Salience Estimation and Faithful Generation:
Modeling Methods for Text Summarization and Generation

Chris Kedzie

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2021

Abstract

Salience Estimation and Faithful Generation:

Modeling Methods for Text Summarization and Generation

Chris Kedzie

This thesis is focused on a particular text-to-text generation problem, automatic summarization, where the goal is to map a large input text to a much shorter summary text. The research presented aims to both understand and tame existing machine learning models, hopefully paving the way for more reliable text-to-text generation algorithms. Somewhat against the prevailing trends, we eschew end-to-end training of an abstractive summarization model, and instead break down the text summarization problem into its constituent tasks. At a high level, we divide these tasks into two categories: *content selection*, or "what to say" and *content realization*, or "how to say it" (McKeown, 1985). Within these categories we propose models and learning algorithms for the problems of *salience estimation* and *faithful generation*.

Salience estimation, that is, determining the importance of a piece of text relative to some context, falls into a problem of the former category, determining what should be selected for a summary. In particular, we experiment with a variety of popular or novel deep learning models for salience estimation in a single document summarization setting, and design several ablation experiments to gain some insight into which input signals are most important for making predictions. Understanding these signals is critical for designing reliable summarization models.

We then consider a more difficult problem of estimating salience in a large document stream, and propose two alternative approaches using classical machine learning techniques from both unsupervised clustering and structured prediction. These models incorporate salience estimates into larger text extraction algorithms that also consider redundancy and previous extraction decisions.

Overall, we find that when simple, position based heuristics are available, as in single document news or research summarization, deep learning models of salience often exploit them to make predictions, while ignoring the arguably more important content features of the input. In more demanding environments, like stream summarization, where heuristics are unreliable, more

semantically relevant features become key to identifying salience content.

In part two, content realization, we assume content selection has already been performed and focus on methods for faithful generation (i.e., ensuring that output text utterances respect the semantics of the input content). Since they can generate very fluent and natural text, deep learning-based natural language generation models are a popular approach to this problem. However, they often omit, misconstrue, or otherwise generate text that is not semantically correct given the input content. In this section, we develop a data augmentation and self-training technique to mitigate this problem. Additionally, we propose a training method for making deep learning-based natural language generation models capable of following a content plan, allowing for more control over the output utterances generated by the model. Under a stress test evaluation protocol, we demonstrate some empirical limits on several neural natural language generation models' ability to encode and properly realize a content plan.

Finally, we conclude with some remarks on future directions for abstractive summarization outside of the end-to-end deep learning paradigm. Our aim here is to suggest avenues for constructing abstractive summarization systems with transparent, controllable, and reliable behavior when it comes to text understanding, compression, and generation. Our hope is that this thesis inspires more research in this direction, and, ultimately, real tools that are broadly useful outside of the natural language processing community.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I have been so incredibly lucky to have been helped by many amazing mentors and colleagues during my time at Columbia. In particular, my advisor Kathy McKeown took a chance on me and gave me many more years than I deserved to explore and learn, for which I am eternally grateful. Fernando Diaz has also been a great intern advisor, mentor, musical collaborator, and someone who talked me off many a proverbial ledge. Sasha Rush, who was my TA in NLP, has been an amazing friend and mentor as well, giving me encouragement and advice throughout my entire time at Columbia. Hal Daumé was always more generous with his time and advice than he needed to be. Umut Özertem was a great intern advisor and kept looking out for me even after the summer had passed.

I'd also like to acknowledge my friends in the NLP lab, department, and various reading groups for all the discussions we have had over the years (my apologies if I forgot anyone): Emily Allaway, Or Biran, Tuhin Chakrabarty, Chad DeChant, Tom Effland, Katy Gero, Chris Hidey, Jeff Jacobs, Giannis Karamanolakis, Faisal Ladhak, Fei-Tzin Lee, Wei-Yun Ma, Jessica Ouyang, Yves Petinot, Jonathan Reeve, Sara Rosenthal, Victor Soto, Karl Stratos, Kapil Thadani, and Elsbeth Turcan.

Finally, I'd like to acknowledge my wife, Alix, who has been unwavering in her support of this journey, and more than anyone has uplifted me through the lows and celebrated the highs.

Dedication

For Alix, my best friend, and true believer that I could ever teach robots to love.

Chapter 1: Introduction

CALIBAN
You taught me language, and my profit on 't
Is I know how to curse. The red plague rid you
For learning me your language!

William Shakespeare, *The Tempest* (I.ii.362-364)

[PROMPT] Q: How many eyes does a horse have?
[GPT-3] A: 4. It has two eyes on the outside and two eyes on the inside.

GPT-3 (Brown et al., 2020), responding to a prompt by Shane (2020)

Somewhere out of the inky-black terminals and unseeable valleys of high-dimensional loss-surfaces, an unexpectedly expressive capacity to generate natural language has emerged in contemporary models of machine learning. In particular, deep learning-based language models have dramatically improved the overall quality of computer generated text, opening the doors to many exciting applications like creative writing tools (Hugging Face, 2019; Samuel, 2019; Seabrook, 2019) and interactive fiction (Robertson, 2019). While these new natural language generation (NLG) methods are quite powerful, they are in practice very difficult to control or constrain. This is unfortunate as it limits responsible application to domains with high fault tolerance, like the those mentioned above and other, essentially low stakes, human guided creative exploration tools. It is unclear if such tools are ready for real deployments in crisis informatics (Starbird and Palen, 2013) or medicine (Gatt et al., 2009).

This thesis is focused on a particular text-to-text generation problem, automatic summarization, where the goal is to map a large input text to a much shorter summary text, and the research presented aims to both understand and tame these machine learning dæmons, hopefully paving the way for more reliable text-to-text generation algorithms. Somewhat against the prevailing trends, we eschew end-to-end training of an abstractive summarization model, and instead break down the text summarization problem into its constituent tasks. At a high level, we divide these tasks into two categories: *content selection*, or "what to say" and *content realization*, or "how to say it" (McKeown, 1985). Within these categories we propose models and learning algorithms for the following problems:

- Content Selection
  - Salience Estimation with Deep Learning Content Selection Models
  - Salience Estimation with Structured Content Selection Models
- Content Realization
  - Data Augmentation and Self-Training for Faithful Neural NLG Models
  - Meaning Representation Linearization Strategies for Content Planning and Controllable Neural NLG Models

The first part, content selection, explores various issues around salience estimation; that is, predicting the importance of an arbitrary unit of text given some surrounding context (e.g., the larger document that contains that text or a user query). In particular, we experiment with a variety of popular and novel deep learning models for salience estimation, and design several ablation experiments to gain some insight into which input signals are most important for making predictions. Understanding these signals is critical for designing reliable summarization models.

We then consider a more difficult problem of estimating salience in a large document stream, and propose two alternative approaches using classical machine learning techniques from unsupervised clustering and structured prediction. These models incorporate salience estimates into larger text extraction algorithms that also consider redundancy and previous extraction decisions.

In part two, content realization, we assume content selection has already been performed and focus on methods for faithful generation, i.e., ensuring that output text utterances respect the semantics of the input content. Since they can generate very fluent and natural text, deep learning-based NLG models are a popular approach to this problem. However, they often omit, misconstrue, or otherwise generate text that is not semantically correct given the input content. In this section, we develop a data augmentation and self-training technique to mitigate this problem. Additionally, we propose a training method for making deep learning-based NLG models capable of following a content plan, allowing for more control over the output utterance generated by the model. Finally, under a stress test evaluation protocol, we demonstrate some empirical limits on several neural NLG models' ability to encode and properly realized a content plan.

It is important to note that the approaches in part two are implemented not for summarization, but for solving *task-oriented dialogue generation* (Mairesse et al., 2010), where the problem is one of modeling the response of a dialogue agent interacting with a human user. The input in this setting is a formal meaning representation of both the communicative goal that the dialogue agent is trying to achieve and the content that is to be realized. While this is technically a data-to-text and not a text-to-text generation problem, the aims of faithful and controllable generation are relevant to both problem classes. With data-to-text generation, however, we get an explicit representation of meaning that makes it easier to measure progress in model faithfulness and control. We therefore consider task-oriented dialogue generation as an idealized form of text-to-text generation, where a content selection module has mapped input text units to a discrete meaning representation for realizing the content. We anticipate that the development of summarization models on discrete semantic representations (Falke et al., 2017; Liao et al., 2018) will continue and that our findings in neural generation methods will carry over to the task of generating summaries from semantic representations (Hardy and Vlachos, 2018). Possible methods of transferring these approaches to the text-to-text regime will be discussed in the final chapter of this thesis.

1.1 Problems in Text-to-Text Generation

Amongst the natural language processing (NLP) research community, machine learning, and especially deep learning, has become the de facto methodological framework for solving text-to-text generation problems. One need only to obtain a large collection of input-output texts, and under this paradigm, the details of the particular generation task can be abstracted away. It is sufficient to re-pose the generation problem as a one of optimization, for which a general purpose neural network model can be trained such that it minimizes the relevant loss function, and in so doing, learns a mapping from the input text to the output text (Sutskever et al., 2014; Bahdanau et al., 2015; Rush et al., 2015; Nallapati et al., 2016; See et al., 2017).

That is to say, the deep learning framework de-emphasizes possessing an algorithmic solution to a problem, and prefers instead a hands off approach: the algorithmic solution is implicit in the dataset, so let the neural network, whose inductive bias is very different from that of the human researcher, learn directly from the data the representations that are most useful to its satisfaction of the optimization criteria.

The logical extension of this central dogma is that the best data is more data (Halevy et al., 2009), and indeed, that is precisely what has happened. Large language model pre-training, where a language model, typically using a transformer architecture (Vaswani et al., 2017), is trained in a self-supervised way on web-scale text, has dramatically expanded the quality of natural language utterances that can be generated by a computational model (Radford et al., 2019; Brown et al., 2020), while also capturing the attention of the popular press (Simonite, 2019; Vincent, 2019). In the NLP research community, pretrained sequence-to-sequence models, the family of deep learning model most commonly used for text-to-text generation, like PEGASUS (Zhang et al., 2019a), T5 (Raffel et al., 2020), or BART (Lewis et al., 2020), have led to impressive gains in the fluency and coherence of modestly-sized paragraphs, as well as automatic task metrics like BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004).

There are downsides to this approach however. By learning only from surface level forms, it is possible for the models to appear like they contain knowledge, but in reality, they are only modeling the probability of word sequences (Bender and Koller, 2020). This can cause them to hallucinate information not present in the input or imply propositions contrary to what was given (Wiseman et al., 2017; Kryscinski et al., 2019; Maynez et al., 2020; Kryscinski et al., 2020). Without an understanding of pragmatics, they cannot know how an argument or chain of propositions holds together, only that they are statistically likely. Moreover, they will learn many implicit and harmful biases of the society that produced the corpora they are trained on, including but not limited to negative stereotypical word associations (Bolukbasi et al., 2016; Nissim et al., 2020) and outright hate speech (Lee, 2016).

With all of that in mind, we would like this thesis to emphasize the importance of articulating understandable steps or sub-tasks on the way to solving the larger summarization or generation problems. While we do not abandon machine learning and deep learning, we instead show how they may be used parsimoniously to solve the actual problems at hand and not simply learn the dataset. Where we do use machine learning models, we try to design experiments which reveal which signals relevant to the actual problem are being used to make predictions, and to establish some empirical limitations on their ability to represent the input data and perform successfully.

We now describe some of the central problems of summarization and text-to-text generation that we are interested in solving. In most summarization tasks, we are generally interested in a text's *salience*, that is to say, the general importance or relevance of a given text unit with respect to its context. Salience is usually the primary dimension of the input data that we wish to measure or predict for determining summary content.

In Chapter 3, we study salience estimation in the *sentence extractive, single document summarization task*, where the goal is to classify which sentences in an input document should be included in an extract summary. In this case, the input document is the context, and the units of text for which we are estimating salience are the document sentences. An extractive summary is a subset of sentences that have maximum salience while satisfying a length budget constraint, typically in the summary word or byte length. While the constrained subset selection problem is interesting and has been studied previously (Goldstein and Carbonell, 1998; McDonald, 2007; Lin and Bilmes, 2010), we focus on modeling the salience estimation task specifically.

We study a variety of popular and novel deep learning architectures for implementing the salience prediction task. Our key contributions here are not only a model architecture, but also our systematic study of the combination of sentence and context (i.e. document) level encoders as well as the manipulation of the input documents to ablate which surface features are available to a given model. Through these input data ablations, we can gain a better understanding of how the salience prediction mechanism is working.

But what about more difficult summarization tasks? In Chapter 4, we study salience estimation for *query focused, streaming, sentence extractive summarization*. In this task, we add a search query and time as additional elements to the summarization problem. As input we are given a time-ordered stream of news articles and a query, typically a notable real-world event, e.g., `"hurricane sandy"`. Our objective is to extract sentences that are relevant to the query event while minimally redundant to previously extracted sentences. Unlike the previous problem, the salience of a given sentence is not constant but monotonically decreases as time progresses. Due to the large volume of input texts, in attempting to model the salience of a sentence we must now also model the redundancy between sentences and previous extraction decisions to perform competently.

We incorporate a salience estimation model into two possible approaches to extracting query-relevant summary sentences from the news stream. The first method uses the salience estimates to bias an affinity propagation clustering algorithm (Dueck, 2009) to identify exemplar sentences which we extract for the summary. The clustering algorithm must trade off representativeness versus the salience of individual sentences when selecting exemplars, i.e. an exemplar must be a good representative of its cluster but also be important under the salience estimator. This approach works in hourly batches, predicting the salience of sentences, clustering them, and then adding the exemplars to a rolling update summary. The salience estimates also adaptively control the number of clusters produced, allowing the model to adapt the number of updates to fit the volume of salient sentences found in the stream.

The first method has several drawbacks. The summary selection is not done to optimize the final summary evaluation measure and the predictions of salience are static and cannot take into account previous decisions the summarizer has made. Additionally, the use of clustering means that there will always be some latency between when important information is known and when we can extract it for the summary. In domains like crisis informatics, minimizing these delays are critical (Starbird and Palen, 2013).

To address these limitations, we recast stream summarization as a sequential decision-making problem (Littman, 1996), where we learn a policy for extracting a sentence based on its estimated salience as well as its relation to previously extracted sentences. The sequential decision-making view opens our model up to exposure bias as the learned policy will suffer from a train-test distribution mismatch if our reference policies are overly optimistic or pessimistic. To mitigate this, we employ a learning-to-search style training regime (Chang et al., 2015) to train a policy to make locally optimal decisions when following either a noisy learned policy or an oracle reference policy. The result is a fully online summarization model whose local decisions positively correlate with good overall summary evaluation measures. Additionally, because the model is greedy, it does not suffer as much from the negative effects of latency as the clustering model does.

In our experiments with single-document extractive summarization, we found that neural models heavily exploited position-based heuristics (i.e., did the sentence occur in the lead paragraph) to determine sentence salience, which arguably does not capture the essence of the summarization problem. In our work on stream summarization, we show that with careful feature and model design, we can capture salience beyond such heuristics. In particular, we show that content, location, and redundancy features can be used to predict salience in this more challenging scenario.

In Chapter 5, we move to problems of content realization, after the content selection process has been performed. Here we focus on the related goals of *faithful* and *controllable* generation using neural NLG models. A neural NLG model is faithful if it can generate utterances that are semantically correct with respect to the information extracted in the content selection stage. One of the central tensions in a neural NLG model is that between the encoder, which creates a representation of the input, and the decoder, which is functionally a language model conditioned on the encoder representation. The decoder language model must simultaneously place high probability on output word sequences that are likely given the training corpus, but also prefer output word sequences that are correlated with the encoder representation of the input sequence. This conflict can lead to hallucinations as the language model may occasionally put more probability mass on a sequence of words that is frequently observed in the training data, but not necessarily licensed by a particular input.

We hypothesize that this failure mode happens in part because the decoder, by predicting next word continuations, is both implicitly planning the layout of the utterance but also trying to satisfy the constraints given by the input, and that alleviating the decoder language model of the planning task may improve the faithfulness. We propose developing controllable neural NLG models, i.e. models that can follow an explicit plan determining the surface realization order of the intended utterance. Controllable models learn to represent the layout of the intended utterance implicitly in their encoder, and thus the decoder language model has less flexibility in selecting the next words, which can lower the chances of hallucinating text and improve the overall faithfulness of the generation model.

We also believe that controllable generation has additional benefits beyond increased faithfulness. For one, it will enable more integration of neural NLG models into large NLG pipelines, (Castro Ferreira et al., 2019). Controllable generation at the level of shallow phrase chunk ordering like we are proposing may also lead to implementations of cognitively plausible discourse ordering theories like Centering Theory (Grosz et al., 1995) or Accessibility Theory (Ariel, 2001), which place constraints on the discourse ordering of entities.

Evaluating the faithfulness of a language generation model for open-world summary generation tasks is non-trivial (Kryscinski et al., 2020; Maynez et al., 2020). In order to simplify things, we study faithful and controllable generation in the context of task-oriented dialogue generation, where, given an explicit representation of a dialogue agent's belief state and goals, we must generate an appropriate natural language utterance. Because the input is an explicit, formalized representation of the meaning of the intended utterance, manual and even automatic checking of the faithfulness of an utterance/meaning representation pair becomes much simpler. Since the concerns of faithful and controllable generation are still incredibly important to generating summaries, we consider the explicit meaning representations as an idealized version of summarization system's content selection stage. Faithfulness is critical for any real summarization application; the reader has to be able to trust that content is correct or it is functionally useless. Controllable generation will further increase reliability and faithfulness, while also allowing the tailoring of the summary to focus on particular user needs, for example targeting generation to focus on a particular set of entities.

For our contribution to faithful generation, we propose a novel data augmentation method for sequence-to-sequence models. We observe that a popular sequence-to-sequence NLG model trained on a task-oriented dialogue generation dataset produces fluent and natural utterances that are unfortunately frequently semantically incorrect with respect to the input meaning representation. We then present evidence that the reason for these errors are spurious correlations between the inputs and outputs in the training data. We also note that by injecting random noise into the unfaithful NLG model we can cause it behave in an uncontrolled but useful manner: it generates utterances that are not faithful to the input but do not exhibit the spurious correlations or exhibit them to a lesser degree. We generate a synthetic corpus of these utterances, and then use another semantic parsing model to give them correct meaning representations. Remarkably, sequence-to-sequence models trained on the union of original training data and the synthetically generated training examples exhibit increased faithfulness without hurting their fluency. We also find that in the union dataset, the problematic spurious correlations are diminished.

While this data augmentation method helps reduce semantic errors, it leaves the surface realization of utterances up to the decoder language model. Our second contribution to neural NLG models is an alignment training method that reliably produces a model capable of following a discourse ordering plan when generating utterances. Our method works by aligning the individual components of a meaning representation to their reference utterances on the training set. Given this alignment, we then map a meaning representation into a linear sequence of tokens, such that the order of the individual components corresponds to the realization order in the training reference. Training an arbitrary sequence-to-sequence model to map this linearized meaning representation to its reference utterance induces the ability to control the model at test time (i.e., we can use a planning model to propose an ordering of the meaning representation's sub-components, and the controllable sequence-to-sequence model will attempt to realize them in that order). To achieve a different ordering, one need only permute the input sequence.

In our experiments, we also evaluate how well models are able to follow adversarially generated plans that do not have human, English language ordering preferences and show that models struggle to realize utterances correctly in this setting. We finally propose another data augmentation scheme to generate constituent phrase data that gives explicit examples of how phrases can be composed into larger units and how that systematically changes the meaning representation. We find that this additional data improves the robustness of the control behavior on these more difficult to follow plans.

1.2 Contributions

We now briefly overview the contributions of this thesis.

1. We propose a systematic evaluation of deep learning models for extractive single document summarizations (Kedzie et al., 2018). Our evaluation of several popular neural architectures shows that:

   - Position features, even when not explicitly represented in the model architecture, are a dominant feature exploited by the model in news and research article summarization. This arguably makes neural models of summarization on these domains a computationally expensive way to identify the lead paragraph of an article.
   - Content features exist across a variety of word classes (e.g., nouns) but are not as strong

of a signal as position.

    - Word embedding averaging is about as effective as recurrent or convolutional sentence encoders. This suggests that current neural summarizaton models are not able to effectively exploit sentence content beyond the notion of a bag-of-words.
    - Autoregressive sentence extraction models (i.e., where prior sentence extraction decisions are fed back into the model before making subsequent decisions) are as effective as non-autoregressive sentence extraction models. This finding implies that at the point of identifying salience, the existing models are not effectively able to exploit the document level context.

2. In the task of query focused, sentence extractive, streaming news summarization, we propose two models for providing extractive update summaries. (Kedzie et al., 2015, 2016)
    - We show that in a setting where position or frequency heuristics are less reliable, we can construct a sentence salience estimation model that effectively uses domain knowledge to identify important content. Additionally, we can embed this salience model in a clustering algorithm, such that sentence selection is done in a way that considers a sentence's salience as well as its representativeness when making extraction decisions. This model is able to identify salient content in a more timely fashion as news comes in than competing approaches.
    - We then show how to learn a greedy sentence extraction policy that optimizes the complete summarization task (as opposed to the previous approach which only learned salience estimation). This policy combines information about sentence salience and redundancy, as well as previous extraction decisions and it yields a high quality extract summary, even while operating in a greedy manner. This method further improves on the clusering approach in terms of identifying salience content with minimal latency.

In the area of data-to-text generation, we make the following contributions to faithful and controllable generation of text from a meaning representation (Kedzie and McKeown, 2019, 2020).

1. We propose a novel data augmentation technique called noise injection and self-training, which allows us to create additional synthetic training examples for the meaning representation to text generation task. We show that the synthetic corpus created by this method, does not possess some of the spurious correlations and dataset creation artifacts present in the human authored training data. Neural NLG models trained on the union of original and synthetic data are more faithful, (i.e., they generate utterances with fewer semantic errors, than more computationally expensive models that rely on discriminative reranking to ensure correctness).

2. We propose an encoder input linearization strategy for sequence-to-sequence NLG models called alignment training. This technique yields neural NLG models that are controllable, i.e., capable of following a discourse ordering plan. We demonstrate that this technique works for both recurrent and transformer based sequence-to-sequence models trained from scratch as well as when fine-tuning a large, pretrained sequence-to-sequence model. We also perform extensive stress testing of the plan following behavior under adversarial conditions, and propose a phrase-based data augmentation method to improve performance in this more challenging setting.

Finally, we conclude with a discussion of the limitations and future directions this work might take. In particular, we focus on how faithful generation might be applied to text-to-text summarization.

1.3 Organization

This thesis is organized as follows. Chapter 2 gives a brief history of the field of NLG and as well as related work in text-to-text generation and summarization. Chapter 3 and Chapter 4 focus on various salience estimation and content selection problems for text summarization with a focus single document summarization and stream summarization respectvely. In Chapter 5, we then discuss our proposed methods for faithful and controllable neural NLG models. Finally, limitations, future work, and concluding thoughts are presented in Chapter 6. While each chapter is working towards the larger narrative and themes of this thesis, they can be read in a self contained manner. To that end, Notational conventions are specific to each chapter and are introduced on a per chapter basis.

Chapter 2: Related Work

In this chapter we attempt to briefly characterize the history of NLG, with a focus on text summarization as well as the development of neural NLG models since these topics are central to the contributions of this thesis. While many bibliographies of the field might start their history of NLG with the beginning of modern computer hardware (i.e., after the development of silicon transistors and the growth of mainframe computing in the 1950s), we briefly highlight instances of algorithmic language generation starting in the middle ages through the early modern period, and how they connect to notable figures in the broader history of computation. We proceed chronologically, starting in antiquity, and then moving to the formulation of modern NLG in the period between 1980 and 2000. We then discuss the growth of data-driven method for text summarization between 2000 and 2014. We conclude with a description of the neural network, a.k.a. deep learning, revolution that arguably began in 2014, and which has had a significant impact on NLP, but especially NLG.

2.1 Natural Language Generation in Antiquity

The desire to build algorithms and machines that generate natural language has an extensive history in both art and science, as well as spiritual practice, and especially soothsaying and prognostication. One early account of a language generation algorithm comes from 14th century historian ‘Abd ar-Rahmān ibn Khaldūn (1332 – 1406), who writes in his universal history, the *Muqaddimah* (1377), of a circular prognostication and divining tool used by Sufi mystics called a zā’irjah.[1] Its practice is

> “a branch of the science of letter magic, practiced among the authorities on letter

[1] Franz Rosenthal in his English translation of the *Muqaddimah* suggests the name is derived from the Persian words zā’icha meaning “horoscope” or “astronomical table” and dā’ira meaning “circle.”

> magic, is the technique of finding out answers from questions by means of connections existing between the letters of the expressions used in the question."

Ibn Khaldūn points to an earlier treatise by the Sufi scholar Abu al-Abbas as-Sabti (1129 – 1204) of Marrakesh as a source of instructions for the device's use, suggesting the practice is at least as old at the 12th century (Rosenthal et al., 1958).

The zā'irjah itself consists of a series of concentric circles divided into 12 sections by six chords. The various segments of the diagram are annotated with letters and numerals. Additionally, the zā'irjah is accompanied by a lookup table mapping letters to numbers. See Figure 2.1 for an example. According to painstaking reconstructions done by Link (2010), a "key poem" was used to pose a question to the zā'irjah and serve as a mnemonic device/mapping of letters to entries in the lookup table. A combination of rules and astronomical observations (antiquity's equivalent of a random seed) were then applied to the key poem to read off series of characters from the zā'irjah. The operator would then interpret those letters into an answer. "The fact that only consonants are written down in Semitic languages permits the meaningful interpretation of many random permutations of symbols," (Link, 2010) suggesting that cherry-picking outputs and over-ascribing intelligence, knowledge, and even wisdom, to a language generation algorithm are as old as the practice of NLG itself.

In a secondary account from a manuscript found at the library of Rabat in Morroco, it is written that a skeptical ibn Khaldūn asked of the device how old it was, "[Is the] zā'irjah [a] recent or [an] ancient science?" Allegedly he received the answer, "The Holy Spirit will depart, its secret having been brought forth / To Idrīs, and through it, he ascended the highest summit," connecting the practice to the sage Idrīs who is one of the eldest ancestors in the Quranic tradition (Rosenthal et al., 1958; Link, 2010).

The teachings of Arabic mystics, including the practice of zā'irjah, as well as the Kabbalistic tradition embodied in the *Sefer Yetzirah* are known to have strongly influenced the Majorcan Christian mystic, Ramon Llull (1232-1315) (Kahn, 1980; Link, 2010; Priani, 2017). Llull, who is regarded as an early philosopher of combinatorics, logic, and computation (Bonner, 2007; Knuth,

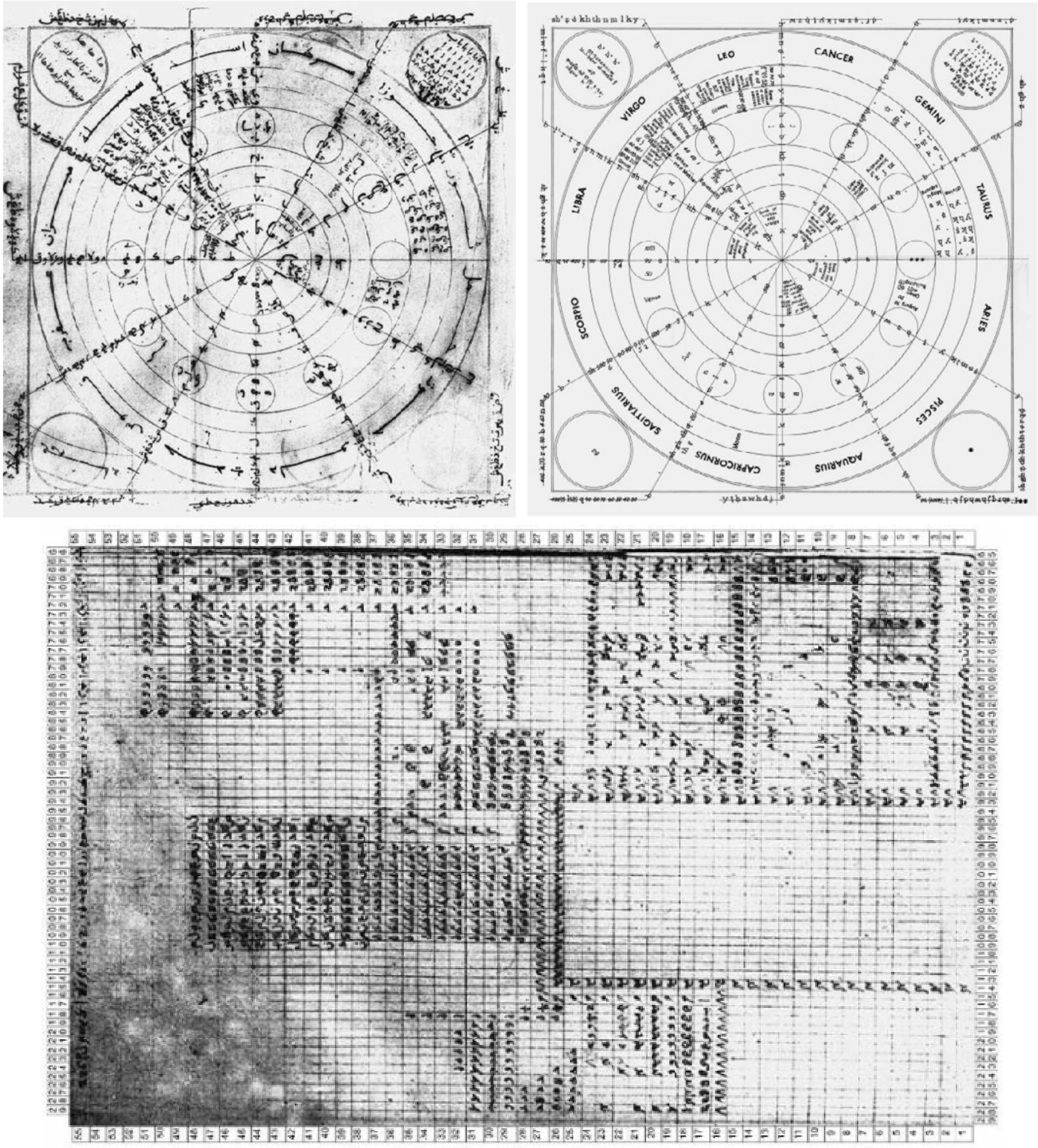


Figure 2.1: A zā’irjah from a 15th century Turkish manuscript of the *Muqaddimah* (top left), its English translation from Rosenthal et al. (1958) (top right), and its lookup table (bottom). Images taken from Link (2010).

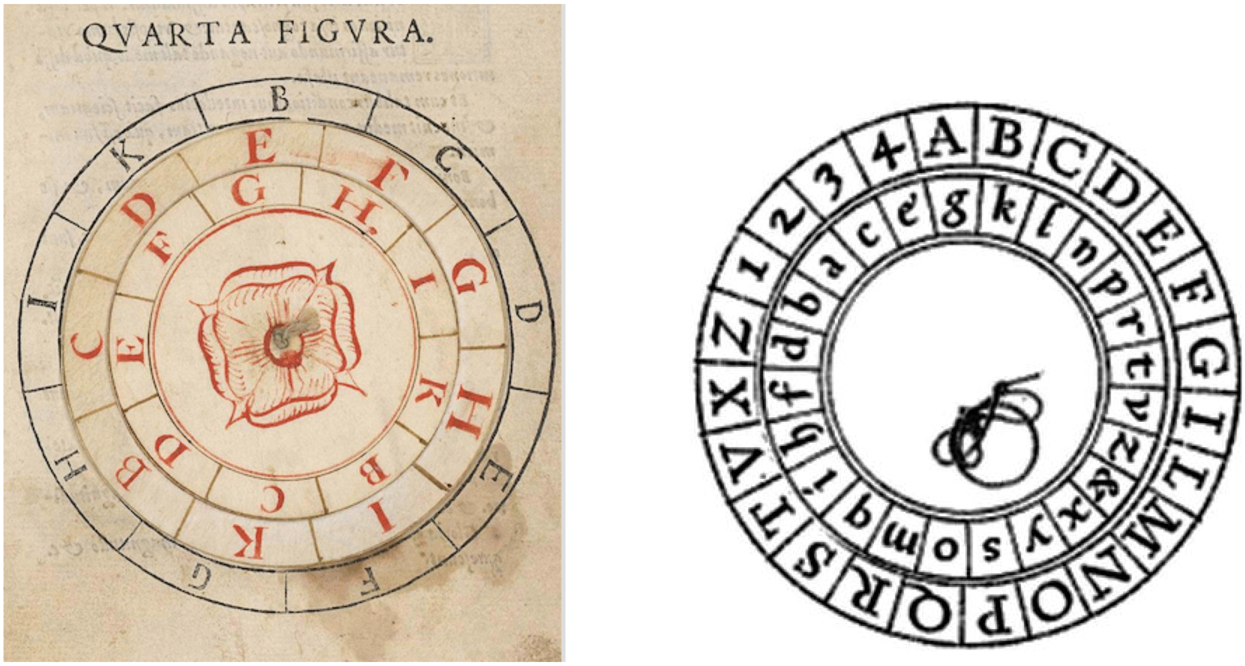

QVARTA FIGVRA.
A B C D E F G I L M N O P Q R S T V X Z 1 2 3 4
a c e g k l n p r t v z & x y s o m q i h f d b

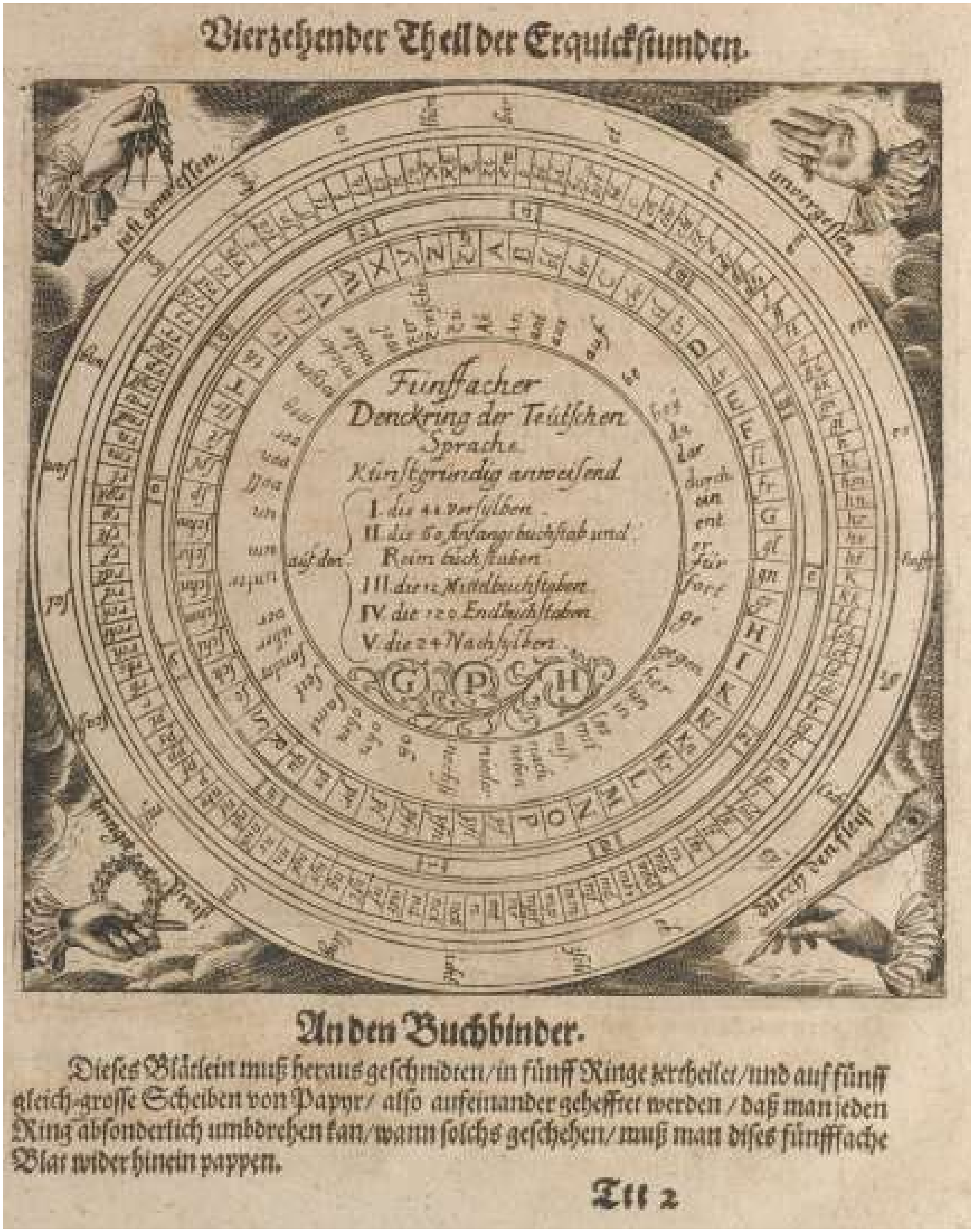
Figure 2.2: (Left) A volvelle from Llull's *Ars Magnus* and (right) Alberti's cipher disk.

2006; Priani, 2017), developed a computational system based on moveable concentric circles made of paper and connected by string. The workings of these *volvelle*[2] are described in his master work, *Ars Magna* (1305). According to his system, concepts were assigned letters which were manipulated to generate new statements involving the concepts, and he claimed could be used to determine the truth of any proposition (Crupi, 2019). Llull's work is also thought to have influenced the polyalphabetic substitution cipher developed by Leon Battista Alberti (1404 – 1472) (see Figure 2.2), the same core cryptographic technology used in the Enigma machine (Kahn, 1980).

Llull's use of the volvelle as generative device was influential throughout medieval Europe, where volvelle were used in both art and science. Arguably they reached their zenith in the baroque works of Georg Philipp Harsdörffer (1607 – 1658). His master work volvelle, *Fünffacher Denckring der Teutschen Sprache* (1651), consisted of five paper discs (depicted in Figure 2.3), and was designed to model German word formation. It was also advertised as an aid in the production of poems and other literary forms (Schäfer, 2006).

While computational devices before modern computing were limited in complexity by their

[2] The name *volvelle* comes from the Latin, literally "to turn."



Vierzehender Theil der Erquickſtunden.

An den Buchbinder.

Dieſes Blätlein muß heraus geſchnidten/ in fünff Ringe zertheilet/ und auf fünff gleich-groſſe Scheiben von Papyr/ alſo aufeinander gehefftet werden/ daß man jeden Ring abſonderlich umbdrehen kan/ wann ſolchs geſchehen/ muß man diſes fünfffache Blat wider hinein pappen.

Ttt 2

Figure 2.3: An illustration of the German word generator volvelle, *Fünffacher Denckring der Teutschen Sprache* (1651) by Georg Philipp Harsdörffer.

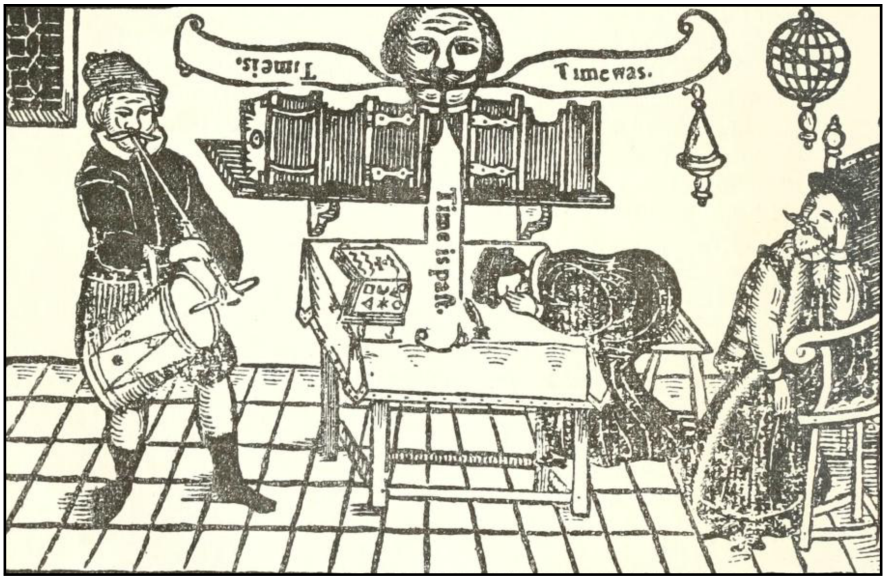


Figure 2.4: A 1630 woodcut depicting Roger Bacon's talking bronze head, a mischievious talking autamata allegedly capable of answering any question. Image taken from Hyman (2016).

construction materials, chiefly paper, the dream of speaking automata was also alive in myth. See for example Figure 2.4, in which a 17th century woodcut print depicts a talking head capable of answering any question. This "brazen head" was allegedly built by the monk Roger Bacon, who in addition to being an early philosopher of science and linguist, might also be considered the first NLP engineer if folklore is true (Hyman, 2016; Hackett, 2020).

2.2 Natural Language Generation from 1980–2000

Returning to the present day, computer aided production of human language closely follows the beginning of modern computing, starting with work on machine translation (MT) systems developed in the 1950s and '60s (Ornstein, 1955; , ALPAC; Hutchins, 2006). Early work on producing extract summaries of research articles also dates back to this period (Luhn, 1958) as well as also notable experiments in generating text purely from syntactic structures (Yngve, 1961).

However, NLG did not begin to coalesce as a distinct subfield until the 1980s which saw the first workshops devoted specifically to NLG and a convergence on the formalisms and problems central to language generation (Reiter and Dale, 1997; McDonald, 2010). NLG researchers of this period were focused on at least four main research programs: (i) linguistically motivated grammars for generation, (ii) frameworks for representing knowledge and concepts, (iii) models of the human receiver of the generated text, and (iv) models of discourse and control for planning the realization of utterances (Mann et al., 1981; McKeown, 1986).

A variety of grammatical formalism were proposed during this period to be the syntactic backbone of language generation algorithms, including Functional Grammar (Halliday, 1985), Transformational Grammar (Chomsky, 1965), Generalized Phrase Structure Grammar (Gazdar et al., 1985), and others. Many frameworks for generation proliferated at this time (often incorporating one of those grammar formalisms), including Knowledge and Modalities Planner (KAMP) (Appelt, 1982), Penman (Hovy, 1993), MUMBLE (McDonald, 1981), TEXT (McKeown, 1982) and others (Mann et al., 1981). While there appears to be a great diversity of approaches, over time the community began converging on a fairly similar pipeline of modules when it came it implementation (Reiter, 1994).

Indeed, most NLG systems from this period could be understood as a pipeline of modules for text planning, sentence planning, and linguistic realization (Reiter and Dale, 1997). In the text planning stage, the concepts to be conveyed are selected, possibly discarding less essential information, and arranged into a discourse plan or ordering. In the sentence planning stage, the concepts from the previous stage are grouped into individual sentences, and lexicalizing concepts and referring expression generation is performed. Finally, in linguistic realization, the intermediate representation from the sentence planning stage is converted into a natural language utterance, often by linearizing and inflecting some syntactic/morphological representation.

Since these systems primarily generated language by starting from non-linguistic and/or semantic representations of concepts, they are often referred to as *concept-to-text* generation or, as more commonly known today, *data-to-text* generation (Gatt and Krahmer, 2018). These systems were applied to a variety of data-to-text problems including weather forecast generation (Goldberg et al., 1994), statistical report generation (i.e. generating a report from numerical or statistical data in a spreadsheet) (Iordanskaja et al., 1992), or as a writing aid to improve the productivity of human authors (Springer et al., 1991; McKeown et al., 1994; Paris et al., 1995). Data-to-text generation came to prominence alongside expert systems (Todd, 1992), including as a means to explain them (Swartout, 1983). As such, these NLG systems suffer from some of the same drawbacks as expert systems, often requiring extensive domain knowledge and manual rule or grammar engineering which became difficult to maintain or amend over time.

In the late 1980s and into the 1990s, as larger text corpora became available and statistical and/or machine learning techniques spread through the community, text-to-text generation (i.e. methods of directly mapping unstructured text inputs to text outputs) increased in popularity. Text-to-text generation is less defined by a specific generation task, method or unifying theory, but on the use of large collections of example input/output text pairs.

Statistical machine translation (SMT) emerges from this period as the dominant success story for machine learning applied to text-to-text generation problems. The availability of digitized bilingual data made it possible to create parallel sentence translation corpora and apply statistical word alignment and translation techniques (Gale and Church, 1993). The canonical IBM translation models were developed in this period (Brown et al., 1988, 1993). Statistical methods marked a stark improvement over attempts at interlingua or semantics based translation systems which often required significant manual effort in pre or post editing (Hutchins, 1994). There were even some limited experiments using learned classifiers to predict sentence salience for summarization (Kupiec et al., 1995).

2.3 The Emergence of Data Driven Extractive Summarization (2000–2014)

While the MT community could take advantage of large (for the time) parallel corpora, it was not until the 2000s that there were readily available collections of documents and their summaries for which to perform the kind of supervised machine learning that was being successfully applied to MT. Beginning in 2001, the Document Understanding Conferences (DUC) began steadily producing collections of documents with reference summaries, bringing together NLP researchers particularly around multi-document summarization (Spärck Jones, 1999; Harman, 2001; Nenkova, 2005).

Many significant summarization systems from this time still relied on unsupervised methods to create extract summaries, usually exploiting document collection statistics to determine the salience of input text units. The representation of text units as TF-IDF weighted bags-of-words (Spärck Jones, 1972) is prevalent in this era. For example, Radev et al. (2000) identify salience sentences in a document cluster by their similarity to the average TF-IDF vector of the document cluster. In Erkan and Radev (2004), sentences are treated as vertices in a fully connected graph, with edge weights determined by the cosine similarity between each sentence's TF-IDF weighted bag-of-words representation. The PageRank algorithm (Page et al., 1999) is then used to determine the graph centrality of each sentence; the sentences most central in the graph are considered the most salient and extracted for the summary.

Other methods used alternative formulations to exploit frequency for summarization, most notably Lin and Hovy (2000) who identified topic signatures using a likelihood ratio test (Dunning, 1993) to identify terms that occurred unusually frequently in a document given a large generic background corpus. After removing stopwords, word frequency on its own has also been shown to be an effective signal for identifying salient sentences (Nenkova and Vanderwende, 2005).

Following on the initial research by Kupiec et al. (1995), other work using supervised learning for summarization begins to emerge from this time (Conroy and O'Leary, 2001; Osborne, 2002; Hirao et al., 2002; Sipos et al., 2012). Here, summarization is framed as a sentence classification task (i.e., which sentences from the input document should be included in a summary).

Most of the text-to-text summarization approaches from the 2000s are primarily extractive systems. While significantly constrained in their expressive quality (i.e. only sentences found in the input can be used to construct the output) especially compared to earlier data-to-text methods, designing data-driven features for unsupervised and supervised statistical machine learning proved to be much more scalable and an easier path to improved summarization performance (Nenkova and McKeown, 2011).

There are some notable works that attempted to perform limited abstractive summarization.

Barzilay and McKeown (2005) developed an unsupervised method of sentence fusion (i.e., combining sentences with the same or overlapping content using their syntactic parses as backbone structure), that has been extended and refined by others (Marsi and Krahmer, 2005; Filippova and Strube, 2008). Heuristically driven phrase deletions were also explored to reduce less salient information in extractive summarization (Jing, 2000; Zajic et al., 2007). In the supervised case, there were several works that attempted to learn to delete non-essential phrase constituents. This was formulated as either a pipeline of learned compression and extraction models (Wang et al., 2013) or as a joint model of extraction and compression (Martins and Smith, 2009; Berg-Kirkpatrick et al., 2011). While sentence fusion was capable of some abstractive rewriting, the other approaches mentioned here predominantly focused on compression, i.e. word or phrase deletion, to generate novel summary text, which is only one of the many ways that human abstractors perform the summarization task (Jing and McKeown, 2000).

Streaming or temporal summarization was first explored in the context of topic detection and tracking (Khandelwal et al., 2001; Allan et al., 2001) and more recently at the Text Retrieval Conference (TREC) (Aslam et al., 2013). Approaches to this problem often perform filtering by estimated salience before relying on multi-document summarization techniques to select text units to construct rolling update summaries (Guo et al., 2013; McCreadie et al., 2014). These pipelines while effective, do not attempt to jointly optimize the salience and selection.

2.4 Neural Natural Language Generation Models (2014–Present)

While feed-forward neural networks had been used previously as part of phrased-based SMT systems (Schwenk et al., 2006), there was an increased interest in the early-mid 2010s around using recurrent neural network (RNN) language models (Mikolov et al., 2010) as a rescoring method for an SMT decoder (Auli et al., 2013; Cho et al., 2014). RNNs could exploit (in theory) unbounded source and target prefix information that was difficult to capture in ngram or feed-forward models. Cho et al. (2014) is particularly notable because they propose separate encoder/decoder RNNs, and while intended for rescoring and not generation directly, this general architecture constitutes the "sequence-to-sequence" backbone of most neural MT (NMT) and neural NLG models.

Shortly thereafter, Sutskever et al. (2014) proposed the now ubiquitous sequence-to-sequence model to perform translation directly. Bahdanau et al. (2015) also proposed a sequence-to-sequence model with an attention mechanism, which both made optimization easier (error feedback, i.e., gradients, could now be routed directly from any decoder word prediction step to any arbitrarily distant timesteps in the encoder) and allowed for visualization of the NMT decoder's alignment with the encoder. NMT models, while conceptually simpler than phrase-based SMT, were starting to achieve state-of-the-art results (Bojar et al., 2016) and wide-spread industry adoption (Wu et al., 2016; Gehring et al., 2017). It did not take long for researchers to adapt the sequence-to-sequence model to other language generation problems, e.g. generating captions from images (Vinyals et al., 2015), sports summaries for box scores Lebret et al. (2016); Wiseman et al. (2017), or from semantic representations (Wen et al., 2015; Dušek and Jurčíček, 2016).

The introduction of the CNN-DailyMail corpus by Hermann et al. (2015) allowed for the application of large-scale training of deep learning models for summarization. In sentence extractive summarization, researchers proposed a variety of hierarchical models with distinct sentence and document level encoder networks that enabled sequential prediction of which sentences or words should be included in the summary (Cheng and Lapata, 2016; Nallapati et al., 2017).

Perhaps more exciting was the surge in abstractive summary generation. Rush et al. (2015) developed an attention-based deep learning model capable of generating headlines from the lead sentence of an article. Subsequently, Nallapati et al. (2016) showed that a sequence-to-sequence model could encode a whole news article and then generate an abstractive summary word by word. Additionally, the line between extractive and abstractive summarization was blurred by the addition of learned copy-mechanisms which could selectively transfer named entities and other out-of-vocabulary terms into the output summary, further improving summary quality (See et al., 2017).

Historically, the field of NLG has relied heavily on grammars and templates to generate text. The fact that neural models could yield reasonably fluent and acceptable summaries given so little pre-specified structure or features is truly impressive. Interestingly, term frequency, was not explicitly represented in either the neural extractive or abstractive summarization models, begging the question as to how they were learning to identify salient content.

Model architecture continued to evolve and in 2017, Vaswani et al. proposed a recurrence-free neural sequence model, built around so-called *transformer* layers, which rely on multiple parallel self- and context-attention mechanisms. This model was designed with optimization speed in mind, and was subsequently used in large scale language model pretraining on web-scale text, spawning the BERT-family of models (Devlin et al., 2019). Large language model pre-training with task-specific fine-tuning became a dominant paradigm in NLP with BERT and its descendants setting records on many downstream text prediction tasks (Ruder et al., 2019).

The transformer layer was also used in the generative pre-training (GPT)-family of models (Radford et al., 2018), which was also trained on web-scale data, but with an auto-regressive language modeling objective. The second generation of these models, GPT-2 (Radford et al., 2019), received notoriety both amongst NLP researchers but also the wider public, as its release was initially delayed given "ethical concerns" about releasing such a powerful language generation model (Vincent, 2019; Seabrook, 2019). GPT-2 exhibited impressive completions of prompt texts, and could be used to generate natural looking paragraphs on arbitrary topics, with longer spans of fluent text than previously thought possible.

GPT-2 could also be fine-tuned to perform task specific conditional generation (Ziegler et al., 2019; Golovanov et al., 2019), however, its architecture was that of a neural language model without distinct encoder/decoder networks; conditional generation was achieved by encoding problem instances as prompts to be continued. Proper sequence-to-sequence variants of the large, transformer-based language model were subsequently developed using various sequence-level denoising autoencoder objectives (Zhang et al., 2019a; Raffel et al., 2020; Lewis et al., 2020). The intention of such models was that they could be fine-tuned for more task-specific sequence-to-sequence problems like summarization, translation, or even arbitrary data-to-text tasks like dialogue generation.

While these models were producing text at a level of quality that had not previously been realized with traditional NLG approaches, a closer examination of model outputs revealed glaring flaws in the semantic correctness of the generated text (Kryscinski et al., 2019, 2020; Maynez et al., 2020). Dušek et al. (2020) found that the quality of neural NLG models can vary significantly, with relatively similar architectures yielding both poor and competitive performance with respect to the semantic correctness of model outputs. In practice most competitive neural NLG models use a variety of beam reranking techniques to improve output faithfulness to the inputs (Dušek and Jurčíček, 2016; Juraska et al., 2018; Wen et al., 2015), as well as copy and coverage mechanisms to improve the recall (See et al., 2017; Elder et al., 2018).

NLG researchers have also begun to explore the degree to which neural models can be constrained to follow discourse plans or other structured objects. Nayak et al. (2017) and Reed et al. (2018) explore several ways of incorporating shallow sentence planning into dialogue generation via the grouping of input sequences into distinct subsequences or by inserting discourse variable tokens into the encoder input sequences to indicate contrasts, comparisons, or other groupings. Balakrishnan et al. (2019) experiment both with tree structured input meaning representations and encoders and compare them to linearized trees with standard sequence-to-sequence models. While these papers find that neural NLG models can consistently follow these discourse ordering constraints, they do not systematically explore how other linearization strategies compare in terms of faithfulness, and they do not evaluate the degree to which a sequence-to-sequence model can follow realization orders not drawn from the training distribution.

Castro Ferreira et al. (2017) compare a neural NLG model using various linearizations of abstract meaning representation (AMR) graphs, including a model-based alignment very similar to the alignment-training linearization presented in this work. However, they evaluate only on automatic quality measures and do not explicitly measure the semantic correctness of the generated text or the degree to which the model realizes the text in the order implied by the linearized input.

Works like Moryossef et al. (2019a,b) and Castro Ferreira et al. (2019) show that treating various planning tasks as separate components in a pipeline, where the components themselves are implemented with neural models, improves the overall quality and semantic correctness of generated utterances relative to a completely end-to-end neural NLG model. However, they do not test the “systematicity” of the neural generation components, i.e. the ability to perform correctly when given an arbitrary or random input from the preceding component, as we do here with the random permutation stress test.

Many papers mention input linearization order anecdotally but do not quantify its impact. For example, Juraska et al. (2018) experiment with random linearization orderings during development, but do not use them in the final model or report results using them, and Gehrmann et al. (2018) report that using a consistent linearization strategy worked best for their models but do not specify the exact order. Juraska et al. (2018) also used sentence level data augmentation, i.e. splitting a multi-sentence example in multiple single sentence examples, similar in spirit to our proposed phrase based method, but they do not evaluate its effect independently. Wiseman et al. (2018) uses an order invariant encoder to produce a latent plan which guides the decoder. Ignoring the encoder and specifying a latent plan would allow for some control over realization order but the degree to which arbitrary realization orders can be achieved is under explored. Additionally, it is not guaranteed that latent plan states uniquely correspond to different meaning representation components.

Chapter 3: Salience Estimation with Deep Learning Content Selection Models

Salience estimation, that is, the prediction of the importance or relevance of a unit of text, is a critical step for any text summarization algorithm (Nenkova and McKeown, 2011). Since the size of the desired output summary is constrained to be much smaller than the original document or documents being summarized, it is necessary to prioritize some information over others when deciding the content of the summary. Estimating the salience of various units of text (i.e., words, phrases, sentences, etc.) enables summarization algorithms to perform this prioritization.

There is no universally agreed upon definition of salience, so its estimation starts on rather shaky epistemological ground. What is most salient will vary significantly from reader to reader, and depend largely on their particular information need and/or prior knowledge (Spärck Jones, 1999). In this chapter, we focus on a supervised learning scenario, where the training corpus consists of a single document paired with a human reference abstract summary prepared by a domain expert. In this setting, we can rely on a data-driven definition of salience; information that the domain expert has put in the summary is most salient. By matching units of text in the input document to corresponding text units in the summary, we label the document text units with a binary judgement of salience (see §3.3.1 for details).

If we set the basic unit of text to be a single sentence and we obtain binary salience judgements in the manner described above, we can model sentence extractive single document summarization as a sequence labeling task (Conroy and O'Leary, 2001). In this formulation, a document is a sequence of sentences, and the task objective is to predict the salience judgment for each sentence. In the simplest of settings, the actual extract summary can be formed by concatenating the sentences labeled as salient.

We refer to the probability of a sentence being labeled as salient as the salience estimate. Historically, most machine learning based methods for salience estimation have use feature-based representations of text units to make salience estimates. Typically, these features make use of word level frequency data (Nenkova and Vanderwende, 2005), information theoretic notions of surprise or topicality (Lin and Hovy, 2000; Daumé III and Marcu, 2006; Louis and Nenkova, 2013; Louis, 2014), as well as position based features (e.g., is the unit of text in the beginning, middle, or end of the document?) (Kupiec et al., 1995; Radev et al., 2000; Conroy and O'Leary, 2001), which are often correlated with human judgements of salience (Nenkova, 2005).

The field of summarization has undergone a revolution driven by the recent popularization of deep learning based models in NLP. Deep learning models have demonstrated empirical successes, achieving state-of-the-art performance in both extractive (Cheng and Lapata, 2016; Nallapati et al., 2017) and abstractive summarization settings (Nallapati et al., 2016; See et al., 2017; Zhang et al., 2019a). Deep learning models also naturally allow for learning hierarchical representations of word, sentence, and document level contexts when performing end-to-end training on the summarization task. However, exactly what kind of information is captured in these representations and how that information affects downstream salience estimation has not been experimentally verified.

In this chapter, we systematically compare several supervised deep learning models of sentence extractive single document summarization. As in prior work, we model a document hierarchically: a document is a sequence of sentences and a sentence is a sequence of words. Each summarization model consists of three layers or modules:

1. The word embedding layer, which maps sequences of words to sequences of fixed dimensional embeddings.

2. The sentence encoder layer, which maps sequences of word embeddings to a sentence embedding.

3. The sentence extractor layer, which maps sequences of sentence embeddings to a sequence of salience judgements.

We systematically compare three different architectures for the sentence encoder and four different sentence extractor architectures. Additionally, we also measure the effect of using fixed pretrained embeddings versus fine-tuning embeddings while training the rest the of model. Various configurations of encoder and extractor modules correspond to both prior work by Cheng and Lapata (2016) or Nallapati et al. (2017) as well as novel summarization models. While prior works have primarily used autoregressive sentence extractor architectures, we propose two non-autoregressive sentence extractors. We evaluate these models across a range of domains including large and small news domains, as well as personal stories, meetings, and medical research articles.

Additionally, we systematically ablate the inputs to models during training to better understand what surface level features are being used to make predictions. Words are tagged with a part of speech tagger and different word classes are replaced with special *unknown* tokens. We can then compare performance of the summarization model with and without access to specific classes of word features (e.g., nouns or verbs). To ablate the implicit effects of sentence position, we compare models trained on the original document to the same model trained on documents with shuffled sentence order. By removing content and position features, we can see their relative impact in the decrease in ROUGE scores on the test set. Moreover, these ablations give us a more intuitive understanding of how models will behave in novel environments. For example, if we know position is an important feature for a model, using it on data that is not position biased will likely result in poor performance.

Our main results reveal:

1. Sentence position bias dominates the learning signal for news summarization, though not for other domains. Summary quality for news is only slightly degraded when content words are omitted from sentence embeddings.

2. Word embedding averaging is as good or better than either recurrent or convolutional encoders for sentence embeddings across all domains.

3. Pre-trained word embeddings are as good, or better than, learned embeddings in five of six

datasets.

4. Non auto-regressive sentence extraction performs as good or better than auto-regressive extraction in all domains.

Taken together, these and other results in the paper suggest that we are over-estimating the ability of deep learning models to learn robust and meaningful content features for summarization. In one sense, this might lessen the burden of applying neural network models of content to other domains; one really just needs in-domain word embeddings. However, if we want to learn something other than where the start of the article is, we will need to design other means of sentence representation, and possibly external knowledge representations, better suited to the summarization task.

3.1 Problem Definition

We now formally define the sentence extractive, single document summarization task as a sequence tagging problem, following Conroy and O'Leary (2001). Let a document $x \in \mathcal{X}$ be a sequence of n sentences,

$$x = [s_1, s_2, \ldots, s_n] \, .$$

Sentences are themselves sequences of words,

$$s_i = \left[w_{i,1}, w_{i,2}, \ldots, w_{i,l_i}\right] ,$$

where $l_i \in \mathbb{N}$ is the length of sentence s_i in words. The words themselves are drawn from a finite vocabulary $\mathcal{V}$.

The binary salience of a sentence s_i is $y_i \in \{0, 1\}$. $y_i = 1$ indicates that sentence s_i is salient and should be included in the extract summary while $y_i = 0$ is assigned to non-salient sentences that should be excluded from the summary. We indicate the vector of salience judgements for the n sentences in x as $\boldsymbol{y} = [y_1, \ldots, y_n] \in \mathcal{Y}$. The objective of this sequence tagging problem is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which maps a document x to a sequence of salience labels $\boldsymbol{y}$. In this work, we learn a probabilistic mapping $P(\boldsymbol{y}|x;\theta)$ where P is a neural network with parameters θ and $P(\cdot|x;\theta) : \mathcal{Y} \rightarrow (0, 1)$.

Prediction is achieved by finding $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} P(\boldsymbol{y}|x;\theta)$, either by approximation or when the structure of P allows, exactly. Additionally, a typical constraint on summarization is that the extract summary not exceed a word budget $c \in \mathbb{N}$, that is, $\sum_{i=1}^{n} \hat{y}_i \cdot l_i \leq c$. Since it is not trivial to incorporate this constraint into the sequence labeling formulation, we instead rely on a greedy heuristic to enforce the budget constraint in practice. More details on test time inference can be found in §3.2.4.

3.2 Models

We implement our salience estimation model $P(\boldsymbol{y}|x;\theta)$ hierarchically following the analogous structure of the documents we are modeling. Every model P proposed in this chapter consists of three modules or layers: *(i)* the word embedding layer, *(ii)* the sentence encoder layer, and *(iii)* the sentence extractor layer. The word embedding layer maps the words in a sentence to a sequence of word embeddings. The sentence encoder layer similarly maps sequences of word embeddings to a sentence embedding. Finally, the sentence extractor maps sequences of sentence embeddings to sequence of salience labels.

Choosing an architecture for each of the modules defines the model. We define several architectures for the sentence encoder and extractor layers and show how particular settings of each correspond to prior summarization models proposed by Cheng and Lapata (2016) and Nallapati et al. (2017). Additionally, we propose two novel sentence extractor layers, and in experiments consider all combinations of sentence encoder/extractor pairings. In the next subsections, we describe each layer in more detail, and conclude this section showing how certain configurations of each layer maps to previously proposed or novel salience estimation models and how the models generate an extract summary at test time (which we refer to as inference).

3.2.1 Word Embedding Layer

The word embedding layer, $\mathrm{Emb}(\cdot\,; \boldsymbol{\Lambda}) : \mathcal{V}^* \rightarrow \mathbb{R}^{* \times D_w}$, maps a sequence of words

$$s_i = \left[w_{i,1}, \ldots, w_{i,l_i}\right]$$

to a sequence of word embeddings

$$\mathbf{V}_i = \left[\mathbf{v}_{i,1}, \ldots, \mathbf{v}_{i,l_i}\right] \in \mathbb{R}^{l_i \times D_w}$$

where the sole parameter $\boldsymbol{\Lambda} \in \mathbb{R}^{|\mathcal{V}| \times D_w}$ is a D_w-dimensional embedding matrix and $\mathbf{v}_{i,j} = \boldsymbol{\Lambda}_{w_{i,j}}$ is the word embedding for word $w_{i,j}$. $\boldsymbol{\Lambda}$ is initialized prior to training the full model with embeddings obtained using the unsupervised Global Vector (GloVe) embedding method on a large collection of text (Pennington et al., 2014). Additionally, $\boldsymbol{\Lambda}$ can be held fixed during training or updated with other model parameters. We use $D_w = 200$-dimensional embeddings in our models.

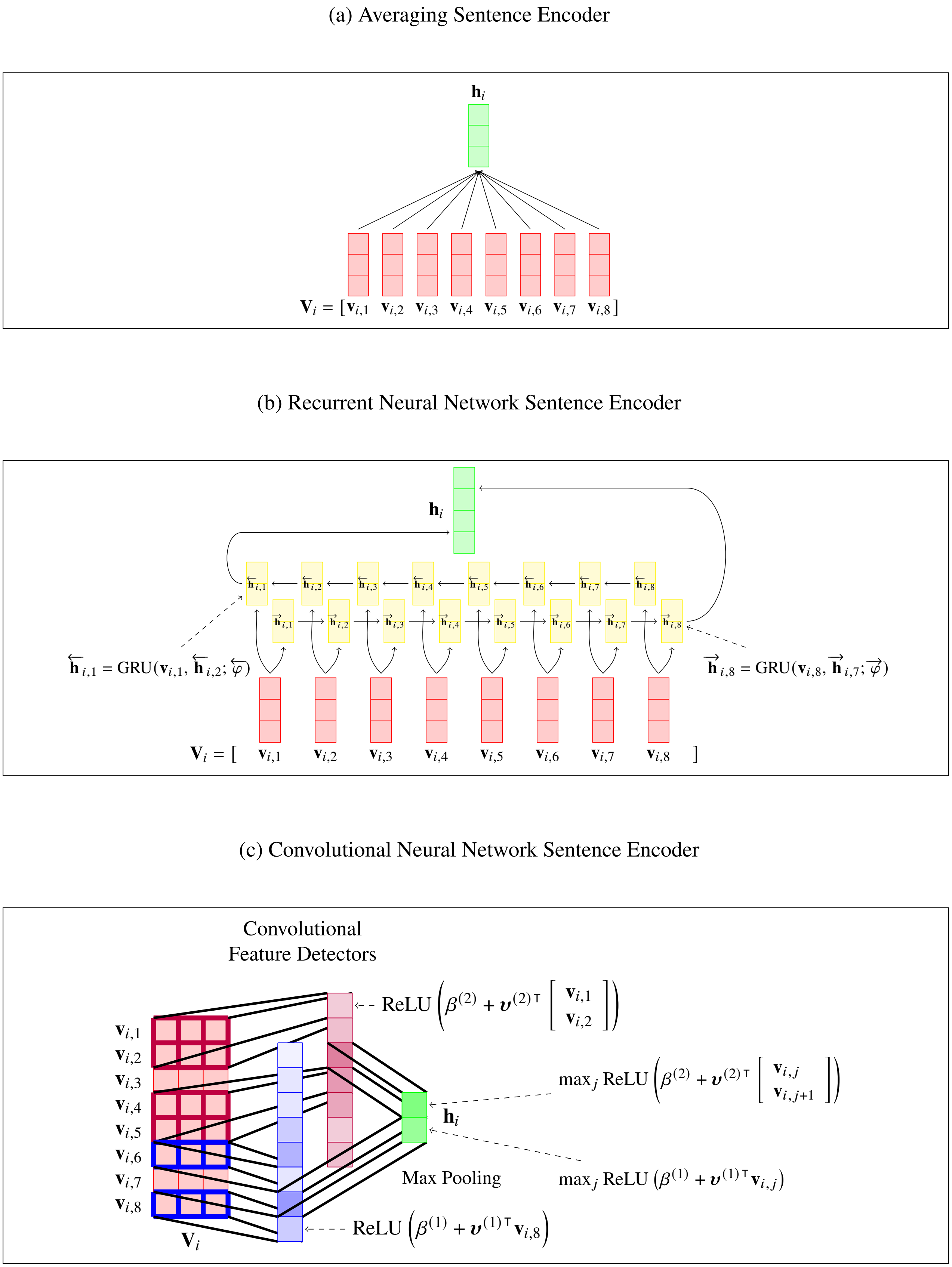
3.2.2 Sentence Encoder Layer

The sentence encoder layer, $\mathrm{SentEnc}(\cdot\,; \xi) : \mathbb{R}^{* \times D_w} \rightarrow \mathbb{R}^{D_s}$, maps a sequence of word embeddings

$$\mathbf{V}_i = \left[\mathbf{v}_{i,1}, \ldots, \mathbf{v}_{i,l_i}\right]$$

to a D_s-dimensional embedding representation of sentence s_i. The set of associated parameters, ξ, depends on the exact architecture for implementing the encoder. We experiment with three architectures for mapping sequences of word embeddings to a fixed length vector: averaging, recurrent neural networks, and convolutional neural networks.

We describe each variant now, and also briefly discuss the trade-offs associated with each architecture. The main distinction amongst the encoders is to how they can exploit the context and structure of the words they are encoding. With the averaging encoder, the resulting sentence embedding captures all words equally, but is insensitive to phrase structure phenomenon like negation.


(a) Averaging Sentence Encoder

(b) Recurrent Neural Network Sentence Encoder

(c) Convolutional Neural Network Sentence Encoder

$\mathbf{h}_i$
$\mathbf{V}_i = [\mathbf{v}_{i,1}\ \mathbf{v}_{i,2}\ \mathbf{v}_{i,3}\ \mathbf{v}_{i,4}\ \mathbf{v}_{i,5}\ \mathbf{v}_{i,6}\ \mathbf{v}_{i,7}\ \mathbf{v}_{i,8}]$
$\mathbf{h}_i$
$\overleftarrow{\mathbf{h}}_{i,1}$ $\overleftarrow{\mathbf{h}}_{i,2}$ $\overleftarrow{\mathbf{h}}_{i,3}$ $\overleftarrow{\mathbf{h}}_{i,4}$ $\overleftarrow{\mathbf{h}}_{i,5}$ $\overleftarrow{\mathbf{h}}_{i,6}$ $\overleftarrow{\mathbf{h}}_{i,7}$ $\overleftarrow{\mathbf{h}}_{i,8}$
$\overrightarrow{\mathbf{h}}_{i,1}$ $\overrightarrow{\mathbf{h}}_{i,2}$ $\overrightarrow{\mathbf{h}}_{i,3}$ $\overrightarrow{\mathbf{h}}_{i,4}$ $\overrightarrow{\mathbf{h}}_{i,5}$ $\overrightarrow{\mathbf{h}}_{i,6}$ $\overrightarrow{\mathbf{h}}_{i,7}$ $\overrightarrow{\mathbf{h}}_{i,8}$
$\overleftarrow{\mathbf{h}}_{i,1} = \mathrm{GRU}(\mathbf{v}_{i,1}, \overleftarrow{\mathbf{h}}_{i,2}; \overleftarrow{\varphi})$
$\overrightarrow{\mathbf{h}}_{i,8} = \mathrm{GRU}(\mathbf{v}_{i,8}, \overrightarrow{\mathbf{h}}_{i,7}; \overrightarrow{\varphi})$
$\mathbf{V}_i = [\ \mathbf{v}_{i,1}\ \mathbf{v}_{i,2}\ \mathbf{v}_{i,3}\ \mathbf{v}_{i,4}\ \mathbf{v}_{i,5}\ \mathbf{v}_{i,6}\ \mathbf{v}_{i,7}\ \mathbf{v}_{i,8}\ ]$
Convolutional
Feature Detectors
$\mathrm{ReLU}\left(\beta^{(2)} + \boldsymbol{\upsilon}^{(2)\intercal} \begin{bmatrix} \mathbf{v}_{i,1} \\ \mathbf{v}_{i,2} \end{bmatrix}\right)$
$\max_j \mathrm{ReLU}\left(\beta^{(2)} + \boldsymbol{\upsilon}^{(2)\intercal} \begin{bmatrix} \mathbf{v}_{i,j} \\ \mathbf{v}_{i,j+1} \end{bmatrix}\right)$
$\mathbf{h}_i$
Max Pooling
$\max_j \mathrm{ReLU}\left(\beta^{(1)} + \boldsymbol{\upsilon}^{(1)\intercal}\mathbf{v}_{i,j}\right)$
$\mathbf{v}_{i,1}$
$\mathbf{v}_{i,2}$
$\mathbf{v}_{i,3}$
$\mathbf{v}_{i,4}$
$\mathbf{v}_{i,5}$
$\mathbf{v}_{i,6}$
$\mathbf{v}_{i,7}$
$\mathbf{v}_{i,8}$
$\mathbf{V}_i$
$\mathrm{ReLU}\left(\beta^{(1)} + \boldsymbol{\upsilon}^{(1)\intercal}\mathbf{v}_{i,8}\right)$

Figure 3.1: Schematics for the averaging, recurrent neural network, and convolutional neural network sentence encoders.

The convolutional encoder can capture local phrase structure but may not be able to capture long range phrase structure. The recurrent encoder can capture long range phrase structure and context but is computationally more expensive than the recurrent encoder. Schematics of each encoder architecture can be found in Figure 3.1.

3.2.2.1 Averaging Sentence Encoder

Under the averaging encoder, a sentence embedding $\mathbf{h}_i \in \mathbb{R}^{D_s}$ is simply the average of its word embeddings,

$$\mathbf{h}_i = \operatorname{SentEnc}(\mathbf{V}_i; \xi) = \frac{1}{l_i} \sum_{j=1}^{l_i} \mathbf{v}_{i,j}. \tag{3.1}$$

The sentence representation here is effectively a bag of word embeddings. There are no parameters associated with this encoder (i.e. $\xi = \emptyset$). The size of the sentence embedding is simply $D_s = D_w = 200$. Dropout with drop probability 0.25 is applied to each word embedding $\mathbf{v}_{i,j}$ during training.

3.2.2.2 Recurrent Neural Network Sentence Encoder

When using the recurrent neural network encoder we apply both forward and backward recurrent neural networks over the word embedding sequences produced by the embedding layer. To obtain the actual sentence embedding, we concatenate the final output step of the forward and backward networks. For the actual recurrence function, we use the gated recurrent unit (GRU)

(Cho et al., 2014). The GRU function, $\mathrm{GRU}(\cdot, \cdot; \varphi) : \mathbb{R}^{D_w} \times \mathbb{R}^{D_h} \rightarrow \mathbb{R}^{D_h}$, is defined as

$$\mathrm{GRU}(\mathbf{v}, \mathbf{h}; \varphi) = (1 - \mathbf{u}) \odot \mathbf{o} + \mathbf{u} \odot \mathbf{h} \tag{3.2}$$

(Reset gate)

$$\mathbf{r} = \sigma\left(\mathbf{W}^{(vr)}\mathbf{v} + \mathbf{b}^{(vr)} + \mathbf{W}^{(hr)}\mathbf{h} + \mathbf{b}^{(hr)}\right) \tag{3.3}$$

(Update gate)

$$\mathbf{u} = \sigma\left(\mathbf{W}^{(vu)}\mathbf{v} + \mathbf{b}^{(vu)} + \mathbf{W}^{(hu)}\mathbf{h} + \mathbf{b}^{(hu)}\right) \tag{3.4}$$

(Candidate state)

$$\mathbf{o} = \tanh\left(\mathbf{W}^{(vo)}\mathbf{v} + \mathbf{b}^{(vo)} + \mathbf{r} \odot \left(\mathbf{W}^{(ho)}\mathbf{h} + \mathbf{b}^{(ho)}\right)\right) \tag{3.5}$$

where $\varphi = \left\{\mathbf{W}^{(ab)}, \mathbf{b}^{(ab)} \middle| a \in \{v, h\}, b \in \{r, u, o\}\right\}$ is the set of GRU parameters with $\mathbf{W}^{(v\cdot)} \in \mathbb{R}^{D_h \times D_w}$, $\mathbf{W}^{(h\cdot)} \in \mathbb{R}^{D_h \times D_h}$, and $\mathbf{b}^{(\cdot)} \in \mathbb{R}^{D_h}$, and $\sigma(x) = \frac{1}{1+e^{-x}}$, $\tanh(x) = \frac{e^x - 1}{e^x + 1}$, and $\odot$ is the Hadamard product.

Under the recurrent neural network encoder, a sentence embedding $\mathbf{h}_i$ is then defined as

$$\mathbf{h}_i = \mathrm{SentEnc}\left(\mathbf{V}_i; \xi\right) = \begin{bmatrix} \overrightarrow{\mathbf{h}}_{i,l_i} \\ \overleftarrow{\mathbf{h}}_{i,1} \end{bmatrix} \tag{3.6}$$

(Forward GRU)

$$\overrightarrow{\mathbf{h}}_{i,0} = \mathbf{0}, \tag{3.7}$$

$$\overrightarrow{\mathbf{h}}_{i,j} = \mathrm{GRU}(\mathbf{v}_{i,j}, \overrightarrow{\mathbf{h}}_{i,j-1}; \overrightarrow{\varphi}) \qquad \forall j \in \{1, \ldots, l_i\} \tag{3.8}$$

(Backward GRU)

$$\overleftarrow{\mathbf{h}}_{i,l_i+1} = \mathbf{0}, \tag{3.9}$$

$$\overleftarrow{\mathbf{h}}_{i,j} = \mathrm{GRU}(\mathbf{v}_{i,j}, \overleftarrow{\mathbf{h}}_{i,j+1}; \overleftarrow{\varphi}) \qquad \forall j \in \{l_i, \ldots, 1\} \tag{3.10}$$

where $[\cdots]$ is the vector concatenation operator and $\overrightarrow{\varphi}$ and $\overleftarrow{\varphi}$ are distinct parameters for the forward and backward GRUs respectively. Collectively the set of parameters for the recurrent neural network sentence encoder is $\xi = \left\{\overrightarrow{\varphi}, \overleftarrow{\varphi}\right\}$. We use $D_h = 300$ dimensional hidden layers for each GRU, making the size of the sentence embedding $D_s = 2D_h = 600$. Dropout with drop probability 0.25 is applied to GRU outputs $\overrightarrow{\mathbf{h}}_{i,j}$ and $\overleftarrow{\mathbf{h}}_{i,j}$ for $j \in \{1, \ldots, l_i\}$ during training.

3.2.2.3 Convolutional Neural Network Sentence Encoder

The convolutional neural network sentence encoder uses a series of convolutional feature maps to encode each sentence. This encoder is similar to the convolutional architecture of Kim (2014) used for text classification tasks. It performs a series of "one-dimensional" convolutions over word embeddings. The kernel width $k \in \mathbb{N}$ of a feature map determines the number of contiguous words that a feature map is sensitive to. For $k = 3$, for example, the feature map would function as a trigram feature detector essentially. We denote a single convolutional feature map of kernel width k as $f_k : \mathbb{R}^{* \times D_w} \to \mathbb{R}$ with

$$f_k(\mathbf{v}_i; \upsilon, \beta) = \max_{j \in \{1 - \lfloor \frac{k}{2} \rfloor, \ldots, l_i + \lfloor \frac{k}{2} \rfloor - k + 1\}} \operatorname{ReLU}\left(\beta + \boldsymbol{\upsilon} \cdot \begin{bmatrix} \mathbf{v}_{i,j} \\ \mathbf{v}_{i,j+1} \\ \vdots \\ \mathbf{v}_{i,j+k-1} \end{bmatrix}\right), \tag{3.11}$$

where $\operatorname{ReLU}(x) = \max(0, x)$ is the rectified linear unit (Nair and Hinton, 2010), $\lfloor \cdot \rfloor$ is the floor operator, and $\boldsymbol{\upsilon} \in \mathbb{R}^{kD_w}$ and $\beta \in \mathbb{R}$ are learned parameters. Note that we use a "zero-padded" convolution (Dumoulin and Visin, 2016). That is, the max operator ranges over $j \in \left\{1 - \left\lfloor \frac{k}{2} \right\rfloor, \ldots, l_i + \left\lfloor \frac{k}{2} \right\rfloor - k + 1\right\}$ instead of $\{1, \ldots, l_i - k + 1\}$, and $\mathbf{v}_{i,j} = \mathbf{0}$ for $j < 1$ and $j > l_i$. Padded convolutions help alleviate the problem of reduced receptive fields on the boundaries of the sequence. See Figure 3.2 for a visual example.

The final sentence embedding $\mathbf{h}_i$ is a concatenation of many convolutional feature maps ranging over multiple kernel widths with each filter having its own distinct sets of parameters. Let $\mathcal{K} = \{k_1, \ldots, k_m\} \subset \mathbb{N}$ be the set of the sentence encoder's m kernel widths, and $D_k \in \mathbb{N}$ be the number of feature maps for kernel width k. The final sentence embedding produced by the convolutional

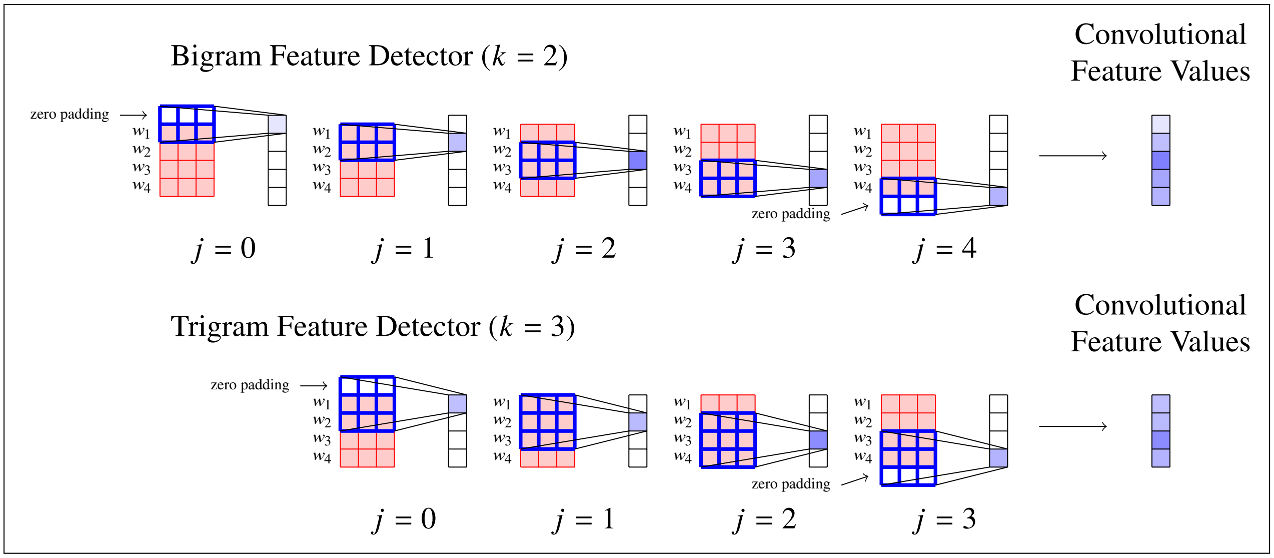

Bigram Feature Detector ($k = 2$)
Convolutional Feature Values
zero padding
w_1
w_2
w_3
w_4
$j = 0$
$j = 1$
$j = 2$
$j = 3$
$j = 4$
Trigram Feature Detector ($k = 3$)
Convolutional Feature Values
$j = 0$
$j = 1$
$j = 2$
$j = 3$

Figure 3.2: Examples of zero padding with bigram ($k = 2$) and trigram ($k = 3$) features for a sequence of length $l_i = 4$.

neural network sentence encoder is defined as

$$\mathbf{h}_i = \text{SentEnc}(\mathbf{v}_i; \xi) = \left[f_{k_1}^{(1)}, \ldots, f_{k_1}^{(D_{k_1})}, f_{k_2}^{(1)}, \ldots, f_{k_2}^{(D_{k_2})}, \ldots, f_{k_m}^{(1)}, \ldots, f_{k_m}^{(D_{k_m})} \right] \tag{3.12}$$

where $f_k^{(j)} = f_k^{(j)}(\mathbf{V}_i; \boldsymbol{\upsilon}^{(j,k)}, \beta^{(j,k)})$ and $\xi = \left\{ \boldsymbol{\upsilon}^{(k,l)}, \beta^{(k,l)} \middle| \forall k, l : k \in \mathcal{K}, l \in \{1, \ldots, D_k\} \right\}$ are the sentence encoder's learned parameters. In our instantiation, we use kernel widths $\mathcal{K} = \{1, \ldots, 6\}$ with corresponding feature maps sizes $D_1 = 25$, $D_2 = 25$, $D_3 = 50$, $D_4 = 50$, $D_5 = 50$, and $D_6 = 50$, making the resulting sentence embedding dimensionality $D_s = 250$. Dropout with drop probability 0.25 is also applied to $\mathbf{h}_i$ during training.

3.2.2.4 Sentence Encoder Trade-offs

The sentence encoder's role is to obtain a vector representation of a finite sequence of word embeddings that is useful for the sentence extraction stage. Therefore, it must aggregate features in the word embedding space that are predictive of salience. Averaging embeddings is not an unreasonable approach to this. Empirically there is evidence that word embedding averaging is a fairly competitive sentence representation generally (Iyyer et al., 2015; Wieting et al., 2016; Arora et al., 2017; Wieting and Gimpel, 2017). In the context of summarization, averaging can be thought of as a noisy OR; if any of the words in a sentence are indicative of salience, this representation should capture them. Computationally, the averaging encoder is the fastest to compute and does not require learning of parameters, reducing the memory and computation time during training.

The recurrent neural network sentence encoder can in theory capture some compositional features of a word sequence that would be difficult or impossible to represent in the averaging encoder (e.g. negation or co-reference). However, this comes at a much heavier computational cost, as recurrent neural networks cannot be fully parallelized due to the inherently sequential nature of their computation.

The convolutional neural network encoder represents a middle ground between the averaging and recurrent neural network encoders. When using modestly sized kernel widths (e.g., 1-5), the receptive window should be sensitive short phrases and some locally scoped negation. It will not be able to capture the longer ranged dependencies that the recurrent neural network encoder would. However, it is much faster to compute than the recurrent neural network as the individual feature maps can be computed completely in parallel.

3.2.3 Sentence Extraction Layer

The role of the sentence extractor, $\operatorname{SentExt}(\cdot;\chi) : \mathbb{R}^{*\times D_s} \rightarrow \mathcal{Y}$, is to map a sequence of sentence embeddings $\mathbf{h}_1, \ldots, \mathbf{h}_n$ produced by the sentence encoder layer to a sequence of salience judgements

$$\boldsymbol{y} = [y_1, \ldots, y_n] \,.$$

The proposed sentence extractors do this by first implementing a probability distribution over salience label sequences conditioned on $\mathbf{h}_1, \ldots, \mathbf{h}_n$, $P(\boldsymbol{y}|\mathbf{h}_1, \ldots, \mathbf{h}_n; \chi)$, and then inferring the (approximate) maximum likelihood sequence, i.e.,

$$\operatorname{SentExt}(\mathbf{h}_1, \ldots, \mathbf{h}_n; \chi) = \hat{\boldsymbol{y}} \approx \operatorname*{arg\,max}_{\boldsymbol{y} \in \mathcal{Y}} P(\boldsymbol{y}|\mathbf{h}_1, \ldots, \mathbf{h}_n; \chi).$$

Previous neural network approaches to sentence extraction have assumed an autoregressive model, leading to the following factorization of the salience label distribution

$$P(y_1, \ldots, y_n | \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi) = \prod_{i=1}^{n} P(y_i | y_1, \ldots, y_{i-1}, \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi),$$

where each prediction y_i is dependent on *all* previous y_j for all $j < i$. We compare two such models proposed by Cheng and Lapata (2016) and Nallapati et al. (2017).

While intuitively it makes sense that previous extraction decisions might affect the probability of extracting subsequent sentences, (e.g., highly salient sentences might cluster together), it has not been empirically investigated whether this dependence is necessary for deep learning models in practice. For example, in the models of Cheng and Lapata (2016) and Nallapati et al. (2017), individual predictions of y_i are made using information from some or all of the sentence embeddings $\mathbf{h}_1, \ldots, \mathbf{h}_n$, such that information about neighboring sentences could be propagated through sentence embedding interactions rather than on previous salience decisions. Additionally, from an efficiency perspective, the autoregressive design prevents parallelization of individual y_i predictions, since they must now be sequentially computed. Motivated by these considerations, we propose two non-autoregressive sentence extractor architectures where individual salience labels y_i are independent of each other, that is,

$$P(y_1, \ldots, y_n | \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi) = \prod_{i=1}^{n} P(y_i | \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi).$$

While the sentence extractor architectures are quite different in their details, under our unified treatment of them here, we view them as producing in their intermediate computations a sequence of contextual sentence embeddings $\mathbf{z}_1, \ldots, \mathbf{z}_n$. Unlike the "context free" sentence embeddings $\mathbf{h}_i$ which are constructed only using words from sentence s_i, each $\mathbf{z}_i$ is computed using information information propagated from neighboring sentence embeddings $\mathbf{h}_1, \ldots, \mathbf{h}_{i-1}$ and $\mathbf{h}_{i+1}, \ldots, \mathbf{h}_n$ and

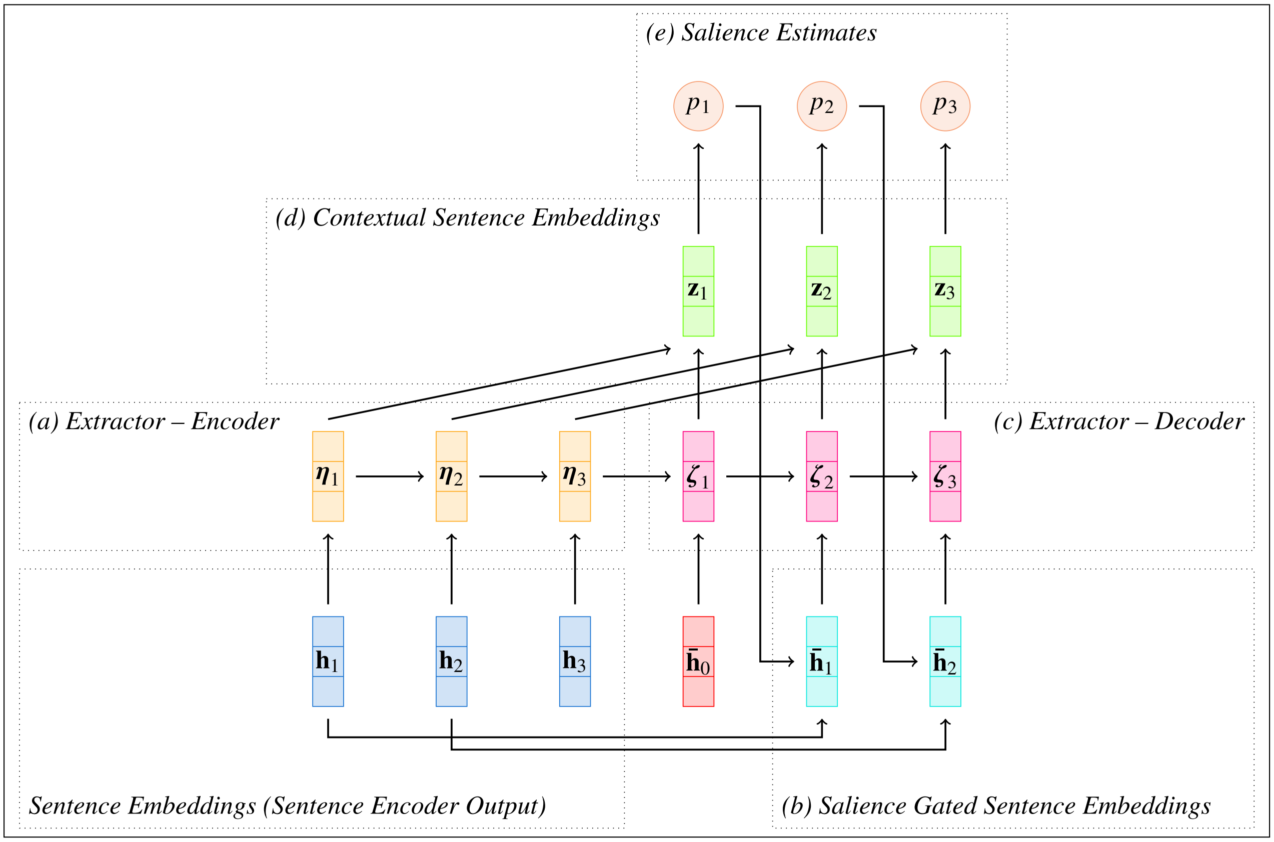


Figure 3.3: Schematic for the Cheng & Lapata sentence extractor.

in the case of the autoregressive models, salience estimates $p_1, \ldots, p_{i-1}$ where

$$p_i = P(y_i = 1 | y_1, \ldots, y_{i-1}, \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi).$$

Additionally, the SummaRunner extractor produces an embedding representation of the document, $\mathbf{d}$, as well as iterative representations of the summary $\mathbf{s}_1, \ldots, \mathbf{s}_{n-1}$ which also affect the creation of the contextual sentence embeddings.

We now describe in detail how the autoregressive sentence extractors (the Cheng & Lapata extractor and the SummaRunner extractor) and our proposed non-autoregressive ones (the RNN extractor and the Seq2Seq extractor), produce the these various representations and make sentence salience estimates.

3.2.3.1 Cheng & Lapata Extractor

The Cheng & Lapata extractor (Cheng and Lapata, 2016) is built around a somewhat idiosyncratic unidirectional sequence-to-sequence model. A schematic outlining the structure of the encoder and closely following the subsequent equations can be found in Figure 3.3.

The encoder is fairly standard. The initial state is initialized to a zero embedding, **0**, and each sentence embedding $\mathbf{h}_i$ is fed into the encoder, to obtain the final encoder hidden state $\boldsymbol{\eta}_n \in \mathbb{R}^{D_x}$. That is,

(Figure 3.3.a) Extractor – Encoder

$$\boldsymbol{\eta}_0 = \mathbf{0} \tag{3.13}$$

$$\boldsymbol{\eta}_i = \text{GRU}(\mathbf{h}_i, \boldsymbol{\eta}_{i-1}; \varphi_\eta) \qquad \forall i : i \in \{1, \ldots, n\}. \tag{3.14}$$

The initial decoder hidden state $\boldsymbol{\zeta}_0 \in \mathbb{R}^{D_x}$ is initialized with the last encoder hidden state, $\boldsymbol{\eta}_n$. The inputs to decoder step i, for $i > 1$, are the salience gated $(i-1)^{\text{th}}$ sentence embeddings,

(Figure 3.3.b) Salience Gated Sentence Embeddings

$$\bar{\mathbf{h}}_{i-1} = p_{i-1}\mathbf{h}_{i-1} \qquad \forall i : i \in \{2, \ldots, n\}, \tag{3.15}$$

where the salience gate is $p_i = P(y_i | y_1, \ldots, y_{i-1}, \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi)$, is the salience estimate computed for sentence s_{i-1}. For the first decoder step (i.e. $i = 1$), since there is no p_0, $\bar{\mathbf{h}}_0$ is a special learned parameter.

The extractor decoder outputs are then computed as,

(Figure 3.3.c) Extractor – Decoder

$$\boldsymbol{\zeta}_0 = \boldsymbol{\eta}_n \tag{3.16}$$

$$\boldsymbol{\zeta}_i = \text{GRU}(\bar{\mathbf{h}}_{i-1}, \boldsymbol{\zeta}_{i-1}; \varphi_\zeta) \qquad \forall i : i \in \{1, \ldots, n\} \tag{3.17}$$

Note in Equation 3.17 that the decoder side GRU input is the sentence embedding from the previous time step, $\mathbf{h}_{i-1}$, weighted by its probability of extraction, p_{i-1}, from the previous step, inducing dependence of each output y_i on all previous outputs $y_1, \ldots, y_{i-1}$.

The contextual sentence embeddings $\mathbf{z}_i$ are then computed by concatenating the encoder and decoder outputs $\boldsymbol{\eta}_i$ and $\boldsymbol{\zeta}_i$ and running them through a feed-forward layer with ReLU activation,

(Figure 3.3.d) Contextual Sentence Embeddings

$$\mathbf{z}_i = \operatorname{ReLU}\left(\mathbf{U}^{(1)} \begin{bmatrix} \boldsymbol{\eta}_i \\ \boldsymbol{\zeta}_i \end{bmatrix} + \mathbf{u}^{(1)}\right) \qquad \forall i : i \in \{1, \ldots, n\}. \tag{3.18}$$

The actual salience estimate for sentence s_i is then computed by feeding $\mathbf{z}_i$ through another feed-forward layer with logistic sigmoid activation,

(Figure 3.3.e) Salience Estimates

$$p_i = P(y_i = 1 | y_1, \ldots, y_{i-1}, \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi) = \sigma\left(\mathbf{U}^{(2)}\mathbf{z}_i + \mathbf{u}^{(2)}\right) \qquad \forall i : i \in \{1, \ldots, n\}. \tag{3.19}$$

The contextual embedding and salience estimate layers have parameters are $\mathbf{U}^{(1)} \in \mathbb{R}^{D_z \times 2D_\chi}$, $\mathbf{u}^{(1)} \in \mathbb{R}^{D_z}$, $\mathbf{U}^{(2)} \in \mathbb{R}^{1 \times D_z}$, and $\mathbf{u}^{(2)} \in \mathbb{R}$. The entire set of learned parameters for the Cheng & Lapata extractor are

$$\chi = \left\{\varphi_\eta, \varphi_\zeta, \bar{\mathbf{h}}_0, \mathbf{U}^{(1)}, \mathbf{u}^{(1)}, \mathbf{U}^{(2)}, \mathbf{u}^{(2)}\right\}.$$

The hidden layer dimensionality of the GRU and the contextual embedding layer is $D_\chi = 300$ and $D_z = 100$, respectively. Dropout with drop probability 0.25 is applied to the GRU outputs ($\boldsymbol{\eta}_i$ and $\boldsymbol{\zeta}_i$), and to $\mathbf{z}_i$.

3.2.3.2 SummaRunner Extractor

Nallapati et al. (2017) proposed a sentence extractor, which we refer to as the SummaRunner Extractor, that factorizes the salience estimates for each sentence into contributions from five

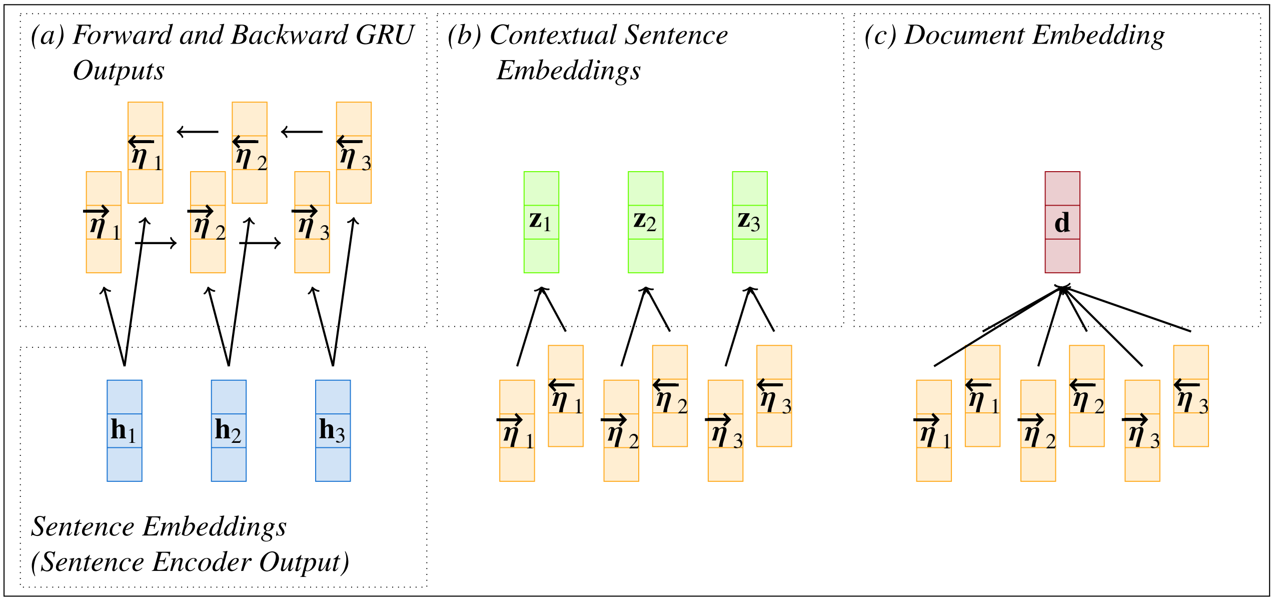


Figure 3.4: SummaRunner contextual sentence embedding and document embeddings.

different sources, which we refer to as salience factors. The salience factors take into account interactions between contextual sentence embeddings and document embeddings or summary embeddings, as well as sentence position embeddings. Salience estimates are made sequentially, starting with the first sentence s_1 and preceding to the last s_n. When computing the salience estimate of sentence s_i, the previous $i - 1$ salience estimates are used to update the summary representation.

In order to construct the contextual sentence embeddings, document embeddings, and summary embeddings, the SummaRunner extractor first runs a bidirectional GRU over the sentence embeddings created by the sentence encoder (visually depicted in Figure 3.4),

(Figure 3.4.a) Forward and Backward GRU Outputs

$$\overrightarrow{z}_0 = \mathbf{0}, \tag{3.20}$$

$$\overrightarrow{z}_i = \operatorname{GRU}(\mathbf{h}_i, \overrightarrow{z}_{i-1}; \overrightarrow{\varphi}) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.21}$$

$$\overleftarrow{z}_{n+1} = \mathbf{0}, \tag{3.22}$$

$$\overleftarrow{z}_i = \operatorname{GRU}(\mathbf{h}_i, \overleftarrow{z}_{i+1}; \overleftarrow{\varphi}) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.23}$$

where $\overrightarrow{z}_i, \overleftarrow{z}_i \in \mathbb{R}^{D_r}$ and $\overrightarrow{\varphi}$ and $\overleftarrow{\varphi}$ are the forward and backward GRU parameters respectively.

The GRU output is concatenated and run through a feed-forward layer to obtain a contextual sentence embedding representation $\mathbf{z}_i \in \mathbb{R}^{D_z}$,

(Figure 3.4.b) Contextual Sentence Embeddings

$$\mathbf{z}_i = \operatorname{ReLU}\left(\mathbf{u}^{(z)} + \mathbf{U}^{(z)} \begin{bmatrix} \overrightarrow{z}_i \\ \overleftarrow{z}_i \end{bmatrix}\right) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.24}$$

where $\mathbf{U}^{(z)} \in \mathbb{R}^{D_z \times 2D_r}$ and $\mathbf{u}^{(z)} \in \mathbb{R}^{D_z}$ are learned parameters.

To construct the document embedding $\mathbf{d}$, the forward and backward GRU outputs are concatenated and averaged before running through a different feed-forward layer,

(Figure 3.4.c) Document Embedding

$$\mathbf{d} = \tanh\left(\mathbf{u}^{(d)} + \mathbf{U}^{(d)}\left(\frac{1}{n}\sum_{i=1}^{n} \begin{bmatrix} \overrightarrow{z}_i \\ \overleftarrow{z}_i \end{bmatrix}\right)\right) \tag{3.25}$$

where $\mathbf{U}^{(d)} \in \mathbb{R}^{D_z \times 2D_r}$ and $\mathbf{u}^{(d)} \in \mathbb{R}^{D_z}$ are learned parameters.

Additionally, an iterative representation of the extract summary at step i, $\mathbf{s}_i$, is constructed by summing the $i-1$ contextual sentence embeddings weighted by their salience estimates,

(Figure 3.5) Summary Embeddings

$$\mathbf{s}_1 = \mathbf{0}, \tag{3.26}$$

$$\mathbf{s}_i = \tanh\left(\sum_{j=1}^{i-1} p_j \cdot \mathbf{z}_j\right) \qquad \forall i : i \in \{2, \ldots, n-1\}, \tag{3.27}$$

where $p_j = P\left(y_j = 1 | y_1, \ldots, y_{j-1}, \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi\right)$ are previously computed salience estimates for sentences $s_1, \ldots, s_{i-1}$.

When computing the salience of a sentence p_i, the SummaRunner model uses the contextual

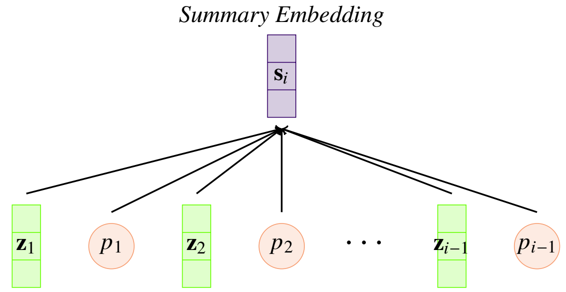


Figure 3.5: SummaRunner iterative summary embeddings.

sentence embedding $\mathbf{z}_i$, the document embedding $\mathbf{d}$, the summary embedding $\mathbf{s}_i$, and the document position i. These representations are used to compute five different factors, which are then summed and run through a logistic sigmoid to the salience estimate. The five factors are named the content factor ($\phi^{(c)}$), the centrality factor ($\phi^{(s)}$),[1] the novelty factor ($\phi^{(n)}$), the fine grained position factor ($\phi^{(fp)}$), and the coarse grained position factor ($\phi^{(cp)}$).

First, there is the content factor which is simply the dot product of the contextual sentence embedding with a learned parameter vector,

(Figure 3.6.a) Content Factor

$$\phi_i^{(c)} = \mathbf{u}^{(c)\intercal} \mathbf{z}_i \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.28}$$

where $\mathbf{u}^{(c)} \in \mathbb{R}^{D_z}$. This term is intended to represent contributions of the individual sentences to their salience.

The centrality factor is intended to capture a sentence's similarity to the document embedding. This interaction computed as

(Figure 3.6.b) Centrality Factor

$$\phi_i^{(s)} = \mathbf{z}_i^\intercal \mathbf{U}^{(s)} \mathbf{d} \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.29}$$

[1] Nallapati et al. (2017) refer to this as the salience factor, but we rename it here to avoid confusion with the model's final predictions which we call salience estimates.

and is mediated by a learned parameter matrix, $\mathbf{U}^{(s)} \in \mathbb{R}^{D_z \times D_z}$.

The third factor, novelty, is similarly an interaction between the contextual sentence embedding and the i-th summary embedding,

(Figure 3.6.c) Novelty Factor

$$\phi_i^{(n)} = -\mathbf{z}_i^\intercal \mathbf{U}^{(n)} \mathbf{s}_i \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.30}$$

where $\mathbf{U}^{(n)} \in \mathbb{R}^{D_z \times D_z}$ is a learned parameter matrix. Note also this factor is multiplied by negative to indicate that high levels of similarity between $\mathbf{z}_i$ and $\mathbf{s}_i$ should be discouraged.

Finally, there are two factors for the fine- and coarse-grained position,

(Figure 3.6.d) Fine-grained Position Factor

$$\phi^{(fp)} = \mathbf{u}^{(fp)^\intercal} \mathbf{g}_i^{(fp)} \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.31}$$

(Figure 3.6.e) Coarse-grained Position Factor

$$\phi^{(cp)} = \mathbf{u}^{(cp)^\intercal} \mathbf{g}_i^{(cp)} \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.32}$$

where $\mathbf{g}_i^{(fp)}$ and $\mathbf{g}_i^{(cp)}$ are embeddings associated with the sentence position and sentence position quartile of the i-th sentence (e.g., sentence s_7 in a document with 12 sentences, would have embeddings $\mathbf{g}_7^{(fp)}$ and $\mathbf{g}_2^{(cp)}$ corresponding to the seventh sentence position and 2nd sentence position quartile respectively). Both $\mathbf{u}^{(fp)}, \mathbf{u}^{(cp)} \in \mathbb{R}^{D_g}$, and $\mathbf{g}_1^{(fp)}, \ldots, \mathbf{g}_{n_{max}}^{(fp)}, \mathbf{g}_1^{(cp)}, \ldots, \mathbf{g}_4^{(cp)} \in \mathbb{R}^{D_g}$ are learned parameters of the SummaRunner extractor, and $n_{max} \in \mathbb{N}$ is the maximum document size in sentences (when handling unusually long documents, sentences with positions greater than n_{max} are all mapped to $\mathbf{g}_{n_{max}}^{(fp)}$).

Each salience estimate p_i is calculated as the sum of those five factors run through a logistic sigmoid function,

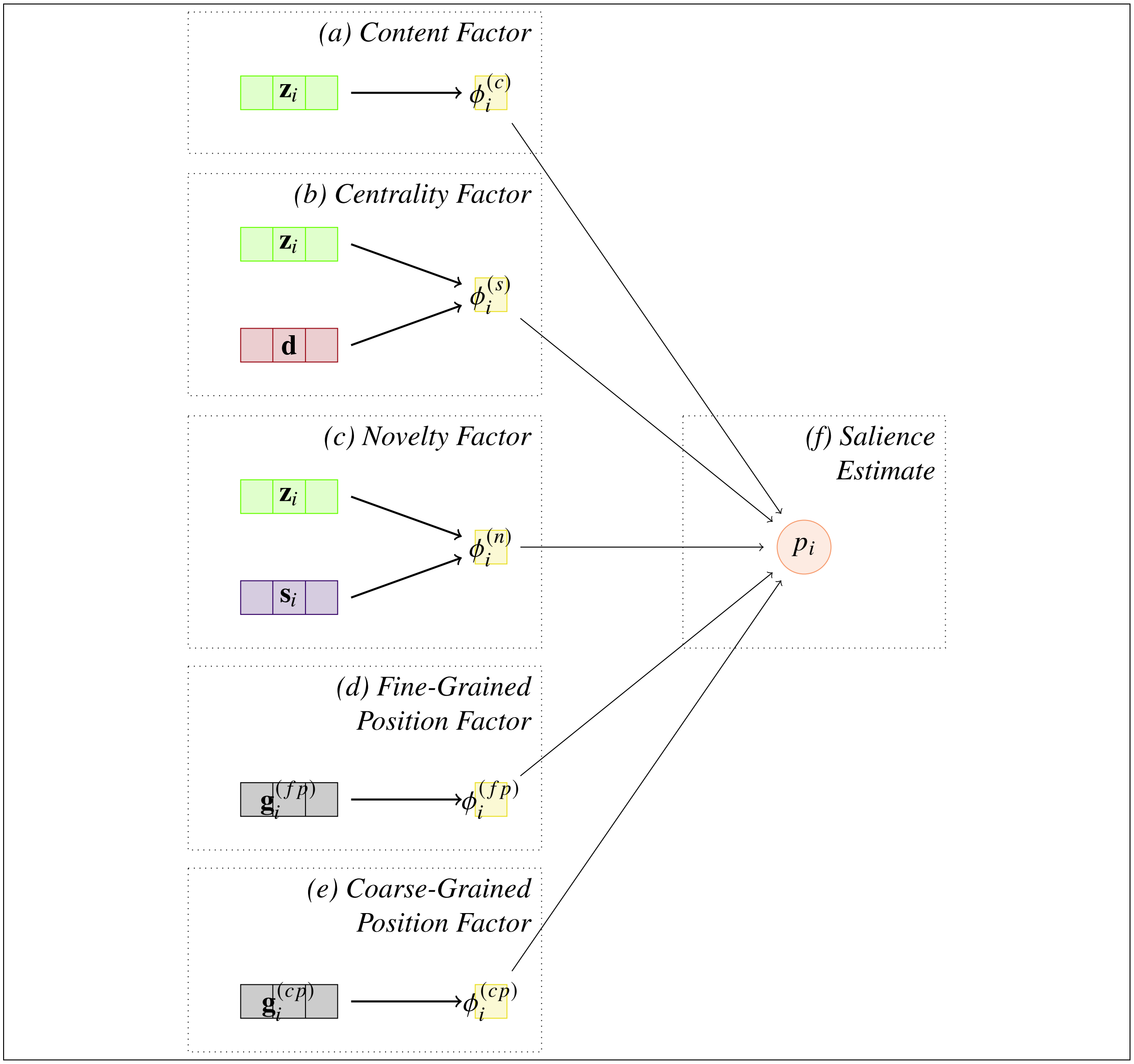


Figure 3.6: Schematic of salience estimation in the SummaRunner extractor.

(Figure 3.6.f) Salience Estimates

$$p_i = P(y_i = 1 | y_1, \ldots, y_{i-1}, \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi) = \sigma\left(\phi_i^{(c)} + \phi_i^{(s)} + \phi_i^{(n)} + \phi_i^{(fp)} + \phi_i^{(cp)}\right)$$
$$\forall i : i \in \{1, \ldots, n\}. \tag{3.33}$$

We should give a bit of caution about interpreting the individual factors along the lines of their given names as the learning procedure does not enforce that their actual values meaningfully correspond to their names. For example, it could be that a sentence's similarity to the summary embedding is actually positively correlated with high salience. In such a case, the model could learn to produce a large negative value for $\mathbf{z}_i^\intercal \mathbf{U}^{(n)} \mathbf{s}_i$, such that the "novelty" factor $\phi_i^{(n)}$ now has large positive values when the i-th sentence is not novel to the summary. In this sense, for a sufficiently over-parameterized network, the negative in novelty term is superfluous as the model can learn around it.

The complete set of parameters for the SummaRunner extractor is

$$\chi = \left\{\overrightarrow{\varphi}, \overleftarrow{\varphi}, \mathbf{U}^{(z)}, \mathbf{u}^{(z)}, \mathbf{U}^{(d)}, \mathbf{u}^{(d)}, \mathbf{u}^{(c)}, \mathbf{U}^{(s)}, \mathbf{U}^{(n)}, \mathbf{u}^{(fp)}, \mathbf{u}^{(cp)}, \mathbf{g}_1^{(fp)}, \ldots, \mathbf{g}_{n_{max}}^{(fp)}, \mathbf{g}_1^{(cp)}, \ldots, \mathbf{g}_4^{(cp)}, \right\}.$$

In our experiments, we set $D_r = 300$, $D_r = 100$, and $D_g = 16$. Dropout with drop probability of 0.25 is applied to the GRU outputs $\overrightarrow{z}_i$ and $\overleftarrow{z}_i$, as well as the contextual sentence embeddings $\mathbf{z}_i$ for all $i \in \{1, \ldots, n\}$.

3.2.3.3 RNN Extractor

Our first proposed non-autoregressive model is a very simple bidirectional recurrent neural network based tagging model (Graves and Schmidhuber, 2005; Wang et al., 2015), which we refer to as the RNN extractor. See Figure 3.7 for a visual depiction of the extractor. As in the SummaRunner extractor, the first step of the RNN extractor is to run a bidirectional recurrent neural network over the sentence embeddings produced by the sentence encoder layer, which produces

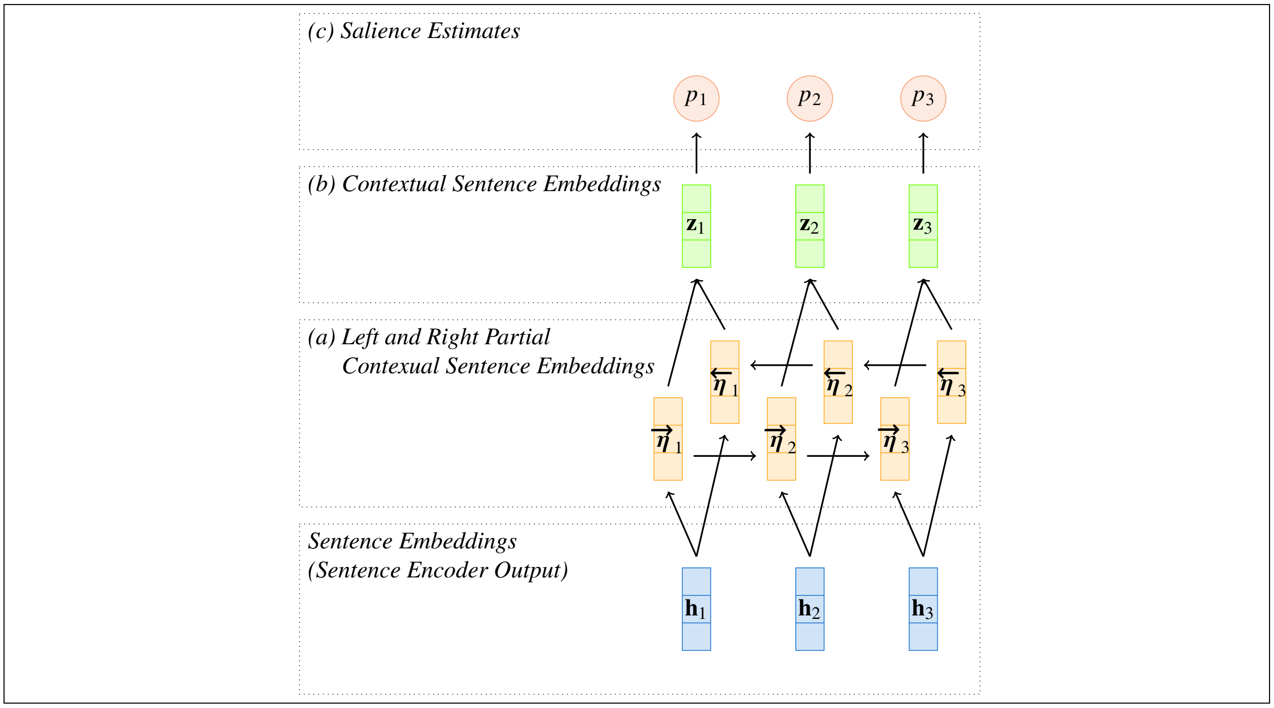

(c) Salience Estimates
p_1
p_2
p_3
(b) Contextual Sentence Embeddings
$\mathbf{z}_1$
$\mathbf{z}_2$
$\mathbf{z}_3$
(a) Left and Right Partial
Contexual Sentence Embeddings
$\overleftarrow{\boldsymbol{\eta}}_1$
$\overleftarrow{\boldsymbol{\eta}}_2$
$\overleftarrow{\boldsymbol{\eta}}_3$
$\overrightarrow{\boldsymbol{\eta}}_1$
$\overrightarrow{\boldsymbol{\eta}}_2$
$\overrightarrow{\boldsymbol{\eta}}_3$
Sentence Embeddings
(Sentence Encoder Output)
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$

Figure 3.7: Schematic for the RNN sentence extractor.

left and right partial contextual embeddings $\overrightarrow{\boldsymbol{\eta}}_i$ and $\overleftarrow{\boldsymbol{\eta}}_i$ respectively,

(Figure 3.7.a) Left and Right Partial Contexual Sentence Embeddings

$$\overrightarrow{\boldsymbol{\eta}}_0 = \mathbf{0}, \tag{3.34}$$

$$\overrightarrow{\boldsymbol{\eta}}_i = \mathrm{GRU}\left(\mathbf{h}_i, \overrightarrow{\boldsymbol{\eta}}_{i-1}; \overrightarrow{\varphi}\right) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.35}$$

$$\overleftarrow{\boldsymbol{\eta}}_{n+1} = \mathbf{0}, \tag{3.36}$$

$$\overleftarrow{\boldsymbol{\eta}}_i = \mathrm{GRU}\left(\mathbf{h}_i, \overleftarrow{\boldsymbol{\eta}}_{i+1}; \overleftarrow{\varphi}\right) \qquad \forall i : i \in \{n, \ldots, 1\}, \tag{3.37}$$

where $\overrightarrow{\boldsymbol{\eta}}_i, \overleftarrow{\boldsymbol{\eta}}_i \in \mathbb{R}^{D_\eta}$, and $\overrightarrow{\varphi}$ and $\overleftarrow{\varphi}$ are the forward and backward GRU parameters.

The left and right partial contextual embeddings of each sentence are then passed through a feed-forward layer to produce contextual sentence embeddings $\mathbf{z}_i$,

(Figure 3.7.b) Contexual Sentence Embeddings

$$\mathbf{z}_i = \mathrm{ReLU}\left(\boldsymbol{W}^{(1)}\begin{bmatrix}\overrightarrow{\boldsymbol{\eta}}_i\\ \overleftarrow{\boldsymbol{\eta}}_i\end{bmatrix} + \boldsymbol{b}^{(1)}\right) \qquad \forall i: \; i \in \{1, \ldots, n\}, \tag{3.38}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{D_z \times 2D_\eta}$ and $\boldsymbol{b}^{(1)} \in \mathbb{R}^{D_z}$ are learned parameters.

Another feed-forward layer with a logistic sigmoid activation computes the actual salience estimates $p_1, \ldots, p_n$ where $p_i = P(y_i = 1|\mathbf{h}_1, \ldots, \mathbf{h}_n; \chi)$ and

(Figure 3.7.c) Salience Estimates

$$p_i = P(y_i = 1|\mathbf{h}_i, \ldots, \mathbf{h}_n; \chi) = \sigma\left(\boldsymbol{W}^{(2)}\mathbf{z}_i + \boldsymbol{b}^{(2)}\right) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.39}$$

where $\boldsymbol{W}^{(2)} \in \mathbb{R}^{1 \times D_z}$ and $\boldsymbol{b}^{(2)} \in \mathbb{R}$ are learned parameters.

The complete set of parameters for the extractor is

$$\chi = \left\{\overrightarrow{\varphi}, \overleftarrow{\varphi}, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}\right\}.$$

In our experiments, we set $D_\eta = 300$ and $D_z = 100$. Dropout with drop probability of 0.25 is applied to $\overrightarrow{\boldsymbol{\eta}}_i$, $\overleftarrow{\boldsymbol{\eta}}_i$, and $\mathbf{z}_i$ for $i \in \{1, \ldots, n\}$.

3.2.3.4 Seq2Seq Extractor

One shortcoming of the RNN extractor is that long range information from one end of the document may not easily be able to affect extraction probabilities of sentences at the other end. Our second proposed model, the Seq2Seq extractor mitigates this problem with an attention mechanism commonly used for neural machine translation (Bahdanau et al., 2015; Luong et al., 2015) and abstractive summarization (See et al., 2017). The Seq2Seq extractor has distinct encoder and decoder bidirectional GRUs which create distinct sequences of encoder contextual embeddings and decoder contextual embeddings (see Figure 3.8 and Figure 3.9 respectively). Using the attention

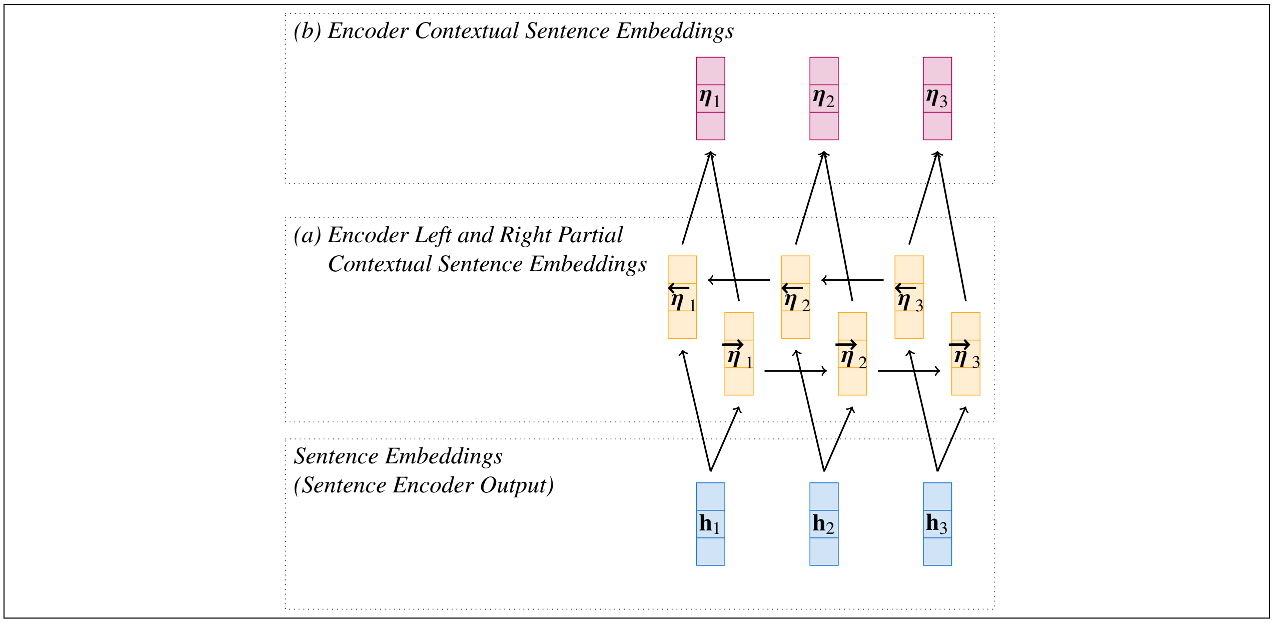

(b) Encoder Contextual Sentence Embeddings
$\boldsymbol{\eta}_1$
$\boldsymbol{\eta}_2$
$\boldsymbol{\eta}_3$
(a) Encoder Left and Right Partial Contextual Sentence Embeddings
$\overleftarrow{\boldsymbol{\eta}}_1$
$\overleftarrow{\boldsymbol{\eta}}_2$
$\overleftarrow{\boldsymbol{\eta}}_3$
$\overrightarrow{\boldsymbol{\eta}}_1$
$\overrightarrow{\boldsymbol{\eta}}_2$
$\overrightarrow{\boldsymbol{\eta}}_3$
Sentence Embeddings (Sentence Encoder Output)
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$

Figure 3.8: Schematic for the encoder contextual sentence embeddings as computed in the Seq2Seq sentence extractor.

mechanism, each decoder contextual embedding $\boldsymbol{\zeta}_i$ attends to the encoder contextual embeddings, $\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n$, to create the final contextual sentence embedding $\mathbf{z}_i$. The final contextual embedding, $\mathbf{z}_i$, thus contains a representation of sentence s_i as well as its relation to the other contextual representations of the remaining sentences. The salience estimate for s_i is then produces by running $\mathbf{z}_i$ through a feed-forward layer with logistic sigmoid output (see Figure 3.10). We now describe in detail the encoder, decoder, and attention/salience estimation layers.

The sentence embeddings produced by the sentence encoder are first encoded by a bidirectional GRU, which produces left and right partial contextual sentence embeddings,

(Figure 3.8.a) Encoder Left and Right Partial Contextual Sentence Embeddings

$$\overrightarrow{\boldsymbol{\eta}}_0 = \mathbf{0}, \tag{3.40}$$

$$\overrightarrow{\boldsymbol{\eta}}_i = \operatorname{GRU}\left(\mathbf{h}_i, \overrightarrow{\boldsymbol{\eta}}_{i-1}; \overrightarrow{\varphi}_\eta\right) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.41}$$

$$\overleftarrow{\boldsymbol{\eta}}_{n+1} = \mathbf{0}, \tag{3.42}$$

$$\overleftarrow{\boldsymbol{\eta}}_i = \operatorname{GRU}\left(\mathbf{h}_i, \overleftarrow{\boldsymbol{\eta}}_{i+1}; \overleftarrow{\varphi}_\eta\right) \qquad \forall i : i \in \{n, \ldots, 1\}, \tag{3.43}$$

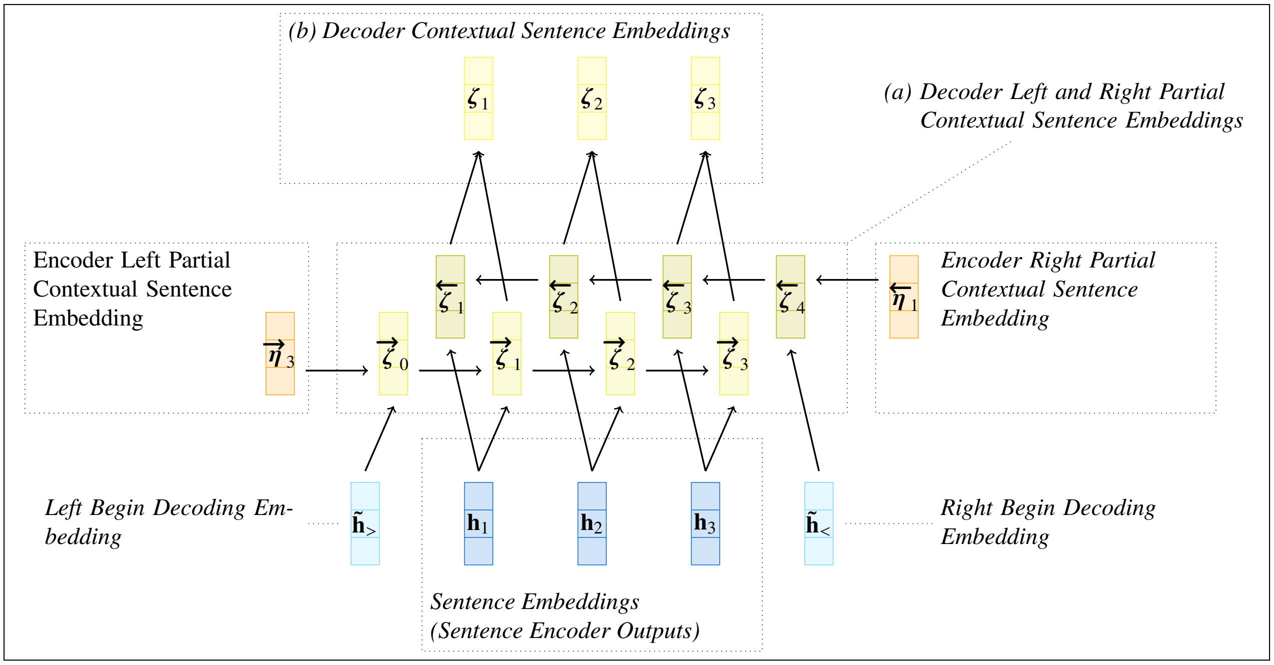

(b) Decoder Contextual Sentence Embeddings
ζ_1
ζ_2
ζ_3
(a) Decoder Left and Right Partial Contextual Sentence Embeddings
Encoder Left Partial Contextual Sentence Embedding
$\overrightarrow{\boldsymbol{\eta}}_3$
$\overleftarrow{\zeta}_1$
$\overleftarrow{\zeta}_2$
$\overleftarrow{\zeta}_3$
$\overleftarrow{\zeta}_4$
$\overrightarrow{\zeta}_0$
$\overrightarrow{\zeta}_1$
$\overrightarrow{\zeta}_2$
$\overrightarrow{\zeta}_3$
$\overleftarrow{\boldsymbol{\eta}}_1$
Encoder Right Partial Contextual Sentence Embedding
Left Begin Decoding Embedding
$\tilde{\mathbf{h}}_>$
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\tilde{\mathbf{h}}_<$
Right Begin Decoding Embedding
Sentence Embeddings (Sentence Encoder Outputs)

Figure 3.9: Schematic for the decoder contextual sentence embeddings as computed by the Seq2Seq sentence extractor.

where $\overrightarrow{\boldsymbol{\eta}}_i, \overleftarrow{\boldsymbol{\eta}}_i \in \mathbb{R}^{D_\chi}$ and $\overrightarrow{\varphi}_\eta$ and $\overleftarrow{\varphi}_\eta$ are the forward and backward encoder GRU parameters respectively. The encoder contextual sentence embeddings are then formed by simply concatenating the encoder left and right partial contextual embeddings,

(Figure 3.8.b) Encoder Contextual Sentence Embeddings

$$\boldsymbol{\eta}_i = \begin{bmatrix} \overrightarrow{\boldsymbol{\eta}}_i \\ \overleftarrow{\boldsymbol{\eta}}_i \end{bmatrix} \qquad \forall i : i \in \{1, \ldots, n\}. \tag{3.44}$$

The final output of each encoder GRU initializes a separate decoder GRU which is then run over the sentence embeddings a second time,

(Figure 3.9.a) Decoder Left and Right Partial Contextual Sentence Embeddings

$$\begin{aligned}
\overrightarrow{\boldsymbol{\zeta}}_0 &= \mathrm{GRU}\left(\tilde{\mathbf{h}}_{>}, \overrightarrow{\boldsymbol{\eta}}_n; \overrightarrow{\varphi}_{\zeta}\right), && && (3.45)\\
\overrightarrow{\boldsymbol{\zeta}}_i &= \mathrm{GRU}(\mathbf{h}_i, \overrightarrow{\boldsymbol{\zeta}}_{i-1}; \overrightarrow{\varphi}_{\zeta}) && \forall i : i \in \{1, \ldots, n\}, && (3.46)\\
\overleftarrow{\boldsymbol{\zeta}}_{n+1} &= \mathrm{GRU}\left(\tilde{\mathbf{h}}_{<}, \overleftarrow{\boldsymbol{\eta}}_1; \overleftarrow{\varphi}_{\zeta}\right), && && (3.47)\\
\overleftarrow{\boldsymbol{\zeta}}_i &= \mathrm{GRU}(\mathbf{h}_i, \overleftarrow{\boldsymbol{\zeta}}_{i+1}; \overleftarrow{\varphi}_{\zeta}) && \forall i : i \in \{1, \ldots, n\}, && (3.48)
\end{aligned}$$

where $\tilde{\mathbf{h}}_{>}$ and $\tilde{\mathbf{h}}_{<}$ are special “begin decoding” input embeddings for the forward and backward decoder respectively, $\overrightarrow{\boldsymbol{\zeta}}_i, \overleftarrow{\boldsymbol{\zeta}}_i \in \mathbb{R}^{D_x}$, and $\overrightarrow{\varphi}_{\zeta}$ and $\overleftarrow{\varphi}_{\zeta}$ are the parameters for the forward and backward decoder GRUs respectively.

The decoder left and right partial contextual sentence embeddings ($\overrightarrow{\boldsymbol{\zeta}}_i$ and $\overleftarrow{\boldsymbol{\zeta}}_i$) are then concatenated to form the decoder contextual sentence embeddings,

(Figure 3.9.b) Decoder Contextual Sentence Embeddings

$$\zeta_i = \begin{bmatrix} \overrightarrow{\boldsymbol{\zeta}}_i \\ \overleftarrow{\boldsymbol{\zeta}}_i \end{bmatrix} \qquad \forall i : i \in \{1, \ldots, n\}. \qquad (3.49)$$

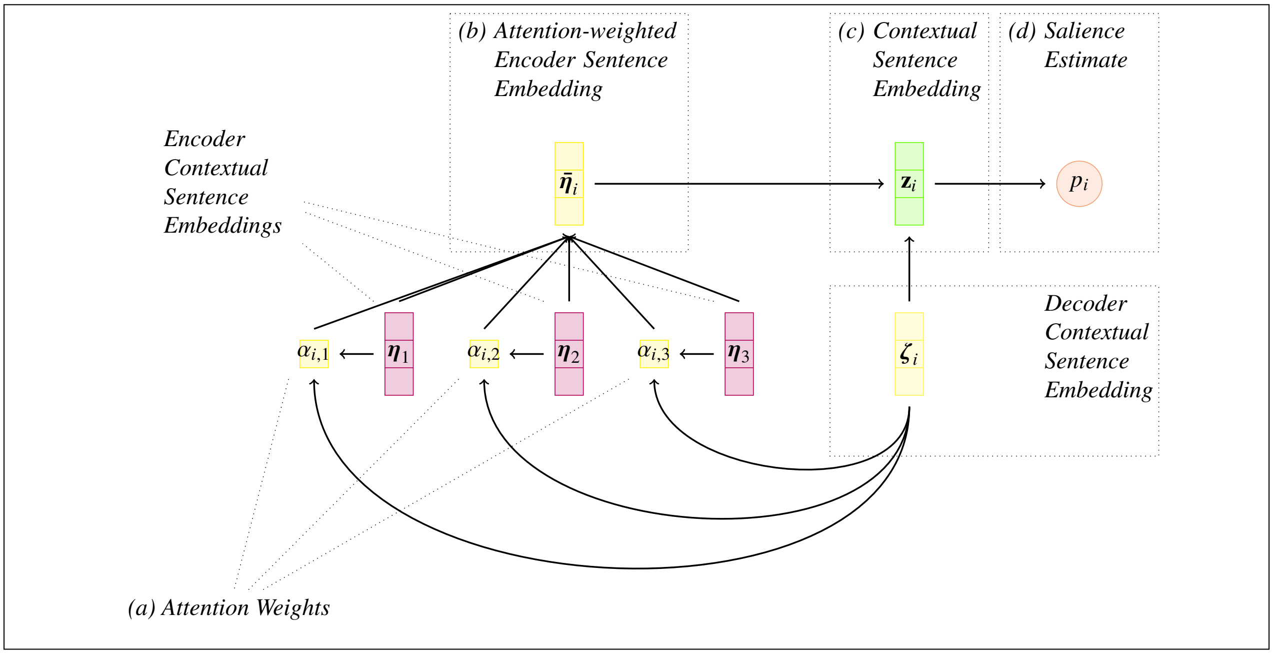


Figure 3.10: Attention layer, contextual sentence embedding, and salience estimation layer for the Seq2Seq extractor.

Each decoder contextual sentence embedding $\boldsymbol{\zeta}_i$ then attends to the encoder contextual sentence embeddings $\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n$, to produce an attention-weighted encoder sentence embedding $\bar{\boldsymbol{\eta}}_i$,

(Figure 3.10.a) Attention Weights

$$\alpha_{i,j} = \frac{\exp\left(\boldsymbol{\zeta}_i \cdot \boldsymbol{\eta}_j\right)}{\sum_{j'=1}^{n} \exp\left(\boldsymbol{\zeta}_i \cdot \boldsymbol{\eta}_{j'}\right)} \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.50}$$

(Figure 3.10.b) Attention-weighted Encoder Sentence Embeddings

$$\bar{\boldsymbol{\eta}}_i = \sum_{j=1}^{n} \alpha_{i,j} \begin{bmatrix} \overrightarrow{\boldsymbol{\eta}}_j \\ \overleftarrow{\boldsymbol{\eta}}_j \end{bmatrix} \qquad \forall i : i \in \{1, \ldots, n\}. \tag{3.51}$$

The attention-weighted encoder sentence embeddings and the decoder contextual sentence embedding are then concatenated and fed through a feed-forward layer to produce the i^{th} contextual sentence embedding $\mathbf{z}_i$, which is itself fed through a final feed-forward layer to compute the i^{th} salience estimate $p_i = P(y_i = 1 | \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi)$,

(Figure 3.10.c) Contextual Sentence Embeddings

$$\mathbf{z}_i = \operatorname{ReLU}\left(\boldsymbol{W}^{(1)} \begin{bmatrix} \bar{\boldsymbol{\eta}}_i \\ \boldsymbol{\zeta}_i \end{bmatrix} + \boldsymbol{b}^{(1)}\right) \qquad \forall i : i \in \{1, \ldots, n\}, \tag{3.52}$$

(Figure 3.10.d) Salience Estimates

$$p_i = P(y_i = 1 | \mathbf{h}_1, \ldots, \mathbf{h}_n; \chi) = \sigma\left(\boldsymbol{W}^{(2)} \mathbf{z}_i + \boldsymbol{b}^{(2)}\right) \qquad \forall i : i \in \{1, \ldots, n\}. \tag{3.53}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{D_z \times 3D_\chi}$, $\boldsymbol{b}^{(1)} \in \mathbb{R}^{D_z}$, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{1 \times D_z}$, and $\boldsymbol{b}^{(2)} \in \mathbb{R}$ are model parameters. The complete set of Seq2Seq extractor parameters is

$$\chi = \left\{ \overrightarrow{\varphi}_\eta, \overleftarrow{\varphi}_\eta, \overrightarrow{\varphi}_\zeta, \overleftarrow{\varphi}_\zeta, \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}, \right\}.$$

In our experiments, we set $D_\chi = 300$, $D_z = 100$. Dropout with drop probability of 0.25 is applied to $\overrightarrow{\boldsymbol{\eta}}_i$, $\overleftarrow{\boldsymbol{\eta}}_i$, $\overrightarrow{\boldsymbol{\zeta}}_i$, $\overleftarrow{\boldsymbol{\zeta}}_i$, and $\mathbf{z}_i$ for all $i \in \{1, \ldots, n\}$.

3.2.3.5 Comparison of Sentence Extractors

All of the extractors rely on running at least one GRU over the sentence embeddings to propagate contextual information of neighboring sentences to $\mathbf{z}_i$. However, only the Cheng & Lapata and SummaRunner extractors rely on $p_1, \ldots, p_{i-1}$ to compute p_i. After the GRU layers are computed in the RNN and Seq2Seq extracts, all salience estimates can be computed in parallel; the Cheng & Lapata and SummaRunner must make their salience predictions sequentially. Additionally, we hypothesize that including this dependence is not necessary to achieve good performance, as similar information about the salience of neighboring sentences will already be captured in the left and right partial contextual sentence embeddings used to construct $\mathbf{z}_i$.

3.2.4 Inference and Summary Generation

Given a setting of encoder and extractor architectures, and the appropriate embedding, encoder, and extractor parameters ($\boldsymbol{\Lambda}$, ξ, and χ respectively), we can compute the probability of a salience label sequence given a document, $P(\boldsymbol{y}|x;\theta)$, as

$$\begin{aligned} P(\boldsymbol{y}|x;\theta) &= P\Big(\boldsymbol{y} \,|\, \mathrm{SentEnc}\left(\mathrm{Emb}\left(s_1;\boldsymbol{\Lambda}\right), \ldots, \mathrm{Emb}\left(s_n;\boldsymbol{\Lambda}\right);\xi\right);\chi\Big) \\ &= \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{(1-y_i)}. \end{aligned}$$

Since even in the autoregressive case discrete extraction decisions are never fed back into the model[2] we can easily find the most likely extractive sequence

$$\hat{\boldsymbol{y}} = \operatorname*{arg\,max}_{\boldsymbol{y} \in \mathcal{Y}} P\left(\boldsymbol{y} | x; \theta\right),$$

where

$$\hat{y}_i = \begin{cases} 1 & \text{if } p_i > 0.5 \\ 0 & \text{otherwise.} \end{cases}$$

This method of inference does not take into account the summary word budget c and the resulting $\hat{\boldsymbol{y}}$ may imply an extract summary that is either too short or too long with respect to the budget constraint. Overly long summaries are less of a problem because they can always be truncated. Short summaries, however, will suffer more in evaluation as ROUGE-recall monotonically increases with summary size until reaching the word budget. Additionally, for fair comparisons between systems, summary lengths should always be the same size (Napoles et al., 2011).

To account for this and obtain a summary of length c, we first compute $p_1, \ldots, p_n$. Then we sort the sentences in descending order $s_{\pi_1}, \ldots, s_{\pi_n}$ where $p_{\pi_i} \geq p_{\pi_{i+1}}$. We then construct the extract summary by selecting the first k sentences such that $\sum_{i=1}^{k-1} l_{\pi_i} < c \leq \sum_{i=1}^{k} l_{\pi_i}$ (i.e., the first k sentences that meet or exceed the length budget). When evaluating or displaying summaries, we show the extracted sentences in the original document order since this improves the coherence, and we truncate the final sentence where it exceeds c.

3.3 Datasets

We perform our experiments across six corpora from varying domains to understand how different biases within each domain can affect content selection. The corpora come from the news domain (CNN-DailyMail, New York Times, DUC), personal narratives domain (Reddit), work-

[2] In the autoregressive extractors, the salience estimates p_i are fed back into the model to compute $p_{i+1}, \ldots, p_n$. However, since these quantities are not determined by whether or not the model actually extracts sentence s_i, they do not create a combinatorial search space.

Dataset	Train	Valid	Test	Refs
CNN/DM	287,113	13,368	11,490	1
NYT	44,382	5,523	6,495	1.93
DUC	516	91	657	2
Reddit	404	24	48	2
AMI	98	19	20	1
PubMed	21,250	1,250	2,500	1

Table 3.1: Sizes of the training, validation, test splits for each dataset and the average number of test set human reference summaries per document.

place meetings (AMI), and medical journal articles (PubMed). See Table 3.1 for dataset statistics.

CNN-DailyMail We use the preprocessing and training, validation, and test splits of See et al. (2017). This corpus is a mix of news on different topics including politics, sports, and entertainment.

New York Times The New York Times (NYT) corpus (Sandhaus, 2008) contains two types of abstracts for a subset of its articles. The first summary is an archival abstract and the second is a shorter online teaser meant to entice a viewer of the webpage to click to read more. From this collection, we take all articles that have a concatenated summary length of at least 100 words. We create training, validation, and test splits by partitioning on dates; we use the year 2005 as the validation data, with training and test partitions including documents before and after 2005 respectively.

DUC We use the single document summarization data from the 2001 and 2002 Document Understanding Conferences (DUC) (Over and Liggett, 2002). We split the 2001 data into training and validation splits and reserve the 2002 data for testing.

AMI The AMI corpus (Carletta et al., 2005) is a collection of real and staged office meetings annotated with text transcriptions, along with abstractive summaries. We use the official train, validation, and test splits as proposed by the dataset authors.

Reddit Ouyang et al. (2017) collected a corpus of personal stories shared on Reddit[3] along with multiple extractive and abstractive summaries. We randomly split this data using roughly three and five percent of the data for validation and testing respectively.

PubMed We created a corpus of 25,000 randomly sampled medical journal articles from the PubMed Open Access Subset.[4] We only included articles if they were at least 1,000 words long and had an abstract of at least 50 words in length. We used the article abstracts as the ground truth human summaries.

3.3.1 Ground Truth Extract Summaries

Since the datasets above typically only have reference abstract summaries, we do not explicitly have document/salience judgement pairs $(x, \boldsymbol{y})$ with which to train a model. In order to obtain $\boldsymbol{y}$, we first construct a "ground truth" reference extract summary $\varepsilon \subseteq x$ by greedily selecting sentences $s_i \in x$ that maximize the ROUGE score (Lin, 2004) with respect to the reference abstract summaries. We then construct the label vector, $\boldsymbol{y} = [y_1, \ldots, y_n]$, by assigning positive salience judgements to those sentences in the extract summary,

$$y_i = \begin{cases} 1 & \text{if } s_i \in \varepsilon \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm for constructing the extract summary and salience judgements from a document and reference abstractive summaries is presented in algorithm 1. It begins by initializing all salience judgements to zero, $\boldsymbol{y} = \boldsymbol{0}$, and the extract summary ε to an empty list (Alg. 1 lines 1-2). It then repeatedly selects the next sentence $s_{\hat{i}}$ from the remaining sentences $s_j \notin \varepsilon$ such that adding $s_{\hat{i}}$ to ε maximally improves the marginal ROUGE score (Alg. 1 lines 3-9). If adding $s_{\hat{i}}$ yields improvement, ε is updated, and $y_{\hat{i}}$ is set to 1 (Alg. 1 lines 6-7). The algorithm terminates when the

[3] www.reddit.com

[4] https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/

Algorithm 1: Salience Label Creation

```
  Data: Input document $x = s_1, \ldots, s_n$, reference abstracts $R$, summary word budget $c$.
1 $y_i \leftarrow 0 \quad \forall i \in 1, \ldots, n$      // Initialize salience judgements to be 0.
2 $\varepsilon \leftarrow [\,]$                     // Initialize summary as empty list.
3 while $\sum_{i=1}^{n} y_i l_i \leq c$ do     // While summary word count ≤ word budget.
4     $\hat{i} \leftarrow \arg\max_{i \in \{1,\ldots,n\}, y_i \neq 1} \textsc{Rouge}(\varepsilon \oplus [s_i], R)$      // Find next best extract.
5     if $\textsc{Rouge}(\varepsilon \oplus [s_{\hat{i}}], R) > \textsc{Rouge}(\varepsilon, R)$ then
6         $\varepsilon \leftarrow \varepsilon \oplus [s_{\hat{i}}]$     // Update extract and salience judgements.
7         $y_{\hat{i}} \leftarrow 1$
8     else
9         break        // No further improvements possible, so end.
  Result: Salience judgements $\boldsymbol{y} = [y_1, \ldots, y_n]$
```

size of extract summary, $\sum_{i=1}^{n} y_i l_i$, excedes the word budget c (Alg. 1 line 3) or adding an additional sentence to ε does not improve the ROUGE score (Alg. 1 line 5). In our experiments, we choose to specifically optimize for the ROUGE-1 recall (i.e. unigram recall) rather than ROUGE-2 recall similarly to other optimization based approaches to summarization (Sipos et al., 2012; Durrett et al., 2016) which found this to be the easier target to learn.

3.4 Experiments

For our main experiments, we train every possible pairing of sentence encoder and extractor architecture ($3 \times 4 = 12$) on each of dataset $\mathcal{D} = \left\{ \left(x^{(1)}, \boldsymbol{y}^{(1)}\right), \ldots, \left(x^{(N)}, \boldsymbol{y}^{(N)}\right) \right\}$. We use the trained models to produce extract summaries for the test set, and we then evaluate summary quality with respect to the reference abstract summaries using ROUGE-2 recall.[5] We use extract summary word budgets of $c = 100$ words for news, and $c = 75$, $c = 290$, and $c = 200$ for Reddit, AMI, and PubMed respectively. We also evaluate using METEOR (Banerjee and Lavie, 2005), which measures precision and recall of reference words while allowing for more matchings on synonymy or morphology. We use the default settings for METEOR. We compute ROUGE with stopwords removed and without stemming, keeping defaults for all other parameters.

For each model configuration, we train five different versions using different random seeds and

[5] ROUGE-1 recall and ROUGE-LCS trend similarity in our experiments so we omit them for space.

report the mean evaluation measure. We estimate statistical significance by first averaging each document level ROUGE or METEOR score over the five random initializations. We then test the difference between the best system on each dataset and all other systems using the approximate randomization test with the Bonferroni correction for multiple comparisons (Riezler and Maxwell, 2005), testing for significance at the 0.05 level.

3.4.1 Training

We train all models to minimize the weighted negative log-likelihood

$$\mathcal{L}(\theta) = -\sum_{(x,\boldsymbol{y})\in\mathcal{D}} \sum_{i=1}^{n} \omega(y_i) \log P\left(y_i | y_1, \ldots, y_{i-1}, x; \theta\right)$$

over the training data $\mathcal{D}$ using stochastic gradient descent with the ADAM optimizer (Kingma and Ba, 2015). Since positive salience labels (i.e. $y_i = 1$) are much rarer than negative salience labels, we reweight the negative log likelihood above, setting

$$\omega(0) = 1 \quad \text{and} \quad \omega(1) = n_0/n_1$$

where n_0 and n_1 are the number of training sentences labeled 0 and 1 respectively. We trained for a maximum of 50 epochs and the best model was selected with early stopping on the validation set according to ROUGE-2. Each epoch constitutes a full pass through the dataset. The average stopping epoch was: CNN-DailyMail, 16.2; NYT, 21.36; DUC, 37.11; Reddit, 36.59; AMI, 19.58; PubMed, 19.84. All experiments were repeated with five random initializations. Unless specified, word embeddings were initialized using pretrained GloVe embeddings (Pennington et al., 2014) and we did not update them during training. Unknown words were mapped to a zero embedding.

We use a learning rate of .0001 and a dropout rate of 0.25 for all dropout layers. We also employ gradient clipping ($-5 < \nabla_\theta < 5$). Weight matrix parameters are initialized using Xavier initialization with the normal distribution (Glorot and Bengio, 2010) and bias terms are set to 0. Hyperparameter settings were found using manual exploration and observing consistent improvements in ROUGE on the validation set. We use a batch size of 32 for all datasets except AMI and PubMed, which are often longer and consume more memory, for which we use sizes two and four respectively.

For Cheng & Lapata based models, we train for half of the maximum epochs with teacher forcing, i.e. we set $p_i = 1$ if $y_i = 1$ in the gold data and 0 otherwise when computing the decoder input $p_i\mathbf{h}_i$. We revert to the predicted model probability during the second half training and during test-time inference.

3.4.2 Baselines

Lead As a baseline we include the lead summary, i.e. taking the first c words of the document as summary, where c is the summary word budget for each dataset (see the first paragraph of §3.4). While incredibly simple, this method is still a competitive baseline for single document summarization, especially on newswire.

Oracle To measure the performance ceiling, we show the ROUGE/METEOR scores using the extractive summary ε which was a bi-product of our algorithm for obtaining salience labels $\boldsymbol{y}$ (see §3.3.1 for details). Essentially, this summary represents an approximate ceiling on ROUGE performance, as it has clairvoyant knowledge of the human reference summaries for each document.

3.5 Results

The results of our main experiment comparing the different extractors/encoders on news and non-news domains are shown in Table 3.2 and Table 3.3 respectively. Overall, we find no major advantage when using the convolutional neural network and recurrent neural network sentence encoders over the averaging encoder. The best performing encoder/extractor pair either uses the averaging encoder (five out of six datasets) or the differences are not statistically significant.

When looking at extractors, the Seq2Seq extractor is either part of the best performing system

Extractor	Encoder	**CNN/DM**		**NYT**		**DUC 2002**	
		M	R-2	M	R-2	M	R-2
Lead	–	24.1	24.4	30.0	32.3	25.1	21.5
RNN	Avg.	**25.2**	25.4	29.8	34.7	**26.8**	22.7
	RNN	25.1	25.4	29.6	34.9	**26.8**	22.6
	CNN	25.0	25.1	29.0	33.7	**26.7**	**22.7**
Seq2Seq	Avg.	**25.2**	**25.6**	**30.5**	**35.7**	**27.0**	**22.8**
	RNN	**25.1**	25.3	30.2	**35.9**	**26.7**	22.5
	CNN	25.0	25.1	29.9	35.1	**26.7**	**22.7**
Cheng & Lapata	Avg.	25.0	25.3	30.4	**35.6**	**27.1**	**23.1**
	RNN	25.0	25.0	**30.3**	**35.8**	**27.0**	**23.0**
	CNN	**25.2**	25.1	29.9	35.0	**26.9**	**23.0**
Summa Runner	Avg.	25.1	25.4	30.2	35.4	26.7	22.3
	RNN	25.1	25.2	30.0	35.5	26.5	22.1
	CNN	24.9	25.0	29.3	34.4	26.4	22.2
Oracle	–	31.1	36.2	35.3	48.9	31.3	31.8

Table 3.2: News domain METEOR (M) and ROUGE-2 recall (R-2) results across all extractor/encoder pairs. Results that are statistically indistinguishable from the best system are shown in bold face.

Extractor	Encoder	**Reddit**		**AMI**		**PubMed**	
		M	R-2	M	R-2	M	R-2
Lead	–	**20.1**	**10.9**	12.3	2.0	15.9	9.3
RNN	Avg.	**20.4**	**11.4**	**17.0**	**5.5**	19.8	17.0
	RNN	**20.2**	**11.4**	16.2	**5.2**	19.7	16.6
	CNN	**20.9**	**12.8**	14.4	3.2	19.9	16.8
Seq2Seq	Avg.	**20.9**	**13.6**	**17.0**	**5.5**	**20.1**	**17.7**
	RNN	**20.5**	**12.0**	16.1	**5.3**	19.7	16.7
	CNN	**20.7**	**13.2**	14.2	2.9	19.8	16.9
Cheng & Lapata	Avg.	**20.9**	**13.6**	**16.7**	**6.1**	**20.1**	**17.7**
	RNN	**20.3**	**12.6**	**16.3**	**5.0**	19.7	16.7
	CNN	**20.5**	**13.4**	14.3	2.8	19.9	16.9
Summa Runner	Avg.	**21.0**	**13.4**	**17.0**	**5.6**	19.9	17.2
	RNN	**20.9**	**12.5**	**16.5**	**5.4**	19.7	16.5
	CNN	**20.4**	**12.3**	14.5	3.2	19.8	16.8
Oracle	–	24.3	16.2	17.8	8.7	24.1	25.0

Table 3.3: Non-news domain METEOR (M) and ROUGE-2 recall (R-2) results across all extractor/encoder pairs. Results that are statistically indistinguishable from the best system are shown in bold face.

Ext.	Emb.	CNN/DM		NYT		DUC		Reddit		AMI		PubMed	
RNN	Fixed	**25.4**		**34.7**		**22.7**		**11.4**		**5.5**		**17.0**	
	F.-T.	25.2	(0.2)	34.3	(0.4)	**22.6**	(0.1)	**11.3**	(0.1)	**5.3**	(0.2)	16.4	(0.6)
Seq2Seq	Fixed	**25.6**		**35.7**		**22.8**		**13.6**		5.5		**17.7**	
	F.-T.	25.3	(0.3)	**35.7**	(0.0)	**22.9**	(-0.1)	**13.8**	(-0.2)	**5.8**	(-0.3)	16.9	(0.8)
C&L	Fixed	**25.3**		**35.6**		**23.1**		**13.6**		**6.1**		**17.7**	
	F.-T.	24.9	(0.4)	35.4	(0.2)	**23.0**	(0.1)	**13.4**	(0.2)	**6.2**	(-0.1)	16.4	(1.3)
Summa Runner	Fixed	**25.4**		**35.4**		**22.3**		**13.4**		**5.6**		**17.2**	
	F.-T.	25.1	(0.3)	35.2	(0.2)	**22.2**	(0.1)	12.6	(0.8)	**5.8**	(-0.2)	16.8	(0.4)

Table 3.4: ROUGE-2 recall across sentence extractors when using fixed pretrained embeddings or when embeddings are fine-tuned (F.-T.) during training. In both cases embeddings are initialized with pretrained GloVe embeddings. All extractors use the averaging sentence encoder. When both fine-tuned and fixed settings are bolded, there is no signifcant performance difference. Difference in scores shown in parenthesis.

(three out of six datasets) or is not statistically distinguishable from the best extractor.

Overall, on the news and medical journal domains, the differences are quite small with the differences between worst and best systems on the CNN/DM dataset spanning only .56 of a ROUGE point. While there is more performance variability in the Reddit and AMI data, there is less distinction among systems: no differences are significant on Reddit and every extractor has at least one configuration that is indistinguishable from the best system on the AMI corpus. This is probably due to the small test size of these datasets.

3.5.1 Ablation Experiments

In addition to our main evaluation above, we also perform several ablation experiments to further understand how the various summarization models perform when certain information is witheld from the model. In particular, we evaluate the effect of fine-tuning word embeddings, part-of-speech (POS) based ablations, and sentence-order shuffling.

Word Embedding Fine-tuning Given that learning a sentence encoder (averaging has no learned parameters) does not yield significant improvement, it is natural to consider whether fine-tuning word embeddings is also necessary. In Table 3.4 we compare the performance of different extractors using the averaging encoder, when the word embeddings are held fixed or fine-tuned during training. In both cases, word embeddings are initialized with GloVe embeddings trained on a combination of Gigaword and Wikipedia. When fine-tuning embeddings, words occurring fewer than three times in the training data are mapped to an unknown token (with learned embedding).

In all but one case, fixed embeddings are as good or better than the fine-tuned embeddings. This is a somewhat surprising finding on the CNN/DM data since it is reasonably large, and learning embeddings should give the models more flexibility to identify important word features.[6] This suggests that we cannot extract much generalizable learning signal from the content other than what is already present from initialization. Even on PubMed, where the language is quite different from the news/Wikipedia articles the GloVe embeddings were trained on, fine-tuning leads to significantly worse results.

Ablation	**CNN/DM**	**NYT**	**DUC**	**Reddit**	**AMI**	**PubMed**
all words	**25.4**	**34.7**	22.7	**11.4**	5.5	**17.0**
-nouns	25.3† (0.1)	34.3† (0.4)	22.3† (0.4)	10.3† (1.1)	3.8† (1.7)	15.7† (1.3)
-verbs	25.3† (0.1)	34.4† (0.3)	22.4† (0.3)	10.8 (0.6)	5.8 (-0.3)	16.6† (0.4)
-adj/adv	25.3† (0.1)	34.4† (0.3)	22.5 (0.2)	9.5† (1.9)	5.4 (0.1)	16.8† (0.2)
-function	25.2† (0.2)	34.5† (0.2)	**22.9**† (-0.2)	10.3† (1.1)	**6.3**† (-0.8)	16.6† (0.4)

Table 3.5: ROUGE-2 recall after removing nouns, verbs, adjectives/adverbs, and function words. Ablations are performed using the averaging sentence encoder and the RNN extractor. Bold indicates best performing system. † indicates significant difference with the non-ablated system. Difference in score from *all words* shown in parenthesis.

POS Ablation It is also not well explored what word features are being used by the encoders. To understand which classes of words were most important we ran an ablation study, selectively removing nouns, verbs (including participles and auxiliaries), adjectives & adverbs, and function words (adpositions, determiners, conjunctions). All datasets were automatically tagged using the spaCy POS tagger[7]. The embeddings of removed words were replaced with a zero vector, preserving the order and position of the non-ablated words in the sentence. Ablations were performed

[6] The AMI corpus is an exception here where learning *does* lead to small performance boosts, however, only in the Seq2Seq extractor is this diference significant; it is quite possible that this is an artifact of the very small test set size.

[7] https://github.com/explosion/spaCy

Ext.	Order	CNN/DM	NYT	DUC	Reddit	AMI	PubMed
Seq2Seq	In-Order	**25.6**	**35.7**	**22.8**	**13.6**	5.5	**17.7**
	Shuffled	21.7 (3.9)	25.6 (10.1)	21.2 (1.6)	**13.5** (0.1)	**6.0** (-0.5)	14.9 (2.8)

Table 3.6: ROUGE-2 recall using models trained on in-order and shuffled documents. Extractor uses the averaging sentence encoder. When both in-order and shuffled settings are bolded, there is no signifcant performance difference. Difference in scores shown in parenthesis.

on training, validation, and test partitions, using the RNN extractor with averaging encoder. Note that while the input to the models has redacted word classes, the produced summaries are evaluated with all words present. See Figure 3.11 for example inputs under the different word class ablations.

Table 3.5 shows the results of the POS tag ablation experiments. While removing any word class from the representation generally hurts performance (with statistical significance), on the news domains, the absolute values of the differences are quite small (.18 on CNN/DM, .41 on NYT, .3 on DUC) suggesting that the model's predictions are not overly dependent on any particular word types.

Qualitatively, we see little difference in outputs produced under the redacted models. For example, see Figure 3.12 where we show output summaries from the different word class redacted models. All four variants identify the lead two sentences as most important, while also including sentences five and six. Three of the summaries also selected sentence 13. The summarizers appear to have selected most content from the front of the article suggesting the lead is identfiable even when different content is ablated.

On the non-news datasets, the ablations have a larger effect (max differences are 1.89 on Reddit, 2.56 on AMI, and 1.3 on PubMed). Removing nouns leads to the largest drop on AMI and PubMed. Removing adjectives and adverbs leads to the largest drop on Reddit, suggesting the intensifiers and descriptive words are useful for identifying important content in personal narratives. Curiously, removing the function word POS class yields a significant improvement on DUC 2002 and AMI.

Sentence Order Shuffling Sentence position is a well known and powerful feature for news summarization (Hong and Nenkova, 2014), owing to the intentional lead bias in the news article

Nouns Redacted

1. [redacted] [redacted] swept toward the [redacted] [redacted] [redacted], and the [redacted] [redacted] alerted its heavily populated south [redacted] to prepare for high [redacted], heavy [redacted] and high [redacted].
2. the [redacted] was approaching from the [redacted] with sustained [redacted] of 75 [redacted] gusting to 92 [redacted] .
3. “There is no [redacted] for [redacted],” civil [redacted] [redacted] [redacted] [redacted] said in a [redacted] [redacted] shortly before [redacted] [redacted].
4. [redacted] said [redacted] of the [redacted] of [redacted] should closely follow [redacted]’s [redacted].

Verbs Redacted

1. Hurricane Gilbert [redacted] toward the Dominican Republic Sunday, and the Civil Defense [redacted] its heavily populated south coast [redacted] [redacted] for high winds, heavy rains and high seas.
2. The storm [redacted] [redacted] from the southeast with sustained winds of 75 mph [redacted] to 92 mph.
3. “There [redacted] no need for alarm,” Civil Defense Director Eugenio Cabral [redacted] in a television alert shortly before midnight Saturday.
4. Cabral [redacted] residents of the province of Barahona [redacted] closely [redacted] Gilbert [redacted] movement.

Adjectives/Adverbs Redacted

1. Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted [redacted] [redacted] [redacted] [redacted] coast to prepare for [redacted] winds, [redacted] rains and [redacted] seas.
2. The storm was approaching from the southeast with [redacted] winds of 75 mph gusting to 92 mph.
3. “[redacted] is no need for alarm,” [redacted] Defense Director Eugenio Cabral said in a television alert [redacted] before midnight Saturday.
4. Cabral said residents of the province of Barahona should [redacted] follow Gilbert’s movement.

Function Words Redacted

1. Hurricane Gilbert swept [redacted] [redacted] Dominican Republic Sunday, [redacted] [redacted] Civil Defense alerted its heavily populated south coast to prepare [redacted] high winds, heavy rains [redacted] high seas.
2. [redacted] storm was approaching [redacted] [redacted] southeast [redacted] sustained winds [redacted] 75 mph gusting [redacted] 92 mph .
3. “There is [redacted] need [redacted] alarm,” Civil Defense Director Eugenio Cabral said [redacted] [redacted] television alert shortly [redacted] midnight Saturday.
4. Cabral said residents [redacted] [redacted] province [redacted] Barahona should closely follow Gilbert’s movement.

Figure 3.11: The first four sentences from a DUC 2002 article (id: d061j-AP880911-0016) under the different word class ablations.

Nouns Redacted

(1) Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas.

(2) The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph.

(5) An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo.

(6) Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night.

(7) The National Hurricane Center in Miami reported its position at 2 a.m. Sunday at latitude 16.1 north, longitude 67.5 west, about 140 miles south of Ponce, Puerto Rico, and 200 miles southeast of Santo Domingo.

Verbs Redacted

(1) Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas.

(2) The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph.

(13) On Saturday, Hurricane Florence was downgraded to a tropical storm and its remnants pushed inland from the U.S. Gulf Coast.

(6) Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night.

(12) San Juan, on the north coast, had heavy rains and gusts Saturday, but they subsided during the night.

(5) An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo.

Adjectives/Adverbs Redacted

(2) The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph.

(1) Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas.

(5) An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo.

(13) On Saturday, Hurricane Florence was downgraded to a tropical storm and its remnants pushed inland from the U.S. Gulf Coast.

(6) Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night.

(14) Residents returned home, happy to find little damage from 80 mph winds and sheets of rain.

Function Words Redacted

(1) Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas.

(2) The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph.

(13) On Saturday, Hurricane Florence was downgraded to a tropical storm and its remnants pushed inland from the U.S. Gulf Coast.

(6) Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night.

(10) Strong winds associated with the Gilbert brought coastal flooding, strong southeast winds and up to 12 feet feet to Puerto Rico's south coast.

(5) An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo.

Figure 3.12: Outputs from the word class ablated models when given the document as input from Figure 3.11. Original document positions of the extracted sentences are shown in parenthesis; sentences that are selected by multiple systems are highlighted in color.

· Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas. · The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph. · An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo. · **On Saturday, Hurricane Florence was downgraded to a tropical storm and its remnants pushed inland from the U.S. Gulf Coast.** · Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night.	· Hurricane Gilbert swept toward the Dominican Republic Sunday, and the Civil Defense alerted its heavily populated south coast to prepare for high winds, heavy rains and high seas. · The storm was approaching from the southeast with sustained winds of 75 mph gusting to 92 mph. · An estimated 100,000 people live in the province, including 70,000 in the city of Barahona, about 125 miles west of Santo Domingo. · Tropical Storm Gilbert formed in the eastern Caribbean and strengthened into a hurricane Saturday night. · **Strong winds associated with the Gilbert brought coastal flooding, strong southeast winds and up to 12 feet feet to Puerto Rico's south coast.**

Table 3.7: Example output of Seq2Seq extractor (left) and Cheng & Lapata Extractor (right). This is a typical example, where only one sentence is different between the two (shown in bold).

writing[8]; it also explains the difficulty in beating the lead baseline for single-document summarization (Nenkova, 2005; Brandow et al., 1995). In examining the generated summaries, we found most of the selected sentences in the news domain came from the lead paragraph of the document. This is despite the fact that there is a long tail of sentence extractions from later in the document in the ground truth extract summaries (31%, 28.3%, and 11.4% of DUC, CNN/DM, and NYT training extract labels come from the second half of the document). Because this lead bias is so strong, it is questionable whether the models are learning to identify important content or just find the start of the document. We conduct a sentence order experiment where each document's sentences are randomly shuffled during training. We then evaluate each model performance on the unshuffled test data, comparing to the model trained on unshuffled data; if the models trained on shuffled data drop in performance, then this indicates the lead bias is the relevant factor.

Table 3.6 shows the results of the shuffling experiments. The news domains and PubMed suffer a significant drop in performance when the document order is shuffled. By comparison, there is no significant difference between the shuffled and in-order models on the Reddit domain, and shuffling actually improves performance on AMI. This suggest that position is being learned by the models in the news/journal article domain even when the model has no explicit position features, and that this feature is more important than either content or function words.

[8] https://en.wikipedia.org/wiki/Inverted_pyramid_(journalism)

3.6 Discussion

Learning content selection for summarization in the news domain is severely inhibited by the lead bias. The summaries generated by all systems described here–the prior work and our proposed simplified models–are highly similar to each other and to the lead baseline. The Cheng & Lapata and Seq2Seq extractors (using the averaging encoder) share 87.8% of output sentences on average on the CNN/DM data, with similar numbers for the other news domains (see Table 3.7 for a typical example). Also on CNN/DM, 58% of the Seq2Seq selected sentences also occur in the lead summary, with similar numbers for DUC, NYT, and Reddit. Shuffling reduces lead overlap to 35.2% but the overall system performance drops significantly; the models are not able to identify important information without position.

The relative robustness of the news domain to part of speech ablation also suggests that models are mostly learning to recognize the stylistic features unique to the beginning of the article, and not the content. Additionally, the drop in performance when learning word embeddings on the news domain suggests that word embeddings alone do not provide very generalizable content features compared to recognizing the lead.

The picture is rosier for non-news summarization where part of speech ablation leads to larger performance differences and shuffling either does not inhibit content selection significantly or leads to modest gains. Learning better word-level representations on these domains will likely require much larger corpora, something which might remain unlikely for personal stories and meetings.

The lack of distinction among sentence encoders is interesting because it echoes findings in the generic sentence embedding literature where word embedding averaging is frustratingly difficult to outperform (Iyyer et al., 2015; Wieting et al., 2016; Arora et al., 2017; Wieting and Gimpel, 2017). The inability to learn useful sentence representations is also borne out in the SummaRunner model, where there are explicit similarity computations between document or summary representations and sentence embeddings; these computations do not seem to add much to the performance as the Cheng & Lapata and Seq2Seq models which lack these features generally perform as well or better. Furthermore, the Cheng & Lapata and SummaRunner extractors both construct a history of previous selection decisions to inform future choices but this does not seem to significantly improve performance over the Seq2Seq extractor (which does not). This suggests that we need to be cautious about the single document summarization task as a test bed for learning summarization models. The input does not appear to be sufficiently rich enough or difficult enough that modeling dependicies in the output needs to be done explicity.

A manual examination of the outputs revealed some interesting failure modes, although in general it was hard to discern clear patterns of behaviour other than lead bias. On the news domain, the models consistently learned to ignore quoted material in the lead, as often the quotes provide color to the story but are unlikely to be included in the summary (e.g. *"It was like somebody slugging a punching bag."*). This behavior was most likely triggered by the presence of quotes, as the quote attributions, which were often tokenized as separate sentences, would subsequently be included in the summary despite also not containing much information (e.g. *Gil Clark of the National Hurricane Center said Thursday*).

3.7 Conclusion

We have presented an empirical study of deep learning-based salience estimation models for summarization. In particular, we examined three different sentence encoders for representing sentences and four different extraction models for predicting sentence salience. Our findings suggest that position heuristics are in many cases exploitable when performing sentence selection, and that models are not making extensive use of content features. As a result, simple encoders like the averaging encoder and simple extractors like the RNN extractor are fairly competitive with more sophisticated models that explicitly model dependencies in the output.

Interestingly, the performance ceiling on extractive, single-document news summarization continues to be raised. When these experiments were carried out, large, pretrained models like BERT (Devlin et al., 2019) were not yet in use. Subsequent work on fine-tuning BERT-based models has shown a modest increase in ROUGE performance on the CNN-DailyMail dataset (Liu and Lapata, 2019). Additionally, Liu and Lapata (2019) show that the BERT based models select from the lead sentences less than the oracle, and draw more frequently from the long tail of the document as well. Meanwhile, a comparable model trained from scratch selected lead sentences much more frequently than the oracle would have (something we also observed). This suggests large, pre-trained language models are better able to exploit features beyond position bias, although it would be interesting to tease this out in more detail in future work.

In either case, it would seem advisable to revise the paradigm of training single document summarization systems to do “generic summarization,” where there is no goal or prior instruction on what is relavent or intended to be searched for. Since this generic task is underspecified, it is relatively easy for models to exploit heuristics as opposed to learning to reason about the salience of a text from features that are more semantically relavent to the task. In the next chapter, we explore this idea further by attempting to summarize a large stream of documents where position heuristics are less useful. Additionally, we develop more wholistic summarization algorithms that can incorporate salience estimates (which could be produced either by a deep learning or classical machine learning based salience estimator) while taking into account features of redundancy or novelty.

Chapter 4: Salience Estimation with Structured Content Selection Models

In many cases, estimating salience is not the entirety of the summarization system's task. Accounting for redundancy is also an important factor in many summarization systems (especially multi-document summarization) since the same information can often be restated multiple times. Additionally, in many situations, salience is dynamic, changing over time with the information need of the summary receiver. In this chapter, we explore ways of incorporating salience predictions into more holistic algorithms for constructing extract summaries using information about text unit redundancy or the summarization system's prior extraction decisions.

Since this approach to summarization is not totally necessary for single document summarization, we motivate the models in this chapter with a more difficult summarization challenge: query focused, sentence extractive, streaming news summarization. In this problem, the summarization system must monitor a stream of news articles and extract sentences, which we call updates, that are relevant to a user query. Collectively, these updates constitute an update summary. As in the last chapter, we rely on a data-driven assignment of update salience, where an update is salient if it contains information that was found in a human authored summary of the query-document stream.

A notable aspect of the stream summarization task is the notion of system time – the summarization system can consider all sentences that have entered the stream before the current system time. Advancing the system time allows the summarizer to observe more sentences from the stream. However, the salience of relevant information decreases monotonically from the earliest time that information was published to the final system time it was actually extracted for the update summary. Because of this, we must extract sentences in an online fashion, attempting to minimize the latency between the time that important information is first published and the time the summarization system extracts that information.

Since there is little supervised data for this task, we rely on a feature-based regression model to provide our salience estimates. The time constraint makes this a particularly challenging task as the typical features for summarization make use of static term frequency. In the streaming case, these features are now constantly evolving with time, and at the start of the stream, estimates of term frequency may not be very reliable. A second but important issue is that salience estimates do not occur in isolation. As we add updates to the summary, the salience of our remaining inputs is likely to change based on redundancy and other factors. Unfortunately, adding summary-sentence interaction features introduces an element of exploration to training a salience estimation model for now various summary configuration and candidate sentence pairs must be considered.

Our two proposed feature-based summarization models deal with these issues in slightly different ways. The first model, the salience-biased affinity propagation (SAP) summarizer (Kedzie et al., 2015), combines independent, sentence-level salience estimates with the affinity propagation clustering algorithm (Frey and Dueck, 2007). Affinity propagation forms clusters by identifying a set of "exemplar" inputs and mapping the remaining inputs to one of the exemplars. Under our modification of the clustering algorithm, we jointly select exemplars that are individually highly salient but also representative of the inputs, adding the resulting exemplars to the update summary.

Our second model, the learning-to-search (L2S) summarizer (Kedzie et al., 2016), allows us to freely incorporate summary/sentence interaction features, as we train the salience model using the learning-to-search regime (Daumé III and Marcu, 2005; Chang et al., 2015) where learning takes place using different exploration policies. Using this method we can learn a summarization policy that makes greedy sentence extraction decisions that also correlate with a good final summary. The L2S summarizer learns to optimize the entire summarization process, jointly estimating the salience of sentences as well as when to extract them, taking into account previous extraction decisions and the candidate update's similarity to the current update summary. Additionally, the L2S summarizer works in a greedy online manner, meaning that it can extract salient content almost as soon as it is published, minimizing the affects of latency. In the next sections, we will introduce the query focused, sentence extractive, streaming news summarization task and dataset, before discussing our proposed SAP and L2S models.

4.1 Task Definition

We now describe the query focused, sentence extractive, streaming news summarization problem in detail. We start with the query, q, which is a brief string describing an event of interest to be summarized. See Table 4.1 for some example queries. All relevance judgements about a sentence are made with respect to the query string. Additionally, the query is used to construct the news stream which we now turn to.

The news stream is an ordered sequence of text approximately relevant to the query (i.e. the results of an information retrieval system). It is useful to be able to talk about this stream from two perspectives, as either a stream of documents, $\mathcal{D}$, or a flat stream of sentences, $\mathcal{S}$. From the document perspective, the stream is an ordered sequence of $m^{(\mathcal{D})}$ documents

$$\mathcal{D}^{(q)} = \left[d_1, d_2, \ldots, d_{m^{(\mathcal{D})}}\right]$$

where each document d_i is itself an ordered sequence of n_i sentences,

$$d_i = \left[s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}\right].$$

Each document d_i also has a timestamp $\tau_i^{(\mathcal{D})}$ which is also shared by all of its sentences $s_{i,j} \in d_i$. The stream is ordered by timestamp, so we have $\tau_i^{(\mathcal{D})} < \tau_{i+1}^{(\mathcal{D})}$ for all $i \in \{1, \ldots, m^{(\mathcal{D})} - 1\}$. From the sentence perspective, the news stream is an ordered sequence of $m^{(\mathcal{S})}$ sentences,

$$\mathcal{S} = \left[s_1, s_2, \ldots, s_{m^{(\mathcal{S})}}\right],$$

where each sentence s_i has a timestamp $\tau_i^{(\mathcal{S})}$ and $\tau_i^{(\mathcal{S})} \leq \tau_{i+1}^{(\mathcal{S})}$ for all $i \in \{1, \ldots, m^{(\mathcal{S})} - 1\}$. The two points of view are equivalent in the sense that the concatenation of $\mathcal{D}$ equals $\mathcal{S}$,

$$d_1 \oplus d_2 \oplus \cdots \oplus d_{m^{(\mathcal{D})}} = \mathcal{S}$$

		Period of Interest			
TREC Year	Wikipedia Page Title	Start Time ($\tau_{\text{«}s\text{»}}$)	Stop Time ($\tau_{\text{«}e\text{»}}$)	Query String (q)	Event Category (c)
2013	2012 Buenos Aires Rail Disaster	02/22/2012 11:33am	03/03/2012 11:33am	*buenos aires train crash*	accident
2013	2012 Pakistan garment factory fires	09/11/2012 1:00pm	09/21/2012 1:00pm	*pakistan factory fire*	accident
2013	2012 Aurora shooting	07/20/2012 6:38am	07/30/2012 6:38am	*colorado shooting*	shooting
2013	Wisconsin Sikh temple shooting	08/05/2012 3:25pm	08/15/2012 3:25pm	*sikh temple shooting*	shooting
2013	Hurricane Isaac (2012)	08/28/2012 4:20pm	09/07/2012 4:20pm	*hurricane isaac*	storm
2013	Hurricane Sandy	10/24/2012 3:00pm	11/03/2012 3:00pm	*hurricane sandy*	storm
2013	June 2012 North American derecho	06/29/2012 3:00pm	07/09/2012 3:00pm	*midwest derecho*	storm
2013	Typhoon Bopha	11/30/2012 2:45pm	12/10/2012 2:45pm	*typhoon bopha*	storm
2013	2012 Guatemala earthquake	11/07/2012 4:35pm	11/17/2012 4:35pm	*guatemala earthquake*	earthquake
2013	2012 Tel Aviv bus bombing	11/21/2012 10:00am	12/01/2012 10:00am	*tel aviv bus bombing*	bombing
2014	Costa Concordia disaster and recovery	01/13/2012 9:45pm	02/01/2012 12:00am	*costa concordia*	accident
2014	Early 2012 European cold wave	01/22/2012 12:00am	02/18/2012 12:00am	*european cold wave*	storm
2014	2013 Eastern Australia floods	01/17/2013 12:00am	01/30/2013 12:00am	*queensland floods*	storm
2014	Boston Marathon bombings	04/15/2013 6:49pm	04/20/2013 11:59pm	*boston marathon bombing*	bombing
2014	Port Said Stadium riot	02/01/2012 1:30pm	02/11/2012 1:30pm	*egyptian riots*	riot
2014	2012 Afghanistan Quran burning protests	02/20/2012 5:30pm	02/28/2012 12:00am	*quran burning protests*	protest
2014	In Amenas hostage crisis	01/16/2013 12:00am	01/20/2013 12:00am	*in amenas hostage crisis*	hostage
2014	2011-13 Russian protests	12/04/2011 12:00am	12/25/2011 12:00am	*russian protests*	protest
2014	2012 Romanian protests	01/12/2012 12:00am	01/26/2012 12:00am	*romanian protests*	protest
2014	2012-13 Egyptian protests	11/18/2012 12:00am	12/01/2012 12:00am	*egyptian protests*	protest
2014	Chelyabinsk meteor	02/15/2013 3:20am	02/25/2013 3:20am	*russia meteor*	impact event
2014	2013 Bulgarian protests against the Borisov cabinet	02/10/2013 12:00am	02/20/2013 11:59pm	*bulgarian protests*	protest
2014	2013 Shahbag protests	02/05/2013 12:00am	02/22/2013 11:59pm	*shahbag protests*	protest
2014	February 2013 nor'easter	02/07/2013 12:00am	02/18/2013 11:59pm	*nor'easter*	storm
2014	Christopher Dorner shootings and manhunt	02/03/2013 12:00am	02/13/2013 7:59am	*Southern California shooting*	shooting
2015	Vauxhall helicopter crash	01/16/2013 7:59am	01/31/2013 7:59am	*vauxhall helicopter crash*	accident
2015	Cyclone Nilam	10/27/2012 12:00am	11/02/2012 12:00am	*cyclone nilam*	storm
2015	2013 Dhaka garment factory collapse	04/24/2013 2:45am	05/04/2013 2:45am	*savar building collapse*	accident
2015	2013 Hyderabad blasts	02/21/2013 1:58pm	03/03/2013 1:58pm	*hyderabad explosion*	bombing
2015	Brazzaville arms dump blasts	03/04/2012 7:00am	03/14/2012 7:00am	*brazzaville explosion*	accident
2015	2012 India blackouts	07/29/2012 9:18pm	08/03/2012 9:18pm	*india power blackouts*	accident
2015	Reactions to Innocence of Muslims	09/11/2012 12:00am	09/30/2012 12:00am	*innocence of muslims protests*	protest
2015	Battle of Konna	01/10/2013 12:00am	01/19/2013 12:00am	*konna battle*	conflict
2015	February 2013 Quetta bombing	02/16/2013 12:00am	02/20/2013 12:00am	*quetta bombing*	bombing
2015	15 April 2013 Iraq attacks	04/15/2013 12:00am	04/20/2013 12:00am	*iraq bombing*	bombing
2015	19 March 2013 Iraq attacks	03/19/2013 12:00am	03/24/2013 12:00am	*iraq bombing*	bombing
2015	2011-12 Los Angeles arson attacks	12/29/2011 9:00am	01/05/2012 9:00am	*los angeles arson*	bombing
2015	2013 Thane building collapse	04/04/2013 12:00am	04/13/2013 12:00am	*thane building collapsed*	accident
2015	2013 United States embassy bombing in Ankara	02/01/2013 12:00am	02/05/2013 12:00am	*suicide bomber ankara*	bombing
2015	22 December 2011 Baghdad bombings	12/21/2011 9:00pm	12/26/2011 9:00pm	*baghdad bomb*	bombing
2015	Aleppo University bombings	01/15/2013 12:00am	01/25/2013 12:00am	*aleppo university explosion*	bombing
2015	Carnival Triumph 2013 Engine Room Fire	02/10/2013 12:00am	02/15/2013 12:00am	*carnival triumph fire*	accident
2015	USS Guardian (MCM-5) January 2013 Grounding	01/17/2013 12:00am	01/22/2013 12:00am	*uss guardian grounding*	accident
2015	2012 Indian Ocean earthquakes	04/11/2012 12:00am	04/16/2012 12:00am	*aceh earthquake*	earthquake
2015	2012 Haida Gwaii earthquake	10/28/2012 3:00am	11/07/2012 3:00am	*haida gwaii earthquake*	earthquake
2015	2012 Catalan independence demonstration	09/11/2012 12:00am	09/16/2012 12:00am	*catalan protest*	protest

Table 4.1: TREC Temporal Summarization shared-task query events for the years 2013-2015. All times are UTC.

and $\tau_i^{(\mathcal{D})} = \tau_j^{(\mathcal{S})}$ for all $s_j \in d_i$.

A query focused, sentence extractive stream summarization model must process the stream sequentially in time and determine for each sentence $s_j \in \mathcal{S}$ whether to extract it or to skip it. That is, the system must decide to add the sentence to a summary of the stream, or to ignore the sentence. While there is no explicit length constraint in the number of updates for a summary, the ideal update covers novel and salient information. A summarizer that extracts many sentences that are not informative or are redundant given prior updates will receive lower scores under the evaluation measures which we describe later in this section.

Crucial to the stream summarization problem is the notion of system time, which indicates what information from the stream has been read and can be used to make extraction predictions. When the system time is $\hat{\tau}_t$, the summarization model can in theory use any information collected from all documents $d_i \in \mathcal{D}$ such that $\tau_i^{(\mathcal{D})} \leq \hat{\tau}_t$, although for more significant query-streams, it may not be practical for a summarization model to store all previous documents. For any sentences not yet extracted, it can similarly decide to extract any sentence $s_j \in \mathcal{S}$ such that $\tau_j^{(\mathcal{S})} \leq \hat{\tau}_t$. Previous extraction decisions, however, cannot be undone. The system time can be incremented by arbitrary positive amounts to $\hat{\tau}_{t+1}$ (i.e. $\hat{\tau}_t < \hat{\tau}_{t+1}$) to allow the summarization model to observe and extract more sentences. Given two timestamps τ_1, τ_2 with $\tau_1 < \tau_2$, we denote the sub-sequence of documents in the stream that fall between them as $\mathcal{D}_{\tau_1:\tau_2}$. Similarly, $\mathcal{S}_{\tau_1:\tau_2}$ indicates the subsequence of sentences with timestamps that fall between τ_1 and τ_2.

We refer to a sentence that has been extracted as an update. Let u_k be the k-th update extracted by the system. u_k has a corresponding timestamp, $\tau_k^{(\mathcal{U})}$ that is equal to the system time that it was extracted by the summarizer. That is, if u_k was extracted at $\hat{\tau}_t$, then $\tau_k^{(\mathcal{U})} = \hat{\tau}_t$. We refer to a collection of K timestamped updates as an update summary, $\mathcal{U} = \left[\left(u_1, \tau_1^{(\mathcal{U})}\right), \ldots, \left(u_K, \tau_K^{(\mathcal{U})}\right)\right]$.

For evaluation purposes we compare the update summary to a human reference abstract summary of the query event. The reference abstract summary, $\mathcal{N} = \left[\left(\gamma_1, \tau_1^{(\mathcal{N})}\right), \ldots, \left(\gamma_r, \tau_r^{(\mathcal{N})}\right)\right]$, contains a series of r sentences describing important facts about the event. We refer to these facts as *nuggets*. Each nugget γ_i has a timestamp $\tau_i^{(\mathcal{N})}$, indicating the reference time that information was published.

We say an update u covers a nugget γ, which we write $\llbracket\gamma\rrbracket \in \llbracket u\rrbracket$, if the information in the nugget γ is described in the update u. Note that it is possible for an update to cover multiple nuggets. For example if we have,

$$\begin{aligned}
\gamma_1 &= \textit{Hurricane Sandy was a category 5 hurricane.}\\
\gamma_2 &= \textit{Hurricane Sandy made landfall on Saturday.}\\
u &= \textit{Hurricane Sandy, which made landfall on Saturday, was upgraded to a category 5 storm.}
\end{aligned}$$

then $\llbracket\gamma_1\rrbracket, \llbracket\gamma_2\rrbracket \in \llbracket u\rrbracket$. Given an update summary $\mathcal{U}$ and a nugget γ, we define the earliest cover,

$$M(\gamma, \mathcal{U}) = \begin{cases} \arg\min_{\left(u_k, \tau_k^{(\mathcal{U})}\right) \in \mathcal{U} : \llbracket\gamma\rrbracket \in \llbracket u_k\rrbracket} \tau_k^{(\mathcal{U})} & \text{if } \exists \left(u_k, \tau_k^{(\mathcal{U})}\right) \in \mathcal{U} : \llbracket\gamma\rrbracket \in \llbracket u\rrbracket \\ \emptyset & \text{otherwise} \end{cases}$$

as the earliest update that covers the nugget γ, or the empty set if no update in the update summary covers γ. We also define the inverse mapping,

$$M^{-1}(u, \mathcal{U}, \mathcal{N}) = \left\{ \left(\gamma_l, \tau_l^{(\mathcal{N})}\right) \in \mathcal{N} \middle| \exists \tau : (u, \tau) = M(\gamma_l, \mathcal{U}) \right\},$$

which is the set of nuggets that have u as its earliest cover.

The objective of the summarization model is to produce an update summary $\mathcal{U}$ such that the set of updates covers the information expressed by the nuggets without containing much repeated information. Additionally, the summarization system should try to minimize the latency between the time information in a nugget is available in the stream and the system time that information is extracted.

We now formalize these evaluation criteria using the official Temporal Summarization evaluation measures (Aslam et al., 2013). Given an update and reference summary, $\mathcal{U}$ and $\mathcal{N}$ respectively, we define the gain of an update as

$$G(u_k, \tau_k^{(\mathcal{U})}, \mathcal{U}, \mathcal{N}) = \sum_{\left(\gamma_j, \tau_j^{(\mathcal{N})}\right) \in M^{-1}(u_k, \mathcal{U}, \mathcal{N})} 1$$

which is essentially the number of nuggets covered by an update u. We also define a latency penalized version of this function,

$$G_L\left(u_i, \mathcal{U}, \mathcal{N}\right) = \sum_{\left(\gamma_j, \tau_j^{(\mathcal{N})}\right) \in M^{-1}(u_k, \mathcal{U}, \mathcal{N})} L\left(\tau_j^{(\mathcal{N})}, \tau_i^{(\mathcal{U})}\right)$$

where $L(\tau^*, \tau) = 1 - \frac{2}{\pi} \arctan\left(\frac{\tau - \tau^*}{3600 * 6}\right)$, $L(\tau^*, \tau) = 1$ when $\tau = \tau^*$, and $L(\tau^*, \tau)$ gradually approaches 0 as $\tau - \tau^*$ increases. In real time, the latency weighting $L\left(\tau_j^{(\mathcal{N})}, \tau_i^{(\mathcal{U})}\right)$ aproaches 2 if u_i covers γ_j over 50 hours before $\tau_j^{(\mathcal{N})}$; $L\left(\tau_j^{(\mathcal{N})}, \tau_i^{(\mathcal{U})}\right)$ approaches 0 if u_i covers γ_j over 50 hours after $\tau_j^{(\mathcal{N})}$. Under the latency weighting, systems receive a gain bonus for covering a nugget γ_j before its reference time $\tau_j^{(\mathcal{N})}$ and a penalty for covering it after this time.

The first metric we use to evaluate a summary is the normalized expected gain,

$$n\mathbb{E}[G]\left(\mathcal{U}\right) = \frac{1}{Z|\mathcal{U}|} \sum_{\left(u_i, \tau_i^{(\mathcal{U})}\right) \in \mathcal{U}} G(u_i, \mathcal{U}, \mathcal{N})$$

where Z is maximum achievable expected gain (computed per query). This metric can be thought of as roughly analogous to precision. We also report a recall focused metric, called comprehensiveness, which is defined as

$$C\left(\mathcal{U}\right) = \frac{1}{|\mathcal{N}|} \sum_{\left(u_i, \tau_i^{(\mathcal{U})}\right) \in \mathcal{U}} G(u_i, \mathcal{U}, \mathcal{N})$$

which measures the percentage of nuggets covered by the update summary. We also report the harmonic mean of normalized expected gain and comprehensiveness,

$$\mathcal{H}(\mathcal{U}) = 2\frac{n\mathbb{E}[G]\,(\mathcal{U}) \times C(\mathcal{U})}{n\mathbb{E}[G]\,(\mathcal{U}) + C(\mathcal{U})}.$$

A latency-penalized version of normalized expected gain, comprehensiveness, and their harmonic mean can be obtained by replacing G with G_L in the above calculations.

4.2 Dataset

For all of our experiments in this chapter, we use data collected or prepared for the TREC 2013, 2014, and 2015 Temporal Summarization shared-tasks (Aslam et al., 2013, 2014, 2015). The task organizers provided both a corpus with which to create the document stream as well as sets of reference query events for training/evaluation.

The corpus for the document stream consisted of the 2014 TREC KBA Stream Corpus (Frank et al., 2012) which contains a 16.1 terabyte set of 1.2 billion timestamped documents crawled from the web between October, 2011 and February 2013.[1] The crawl includes a variety of news articles, forum data, and blog pages. We only use the news portion of this dataset in our experiments. Because the documents in this dataset are timestamped, we can simulate a document stream of online news for the 2011-2013 time period.

The reference query events were manually curated by the track organizers from world news events that were significant enough to have their own Wikipedia page. See Table 4.1 for the complete list of reference query events collected for the shared-task. The task organizers also created the query string and determined a suitable time period of interest, i.e. the time duration of the event, for all reference query events. For each event we create a stream of relevant documents from the KBA Stream Corpus by selecting only those documents that contain the complete set of query terms and whose timestamps fall within the period of interest.

Assessors at the National Institute of Standards and Technology (NIST) constructed a ground

[1] http://streamcorpus.org/

truth set of nuggets for each reference event by extracting important snippets from the introduction of the event's associated Wikipedia page. Assessors used the revision history to identify important nugget texts, and also used the revision timestamps to establish the reference timestamps $\tau_i^{(\mathcal{N})}$ for each nugget γ_i. More detail on this process can be found in the official shared-task description (Aslam et al., 2013).

4.3 The Salience-biased Affinity Propagation (SAP) Summarizer

We now present our first streaming summarization model, the salience-biased affinity propagation (SAP) summarizer. The SAP summarizer predicts sentence salience with respect to a query q, and integrates these predictions into a clustering based multi-document summarization system. We demonstrate that combining salience with clustering produces more relevant summaries compared to baselines using clustering or salience alone. Our experiments suggest that this is because our system is better able to adapt to dynamic changes in input volume that adversely affect methods that use redundancy as a proxy for salience.

In addition to the tight integration between clustering and salience prediction, our approach also exploits knowledge about the event to determine salience. Thus, salience represents both how typical a sentence is of the event category (e.g., industrial accident, hurricane, riot) and whether it specifies information about this particular event. Our feature representation includes a set of language models, one for each event category, to measure the typicality of the sentence with regard to the current event, the physical distance of mentioned locations from the center of the event, and the change in word frequencies over the time of the event. While we evaluate these features in the domain of disasters, this approach is applicable to any streaming summarization task.

We evaluate the SAP summarizer with two main experiments. First, we present the results of our model in the TREC 2014 Temporal Summarization shared-task (Aslam et al., 2014), where the SAP summarizer achieved top performance on the main evaluation metric ($\mathcal{H}$), and was also shown to have higher precision relative to other participant systems. Second, we perform our own independent evaluation, and show our approach achieves a statistically significant improvement in

Algorithm 2: Salience-biased Affinity Propagation (SAP) Summarizer

```
Input:  query string $q$, query category $c$, stream $\mathcal{S}$, period of interest $(\tau_{«s»}, \tau_{«e»})$
Output: update summary $\mathcal{U}$

/* Initialize empty update summary and system start time.
   */
$\mathcal{U} \leftarrow []$
$t \leftarrow 1$
$\hat{\tau}_0 \leftarrow \tau_{«s»}$
$\hat{\tau}_1 \leftarrow \tau_{«e»} + \delta_{hour}$
while $\hat{\tau}_t < \tau_{«e»}$ do
    /* Predict salience (§4.3.2)                                  */
    $\hat{\mathbf{y}}_t \leftarrow []$
    for $s_i \in \mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$ do
        $\hat{\mathbf{y}}_t \leftarrow \hat{\mathbf{y}} \oplus [f(\phi(s_i, q, c))]$
    /* Select exemplars with SAP clustering (§4.3.3)              */
    $\mathcal{E}_t \leftarrow$ APCluster($\mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}, \hat{\mathbf{y}}_t$)
    /* Select next updates (§4.3.4)                               */
    $\mathcal{U}_t \leftarrow$ FilterRedundant($\mathcal{E}_t, \mathcal{U}$)
    for $u \in \mathcal{U}_t$ do
        $\mathcal{U} \leftarrow \mathcal{U} \oplus [(u, \hat{\tau}_t)]$
    $\hat{\tau}_{t+1} \leftarrow \hat{\tau}_t + \delta_{hour}$
    $t \leftarrow t + 1$
```

ROUGE scores compared to multiple baselines in addition to the expected gain and comprehensiveness metrics. We also perform a feature ablation experiment to see which features are most important in our salience estimation component.

4.3.1 Summarization Model

The summarizer takes as input the query string q and category c, as well as the period of interest, $(\tau_{«s»}, \tau_{«e»})$, i.e., the time period of the stream on which to run the summarizer. The summarizer works by incrementing the system time in hourly batches, processing the newly observed sentences, and selecting some of them to be updates, which are then added to the update summary. When the system time exceeds $\tau_{«e»}$, the summarizer terminates, returning the completed update summary.

Pseudo-code for the SAP summarizer is shown in algorithm 2. The algorithm starts with an empty update summary $\mathcal{U}$ and initial system time $\hat{\tau}_1 = \tau_{«s»} + \delta_{hour}$ (Alg. 2 lines 1–4). System time is incremented in hourly intervals, i.e. $\hat{\tau}_t - \hat{\tau}_{t-1} = \delta_{hour}$. At each time $\hat{\tau}_t$, we process the sentences that entered the stream in the last hour, $\mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$, by performing the following actions:

1. predict the salience $\hat{y}_i$ of sentences $s_i \in \mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$ (Alg. 2 lines 7–9, §4.3.2),
2. select a set of exemplar sentences $\mathcal{E}_t \subset \mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$ by combining affinity propagation clustering with salience predictions (Alg. 2 line 11, §4.3.3),
3. add the most novel and salient exemplars, $\mathcal{U}_{\hat{\tau}_t}$, to the update summary $\mathcal{U}$ (Alg. 2 lines 13–15, §4.3.4).

Once the system time $\hat{\tau}_t$ exceeds the period of interest (i.e., $\hat{\tau}_t > \tau_{«e»}$), the summarizer returns collected set of updates $\mathcal{U}$ as the summary of the event.

4.3.2 Salience Estimation

4.3.2.1 Model

Given a sentence $s \in \mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$ from the current hourly batch, the salience estimation model determines how important or relevant it's content is with respect to the query. Since we do not have manual assessments of query-sentence salience for the overwhelming majority of the sentences in the KBA Stream Corpus, we rely on an automatic measure for determining the salience targets we wish to predict.

Let $\mathcal{N}$ be a timestamped collection of reference nuggets for query q. We define the salience of a sentence s with respect to q to be the degree to which it reflects an event's reference nuggets, which we define as its maximum nugget similarity,

$$\text{salience}\,(s, q) = \max_{\left(\gamma_i, \tau_i^{(\mathcal{N})}\right) \in \mathcal{N}} \text{similarity}\,(s, \gamma_i)\,. \tag{4.1}$$

We implement the similarity function using the cosine-similarity of latent-space vectors associated with s and γ_j using weighted matrix factorization (WMF) (Srebro and Jaakkola, 2003; Guo and Diab, 2012). Given a term-sentence matrix $\mathbf{K} \in \mathbb{R}^{D_w \times D_s}$ where $\mathbf{K}_{i,j}$ is the TF-IDF weight of term i in sentence j, WMF finds a low-rank approximation $\mathbf{P}^\intercal \mathbf{Q} \approx \mathbf{K}$ where $\mathbf{P} \in \mathbb{R}^{D_h \times D_w}$ and $\mathbf{Q} \in \mathbb{R}^{D_h \times D_s}$ are projection matrices into the latent term and document spaces respectively. The projection matrices are found by minimizing the weighted reconstruction error of $\mathbf{K}$ under a least squares objective, i.e.,

$$\mathcal{L}\left(\mathbf{P}, \mathbf{Q}\right) = \sum_{i=1}^{D_w} \sum_{j=1}^{D_s} \mathbf{W}_{i,j} \left(\mathbf{K}_{i,j} - \left(\mathbf{P}^\intercal \mathbf{Q}\right)_{i,j}\right)^2 + \text{reg.}$$

Following Guo and Diab (2012), we set the weight $\mathbf{W}_{i,j}$ to

$$\mathbf{W}_{i,j} = \begin{cases} 0.01 & \text{if } \mathbf{K}_{i,j} = 0, \\ 1 & \text{otherwise} \end{cases}$$

which focuses the reconstruction on non-zero entries in $\mathbf{K}$. The intuition here is that $\mathbf{K}$ is sparse so error in modeling the 0 entries, which we care least about, will dominate the loss. By down-weighting those entries, the projection matrices must better represent how terms are positively associated to the documents they occur in.[2] Let $\psi\left(s\right), \psi\left(\gamma\right) \in \mathbb{R}^{D_w}$ be projections into TF-IDF weighted bag-of-words space of s and γ respectively. The similarity of s and γ then is defined as,

$$\text{similarity}\left(s, \gamma\right) = \text{cosine-similarity}\left(\mathbf{P} \cdot \psi\left(s\right), \mathbf{P} \cdot \psi\left(\gamma\right)\right). \tag{4.2}$$

Since the summarizer will not have knowledge of the reference summary $\mathcal{N}$ at test time, we must estimate Equation 4.1 without it. To that end, we fit a regression model $f\left(\cdot\right)$ to estimate the

[2] WMF is similar to latent semantic analysis (LSA) (Dumais et al., 1988) which uses a truncated singular-value decomposition to obtain term and sentence projections but does not reweight non-zero entries. Interestingly, WMF is also similar to the global vector (GloVe) embedding method (Pennington et al., 2014) which minimizes a weighted least squares objective of a term-by-term log cooccurrence matrix.

salience of a sentence with respect to the query, i.e.,

$$\text{salience}\,(s, q) \approx f\,(\phi\,(s, q, c)) \tag{4.3}$$

using features, $\phi\,(s, q, c)$, of the sentence, query, and query category.

We opt to use a Gaussian process (GP) regression model (Rasmussen and Williams, 2005) with a radial basis function (RBF) kernel for the salience prediction task. Our features fall naturally into five groups (which we describe below) and we use a separate RBF kernel for each, using the sum of each feature group RBF kernel as the final input to the GP model.

Given our feature representation of the input sentences, we need only target salience values for model learning. For each query event in our training data, we sample a set of sentences and each sentence's salience is computed according to Equation 4.1. This results in a training set of sentences with their feature representations and target salience values, to which we fit the salience estimator.

4.3.2.2 Features

We want our model to be predictive of sentence salience across different event instances so we avoid event-specific lexical features. Instead, we extract features such as language model scores, geographic relevance, and temporal relevance from each sentence; these features in our initial model development were consistently helpful across specific event instances and categories.

Basic Features We employ several basic features that have been used previously in supervised models to rank sentence salience (Kupiec et al., 1995; Conroy and O'Leary, 2001). These include sentence length, the number of capitalized words normalized by sentence length, document position, and number of named entities. The data stream comprises text extracted from raw html documents; these features help to down-weight sentences that are not content (e.g. web page titles, links to other content) or more heavily weight important sentences (e.g., that appear in prominent positions such as paragraph initial or article initial).

Query Features Query features measure the relationship between the sentence and the event query and category. These include the number of query words present in the sentence in addition to the number of event category synonyms, hypernyms, and hyponyms using WordNet (Miller, 1995). For example, for event category *earthquake*, we match sentence terms “quake”, “temblor”, “seism”, and “aftershock”.

Language Model Features Language models allow us to measure the likelihood of producing a sentence from a particular source. We consider two different language models to obtain features. The first model is estimated from a corpus of generic news articles (we used the 1995-2010 Associated Press section of the Gigaword corpus (Graff and Cieri, 2003)). This model is intended to assess the general writing quality (grammaticality, word usage) of an input sentence and helps our model to select sentences written in the newswire style.

The second model is estimated from text specific to our event categories. For each event category we create a corpus of related documents using pages and subcategories listed under a related Wikipedia category. For example, the language model for event category *earthquake* is estimated from Wikipedia pages under *Category:Earthquakes*. Table 4.1 lists the event categories for each of the events in our dataset. These models are intended to detect sentences similar to those appearing in summaries of other events in the same category (e.g., most earthquake summaries are likely to include higher probability for ngrams including the token ‘magnitude’). While we focus our system on the language of news and disaster, we emphasize that the use of language modeling can be an effective feature for multi-document summarization for other domains that have related text corpora.

We use the SRILM toolkit (Stolcke, 2002) to implement a 5-gram Kneser-Ney model for both the background language model and the event category specific language models. For each sentence we use the average token log probability under each model as a feature.

Geographic Relevance Features The events in our corpus are all phenomena that affect some part of the world. Where possible, we would like to capture a sentence’s proximity to the event, i.e. when a sentence references a location, it should be close to the geographic area of the event.

There are two particular challenges to using geographic features in our present setting. First, we do not know where the event is, and second, most sentences do not contain references to a location. We address the first issue by extracting all locations (using contiguous token spans tagged with a location tag under a named-entity tagger) from documents relevant to the event at the current hour (i.e. $d_i \in \mathcal{D}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$) and looking up their latitude and longitude using a publicly available geo-location service. Since the documents are at least somewhat relevant to the event, we assume in aggregate the locations should give us a rough area of interest. The locations are clustered and we treat the resulting cluster centers as the event locations for the current time.[3]

The second issue arises from the fact that the majority of sentences in our data do not contain explicit references to locations. Our intuition is that geographic relevance is important in the disaster domain, and we would like to take advantage of the sentences that do have location information present. To make up for this imbalance, we instead compute an overall location for the document and derive geographic features based on the document's proximity to the event in question. These features are assigned to all sentences in the document.

Our method of computing document-level geographic relevance features is as follows. Using the locations in each document, we compute the median distance to the nearest event location. Because document position is a good indicator of importance we also compute the distance of the first mentioned location to the nearest event location. All sentences in the document take as features these two distance calculations. Because some events can move, we also compute these distances to event locations from the previous hour.

Temporal Relevance Features As we track events over time, it is likely that the coverage of the event may die down, only to spike back up when there is a breaking development. Identifying terms that are "bursty," i.e. suddenly peaking in usage, can help to locate novel sentences that are part of the most recent reportage and have yet to fall into the background.

We compute the IDF_t for each hour sub-sequence, $\mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$. For each sentence $s \in \mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$,

[3] The great-circle distance (`https://en.wikipedia.org/wiki/Great-circle_distance`), which is the shortest arc between two points projected onto the surface of a sphere, is used as the distance metric for clustering. Clustering is done with affinity propagation clustering.

the average TF-IDF$_t$ is taken as a feature. Additionally, we use the difference between average TF-IDF$_t$ and average TF-IDF$_{t-i}$ for $i \in \{1, \ldots, 24\}$ to measure how the TF-IDF scores for the sentence have changed over the last 24 hours, i.e. we keep the sentence term frequencies fixed and compute the difference in IDF. Large changes in IDF value indicate the sentence contains bursty terms. We also use the time (in hours) since the event started as a feature.

4.3.3 Affinity Propagation Clustering

Once we have predicted the salience for a batch of sentences, we must now select a set of update candidates, i.e. sentences that are both salient and representative of the current batch. To accomplish this, we combine the output of our salience prediction model with the affinity propagation clustering algorithm (Frey and Dueck, 2007).

Affinity propagation (AP) identifies a subset of data points as exemplars and forms clusters by assigning the remaining points to one of the exemplars. Let

$$\mathcal{X} = \{s_1, \ldots, s_n\}$$

be a set of n data points to be clustered and let

$$\boldsymbol{\varepsilon} = [\varepsilon_1, \ldots, \varepsilon_n]$$

be a vector of corresponding exemplar assignments, i.e. $\varepsilon_i \in \{1, \ldots, n\}$ and $\varepsilon_i = k$ means that s_k is the exemplar for s_i.

AP attempts to maximize the constrained net similarity objective,

$$\mathcal{L}(\boldsymbol{\varepsilon}) = e^{\left(\sum_{i=1}^{n} \operatorname{sim}\left(s_i, s_{\varepsilon_i}\right) + \sum_{k=1}^{n} \log g_k(\boldsymbol{\varepsilon})\right)}$$

where $\text{sim}\left(s_i, s_{\varepsilon_i}\right) \in \mathbb{R}$ measures the affinity of s_i for its exemplar s_{ε_i} and

$$g_k(\boldsymbol{\varepsilon}) = \begin{cases} 0 & \varepsilon_k \neq k \text{ but } \exists i : \varepsilon_i = k \\ 1 & \text{otherwise} \end{cases}$$

expresses the constraint that if s_k is some point's exemplar, it must be its own exemplar, i.e. all clusters must have one exemplar. The affinities can be interpreted as a log-probabilities of the exemplar-assignments, i.e. $p(\boldsymbol{\varepsilon}) = \frac{\mathcal{L}(\varepsilon)}{\sum_{\varepsilon'} \mathcal{L}(\varepsilon')}$. Frey and Dueck (2007) show the net similarity objective can be viewed as a factor graph and an optimal configuration can be found using max-product message passing. For the affinity function sim, we use

$$\text{sim}\left(s_i, s_{\varepsilon_i}\right) = \begin{cases} f\left(\phi\left(\mathbf{x}_i, q, c\right)\right) & \text{if } i = \varepsilon_i \\ \text{similarity}\left(s_i, s_{\varepsilon_i}\right) & \text{otherwise} \end{cases}$$

where $f\left(\phi\left(s_i, q, c\right)\right)$ is the salience estimate of sentence s_i (Equation 4.3) and $\text{similarity}\left(s_i, s_{\varepsilon_i}\right)$ is the WMF-based similarity method (Equation 4.2).

AP has several useful properties relevant to our stream summarization task. Chiefly, the number of clusters k is not a model hyper-parameter. Given that our task requires clustering many batches of data with potentially large variations in the volume of data per batch, searching for an optimal k would be computationally prohibitive. With AP, k is determined by the self-affinity, $\text{sim}(s_i, s_i)$, of the data, with lower overall values of $\text{sim}(s_i, s_i)$ yielding a smaller number of clusters. When the volume of input is high but the salience predictions are low, the self-affinity term will guide AP toward a solution with fewer clusters; *vice-versa* when input is very salient on average but the volume of input is low. The adaptive nature of our model differentiates our method from most other update summarization systems.

In the summarization pipeline, given a batch of sentences $\mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}$ and their salience estimates $\hat{\mathbf{y}}_t$, we run the AP clustering algorithm and obtain the set of exemplars $\mathcal{E}_t = \text{APCluster}(\mathcal{S}_{\hat{\tau}_{t-1}:\hat{\tau}_t}, \hat{\mathbf{y}}_t)$. The exemplars $\mathcal{E}_t$ are then passed to a redundancy filtering stage which we describe next.

4.3.4 Redundancy Filtering and Update Selection

The exemplar sentences from the exemplar selection stage are the most salient and representative of the input for the current hour. However, we need to reconcile these sentences with updates from the previous hours to ensure that the most salient and least redundant updates are selected. To ensure that only the most salient updates are selected we apply a minimum salience threshold; after exemplar sentences have been identified, any exemplars whose salience is less than λ_{sal} are removed from consideration.

Next, to prevent adding updates that are redundant with previous output updates, we filter out exemplars that are too similar to previous updates. The exemplars are examined sequentially in order of decreasing salience and a similarity threshold is applied, where the exemplar is ignored if its maximum semantic similarity to any previous updates in the summary is greater than λ_{sim}. Exemplars that pass these thresholds (indicated as $\mathcal{U}_t$ in Alg. 2 line 13) are selected as updates and added to the summary.

4.3.5 TREC 2014 Experiments and Results

We submitted three different model variations to the 2014 TREC Temporal Summarization shared-task (Aslam et al., 2015). The models were submitted under the Team ID *cunlp* and given the following Run IDs:

1. 1APSalRed — the SAP summarizer where salience predictions are penalized by similarity to prior updates. Let $\hat{y}_k = f\left(\phi\left(u_k, q, c\right)\right)$ for each update in $\mathcal{U}$. The salience estimate for a new sentence s_i is $f\left(\phi\left(s_i, q, c\right)\right) - \sum_{\left(u_k, \tau_k^{(\mathcal{U})}\right) \in \mathcal{U}} \hat{y}_k \cdot \text{similarity}\left(s_i, u_k\right)$.

2. 2APSal — the SAP summarizer.

3. 3AP — the summarizer with no salience estimation, all non-singleton exemplars are passed on to the redundancy filter and update selection stage.

We use the 10 TREC 2013 events to train/tune our submitted models. Three of the events were selected at random as a development set. All system salience and similarity threshold parameters

TeamID	RunID	$n\mathbb{E}[G](\mathcal{U})$	$C(\mathcal{U})$	$C_L(\mathcal{U})$	$\mathcal{H}_L(\mathcal{U})$
cunlp	**2APSal**	0.0631	0.3220	1.2068	**0.1162**
BJUT	Q1	**0.0657**	0.4088	1.1491	0.1110
BJUT	Q2	0.0632	0.3979	1.1669	0.1091
BJUT	Q0	0.0632	0.3979	1.1669	0.1091
uogTr	uogTr2A	0.0467	0.4453	1.2322	0.0986
uogTr	uogTr4AC	0.0347	0.4539	1.2751	0.0793
uogTr	uogTr4ARas	0.0387	0.3691	1.2328	0.0772
IRIT	KW30H5NW3600	0.0383	0.3521	1.2221	0.0723
IRIT	KW30H5NW300	0.0378	0.3538	1.2208	0.0714
uogTr	uogTr4A	0.0281	**0.4733**	1.2522	0.0677
average		0.0327	0.3615	1.2943	0.0620
IRIT	KW80H5NW3600	0.0289	0.3764	1.2191	0.0604
IRIT	KW30H10NW300	0.0298	0.3780	1.2617	0.0602
cunlp	**1APSalRed**	0.0325	0.3058	1.1507	0.0602
IRIT	KW80H5NW300	0.0285	0.3806	1.2164	0.0596
ICTNET	run3	0.0531	0.1081	0.7004	0.0530
BUPT_PRIS	Cluster4	0.0155	0.2692	1.9140	0.0508
IRIT	KW80H10NW300	0.0225	0.4012	1.2621	0.0503
BUPT_PRIS	Cluster3	0.0115	0.3380	1.9165	0.0407
cunlp	**3AP**	0.0174	0.4265	1.3689	0.0403
ICTNET	run2	0.0418	0.0934	0.6266	0.0311
BUPT_PRIS	Cluster2	0.0059	0.3728	1.9170	0.0222
ICTNET	run4	0.0079	0.4070	1.2364	0.0178
ICTNET	run1	0.0070	0.4090	1.2314	0.0160
BUPT_PRIS	Cluster1	0.0033	0.4369	**1.9175**	0.0127

Table 4.2: Official TREC 2014 Temporal Summarization shared-task results using manual update/nugget matches.

are tuned on the development set to maximize Rouge-2 F1 scores with respect to the reference summaries $\mathcal{N}$. We fit separate salience models using 1,000 sentences randomly sampled from each of the seven remaining TREC 2013 query events using its associated document stream. When predicting the salience for a new sentence at test time, we use the average prediction of all seven models.

Participant systems were run on 15 query events and associated document streams curated for the evaluation. Updates from the participant systems were pooled and manually matched to reference nuggets by NIST assessors. These matches were used to compute the shared-task evaluation metrics (§4.3.1): normalized expected gain, $n\mathbb{E}[G](\mathcal{U})$, comprehensiveness $C(\mathcal{U})$, latency-penalized comprehensiveness $C_L(\mathcal{U})$, and the overall track evaluation measure, the harmonic mean $\mathcal{H}_L(\mathcal{U})$ of the latency-penalized $n\mathbb{E}[G](\mathcal{U})$ and $C(\mathcal{U})$ variants.

To give a qualitative flavor of our produced summaries, we show excerpts of two summaries produced by the 2APSal, i.e. SAP, model, in Figure 4.1. Results from the official TREC 2014 Temporal Summarization shared-task are shown in Table 4.2. We see that our 2APSal model had the highest overall $\mathcal{H}_L(\mathcal{U})$. It was also one of the most precise models along with BJUT runs Q1, Q2, and Q0. The 2APSal and the BJUT runs return far fewer updates on average while still managing to include updates that matched to nuggets. The 2APSal run returned 381.4 updates per query on average while the overall track average was 8,528.55 updates per query on average.

When we look at 3AP, which had no salience component, it had higher recall (i.e. comprehensiveness) than 2APSal but was much less precise, returning 5,967 updates on average. This suggests that the salience estimation was important for guiding the summarizer.

The performance of 1APSalRed was between 3AP and 2APSal. The redundancy penalty meant that the average self-similarities were flatter than those of 2APSal yielding more diverse updates, which were not as likely to be matched to ground-truth nuggets, and therefore hurting the expected gain metrics.

All three of our proposed models had a latency-penalized comprehensiveness, $C_L(\mathcal{U})$, above one (1.1507 – 1.3689). Values above one indicate the average update that matched a nugget was selected before that information made it into Wikipedia (and confusingly getting a latency reward for early returns). In this case our proposed model was finding nugget information around three to five hours ahead of its publication in Wikipedia.

4.3.6 Automatic Experiments

Independent of the official TREC shared-task evaluation we also perform our own experiments using automatically computed metrics. In particular, we compute ROUGE with respect to the reference summaries $\mathcal{N}$, and expected gain and comprehensiveness computed using automatically

Hurricane Sandy

- The forecast map shows Sandy reaching eastern Cuba by early Thursday before heading to the Bahamas.
- Jamaica’s government issued a hurricane warning on Tuesday morning and announced schools would be closed on Wednesday.
- Dangerous flash floods and mudslides set off by Sandy were a threat for the island of roughly 2.7 million inhabitants, Jamaica’s meteorological service said.
- A few reliable forecast models bring Sandy close enough to the coast to produce heavy rains, strong winds and beach erosion.
- Max winds are 65 mph with strengthening to a hurricane expected in the next 12 to 18 hours.
- The two international airports, as well as schools and businesses, will remain closed today until the hurricane warning, which is now in effect for the island, is lifted.

2012 Pakistan Garment Factory Fires

- The fire broke out when people in the building were trying to start their generator after the electricity went out.
- Pakistani television showed pictures of what appeared to be a three-story building with flames leaping from the top-floor windows and smoke billowing into the night sky.
- The people went to the back side of the building but there was no access, so we had to made forceful entries and rescue the people, said Numan Noor, a firefighter on the scene.
- “We have recovered 63 bodies, including three found when we reached the basement of the building,” Karachi fire chief Ehtesham Salim told AFP on Wednesday.
- Salim added that the blaze was Karachi’s “biggest fire in terms of deaths in decades.”
- The garment trade as a whole is vital to Pakistan’s shaky economy.

2012 Romanian Protests

- Clashes between riot police and demonstrators have also erupted in the Romanian capital Bucharest for a third day in a row.
- BOC urged Romanians to understand that tough austerity measures are needed to avoid a default.
- More than 1,000 protesters rallied in Bucharest’s main university square, blocking traffic.
- Bucharest : a Romanian medical official says 59 people suffered injuries as days of protests against the government and austerity measures turned violent.
- Most Romanians agree that only fundamental reform can save the country’s ailing and corrupt system.
- Many Romanians feared this would only increase corruption and create a further divide between the classes, leading to a two-tier system in which only the wealthy would be able to afford care.

Figure 4.1: SAP summary excerpts.

matched update-nugget pairs.

We evaluate our system on two metrics: ROUGE and the unnormalized expected gain, $\mathbb{E}[G](\mathcal{U})$, and the comprehensiveness, $C(\mathcal{U})$ using automatically matched updates/nuggets. We also perform a feature ablation study and evaluate the resulting performance on ROUGE.

In this evaluation, we use the TREC 2013 and 2014 events, 25 events in total. One of the events is not actually covered by the KBA Stream Corpus so we discard. From the remaining 24, we set aside three events to use as a development set. All system salience and similarity threshold parameters are tuned on the development set to maximize ROUGE-2 F1 scores. We train a salience model for each event using 1000 sentences randomly sampled from the event's document stream. We perform a leave-one-out evaluation of each event. At test time, we predict a sentence's salience using the average predictions of the 23 other models.

ROUGE Evaluation Model summaries for each event were constructed by concatenating the event's nuggets. Generally, ROUGE evaluation assumes both model and system summaries are of a bounded length. Since our systems are summarizing events over a span of two weeks time, the total length of our system output is much longer than the model. To address this, for each system/event pair, we sample with replacement 1,000 random summaries of length less than or equal to the model summary (truncating the last sentence when neccessary). The final ROUGE scores for the system are the average scores from these 1,000 samples.

Because we are interested in system performance over time, we also evaluate systems at 12 hour intervals using the same regime as above. The reference summaries in this case are retrospective (i.e., covering the entire period of interest), and this evaluation reveals how quickly systems can cover information in the reference.

Expected Gain and Comprehensiveness In order to compute the expected gain and comprehensiveness, we need to know which updates cover which nuggets. While this was done manually with an extensive pool of manual annotators for the TREC shared task, we determine the covering automatically. A cover exists if the WMF-based similarity of the update/nugget pair is above a threshold. Determining an optimal threshold to count covers is difficult so we evaluate at threshold values ranging from .5 to 1, where values closer to 1 are more conservative estimates of performance. A manual inspection of matches suggests that similarity values around .7 produce reasonable matches. The average similarity of manual matches performed by NIST assessors was much lower at approximately .25, increasing our confidence in the automatic matches in the .5–1 range.

Note that we report expected gain here and not normalized expected gain. This allows us to use the more intuitive interpretation of expected gain which is the average number of unique, non-redundant nuggets each update covers. For example, $\mathbb{E}[G](\mathcal{U}) = 1$ would indicate on average that each update covered one novel nugget, while $\mathbb{E}[G](\mathcal{U}) = 0.5$ would mean on average every two updates covers a novel nugget.

4.3.6.1 Model Comparisons

We refer to our complete model as SAP. We compare this model against several variants and baselines intended to measure the contribution of different components. All thresholds for all runs are tuned on the development set.

Affinity Propagation only (AP) The purpose of this model is to directly measure the effect of integrating salience and clustering by providing a baseline that uses the identical clustering component, but without the salience information. In this model, input sentences are *a priori* equally likely to be exemplars; the self-affinity values are uniformly set as the median value of the input similarity scores, as is commonly used in the AP literature (Frey and Dueck, 2007). After clustering a sentence batch, the exemplars are examined in order of increasing time since event start and selected as updates if their maximum similarity to the previous updates is less than λ_{sim}, as in the novelty filtering stage of SAP.

Hierarchical Agglomerative Clustering (HAC) We provide another clustering baseline, single-linkage hierarchical agglomerative clustering. We include this baseline to show that SAP is not

	ROUGE-1			ROUGE-2		
Model	Recall	Prec.	F_1	Recall	Prec.	F_1
SAP	**0.282**	**0.344**	**0.306**	**0.045**	**0.056**	**0.049**
AP	0.245	0.285	0.263	0.033	0.038	0.035
RS	0.230	0.271	0.247	0.031	0.037	0.034
HAC	0.169	0.230	0.186	0.017	0.024	0.019

Table 4.3: Model ROUGE performance.

just an improvement over AP but centrality driven methods in general. HAC was chosen over other clustering approaches because the number of clusters is not an explicit hyper-parameter. To produce flat clusters from the hierarchical clustering, we flatten the HAC dendrogram using the cophenetic distance criteria, i.e. observations in each flat cluster have no greater a cophenetic distance than a threshold. Cluster centers are determined to be the sentence with highest cosine similarity to the flat cluster mean. Cluster centers are examined in time order and are added to the summary if their similarity to previous updates is below a similarity threshold λ_{sim} as is done in the AP model.

Rank by Salience (RS) We also isolate the impact of our salience model in order to demonstrate that the fusion of clustering and salience prediction improves over predicting salience alone. In this model we predict the salience of sentences as in step 1 for SAP. We omit the clustering phase (step 2). Updates are selected identically to step 3 of SAP, proceeding in order of decreasing salience, selecting updates that are above a salience threshold λ_{sal} and below a similarity threshold λ_{sim} with respect to the previously selected updates.

4.3.7 Results

ROUGE Table 4.3 shows our results for system output samples against the full summary of nuggets using ROUGE. SAP improves over the individual component systems, i.e. affinity propagation only (AP) or salience prediction only (RS), suggesting the combination of these two components is beneficial. This improvement is statistically significant for ROUGE $-$ 1 and ROUGE $-$ 2

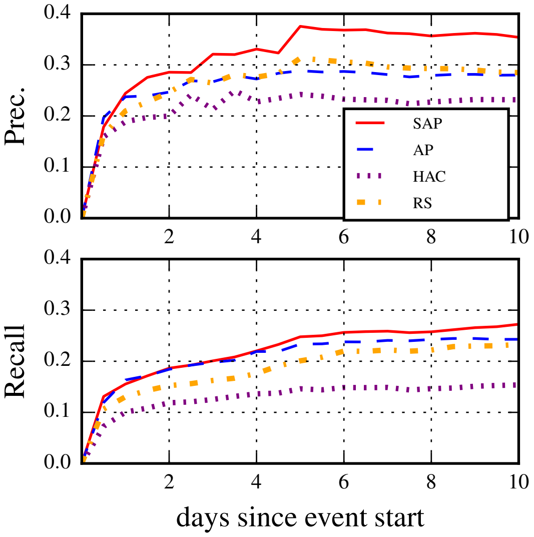


Figure 4.2: System ROUGE-1 performance over time.

precision, recall, and F-measures at the $\alpha = .01$ level using the Wilcoxon signed-rank test. The full system or the individual components AP and RS also outperform the alternative clustering method HAC.

SAP maintains its performance above the baselines over time as well. Figure 4.2 shows the ROUGE-1 scores over time. We show the difference in unigram precision (bigram precision is not shown but it follows similar curve). Within the initial days of the event, SAP is able to take the lead over the other systems in ngram precision. The SAP model is better able to find salient updates earlier on; for news and crisis informatics, this is an especially important quality of the model. Moreover, the SAP's recall is not diminished by the high precision and remains competitive with AP. Over time SAP's recall also begins to pull away, while the other models begin to plateau.

Expected Gain and Comprehensiveness Figure 4.3 shows the expected gain across a range of similarity thresholds, where thresholds closer to 1 are more conservative estimates. The ranking of the systems remains constant across the sweep with SAP beating all baseline systems. Predicting salience in general is helpful for keeping a summary on topic as the RS approach outperforms the clustering only approaches on expected gain.

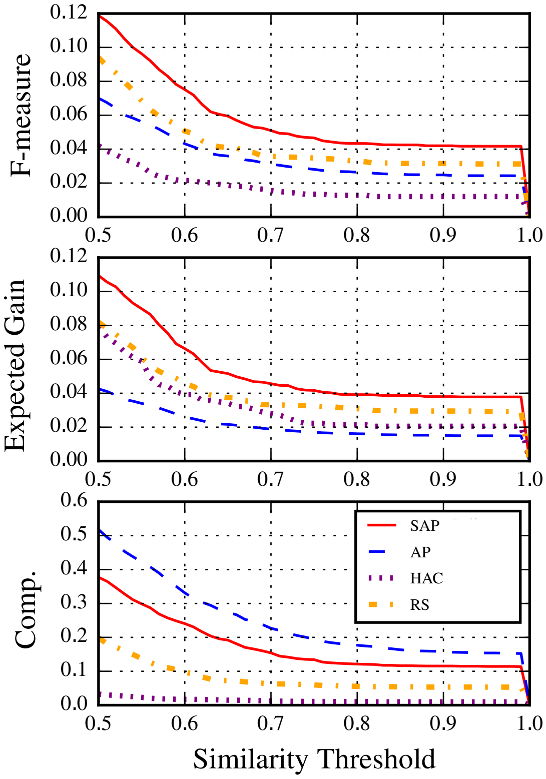


Figure 4.3: Expected Gain and Comprehensiveness performance.

When looking at the comprehensiveness of the summaries AP outperforms SAP. The compromise encoded in the SAP objective function, between being representative and being salient, is seen clearly here where the performance of the SAP methods is lower bounded by the salience focused RS system and upper bounded by the clustering only AP system. Overall, SAP achieves the best balance of these two metrics.

4.3.8 Feature Ablation

Table 4.4 shows the results of our feature ablation tests. Removing the language models yields a statistically significant drop in both ngram recall and F-measure.

Removing the language model and geographic relevance features leads to a statistically significant drop in ROUGE-1 F1 scores. Unfortunately, this is not the case for the temporal relevance

	ROUGE-1			ROUGE-2		
Model	Recall	Prec.	F_1	Recall	Prec.	F_1
Full System	0.282	0.344	0.306	0.045	0.056	0.049
No Basic	0.263	$0.380^{\dagger}$	0.294	0.046	$0.068^{\dagger\dagger}$	$0.051^{\dagger}$
No LM	$0.223^{\dagger}$	0.361	$0.254^{\dagger}$	$0.033^{\dagger}$	0.056	$0.038^{\dagger}$
No Time	$0.297^{\dagger}$	$0.367^{\dagger\dagger}$	$0.322^{\dagger}$	$0.052^{\dagger\dagger}$	$0.064^{\dagger\dagger}$	$0.056^{\dagger\dagger}$
No Geo	$0.232^{\dagger\dagger}$	0.381	$0.265^{\dagger}$	$0.037^{\dagger}$	0.065	0.042
No Query	0.251	0.377	0.280	0.043	$0.068^{\dagger}$	0.048

Table 4.4: Feature ablation ROUGE performance. † indicates statistically significant difference from full model at the $\alpha = .05$ level. †† indicates statistically significant difference from full model at the $\alpha = .01$ level.

features. We surmise that these features are too strongly correlated with each other, i.e. the differences in TF-IDF between hours are definitely not i.i.d. variables.

Interestingly, removing the basic features leads to an increase in both unigram and bigram precision; in the bigram case this is enough to cause a statistically significant increase in F-measure over the full model. In other words, the generic features actually lead to an inferior model when we can incorporate more appropriate domain specific features. This result perhaps echoes the claim of Spärck Jones (1999) that generic approaches to summarization are unlikely to produce truly useful summaries.

4.4 Learning-to-Search (L2S) Summarizer

While our previous summarization model proved reasonably capable of summarizing events over time, by processing the stream in hourly batches it was limited in its ability to respond quickly to unfolding events. One reason for this limitation is an implicit assumption in that model, and most multi-document summarization models, that frequency of a unit of text is a proxy for its salience. In retrospective summarization of static document collections, this is a reasonable assumption, and has been exploited successfully in various ways: TF-IDF term weightings, document language models derived from frequency, and random-walks on sentence-graphs whose edges are determined by frequently co-occurrring terms (Lin and Hovy, 2000; Radev et al., 2000; Erkan and

Radev, 2004; Mihalcea and Tarau, 2004; Nenkova and Vanderwende, 2005; Daumé III and Marcu, 2006).

In the streaming setting these proxy features are constantly evolving. There may be fallow periods in an event where nothing happens and then sudden bursts of activity. The behavior of SAP is unsuited for this, in that it is run every hour regardless. In the SAP model, by collecting an hour's worth of documents and performing a salience-biased clustering, we try to walk the line between using clustering, where the frequency of like text units are effectively votes for the most salient unit, and predictions about salience from our regression model, which makes more use of the semantics of the query event and text unit under analysis. However, as we showed in the feature ablation, incorporating time-based frequency features made the model worse. While we are able to incorporate salience estimate successfully into the summarization model with SAP, we have yet to successfully provide a learning-based of model the entire summarization process.

In this section, we attempt to overcome these limitations, removing the clustering component from the update selection, and develop a fully online streaming summarization model, one that learns to make extraction decisions immediately upon seeing the next sentence from $\mathcal{S}$, using features derived from the entire observed document stream, the state of the current update summary, and the model's prior extraction decisions. To that end, we present a novel streaming-document summarization system based on sequential decision making. Specifically, we adopt the "learning to search" approach, a technique which adapts methods from reinforcement learning for structured prediction problems (Daumé III et al., 2009; Ross and Bagnell, 2010). In this framework, we cast streaming summarization as a form of greedy search and train our system to imitate the behavior of an oracle summarization system. Effectively, we train a linear classifier to predict when to extract a sentence $s \in \mathcal{S}$ using features of the sentence, query, previous summary updates, and other aggregate statistics collected from the stream up to the current time $\hat{\tau}_t$.

As in the previous section, we report both the TREC 2015 Temporal Summarization shared-task manual evaluation as well as our own independent automatic evaluation. In our evaluation we demonstrate a 28.3% improvement in summary $\mathcal{H}_L(\mathcal{U})$ and a 43.8% improvement in time-sensitive $\mathcal{H}_L(\mathcal{U})$ metrics against several state-of-the-art baselines. In shared-task evaluation of our system at TREC 2015, where we were the second-best performing team in the main evaluation, and top system for a secondary evaluation run on a pre-filtered, highly relevant document stream.

4.4.1 Stream Summarization as Sequential Decision Making

We could naïvely treat the stream summarization problem as classification and predict which sentences to extract or skip. However, this would make it difficult to take advantage of many features (e.g. sentence novelty with regard to previous updates). What is more concerning, however, is that the classification objective for this task is somewhat ill-defined: successfully predicting extract on one sentence changes the true label (from extract to skip) for sentences that contain the same information but occur later in the stream.

In this work, we pose stream summarization as a greedy search over a binary branching tree where each level corresponds to a position in the stream (see Figure 4.4). The height of the tree corresponds to the length of stream. A path through the tree is determined by the system extract and skip decisions.

When treated as a sequential decision making problem, our task reduces to defining a policy for selecting a sentence based on its features as well as the features of its ancestors (i.e., all of the observed sentences and previous extraction decisions). The union of these features represents the current state in the decision making process.

The feature representation provides state abstraction both within a given query's search tree as well as to states in other queries' search trees, and also allows for complex interactions between the current update summary, candidate sentences, and stream dynamics unlike the classification approach.

In order to learn an effective policy for a query q, we can take one of several approaches. We could use a simulator to provide feedback to a reinforcement learning algorithm. Alternatively, if provided access to an evaluation algorithm at training time, we can simulate (approximately) optimal decisions. That is, using the training data, we can define an *oracle policy* that is able to

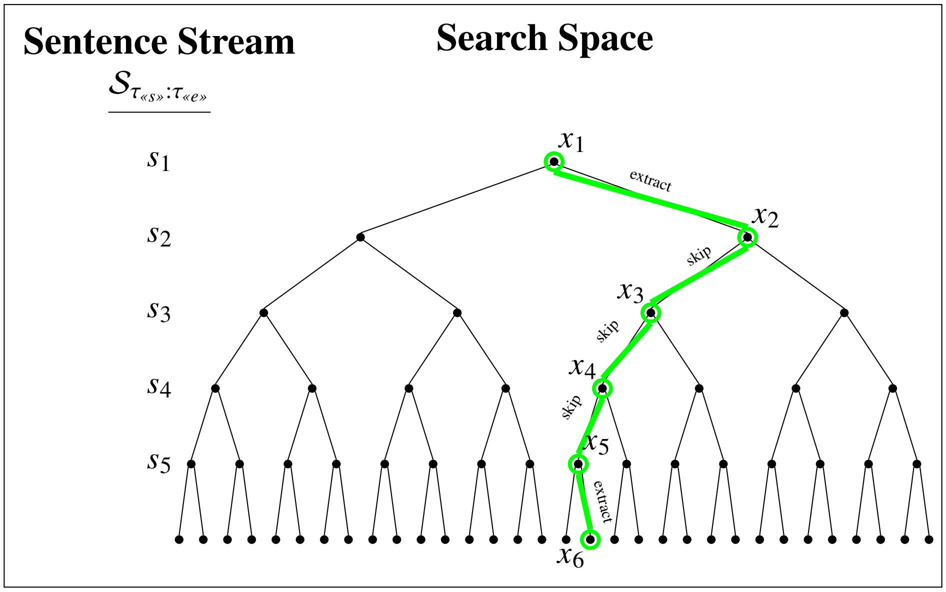


Figure 4.4: Search space for a stream of 5 sentences. Left branches skip the current sentence. Right branches indicate extracting the current sentence as an update. The path in green corresponds to one trajectory through this space consisting of extracting sentence s_1, then skipping sentences $s_2, \ldots, s_4$ and extracting sentence s_5.

omnisciently determine which sentences to extract and which to skip. Moreover, it can make these determinations by starting at the root or at an arbitrary node in the tree, allowing us to observe optimal performance in states unlikely to be reached by the oracle. We adopt *locally optimal learning to search* to learn our model from the oracle policy (Chang et al., 2015).

4.4.2 Policy-based Stream Summarization

In the induced search problem, each search state $x_t \in \mathcal{X}$ corresponds to observing the first t sentences in the stream $[s_1, \ldots, s_t] \subset \mathcal{S}$ and a sequence of $t - 1$ actions $y_1, \ldots, y_{t-1}$. For each state $x_t \in \mathcal{X}$, the set of actions is $y_t \in \{0, 1\}$ with $y_t = 1$ indicating we extract the t-th sentence and add it to our update summary, and $y_t = 0$ indicating we skip it. For simplicity of exposition, we assume a fixed length stream of size T but this is not strictly necessary.

A policy, $f : \mathcal{X} \rightarrow \{0, 1\}$, is a mapping from states to an extraction decision. Given a policy, the policy-based stream summarization algorithm (algorithm 3) is trivial, iterating sequentially over sentences in the stream, and adding each sentence to the update summary if it is the current

Algorithm 3: Policy-based Stream Summarization

```
Input: query string $q$, query category $c$, stream $\mathcal{S}$, period of interest ($\tau_{«s»}$, $\tau_{«e»}$),
       summarization policy $f$
Output: update summary $\mathcal{U}$
$\mathcal{U} \leftarrow \{\}$
for $s_t \in \mathcal{S}_{\tau_{«s»}:\tau_{«e»}}$ do
    $x_t \leftarrow (s_1, \ldots, s_t, y_1, \ldots, y_{t-1}, q, c)$
    $y_t \leftarrow f(x_t)$
    if $y_t = 1$ then
        $\mathcal{U} \leftarrow \mathcal{U} \cup \left\{\left(s_t, \tau_t^{(\mathcal{S})}\right)\right\}$
```

action determined by f. In practice, we use a linear cost-sensitive classifier to implement f_θ, with

$$y_t = f_\theta(x_t) = \operatorname*{arg\,min}_{y \in \{0,1\}} \phi(x_t, y) \cdot \theta_y$$

where we encode each state-potential action pair (x_t, y) as a d-dimensional feature vector $\phi(x_t, y) \in \mathbb{R}^d$ and $\theta_0, \theta_1 \in \mathbb{R}^d$ are learned parameters. Note that $\phi(x_t, y) \cdot \theta_y$ is a linear regression to predict the cost, which we define shortly, associated with taking action y in state x_t. Given estimates of our two available actions, extracting a sentence or ignoring it, our policy f_f selects the action with minimum cost.

Before we can define cost more concretely, we must first introduce some additional notation and concepts. For a given sequence of states $x_1, \ldots, x_T$ explored by a policy f, let $\boldsymbol{y} = [y_1, \ldots, y_T]$ be the associated sequence of actions taken by f, i.e. $y_t = f(x_t)$. A loss function, $\ell : \{0, 1\}^T \rightarrow \mathbb{R}$, measures the quality of an action sequence $\boldsymbol{y}$. In our present case this might be the negative ROUGE score of the update summary that results from $\boldsymbol{y}$ or some other relevant measure of performance.

Let f be a policy, $x_1, \ldots, x_T$ a sequence of states explored by f, and the corresponding action sequence $\boldsymbol{y}$. We also define a second policy, $f^{(o)}$, which we call the roll-out policy and which may or may not be distinct from f. For any state x_t we can then define two additional action sequences,

$$\hat{\boldsymbol{y}}^{(t,\hat{y},f^{(o)})} = \left[y_1, \ldots, y_{t-1}, \hat{y}_t, y_{t+1}^{(t,\hat{y},f^{(o)})}, \ldots, y_T^{(t,\hat{y},f^{(o)})},\right] \quad \forall \hat{y}_t \in \{0, 1\}$$

where $\hat{\boldsymbol{y}}^{\left(t,\hat{y},f^{(o)}\right)}$ is the sequence of actions that result from following f on the first $t-1$ states, taking action $\hat{y}_t$ at state x_t and then following $f^{(o)}$ on the remaining $T-t$ states. That is, for $k > t$, $y^{\left(,k,f^{(o)}\right)} = f^{(o)}(x'_k)$, where x'_k is the state that results from taking action $\hat{y}_t$ in x_t and following $f^{(o)}$ on a sequence of states $x'_{t+1}, \ldots, x'_k$. The loss, $\ell\left(\hat{\boldsymbol{y}}^{\left(t,0,f^{(o)}\right)}\right)$, then reflects the evaluation measure for an update summary where sentence s_t was not extracted, and $f^{(o)}$ completed the summary of the stream. Similarly, $\ell\left(\hat{\boldsymbol{y}}^{\left(t,1,f^{(o)}\right)}\right)$ reflects the evaluation measure for an update summary where sentence s_t was extracted, and $f^{(o)}$ completed the summary of the stream.

We can now define the cost of an action y in state x_t as

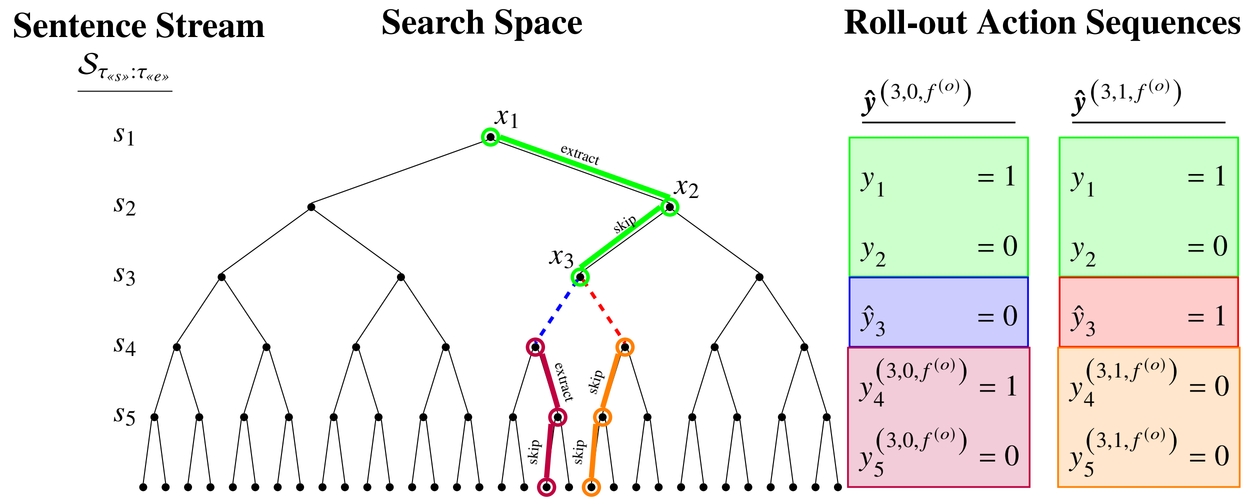
$$c(x_t, y) = \ell\left(\hat{\boldsymbol{y}}^{\left(t,y,f^{(o)}\right)}\right) - \min_{y' \in \{0,1\}} \ell\left(\hat{\boldsymbol{y}}^{\left(t,y',f^{(o)}\right)}\right).$$

Note that the cost is also a function of $f^{(o)}$, which determines how the action sequence is completed after x_t. The costs connect the overall summary loss $\ell\left(\hat{\boldsymbol{y}}^{\left(t,\hat{y},f^{(o)}\right)}\right)$ to a particular action in x_t that builds the summary. We depict an example of the cost computation in Figure 4.5. We discuss how to collect costs and learn θ_0 and θ_1 such that they are good estimators of cost in the next section.

4.4.3 The Locally-Optimal Learning-to-Search Algorithm

In a perfect world, we would have a training set of state-action pairs, (x_t, y_t), and their associated costs $c(x_t, y_t)$, drawn from a distribution of states similar to the one our final learned policy would produce. With these states, actions, and costs in hand we could fit two linear regressions, one for estimating the cost of extract actions and one for skip actions from a given state's feature representation; we would immediately have a suitable stream summarization policy. Unfortunately, we do not have a reasonable distribution of state-action pairs let alone the associated costs. Instead we turn to the locally-optimal learning-to-search (LOLS) algorithm (Chang et al., 2015), presented in algorithm 4, to iteratively refine our learned policy as it attempts to follow an oracle policy.

At a high level, we begin with an initially random learned policy. We leverage a heuristic oracle stream summarizer and our learned policy to collect state-action pairs and their costs from


Sentence Stream
Search Space
Roll-out Action Sequences
$S_{\tau_{«s»}:\tau_{«e»}}$
s_1
s_2
s_3
s_4
s_5
x_1
x_2
x_3
extract
skip
extract
skip
skip
skip
$\hat{\boldsymbol{y}}^{(3,0,f^{(o)})}$
$y_1 = 1$
$y_2 = 0$
$\hat{y}_3 = 0$
$y_4^{(3,0,f^{(o)})} = 1$
$y_5^{(3,0,f^{(o)})} = 0$
$\hat{\boldsymbol{y}}^{(3,1,f^{(o)})}$
$y_1 = 1$
$y_2 = 0$
$\hat{y}_3 = 1$
$y_4^{(3,1,f^{(o)})} = 0$
$y_5^{(3,1,f^{(o)})} = 0$

Computing Costs $c(x_3, 0)$ and $c(x_3, 1)$

Step 1. Roll-in to x_3 with policy f.
Use policy f to explore to state x_3, shown in green in the search space above.

Step 2. Roll-out with $f^{(o)}$ to create action sequences $\hat{\boldsymbol{y}}^{(3,0,f^{(o)})}$ and $\hat{\boldsymbol{y}}^{(3,1,f^{(o)})}$.
For each action $\hat{y}_3 \in \{0, 1\}$, use $f^{(o)}$ to complete the roll-outs after having made action $\hat{y}_3$ in state x_3 (shown in purple and orange respectively) and create alternative action sequences $\hat{\boldsymbol{y}}^{(3,0,f^{(o)})}$ and $\hat{\boldsymbol{y}}^{(3,1,f^{(o)})}$ (shown on the right).

Step 3. Compute losses.
After completing the roll-outs, compute losses $\ell\left(\hat{\boldsymbol{y}}^{(3,0,f^{(o)})}\right)$ and $\ell\left(\hat{\boldsymbol{y}}^{(3,1,f^{(o)})}\right)$.

Step 4. Compute costs.
Compute

$$c(x_3, 0) = \ell\left(\hat{\boldsymbol{y}}^{(3,0,f^{(o)})}\right) - \min_{y' \in \{0,1\}} \ell\left(\hat{\boldsymbol{y}}^{(3,y',f^{(o)})}\right)$$

and

$$c(x_3, 1) = \ell\left(\hat{\boldsymbol{y}}^{(3,1,f^{(o)})}\right) - \min_{y' \in \{0,1\}} \ell\left(\hat{\boldsymbol{y}}^{(3,y',f^{(o)})}\right).$$

Figure 4.5: Example of computing costs of actions at x_3 using roll-out policy $f^{(o)}$.

the training set query-streams, with our learned policy gradually learning to imitate the heuristic oracle. Using both the oracle and learned policy to sample state-action-cost triples is beneficial as it exposes the learned policy to a mix of states it is likely to encounter when following its own actions, but also state-actions of the better performing reference summarizer. Exploring with the learned policy alone may be less optimal because it may overestimate the costs of good decisions since its roll-outs at the beginning of training are likely to be quite bad. Exploring with only the oracle would also be harmful as the distribution of states it produces will reflect an optimistic level of performance from learned policy that it will not be able to match in practice.

4.4.3.1 Oracle Policy

For a given policy f, training query q, stream $\mathcal{S}$, and reference summary $\mathcal{N}$, we construct a greedy heuristic oracle policy f^*_q. Let $\mathcal{U}_t$ be the update summary at state x_t reached by f. Additionally, let

$$\mathcal{N}_{[\![\mathcal{U}_t]\!]} = \left\{ \gamma_i \middle| \left(\gamma_i, \tau_i^{(\mathcal{N})}\right) \in \mathcal{N}, \exists \left(u_j, \tau_j^{(\mathcal{U})}\right) \in \mathcal{U}_t : [\![\gamma_i]\!] \in [\![u_j]\!] \right\}$$

be the set of nuggets covered by the updates in the update summary at x_t, and let

$$\mathcal{N}_{[\![s_t]\!]} = \left\{ \gamma_i \middle| \left(\gamma_i, \tau_i^{(\mathcal{N})}\right) \in \mathcal{N}, [\![\gamma_i]\!] \in [\![s_t]\!] \right\}$$

be the set of nuggets covered by s_t. The oracle f^*_q, which has clairvoyant knowledge of these sets, performs the following actions given a state x_t,

$$f^*_q(x_t) = \begin{cases} 1 & \text{if } \exists \gamma : \gamma \in \mathcal{N}_{[\![s_t]\!]} \wedge \gamma \notin \mathcal{N}_{[\![\mathcal{U}_t]\!]} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, the oracle policy will extract s_t at state x_t if s_t covers any nuggets not yet covered by updates in the update summary $\mathcal{U}_t$. We describe how we obtain comprehensive nugget/sentence covers, i.e. the sets $\mathcal{N}_{[\![s_t]\!]}$ in §4.4.5.

4.4.3.2 Loss Function

We design our loss function to penalize policies that over- or under-generate. Let $x_1, \ldots, x_T$ be the sequence of states associated with the action sequence $\boldsymbol{y}$. We define the loss of a sequence as

$$\ell(\boldsymbol{y}) = 1 - 2 \times \frac{\boldsymbol{y}^{\intercal}\boldsymbol{y}^*}{\sum_{i=1}^{T} y_i + \sum_{j=1}^{T} y_j^*}$$

where $\boldsymbol{y}^* = \left[y_1^*, \ldots, y_T^*\right]$ is a reference sequence of consisting of the one-step optimal deviations under the oracle policy, i.e. $y_t^* = f_q^*(x_t)$.

Note that ℓ is the complement of the Dice coefficient. This encourages not only local agreement between policies (the numerator of the second term) but that the learned and oracle policy should generate roughly the same number of updates (the denominator in the second term).

4.4.3.3 Learning with LOLS

We now step through the LOLS algorithm in detail. Our initial policy parameters θ_0 and θ_1 are randomly initialized (Algorithm 4 line 1). The LOLS algorithm works by iteratively using the current learned policy f_θ to explore state sequences from a training stream $\mathcal{S}^{(q)}$ (Algorithm 4 line 6). At each state x_t, a roll-out policy $f^{(o)}$ is selected at random from f_θ and the oracle f_q^* (Algorithm 4 line 8). Losses are then computed for each continuation sequence $\hat{\boldsymbol{y}}^{\left(t,0,f^{(o)}\right)}$ and $\hat{\boldsymbol{y}}^{\left(t,1,f^{(o)}\right)}$ (Algorithm 4 line 9). Costs for each action are then computed and state features and costs are cached in Γ (Algorithm 4 lines 11–12). After the roll-in has explored all T states, we update the policy parameters θ by performing gradient descent on the squared error of the linear regressor on the cached feature-cost pairs in Γ (Algorithm 4 lines 13–14).

Algorithm 4: Locally Optimal Learning-to-Search for Stream Summarization

```
Input: training dataset of query-relevant streams and oracle policies $\{\mathcal{S}_q, f^*_q\}_{q \in Q}$, number
       of training epochs $N$, learning rate $\lambda$, and a mixture parameter $\beta \in (0, 1)$ for
       selecting a roll-out policy.
Output: learned summarization policy $f_\theta$
Initialize $\theta$
for $n \leftarrow 1, 2, \ldots, N$ do
    for $q \in Q$ do
        $\Gamma \leftarrow \{\}$
        for $t \in \{1, \ldots, T\}$ do
            Roll-in by executing $f_\theta$ on $\mathcal{S}_q$ for $t - 1$ rounds and reach state $s_t$.
            for $y \in \{0, 1\}$ do
                Pick roll-out policy $f^{(o)} \leftarrow \begin{cases} f^*_q & \text{with probability } \beta \\ f_\theta & \text{with probability } 1 - \beta. \end{cases}$.
                Compute roll-out action sequence $\boldsymbol{y}^{(t,y,f^{(o)})}$.
            for $y \in \{0, 1\}$ do
                $c(x_t, y) \leftarrow \ell\left(\boldsymbol{y}^{(t,y,f^{(o)})}\right) - \min_{y' \in \{0,1\}} \ell\left(\boldsymbol{y}^{(t,y',f^{(o)})}\right)$
                $\Gamma \leftarrow \Gamma \cup \{(\phi(x_t, y), c(x_t, y), y)\}$
        for $(\phi(x, y), c(x, y), y) \in \Gamma$ do
            $\theta_y \leftarrow \theta_y - \lambda \nabla_{\theta_y} (c(x, y) - \phi(x, y) \cdot \theta_y)^2$
Result: $f_\theta$
```

4.4.4 Features

As mentioned in the previous section, we represent each state as a feature vector. In general, at state x_t, these features are functions of the current sentence s_t, the stream history $s_1, \ldots, s_{t-1}$, the query string q and category c, and/or the decision history $y_1, \ldots, y_{t-1}$. We refer to features only determined by s_t, q, and c as static features and all others as dynamic features.

4.4.4.1 Static Features

Basic Features Our most basic features look at the length in words of a sentence, its position in the document, and the ratio of specific named entity tags to non-named entity tokens. We also compute the average number of sentence tokens that match the event query words and synonyms using WordNet.

Language Model Features Similar to §4.3, we compute the average token log probability of the sentence on two language models: i) an event category specific language model and ii) a general newswire language model. The first language model is built from Wikipedia articles relevant to the event-type domain. The second model is built from the New York Times and Associate Press sections of the Gigaword-5 corpus (Graff and Cieri, 2003).

Single Document Summarization Features These features are computed using the current sentence's document as a context and are also commonly used as ranking features in other document summarization systems. Where a sentence representation is needed, we use both TF-IDF bag-of-words representation as well as the latent vector representation under the WMF method described in §4.3.2.

We compute SumBasic features (Nenkova and Vanderwende, 2005): the average and sum of unigram probabilities in a sentence. We compute the arithmetic and geometric means of the sentence's cosine distance to the other sentences of the document (Guo et al., 2013). We refer to this quantity as novelty and compute it with both vector representations. We also compute the centroid rank (Radev et al., 2000) and LexRank of each sentence (Erkan and Radev, 2004), again using both vector representations.

Summary Content Probability For a subset of the stream sentences, we have manual judgements as to whether they match to reference summary nuggets or not. We use this data (restricted to sentences from the training query streams), to train a decision tree classifier, using the sentences' term ngrams as the classifier features. As this data is aggregated across the training queries, the purpose of this classifier is to capture the importance of general ngrams predictive of summary worthy content. Using this classifier, we obtain the probability that the current sentence $\mathbf{x}_t$ contains summary content and use this as a model feature.

4.4.4.2 Dynamic Features

Stream Language Models We maintain several unigram language models that are updated with each new document in the stream. Using these counts, we compute the sum, average, and maximum token probability of the non-stop words in the sentence. We compute similar quantities restricted to *person*, *location*, and *organization* named entities as well.

Update Similarity The average and maximum cosine similarity of the current sentence to all previous updates is computed under both the TF-IDF bag-of-words and latent vector representation. We also include indicator features for when the set of updates is empty (i.e. at the beginning of a run) and when either similarity is 0.

Document Frequency We also compute the hour-to-hour percent change in document frequency of the stream. This feature helps gauge breaking developments in an unfolding event. As this feature is also heavily affected by the daily news cycle (larger average document frequencies in the morning and evening) we compute the 0-mean/unit-variance of this feature using the training streams to find the mean and variance for each hour of the day.

Feature Interactions Many of our features are helpful for determining the importance of a sentence with respect to its document. However, they are more ambiguous for determining importance to the event as a whole. For example, it is not clear how to compare the document level PageRank of sentences from different documents. To compensate for this, we leverage two features which we believe to be good global indicators of update selection: the summary content probability and the document frequency. These two features are proxies for detecting (1) a good summary sentences (regardless of novelty with respect to other previous decisions) and (2) when an event is likely to be producing novel content. We compute the conjunctions of all previously mentioned features with the summary content probability and document frequency separately and together.

4.4.5 Expanding Relevance Judgments

Because of the large size of the corpus, less than 1% of the sentences received manual review by NIST assessors during the 2013-15 shared-task evaluations. In order to increase the amount of data for training and evaluation of our system, we augmented the manual sentence-nugget match judgements with automatic matches. Sentence-nugget matches are critical for our experiments because they enable us to compute evaluation metrics, but also the oracle summarization policy used in the LOLS algorithm.

In order to automatically tag sentences in the corpus with additional nugget matches, we trained a separate decision-tree classifier for each nugget with more than 10 manual sentence matches. Manually matched sentences were used as positive training data and an equal number of manually judged non-matching sentences were used as negative examples. Sentence n-grams (1-5), percentage of nugget terms covered by the sentence, semantic similarity of the sentence to nugget were used as features, along with an interaction term between the semantic similarity and coverage. After training the classifiers we used them to automatically tag corpus sentences with nugget matches. When augmenting the relevance judgments with these nugget match labels, we only include those that have a probability greater than 90% under the classifier. For evaluation, the summarization system only has access to the query and the document stream, without knowledge of any nugget matches (manual or automatic).

4.4.6 TREC 2015 Experiments and Results

There were three tasks at the TREC 2015 Temporal Summarization shared-task evaluation (Aslam et al., 2015):

1. **Full Filtering and Summarization** Participants must summarize a **very** high volume stream of documents where very few documents are likely to be similar to the query.

2. **Partial Filtering and Summarization** Participants must summarize a high volume stream of documents that has been pre-filtered for query relevance by an IR component developed by the task organizers.

3. **Summarization Only** Participants must summarize a low-volume stream of on-topic documents.

We participated in tasks 1 and 2, submitting, with Team ID *cunlp*, the following Run IDs:

- **Task 1. Full Filtering and Summarization**

  - 2LtoSnofltr20 – The L2S summarizer, where documents were truncated to there first 20 sentences.

- **Task 2. Partial Filtering and Summarization**
  - 1LtoSfltr20 – The L2S summarizer, where documents were truncated to the first 20 sentences.
  - 3LtoSfltr5 – The L2S summarizer, where documents were truncated to their first 5 sentences.
  - 4APSAL – Our SAP summarizer that was the overall best system at TREC 2014. We updated the salience predictions with additional training data from the TREC 2014 Temporal Summarization query events.

To train the L2S summarizer we randomly selected 3 events to be a development set and used the remaining 21 events from the 2013-2014 Temporal Summarization query events as our training set. The document streams for the events are unfortunately too large to be used directly with algorithm 4. In order to make training time reasonable yet representative, we downsample each stream to a length of 100 sentences. The downsampling is done uniformly over the entire stream. This is repeated 10 times for each training event to create a total of 210 training streams. In the event that a downsample contains no nuggets (either human or automatically labeled) we resample until at least one exists in the sample. We select the best model iteration based on the automatically computed $\mathcal{H}(\mathcal{U})$ on the development set. We show an example summary produced by the L2S system in Figure 4.6.

Table 4.5 shows the official results for task 1, the full filtering and summarization task. Our L2S run, 2LtoSnofltr20, was the top run in this track, although only one other team submitted runs due to the high computational cost of running on the large document stream. 2LtoSnofltr20 had the lowest precision (i.e., $n\mathbb{E}[G](\mathcal{U})$) of the submitted runs, and only achieved average recall (i.e., $C(\mathcal{U})$). However, it had lower latency, yielding the highest latency weighted comprehensiveness, $C_L(\mathcal{U})$; on the task's overall summary measure $\mathcal{H}_L(\mathcal{U})$, 2LtoSnofltr20 is ranked first, largely due

4:19 p.m.	Two explosions shattered the euphoria of the Boston Marathon finish line on Monday, sending authorities out on the course to carry off the injured while the stragglers were rerouted away...
4:31 p.m.	Police in New York City and London are stepping up security following explosions at the Boston Marathon.
4:31 p.m.	A senior U.S. intelligence official says two more explosive devices have been found near the scene of the Boston marathon where two bombs detonated earlier.
5:10 p.m.	Several candidates for Massachusetts' Senate special election have suspended campaign activity in response to the explosions...

Figure 4.6: Excerpt of the L2S summary for the query *boston marathon bombing* generated from an input document stream.

Team ID	Run ID	$n\mathbb{E}[G](\mathcal{U})$	$C(\mathcal{U})$	$C_L(\mathcal{U})$	$\mathcal{H}_L(\mathcal{U})$
cunlp	**2LtoSnofltr20**	0.1224	0.4691	**0.8086**	**0.1531**
CWI	IGnPrecision	0.1894	0.4678	0.6273	0.1396
average		0.1533	0.4575	0.6507	0.1279
CWI	IGn	0.1620	**0.5137**	0.6538	0.1248
CWI	docs	0.1242	0.4680	0.6658	0.1222
CWI	titles	**0.1915**	0.3107	0.5171	0.1150

Table 4.5: Official TREC 2015 Temporal Summarization Task 1 results using manual update/nugget matches.

to its low latency. Here we can see that fully online/greedy nature of L2S summarizer pays off in terms of latency, as salient content is identified relativey quickly once it has entered the stream.

Table 4.6 shows the official results for Task 2, the partial filtering and summarization task. The top runs by WaterlooClarke examined only the titles or first sentences of the documents, while our systems did not use titles, and used the first five or twenty sentences of documents (3LtoSfltr5 and 3LtoSfltr20 respectively). The WaterlooClarke runs had significantly higher $n\mathbb{E}[G](\mathcal{U})$ than the other systems but lower than average $C(\mathcal{U})$. Our L2S runs, 3LtoSfltr5 and 3LtoSfltr20, had higher $C(\mathcal{U})$, although not the highest overall. The fourth and five place overall performance (i.e., $\mathcal{H}_L(\mathcal{U})$) by 3LtoSfltr5 and 3LtoSfltr20 are due both to the high $C(\mathcal{U})$ but also low-latency at which it added important information to the summary.

Team ID	Run ID	$n\mathbb{E}[G](\mathcal{U})$	$C(\mathcal{U})$	$C_L(\mathcal{U})$	$\mathcal{H}_L(\mathcal{U})$
WaterlooClarke	UWCTSRun1	0.2350	0.3520	0.6612	**0.1762**
WaterlooClarke	UWCTSRun3	0.2252	0.3421	**0.6643**	0.1718
WaterlooClarke	UWCTSRun2	**0.2872**	0.2584	0.6551	0.1710
cunlp	**3LtoSfltr5**	0.1371	0.4870	0.6392	0.1282
cunlp	**1LtoSfltr20**	0.1203	0.5372	0.6287	0.1100
IRIT	FS1A	0.0849	0.4959	0.6051	0.0719
cunlp	**4APSAL**	0.1011	0.4584	0.5108	0.0674
average		0.0666	0.4342	0.4697	0.0499
IRIT	FS2A	0.0518	0.5899	0.6285	0.0476
BJUT	DMSL1NMF2	0.0445	0.6123	0.4539	0.0354
BJUT	DMSL1AP1	0.0413	**0.6155**	0.4701	0.0338
l3sattrec15	l3sattrec15run1	0.0408	0.3612	0.3743	0.0268
l3sattrec15	l3sattrec15run3	0.0400	0.3669	0.3712	0.0262
IRIT	FS1B	0.0422	0.2939	0.3913	0.0259
IRIT	FS2B	0.0306	0.3391	0.4491	0.0239
USI	InL2DecrQE1ID1	0.0182	0.5713	0.5806	0.0196
USI	InL2DecrQE2ID2	0.0169	0.5758	0.5836	0.0184
udel_fang	WikiOnlyFS2	0.0206	0.5819	0.4600	0.0176
udel_fang	ProfOnlyFS3	0.0258	0.5294	0.4122	0.0174
USI	InL2StabQE2ID3	0.0171	0.6133	0.5238	0.0170
udel_fang	WikiProfMixFS1	0.0189	0.5965	0.4660	0.0166
l3sattrec15	l3sattrec15run2	0.0283	0.2276	0.2560	0.0164
USI	InL2IncrQE2ID4	0.0179	0.5837	0.2888	0.0108

Table 4.6: Official TREC 2015 Temporal Summarization Task 2 results using manual update/nugget matches.

The SAP run, 4APSAL, was also above the track average for all evaluation measures. It is, however, dominated by both L2S runs on all measures. This suggests that our L2S is an overall improvement over the SAP model in both precision, recall, and latency.

4.4.7 Automatic Experiments

We evaluate our method on the publicly available TREC Temporal Summarization Track data using the data from the 2013, 2014, and 2015 years of the shared task. This collection contains 44 unique query events. To evaluate our model, we randomly select five events to use as a development set and then perform a leave-one-out style evaluation on the remaining 39 events.

As we did in the official Temporal Summarization submission, we downsample each training stream to a length of 100 sentences. The downsampling is done uniformly over the entire stream. This is repeated 10 times for each training event to create a total of 380 training streams. In the event that a downsample contains no nuggets (either human or automatically labeled) we resample until at least one exists in the sample. We select the best model iteration for each training fold based on the automatically computed $\mathcal{H}(\mathcal{U})$ on the development set.

4.4.7.1 Baselines and Model Variants

We refer to our "learning to search" model in the results as L2S. We compare our proposed model against several baselines and extensions.

Cosine Similarity Threshold WaterlooClarke, the top performing team at TREC 2015 used a heuristic method that only examined article first sentences, selecting those that were below a cosine similarity threshold to any of the previously selected updates. We implemented a variant of that approach using the latent-vector representation used throughout this work. The development set was used to set the threshold. We refer to this model as COS.

SAP We also compare to the SAP model described in §4.3. In this evaluation.

Learning2Search+Cosine Similarity Threshold In this model, which we refer to as L2S-COS, we run L2S as before, but filter the resulting updates using the same cosine similarity thresh-

	unpenalized			latency-penalized			
	$n\mathbb{E}[G](\mathcal{U})$	$C(\mathcal{U})$	$\mathcal{H}(\mathcal{U})$	$n\mathbb{E}[G_L](\mathcal{U})$	$C_L(\mathcal{U})$	$\mathcal{H}_L(\mathcal{U})$	$\lvert\mathcal{U}\rvert$
SAP	$\mathbf{0.119}^{c}$	0.09	0.094	0.105	0.088	0.088	8.333
COS	0.075	0.176^{s}	0.099	0.095	0.236^{s}	0.128^{s}	$145.615^{s,f}$
L2S	0.097	$\mathbf{0.207}^{s,f}$	0.112	0.136^{c}	$\mathbf{0.306}^{s,c,f}$	0.162^{s}	$89.872^{s,f}$
L2S-COS	$0.115^{c,l}$	0.189^{s}	$\mathbf{0.127}^{s,c,l}$	$\mathbf{0.162}^{s,c,l}$	0.276^{s}	$\mathbf{0.184}^{s,c,l}$	$29.231^{s,c}$

Figure 4.7: Average system performance and average number of updates per event. Superscripts indicate significant improvements ($p < 0.05$) between the run and competing algorithms using the paired randomization test with the Bonferroni correction for multiple comparisons (s: APSAL, c: COS, l: LS, f: LS-COS).

old method as in COS. The threshold was also tuned on the development set.

4.4.8 Results

Results for system runs are shown in Figure 4.7. On average, L2S and L2S-COS achieve higher $\mathcal{H}(\mathcal{U})$ scores than the baseline systems in both latency penalized and unpenalized evaluations. For L2S-COS, the difference in mean $\mathcal{H}(\mathcal{U})$ score was significant compared to all other systems (for both latency settings). Intuitively, L2S has higher comprehensiveness than L2S-COS; adding the the cosine similairt filter to L2S reduces the comprehensiveness, but increases the average gain of the updates by a larger amount, yielding an improved harmonic mean of the two metrics ($\mathcal{H}(\mathcal{U})$).

SAP achieved the overall highest expected gain, partially because it was the tersest system we evaluated (at 8 updates per query on average). However, only COS was statistically significantly worse than it on this measure. We also see that SAP suffers from the latency-weighted evaluation, receiving latency penalties for retrieving updates after that information had been added to Wikipedia. By comparison, all other systems are actually rewarded in the latency-weighted evaluation, as they consistently retrieve information before it is published in Wikipedia. While SAP had previously beaten Wikiepdia in previous evaluations, the added events for the 2015 Temporal Summarization had less media coverage, suggesting the clustering based approach is less suited for lower-volume news streams.

	latency-penalized		
	$n\mathbb{E}[G_L](\mathcal{U})$	$C_L(\mathcal{U})$	$\mathcal{H}_L(\mathcal{U})$
COS	0.095	**0.236**	0.128
L2S-FS	0.164	0.220	0.157
L2S-COS-FS	**0.207**	0.18	**0.163**

Table 4.7: Average system performance. L2S-FS and L2S-COS-FS runs are trained and evaluated on first sentences only (like the COS system). Ranking is consistent with unpenalized results.

In comprehensiveness, L2S recalls on average a fifth of the nuggets for each event. This is even more impressive when compared to the average number of updates produced by each system; while COS achieves similar comprehensiveness, it takes on average about 1.6 times more updates than L2S and almost 5 times more updates than L2S-COS. The output size of COS stretches the limit of the term "summary," which is typically far shorter than 145 sentences in length. This is especially important if the intended application is negatively affected by verbosity (e.g. crisis monitoring).

4.4.9 Discussion

Since COS only considers the first sentence of each document, it may miss relevant sentences below the article's lead. In order to confirm the importance of modeling the oracle, we also trained and evaluated the L2S based approaches on first sentence only streams. Table 4.7 shows the latency penalized results of the first sentence only runs. The L2S approaches still dominate COS and receive larger positive effects from the latency penalty despite also being restricted to the first sentence. Clearly having a model (beyond similarity) of what to select is helpful. Ultimately we do much better when we can look at the whole document.

We also performed an error analysis to further understand how each system operates. Figure 4.8 shows the errors made by each system on the test streams. Errors were broken down into four categories. *Miss lead* and *miss body* errors occur when a system skips a sentence containing a novel nugget in the lead or article body respectively. An *empty* error indicates an update was selected that contained no nugget. *Duplicate* errors occur when an update contains nuggets but

	Miss (Lead)	Miss (Body)	Empty	Duplicates	Total
SAP	29.6%	68.7%	1.6%	0.1%	15,986
Cos	17.8%	39.4%	41.1%	1.7%	12,873
L2S-FS	25.4%	71.7%	2.0%	0.9%	13,088
L2S-Cos-FS	27.9%	70.8%	1.0%	0.2%	15,756
L2S	19.6%	55.3%	19.9%	5.1%	13,380
L2S-Cos	24.6%	66.7%	7.5%	1.2%	11,613

Figure 4.8: Percent of errors made and total errors on test set.

none are novel.

Overall, errors of the miss type are most common and suggest future development effort should focus on summary content identification. About a fifth to a third of all system error comes from missing content in the lead sentence alone.

After misses, empty errors (false positives) are the next largest source of error. Cos was especially prone to empty errors (41% of its total errors). L2S is also vulnerable to empties (19.9%) but after applying the similarity filter and restriting to first sentences, these errors can be reduced dramatically (to 1%).

Surprisingly, duplicate errors are a minor issue in our evaluation. This is not to suggest we should ignore this component, however, as efforts to increase recall (reduce miss errors) are likely to require more robust redundancy detection.

4.5 Conclusion

In this chapter we introduced two stream summarization algorithms, where we focused on incorporating salience predictions into the update selection stage. In the first model, SAP, we were able to incorporate these predictions into the affinity propagation clustering algorithm, thereby balancing the twin objectives of selecting representative updates while ensuring the updates contained essential information that was relevant to the query event. In the second method, L2S, we were able to train a stream summarization policy which predicts the salience of potential updates using a feature representation of the current update summary and stream state. This method improves over SAP in that it can make predictions using dynamically updated features based on previous behavior and can make decisions on updates immediately without needing to collect a sizeable cache of sentences for clustering.

Future work could focus on a number of improvements. One of the most important ones would be to improve the understanding of novelty. Currently, we do not have any explicit handling of information that is refined or updated over time. For example, the number of reported casualities in a disaster typically changes over time as more cases are reported and nubmers are refined. Under our current models these statements might all look very similar since they have a fairly boilerplate text, e.g. for we have the following nuggets for a train crash event presented in chronological order,

- 55 dead
- 50 confirmed dead
- 51 confirmed dead.

As this information is textually similar, after a system selects one of these updates, it may not extract others.

Also as issues of trust and veracity in reporting in online sources have become more important, it might also be necessary to model the reliability of an information source. In this case some numbers are reported and some are also confirmed. Our model currently has no distinction between official and unofficial sourcing, which would be necessary in any real implemenation of these models.

We also clearly have room to improve salience predictions. The expected gain of our systems hovered around 0.1, meaning it would take on average ten updates to produce a novel piece of information. As our error analysis showed, misses were the most prevalent error. All implemented systems have the highest error rates when trying to find information in the body of an article. This echoes the findings of our previous chapter, where position bias is predominant feature being exploited. More efforts on modeling the salience of sub-events and knock-on effects of a query event might be beneficial here.

Chapter 5: Faithful and Controllable Generation

Up to this point, we have focused on content selection in a text-to-text generation system while relying on a trivial text generation algorithm, copying or extracting text units verbatim from the input, to perform the actual generation task. In this chapter, we move to modeling the actual language generation process after the content selection stages (i.e. Chapters 3 and 4) have been performed. We focus in particular on the sequence-to-sequence model as a means of mapping a representation of selected content to an appropriate natural language utterance. Sequence-to-sequence models are a family of deep learning models with bi-partite structure, possessing an encoder network which represents the input and a decoder which generates output from encoder's state (Sutskever et al., 2014). We tackle two issues, faithfulness and control, which are necessary prerequisites for any practically useful sequence-to-sequence-based model of natural language generation. These concepts, which we define in more detail later in this chapter, can be broadly construed as ensuring the decoder (i.e., the language model) in a sequence-to-sequence model is constrained to generate utterances that respect the semantics of the input (i.e. ensuring model faithfulness) while following a proscribed discourse structure for the selected content (i.e. ensuring model control).

When data is plentiful, the sequence-to-sequence paradigm has proven to be incredibly adaptable to a variety of problem domains, and in the research literature it has become the standard method for a host of language generation tasks (Xu et al., 2015; Dušek and Jurčíček, 2016; Vaswani et al., 2017; Fan et al., 2018). Recent evaluations of end-to-end trained sequence-to-sequence models for dialogue generation have shown that they are capable of learning very natural text realizations of formal meaning representation. In many cases, they beat rule and template based systems on human and automatic measures of quality and naturalness (Dušek et al., 2020).

However, this powerful generation capability comes with a cost; deep learning-based language models are notoriously difficult to control, often producing quite fluent but semantically misleading

$$\begin{bmatrix}\textsc{Inform}\\ \text{name=Aromi}\\ \text{eat\_type=coffee shop}\\ \text{customer\_rating=5 out of 5}\\ \text{food=English}\\ \text{area=city centre}\\ \text{family\_friendly=yes}\end{bmatrix}$$

(a) *Inform* dialog act.

- Aromi coffee shop serves English food in a family-friendly atmosphere near the city center and has a customer rating of 5 out of 5.
- The Aromi coffee shop is family-friendly and serves English food. It has a customer rating of 5 out of 5 and is located near the center of the city.

(b) Natural language utterances.

$$\begin{bmatrix}\textsc{Give Opinion}\\ \text{name=Little Nightmares}\\ \text{rating=good}\\ \text{genres=[}\\ \quad\text{adventure,}\\ \quad\text{platformer,}\\ \quad\text{puzzle}\\ \text{]}\\ \text{player\_perspective=side view}\end{bmatrix}$$

(c) *Give Opinion* dialog act.

- Adventure games that combine platforming and puzzles can be frustrating to play, but the side view perspective is perfect for them. That's why I enjoyed playing Little Nightmares.
- Little Nightmares is a pretty cool game that has kept me entertained. It's an adventure side-scrolling platformer with some puzzle elements to give me a bit of a challenge.

(d) Natural language utterances.

$$\begin{bmatrix}\textsc{Compare}\\ \text{name}_1\text{=Erebus 92}\\ \text{resolution}_1\text{=720p}\\ \text{family}_1\text{=W2}\\ \\ \text{name}_2\text{=Helios 96}\\ \text{resolution}_2\text{=1080p}\\ \text{family}_2\text{=L5}\end{bmatrix}$$

(e) *Compare* dialog act.

- Tthe Helios 96 tv has a 1080p resolution in the L5 family while the Erebus 92 has a 720p resolution in the W2 family.
- Compared to Erebus 92 which has 720p resolution and is in the W2 product family, Helios 96 has 1080p resolution and is in the L5 product family. Which one do you prefer?

(f) Natural language utterances.

$$\begin{bmatrix}\textsc{Confirm}\\ \text{type=laptop}\\ \text{drive\_range=medium}\end{bmatrix}$$

(g) *Confirm* dialog act.

- Just to verify. The laptop needs to have a medium drive range, correct?
- Let me confirm, a laptop in the medium drive range right?

(h) Natural language utterances.

Figure 5.1: Example meaning representations (left) and their reference utterances (right) from the restaurant, video game, tv, and laptop domains.

outputs. For instance, Dušek et al. (2020) note that in the E2E Generation Challenge shared-task, sequence-to-sequence models were both some of the best and worst performers. One submission, a vanilla sequence-to-sequence model, scored first in human evaluations of naturalness but last in quality (which they define as semantic completeness and grammaticality). In order for such models to truly be useful, they must be capable of correctly generating utterances for novel meaning representations at test time. In practice, even with delexicalization (Dušek and Jurčíček, 2016; Juraska et al., 2018), copy and coverage mechanisms (Elder et al., 2018), and over-generation plus reranking (Dušek and Jurčíček, 2016; Juraska et al., 2018), deep learning-based language generation models still produce errors (Dušek et al., 2020).

We study sequence-to-sequence model faithfulness on the *task-oriented dialog generation* problem (Mairesse et al., 2010; Wen et al., 2015; Dušek et al., 2018), where a natural language generation model must map a meaning representation (i.e., a dialogue act with an associated set of attribute-value pairs[1]) to an appropriate natural language utterance (see Figure 5.1 for examples). In the context of our broader work on text-to-text generation, we think of the meaning representation input as an idealized representation of the content selection stage in a text-to-text generation model. Studying faithfulness and control in the closed-world domains of task-oriented dialog generation allows us to make meaningful progress while minimizing unnecessary complexity.

For instance, natural language summaries often contain information not explicitly represented in the input. The source of this content is either from common sense knowledge, generic or domain specific knowledge, or new facts deduced from any combination of the input and prior knowledge (Wiseman et al., 2017; Wang, 2019). Evaluating the faithfulness of a neural language generation model in this context is extremely difficult because it is not clear if a generated utterance is due to the decoder language model or the encoder's representation of the input.

Instead, the task-oriented dialogue generation problems we study are developed to be closed-world, narrow domain settings, where the totality of the information needed to generate an utterance is represented by the meaning representation. Additionally, the semantics of the meaning

[1] In the literature and in industry, dialogue acts are sometimes called "intents," and attribute-value pairs as "slots" and "slot-fillers" or "entities."

representation are explicit; there is no information that needs to be realized by the language generation component that requires additional inferences from the input.

We call a natural language generation model that generates utterances that are semantically correct with respect to the input meaning representation, a faithful generation model. We posit that sequence-to-sequence models do not learn representations of the input meaning representation that correspond to basic features of the nature of utterance data, chiefly that phrases denote fragments of meaning representation which can be recombined with other fragments to systematically create new meanings/utterances. Instead, the learned representations are highly idiosyncratic, and often reflect spurious correlations and artifacts of the dataset that do not generalize well outside the training data. This issue is symptomatic of neural models' lack of systematicity, which in turn leads to unfaithful language generation models (Lake and Baroni, 2018).

To overcome these issues, we propose a novel data augmentation scheme, called *noise-injection sampling and self-training*, to create synthetic meaning representation/utterance pairs which break spurious correlations in the training dataset. Our method makes use of a vanilla sequence-to-sequence natural language generation model, i.e. the kind described by Dušek et al. (2020) that produces natural but semantically incorrect utterances, and a meaning representation parser, both of which can be trained on the same parallel data. We then use a noise-injection sampling method (Cho, 2016) that allows us to generate semantically diverse yet syntactically well formed utterances from the natural language generation model. We obtain corrected meaning representations for these sampled utterances using the meaning representation parser. Using this procedure we can generate a large collection of synthetic data points which show a reduction in spurious correlations between the size of a meaning representation and its content or between different pairs of attribute-values. Training a new sequence-to-sequence model on the union of the original training and novel synthetic data yields a more faithful generation model with substantially reduced semantic errors.

While a faithful sequence-to-sequence model produces semantically correct output, in general it is free to let the surface realization order of the attribute-values be determined by its language model. We show that we can actually control the realization order by properly "linearizing" the

$$\begin{bmatrix} \textsc{Inform} \\ \text{name=Aromi} \\ \text{area=city centre} \\ \text{eat\_type=coffee shop} \end{bmatrix}$$

(a) *Inform* dialog act.

Plan A: name → eat_type → area
Realization: *Aromi is a coffee shop in the city centre.*

Plan B: eat_type → name → area
Realization: *There is a coffee shop called Aromi in the city centre.*

Plan C: eat_type → area → name
Realization: *For coffee in the centre of the city, try Aromi.*

(b) Three different utterance plans with example realizations for the *Inform* dialogue act (left).

Figure 5.2: Examples of controllable generation. (a) A meaning representation of an *Inform* dialogue act. (b) Three hypothetical utterance plans and their realizations for the example dialogue act.

meaning representation, that is, converting the meaning representation into a linear sequence of discrete tokens, before feeding it into the encoder. Our proposed *alignment training* linearization strategy for converting a meaning representation to an encoder input sequence yields a highly controllable generation model, effectively moving implicit utterance planning from the decoder to the encoder. We consider controllable generation models to be a subset of faithful generation models that can follow an externally provided discourse ordering plan. See Figure 5.2 for examples of such plans in the context of task-oriented dialogue generation. We find that alignment training endows both recurrent and transformer-based sequence-to-sequence architectures with the controllable generation property as well as when fine tuning a large, pretrained sequence-to-sequence model.

While most contemporary research practice prefers end-to-end solutions that leave planning implicit, we argue that such fine grained control in a sequence-to-sequence model is highly desirable. Not only would it enable drawing deeper connections between sequence-to-sequence models and the extensive literature on sentence or utterance level planning for language generation (Reiter and Dale, 2000; Walker et al., 2001; Stone et al., 2003), it would also allow for neural implementations of various psycho-linguistic theories of discourse (e.g., Centering Theory (Grosz et al., 1995), or Accessibility Theory (Ariel, 2001)). This could, in turn, encourage the validation and/or refinement of additional psychologically plausible models of language production.

Ensuring robustness of the control behavior is also necessary to reliably incorporate neural language generation models into larger language generation pipelines (Moryossef et al., 2019a,b; Castro Ferreira et al., 2019). However, as previously mentioned, neural models do not learn systematic representations of the input, which can lead to errors in faithfulness or plan following when generating from ordering plans not well represented in the training data. To mitigate this, we propose a phrase-based data augmentation scheme to collect additional examples that give explicit supervision of how constituent phrases compose, and how that composition can systematically change the meaning (e.g. prepending "not" to a phrase systematically negates its meaning). We show under extensive stress testing with randomly generated plans that this data-augmentation improves the robustness of control.

In what follows, we introduce the meaning representations used for task-oriented dialogue generation in more detail (§5.1) and provide some background on sequence-to-sequence modeling for meaning representation-to-text generation (§5.2). We then turn to our main contributions, our noise-injection sampling and self-training data-augmentation method for faithful generation (§5.3), and alignment training linearization for controllable generation (§5.4), before concluding.

5.1 Meaning Representations for Task-Oriented Dialogue Generation

5.1.1 Meaning Representation Structure

In this chapter, we use several domain specific meaning representations to formally represent the input to the surface realization model. While specifics of the meaning representation can vary from domain to domain, the overall structure of the meaning representation is fairly straightforward, borrowing from a common format used frequently in the natural language generation literature (Mairesse et al., 2010; Gašić et al., 2014; Wen et al., 2015; Novikova et al., 2017; Juraska et al., 2019). Each meaning representation has a dialogue act, which expresses the communicative goal or intent, and zero or more attribute-value pairs which further define the semantics of the desired utterance.

See Figure 5.1a for an example meaning representation from the restaurant domain. The dialogue act, in this case to inform a user, is the first item and is written in SMALLCAPS style. The attributes are “name,” “eat_type,” “customer_rating,” “food,” “area,” and “family_friendly.” Their associated values are “Aromi,” “coffee shop,” “5 out of 5,” “English,” “city centre,” and “yes” respectively. In this case the attributes are referring to the restaurant about which a hypothetical dialogue agent is trying to inform a user.

In our setting, dialogue acts are predominantly declarative (e.g., Figure 5.1a or Figure 5.1c), but also include interrogatives (e.g., Figure 5.1g), and some that may be a mix of both (e.g., Figure 5.1e where the second reference ends in a question about user preference). Additionally, we also have semantically vacuous “chit-chat” dialogue acts like GREETING and GOODBYE which are expected to begin and end, respectively, a series of exchanges with the dialogue agent.

The kinds of values that can fill an attribute are typically categorical variables. For example, in the restaurant domain, the attribute “food” may take values from a closed list of food types such as the set

$$\{\text{“Chinese”}, \text{“English”}, \text{“French”}, \text{“Fast food”}, \text{“Indian”}, \text{“Italian”}, \text{“Japanese”}\}.$$

Other value types include list-valued attributes, numerical values, or free text (see Figure 5.3 for examples of each). For list-valued attributes, the value is a list of items drawn from a closed set. For example, in the video game domain, a video game can belong to several genres simultaneously. For our purposes, we treat each value in the list as a distinct attribute-value pair. So in the case of Figure 5.3a, we treat it is if it had the following meaning representation,

INFORM
name=Portal 2
esrb=E 10+ (for Everyone 10 and Older)
genres=[platformer, puzzle, shooter]
player_perspective=[first person]
has_multiplayer=yes
	Portal 2 was rated E 10+ (for Everyone 10 and Older). It is a puzzle platformer FPS with multiplayer.

(a) An example of list-valued attributes (genres and player_perspective) from the video game domain. Note that the acronym FPS means "first person shooter" which realizes both the player_perspective attribute-valueand a genre attribute-value.

REQUEST
specifier="dull"
has_multiplayer=yes
	What's the most dull multiplayer game you've ever played?

(b) An example of a free-text valued attribute (specifier) from the video game domain. The specifier value can be any adjective.

INFORM COUNT
count=58
type=laptop
is_for_business_computing=true
weight_range=don't care
drive_range=don't care
	There are 58 laptops used for business computing if you do not care what weight range or drive range they have.

(c) An example of a numeric-valued attribute (count) from the laptop domain.

Figure 5.3: Examples of various attribute-value types paired with an example realization.

$$\left[\!\!\left[\begin{array}{l}\textsc{Inform}\\ \text{name=Portal 2}\\ \text{esrb=E 10+ (for Everyone 10 and Older)}\\ \text{genres=platformer}\\ \text{genres=puzzle}\\ \text{genres=shooter}\\ \text{player\_perspective=first person}\\ \text{has\_multiplayer=yes}\end{array}\right]\!\!\right]$$

Additionally, not all attributes need to be specified. In which case, the utterance should not mention them.

The term "meaning representation" is somewhat of a misnomer as the representations might better be characterized as a pragmatic construct (i.e. a representation of the dialogue agent's intentional state). The attribute-values, on the other hand, are a semantic construct, representing the semantic value or propositional content of the sentences in the utterance. In other words, from the perspective of formal semantics,

- *The Aromi is a coffee shop in the city centre.*
- *Just to confirm, the coffee shop in the city centre is called Aromi?*
- *What about Aromi, the coffee shop in the city centre?*

all share the same semantic value. The "meaning" of the above utterances as a statement of first-order logic might look something like,

$$\exists x : \text{isCoffeeShop}(x) \wedge \text{namedAromi}(x) \wedge \text{inCityCentre}(x).$$

We could represent this statement in our present setting as a "meaning representation without a dialogue act," i.e.,

$$\left[\!\!\left[\begin{array}{l}\text{—}\\ \text{name=Aromi}\\ \text{eat\_type=coffee shop}\\ \text{area=city centre}\end{array}\right]\!\!\right]$$

which, when combined with one of the dialogue acts INFORM, CONFIRM, or RECOMMEND, yields the pragmatic sense of the respective utterances above.

5.1.2 Relating Between Meaning Representations and Utterances

Let $\mu \in \mathcal{M}$ be a meaning representation, and let $\mathbf{y} = [y_1, \ldots, y_n] \in \mathcal{Y}$ be an utterance, i.e. sequence of n tokens from a vocabulary $\mathcal{V}_\mathcal{Y}$ and $\mathcal{Y} = \mathcal{V}_\mathcal{Y}^*$. We say that an utterance $\mathbf{y}$ *denotes* a meaning representation μ, which we write $[\![\mathbf{y}]\!] = \mu$ if the propositional content of the utterance and the meaning representation are the same, i.e. the attribute-values implied by $\mathbf{y}$ and explicitly listed by μ are the same. We can make similar statements about a sub-sequence of an utterance. Let $\mathbf{y}_{i:i+j} = \left[y_i, y_{i+1}, \ldots, y_{i+j}\right]$ be a sub-sequence of $j + 1$ tokens starting at token i. We then have $\left[\!\left[\mathbf{y}_{i:i+j}\right]\!\right] = \mu'$ for some $\mu' \in \mathcal{M} \cup \emptyset$. When an utterance or sub-sequence $\mathbf{y}$ contains no propositional content or is otherwise not a meaningful statement, we have $[\![\mathbf{y}]\!] = \emptyset$.

As an example, consider the following meaning representation,

$$\mu = \left[\!\!\left[\begin{array}{l}\text{INFORM}\\ \text{name=The Vaults}\\ \text{eat\_type=pub}\\ \text{near=Café Adriatic}\\ \text{family\_friendly=no}\end{array}\right]\!\!\right]$$

and the utterance,

$$\mathbf{y} = [\textit{The}, \textit{Vaults}, \textit{pub}, \textit{is}, \textit{near}, \textit{Café}, \textit{Adriatic}, ., \textit{It}, \textit{is}, \textit{not}, \textit{a}, \textit{good}, \textit{place}, \textit{for}, \textit{families}, .] \,.$$

Clearly, $[\![\mathbf{y}]\!] = \mu$. But we can also look at the meanings of individual phrases,

$$\begin{aligned}
\llbracket \mathbf{y}_{1:2} \rrbracket = \llbracket [\mathit{The}, \mathit{Vaults}] \rrbracket &= \left\llbracket \begin{array}{l} \text{—} \\ \text{name=The Vaults} \end{array} \right\rrbracket \\
\llbracket \mathbf{y}_{3:3} \rrbracket = \llbracket [\mathit{pub}] \rrbracket &= \left\llbracket \begin{array}{l} \text{—} \\ \text{eat\_type=pub} \end{array} \right\rrbracket \\
\llbracket \mathbf{y}_{5:7} \rrbracket = \llbracket [\mathit{near}, \mathit{Café}, \mathit{Adriatic}] \rrbracket &= \left\llbracket \begin{array}{l} \text{—} \\ \text{near=Café Adriatic} \end{array} \right\rrbracket \\
\llbracket \mathbf{y}_{11:16} \rrbracket = \llbracket [\mathit{not}, a, \mathit{good}, \mathit{place}, \mathit{for}, \mathit{families}] \rrbracket &= \left\llbracket \begin{array}{l} \text{—} \\ \text{family\_friendly=no} \end{array} \right\rrbracket .
\end{aligned}$$

Note that it is not the case that $\llbracket \mathbf{y} \rrbracket = \mu \Rightarrow \llbracket \mathbf{y}_{i:i+j} \rrbracket \subseteq \mu$. Consider in the example above $\mathbf{y}_{11:16}$ its sub-sequence $\mathbf{y}_{12:16} = [a, \mathit{good}, \mathit{place}, \mathit{for}, \mathit{families}]$ which have the following denotations,

$$\llbracket \mathbf{y}_{11:16} \rrbracket = \left\llbracket \begin{array}{l} \text{—} \\ \text{family\_friendly=yes} \end{array} \right\rrbracket \neq \llbracket \mathbf{y}_{12:16} \rrbracket = \left\llbracket \begin{array}{l} \text{—} \\ \text{family\_friendly=no} \end{array} \right\rrbracket$$

It is also important to note that the attribute-values are unordered and do not necessarily reflect the realization order of the utterance.

In the datasets used for this chapter, μ are provided with one or more reference utterances, $\mathbf{y}^{(1)}, \ldots \mathbf{y}^{(k)}$ and that for each reference $\mathbf{y}^{(i)}$, we have that each attribute-value in μ can be mapped to an utterance sub-sequence that denotes it. Occasionally this is not the case in the available training data. For example, some attribute-values may have several possible groundings (see Figure 5.4a) or be realized using inferential knowledge not explicitly represented in the meaning representation (see Figure 5.4b).

While such examples may exist in the training data, we consider model generation of such phenomena to constitute a failure to faithfully generate an utterance. In the overwhelming majority of cases, each attribute-value is explicitly and uniquely grounded in the target utterances, this makes surface realization from meaning representations a useful task to study faithful generation. The baseline task of correctly generating all attribute-values appropriately for the dialogue act is hard enough, and it in this setting we do not have to worry about ungrounded information or information that is not explicitly represented in the meaning representations but is deducible from the meaning representation (Wiseman et al., 2017).

INFORM
name=Wildwood
food=Italian
eat_type=pub
price_range=£20-25
customer_rating=high

Wildwood is an Italian pub with a price range of £20-25. The pub has a very high customer rating.

(a) The attribute-value eat_type=pub occurs in multiple locations of the utterance.

INFORM
name=The Waterman
food=Japanese
price_range=high
area=riverside

Near the river there is an expensive sushi place called The Waterman.

(b) The utterance claims the restaurant serves sushi even though this is not stated in the meaning representation. Not all Japanese restaurants serve sushi so this inference is not justified.

Figure 5.4: Example training set errors.

5.2 Modeling Meaning Representation-to-Text Generation with Sequence-to-Sequence Architectures

5.2.1 Sequence-to-Sequence Modeling

We approach the problem of mapping a meaning representation to a natural language utterance with a variety of popular sequence-to-sequence architectures. A sequence-to-sequence model is a neural network with parameters θ that implements a probabilistic mapping, $p(\cdot|\cdot;\theta) : \mathcal{X} \times \mathcal{Y} \rightarrow (0, 1)$, from input sequences

$$\mathbf{x} = [x_1, \ldots, x_m] \in \mathcal{X}$$

to output sequences

$$\mathbf{y} = [y_1, \ldots, y_n] \in \mathcal{Y}.$$

Tokens from the input sequence are drawn from a finite vocabulary $\mathcal{V}_\mathcal{X}$ and input sequences its Kleene closure $\mathcal{V}_\mathcal{X}^* = \mathcal{X}$. Analogously, tokens from the output sequence are drawn from a distinct, finite vocabulary $\mathcal{V}_{\mathcal{Y}}$ and output sequences its Kleene closure $\mathcal{V}_{\mathcal{Y}}^* = \mathcal{Y}$. For clarity we occasionally omit θ in subsequent equations.

Typically, p is implemented as a bipartite network consisting of distinct encoder and decoder networks Enc and Dec respectively. The encoder Enc : $\mathcal{X} \rightarrow \mathbb{R}^{*\times D_{\mathcal{E}}}$ is a mapping of an input sequence $\mathbf{x}$ of m tokens to m corresponding vectors $\mathbf{h}_1, \ldots \mathbf{h}_m \in \mathbb{R}^{D_{\mathcal{E}}}$ and

$$p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}|\operatorname{Enc}(\mathbf{x})) = p(\mathbf{y}|\mathbf{h}_1, \ldots, \mathbf{h}_m).$$

The decoder Dec : $\mathcal{V}_{\mathcal{Y}}^+ \times \mathbb{R}^{*\times D_{\mathcal{E}}} \rightarrow (0, 1)$ then is a mapping of previously generated tokens $\mathbf{y}_{1:i-1} = [y_1, \ldots, y_{i-1}]$ and encoder states $\mathbf{h}_1, \ldots, \mathbf{h}_m$ to a probability distribution over the output vocabulary $\mathcal{V}_{\mathcal{Y}}$, where

$$p(y_i|\mathbf{y}_{1:i-1}, \mathbf{x}) = \operatorname{Dec}\left(\mathbf{y}_{1:i}, \operatorname{Enc}(\mathbf{x})\right) \quad \text{and} \quad \sum_{y \in \mathcal{V}_{\mathcal{Y}}} p\left(y|\mathbf{y}_{1:i-1}, \operatorname{Enc}(\mathbf{x})\right) = 1.$$

Hence, $p(\cdot|\mathbf{x})$ is a conditional language model over utterance tokens that factorizes in a left-to-right fashion, i.e.,

$$p\left(\mathbf{y}|\mathbf{x}\right) = \prod_{i=1}^{n} p\left(y_i|\mathbf{y}_{1:i-1}, \mathbf{x}\right).$$

Notice that the “inputs” and “outputs” to the sequence-to-sequence model are sequences of tokens, $\mathbf{x}$ and $\mathbf{y}$ respectively. In order to use a sequence-to-sequence model for natural language generation from a meaning representation we need only map our desired inputs and outputs to sequences of discrete tokens. In English, the desired output is relatively straightforward to represent as a sequence as an English language utterance can naturally be represented as a sequence of word tokens. In practice, we also indicate full sentence stops with a special token «■»[2] and prepend and append distinguished tokens *«s»* and *«e»*, respectively, to indicate the start and end of the utterance,

[2] We add an explicit sentence stop symbol because occasionally utterances have non-sentence final periods and we want there to be no ambiguity about sentence stops produced by the model.

as well as lower-case all tokens. As an example, the utterance

The Vaults pub is near Café Adriatic. It is not a good place for families.

would be represented as

$$\mathbf{y} = [\textit{«s»}, \textit{the}, \textit{vaults}, \textit{pub}, \textit{is}, \textit{near}, \textit{café}, \textit{adriatic}, \textit{.}, \textit{«}\blacksquare\textit{»}, \textit{it}, \textit{is}, \textit{not}, \textit{a}, \textit{good}, \textit{place}, \textit{for}, \textit{families}, \textit{.}, \textit{«e»}]\ .$$

The meaning representation is not itself a sequence, however, so we cannot apply it to a sequence-to-sequence model directly. Instead it must first be "linearized," or mapped to a linear sequence of tokens. We refer to a function $\pi : \mathcal{M} \rightarrow \mathcal{X}$, as a linearization strategy. We experiment with several linearization strategies in this chapter, however, all of them operate over the same domain, $\mathcal{V}_{\mathcal{X}}^{+}$, where $\mathcal{V}_{\mathcal{X}}$ consists of distinct tokens for each dialogue act and attribute-value pairs. As an example consider the following meaning representation,

$$\mu = \left[\!\!\left[\begin{array}{l} \textsc{Inform} \\ \text{name=Aromi} \\ \text{area=city centre} \\ \text{eat\_type=coffee shop} \end{array} \right]\!\!\right]$$

and some possible linearizations,

$$\pi_1(\mu) = \mathbf{x} = [\textit{inform}, \textit{name=Aromi}, \textit{eat\_type=coffee shop}, \textit{area=city centre}]$$

$$\pi_2(\mu) = \mathbf{x} = [\textit{inform}, \textit{eat\_type=coffee shop}, \textit{name=Aromi}, \textit{area=city centre}]$$

$$\pi_3(\mu) = \mathbf{x} = [\textit{inform}, \textit{eat\_type=coffee shop}, \textit{area=city centre}, \textit{name=Aromi}]\ .$$

In practice, regardless of the choice of linearization strategy, we prepend a start token, *«s»*, and append and a stop token, *«e»*, to all input token sequences, e.g.

$$\mathbf{x} = [\textit{«s»}, \textit{inform}, \textit{name=Aromi}, \textit{eat\_type=coffee shop}, \textit{area=city centre}, \textit{«e»}]\ .$$

The encoder and decoder networks of the sequence-to-sequence model can be implemented with a variety of architectures. We use two such architectures, the gated recurrent unit (GRU) (Cho et al., 2014) and the transformer (Vaswani et al., 2017). Since we use standard variants, we defer explicit model definitions to Appendix A and Appendix B for GRU- and transformer-based sequence-to-sequence architectures respectively. While the GRU must be given an explicit linearization to process any input, the transformer variant does not have to be sensitive to linearization order. However, when the transformer uses position embeddings, which is the standard configuration, it is sensitive to linearization order. We always use position embeddings in this work as Vaswani et al. (2017) found the model did not work as well without them.

5.2.2 Learning

Given a dataset of meaning representation/utterance pairs, $\mathcal{D} = \left\{\left(\mu^{(1)}, \mathbf{y}^{(1)}\right), \ldots, \left(\mu^{(N)}, \mathbf{y}^{(N)}\right)\right\}$, and a linearization strategy, π, the parameters, θ, of p can be learned by approximately minimizing the negative log-likelihood of the data,

$$\hat{\theta} \approx \operatorname*{arg\,min}_{\theta \in \Theta} -\frac{1}{N} \sum_{(\mu, \mathbf{y}) \in \mathcal{D}} \log p\left(\mathbf{y} | \pi\left(\mu\right); \theta\right).$$

In practice, this is done with some variant of mini-batch stochastic gradient descent, e.g., Adam (Kingma and Ba, 2015). Additionaly, label smoothing (Szegedy et al., 2016) and non-linear learning rates are often necessary in practice for optimizing the transformer-based sequence-to-sequence model. Dropout is also typically applied to both the GRU and transformer models during training.

5.2.3 Inference

As stated above, $p\left(\cdot | \pi\left(\mu\right)\right)$ is a conditional language model. Given some meaning representation μ, a natural utterance one might want to infer is the maximum a posteriori (MAP) utterance

$$\hat{\mathbf{y}} = \operatorname*{arg\,max}_{\mathbf{y} \in n} \log p\left(\mathbf{y} | \pi\left(\mu\right)\right)$$

Algorithm 5: Beam Search

```
Data:
    $\mathbf{x}$ : input sequence
    $k$ : beam size
    $T$ : maximum utterance length
    $p$ : generation model for scoring candidates.
    rerank-score : reranking hypothesis scoring function.
$\mathcal{H} \leftarrow \{[«s»]\}$
$\mathcal{H}_{complete} \leftarrow \{\}$
for $i = 1, \ldots, T$ do
    $\mathcal{H}_{new} \leftarrow \{\}$
    for $\mathbf{y} = [y_1, \ldots, y_i] \in \mathcal{H}$ do
        if $y_i = «e»$ then
            $\mathcal{H}_{complete} \leftarrow \mathcal{H}_{complete} \cup \{\mathbf{y}\}$
            $\mathcal{H} \leftarrow \mathcal{H} \setminus \{\mathbf{y}\}$
        for $y' \in \mathcal{V}_{\mathcal{Y}}$ do
            $\mathcal{H}_{new} \leftarrow \mathcal{H}_{new} \cup \{[y_1, \ldots, y_i, y']\}$
    $\mathcal{H} \leftarrow \mathrm{Top}_k(\mathcal{H}_{new}, p(\cdot|\mathbf{x}))$
Result: $\mathrm{Top}_1(\mathcal{H}_{complete}, \text{rerank-score}(\cdot|\mathbf{x}))$
```

under the model.[3] Unfortunately, the search implied by the arg max is intractable. Instead an approximate search is performed. The most commonly used search is called *beam search* (Reddy, 1977) or beam decoding. Under beam search, a set $\mathcal{H}$ of k-best candidate utterances is maintained throughout the search. k is referred to as the beam size or beam width. At each step i of the search, the next word continuations are computed for each candidate utterance prefix, yielding $k \times |\mathcal{V}_{\mathcal{Y}}|$ utterances, from which the top-k under some search criterion are selected, and $\mathcal{H}$ is updated the with the k utterances of length $i + 1$. When a completed utterance enters $\mathcal{H}$ (i.e., $y_{i+1} = «e»$), it is added to a set of completed utterances, $\mathcal{H}_{complete}$, and removed from $\mathcal{H}$. After exploring the maximum number of steps T (or some other heuristic stopping criterion), $\mathcal{H}_{complete}$ is reranked according some heuristic reranking criteria, and the top-ranked utterance is returned. When the beam size is 1, we refer to the algorithm as greedy search or greedy decoding. See algorithm 5 for a formal description of the algorithm.

[3] Technically, we are only considering valid finite utterances $\mathbf{y} = [y_1, \ldots, y_n] \in \mathcal{Y}$ with $y_1 = «s»$ and $y_n = «e»$.

Common reranking criteria include the length-normalized log likelihood,

$$\text{rerank-score}(\mathbf{y}, \mu) = \frac{\sum_{i=1}^{|\mathbf{y}|} \log p(y_i|\mathbf{y}_{1:i-1}, \pi(\mu))}{|\mathbf{y}|}$$

or a mixture of model likelihood and an auxiliary language model, p_{LM},

$$\text{rerank-score}(\mathbf{y}, \mu) = \log p(\mathbf{y}|\pi(\mu)) + \lambda \log p_{LM}(\mathbf{y}).$$

The latter method is popular in machine translation where it is easier to obtain a large monolingual corpus with which to train a language model than it is to obtain a large parallel corpus for training the translation model (Xie, 2017). When using sequence-to-sequence models for the meaning representation-to-text generation problem, practitioners often incorporate a discriminative meaning representation parser, $q(\cdot|\cdot) : \mathcal{M} \times \mathcal{Y} \rightarrow (0, 1)$, in the reranker,

$$\text{rerank-score}(\mathbf{y}, \mu) = \log q(\mu|\mathbf{y}),$$

which can help to select the most semantically correct utterances from the beam candidates. Under this setting, for a candidate utterance $\hat{\mathbf{y}} \in \mathcal{H}_{\text{complete}}$ obtained with $p\left(\cdot|\pi(\mu)\right)$ using beam search, $q\left(\mu|\hat{\mathbf{y}}\right)$ gives the probability that $[\![\hat{\mathbf{y}}]\!] = \mu$ under the model q.

Despite its wide adoption and empirical success, however, there are many known issues with beam search. The output may repeat phrases or words (Holtzman et al., 2019), or may never even terminate (Welleck et al., 2020). While these issues are often linked to differences in the maximum likelihood learning objective and test-time search procedure (Lafferty et al., 2001; Andor et al., 2016), the problems are possibly deeper as increasing the beam size often leads to worse empirical performance (Koehn and Knowles, 2017). In fact, the biases present in beam search are actually beneficial when compared to exact search (Stahlberg and Byrne, 2019). Additionally, it is well known that the set of output beam candidates may lack diversity and only differ by a small number of words (Sordoni et al., 2015; Galley et al., 2015; Li et al., 2016; Vinyals and Le, 2015; Serban et al., 2016).

5.2.4 Sampling

As an alternative to deterministic decoding, one may sample from the conditional distribution, $p\left(\cdot|\pi(\mu)\right)$. The typical method for doing this is called ancestral sampling. Ancestral sampling is very similar to greedy decoding, and works by sequentially sampling the next word $y_{i+1} \sim p\left(\cdot|\mathbf{y}_{1:i}, \pi(\mu)\right)$, and terminating when $y_{i+1} = \text{«}e\text{»}$. There are several modifications one might make to ancestral sampling in practice. To encourage more diversity in the sampled outputs, a temperature parameter τ is sometimes added to the final softmax layer,

$$p\left(y_{i+1}|\mathbf{y}_{1:i}, \pi(\mu); \tau)\right) = \operatorname{softmax}\left(\frac{\mathbf{W}^{(o)}\mathbf{g}_i + \mathbf{b}^{(o)}}{\tau}\right)_{y_{i+1}}.$$

As τ tends toward $+\infty$, the conditional distribution becomes less peaked and the differences in probability between any two words diminish, making it easier to sample an unusual continuation of the utterance sequence. In the positive limit, each word becomes equally likely,

$$\lim_{\tau \rightarrow +\infty} p\left(y|\mathbf{y}_{1:i}, \pi(\mu); \tau)\right) = \frac{1}{|\mathcal{V}_{\mathcal{Y}}|}.$$

As τ approaches zero, the distribution becomes a "one-hot" distribution,

$$\lim_{\tau \rightarrow +0} p\left(y|\mathbf{y}_{1:i}, \pi(\mu); \tau)\right) = \mathbb{1}\{y = \arg\max_{y'} p\left(y'|\mathbf{y}_{1:i}, \pi(\mu)\right)\}$$

where the probability is zero for every word except the most likely next word in the original distribution, which now has probability one.

While ancestral sampling can lead to more diverse outputs, the next word distributions are often quite peaked meaning most of the vocabulary accounts for less than 0.05 of the cumulative distribution function. For a 20 word sentence, this means that on average at least one word will be sampled from the long-tail, effectively choosing a word uniformly at random from the vocabulary. To avoid this issue, two heuristic modifications are often made to ancestral sampling, *top-k* sampling (Fan et al., 2018; Holtzman et al., 2018; Radford et al., 2019) and *nucleus* sampling (Holtzman et al., 2019).

In top-k sampling, $p(y|\mathbf{y}_{1:i}, \pi(\mu))$ is restricted to the top k most likely words. Let $\mathcal{T}_i^{(k)} \subset \mathcal{V}_\mathcal{Y}$ be the set of k most likely next words at sampling step i, i.e.,

$$\mathcal{T}_i^{(k)} = \operatorname*{arg\,max}_{S \subset \mathcal{V}_\mathcal{Y}, |S|=k} \sum_{y \in S} \log p(y|\mathbf{y}_{1:i}, \pi(\mu)).$$

The next word y_{i+1} is then sampled from the following distribution,

$$p\left(y_{i+1}|\mathbf{y}_{1:i}, \pi(\mu); \mathcal{T}_i^{(k)}\right) = \begin{cases} \frac{p(y_{i+1}|\mathbf{y}_{1:i}, \pi(\mu))}{\sum_{y' \in \mathcal{T}_i^{(k)}} p(y'|\mathbf{y}_{1:i}, \pi(\mu))} & y_{i+1} \in \mathcal{T}_i^{(k)} \\ 0 & \text{otherwise.} \end{cases}$$

Holtzman et al. (2019) show that picking the right k for top-k sampling is difficult because the next word distribution can alternate from very flat (which would suggest a large k) to very peaked (which would suggest a small k). Instead they propose restricting the subset of vocabulary to sample from to the smallest set of words such that their cumulative probability is greater than a threshold p,

$$\mathcal{N}_i^{(p)} = \operatorname*{arg\,min}_{\substack{S \subset \mathcal{V}_\mathcal{Y} \\ \sum_{y \in S} p(y|\mathbf{y}_{1:i}, \pi(\mu)) \geq p}} |S|.$$

The sampling distribution for this method which they call nucleus sampling, is computed similarly to top-k sampling,

$$p\left(y_{i+1}|\mathbf{y}_{1:i}, \pi(\mu); \mathcal{N}_i^{(p)}\right) = \begin{cases} \frac{p(y_{i+1}|\mathbf{y}_{1:i}, \pi(\mu))}{\sum_{y' \in \mathcal{N}_i^{(p)}} p(y'|\mathbf{y}_{1:i}, \pi(\mu))} & y_{i+1} \in \mathcal{N}_i^{(p)} \\ 0 & \text{otherwise.} \end{cases}$$

Nucleus sampling helps avoid sampling from the long tail of the distribution while still producing diverse samples.

5.3 Faithful Generation Through Data-Augmentation: Noise-Injection Sampling and Self-Training

We now formally define faithfulness as it relates to the meaning representation to text generation problem. Let $G : \mathcal{M} \rightarrow \mathcal{Y}$ be an arbitrary mapping from meaning representations to utterances. We say that a mapping G is faithful if

$$[\![G(\mu)]\!] = \mu \quad \forall \mu : \mu \in \mathcal{M}.$$

In words, G is faithful if the propositional content of μ (i.e., the semantics of the attribute-value pairs in μ) is correctly expressed by the generated utterance $\hat{\mathbf{y}} = G(\mu)$ for any well-formed meaning representation μ.

If G is implemented with templates as in Puzikov and Gurevych (2018), it is possible to design a faithful mapping. However, it is possible that faithfulness and naturalness are in tension, as the method of Puzikov and Gurevych (2018) did not perform as highly on human judgements of naturalness.

It is well known that implementing G with a neural model p and an inference procedure such as beam search are not sufficient to obtain a faithful model. Beam search, which only expands candidates whose next word continuations are highly probable, tends to produce low-perplexity utterances (Serban et al., 2016). Low-perplexity utterances may satisfy perceived notions of quality (Meister et al., 2020), but this is not a sufficient condition for semantic correctness. As mentioned in §5.2.3, a common approach to make p more faithful is to perform overgeneration with reranking (Dušek and Jurčíček, 2016; Juraska et al., 2018; Dušek et al., 2019; Dušek et al., 2020). The k-best list of utterances $\mathcal{H}_{complete} = \left\{\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(k)}\right\}$ is produced using beam search with $p\left(\cdot|\pi(\mu)\right)$. Then a discriminative meaning representation parser, $q(\cdot|\cdot) : \mathcal{M} \times \mathcal{Y} \rightarrow (0, 1)$, is used to rerank $\mathcal{H}_{complete}$ such that the final output utterance is

$$\hat{\mathbf{y}} = \operatorname*{arg\,max}_{\mathbf{y} \in \mathcal{H}_{complete}} q\left(\mu|\mathbf{y}\right).$$

In this setting the inference procedure selects the utterance that most likely denotes the input μ under q. While this reduces the risk of generating an incorrect utterance with respect to μ, it can still fail when either q is not accurate or when $\mathcal{H}_{complete}$ does not contain a completely correct utterance.

The critical issue here is that the final beam search hypothesis set may not contain any completely semantically correct utterances. This in part happens because neural models on natural language data[4] do not naïvely exhibit the quality of systematicity (Fodor and Pylyshyn, 1988; Phillips, 1998; Marcus, 2003; Lake and Baroni, 2018; McCoy et al., 2019; Gardner et al., 2020). A model displays systematicity if the capability of the model to perform a task implies that it can perform other structurally related tasks successfully. On natural language data, a model with systematicity should be able to exploit the algebraic and compositional nature of natural language to make correct inferences. Lake and Baroni (2018) give the example of a human speaker that understands the concept of "twice" or "again," who, upon learning a novel verb, "to dax," immediately understands the meaning of "daxed twice" or "to dax again" even though they have never seen examples of these compositions before. Empirically, they demonstrate that a recurrent neural network based text-to-meaning representation model does not possess this ability and frequently fails to generalize to novel compositions even where the individual constituents of the compositions are well represented in the training data. Bastings et al. (2018) show that this also holds going the other direction, from meaning representation to text, the more relevant direction for our present discussion.

In our case, this lack of systematicity manifests itself as a failure to realize individual attribute-values that are well represented in the training data when those attribute-values occur in novel combinations in a meaning representation at test time. We use as a case-study, the attribute-value *near=Burger King*. which in our restaurant domain dataset, the E2E Challenge dataset (Dušek et al., 2018), denotes that a restaurant x is near Burger King.

The attribute-value *near=Burger King* appears frequently in the training data in longer meaning

[4] Here we are referring to both sequence-to-sequence models but also sequence classification (Kim, 2014; McCoy et al., 2019) and sequence-pair classification (He et al., 2019).

representation/utterance pairs. In fact *near=Burger King* is positively associated with meaning representations where seven or eight attributes are specified (there are eight total unique attributes on this dataset). See Figure 5.5 where we plot the point-wise mutual information (PMI) (Church and Hanks, 1990) of the occurrence of the *near=Burger King* and the occurrence of a meaning representation of a particular size on the training set, where the size of a meaning representation $|\mu|$ is the number of attribute-value pairs in μ. The PMI is computed as

$$\mathrm{PMI}(\text{near=Burger King}, |\mu| = k) = \log \frac{p(\text{near=Burger King}, |\mu| = k)}{p(\text{near=Burger King})p(|\mu| = k)} \quad \forall k : k \in \{3, \ldots, 8\},$$

with

$$\begin{aligned} p(\text{near=Burger King}) &= \frac{\sum_{(\mu,\mathbf{y})\in\mathcal{D}} \mathbb{1}\{\text{near=Burger King} \in \mu\}}{|\mathcal{D}|} \\ p(|\mu| = k) &= \frac{\sum_{(\mu,\mathbf{y})\in\mathcal{D}} \mathbb{1}\{|\mu| = k\}}{|\mathcal{D}|} \\ p(\text{near=Burger King}, |\mu| = k) &= \frac{\sum_{(\mu,\mathbf{y})\in\mathcal{D}} \mathbb{1}\{\text{near=Burger King} \in \mu \wedge |\mu| = k\}}{|\mathcal{D}|}. \end{aligned}$$

From Figure 5.5 we can see that *near=Burger King* is negatively associated with smaller meaning representations, suggesting a neural model will struggle to generate utterances for it in the smaller meaning representation regime.

To demonstrate this, we trained a uni-directional GRU generation model on the training corpus and then tried to generate an utterance for the following meaning representation,

$$\mu = \left[\!\!\left[\begin{array}{l} \textsc{Inform} \\ \text{name=Alimentum} \\ \text{near=Burger King} \\ \text{area=city centre} \\ \text{family\_friendly=no} \end{array} \right]\!\!\right],$$

using beam search. Notice that in this case $|\mu| = 4$, indicating that the occurrence of *near=Burger King* is a relatively novel situation given the training set. We generated some beam search candidates we show below, underlining the phrases that are not semantically correct given the meaning

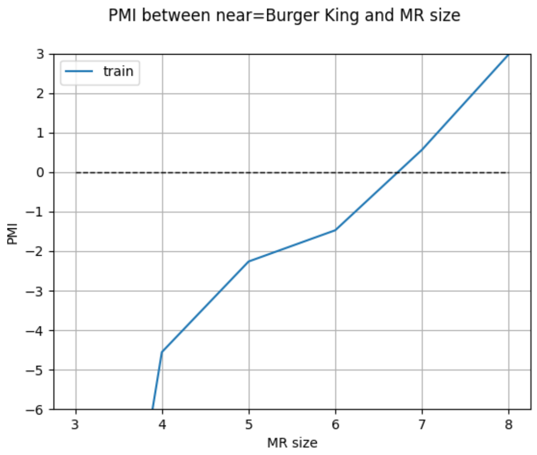


Figure 5.5: PMI between near=Burger King and meaning representation size on the E2E Challenge dataset. 0 on the *y*-axis indicates the two variables are independent.

representation,

1. Alimentum is located in the city centre near the Express by Holiday Inn. It is not family-friendly.
2. Alimentum is located in the city centre near the Yippee Noodle Bar. It is not family-friendly.
3. Alimentum is located in the city centre near the Raja Indian Cuisine. It is not family-friendly.
4. Alimentum is not family-friendly. It is located in the city centre near the Yippee Noodle Bar.
5. The Alimentum is located in the city centre near the Express by Holiday Inn. It is not family-friendly.

Right away we are confronted by their homogeneity; utterances 1,2,3 and 5 have the same syntactic structure, varying only in the phrase “near X.” Utterances 1 and 5 differ only by a single word (the initial article *The* in 5). Most importantly, none of them correctly specify that the Alimentum is near Burger King. Even with a beam size of 128, the name “Burger King” is never generated by the model![5]

[5] A beam size of 128 would be impractical for most applications. Beam sizes are typically from 4-10 in most works.

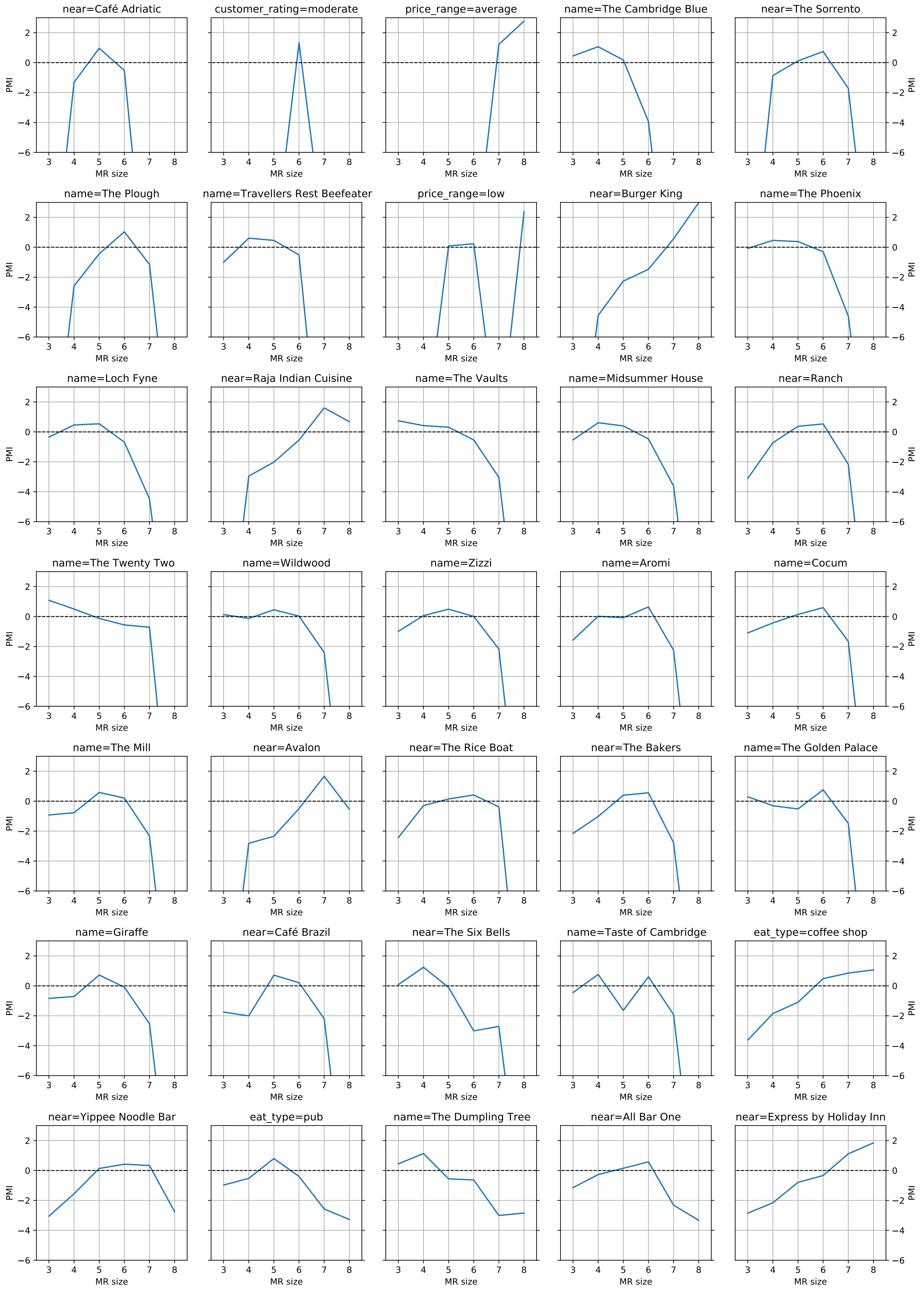


Figure 5.6: PMI between various attribute-values and meaning representation size on the E2E Challenge dataset. 0 on the *y*-axis indicates the two variables are independent.

This is more frustrating because there are plenty of training examples where even a coarse understanding of phrase structure would allow construction of a correct utterance for this case. For instance, we observe utterances containing "near Burger King" like this,

> *The Eagle is a low rated coffee shop **near Burger King** and the riverside that is family friendly and is less than £20 for Japanese food.*

while also seeing

> *Alimentum is located in the city centre **near Yippee Noodle Bar**. . . .*

where a correct utterance could be created by substituting "Burger King" in the latter instance, e.g.,

> *Alimentum is located in the city centre **near Burger King**.*

Unfortunately, the GRU model does not learn to substitute the correct prepositional phrase. Given that correct examples seem constructable from constituent phrases, it suggests that a data-augmentation approach might help to generate additional training examples that do not possess some of the spurious correlations between attribute-values and input size.

Indeed, the compositional data-augmentation scheme proposed by Andreas (2020) demonstrates improved model systematicity. Unfortunately, a rule based system of recombination risks creating disfluencies in the utterances that could potentially reduce the fluency of the learned model. Additionally, the number of spurious associations in the dataset are numerous; see Figure 5.6 for 35 of the total 79 attribute-value pairs for the E2E dataset. They all have some spurious association with meaning representation size. And we haven't even explored other associations that might exist (e.g. between two attribute-value pairs[6]). In the following subsections, we explore what an ideal data-augmentation policy might look like and then give a practical implementation of it.

[6] One might argue that it is OK for there to be correlations between the attribute-values, e.g., maybe there are fewer family friendly restaurants in the city centre. However, we often expect an NLG system to respond systematically – given an arbitrary set of attribute-values, the NLG component should realize them all correctly, and under this constraint it is desirable that any particular pairing of attribute values is independent in the NLG model.

5.3.1 An Idealized Data-Augmentation Protocol

We now introduce an idealized data-augmentation protocol and discuss some potential pitfalls and bottlenecks before proposing our implementation of it. Let $\mathcal{D}_{\mathcal{M}}$ and $\mathcal{D}_{\mathcal{Y}}$ be the empirical distributions (i.e. training dataset distributions) over the meaning representations and utterances respectively. The empirical distributions exhibit various dataset creation/annotation artifacts. For example, we have that some attributes are correlated with length (i.e., $\mathcal{D}_{\mathcal{M}}(a \in \mu, |\mu| = k) \neq \mathcal{D}_{\mathcal{M}}(a \in \mu)\mathcal{D}_{\mathcal{M}}(|\mu| = k)$) or certain attributes with each other (i.e., $\mathcal{D}_{\mathcal{M}}(a_1, a_2 \in \mu) \neq \mathcal{D}_{\mathcal{M}}(a_1 \in \mu)\mathcal{D}_{\mathcal{M}}(a_2 \in \mu)$).

Ideally, we could construct novel meaning representation examples such that their distributions did not display these correlations. Let us assume we have such a distribution, $\mathcal{D}_{\mu}^{-1}$, from which we can sample novel utterances. Given a sample $\mathcal{D}_{\mu}^{-1}$, we would then need a conditional distribution $Y(\tilde{\mu})$ from which to draw the appropriate companion utterance $\tilde{\mathbf{y}}$ such that $[\![\tilde{\mathbf{y}}]\!] = \tilde{\mu}$ while the naturalness/grammaticality of $\tilde{\mathbf{y}}$ was consistent with the empirical distribution, i.e. $S(Y) \approx S(\mathcal{D}_{\mathcal{Y}})$ where S is a projection of utterances into a syntactic space that is independent of the content. Having these two distributions, we could follow the simple data-augmentation protocol in algorithm 6 to obtain a more systematic language generation model p_*.

Coming up with a meaning representation distribution, $\mathcal{D}_{\mu}^{-1}$, is fairly straightforward. For example we could just sample the size of the meaning representation, k, uniformly at random, then sample the k attributes uniformly at random without replacement. This would ensure that attributes and meaning representation size are independent and ensure that attributes are not correlated with each other. To make up for the fact that some attribute-values are over-represented in the training set, we could sample values inversely proportional to their empirical frequency. This results in the following data generation process for

$$\tilde{\mu} = \left[\!\!\left[\begin{array}{c} \delta \\ a_1{=}v_1 \\ \vdots \\ a_k{=}v_k \end{array} \right]\!\!\right] \sim \mathcal{D}_{\mu}^{-1},$$

Algorithm 6: Idealized Data-Augmentation and Training

```
$\mathcal{A} \leftarrow \{\}$
while $|\mathcal{A}| < N$ do
    $\tilde{\mu} \sim \mathcal{D}_{\mu}^{-1}$
    $\tilde{\mathbf{y}} \sim Y(\tilde{\boldsymbol{\mu}})$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\tilde{\mu}, \tilde{\mathbf{y}})\}$
$p_* \leftarrow \mathrm{Train}(\mathcal{D} \cup \mathcal{A})$
Result: $p_*$
```

(1) Draw a dialogue act.

$$\delta \sim \mathrm{Uniform}(\{\delta_1, \delta_2, \ldots\})$$

(2) Draw a meaning representation size k.

$$k \sim \mathrm{Uniform}(\{k_{min}, \ldots, k_{max}\})$$

(3) Sample k attributes without replacement.

$$a_i \sim \mathrm{Uniform}(\{name, \ldots, near, eat\_type\} \setminus \{a_1, \ldots, a_{i-1}\}) \quad \forall i : i \in \{1, \ldots, k\}$$

(4) Sample a value v_i for each attribute a_i.

$$v_i \sim \mathrm{Categorical}\left(\mathrm{count}(v_1)^{-1}, \mathrm{count}(v_2)^{-1}, \ldots\right) \quad \forall i : i \in \{1, \ldots, k\}.$$

Unfortunately, it is not clear how we implement utterance distribution $Y(\tilde{\mu})$ since if we had an utterance generation method that could respond systematically to non-training data distributed meaning representations, we wouldn't need to perform data augmentation in the first place. As a starting point, we consider ways of generating samples from a base model p_0 trained on the available training data, i.e. $p_0 = \mathrm{Train}(\mathcal{D})$.

5.3.2 Conditional Utterance Sampling for Data-Augmentation

We cannot use p_0 with beam search as we saw previously; there are some meaning representations that p_0 won't be able to create utterances for (as we saw with *near=Burger King*). We could try a variant of ancestral sampling, but it is difficult for ancestral sampling schemes to produce extremely different outputs that break from spurious associations learned from the training distribution without hurting fluency.

The fundamental issue with ancestral sampling is that the randomness of the model is located at the word selection stage. This means that in the middle of generating a phrase it is possible for a disfluent word to be selected, which can disrupt the current phrase but also destabilize subsequent generation steps as the model tries to recover from the unusual selection. Ideally, randomness in a model would occur earlier in determining the "topicality" or "aboutness" (you might even say content selection) of the generated utterance.

Beyond the conditioning input $\pi(\mu)$, the content that is to be generated is implicitly represented by inner hidden states of the model. In Cho (2016), they argue that the hidden states, $\mathbf{g}_i$, of the sequence-to-sequence decoder lie on a manifold, as a requirement of learning the next word prediction, i.e. $\hat{y} = \arg\max_y p(y|\mathbf{y}_{1:i}, \pi(\mu)) = \arg\max_y \left(\mathbf{W}^{(o)}\mathbf{g}_i + \mathbf{b}^{(o)}\right)_y$ implies that $\hat{y}$ must be linearly separable from other words $y' \in \mathcal{V}_\mathcal{Y}$ along the hidden state manifold. The implication is that moving about the manifold will change the "topicality" of the distribution $p(y|\mathbf{y}_{1:i}, \pi(\mu))$. They further suggest adding Gaussian noise to $\mathbf{g}_i$ as a way to obtain random samples from p, which we refer to as *noise-injection sampling*. While Cho (2016) used noise-injection sampling as a means to generate semantically correct but syntactically diverse outputs, one of our contributions is to use this scheme as a means to generate semantically divergent outputs (that maintain grammaticality), for use as a data-augmentation tool.

We show the noise-injection sampling algorithm in Figure 5.7 along with greedy decoding and ancestral sampling to emphasize the how the location of the stochasticity moves from the next word selection (Alg. 8 line 8) to a perturbation of the hidden state (Alg. 9 line 7). Note that in line 7 of the noise injection sampling algorithm, the standard deviation of the normal distribution, $\frac{\sigma}{i}$, is scaled by the decoder step i and in the limit turns to zero, i.e. $\lim_{i \to +\infty} \mathbf{g} + \boldsymbol{\epsilon}_i = \mathbf{g}$. The inuition behind this scaling is that we add the most noise at the first steps of decoding, which encourages the decoder to start from a topically novel region of the hidden state manifold. As the decoding proceeds, the noise reduces along with the chances of sending the decoder off the manifold and destabilizing the decoding, and gradually we converge on the behavior of greedy decoding.

Deterministic operation   Stochastic operation

Alg. 7: Greedy Decoding

```
$\mathbf{h}_{1:m} \leftarrow \text{enc}(\pi(\mu))$
$\hat{y}_1 \leftarrow$ «s»
$\hat{\mathbf{y}} \leftarrow [\hat{y}_1]$
$i \leftarrow 1$
while $\hat{y}_i \neq$ «e» do
    $\mathbf{g}_i \leftarrow \text{dec}(\hat{\mathbf{y}}, \mathbf{h}_{1:m})$

    $\hat{y}_{i+1} \leftarrow \arg\max_y p(y|\mathbf{g}_i)$
    $\hat{\mathbf{y}} \leftarrow \hat{\mathbf{y}} \oplus [\hat{y}_{i+1}]$
    $i \leftarrow i + 1$
Result: $\hat{\mathbf{y}}$
```

Alg. 8: Ancestral Sampling

```
$\mathbf{h}_{1:m} \leftarrow \text{enc}(\pi(\mu))$
$\hat{y}_1 \leftarrow$ «s»
$\hat{\mathbf{y}} \leftarrow [\hat{y}_1]$
$i \leftarrow 1$
while $\hat{y}_i \neq$ «e» do
    $\mathbf{g}_i \leftarrow \text{dec}(\hat{\mathbf{y}}, \mathbf{h}_{1:m})$

    $\hat{y}_{i+1} \sim p(\cdot|\mathbf{g}_i)$
    $\hat{\mathbf{y}} \leftarrow \hat{\mathbf{y}} \oplus [\hat{y}_{i+1}]$
    $i \leftarrow i + 1$
Result: $\hat{\mathbf{y}}$
```

Alg. 9: Noise Injection Sampling

```
$\mathbf{h}_{1:m} \leftarrow \text{enc}(\pi(\mu))$
$\hat{y}_1 \leftarrow$ «s»
$\hat{\mathbf{y}} \leftarrow [\hat{y}_1]$
$i \leftarrow 1$
while $\hat{y}_i \neq$ «e» do
    $\mathbf{g}_i \leftarrow \text{dec}(\hat{\mathbf{y}}, \mathbf{h}_{1:m})$
    $\boldsymbol{\epsilon_i} \sim \text{Normal}(\mathbf{0}, \frac{\sigma}{i})$
    $\hat{y}_{i+1} \leftarrow \arg\max_y p(y|\mathbf{g}_i + \boldsymbol{\epsilon_i})$
    $\hat{\mathbf{y}} \leftarrow \hat{\mathbf{y}} \oplus [\hat{y}_{i+1}]$
    $i \leftarrow i + 1$
Result: $\hat{\mathbf{y}}$
```

Figure 5.7: A comparison of greedy decoding, ancestral sampling, and noise injection sampling.

We can understand noise-injection sampling as a compromise between greedy decoding and ancestral sampling; rather than draw a sequence of utterance tokens stochasticity, we instead draw a sequence of hidden state spaces. Given the sequence of hidden state spaces, the corresponding sequence of utterance tokens is deterministically decided by the most likely next token given the last hidden state. This next word selection strategy helps to avoid disfluent continuations.

In Figure 5.8 we show examples of samples obtained with noise-injection sampling as well as some ancestral sampling schemes. We can see that the ancestral sampling examples are not very diverse. The noise-injection sampling example, however, semantically diverges from the input while maintaining fluency. It was even able to generate an utterance containing the phrase "near Burger King" which is was practically impossible to generate with beam search.

In Table 5.1, we show the probability of generating the example

«s» the waterman is not family friendly and is located near burger king . «e»

under the various sampling schemes. In our present case, top-k and nucleus sampling have very similar distributions to the ancestral sampling distribution ($p(y_{i+1}|\mathbf{g}_i)$). All three of these techniques assign a very low probability to generating the example utterance, and in the case of nucleus sampling, it only gives non-zero probability when using a nucleus of 0.99 cumulative probability

Ancestral Sampling
«s» the eagle is a non family - friendly italian food establishment . «e»
«s» the eagle is a italian food place and is not family - friendly . «e»
«s» some italian food can be found at the eagle . «■» it 's not family - friendly . «e»
«s» the eagle serves italian food . «■» it has a «?» «?» and is not family friendly . «e»
«s» the eagle is a family friendly place for italian food . «e»

Top-K Sampling ($k = 100$)
«s» the eagle serves italian food . «■» it is not family - friendly . «e»
«s» the eagle serves italian cuisine . «■» it is not family - friendly . «e»
«s» the eagle has italian food and is not family - friendly «e»
«s» the eagle is italian place . «■» it is not family - friendly . «e»
«s» the eagle provides fast food . «■» it is not family - friendly . «e»

Nucleus Sampling ($p = 0.95$)
«s» the eagle serves italian food and is not family - friendly . «e»
«s» the eagle is not family - friendly and serves italian food . «e»
«s» the eagle is not family - friendly . «■» they serve italian food . «e»
«s» italian food is served at the eagle . «■» not family - friendly . «e»
«s» the eagle is a good place to eat italian food . «■» it is not family - friendly . «e»

Noise-Injection Sampling ($\sigma = 2.0$)
«s» the eagle in the city centre . «■» it is not family - friendly . «■» it is located near the burger king . «e»
«s» the eagle serves italian food . «e»
«s» the waterman is not family friendly and is located near burger king . «e»
«s» the eagle is located near the burger king . «e»
«s» the eagle is a non family - friendly italian food place . «e»

Figure 5.8: Example samples taken after conditioning on the following meaning representation: ⟦INFORM; name=The Eagle; food=Italian; family_friendly=yes⟧.

(i.e. $\mathcal{N}_i^{(.99)}$)! In particular, noise-injection sampling puts much more probability mass on generating relatively rare attribute-value realizations ($i = 2$, "waterman" and $i = 11$, "burger"). This aspect of noise-injection sampling makes it very attractive for data-augmentation as we can use it to create semantically novel utterances that are not represented in the training dataset, while still producing fluent outputs.

5.3.3 A Practical Data-Augmentation Protocol

Because of its ability to generate semantically divergent and novel outputs while maintaining fluency, we adopt this noise-injection sampling as our method of sampling utterances, Y, for data-

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
y_{i+1}	the	waterman	is	not	family	friendly	and	is	located	near	burger	king	.	*«e»*
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{T}_i^{(5)}\right)$	0.874	0.004	0.380	0.397	0.915	0.147	0.338	0.111	0.147	0.168	0.000	0.954	0.931	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{T}_i^{(25)}\right)$	0.792	0.004	0.344	0.371	0.877	0.147	0.327	0.101	0.111	0.148	0.001	0.935	0.911	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{T}_i^{(50)}\right)$	0.778	0.004	0.339	0.366	0.872	0.147	0.326	0.100	0.105	0.146	0.001	0.930	0.910	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{T}_i^{(75)}\right)$	0.772	0.004	0.338	0.364	0.870	0.147	0.326	0.100	0.103	0.145	0.001	0.928	0.909	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{T}_i^{(100)}\right)$	0.768	0.004	0.337	0.363	0.869	0.147	0.326	0.100	0.102	0.144	0.001	0.926	0.909	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{N}_i^{(.95)}\right)$	0.796	0.000	0.352	0.377	0.909	0.148	0.342	0.104	0.103	0.150	0.000	0.964	0.950	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{N}_i^{(.96)}\right)$	0.789	0.000	0.349	0.374	0.898	0.148	0.338	0.103	0.102	0.149	0.000	0.954	0.939	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{N}_i^{(.97)}\right)$	0.781	0.000	0.345	0.370	0.892	0.148	0.333	0.102	0.101	0.148	0.000	0.947	0.931	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{N}_i^{(.98)}\right)$	0.773	0.000	0.342	0.367	0.882	0.148	0.332	0.101	0.100	0.146	0.000	0.937	0.924	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i;\mathcal{N}_i^{(.99)}\right)$	0.765	0.004	0.338	0.363	0.874	0.148	0.329	0.100	0.099	0.145	0.001	0.928	0.917	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i\right)$	0.758	0.004	0.335	0.359	0.865	0.147	0.326	0.099	0.098	0.143	0.001	0.919	0.908	0.810
$p\left(y_{i+1} \mid \mathbf{g}_i+\boldsymbol{\epsilon}_i\right)$	0.321	0.170	0.408	0.489	0.785	0.514	0.459	0.562	0.440	0.731	0.599	0.972	0.903	0.984

Table 5.1: Word selection probabilities when using ancestral sampling, top-k sampling (for $k \in \{5, 25, 50, 75, 100\}$), nucleus samplling (for $p \in \{0.95, 0.96, 0.97, 0.98, 0.99\}$), and noise-injection sampling ($\sigma = 2.0$).

Alg. 10: Data Augmentation with Noise-Injection Sampling and Self-Training

```
   Data: training dataset $\mathcal{D}$, number of synthetic datapoints to generate $N$
1  $p_0 \leftarrow \text{Train}_{\mathbf{y}}\,(\mathcal{D})$
2  $q \leftarrow \text{Train}_{\mu}\,(\mathcal{D})$
3  $\mathcal{A} \leftarrow \{\}$
4  while $|\mathcal{A}| < N$ do
5      $\tilde{\mu} \sim \mathcal{D}_{\mu}^{-1}$
6      $\tilde{\mathcal{Y}}_{200} \leftarrow \left\{\tilde{\mathbf{y}}^{(i)} \sim p_0\left(\cdot|\pi(\tilde{\mu}), \boldsymbol{\epsilon}^{(i)}\right) \quad \forall i : i \in \{1, \ldots, 200\}\right\}$
7      $\tilde{\mathcal{Y}}_{20} \leftarrow \text{Top}_{20}\left(\tilde{\mathcal{Y}}_{200}, \lambda \mathbf{y} : \frac{\log p_0(\mathbf{y}|\pi(\tilde{\mu}), \boldsymbol{\epsilon})}{|\mathbf{y}|}\right)$
8      for $\hat{\mathbf{y}} \in \tilde{\mathcal{Y}}_{20}$ do
9          $\hat{\mu} \leftarrow q\,(\hat{\mathbf{y}})$
10         if $\neg\,\text{Filter}\,(\hat{\mu}, \hat{\mathbf{y}})$ then
11             $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\hat{\mu}, \hat{\mathbf{y}})\}$
12 $p_* \leftarrow \text{Train}_{\mathbf{y}}(\mathcal{D} \cup \mathcal{A})$
   Result: $p_*$
```

augmentation. We show our actual data-augmentation scheme in algorithm 10 and now walk through some of the implementation details.

Train base generator p_0 and meaning representation parser q. The algorithm begins by training the base generator (i.e., naïve sequence-to-sequence model) and meaning representation parser q. Both models are trained on the same data, with the only real change to the Train sub-routine being which part of a training example is the output and which is the input. Alternatively, q can also be implemented using regular-expression-based rules. We defer detailed explanation of q until the experiments; it suffices to understand q as a mapping from utterances to meaning representations.

Sampling a meaning representation, $\tilde{\mu}$ We use the meaning representation sampling scheme described in §5.3.1 to implement the distribution $\mathcal{D}_{\mu}^{-1}$.

Generating a novel utterance from p_0 with noise-injection sampling. In line 6 we take 200 noise-injection samples from p_0 to construct a candidate set of utterances, $\tilde{\mathcal{Y}}_{200}$. We use $\sigma = 2.0$ after manually experimenting with a range of values from 0.1-3.0 since it gave reasonably fluent outputs while also generating semantically divergent outputs.. phrases with no realized attributes. From these we use only the top 20 utterances, $\tilde{\mathcal{Y}}_{20}$ by average log-likelihood, $\frac{\log p(\tilde{\mathbf{y}}^{(i)}|\pi(\tilde{\mu}),\boldsymbol{\epsilon}^{(i)})}{|\tilde{\mathbf{y}}^{(i)}|}$ (line 7). We do this selection step so as to be extra cautious and avoid adding any potentially disfluent utterances to $\mathcal{A}$.

Predict meaning representation $\hat{\mu}$ from $\hat{\mathbf{y}}$. Because the noise-injection sampling produces highly semantically divergent utterances, it is unlikely that $[\![\hat{\mathbf{y}}]\!] = \tilde{\mu}$. Instead we use the meaning representation parser, q, to recover the most likely meaning representation, $\hat{\mu} = q(\hat{\mathbf{y}})$. More details about q will be provided in §5.3.6.

Check synthetic datapoint $(\hat{\mu}, \hat{\mathbf{y}})$. We do one last quality check on the synthetic example $(\hat{\mu}, \hat{\mathbf{y}})$ before adding it to the augmented dataset, $\mathcal{A}$. We make sure that the probability of $\hat{\mu}$ under q is above 0.5 when using a model-based meaning representation parser. When using a rule-based meaning representation parser, we check to make sure that there are no repeated attribute-value-pairs in $\hat{\mathbf{y}}$, e.g., "Aromi is a coffee shop and it is a coffee shop," by discarding any utterances that trigger multiple rules for any attribute. Meaning representation/utterances that have been previously generated are also discarded. If the meaning representation/utterance pair passes these final quality checks, we add it to $\mathcal{A}$.

Train an augmented generator p_* on $\mathcal{D} \cup \mathcal{A}$. After generating a synthetic dataset, $\mathcal{A}$, we train a new generation model, p_*, on the union of the original training data and the newly generated synthetic data. We refer to this model as the augmented generator and, as we will show empirically, the augmented generator is more faithful than the base generator, p_0. We call this process self-training because p_* and p_0 share the same architecture, and p_* is trained on data produced by p_0. In theory, we could repeat this process similar to iterative back-translation (Hoang et al., 2018), using p_0 to produce a p_1 which could then produce a p_2 and so on. However, we did not experiment with this because we found that p_* improved in faithfulness significantly over p_0 after one pass through algorithm 10.

Dataset	Train	Valid	Test	Unique Dialogue Acts	Unique Attribute Values
E2E Chal.	42,061	4,672	4,693	1	8
Laptops	15,888	5,298	5,297	14	19
TVs	8,442	2,814	2,812	14	15

Table 5.2: Dataset statistics for noise-injection and self-training experiments.

5.3.4 Datasets

We experimentally validate the noise-injection sampling and self-training data-augmentation scheme on three recent dialogue generation datasets, the E2E Challenge dataset (Novikova et al., 2017) and the Laptops and TVs datasets (Wen et al., 2016). Each dataset consists of meaning representations paired with one or more reference utterances. All attribute values come from a closed vocabulary.

The three datasets also represent different training size conditions; with the E2E Challenge dataset representing the "large data" training condition and the Laptops and TVs dataset representing "small data" conditions. See Table 5.2 for dataset size statistics. The E2E Challenge dataset has only one dialogue act, INFORM, and its training meaning representations contain three to eight unique attributes. The Laptops and TVs datasets contain a more diverse set of meaning representation/utterance pairs. There are 14 unique dialogue acts. The number of minimum and maximum attributes varies according to the dialogue act. See Table 5.3 for a list of the unique dialogue acts and attributes for the three training sets.

Delexicalization Prior work using neural natural language generation models often relies on delexicalization, that is, replacing realizations of named-entity or numeric values in an utterance with a placeholder token, in order to alleviate data sparsity issues and yield better generalization when a generating utterances about named-entities not seen in the training dataset. For example, on the E2E Challenge dataset, the name and near attributes are often delexicalized because they are proper names of establishments that are simple to find and replace in the utterance. When delexicalizing the name and near attributes, the fully lexicalized utterance

Dataset	Dialog Acts	Attributes	
E2E Challenge	INFORM	name	
		near	
		eat_type	
		food	
		area	
		price_range	
		customer_rating	
		family_friendly	
Laptops	INFORM	family	weight
	INFORMONLYMATCH	price_range	platform
	INFORMONMATCH	battery_rating	memory
	INFORMALL	drive_range	drive
	INFORMCOUNT	weight_range	processor
	INFORMNOINFO	is_for_business_computing	
	RECOMMEND	name	
	COMPARE	type	
	SELECT	price	
	SUGGEST	warranty	
	CONFIRM	battery	
	REQUEST	design	
	REQUESTMORE	dimension	
	GOODBYE	utility	
TVs	INFORM	family	audio
	INFORMONLYMATCH	price_range	
	INFORMONMATCH	screen_size_range	
	INFORMALL	eco-rating	
	INFORMCOUNT	hdmi-port	
	INFORMNOINFO	has_usb-port	
	RECOMMEND	name	
	COMPARE	type	
	SELECT	price	
	SUGGEST	resolution	
	CONFIRM	power_consumption	
	REQUEST	accessories	
	REQUESTMORE	color	
	GOODBYE	screen_size	

Table 5.3: The dialogue acts and attributes for the E2E Challenge, Laptops, and TVs datasets.

Near The Six Bells is a venue that is children friendly named The Golden Curry.

can be delexicalized as

Near «near» is a venue that is children friendly named «name».

Delexicalized utterances can be re-lexicalized as a post-processing step, where the placeholder token is replaced with the correct value text.

On the E2E Challenge dataset, we experiment with delexicalization of the *Name* and *Near* attributes since they have a relatively large vocabulary of valid slot fillers, some of which are only seen infrequently in the training data; it can be difficult for fully lexicalized models to produce some of the rarer location names for these attributes. However, since delexicalization might be difficult or impossible in other domains, we implement both delexicalized and lexicalized versions of the generation models on the E2E dataset to more fully evaluate the self-training method.

The evaluation script for the Laptops and TVs datasets uses delexicalization to evaluate attribute realization error, and so we use it here to be consistent with prior work, delexicalizing all possible attributes.

While delexicalization effectively solves some problems in faithful generation (e.g., the difficulty in generating the phrase *"near Burger King"*), it is difficult to apply to attributes that are not realized by a small vocabulary of names or phrases. Even in those cases it introduces extra complexity if the surrounding context will depend on the particular value in any way (e.g., *"near an «name»"* would not be grammatical if replacing the *«name»* placeholder with *"Burger King"*).

5.3.5 Text Generation Models

We use a two-layer, unidirectional GRU architecture with Bahdanau style attention for our sequence-to-sequence meaning representation-to-text model. We set $D_w = D_h = D_{\mathcal{E}} = D_{\mathfrak{D}} = 512$, that is, we use 512-dimensional embedding and hidden states as described in Appendix A.

We fit model parameters, θ, by minimizing the negative log-likelihood of the training set, $\mathcal{D}$, i.e.

$$\mathcal{L}(\theta) = -\sum_{(\mu,\mathbf{y})\in\mathcal{D}} \log p\left(\mathbf{y}|\pi(\mu);\theta\right).$$

Our choice of linearization strategy, π, differs slightly for the E2E Challenge and ViGGO corpora. For the former, we arbitrarily and consistently order the eight attribues, explicitly representing absent attribute-values with a *N/A* token.[7] For example, for the meaning representation

$$\mu = \left[\!\!\left[\begin{array}{l}\textsc{Inform}\\ \text{name=The Mill}\\ \text{near=Avalon}\\ \text{food=Italian}\end{array}\right]\!\!\right],$$

we would have the following linearization,

$$\pi(\mu) = \left[\begin{array}{l}\textit{«s»,}\\ \textit{eat\_type=N/A,}\\ \textit{near=Avalon,}\\ \textit{area=N/A,}\\ \textit{family\_friendly=N/A,}\\ \textit{customer\_rating=N/A,}\\ \textit{price\_range=N/A,}\\ \textit{food=Italian,}\\ \textit{name=The Mill}\\ \textit{«e»}\end{array}\right].$$

We omit the dialogue act since the E2E Challenge dataset only has one, INFORM. When using the delexicalized model variant, we omit the name attribute since it is always present, and only indicate that the near attribute is present with a placeholder token, yielding

[7] In our initial experiments, including absent attributes in the input this way performed slightly better than omitting them.

$$\pi(\mu) = \left[\begin{array}{l} \textit{«s»}, \\ \textit{eat\_type=N/A}, \\ \textit{near=«present»}, \\ \textit{area=N/A}, \\ \textit{family\_friendly=N/A}, \\ \textit{customer\_rating=N/A}, \\ \textit{price\_range=N/A}, \\ \textit{food=Italian}, \\ \textit{«e»} \end{array}\right].$$

For the Laptops and TVs corpus, we similarly determine an arbitrary ordering but omit any absent attribute-values since there are too many to represent all of them explicitly. Additionally, since there are multiple dialogue acts we prepend a token representing the dialogue act to the start of the sequence. As an example, for the following meaning representation

$$\mu = \left[\!\!\left[\begin{array}{l} \textsc{InformCount} \\ \text{count=40} \\ \text{family=don't care} \\ \text{battery\_rating=excellent} \end{array}\right]\!\!\right]$$

we would have the following linearization,

$$\pi(\mu) = [\textit{«s»}, \textit{inform\_count}, \textit{count=«NUM»},\textit{family=don't care}, \textit{battery\_rating=excellent}, \textit{«e»}]\,.$$

When generating utterances for evaluation (i.e. not for use in noise-injection sampling) we use either greedy decoding or beam decoding with a beam size of eight. The beam search terminates after eight candidates have been generated; the candidates are reranked by average token log-likelihood, $\frac{\log p(\mathbf{y}|\pi(\mu))}{|\mathbf{y}|}$. In these experiments, we **do not** use a discriminative reranker to ensure the faithfulness of the selected beam candidate.

5.3.6 Meaning Representation Parsing Models

Given a novel utterance $\hat{\mathbf{y}}$ sampled from p_0, we need to reliably parse the implied meaning representation $\hat{\mu} = q(\hat{\mathbf{y}})$, where q is our parsing model. We have two things going for us in our experimental setting. First, even with noise-injection sampling, model outputs are fairly patterned, reducing the variability of the utterances we need to parse in practice; a meaning representation parser on real human data would need to be much more robust.

Second, the meaning representation in this study are flat lists of attributes that are somewhat independent of each other. We only need to detect the presence of each attribute and its value. For the Laptops and TVs datasets we also need to recover the dialog act but these also are signaled by a fairly limited repertoire of cues, e.g. "we recommend." Given this, we experiment with both hand crafted regular expression rules and learned classifiers to predict the value of an attribute if present or that it is missing.

Rule-based parser ($q_{\Re}$) We design hand-crafted regular expression based rules to match for the presence of key phrases for each of the attributes and dialouge acts in the datasets while also checking to make sure that there is only one match per attribute.

To construct the rules, we look through both the training data references as well as the generation model outputs as this is what the rules will be operating on in practice. For each lexicalized attribute (and dialogue act) we develop a list of regular expressions such as,

`/is (family|kid|child) friendly/` ⇒ family_friendly=yes.

For the delexicalized attributes, we simply check for the presence of the placeholder token.

We design these rules to be high precision, as it is safer to miss out on more obscure varieties of utterance to avoid adding incorrectly parsed data points. However, in many cases the rules are also high recall as well. The average F-score on the E2E validation set is 0.93.

Classifier-based parser (q_{ϕ}) It is perhaps too optimistic to believe we can construct reasonable rules in all cases. Rule creation quickly becomes tedious and for more complex meaning representations, this would become a bottleneck. To address these concerns, we also study the feasibility of using learned classifiers to predict the presence and value of the attributes. For each attribute in the E2E dataset, we trained a separate convolutional neural network classifier to predict the correct attribute value (or *n/a* if the attribute is not present) from an utterance.

The architecture largely follows that of Kim (2014). Let $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}_y| \times D_w}$ be an embedding matrix for the utterance token vocabulary, $\mathbf{W}$, with each token $y \in \mathcal{V}_y$ associated with a row in $\mathbf{W}$, which we indicate with $\mathbf{W}_y \in \mathbb{R}^{D_w}$. For each attribute a, the set of possible values (including *n/a*) is denoted $\mathcal{V}_a$.

Given an utterance $\mathbf{y} = [y_1, \ldots, y_n]$, we first embed the utterance tokens to obtain a sequence of word embeddings,

$$\mathbf{w}_1, \ldots, \mathbf{w}_n = \mathbf{W}_{y_1}, \ldots, \mathbf{W}_{y_n}.$$

We then apply a series of unigram, bigram, and trigram convolutional filters (i.e., convolutional feature widths k of 1, 2, and 3 respectively) each with D_f output features, which are computed as,

$$h_{k,i} = \max_{j \in \left\{1 - \left\lfloor \frac{k}{2} \right\rfloor, \ldots, n + \left\lfloor \frac{k}{2} \right\rfloor - k + 1\right\}} \operatorname{ReLU}\left(b^{(k,i)} + \mathbf{u}^{(k,i)} \cdot \begin{bmatrix} \mathbf{w}_j \\ \mathbf{w}_{j+1} \\ \vdots \\ \mathbf{w}_{j+k-1} \end{bmatrix} \right) \quad \forall k, i : \begin{array}{l} k \in \{1, 2, 3\}, \\ i \in \{1, \ldots, D_f\} \end{array}$$

where $b^{(k,i)} \in \mathbb{R}$ and $\mathbf{u}^{(k,i)} \in \mathbb{R}^{kD_w}$ are learned parameters and we use the same zero-padded convolution described in §3.2.2.3 with $\mathbf{w}_i = \mathbf{0}$ for $i < 1$ and $i > n$. The individual convolutional features are collected in a hidden state encoding of the utterance, $\mathbf{h} \in \mathbb{R}^{kD_f}$, with

$$\mathbf{h} = \left[h_{1,1}, \ldots, h_{1,D_f}, h_{2,1}, \ldots, h_{2,D_f}, h_{3,1}, \ldots, h_{3,D_f}\right].$$

The hidden state is then fed through a two layer feed-forward network to compute the probability of a particular attribute value,

$$q_a\left(v|\mathbf{y}\right) = \operatorname{softmax}\left(\mathbf{U}^{(a,2)}\left(\mathbf{U}^{(a,1)}\mathbf{h} + \mathbf{b}^{(a,1)}\right) + \mathbf{b}^{(a,2)}\right)_v$$

where $\mathbf{U}^{(a,1)} \in \mathbb{R}^{D_w \times k D_w}$, $\mathbf{b}^{(a,1)} \in \mathbb{R}^{D_w}$, $\mathbf{U}^{(a,2)} \in \mathbb{R}^{|\mathcal{V}_a| \times D_w}$, and $\mathbf{b}^{(a,2)} \in \mathbb{R}^{|\mathcal{V}_a|}$ are learned parameters. If μ contains a m attribute-value pairs, $a_1 = v_1, \ldots, a_m = v_m$, the probability of μ under the parsing model is $q(\mu|\mathbf{y}) = \prod_{i=1}^{m} q_{a_i}(v_i|\mathbf{y})$.

Each attribute classifier has distinct parameters and is trained on the training set but minimizing the negative log-likelihood,

$$\mathcal{L}(\phi) = -\sum_{(a=v,\mathbf{y}) \in \mathcal{D}} \log q_a(v|\mathbf{y};\phi),$$

using minibatch stochastic gradient descent on the training set, $\mathcal{D}$.

During training we apply dropout (with drop rate of 0.25) to the embedding layer, convolutional filter outputs, and hidden layers. We train for 30 epochs with gradient descent using a learning rate of 0.25 and weight decay penalty of 0.0001, using validation set F1 as our model selection criterion. The average E2E validation F-score is 0.94.

5.3.7 Experiments

5.3.7.1 E2E Challenge

We train base generators p_0 on the original training data $\mathcal{D}$, with and without delexicalizing the name and near attributes. We train for 500 epochs with gradient descent. We use a batch size of 128, with a learning rate of 0.25, weight decay penalty of 0.0001, and a dropout probability of 0.25. We select the best model iteration using validation set BLEU score.[8]

Using the self-training method outlined in §5.3.3, we create augmented datasets using either $q_{\mathfrak{R}}$ or q_ϕ, which we refer to as $\mathcal{A}_{q_{\mathfrak{R}}}$ and $\mathcal{A}_{q_\phi}$ respectively We only use the model parser, q_ϕ, in the delexicalized setting. We repeat the while loop in the data-augmentation algorithm 25,000 times for each valid MR size $3, \ldots, 8$, yielding 1,591,788 additional samples for the lexicalized $p_0/q_{\mathfrak{R}}$ pairing, and 501,909 for $p_0/q_{\mathfrak{R}}$ and 384,436 for p_0/q_ϕ delexicalized pairings.[9]

[8] We use the official shared task script to compute automatic quality metrics on the E2E dataset.

[9] The number of additional examples generated by the delexicalized approaches is an order of magnitude smaller than the lexicalized case because we discard duplicate generated utterances. With delexicalization, duplicates are more frequently generated by the model.

For both $\mathcal{D} \cup \mathcal{A}_{q_{\mathfrak{R}}}$ and $\mathcal{D} \cup \mathcal{A}_{q_\phi}$ we train new generators p_* using the same training setting as above (although we terminate training after 50 epochs because the models converge much faster with the additional data). We report BLEU, ROUGE-L, and METEOR on the E2E Challenge test set, using the official shared-task evaluation script. We show results for both greedy decoding and beam decoding with beam size 8 under p_0 and p_* models. We compare our models to the best sequence-to-sequence model, Slug (Juraska et al., 2018), the best grammar-rule based model, DANGNT (Nguyen and Tran, 2018), and the best template based model, TUDA (Puzikov and Gurevych, 2018), as determined during the shared task evaluation (Dušek et al., 2019).

5.3.7.2 Laptops and TVs

We perform similar experiments on the Laptops and TVs datasets. We train a separate p_0 model for each dataset for 300 epochs with a learning rate of 0.1 for Laptops and 0.25 for TVs. The weight decay penalty is 0.0001 and dropout probability is 0.25. Best model iteration is determined by validation set BLEU score. As in the E2E experiments, we create an augmented dataset for both the Laptops and TVs dataset using the method outlined in §5.3.3. We then train new generators p_* on the union of original training data and the augmented dataset.

We repeat the while loop in the noise-injection sampling algorithm 25,000 times for each dialogue act and legal dialogue act size.[10] We obtain 373,468 and 33,478 additional samples for the Laptops and TVs datasets respectively.

We automatically evaluate our models using the evaluation script of (Wen et al., 2016), which computes BLEU scores, as well as slot alignment error rate (since this dataset is almost fully delexicalized, it simply checks for the presence of the correct attribute placeholders according to the MR). We compare again to the Slug model as well as the Semantically Conditioned LSTM (SCLSTM) (Wen et al., 2015) which report state-of-the-art results on these datasets.

[10] A number of attributes S is "legal" if we observe a training instance with that dialogue act instance with S attributes in the original training data.

Model				BLEU	ROUGE-L	METEOR
		Slug		66.19	67.72	44.54
		DANGNT		59.90	66.34	43.46
		TUDA		56.57	66.14	45.29
Delex.	Base Gen. (p_0)		greedy	66.91	68.27	44.95
			beam	**67.13**	**68.91**	45.15
	Aug. Gen. (p_*)	Rule Parser ($q_{\mathfrak{R}}$)	greedy	65.57	67.71	45.56
			beam	66.28	68.08	**45.78**
		Model Parser (q_ϕ)	greedy	63.76	67.31	44.94
			beam	64.23	67.54	45.17
Lex.	Base Gen. (p_0)		greedy	60.35	64.51	41.82
			beam	61.81	65.83	42.69
	Aug. Gen. (p_*)	Rule Parser ($q_{\mathfrak{R}}$)	greedy	64.74	68.21	44.46
			beam	64.81	67.83	44.39

Table 5.4: Automatic quality metrics on the E2E test set. Baseline methods all rely on at least partial delexicalization, puting our lexicalized models at a relative disadvantage.

Model				Name	Near	Family Friendly	Area	Customer Rating	Food	Price Range	Eat Type	All
			Slug	0	0	6	1	6	10	35	9	67
			DANGNT	0	0	18	0	0	0	0	58	76
			TUDA	0	0	0	0	0	0	0	0	**0**
delex.	p_0		greedy	0	0	23	23	16	26	27	0	115
			beam	0	0	60	3	9	3	8	0	83
	p_*	$q_{\mathfrak{R}}$	greedy	0	0	0	0	0	0	0	0	**0**
			beam	0	0	0	0	0	0	0	0	**0**
		q_ϕ	greedy	0	0	1	0	8	1	9	0	19
			beam	0	0	0	0	3	0	0	0	3
lex.	p_0		greedy	145	141	14	15	2	14	2	0	333
			beam	155	124	62	0	0	0	0	0	341
	p_*	$q_{\mathfrak{R}}$	greedy	0	0	2	0	0	125	0	0	127
			beam	0	2	0	0	0	119	0	0	121

Table 5.5: Attribute realization errors on the E2E test set. The Slug model and our delexicalized models delexicalize the NAME and NEAR slots, thus making 0 errors on these attributes. DANGNT and TUDA models perform complete delexicalization.

5.3.8 Results

5.3.8.1 E2E Challenge

Automatic evaluation metrics are shown in Table 5.4. Surprisingly, p_0 using greedy decoding surpasses all of the baseline systems on all three automatic metrics. This is quite shocking as the Slug baseline ensembles three different sequence-to-sequence models producing 10 outputs each using beam search and reranking based on slot alignment to select the final generation output. The $p_*/q_{\mathfrak{R}}$ model remains competitive with Slug, again even using greedy decoding. The p_*/q_ϕ starts under-performing Slug on BLEU score but remains competitive on ROUGE-L and METEOR again when using greedy decoding. Overall the augmented training data tends to hurt generation with respect to automatic quality measures. In this regard, the added noise of the model-based parser, q_ϕ, exacerbates things as it reduces quality more than the rule-based parser, $q_{\mathfrak{R}}$.

In the lexicalized setting, p_0 produces lower quality output than the Slug system. However, the augmented training procedure increases the quality of the lexicalized p_* model which beats Slug on ROUGE-L.

The automatic quality evaluations are somewhat misleading, however. To gain more insight into model performance we apply our rule based parser to estimate attribute realization error for all system outputs on the test set, similarly to Dušek et al. (2019) (e.g., if the MR specifies food=French, we check to make sure the generated utterance says so). The results of this evaluation are shown in Table 5.5. Immediately, it is revealed that p_0 is far worse than the baseline methods making 115 and 83 errors using greedy and beam decoding respectively.

It is here that we see the benefits of the data-augmentation. The $p_*/q_{\mathfrak{R}}$ model achieves zero test set errors even when using the greedy decoding. The p_*/q_ϕ model is slightly worse (in agreement with the automatic quality measurements), but its greedy search is still superior to the more sophisticated Slug decoder, achieving 19 total test set errors compared to Slug's 67 errors.

The lexicalized p_0 model has especially high error rates, particularly on the name and near attributes. With augmented data training, the p_* model reduces these errors to zero when using

			Laptops		TVs	
Model			BLEU	Err.	BLEU	Err.
SCLSTM			51.16	0.79%	**52.65**	2.31%
Slug			**52.38**	1.55%	52.26	1.67%
Base Gen. (p_0)		beam	37.13	0.72%	32.63	0.72%
Aug. Gen. (p_*)	Rule Parser ($q_{\mathfrak{R}}$)	greedy	37.21	**0.13%**	32.43	0.28%
		beam	37.19	0.14%	32.59	**0.20%**

Table 5.6: BLEU and automatic attribute error on the Laptops and TVs datasets.

greedy search and 2 with beam search. Unfortunately, the augmented training is more unstable in the lexicalized setting, as it produces a large spike in food attribute errors, although the p_* models still have lower overall error than p_0.

5.3.8.2 Laptops and TVs

The results are more mixed here. Our BLEU scores are about 15 points below the baselines on the Laptops dataset and 20 points below the baselines on the TVs dataset. Upon examining the evaluation script in detail we see that BLEU score is calculated using 5 model outputs which Juraska et al. (2018) and Wen et al. (2016) do. We only produce the 1-best output at test time, perhaps explaining the difference.

Looking through our model outputs we see mostly good utterances, often nearly exactly matching the references. Our models outperform the state of the art models on errors. The best state of the art models make errors by generating sentences that do not match the input representation 0.79% and 1.67% of the time on the Laptops and TVs datasets respectively. Our p_* model reduces that error to only 0.13% and 0.20%.

5.3.9 Experiment Human Evaluation

E2E Dataset We had two undergraduate students not involved with the research look at 100 random test set utterances for six of our model variants. They were shown both the Slug output and one of our model outputs and asked to select which output was of better linguistic quality and correctness or indicate that they were equally good. We resolved disagreements in favor of the

	Correct.			Quality		
Model	>	=	<	>	=	<
delex. p_0 b	7	89	4	1	96	3
delex. $p_* \; q_{\mathfrak{R}}$ g	7	93	0	0	100	0
delex. $p_* \; q_{\mathfrak{R}}$ b	7	93	0	0	100	0
delex. $p_* \; q_{\phi}$ g	5	95	0	0	100	0
delex. $p_* \; q_{\phi}$ b	8	92	0	0	100	0
lex. $p_* \; q_{\mathfrak{R}}$ g	8	90	2	0	100	0

Table 5.7: Human correctness and quality judgments (%). Comparisons are better than (>), equal to (=), and worse than (<) the baseline Slug model. (g) and (b) indicate greedy and beam decoding respectively.

	Correct.			Quality		
Model	>	=	<	>	=	<
delex. $p_* \; q_{\mathfrak{R}}$ g	0	100	0	2	91	7

Table 5.8: Human correctness and quality judgments (%). Comparisons are better than (>), equal to (=), and worse than (<) the test set references.

baseline, i.e. if any annotator thought the baseline was better we considered it so. If an annotator marked one of our systems as better and the other marked it as equal, we considered it equal to the baseline. Inter-annotator agreement was high, with 92% agreement on correctness and 88% agreement on quality.

Table 5.7 shows the results of the evaluation. We find that the p_* model outputs are indistinguishable from the Slug model in terms of linguistic quality, regardless of the setting. In terms of correctness, the lexicalized p_* model is as good as or better than the Slug model 98% of the time. When using the delexicalized models, we don't even need beam search. The delexicalized p_* greedy decoder is as good as or better than Slug 100% of the time.

Laptops Dataset We had the same annotators look at 100 random *Inform* DAs from the Laptops test set since they are the majority DA type and we could use the same annotator guidelines from the E2E experiment. We do not have access to the Slug or SCLSTM outputs on this dataset, so we compared to one of the two test set reference sentences (picking at random) vs. the $p_*/q_{\mathfrak{R}}$ with greedy decoding. Table 5.8 shows the results. Despite the low BLEU scores, we find our outputs to be of comparable quality to references 91% of the time. Moreover, they are equally as correct as the human references 100% of the time. Annotators agreed 99% and 87% of the time on correctness and quality respectively.

5.3.10 Analysis

We hypothesize that self-training improves the correctness of outputs by sacrificing some expressiveness of the model. For example, p_* BLEU scores on the E2E dataset drop by at least 0.8 as compared to p_0 with beam search. We see a similar pattern on the TVs dataset. Self-training increases automatic metrics in the lexicalized setting, but this could be attributable to reductions in name and near realization errors, which are orthogonal to the syntactic diversity of generation.

To better quantify these effects we report the average length in words, average number of sentences, and average revised Developmental Level (D-Level) score according to the D-Level analyser (Lu, 2009) on the E2E Challenge test set outputs. The D-Level analyser automatically categorizes the syntactic complexities of an utterance into one of eight categories, with eight being the most complex, based on the revised Developmental Level scale (Rosenberg and Abbeduto, 1987; Covington et al., 2006).

Table 5.9 shows the statistics for the E2E test set outputs. In the lexicalized setting, the mean D-Level results support our hypothesis; syntactic complexity of test set outputs decreases from p_0 to p_*. In the delexicalized setting this is somewhat true; three of the p_* models have lower mean D-Level scores than p_0 with greedy decoding. Curiously, p_*/q_ϕ with beam search has the highest overall syntactic complexity of any our model variants, at odds with our hypothesis. No models are as syntactically complex as the human references, but our models come closest, with a mean D-Level category of 1.87 using the delex. p_*/q_ϕ model with beam decoder.

We also see that $p_*/q_{\mathfrak{R}}$ models are over two sentences in length on average while the human references are under two sentences, suggesting they are more often falling back to simple but reliable ways to realize attributes (e.g., appending "It is a family-friendly venue.").

One curious observation about the self-training procedure is that it leads to a convergence in

Model	Words	Sents	Mean D-Level
Human Refs.	24.06	1.76	**2.25**
Slug	24.20	1.86	1.39
lex. p_0 greedy	25.73	2.18	**1.84**
lex. p_0 beam	26.00	2.20	1.50
lex. p_* $q_{\mathfrak{R}}$ greedy	26.01	2.20	1.39
lex. p_* $q_{\mathfrak{R}}$ beam	26.04	2.17	1.45
delex. p_0 greedy	24.83	2.10	1.79
delex. p_0 beam	24.51	2.03	1.48
delex. p_* $q_{\mathfrak{R}}$ greedy	26.50	2.29	1.74
delex. p_* $q_{\mathfrak{R}}$ beam	26.46	2.28	1.74
delex. p_* q_ϕ greedy	25.33	1.76	1.77
delex. p_* q_ϕ beam	25.49	1.75	**1.87**

Table 5.9: Words/sentences per utterance and mean D-Level score of model outputs on the E2E dataset.

output complexity of greedy and beam decoding. The differences between mean D-Level score on the p_0 models is 0.34 and 0.31 in the lexicalized and delexicalized settings respectively. This shrinks to 0.0 and 0.1 in the delexicalized p_* settings and 0.06 for lexicalized p_*, suggesting that the model probability distributions are sharpening around a smaller set of output structures.

That our simple models with greedy search and no semantic control mechanisms can perform as reliably as more sophisticated models suggests that in standard training regimes we are often not fully learning from all information available in the training data. Via sampling we can uncover novel recombinations of utterances that are only implied by the provided references.

These recombinations are helpful. When we compare the PMI of various attribute-values to meaning representation size when using the original training data (i.e., the plots we showed in Figure 5.5 and Figure 5.6) against the union of original and synthetic data produced by noise-injection sampling (show in Figure 5.9), we see that most plots are much closer to 0 with the union of datasets, indicative of greater independence between length and a particular attribute-value, and that this spurious association has been lessened considerably if not removed.

Length is not the only spurious correlation present in the original training dataset that can be mitigated by the synthetic datasets. In Figure 5.10 we plot the PMI between the occurrence of any

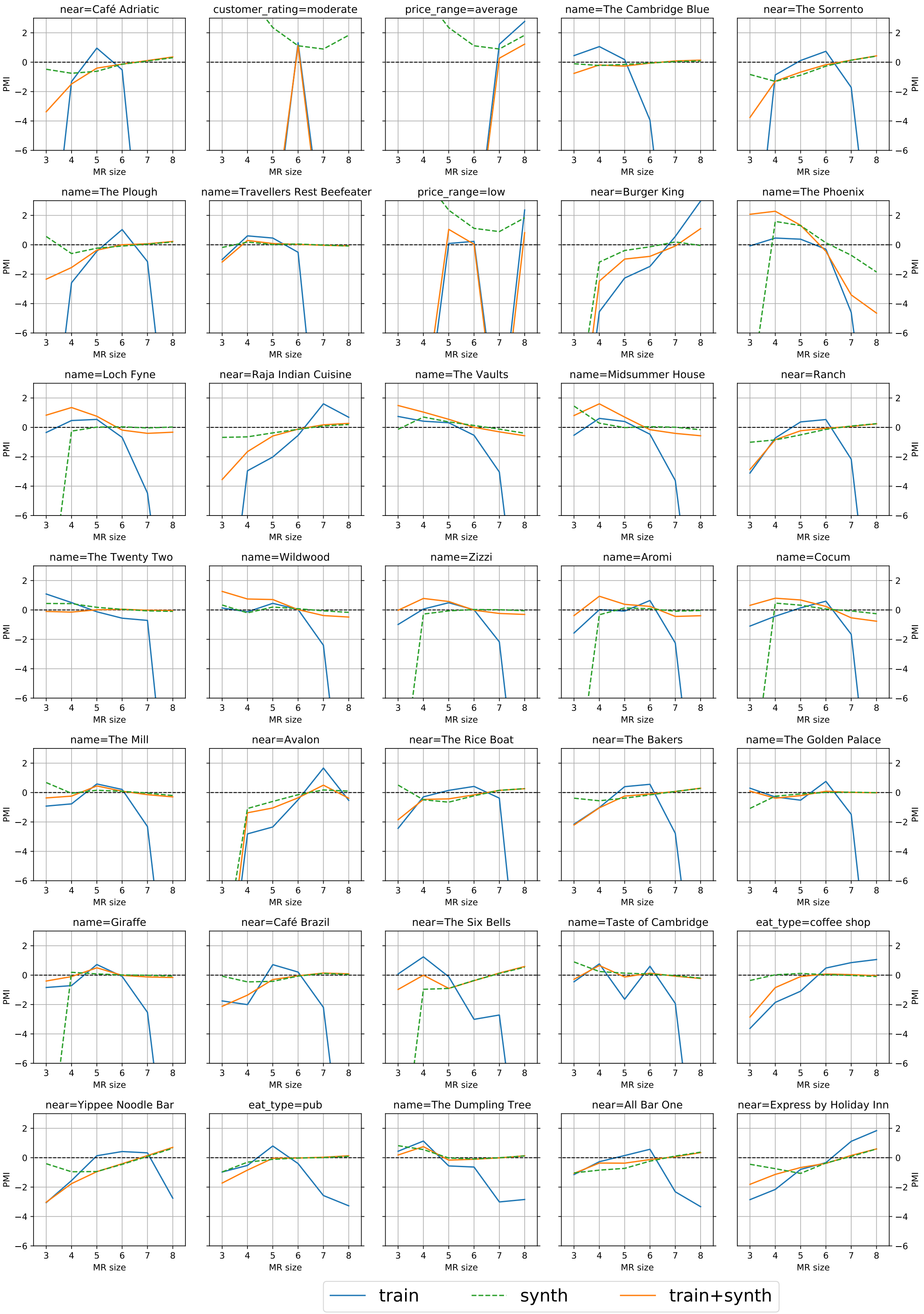


Figure 5.9: PMI between various attribute-values and meaning representation size on the E2E Challenge dataset (blue), synthetic dataset (dashed green), and their union (orange). 0 on the y-axis indicates the two variables are independent.

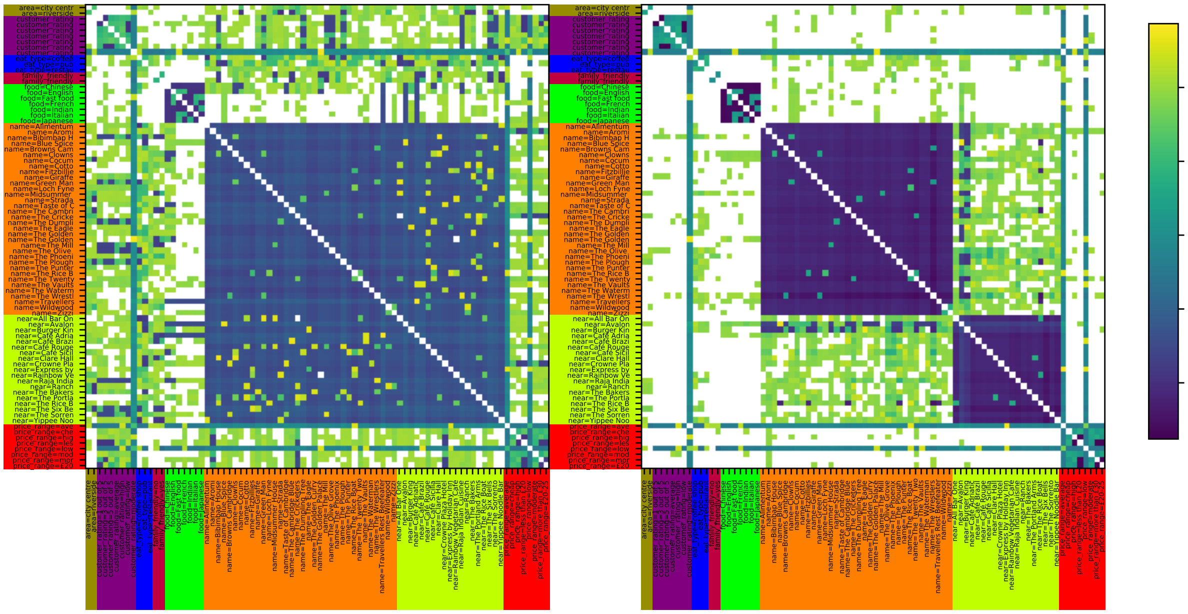


Figure 5.10: PMI between E2E Challenge attribute-values on the original training data (left) and the union of the training and synthetic data (right). PMI between $(-.25, .25)$ are colored white and suggest relative independence between the two attribute-value pairs. Color blocks on the x and y axis labels correspond to groups of values for the same attribute. E.g., orange are all the values for the name attribute.

two attribute-values, e.g. PMI(name=The Eagle,near=Burger King), on the original training data and union of original and synthetic data. Anti-correlation, i.e. extremely negative PMI values, along the diagonal are expected as attribute-values in the E2E Challenge dataset are mutually exclusive and don't usually co-occur (modulo human annotator error). We show PMI in the range of $(-.25, .25)$ as white indicating roughly no strong association. In the ideal dataset, aside from anti-correlation among values for the same attribute, we would like most the PMI values to be close to 0. When comparing the PMI from the original dataset (left) the union of original and synthetic, we see much more whitespace in the latter suggesting there are fewer spurious associations between attribute-values on the union dataset.

Producing training datasets with fewer spurious associations appears to be highly beneficial when training sequence-to-sequence models for text generation as we observe reduced semantic errors, and improved performance of greedy decoding compared to more computationally intensive inference procedures.

5.4 Alignment Training for Controllable Generation

In the previous section, we showed how to make an arbitrary sequence-to-sequence model more likely to generate semantically correct utterances using data-augmentation. While the resultant generation model is more faithful, it still lacks even coarse-grained control over the organization of the generated utterance. What's more, there's no guarantee that small changes to the input don't lead to dramatically different outputs. For example, changing a boolean attribute, e.g. changing *family_friendly=yes* to *family_friendly=no*, may lead to dramatically different syntactic structure in the output. This is because the structure or plan of the utterance is only determined implicitly by the sequence-to-sequence decoder's language model.

In this section, we show how to make a sequence-to-sequence controllable, which we achieve through a particular linearization of the input meaning representation, a linearization strategy we call *alignment training*. That is, we can specify the order in which the attribute-values of an input meaning representation are to be realized in the utterance. See Figure 5.2 for example realizations from a controllable generation model that follow three different permutations of name, eat_type, and area attributes. Through evaluation on two dialogue generation benchmarks we show that alignment training yields high levels of control in both GRU and Transformer models. This holds when models follow either a separate planning model or a human provided plan.

We also propose using a phrase-based data augmentation method to further improve the robustness of control. We further evaluate the control mechanism on randomly generated plans which are much harder to follow than human or model provided plans. We find that phrase-based data augmentation helps sequence-to-sequence models follow these more difficult plans.

5.4.1 Alignment Training Linearization

Unlike the arbitrary linearization used in §5.3.5, alignment training linearization is not solely a function of μ, but is determined by both μ and a reference utterance $\mathbf{y}$. Given a $(\mu, \mathbf{y})$ pair, the alignment training linearization finds a linearization π such that the order of the attribute-values in

$\pi(\mu)$ corresponds to the order in which they are realized in $\mathbf{y}$.

Figure 5.11 shows some examples of the alignment training linearization, including some special cases. When linearizing list-valued attributes, for instance, we treat them as distinct attribute-value pairs (Figure 5.11.a). Occasionally, we encounter repeated attribute-values in the training set, and in that case we include extra attribute-value pairs in the corresponding location in the linearization (Figure 5.11.b). We also ignore any instances of ungrounded information, as in example Figure 5.11.c where *food=Japanese* is not mentioned in the reference utterance. has no explicit representation.

In Figure 5.12 we show the steps of our procedure for obtaining the alignment training linearization, given a reference utterance $\mathbf{y}$. The first step is to tag the utterance tokens $\mathbf{y} = [y_1, \ldots, y_n]$ with a corresponding tag sequence $\mathbf{t} = [t_1, \ldots, t_n]$ where each tag t_i is equal to an attribute-value $x_j \in \mu$ or the null tag $\emptyset$. We assume that we have access to such a tagger $T : \mathcal{Y} \rightarrow \mathcal{X}$ (see §5.4.3.1 for implementation details). After producing the tag sequence $\mathbf{t}^{(1)} = T(\mathbf{y})$ (Figure 5.12b), we then group contiguous sequences of tags sharing the same tag value, discarding any null tag sequences to obtain the sequence of subsequences $\mathbf{t}^{(2)} = \left[\mathbf{t}^{(1)}_{i_1:j_1}, \ldots, \mathbf{t}^{(1)}_{i_m:j_m},\right]$ (Figure 5.12c). Finally, $\mathbf{x}$ is constructed by by prepending the dialogue act x_0 of μ to the ordered sequence of attribute-value pairs $x_1, \ldots, x_m$ implied by $\mathbf{t}^{(1)}_{i_1}, \ldots, \mathbf{t}^{(1)}_{i_m}$ (Figure 5.12d).

At test time, the generation model is only presented with a meaning representation μ and we don't have a reference utterance $\mathbf{y}$ with which to apply the alignment training linearization. In this case, we can use an utterance planning model $U : \mathcal{M} \rightarrow \mathcal{X}$ to map a meaning representation μ to a linear sequence $\mathbf{x}$. Alternatively, we can use the test set reference to obtain the alignment training linearization; this represents an unrealistically optimistic case where the model has clairvoyant known of the discourse ordering preferred by a human. In either case, we refer to a linearization $\mathbf{x}$ obtained either from U or a human reference as an utterance plan since a generation model trained with alignment training linearizations will attempt to follow it during the generation of the utterance.

(a) Example of an alignment training linearization for a meaning representation with a list-valued attribute, *genres*. Note also that the *rating* attribute for ViGGO examples is not aligned but always appended after the dialogue act (see §5.4.3.1 for details).

$$\mu = \begin{bmatrix} \text{GIVE OPINION} \\ \text{name=Little Nightmares} \\ \text{rating=good} \\ \text{genres=[} \\ \quad \text{adventure,} \\ \quad \text{platformer,} \\ \quad \text{puzzle} \\ \text{]} \\ \text{player\_perspective=side view} \end{bmatrix} \qquad \pi(\mu) = \begin{bmatrix} \textit{«s»,} \\ \textit{give\_opinion,} \\ \textit{rating=good,} \\ \textit{name=Little Nightmares,} \\ \textit{genres=adventure,} \\ \textit{player\_perspective=side view,} \\ \textit{genres=platformer,} \\ \textit{genres=puzzle,} \\ \textit{«e»} \end{bmatrix}$$

y = *Little Nightmares is a pretty cool game that has kept me entertained. It's an adventure side-scrolling platformer with some puzzle elements to give me a bit of a challenge.*

(b) Example of an alignment training linearization with repeated attribute-values. In this case, the *name* attribute is realized twice and so it appears twice in the linearization.

$$\mu = \begin{bmatrix} \text{INFORM} \\ \text{name=Aromi} \\ \text{eat\_type=coffee shop} \\ \text{customer\_rating=5 out of 5} \\ \text{food=English} \\ \text{area=city centre} \\ \text{family\_friendly=yes} \end{bmatrix} \qquad \pi(\mu) = \begin{bmatrix} \textit{«s»,} \\ \textit{inform,} \\ \textit{name=Aromi,} \\ \textit{eat\_type=coffee shop,} \\ \textit{family\_friendly=yes,} \\ \textit{food=English,} \\ \textit{name=Aromi,} \\ \textit{customer\_rating=5 out of 5,} \\ \textit{area=city centre,} \\ \textit{«e»} \end{bmatrix}$$

y = *The Aromi coffee shop is family-friendly and serves English food. Aromi has a customer rating of 5 out of 5 and is located near the center of the city.*

(c) Example alignment training linearization where an attribute-value is not grounded in the reference utterance. In this case, *food=Japanese* is not present in the linearization.

$$\mu = \begin{bmatrix} \text{INFORM} \\ \text{name=The Waterman} \\ \text{food=Japanese} \\ \text{price\_range=high} \\ \text{area=riverside} \end{bmatrix} \qquad \pi(\mu) = \begin{bmatrix} \textit{«s»,} \\ \textit{inform,} \\ \textit{area=riverside,} \\ \textit{price\_range=high,} \\ \textit{name=The Waterman,} \\ \textit{«e»} \end{bmatrix}$$

y = *Near the river there is an expensive sushi place called the Waterman.*

Figure 5.11: Example meaning representation/utterance pairs (μ, **y**) and their alignment training linearization $\pi(\mu)$.

(a) Input Utterance

$$\begin{array}{lcccccccccccc} \mathbf{y} = [& y_1, & y_2, & y_3, & y_4, & y_5, & y_6, & y_7, & y_8, & y_9, & y_{10}, & y_{11}, & y_{12}] \\ & \text{For} & \text{coffee} & \text{in} & \text{the} & \text{centre} & \text{of} & \text{the} & \text{city} & , & \text{try} & \text{Aromi} & . \end{array}$$

(b) Tagged Utterance

$$\begin{array}{lcccccccccccc} \mathbf{t}^{(1)} = [& t_1, & t_2, & t_3, & t_4, & t_5, & t_6, & t_7, & t_8, & t_9, & t_{10}, & t_{11}, & t_{12}] \\ & \text{eat\_type=coffee shop} & \text{eat\_type=coffee shop} & \text{area=city centre} & \text{area=city centre} & \text{area=city centre} & \text{area=city centre} & \text{area=city centre} & \text{area=city centre} & \emptyset & \emptyset & \text{name=Aromi} & \emptyset \end{array}$$

(c) MR Segmented Tags

$$\mathbf{t}^{(2)} = \left[\left[t_1, \quad t_2 \right]_1, \left[t_3, \quad t_4, \quad t_5, \quad t_6, \quad t_7, \quad t_8 \right]_2, \qquad \left[t_{11} \right]_3 \quad \right]$$

(d) Alignment Training Linearization

$$\begin{array}{lcccc} \mathbf{x} = [& x_0, & x_1, & x_2, & x_3] \\ & \text{inform,} & \text{eat\_type=coffee shop,} & \text{area=city centre,} & \text{name=Aromi} \end{array}$$

Figure 5.12: Example steps of the alignment training linearization algorithm for producing a linearized meaning representation **x**.

5.4.1.1 Alternative Linearization Strategies

In our experiments, we compare alignment training to three other linearization strategies, which we describe below. These linearizations, while sensible methods of mapping a meaning representation to a linear sequence of tokens, have no correspondence between the meaning representation linearization and surface realization order. Because of this, sequence-to-sequence models trained using these linearization strategies are not controllable. These linearization strategies may have some effect on the faithfulness when compared to each other and alignemnt training, so evaluation of this modelling choice has additional benefits beyond benchmarking alignment training. See Figure 5.13 for examples of the different linearization strategies on the same meaning representation/utterance pair.

Random (RND) In the RANDOM linearization (RND), we randomly order the attribute-value pairs for a given meaning representation. This strategy serves as a baseline for determining if linearization ordering matters at all for faithfulness. RND is similar to token level noise used in sequential denoising autoencoders (Wang et al., 2019) and might even improve faithfulness. During training, we resample the ordering for each example at every epoch so as not to over fit to a particular random ordering. We do not resample the validation set in order to obtain stable results from which to pick the best model.

Increasing Frequency (IF) In the INCREASING FREQUENCY linearization (IF), we order the attribute-value pairs by increasing frequency of occurrence in the training data i.e. $\text{count}(a_i = v_i) \leq \text{count}(a_{i+1} = v_{i+1})$. We hypothesize that placing frequently occurring items in a consistent location may make it easier for the generation model to realize those items correctly, possibly at the expense of rarer items.

Fixed Position (FP) We take consistency one step further and create a fixed ordering of all attributes, *n.b.* not attribute-values, ordering them in increasing frequency of occurrence on the training set (i.e. every instance has the same order of attributes in the encoder input). In this FIXED

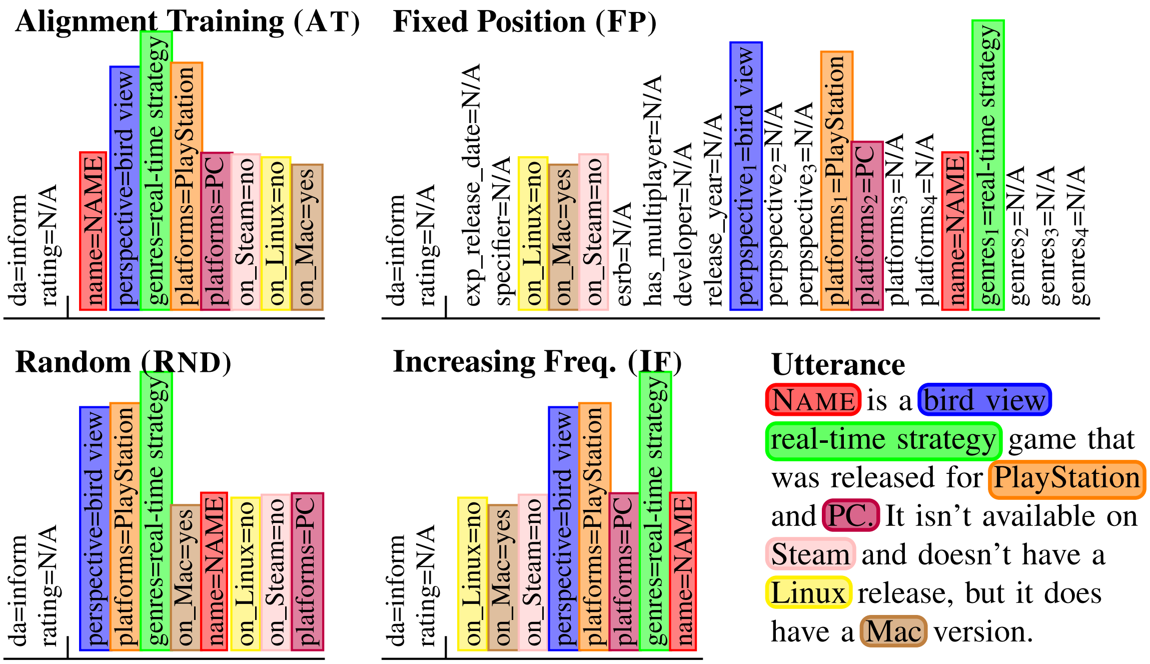
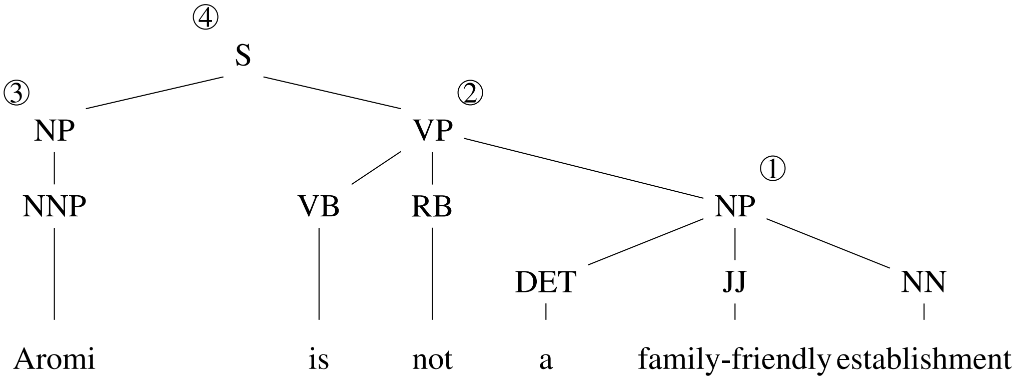


Figure 5.13: Example meaning representation linearization strategies for an utterance (lower right) from the ViGGO training set.

POSITION linearization (FP), attributes that are not present in an meaning representation are explicitly represented with an *N/A* value. For list-valued slots, we determine the maximum length list in the training data and create that many repeated slots in the input sequence. This linearization is feasible for datasets with a modest number of unique attributes (in our case ViGGO has 14 attributes and the E2E Challenge corpus has eight) but would not easily scale to 10s, 100s, or larger attribute vocabularies.

5.4.2 Phrase-based Data Augmentation

While the alignment training linearization induces control in a sequence-to-sequence model, the resulting model will still likely have trouble following utterance plans that are not well represented in the training data. As we discussed in §5.3, sequence-to-sequence models do not seem understand the compositional nature of phrase structure in language data. With this problem in mind, we propose a phrase-based data-augmentation method for creating additional training examples from constituent phrases of the training data. In doing so, we directly expose the model to



	Meaning Representation (μ)	Utterance ($\mathbf{y}$)
①	⟦ Inform family_friendly=yes ⟧	[*«s»*, *a*, *family-friendly*, *establishment*, *«e»*]
②	⟦ Inform family_friendly=no ⟧	[*«s»*, *is*, *not*, *a*, *family-friendly*, *establishment*, *«e»*]
③	⟦ Inform name=Aromi ⟧	[*«s»*, *aromi*, *«e»*]

④	⟦ Inform
	name=Aromi
	family_friendly=no ⟧
		[*«s»*, *aromi*, *is*, *not*, *a*, *family-friendly*, *establishment*, *«e»*]

Figure 5.14: Example training instances produced from the phrase-based data augmentation protocol. The constituent parse is shown above. Numbered phrase nodes correspond to the phrase examples created in the table below.

instances of syntactic composition, and how that composition systematically changes the semantics of the utterance.

We parse all training utterances and create additional training utterances from constituent phrases governed by NP, VP, ADJP, ADVP, PP, S, Sbar non-terminals.[11] Because a phrase may mean something different than the larger utterance it is embedded in, we apply the utterance tagger used for alignment training (see §5.4.3.1) to obtain the correct attribute-values denoted by the phrase. Since the tagger does not predict the dialogue act, we assign the dialogue act of the original training utterance to the new phrase's meaning representation. If a new phrase example obtained by this process does not have any attribute predicted by the tagger, we discard it.

Because we reclassify the meaning representation of phrases using the utterance tagger, the augmented data includes examples of how to negate binary attributes. See for example in Figure 5.14 where we extract the noun phrase "a family-friendly establishment" which implies *family_friendly=yes* and its composition with a verb phrase "is not", which changes the meaning to *family_friendly=no*.

When presenting the linearized meaning representationof phrase examples to the model encoder we prepend and append phrase specific start and stop tokens respectively (e.g., *«s-NP»* and *«e-NP»*) to prevent the model from ever producing an incomplete sentence when generating for a complete meaning representation.

5.4.3 Datasets

We run our alignment training experiments on the E2E Challenge dataset as well as the more recently released ViGGO corpus (Juraska et al., 2019) another English language, task-oriented dialogue dataset.[12] The ViGGO corpus comes from the video game domain (i.e., conversations with a video game recommendation agent) and contains 14 attribute types and nine dialogue acts. In addition to binary and categorical valued attributes, the corpus also features list-valued attributes which can have a variable number of values, and an open-class specifier attribute.

[11] We used the Stanford CoreNLP parser v3.9.2.

[12] `https://nlds.soe.ucsc.edu/viggo`

Dataset	Train	Augmented	Valid	Test
E2E Challenge	33,523	443,192	4,299	4,693
ViGGO	5,103	67,445	714	1,083

Table 5.10: Dataset sizes (including data augmentation) after correcting the training and validation instances.

5.4.3.1 Meaning Representation/Utterance Alignments

The original datasets do not have alignments between individual attribute-value pairs and the subsequences of the utterances they occur in, which we need for the alignment training linearization strategy. We manually developed a list of heuristic pattern matching rules (e.g., "not kid-friendly" → *family_friendly=no*) which we use to tag the utterance tokens. For ViGGO, we started from scratch, but for the E2E Challenge dataset we greatly expanded the rule-set created by Dušek et al. (2019). To ensure the correctness of the rules, we iteratively added new matching rules, ran them on the training and validation sets, and verified that they produced the same meaning representation as was provided in the dataset. This process took the author roughly two weeks to produce approximately 25,000 and 1,500 rules for the E2E and ViGGO datasets respectively. Note that the large number of rules is obtained programmatically, i.e. creating template rules and inserting matching keywords or phrases (e.g., enumerating variants such as *not kid-friendly*, *not child-friendly*, *not family-friendly*, etc.).

In cases where the matching rules produced different meaning representations than provided in the original dataset, we manually checked them. If the rule was incorrect, we added a new rule to account for the exception. In many cases in the E2E Challenge dataset and several times in the ViGGO corpus, we found the rule to be correct and the meaning representation to be incorrect for the given utterance. In those cases, we used the corrected meaning representations for training and validation. We do not modify the test sets in any way. We follow Dušek et al. (2019) and remove from the training and validation sets any modified examples that share a meaning representation also found in the test set. This creates slightly different training and validation set numbers for the E2E Challenge dataset than in the faithful generation experiments. See Table 5.10

for statistics. We use the matching rules to develop a rule-based utterance tagger to implement the alignment training linearization, phrase-based data augmentation protocol, and as a reranker when generating utterances in our experiments.

For most cases, the attribute-values uniquely correspond to a non-overlapping subsequences of the utterance. The *rating* attribute in the ViGGO dataset, however, could have multiple reasonable mappings to the utterance, so we treat it in practice like an addendum to the dialogue act, occurring directly after the dialogue act as part of a "header" section in any meaning representation linearization strategy (see Figure 5.13 where *rating=N/A* occurs after the dialogue act regardless of choice of linearization strategy).

Delexicalization The ViGGO corpus is relatively small and the attributes name, developer, release_year, expected_release_date, and specifier can have values that are only seen several times during training. Neural models often struggle to learn good representations for infrequent inputs, which can, in turn, lead to poor test-set generalization. To alleviate this, we delexicalize these values in the utterance. That is, we replace them with an attribute specific placeholder token.

Additionally, for specifier whose values come from the open class of adjectives, we represent the specified adjective with a placeholder which marks two features, whether it is consonant (C) or vowel initial (V) (e.g. "dull" vs. "old") and whether it is in regular (R) or superlative (S) form (e.g. "dull" vs. "dullest") since these features can effect the surrounding context in which the adjective is realized. See the following lexicalized/delexicalized examples:

- specifier=oldest – vowel initial, superlative
  - *What is the oldest game you've played?*
  - *What is the SPECIFIER_V_S game you've played?*
- specifier=old – vowel initial, regular
  - *What is an old game you've played?*
  - *What is an SPECIFIER_V_R game you've played?*

- specifier=new – consonant initial, regular
    - *What is a new game you've played?*
    - *What is a SPECIFIER_C_R game you've played?*

Under this delexicalization scheme, models can learn the appropriate articles (if any) to use before realizing a particular specifier value.

All generated delexicalized utterances are post-processed with the corresponding attribute-values before computing evaluation metrics (i.e., they are re-lexicalized with the appropriate value strings from the input meaning representation). Unlike in the faithful generation experiments, we do not perform any delexicalization of the E2E Challenge corpus.

5.4.4 Generation Models

We examine the effects of linearization strategy and data augmentation on biGRU (see Appendix A) and transformer (see Appendix B) based sequence-to-sequence models. See Table 5.11 for the set of hyper-parameters that we explored for each model and Table 5.12 and Table 5.13 for the winning hyper-parameter settings for the biGRU and transformer models respectively. Hyper-parameters were found using grid-search, selecting the model with best validation BLEU score. We performed a separate grid-search for each architecture-linearization strategy pairing in case there was no one best hyper-parameter setting. We used a batch size of 128 for all biGRU and Transformer models and trained for at most 700 epochs.

Additionally, we fine-tune BART Lewis et al. (2020), a large, pretrained transformer-based sequence-to-sequence model. We stop fine-tuning after validation set cross-entropy stops decreasing. We use the same settings as the fine-tuning for the CNN-DailyMail summarization task, although we modify the maximum number of updates to be roughly to be equivalent to 10 epochs on the training set when using a 500 token batch size, since the number of updates effects the learning rate scheduler. We selected the model iterate with lowest validation set cross-entropy.

While BART is unlikely to have seen any linearized MR in its pretraining data, its use of

Hyperparameter	biGRU	Transformer
Layers	1, 2	1, 2
Label Smoothing	0.0, 0.1	0.0, 0.1
Weight Decay	0, 10^{-}5	—
Optimizer/Learning Rate	Adam/10^{-3}, Adam/10^{-4}, Adam/10^{-5}, SGD/0.5, SGD/0.25, SGD/0.1	Adam with the learning rate schedule from Rush (2018) (factor=1, warmup=8000)
Tied Decoder Embeddings	tied, untied	tied, untied
Attention	Bahdanau, General	—

Table 5.11: Hyperparameter search space for biGRU and transformer architectures.

	Model	L	LS	WD	Optim.	LR	Attn	D_w	D_h	$D_{\mathcal{E}}$	$D_{\mathcal{D}}$	Drop.	Params
E2E	Rnd	2	0.1	10^{-5}	Adam	10^{-5}	Bahd.	512	512	1024	512	0.1	14,820,419
	Fp	2	0.1	10^{-5}	SGD	0.1	Bahd.	512	512	1024	512	0.1	14,820,003
	If	2	0.1	0.0	SGD	0.5	Gen.	512	512	1024	512	0.1	14,557,763
	If+p	2	0.1	0.0	SGD	0.5	Gen.	512	512	1024	512	0.1	14,557,763
	At	2	0.1	10^{-5}	Adam	10^{-5}	Bahd.	512	512	1024	512	0.1	14,820,419
	At+p	2	0.1	10^{-5}	Adam	10^{-5}	Bahd.	512	512	1024	512	0.1	14,820,419
ViGGO	Rnd	2	0.1	10^{-5}	SGD	0.25	Gen.	512	512	1024	512	0.1	14,274,865
	Fp	1	0.1	10^{-5}	Adam	10^{-5}	Bahd.	512	512	1024	512	0.1	7,718,193
	If	1	0.0	0.0	SGD	0.5	Bahd.	512	512	1024	512	0.1	7,712,049
	If+	1	0.0	0.0	SGD	0.5	Bahd.	512	512	1024	512	0.1	7,712,049
	At	2	0.1	0.0	Adam	10^{-5}	Bahd.	512	512	1024	512	0.1	14,537,521
	At+p	2	0.1	0.0	Adam	10^{-5}	Bahd.	512	512	1024	512	0.1	14,537,521

Table 5.12: Winning hyperparameter settings for biGRU models. L, LS, and WD indicate number of layers, label smoothing, and weight decay respectively. All models use untied embeddings. Drop. indicates dropout (i.e. drop probability).

	Model	Layers	LS	Emb.	Params	D_w	D_h	$D_{\mathcal{E}}$	$D_{\mathfrak{D}}$	Dropout
E2E	RND	1	0.1	tied	7,966,787	512	2048	512	512	0.1
	FP	1	0.1	tied	7,970,371	512	2048	512	512	0.1
	IF	1	0.1	untied	8,525,379	512	2048	512	512	0.1
	IF+P	1	0.1	untied	8,525,379	512	2048	512	512	0.1
	AT	2	0.1	untied	15,881,795	512	2048	512	512	0.1
	AT+P	2	0.1	untied	15,881,795	512	2048	512	512	0.1
ViGGO	RND	2	0.0	untied	15,598,897	512	2048	512	512	0.1
	FP	2	0.1	untied	15,605,041	512	2048	512	512	0.1
	IF	2	0.1	untied	15,598,897	512	2048	512	512	0.1
	IF+P	2	0.1	untied	15,598,897	512	2048	512	512	0.1
	AT	2	0.1	untied	15,598,897	512	2048	512	512	0.1
	AT+P	2	0.1	untied	15,598,897	512	2048	512	512	0.1

Table 5.13: Winning hyperparameter settings for transformer models (trained from scratch). L and LS indicate number of layers and label smoothing respectively. Drop. indicates dropout (i.e. drop probability). All models trained with the Adam optimizir with the learning rate schedule from Rush (2018) (factor=1, warmup=8000).

sub-word encoding allows it to encode arbitrary strings. Rather than extending it's encoder input vocabulary to add the MR tokens, we simply format the input MR as a string (in the corresponding linearization order), e.g. "inform rating=good name=NAME platforms=PC platforms=Xbox".

5.4.5 Utterance Planner Model

We experiment with three approaches to creating a test-time utterance plan for the alignment training models. The first is a bigram language model (BGUP) over attribute-value sequences. Attribute-value bigram counts are estimated from the training data (using Lidstone smoothing (Chen and Goodman, 1996) with $\alpha = 10^{-6}$) according to the ordering determined by the matching rules (i.e. the alignment-training ordering).

The second model is a recurrent neural network based sequence-to-sequence model, which we refer to as the neural utterance planner (NUP). We train the NUP to map IF ordered attribute-values to the alignment training ordering. We grid-search model hyperparameters, selecting the model with highest average Kendall's τ (Kendall, 1938) on the validation set alignment training orderings. See Table 5.14 for the hyperparameter search space and Table 5.15 to see the chosen

Hyperparameter	Search Space
Layers	1, 2
Learning Rate	10^{-3}, 10^{-4}, 10^{-5}
RNN Cell	GRU, LSTM
Encoder direction	uni-, bi-
Label Smoothing	0.0, 0.1

Table 5.14: Hyperparameter search space for the neural utterance planner (NUP).

Dataset	L	Enc. Dir.	RNN Cell	LR	LS	Attn.	D_w	D_h	$D_{\mathcal{E}}$	$D_{\mathcal{D}}$	Dropout
E2E	1	bi-	LSTM	10^{-5}	0.1	Bahd.	512	512	1024	512	0.1
ViGGO	1	uni-	LSTM	10^{-4}	0.1	Bahd.	512	512	1024	512	0.1

Table 5.15: Winning hyperparameter options for the neural utterance planner (NUP) model.

hyperparameter setting. We used a batch size of 128, the Adam optimizer, and trained for at most 50 epochs. Unlike the BGUP model, the NUP model also conditions on the dialogue act, so it can learn ordering preferences that differ across dialogue acts.

For both BGUP and NUP, we use beam search (with beam size 32) to generate candidate utterance plans. The beam search is constrained to only generate attribute-value pairs that are given in the supplied meaning representation, and to avoid generating repeated attributes. The search is not allowed to terminate until all attribute-values in the meaning representation are generated. Beam candidates are ranked by log likelihood. We show validation and test set Kendall's τ to the reference utterance for both planning models in Table 5.16. A Kendall's τ of 1.0 indicates that the planner exactly follows the human reference order while 0.0 indicates a random order relative to the human reference. $\tau = -1$ indicates the model produces the reverse order of the human reference plan. We see that the NUP produces utterance plans that are closer in order to the human reference on both the E2E Challenge and ViGGO datasets.

The final ordering we propose is the ORACLE ordering, i.e. the utterance plan implied by the human-authored test-set reference utterances. This plan represents the model performance if it had a priori knowledge of the reference utterance plan. When a test example has multiple references, we select the most frequent ordering in the references, breaking ties according to BGUP log-

Dataset	Model	Valid	Test
ViGGO	BGUP	0.417	0.347
	NUP	0.739	0.651
E2E	BGUP	0.433	0.432
	NUP	0.502	0.447

Table 5.16: Validation and test set Kendall's τ for BGUP and NUP models.

likelihood.

5.4.6 Experiments

5.4.6.1 Test-Set Evaluation

In our first experiment, we compare performance of the proposed models and linearization strategies on the E2E Challenge and ViGGO test sets. We refer to models using the alignment training linearization strategy as AT+BGUP, AT+NUP, or AT+ORACLE depending on whether the model is following the bigram planner, neural planner, or human reference plan respectively. For the IF and AT+NUP models we also include variants trained on the union of original training data and phrase-augmented data (see §5.4.2), which we denote +P.

Evaluation Measures For automatic quality measures, we report BLEU and ROUGE-Lscores using the official E2E Challenge evaluation script.[13] Additionally, we use the rule-based utterance tagger to automatically annotate the attribute-value spans of the model generated utterances, and then manually verify/correct them. With the attribute-value annotations in hand we compute the number of missing, wrong, or added attribute-values for each model. From these counts, we compute the semantic error rate (SER) (Dušek et al., 2020) where

$$\text{SER} = \frac{\#missing + \#wrong + \#added}{\#attributes}.$$

[13] `https://github.com/tuetschek/e2e-metrics`

On ViGGO, we do not include the rating attribute in this evaluation since we consider it part of the dialogue act. Additionally, for AT variants, we report the order accuracy (OA) as the percentage of generated utterances that correctly follow the provided utterance plan. Utterances with wrong or added attribute values are counted as not following the utterance plan.

All models are trained five times with different random seeds; we report the mean of all five runs. We report statistical significance using Welch's t-test (Welch, 1947), comparing the score distribution of the five runs from the best linearization strategy against all other strategies at the 0.05 level.

Baselines On the ViGGO dataset we compare to the transformer baseline of Juraska et al. (2019), which used a beam search of size 10 and heuristic attribute reranker (similar to our attribute-value matching rules). On the E2E Challenge dataset, we report the results of TGen+ (Dušek et al., 2019), an LSTM-based sequence-to-sequence model, which also uses beam search with a matching rule based reranker to select the most semantically correct utterance and is trained on a cleaned version of the corpus (similar to our approach).

5.4.6.2 Random Permutation Stress Test

Differences between an AT model following an utterance planner model and the human oracle are often small so we do not learn much about the limits of controllability of such models, or how they behave in extreme conditions (i.e. on an arbitrary, random utterance plan, not drawn from the training data distribution). In order to perform such an experiment we generate random utterance plans (i.e. permutations of attribute-values) and have the AT models generate utterances for them, which we evaluate with respect to SER and OA (we lack ground truth references with which to evaluate BLEU or ROUGE-L). We generate random permutations of size $3, 4, \ldots, 8$ on the E2E dataset, since there are 8 unique attributes on the E2E dataset. For ViGGO we generate permutations of size $3, 4, \ldots, 10$ (96% of the ViGGO training examples fall within this range). For each size we generated 100 random permutations and all generated plans were given the INFORM dialogue act. In addition to running the AT models on these random permutations, we also compare them to the same model after using the NUP to reorder them into an easier[14] ordering.

5.4.6.3 Human Evaluation

In our final experiment, we had human evaluators rank the 100 outputs of the size 5 random permutations for three BART models on both datasets: (i) AT+P model with NUP, (ii) AT+P model, and (iii) AT model. The first model, which uses an utterance planner, is likely to be more natural since it doesn't have to follow the random order, so it serves as a ceiling. The second and third models will try to follow the random permutation ordering, and are more likely to produce unnatural transitions between awkward sequences of attribute-values. Differences between these models will allow us to understand how the phrase-augmented data affects the fluency of the models. The annotators were asked to rank outputs by their naturalness/fluency. Each set was annotated twice by different annotators so we can compute agreement.

5.4.7 Results

AT models accurately follow utterance plans. See Table 5.17 and Table 5.18 for results on E2E Challenge and ViGGO test sets respectively. The best non-ORACLE results are show in bold for each model and results that are not different with statistical significance to the best results are underlined. We see that the AT+NUP strategy consistently receives the lowest semantic error rate and highest order accuracy, regardless of architecture or dataset, suggesting that alleviating the model's decoder of content planning is highly beneficial to avoiding errors. The Transformer AT model is able to consistently achieve virtually zero semantic error on the E2E Challenge dataset using either the bigram or neural planner model.

We also see that fine-tuned BART is able to learn to follow an utterance plan as well. When following the neural utterance planner, BART is highly competitive with the trained from scratch Transformer on the E2E Challenge dataset and surpassing it on the ViGGO dataset in terms of

[14] Easier in the sense that the NUP re-ordering is closer to the training set distribution of AT utterance plans.

semantic error rate.

Generally, the AT models had a smaller variance in test-set evaluation measures over the five random initializations as compared to the other strategies. This is reflected in some unusual equivalency classes by statistical significance. For example, on the E2E Challenge dataset biGRU models, the AT+NUP+P strategy achieves 0% semantic error and is significantly different than all other linearization strategies **except** the FP strategy even though the absolute difference in score is 6.54%. This is unusual because the AT+NUP+P strategy **is** significantly different from AT+NUP but the absolute difference is only 0.26%. This happens because the variance in test-set results is higher for FP making it harder to show significance with only five samples.

Transformer-based models are more faithful than biGRU on RND, FP, and IF linearizations. On the ViGGO dataset, BART and Transformer IF achieve 1.86% and 7.50% semantic error rate respectively, while the biGRU IF model has 19.20% semantic error rate. These trends hold for FP and RND, and on the E2E dataset as well. Because there is no sequential correspondence in the input, it is possible that the recurrence in the biGRU makes it difficult to ignore spurious input ordering effects. Additionally, we see that RND does offer some benefits of denoising; RND models have lower semantic error rate than IF models in 3 of 6 cases and FP models in 5 out of 6 cases.

Model based plans are easier to follow than human reference plans. On E2E, there is very little difference in semantic error rate when following either the bigram-based utterance planner, BGUP, or neural utterance planner, NUP. This is also true of the ViGGO BART models as well. In the small data (i.e. ViGGO) setting, biGRU and Transformer models achieve better semantic error rate when following the neural utterance planner. In most cases, neural utterance planner models have slightly higher BLEU and ROUGE-L than the bigram utterance planner, suggesting the neural planner produces utterance plans closer to the reference orderings. The neural and bigram planner models have slightly lower semantic error rate than when following the ORACLE utterance plans.

[15] Since their model does not realize specifier attributes, we do not include them in SER calculation. When including them, their model achieves 2.6% SER.

	Model	B↑	R↑	SER↓	OA↑
	TGen+ Dušek et al. (2019)	66.0	67.6	0.03	—
biGRU	RND	**66.8**	68.3	2.64	—
	FP	63.4	65.6	6.54	—
	IF	59.2	62.7	12.64	—
	IF+P	65.8	68.1	0.24	—
	AT+BGUP	66.4	68.3	0.26	98.2
	AT+NUP	66.3	68.9	0.26	98.3
	AT+NUP+P	66.5	**69.1**	**0.00**	**100.0**
	AT ORACLE	69.8	77.3	0.84	94.3
Transformer	RND	**67.4**	68.2	1.06	—
	FP	**67.4**	68.7	3.10	—
	IF	67.1	68.1	0.66	—
	IF+P	66.8	68.3	0.28	—
	AT+BGUP	66.8	68.4	**0.00**	99.9
	AT+NUP	67.0	69.0	**0.00**	**100.0**
	AT+NUP+P	66.7	**69.1**	**0.00**	**100.0**
	AT ORACLE	69.3	77.0	0.76	95.0
BART	RND	66.5	68.3	0.14	—
	FP	65.5	67.2	0.16	—
	IF	65.6	67.4	0.18	—
	IF+P	65.9	68.2	0.30	—
	AT+BGUP	66.2	68.7	0.20	98.6
	AT+NUP	**66.6**	69.2	0.20	98.6
	AT+NUP+P	66.3	**69.3**	**0.00**	**100.0**
	AT ORACLE	68.3	77.1	0.70	95.3

Table 5.17: E2E Challenge test set (B) BLEU, (R) ROUGE-L, SER, and OA. All numbers are percents.

	Model	B↑	R↑	SER↓	OA↑
	Transformer Juraska et al. (2019)	52.1	63.8	1.60[15]	—
biGRU	RND	50.2	61.6	12.56	—
	FP	50.2	61.0	17.12	—
	IF	50.2	61.3	19.20	—
	IF+P	49.5	61.6	12.46	—
	AT+BGUP	48.5	58.5	3.40	89.8
	AT+NUP	51.8	62.6	**1.58**	93.7
	AT+NUP+P	**52.4**	**62.7**	1.62	**94.3**
	AT ORACLE	54.1	65.5	2.42	92.2
Transformer	RND	52.0	62.9	9.62	—
	FP	**52.6**	63.0	8.70	—
	IF	52.3	62.6	7.50	—
	IF+P	52.3	**63.1**	4.24	—
	AT+BGUP	48.7	59.2	4.68	79.1
	AT+NUP	51.6	62.4	2.70	88.3
	AT+NUP+P	51.1	62.0	**2.28**	**89.8**
	AT ORACLE	53.2	65.0	4.08	83.0
BART	RND	43.7	55.1	1.50	—
	FP	47.0	58.9	1.68	—
	IF	43.1	54.4	1.86	—
	IF+P	**49.1**	**59.7**	1.78	—
	AT+BGUP	43.8	54.0	0.52	**98.3**
	AT+NUP	45.5	57.6	0.54	98.2
	AT+NUP+P	48.5	59.2	**0.46**	98.1
	AT ORACLE	47.1	60.4	0.82	97.2

Table 5.18: ViGGO test set (B) BLEU, (R) ROUGE-L, SER, and OA. All numbers are percents.

	E2E		ViGGo	
Model	SER↓	OA↑	SER↓	OA↑
biGRU	1.14	94.44	13.58	46.72
+P	0.54	97.34	14.46	49.26
+NUP	0.22	98.72	9.62	62.04
+NUP+P	**0.02**	**99.86**	**8.98**	**64.50**
Transformer	0.78	95.20	28.34	18.70
+P	0.40	98.10	25.72	18.10
+NUP	0.08	99.64	24.18	31.34
+NUP+P	**0.02**	**99.86**	**21.64**	**31.86**
BART	0.42	97.78	2.30	82.00
+P	0.22	98.78	1.82	87.98
+NUP	0.64	96.52	1.34	91.40
+NUP+P	**0.20**	**99.02**	**0.76**	**95.32**

Table 5.19: Random permutation stress test of At models.

This suggests that the model-based planners are producing orders more commonly seen in the training data, similar to how neural language generators frequently learn the least interesting, lowest entropy responses (Serban et al., 2016). On the other hand, when given the Oracle orderings, models achieve much higher word overlap with the reference, e.g. achieving an E2E Rouge-L ≥ 77.

Phrase-training reduces SER. We see that phrase data improves semantic error rate in 8 out of 12 cases, with the largest gains coming from the biGRU If model. Where the base semantic error rate was higher, phrase training has a more noticeable effect. After phrase training, all E2E models are operating at near zero semantic error rate and almost perfectly following the neural utterance planner. Model performance on ViGGO is more varied, with phrase training slighting hurting the biGRU At+NUP model, but otherwise helping performance.

Random Permutation Stress Test Results of the random permutation experiment are shown in Table 5.19. Overall, all models have an easier time following the neural utterance planner's reordering of the random permutations. Phrase training also generally improved semantic error

	Model	1	2	3	Avg.
E2E	AT+NUP+P	61.5	16.5	22.0	**1.61**
	AT+P	30.0	44.0	26.0	1.96
	AT	25.0	49.5	25.5	2.01
ViGGO	AT+NUP+P	57.5	27.5	15.0	**1.58**
	AT+P	10.0	29.5	60.5	2.51
	AT	43.0	46.0	11.0	1.68

Table 5.20: Human Evaluation results. Table shows the percent of times each model was ranked 1 (best), 2, 3 (worst) in terms of naturalness and average rank.

rate. All models perform quite well on the E2E permutations. With phrase-training, all E2E models achieve less than 0.6% semantic error rate following random utterance plans. Starker differences emerge on the ViGGO dataset. The biGRU+NUP+P model achieves a 8.98% semantic error rate and only correctly follows the given order 64.5% of the time, which is a large decrease in performance compared to the ViGGO test set.

Human Evaluation Results of the human evaluation are shown in Table 5.20. We show the number of times each system was ranked 1 (most natural), 2, or 3 (least natural) and the average rank overall. Overall, we see that BART with the neural utterance planner and phrase-augmentation training is preferred on both datasets, suggesting that the utterance planner is producing natural orderings of the attribute-values, and the model can generate reasonable output for it. On the E2E dataset, we also see small differences in between the AT+P and AT models suggesting that when following an arbitrary ordering, the phrase-augmented model is about as natural as the non-phrase trained model. This is encouraging as the phrase trained model has lower semantic error rates. On the ViGGO dataset we do find that the phrase trained model is less natural, suggesting that in the small data setting, phrase-training may hurt fluency when trying to follow a difficult utterance plan.

For agreement we compute average Kendall's τ between each pair of annotators for each dataset. On E2E, we have $\tau = .853$ and ViGGO we have $\tau = .932$ suggesting very strong agreement.

5.4.8 Discussion

One consistently worrying sign throughout the first two experiments is that the automatic metrics are not good indicators of semantic correctness. For example the ROUGE-L score of the E2E AT ORACLE models is about 8 points higher than the AT+NUP models, but the AT+NUP models make fewer semantic errors. Other similar examples can be found where the automatic metric would suggest picking the more error prone model over another. As generating fluent text becomes less of a difficult a problem, these shallow ngram overlap methods will cease to suffice as distinguishing criteria.

The second experiments also reveal limitations in the controllable model's ability to follow arbitrary orderings. The biGRU and Transformer models in the small-data ViGGO setting are not able to generalize effectively on non-training distribution utterance plans. BART performance is much better here, but is still hovering around 2% semantic error rate and only roughly 88% of outputs conform to the intended utterance plan. Thankfully, if an exact ordering is not required, using the neural utterance planner to propose an order leads to more semantically correct outputs.

5.4.9 Limitations

While we are able to achieve very low test-set SER for both corpora, we should caution that this required extensive manual development of matching rules to produce meaning representation/utterance alignments, which in turn resulted in significant cleaning of the training datasets. We chose to do this over pursuing a model based strategy of aligning utterance subspans to attribute-values because we wanted to better understand how systematically S2S models can represent arbitrary order permutations independent of alignment model error. Also we should note that data cleaning can yield more substantial decreases in semantic errors (Dušek et al., 2019; Wang, 2019) and is an important consideration in any practical neural NLG.

5.5 Conclusion

In this chapter we focused on two problems in natural language generation from a meaning representation using sequence-to-sequence models: faithful generation and controllable generation. For the former we proposed a data-augmentation protocol called noise-injection sampling and self-training, which enabled us to use an unfaithful sequence-to-sequence based language generation model and a meaning representation parser to generate fluent but semantically divergent synthetic training instances, which when added to the original training data, improved the faithfulness of subsequent models.

For the former problem of controllable generation, we showed that alignment of the input meaning representation to the reference utterance realization order yields high degrees of control in several popular sequence-to-sequence model variates. Additionally, we also see that data-augmentation is useful in making the model more robust when following difficult utterance plans.

In future work, we hope to focus more on changes to the models themselves to make them more explicitly aware of the compositional nature of the language they are modeling and how that can effect faithfulness and control. We are currently achieving this through data-augmentation. However, we worry that data-augmentation will be difficult to scale to more complex meaning representations, and that it will be difficult to adequately represent more combinatorially complex semantic formalisms explicitly.

Chapter 6: Conclusion

In this thesis we presented a variety of contributions to two open problems in automatic text summarization, salience estimation for content selection and reliable text generation from formalized representations of content with neural NLG models. In the first two chapters we studied salience estimation from two perspectives, single document summarization and query focused stream summarization. In the first case, we evaluated several deep learning architectures for estimating sentence salience. In addition to proposing a hierarchical modeling framework (i.e., as word embedding, sentence encoder, and sentence extraction layers) which included prior summarization models as well as some novel ones, we were also interested in a thorough exploration of model behavior on the sentence salience estimation task.

Our evaluation considered both aspects of the model architecture as well as ablations of the dataset. We find that the models studied, regardless of architecture, are likely exploiting artifacts of the dataset or other shallow heuristics to make predictions rather than any deep content understanding. We summarize our evidence for this briefly. First at the level of sentence encoder, we find that differences between different sentence encoder architectures are small and that the averaging encoder is frequently the best performing encoder when keeping the extractor fixed. That averaging works well implies that the mere presence of a word is more important than its long range context (detectable by the recurrent encoder) or its local context (detectable by the convolutional encoder) and for the summarization models studied, it is sufficient to represent a sentence as a bag of embeddings.

At the sentence extractor level we find limited evidence that the models are able to make use of document level context or prior actions effectively. We compared two extractors that made sequential predictions of sentence salience where previous extractive decisions were fed as inputs back into the model to make subsequent predictions, i.e., two autoregressive extractors, against two extractors that made independent predictions, i.e., two non-autoregressive models. What we saw was that previous extraction decisions were not decisive in ROUGE performance, with both autoregressive and non-autoregressive models achieving similar ROUGE scores. This suggests that representing what has been previously selected for the summary does not affect subsequent decisions much, and that the models struggle to exploit long range context effectively.

Finally, when we ablate different word types, we see very small differences in ROUGE scores on news data, suggesting that the model is not overly dependent on entities or events (i.e., nouns or verbs) being available in the sentence to make predictions. When removing position as a viable feature by shuffling sentence order, we finally see large decreases in ROUGE. At least for news and to slightly less degree medical documents, we see that position is implicitly detectable by the models and that it is driving much of the ROUGE performance.

On datasets where position is less reliable, we see that the order shuffling experiments have less of an effect. On the Reddit personal narratives, there is no significant difference between the shuffled and non-shuffled model and on the AMI work place meeting corpus, we see that performance actually improves significantly. On these datasets, removing different content types yields slightly larger differences in performance, also giving credence to the idea that these models are focusing more content.

In our second chapter, which introduces a more complex news summarization task, we see that modeling content as well as prior extraction decisions becomes more important as position becomes unreliable, especially if you can exploit event specific prior knowledge. Our SAP summarization model, which estimates salience using a mix of general purpose and domain specific features improves over clustering or predicting salience alone. Through feature ablation we see that newswire and domain language model scores are the most important feature group for predicting salience. The basic features which make use of sentence position are least important.

We find in both our own evaluation as well as the TREC Temporal Summarization official evaluation that the SAP model leads to higher expected gain (i.e., precision) than a similar summarization system with only the clustering component. While the SAP model does have less comprehensiveness than a clustering only approach, in the harmonic mean of the two metrics it achieves higher results. Additionally, when taking latency into account, we find that the salience estimation model helps the clustering component quickly identify the most important sentences.

While the SAP model integrates a salience estimate component into a clustering algorithm, it does not optimize the update selection stage with the overall quality of the summary in mind. Effectively, each round of update selections are independent of each other. Our second model proposed for the stream summarization task embeds the salience estimation component into a greedy sentence extraction policy, which learns from exploration and takes into account previous actions and other dynamic features of the summarization system, while optimizing an overall measure of summary quality. We again find in both TREC Temporal Summarization evaluations and our own independent evaluations that the L2S summarizer is able to identify salience content more quickly than other models while maintaining sufficient recall to be competitive with other models. Being able to identify information in a timely manner is a key aspect of stream summarization as information becomes less useful to users as time goes on (Yom-Tov and Diaz, 2011).

We next turn our attention to text generation. Rather than pursue an end-to-end neural strategy, we instead focus on NLG from discrete meaning representations, which we consider to be an idealized form of the content selection stage of an extractive summarization system. Having an explicit representation of semantics allows us to more easily study issues of faithfulness and control in neural NLG models. To that end, we show that naïve sequence-to-sequence models produce fluent but semantically incorrect utterances and that this is possibly due to spurious correlations and artifacts of the training data. We then propose a noise-injection and self-training scheme for data augmentation. Using the initial naïve sequence-to-sequence model (and a meaning representation parser) we create synthetic training examples which do not possess these artifacts, or do so to a lesser degree. A subsequent sequence-to-sequence model trained on the union of the original and synthetic data produces a more faithful NLG model. Additionally, we observe that greedy decoding converges on beam decoding in terms of semantic correctness suggesting the resulting model would be more efficient to run in practice.

We also introduce the alignment training procedure for making a neural NLG model controllable at the level of shallow phrase/discourse entity ordering. We find controllable neural NLG models are able to follow discourse plans produced by discourse ordering models, and to a slightly lesser degree, human discourse ordering plans. Additionally, we find that BART, a large, pretrained sequence-to-sequence language model, when fine-tuned with alignment training, produces highly controllable NLG models on both large and small data settings. We also find that alignment training has positive benefits in terms of model faithfulness with alignment trained models more frequently producing fewest semantic errors.

We further stress test our models on difficult, random permutation plans to test how models follow a truly arbitrary plan absent of English language ordering biases. We find that there are some gaps in our models' ability to represent and follow arbitrary plans, especially for trained from scratch models in the small data condition. BART performs relatively well in either large or small data condition but does see some performance decrease in semantic correctness and order accuracy. We also find that phrase-based data augmentation can help reduce the semantic error rate in this more adversarial setting.

6.1 Limitations and Open Problems for Abstractive Summarization Beyond End-to-End Neural Models

One glaring omission from this thesis is that we never develop a fully abstractive summarization system. That is, we have not presented a bipartite model such that the first component takes as input a set of text units and selects the contents for a summary, and the second component then generates the summary from the selected contents. We have consciously avoided bridging this gap as it is a difficult one. Instead we have focused on the content selection and faithful generation problems in isolation, which are areas where we could make immediate contributions. We prefer pursuing these sub-problems directly where errors can be isolated and the effects of interventions accurately measured.

We do believe that pursuing a summarization system that works by first mapping input text units to a logical meaning representation, and then uses a separate NLG model to convert that meaning representation to a natural language summary is a worthwhile goal. What's more, all of the components exist to build this system with the caveat that they currently have relatively high error rates. We now describe some of these components and their current limitations, some of which we highlight again in future works as promising avenues to explore.

Semantic Parsing We will need a reliable way of mapping text units to a semantic representation, like abstract meaning representation (AMR) (Banarescu et al., 2013), discourse representation theory (DRT) (Basile et al., 2012), or concept maps (Yang et al., 2020). Work on semantic parsing is an active area which continues to improve, but the current state-of-the-art of such systems is still currently lacking. At the current levels of accuracy, errors in semantic parsing would dominate any downstream results (Zhang et al., 2019b).

Summaries of Meaning Representations Even with high accuracy semantic parsing, we would still need a method of mapping an input meaning representation to a summary meaning representation. While there is some initial work on this task (Falke et al., 2017; Liao et al., 2018; Falke and Gurevych, 2019), it is far from a solved problem. Existing collections of summary meaning representations are relatively small, and it remains difficult to plausibly use such collections to build models for generic news summarization let alone other domains.

Faithful Generation of Complex Meaning Representations There has been a growing number of works that focus on generating text from more complicated structures than our relatively simple meaning representations used in this thesis. In particular, generating from AMR (Castro Ferreira et al., 2017; Wang et al., 2020b) or tree structures (Balakrishnan et al., 2019), have shown increased progress, in keeping with our suggestion at the start of the thesis that generating text conditioned on arbitrary inputs has gotten easier with neural models. There has even been work looking generating summaries from their AMR graphs with neural models (Hardy and Vlachos, 2018).

However, with increased complexity of the meaning representation, it becomes difficult to evaluate the faithfulness of generated outputs. In the datasets studied in this thesis, most of the realizations of any individual attribute-values were relatively independent of other attribute-values. Presumably, accurate meaning representations of real summaries would be much more complex. Validating the faithfulness of complex news events or rhetorical arguments might require representing contingency, uncertainty, or counter-factual conditions, as well as the plausibility of arguments. To solve these issues we might benefit from borrowing ideas from argumentation mining (e.g., Chakrabarty et al. (2019)) to better understand how multiple pieces of evidence are used to bolster claims and make larger points.

6.2 Why not an end-to-end neural abstractive summarization model?

While unusual for summarization research in 2020, we have not advocated for an end-to-end neural abstractive summarization model. We argue that learning an end-to-end summarization model (while possibly delivering flashy and impressive short term results) is unlikely to result in summarization reliable enough for use outside of the NLP research community.

First, it has been shown repeatedly, that while large neural language models might deliver impressive test set results they often exploit shallow heuristics and frequently fail to generalize in ways that suggest they are learning hidden syntactic or semantic representations that are cognitively plausible (Fodor and Pylyshyn, 1988; Marcus, 2003; Lake and Baroni, 2018; McCoy et al., 2019; Linzen, 2020).

Second, the summarization task as often studied in NLP (i.e., learning to map input texts to summary text from a large parallel corpus) is underspecified. Real summaries are grounded in an extrinsic task and serve a particular human informational need or goal (Spärck Jones, 1999; McKeown et al., 2005). Without explicitly specifying these goals both as model inputs and in evaluation, you can’t say whether a summarization system that exploits position bias over a deeper semantic understanding of the text is good or bad. For example, if the goal is, “help me get the gist of the morning newspaper faster,” then perhaps showing the user the lead paragraphs of each article in the paper is actually a quite good summary. If the intended goal is “a summary of my doctor's appointment with an emphasis on the evidence for my having contracted COVID-19," then it is unlikely that the semantic understanding necessary to complete that task can be learned simply from amassing a collection of parallel input/summary texts (Bender and Koller, 2020; Bisk et al., 2020).

While automatic methods of evaluating summary faithfulness are currently being explored through question-answering (Wang et al., 2020a; Durmus et al., 2020) or estimation of ROUGE or BLEU scores (Zhang et al., 2020; Sellam et al., 2020), they suffer from the same drawbacks just mentioned. Underlying them is same paradigm of training large neural models on parallel corpora, which because of the two points mentioned will often fail to generalize correctly on the long-tail input instances. Additionally, these methods serve as post-hoc correctives, and do not help to attribute errors in an abstractive generation models outputs to its decoder or encoder representations.

6.3 Future Work

Given our discussion of limitations, we believe the following research directions would yield meaningful future contributions to the automatic summarization literature.

Semantic Representations for Summarization Our work assumes that research on broad coverage semantic parsing will improve to a point such that high accuracy logical representations will be automatically producible for arbitrary newswire text. However, a variety of open problems remain around appropriate meaning representations for the summarization problem. The formalisms for semantics in NLP have realistically only been annotated at the level of sentences and aggregating these representations into document or multi-document level representations is challenging, even for humans (O'Gorman et al., 2018).

When performing content selection, we could perform salience estimation and selection directly from the text (as we currently do in this thesis) but then pass on the automatically parsed meaning representations of the extracted text units to the NLG component. A more interesting alternative is that salience estimation and content selection could be done in the meaning representation space; a small but growing subset of the summarization literature has explored this variant of the content selection problem (Falke et al., 2017; Liao et al., 2018; Falke and Gurevych, 2019). These works largely frame summarization from a semantic or concept graph as one of graph compression, selectively deleting unimportant nodes from the graph representation of the input. However, compression through deletion is only one of many abstraction methods employed by human summarizers (Jing and McKeown, 2000). Expansion of existing semantic formalisms with a catalogue of representation transformations that capture not only deletion but also aggregation, metonymy, synecdoche, generalization, or other forms of abstraction would be of great benefit to the development of summarization from semantic graphs/meaning representations.

Psychological/Motivational Factors in Summarization We showed that existing models struggle to find all the salient information in the input, especially when the document structure or position cues are relatively weak. Arguably, simply learning a salient content model from a large collection of document-summary pairs (a practice that is presently de rigueur for end-to-end neural summarization models) is likely not sufficient since the goal of the reader or the intended use of the summary remains underspecified in most contemporary research.

For instance, a clinician's notes are not simply a truncated list of the k most frequently mentioned symptoms by the patient. Instead, they are a summary of both the dialogue between the doctor and patient, but focused by doctor's deductive procedure for determining the specific malady afflicting the patient. That is, during the appointment the doctor maintains a set of possible hypothesis diagnoses, and gradually rules out some while increasing confidence in others through their questioning of the patient (Pivovarov and Elhadad, 2015). Having a clinical notes summarization model with an explicit representation of this deductive procedure would possibly improve the ability to contextually identify salient questions and answers. One could imagine similar scenarios in other domains. In financial news summarization, for example, the salience of particular events depends heavily on an individual's particular investment position.

In any case, we expect that incorporating an explicit model of the summary reader's beliefs and world knowledge into the salience estimation component of a summarization system seems likely to improve the accuracy of content selection. Additionally, it may also improve model scrutability. We believe work on concept modeling (Bosselut et al., 2019) might be a fruitful way to explore this idea in summarization.

Better Inductive Biases in Generation Models Our generation models improved when we could reliably create synthetic examples through data augmentation. In practice, data augmentation is difficult to get right as any errors (e.g., disfluencies or bad meaning representation/utterance pairings) risk hurting the model more than they benefit. Additionally, they have the flavor of a brute force solution, where we attempt to systematically generate all gaps in the training data. While this can work for simple datasets it is unlikely to scale to generation from more complex combinatorial objects (i.e., meaning representations with recursive structure like graphs and trees).

In future work, we would like to explore ways of building in better inductive biases into the model. In particular we would like to adapt some forms of contrastive learning popular in the vision community (Chen et al., 2020) to the NLG setting. Under the contrastive learning framework, constraints are put on the model representation via auxiliary discrimination tasks. In the image domain, this might mean imposing a constraint that semantics preserving transformations (e.g., rotation) of the same image have similar hidden representations under the model, with the intuition being that an image classifier should produce the same classification probabilities even if the image is arbitrarily rotated.

In NLP, these kinds of semantics preserving transformations are somewhat harder to obtain, but we can begin to imagine some feasible ones that might work for the NLG problem as we have posed it. In our controllable generation model, the encoder input sequence is actually serving double duty as both a representation of the meaning representation but also the utterance plan. That is,

$$\pi_1(\mu) = [\mathit{inform}, \mathit{name}{=}\mathit{Aromi}, \mathit{eat\_type}{=}\mathit{coffee\ shop}, \mathit{area}{=}\mathit{city\ centre}]$$

and

$$\pi_2(\mu) = [\mathit{inform}, \mathit{eat\_type{=}coffee\ shop}, \mathit{area{=}city\ centre}, \mathit{name{=}Aromi}]$$

both correspond to two different utterances plans for the same meaning representation,

$$\mu = \left[\!\!\left[\begin{array}{l}\textsc{Inform}\\ \text{name=Aromi}\\ \text{area=city centre}\\ \text{eat\_type=coffee shop}\end{array}\right]\!\!\right].$$

As we currently have it, the encoders of our neural NLG models would produce different representations, i.e., $\text{enc}\,(\pi_1(\mu)) \neq \text{enc}\,(\pi_2(\mu))$. In the spirit of contrastive learning, however, it might be useful to differentiate the part of the hidden state that corresponds to the propositional contents of the meaning representation and the part that corresponds to the plan. We might specify that the encoder produce two hidden states, one representing the meaning representation and the other representing the plan: $\text{enc}\,(\pi_1(\mu)) = \left(\mathbf{h}_\mu, \mathbf{h}_\pi\right)$. For two different plans for the same meaning representation,

$$\begin{aligned}\text{enc}\,(\pi_1(\mu)) &= \left(\mathbf{h}_\mu^{(1)}, \mathbf{h}_\pi^{(1)}\right)\\ \text{enc}\,(\pi_2(\mu)) &= \left(\mathbf{h}_\mu^{(2)}, \mathbf{h}_\pi^{(2)}\right)\end{aligned}$$

we would enforce the constraint during training that

$$\mathbf{h}_\mu^{(1)} = \mathbf{h}_\mu^{(2)} \quad \text{and} \quad \mathbf{h}_\pi^{(1)} \neq \mathbf{h}_\pi^{(2)}.$$

For another meaning representation that has the same attributes, but different values,

$$\mu' = \left[\!\!\left[\begin{array}{l}\textsc{Inform}\\ \text{name=Bar Central}\\ \text{area=riverside}\\ \text{eat\_type=pub}\end{array}\right]\!\!\right] \quad \text{with} \quad \text{enc}\,(\pi_1(\mu')) = \left(\mathbf{h}_\mu^{(3)}, \mathbf{h}_\pi^{(3)}\right),$$

we would have

$$\mathbf{h}_{\mu}^{(3)} \neq \mathbf{h}_{\mu}^{(1)} = \mathbf{h}_{\mu}^{(2)} \quad \text{and} \quad \mathbf{h}_{\pi}^{(3)} = \mathbf{h}_{\pi}^{(1)} \neq \mathbf{h}_{\pi}^{(2)}.$$

The intuition here is that by creating distinct representations of the content and the plan, the encoder can amortize learning how to represent specific attribute orderings independent of the values of the attributes, while the values themselves can reside in $\mathbf{h}_{\mu}$.

Additionally, it would useful to explore how contrastive learning and other semantics preserving transformations can be represented in the encoding of more complex meaning representations. Work on generating text from trees (Balakrishnan et al., 2019) or arbitrary graph structures like AMR (Wang et al., 2020b) might benefit from imposing such constraints on the encoder representation.

6.4 Final Remarks

We have studied in this thesis several problems related to text-to-text generation generally, and the summarization problem specifically. Our hope is that by breaking down the summarization into sub-tasks we have revealed important but tractable problems on the way to more reliable text-to-text generation and summarization.

We also hope that we have given some useful experimental setups for revealing how deep learning models perform content selection. We think this level of investigation should be carried out whenever deep learning based summarization models are proposed. In general, we think that knowing how a model produces an answer to a question is just as important as the model getting the correct answer. For instance, if we know that the model is exploiting position heuristics to identify important information, we should not expect it to perform well when those signals are not present.

On the generation side, we hope we have emphasized the importance of semantic correctness and control when considering a neural NLG model. While ngram overlap based metrics ROUGE and BLEU were relatively reliable metrics of summary quality when summarization models were primarily extractive, with the move to powerful neural NLG models that can generate fluent and natural text, these kinds of metrics become less discriminating. We think over time they should be de-emphasized in favor of some kind of semantic evaluation, preferably a manual one. We are still not in a world where neural NLG models can be be used reliably in industrial text generation settings. We hope that work continues in the area of faithfulness and control so that this situation changes.

Code and data for many of the experiments presented in this thesis can be found at `http://cs.columbia.edu/~kedzie/`. Additionally, conference papers corresponding to individual chapter contents can also be found there. Questions with regard to this thesis or any of the linked materials should be directed to the author.

Bibliography

James Allan, Rahul Gupta, and Vikas Khandelwal. 2001. Temporal Summaries of News Topics. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, page 10–18, New York, NY, USA. Association for Computing Machinery.

Automatic Language Processing Advisory Committee (ALPAC). 1966. *Language and Machines: Computers in Translation and Linguistics: a Report*, volume 1416. National Academy of Sciences, National Research Council.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Jacob Andreas. 2020. Good-Enough Compositional Data Augmentation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7556–7566, Online. Association for Computational Linguistics.

Douglas E. Appelt. 1982. Planning Natural-Language Utterances. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*, AAAI'82, page 59–62. AAAI Press.

Mira Ariel. 2001. Accessibility theory: An overview. In *Text Representation: Linguistic and psycholinguistic aspects*, volume 8, pages 29–87. John Benjamins.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A Simple but Tough-to-Beat Baseline for Sentence Embeddings. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Javed A. Aslam, Fernando Diaz, Matthew Ekstrand-Abueg, Richard McCreadie, Virgil Pavlu, and Tetsuya Sakai. 2015. TREC 2015 Temporal Summarization Track Overview. In *Proceedings of The Twenty-Fourth Text REtrieval Conference, TREC 2015, Gaithersburg, Maryland, USA, November 17-20, 2015*, volume 500-319 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Javed A. Aslam, Matthew Ekstrand-Abueg, Virgil Pavlu, Fernando Diaz, Richard McCreadie, and Tetsuya Sakai. 2014. TREC 2014 Temporal Summarization Track Overview. In *Proceedings of The Twenty-Third Text REtrieval Conference, TREC 2014, Gaithersburg, Maryland, USA, November 19-21, 2014*, volume 500-308 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Javed A. Aslam, Matthew Ekstrand-Abueg, Virgil Pavlu, Fernando Diaz, and Tetsuya Sakai. 2013. TREC 2013 Temporal Summarization. In *Proceedings of The Twenty-Second Text REtrieval*

Conference, TREC 2013, Gaithersburg, Maryland, USA, November 19-22, 2013, volume 500-302 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. 2013. Joint Language and Translation Modeling with Recurrent Neural Networks. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1044–1054, Seattle, Washington, USA. Association for Computational Linguistics.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR*, abs/1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Anusha Balakrishnan, Jinfeng Rao, Kartikeya Upasani, Michael White, and Rajen Subba. 2019. Constrained Decoding for Neural NLG from Compositional Representations in Task-Oriented Dialogue. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 831–844, Florence, Italy. Association for Computational Linguistics.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, Sofia, Bulgaria. Association for Computational Linguistics.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Regina Barzilay and Kathleen R. McKeown. 2005. Sentence Fusion for Multidocument News Summarization. *Computational Linguistics*, 31(3):297–328.

Valerio Basile, Johan Bos, Kilian Evang, and Noortje Venhuizen. 2012. Developing a large semantically annotated corpus. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3196–3200, Istanbul, Turkey. European Language Resources Association (ELRA).

Jasmijn Bastings, Marco Baroni, Jason Weston, Kyunghyun Cho, and Douwe Kiela. 2018. Jump to better conclusions: SCAN both left and right. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 47–55, Brussels, Belgium. Association for Computational Linguistics.

Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online. Association for Computational Linguistics.

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly Learning to Extract and Compress. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 481–490, Portland, Oregon, USA. Association for Computational Linguistics.

Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph Turian. 2020. Experience Grounds Language. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8718–8735, Online. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 Conference on Machine Translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. 2016. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4356–4364, Red Hook, NY, USA. Curran Associates Inc.

Anthony Bonner. 2007. *The Art and Logic of Ramon Llull: A User's Guide*. Studien und Texte zur Geistesgeschichte des Mittelalters. Brill.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. 2019. COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4762–4779, Florence, Italy. Association for Computational Linguistics.

Ronald Brandow, Karl Mitze, and Lisa F. Rau. 1995. Automatic Condensation of Electronic Publications by Sentence Selection. *Information Processesing & Management*, 31(5):675–685.

P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, R. Mercer, and P. Roossin. 1988. A Statistical Approach to Language Translation. In *Coling Budapest 1988 Volume 1: International Conference on Computational Linguistics*.

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz

Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners.

Jean Carletta, Simone Ashby, Sebastien Bourban, Mike Flynn, Mael Guillemot, Thomas Hain, Jaroslav Kadlec, Vasilis Karaiskos, Wessel Kraaij, Melissa Kronenthal, Guillaume Lathoud, Mike Lincoln, Agnes Lisowska, Iain McCowan, Wilfried Post, Dennis Reidsma, and Pierre Wellner. 2005. The AMI Meeting Corpus: A Pre-Announcement. In *Proceedings of the Second International Conference on Machine Learning for Multimodal Interaction*, MLMI'05, page 28–39, Berlin, Heidelberg. Springer-Verlag.

Thiago Castro Ferreira, Iacer Calixto, Sander Wubben, and Emiel Krahmer. 2017. Linguistic realisation as machine translation: Comparing different MT models for AMR-to-text generation. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 1–10, Santiago de Compostela, Spain. Association for Computational Linguistics.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 552–562, Hong Kong, China. Association for Computational Linguistics.

Tuhin Chakrabarty, Christopher Hidey, Smaranda Muresan, Kathy McKeown, and Alyssa Hwang. 2019. AMPERSAND: Argument Mining for PERSuAsive oNline Discussions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2933–2943, Hong Kong, China. Association for Computational Linguistics.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. 2015. Learning to Search Better than Your Teacher. volume 37 of *Proceedings of Machine Learning Research*, pages 2058–2066, Lille, France. PMLR.

Stanley F. Chen and Joshua Goodman. 1996. An Empirical Study of Smoothing Techniques for Language Modeling. In *34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318, Santa Cruz, California, USA. Association for Computational Linguistics.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607, Virtual. PMLR.

Jianpeng Cheng and Mirella Lapata. 2016. Neural Summarization by Extracting Sentences and Words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 484–494, Berlin, Germany. Association for Computational Linguistics.

Kyunghyun Cho. 2016. Noisy Parallel Approximate Decoding for Conditional Recurrent Language model. *CoRR*, abs/1605.03835.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT Press.

Kenneth Ward Church and Patrick Hanks. 1990. Word Association Norms, Mutual Information, and Lexicography. *Computational Linguistics*, 16(1):22–29.

John M. Conroy and Dianne P. O'Leary. 2001. Text Summarization via Hidden Markov Models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, page 406–407, New York, NY, USA. Association for Computing Machinery.

Michael A Covington, Congzhou He, Cati Brown, Lorina Naçi, and John Brown. 2006. How Complex Is That Sentence? A Proposed Revision of the Rosenberg and Abbeduto D-Level Scale.

Gianfranco Crupi. 2019. Volvelles of knowledge. Origin and development of an instrument of scientific imagination (13th-17th centuries). *JLIS.it*, 10(2):1–27.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-Based Structured Prediction. *Mach. Learn.*, 75(3):297–325.

Hal Daumé III and Daniel Marcu. 2005. Learning as Search Optimization: Approximate Large Margin Methods for Structured Prediction. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 169–176, New York, NY, USA. Association for Computing Machinery.

Hal Daumé III and Daniel Marcu. 2006. Bayesian Query-Focused Summarization. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 305–312, Sydney, Australia. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Delbert Dueck. 2009. *Affinity Propagation: Clustering Data by Passing Messages*. Ph.D. thesis.

Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. 1988. Using Latent Semantic Analysis to Improve Access to Textual Information. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '88, page 281–285, New York, NY, USA. Association for Computing Machinery.

Vincent Dumoulin and Francesco Visin. 2016. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Ted Dunning. 1993. Accurate Methods for the Statistics of Surprise and Coincidence. *Computational Linguistics*, 19(1):61–74.

Esin Durmus, He He, and Mona Diab. 2020. FEQA: A Question Answering Evaluation Framework for Faithfulness Assessment in Abstractive Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5055–5070, Online. Association for Computational Linguistics.

Greg Durrett, Taylor Berg-Kirkpatrick, and Dan Klein. 2016. Learning-Based Single-Document Summarization with Compression and Anaphoricity Constraints. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1998–2008, Berlin, Germany. Association for Computational Linguistics.

Ondřej Dušek, David M. Howcroft, and Verena Rieser. 2019. Semantic Noise Matters for Neural Natural Language Generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 421–426, Tokyo, Japan. Association for Computational Linguistics.

Ondřej Dušek and Filip Jurčíček. 2016. Sequence-to-Sequence Generation for Spoken Dialogue via Deep Syntax Trees and Strings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 45–51, Berlin, Germany. Association for Computational Linguistics.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2018. Findings of the E2E NLG challenge. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 322–328, Tilburg University, The Netherlands. Association for Computational Linguistics.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2020. Evaluating the state-of-the-art of End-to-End Natural Language Generation: The E2E NLG challenge. *Computer Speech & Language*, 59:123 – 156.

Henry Elder, Sebastian Gehrmann, Alexander O'Connor, and Qun Liu. 2018. E2E NLG Challenge Submission: Towards Controllable Generation of Diverse Natural Language. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 457–462, Tilburg University, The Netherlands. Association for Computational Linguistics.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

Günes Erkan and Dragomir R. Radev. 2004. LexRank: Graph-Based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22(1):457–479.

Tobias Falke and Iryna Gurevych. 2019. Fast Concept Mention Grouping for Concept Map-based Multi-Document Summarization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 695–700, Minneapolis, Minnesota. Association for

Computational Linguistics.

Tobias Falke, Christian M. Meyer, and Iryna Gurevych. 2017. Concept-Map-Based Multi-Document Summarization using Concept Coreference Resolution and Global Importance optimization. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 801–811, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Katja Filippova and Michael Strube. 2008. Sentence Fusion via Dependency Graph Compression. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 177–185, Honolulu, Hawaii. Association for Computational Linguistics.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition*, 28(1):3 – 71.

John R. Frank, Max Kleiman-Weiner, Daniel A. Roberts, Feng Niu, Ce Zhang, Christopher Ré, and Ian Soboroff. 2012. Building an Entity-Centric Stream Filtering Test Collection for TREC 2012. In *Proceedings of The Twenty-First Text REtrieval Conference, TREC 2012, Gaithersburg, Maryland, USA, November 6-9, 2012*, volume 500-298 of *NIST Special Publication*. National Institute of Standards and Technology (NIST).

Brendan J. Frey and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science*, 315(5814):972–976.

William A. Gale and Kenneth W. Church. 1993. A Program for Aligning Sentences in Bilingual Corpora. *Computational Linguistics*, 19(1):75–102.

Michel Galley, Chris Brockett, Alessandro Sordoni, Yangfeng Ji, Michael Auli, Chris Quirk, Margaret Mitchell, Jianfeng Gao, and Bill Dolan. 2015. deltaBLEU: A discriminative metric for generation tasks with intrinsically diverse targets. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 445–450, Beijing, China. Association for Computational Linguistics.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. 2020. Evaluating Models' Local Decision Boundaries via Contrast Sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1307–1323, Online. Association for Computational Linguistics.

M. Gašić, Dongho Kim, Pirros Tsiakoulis, Catherine Breslin, Matthew Henderson, Martin Szummer, Blaise Thomson, and Steve Young. 2014. Incremental on-line adaptation of POMDP-based dialogue managers to extended domains. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 140–144. ISCA.

Albert Gatt and Emiel Krahmer. 2018. Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61(1):65–170.

Albert Gatt, François Portet, Ehud Reiter, Jim Hunter, Saad Mahamood, Wendy Moncur, and Somayajulu Sripada. 2009. From Data to Text in the Neonatal Intensive Care Unit: Using NLG Technology for Decision Support and Information Management. *AI Communications*, 22(3):153–186.

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Harvard University Press.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252, International Convention Centre, Sydney, Australia. PMLR.

Sebastian Gehrmann, Falcon Dai, Henry Elder, and Alexander Rush. 2018. End-to-End Content and Plan Selection for Data-to-Text Generation. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 46–56, Tilburg University, The Netherlands. Association for Computational Linguistics.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. JMLR Workshop and Conference Proceedings.

Eli Goldberg, Norbert Driedger, and Richard I. Kittredge. 1994. Using Natural-Language Processing to Produce Weather Forecasts. *IEEE Expert: Intelligent Systems and Their Applications*, 9(2):45–53.

Jade Goldstein and Jaime Carbonell. 1998. Summarization: (1) Using MMR for Diversity-Based Reranking and (2) Evaluating Summaries. In *TIPSTER TEXT PROGRAM PHASE III: Proceedings of a Workshop held at Baltimore, Maryland, October 13-15, 1998*, pages 181–195, Baltimore, Maryland, USA. Association for Computational Linguistics.

Sergey Golovanov, Rauf Kurbanov, Sergey Nikolenko, Kyryl Truskovskyi, Alexander Tselousov, and Thomas Wolf. 2019. Large-Scale Transfer Learning for Natural Language Generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6053–6058, Florence, Italy. Association for Computational Linguistics.

David Graff and C Cieri. 2003. English Gigaword Corpus. *Linguistic Data Consortium.*

Alex Graves and Jürgen Schmidhuber. 2005. 2005 Special Issue: Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5–6):602–610.

Barbara J. Grosz, Aravind K. Joshi, and Scott Weinstein. 1995. Centering: A Framework for Modeling the Local Coherence of Discourse. *Computational Linguistics*, 21(2):203–225.

Qi Guo, Fernando Diaz, and Elad Yom-Tov. 2013. Updating Users about Time Critical Events. In *Advances in Information Retrieval - 35th European Conference on IR Research, ECIR 2013, Moscow, Russia, March 24-27, 2013. Proceedings*, volume 7814 of *Lecture Notes in Computer Science*, pages 483–494. Springer.

Weiwei Guo and Mona Diab. 2012. Weiwei: A Simple Unsupervised Latent Semantics based Approach for Sentence Similarity. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 586–590, Montréal, Canada. Association for Computational Linguistics.

Jeremiah Hackett. 2020. Roger bacon. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*, summer 2020 edition. Metaphysics Research Lab, Stanford University.

Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24(2):8–12.

M. A. K. Halliday. 1985. *An Introduction To Functional Grammar*. Edward Arnold.

Hardy and Andreas Vlachos. 2018. Guided Neural Language Generation for Abstractive Summarization using Abstract Meaning Representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 768–773, Brussels, Belgium. Association for Computational Linguistics.

Donna Harman. 2001. The Importance of Focused Evaluations: a Case Study of TREC and DUC. In *Proceedings of the Third Second Workshop Meeting on Evaluation of Chinese & Japanese Text Retrieval and Text Summarization, NTCIR-2, Tokyo, Japan, March 7-9, 2001*. National Institute of Informatics (NII).

He He, Sheng Zha, and Haohan Wang. 2019. Unlearn Dataset Bias in Natural Language Inference by Fitting the Residual. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 132–142, Hong Kong, China. Association for Computational Linguistics.

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1693–1701, Cambridge, MA, USA. MIT Press.

Tsutomu Hirao, Hideki Isozaki, Eisaku Maeda, and Yuji Matsumoto. 2002. Extracting Important Sentences with Support Vector Machines. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative Back-Translation for Neural Machine Translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, Melbourne, Australia. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649, Melbourne, Australia. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019. The Curious Case of Neural Text Degeneration. *CoRR*, abs/1904.09751.

Kai Hong and Ani Nenkova. 2014. Improving the Estimation of Word Importance for News Multi-Document Summarization. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 712–721, Gothenburg, Sweden. Association for Computational Linguistics.

Eduard H. Hovy. 1993. NATURAL LANGUAGE PROCESSING BY THE PENMAN PROJECT AT USC/ISI. Technical report, University of Southern California/Information Sciences Institute.

Hugging Face. 2019. *Write with Transformer*.

John Hutchins. 1994. Research methods and system designs in machine translation: a ten-year review, 1984-1994. In *Machine Translation: Ten Years On*, pages 1–16.

John Hutchins. 2006. Machine Translation: History. *Encyclopedia of Language & Linguistics*, pages 375–383.

Wendy Beth Hyman, editor. 2016. *The Automaton in English Renaissance Literature*. Routledge.

L. Iordanskaja, M. Kim, R. Kittredge, B. Lavoie, and A. Polguere. 1992. Generation of extended bilingual statistical reports. In *COLING 1992 Volume 3: The 15th International Conference on Computational Linguistics*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China. Association for Computational Linguistics.

Hongyan Jing. 2000. Sentence Reduction for Automatic Text Summarization. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, ANLC '00, page 310–315, USA. Association for Computational Linguistics.

Hongyan Jing and Kathleen R. McKeown. 2000. Cut and Paste Based Text Summarization. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

Juraj Juraska, Kevin Bowden, and Marilyn Walker. 2019. ViGGO: A Video Game Corpus for Data-

To-Text Generation in Open-Domain Conversation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 164–172, Tokyo, Japan. Association for Computational Linguistics.

Juraj Juraska, Panagiotis Karagiannis, Kevin Bowden, and Marilyn Walker. 2018. A Deep Ensemble Model with Slot Alignment for Sequence-to-Sequence Natural Language Generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 152–162, New Orleans, Louisiana. Association for Computational Linguistics.

David Kahn. 1980. On the Origin of Polyalphabetic Substitution. *Isis*, 71(1):122–127.

Chris Kedzie, Fernando Diaz, and Kathleen McKeown. 2016. Real-Time Web Scale Event Summarization Using Sequential Decision Making. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, page 3754–3760. AAAI Press.

Chris Kedzie and Kathleen McKeown. 2019. A Good Sample is Hard to Find: Noise Injection Sampling and Self-training for Neural Language Generation Models. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 584–593, Tokyo, Japan. Association for Computational Linguistics.

Chris Kedzie and Kathleen McKeown. 2020. Controllable Meaning Representation to Text Generation: Linearization and Data Augmentation Strategies. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5160–5185, Online. Association for Computational Linguistics.

Chris Kedzie, Kathleen McKeown, and Hal Daumé III. 2018. Content Selection in Deep Learning Models of Summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1818–1828, Brussels, Belgium. Association for Computational Linguistics.

Chris Kedzie, Kathleen McKeown, and Fernando Diaz. 2015. Predicting Salient Updates for Disaster Summarization. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1608–1617, Beijing, China. Association for Computational Linguistics.

M. G. Kendall. 1938. A NEW MEASURE OF RANK CORRELATION. *Biometrika*, 30(1-2):81–93.

Vikash Khandelwal, Rahul Gupta, and James Allan. 2001. An Evaluation Corpus for Temporal Summarization. In *Proceedings of the First International Conference on Human Language Technology Research*.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Donaled E. Knuth. 2006. *The Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees–History of Combinatorial Generation*. Pearson Education.

Philipp Koehn and Rebecca Knowles. 2017. Six Challenges for Neural Machine Translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Wojciech Kryscinski, Nitish Shirish Keskar, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Neural Text Summarization: A Critical Evaluation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 540–551, Hong Kong, China. Association for Computational Linguistics.

Wojciech Kryscinski, Bryan McCann, Caiming Xiong, and Richard Socher. 2020. Evaluating the Factual Consistency of Abstractive Text Summarization. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9332–9346, Online. Association for Computational Linguistics.

Julian Kupiec, Jan Pedersen, and Francine Chen. 1995. A Trainable Document Summarizer. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, page 68–73, New York, NY, USA. Association for Computing Machinery.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Brenden Lake and Marco Baroni. 2018. Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks. volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882, Stockholmsmässan, Stockholm Sweden. PMLR.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural Text Generation from Structured Data with Application to the Biography Domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213, Austin, Texas. Association for Computational Linguistics.

Peter Lee. 2016. Learning from Tay's introduction. *blogs.microsoft.com*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016. A Diversity-Promoting Objective Function for Neural Conversation Models. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 110–119, San Diego, California. Association for Computational Linguistics.

Kexin Liao, Logan Lebanoff, and Fei Liu. 2018. Abstract Meaning Representation for Multi-Document Summarization. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1178–1190, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Chin-Yew Lin and Eduard Hovy. 2000. The Automated Acquisition of Topic Signatures for Text Summarization. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*.

Hui Lin and Jeff Bilmes. 2010. Multi-document Summarization via Budgeted Maximization of Submodular Functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, Los Angeles, California. Association for Computational Linguistics.

David Link. 2010. Scrambling T-R-U-T-H: Rotating Letters as a Material Form of Thought. In Siegfried Zielinski and Eckhard Fuerlus, editors, *Variantology 4. On Deep Time Relations of Arts, Sciences and Technologies in the Arabic-Islamic World and Beyond*, pages 215–266. Walther König.

Tal Linzen. 2020. How Can We Accelerate Progress Towards Human-like Linguistic Generalization? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5210–5217, Online. Association for Computational Linguistics.

Michael Lederman Littman. 1996. *Algorithms for Sequential Decision-Making*. Ph.D. thesis.

Yang Liu and Mirella Lapata. 2019. Text Summarization with Pretrained Encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3730–3740, Hong Kong, China. Association for Computational Linguistics.

Annie Louis. 2014. A Bayesian Method to Incorporate Background Knowledge during Automatic Text Summarization. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 333–338, Baltimore, Maryland. Association for Computational Linguistics.

Annie Louis and Ani Nenkova. 2013. Automatically Assessing Machine Summary Content Without a Gold Standard. *Computational Linguistics*, 39(2):267–300.

Xiaofei Lu. 2009. Automatic measurement of syntactic complexity in child language acquisition. *International Journal of Corpus Linguistics*, 14(1):3–28.

Hans Peter Luhn. 1958. The Automatic Creation of Literature Abstracts. *IBM Journal of research and development*, 2(2):159–165.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. 2010. Phrase-Based Statistical Language Generation Using Graphical Models and Active Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1552–1561, Uppsala, Sweden. Association for Computational Linguistics.

William C. Mann, Madeline Bates, Barbara J. Grosz, David D. McDonald, and Kathleen R. McKeown. 1981. Text Generation: The State of the Art and the Literature. Technical report, University of Southern California/Information Sciences Institute.

Gary F Marcus. 2003. *The Algebraic Mind: Integrating Connectionism and Cognitive Science*. MIT Press.

Erwin Marsi and Emiel Krahmer. 2005. Explorations in Sentence Fusion. In *Proceedings of the Tenth European Workshop on Natural Language Generation (ENLG-05)*.

André Martins and Noah A. Smith. 2009. Summarization with a Joint Model for Sentence Extraction and Compression. In *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, pages 1–9, Boulder, Colorado. Association for Computational Linguistics.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On Faithfulness and Factuality in Abstractive Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2014. Incremental Update Summarization: Adaptive Sentence Selection Based on Prevalence and Novelty. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, page 301–310, New York, NY, USA. Association for Computing Machinery.

David D. McDonald. 1981. MUMBLE: A Flexible System for Language Production. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 1062, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

David D. McDonald. 2010. Natural language generation. *Handbook of Natural Language Processing*, 2:121–144.

Ryan McDonald. 2007. A Study of Global Inference Algorithms in Multi-Document Summarization. In *Proceedings of the 29th European Conference on IR Research*, ECIR'07, page 557–564, Berlin, Heidelberg. Springer-Verlag.

Kathleen McKeown. 1985. Discourse Strategies for Generating Natural-Language Text. *Artificial Intelligence*, 27(1):1–41.

Kathleen McKeown. 1986. Language Generation: Applications, Issues, and Approaches. *Proceedings of the IEEE*, 74:961 – 968.

Kathleen McKeown, Karen Kukich, and James Shaw. 1994. Practical Issues in Automatic Documentation Generation. In *Fourth Conference on Applied Natural Language Processing*, pages 7–14, Stuttgart, Germany. Association for Computational Linguistics.

Kathleen McKeown, Rebecca J. Passonneau, David K. Elson, Ani Nenkova, and Julia Hirschberg. 2005. Do Summaries Help? a task-based evaluation of multi-document summarization. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, page 210–217, New York, NY, USA. Association for Computing Machinery.

Kathleen R. McKeown. 1982. The TEXT System for Natural Language Generation: An Overview. In *20th Annual Meeting of the Association for Computational Linguistics*, pages 113–120, Toronto, Ontario, Canada. Association for Computational Linguistics.

Clara Meister, Ryan Cotterell, and Tim Vieira. 2020. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2173–2185, Online. Association for Computational Linguistics.

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing Order into Text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411, Barcelona, Spain. Association for Computational Linguistics.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. In *INTERSPEECH*, pages 1045–1048. ISCA.

George A. Miller. 1995. WordNet: A Lexical Database for English. *Communications of the ACM*, 38:39–41.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019a. Improving Quality and Efficiency in Plan-based Neural Data-to-text Generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 377–382, Tokyo, Japan. Association for Computational Linguistics.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019b. Step-by-Step: Separating Planning from

Realization in Neural Data-to-Text Generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2267–2277, Minneapolis, Minnesota. Association for Computational Linguistics.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA. Omnipress.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3075–3081. AAAI Press.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Courtney Napoles, Benjamin Van Durme, and Chris Callison-Burch. 2011. Evaluating Sentence Compression: Pitfalls and Suggested Remedies. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*, pages 91–97, Portland, Oregon. Association for Computational Linguistics.

Neha Nayak, Dilek Hakkani-Tür, Marilyn Walker, and Larry Heck. 2017. To Plan or not to Plan? Discourse Planning in Slot-Value Informed Sequence to Sequence Models for Language Generation. In *Proceedings of Interspeech 2017*, pages 3339–3343.

Ani Nenkova. 2005. Automatic Text Summarization of Newswire: Lessons Learned from the Document Understanding Conference. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, page 1436–1441. AAAI Press.

Ani Nenkova and Kathleen R. McKeown. 2011. Automatic Summarization. *Foundations and Trends in Information Retrieval*, 5(2-3):103–233.

Ani Nenkova and Lucy Vanderwende. 2005. The Impact of Frequency on Summarization. Technical report, Microsoft Research.

Dang Tuan Nguyen and Trung Tran. 2018. Structure-based Generation System for E2E NLG Challenge.

Malvina Nissim, Rik van Noord, and Rob van der Goot. 2020. Fair is Better than Sensational: Man Is to Doctor as Woman Is to Doctor. *Computational Linguistics*, 46(2):487–497.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany. Association for Computational Linguistics.

Tim O'Gorman, Michael Regan, Kira Griffitt, Ulf Hermjakob, Kevin Knight, and Martha Palmer. 2018. AMR Beyond the Sentence: the Multi-sentence AMR corpus. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3693–3702, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Jacob Ornstein. 1955. Mechanical Translation: New Challenge to Communication. *Science*, 122(3173):745–748.

Miles Osborne. 2002. Using maximum entropy for sentence extraction. In *Proceedings of the ACL-02 Workshop on Automatic Summarization*, pages 1–8, Phildadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Jessica Ouyang, Serina Chang, and Kathy McKeown. 2017. Crowd-Sourced Iterative Annotation for Narrative Summarization Corpora. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 46–51, Valencia, Spain. Association for Computational Linguistics.

P. Over and W. Liggett. 2002. Introduction to DUC: An Intrinsic Evaluation of Generic News Text Summarization Systems.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Cecile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley, Lyn Pemberton, Richard Power, and Donia Scott. 1995. A Support Tool for Writing Multilingual Instructions. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1398–1404, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Steven Phillips. 1998. Are Feedforward and Recurrent Networks Systematic? Analysis and Implications for a Connectionist Cognitive Architecture. *Connection Science*, 10(2):137–160.

Rimma Pivovarov and Noémie Elhadad. 2015. Automated methods for the summarization of electronic health records. *Journal of the American Medical Informatics Association*, 22(5):938–947.

Ernesto Priani. 2017. Ramon Llull. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*, Spring 2017 edition. Metaphysics Research Lab, Stanford University.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2E NLG Challenge: Neural Models vs. Templates. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 463–471, Tilburg University, The Netherlands. Association for Computational Linguistics.

Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. 2000. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies. In *NAACL-ANLP 2000 Workshop: Automatic Summarization*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press.

Raj Reddy. 1977. Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort at Carnegie-Mellon University. Technical report, Department of Computer Science. Carnegie-Mellon University.

Lena Reed, Shereen Oraby, and Marilyn Walker. 2018. Can Neural Generators for Dialogue Learn Sentence Planning and Discourse Structuring? In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 284–295, Tilburg University, The Netherlands. Association for Computational Linguistics.

Ehud Reiter. 1994. Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation*.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press.

Stefan Riezler and John T. Maxwell. 2005. On Some Pitfalls in Automatic Evaluation and Significance Testing for MT. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 57–64, Ann Arbor, Michigan. Association for Computational Linguistics.

Adi Robertson. 2019. The infinite text adventure AI Dungeon 2 is now easy to play online: Dive into an adventure game powered by dream logic. *The Verge*.

Sheldon Rosenberg and Leonard Abbeduto. 1987. Indicators of linguistic competence in the peer group conversational behavior of mildly retarded adults. *Applied Psycholinguistics*, 8(1):19–32.

F. Rosenthal, A. R. I. Khaldūn, and N. J. Dawood. 1958. *The Muqaddimah : an introduction to history ; in three volumes.* Bollingen Series (General) Series. Pantheon Books.

Stephane Ross and Drew Bagnell. 2010. Efficient Reductions for Imitation Learning. volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy. JMLR Workshop and Conference Proceedings.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer Learning in Natural Language Processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18.

Alexander Rush. 2018. The Annotated Transformer. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 52–60, Melbourne, Australia. Association for Computational Linguistics.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A Neural Attention Model for Abstractive Sentence Summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

Sigal Samuel. 2019. How I'm using AI to write my next novel. *Vox*.

Evan Sandhaus. 2008. The new york times annotated corpus. *Linguistic Data Consortium, Philadelphia*, 6(12):e26752.

Jörgen Schäfer. 2006. Literary Machines Made in Germany. German Proto-Cybertexts from the Baroque Era to the Present. *The Cybertext Yearbook Database. Theme Issue on Ergodic Histories*, pages 1–69.

Holger Schwenk, Daniel Dechelotte, and Jean-Luc Gauvain. 2006. Continuous Space Language Models for Statistical Machine Translation. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 723–730, Sydney, Australia. Association for Computational Linguistics.

John Seabrook. 2019. The Next Word. *The New Yorker*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning Robust Metrics for Text Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.

Iulian V. Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2016. Building End-to-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 3776–3783. AAAI Press.

Janelle Shane. 2020. Facts about whales. *aiweirdness.com*.

Tom Simonite. 2019. The AI Text Generator That's Too Dangerous to Make Public. *Wired*.

Ruben Sipos, Pannaga Shivaswamy, and Thorsten Joachims. 2012. Large-margin learning of submodular summarization models. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 224–233, Avignon, France. Association for Computational Linguistics.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A Neural Network Approach to Context-Sensitive Generation of Conversational Responses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 196–205, Denver, Colorado. Association for Computational Linguistics.

Karen Spärck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21.

Karen Spärck Jones. 1999. Automatic Summarizing: Factors and Directions. In Inderjeet Mani and Mark T. Maybury, editors, *Advances in Automatic Text Summarization*, chapter 1, pages 1–12. The MIT Press.

Stephen Springer, Paul Buta, and T. C. Wolf. 1991. Automatic Letter Composition for Customer Service. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, pages 67–83.

Nathan Srebro and Tommi Jaakkola. 2003. Weighted Low-Rank Approximations. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 720–727. AAAI Press.

Felix Stahlberg and Bill Byrne. 2019. On NMT Search Errors and Model Errors: Cat Got Your Tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3356–3362, Hong Kong, China. Association for Computational Linguistics.

Kate Starbird and Leysia Palen. 2013. Working and Sustaining the Virtual "Disaster Desk". In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, page 491–502, New York, NY, USA. Association for Computing Machinery.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *7th International Conference on Spoken Language Processing, ICSLP2002 - INTERSPEECH 2002, Denver, Colorado, USA, September 16-20, 2002*. ISCA.

Matthew Stone, Christine Doran, Bonnie Webber, Tonia Bleam, and Martha Palmer. 2003. Microplanning with Communicative Intentions: The SPUD System. *Computational Intelligence*, 19(4):311–381.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, volume 27, pages 3104–3112. Curran Associates, Inc.

William R. Swartout. 1983. XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3):285–325.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

Bryan S. Todd. 1992. *An Introduction to Expert Systems*. Oxford University Computing Laboratory, Programming Research Group.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

James Vincent. 2019. OpenAI has published the text-generating AI it said was too dangerous to share. *The Verge*.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. 2015. Show and Tell: A Neural Image Caption Generator. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164.

Oriol Vinyals and Quoc V. Le. 2015. A Neural Conversational Model.

Marilyn A. Walker, Owen Rambow, and Monica Rogati. 2001. SPoT: A Trainable Sentence Planner. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Alex Wang, Kyunghyun Cho, and Mike Lewis. 2020a. Asking and Answering Questions to Evaluate the Factual Consistency of Summaries. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5008–5020, Online. Association for Computational Linguistics.

Hongmin Wang. 2019. Revisiting Challenges in Data-to-Text Generation with Fact Grounding. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 311–322, Tokyo, Japan. Association for Computational Linguistics.

Liang Wang, Wei Zhao, Ruoyu Jia, Sujian Li, and Jingming Liu. 2019. Denoising based Sequence-to-Sequence Pre-training for Text Generation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4003–4015, Hong Kong, China. Association for Computational Linguistics.

Lu Wang, Hema Raghavan, Vittorio Castelli, Radu Florian, and Claire Cardie. 2013. A Sentence Compression Based Framework to Query-Focused Multi-Document Summarization. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1384–1394, Sofia, Bulgaria. Association for Computational Linguistics.

Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. 2015. Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network.

Tianming Wang, Xiaojun Wan, and Shaowei Yao. 2020b. Better AMR-To-Text Generation with Graph Structure Reconstruction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 3919–3925. International Joint Conferences on Artificial Intelligence Organization.

B. L. Welch. 1947. THE GENERALIZATION OF 'STUDENT'S' PROBLEM WHEN SEVERAL DIFFERENT POPULATION VARLANCES ARE INVOLVED. *Biometrika*, 34(1-2):28–35.

Sean Welleck, Ilia Kulikov, Jaedeok Kim, Richard Yuanzhe Pang, and Kyunghyun Cho. 2020. Consistency of a Recurrent Language Model With Respect to Incomplete Decoding. pages 5553–5568.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016. Multi-domain neural network language generation for spoken dialogue systems. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 120–129, San Diego, California. Association for Computational Linguistics.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721, Lisbon, Portugal. Association for Computational Linguistics.

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2016. Towards Universal Paraphrastic Sentence Embeddings. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

John Wieting and Kevin Gimpel. 2017. Revisiting Recurrent Networks for Paraphrastic Sentence Embeddings. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 2078–2088. Association for Computational Linguistics.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2017. Challenges in Data-to-Document Generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263, Copenhagen, Denmark. Association for Computational Linguistics.

Sam Wiseman, Stuart Shieber, and Alexander Rush. 2018. Learning Neural Templates for Text Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language*

Processing, pages 3174–3187, Brussels, Belgium. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, abs/1609.08144.

Ziang Xie. 2017. Neural Text Generation: A Practical Guide. *CoRR*, abs/1711.09534.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France. PMLR.

Carl Yang, Jieyu Zhang, Haonan Wang, Bangzheng Li, and Jiawei Han. 2020. Neural Concept Map Generation for Effective Document Classification with Interpretable Structured Summarization. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1629–1632, New York, NY, USA. Association for Computing Machinery.

Victor H. Yngve. 1961. Random Generation of English Sentences. In *Proceedings of the International Conference on Machine Translation of Languages and Applied Language Analysis*, National Physical Laboratory, Teddington, UK.

Elad Yom-Tov and Fernando Diaz. 2011. Out of Sight, Not out of Mind: On the Effect of Social and Physical Detachment on Information Need. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 385–394, New York, NY, USA. Association for Computing Machinery.

David Zajic, Bonnie J. Dorr, Jimmy Lin, and Richard Schwartz. 2007. Multi-Candidate Reduction: Sentence Compression as a Tool for Document Summarization Tasks. *Information Processing & Management*, 43(6):1549–1570.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J Liu. 2019a. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. *arXiv preprint arXiv:1912.08777*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019b. AMR Parsing as Sequence-to-Graph Transduction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 80–94, Florence, Italy. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating Text Generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Zachary M. Ziegler, Luke Melas-Kyriazi, Sebastian Gehrmann, and Alexander M. Rush. 2019.

Encoder-Agnostic Adaptation for Conditional Language eneration. *CoRR*, abs/1908.06938.

Appendix A: GRU-based Sequence-to-Sequence Architecture

The GRU is a form of reccurent neural network (Elman, 1990) that operates over discrete sequences, which upon receiving a new input token, updates a "hidden state" or internal representation using the current input and the previous hidden state. In the sequence-to-sequence paradigm, both the encoder and decoder are built upon distinct GRU layers.

The encoder consists of an embedding layer which maps the discrete input sequence to a sequence embeddings. The encoder input embedding sequence is then fed through one or more GRU layers. Optionally, the encoder GRUs can be run uni-directionally (i.e., proceeding left-to-right), or bi-directionally (i.e. distinct left-to-right and right-to-left GRUs process the input sequence and concatenate the output). We describe both cases below. After encoding the input, the final state of the encoder is optionally run through a bridge layer to project it to a compatible size for the decoder.

The decoder also has an embedding layer which it uses to map previously generated output tokens to embeddings which are then fed into the one or more uni-directional decoder GRUs. The decoder hidden state at each step attends to the encoder hidden states, producing an "attention vector," i.e. a weighted sum of the encoder hidden states. The decoder state and the attention vector are concatenated and fed through a feed-forward layer with softmax output to produce a probability distribution over the next token.

See Figure A.1 for a schematic example of the GRU-based sequence-to-sequence model. We now describe the individual components in detail.

Encoder Embedding Layer Let $\mathbf{x} = [x_1, \ldots, x_m]$ be a linearized meaning representation token sequence. Before feeding $\mathbf{x}$ into the encoder GRU layer, we first embed each token. Let $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}_X| \times D_w}$ be the encoder input embedding matrix, where each row, $\mathbf{m}_i$, is a D_w-dimensional

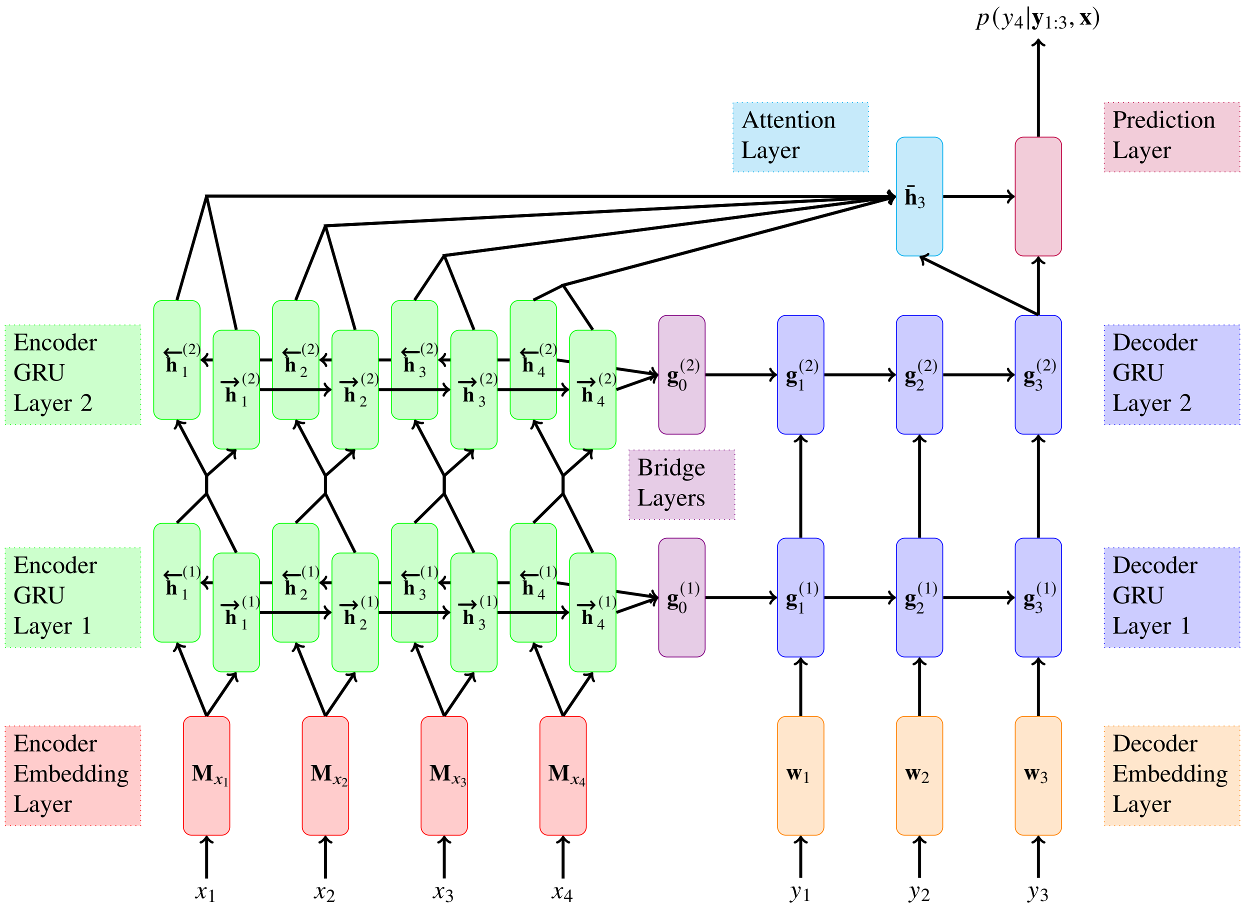

$p(y_4|\mathbf{y}_{1:3}, \mathbf{x})$
Attention Layer
Prediction Layer
$\bar{\mathbf{h}}_3$
Encoder GRU Layer 2
$\overleftarrow{\mathbf{h}}_1^{(2)}$ $\overrightarrow{\mathbf{h}}_1^{(2)}$ $\overleftarrow{\mathbf{h}}_2^{(2)}$ $\overrightarrow{\mathbf{h}}_2^{(2)}$ $\overleftarrow{\mathbf{h}}_3^{(2)}$ $\overrightarrow{\mathbf{h}}_3^{(2)}$ $\overleftarrow{\mathbf{h}}_4^{(2)}$ $\overrightarrow{\mathbf{h}}_4^{(2)}$
$\mathbf{g}_0^{(2)}$ $\mathbf{g}_1^{(2)}$ $\mathbf{g}_2^{(2)}$ $\mathbf{g}_3^{(2)}$
Decoder GRU Layer 2
Bridge Layers
Encoder GRU Layer 1
$\overleftarrow{\mathbf{h}}_1^{(1)}$ $\overrightarrow{\mathbf{h}}_1^{(1)}$ $\overleftarrow{\mathbf{h}}_2^{(1)}$ $\overrightarrow{\mathbf{h}}_2^{(1)}$ $\overleftarrow{\mathbf{h}}_3^{(1)}$ $\overrightarrow{\mathbf{h}}_3^{(1)}$ $\overleftarrow{\mathbf{h}}_4^{(1)}$ $\overrightarrow{\mathbf{h}}_4^{(1)}$
$\mathbf{g}_0^{(1)}$ $\mathbf{g}_1^{(1)}$ $\mathbf{g}_2^{(1)}$ $\mathbf{g}_3^{(1)}$
Decoder GRU Layer 1
Encoder Embedding Layer
$\mathbf{M}_{x_1}$ $\mathbf{M}_{x_2}$ $\mathbf{M}_{x_3}$ $\mathbf{M}_{x_4}$
$\mathbf{w}_1$ $\mathbf{w}_2$ $\mathbf{w}_3$
Decoder Embedding Layer
x_1 x_2 x_3 x_4
y_1 y_2 y_3

Figure A.1: Schematic of the bi-directional GRU-based sequence-to-sequence model.

embedding for a token in $y \in \mathcal{V}_{\mathcal{X}}$, i.e.,

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_{|\mathcal{V}_{\mathcal{X}}|} \end{bmatrix}.$$

We assume each element $x \in \mathcal{V}_{\mathcal{X}}$ is uniquely identified with a row $i \in \{1, \ldots, |\mathcal{V}_{\mathcal{X}}|\}$. We indicate the embedding of x as $\mathbf{M}_x = \mathbf{m}_i$. The input to the encoder GRU layer then is

$$\left[\mathbf{h}_1^{(0)}, \ldots, \mathbf{h}_m^{(0)}\right] = \left[\mathbf{M}_{x_1}, \ldots, \mathbf{M}_{x_m}\right].$$

Encoder Uni-directional GRU Layers We then compute the GRU hidden states. The encoder can have an arbitray number of layers L. For each layer $l \in \{1, \ldots, L\}$ we compute,

$$\begin{aligned} \mathbf{h}_0^{(l)} &= \mathbf{0} \\ \mathbf{h}_i^{(l)} &= \mathrm{GRU}(\mathbf{h}_i^{(l-1)}, \mathbf{h}_{i-1}^{(l)}; \theta_{\mathcal{E}}^{(l)}) && \forall i : i \in \{1, \ldots, m\} \end{aligned}$$

where $\theta_{\mathcal{E}}^{(l)}$ are the GRU encoder parameters[1] for the l-th layer and $\mathbf{h}_i^{(l)} \in \mathbb{R}^{D_h}$. The encoder GRU layers output the sequence of hidden states, $\mathbf{h}_1, \ldots, \mathbf{h}_m$, used by the decoder to represent the input; in the uni-directional case, these are simply the last GRU layer outputs, i.e. $\mathbf{h}_i = \mathbf{h}_i^{(L)} \in \mathbb{R}^{D_{\mathcal{E}}}$ (in the uni-directional case, $D_h = D_{\mathcal{E}}$).

Encoder Bi-directional GRU Layers The uni-directional encoder may suffer from a recency bias when creating the initial state for the decoder and for longer input sequences the encoder may "forget" information encoded in the early hidden states. In practice to alleviate this another GRU is run in the opposite direction and its outputs are concatenated. For the first layer, we have,

(Encoder Forward GRU)

$$\begin{aligned} \overrightarrow{\mathbf{h}}_0^{(1)} &= \mathbf{0}, \\ \overrightarrow{\mathbf{h}}_i^{(1)} &= \mathrm{GRU}\left(\mathbf{h}_i^{(0)}, \overrightarrow{\mathbf{h}}_{i-1}^{(1)}; \theta_{\overrightarrow{\mathcal{E}}}^{(l)}\right), && \forall i : i \in \{1, \ldots, m\} \end{aligned}$$

(Encoder Backward GRU)

$$\begin{aligned} \overleftarrow{\mathbf{h}}_0^{(1)} &= \mathbf{0}, \\ \overleftarrow{\mathbf{h}}_i^{(1)} &= \mathrm{GRU}\left(\mathbf{h}_i^{(0)}, \overleftarrow{\mathbf{h}}_{i+1}^{(1)}; \theta_{\overleftarrow{\mathcal{E}}}^{(l)}\right) && \forall i : i \in \{1, \ldots, m\} \\ \mathbf{h}_i^{(1)} &= \begin{bmatrix} \overrightarrow{\mathbf{h}}_i^{(1)} \\ \overleftarrow{\mathbf{h}}_i^{(1)} \end{bmatrix} && \forall i : i \in \{1, \ldots, m\} \end{aligned}$$

[1] See Equation 3.2 for the definition of the GRU function.

where $\theta_{\overrightarrow{\mathcal{E}}}$ and $\theta_{\overleftarrow{\mathcal{E}}}$ are forward and backward GRU parameters respectively, $\overrightarrow{\mathbf{h}}_i^{(1)}, \overleftarrow{\mathbf{h}}_i^{(1)} \in \mathbb{R}^{D_h}$, and first layer hidden state, $\mathbf{h}_i^{(1)} \in \mathbb{R}^{2D_h}$, is a concatenation of the forward and backward hidden states at step i. The subsequent layers are computed similarly, but the input to the GRUs are $2D_h$-dimensional. Like before, the encoder outputs are the hidden state outputs of the last layer, $\mathbf{h}_i = \mathbf{h}_i^{(L)} \in \mathbb{R}^{D_\mathcal{E}}$ where $D_\mathcal{E} = 2D_h$.

Decoder Embedding Layer We then embed the utterance token sequence, $\mathbf{y} = [y_1, \ldots, y_n]$, before feeding it to the decoder GRU layers.. Let $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}_\mathcal{Y}| \times D_w}$ be an embedding matrix of the utterance tokens $y \in \mathcal{V}_\mathcal{Y}$ defined analogously to the encoder embedding matrix $\mathbf{M}$. The input to the decoder then is

$$\left[\mathbf{g}_1^{(0)}, \ldots, \mathbf{g}_{n-1}^{(0)}\right] = \left[\mathbf{W}_{y_1}, \ldots, \mathbf{W}_{y_{n-1}}\right].^2$$

Bridge Layer We initialize the decoder hidden state with the final (i.e. m-th) state of the encoder GRU. In the case where $D_\mathcal{E} \neq D_h$, we need to project $\mathbf{h}_m^{(l)}$ to D_h dimensions,

$$\mathbf{g}_0^{(l)} = \begin{cases} \tanh\left(\mathbf{W}_l^{(br)}\mathbf{h}_m^{(l)} + \mathbf{b}_l^{(br)}\right) & D_\mathcal{E} \neq D_h \\ \mathbf{h}_m^{(l)} & \text{otherwise} \end{cases},$$

where $\mathbf{W}_l^{(br)} \in \mathbb{R}^{D_h \times D_\mathcal{E}}$ and $\mathbf{b}_l^{(br)} \in \mathbb{R}^{D_h}$ for $l \in \{1, \ldots, L\}$ are the weight and bias parameters for the "bridge layer" between the encoder and decoder networks.

Decoder GRU Layers The decoder GRU is then computed analogously to the uni-directional encoder GRU,

$$\mathbf{g}_i^{(l)} = \operatorname{GRU}\left(\mathbf{g}_i^{(l-1)}, \mathbf{g}_{i-1}^{(l)}; \theta_\mathcal{D}^{(l)}\right) \qquad \forall i : i \in \{1, \ldots, n-1\},$$

[2] The decoder input sequences have length $n - 1$ since the n^{th} token is always the stop token *«e»*, which is never fed into the decoder input.

where $\theta^{(l)}_{\mathcal{D}}$ are the decoder GRU parameters and $\mathbf{g}^{(l)}_i \in \mathbb{R}^{D_h}$ for $i \in \{1, \ldots, n-1\}$ and $l \in \{1, \ldots, L\}$. The decoder outputs, $\mathbf{g}_1, \ldots, \mathbf{g}_{n-1}$, are the decoder hidden states of the last decoder layer, i.e. $\mathbf{g}_i = \mathbf{g}^{(L)}_i \in \mathbb{R}^{D_{\mathcal{D}}}$ where $D_{\mathcal{D}} = D_h$.

Attention Layer As mentioned before, one drawback of the recurrent neural network design is that information from earlier states my not be preserved in later states. To ameliorate this, the attention mechanism was proposed to allow an arbitrary decoder state to retrieve information from an arbitrary encoder state (Bahdanau et al., 2015). This works by taking a weighted average of the encoder states,

$$\bar{\mathbf{h}}_i = \sum_{j=1}^{m} \alpha_{i,j} \mathbf{h}_j \qquad \forall i : i \in \{1, \ldots, n-1\}$$

where $\alpha_{i,j} \in (0, 1)$ is proportional to a score function $s(\mathbf{g}_i, \mathbf{h}_j)$ which measures some notion of "relevance" for decoder state i to encoder start j,

$$\alpha_{i,j} = \frac{\exp s(\mathbf{g}_i, \mathbf{h}_j)}{\sum_{j'=1}^{m} \exp s(\mathbf{g}_i, \mathbf{h}_{j'})} \quad \forall i, j : j \in \{1, \ldots, m\}, i \in \{1, \ldots, n-1\}.$$

There are several popular ways to implement s which we consider; we refer to three of them using the names given in Luong et al. (2015). When the encoder and decoder hidden states are of the same dimension, the simplest function is just the dot product, which we refer to as "dot-style" attention,

(Dot-Style Attention)

$$s(\mathbf{g}_i, \mathbf{h}_j) = \mathbf{g}_i \cdot \mathbf{h}_j.$$

If they are not the same dimension, one can insert a parameter matrix in place of the dot product,

(General-Style Attention)

$$s(\mathbf{g}_i, \mathbf{h}_j) = \mathbf{g}_i \mathbf{K} \mathbf{h}_j$$

where $\mathbf{K} \in \mathbb{R}^{D_{\mathcal{D}} \times D_{\mathcal{E}}}$ is a learned parameter of the model.

The third method called "concat" by Luong et al. (2015) but also commonly referred to as "Bahdanau," since it was introduced in the Bahdanau et al. (2015), uses a feed-forward layer to project the pair of states down to a scalar,

(Concat-Style Attention)

$$s(\mathbf{g}_i, \mathbf{h}_j) = \mathbf{k} \cdot \tanh\left(\mathbf{K}\begin{bmatrix} \mathbf{g}_i \\ \mathbf{h}_j \end{bmatrix}\right),$$

where $\mathbf{K} \in \mathbb{R}^{D_h \times (D_{\mathcal{D}} + D_{\mathcal{E}})}$ and $\mathbf{k} \in \mathbb{R}^{D_h}$ are learned parameters.

Prediction Layer Finally, the attention output $\bar{\mathbf{h}}_i$ and decoder state $\mathbf{g}_i$ are run through a two layer feed-forward network to produce a distribution over the utterance token vocabulary $\mathcal{V}_{\mathcal{Y}}$,

$$p\left(y_{i+1} | \mathbf{y}_{1:i}, \pi(\mu); \theta\right) = \operatorname{softmax}\left(\mathbf{W}^{(2)} \cdot \tanh\left(\mathbf{W}^{(1)}\begin{bmatrix} \mathbf{g}_i \\ \mathbf{h}_j \end{bmatrix} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right)_{y_{i+1}} \quad \forall i : i \in \{1, \ldots, n{-}1\}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{D_h \times (D_{\mathcal{D}} + D_{\mathcal{E}})}$, $\mathbf{b}^{(1)} \in \mathbb{R}^{D_h}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{|\mathcal{V}_{\mathcal{Y}}| \times D_h}$, and $\mathbf{b}^{(2)} \in \mathbb{R}^{|\mathcal{V}_{\mathcal{Y}}|}$ are learned parameters and we associate each utterance token y with a unique element in the final softmax distribution (similar to how we indexed into the embeddings matrices **M** and **W**).

Appendix B: Transformer-based Sequence-to-Sequence Architecture

The transformer sequence-to-sequence model eschews the recurrence as a mechanism for propagating information, and instead leans solely on several attention mechanisms to learn representations of the input sequence $\mathbf{x}$ as well as the decoder input prefix $\mathbf{y}_{1:i}$. Like the GRU, each encoder and decoder consist of L distinct layers which are applied to $\mathbf{x}$ and $\mathbf{y}_{1:i}$ respectively. Ultimately, the decoder outputs are used to compute the next word probability, $p(y_{i+1}|\mathbf{y}_{1:i}, \mathbf{x})$.

A schematic diagram of the transformer-based sequence-to-sequence model is shown in Figure B.1. Like the GRU schematic, the model diagram is color-coded to correspond with the text descriptions of each component. While the transformer looks significantly more complex than the GRU architecture, it is fundamentally built around only three different kinds of neural network layers, *(i)* multi-head attention, *(ii)* feed-forwardlayers, and *(iii)* layer normalization. Before describing the encoder and decoder layers in detail, we first describe these basic components and how they form the various "block" structures which are employed throughout the model.

B.1 Transformer Components

Multi-Head Attention The first basic layer to be defined is the multi-head attention layer. We begin by describing "single-head" attention from the point of view of a soft key-value store and then generalize to the "multi-head" case. In this view, we assume we want to attend to a sequence of m items. Each item has two representations, a key representation and a value representation, which are written collectively as rows in a key and value matrix, $\mathbf{K} \in \mathbb{R}^{m \times D_w}$ and $\mathbf{V} \in \mathbb{R}^{m \times D_w}$ respectively. We then have n query items that will each individually attend to the m items; we similarly represent the query items as rows in a matrix $\mathbf{Q} \in \mathbb{R}^{n \times D_w}$. An attention layer, denoted $\mathrm{Attn} : \mathbb{R}^{n \times D_w} \times \mathbb{R}^{m \times D_w} \times \mathbb{R}^{m \times D_w} \rightarrow \mathbb{R}^{n \times D_w}$, then computes an attention weighted read of the value

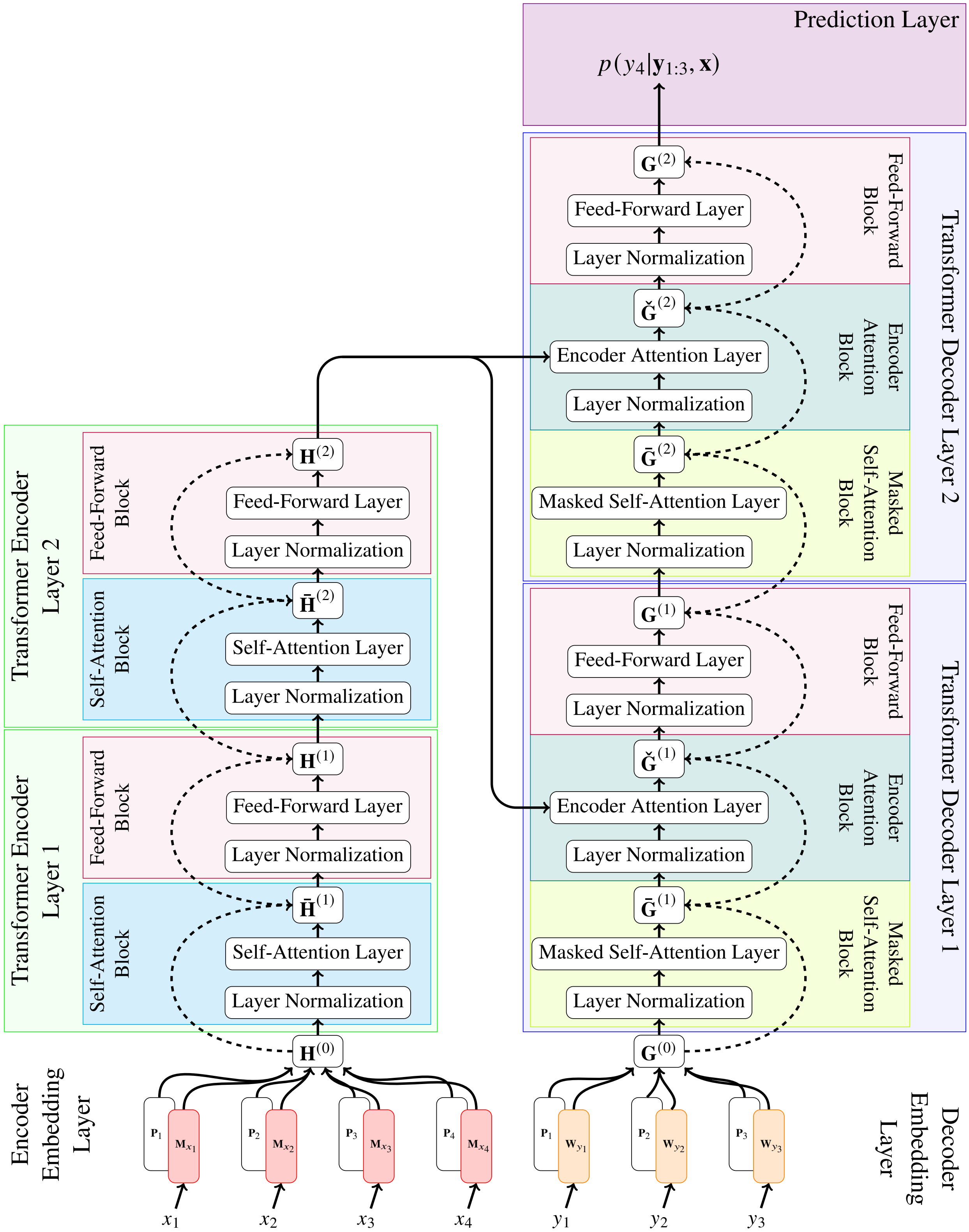


Figure B.1: A schematic diagram of a two layer transformer-based sequence-to-sequence model. Dashed lines indicate skip-connections.

matrix as,

$$\mathrm{Attn}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_w}}\right)\mathbf{V},$$

where

$$\mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_w}}\right)_{i,j} = \frac{\exp\left(D_w^{-\frac{1}{2}}\mathbf{q}_i \cdot \mathbf{k}_j\right)}{\sum_{j'=1}^{m}\exp\left(D_w^{-\frac{1}{2}}\mathbf{q}_i \cdot \mathbf{k}_{j'}\right)} \quad \forall i,j : i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}.$$

Because the m items have distinct key and value matrices, representation of similarity between a query and a key can be different than the value that is produced in the output, unlike the attention mechanisms discussed in the GRU decoder, where essentially, the key and values were identical representations.

The idea behind multi-head attention is to compute D_a distinct attention operations by first projecting the query, keys, and values down to a smaller representation. That is, given projection matrices, $\mathbf{W}^{(Q_{in},k)}, \mathbf{W}^{(K_{in},k)}, \mathbf{W}^{(V_{in},k)} \in \mathbb{R}^{D_w \times \frac{D_w}{D_a}}$ for $k \in \{1,\ldots,D_a\}$, and $\mathbf{W}^{(V_{out})} \in \mathbb{R}^{D_w \times D_w}$, the multi-headed attention layer computes

$$\mathrm{MultiHeadAttn}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \begin{pmatrix} \mathrm{Attn}\left(\mathbf{Q}\mathbf{W}^{(Q_{in},1)}, \mathbf{K}\mathbf{W}^{(K_{in},1)}, \mathbf{V}\mathbf{W}^{(V_{in},1)}\right) \\ \oplus \quad \mathrm{Attn}\left(\mathbf{Q}\mathbf{W}^{(Q_{in},2)}, \mathbf{K}\mathbf{W}^{(K_{in},2)}, \mathbf{V}\mathbf{W}^{(V_{in},2)}\right) \\ \vdots \\ \oplus \quad \mathrm{Attn}\left(\mathbf{Q}\mathbf{W}^{(Q_{in},D_a)}, \mathbf{K}\mathbf{W}^{(K_{in},D_a)}, \mathbf{V}\mathbf{W}^{(V_{in},D_a)}\right) \end{pmatrix}\mathbf{W}^{(V_{out})}$$

where $\oplus$ indicates column-wise concatenation. Each use of a multi-head attention layer uses distinct projection matrices and are learned parameters of the model.

Additionally, there is a masked variant of attention, MaskedMultiHeadAttn where the individual attention layers are computed as

$$\mathrm{Attn}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T \odot \mathbf{M}}{\sqrt{D_w}}\right)\mathbf{V}$$

where $\mathbf{M} \in \mathbb{R}^{n \times m}$ is a lower triangular matrix, i.e. values on or below the diagonal are 1 and all other values are $-\infty$. The masked multi-head attention is used for the decoder self-attention and prevents the i-th decoder step from attending to future steps, i.e. giving it clairvoyant knowledge of future tokens.

Feed-Forward Layer The next bulding block is a single hidden layer feed-forward network, $\mathrm{FF} : \mathbb{R}^{* \times D_w} \rightarrow \mathbb{R}^{* \times D_w}$, with ReLU activation in its hidden layer and no activation in its output. Let the input to the layer be a sequence of m vectors, represented as rows in a matrix $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_m] \in \mathbb{R}^{m \times D_w}$. The output of the FF layer is then computed

$$\mathrm{FF}\left(\mathbf{H}; \mathbf{W}^{(i)}, \mathbf{W}^{(j)}, \mathbf{b}^{(i)}, \mathbf{b}^{(j)}\right) = \mathrm{ReLU}\left(\mathbf{H}\mathbf{W}^{(i)} + \mathbf{b}^{(i)}\right)\mathbf{W}^{(j)} + \mathbf{b}^{(j)}.$$

where $\mathbf{W}^{(i)} \in \mathbb{R}^{D_w \times D_h}$, $\mathbf{b}^{(i)} \in \mathbb{R}^{D_h}$, $\mathbf{W}^{(j)} \in \mathbb{R}^{D_h \times D_w}$, $\mathbf{b}^{(j)} \in \mathbb{R}^{D_w}$ are learned parameters and matrix-vector additions are broadcast across the matrix rows (i.e. $\mathbf{H} + \mathbf{b} = [\mathbf{h}_1 + \mathbf{b}; \cdots \mathbf{h}_m + \mathbf{b}]$)

Layer Normalization The final component is layer normalization (Ba et al., 2016). Let $\mathbf{h} = \left[h_1, \ldots, h_{D_w}\right] \in \mathbb{R}^{D_w}$ be a vector, representing an embedding of an item with D_w features, and with mean and standard deviation

$$\bar{h} = \frac{1}{D_w} \sum_{i=1}^{D_w} h_i \quad \text{and} \quad h_\sigma = \left(\frac{1}{D_w - 1} \sum_{k=1}^{D_w} \left(h_k - \bar{h}\right)^2 + \epsilon \right)^{\frac{1}{2}},$$

respectively (the ϵ term is a small constant for numerical stability, set to 10^{-5}). Layer normalization, $\mathrm{LN} : \mathbb{R}^{D_w} \rightarrow \mathbb{R}^{D_w}$, normalizes the input have zero mean/unit variance before scaling each element and adding a bias,

$$\mathrm{LN}(\mathbf{h}; \mathbf{a}, \mathbf{b}) = \mathbf{a} \odot \left(\mathbf{h} - \bar{h}\right) \cdot h_\sigma^{-1} + \mathbf{b}$$

where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{D_w}$ are learned parameters and $\odot$ is the element-wise product. When applying layer normalization to a matrix $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_m] \in \mathbb{R}^{m \times D_w}$, where $\mathbf{H}$ is a sequence m embeddings with

D_w features, layer normalization is applied independently to each row,

$$\mathrm{LN}(\mathbf{H}; \mathbf{a}, \mathbf{b}) = \mathrm{LN}\left(\begin{bmatrix} \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_m \end{bmatrix}; \mathbf{a}, \mathbf{b}\right) = \begin{bmatrix} \mathrm{LN}\left(\mathbf{h}_1; \mathbf{a}, \mathbf{b}\right) \\ \vdots \\ \mathrm{LN}\left(\mathbf{h}_m; \mathbf{a}, \mathbf{b}\right) \end{bmatrix}.$$

B.2 Transformer Processing Blocks

Each encoder and decoder transformer layer consists of several "processing blocks" which we define now. Each processing block uses some of the basic components defined in the previous subsection. There are fourt distinct block types, a FEEDFORWARDBLOCK, a SELFATTENTIONBLOCK, a MASKEDSELFATTENTIONBLOCK, and an ENCODERATTENTIONBLOCK. Each block consists of three layers

1. Layer Normalization
2. Processing Layer
3. Skip-Connections

where the processing layer is determined by the block type. For instance, let $\mathbf{H} \in \mathbb{R}^{m \times D_w}$ be a matrix with its rows representing a sequence of m vectors; the FEEDFORWARDBLOCK is defined as

Feed-Forward Block

$$\begin{aligned} &\textit{(Layer Normalization)} && \check{\mathbf{H}} = \mathrm{LN}\left(\mathbf{H}\right) \\ &\textit{(Processing Layer)} && \bar{\mathbf{H}} = \mathrm{FF}\left(\check{\mathbf{H}}\right) \\ &\textit{(Skip-Connection)} && \hat{\mathbf{H}} = \mathbf{H} + \bar{\mathbf{H}} \end{aligned}$$

$$\mathrm{FEEDFORWARDBLOCK}(\mathbf{H}) = \hat{\mathbf{H}}.$$

The SELFATTENTIONBLOCK and MASKEDSELFATTENTIONBLOCKs are similarly defined as,

Self-Attention Block

$$\begin{aligned} \textit{(Layer Normalization)} \qquad & \check{\mathbf{H}} = \mathrm{LN}\left(\mathbf{H}\right) \\ \textit{(Processing Layer)} \qquad & \bar{\mathbf{H}} = \mathrm{MultiHeadAttn}\left(\check{\mathbf{H}}, \check{\mathbf{H}}\right) \\ \textit{(Skip-Connection)} \qquad & \hat{\mathbf{H}} = \mathbf{H} + \bar{\mathbf{H}} \end{aligned}$$

$$\textsc{SelfAttentionBlock}(\mathbf{H}) = \hat{\mathbf{H}}$$

and

Masked Self-Attention Block

$$\begin{aligned} \textit{(Layer Normalization)} \qquad & \check{\mathbf{H}} = \mathrm{LN}\left(\mathbf{H}\right) \\ \textit{(Processing Layer)} \qquad & \bar{\mathbf{H}} = \mathrm{MaskedMultiHeadAttn}\left(\check{\mathbf{H}}, \check{\mathbf{H}}\right) \\ \textit{(Skip-Connection)} \qquad & \hat{\mathbf{H}} = \mathbf{H} + \bar{\mathbf{H}} \end{aligned}$$

$$\textsc{MaskedSelfAttentionBlock}(\mathbf{H}) = \hat{\mathbf{H}}.$$

Finally, let $\mathbf{G} \in \mathbb{R}^{n \times D_w}$ be a sequence of n embeddings. The ENCODERATTENTIONBLOCK is defined as,

Encoder Attention Block

$$\begin{aligned}
&\textit{(Layer Normalization)} && \check{\mathbf{G}} = \mathrm{LN}\left(\mathbf{G}\right) \\
&\textit{(Processing Layer)} && \bar{\mathbf{G}} = \mathrm{MultiHeadAttn}\left(\check{\mathbf{G}}, \mathbf{H}\right) \\
&\textit{(Skip-Connection)} && \hat{\mathbf{G}} = \mathbf{G} + \bar{\mathbf{G}}
\end{aligned}$$

$$\textsc{EncoderAttentionBlock}(\mathbf{G}, \mathbf{H}) = \hat{\mathbf{G}}.$$

B.3 The Transformer Encoder and Decoder Layers

We now describe the actual transformer-based sequence-to-sequence model using the blocks defined previously. We start first with the encoder and decoder input layers.

Encoder Embedding Layer and Decoder Embedding Layer Let $\mathbf{x} = [x_1, \ldots, x_m]$ and $\mathbf{y} = [y_1, \ldots, y_{n-1}]$ be input and output sequences, with elements x_i and y_i drawn from vocabularies $\mathcal{V}_\mathcal{X}$ and $\mathcal{V}_\mathcal{Y}$ respectively. With each vocabulary we associate an embedding matrix, $\mathbf{M} \in \mathbb{R}^{|\mathcal{V}_\mathcal{X}| \times D_w}$ and $\mathbf{W} \in \mathbb{R}^{|\mathcal{V}_\mathcal{Y}| \times D_w}$ respectively. Let $\mathbf{M}_x \in \mathbb{R}^{D_w}$ denote the D_w-dimensional embedding for each $x \in \mathcal{V}_\mathcal{X}$; similarly, let $\mathbf{W}_y \in \mathbb{R}^{D_w}$ denote the D_w-dimensional embedding for each $y \in \mathcal{V}_\mathcal{Y}$.

Additionally let $\mathbf{P} \in \mathbb{R}^{m_{max} \times D_w}$ be a sinusoidal position embedding matrix defined elementwise with

$$\begin{aligned}
\mathbf{P}_{i,j} &= \sin\left(\frac{i}{10,000^{\frac{2(j-1)}{D_w}}}\right) && \forall i, j : i \in \{1, \ldots, m_{max}\}, j \in \{1, 3, \ldots, D_w - 1\} \\
\mathbf{P}_{i,j} &= \cos\left(\frac{i}{10,000^{\frac{2(j-1)}{D_w}}}\right) && \forall i, j : i \in \{1, \ldots, m_{max}\}, j \in \{2, 4, \ldots, D_w\}.
\end{aligned}$$

$\mathbf{P}$ is not updated during training and its rows function as an encoding of the relative position of token in the encoder/decoder inputs. The number of total positions m_{max} is a hyperparameter and represented the longest sequence the encoder/decoder can take as input.

Before being fed into the transformer encoder/decoder, each sequence is embedded in its respective embedding space and the corresponding position embeddings are added,

$$\mathbf{H}^{(0)} = \begin{bmatrix} \mathbf{M}_{x_1} + \mathbf{P}_1 \\ \vdots \\ \mathbf{M}_{x_m} + \mathbf{P}_m \end{bmatrix} \quad \text{and} \quad \mathbf{G}^{(0)} = \begin{bmatrix} \mathbf{W}_{y_1} + \mathbf{P}_1 \\ \vdots \\ \mathbf{W}_{y_{n-1}} + \mathbf{P}_{n-1} \end{bmatrix}.$$

Transformer Encoder A transformer encoder layer consists of a SELFATTENTIONBLOCK followed by a FEEDFORWARDBLOCK. A transformer encoder with L layers then computes

$$\begin{aligned} \bar{\mathbf{H}}^{(l)} &= \textsc{SelfAttentionBlock}^{(l)}\left(\mathbf{H}^{(l-1)}\right) \\ \mathbf{H}^{(l)} &= \textsc{FeedForwardBlock}^{(l)}\left(\bar{\mathbf{H}}^{(l)}\right) \end{aligned}$$

for $l \in \{1, \ldots, L\}$. We indicate the final encoder output as $\mathbf{H} = \mathbf{H}^{(L)}$.

Transformer Decoder A transformer deccoder layer consists of a MASKEDSELFATTENTIONBLOCK followed by an ENCODERATTENTIONBLOCK and a FEEDFORWARDBLOCK. Note that the MASKEDSELFATTENTIONBLOCK means that though we can compute all decoder states in parallel during training, it is equivalent to computing each decoder state sequentially (which we must do at test time). A transformer deccoder with L layers is computed as

$$\begin{aligned} \check{\mathbf{G}}^{(l)} &= \textsc{MaskedSelfAttentionBlock}^{(l)}\left(\mathbf{G}^{(l-1)}\right) \\ \bar{\mathbf{G}}^{(l)} &= \textsc{EncoderAttentionBlock}^{(l)}\left(\check{\mathbf{G}}^{(l)}, \mathbf{H}\right) \\ \mathbf{G}^{(l)} &= \textsc{FeedForwardBlock}^{(l)}\left(\bar{\mathbf{G}}^{(l)}\right) \end{aligned}$$

for $l \in \{1, \ldots, L\}$. We indicate the final decoder output as $\mathbf{G} = \mathbf{G}^{(L)}$.

Prediction Layer Let $\mathbf{g}_i$ be the i-th row of $\mathbf{G}$ corresponding to the decoder representation of the i-th decoder state. The probability of the next word is

$$p\left(y_{i+1}|\mathbf{y}_{1:i},\mathbf{x}\right) = \text{softmax}\left(\mathbf{W}^{(o)}\mathbf{g}_i + \mathbf{b}^{(o)}\right)_{y_{i+1}} \qquad \forall i : i \in \{1, \ldots, n-1\}$$

where $\mathbf{W}^{(o)} \in \mathbb{R}^{|\mathcal{V}_\mathcal{Y}| \times D_w}$ and $\mathbf{b}^{(o)} \in \mathbb{R}^{D_w}$ are learned parameters. Each block from each encoder and decoder layer has separate learned parameters. Because each operation in the transformer is built around matrix multiplication, its computation can be parallelized more heavily than recurrent neural network models like the GRU architecture.