AN ABSTRACT OF THE DISSERTATION OF

MohammadReza Ghaeini for the degree of Doctor of Philosophy in Computer Science presented on February 04, 2020.

Title: Improving and Understanding Deep Models for Natural Language Comprehension

Abstract approved: ____

Xiaoli Z. Fern

Natural Language Comprehension is a challenging domain of Natural Language Processing. To improve a model's language comprehension/understanding, one approach would be to enrich the structure of the model to enhance its capability in learning the latent rules of the language.

In this dissertation, we will first introduce several deep models for a variety of natural language comprehension tasks including natural language inference and question answering. Previous approaches employ reading mechanisms that do not fully exploit the interdependencies between the input sources like "premise and hypothesis" or "document and query". In contrast, we explore more sophisticated reading mechanisms to efficiently model the relationships between input sources (e.g. "premise and hypothesis" or "document and query"). These mechanisms and models yield better empirical performances, however, due to the black-box nature of deep learning, it is difficult to assess whether the improved models indeed acquire a better understanding of language. Meanwhile, data is often plagued by meaningless or even harmful statistical biases and deep models might achieve high performance by focusing on the biases. This motivates us to study methods for "peaking inside" the black-box deep models to provide explanation and understanding of the models' behavior. The proposed method (a.k.a. saliency) takes a step toward explaining deep learning-based models based on gradient of the model output with respect to different components like the input layer and intermediate layers. Saliency reveals interesting insights and identifies critical information contributing to the model decisions. Besides proposing a model-agnostic interpretation method (saliency), we study model-dependent interpretation solutions and propose two interpretable designs and structures. Finally, we introduce a novel mechanism (saliency learning), which learns from ground-truth explanation signal such that the learned model will not only make the right prediction but also for the right reason. Our experimental results on multiple tasks and datasets demonstrate the effectiveness of the proposed methods, which produce more faithful to right reasons and evidences predictions while delivering better results compared to traditionally trained models.

Improving and Understanding Deep Models for Natural Language Comprehension

by

MohammadReza Ghaeini

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented February 04, 2020
Commencement June 2020

Doctor of Philosophy dissertation of MohammadReza Ghaeini presented on February 04, 2020.

APPROVED:

Major Professor, representing Computer Science

Head of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

MohammadReza Ghaeini, Author

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere appreciation and gratitude to my dear advisor, Xiaoli Fern for all her help and guidance. Over the past five years, I have always counted on her flawless advisory feedback and I will continue to learn from her. She always has been so supportive and genuinely helpful through all my PhD program.

I also would like to thank my PhD committee members, Dr. Prasad Tadepalli, Dr. Liang Huang, Dr. David Hendrix, and Dr. Kipp Shearman, for all of their invaluable feedback, support, and for selflessly agreeing to serve on my committee.

During my years working at Oregon State University, it has been a true pleasure to be colleagues and office mates with Hamed Shahbazi, Medisa Danaee, Walker Orr, and Chao Ma. Especially, I am truly appreciative of my dear friend Hamed Shahbazi for all helpful discussions, suggestion, and collaboration.

Friendship is one of the most important aspects of my life and I could not make it here without my beloved friends who are like my family here. So, I am very thankful to each and every one of them. I am happy to have every one of you by my side on this journey. In particular, I would like to name a few of them here as a token of appreciation for their friendship: Elham Mirkoohi, Shahrokh Shahi, Shiva Keyvanfar, Farzad Zafarani, Lily Ranjbar, Ali Jafarnejad, Mehran Amiri, Shiva Bahrami, Meysam Nezafati, Arash Moradi, Amir Afsharinejad, Forough Khonsari, Mahtab Aboufazeli, Amir Azarbakht, and Parisa Ataei.

Finally, I must express my very profound gratitude to my parents and my sister for providing me with unfailing support and continuous encouragement throughout my years of study. Especially, my dear father who I, unfortunately, lost during my years of study. He will be truly missed but his memory will always be in my heart. This accomplishment would not have been possible without them and their sacrifices.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

Chapter 1: Introduction

In this dissertation, we focus on improving and understanding deep learning-based models for Natural Language Comprehension (NLC). Figure 1.1 demonstrates the chain of thoughts that form this work. Natural Language Comprehension is a central problem of Natural Language Processing. NLC is the key factor to obtain good and reliable performance in a variety of NLP tasks. One way to improve natural language comprehension and understanding would be enriching the structure of the model to enhance its capability to learn the latent rules of the language. The way we handle and encode input sources critically influence the model's behavior and performance. If the model misses important information and details in the input encoding stage, the rest of the model could not recover the missing information and clues.

The first part of this dissertation focuses on proposing more sophisticated reading mechanisms to efficiently model the relationship and dependency between input sources. Here we start our study with the Natural Language Inference task (Chapter 3). Natural Language Inference (NLI; a.k.a. Recognizing Textual Entailment, or RTE) is an important and challenging task for natural language understanding [59]. The goal of NLI is to identify the logical relationship (*entailment*, *neutral*, or *contradiction*) between a premise and a corresponding hypothesis. Existing approaches mostly rely on simple reading mechanisms for independent encoding of the premise and hypothesis. Instead, we propose a novel dependent reading bidirectional LSTM network (DR-BiLSTM) to

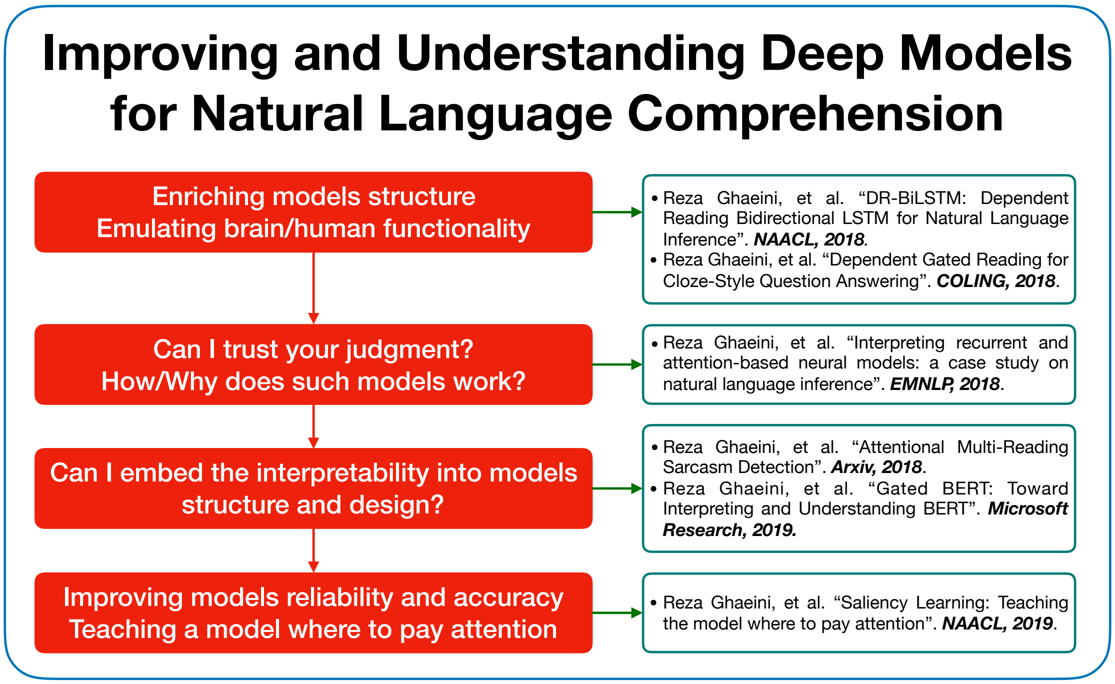


Figure 1.1: Research and dissertation chain of thoughts.

efficiently model the relationship between a premise and a hypothesis during encoding and inference. Our evaluation shows that DR-BiLSTM achieves the new state-of-the-art scores on the Stanford NLI dataset.

In Chapter 4 we test the observed positive impact of using more sophisticated reading mechanisms on a different task. Cloze-Style question answering could be considered as one of the tasks that essentially studies Human Language Comprehension. It requires semantic understanding and reasoning over clues. The goal of this task is to read and comprehend the given document and answer queries. Previous works employ reading mechanisms that do not fully exploit the interdependency between the document and the query. In Chapter 4, we propose a novel *dependent gated reading* bidirectional GRU network (DGR) to efficiently model the relationship between the document and the query during encoding and decision making. Our evaluation shows that DGR obtains highly competitive performance on well-known machine comprehension benchmarks such as the Children's Book Test (CBT-NE and CBT-CN) and Who DiD What (WDW, Strict and Relaxed).

Up until this point, we showed that the aforementioned mechanisms and models yield better empirical performances. However, due to the black-box nature of deep learning, it is difficult to assess whether the improved models indeed acquire a better language understanding. Meanwhile, data is often plagued by meaningless or even harmful statistical biases and deep models might achieve high performance by focusing on the biases [1, 94, 32, 45]. This motivates us to study methods for "peaking inside" the black-box deep models to provide explanation and understanding of the models' behavior. In the second part of this dissertation, we focus on interpretability and understanding the model's behavior and performance. In Chapter 5, we take a step toward explaining deep learning based models through a case study on a popular neural model for NLI. In particular, we propose to interpret the intermediate layers of NLI models by visualizing the saliency of attention and LSTM gating signals. We present several examples for which our methods reveal interesting insights and identify the critical information contributing to the model's decisions.

Interpretability could also be implanted in the model structure and design (model-dependent interpretability). In Chapter 6, we aim to propose an interpretable model for Sarcasm Detection. Recognizing sarcasm often requires a deep understanding of multiple sources of information, including the utterance, the conversational context, and real-world facts. Most of the current sarcasm detection systems consider only the utterance in isolation. There are some limited attempts toward taking into account the conversational context. In this work, we propose an interpretable end-to-end model that combines information from both the utterance and the conversational context to detect sarcasm, and demonstrate its effectiveness through empirical evaluations. We also study the behavior of the proposed model to provide explanations for the model's decisions. Importantly, our model is capable of determining the impact of utterance and conversational context on the model's decisions. A similar goal is pursued in the work described in Chapter 7, in which we propose an interesting yet simple modification to a well-known and widely-used model called BERT [18]. Our modification (we refer to the modified BERT as Gated BERT) introduces interpretability features to this model. In addition to shedding a light on behavior, role, and impact of different layers of BERT for disambiguation and decision making of different tasks, the proposed modification also yields better performance both with and without fine-tuning of the embedding and transformer layer. We evaluate the Gated BERT and BERT on a variety of NLP tasks using GLUE benchmarks [97]. Moreover, we provide a demo of this work that provides a wide range of features for understanding and studying the behavior of Gated BERT and BERT.

Aforementioned methods are helpful for understanding model's behavior and assessing the reliability of the model's predictions. But, such methods do not fix and improve the model's reliability. In Chapter 8, we propose a method to teach the model to make the right prediction for the right reason by providing explanation training signal and ensuring the alignment of the model's explanation with the ground truth explanation.

Our experimental results on multiple tasks and datasets demonstrate the effectiveness of the proposed method, which produces more reliable predictions while delivering better results compared to traditionally trained models.

Chapter 2: Background and Preliminaries

2.1 Artificial Neural Networks

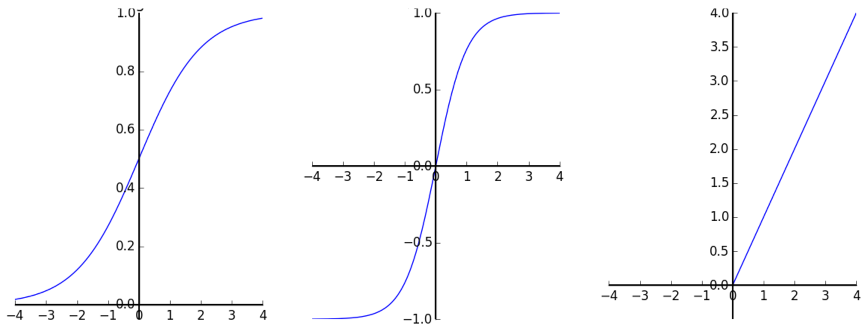
Artificial neural networks (ANNs) are a class of artificial intelligence and machine learning models which intent to mathematically model human brain. ANNs model potentially a very complex function by connecting a series of layers each of which is a linear transformation followed by an element-wise non-linearity like sigmoid or tanh. Aforementioned non-linearity is called *activation function* which is inspired by activation behavior of biological neurons.

Usually, in neural networks the goal is to learn a set of wight matrices Ws and bias terms bs. The output of a simple layer in a neural network with non-linear activation function f, and input vector x is defined as:

$$y = f(Wx + b) \tag{2.1}$$

There are different choices for activation function in a neural network. Sigmoid ($\sigma(x)$), hyperbolic tangent(*tanh*(x)), and rectifier function (*ReLU*(x)) are the most common activation functions for neural networks, which have different behaviors and characteristics. All non-linear activation functions are shown in Figure 2.1.

The most basic form of such a neural network is the multilayer perceptron classifier (*MLP*), which is shown in Figure 2.2. MLP consists of an input layer, one or more



$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \qquad ReLU(x) = \max(0, x)$$

Figure 2.1: Three most commonly used non-linear activation functions in neural networks

fully-connected hidden layers, and finally a prediction layer at the top, which uses Softmax (Equation 2.2) to compute the probability of each class given the input, x. In this layer an error function is minimized. Standard examples for this error function are the cross-entropy error function for classification and the least squares error function for regression.

$$P(y = i \mid x) = \frac{e^{x^T w_i}}{\sum_{k=1}^{n} e^{x^T w_k}} \tag{2.2}$$

The most common application of the neural network is classification, where the goal is to learn model to classify the given input to specific classes. In such a case, the parameters of the network (set of weights and biases) are learned through back-propagated gradient descent to minimize a loss function. This procedure is called back-propagation.

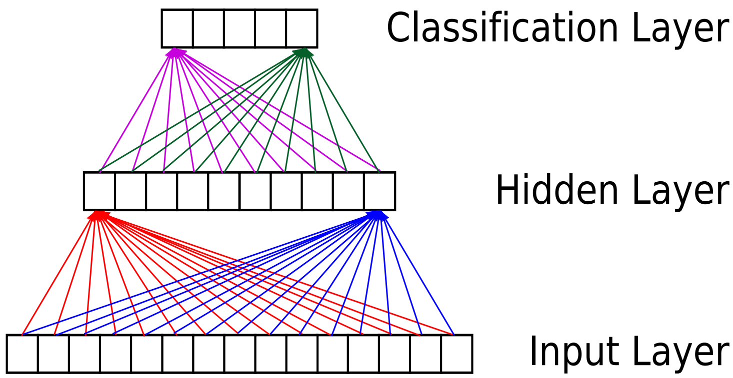


Figure 2.2: Representation of a basic multilayer perceptron classifier (MLP) with one hidden layer and multiple class classification.

2.2 Word Embedding

Typically, neural networks work with dense fixed-dimension data vectors from a continuous feature set. But natural language processing tasks typically involve discrete features, such as words, n-grams or co-occurrence of words. In machine learning, such features are often represented by sparse vectors like binary-valued (e.g. one-hot representation for words) or count-valued vectors with very high dimensionality. Features of this type are typically unsuitable for neural networks because of their inherent sparsity, which makes the learning intractable, furthermore, they also make the network unnecessarily complex in term of the dimensionality. The point is that a well-designed relatively low-dimensional continuous representation of such discrete feature could encode various relations and similarities between existing discrete entities. Moreover, neural networks perform much better with this type of input since neural network can extract complex relations due to their high non-linearity.

The aforementioned advantages lead NLP community to develop various models and structures that extract and learn such relatively low-dimensional vector to represent words, which are called word embeddings in the literature. There are many powerful word embeddings that encode different aspects of words. A common way to learn word embedding is to learn them in an unsupervised manner and rely on co-occurrence of words in a very large set of valid sentences. Bengio et al. (2003), Collobert and Weston (2008), Word2Vector method (2013), Mikolov et al. (2013), and the GloVe method (2014) are examples of successful and powerful word embeddings [6, 14, 63, 64, 70]. Mikolov et al. use word embeddings to encode concept of the words in a way such that we can remove or add a characteristic (like sex) of words in order to reach the new word with desired characteristic. For instance, $W_{King} - W_{Man} + W_{Woman} \approx W_{Queen}$.

In NLP tasks, word embeddings instead of one-hot vectors, are fed to the network at the input layer. Also, word embeddings typically are treated as additional parameters to the network. In such cases, a projection layer transfers indices of the words into their word embeddings. The pre-initialized word embeddings will be updated during the training to reach a task specific word embedding. We can initialize word embedding with learned word embeddings like word2vec or initialize them randomly. This procedure is called *fine tuning*. Fine tuning usually is an essential step in the training of a network since we may need to pay attention to different aspect of words for different tasks. For example, in a parsing task, words “good” and “bad” should have similar embeddings because if we replace “good” with “bad”, the parse should be the same. However, in a task like sentiment analysis, words “good” and “bad” should have very different embeddings since they might change the final sentiment of a sentence. As such, we cannot have an universal word embedding that performs appropriately in all

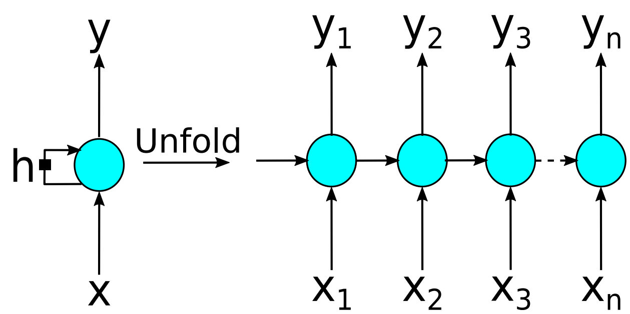


Figure 2.3: Structure of a Recurrent Neural Network (RNN). Folded and unfolded representations respectively

NLP tasks and we need to update them during the training to reach task specific word embeddings.

Finally, we should note that continuous vector representation can be learned for other discrete features, such as part-of-speech tags, name-entity tags, etc too. In this settings, the representations are initialized randomly and then learned during the training procedure.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) represent a class of neural networks that handle sequential data with variable sizes. In a recurrent network, a shared network architecture is applied repeatedly to a sequence of data with a history at each step being produced by the previous time step. The main idea for the recurrent neural network is to process the data in a sequential manner and remember important aspect of the data over time considering the whole sequence. Recurrent networks can be considered as the most essential deep learning architecture for natural language processing, because natural languages

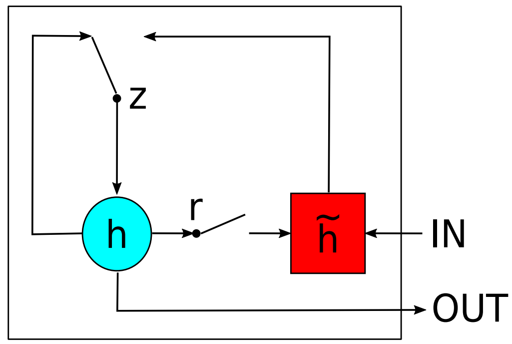


Figure 2.4: Structure of Gated Recurrent Unit (GRU)

are sequential in nature with variable sizes.

The general structure of a recurrent neural network (folded and unfolded representation) can be seen in Figure 2.3. Also, the computation formula of the simple RNN (Vanilla RNN) at time step t is as follows:

$$h_t = f(Wx_t + Uh_{t-1} + b) \tag{2.3}$$

where W, U are the weights, b is the bias, f stands for the activation function, x_t and h_t are the RNN input and output at time step t respectively and h_{t-1} is the previous hidden state.

2.3.1 Gated Recurrent Unit

Gated Recurrent Unit (*GRU*) is an extension of recurrent neural networks which uses gating in order to determine how much of the current input should influence the hidden state and how much of the previous hidden state (h_{t-1}) should be remembered or

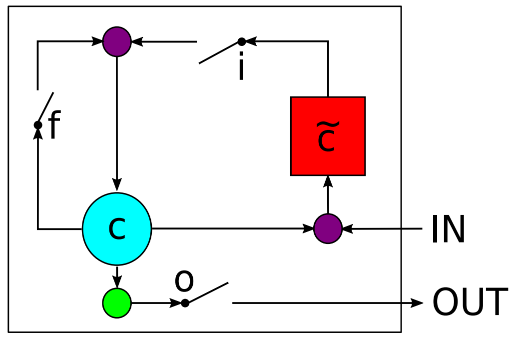


Figure 2.5: Structure of Long Short-Term Memory (LSTM)

forgotten. Figure 2.4 depicts the structure of GRU and also Equation 2.4 shows the computation of GRU.

$$
\begin{aligned}
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
\tilde{h}_t &= f(W x_t + U(r_t \odot h_{t-1})) \\
h_t &= (1 - z_t) h_{t-1} + z_t \tilde{h}_t
\end{aligned}
\tag{2.4}
$$

where Us and Ws are set of weights, bs are set of biases, $\odot$ is the element-wise product, σ is the sigmoid function, f is the activation of GRU and r_t, z_t, h_t, $\tilde{h}_t$ and x_t stand for the reset gate, update gate, hidden state, candidate hidden state and the GRU input at time step t respectively.

2.3.2 Long Short-Term Memory

Long Short-Term Memory (*LSTM*) is an extension of recurrent neural networks which uses gaiting in order to determine how much of the current input should influence LSTM memory cell, how much of the previous memory cell (C_{t-1}) should be remembered or forgotten and how much of the current memory cell should be passed as the output of the LSTM. Figure 2.5 illustrates the structure of LSTM and also Equation 2.5 presents the computation of LSTM.

$$\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
\tilde{C}_t &= f(W_c x_t + U_c h_{t-1} + b_c) \\
C_t &= i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o) \\
h_t &= o_t \odot f(C_t)
\end{aligned} \tag{2.5}$$

where V_o, Us and Ws are set of weights, bs are set of biases, $\odot$ is the element-wise product, σ is the sigmoid function, f is the activation of LSTM and i_t, f_t, o_t, h_t, C_t $\tilde{C}_t$ and x_t stand for input gate, forget gate, output gate, LSTM output, memory cell, candidate memory cell and LSTM input at time step t respectively.

Chapter 3: DR-BiLSTM: Dependent Reading Bidirectional LSTM for NLI

This chapter describes the work in Ghaeini et al. (2018a) [26].

3.1 Introduction

Natural Language Inference (NLI; a.k.a. Recognizing Textual Entailment, or RTE) is an important and challenging task for natural language understanding [59]. The goal of NLI is to identify the logical relationship (*entailment*, *neutral*, or *contradiction*) between a premise and a corresponding hypothesis. Table 3.1 shows few example relationships from the Stanford Natural Language Inference (SNLI) dataset [7].

Recently, NLI has received a lot of attention from the researchers, especially due to the availability of large annotated datasets like SNLI [7]. Various deep learning models have been proposed that achieve successful results for this task [30, 99, 11, 103, 69, 106, 82]. Most of these existing NLI models use attention mechanism to jointly interpret and align the premise and hypothesis. Such models use simple reading mechanisms to encode the premise and hypothesis independently. However, such a complex task require explicit modeling of dependency relationships between the premise and the hypothesis during the encoding and inference processes to prevent the network from the loss of relevant, contextual information. In this work, we refer to such strategies as *dependent reading*.

P[a]	A senior is waiting at the window of a restaurant that serves sandwiches.	Relationship
H[b]	A person waits to be served his food.	Entailment
	A man is looking to order a grilled cheese sandwich.	Neutral
	A man is waiting in line for the bus.	Contradiction

[a]**P**, Premise.
[b]**H**, Hypothesis.

Table 3.1: Examples from the SNLI dataset.

There are some alternative reading mechanisms available in the literature [82, 80] that consider dependency aspects of the premise-hypothesis relationships. However, these mechanisms have two major limitations:

- So far, they have only explored dependency aspects during the encoding stage, while ignoring its benefit during inference.
- Such models only consider encoding a hypothesis depending on the premise, disregarding the dependency aspects in the opposite direction.

We propose a dependent reading bidirectional LSTM (DR-BiLSTM) model to address these limitations. Given a premise u and a hypothesis v, our model first encodes them considering dependency on each other ($u|v$ and $v|u$). Next, the model employs a soft attention mechanism to extract relevant information from these encodings. The augmented sentence representations are then passed to the inference stage, which uses a similar dependent reading strategy in both directions, i.e. $u \rightarrow v$ and $v \rightarrow u$. Finally, a decision is made through a multi-layer perceptron (MLP) based on the aggregated information.

Our experiments on the SNLI dataset show that DR-BiLSTM achieves the best single model and ensemble model performance obtaining improvements of a considerable margin of 0.4% and 0.3% over the previous state-of-the-art single and ensemble models, respectively.

Furthermore, we demonstrate the importance of a simple preprocessing step performed on the SNLI dataset. Evaluation results show that such preprocessing allows our single model to achieve the same accuracy as the state-of-the-art ensemble model and improves our ensemble model to outperform the state-of-the-art ensemble model by a remarkable margin of 0.7%. Finally, we perform an extensive analysis to clarify the strengths and weaknesses of our models.

3.2 Related Work

Early studies use small datasets while leveraging lexical and syntactic features for NLI [59]. The recent availability of large-scale annotated datasets [7, 101] has enabled researchers to develop various deep learning-based architectures for NLI.

Parikh et al. (2016) propose an attention-based model [4] that decomposes the NLI task into sub-problems to solve them in parallel. They further show the benefit of adding intra-sentence attention to input representations. [11] explore sequential inference models based on chain LSTMs with attentional input encoding and demonstrate the effectiveness of syntactic information. We also use similar attention mechanisms. However, our model is distinct from these models as they do not benefit from dependent reading strategies.

Rocktaschel et al. (2015) use a word-by-word neural attention mechanism while [82] propose re-read LSTM units by considering the dependency of a hypothesis on the information of its premise ($v|u$) to achieve promising results. However, these models suffer from weak inferencing methods by disregarding the dependency aspects from the opposite direction ($u|v$). Intuitively, when a human judges a premise-hypothesis relationship, s/he might consider back-and-forth reading of both sentences before coming to a conclusion. Therefore, it is essential to encode the premise-hypothesis dependency relations from both directions to optimize the understanding of their relationship.

Wang et al. (2017) propose a bilateral multi-perspective matching (BiMPM) model, which resembles the concept of matching a premise and hypothesis from both directions. Their matching strategy is essentially similar to our attention mechanism that utilizes relevant information from the other sentence for each word sequence. They use similar methods as [11] for encoding and inference, without any dependent reading mechanism.

Although NLI is well studied in the literature, the potential of dependent reading and interaction between a premise and hypothesis is not rigorously explored. In this work, we address this gap by proposing a novel deep learning model (DR-BiLSTM). Experimental results demonstrate the effectiveness of our model.

3.3 Model

Our proposed model (DR-BiLSTM) is composed of the following major components: input encoding, attention, inference, and classification. Figure 3.1 demonstrates a high-level view of our proposed NLI framework.

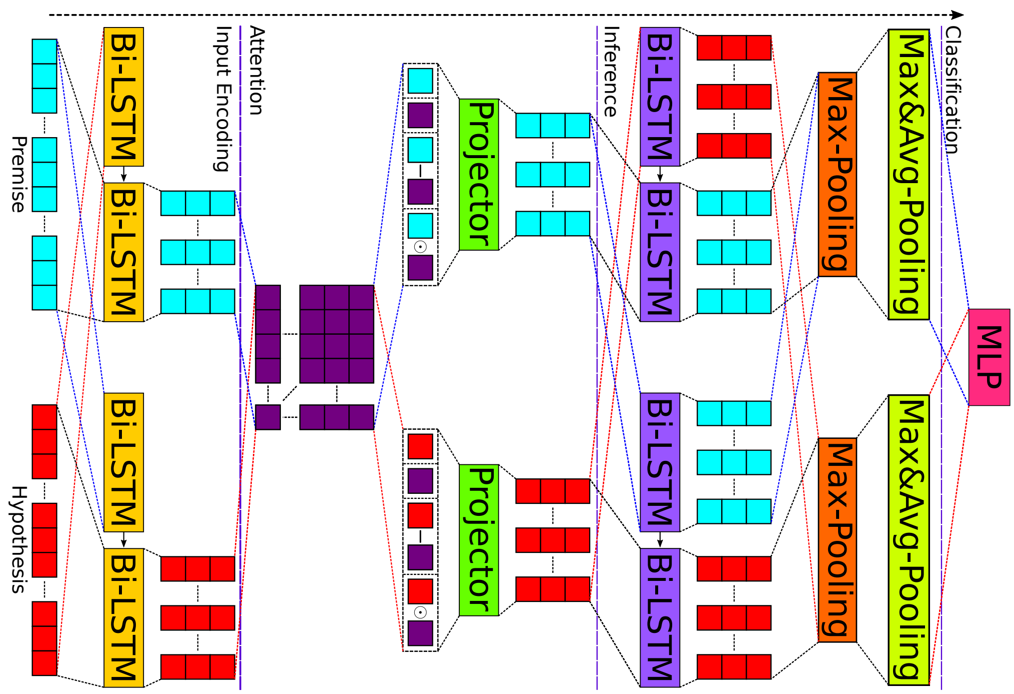


Figure 3.1: A high-level view of DR-BiLSTM. The data (premise u and hypothesis v, depicted with cyan and red tensors respectively) flows from bottom to top. Relevant tensors are shown with the same color and elements with the same colors share parameters.

Let $u = [u_1, \cdots, u_n]$ and $v = [v_1, \cdots, v_m]$ be the given premise with length n and hypothesis with length m respectively, where $u_i, v_j \in \mathbb{R}^r$ is an word embedding of r-dimensional vector. The task is to predict a label y that indicates the logical relationship between premise u and hypothesis v.

3.3.1 Input Encoding

RNNs are the natural solution for variable length sequence modeling, consequently, we utilize a bidirectional LSTM (BiLSTM) [39] for encoding the given sentences. For ease of presentation, we only describe how we encode u depending on v. The same procedure is utilized for the reverse direction $(v|u)$.

To dependently encode u, we first process v using the BiLSTM. Then we read u through the BiLSTM that is initialized with previous reading final states (memory cell and hidden state). Here we represent a word (e.g. u_i) and its context depending on the other sentence (e.g. v). Equations 3.1 and 3.2 formally represent this component.

$$
\begin{aligned}
\bar{v}, s_v &= \textit{BiLSTM}(v, 0) \\
\hat{u}, - &= \textit{BiLSTM}(u, s_v)
\end{aligned}
\tag{3.1}
$$

$$
\begin{aligned}
\bar{u}, s_u &= \textit{BiLSTM}(u, 0) \\
\hat{v}, - &= \textit{BiLSTM}(v, s_u)
\end{aligned}
\tag{3.2}
$$

where $\{\bar{u} \in \mathbb{R}^{n \times 2d}, \hat{u} \in \mathbb{R}^{n \times 2d}, s_u\}$ and $\{\bar{v} \in \mathbb{R}^{m \times 2d}, \hat{v} \in \mathbb{R}^{m \times 2d}, s_v\}$ are the independent reading sequences, dependent reading sequences, and BiLSTM final state of independent reading of u and v respectively. Note that, "$-$" in these equations means that we do not care about the associated variable and its value. BiLSTM inputs are the word embedding sequences and initial state vectors. $\hat{u}$ and $\hat{v}$ are passed to the next layer as the output of the input encoding component.

The proposed encoding mechanism yields a richer representation for both premise and hypothesis by taking the history of each other into account. Using a max or average pooling over the independent and dependent readings does not further improve our model. This was expected since dependent reading produces more promising and relevant encodings.

3.3.2 Attention

We employ a soft alignment method to associate the relevant sub-components between the given premise and hypothesis. In deep learning models, such purpose is often achieved with a soft attention mechanism. Here we compute the unnormalized attention weights as the similarity of hidden states of the premise and hypothesis with Equation 3.3 (energy function).

$$e_{ij} = \hat{u}_i \hat{v}_j^T, \quad i \in [1, n], j \in [1, m] \tag{3.3}$$

where $\hat{u}_i$ and $\hat{v}_j$ are the dependent reading hidden representations of u and v respectively which are computed earlier in Equations 3.1 and 3.2. Next, for each word in either premise or hypothesis, the relevant semantics in the other sentence is extracted and composed according to e_{ij}. Equations 3.4 and 3.5 provide formal and specific details of this procedure.

$$\tilde{u}_i = \sum_{j=1}^{m} \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{ik})} \hat{v}_j, \quad i \in [1, n] \tag{3.4}$$

$$\tilde{v}_j = \sum_{i=1}^{n} \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{kj})} \hat{u}_i, \quad j \in [1, m] \tag{3.5}$$

where $\tilde{u}_i$ represents the extracted relevant information of $\hat{v}$ by attending to $\hat{u}_i$ while $\tilde{v}_j$

represents the extracted relevant information of $\hat{u}$ by attending to $\hat{v}_j$.

To further enrich the collected attentional information, a trivial next step would be to pass the concatenation of the tuples $(\hat{u}_i, \tilde{u}_i)$ or $(\hat{v}_j, \tilde{v}_j)$ which provides a linear relationship between them. However, the model would suffer from the absence of *similarity* and *closeness* measures. Therefore, we calculate the difference and element-wise product for the tuples $(\hat{u}_i, \tilde{u}_i)$ and $(\hat{v}_j, \tilde{v}_j)$ that represent the similarity and closeness information respectively [11, 50].

The difference and element-wise product are then concatenated with the computed vectors, $(\hat{u}_i, \tilde{u}_i)$ or $(\hat{v}_j, \tilde{v}_j)$, respectively. Finally, a feedforward neural layer with ReLU activation function projects the concatenated vectors from $8d$-dimensional vector space into a d-dimensional vector space (Equations 3.6 and 3.7). This helps the model to capture deeper dependencies between the sentences besides lowering the complexity of vector representations.

$$\begin{aligned} a_i &= [\hat{u}_i, \tilde{u}_i, \hat{u}_i - \tilde{u}_i, \hat{u}_i \odot \tilde{u}_i] \\ p_i &= \boldsymbol{ReLU}(W_p a_i + b_p) \end{aligned} \tag{3.6}$$

$$\begin{aligned} b_j &= [\hat{v}_j, \tilde{v}_j, \hat{v}_j - \tilde{v}_j, \hat{v}_j \odot \tilde{v}_j] \\ q_j &= \boldsymbol{ReLU}(W_p b_j + b_p) \end{aligned} \tag{3.7}$$

Here $\odot$ stands for element-wise product while $W_p \in \mathbb{R}^{8d \times d}$ and $b_p \in \mathbb{R}^d$ are the trainable weights and biases of the projector layer respectively.

3.3.3 Inference

During this phase, we use another BiLSTM to aggregate the two sequences of computed matching vectors, p and q from the attention stage (Section 3.3.2). This aggregation is performed in a sequential manner to avoid losing effect of latent variables that might rely on the sequence of matching vectors.

Instead of aggregating the sequences of matching vectors individually, we propose a similar dependent reading approach for the inference stage. We employ a BiLSTM reading process (Equations 3.8 and 3.9) similar to the input encoding step discussed in Section 3.3.1. But rather than passing just the dependent reading information to the next step, we feed both independent reading ($\bar{p}$ and $\bar{q}$) and dependent reading ($\hat{p}$ and $\hat{q}$) to a max pooling layer, which selects maximum values from each sequence of independent and dependent readings ($\bar{p}_i$ and $\hat{p}_i$) as shown in Equations 3.10 and 3.11. The main intuition behind this architecture is to maximize the inferencing ability of the model by considering both independent and dependent readings.

$$\begin{aligned} \bar{q}, s_q &= \textit{BiLSTM}(q, 0) \\ \hat{p}, - &= \textit{BiLSTM}(p, s_q) \end{aligned} \tag{3.8}$$

$$\begin{aligned} \bar{p}, s_p &= \textit{BiLSTM}(p, 0) \\ \hat{q}, - &= \textit{BiLSTM}(q, s_p) \end{aligned} \tag{3.9}$$

$$\tilde{p} = \textit{MaxPooling}(\bar{p}, \hat{p}) \tag{3.10}$$

$$\tilde{q} = \textit{MaxPooling}(\bar{q}, \hat{q}) \tag{3.11}$$

Here $\{\bar{p} \in \mathbb{R}^{n\times 2d}, \hat{p} \in \mathbb{R}^{n\times 2d}, s_p\}$ and $\{\bar{q} \in \mathbb{R}^{m\times 2d}, \hat{q} \in \mathbb{R}^{m\times 2d}, s_q\}$ are the independent reading sequences, dependent reading sequences, and BiLSTM final state of independent reading of p and q respectively. BiLSTM inputs are the word embedding sequences and initial state vectors.

Finally, we convert $\tilde{p} \in \mathbb{R}^{n\times 2d}$ and $\tilde{q} \in \mathbb{R}^{m\times 2d}$ to fixed-length vectors with pooling, $U \in \mathbb{R}^{4d}$ and $V \in \mathbb{R}^{4d}$. As shown in Equations 3.12 and 3.13, we employ both max and average pooling and describe the overall inference relationship with concatenation of their outputs.

$$U = [\textit{MaxPooling}(\tilde{p}), \textit{AvgPooling}(\tilde{p})] \tag{3.12}$$

$$V = [\textit{MaxPooling}(\tilde{q}), \textit{AvgPooling}(\tilde{q})] \tag{3.13}$$

3.3.4 Classification

Here, we feed the concatenation of U and V ($[U, V]$) into a multilayer perceptron (MLP) classifier that includes a hidden layer with *tanh* activation and *softmax* output layer. The model is trained in an end-to-end manner.

$$\textit{Output} = \textit{MLP}([U, V]) \tag{3.14}$$

3.4 Experiments and Evaluation

3.4.1 Dataset

The Stanford Natural Language Inference (SNLI) dataset contains $570K$ human annotated sentence pairs. The premises are drawn from the Flickr30k [72] corpus, and then the hypotheses are manually composed for each relationship class (*entailment*, *neutral*, *contradiction*, and -). The "-" class indicates that there is no consensus decision among the annotators, consequently, we remove them during the training and evaluation following the literature. We use the same data split as provided in [7] to report comparable results with other models.

3.4.2 Experimental Setup

We use pre-trained 300-D Glove $840B$ vectors [70] to initialize our word embedding vectors. All hidden states of BiLSTMs during input encoding and inference have 450 dimensions ($r = 300$ and $d = 450$). The weights are learned by minimizing the log-loss on the training data via the Adam optimizer [48]. The initial learning rate is 0.0004. To avoid overfitting, we use dropout [86] with the rate of 0.4 for regularization, which is applied to all feedforward connections. During training, the word embeddings are updated to learn effective representations for the NLI task. We use a fairly small batch size of 32 to provide more exploration power to the model. Our observation indicates that using larger batch sizes hurts the performance of our model.

3.4.3 Ensemble Strategy

Ensemble methods use multiple models to obtain better predictive performance. Previous works typically utilize trivial ensemble strategies by either using majority votes or averaging the probability distributions over the same model with different initialization seeds [99, 30].

By contrast, we use weighted averaging of the probability distributions where the weight of each model is learned through its performance on the SNLI development set. Furthermore, the differences between our models in the ensemble originate from: 1) variations in the number of dependent readings (i.e. 1 and 3 rounds of dependent reading), 2) projection layer activation (*tanh* and *ReLU* in Equations 3.6 and 3.7), and 3) different initialization seeds. To be exact, we use the following configurations in our ensemble model study:

- DR-BiLSTM (with different initialization seeds): here, we consider 6 DR-BiLSTMs with different initialization seeds.
- *tanh*-Projection: same configuration as DR-BiLSTM, but we use *tanh* instead of *ReLU* as the activation function in Equations 3.6 and 3.7:

$$p_i = tanh(W_p a_i + b_p) \tag{3.15}$$

$$q_j = tanh(W_p b_j + b_p) \tag{3.16}$$

- DR-BiLSTM (with 1 round of dependent reading): same configuration as DR-BiLSTM, but we do not use dependent reading during the inference process. In other words, we use $\tilde{p} = \bar{p}$ and $\tilde{q} = \bar{q}$ instead of Equations 3.10 and 3.11 respectively.

- DR-BiLSTM (with 3 rounds of dependent reading): same configuration as the above, but we use 3 rounds of dependent reading. Formally, we replace Equations 3.1 and 3.2 with the following equations respectively:

$$\begin{aligned} -, s_v &= \textit{BiLSTM}(v, 0) \\ -, s_{vu} &= \textit{BiLSTM}(u, s_v) \\ -, s_{vuv} &= \textit{BiLSTM}(v, s_{vu}) \\ \hat{u}, - &= \textit{BiLSTM}(u, s_{vuv}) \end{aligned} \tag{3.17}$$

$$\begin{aligned} -, s_u &= \textit{BiLSTM}(u, 0) \\ -, s_{uv} &= \textit{BiLSTM}(v, s_u) \\ -, s_{uvu} &= \textit{BiLSTM}(u, s_{uv}) \\ \hat{v}, - &= \textit{BiLSTM}(v, s_{uvu}) \end{aligned} \tag{3.18}$$

Our final ensemble model, DR-BiLSTM (Ensemble) is the combination of the following 6 models: tanh-Projection, DR-BiLSTM (with 1 round of dependent reading), DR-BiLSTM (with 3 rounds of dependent reading), and 3 DR-BiLSTMs with different initialization seeds.

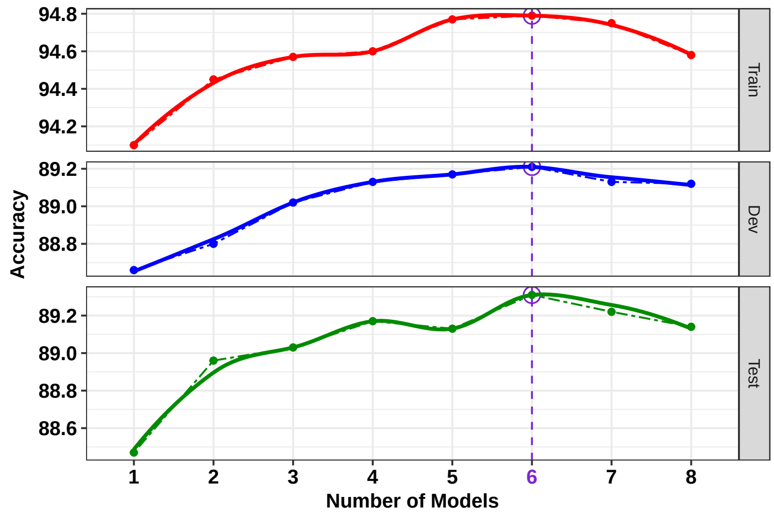


Figure 3.2: Performance of n ensemble models reported for training (red, top), development (blue, middle), and test (green, bottom) sets of SNLI. For n number of models, the best performance on the development set is used as the criteria to determine the final ensemble. The best performance on development set (89.22%) is observed using 6 models and is henceforth considered as our final DR-BiLSTM (Ensemble) model.

The main intuition behind this design is that the effectiveness of a model may depend on the complexity of a premise-hypothesis instance. For a simple instance, a simple model could perform better than a complex one, while a complex instance may need further consideration toward disambiguation. Consequently, using models with different rounds of dependent readings in the encoding stage should be beneficial.

Figure 3.2 demonstrates the observed performance of our ensemble method with different number of models. The performance of the models are reported based on the best obtained accuracy on the development set. We also study the effectiveness of other ensemble strategies e.g. majority voting and averaging the probability distribution strategies for ensemble models using the same set of models as our weighted averaging

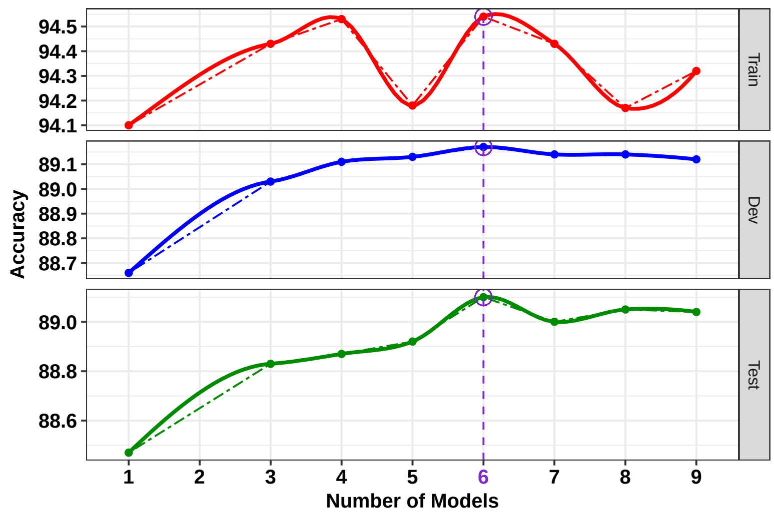


Figure 3.3: Performance of n ensemble models using majority voting on natural language inference reported for training set (red, top), development set (blue, middle), and test set (green, bottom) of SNLI. The best performance on development set is used as the criteria to determine the final ensemble. The best performance on development set is observed using 6 models.

ensemble method. Figure 3.3 shows the behavior of the majority voting strategy with different number of models. Interestingly, the best development accuracy is also observed using 6 individual models including tanh-Projection, DR-BiLSTM (with 1 round of dependent reading), DR-BiLSTM (with 3 rounds of dependent reading), and 3 DR-BiLSTMs with varying initialization seeds that are different from our DR-BiLSTM (Ensemble) model. We should note that our weighted averaging ensemble strategy performs better than the majority voting method in both development set and test set of SNLI, which indicates the effectiveness of our approach. Furthermore, our method could show more consistent behavior for training and test sets when we increased the number of models (Figure 3.2). According to our observations, averaging the probability distribu-

Original Sentence	**Corrected Sentence**
Froends ride in an open top vehicle together.	***Friends*** ride in an open top vehicle together.
A middle ***easten*** store.	A middle ***eastern*** store.
A woman is looking at a ***phtographer***	A woman is looking at a ***photographer***
The mother and daughter are ***fighitn***.	The mother and daughter are ***fighting***.
Two ***kiled*** men hold bagpipes	Two ***killed*** men hold bagpipes
A woman escapes a from a hostile ***enviroment***	A woman escapes a from a hostile ***environment***
Two ***daschunds*** play with a red ball	Two ***dachshunds*** play with a red ball
A black dog is running through a ***marsh-like*** area.	A black dog is running through a ***marsh like*** area.
a singer wearing a ***jacker*** performs on stage	a singer wearing a ***jacket*** performs on stage
There is a ***sculture***	There is a ***sculpture***
Taking a ***neverending*** break	Taking a ***never ending*** break
The woman has sounds ***emanting*** from her mouth.	The woman has sounds ***emanating*** from her mouth.
the lady is ***shpping***	the lady is ***shopping***
A Bugatti and a ***Lambourgini*** compete in a road race.	A Bugatti and a ***Lamborghini*** compete in a road race.

Table 3.2: Examples of original sentences that contain erroneous words (misspelled) in the test set of SNLI along with their corrected counterparts. Erroneous words are shown in ***bold and italic***.

tions fails to improve the development set accuracy using two and three models, so we did not study it further.

3.4.4 Preprocessing

We perform a trivial preprocessing step on SNLI to recover some out-of-vocabulary words found in the development set and test set. Note that our vocabulary contains all words that are seen in the training set, so there is no out-of-vocabulary word in it. The SNLI dataset is not immune to human errors, specifically, misspelled words. We noticed that misspelling is the main reason for some of the observed out-of-vocabulary words. Consequently, we simply fix the unseen misspelled words using Microsoft spell-checker (other approaches like edit distance can also be used). Moreover, while dealing with an unseen word during evaluation, we try to: 1) replace it with its lower case, or 2) split the word when it contains a "-" (e.g. "marsh-like") or starts with "un" (e.g. "unloading"). If we still could not find the word in our vocabulary, we consider it as an *unknown* word.

Table 3.2 shows some erroneous sentences from the SNLI test set along with their corrected equivalents (after preprocessing). Later, we demonstrate the importance and impact of such trivial preprocessing.

3.4.5 Results

Table 3.3 shows the accuracy of the models on training and test sets of SNLI. The first row represents a baseline classifier presented by Bowman et al. (2015) [7] that utilizes handcrafted features. All other listed models are deep-learning based. The gap between the traditional model and deep learning models demonstrates the effectiveness of deep learning methods for this task. We also report the estimated human performance on the SNLI dataset, which is the average accuracy of five annotators in comparison to the gold labels [30]. It is noteworthy that recent deep learning models surpass the human performance in the NLI task.

As shown in Table 3.3, previous deep learning models (rows 2-19) can be divided into three categories: 1) sentence encoding based models (rows 2-7), 2) single inter-sentence attention-based models (rows 8-16), and 3) ensemble inter-sentence attention-based models (rows 17-19). We can see that inter-sentence attention-based models perform better than sentence encoding based models, which supports our intuition. Natural language inference requires a deep interaction between the premise and hypothesis.

Model	Accuracy	
	Train	**Test**
Bowman et al. (2015) [7] (Feature)	99.7%	78.2%
Bowman et al. (2015) [7]	83.9%	80.6%
Vendrov et al. (2015) [92]	98.8%	81.4%
Mou et al. (2016) [65]	83.3%	82.1%
Bowman et al. (2016) [8]	89.2%	83.2%
Liu et al. (2016) [58]	84.5%	84.2%
Yu and Munkhdalai (2017) [103]	86.2%	84.6%
Rocktaschel et al. (2015) [80]	85.3%	83.5%
Wang et al. (2016) [98]	92.0%	86.1%
Liu et al. (2016) [57]	88.5%	86.3%
Parikh et al. (2016) [69]	90.5%	86.8%
Yu and Munkhdalai (2017) [104]	88.5%	87.3%
Sha et al. (2015) [82]	90.7%	87.5%
Wang et al. (2017) [99] (Single)	90.9%	87.5%
Chen et al. (2017) [11] (Single)	92.6%	88.0%
Gong et al. (2017) [30] (Single)	91.2%	88.0%
Chen et al. (2017) [11] (Ensemble)	93.5%	88.6%
Wang et al. (2017) [99] (Ensemble)	93.2%	88.8%
Gong et al. (2017) [30] (Ensemble)	92.3%	88.9%
Human Performance (Estimated)	97.2%	87.7%
DR-BiLSTM (Single)	94.1%	**88.5%**
DR-BiLSTM (Single)+Process	94.1%	**88.9%**
DR-BiLSTM (Ensemble)	94.8%	**89.3%**
DR-BiLSTM (Ensem.)+Process	94.8%	**89.6%**

Table 3.3: Accuracies of the models on the training set and test set of SNLI. DR-BiLSTM (Ensemble) achieves the accuracy of 89.3%, the best result observed on SNLI, while DR-BiLSTM (Single) obtains the accuracy of 88.5%, which considerably outperforms the previous non-ensemble models. Also, utilizing a trivial preprocessing step yields to further improvements of 0.4% and 0.3% for single and ensemble DR-BiLSTM models respectively.

Inter-sentence attention-based approaches can provide such interaction while sentence encoding based models fail to do so.

To further enhance the modeling of interaction between the premise and hypothesis for efficient disambiguation of their relationship, we introduce the dependent reading strategy in our proposed DR-BiLSTM model. The results demonstrate the effectiveness of our model. DR-BiLSTM (Single) achieves 88.5% accuracy on the test set which is noticeably the best reported result among the existing single models for this task. Note that the difference between DR-BiLSTM and [11] is statistically significant with a p-value of < 0.001 over the *Chi-square* test[1].

To further improve the performance of NLI systems, researchers have built ensemble models. Previously, ensemble systems obtained the best performance on SNLI with a huge margin. Table 3.3 shows that our proposed single model achieves competitive results compared to these reported ensemble models. Our ensemble model considerably outperforms the current state-of-the-art by obtaining 89.3% accuracy.

Up until this point, we discussed the performance of our models where we have not considered preprocessing for recovering the out-of-vocabulary words. In Table 3.3, "DR-BiLSTM (Single) $+$ Process", and "DR-BiLSTM (Ensem.) $+$ Process" represent the performance of our models on the preprocessed dataset. We can see that our preprocessing mechanism leads to further improvements of 0.4% and 0.3% on the SNLI test set for our single and ensemble models respectively. In fact, our single model ("DR-BiLSTM (Single) $+$ Process") obtains the state-of-the-art performance over both reported single and ensemble models by performing a simple preprocessing step. Furthermore, "DR-BiLSTM (Ensem.) $+$ Process" outperforms the existing state-of-the-art

[1]Chi-square test (χ^2 test) is used to determine if there is a significant difference between two categorical variables (i.e. models' outputs).

Model	Dev Acc[a]	p-value
DR-BiLSTM	**88.69%**	-
DR-BiLSTM - hidden MLP	88.45%	<0.001
DR-BiLSTM - average pooling	88.50%	<0.001
DR-BiLSTM - max pooling	88.39%	<0.001
DR-BiLSTM - element-wise product	88.51%	<0.001
DR-BiLSTM - difference	88.24%	<0.001
DR-BiLSTM - difference & element-wise product	87.96%	<0.001
DR-BiLSTM - inference pooling	88.46%	<0.001
DR-BiLSTM - dep. infer[b]	88.43%	<0.001
DR-BiLSTM - dep. enc[c]	88.26%	<0.001
DR-BiLSTM - dep. enc & infer	88.20%	<0.001

[a]Dev Acc, Development Accuracy.
[b]dep. infer, dependent reading inference.
[c]dep. enc, dependent reading encoding.

Table 3.4: Ablation study results. Performance of different configurations of the proposed model on the development set of SNLI along with their p-values in comparison to DR-BiLSTM (Single).

remarkably (0.7% improvement). For more comparison and analyses, we use "DR-BiLSTM (Single)" and "DR-BiLSTM (Ensemble)" as our single and ensemble models in the rest of the work.

3.4.6 Ablation and Configuration Study

We conducted an ablation study on our model to examine the importance and effect of each major component. Then, we study the impact of BiLSTM dimensionality on the performance of the development set and training set of SNLI. We investigate all settings on the development set of the SNLI dataset.

Table 3.4 shows the ablation study results on the development set of SNLI along with the statistical significance test results in comparison to the proposed model, DR-BiLSTM. We can see that all modifications lead to a new model and their differences are statistically significant with a p-value of < 0.001 over *Chi square* test.

Table 3.4 shows that removing any part from our model hurts the development set accuracy which indicates the effectiveness of these components. Among all components, three of them have noticeable influences: max pooling, difference in the attention stage, and dependent reading.

Most importantly, the last four study cases in Table 3.4 (rows 8-11) verify the main intuitions behind our proposed model. They illustrate the importance of our proposed dependent reading strategy which leads to significant improvement, specifically in the encoding stage. We are convinced that the importance of dependent reading in the encoding stage originates from its ability to focus on more important and relevant aspects of the sentences due to its prior knowledge of the other sentence during the encoding procedure.

Figure 3.4 shows the behavior of the proposed model accuracy on the training set and development set of SNLI. Since the models are selected based on the best observed development set accuracy during the training procedure, the training accuracy curve (red, top) is not strictly increasing. Figure 3.4 demonstrates that we achieve the best performance with 450-dimensional BiLSTMs. In other words, using BiLSTMs with lower dimensionality causes the model to suffer from the lack of space for capturing proper information and dependencies. On the other hand, using higher dimensionality leads to overfitting which hurts the performance on the development set. Hence, we use 450-dimensional BiLSTM in our proposed model.

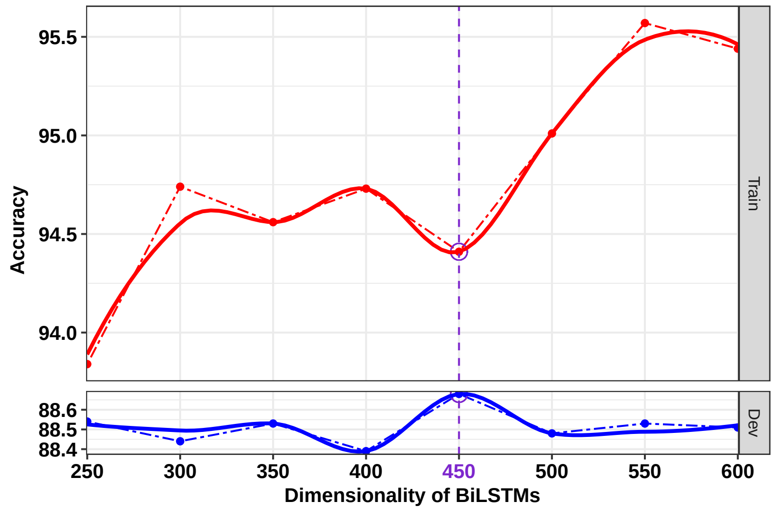


Figure 3.4: Impact of BiLSTM dimensionality in the proposed model on the training set (red, top) and development set (blue, bottom) accuracies of the SNLI dataset.

3.4.7 Analysis

We first investigate the performance of our models categorically. Then, we show a visualization of the energy function in the attention stage (Equation 3.3) for an instance from the SNLI test set.

To qualitatively evaluate the performance of our models, we design a set of annotation tags that can be extracted automatically. This design is inspired by the reported annotation tags in [101]. The specifications of our annotation tags are as follows:

- **High Overlap:** premise and hypothesis sentences share more than 70% tokens.
- **Regular Overlap:** sentences share between 30% and 70% tokens.
- **Low Overlap:** sentences share less than 30% tokens.

- **Long Sentence:** either sentence is longer than 20 tokens.
- **Regular Sentence:** premise or hypothesis length is between 5 and 20 tokens.
- **Short Sentence:** either sentence is shorter than 5 tokens.
- **Negation:** negation is present in a sentence.
- **Quantifier:** either of the sentences contains one of the following quantifiers: much, enough, more, most, less, least, no, none, some, any, many, few, several, almost, nearly.
- **Belief:** either of the sentences contains one of the following belief verbs: know, believe, understand, doubt, think, suppose, recognize, forget, remember, imagine, mean, agree, disagree, deny, promise.

Table 3.5 shows the frequency of aforementioned annotation tags in the SNLI test set along with the performance (accuracy) of ESIM [11], DR-BiLSTM (Single), and DR-BiLSTM (Ensemble). Table 3.5 can be divided into four major categories: 1) gold label data, 2) word overlap, 3) sentence length, and 4) occurrence of special words. We can see that DR-BiLSTM (Ensemble) performs the best in all categories which matches our expectation. Moreover, DR-BiLSTM (Single) performs noticeably better than ESIM in most of the categories except "Entailment", "High Overlap", and "Long Sentence", for which our model is not far behind (gaps of 0.2%, 0.5%, and 0.9%, respectively). It is noteworthy that DR-BiLSTM (Single) performs better than ESIM in more frequent categories. Specifically, the performance of our model in "Neutral", "Negation", and "Quantifier" categories (improvements of 1.4%, 3.5%, and 1.9%, respectively) indicates

Annotation Tag	Frequency	ESIM	DR(S)[b]	DR(E)[c]
Entailment	34.3%	90.0%	89.8%	90.9%
Neutral	32.8%	83.7%	85.1%	85.6%
Contradiction	32.9%	90.0%	90.5%	91.4%
High Overlap	24.3%	91.2%	90.7%	92.1%
Reg. Overlap	33.7%	87.1%	87.9%	88.8%
Low Overlap	45.4%	87.0%	87.8%	88.4%
Long Sentence	6.4%	92.2%	91.3%	91.9%
Reg. Sentence	74.9%	87.8%	88.4%	89.2%
Short Sentence	19.9%	87.6%	88.1%	89.3%
Negation	2.1%	82.2%	85.7%	87.1%
Quantifier	8.7%	85.5%	87.4%	87.6%
Belief	0.2%	78.6%	78.6%	78.6%

[b] DR(S), DR-BiLSTM (Single).
[c] DR(E), DR-BiLSTM (Ensemble).

Table 3.5: Categorical performance analyses (accuracy) of ESIM [11], DR-BiLSTM (DR(S)) and Ensemble DR-BiLSTM (DR(E)) on the SNLI test set.

the superiority of our model in understanding and disambiguating complex samples. Our investigations indicate that ESIM generates somewhat uniform attention for most of the word pairs while our model could effectively attend to specific parts of the given sentences and provide more meaningful attention. In other words, the dependent reading strategy enables our model to achieve meaningful representations, which leads to better attention to obtain further gains on such categories like Negation and Quantifier sentences.

Next, we show a visualization of the normalized (min-max normalization) attention weights (energy function, Equation 3.3) of our model in Figure 3.5. We show a sentence pair, where the premise is "*Male in a blue jacket decides to lay the grass.*", and the hypothesis is "*The guy in yellow is rolling on the grass.*", and its logical relationship is *contradiction*. Figure 3.5 indicates the model's ability in attending to critical pairs

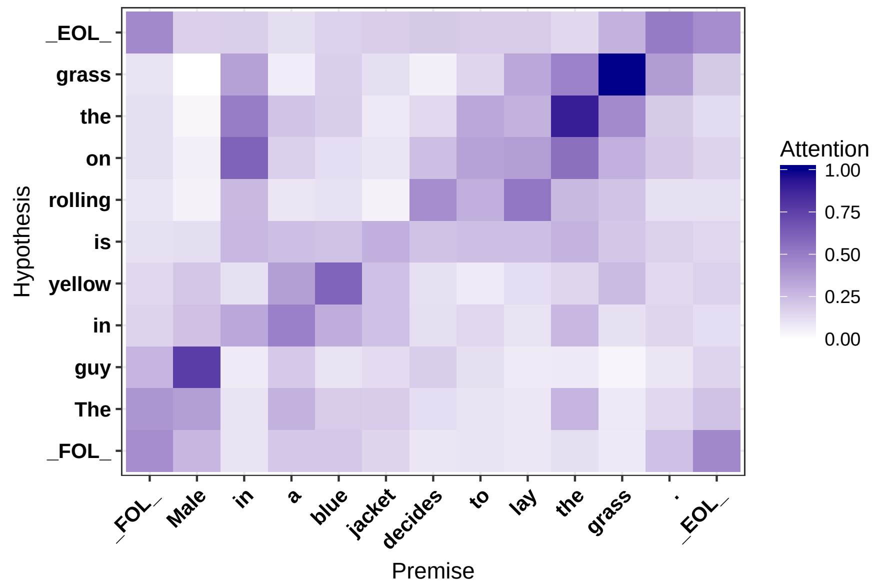
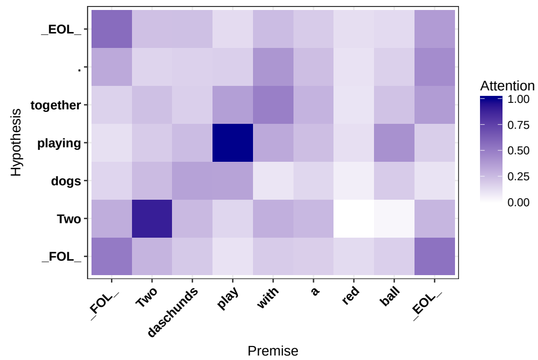


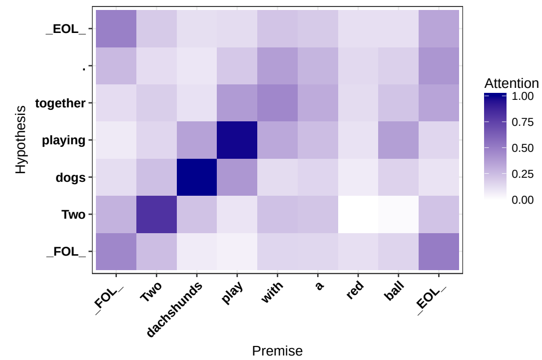
Figure 3.5: Normalized attention weights for a sample from the SNLI test set. Darker color illustrates higher attention.

of words like <Male, guy>, <decides, rolling>, and <lay, rolling>. Finally, high attention between {decides, lay} and {rolling}, and {Male} and {guy} leads the model to correctly classify the sentence pair as *contradiction*. Note that we add two dummy notations (i.e. _FOL_, and _EOL_) to all sentences which indicate their beginning and end.

Furthermore, we show the energy function (Equation 3.3) visualizations of 6 examples from Table 3.2 in Figures 3.6, 3.7, 3.8, 3.9, 3.10, and 3.11. Each figure presents the visualization of an original erroneous sample along its corrected version. These figures clearly illustrate that fixing the erroneous words leads to producing correct attentions over the sentences. This can be observed by comparing the attention for the erroneous words and corrected words, e.g. “daschunds” and “dachshunds” in the premise of Fig-



(a) Erroneous sample (daschunds in premise).



(b) Fixed sample (dachshunds in premise).

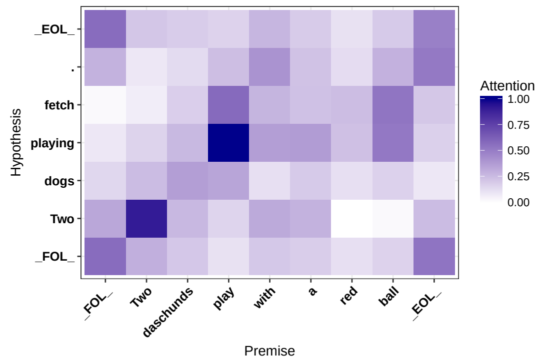
Figure 3.6: Visualization of the energy function for one erroneous sample (a) and the fixed sample (b). The gold label is *Entailment*. Our model returns *Contradiction* for the erroneous sample, but correctly classifies the fixed sample.

ures 3.6 and 3.7.

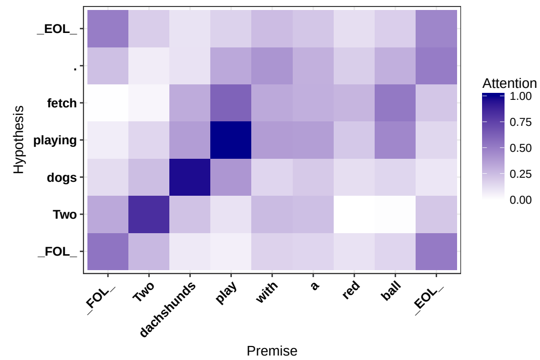
Finally we investigate the normalized attention weights of DR-BiLSTM and ESIM for four samples that belong to Negation and/or Quantifier categories (Figures 3.12 - 3.15). Each figure illustrates the normalized energy function of DR-BiLSTM (left diagram) and ESIM (right diagram) respectively. Provided figures indicate that ESIM assigns somewhat similar attention to most of the pairs while DR-BiLSTM focuses on specific parts of the given premise and hypothesis.

3.5 Conclusion

We propose a novel natural language inference model (DR-BiLSTM) that benefits from a dependent reading strategy and achieves the state-of-the-art results on the SNLI dataset. We also introduce a sophisticated ensemble strategy and illustrate its effectiveness through



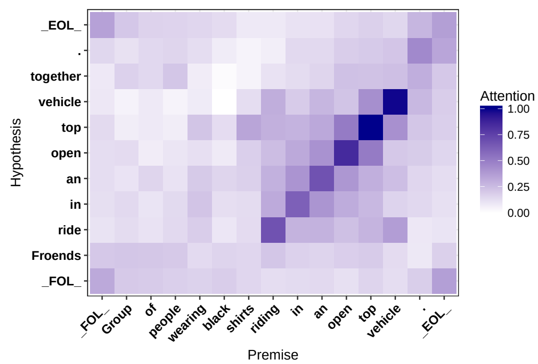
(a) Erroneous sample (daschunds in premise).



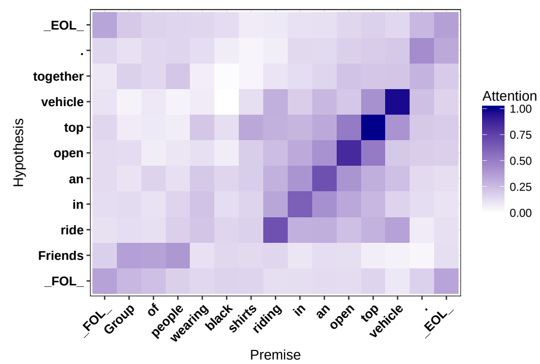
(b) Fixed sample (dachshunds in premise).

Figure 3.7: Visualization of the energy function for one erroneous sample (a) and the fixed sample (b). The gold label is *Neutral*. Our model returns *Contradiction* for the erroneous sample, but correctly classifies the fixed sample.

experimentation. Moreover, we demonstrate the importance of a simple preprocessing step on the performance of our proposed models. Evaluation results show that the preprocessing step allows our DR-BiLSTM (single) model to outperform all previous single and ensemble methods. Similar superior performance is also observed for our DR-BiLSTM (ensemble) model. We show that our ensemble model outperforms the existing state-of-the-art by a considerable margin of 0.7%. Finally, we perform an extensive analysis to demonstrate the strength and weakness of the proposed model, which would pave the way for further improvements in this domain.

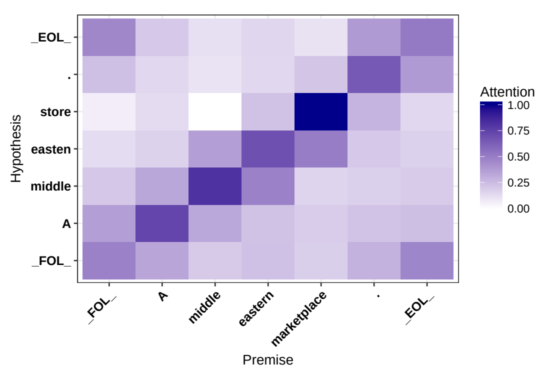


(a) Erroneous sample (Froends in hypothesis).

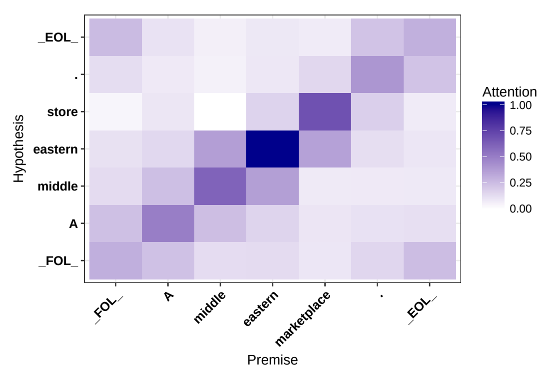


(b) Fixed sample (Friends in hypothesis).

Figure 3.8: Visualization of the energy function for one erroneous sample (a) and the fixed sample (b). The gold label is *Neutral*. Our model returns *Entailment* for the erroneous sample, but correctly classifies the fixed sample.

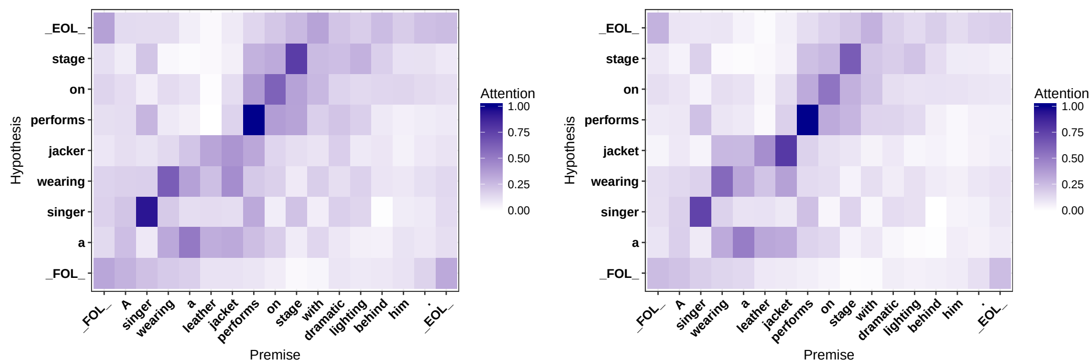


(a) Erroneous sample (easten in hypothesis).



(b) Fixed sample (eastern in hypothesis).

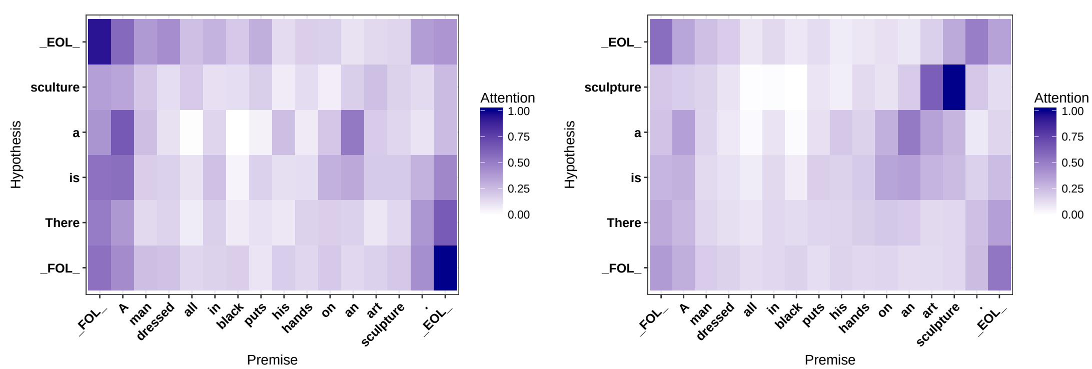
Figure 3.9: Visualization of the energy function for one erroneous sample (a) and the fixed sample (b). The gold label is *Entailment*. Our model returns *Contradiction* for the erroneous sample, but correctly classifies the fixed sample.



(a) Erroneous sample (jacker in hypothesis).

(b) Fixed sample (jacket in hypothesis).

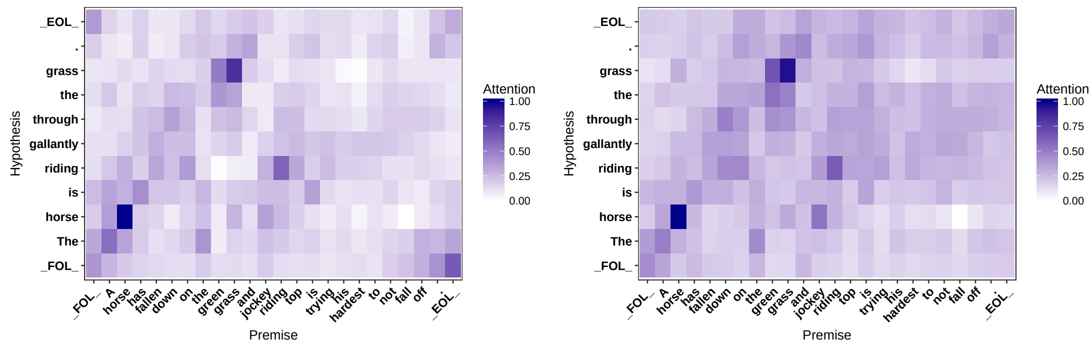
Figure 3.10: Visualization of the energy function for one erroneous sample (a) and the fixed sample (b). The gold label is *Entailment*. Our model returns *Neutral* for the erroneous sample, but correctly classifies the fixed sample.



(a) Erroneous sample (sculture in hypothesis).

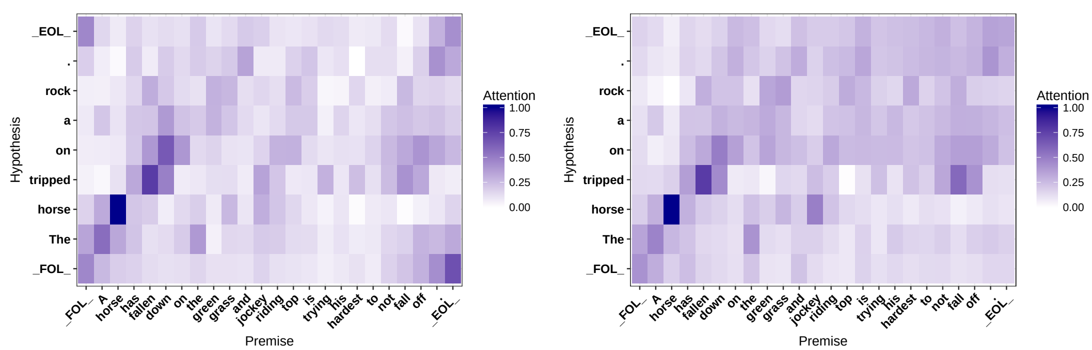
(b) Fixed sample (sculpture in hypothesis).

Figure 3.11: Visualization of the energy function for one erroneous sample (a) and the fixed sample (b). The gold label is *Entailment*. Our model returns *Neutral* for the erroneous sample, but correctly classifies the fixed sample.



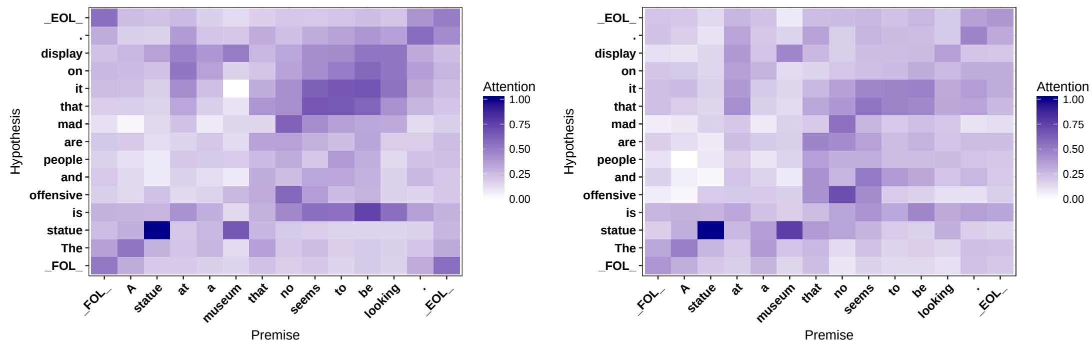
(a) Normalized attention of DR-BiLSTM. (b) Normalized attention of ESIM

Figure 3.12: Visualization of the normalized attention weights of DR-BiLSTM (a) and ESIM (b) models for one sample from the SNLI test set. This sample belongs to the Negation category. The gold label is *Contradiction*. Our model returns *Contradiction* while ESIM returns *Entailment*.



(a) Normalized attention of DR-BiLSTM. (b) Normalized attention of ESIM

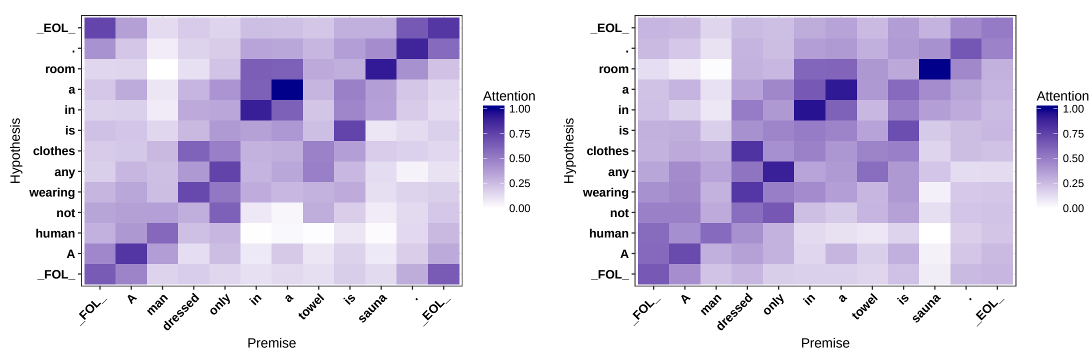
Figure 3.13: Visualization of the normalized attention weights of DR-BiLSTM (a) and ESIM (b) models for one sample from the SNLI test set. This sample belongs to the Negation category. The gold label is *Contradiction*. Our model returns *Contradiction* while ESIM returns *Entailment*.



(a) Normalized attention of DR-BiLSTM.

(b) Normalized attention of ESIM

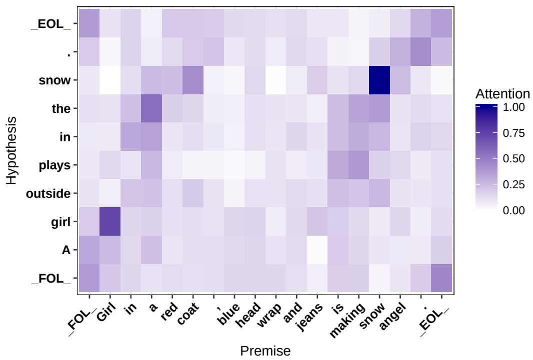
Figure 3.14: Visualization of the normalized attention weights of DR-BiLSTM (a) and ESIM (b) models for one sample from the SNLI test set. This sample belongs to both Negation and Quantifier categories. The gold label is *Neutral*. Our model returns *Neutral* while ESIM returns *Contradiction*.



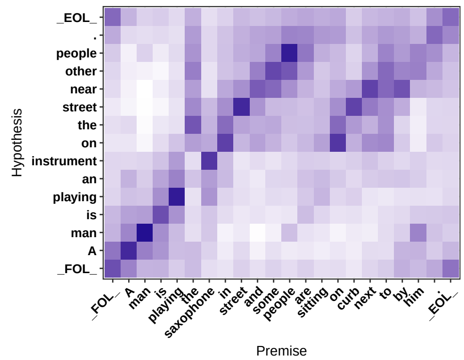
(a) Normalized attention of DR-BiLSTM.

(b) Normalized attention of ESIM

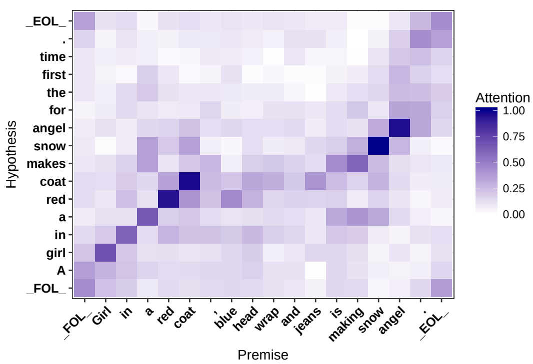
Figure 3.15: Visualization of the normalized attention weights of DR-BiLSTM (a) and ESIM (b) models for one sample from the SNLI test set. This sample belongs to both Negation and Quantifier categories. The gold label is *Entailment*. Our model returns *Entailment* while ESIM returns *Contradiction*.



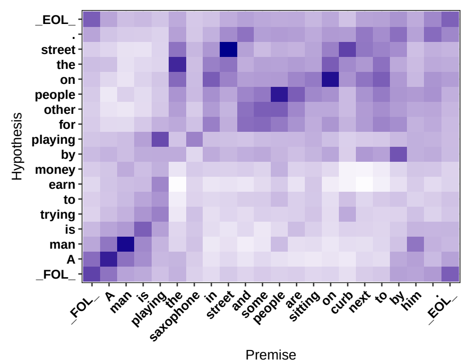
(a) Instance 1 - Entailment relationship.



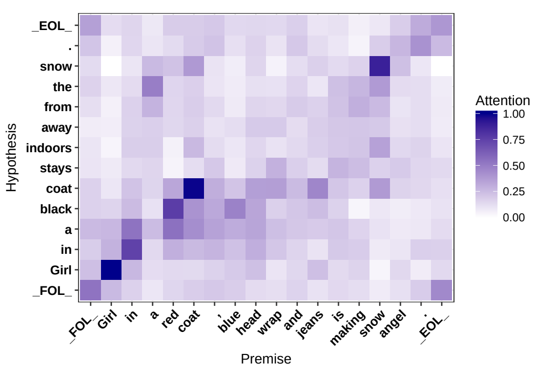
(b) Instance 2 - Entailment relationship.



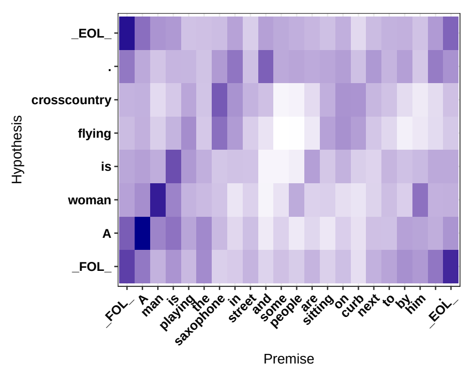
(c) Instance 1 - Neutral relationship.



(d) Instance 2 - Neutral relationship.

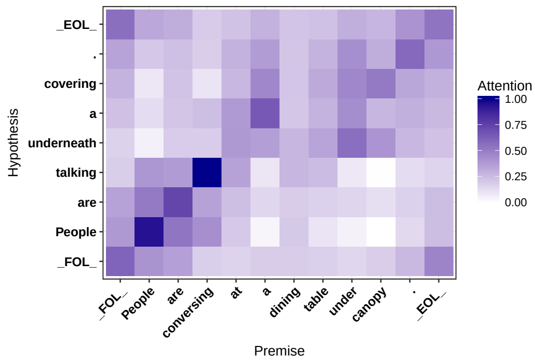


(e) Instance 1 - Contradiction relationship.

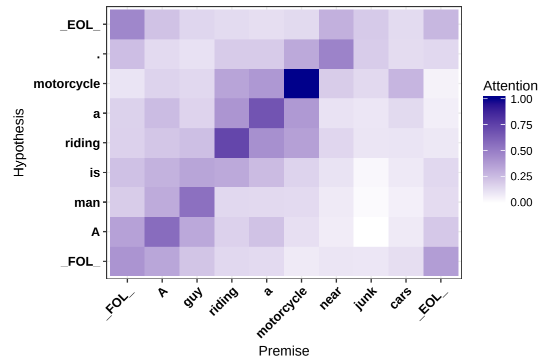


(f) Instance 2 - Contradiction relationship.

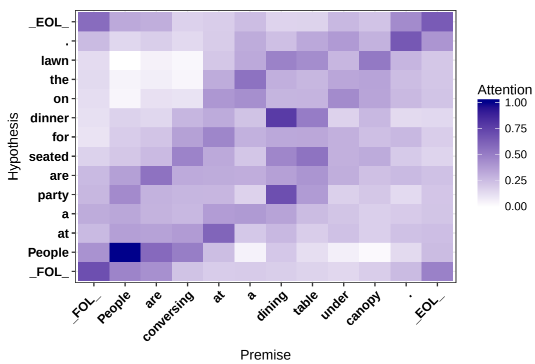
Figure 3.16: Normalized attention weights for 6 data samples from the test set of SNLI dataset. (a,c,e) and (b,d,f) represent the normalized attention weights for *Entailment*, *Neutral*, and *Contradiction* logical relationships of two premises (Instance 1 and 2) respectively. Darker color illustrates higher attention.



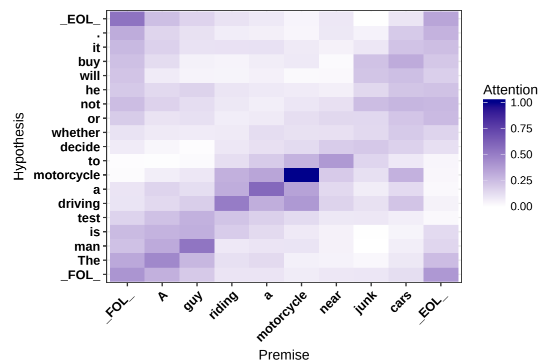
(a) Instance 3 - Entailment relationship.



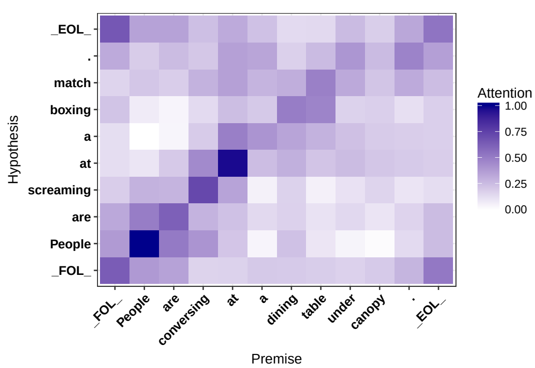
(b) Instance 4 - Entailment relationship.



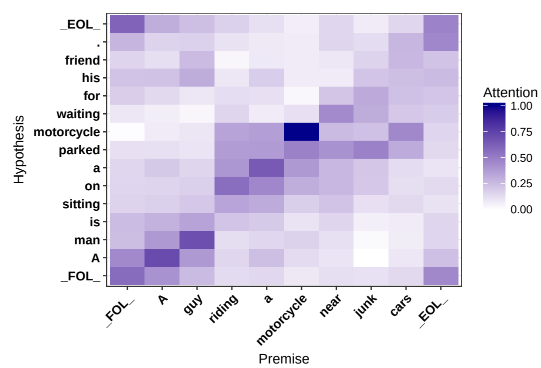
(c) Instance 3 - Neutral relationship.



(d) Instance 4 - Neutral relationship.

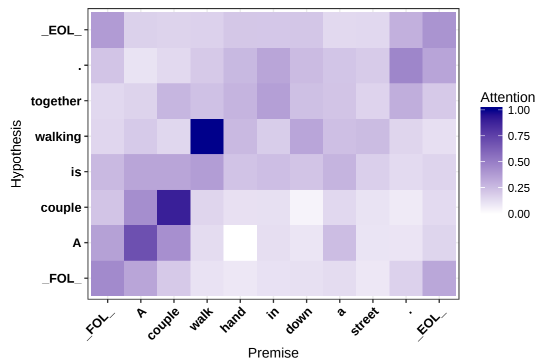


(e) Instance 3 - Contradiction relationship.

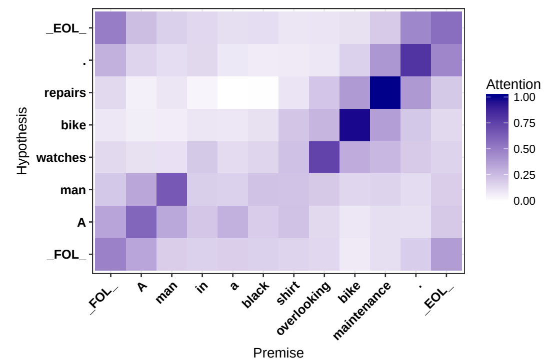


(f) Instance 4 - Contradiction relationship.

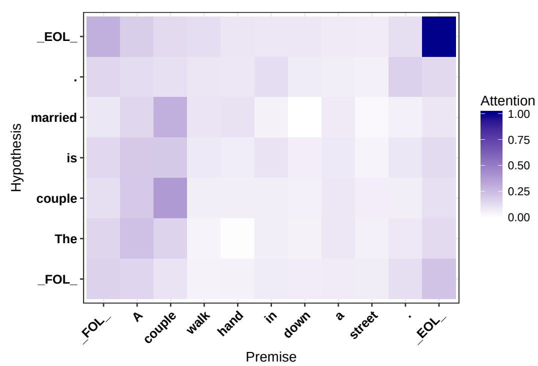
Figure 3.17: Normalized attention weights for 6 data samples from the test set of SNLI dataset. (a,c,e) and (b,d,f) represent the normalized attention weights for *Entailment*, *Neutral*, and *Contradiction* logical relationships of two premises (Instance 3 and 4) respectively. Darker color illustrates higher attention.



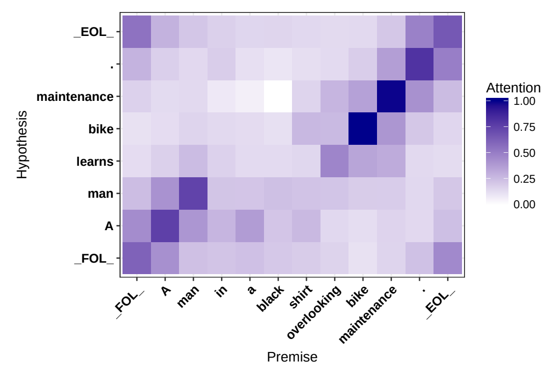
(a) Instance 5 - Entailment relationship.



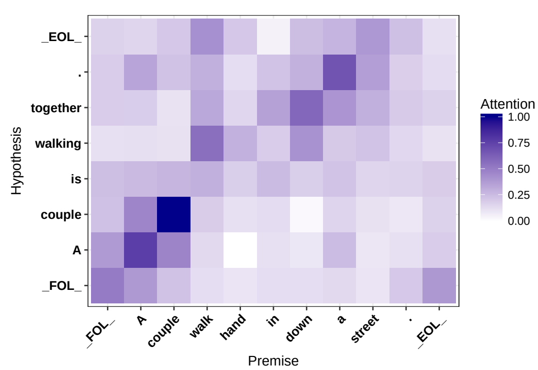
(b) Instance 6 - Entailment relationship.



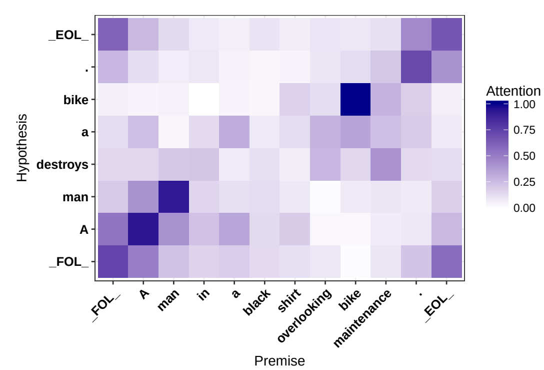
(c) Instance 5 - Neutral relationship.



(d) Instance 6 - Neutral relationship.



(e) Instance 5 - Contradiction relationship.



(f) Instance 6 - Contradiction relationship.

Figure 3.18: Normalized attention weights for 6 data samples from the test set of SNLI dataset. (a,c,e) and (b,d,f) represent the normalized attention weights for *Entailment*, *Neutral*, and *Contradiction* logical relationships of two premises (Instance 5 and 6) respectively. Darker color illustrates higher attention.

Chapter 4: Dependent gated reading for cloze-style question answering

This chapter describes the work in Ghaeini el al. (2018b) [22].

4.1 Introduction

Human language comprehension is an important and challenging task for machines that requires semantic understanding and reasoning over clues. The goal of this general task is to read and comprehend the given document and answer queries.

Recently, the cloze-style reading comprehension problem has received increasing attention from the NLP community. A cloze-style query [88] is a short passage of text containing a blank part, which we must fill with an appropriate token based on the reading and understanding of a related document. The recent introduction of several large-scale datasets of cloze-style question answering made it feasible to train deep learning systems for such task [67, 37, 36]. Various deep learning models have been proposed and achieved reasonable results for this task [102, 19, 66, 15, 90, 44, 10, 16, 85]. The success of recent models are mostly due to two factors: 1) Attention mechanisms [4], which allow the model to sharpen its understanding and focus on important and appropriate subparts of the given context; 2) Multi-hop architectures, which read the document and/or the query in multiple passes, allowing the model to re-consider and refocus its understanding in later iterations. Intuitively, both attention mechanisms and multi-hop reading fulfill the necessity of considering the dependency aspects of the given document and the query. Such a consideration enables the model to pay attention to the relevant information and ignore the irrelevant details. Human language comprehension is often performed by jointly reading the document and query to leverage their dependencies and stay focused in reading and avoid losing relevant contextual information. Current state-of-the-art models also attempt to capture this by using the reading of the query to guide the reading of the document [102, 19], or using the memory of the document to help interpret the query [66]. However, these systems only consider uni-directional dependencies. Our primary hypothesis is that we can gain further improvements by considering bidirectional dependencies.

In this work, we present a novel multi-hop neural network architecture, called Dependent Gated Reading (DGR), which addresses the aforementioned gap and performs dependent reading in both directions. Our model begins with an initial reading step that encodes the given query and document, followed by an iterative reading module (multi-hop) that employs soft attention to extract the most relevant information from the document and query encodings to augment each other's representation, which are then passed onto the next iteration of reading. Finally, the model performance a final round of attention allocate and aggregate to rank all possible candidates and make prediction.

We evaluate our model on well-known machine comprehension benchmarks such as the Children's Book Test (CBT-NE & CBT-CN), and Who DiD What (WDW, Strict & Relaxed). Our experimental results indicate the effectiveness of DGR by achieving state-of-the-art results on CBT-NE, WDW-Strict, and WDW-Relaxed. In summary, our contributions are as follows: 1) we propose a new deep learning architecture to address the existing gap of reading dependencies between the document and the query. The proposed model outperforms the state-of-the-art for CBT-NE, WDW-Strict, and WDW-Relaxed by $0.5\%, 0.8\%$, and 0.3% respectively; 2) we perform an ablation study and analysis to clarify the strengths and weaknesses of our model while enriching our understanding of the language comprehension task.

4.2 Related Work

The availability of large-scale datasets [67, 37, 36] has enabled researchers to develop various deep learning-based architectures for language comprehension tasks such as cloze-style question answering.

Sordoni et al. (2016) propose an *Iterative Alternative Attention* (IAA) reader. IAA is a multi-hop comprehension model which uses a GRU network to search for correct answers from the given document. IAA is the first model that does not collapse the query into a single vector. It deploys an iterative alternating attention mechanism that collects evidence from both the document and the query.

Kadlec et al. (2016) Introduce a single-hop model called *Attention Sum Reader* (AS Reader) that uses two bi-directional GRUs (Bi-GRU) to independently encode the query and the document. It then computes a probability distribution over all document tokens by taking the softmax of the dot product between the query and the token representations. Finally, it introduces a *pointer-sum attention* aggregation mechanism to aggregate the probability of multiple appearances of the same candidate. The candidate with the highest probability will be considered as the answer. Cui et al. (2017) introduce a similar single-hop model called *attention-over-attention* (AOA) reader which uses a two-way attention mechanism to allow the query and document to mutually attend to one another.

Trischler et al. (2016) introduce EpiReader [90], which uses AS Reader to first narrow down the candidates, then replaces the query placeholder with each candidate to yield a different query statement, and estimate the entailment between the document and the different query statements to predict the answer.

Munkhdalai and Yu (2017) (NSE) propose a computational hypothesis testing framework based on memory augmented neural networks. They encode the document and query independently at the beginning and then re-encode the query (but not the document) over multiple iterations (hops). At the end of each iteration, they predict an answer. The final answer is the candidate that obtains the highest probability over all iterations.

Dhingra et al. (2017) extend the AS Reader by proposing *Gated Attention Reader* (GA Reader). GA Reader uses a multi-hop architecture to compute the representation of the documents and query. In each iteration the query is encoded independent of the document and previous iterations, but the document is encoded iterative considering the previous iteration as well as an attention mechanism with multiplicative gating to generate query-specific document representations. GA reader uses the same mechanism for making the final predictions as the AS reader. Yang et al. (2017) further extend the GA Reader with a fine grained gating approach that uses external semantic and syntactic features (i.e. NER, POS, etc) of the tokens to combine the word and character level embeddings and produce a final representation of the words.

Among the aforementioned models, the GA Reader is the closest to our model in that we use a similar architecture that is multi-hop and performs iterative reading. The main distinct between our model and the GA Reader is the reading and encoding of the query. Instead of performing independent reading of query in each iteration, our reading and encoding of the query not only depends on the document but also the reading of previous iterations.

Although cloze-style question answering task is well studied in the literature, the potential of dependent reading and interaction between the document and the query is not rigorously explored. In this work, we address this gap by proposing a novel deep learning model (DGR). Experimental results demonstrate the effectiveness of our model.

4.3 Dependent Gated Reading

Figure 4.1 depicts a high-level view of our proposed Dependent Gate Reading (DGR) model, which follows a fairly standard multi-hop architecture, simulating the multi-step reading and comprehension process of humans.

The input to our model at the training stage can be represented as a tuple (D, Q, C, a), where $D = [d_1, \cdots, d_n]$ is the document of length n, $Q = [q_1, \cdots, q_m]$ is the query of length m with a placeholder, $C = [c_1, \cdots, c_g]$ is a set of g candidates and $a \in C$ is the ground truth answer. Here we assume d_i, q_j are some form of embedding of the individual tokens of document and query. At the testing stage, given the input document D, query Q and candidate set C, the goal is to choose the correct candidate a among C for the placeholder in Q.

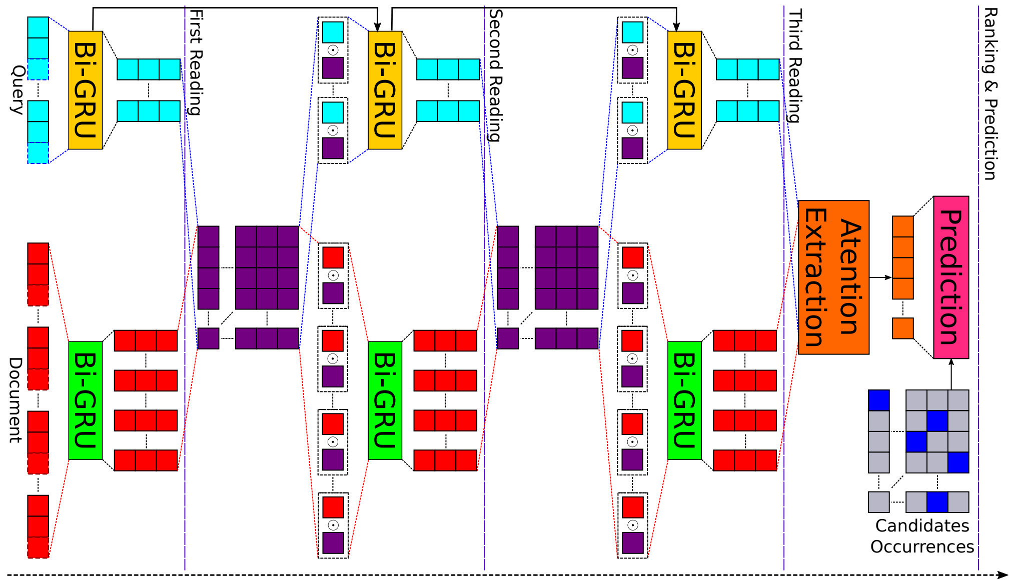


Figure 4.1: A high-level view of dependent gated reading model (DGR). The data (document d and query q, depicted with red and cyan tensors respectively) flows from left to right. At the first (input) layer, the word representations are shown with black solid borders while the character representations are shown with colored dashed borders. The figure is color coded; relevant tensors and elements are shown with the same color. Note that none of the elements share parameters. The purple matrices extract relevant information between document and query representations. The black arrows between the query Bi-GRUs (yellow ones) pass the final hidden state of a Bi-GRU to another one as initialization value for its hidden state.

DGR can be divided to two major parts: Multi-hop Reading, and Ranking & Prediction.

4.3.1 Multi-hop Reading of Document and Query

Recurrent networks provide a natural solution for modeling variable length sequences. Consequently, we use bi-directional Gated Recurrent Units (Bi-GRUs) [13] as the main building blocks for encoding the given document and query. For the initial step of our multi-hop reading, the document D and the query q are read with two separate Bi-GRUs (Equations 4.1 and 4.2) where $\hat{d}^0 \in \mathbb{R}^{n \times r}$ and $\hat{q}^0 \in \mathbb{R}^{m \times r}$ are the first Bi-GRU reading sequences of D and Q respectively. h^0 consists of two parts, h_f^0 and h_b^0, which record the final output of forward and backward GRU reading of Q respectively. Note that "$-$" in equations means that we do not care about the associated variable and its value.

$$\hat{d}^0, - = BiGRU_{d0}(D, 0) \tag{4.1}$$

$$\hat{q}^0, h^0 = BiGRU_{q0}(Q, 0) \tag{4.2}$$

We use $s \in [0, S]$ to denote the reading iteration, with $S+1$ total iterations. For the initial iteration ($s = 0$), both Bi-GRUs are fed with a zero vector for the initial hidden state as shown in Equations 4.1 and 4.2. Once the document and query encodings ($\hat{d}^s$ and $\hat{q}^s$ respectively) are computed, we employ a soft alignment method to associate the relevant sub-components between the given document and query. In deep learning models, this is often achieved with a soft attention mechanism. We follow the same soft attention mechanism as used in the GA reader [19], which is described below for completeness.

Given $\hat{d}^s$ and $\hat{q}^s$, we first compute the unnormalized attention weights between the i-th token of the document and the j-th token of the query as the similarity between the corresponding hidden states with Equation 4.3 (energy function).

$$e_{ij}^s = (\hat{d}_i^s)^T \hat{q}_j^s, \quad \forall i \in [1, n], \forall j \in [1, m], \forall s \in [0, S-1] \tag{4.3}$$

For each document token and query token, the most relevant semantics from the other context are extracted and composed based on $e_{ij}^s \in \mathbb{R}^{n \times m}$. Equations 4.4 and 4.5 provide the specific details of this procedure where $\tilde{d}_i^s \in \mathbb{R}^r$ represents the extracted information from the current reading of the query, $\hat{q}^s$, that is most relevant to the i-th document token by attending to $\hat{d}_i^s$. Similarly $\tilde{q}_j^s \in \mathbb{R}^r$ represents, for the j-th query token, the extracted relevant document information from $\hat{d}^s$ by attending to $\hat{q}_j^s$.

$$\tilde{d}_i^s = \sum_{j=1}^{m} \frac{\exp(e_{ij}^s)}{\sum_{k=1}^{m} \exp(e_{ik}^s)} \hat{q}_j^s, \quad \forall i \in [1, n], \forall s \in [0, S-1] \tag{4.4}$$

$$\tilde{q}_j^s = \sum_{i=1}^{n} \frac{\exp(e_{ij}^s)}{\sum_{k=1}^{n} \exp(e_{kj}^s)} \hat{d}_i^s, \quad \forall j \in [1, m], \forall s \in [0, S-1] \tag{4.5}$$

To incorporate the context information, we use element-wise product of the tuples $(\hat{d}_i^s, \tilde{d}_i^s)$ or $(\hat{q}_j^s, \tilde{q}_j^s)$ to produce a new representation of the hidden states for the document and the query as described in Equations 4.6 and 4.7.

$$u_i^s = \hat{d}_i^s \odot \tilde{d}_i^s, \quad \forall s \in [0, S-1] \tag{4.6}$$

$$v_j^s = \hat{q}_j^s \odot \tilde{q}_j^s, \quad \forall s \in [0, S-1] \tag{4.7}$$

Here $\odot$ stands for element-wise product, and $u^s \in \mathbb{R}^r$ and $v^s \in \mathbb{R}^r$ are the new encodings of the document and query respectively.Note that GA-reader uses the same mechanism to update the document encoding but does not change the query representation according to the document.

We then pass the new document (u^s) and query (v^s) embeddings to the Bi-GRUs for the next iteration $s+1$. Note that for query reading, we feed, h^s, the final hidden state of the previous reading (without document based updates) to the Bi-GRU of the next iteration as the initial hidden state. Intuitively, h^s provides a summary understanding of the query from the previous iteration, without the document modulated updates. By considering both h^s and v^s, this encoding mechanism provides a richer representation of the query. This is formally described by Equations 4.8 and 4.9.

$$\hat{d}^{s+1}, - = BiGRU_{ds}(u^s, 0), \forall s \in [0, S-1] \tag{4.8}$$

$$\hat{q}^{s+1}, h^{s+1} = BiGRU_{qs}(v^s, h^s), \forall s \in [0, S-1] \tag{4.9}$$

We should note that using the following configuration variations did not yield any improvement to our model: 1) Other choices for gating aggregation strategy (Equations 4.6 and 4.7) like addition, concatenation, or applying a transformation function on different sub-members of $\{$element-wise product, concatenation and difference$\}$. 2) Residual connection.

4.3.2 Ranking & Prediction

Given the final document and query encodings, $\hat{d}^S$ and $\hat{q}^S$, the final stage of our model computes a score for each candidate $c \in C$. This part of our model use the same *point sum attention* aggregation operation as introduced by the Attention Sum (AS) reader [44], which is also used by the GA reader [19].

Let idx be the position of the the placeholder in Q, and $\hat{q}^S_{idx}$ be the associated hidden embedding of the placeholder in the given query. We first compute the probability of each token in the document to be the desired answer by computing the dot product between $\hat{q}^S_{idx}$ and $\hat{d}^S_j$ for $j = 1, ..., n$ and then normalize with the softmax function:

$$y = \text{softmax}((\hat{q}^S_{idx})^T \hat{d}^S) \tag{4.10}$$

where $y \in \mathbb{R}^n$ gives us a normalized attention/probability over all tokens of the document. Next, the probability of each particular candidate $c \in C$ for being the answer is computed by aggregating the document-level attentions of all positions in which c appears:

$$p(c|D, Q) \propto \sum_{i \in I(c,D)} y_i, \quad \forall c \in C \tag{4.11}$$

where $I(c, D)$ indicates the positions that candidate c appears in the document D (Candidate Occurrences in Figure 4.1). Finally the prediction is given by $a^* = \mathit{argmax}_{c \in C}\, p(c|D, Q)$.

Key differences from the GA reader. Given the strong similarity between our model and the GA reader, it is worth highlighting the three key differences between the two models: (a) Document gated query reading: we compute a document-specific query representations to pass to the next query reading step; (b) Dependent query reading: in each iteration, the input to the query BiGRU comes from the document gated encoding of the query from the last iteration whereas the GA Reader reads the queries independently in all iterations; (c) Dependent query BiGRU initialization: the query BiGRU is initialized with the final hidden states of the query BiGRU from the previous iteration. These key differences in query encoding are designed to better capture the interdependences between query and document and produce richer and more relevant representations of the query and enhance the comprehension and query answering performance.

4.3.3 Further Enhancements

Following the practice of GA reader, we included several enhancements which have been shown to be helpful in previous work.

Question Evidence Common Word Feature. To generate the final document encoding $\hat{d}^S$, an additional modification of u^{S-1} is introduced before applying Equation 4.8. Specifically, an additional *Question Evidence Common Word Feature* (qe-comm) [54] is introduced for each document token, indicating whether the token is present in the query. Assume f_i stands for the qe-comm feature of the i-th document token, therefore, $u_i^{S-1} = [u_i^{S-1}, f_i]$.

Character-level embeddings. Word-level embeddings are good at representing the semantics of the tokens but suffers from out-of-vocabulary (OOV) words and is incapable of representing sub-word morphologies. Character-level embeddings effectively address such limitations [56, 20]. In this work, we represent a token by concatenating its word embedding and character embedding. To compute the character embedding of a token $w = [x_1, \cdots, x_l]$, we pass w to two GRUs in forward and backward directions respectively. Their outputs are then concatenated and passed through a linear transformation to form the character embedding of the token.

	CBT-NE	CBT-CN	WDW-Strict	WDW-Relaxed
# training set	108,719	120,769	127,786	185,978
# development set	2,000	2,000	10,000	10,000
# test set	2,500	2,500	10,000	10,000
# vocabulary	53,063	53,185	347,406	308,02
max document length	1,338	1,338	3,085	3,085

Table 4.1: Dataset statistics

4.4 Experiments and Evaluation

4.4.1 Datasets

We evaluate the DGR model on three large-scale language comprehension datasets, Children's Book Test Named Entity (CBT-NE), Common Noun (CBT-CN), and Who Did What (WDW) Strict and Relaxed.

The first two datasets are formed from two subsets of the Children's Book Test (CBT) [37]. Documents in CBT consist of 20 contiguous sentences from the body of a popular children's book, and queries are formed by replacing a token from the 21^{st} sentence with a placeholder. We experiment on subsets where the replaced token is either a named entity (CBT-NE) or common noun (CBT-CN). Other subsets of CBT have also been studied previously but because simple language models have been able to achieve human-level performance on them, we ignore such subsets [37].

The Who Did What (WDW) dataset [67] is constructed from the LDC English Gigaword newswire corpus. Each sample in WDW is formed from two independent articles. One article is considered as the passage to be read and the other article on the same subject is used to form the query. Missing tokens are always person named entities. For this dataset, samples that are easily answered by simple systems are filtered out, which makes the task more challenging. There are two versions for the training set (Strict and Relaxed) while using the same development and test sets. Strict is a small but focused/-clean training set while Relaxed is a larger but more noisy training set. We experiment on both of these training sets and report corresponding results on both settings. Statistics of all the aforementioned datasets are summarized in Table 4.1.

Other datasets for this task include CNN and Daily Mail News [36]. Because previous models already achieved human-level performance on these datasets, following Munkhdalai and Yu [66], we do not include them in our study.

4.4.2 Training Details & Experimental Setup

We use pre-trained 100-D Glove $6B$ vectors [70] to initialize our word embeddings while randomly initializing the character embedding. All hidden states of BiGRUs have 128 dimensions ($o = 100$ and $r = 128$). The weights are learned by minimizing the negative log-loss (Equation 4.12) on the training data via the Adam optimizer [48]. The learning rate is 0.0005. To avoid overfitting, we use dropout [86] with rate of 0.4 and 0.3 for CBT and WDW respectively as regularization, which is applied to all feedforward connections. While we fix the word embedding, character embeddings are updated during the training to learn effective representations for this task. We use a fairly small batch size of 32 to provide more exploration power to the model.

$$L = \sum_{i} -\log(p(a|D, Q)) \tag{4.12}$$

Method	Test Accuracy(%)			
	CBT-NE	CBT-CN	WDW-Strict	WDW-Relaxed
AS Reader [44]	68.6%	63.4%	57.0%	59.0%
EpiReader [90]	69.7%	67.4%	-	-
IAA Reader [85]	68.6%	69.2%	-	-
AOA Reader [15]	72.0%	69.4%	-	-
GA Reader [19]	74.9%	70.7%	71.2%	72.6%
AS Reader (Ensemble) [44]	70.6%	68.9%	-	-
EpiReader (Ensemble) [90]	71.8%	70.6%	-	-
IAA Reader (Ensemble) [85]	72.0%	71.0%	-	-
AOA Reader (Ensemble) [15]	74.5%	70.8%	-	-
NSE (T=1) [66]	71.1%	69.7%	65.5%	65.3%
NSE Query Gating (T=2) [66]	71.5%	70.7%	65.1%	65.5%
NSE Query Gating (T=6) [66]	71.4%	72.0%	65.7%	65.8%
NSE Adaptive Comp. (T=2) [66]	72.1%	71.2%	65.4%	66.0%
NSE Adaptive Comp. (T=12) [66]	73.2%	71.4%	66.2%	66.7%
FG [102]	74.9%	72.0%	71.7%	72.6%
DGR	**75.4%**	70.7%	**72.0%**	**72.9%**

Table 4.2: Performance of proposed model (DGR) on the test set of CBT-NE, CBT-CN, WDW-Strict, and WDW-Relaxed datasets.

4.4.3 Results

Table 4.2 shows the test accuracy of the models on CBT-NE, CBT-CN, WDW-Strict, and WDW-Relaxed. We divide the previous models into four categories: 1) Single models (rows 1-5), 2) Ensemble models (rows 6-9), 3) NSE models (rows 10-14), and 4) the FG model (row 15). Table 4.2 primarily focuses on comparing models that do not rely on any NLP toolkit features (i.e. POS, NER, etc), with the exception of the FG model which uses additional information about document tokens including POS, NER and word frequency information to produce the embedding of the token.

From Table 4.2, we can see that DGR achieves the state-of-the-art results on all aforementioned datasets expect for CBT-CN. The targets of CBT-NE, WDW-Strict, and WDW-Relaxed are all Named Entities while the CBT-CN focuses on Common Noun. We believe that our architecture is more suitable for Named Entity targeted comprehension tasks. This phenomenon warrants a closer look in future work. Comparing GA Reader, FG, and DGR (the three models with similar architectures), we see that FG outperform the GA Reader on CBT-CN and WDW-Strict datasets while DGR outperforms both FG and GA Reader results on CBT-NE, WDW-Strict, WDW-Relaxed datasets with noticeable margins. This suggests that while the NLP toolkit features such as POS and NER could help the performance of the comprehension models (specially in CBT-CN), capturing richer dependency interaction between document and query appears to play a more important role for comprehension tasks focusing on Named Entities.

Finally, For each of the three datasets on which our model achieves the state-of-the-art performance, we conducted the one-sided McNemar's test to verify the statistical significance of the performance improvement over the main competitor (GA reader). The obtained p-values are 0.03, 0.003, and 0.011 for CBT-NE, WDW-Strict, and WDW-Relaxed respectively, indicating that the performance gain by DGR is statistically significant.

4.4.4 Ablation Study

We conducted an ablation study on our model to examine the importance and the effect of proposed strategies. We investigate all settings on the development set of the CTB-NE, CBT-CN, WDW-Strict, and WDW-Relaxed datasets. Consider the three key

Method	Development Accuracy(%)			
	CBT-NE	CBT-CN	WDW-Strict	WDW-Relaxed
1) DGR	**77.90**	**73.80**	**71.78**	**72.26**
2) DGR - (a)	75.60	72.25	71.04	71.82
3) DGR - (c)	77.50	72.45	71.29	71.93
4) DGR - (a) & (b)	77.85	73.05	71.67	72.20
5) DGR - (a) & (c)	76.00	72.85	71.37	72.13
6) DGR - (a) & (b) & (c)	77.65	73.00	71.61	72.16

Table 4.3: Ablation study results. Performance of different configurations of the proposed model on the development set of the CBT-NE, CBT-CN, WDW-Strict, and WDW-Relaxed datasets

differences of our method from the GA Reader: (a) Document gated query reading — here we compute a document-specific query representations to pass to the next reading layer; (b) Dependent query reading — the query readings are dependent from one layer to the next as the input to the next reading layer comes from the output of previous layer; (c) Dependent BiGRU initialization — query BiGRUs of a later layer are initialized with the final hidden states of previous layer's query BiGRU.

Table 4.3 shows the ablation study results on the development set of CBT-NE, CBT-CN, WDW-Strict, and WDW-Relaxed for a variety of DGR configurations by removing one or more of the key differences with GA reader. Note that by all removing all three difference elements, configuration 6 reduces to the GA reader.

According to Table 4.3, DGR achieves the best development accuracy on all datasets which indicates that collectively, the three elements lead to improved effectiveness.

Effect of document dependent reading. Configuration 2 removes the document dependent reading, and retains the other two elements. Interestingly, this configuration achieved the worst performance among all variations. Without proper guiding from the document side, iteratively reading the query actually leads to worse performance than independent query reading. This suggests that document dependent reading is a critical element that helps achieve better reading of query.

Effect of Dependent Query BiGRU initialization. In Configuration 3, we remove the dependent query BiGRU initialization, which results in a performance loss ranging from 0.33% (WDW-relaxed) to 1.35% (CBT-CN), suggesting that this connection provides important information that helps the reading of the query. Note that simply adding dependent query BiGRU initialization to GA reader (configuration 4) leads to a slight improvement over GA reader, which again confirms the usefulness of this channel of information.

Effect of dependent query reading. Unfortunately, we cannot only remove (b) from our model because it will cause dimension mismatch between the document and query representation preventing the gating operation for computing the document gated query representation. Instead, we compare the GA reader (configuration 6) with configure 5, which adds dependent query reading to the GA reader. We can see that adding the dependent query reading to the GA reader actually leads to a slight performance loss. Note that further including document gated reading (configuration 3) improves the performance on CBT-NE, but still fails to outperform GA reader. This points to a potential direction to further improve our model by designing a new mechanism that is capable of document dependent gating without the dependent query reading.

4.4.5 Rule-based Disambiguation Study

Here, we present a simple rule-based detection strategy for CBT-NE dataset which disambiguates about 30% and 18% of the samples in CBT-NE development and test sets. For each query q, assume w is previous/next next word in the placeholder which start with upper case character. If such a w exists, we look for w in the document d and collect all words that could appears next/before w. After removing all collected words that are not in the candidate list C, the samples is disambiguated and solved if we end up with a single word (answer). We refer to the set of such samples as disambiguated set. Table 4.4 shows the statistics of this rule-based strategy on the rule-based disambiguated test set of CBT-NE. Furthermore, Table 4.5 shows a data sample in CBT-NE that is correctly disambiguated with our rule-based approach.

Set	Correct Disambiguation(%)	Wrong Disambig.(%)	Total Disambig.
Development	29.65%	0.1%	595
Test	18.36%	0.12%	462

Table 4.4: Statistics and performance of the proposed rule-based strategy on CBT-NE dataset.

Figure 4.2 shows the performance of DGR and its variations on the set of data samples in CBT-NE test set that could be disambiguated with the proposed rule-based strategy. Although we use the lower case words in the training process, all models perform substantially well on disambiguating such samples. This observation could demonstrate the effectiveness of the general architecture.

doc[a]	1 Instead of answering , **Jimmy** Skunk began to laugh . 2 “ Who ’s a bug ? ” 3 demanded Old Mr. Toad , more crossly than before . 4 “ There is n’t any bug , Mr. Toad , and I beg your pardon , ” replied **Jimmy** , remembering his politeness . 5 “ I just thought there was . 6 You see , I did n’t know you were under that piece of bark . 7 I hope you will excuse me , Mr. Toad . 8 Have you seen any fat beetles this morning ? ” 9 “ No , ” said Old Mr. Toad grumpily , and yawned and rubbed his eyes . 10 “ Why , ” exclaimed **Jimmy** Skunk , “ I believe you have just waked up ! ” 11 “ What if I have ? ” 12 demanded Old Mr. Toad . 13 “ Oh , nothing , nothing at all , Mr. Toad , ” replied **Jimmy** Skunk , “ only you are the second one I ’ve met this morning who had just waked up . ” 14 “ Who was the other ? ” 15 asked Old Mr. Toad . 16 “ Mr. Blacksnake , ” replied **Jimmy** . 17 “ He inquired for you . ” 18 Old Mr. Toad turned quite pale . 19 “ I – I think I ’ll be moving along , ” said he . 20 XVII OLD MR. TOAD ’S MISTAKE If is a very little word to look at , but the biggest word you have ever seen does n’t begin to have so much meaning as little “ if . ”
query	21 If **Jimmy** @placeholder had n’t ambled down the Crooked Little Path just when he did ; if he had n’t been looking for fat beetles ; if he had n’t seen that big piece of bark at one side and decided to pull it over ; if it had n’t been for all these “ ifs , ” why Old Mr. Toad would n’t have made the mistake he did , and you would n’t have had this story .
cands[b]	Blacksnake, Jimmy, Mr., Skunk, Toad, XVII, bug, morning, pardon, second
ans[c]	Skunk
pred[d]	Skunk

[a] doc, Document
[b] cands, Candidates
[c] ans, Answer
[d] pred, Prediction

Table 4.5: Example of a disambiguated sample in CBT-NE dataset with the proposed rule-based approach.

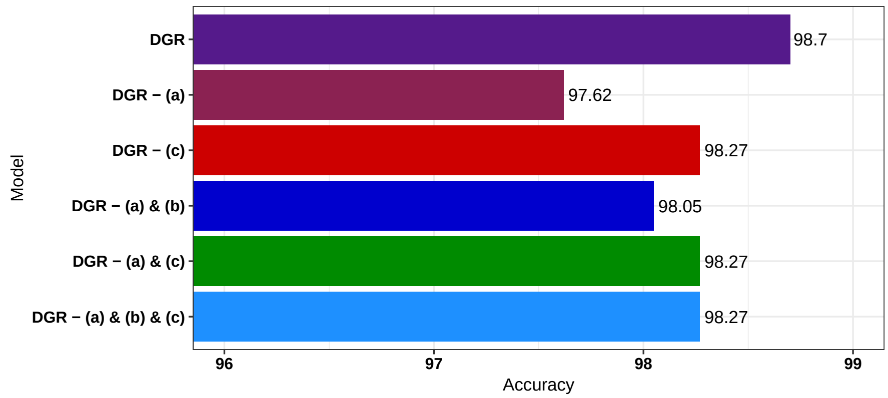


Figure 4.2: Performance of DGR and its variations on the rule-based disambiguated test set of CBT-NE.

4.4.6 Analysis

In this section, We first investigate the performance of DGR and its variations on two attributes: the *document length*, and *query length*. Then we show a layer-wise visualization of the energy function (Equation 4.3) for an instance from the CBT-NE dataset.

4.4.6.1 Length Study

Among the four datasets that we use in this work, WDW-Relaxed is the biggest and the most noisy one which makes it as a good candidate for analyzing the trend and behavior of our models.

Figure 4.3 depicts the performance of DGR and its variations against the length of document (left), and the length of query (right). A bar on top of each diagram indicates the frequency of samples in each intervals. Each data sample is added to the closet interval.

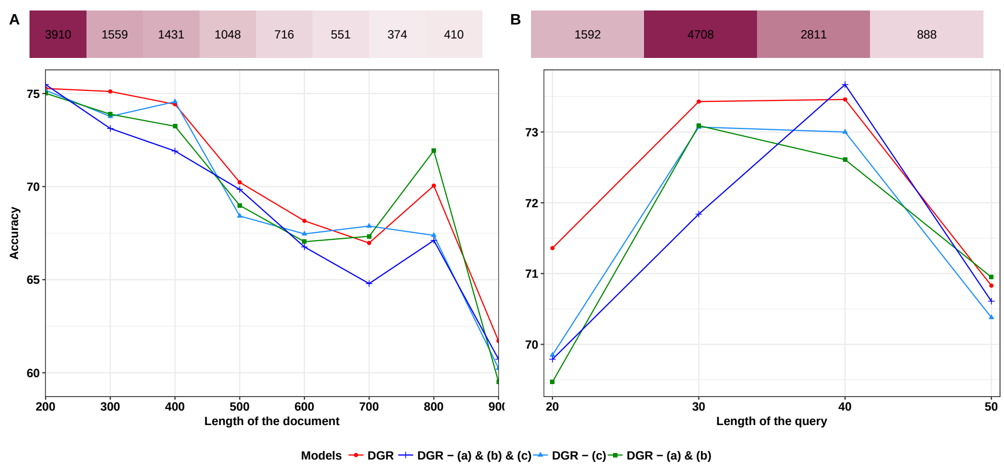


Figure 4.3: Test accuracy of DGR and its variations against the length of the document (A), and length of the query (B) on the WDW-Relaxed dataset. The bar on top of each figure indicates the number of samples in each interval. Darker color in the bars illustrates more samples.

Overall Figure 4.3 suggests that DGR achieves highly competitive performance across different document and query lengths in comparison to the other variations including the GA reader. In particular, DGR perform better or similarly to the GA reader ("DGR - (a) & (b) & (c)") in all categories except when query length is between 30 and 40 where GA reader wins with a small margin. Furthermore, we see that "DGR - (a) & (b)" wins over "DGR - (a) & (b) & (c)" in most document length categories. This suggests the positive effect of the connection offered by (c), especially for longer documents.

4.4.6.2 Attention Study

To gain insights into the influence of the proposed strategies on the internal behavior of the model, we analyze the attention distribution at intermediate layers. We show a visualization of layer-wise normalized aggregated attention weights (energy function, Equation 4.3) for candidate set over the query (Figures 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12). In each figure, the top plots show the layer-wise attention of DGR and the bottom plots show the layer-wise attention of the GA reader, i.e., "DGR - (a) & (b) & (c)". Moreover, the left and middle plot show the aggregated attention of candidates over the whole query while the right plot depicts the aggregated attention of the candidates for the placeholder in the query in the final layer.

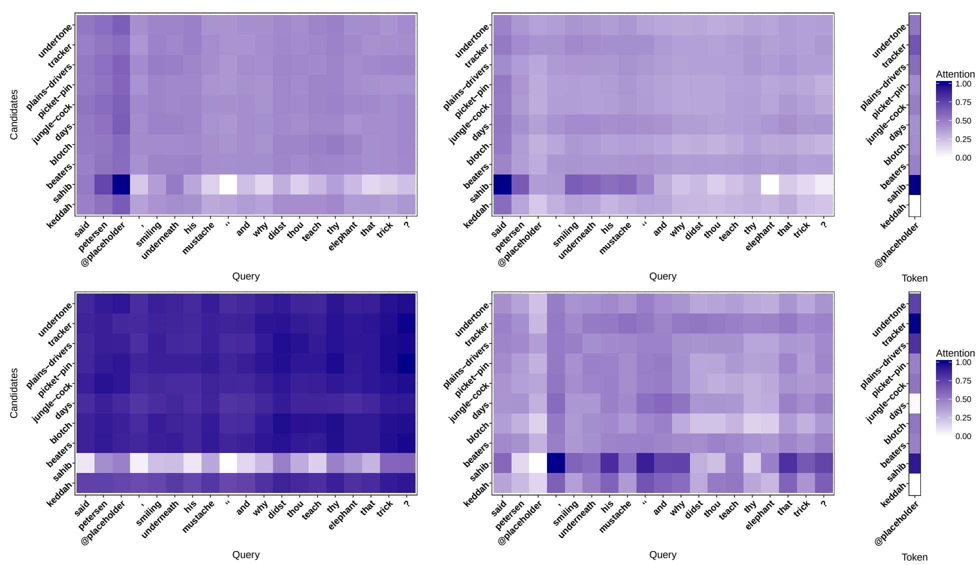


Figure 4.4: Layer-wise normalized attention visualization of "DGR" (top) and "DGR - (a) & (b) & (c)" (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is "sahib".

A generic pattern observed in our study is that GA reader tends to generate more uniform attention distributions while DGR produces more focused attention. In other words, each layer of DGR tends to focus on different sub-parts and examine different hypotheses, illustrating the significant impact of the proposed strategies on the attention mechanism.

4.5 Conclusion

We proposed a novel cloze-style question answering model (DGR) that efficiently model the relationship between the document and the query. Our model achieves the the state-of-the-art results on several large-scale benchmark datasets such as CBT-NE, WDW-Strict, and WDW-Relaxed. Our extensive analysis and ablation studies confirm our hypothesis that using a more sophisticated method for modeling the interaction between document and query could yield further improvements.

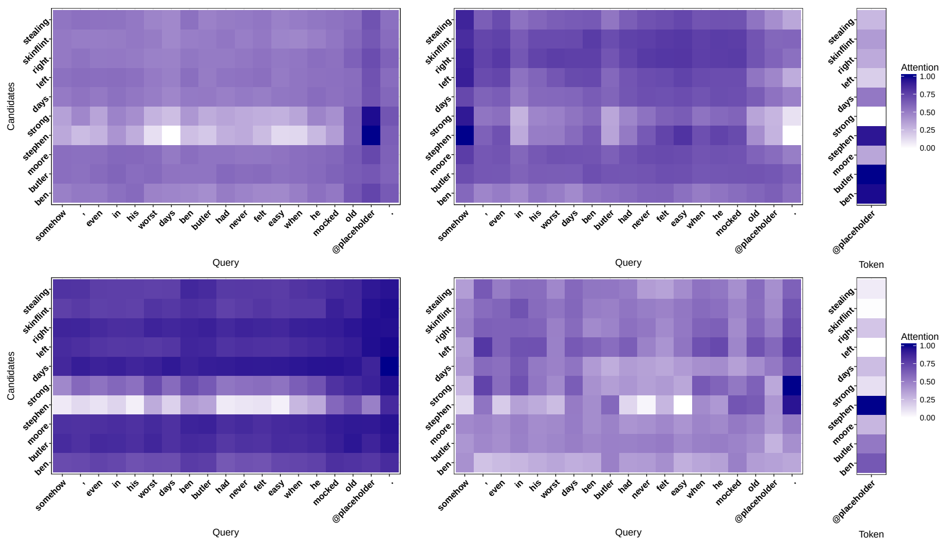


Figure 4.5: Layer-wise normalized attention visualization of “DGR” (top) and “DGR - (a) & (b) & (c)” (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is “butler”.

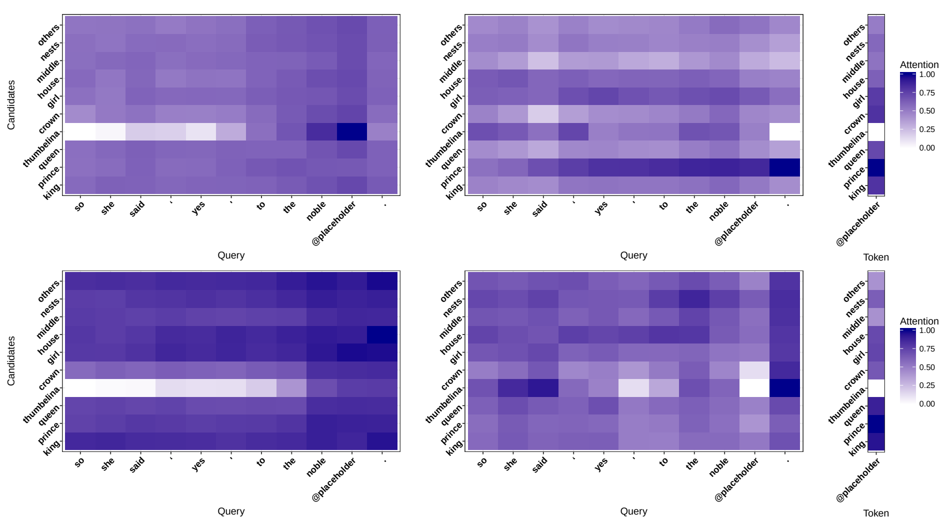


Figure 4.6: Layer-wise normalized attention visualization of “DGR” (top) and “DGR - (a) & (b) & (c)” (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is “prince”.

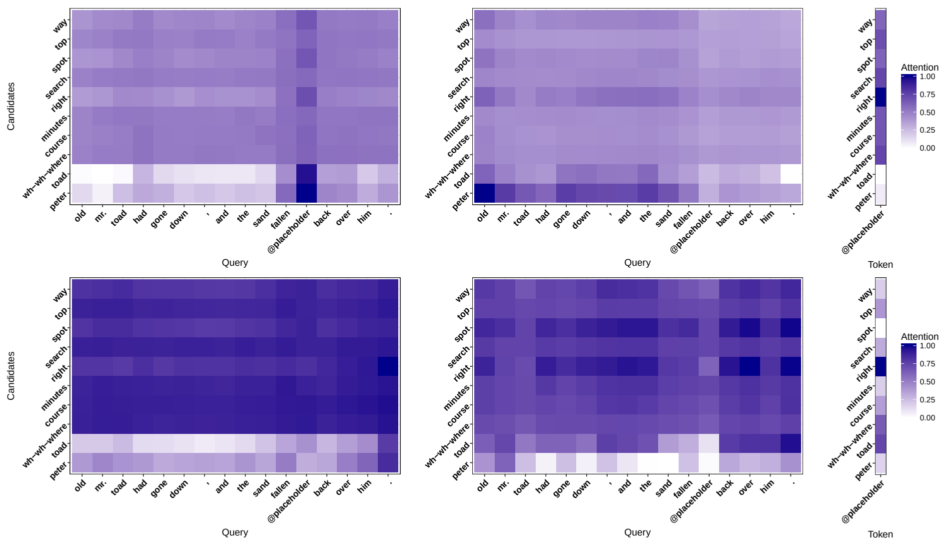


Figure 4.7: Layer-wise normalized attention visualization of "DGR" (top) and "DGR - (a) & (b) & (c)" (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is "right".

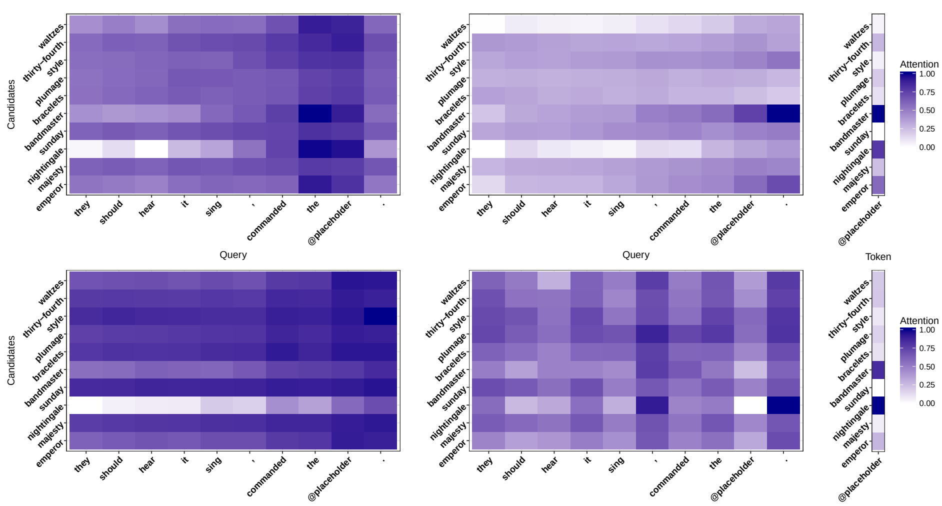


Figure 4.8: Layer-wise normalized attention visualization of "DGR" (top) and "DGR - (a) & (b) & (c)" (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is "bandmaster".

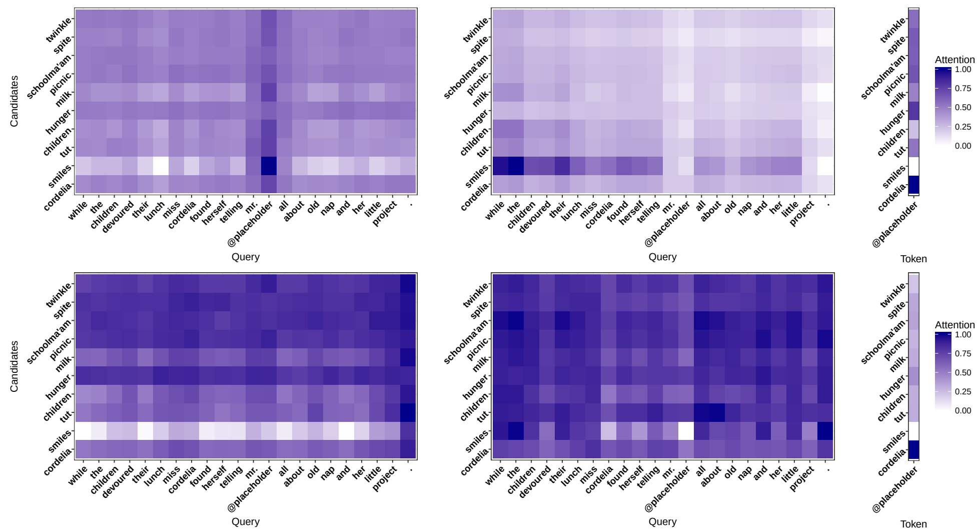


Figure 4.9: Layer-wise normalized attention visualization of "DGR" (top) and "DGR - (a) & (b) & (c)" (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is "cordelia".

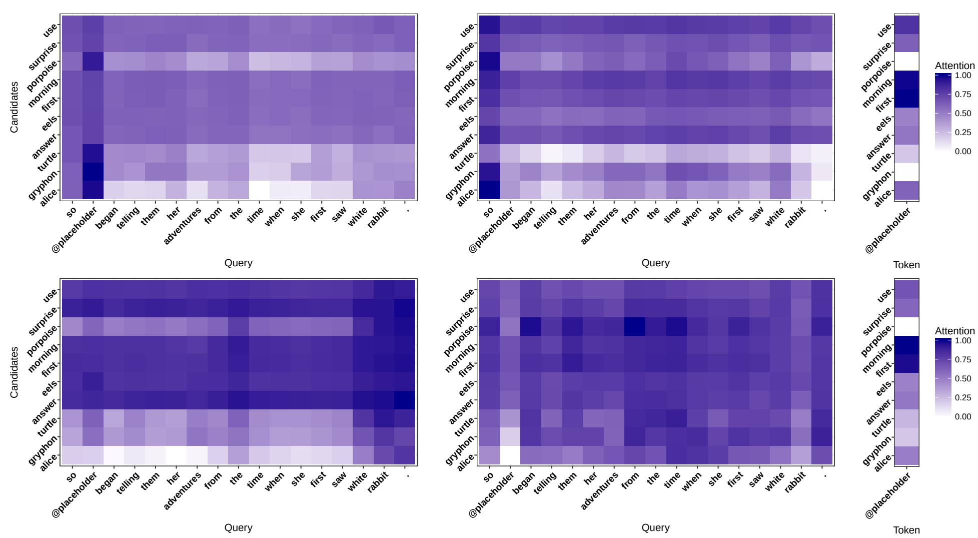


Figure 4.10: Layer-wise normalized attention visualization of "DGR" (top) and "DGR - (a) & (b) & (c)" (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is "first".

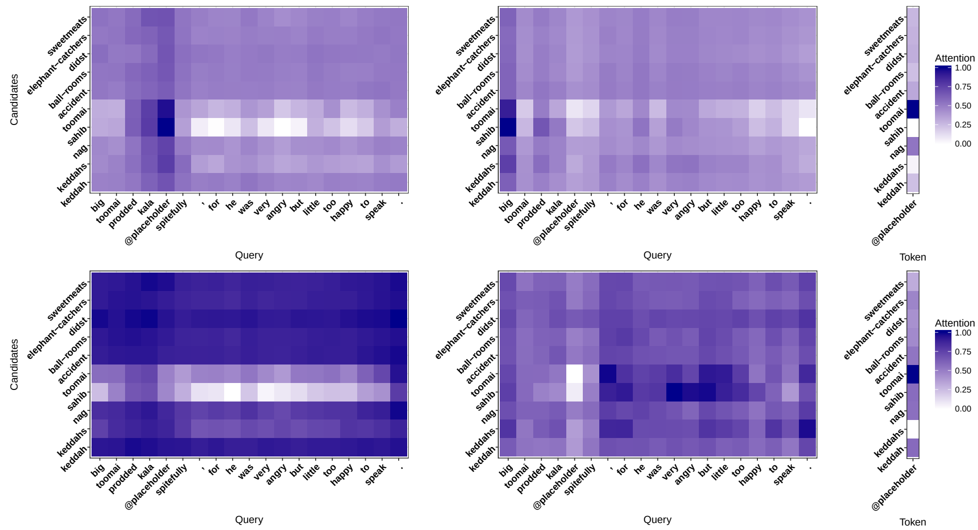


Figure 4.11: Layer-wise normalized attention visualization of “DGR” (top) and “DGR - (a) & (b) & (c)” (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is “toomai”.

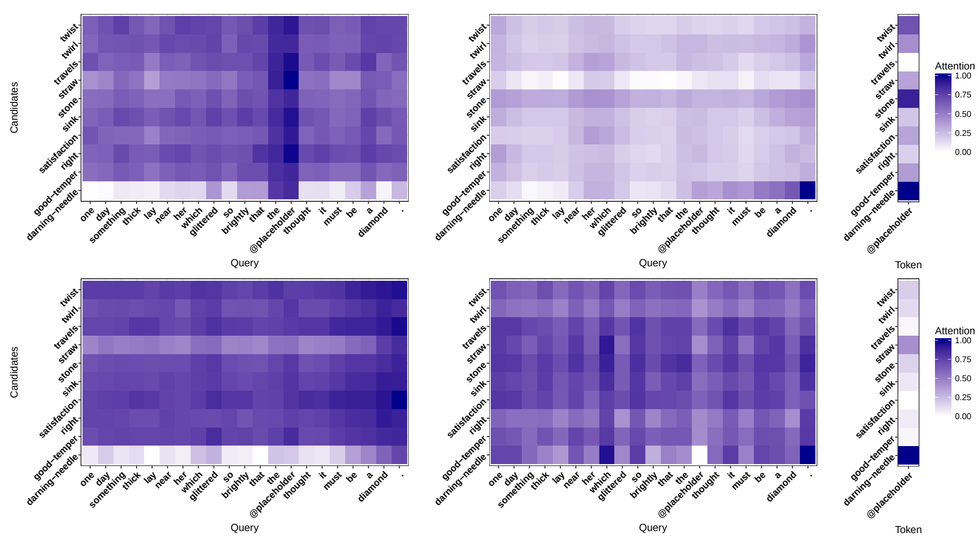


Figure 4.12: Layer-wise normalized attention visualization of “DGR” (top) and “DGR - (a)& (b) & (c)” (bottom) for a sample from the CBT-NE test set. Darker color illustrates higher attention. Figures only show the aggregated attention of candidates. The gold answer is “darning-needle”.

Chapter 5: Interpreting Recurrent and Attention-based Neural Models: A Case Study on NLI

This chapter describes the work in Ghaeini et al. (2018c) [25].

5.1 Introduction

Deep learning has achieved tremendous success for many NLP tasks. However, unlike traditional methods that provide optimized weights for human understandable features, the behavior of deep learning models is much harder to interpret. Due to the high dimensionality of word embeddings, and the complex, typically recurrent architectures used for textual data, it is often unclear how and why a deep learning model reaches its decisions.

There are a few attempts toward explaining/interpreting deep learning-based models, mostly by visualizing the representation of words and/or hidden states, and their importances (via saliency or erasure) on shallow tasks like sentiment analysis and POS tagging [52, 3, 53, 75]. In contrast, we focus on interpreting the gating and attention signals of the intermediate layers of deep models in the challenging task of Natural Language Inference. A key concept in explaining deep models is saliency, which determines what is critical for the final decision of a deep model. So far, saliency has only been used to illustrate the impact of word embeddings. In this work, we extend this concept to the intermediate layer of deep models to examine the saliency of attention as well as the LSTM gating signals to understand the behavior of these components and their impact on the final decision. We make two main contributions. First, we introduce new strategies for interpreting the behavior of deep models in their intermediate layers, specifically, by examining the saliency of the attention and the gating signals. Second, we provide an extensive analysis of the state-of-the-art model for the NLI task and show that our methods reveal interesting insights not available from traditional methods of inspecting attention and word saliency.

In this work, our focus was on NLI, which is a fundamental NLP task that requires both understanding and reasoning. Furthermore, the state-of-the-art NLI models employ complex neural architectures involving key mechanisms, such as attention and repeated reading, widely seen in successful models for other NLP tasks. As such, we expect our methods to be potentially useful for other natural understanding tasks as well.

5.2 Task and Model

In NLI [7], we are given two sentences, a premise and a hypothesis, the goal is to decide the logical relationship (*Entailment*, *Neutral*, or *Contradiction*) between them.

Many of the top performing NLI models [26, 87, 71, 61, 30, 99, 11] are variants of the ESIM model [11], which we choose to analyze in this work. ESIM reads the sentences independently using LSTM at first, and then applies attention to align/contrast the sentences. Another round of LSTM reading then produces the final representations, which are compared to make the prediction. Detailed description of ESIM can be found

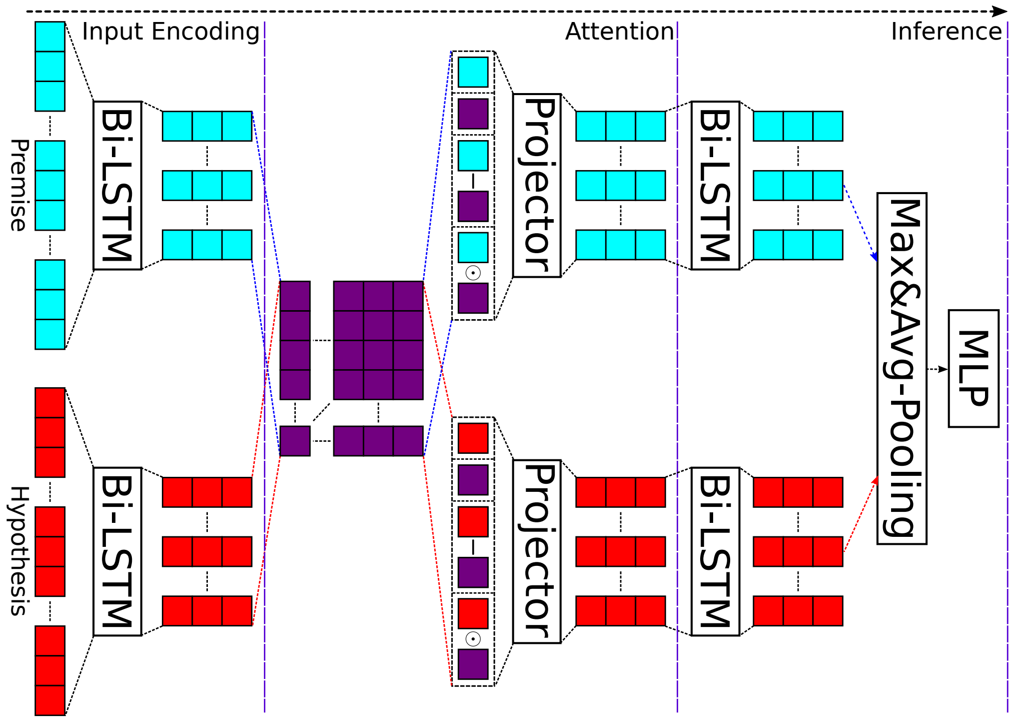


Figure 5.1: A high-level view of ESIM model.

in the next subsection.

Using the SNLI [7] data, we train two variants of ESIM, with dimensionality 50 and 300 respectively, referred to as ESIM-50 and ESIM-300 in the remainder of this work.

5.2.1 ESIM

Here we describe the ESIM model. We divide ESIM to three main parts: 1) input encoding, 2) attention, and 3) inference. Figure 5.1 demonstrates a high-level view of the ESIM framework.

Let $u = [u_1, \cdots, u_n]$ and $v = [v_1, \cdots, v_m]$ be the given premise with length n and hypothesis with length m respectively, where $u_i, v_j \in \mathbb{R}^r$ are word embeddings of r-dimensional vector. The goal is to predict a label y that indicates the logical relationship between premise u and hypothesis v. Below we briefly explain the aforementioned parts.

5.2.1.1 Input Encoding

It utilizes a bidirectional LSTM (BiLSTM) for encoding the given premise and hypothesis using Equations 5.1 and 5.2 respectively.

$$\hat{u} = \textit{BiLSTM}(u) \tag{5.1}$$

$$\hat{v} = \textit{BiLSTM}(v) \tag{5.2}$$

where $\hat{u} \in \mathbb{R}^{n \times 2d}$ and $\hat{v} \in \mathbb{R}^{m \times 2d}$ are the reading sequences of u and v respectively.

5.2.1.2 Attention

It employs a soft alignment method to associate the relevant sub-components between the given premise and hypothesis. Equation 5.3 (energy function) computes the unnormalized attention weights as the similarity of hidden states of the premise and hypothesis.

$$e_{ij} = \hat{u}_i \hat{v}_j^T, \quad i \in [1,n], j \in [1,m] \tag{5.3}$$

where $\hat{u}_i$ and $\hat{v}_j$ are the hidden representations of u and v respectively which are computed earlier in Equations 5.1 and 5.2. Next, for each word in either premise or hypothesis, the relevant semantics in the other sentence is extracted and composed according to e_{ij}. Equations 5.4 and 5.5 provide formal and specific details of this procedure.

$$\tilde{u}_i = \sum_{j=1}^{m} \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{ik})} \hat{v}_j, \quad i \in [1,n] \tag{5.4}$$

$$\tilde{v}_j = \sum_{i=1}^{n} \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{kj})} \hat{u}_i, \quad j \in [1,m] \tag{5.5}$$

where $\tilde{u}_i$ represents the extracted relevant information of $\hat{v}$ by attending to $\hat{u}_i$ while $\tilde{v}_j$ represents the extracted relevant information of $\hat{u}$ by attending to $\hat{v}_j$. Next, it passes the enriched information through a projector layer which produce the final output of attention stage. Equations 5.6 and 5.7 formally represent this process.

$$\begin{aligned} a_i &= [\hat{u}_i, \tilde{u}_i, \hat{u}_i - \tilde{u}_i, \hat{u}_i \odot \tilde{u}_i] \\ p_i &= \boldsymbol{ReLU}(W_p a_i + b_p) \end{aligned} \tag{5.6}$$

$$\begin{aligned} b_j &= [\hat{v}_j, \tilde{v}_j, \hat{v}_j - \tilde{v}_j, \hat{v}_j \odot \tilde{v}_j] \\ q_j &= \boldsymbol{ReLU}(W_p b_j + b_p) \end{aligned} \tag{5.7}$$

Here $\odot$ stands for element-wise product while $W_p \in \mathbb{R}^{8d \times d}$ and $b_p \in \mathbb{R}^d$ are the trainable weights and biases of the projector layer respectively. p and q indicate the output of attention devision for premise and hypothesis respectively.

5.2.1.3 Inference

During this phase, it uses another BiLSTM to aggregate the two sequences of computed matching vectors, p and q from the attention stage (Equations 5.8 and 5.9).

$$\hat{p} = \textit{BiLSTM}(p) \tag{5.8}$$

$$\hat{q} = \textit{BiLSTM}(q) \tag{5.9}$$

where $\hat{p} \in \mathbb{R}^{n \times 2d}$ and $\hat{q} \in \mathbb{R}^{m \times 2d}$ are the reading sequences of p and q respectively. Finally the concatenation max and average pooling of $\hat{p}$ and $\hat{q}$ are pass through a multi-layer perceptron (MLP) classifier that includes a hidden layer with *tanh* activation and *softmax* output layer. The model is trained in an end-to-end manner.

5.3 Visualization of Attention and Gating

In this work, we are primarily interested in the internal workings of the NLI model. In particular, we focus on the attention and the gating signals of LSTM readers, and how they contribute to the decisions of the model.

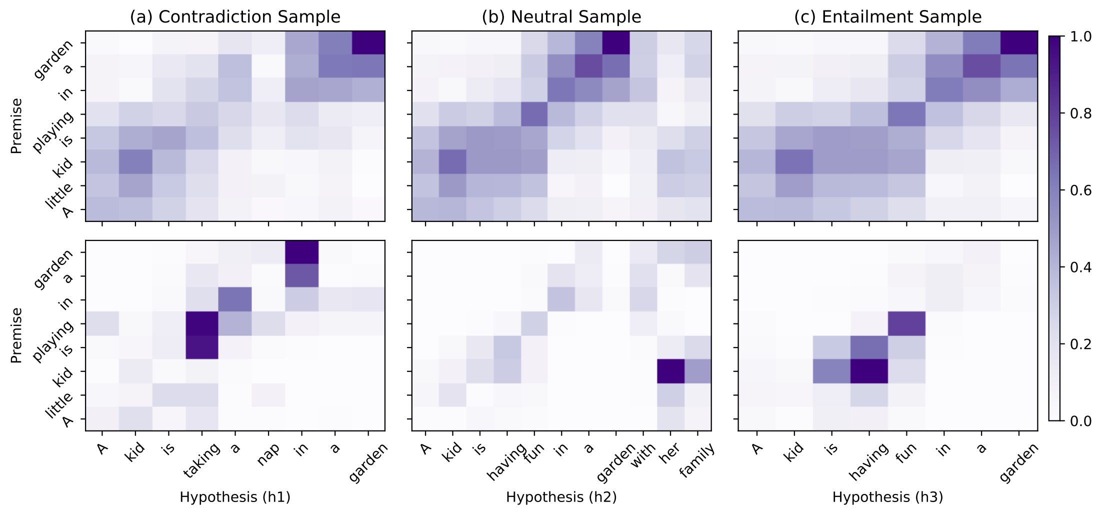


Figure 5.2: Normalized attention and attention saliency visualization. Each column shows visualization of one sample. Top plots depict attention visualization and bottom ones represent attention saliency visualization. Predicted (the same as Gold) label of each sample is shown on top of each column.

5.3.1 Attention

Attention has been widely used in many NLP tasks [22, 19, 4] and is probably one of the most critical parts that affects the inference decisions. Several pieces of prior work in NLI have attempted to visualize the attention layer to provide some understanding of their models [26, 69]. Such visualizations generate a heatmap representing the similarity between the hidden states of the premise and the hypothesis (Equation 5.3). Unfortunately the similarities are often the same regardless of the decision.

Let us consider the following example, where the same premise *"A kid is playing in the garden"*, is paired with three different hypotheses:

h1: *A kid is taking a nap in the garden* h2: *A kid is having fun in the garden with her family*

h3: *A kid is having fun in the garden*

Note that the ground truth relationships are Contradiction, Neutral, and Entailment, respectively.

The first row of Figure 5.2 shows the visualization of normalized attention for the three cases produced by ESIM-50, which makes correct predictions for all of them. As we can see from the figure, the three attention maps are fairly similar despite the completely different decisions. The key issue is that the attention visualization only allows us to see how the model aligns the premise with the hypothesis, but does not show *how such alignment impacts the decision.* This prompts us to consider the saliency of attention.

5.3.1.1 Attention Saliency

The concept of saliency was first introduced in vision for visualizing the spatial support on an image for a particular object class [83]. In NLP, saliency has been used to study the importance of words toward a final decision [52] .

We propose to examine the saliency of attention. Specifically, given a premise-hypothesis pair and the model's decision y, we consider the similarity between a pair of premise and hypothesis hidden states e_{ij} as a variable. The score of the decision $S(y)$ is thus a function of e_{ij} for all i and j. The saliency of e_{ij} is then defined to be $|\frac{\partial S(y)}{\partial e_{ij}}|$.

The second row of Figure 5.2 presents the attention saliency map for the three examples acquired by the same ESIM-50 model. Interestingly, the saliencies are clearly different across the examples, each highlighting different parts of the alignment. Specifically, for h1, we see the alignment between "is playing" and "taking a nap" and the alignment of "in a garden" to have the most prominent saliency toward the decision of Contradiction. For h2, the alignment of "kid" and "her family" seems to be the most salient for the decision of Neutral. Finally, for h3, the alignment between "is having fun" and "kid is playing" have the strongest impact toward the decision of Entailment.

From this example, we can see that by inspecting the attention saliency, we effectively pinpoint which part of the alignments contribute most critically to the final prediction whereas simply visualizing the attention itself reveals little information.

5.3.1.2 Comparing Models

In the previous examples, we study the behavior of the same model on different inputs. Now we use the attention saliency to compare the two different ESIM models: ESIM-50 and ESIM-300.

Consider two examples with a shared hypothesis of *"A man ordered a book"* and premise:

p1: *John ordered a book from amazon*

p2: *Mary ordered a book from amazon*

Here ESIM-50 fails to capture the gender connections of the two different names and predicts Neutral for both inputs, whereas ESIM-300 correctly predicts Entailment for the first case and Contradiction for the second.

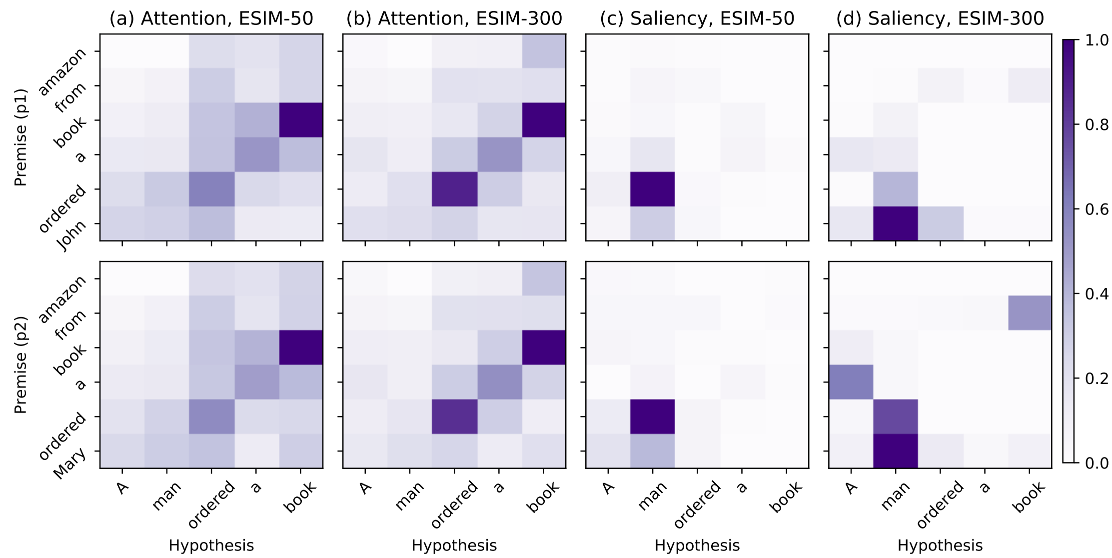


Figure 5.3: Normalized attention and attention saliency visualizations of two examples (p1 and p2) for ESIM-50 (a) and ESIM-300 (b) models. Each column indicates visualization of a model and each row represents visualization of one example.

In the first two columns of Figure 5.3 (column a and b) we visualize the attention of the two examples for ESIM-50 (left) and ESIM-300 (right) respectively. Although the two models make different predictions, their attention maps appear qualitatively similar.

In contrast, columns 3-4 of Figure 5.3 (column c and d) present the attention saliency for the two examples by ESIM-50 and ESIM-300 respectively. We see that for both examples, ESIM-50 primarily focused on the alignment of "ordered", whereas ESIM-300 focused more on the alignment of "John" and "Mary" with "man". It is interesting to note that ESIM-300 does not appear to learn significantly different similarity values compared to ESIM-50 for the two critical pairs of words ("John", "man") and ("Mary", "man") based on the attention map. The saliency map, however, reveals that the two models use these values quite differently, with only ESIM-300 correctly focusing on them.

5.3.1.3 More Attention Study

Here we provide more examples on the NLI task which intend to examine specific behavior in this model. Such examples (Figures 5.4, 5.5, 5.6, 5.7, 5.8) indicate interesting observation that we can analyze them in the future works. Table 1 shows the list of all example.

ID	Premise	Hypothesis	Gold	Prediction	Category
1	Six men, two with shirts and four without, have taken a break from their work on a building.	Seven men, two with shirts and four without, have taken a break from their work on a building.	Contradiction	Contradiction	Counting
2	two men with shirts and four men without, have taken a break from their work on a building.	Six men, two with shirts and four without, have taken a break from their work on a building.	Entailment	Entailment	Counting
3	Six men, two with shirts and four without, have taken a break from their work on a building.	Six men, four with shirts and two without, have taken a break from their work on a building.	Contradiction	Contradiction	Counting
4	A man just ordered a book from amazon.	A man ordered a book yesterday.	Neutral	Neutral	Chronology
5	A man ordered a book from amazon 30 hours ago.	A man ordered a book yesterday.	Entailment	Entailment	Chronology

Table 5.1: Examples along their gold labels, ESIM-50 predictions and study categories.

5.3.2 LSTM Gating Signals

LSTM gating signals determine the flow of information. In other words, they indicate how LSTM reads the word sequences and how the information from different parts is captured and combined. LSTM gating signals are rarely analyzed, possibly due to their high dimensionality and complexity. In this work, we consider both the gating signals

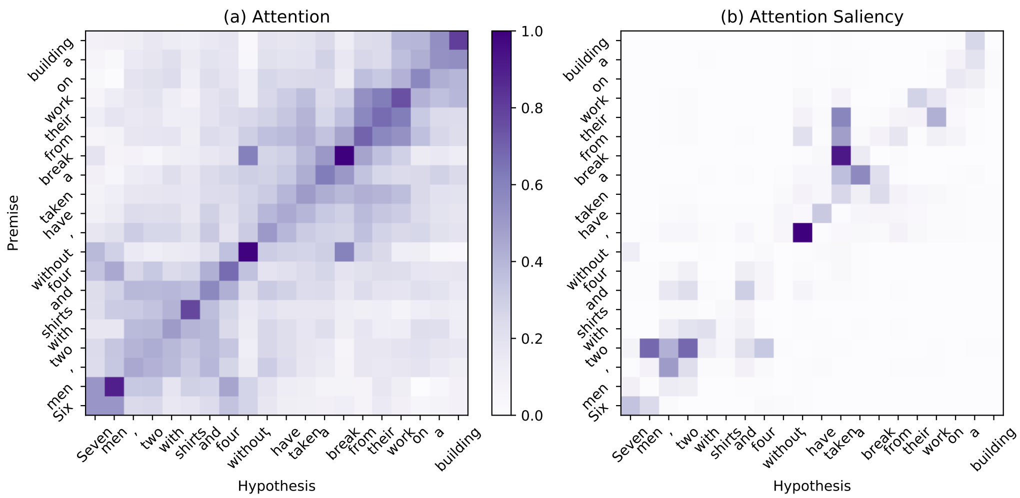


Figure 5.4: Normalized attention (a) and saliency attention (b) visualizations of Example 1. The gold relationship for this example is Contradiction. ESIM-50 also predicts Contradiction for this example.

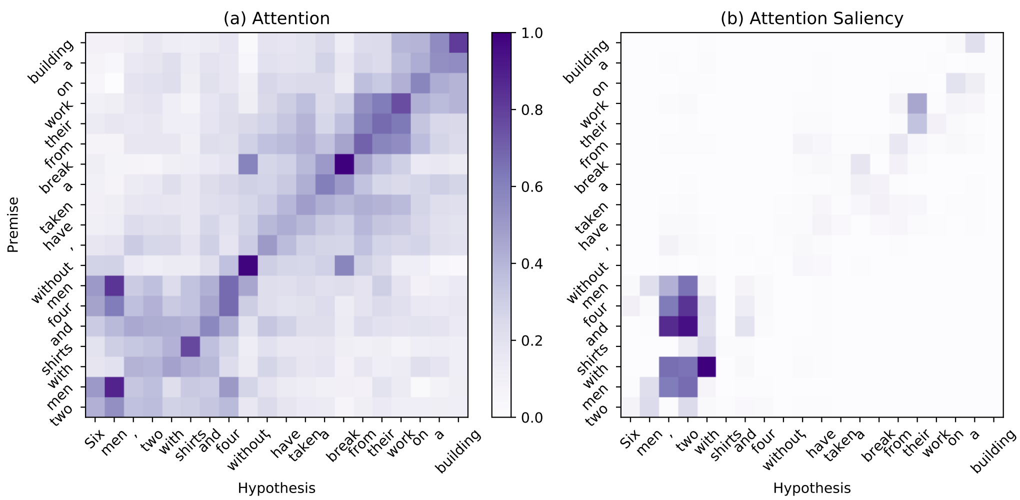


Figure 5.5: Normalized attention (a) and saliency attention (b) visualizations of Example 2. The gold relationship for this example is Entailment. ESIM-50 also predicts Entailment for this example.

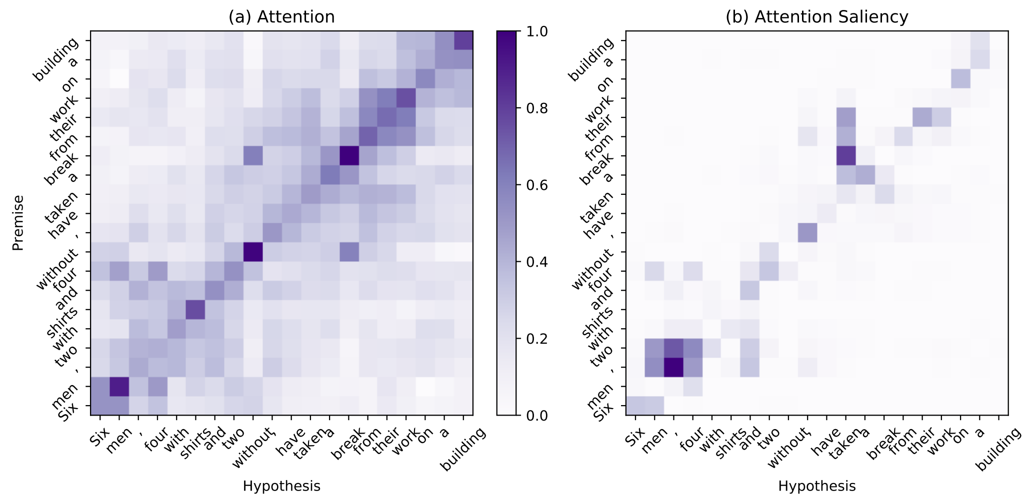


Figure 5.6: Normalized attention (a) and saliency attention (b) visualizations of Example 3. The gold relationship for this example is Contradiction. ESIM-50 also predicts Contradiction for this example.

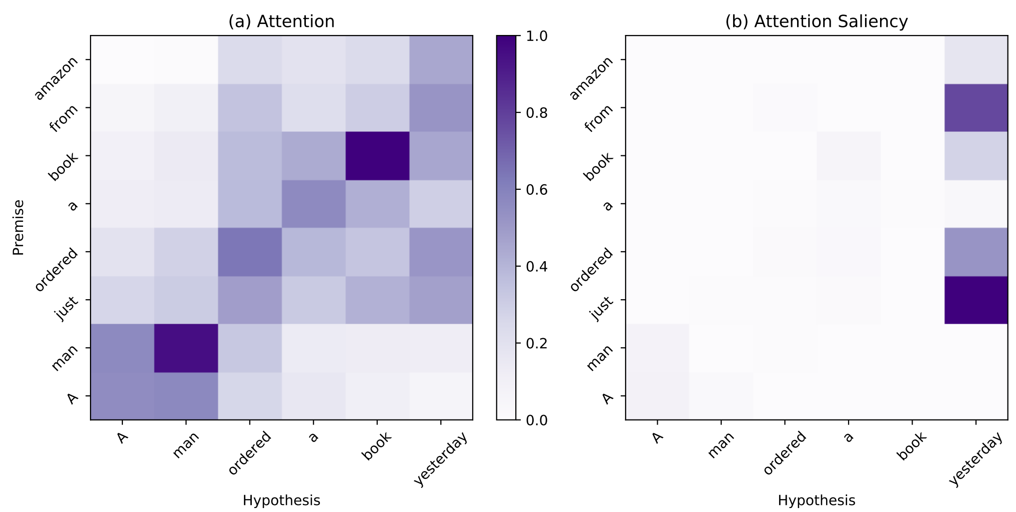


Figure 5.7: Normalized attention (a) and saliency attention (b) visualizations of Example 4. The gold relationship for this example is Neutral. ESIM-50 also predicts Neutral for this example.

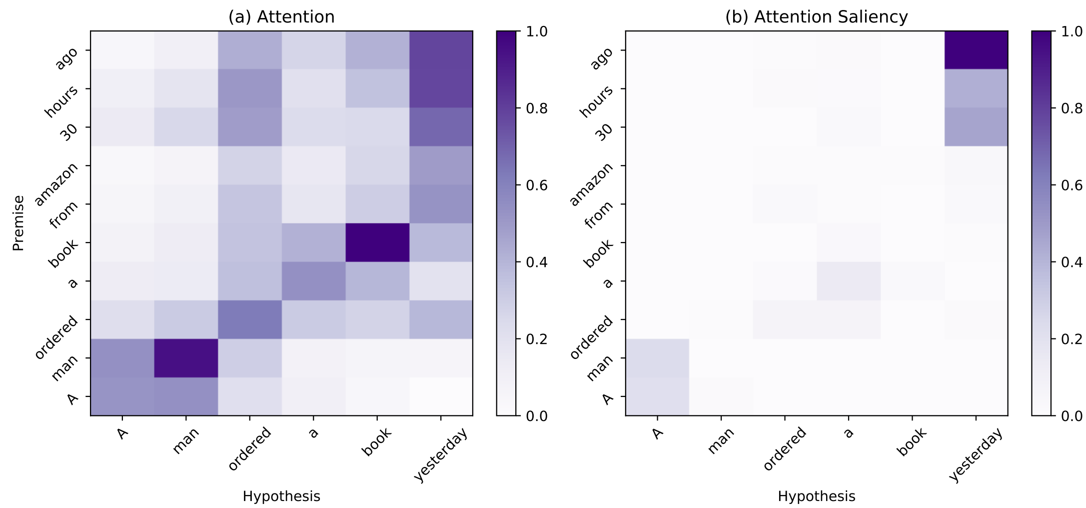


Figure 5.8: Normalized attention (a) and saliency attention (b) visualizations of Example 5. The gold relationship for this example is Entailment. ESIM-50 also predicts Entailment for this example.

and their saliency, which is computed as the partial derivative of the score of the final decision with respect to each gating signal.

Instead of considering individual dimensions of the gating signals, we aggregate them to consider their norm, both for the signal and for its saliency. Note that ESIM models have two LSTM layers, the first (input) LSTM performs the input encoding and the second (inference) LSTM generates the representation for inference.

In Figure 5.9 we plot the normalized signal and saliency norms for different gates (input, forget, output)[1] of the Forward input (bottom three rows) and inference (top three rows) LSTMs. These results are produced by the ESIM-50 model for the three examples of Section 3.1, one for each column.

[1]We also examined the memory cell but it shows very similar behavior with the output gate and is hence omitted.

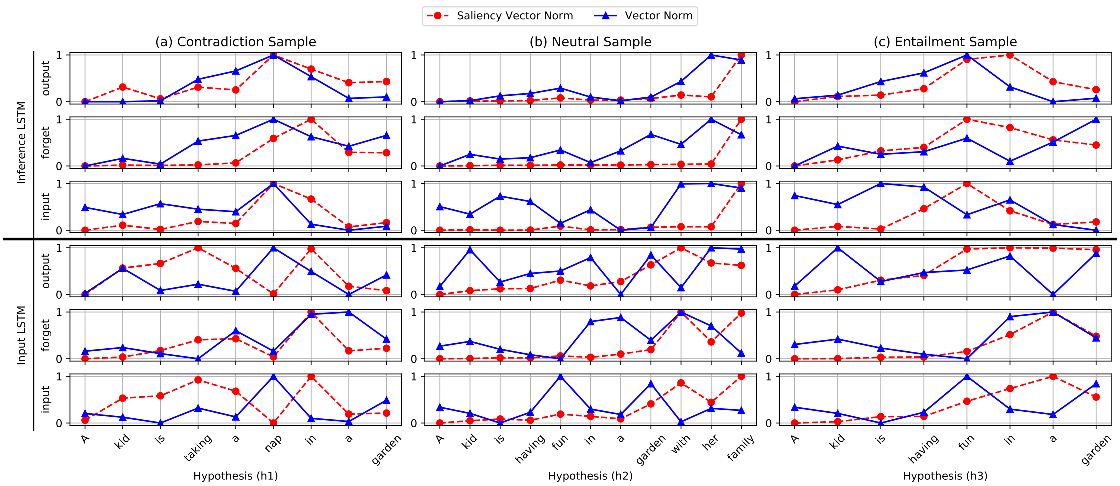


Figure 5.9: Normalized signal and saliency norms for the input and inference LSTMs (forward) of ESIM-50 for three examples. The bottom (top) three rows show the signals of the input (inference) LSTM. Each row shows one of the three gates (input, forget and output).

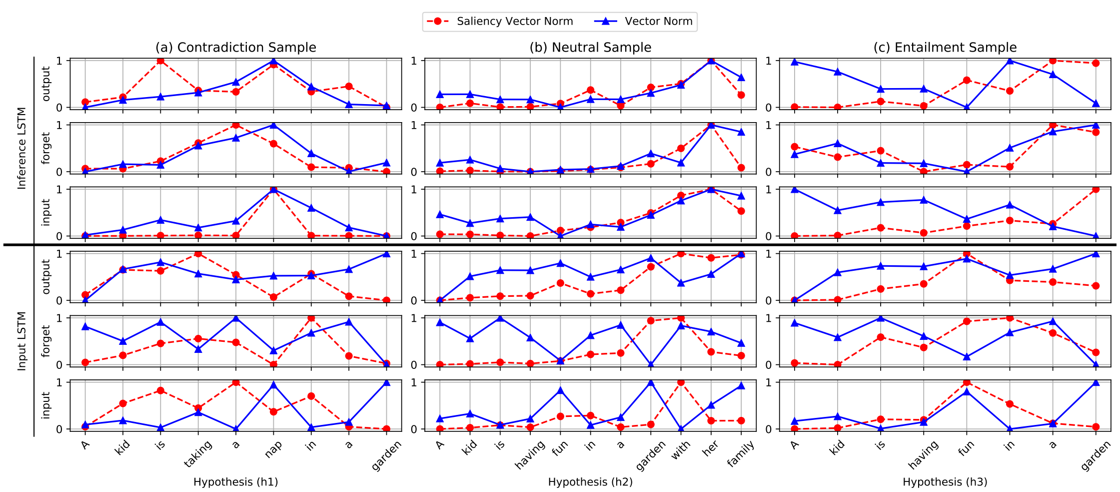


Figure 5.10: Normalized signal and saliency norms for the input and inference LSTMs (backward) for three examples, one for each column. The bottom (top) three rows show the signals of the input (inference) LSTM, where each row shows one of the three gates (input, forget and output).

From the figure, we first note that the saliency tends to be somewhat consistent across different gates within the same LSTM, suggesting that we can interpret them jointly to identify parts of the sentence important for the model's prediction.

Comparing across examples, we see that the saliency curves show pronounced differences across the examples. For instance, the saliency pattern of the Neutral example is significantly different from the other two examples, and heavily concentrated toward the end of the sentence ("with her family"). Note that without this part of the sentence, the relationship would have been Entailment. The focus (evidenced by its strong saliency and strong gating signal) on this particular part, which presents information not available from the premise, explains the model's decision of Neutral.

Comparing the behavior of the input LSTM and the inference LSTM, we observe interesting shifts of focus. In particular, we see that the inference LSTM tends to see much more concentrated saliency over key parts of the sentence, whereas the input LSTM sees more spread of saliency. For example, for the Contradiction example, the input LSTM sees high saliency for both "taking" and "in", whereas the inference LSTM primarily focuses on "nap", which is the key word suggesting a Contradiction. Note that ESIM uses attention between the input and inference LSTM layers to align/contrast the sentences, hence it makes sense that the inference LSTM is more focused on the critical differences between the sentences. This is also observed for the Neutral example as well.

It is worth noting that, while revealing similar general trends, the backward LSTM can sometimes focus on different parts of the sentence (Figure 5.10), suggesting the forward and backward readings provide complementary understanding of the sentence.

5.4 Conclusion

We propose new visualization and interpretation strategies for neural models to understand how and why they work. We demonstrate the effectiveness of the proposed strategies on a complex task (NLI). Our strategies are able to provide interesting insights not achievable by previous explanation techniques. Our future work will extend our study to consider other NLP tasks and models with the goal of producing useful insights for further improving these models.

Chapter 6: Attentional Multi-Reading Sarcasm Detection

This chapter describes the work in Ghaeini et al. (2018d) [24].

6.1 Introduction

Recently, dialogue systems have received a lot of attention from researchers. Unfortunately, existing approaches often fail to detect sarcastic user comments in order to provide proper responses.

Sarcasm detection is an important and challenging task for natural language understanding. The goal of sarcasm detection is to determine whether a sentence is sarcastic or non-sarcastic. Sarcasm is a type of phenomenon with specific perlocutionary effects on the hearer [33], such as to break their pattern of expectation. Consequently, correct understanding of sarcasm often requires a deep understanding of multiple sources of information, including the utterance, the conversational context, and, frequently some real world facts. Table 6.1 shows three different sarcastic samples from the SARC dataset [47], each of which requires a different source of information for disambiguation.

Existing approaches for sarcasm detection primarily focus on lexical, pragmatic cues (e.g. interjections, punctuations, sentimental shift etc.) found in utterance [49, 41]. In contrast, the natural language understanding aspect of sarcasm detection could be more robust, interesting and challenging. Moreover, most sarcasm detection systems have

<table>
<tr><th>Type</th><th colspan="2">Sample</th></tr>
<tr><td rowspan="2">U.S.[a]</td><td>C[d]</td><td>just don't. if you are telling anyone else what they can and can't put on their bodies, just don't</td></tr>
<tr><td>R[e]</td><td>we're on Reddit, don't you know we control everything people do?</td></tr>
<tr><td rowspan="2">C.D.[b]</td><td>C</td><td>who else thinks that javascript alert is an annoying, lazy, and ugly way to notify me of something on your site.</td></tr>
<tr><td>R</td><td>it's a useful debugging tool</td></tr>
<tr><td rowspan="2">E.K.D.[c]</td><td>C</td><td>till that some cattle ranchers in south dakota lost between 20% - 50% of their livestock in winter storm atlas, and may not be eligible for insurance due to the expiration of the farm bill and federal government shutdown.</td></tr>
<tr><td>R</td><td>this is clearly barrack hussein obama's fault, since he refuses to modify the aca and obamacare.</td></tr>
</table>

[a]**U.S.**, Utterance Sufficient.
[b]**C.D.**, Conversation Dependent.
[c]**E.K.D.**, External Knowledge Dependent.
[d]**C**, Comment.
[e]**R**, Response.

Table 6.1: Different types of sarcastic examples from the SARC dataset. Each data sample contains a comment and response. Important and influential tokens are shown in blue.

considered utterances in isolation [17, 31, 55, 79, 60, 42, 29, 43, 27, 73, 2, 34]. However, even humans have difficulty in recognizing sarcastic intent when considering an utterance in isolation [96]. There are some limited attempts toward taking the conversational context into account [28] by using a variety of LSTMs [39] to encode both context and reply sentences. Still such approaches only focuses on the conversation dependent samples.

In this work, we propose an end-to-end model that combines information from both the utterance and the conversational context to detect sarcasm. Considering the utterance beside the conversational context enables the model to (1) properly handle utterance-sufficient samples, (2) automatically extract lexical and grammatical features from the utterance. First, We demonstrate the effectiveness of our model through empirical evaluations on the SARC dataset [47], the largest available dataset for sarcasm detection. Next, we illustrate the impact of different aspects of the proposed model through an ablation study. Finally, we present an extensive data analysis to (1) provide explanations regarding our model's decisions and behavior by visualizing attention and attention saliency[25]; (2) study the impact and effect of utterance and the conversational context on our model's final prediction. In summary, our contributions are as follows:

- Proposing a novel end-to-end and interpretable deep learning model that combines information from both the utterance and conversational context in parallel.
- Illustrating the impact of the proposed model's component through an extensive ablation study.
- Explaining the model's behavior and predictions by visualization of the attention and attention saliency.
- Examining the impact of utterance and conversational context on the model's final predictions.

6.2 Related Work

Automatic sarcasm detection is a relatively recent field of research. Early studies use small datasets and leverage lexical and syntactic features for sarcasm detection [41].

Here we classify the previous works into three categories, isolate-utterance based, contextual-feature based, and conversation based sarcasm detection models.

- **Isolate-utterance based:** Most existing sarcasm detection systems consider the utterances in isolation [17, 31, 55, 79, 60, 42, 29, 43, 27]. Methods in this category commonly rely on hand-designed features, syntactic patterns, and lexical cues.

- **Contextual-feature based:** Wallace et al. (2014) illustrates the necessity of using contextual information in sarcasm detection by showing how traditional classifiers fail in instances where humans also require additional context. Consequently, researchers recently started to exploit contextual information for sarcasm detection. In particular, contextual information about authors, topics or conversational context have been considered [46, 5, 95, 74, 73, 105, 2, 34]. Such techniques rely on either feature engineering or embedding-based representation via deep learning.

  These approaches benefit from contextual information in a pipelined and feature based manner. We should note that user profiling has been shown to have noticeable impact on sarcasm detection [34]. However, user profiling is not always possible. In this work, we are primarily interested in the language side of the sarcasm detection and aim to provide an end-to-end user/author independent system that could be used in a variety of applications, especially dialogue systems and chat boxes.

- **Conversation-based:** The last category of methods aims to detect sarcasm based on the understanding of the conversation (other than simply extracting features from the context). To the best of our knowledge, there is just one conversation dependent sarcasm detection system [28], which focuses on modeling conversational context using a variety of LSTMs to help sarcasm detection. They effectively demonstrated the importance and impact of considering conversational context for sarcasm detection.

Among all previous works, Ghosh et al. (2017) and our system share similar intuition and motivation. However, we utilize a different deep learning architecture to address sarcasm detection. Furthermore, we consider the utterance in both isolation and conversation dependent settings. Such a strategy allows the model to (1) extract lexical and grammatical features from the utterance, and (2) selectively attend to the proper source of information. Finally, we evaluate our system with a much larger and broader dataset that could lead to more robust and unbiased evaluation.

6.3 Model

The inputs to our model are $u = [u_1, \cdots, u_n]$ and $v = [v_1, \cdots, v_m]$, which are the given comment (length n) and response (length m) respectively. Here $u_i, v_j \in \mathbb{R}^r$ are r-dimensional word embedding vectors. The goal is to predict a label y that indicates whether the response v is sarcastic or non-sarcastic.

Our proposed model (**A**ttentional **M**ulti-**R**eading system; AMR) consists of an utterance-only (left side) part and a conversation-dependent (right side) part, formulated with the following major components: input encoding, attention, re-reading, and classification. Figure 6.1 demonstrates a high-level view of our proposed AMR framework.

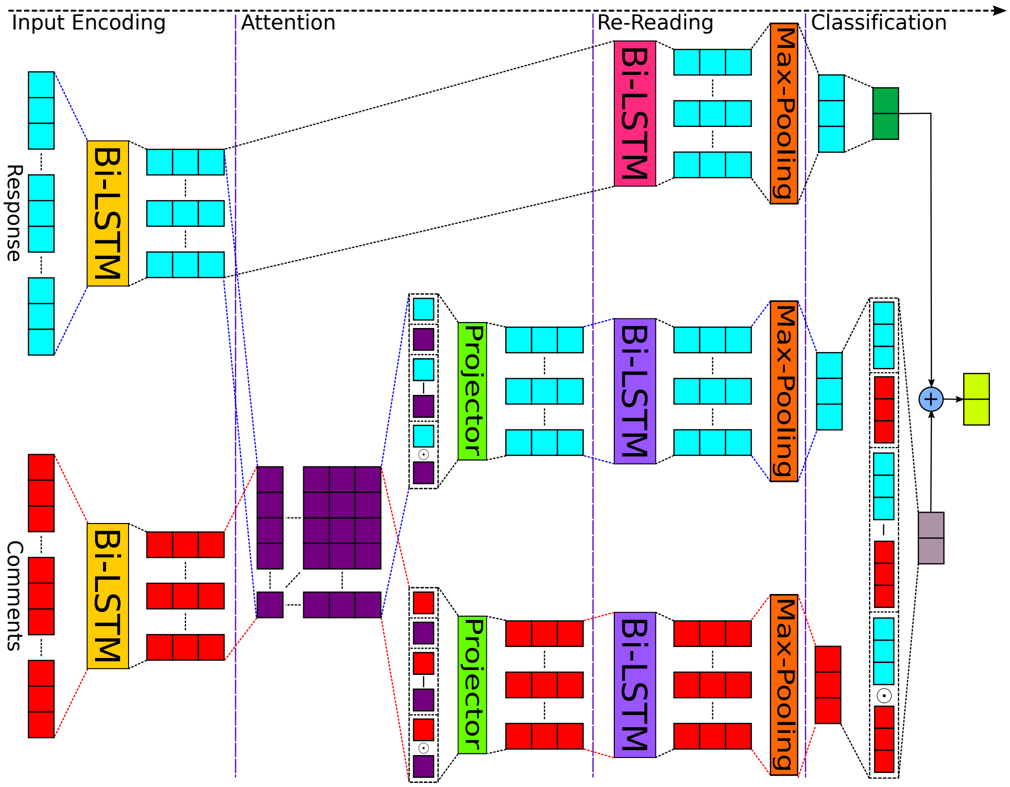


Figure 6.1: A high-level view of our model (AMR). The data (comment u and response v, depicted with red and cyan/blue tensors respectively) flows from bottom to top. Relevant tensors are shown with the same color and elements with the same colors share parameters. The left part shows the utterance-only part and the right part represents the conversation-dependent part of AMR.

6.3.1 Input Encoding

RNNs provide a natural solution for modeling variable length sequences and have shown to be successful in various NLP tasks [22, 26, 4, 21]. Consequently, we utilize a bidirectional LSTM (BiLSTM) [39] for encoding the given comment and response. Here we simply read and encode the comment and response using a BiLSTM. Equations 6.1 and 6.2 formally represent this component.

$$\bar{u} = \textit{BiLSTM}(u) \tag{6.1}$$

$$\bar{v} = \textit{BiLSTM}(v) \tag{6.2}$$

where $\bar{u} \in \mathbb{R}^{n \times 2d}$ and $\bar{v} \in \mathbb{R}^{m \times 2d}$ are the BiLSTM reading sequences of u and v respectively.

6.3.2 Attention

Here we employ a soft alignment method to associate the relevant sub-components between the given comment and response. The unnormalized attention weights are computed as the similarity of the hidden states of the comment and response as shown in Equation 6.3 (energy function).

$$e_{ij} = \bar{u}_i \bar{v}_j^T, \quad i \in [1, n], j \in [1, m] \tag{6.3}$$

where $\bar{u}_i$ and $\bar{v}_j$ are the hidden representations of u and v respectively which are computed earlier in Equations 6.1 and 6.2 respectively. Next, for each word in either comment or response, the relevant semantics in the other sentence is extracted and composed according to e_{ij} as shown in Equations 6.4 and 6.5.

$$\tilde{u}_i = \sum_{j=1}^{m} \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{ik})} \bar{v}_j, \quad i \in [1, n] \tag{6.4}$$

$$\tilde{v}_j = \sum_{i=1}^{n} \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{kj})} \bar{u}_i, \quad j \in [1, m] \tag{6.5}$$

where $\tilde{u}_i$ represents the extracted relevant information of $\bar{v}$ by attending to $\bar{u}_i$ while $\tilde{v}_j$ represents the extracted relevant information of $\bar{u}$ by attending to $\bar{v}_j$.

6.3.2.1 Attention Augmentation and Projection

To utilize the collected attentional information $\tilde{u}_j$ and $\tilde{v}_j$, a trivial next step would be to concatenate them with $\bar{u}_i$ and $\bar{v}_j$ respectively. More over, it is often interesting to compare and contrast the information from the comment and the response in order to detect sarcasm. Hence, we calculate the element-wise difference and element-wise and include these vectors for further consideration. We concatenate all the vectors and represent the comment and response as $[\bar{u}_i, \tilde{u}_i, \bar{u}_i - \tilde{u}_i, \bar{u}_i \odot \tilde{u}_i]$ and $[\bar{v}_j, \tilde{v}_j, \bar{v}_j - \tilde{v}_j, \bar{v}_j \odot \tilde{v}_j]$ with $i = 1, ..., n$ and $j = 1, ..., m$ respectively. Finally, a feed-forward neural layer with the ReLU activation function projects the concatenated vectors from the $8d$-dimensional vector space into a d-dimensional vector space (Equations 6.6 and 6.7). This projection layer serves the dual purpose of both helping the model to capture deeper dependencies between the comment and response and lowering the complexity of vector representations.

$$p_i = \textbf{\textit{ReLU}}(W_c([\bar{u}_i, \tilde{u}_i, \bar{u}_i - \tilde{u}_i, \bar{u}_i \odot \tilde{u}_i]) + b_c) \tag{6.6}$$

$$q_j = \textbf{\textit{ReLU}}(W_c([\bar{v}_j, \tilde{v}_j, \bar{v}_j - \tilde{v}_j, \bar{v}_j \odot \tilde{v}_j]) + b_c) \tag{6.7}$$

Here $\odot$ stands for element-wise product while $W_c \in \mathbb{R}^{8d \times d}$ and $b_c \in \mathbb{R}^d$ are the trainable weights and biases of the projector layers respectively.

6.3.3 Re-Reading

During this phase, two BiLSTMs are used. First, we use a shared BiLSTM (*BiLSTM*$_c$) to aggregate the sequences of computed matching vectors, p and q from the *Attention* stage. This aggregation is performed in a sequential manner to ensure that sequential information in the latent variables is retained. Second, We use another BiLSTM to re-read and re-encode the previous encoding of the response from the *Input Encoding* section ($\bar{v}$). Such a re-reading process is helpful toward achieving a deeper and more meaningful representation for the response when considered in isolation. The Re-Reading procedure is done through Equations 6.8, 6.9, and 6.10.

$$\bar{p} = \textbf{\textit{BiLSTM}}_c(p) \tag{6.8}$$

$$\bar{q} = \textbf{\textit{BiLSTM}}_c(q) \tag{6.9}$$

$$\bar{x} = \textbf{\textit{BiLSTM}}_u(\bar{v}) \tag{6.10}$$

Finally, we convert $\bar{p} \in \mathbb{R}^{n \times 2d}$, $\bar{q} \in \mathbb{R}^{m \times 2d}$ and $\bar{x} \in \mathbb{R}^{m \times 2d}$ to fixed-length vectors using a max pooling layer (Equations 6.11, 6.12, and 6.13).

$$\tilde{p} = \mathit{MaxPooling}(\bar{p}) \tag{6.11}$$

$$\tilde{q} = \mathit{MaxPooling}(\bar{q}) \tag{6.12}$$

$$\tilde{x} = \mathit{MaxPooling}(\bar{x}) \tag{6.13}$$

where $\tilde{p} \in \mathbb{R}^{2d}$, $\tilde{q} \in \mathbb{R}^{2d}$ are the final and fixed representations of the comment and the response produced via conversation-dependent reading (the right part of the model), and $\tilde{x} \in \mathbb{R}^{2d}$ is a separate representation of the response produced by the utterance-only reading (the left portion of the model).

6.3.4 Classification

To make final prediction, we consider both the utterance-only representation as well as the conversation dependent representations. Equation 6.14 represents a feed-forward layer that computes the utterance-only prediction from $\tilde{x}$. For the conversation-dependent part, we enrich the extracted information from the comment and response by incorporating the difference and element-wise product of $\tilde{p}$ and $\tilde{q}$ respectively. Equation 6.15 formally describes the prediction procedure for the conversation-dependent part.

$$o_u = U_u \tilde{x} + a_u \tag{6.14}$$

$$o_c = U_c([\tilde{p}, \tilde{q}, \tilde{p} - \tilde{q}, \tilde{p} \odot \tilde{q}]) + a_c \tag{6.15}$$

where $U_u \in \mathbb{R}^{2d \times 2}$, $U_c \in \mathbb{R}^{8d \times 2}$, $a_u \in \mathbb{R}^2$ and $a_c \in \mathbb{R}^2$ are the trainable weights and biases of the prediction layers respectively. Finally, we combine both predictions (i.e. o_u and o_c) using a trainable weight α (Equation 6.16).

$$output = \textbf{\textit{Softmax}}(o_u + \alpha o_c) \tag{6.16}$$

The model is trained in an end-to-end manner. More detailed information about the architecture and training can be found in the following section.

6.4 Experiments and Evaluation

6.4.1 Dataset

SARC[1] [47] is a self-annotated corpus for sarcasm detection. SARC is the largest available sarcasm detection dataset for this task and contains more than a million of sarcastic/non-sarcastic samples extracted from Reddit[2]. Every instance in SARC is a response to a set of comments. The response is annotated by its author as either sarcastic or non-sarcastic. In this work, we concatenate all of the available comments for each

[1] http://nlp.cs.princeton.edu/SARC/
[2] https://www.reddit.com/

		non-sarcastic	sarcastic
Train	Data Size	128,541	128,541
	# Avg. Comment	60.9	60.9
	# Avg. Response	55.0	54.5
Test	Data Size	32,333	32,333
	# Avg. Comment	60.8	60.8
	# Avg. Response	55.8	54.7
Vocabulary		95,043	

Table 6.2: SARC main balanced V2.0 statistics.

response in chronological order into a single comment.

We evaluate our system on the latest version of the balanced SARC (SARC V2.0, Main balanced). Due to the lack of a pre-defined validation set, we randomly hold out 10% of the training set data as our validation set. All hyper-parameters are tuned based on the performance on the validation set. Table 6.2 shows the SARC (V2.0) dataset statistics.

The motivation behind using the SARC dataset as our primary benchmark is three-fold: (1) SARC is the largest available dataset for sarcasm detection. Consequently, SARC is the most appropriate dataset for training a sophisticated deep-learning based model. Also, due to its size, the evaluation results could be considered more robust and unbiased. (2) SARC is specifically developed to investigate the necessity of contextual information in sarcasm detection in realistic settings. This characteristic aligns well with the motivation of our work. (3) This dataset is author-annotated and has a small false-positive rate for the sarcastic labels [47], thus providing reliable annotations. Importantly, its self-annotation characteristic avoid annotation errors induced by third-party annotators.

6.4.2 Experimental Setup

We use the pre-trained 300-D Glove $840B$ vectors [70] to initialize our word embedding vectors. All hidden states of BiLSTMs for both input encoding and re-reading have 300 dimensions ($r = 300$ and $d = 300$). The weights are learned by minimizing the log-loss (Equation 6.17) on the training data via the Adam optimizer [48]. The initial learning rate is 0.0001. To avoid overfitting, we use dropout [86] with the rate of 0.5 for regularization, which is applied to all feedforward connections. During training, the word embeddings are updated to learn effective representations for the sarcasm detection task. We use a fairly small batch size of 32 to provide more exploration power to the model. We consider 200 and 100 as the maximum acceptable length of the comment and response respectively ($n \leq 200$ and $m \leq 100$). In other words, only 200 and 100 words of the given comment and response is processed and the rest (in case of existence) are thrown away.

$$
\begin{aligned}
y_i^* &= \textit{argmax}(output_i) \\
l &= -\frac{1}{N}\sum_{i=0}^{N}(y_i \log(y_i^*) + (1 - y_i)\log(1 - y_i^*))
\end{aligned}
\tag{6.17}
$$

6.4.3 Results

Here we evaluate our model based on two versions of SARC. (1) [34] is the most recent work that use SARC dataset for evaluation. It is not clear which version of SARC is

Model	Test Set	
	F1	Accuracy
(1) Bag of Words	64%	63%
(2) CNN	66%	65%
(3) CASCADE − Personality Feature	66%	68%
(4) CNN-SVM [73]	68%	68%
(5) CUE-CNN [2]	69%	70%
(6) CASCADE [34]	77%	77%
(7) **Ours (AMR)**	**68%**	**70%**

Table 6.3: F1-measures and Accuracies of models on the test set of SARC$_{csd}$. The second three (4,5, and 6) models benefit from personality feature (their results are shown in blue). Whereas the first three models (1,2, and 3), similar to our model; only rely on response or response and comment. Our models (AMR) achieves the F1-measure and accuracy of 68% and 70% respectively, the best results observed on SARC$_{csd}$ among similar methods which does not use personality features.

used, but they have released their train and test sets[3]. We refer to this dataset as SARC$_{csd}$ in the rest of this work. In this sub-section (Results), we use SARC$_{csd}$ to compare our system with the reported performances in [34]. (2) We use the SARC V2.0 in next section (Ablation and Configuration Study) to report standard results on SARC V2.0 and compare the performance of different configurations of our model.

Table 6.3 shows the F1-measures and accuracies of models on the test set of SARC$_{csd}$. The first row shows the results of a baseline classifier using the bag-of-words method. All other listed models are deep learning based. The second model is a simple CNN applied to the given utterance/response. The third system is the CASCADE model [34] without using the personality features. This system use the context in a pipeline manner via a discourse feature vector. The next three reported models benefit from stylometric

[3]https://github.com/SenticNet/CASCADE–Contextual-Sarcasm-Detection

and personality features (The result of such methods are shown in blue).

Bag-of-words approach obtained the lowest performance whereas all deep learning based models outperform it. Among all deep learning ones, the CNN baseline has the lowest performance. The CNN baseline only relies on the given utterance/response highlighting the impact and importance of considering both comment and response in the disambiguation process.

Comparing methods that benefit from personality features and user profiling (4,5, and 6) with the ones that do not (1,2, and 3), it is clear that such features are very helpful for sarcasm detection. However, user profiling helps a model primarily by providing information about the user's behavior or how the user forms sarcastic sentences. In other words, it does not really enrich the model's capability toward understanding what constructs sarcasm in general. More over, user history and information may not always be available for extracting such features. Importantly, one of the main goals of this work is to move toward solving the sarcasm **understanding** issue in a dialog system. In particular, we are mainly interested in the language understanding aspect of sarcasm detection. As such, we aim to build an end-to-end system that does not depend on any additional information or assumption (user profiling, topic modeling, etc.) other that the sequence of the sentences (the conversation). Due to these considerations, the fair comparison would be comparing the results of our system with the fist three models in Table 6.3, which demonstrates the effectiveness of our models.

From Table 6.3 we can see that AMR achieves an F1-measure and accuracy of 68% and 70% respectively on the test set of SARC$_{csd}$, which are the best reported results among the existing comparable baselines for sarcasm detection. Here we obtain 2% im-

Models	SARC V2.0 Test Set			
	Precision	**Recall**	**F1-Measure**	**Accuracy**
(01) AMR	69.33%	69.64%	**69.48%**	**69.45%**
(02) Conversation-dependent	70.23%	66.36%	68.24%	69.11%
(03) Utterance-only	70.86%	64.66%	67.62%	69.04%
(04) AMR − Attention	69.39%	68.79%	69.09%	69.22%
(05) AMR − Re-Reading	72.93%	60.20%	65.96%	68.93%
(06) AMR − Re-Reading − Attention	**74.76%**	55.31%	63.58%	68.32%
(07) AMR − difference	70.07%	67.53%	68.78%	69.34%
(08) AMR − Element-Wise product	70.41%	67.01%	68.67%	69.42%
(09) AMR − E-W product − difference	71.19%	65.50%	68.23%	**69.45%**
(10) AMR with only E-W product	70.75%	65.05%	67.78%	69.07%
(11) AMR − train embedding	67.22%	**69.68%**	68.43%	67.85%

Table 6.4: Ablation study results. Precision, Recall, F1-Measure, and Accuracy of different models on the test set of SARC V2.0.

provement on both F1-measure and accuracy on the test data of SARC$_{csd}$ in comparison with the previous state-of-the-art system; CASCADE without personality feature (row 3 in the Table 6.3). It is interesting to note that although we do not employ user profiling, our performance is similar and competitive with several baselines that use user profiling (CNN-SVM [73] and CUE-CNN [2]).

6.4.4 Ablation and Configuration Study

In this section, we conduct an ablation and configuration study of our model to examine the importance and effect of each major component. We report the performance (Precision, Recall, F1-Measure, and Accuracy) of different variants of our model on the test set of SARC V2.0 in Table 6.4.

The first row shows the performance of the proposed model, AMR. Rows 2 and 3 study the impact of the conversation-dependent and utterance-only parts of the models. Rows 4-6 examine the impact of attention and re-reading stages by removing either one (rows 4 and 5) or both components (row 6). Rows 7-10 investigate the effect of data augmentation in attention and classification of conversation-dependent part of the proposed model. Specifically, we consider removing the different data augmentations shown in Equation 6.6, 6.7, and 6.15. Finally, row 11 shows the result of our model without fine-tuning the word embedding during the training procedure.

First, we compare the models based on their F1-Measure and Accuracy. The results show that removing any part of our model leads to reduced test set performance both in terms of F1-Measure and accuracy (expect for row 9 where accuracy remained the same), indicating the usefulness of these components in general.

We observe that AMR performs noticeably better than both *Utterance-only* and *Conversation-dependent* configurations, validating the intuition of our design. It is noteworthy that *Conversation-dependent* model performs better than the other one, suggesting the importance of considering the conversation and context for this task. Comparison of rows 4, 5, and 6 suggests that although both of *Attention* and *Re-Reading* are important, but *Re-Reading* has a more significant impact on the performance of AMR.

A closer look into the precisions and recalls of the different models suggests an interesting trend — removing different components of the model typically leads to improved precision in sarcasm detection but suffers from significantly reduced recall. This is evidenced by the results of the first 10 rows. Comparing the first three rows, it is interesting to note that either part of the model (conversation-dependent or utterance-only) individually achieves slightly higher precision but significantly lower recall. The fact that by combining the two our model was able to achieve significantly improved recall suggests that the two parts were able to detect different types of sarcasms, which is consistent with our intuition.

Removing fine-tuning of the word embedding during the training has an opposite effect with reduced precision but little or no impact on the recall. This suggests that by fine tuning the word embeddings for the sarcasm detection task, we were able to increase the specificity of the sarcasm detector without sacrificing the sensitivity.

6.5 Analysis

In this section, we first show visualization of the energy functions (i.e. attention) in the attention stage (Equation 6.3) and its saliency for an instance from the SARC V2.0 test set. Next, we study the performance of our system (Utterance-only, Conversation-dependent and AMR) against the length of comment and response.

6.5.1 Attention Study

Here we show a visualization of the normalized attention (Equation 6.3) and normalized attention saliency[4] in Figure 6.2.

We show a comment and response pair, where the comment is *"man accidentally shoots himself when concealed weapon goes off in movie theater."*, and the response is *"just another responsible gun owner exercising his rights under the 2nd amendment."* which is a sarcastic response and AMR identifies it as sarcastic response as well. At-

[4]For more details refer to Ghaeini et al. (2018c)[25]

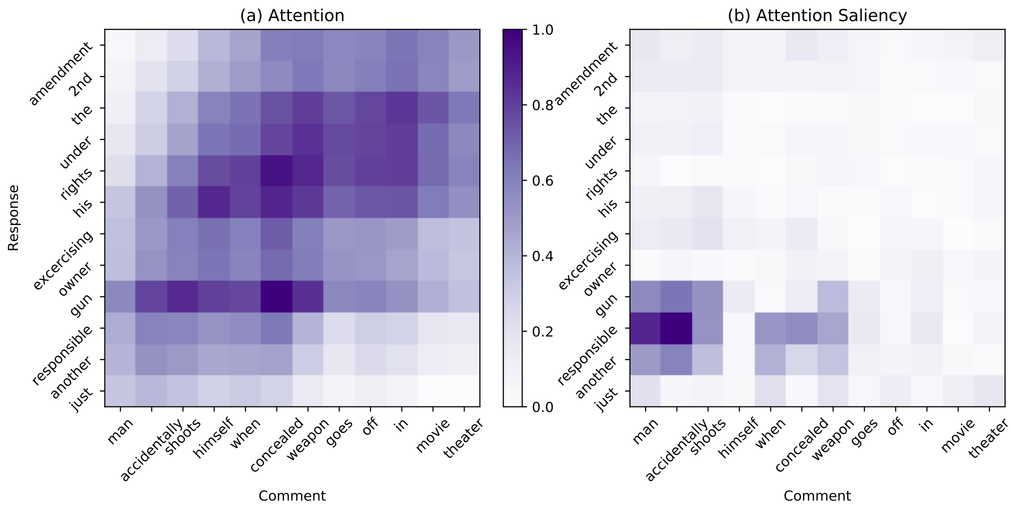


Figure 6.2: Normalized attention (a, top) and normalized attention saliency (b, bottom) visualization for a sarcastic instance from the test set of SARC V2.0.

tention visualization in Figure 6.2 indicates that the model could successfully attend to relevant pairs of words like <gun, shoots>, <gun, concealed>, <gun, weapon>, <his, himself>, etc. However, still we cannot clearly explain the model's prediction. Thus we use the attention saliency to visualize the impact of each word pair toward the model's prediction. Attention saliency is the absolute value of the partial derivative of the model prediction respect to the attention. Larger saliency indicates stronger impact on the model's prediction. According to the attention saliency visualization in Figure 6.2 (b), the phrase pair of <another responsible gun, man accidentally shoots> has the highest impact toward identifying the aforementioned example as sarcastic, which is consistent with human intuition. This demonstrates and verifies the model's ability in understanding comment and response and then utilizing the crucial relationships between the comment and response for identifying sarcastic responses. The word "responsible" in the

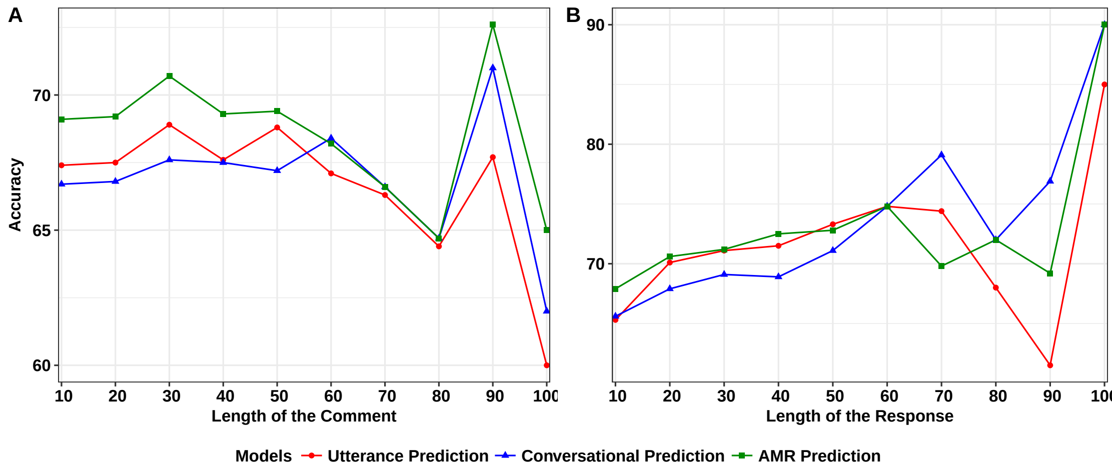


Figure 6.3: Test accuracy of AMR and its sub-parts (Utterance-only and Conversation-dependent) against the length of the comment (A) and response (B).

response appears to be the key phrase that deliver the sarcastic intent of the response — when paired with the phrase "man accidentally shoots" we see the highest saliency, suggesting the most significant impact toward the final prediction.

6.5.2 Length Study

One of the advantage of our model is its prediction interpretability. AMR contains two major parts; *Utterance-only* and *Conversation-dependent*. Each part makes its own prediction. Then AMR combines utterance-only and conversation-dependent predictions using a trainable variable α to obtains its final prediction. Consequently, the impact of each part toward the final prediction can be computed. In other words, we can determine which part affects the final prediction the most.

Figure 6.3 depicts the performance of AMR (green line), Utterance-only part (red line), and Conversation-dependent part (blue lines) against length of the comment (A, left), and length of the response (B, right) respectively.

According to Figure 6.3, the utterance-only part provides more accurate predictions for short comments ($n \leq 50$). We believe that the utterance-only part of AMR is capable of automatically extracting useful lexical and grammatical cues from utterance which could be beneficial for detecting sarcastic utterances/responses. Consequently, among samples with short comment; thus less contextual information, the utterance-only part shows better performance. It is noteworthy that the performance of AMR is almost always higher than both utterance-only and conversation-dependent parts. However, the conversation-dependent part performs better for longer comments ($50 < n \leq 200$). This observation is consistent with our expectation because long comments are more likely to have relevant and crucial information for determining the sarcastic intent of the response. Such an analysis verifies the intuition behind the design our model.

Despite of the plot A in Figure 6.3, plot B does not reflect a very coherent behavior and trend among the reported settings. Interestingly, for the very short responses category ($m \leq 10$) which is also the most frequent response category, the conversation-dependent part performs better than the utterance-only part. Due to lack of information in very short responses, disambiguation of such samples are usually reliant on the comment. If we ignore the aforementioned category ($m \leq 10$), plot B illustrates similar behavior and trend for utterance-only and conversation-dependent parts. The utterance-only part perform better for short responses ($10 < m \leq 50$) and the conversation-dependent part beats the utterance-only part for long responses ($50 < m \leq 100$).

Overall, Figure 6.3 suggests that the conversation-dependent part performs better when (1) we do not have enough information in the response ($m \leq 10$) or (2) the response or the comment is too long ($n, m > 50$). We believe that in case of dealing with long comment or response, we require some guidance for attending to the important and influential sub-parts of the comment or response. Such a goal can be achieved by utilizing an attention mechanism on both comment and response.

6.6 Conclusion

We propose a novel interpretable end-to-end sarcasm detection model that benefits from both the utterance and the conversational context in parallel. Our evaluations successfully demonstrate the effectiveness of the proposed model. We provide an extensive oblation study that illustrates and justifies the importance and impact of different components of the proposed model. Moreover, we study the model's behavior by visualizing attention and attention saliency. Finally, we present an interesting data analysis to examine the impact of utterance and conversational context on the model's predictions. Our future work will extend our study to include the world fact information in the disambiguation procedure to produce more robust and accurate predictions.

Chapter 7: Gated BERT: Toward Interpreting and Understanding BERT

This chapter describes an unpublished work done during an internship at Microsoft Research in 2019.

7.1 Introduction

BERT (Bidirectional Encoder Representation from Transformers) [18] is a bidirectional variant of Transformer networks [91]. BERT can be fine-tuned for a wide range of NLP tasks such as natural language inference, sentiment analysing, and paraphrase identification without substantial modification. One of the main baselines for evaluating performance of BERT is the GLUE (General Language Understanding Evaluation) benchmark [97]. The GLUE benchmark contains a variety of sentence- or sentence-pair language understanding tasks such as Linguistic Acceptability, Sentiment Analysing, Paraphrase Identification, Natural Language Inference, etc. The noticeable performance improvement of BERT on the GLUE benchmark compared to previous state-of-the-art methods has attracted a lot of attention to BERT. However, it is unclear how and why it actually works.

There are a few attempts toward studying the behavior of BERT [93, 62, 76, 40, 89]. Voita et al. (2019) and Michel et al. (2019) focus on studying the necessity having all attention heads and layers. Their observations suggest that many of attention heads can be eliminated without a noticeable drop in performance of specific tasks [93, 62]. Coenen et al. (2019), Jawahar et al. (2019), and Tenney et al. (2019) investigate capability of BERT in capturing different linguistic features. Coenen et al. (2019) find evidence of a fine-grained geometric representation of word senses [76]. Jawahar et al. (2019) provide evidences that BERT intermediate layers encode a rich hierarchy of linguistic information, with surface features at the bottom, syntactic features in the middle and semantic features at the top [40]. Finally, Tenney et al. (2019) demonstrate that BERT is capable of extracting linguistic features such as POS tagging, parsing, NER, semantic roles, and coreference [89].

In this work, borrowing from ELMo [71], we introduce a variation of BERT named Gated BERT. The intuition behind Gated BERT is to shed a light on the behaviour of BERT to help us in obtaining more powerful and more reliable method in future. To achieve this goal, we changed the task disambiguation part of BERT – which is simply passing the vector representation of "[CLS]" token of the last layer to a linear feedforward layer – to a layer-wise gated mechanism. Here, we introduce a weight for every layer of the BERT which determines how much that specific layer should influence input of the task disambiguation part (The linear feedforward layer). Each task has its own set of layer weights, so by looking at the values and their update trends, we can approximately identify the purpose and importance of different layers for different tasks. Moreover, we observe improvement on BERT performance on the development and test sets of most GLUE tasks. In parallel, we study the necessity of having all 24 layers of the BERT (large version). We would like to shrink the model while preserving its performance and capability to make BERT feasible to be used in real-world applications.

Finally, we describe the implemented demo for this work which provides a variety of interpretation features to this work.

7.2 Preliminary: BERT

Figure 7.1 depicts a high-level view of the BERT model for classification and regression tasks. BERT for classification and regression tasks could be divided to three major components: Embedding, Transformer Layers, and Prediction.

7.2.1 Embedding

For a given token, its embedding is constructed by summing the corresponding token, segment, and position embeddings. BERT is not a sequential model and it is not capable of distinguishing occurrence of a token in different positions. To simulate the sequential nature of textual data, BERT uses position embeddings. Moreover, the segment embedding enables BERT model to handle cases when we have different segments of data in the input. For example, when the input source is a sentence-pair (e.g. natural language inference that we have "premise" and "hypothesis"). So, each sentence can have a different segment id and embedding which help distinguishing tokens and occurrences in different sentences and segments. Equation 7.1 describes the embedding construction of BERT model.

$$e_t = w_t + p_t + s_t; \quad w_t, p_t, s_t \in \mathbb{R}^d \tag{7.1}$$

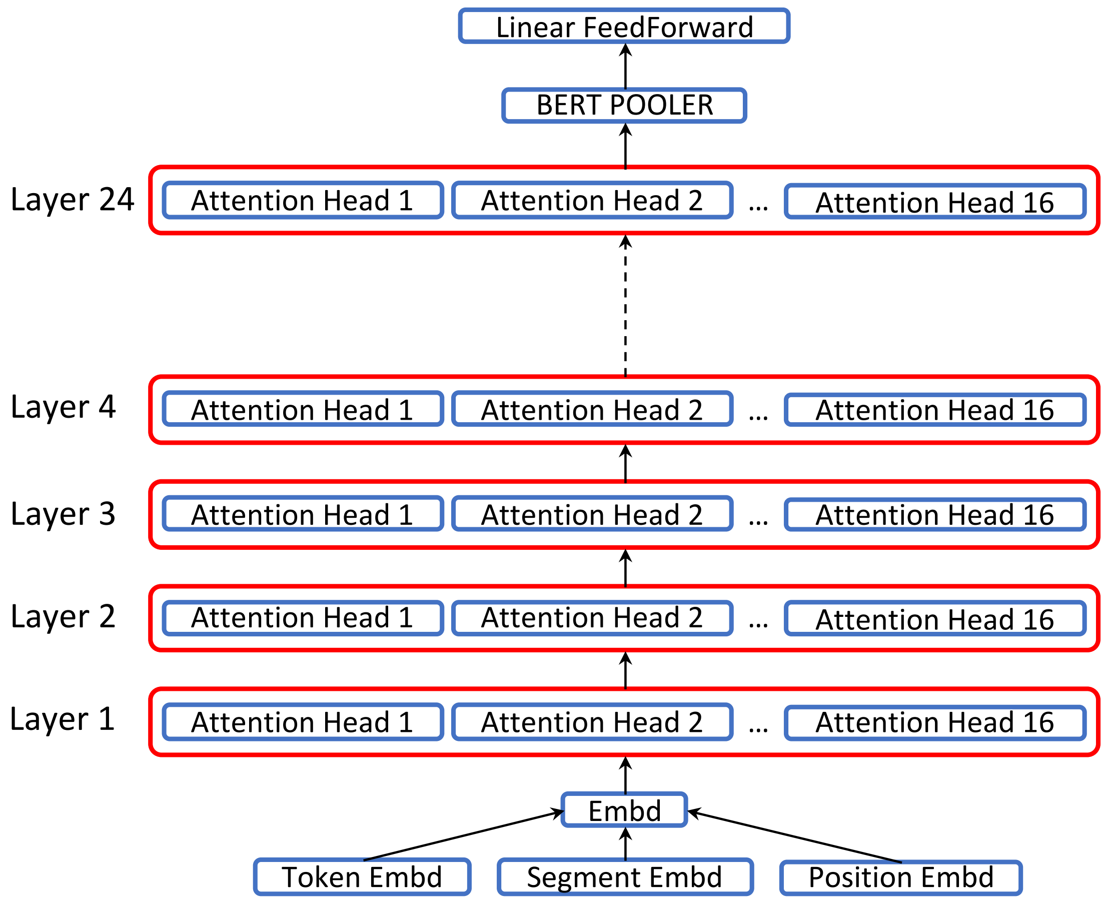


Figure 7.1: A high-level view of BERT model.

where $w_t, p_t, s_t \in \mathbb{R}^d$ are token embedding, position embedding, and segment embedding of token "t" respectively and d is the embedding size and dimension.

7.2.2 Transformer Layer

BERT has two main variation; BERT-base and BERT-large (we use the BERT-large variation in this work). BERT-base and BERT-large have 12 and 24 layers of Transformer network [91] respectively. We omit an exhaustive explanation of Transformer network but in short, Equations 7.2 and 7.3 represent the Transformer network. The Transformer network is a Multi-Head Attention (as shown in Equation 7.2) and each attention head is a Scaled Dot-Product Attention (as shown in Equation 7.3) [1].

$$\begin{aligned} \textit{MultiHead}(Q,K,V) &= \textit{Concat}(\textit{head}_1,\cdots,\textit{head}_h)W^O \\ \textit{where head}_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{7.2}$$

$$\textit{Attention}(Q,K,V) = \textit{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{7.3}$$

Here, Q, K, V are the query, key, and value representation respectively. But in BERT, all these representation are the same ($Q = K = V$). Also, the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d \times d_k}, W_i^K \in \mathbb{R}^{d \times d_k}, W_i^V \in \mathbb{R}^{d \times d_k}$, and $W^O \in \mathbb{R}^{hd_k \times d}$.

7.2.3 Prediction

The first token of every input sequence is always a special classification token ("[CLS]"). The final hidden state corresponding to this token is used as the aggregate sequence representation for disambiguation and decision making of downstream tasks. This special token is feed into a linear feedforward layer. Equation 7.4 describes the prediction process.

[1] Please refer to Vaswani et al. (2017) [91] for more details.

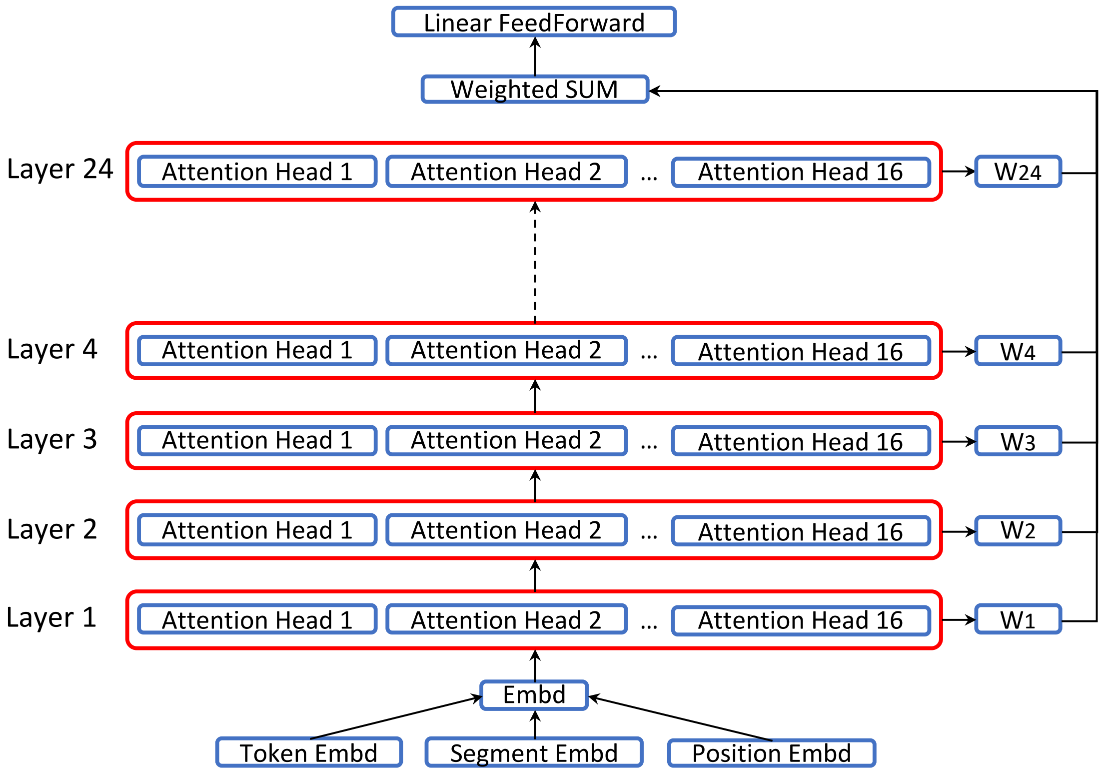


Figure 7.2: A high-level view of Gated BERT (G-BERT) model.

$$Prediction = FF(h_0^L); \quad L = 24, \quad |Prediction| = \#classes \tag{7.4}$$

7.3 Gated BERT

Here, we propose a simple modification to BERT which could potentially yield better performance and also shed a light on behavior of the BERT. As described in Section 7.2.3, BERT uses the final hidden state corresponding to “[CLS]” token (h_0^L) for disam-

Sentence	Label
Rusty talked about himself only after Mary did talk about him.	Correct
Here's a knife with which for you to cut up the onions.	Incorrect

Table 7.1: Two data samples from the CoLA corpus.

biguation and decision making. Here, we introduce a set of weights associated to each layer of BERT ($\alpha_i \in \mathbb{R}$) and then modify the prediction mechanism of BERT (Equation 7.4) to Equation 7.5. According to Equation 7.5, the weighted sum of "[CLS]" tokens of all layers is passed into the linear feedforward layer for disambiguation. Therefore, captured information in each layer could influence the disambiguation and decision making process. Such a method introduces an interpretability capability to the proposed method (Gated BERT). Figure 7.2 illustrates the proposed Gated BERT model.

$$\textit{Prediction} = \textit{FF}(\sum_{i=1}^{L} \alpha_i h_0^i); \quad L = 24 \tag{7.5}$$

7.4 Experiments and Evaluation

7.4.1 Dataset

We evaluate BERT and Gated BERT on multiple tasks from the GLUE benchmark; CoLA, MNLI, MRPC, QNLI, RTE, SST-2, STS-B. Below we describe these tasks and datasets in details:

- CoLA (The Corpus of Linguistic Acceptability): CoLA is a set of 10,657 English sentences labeled as grammatical or ungrammatical from published linguistics literature [100]. The public version contains 9594 sentences belonging to training

ID	Sentence	Label
1	He said the foodservice pie business doesn't fit the company's long-term growth strategy.	Paraphrase
	The foodservice pie business does not fit our long-term growth strategy.	
2	No dates have been set for the civil or the criminal trial.	Non-Paraphrase
	No dates have been set for the criminal or civil cases, but Shanley has pleaded not guilty.	

Table 7.2: Two data samples from the MRPC corpus.

ID	Sentence	Label
1	What came into force after the new constitution was herald?	Entailment
	As of that day, the new constitution heralding the Second Republic came into force.	
2	What is the minimum required if you want to teach in Canada?	Non-Entailment
	In most provinces a second Bachelor's Degree such as a Bachelor of Education is required to become a qualified teacher	

Table 7.3: Two data samples from the QNLI corpus.

Sentence	Label
it's a charming and often affecting journey.	Positive
or doing last year's taxes with your ex-wife.	Negative

Table 7.4: Two data samples from the SST-2 corpus.

ID	Sentence	Score
1	A man with a hard hat is dancing.	5.00
	A man wearing a hard hat is dancing.	
2	A panda is climbing.	1.60
	A man is climbing a rope.	
3	A woman is taking a picture.	0.25
	A man is playing a guitar.	
4	A man is playing a flute.	0.00
	A man is playing a flute.	

Table 7.5: Four data samples from the STS-B corpus.

and development sets, and excludes 1063 sentences belonging to a held out test set. Table 7.1 shows two examples from this corpus.

- MNLI (Multi-Genre Natural Language Inference): MultiNLI is a crowd-sourced collection of 433k sentence pairs annotated with textual entailment information [101]. MNLI is modeled on the SNLI corpus (see section 3.4.1 for more details), but differs in covering a range of genres of spoken and written text, and supports a distinctive cross-genre generalization evaluation.

- MRPC (Microsoft Research Paraphrase Corpus): MRPC corpus is a paraphrase identification dataset. The goal of this task is to identify if two sentences are paraphrases of each other. The evaluation metric for MRPC is accuracy and F1. Table 7.2 shows two examples from this corpus.

- QNLI (Question Natural Language Inference): QNLI task is similar to MNLI in nature. Given a question and a sentence, the goal of QNLI is to determine if the question can be answered by the given sentence (*entailment*) or not (*non-entailment*). Table 7.3 shows two examples from this corpus.

- RTE (Recognizing Textual Entailment): The goal of RTE is the same as MNLI and SNLI.

- SST-2 (The Stanford Sentiment Treebank): This is a sentiment analysing task. SST-2 contains fine grained sentiment labels for 215,154 phrases in the parse trees of 11,855 sentences [84]. Table 7.4 shows two examples from this corpus.

Set	CoLA	MNLI	MRPC	QNLI	RTE	SST-2	STS-B
Train	8,551	392,702	3,668	104,743	2,490	67,349	5,749
Dev	1,043	19,647	408	5,463	277	872	1,500
Test	1,063	19,643	1,725	5,463	3,000	1,821	1,379

Table 7.6: GLUE benchmark Data Statistics

- STS-B (Semantic Textual Similarity Benchmark): This is a semantic textual similarity task and it measures the relatedness of two sentences. The evaluation criterion for this task is Pearson correlation. Table 7.5 shows four examples from this corpus. This is the only regression task in this work.

Finally, Table 7.6 illustrates data statistics of the described datasets.

7.4.2 Training

BERT and Gated BERT (embeddings and 24 transformer layers) are initialized using the pre-trained weights. We train the BERT and Gated BERT with the same settings and compare their results when (1) embeddings and 24 layers weights are fixed (first category; Fixed BERT and G-BERT) and when (2) we fine-tune whole parameters (second category; BERT and G-BERT + Fine-Tuning). Figure 7.3 and 7.4 demonstrate fixed parameters and trainable parameters of Fixed BERT and Gated BERT respectively. The gray parts are fixed and the blue parts are trainable and will be updated during the training. Also, Figure 7.5 represents a high-level illustration of Fixed BERT and Gated BERT training procedure and parts.

We use two initialization methods for the introduced weights ($\alpha_i \in \mathbb{R}$). The first method is called *average initialization* ("*avg*" for short). In this method, all weight are

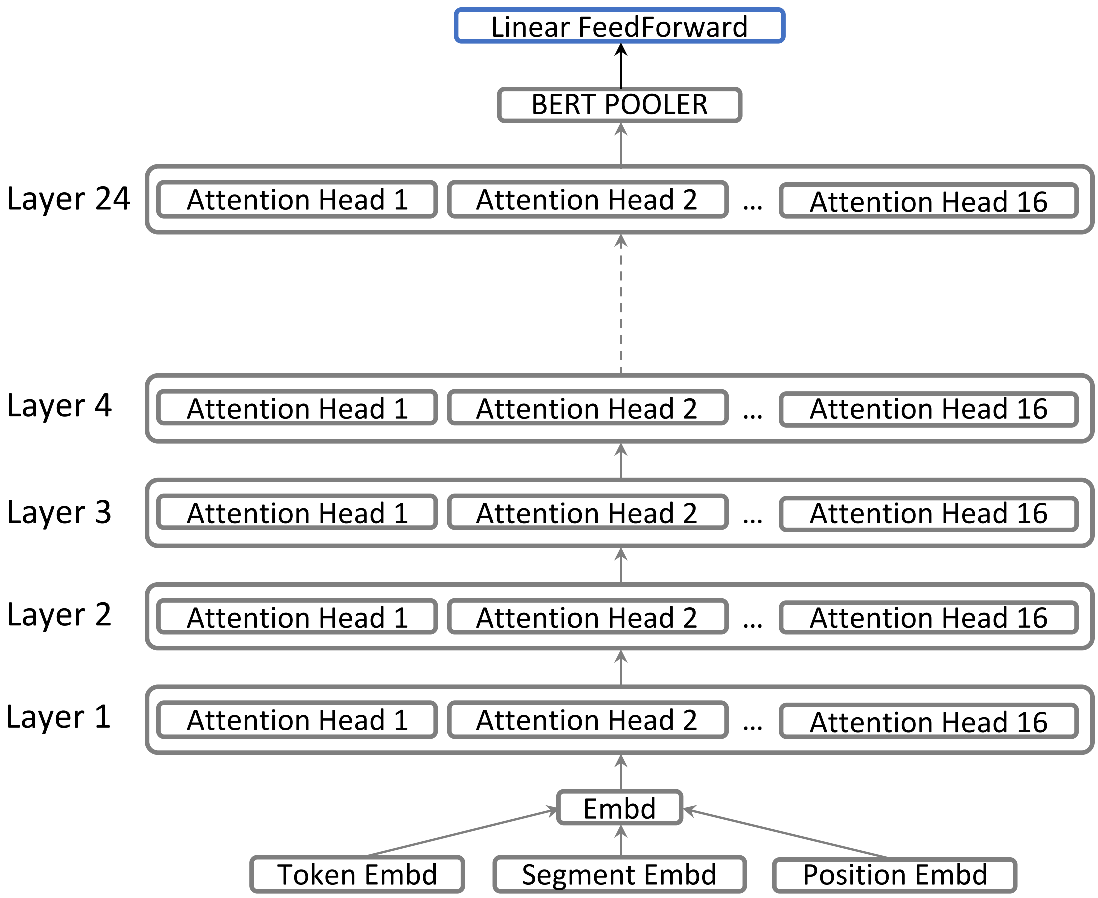


Figure 7.3: Demonstration of fixed and trainable parts of the Fixed BERT. Gray parts are fixed and Blue parts are trainable and will be updated during the training.

uniformly sampled from $[\frac{1}{L} - 0.001, \frac{1}{L} + 0.001]$ distribution where L is the number of transformer layers (Figure 7.6). The second one is called *last initialization* ("*last*" for short). In this method, last layer weight (α_L) is set to 1.0 and the rest of them ($\alpha_i, i \in \{1, \cdots, L-1\}$) are uniformly sampled from $[-0.001, 0.001]$ distribution (Figure 7.7).

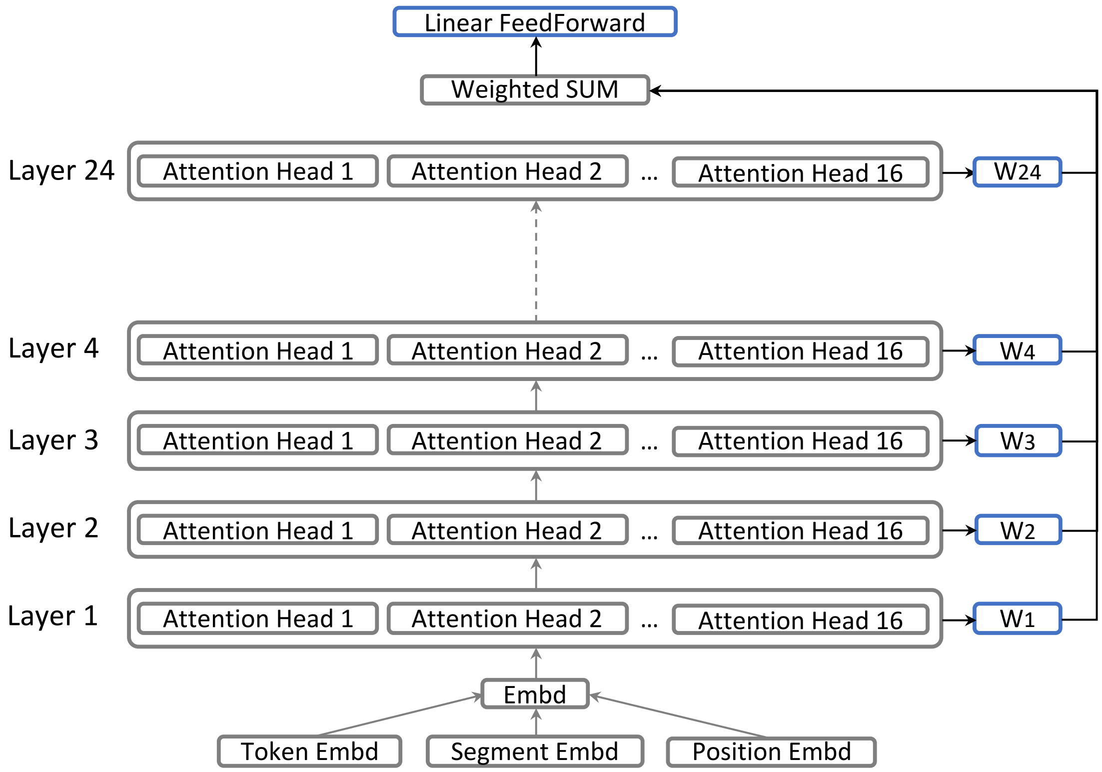


Figure 7.4: Demonstration of fixed and trainable parts of the Gated BERT. Gray parts are fixed and Blue parts are trainable and will be updated during the training.

7.4.3 Experimental Results

Tables 7.7 and 7.8 shows the development results of Fixed BERT, Gated BERT (average initialization), Gated BERT (last initialization), BERT, Gated BERT (average initialization) + Fine-Tuning, and Gated BERT (last initialization) + Fine-Tuning on development set and test set of GLUE benchmark respectively.

According to Tables 7.7 and 7.8, the general trend indicates that Gated BERT performs better than BERT (with and without fine-tuning). Moreover, Gated BERT (aver-

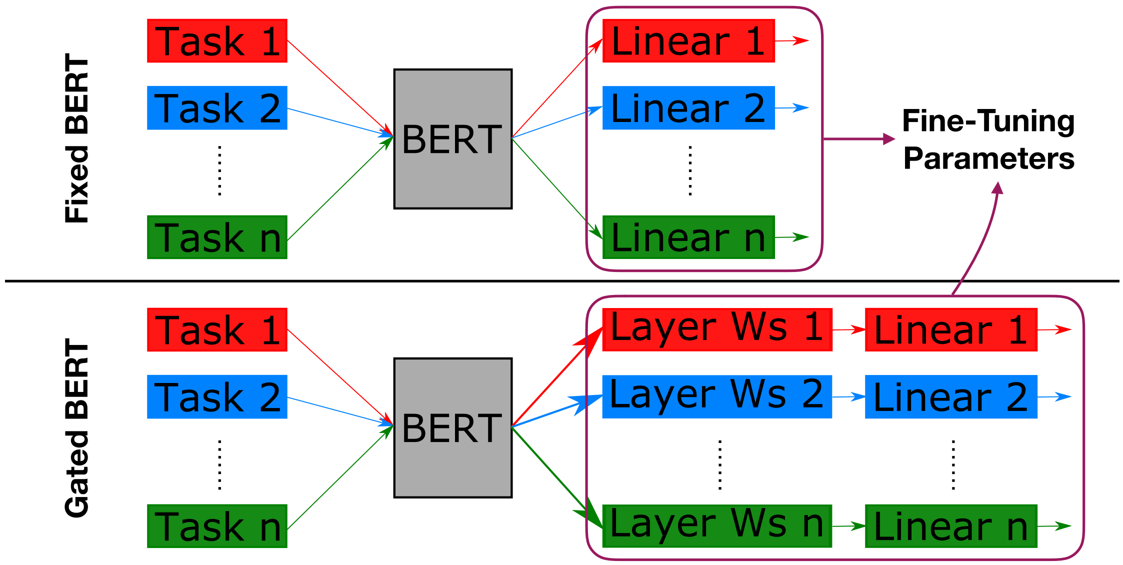


Figure 7.5: High-level demonstration of fixed and trainable parts of the Fixed BERT and Gated BERT.

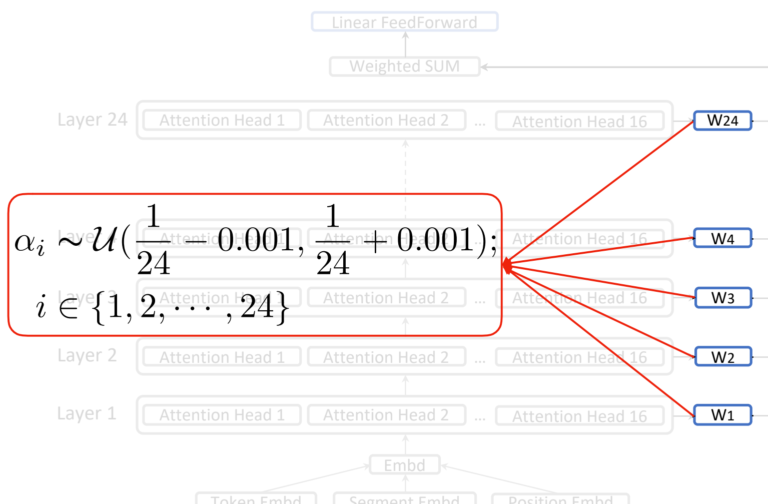


Figure 7.6: Layer weights *average initialization* visualization.

age initialization) performs better than Gated BERT (last initialization). This observation suggests that average initialization yields higher weight exploration and capability

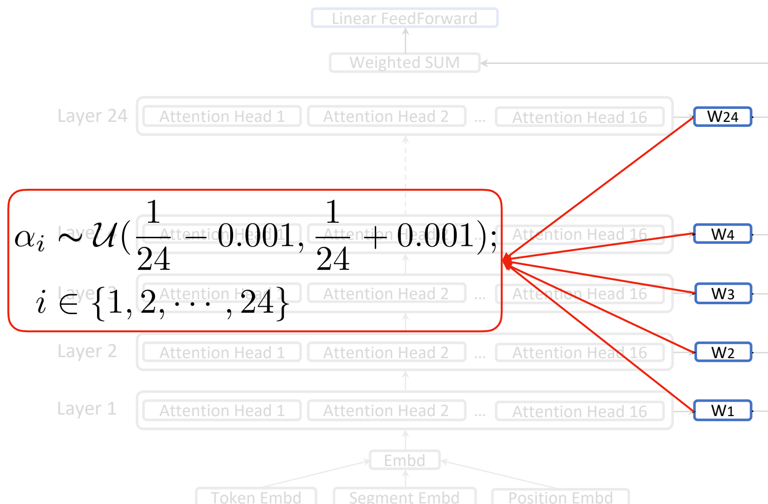


Figure 7.7: Layer weights *last initialization* visualization.

Model	CoLA (mcc)	MNLI (m/mm)	MRPC (f1/acc)	QNLI (acc)	RTE (acc)	SST-2 (acc)	STS-B (pearson /spearman)
Fixed BERT	49.3	67.3/68.4	84.1/75.5	80.3	59.9	90.7	84.5/84.6
G-BERT(avg)	46.9	74.0/74.5	84.0/75.0	85.0	64.3	91.4	86.5/86.1
G-BERT(last)	43.9	71.4/72.5	84.5/75.7	82.5	60.6	90.9	84.9/84.9
BERT	59.2	86.6/86.6	87.9/82.8	92.4	67.5	94.6	87.9/91.6
G-BERT(avg) + Fine-Tuning	64.3	86.6/86.5	91.9/88.5	92.4	74.7	93.6	90.6/90.3
G-BERT(last) + Fine-Tuning	61.4	86.3/86.2	90.9/86.8	92.4	72.2	94.0	90.8/90.5

Table 7.7: Performance of Fixed BERT and Gated BERT models on the development set of GLUE tasks.

to obtain better performance. We believe Gated BERT (last initialization) stuck in a local minimum and forced using mostly last layer for disambiguation and decision making.

Model	CoLA (mcc)	MNLI (m/mm)	MRPC (f1/acc)	QNLI (acc)	RTE (acc)	SST-2 (acc)	STS-B (S/P)	Score
Fixed BERT	43.7	67.6/68.2	82.6/74.0	79.9	60.5	91.9	77.2/74.1	69.5
G-BERT(avg)	41.8	74.0/73.7	83.2/74.7	84.5	63.9	92.1	80.5/77.7	71.3
G-BERT(last)	43.4	71.7/72.1	82.6/74.0	82.4	61.4	91.3	78.8/75.9	70.4
BERT	59.8	86.0/85.4	87.4/82.4	92.1	67.1	94.5	86.0/84.8	77.4
G-BERT(avg) + Fine-Tuning	60.6	86.0/85.2	89.0/85.0	92.3	69.0	94.3	87.4/86.3	78.1
G-BERT(last) + Fine-Tuning	57.1	85.5/85.2	89.2/84.9	92.4	68.8	94.2	87.5/86.5	77.7

Table 7.8: Performance of Fixed BERT and Gated BERT models on the test set of GLUE tasks.

7.4.4 Analysis

7.4.4.1 Layer Influence and Understanding

Here, we visualize the normalized layer gate weights for Gated BERT (average initialization) and Gated BERT (average initialization) + Fine-Tuning across different tasks (Figures 7.8 and 7.9 respectively). Among all GLUE tasks, STS-B is the only regression one and we observe different pattern and behavior for this task. For the rest of tasks (classification ones) we have a similar trends suggesting that top 8 layers are the most effective layers for disambiguation and decision making. This observation and hypothesis is verified by the trends and behavior of layer gate weights for Gated BERT + Fine-Tuning in Figure 7.9. Note that after fine-tuning, all weights are changed and behavior of the models are not comparable across GLUE tasks, but still, Figure 7.9 demonstrates that fine-tuning all parameters yields even more attention on the top layers of the model across all tasks.

This observation suggests an interesting approach for model size reduction and speed

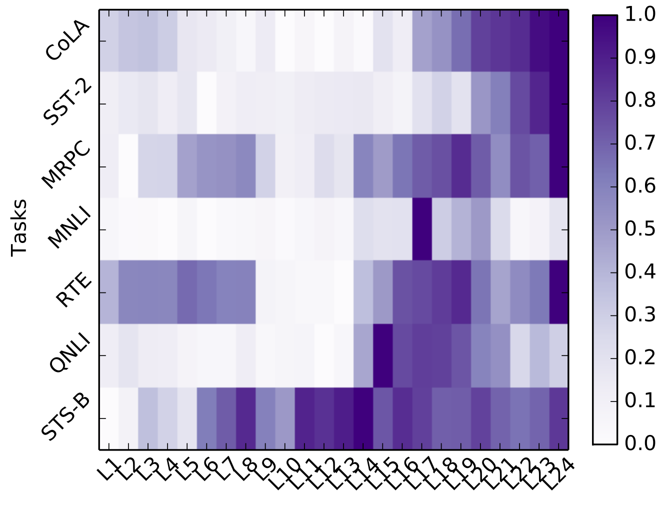


Figure 7.8: Normalized layer gate weights of Gated BERT model (average initialization) for GLUE tasks. Darker color illustrates higher weight value.

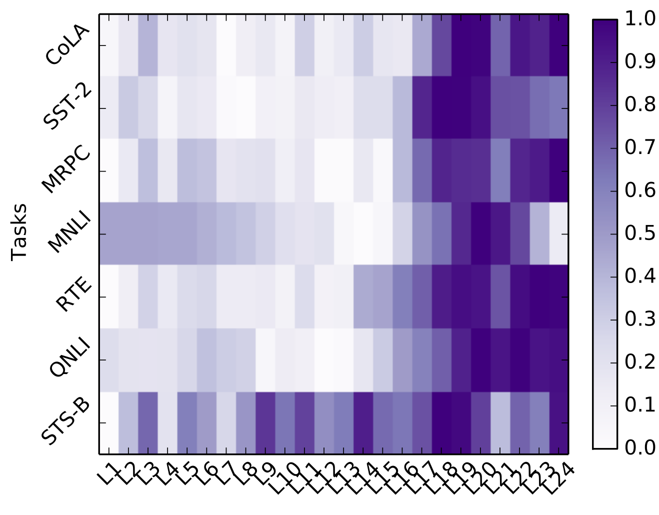


Figure 7.9: Normalized layer gate weights of Gated BERT (average initialization) + Fine-Tuning for GLUE tasks. Darker color illustrates higher weight value.

up in inference. Theoretically, we can model the behavior of the first 14 layers of BERT or Gated BERT by one or two layers using student-teacher methods [38, 9].

Model	CoLA (mcc)	MNLI (m/mm)	MRPC (f1/acc)	QNLI (acc)	RTE (acc)	SST-2 (acc)	STS-B (pearson /spearman)
0 Layer Drop	42.2	72.6/73.1	83.8/74.0	83.9	62.8	90.1	85.8/85.6
1 Layer Drop	42.2	72.2/72.9	83.5/74.0	83.8	63.5	90.1	85.9/85.6
2 Layer Drop	42.2	70.1/71.4	83.6/74.0	83.8	63.2	89.9	84.8/85.6
6 Layer Drop	38.4	70.8/71.9	83.9/74.3	83.8	65.0	87.5	85.5/85.3
12 Layer Drop	0.0	59.1/60.5	82.6/71.3	77.8	65.0	79.5	75.1/75.9
18 Layer Drop	0.0	56.3/56.5	81.2/68.4	65.6	60.6	78.7	15.4/13.2

Table 7.9: Performance of Gated BERT model on the development set of GLUE tasks when 0, 1, 2, 6, 12, and 18 top layers of the Gated BERT model are dropped.

7.4.4.2 Layer Importance

Here, we study the impact of removing top layers of the Gated BERT. Table 7.9 shows performance of Gated BERT on the development set of GLUE tasks when 0, 1, 2, 6, 12, and 18 top layers of the Gated BERT are dropped (different initialization seed is used for this experiment). According to Table 7.9, removing more than two layers from the top of the Gated BERT model yields a noticeable drop in performance on GLUE tasks. Figures 7.10, 7.11, 7.12, 7.13, and 7.14 indicates normalized layer gate weights for Gated BERT (average initialization) when 0, 1, 2, 6, and 12 top layers of the Gated BERT are dropped respectively. Aforementioned figures suggest that removing up to 6 layers does not yield a significant change in the behavior and trend of normalized layer gate weights of the Gated BERT model (while causing noticeable impacts on the performance) but removing 12 layers yields a drastic change of the behavior and trend, again, verifying the importance of the top layers of the model.

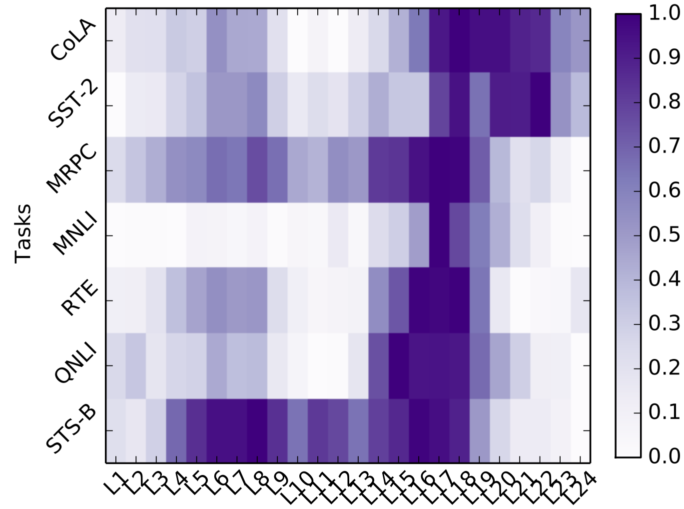


Figure 7.10: Normalized layer gate weights of Gated BERT model (average initialization) for GLUE tasks when none of the layers are dropped. Darker color illustrates higher weight value.

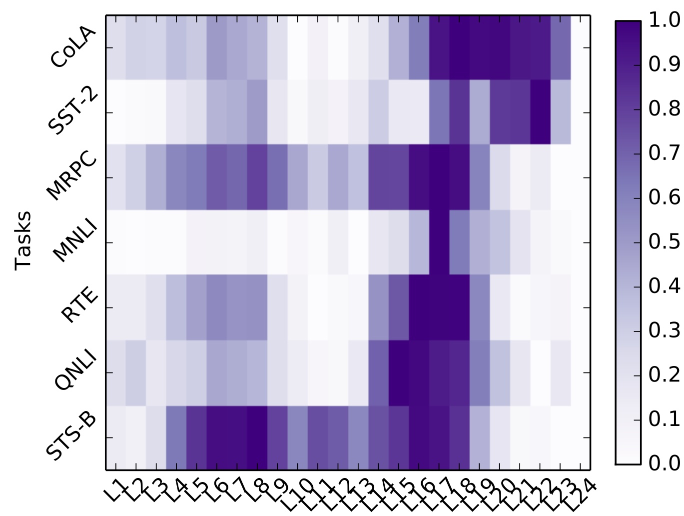


Figure 7.11: Normalized layer gate weights of Gated BERT model (average initialization) for GLUE tasks when the last top layer (layer 24) is dropped. Darker color illustrates higher weight value.

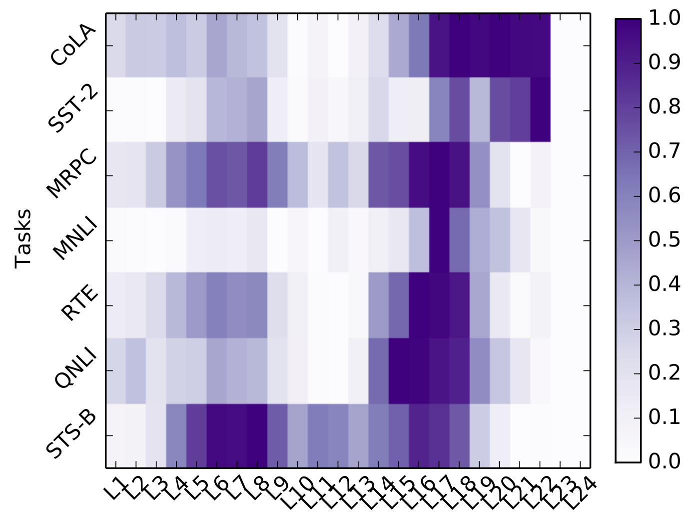


Figure 7.12: Normalized layer gate weights of Gated BERT model (average initialization) for GLUE tasks when the last two top layers (layers 23 and 24) are dropped. Darker color illustrates higher weight value.

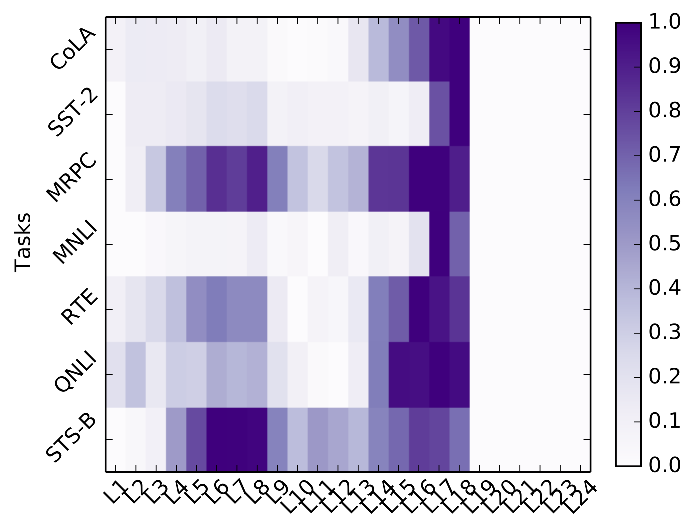


Figure 7.13: Normalized layer gate weights of Gated BERT model (average initialization) for GLUE tasks when the last six top layers (layers 19 to 24) are dropped. Darker color illustrates higher weight value.

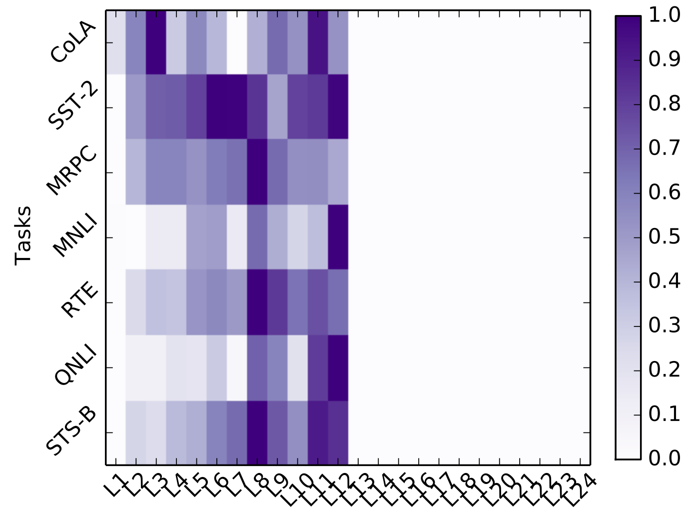


Figure 7.14: Normalized layer gate weights of Gated BERT model (average initialization) for GLUE tasks when the last 12 top layers (layers 13 to 24; second half of the model) are dropped. Darker color illustrates higher weight value.

7.5 Demo

Here, we describe the demo that is implemented for this work. This demo is capable of effectively visualizing the behavior of BERT and Gated BERT for all layers and components. In other words, we can potentially examine and study the behavior of any layer and components of these models. Figure 7.15 shows a screenshot of the demo for a sample from SST-2 corpus. The given sample and sentence in Figure 7.15 is “or doing last year taxes with your ex-wife.”. Figure 7.16 is a screenshot of the result page of the demo. Here, the system has predicted that the given sentence is Negative. The demo can help us to obtain a better understanding of the intuition behind the model prediction. Figure 7.17 illustrates inspection of the embedding layer for the given sentence. The first row from bottom shows the normalized embedding weight values. The middle row

Figure 7.15: First page of the demo for the sentiment analysing task (SST-2)

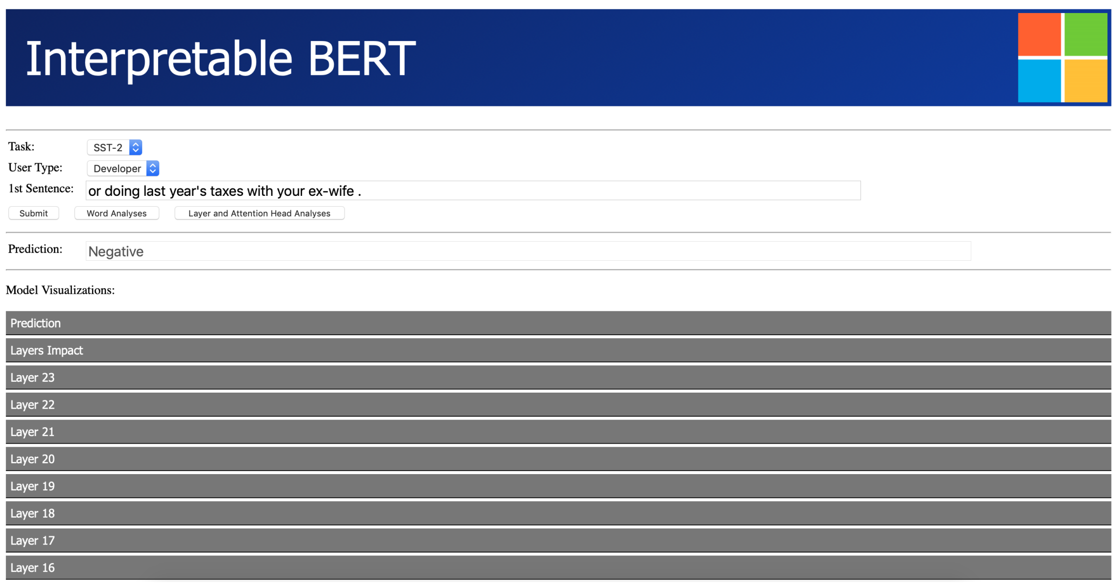


Figure 7.16: Result page of the demo for the sentiment analysing task (SST-2)

depicts the normalized gradient/saliency of the embeddings. Lastly, the third row from bottom indicates the normalized Taylor score of the embeddings. The Taylor score is defined as the multiplication of weight value and the gradient/saliency. Figure 7.17 suggests that "taxes" has the highest weight value, gradient/saliency, and Taylor score. Therefore, "taxes" could be considered as the key point and main factor for making the Negative prediction in this example.

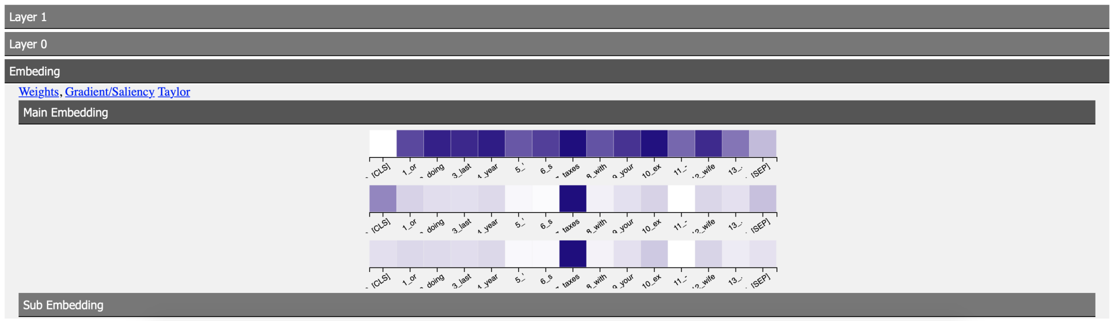


Figure 7.17: Visualization of word embeddings weights, gradient/saliency of word embeddings, and Taylor value of word embeddings for a sample from SST-2 corpus.

To test this hypothesis, we can use the “Word Analyses” component of the demo. The “Word Analysis” component provides the capability of modifying words in automatic (e.g. removal, zeroing out, unknown replacement, wordnet sampling, and n-gram sampling) and manual ways. Figures 7.18 and 7.19 demonstrate model behavior before and after removing the word “taxes”. Change of model prediction from “Negative” to “Positive” after removing “taxes” confirms the importance of “taxes” in the model prediction.

Finally, the “Layer and Attention Head Analyses” component delivers capabilities to modify the structure of the model (turning different parts off and on) to study impact of different parts on the model prediction. A screenshot of this component and page is shown in Figure 7.20.

This demo is generally model and task-independent. So, it can be adjusted to visualize other models and tasks. The implemented demo could be considered as a good toolbox for interpreting and debugging the behaviour of deep models.

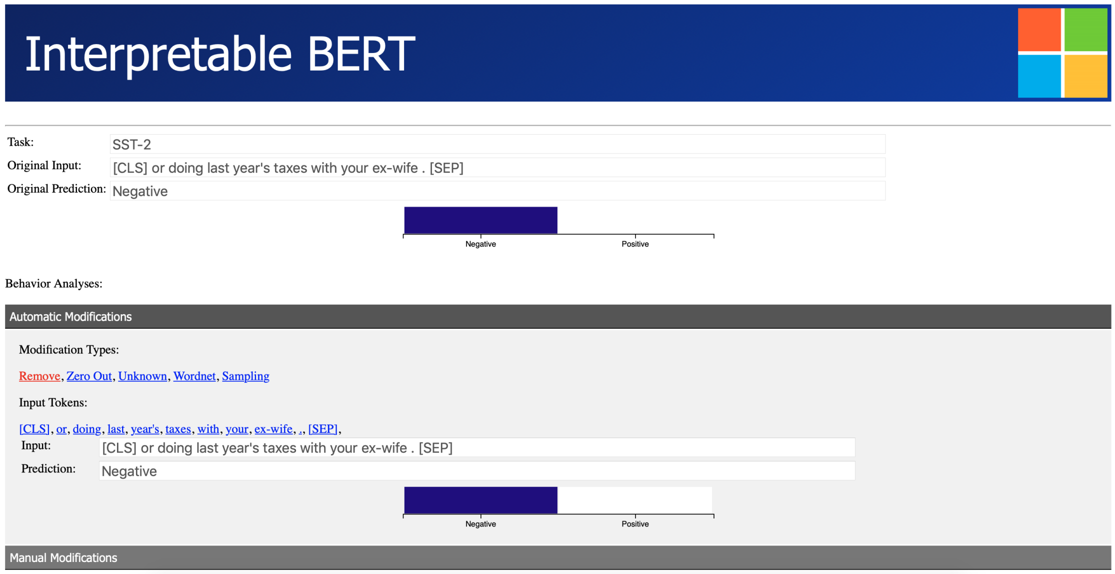


Figure 7.18: Word analysis page of the demo for a sample from SST-2 corpus.

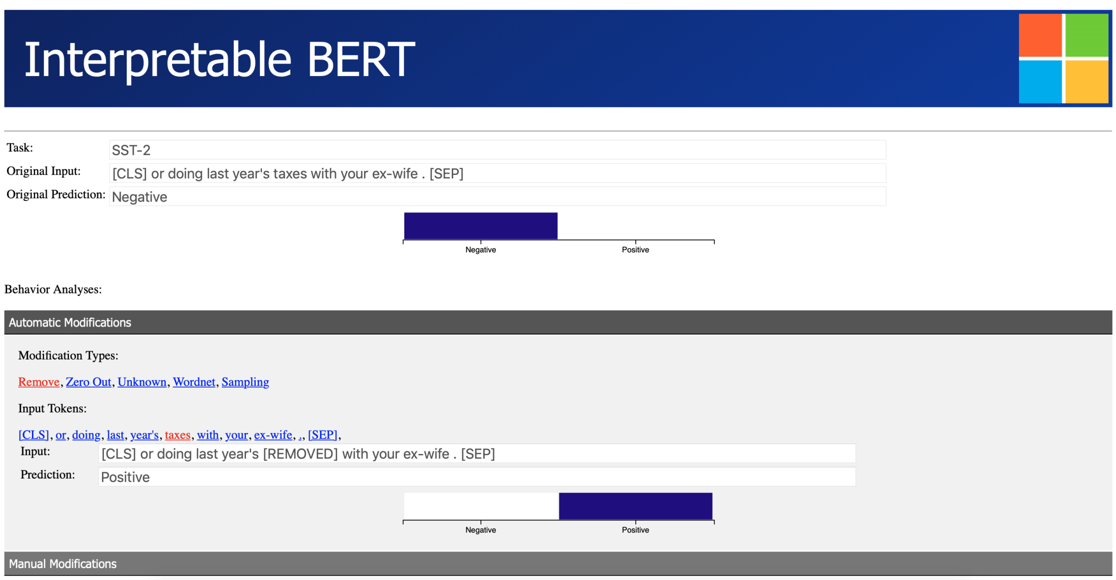


Figure 7.19: Word analysis page of the demo for a sample from SST-2 corpus when the work “taxes” has been removed and the sentiment has been changed from Negative to Positive.

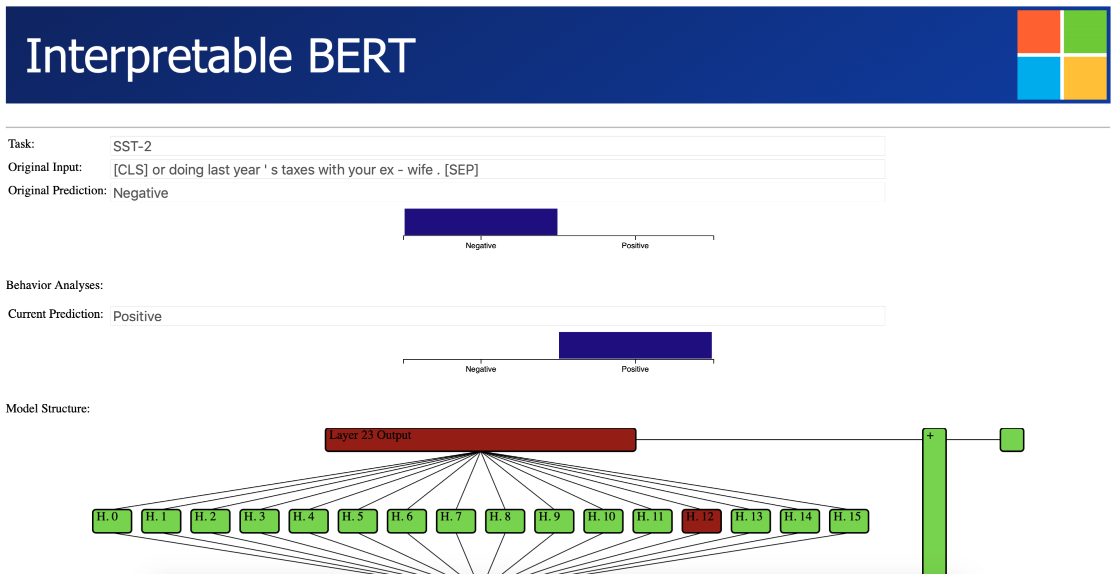


Figure 7.20: Layer and attention analysis page of the demo for a sample from SST-2 corpus.

7.6 Conclusion

We propose an interesting modification to the BERT which add interpretability features to this well-known model. While providing insights for the impact of different layers of the BERT for disambiguation and decision making of a variety of tasks, the proposed structure yield better performance with and without fine-tuning the embeddings and transformer layer weights. We evaluate BERT and Gated BERT on a wide range of NLP tasks using GLUE benchmark. Moreover, we implement an effective demo for this work which provides many practical features for debugging, understanding, and studying the behavior of BERT, Gated BERT, and potentially other deep models.

Chapter 8: Saliency Learning: Teaching the Model Where to Pay Attention

This chapter describes the work in Ghaeini et al. (2019) [23].

8.1 Introduction

It is unfortunate that our data is often plagued by meaningless or even harmful statistical biases. When we train a model on such data, it is possible that the classifier focuses on irrelevant biases to achieve high performance on the biased data. Recent studies demonstrate that deep learning models noticeably suffer from this issue [1, 94, 32]. Due to the black-box nature of deep models and the high dimensionality of their inherent representations, it is difficult to interpret and trust their behaviour and predictions. Recent work on explanation and interpretation has introduced a few approaches [83, 77, 51, 52, 53, 25, 78] for explanation. Such methods provide insights toward the model's behaviour, which is helpful for detecting biases in our models. However, they do not correct them. Here, we investigate how to incorporate explanations into the learning process to ensure that our model not only makes correct predictions but also makes them for the right reason.

Specifically, we propose to train a deep model using both ground truth labels and additional annotations suggesting the desired explanation. The learning is achieved via a novel method called *saliency learning*, which regulates the model's behaviour using saliency to ensure that the most critical factors impacting the model's prediction are aligned with the desired explanation.

Our work is closely related to Ross el al. (2017) [81], which also uses the gradient/saliency information to regularize model's behaviour. However, we differ in the following points: 1) Ross el al. (2017) [81] is limited to regularizing model with gradient of the model's input. In contrast, we extend this concept to the intermediate layers of deep models, which is demonstrated to be beneficial based on the experimental results; 2) Ross el al. (2017) [81] considers annotation at the dimension level, which is not appropriate for NLP tasks since the individual dimensions of the word embeddings are not interpretable; 3) most importantly, Ross el al. (2017) [81] learns from annotations of *irrelevant* parts of the data, whereas we focus on positive annotations identifying parts of the data that contributes positive evidence toward a specific class. In textual data, it is often unrealistic to annotate a word (even a stop word) to be completely irrelevant. On the other hand, it can be reasonably easy to identify group of words that are positively linked to a class.

We make the following contributions: 1) we propose a new method for teaching the model where to pay attention; 2) we evaluate our method on multiple tasks and datasets and demonstrate that our method achieves more reliable predictions while delivering better results than traditionally trained models; 3) we verify the sensitivity of our saliency-trained model to perturbations introduced on part of the data that contributes to the explanation.

8.2 Background: Saliency

The concept of saliency was first introduced in vision for visualizing the spatial support on an image for particular object class [83]. Considering a deep model prediction as a differentiable model f parameterized by θ with input $X \in \mathbb{R}^{n \times d}$. Such a model could be described using the Taylor series as follow:

$$f(x) = f(a) + f^{'}(a)(x - a) + \frac{f^{''}(a)}{2!}(x - a)^2 + \dots \tag{8.1}$$

By approximating that a deep model is a linear function, we could use the first order Taylor expansion.

$$f(x) \approx f^{'}(a)x + b \tag{8.2}$$

According to Equation 8.2, the first derivative of the model's prediction respect to its input ($f^{'}(a)$ or $\frac{\partial f}{\partial x}|_{x=a}$) describes the model's behaviour near the input. To make it more clear, bigger derivative/gradient indicates more impact and contribution toward the model's prediction. Consequently, the large-magnitude derivative values determine elements of input that would greatly affect $f(x)$ if changed.

8.3 Saliency-based Explanation Learning

Our goal is to teach the model where to pay attention in order to avoid focusing on meaningless statistical biases in the data. In this work, we focus on positive explanations. In other words, we expect the explanation to highlight information that contributes positively towards the label. For example, if a piece of text contains the mention of a particular event, then the explanation will highlight parts of the text indicating the event, not non-existence of some other events. This choice is because positive evidence is more natural for human to specify.

Formally, each training example is a tuple (X, y, Z), where $X = [X_1, X_2, \ldots, X_n]$ is the input text (length n), y is the ground-truth label, and $Z \in \{0, 1\}^n$ is the ground-truth explanation as a binary mask indicating whether each word contributes positive evidence toward the label y.

Recent studies have shown that the model's predictions can be explained by examining the saliency of the inputs [83, 35, 81, 52] as well as other internal elements of the model [25]. Given an example, for which the model makes a prediction, the saliency of a particular element is computed as the derivative of the model's prediction with respect to that element. Saliency provides clues as to where the model is drawing strong evidence to support its prediction. As such, if we constrain the saliency to be aligned with the desired explanation during learning, our model will be coerced to pay attention to the right evidence.

In computing saliency, we are dealing with high-dimensional data. For example, each word is represented by an embedding of d dimensions. To aggregate the contribution of all dimensions, we consider sum of the gradients of all dimensions as the overall vector/embedding contribution. For the i-th word, if $Z[i] = 1$, then its vector should have a positive gradient/contribution, otherwise the model would be penalized. To accomplish this, we incorporate a saliency regularization term to the model cost function using hinge loss. Equation 8.3 describes our cost function evaluated on a single example

(X, y, Z).

$$\mathcal{C}(\theta, X, y, Z) = \mathcal{L}(\theta, X, y) + \lambda \sum_{i=1}^{n} \max\left(0, -Z_i \sum_{j=1}^{d} \frac{\partial f_\theta(X, y)}{\partial X_{i,j}}\right) \tag{8.3}$$

where $\mathcal{L}$ is a traditional model cost function (e.g. cross-entropy), λ is a hyper parameter, f specifies the model with parameter θ, and $\frac{\partial f}{\partial X_{i,j}}$ represents the saliency of the j-th dimension of word X_i. The new term in the $\mathcal{C}$ penalizes negative gradient for the marked words in Z (contributory words).

Since $\mathcal{C}$ is differentiable respect to θ, it can be optimized using existing gradient-based optimization methods. It is important to note that while Equation 8.3 only regularizes the saliency of the input layer, the same principle can be applied to the intermediate layers of the model [25] by considering the intermediate layer as the input for the later layers.

Note that if $Z = 0$ then $\mathcal{C} = \mathcal{L}$. So, in case of lacking proper annotations for a specific sample or sequence, we can simply use 0 as its annotation. This property enables our method to be easily used in semi-supervised or active learning settings.

8.4 Tasks and Datasets

To teach the model where to pay attention, we need ground-truth explanation annotation Z, which is difficult to come by. As a proof of concept, we modify two well known real tasks (Event Extraction and Cloze-Style Question Answering) to simulate approximate annotations for explanation. Here, we first describe the main and real Event Extraction and Close-Style Question Answering tasks (before our modification). Next, we illustrate the modified tasks and provide data statistics of the modified version of ACE, ERE, CBT-NE, and CBT-CN datasets in Table 8.1.

The real Event Extraction and Close-Style Question Answering tasks are defiend as follow:

- **1) Event Extraction:** Given a set of ontologized event types (e.g. Movement, Transaction, Conflict, etc.), the goal of event extraction is to identify the mentions of different events along with their types from natural texts [12, 21, 68].

- **2) Cloze-Style Question Answering:** Documents in CBT consist of 20 contiguous sentences from the body of a popular children book and queries are formed by replacing a token from the 21^{st} sentence with a blank. Given a document, a query, and a set of candidates, the goal is to find the correct replacement for blank in the query among the given candidates. To avoid having too many negative examples in our modified datasets, we only consider sentences that contain at least one candidate. To be more clear, each sample from the CBT dataset is split to at most 20 samples – each sentence of the main sample as long as it contains one of the candidates [90, 44, 15, 19, 22].

We define the modified tasks as follows:

- **1) Event Extraction:** Given a sentence, the goal is to determine whether the sentence contains an event. Note that event extraction benchmarks contain the annotation of event triggers, which we use to build the annotation Z. In particular, the Z value of every word is annotated to be zero unless it belongs to an event trig-

Dataset	Sample Count			
	Train		Test	
	P.[a]	N.[b]	P.	N.
ACE	3.2K	15K	293	421
ERE	3.1K	4K	2.7K	1.91K
CBT-NE	359K	1.82M	8.8K	41.1K
CBT-CN	256K	2.16M	5.5K	44.4K

[a] Positive Sample Count
[b] Negative Sample Count

Table 8.1: Dataset statistics of the modified tasks and datasets.

ger. For this task, we consider two well known event extraction datasets, namely ACE 2005 and Rich ERE 2015.

- **2) Cloze-Style Question Answering:** Given a sentence and a query with a blank, the goal is to determine whether the sentence contains the correct replacement for the blank. Here, annotation of each word is zero unless it belongs to the gold replacement. For this task, we use two well known cloze-style question answering datasets: Children Book Test Named Entity (CBT-NE) and Common Noun (CBT-CN) [37].

Here, we only consider the simple binary tasks as a first attempt to examine the effectiveness of our method. However, our method is not restricted to binary tasks. In multi-class problems, each class can be treated as the positive class of the binary classification. In such a setting, each class would have its own explanation and annotation Z.

Note that for both tasks if an example is negative, its explanation annotation will be all zero. In other words, for negative examples we have $\mathcal{C} = \mathcal{L}$.

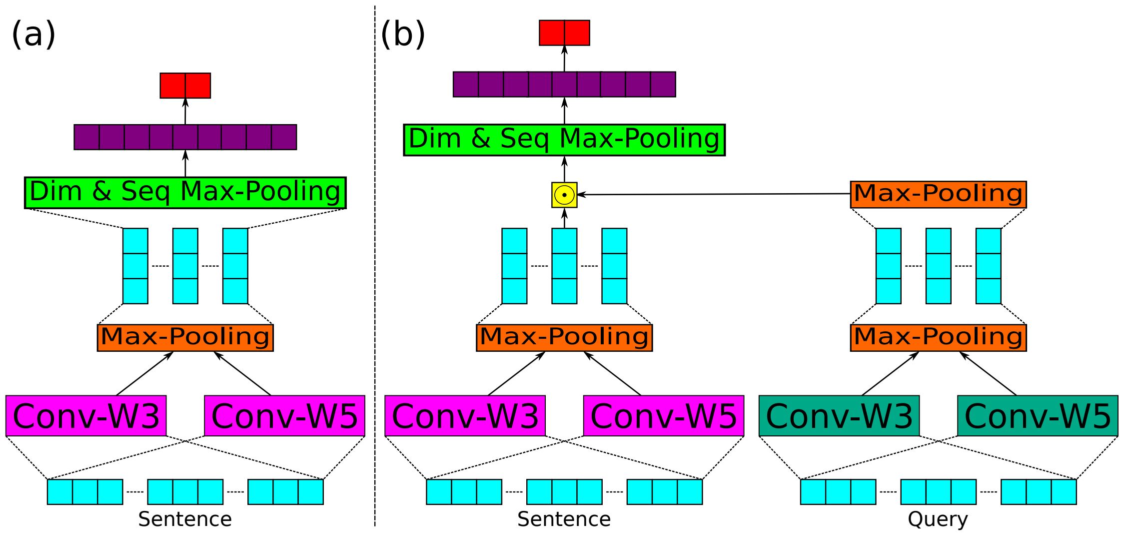


Figure 8.1: A high-level view of the models used for event extraction (a) and question answering (b).

8.5 Model

We use simple CNN based models to avoid complexity. Figure 8.1 illustrates the models used in this work. Both models have a similar structure. The main difference is that QA has two inputs (sentence and query). We first describe the event extraction model followed by the QA model.

Figure 8.1 (a) shows the event extraction model. Given a sentence $W = [w_1, \dots, w_n]$ where $w_i \in \mathbb{R}^d$, we first pass the embeddings to two CNNs with feature size of d and window size of 3 and 5. Next we apply max-pooling to both CNN outputs. It will give us the representation $I \in \mathbb{R}^{n \times d}$, which we refer to as the *intermediate representation*. Then, we apply sequence-wise and dimension-wise max-poolings to I to capture $D_{seq} \in \mathbb{R}^d$ and $D_{dim} \in \mathbb{R}^n$ respectively. D_{dim} will be referred as *decision representation*. Finally we pass the concatenation of D_{seq} and D_{dim} to a feed-forward layer for prediction.

Figure 8.1 (b) depicts the QA model. The main difference is having *query* as an extra input. To process the query, we use a similar structure to the main model. After CNNs and max-pooling we end up with $Q \in \mathbb{R}^{m \times d}$ where m is the length of query. To obtain a sequence independent vector, we apply another max-pooling to Q resulting in a query representation $q \in \mathbb{R}^{d}$. We follow a similar approach to in event extraction for the given sentence. The only difference is that we apply a dot product between the *intermediate representations* and query representation ($I_i = I_i \odot q$).

As mentioned previously, we can apply saliency regularization to different levels of the model. In this work, we apply saliency regularization on the following three levels: 1) Word embeddings (W). 2) Intermediate representation (I). 3) Decision representation (D_{dim}). Note that the aforementioned levels share the same annotation for training. For training details please refer to Section 8.7.

8.6 Experiments and Analysis

8.7 Training

All hyper-parameters are tuned based on the development set. We use pre-trained $300-D$ Glove $840B$ vectors [70] to initialize our word embedding vectors. All hidden states and feature sizes are 300 dimensions ($d = 300$). The weights are learned by minimizing the cost function on the training data via Adam optimizer. The initial learning rate is 0.0001 and $\lambda = 0.5, 0.7, 0.4,$ and 0.35 for ACE, ERE, CBT-NE, and CBT-CN respectively. To avoid overfitting, we use dropout with a rate of 0.5 for regularization, which is applied to all feedforward connections. During training, the word embeddings are updated to learn effective representations for each task and dataset. We use a fairly small batch size of 32 to provide more exploration power to the model. Finally, Equation 8.4 indicates the the cost function that is used for the training where W, I, and D_{dim} are the *word embeddings*, *Intermediate representation*, and *Decision representation* respectively.

$$
\begin{aligned}
\mathcal{C}(\theta, X, y, Z) = {} & \mathcal{L}(\theta, X, y) \\
& + \lambda \sum_{i=1}^{n} \max\left(0, -Z_i \sum_{j=1}^{d} \frac{\partial f_W(W, y)}{\partial W_{i,j}}\right) \\
& + \lambda \sum_{i=1}^{n} \max\left(0, -Z_i \sum_{j=1}^{d} \frac{\partial f_I(I, y)}{\partial I_{i,j}}\right) \\
& + \lambda \sum_{i=1}^{n} \max\left(0, -Z_i \frac{\partial f_{D_{dim}}(D_{dim}, y)}{\partial D_{dim,i}}\right)
\end{aligned}
\tag{8.4}
$$

8.7.1 Performance

Table 8.2 shows the performance of the trained models on ACE, ERE, CBT-NE, and CBT-CN datasets using the aforementioned models with and without saliency learning. The results indicate that using saliency learning yields better accuracy and F1 measure on all four datasets. It is interesting to note that saliency learning consistently helps the models to achieve noticeably higher precision without hurting the F1 measure and accuracy. This observation suggests that saliency learning is effective in providing proper guidance for more accurate predictions – Note that here we only have guidance for

Dataset	Saliency Learning (S)	Precision	Recall	F1	Accuracy
ACE	No	66.0	**77.5**	71.3	74.4
	Yes	**70.1**	76.1	**73.0**	**76.9**
ERE	No	85.0	86.6	85.8	83.1
	Yes	**85.8**	**87.3**	**86.6**	**84.0**
CBT-NE	No	55.6	**76.3**	64.3	75.5
	Yes	**57.2**	74.5	**64.7**	**76.5**
CBT-CN	No	47.4	**39.0**	42.8	77.3
	Yes	**48.3**	38.9	**43.1**	**77.7**

Table 8.2: Performance of trained models on multiple datasets using traditional method and saliency learning.

positive prediction. To verify the statistical significance of the observed performance improvement over traditionally trained models without saliency learning, we conducted the one-sided McNemar's test. The obtained p-values are 0.03, 0.03, 0.0001, and 0.04 for ACE, ERE, CBT-NE, and CBT-CN respectively, indicating that the performance gain by saliency learning is statistically significant.

8.7.2 Saliency Accuracy

In this section, we examine how well does the saliency of the trained model aligns with the annotation. To this end, we define a metric called *saliency accuracy* (s_{acc}), which measures what percentage of all positive positions of annotation Z indeed obtain a positive gradient. Formally, $s_{acc} = 100\frac{\sum_i \delta(Z_i G_i > 0)}{\sum_i Z_i}$ where G_i is the gradient of element i and δ is the indicator function.

Table 8.3 shows the saliency accuracy at different layers of the trained model with and without saliency learning. According to Table 8.3, our method achieves a much

Dataset	S.	W.[a]	I.[b]	D.[c]
ACE	No	61.60	66.05	63.27
	Yes	**99.26**	**77.92**	**65.49**
ERE	No	51.62	56.71	44.37
	Yes	**99.77**	**77.45**	**51.78**
CBT-NE	No	52.32	65.38	68.81
	Yes	**98.17**	**98.34**	**95.56**
CBT-CN	No	47.78	53.68	45.15
	Yes	**99.13**	**98.94**	**97.06**

[a]Word Level Saliency Accuracy.
[b]Intermediate Level Saliency Accuracy.
[c]Decision Level Saliency Accuracy.

Table 8.3: Saliency accuracy of different layer of our models trained on ACE, ERE, CBT-NE, CBT-CN.

higher saliency accuracy for all datasets indicating that the learning was indeed effective in aligning the model saliency with the annotation. In other words, important words will have positive contributions in the saliency-trained model, and as such, it learns to focus on the right part(s) of the data. This claim can also be verified by visualizing the saliency, which is provided in the next section.

8.7.3 Saliency Visualization

Here, we visualize the saliency of three positive samples from the ACE dataset for both the traditionally trained (Baseline Model) and the saliency-trained model (saliency-trained Model). Table 8.4 shows the top 6 salient words (words with highest saliency/gradient) of three positive samples along with their contributory words (annotation Z), the baseline model prediction (P_B), and the saliency-trained model prediction (P_S).

id	Baseline Model	Saliency-trained Model	Z	P_B	P_S
1	The judge at Hassan's extradition hearing said that he found the French handwriting report very problematic, very confusing, and with suspect conclusions.	The judge at Hassan 's extradition hearing said that he found the French handwriting report very problematic, very confusing, and with suspect conclusions.	extradition hearing said	1	1
2	Solana said the EU would help in the humanitarian crisis expected to follow an attack on Iraq.	Solana said the EU would help in the humanitarian crisis expected to follow an attack on Iraq .	attack	1	1
3	The trial will start on March 13 , the court said .	The trial will start on March 13 , the court said.	trial	1	1

Table 8.4: Top 6 salient words visualization of data samples from ACE for the baseline and the saliency-trained models.

Darker red color indicates more salient words. According to Table 8.4, both models correctly predict 1 and the saliency-trained model successfully pays attention to the expected meaningful words while the baseline model pays attention to mostly irrelevant ones. More analyses are provided in section 8.7.5.

8.7.4 Verification

Up to this point, we show that using saliency learning yields noticeably better precision, F1 measure, accuracy, and saliency accuracy. Here, we aim to verify our claim that saliency learning coerces the model to pay more attention to the critical parts. The annotation Z describes the influential words toward the positive labels. Our hypothesis is that *removing such words would cause more impact on the saliency-trained models* since by training, they should be more sensitive to these words. We measure the impact

Dataset	S.	TPR_0^a	TPR_1^b	ΔTPR^c
ACE	No	77.5	52.2	32.6
	Yes	76.1	45.0	**40.9**
ERE	No	86.6	73.2	15.4
	Yes	87.3	70.6	**19.1**
CBT-NE	No	76.3	30.2	60.4
	Yes	74.5	28.5	**61.8**
CBT-CN	No	39.0	16.6	57.4
	Yes	38.9	15.4	**60.4**

[a] True Positive Rate (before removal).
[b] TPR after removing the critical word(s).
[c] TPR change rate.

Table 8.5: True positive rate and true positive rate change of the trained models before and after removing the contributory word(s).

as the percentage change of the model's true positive rate. This measure is chosen because negative examples do not have any annotated contributory words, and hence we are particularly interested in how removing contributory words of positive examples would impact the model's true positive rate (TPR).

Table 8.5 shows the outcome of the aforementioned experiment, where the last column lists the TPR reduction rates. From the table, we see a consistently higher rate of TPR reduction for saliency-trained models compared to traditionally trained models, suggesting that the saliency-trained models are more sensitive to the perturbation of the contributory word(s) and confirming our hypothesis.

It is worth noting that we observe less substantial change to the true positive rate for the event task. This is likely due to the fact that we are using trigger words as simulated explanations. While trigger words are clearly related to events, there are often other words in the sentence relating to events but not annotated as trigger words.

8.7.5 More Saliency Visualization

In this section, we empirically analyze the traditionally trained (Baseline Model) and the saliency-trained model (saliency-trained Model) behaviour by observing the saliency of 19 positive samples from ACE and ERE datasets. Tables 8.6 and 8.7 show the top 6 salient words (words with highest saliency/gradient) of positive samples from ACE or ERE dataset along with their contributory word(s) (Z), the baseline model prediction (P_B), and the saliency-trained model prediction (P_S). Darker red color indicates more salient words. Our observations could be divided into six categories as follow:

- Samples 1-4 (and also samples in Table 8.4): Both models correctly predict 1 for these samples. The saliency-trained model successfully pays attention to the expected meaningful words while the baseline model pays attention to mostly irrelevant ones.

- Samples 5-8: Both models correctly predict 1 and pays attention to the contributory words. Yet, we observe lower saliency for important words and higher saliency for irrelevant ones.

- Samples 9-10: Here, the baseline model fails to pay attention to the contributory words and predicts 0 while the saliency-trained model one successfully pays attention to them and predicts 1.

- Samples 11-14: Although the models have high saliency for the contributory words, still they could not correctly disambiguate these samples. This observation suggests that having high saliency for important words does not guarantee positive prediction. High saliency for these words indicate their positive contribution toward the positive prediction but still, the model might consider higher probability for negative prediction.

- Samples 15-17: Here, only the baseline model could correctly predict 1. However, the baseline model does not pay attention to the contributory words. In other words, the explanation does not support the prediction (unreliable).

- Samples 18-19: Not always the saliency-trained model could pay proper attention to the contributory words. In these examples, the baseline model has high saliency for contributory words. It is worth noting that when the saliency-trained model does not have high saliency for contributory words, it does not predict 1. Such observation could suggest that the saliency-trained model predictions are more reliable. The aforementioned claim is also verified by consistently obtaining noticeably higher precision for all datasets and tasks (Section 8.7.1 and Table 8.2).

8.8 Conclusion

In this work, we proposed *saliency learning*, a novel approach for teaching a model where to pay attention. We demonstrated the effectiveness of our method on multiple tasks and datasets using simulated explanations. The results show that saliency learning enables us to obtain better precision, F1 measure and accuracy on these tasks and datasets. Further, it produces models whose saliency is more properly aligned with the desired explanation. In other words, *saliency learning* gives us more reliable predictions

id	Baseline Model	Saliency-trained Model	Z	P_B	P_S
1	India 's has been reeling under a heatwave since mid-May which has killed 1,403 people.	India 's has been reeling under a heatwave since mid-May which has killed 1,403 people .	killed	1	1
2	Retired General Electric Co. Chairman Jack Welch is seeking work-related documents of his estranged wife in his high-stakes divorce case .	Retired General Electric Co. Chairman Jack Welch is seeking work-related documents of his estranged wife in his high-stakes divorce case .	Retired divorce	1	1
3	The following year, he was acquitted in the Guatemala case, but the U.S. continued to push for his prosecution.	The following year, he was acquitted in the Guatemala case , but the U.S. continued to push for his prosecution .	acquitted case	1	1
4	In 2011, a Spanish National Court judge issued arrest warrants for 20 men , including Montano,suspected of participating in the slaying of the priests.	In 2011, a Spanish National Court judge issued arrest warrants for 20 men, including Montano,suspected of participating in the slaying of the priests.	issued slaying arrest	1	1
5	Slobodan Milosevic's wife will go on trial next week on charges of mismanaging state property during the former president's rule, a court said Thursday.	Slobodan Milosevic's wife will go on trial next week on charges of mismanaging state property during the former president 's rule, a court said Thursday .	trial charges former	1	1
6	Iraqis mostly fought back with small arms, pistols, machine guns and rocket-propelled grenades .	Iraqis mostly fought back with small arms, pistols, machine guns and rocket-propelled grenades.	fought	1	1
7	He will then stay on for a regional summit before heading to Saint Petersburg for celebrations marking the 300th anniversary of the city's founding .	He will then stay on for a regional summit before heading to Saint Petersburg for celebrations marking the 300th anniversary of the city's founding.	heading summit	1	1

Table 8.6: Top 6 salient words visualization of samples from ACE and ERE for the baseline and the saliency-trained models.

id	Baseline Model	Saliency-trained Model	Z	P_B	P_S
8	But the Saint Petersburg summit ended without any formal declaration on Iraq .	But the Saint Petersburg summit ended without any formal declaration on Iraq .	summit	1	1
9	From greatest moment of his life to divorce in 3 years or less.	From greatest moment of his life to divorce in 3 years or less.	divorce	0	1
10	The student, who was 18 at the time of the alleged sexual relationship, testified under a pseudonym .	The student, who was 18 at the time of the alleged sexual relationship , testified under a pseudonym.	testified	0	1
11	U.S. aircraft bombed Iraqi tanks holding bridges close to the city .	U.S. aircraft bombed Iraqi tanks holding bridges close to the city.	bombed	0	0
12	However , no blasphemy convict has ever been executed in the country .	However, no blasphemy convict has ever been executed in the country .	executed	0	0
13	Gul 's resignation had been long expected .	Gul 's resignation had been long expected .	resignation	0	0
14	aside from purchasing alcohol, what rights don't 18 year olds have?	aside from purchasing alcohol , what rights don't 18 year olds have?	purchasing	0	0
15	He also ordered him to have no contact with Shannon Molden.	He also ordered him to have no contact with Shannon Molden .	ordered contact	1	0
16	This means your account is once again active and operational, Riaño wrote Colombia Reports.	This means your account is once again active and operational , Riaño wrote Colombia Reports .	wrote	1	0
17	I am a Christian as is my ex husband yet we are divorced.	I am a Christian as is my ex husband yet we are divorced .	divorced ex	1	0
18	Taylor acknowledged in his testimony that he ran up toward the pulpit with a large group and followed the men outside.	Taylor acknowledged in his testimony that he ran up toward the pulpit with a large group and followed the men outside.	testimony followed ran	1	0
19	The note admonished Jasper Molden , and his then-fiancée, Shannon Molden .	The note admonished Jasper Molden , and his then-fiancée , Shannon Molden.	note	0	0

Table 8.7: Top 6 salient words visualization of samples from ACE and ERE for the baseline and the saliency-trained models.

while delivering better performance than traditionally trained models. Finally, our verification experiments illustrate that the saliency-trained models show higher sensitivity to the removal of contributory words in a positive example. For future work, we will extend our study to examine saliency learning on NLP tasks in an active learning setting where real explanations are requested and provided by a human.

Chapter 9: Summary

This dissertation describes methods and solutions for improving and understanding deep models with a special focus on Natural Language Comprehension (NLC) tasks. First, we attempt to improve a model's language comprehension/understanding by enriching the structure of the model to enhance its capability in learning the latent rules of the language. More specifically, we focus on pairwise input source tasks like Natural Language Inference (NLI) and Cloze-Style Question Answering and propose the idea of conditional/dependent encoding and reading. The intuition behind the proposed methodology is to efficiently model the relationships between input sources (e.g. "premise and hypothesis" or "document and query"). We demonstrate the positive impact of conditional/dependent encoding by obtaining better empirical performances, However, due to the black-box nature of deep learning, we can not conclude that the proposed methods yield better language understanding. This motivates us to study methods for "peaking inside" the black-box deep models to provide explanation and understanding of the models' behaviour. The proposed method (a.k.a. saliency) takes a step toward explaining deep models based on gradient of the model output with respect to different components like the input layer and intermediate layers. We demonstrate the effectiveness of the proposed explanation method on a complex task (NLI). Saliency reveals interesting insights and identifies critical information contributing to the model decisions. Besides proposing a model-agnostic interpretation method (saliency), we study model-dependent/model-embedded interpretation solutions and propose two interpretable designs and structures; *Attentional Multi-Reading Sarcasm Detection* and *Gated BERT*. Our evaluations successfully demonstrate the superiority of the proposed interpretable models by obtaining better performance while delivering explanation and interpretation. Moreover, we develop and release an interesting demo/toolkit which could be easily adjusted for other tasks and structures. The developed demo/toolkit provides many helpful insights for understanding and debugging a model. Finally, we introduce saliency learning; a novel approach for teaching a model where to pay attention to make the right prediction for the right reason. Our experimental results on multiple tasks and datasets demonstrate the effectiveness of the proposed method, which produce more reliable predictions while delivering better results compared to traditionally trained models.

Interpretation and explanation is a new line of research and we are yet far behind the perfection. The future works of this study could be categorized as below:

1. Investigating faithfull explanation strategies which unlike saliency do not require the linear approximation.

2. Incorporating explanation and interpretation into models' design and structure in order to obtain better and more reliable results.

3. Examining saliency learning in active learning and semi-supervised setting where real explanations are requested and provided by a human.

Bibliography

[1] Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the behavior of visual question answering models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1955–1960, 2016.

[2] Silvio Amir, Byron C. Wallace, Hao Lyu, Paula Carvalho, and Mário J. Silva. Modelling context with user embeddings for sarcasm detection in social media. *In Proceeding of CoNLL, 2016.*, pages 167–177, 2016.

[3] Leila Arras, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. Explaining recurrent neural network predictions in sentiment analysis. In *Proceedings of the 8th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@EMNLP 2017, Copenhagen, Denmark, September 8, 2017*, pages 159–168, 2017.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[5] David Bamman and Noah A. Smith. Contextualized sarcasm detection on twitter. *In Proceeding of ICWSM, 2015.*, pages 574–577, 2015.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[7] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642, 2015.

[8] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. A fast unified model for parsing and sentence understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[9] Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 535–541, 2006.

[10] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[11] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced LSTM for natural language inference. 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 1657–1668, 2017.

[12] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 167–176, 2015.

[13] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[14] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. *ICML*, pages 160–167, 2008.

[15] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. 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 593–602, 2017.

[16] Yiming Cui, Ting Liu, Zhipeng Chen, Shijin Wang, and Guoping Hu. Consensus attention-based neural networks for chinese reading comprehension. In *COL-*

ING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan, pages 1777–1786, 2016.

[17] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Semi-supervised recognition of sarcasm in twitter and amazon. *In Proceeding of CoNLL, 2010.*, pages 107–116, 2010.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[19] Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. 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 1832–1846, 2017.

[20] Bhuwan Dhingra, Zhong Zhou, Dylan Fitzpatrick, Michael Muehl, and William W. Cohen. Tweet2vec: Character-based distributed representations for social media. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.

[21] Reza Ghaeini, Xiaoli Z. Fern, Liang Huang, and Prasad Tadepalli. Event nugget detection with forward-backward recurrent neural networks. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.

[22] Reza Ghaeini, Xiaoli Z. Fern, Hamed Shahbazi, and Prasad Tadepalli. Dependent gated reading for cloze-style question answering. In *Proceedings of the 27th International Conference on Computational Linguistics, COLING 2018, Santa Fe, New Mexico, USA, August 20-26, 2018*, pages 3330–3345, 2018.

[23] Reza Ghaeini, Xiaoli Z. Fern, Hamed Shahbazi, and Prasad Tadepalli. Saliency learning: Teaching the model where to pay attention. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational*

Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 4016–4025, 2019.

[24] Reza Ghaeini, Xiaoli Z. Fern, and Prasad Tadepalli. Attentional multi-reading sarcasm detection. *CoRR*, abs/1809.03051, 2018.

[25] Reza Ghaeini, Xiaoli Z. Fern, and Prasad Tadepalli. Interpreting recurrent and attention-based neural models: a case study on natural language inference. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4952–4957, 2018.

[26] Reza Ghaeini, Sadid A. Hasan, Vivek V. Datla, Joey Liu, Kathy Lee, Ashequl Qadir, Yuan Ling, Aaditya Prakash, Xiaoli Z. Fern, and Oladimeji Farri. Dr-bilstm: Dependent reading bidirectional LSTM for natural language inference. *NAACL HLT 2018, The 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018.

[27] Aniruddha Ghosh and Tony Veale. Fracking sarcasm using neural network. *Proceedings of the 7th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@NAACL-HLT, 2016.*, pages 161–169, 2016.

[28] Debanjan Ghosh, Alexander Richard Fabbri, and Smaranda Muresan. The role of conversation context for sarcasm detection in online interactions. *In Proceeding of SIGdial Meeting on Discourse and Dialogue, 2017.*, pages 186–196, 2017.

[29] Debanjan Ghosh, Weiwei Guo, and Smaranda Muresan. Sarcastic or not: Word embeddings to predict the literal or sarcastic meaning of words. *In Proceeding of EMNLP, 2015.*, pages 1003–1012, 2015.

[30] Yichen Gong, Heng Luo, and Jian Zhang. Natural language inference over interaction space. *CoRR*, abs/1709.04348, 2017.

[31] Roberto I. González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in twitter: A closer look. *In Proceeding of ACL, 2011.*, pages 581–586, 2011.

[32] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman, and Noah A. Smith. Annotation artifacts in natural language inference data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 107–112, 2018.

[33] Henk Haverkate. A speech act analysis of irony. *Journal of Pragmatics.*, 14(1):77–109, 1990.

[34] Devamanyu Hazarika, Soujanya Poria, Sruthi Gorantla, Erik Cambria, Roger Zimmermann, and Rada Mihalcea. CASCADE: contextual sarcasm detection in online discussion forums. *In Proceeding of COLING, 2018.*, pages 1837–1848, 2018.

[35] Yotam Hechtlinger. Interpretation of prediction models using the input gradient. *CoRR*, abs/1611.07634, 2016.

[36] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1693–1701, 2015.

[37] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *CoRR*, abs/1511.02301, 2015.

[38] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[40] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3651–3657, 2019.

[41] Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Automatic sarcasm detection: A survey. *ACM Comput. Surv.*, 50(5):73:1–73:22, 2017.

[42] Aditya Joshi, Vinita Sharma, and Pushpak Bhattacharyya. Harnessing context incongruity for sarcasm detection. *In Proceeding of ACL, 2015.*, pages 757–762, 2015.

[43] Aditya Joshi, Vaibhav Tripathi, Kevin Patel, Pushpak Bhattacharyya, and Mark James Carman. Are word embedding-based features useful for sarcasm detection? *In Proceeding of EMNLP, 2016.*, pages 1006–1011, 2016.

[44] Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. Text understanding with the attention sum reader network. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[45] Divyansh Kaushik, Eduard H. Hovy, and Zachary C. Lipton. Learning the difference that makes a difference with counterfactually-augmented data. *CoRR*, abs/1909.12434, 2019.

[46] Anupam Khattri, Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Your sentiment precedes you: Using an author's historical tweets to predict sarcasm. *Proceedings of the 6th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@EMNLP, 2015.*, pages 25–30, 2015.

[47] Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. A large self-annotated corpus for sarcasm. *In Proceeding of LREC, 2018.*, 2018.

[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[49] Roger J Kreuz and Gina M Caucci. Lexical influences on the perception of sarcasm. *Proceedings of the Workshop on computational approaches to Figurative Language.*, pages 1–4, 2007.

[50] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1378–1387, 2016.

[51] Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. Rationalizing neural predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural*

Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016, pages 107–117, 2016.

[52] Jiwei Li, Xinlei Chen, Eduard H. Hovy, and Dan Jurafsky. Visualizing and understanding neural models in NLP. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 681–691, 2016.

[53] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *CoRR*, abs/1612.08220, 2017.

[54] Peng Li, Wei Li, Zhengyan He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. Dataset and neural recurrent sequence labeling model for open-domain factoid question answering. *arXiv preprint arXiv:1607.06275*, 2016.

[55] Christine Liebrecht, Florian Kunneman, and Antal van den Bosch. The perfect solution for detecting sarcasm in tweets #not. *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, WASSA@NAACL-HLT, 2013.*, pages 29–37, 2013.

[56] Wang Ling, Chris Dyer, Alan W. Black, Isabel Trancoso, Ramon Fermandez, Silvio Amir, Luís Marujo, and Tiago Luís. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1520–1530, 2015.

[57] Pengfei Liu, Xipeng Qiu, Jifan Chen, and Xuanjing Huang. Deep fusion lstms for text semantic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*, 2016.

[58] Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR*, abs/1605.09090, 2016.

[59] Bill MacCartney and Christopher D. Manning. Modeling semantic containment and exclusion in natural language inference. In *COLING 2008, 22nd International Conference on Computational Linguistics, Proceedings of the Conference, 18-22 August 2008, Manchester, UK*, pages 521–528, 2008.

[60] Diana Maynard and Mark A. Greenwood. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. *In Proceeding of LREC, 2014.*, pages 4238–4243, 2014.

[61] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6297–6308, 2017.

[62] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14014–14024, 2019.

[63] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[64] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA*, pages 746–751, 2013.

[65] Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.

[66] Tsendsuren Munkhdalai and Hong Yu. Reasoning with memory augmented neural networks for language comprehension. *ICLR*, abs/1610.06454, 2017.

[67] Takeshi Onishi, Hai Wang, Mohit Bansal, Kevin Gimpel, and David A. McAllester. Who did what: A large-scale person-centered cloze dataset. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language*

Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016, pages 2230–2235, 2016.

[68] John Walker Orr, Prasad Tadepalli, and Xiaoli Z. Fern. Event detection with neural networks: A rigorous empirical evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 999–1004, 2018.

[69] Ankur P. Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2249–2255, 2016.

[70] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[71] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

[72] Bryan A. Plummer, Liwei Wang, Chris M. Cervantes, Juan C. Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2641–2649, 2015.

[73] Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. A deeper look into sarcastic tweets using deep convolutional neural networks. *In Proceeding of COLING, 2016.*, pages 1601–1612, 2016.

[74] Ashwin Rajadesingan, Reza Zafarani, and Huan Liu. Sarcasm detection on twitter: A behavioral modeling approach. *In Proceeding of ACM International Conference on Web Search and Data Mining, WSDM, 2015.*, pages 97–106, 2015.

[75] Marek Rei and Anders Søgaard. Zero-shot sequence labeling: Transferring knowledge from sentences to tokens. In *Proceedings of the 2018 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers), pages 293–302, 2018.

[76] Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B. Viégas, Andy Coenen, Adam Pearce, and Been Kim. Visualizing and measuring the geometry of BERT. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8592–8600, 2019.

[77] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. ”why should I trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[78] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1527–1535, 2018.

[79] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as contrast between a positive sentiment and negative situation. *In Proceeding of EMNLP, 2013, A meeting of SIGDAT, a Special Interest Group of the ACL.*, pages 704–714, 2013.

[80] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomás Kociský, and Phil Blunsom. Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664, 2015.

[81] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 2662–2670, 2017.

[82] Lei Sha, Baobao Chang, Zhifang Sui, and Sujian Li. Reading and thinking: Re-read LSTM unit for textual entailment recognition. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Con-*

ference: Technical Papers, December 11-16, 2016, Osaka, Japan, pages 2870–2879, 2016.

[83] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[84] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642, 2013.

[85] Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *CoRR*, abs/1606.02245, 2016.

[86] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[87] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. A compare-propagate architecture with alignment factorization for natural language inference. *CoRR*, abs/1801.00102, 2018.

[88] Wilson L Taylor. “cloze procedure”: a new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433, 1953.

[89] Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4593–4601, 2019.

[90] Adam Trischler, Zheng Ye, Xingdi Yuan, Philip Bachman, Alessandro Sordoni, and Kaheer Suleman. Natural language comprehension with the epireader. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 128–137, 2016.

[91] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you

need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[92] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. *CoRR*, abs/1511.06361, 2015.

[93] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5797–5808, 2019.

[94] Soumya Wadhwa, Varsha Embar, Matthias Grabmair, and Eric Nyberg. Towards inference-oriented reading comprehension: Parallelqa. *CoRR*, abs/1805.03830, 2018.

[95] Byron C. Wallace, Do Kook Choe, and Eugene Charniak. Sparse, contextually informed models for irony detection: Exploiting user communities, entities and sentiment. *In Proceeding of ACL, 2015.*, pages 1035–1044, 2015.

[96] Byron C. Wallace, Do Kook Choe, Laura Kertz, and Eugene Charniak. Humans require context to infer ironic intent (so computers probably do, too). *In Proceeding of ACL, 2014.*, pages 512–516, 2014.

[97] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[98] Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego California, USA, June 12-17, 2016*, pages 1442–1451, 2016.

[99] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4144–4150, 2017.

[100] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. Neural network acceptability judgments. *TACL*, 7:625–641, 2019.

[101] Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broadcoverage challenge corpus for sentence understanding through inference. *CoRR*, abs/1704.05426, 2017.

[102] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W. Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *ICLR*, abs/1611.01724, 2017.

[103] Hong Yu and Tsendsuren Munkhdalai. Neural semantic encoders. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 397–407, 2017.

[104] Hong Yu and Tsendsuren Munkhdalai. Neural tree indexers for text understanding. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 1: Long Papers*, pages 11–21, 2017.

[105] Meishan Zhang, Yue Zhang, and Guohong Fu. Tweet sarcasm detection using deep neural network. *In Proceeding of COLING, 2016.*, pages 2449–2460, 2016.

[106] Kai Zhao, Liang Huang, and Mingbo Ma. Textual entailment with structured attentions and composition. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pages 2248–2258, 2016.

APPENDICES

Appendix A: Source Code for Gated BERT (G-BERT)

The source code for major parts of the Gated BERT (G-BERT) and the demo are described here for reference. In the hope that future researchers will be able to build from the work, the full source code has also been published and made completely open source at: https://github.com/rezaghaeini/Gated_BERT

A.1 pyencoder/modeling_bert.py

```
class BertModel(BertPreTrainedModel):
    def __init__(self, config):
        super(BertModel, self).__init__(config)
        self.config = config
        self.embeddings = BertEmbeddings(config)
        self.encoder = BertEncoder(config)
        self.pooler = BertPooler(config)
        self.head_mask_size = (config.num_hidden_layers, config.num_attention_heads)
        self.layer_mask_size = (config.num_hidden_layers,)
        self.head_init_limit = math.sqrt(6. / (self.head_mask_size[0] + self.head_mask_size[1]))
        self.layer_init_limit = math.sqrt(6. / (self.layer_mask_size[0] + 1))
        self.gate_dropout = nn.Dropout(config.hidden_dropout_prob)
```

```
        self.eps = 1e-6
        self.temp = 0.33
        self.gamma = -0.1
        self.zeta = 1.1
        self.layer_fix_mask = None
        self.gamma_zeta_ratio = math.log(-self.gamma / self.zeta)
        if config.freeze_encoder:
            if not config.keep_part_grads:
                for p in self.embeddings.word_embeddings.parameters()
:
                    p.requires_grad = False
                for p in self.embeddings.token_type_embeddings.
parameters():
                    p.requires_grad = False
                for p in self.embeddings.position_embeddings.
parameters():
                    p.requires_grad = False
            for layer in self.encoder.layer:
                for p in layer.parameters():
                    p.requires_grad = False
        if config.smart_head:
            if type(config.num_labels) == list:
                self.head_weights = nn.ParameterList()
                for _ in range(len(config.num_labels)):
                    self.head_weights.append(self._init_gate_weights(
config.shm_reg_type, self.head_mask_size, self.head_init_limit))
            else:
```

```
                self.head_weights = self._init_gate_weights(config.shm_reg_type, self.head_mask_size, self.head_init_limit)
        if config.smart_pooling:
            if type(config.num_labels) == list:
                self.layer_weights = nn.ParameterList()
                for _ in range(len(config.num_labels)):
                    self.layer_weights.append(self._init_gate_weights(config.lp_reg_type, self.layer_mask_size, self.layer_init_limit, config.lp_init_method, True))
            else:
                self.layer_weights = self._init_gate_weights(config.lp_reg_type, self.layer_mask_size, self.layer_init_limit, config.lp_init_method, True)
        self.keep_part_grads = config.keep_part_grads
        self.layer_mask_weight = None
        self.apply(self.init_weights)

    def gate_mask(self, w, strategy, mask_size, device, hard=None):
        def clipped_concrete(x):
            return torch.min(torch.max(x, torch.zeros_like(x)), torch.ones_like(x))
        if strategy == 'L0':
            if self.training and (not self.config.noiseless_L0):
                u = Variable(torch.zeros(mask_size, device=device).uniform_(self.eps, 1-self.eps), requires_grad=False).view(-1)
                concrete = torch.sigmoid((torch.log(u) - torch.log(1 - u) + w.view(-1)) / self.temp)
```

```
            else:
                concrete = torch.sigmoid(w.view(-1) / self.temp)
            stretched_concrete = concrete * (self.zeta - self.gamma) + self.gamma
            clipped_concrete = clipped_concrete(stretched_concrete)
            if self.config.hard_L0 or (hard is not None and hard):
                hard_concrete = Variable(torch.gt(clipped_concrete, 0.5).to(dtype=torch.float32, device=device), requires_grad=False)
                clipped_concrete = clipped_concrete + Variable((hard_concrete-clipped_concrete), requires_grad=False)
            return clipped_concrete.view(mask_size)
        else:
            if hard is not None and hard:
                hard_w = Variable(torch.gt(w, 0.5).to(dtype=torch.float32, device=device), requires_grad=False)
                return  w + Variable((hard_w - w), requires_grad=False)
            else:
                return w

    def _init_gate_weights(self, strategy, mask_size, init_limit, init_method='norm', is_layers=False):
        if strategy == 'L0':
            return nn.Parameter(
                torch.zeros(mask_size).uniform_(-2*(self.config.L0_start_mask)*init_limit, 2*(1-self.config.L0_start_mask)*init_limit))
```

```
        else:
            if is_layers:
                if init_method == 'norm':
                    center = 1.0 / self.config.num_hidden_layers
                    t = torch.zeros(mask_size).uniform_(center-0.001, center+0.001)
                    return nn.Parameter(t)
                else:
                    init_part = init_method.split('_')
                    one_idx = -1 if len(init_part) < 2 else int(init_part[-1])
                    t = torch.zeros(mask_size).uniform_(-0.001, 0.001)
                    t[one_idx] = 1.0
                    return nn.Parameter(t)
            else:
                return nn.Parameter(
                    torch.ones(mask_size))

    def _resize_token_embeddings(self, new_num_tokens):
        old_embeddings = self.embeddings.word_embeddings
        new_embeddings = self._get_resized_embeddings(old_embeddings, new_num_tokens)
        self.embeddings.word_embeddings = new_embeddings
        return self.embeddings.word_embeddings

    def _prune_heads(self, heads_to_prune):
```

```
        """ Prunes heads of the model.
            heads_to_prune: dict of {layer_num: list of heads to prune in this layer}
            See base class PreTrainedModel
        """
        for layer, heads in heads_to_prune.items():
            self.encoder.layer[layer].attention.prune_heads(heads)

    def _prune_layers(self, layer_mask):
        self.layer_fix_mask = layer_mask

    def forward(self, input_ids, token_type_ids=None, attention_mask=None, position_ids=None, head_mask=None, task_id=None, layer_mask=None):
        if self.config.smart_head:
            if task_id is None:
                head_mask = self.gate_mask(
                                self.head_weights,
                                self.config.shm_reg_type,
                                self.head_mask_size,
                                input_ids.device) if head_mask is None else (self.gate_mask(self.head_weights, self.config.shm_reg_type, self.head_mask_size, input_ids.device) * head_mask)
            else:
                head_mask = self.gate_mask(
                                self.head_weights[task_id],
                                self.config.shm_reg_type,
```

```
                                self.head_mask_size,
                                input_ids.device) if head_mask is None else (self.gate_mask(self.head_weights[task_id], self.config.shm_reg_type, self.head_mask_size, input_ids.device) * head_mask)
            if self.config.gate_dropout == 1:
                head_mask = self.gate_dropout(head_mask)
        if attention_mask is None:
            attention_mask = torch.ones_like(input_ids)
        if token_type_ids is None:
            token_type_ids = torch.zeros_like(input_ids)

        # We create a 3D attention mask from a 2D tensor mask.
        # Sizes are [batch_size, 1, 1, to_seq_length]
        # So we can broadcast to [batch_size, num_heads, from_seq_length, to_seq_length]
        # this attention mask is more simple than the triangular masking of causal attention
        # used in OpenAI GPT, we just need to prepare the broadcast dimension here.
        extended_attention_mask = attention_mask.unsqueeze(1).unsqueeze(2)

        # Since attention_mask is 1.0 for positions we want to attend and 0.0 for
        # masked positions, this operation will create a tensor which is 0.0 for
```

```
        # positions we want to attend and -10000.0 for masked positions.
        # Since we are adding it to the raw scores before the softmax, this is
        # effectively the same as removing these entirely.
        extended_attention_mask = extended_attention_mask.to(dtype=next(self.parameters()).dtype) # fp16 compatibility
        extended_attention_mask = (1.0 - extended_attention_mask) * -10000.0

        # Prepare head mask if needed
        # 1.0 in head_mask indicate we keep the head
        # attention_probs has shape bsz x n_heads x N x N
        # input head_mask has shape [num_heads] or [num_hidden_layers x num_heads]
        # and head_mask is converted to shape [num_hidden_layers x batch x num_heads x seq_length x seq_length]
        if head_mask is not None:
            if head_mask.dim() == 1:
                head_mask = head_mask.unsqueeze(0).unsqueeze(0).unsqueeze(-1).unsqueeze(-1)
                head_mask = head_mask.expand(self.config.num_hidden_layers, -1, -1, -1, -1)
            elif head_mask.dim() == 2:
                head_mask = head_mask.unsqueeze(1).unsqueeze(-1).unsqueeze(-1)  # We can specify head_mask for each layer
```

```
        head_mask = head_mask.to(dtype=next(self.parameters()).dtype) # switch to fload if need + fp16 compatibility
    else:
        head_mask = [None] * self.config.num_hidden_layers

    embedding_output = self.embeddings(input_ids, position_ids=position_ids, token_type_ids=token_type_ids)
    encoder_outputs = self.encoder(embedding_output,
                                   extended_attention_mask,
                                   head_mask=head_mask)
    sequence_output = encoder_outputs[0]
    if self.config.smart_pooling:
        if task_id is None:
            layer_weight = self.gate_mask(self.layer_weights, self.config.lp_reg_type, self.layer_mask_size, input_ids.device)
        else:
            layer_weight = self.gate_mask(self.layer_weights[task_id], self.config.lp_reg_type, self.layer_mask_size, input_ids.device)
        if self.layer_fix_mask is not None:
            layer_weight = layer_weight * torch.FloatTensor(self.layer_fix_mask).to(device=input_ids.device)
        if layer_mask is not None:
            layer_weight = layer_weight * torch.FloatTensor(layer_mask).to(device=input_ids.device)
            # layer_mask.to(dtype=next(self.parameters()).dtype)
        if self.config.gate_normalize == 1:
```

```
                layer_weight = nn.Softmax(dim=-1)(layer_weight)
            if self.keep_part_grads and (layer_mask is None):
                self.layer_mask_weight = layer_weight
                self.layer_mask_weight.retain_grad()
            if self.config.gate_dropout == 1:
                layer_weight = self.gate_dropout(layer_weight)
            layer_weight = layer_weight.unsqueeze(-1).unsqueeze(-1).expand(self.config.num_hidden_layers, -1, self.config.hidden_size)
            sequence_output = torch.sum((torch.stack(encoder_outputs[-1]) * layer_weight), dim=0).unsqueeze(1)
        pooled_output = self.pooler(sequence_output)
        outputs = (sequence_output, pooled_output,) + encoder_outputs[1:]  # add hidden_states and attentions if they are here
        return outputs  # sequence_output, pooled_output, (hidden_states), (attentions)

class BertForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):
        super(BertForSequenceClassification, self).__init__(config)
        self.num_labels = config.num_labels
        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, self.config.num_labels)
        self.prediction_vals = None
        self.apply(self.init_weights)

```

```python
    def forward(self, input_ids, token_type_ids=None, attention_mask=None, labels=None, position_ids=None, head_mask=None, layer_mask=None):
        outputs = self.bert(input_ids, position_ids=position_ids, token_type_ids=token_type_ids, attention_mask=attention_mask, head_mask=head_mask, layer_mask=layer_mask)
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        self.prediction_vals = logits
        outputs = (logits,) + outputs[2:]  # add hidden states and attention if they are here
        if labels is not None:
            if self.num_labels == 1:
                #  We are doing regression
                loss_fct = MSELoss()
                loss = loss_fct(logits.view(-1), labels.view(-1))
            else:
                loss_fct = CrossEntropyLoss()
                loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            outputs = (loss,) + outputs

        return outputs
```

A.2 demo/async_demo.py

```
# -*- coding: utf-8 -*-
import json
import argparse
import numpy as np
import time, sys, os
from random import randrange
from nltk.corpus import wordnet
from .demo_model_bridge import Bridge
from flask import Flask, render_template, request
from flask_socketio import SocketIO, emit, disconnect

ngram_distribution = None
mask_spec_chars = False
head_count, layer_count = 0, 0
args, model_bridge = None, None
model_width = 1210
model_height = 5280

app = Flask(__name__)
app.config['SECRET_KEY'] = 'secret'
socketio = SocketIO(app)
model_type = 'player_norm'

def extract_ngrams():
    def add_to_dict(key_str, _dict):
        if key_str in _dict:
            _dict[key_str] += 1
```

```
    else:
        _dict[key_str] = 1
    return _dict
def add_ngram(key_cat, token, _dict):
    if key_cat in _dict:
        _dict[key_cat] = add_to_dict(token, _dict[key_cat])
    else:
        _dict[key_cat] = add_to_dict(token, {})
    return _dict
ngram_dict = {'2gram': {}, '3gram': {}, 'vocab': {}}
for line in open(args.ngram_source):
    tokens = line.rstrip().lower().split(" ")
    if len(tokens) == 1:
        if ('-' not in tokens[0]) and ('_' not in tokens[0]):
            ngram_dict['vocab'] = add_to_dict(tokens[0],
ngram_dict['vocab'])
        ngram_dict['2gram'] = add_ngram("b_", tokens[0],
ngram_dict['2gram'])
        ngram_dict['2gram'] = add_ngram("a_", tokens[0],
ngram_dict['2gram'])
        ngram_dict['3gram'] = add_ngram("_", tokens[0],
ngram_dict['3gram'])
    else:
        for i in range(len(tokens)):
            if ('-' not in tokens[i]) and ('_' not in tokens[i]):
                ngram_dict['vocab'] = add_to_dict(tokens[i],
ngram_dict['vocab'])
```

```
                if i == 0:
                    ngram_dict['2gram'] = add_ngram("a_", tokens[0], ngram_dict['2gram'])
                    ngram_dict['2gram'] = add_ngram("b_%s"%tokens[1], tokens[0], ngram_dict['2gram'])
                    ngram_dict['3gram'] = add_ngram("_%s"%tokens[1], tokens[0], ngram_dict['3gram'])
                elif i == len(tokens)-1:
                    ngram_dict['2gram'] = add_ngram("a_%s"%tokens[i-1], tokens[i], ngram_dict['2gram'])
                    ngram_dict['2gram'] = add_ngram("b_", tokens[i], ngram_dict['2gram'])
                    ngram_dict['3gram'] = add_ngram("%s_"%tokens[i-1], tokens[i], ngram_dict['3gram'])
                else:
                    ngram_dict['2gram'] = add_ngram("a_%s"%tokens[i-1], tokens[i], ngram_dict['2gram'])
                    ngram_dict['2gram'] = add_ngram("b_%s"%tokens[i+1], tokens[i], ngram_dict['2gram'])
                    ngram_dict['3gram'] = add_ngram("%s_%s"%(tokens[i-1], tokens[i+1]), tokens[i], ngram_dict['3gram'])
    with open(args.ngram_distribution ,'w') as f:
        json.dump(ngram_dict, f)

def mask_special_chars(data, idx, mask=False):
    if mask:
        if data.min() < 0:
```

```
            data = data - data.min()
        if len(data.shape) == 1:
            for i in idx:
                data[i] = 0
        elif len(data.shape) == 2:
            for i in idx:
                data[i,:] = 0
                data[:,i] = 0
        return data
    else:
        return data

def normalization(v, ax=None, zero_one=True, doAbs=False):
    if v is None:
        raise RuntimeError("array is None!")
    v = np.array(v, dtype='float32')
    assert len(v.shape) <= 2
    if doAbs:
        v = np.abs(v)
    max_value = np.max(v, axis=ax)
    min_value = np.min(v, axis=ax)
    if ax is None:
        det = (max_value - min_value)
        det = det if det > 0 else 1
        v = (v - min_value) / det
    elif ax == 1 or ax == -1:
        det = (max_value - min_value)[:,None]
```

```
        det = np.where(det==0, 1, det)
        v = (v - min_value[:,None]) / det
    else:
        det = (max_value - min_value)[None,:]
        det = np.where(det==0, 1, det)
        v = (v - min_value[None,:]) / det
    if not zero_one:
        v = (2 * v) - 1
    return v.flatten().tolist()

# General run of the model
def interpretation_extraction(inp1, inp2, pairwise, task, user,
    mask_special=None):
    if mask_special is None:
        mask_special = mask_spec_chars
    data_list =  [inp1, inp2] if pairwise else [inp1]
    data_batch, input_text = model_bridge.parse(data_list, task)
    special_idx = [0]
    for i in range(len(input_text)):
        if input_text[i] == '[SEP]':
            special_idx.append(i)
    model_info = model_bridge._demo_run(task, data_batch, user)

    max_word_len = (max([len(w) for w in input_text]))
    json_dict = {
        "head_names": ["Head %d"%i for i in range(head_count)],
        "layer_names": ["Layer %d"%i for i in range(layer_count)],
```

```
        "head_count": head_count,
        "y_margin": 9*max_word_len,
        "x_margin": int(5.5*max_word_len),
        "len": len(input_text),
        "x": ["%d_%s"%(i, input_text[i]) for i in range(len(input_text))],
        "classes": model_bridge.get_class_names(task),
        "logit": normalization(model_info['logit']),
        "prediction": model_bridge.get_prediction_string(task, model_info["prediction"]),
        "layers": [],
        "layers_impact_W": normalization(model_info['layer_weight_impact']['w']),
        "layers_impact_G": normalization(model_info['layer_weight_impact']['g']),
        "layers_impact_T": normalization(np.multiply(model_info['layer_weight_impact']['w'], model_info['layer_weight_impact']['g'])),
        "embedding_W_main": normalization(mask_special_chars(
            np.abs(model_info['embedding']['w']).sum(axis=-1), special_idx, mask_special)),
        "embedding_G_main": normalization(mask_special_chars(
            np.abs(model_info['embedding']['g']).sum(axis=-1), special_idx, mask_special)),
        "embedding_T_main": normalization(mask_special_chars(
            np.abs(np.multiply(model_info['embedding']['w'], model_info['embedding']['g'])).sum(axis=-1), special_idx,
```

```
mask_special)),
         "sub_embedding_WG": []
 }
 if user == "Developer":
     for i in range(layer_count):
         layer_dict = {"idx": i,
                        "W_output": normalization(
mask_special_chars(np.abs(model_info['attetion_layer_%d'%i]['
output']['w']).sum(axis=-1), special_idx, mask_special)),
                        "G_output": normalization(
mask_special_chars(np.abs(model_info['attetion_layer_%d'%i]['
output']['g']).sum(axis=-1), special_idx, mask_special)),
                        "T_output": normalization(
mask_special_chars(np.abs(np.multiply(model_info['attetion_layer_%
d'%i]['output']['w'], model_info['attetion_layer_%d'%i]['output'][
'g'])).sum(axis=-1), special_idx, mask_special)),
                        "W_Head": [],
                        "G_Head": [],
                        "T_Head": []
                       }
         W_impact, G_impact, T_impact = [], [], []
         for j in range(head_count):
             layer_dict["W_Head"].append(normalization(
mask_special_chars(np.abs(model_info['attetion_layer_%d'%i]['head_
%d_probs'%j]['w']), special_idx, mask_special)))
             W_impact.append(mask_special_chars(model_info['
attetion_layer_%d'%i]['head_%d_probs'%j]['w'], special_idx,
```

```
mask_special).sum())
                layer_dict["G_Head"].append(normalization(mask_special_chars(np.abs(model_info['attetion_layer_%d'%i]['head_%d_probs'%j]['g']), special_idx, mask_special)))
                G_impact.append(mask_special_chars(model_info['attetion_layer_%d'%i]['head_%d_probs'%j]['g'], special_idx, mask_special).sum())
                layer_dict["T_Head"].append(normalization(mask_special_chars(np.abs(np.multiply(model_info['attetion_layer_%d'%i]['head_%d_probs'%j]['w'], model_info['attetion_layer_%d'%i]['head_%d_probs'%j]['g'])), special_idx, mask_special)))
                T_impact.append(mask_special_chars(np.abs(np.multiply(model_info['attetion_layer_%d'%i]['head_%d_probs'%j]['w'], model_info['attetion_layer_%d'%i]['head_%d_probs'%j]['g'])), special_idx, mask_special).sum())
            layer_dict["W_impact"] = normalization(np.array(W_impact))
            layer_dict["G_impact"] = normalization(np.array(G_impact))
            layer_dict["T_impact"] = normalization(np.array(T_impact))
            json_dict["layers"].append(layer_dict)

    json_dict["sub_embedding_WG"] = [
                                    {"name": "Word",
                                     "W": normalization(mask_special_chars(np.abs(model_info['words_embedding']['w']).sum(
```

```
axis=-1), special_idx, mask_special)),
                                          "G": normalization(
mask_special_chars(np.abs(model_info['words_embedding']['g']).sum(
axis=-1), special_idx, mask_special)),
                                          "T": normalization(
mask_special_chars(np.abs(np.multiply(model_info['words_embedding'
]['w'], model_info['words_embedding']['g'])).sum(axis=-1),
special_idx, mask_special))
                                         },
                                         {"name": "Position",
                                          "W": normalization(
mask_special_chars(np.abs(model_info['position_embedding']['w']).
sum(axis=-1), special_idx, mask_special)),
                                          "G": normalization(
mask_special_chars(np.abs(model_info['position_embedding']['g']).
sum(axis=-1), special_idx, mask_special)),
                                          "T": normalization(
mask_special_chars(np.abs(np.multiply(model_info['
position_embedding']['w'], model_info['position_embedding']['g']))
.sum(axis=-1), special_idx, mask_special))
                                         },
                                         {"name": "Type",
                                          "W": normalization(
mask_special_chars(np.abs(model_info['token_type_embedding']['w'])
.sum(axis=-1), special_idx, mask_special)),
                                          "G": normalization(
mask_special_chars(np.abs(model_info['token_type_embedding']['g'])
```

```
    .sum(axis=-1), special_idx, mask_special)),
                                    "T": normalization(
    mask_special_chars(np.abs(np.multiply(model_info['
    token_type_embedding']['w'], model_info['token_type_embedding']['g
    '])).sum(axis=-1), special_idx, mask_special))
                                    }
                                   ]
    return json_dict

# Automatic word modifications
def wordnet_token(org_input, idx):
    synonyms = []
    for syn in wordnet.synsets(org_input[idx].lower()):
        for l in syn.lemmas():
            w = l.name()
            if w not in synonyms and (w.lower()!=org_input[idx].lower
    ()) and (len(w.split(' ')) == 1) and (len(w.split('_')) == 1) and
    (len(w.split('-')) == 1):
                synonyms.append(w.lower())
    if len(synonyms) == 0:
        return org_input[idx]
    random_value = randrange(len(synonyms))
    return synonyms[random_value]

def sampling_token(org_input, idx):
    global ngram_distribution
    if ngram_distribution is None:
```

```
        with open(args.ngram_distribution ,'r') as f:
            ngram_distribution = json.load(f)
    previous_word, next_word = '', ''
    if (idx > 0) and (org_input[idx-1] != '[CLS]') and (org_input[idx-1] != '[SEP]'):
        previous_word = org_input[idx-1]
    if (idx < len(org_input)) and (org_input[idx+1] != '[CLS]') and (org_input[idx+1] != '[SEP]'):
        next_word = org_input[idx+1]
    candidate_list = {}
    if "a_%s"%previous_word in ngram_distribution['2gram']:
        candidate_list = ngram_distribution['2gram']["a_%s"%previous_word]
        if "b_%s"%next_word in ngram_distribution['2gram']:
            tmp = ngram_distribution['2gram']["b_%s"%next_word]
            for k in tmp.keys():
                if k in candidate_list:
                    candidate_list[k] += tmp[k]
                else:
                    candidate_list[k] = tmp[k]
        if "%s_%s"%(previous_word,next_word) in ngram_distribution:
             tmp = ngram_distribution['3gram']["%s_%s"%(previous_word,next_word)]
             for k in tmp.keys():
                 if k in candidate_list:
                     candidate_list[k] += tmp[k]
                 else:
```

```
                    candidate_list[k] = tmp[k]
    elif "b_%s"%next_word in ngram_distribution['2gram']:
        candidate_list = ngram_distribution['2gram']["b_%s"%next_word]
    if org_input[idx] in candidate_list:
        del candidate_list[org_input[idx]]
    elim_list = []
    for k in candidate_list.keys():
        if ('-' in k) or ('_' in k):
            elim_list.append(k)
    for k in elim_list:
        del candidate_list[k]
    if len(candidate_list) > 0:
        freq_sum = 0
        for k in candidate_list.keys():
            freq_sum += candidate_list[k]
        random_value = randrange(freq_sum) + 1
        counter = 0
        for k in candidate_list.keys():
            counter += candidate_list[k]
            if counter >= random_value:
                return k
    else:
        freq_sum = 0
        for k in ngram_distribution['vocab'].keys():
            freq_sum += ngram_distribution['vocab'][k]
        random_value = randrange(freq_sum) + 1
```

```
        counter = 0
        for k in ngram_distribution['vocab'].keys():
            counter += ngram_distribution['vocab'][k]
            if counter >= random_value:
                return k

def _word_modification(method, org_input, idx):
    if method == 'Remove':
        return '[REMOVED]'
    elif method == 'Zero Out':
        return '[ZERO]'
    elif method == 'Unknown':
        return '[UNK]'
    elif method == 'Wordnet':
        return wordnet_token(org_input, idx)
    elif method == 'Sampling':
        return sampling_token(org_input, idx)
    else:
        raise RuntimeError('The modification method is not defined.')
    return org_input[idx]

def word_modification_process(inp1, inp2, pairwise, task, method=None
    , modif_inp1=None, modif_inp2=None):
    data_list =  [inp1, inp2] if pairwise else [inp1]
    word_modification = True if (method is not None) and (method in [
    'Remove', 'Zero Out', 'Unknown']) else False
    modif_data_list = None
```

```
    if word_modification:
        modif_data_list = [modif_inp1, modif_inp2] if pairwise else [modif_inp1]
    elif method is not None:
        data_list = [modif_inp1, modif_inp2] if pairwise else [modif_inp1]
    data_batch, input_text = model_bridge.parse(data_list, task, word_modification, modif_data_list, word_analyses=True)
    if word_modification:
        input_text = ['[CLS]'] + ((modif_inp1.split(' ') + ['[SEP]'] + modif_inp2.split(' ')) if pairwise else modif_inp1.split(' ')) + ['[SEP]']
    model_info = model_bridge._demo_word_change_run(task, data_batch)
    return input_text, model_info['prediction'], normalization(model_info['logit'])

# Structure modification
def structure_modification_process(inp1, inp2, pairwise, task, head_mask=None, layer_mask=None):
    data_list =  [inp1, inp2] if pairwise else [inp1]
    data_batch, input_text = model_bridge.parse(data_list, task)
    model_info = model_bridge._demo_structure_change_run(task, data_batch, head_mask=head_mask, layer_mask=layer_mask)
    return input_text, model_info['prediction'], normalization(model_info['logit'])

# Main Templates
```

```
@app.route("/")
def index():
    info = {
        "task_set": model_bridge.task_list,
        "task_pair": model_bridge.task_pair_list,
        "task_count": len(model_bridge.task_list),
        "selected_task_id": 0,
        "selected_user": "Developer",
        "input01": "",
        "input02": ""
    }
    return render_template("async_demo.html", info=info)

@app.route("/", methods=['POST'])
def my_from_post():
    inp1 = request.form['input01'].lower()
    inp2 = request.form['input02'].lower()
    task = request.form['taskcombo']
    user = request.form['usercombo']
    pairwise = (model_bridge.task_pair_list[model_bridge.task_list.
    index(task)] == "1")
    if request.form['submit'] == 'Submit':
        json_dict = interpretation_extraction(inp1, inp2, pairwise,
    task, user)
        info = {
            "task_set": model_bridge.task_list,
            "task_pair": model_bridge.task_pair_list,
```

```
            "task_count": len(model_bridge.task_list),
            "selected_task_id": model_bridge.task_list.index(task),
            "selected_user": user,
            "input01": inp1,
            "input02": inp2,
            "prediction": json_dict["prediction"],
            "head_count": head_count if len(json_dict["layers"]) > 0 else 0,
            "layer_idx": range(len(json_dict["layers"])-1, -1, -1),
            "sub_embedding_WG": ["Word", "Position", "Type"] if len(json_dict["sub_embedding_WG"]) > 0 else [],
            "json": json_dict
        }
        return render_template("async_lazy_response_d3.html", info=info)
    elif request.form['submit'] == 'Word Analyses':
        token_list, prediction, logit = word_modification_process(inp1, inp2, pairwise, task)
        info = {
            "task": task,
            "pairwise": pairwise,
            "token_list": token_list,
            "token_list_len": len(token_list),
            "token_list_cat": ['static' if (x == '[CLS]' or x == '[SEP]') else 'multi' for x in token_list],
            "original_input": ' '.join(token_list),
            "input01": inp1,
```

```
            "input02": inp2,
            "classes": model_bridge.get_class_names(task),
            "org_prediction": model_bridge.get_prediction_string(task, prediction),
            "org_logit_vector": logit
        }
        return render_template("async_word_analyze_d3.html", info=info)
    elif request.form['submit'] == 'Layer and Attention Head Analyses':
        token_list, prediction, logit = structure_modification_process(inp1, inp2, pairwise, task)
        info = {
            "model_width": model_width,
            "model_height": model_height,
            "task": task,
            "classes": model_bridge.get_class_names(task),
            "original_input": ' '.join(token_list),
            "org_prediction": model_bridge.get_prediction_string(task, prediction),
            "org_logit_vector": logit
        }
        return render_template("async_structure_analyze_d3.html", info=info)

# Word Analyses Template
@socketio.on('change_modification_type', namespace='/word_analyze')
```

```python
def change_modification_type_message(message):
    task = message['task']
    method = message['type']
    org_input = message['org_input'].split(' ')
    cur_input = message['cur_input'].split(' ')
    pairwise = (model_bridge.task_pair_list[model_bridge.task_list.
   index(task)] == "1")
    inps, modif_inps, idx = [[], []], [[], []], 0
    for i, [ow, cw] in enumerate(zip(org_input, cur_input)):
        if ow != '[CLS]' and ow != '[SEP]':
            if ow == cw:
                inps[idx].append(ow)
                modif_inps[idx].append(ow)
            else:
                inps[idx].append(ow)
                modif_inps[idx].append(_word_modification(method,
   org_input, i))
        elif ow == '[SEP]':
            idx += 1
    token_list, prediction, logit = word_modification_process(' '.
   join(inps[0]), ' '.join(inps[1]), pairwise,
                                                task, method, ' '.join(
   modif_inps[0]), ' '.join(modif_inps[1]))
    response = {'text': ' '.join(token_list),
                'prediction': model_bridge.get_prediction_string(task
   , prediction),
                'logit': logit}
```

```
    emit('auto_response', response)

@socketio.on('change_words', namespace='/word_analyze')
def change_words_message(message):
    task = message['task']
    method = message['type']
    word_idx = int(message['word_idx'].split('_')[1])
    org_input = message['org_input'].split(' ')
    cur_input = message['cur_input'].split(' ')
    pairwise = (model_bridge.task_pair_list[model_bridge.task_list.
   index(task)] == "1")
    inps, modif_inps, idx = [[], []], [[], []], 0
    for i, [ow, cw] in enumerate(zip(org_input, cur_input)):
        if cw != '[CLS]' and cw != '[SEP]':
            if i != word_idx:
                inps[idx].append(ow)
                modif_inps[idx].append(cw)
            else:
                inps[idx].append(ow)
                if ow == cw:
                    modif_inps[idx].append(_word_modification(method,
    org_input, i))
                else:
                    modif_inps[idx].append(ow)
        elif cw == '[SEP]':
            idx += 1
```

```
    token_list, prediction, logit = word_modification_process(' '.join(inps[0]), ' '.join(inps[1]), pairwise,
                                                                  task, method, ' '.join(modif_inps[0]), ' '.join(modif_inps[1]))
    response = {'text': ' '.join(token_list),
                'prediction': model_bridge.get_prediction_string(task, prediction),
                'logit': logit}
    emit('auto_response', response)

@socketio.on('new_input', namespace='/word_analyze')
def new_input_message(message):
    task = message['task']
    pairwise = (model_bridge.task_pair_list[model_bridge.task_list.index(task)] == "1")
    inp1 = message['input01'].lower()
    inp2 = message['input02'].lower() if pairwise else ""
    _, prediction, logit = word_modification_process(inp1, inp2, pairwise, task)
    response = {'prediction': model_bridge.get_prediction_string(task, prediction),
                'logit': logit}
    emit('manual_response', response)

# Structure Analyses Template
@socketio.on('connect', namespace='/structure_analyze')
def structure_analyze_connect():
```

```
    graph = model_bridge.get_model_graph()
    emit('connect_response', graph)

@socketio.on('change_structure', namespace='/structure_analyze')
def structure_change_message(message):
    task = message['task']
    _input = message['input'].split(' ')
    pairwise = (model_bridge.task_pair_list[model_bridge.task_list.index(task)] == "1")
    inps, idx = [[], []], 0
    if pairwise:
        for w in _input:
            if w != '[CLS]' and w != '[SEP]':
                inps[idx].append(w)
            elif w == '[SEP]':
                idx += 1
    else:
        inps[0] = _input[1:-1]
    head_status = message['head_status']
    head_mask = [[1]*head_count]*layer_count
    active_heads = True
    for i in range(layer_count):
        for j in range(head_count):
            if ("Layer_%d_Head_%d"%(i,j) in head_status) and (head_status["Layer_%d_Head_%d"%(i,j)] == 0):
                head_mask[i][j] = 0
                active_heads = False
```

```
    if active_heads:
        head_mask = None
    layer_status = message['layer_status']
    layer_mask = []
    active_layers = True
    for i in range(layer_count):
        if ("Layer_%d_Collector"%(i) in layer_status) and (layer_status["Layer_%d_Collector"%(i)] == 0):
            layer_mask.append(0.)
            active_layers = False
        else:
            layer_mask.append(1.)
    if active_layers:
        layer_mask = None
    _, prediction, logit = structure_modification_process(' '.join(inps[0]), ' '.join(inps[1]), pairwise, task, head_mask, layer_mask)
    response = {'prediction': model_bridge.get_prediction_string(task, prediction),
                'logit': logit}
    emit('change_response', response)

@app.after_request
def add_header(r):
    """
    Add headers to both force latest IE rendering engine or Chrome Frame,
```

```
    and also to cache the rendered page for 10 minutes.
    """
    r.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"
    r.headers["Pragma"] = "no-cache"
    r.headers["Expires"] = "0"
    r.headers['Cache-Control'] = 'public, max-age=0'
    return r

def main(inp_args):
    global model_bridge, args, head_count, layer_count
    args = inp_args
    if args.ngram_extraction:
        extract_ngrams()
    else:
        if args.model_type != "":
            model_type = args.model_type
        model_bridge = Bridge(model_type)
        head_count = model_bridge.head_count
        layer_count = model_bridge.layer_count
        socketio.run(app, host=args.ip, port=args.port)
```

A.3 demo/demo_model_bridge.py

```
import re
import torch
import numpy as np
```

```python
from run_tasks import (compute_metrics, convert_examples_to_features, output_modes, processors, InputExample, InputFeatures)
from pyencoder import (BertConfig, BertForSequenceClassification, BertForMultiSequenceClassification)
from pytorch_transformers import (WEIGHTS_NAME, BertTokenizer)
from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler, TensorDataset)

class Bridge(object):
    def pre_process_modif(self, main_text, modif_text):
        main_text = main_text.split(' ')
        modif_text = modif_text.split(' ')
        assert len(main_text) == len(modif_text)
        for i in range(len(main_text)):
            if modif_text[i] == '[REMOVED]':
                main_text[i] = ''
        text = ' '.join(main_text)
        text = re.sub(r"  +", r" ", text)
        return text

    def parse(self, data_list, task, word_modification=False, modif_data_list=None, word_analyses=False):
        if word_modification:
            data_list[0] = self.pre_process_modif(data_list[0], modif_data_list[0])
            if len(data_list)>1:
```

```
            data_list[1] = self.pre_process_modif(data_list[1], modif_data_list[1])
        examples = [InputExample(guid=0, text_a=data_list[0], text_b=(data_list[1] if len(data_list)>1 else None), label="0")]
        tokenizer = self.model_list[task][1]
        token_count = 4 + len(data_list[0].split(' ')) + ((1 + len(data_list[1].split(' '))) if len(data_list)>1 else 0)
        features, tokens = convert_examples_to_features(examples, [], -1, tokenizer, "regression", cls_token_at_end=False, cls_token=tokenizer.cls_token, sep_token=tokenizer.sep_token, cls_token_segment_id=0, pad_on_left=False, pad_token_segment_id=0, pass_text=True, only_split=word_analyses)
        # Convert to Tensors and build dataset
        all_input_ids = torch.tensor([f.input_ids for f in features], dtype=torch.long)
        all_input_mask = torch.tensor([f.input_mask for f in features], dtype=torch.long)
        all_segment_ids = torch.tensor([f.segment_ids for f in features], dtype=torch.long)
        if word_modification:
            modif_tokens = ['[CLS]'] + ((modif_data_list[0].split(' ') + ['[SEP]'] + modif_data_list[1].split(' ')) if len(modif_data_list)>1 else modif_data_list[0].split(' ')) + ['[SEP]']
            if '[ZERO]' in modif_tokens:
                for i in range(len(modif_tokens)):
                    if modif_tokens[i] == '[ZERO]':
                        all_input_mask[0][i] = 0
```

```
                    all_input_ids[0][i] = tokenizer._convert_token_to_id(tokenizer.mask_token)
            elif '[UNK]' in modif_tokens:
                for i in range(len(modif_tokens)):
                    if modif_tokens[i] == '[UNK]':
                        all_input_ids[0][i] = tokenizer._convert_token_to_id(tokenizer.unk_token)

        batch = [all_input_ids, all_input_mask, all_segment_ids]
        return batch, tokens

    # RUN MODELS
    def _demo_run(self, task, batch, user):
        self.model_list[task][2].eval()
        batch = tuple(t.to(self.device) for t in batch)
        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'token_type_ids': batch[2],
                  'labels':         None}
        outputs = self.model_list[task][2](**inputs)
        logits = outputs[0]
        pred = logits.detach().cpu().numpy()[0]
        dy_dl = torch.ones((1,1)).to(self.device)
        tmp_model = self.model_list[task][2].module if hasattr(self.model_list[task][2], 'module') else self.model_list[task][2]
        # compute the gradient of the output respect to desired units and components of the model
```

```
        if task != "STS-B":
            tmp_model.prediction_vals[:,pred[0]].backward(dy_dl)
        else:
            tmp_model.prediction_vals.backward(dy_dl)
        info = {'prediction': pred[0],
                'logit': logits.detach().cpu().numpy()[0],
                'embedding': {'w': tmp_model.bert.embeddings.embeddig_list[3].detach().cpu().numpy()[0],
                              'g': tmp_model.bert.embeddings.embeddig_list[3].grad.cpu().numpy()[0]
                             }
               }
        if tmp_model.bert.layer_mask_weight is None:
            info['layer_weight_impact'] = {'w': np.array([0]*(self.layer_count-1)+[1]),
                                           'g': np.array([0]*(self.layer_count-1)+[1])}
        else:
            info['layer_weight_impact'] = {'w': tmp_model.bert.layer_mask_weight.detach().cpu().numpy(),
                                           'g': tmp_model.bert.layer_mask_weight.grad.cpu().numpy()}
        if user == 'Developer':
            info['words_embedding'] = {
                            'w': tmp_model.bert.embeddings.embeddig_list[0].detach().cpu().numpy()[0],
```

```
                        'g': tmp_model.bert.embeddings.embeddig_list[0].grad.cpu().numpy()[0]
                     }
        info['position_embedding'] = {
                        'w': tmp_model.bert.embeddings.embeddig_list[1].detach().cpu().numpy()[0],
                        'g': tmp_model.bert.embeddings.embeddig_list[1].grad.cpu().numpy()[0]
                     }
        info['token_type_embedding'] = {
                        'w': tmp_model.bert.embeddings.embeddig_list[2].detach().cpu().numpy()[0],
                        'g': tmp_model.bert.embeddings.embeddig_list[2].grad.cpu().numpy()[0]
                     }
        for _layer in range(self.layer_count):
            info['attetion_layer_%d'%_layer] = {}
            info['attetion_layer_%d'%_layer]['output'] = {
                                        'w': tmp_model.bert.encoder.layer[_layer].attention.self.context_output.detach().cpu().numpy()[0],
                                        'g': tmp_model.bert.encoder.layer[_layer].attention.self.context_output.grad.cpu().numpy()[0]
                                    }
            att_probs = tmp_model.bert.encoder.layer[_layer].attention.self.att_probs.detach().cpu().numpy()[0]
```

```
                att_probs_grad = tmp_model.bert.encoder.layer[_layer].attention.self.att_probs.grad.cpu().numpy()[0]
                for _head in range(self.head_count):
                    info['attetion_layer_%d'%_layer]['head_%d_probs'%_head] = {
                        'w': att_probs[_head],
                        'g': att_probs_grad[_head]
                    }
        return info

    def _demo_word_change_run(self, task, batch):
        self.model_list[task][2].eval()
        batch = tuple(t.to(self.device) for t in batch)
        with torch.no_grad():
            inputs = {'input_ids':      batch[0],
                      'attention_mask': batch[1],
                      'token_type_ids': batch[2],
                      'labels':         None}
            outputs = self.model_list[task][2](**inputs)
            logits = outputs[0]
            pred = np.squeeze(logits.detach().cpu().numpy())
            info = {'prediction': pred,
                    'logit': logits.detach().cpu().numpy()[0],}
            return info

```

```
    def _demo_structure_change_run(self, task, batch, head_mask=None,
     layer_mask=None):
        self.model_list[task][2].eval()
        batch = tuple(t.to(self.device) for t in batch)
        with torch.no_grad():
            inputs = {'input_ids':      batch[0],
                      'attention_mask': batch[1],
                      'token_type_ids': batch[2],
                      'labels':         None,
                      'head_mask':      None if head_mask is None
    else torch.FloatTensor(head_mask).to(device=self.device),
                      'layer_mask':     None if layer_mask is None
    else layer_mask}
            outputs = self.model_list[task][2](**inputs)
            logits = outputs[0]
            pred = np.squeeze(logits.detach().cpu().numpy())
            info = {'prediction': pred,
                    'logit': logits.detach().cpu().numpy()[0],}
            return info
```

A.4 HTML Script

```
<script>
    var json = {{info["json"]|safe}}

    var margin = {top: 5, right: 5, bottom: {{info["json"]["
  x_margin"]|safe}}, left: {{info["json"]["y_margin"]|safe}} },
```

```
    width = 35*{{info["json"]["len"]|safe}},
    height = 35*{{info["json"]["len"]|safe}},
    vector_height = 35;

  // Labels of row and columns
  var sentence = {{ info["json"]["x"]|safe }}
  var classes = {{info["json"]["classes"]|safe}}
  var heads = {{info["json"]["head_names"]|safe}}
  var layers = {{info["json"]["layer_names"]|safe}}

  // Build X scales and axis:
  var x = d3.scaleBand()
    .range([ 0, width ])
    .domain(sentence)
    .padding(0.01);

  // Build X scales and axis:
  var c_x = d3.scaleBand()
    .range([ 0, width ])
    .domain(classes)
    .padding(0.01);

  var l_x = d3.scaleBand()
    .range([ 0, width ])
    .domain(layers)
    .padding(0.01);

```

```
// Build X scales and axis:
var h_x = d3.scaleBand()
  .range([ 0, width ])
  .domain(heads)
  .padding(0.01);

// Build X scales and axis:
var y = d3.scaleBand()
  .range([ height, 0 ])
  .domain(sentence)
  .padding(0.01);

var v_y = d3.scaleBand()
  .range([ vector_height, 0 ])
  .domain([''])
  .padding(0.01);

// Build color scale
var myColor = d3.scaleLinear()
  .range(["white", "#250082"])
  .domain([0,1])

function draw_vector(id_str, data_array, x_axis, x_lbls){
  var svg = d3.select(id_str)
    .append("svg")
      .attr("width", width + margin.left + margin.right)
```

```
          .attr("height", vector_height + margin.top + margin.
bottom)
        .append("g")
          .attr("transform", "translate(" + margin.left + "," +
margin.top + ")");
      svg.append("g")
        .attr("transform", "translate(0," + vector_height + ")")
        .call(d3.axisBottom(x_axis))
        .selectAll("text")
          .style("text-anchor", "end")
          .attr("transform", "rotate(-35)");
      var vector = svg.selectAll()
        .data(data_array, function(d, i) {return x_lbls[(i%x_lbls.
length)]+':'+'';})
        .enter()
          .append("rect")
          .attr("x", function(d, i) { return x(x_lbls[(i%x_lbls.
length)]) })
          .attr("y", function(d, i) { return v_y('') })
          .attr("width", x.bandwidth() )
          .attr("height", v_y.bandwidth() )
          .style("fill", function(d, i) { return myColor(d)} )

      vector.append("title")
        .text(function(d) { return "value: " + d; });
    }
    function draw_matrix(id_str, data_array){
```

```
      var svg = d3.select(id_str)
        .append("svg")
          .attr("width", width + margin.left + margin.right)
          .attr("height", height + margin.top + margin.bottom)
        .append("g")
          .attr("transform",
            "translate(" + margin.left + "," + margin.top + ")");
      svg.append("g")
        .attr("transform", "translate(0," + height + ")")
        .call(d3.axisBottom(x))
        .selectAll("text")
          .style("text-anchor", "end")
          .attr("transform", "rotate(-35)");
      svg.append("g")
        .call(d3.axisLeft(y));
      var heatmap = svg.selectAll()
        .data(data_array, function(d, i) {return sentence[(i%
sentence.length)]+':'+sentence[Math.floor(i/sentence.length)];})
        .enter()
          .append("rect")
          .attr("x", function(d, i) { return x(sentence[(i%sentence
.length)]) })
          .attr("y", function(d, i) { return y(sentence[Math.floor(
i/sentence.length)]) })
          .attr("width", x.bandwidth() )
          .attr("height", y.bandwidth() )
          .style("fill", function(d, i) { return myColor(d); })
```

```
      heatmap.append("title")
        .text(function(d) { return "value: " + d; });
    }

    function draw_layer_weight_impact(){
      draw_vector("#W_layers_impact_div", json.layers_impact_W, l_x
, layers);
      draw_vector("#G_layers_impact_div", json.layers_impact_G, l_x
, layers);
      draw_vector("#T_layers_impact_div", json.layers_impact_T, l_x
, layers);
    }
    function remove_layer_weight_impact(){
      $("#W_layers_impact_div").empty();
      $("#G_layers_impact_div").empty();
      $("#T_layers_impact_div").empty();
    }
    function draw_main_embedding(){
      draw_vector("#W_Embd", json.embedding_W_main, x, sentence);
      draw_vector("#G_Embd", json.embedding_G_main, x, sentence);
      draw_vector("#T_Embd", json.embedding_T_main, x, sentence);
    }
    function remove_main_embedding(){
      $("#W_Embd").empty();
      $("#G_Embd").empty();
      $("#T_Embd").empty();
```

```
    }
    function draw_sub_embedding(){
      json.sub_embedding_WG.forEach(function(d){
        draw_vector("#W_Subembd_"+d.name+"_div", d.W, x, sentence)
        draw_vector("#G_Subembd_"+d.name+"_div", d.G, x, sentence)
        draw_vector("#T_Subembd_"+d.name+"_div", d.T, x, sentence)
      });
    }
    function remove_sub_embedding(){
      json.sub_embedding_WG.forEach(function(d){
        $("#W_Subembd_"+d.name+"_div").empty();
        $("#G_Subembd_"+d.name+"_div").empty();
        $("#T_Subembd_"+d.name+"_div").empty();
      });
    }
    function draw_layer_output(_dict, idx){
      draw_vector("#W_L_"+idx+"_output_div", _dict.W_output, x, sentence);
      draw_vector("#G_L_"+idx+"_output_div", _dict.G_output, x, sentence);
      draw_vector("#T_L_"+idx+"_output_div", _dict.T_output, x, sentence);
    }
    function remove_layer_output(idx){
      $("#W_L_"+idx+"_output_div").empty();
      $("#G_L_"+idx+"_output_div").empty();
      $("#T_L_"+idx+"_output_div").empty();
```

```
    }
    function draw_layer_impact(_dict, idx){
      draw_vector("#W_L_"+idx+"_impact_div", _dict.W_impact, h_x, heads);
      draw_vector("#G_L_"+idx+"_impact_div", _dict.G_impact, h_x, heads);
      draw_vector("#T_L_"+idx+"_impact_div", _dict.T_impact, h_x, heads);
    }
    function remove_layer_impact(idx){
      $("#W_L_"+idx+"_impact_div").empty();
      $("#G_L_"+idx+"_impact_div").empty();
      $("#T_L_"+idx+"_impact_div").empty();
    }
    function draw_attention_head(_dict, layer_idx){
      var i;
      for (i = 0; i < json.head_count; i++) {
        draw_matrix("#W_L_"+layer_idx+"_head_"+i+"_div", _dict.W_Head[i]);
        draw_matrix("#G_L_"+layer_idx+"_head_"+i+"_div", _dict.G_Head[i]);
        draw_matrix("#T_L_"+layer_idx+"_head_"+i+"_div", _dict.T_Head[i]);
      }
    }
    function remove_attention_head(layer_idx){
      var i;
```

```
      for (i = 0; i < json.head_count; i++) {
        $("#W_L_"+layer_idx+"_head_"+i+"_div").empty();
        $("#G_L_"+layer_idx+"_head_"+i+"_div").empty();
        $("#T_L_"+layer_idx+"_head_"+i+"_div").empty();
      }
    }

    draw_vector("#logit_div", json.logit, c_x, classes);
    draw_layer_weight_impact();

    $(function(){
      $(".collapsible").click(function(e){
        this.classList.toggle("active");
        var parent = this.parentElement;
        var content = this.nextElementSibling;
        if (content.style.maxHeight){
          content.style.maxHeight = null;
          console.log($(e.target).attr("cat"))
          if ($(e.target).attr("cat") == "main_embedding"){
            remove_main_embedding();
          } else if ($(e.target).attr("cat") == "sub_embedding"){
            remove_sub_embedding();
          } else if ($(e.target).attr("cat") == "layer_output"){
            idx = $(e.target).attr("layer_idx")
            remove_layer_output(idx);
          } else if ($(e.target).attr("cat") == "layer_impact"){
            idx = $(e.target).attr("layer_idx")
```

```
            remove_layer_impact(idx);
          } else if ($(e.target).attr("cat") == "head_output"){
            idx = $(e.target).attr("layer_idx")
            remove_attention_head(idx);
          }
        } else {
          if ($(e.target).attr("cat") == "main_embedding"){
            draw_main_embedding();
          } else if ($(e.target).attr("cat") == "sub_embedding"){
            draw_sub_embedding();
          } else if ($(e.target).attr("cat") == "layer_output"){
            idx = parseInt($(e.target).attr("layer_idx"), 10);
            draw_layer_output(json.layers[idx], idx);
          } else if ($(e.target).attr("cat") == "layer_impact"){
            idx = parseInt($(e.target).attr("layer_idx"), 10);
            draw_layer_impact(json.layers[idx], idx);
          } else if ($(e.target).attr("cat") == "head_output"){
            idx = parseInt($(e.target).attr("layer_idx"), 10);
            draw_attention_head(json.layers[idx], idx);
          }
          content.style.maxHeight = content.scrollHeight + "px";
          if (parent.className == "content"){
            parent.style.maxHeight = parent.scrollHeight + content.scrollHeight + "px";
          } else if (parent.className == "embd_content"){
            var superparent = parent.parentElement;
```

```
              superparent.style.maxHeight = superparent.scrollHeight
+ content.scrollHeight + "px";
            }
          }
        });
      });
      $(function() {
        $("#taskcombo").on("change", function(e) {
          var s2_sts = $("option:selected", this).attr("s2_sts");
          if (s2_sts == "1"){
            $(input02_div).show();
          } else {
            $(input02_div).hide();
          }
        }).change();
      });
      $(function() {
        $("#usercombo").on("change", function(e) {
          var utype = $("option:selected", this).text();
          if (utype == "Developer"){
            $("div[cat=devop]").show()
          } else {
            $("div[cat=devop]").hide()
          }
        }).change();
      });
      $(function(){
```

```
      $("a[item-id]").click(function(e){
        this.classList.toggle("select");
        $("div[item-id="+$(e.target).attr("item-id")+"]").toggle()
      });
    });
  </script>
```

Listing A.1: JavaScript for a HTML file (async_lazy_response_d3.html) of the Demo