Université de Montréal

Recurrent Neural Models and Related Problems in Natural Language Processing

par Saizheng Zhang

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des arts et des sciences
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

avril, 2019

© Saizheng Zhang, 2019.

Résumé

Le réseau de neurones récurrent (RNN) est l'un des plus puissants modèles d'apprentissage automatique spécialisés dans la capture des variations temporelles et des dépendances de données séquentielles. Grâce à la résurgence de l'apprentissage en profondeur au cours de la dernière décennie, de nombreuses structures RNN innovantes ont été inventées et appliquées à divers problèmes pratiques, en particulier dans le domaine du traitement automatique du langage naturel (TALN). Cette thèse suit une direction similaire, dans laquelle nous proposons de nouvelles perspectives sur les propriétés structurelles des RNN et sur la manière dont les modèles RNN récemment proposés peuvent stimuler le developpement de nouveaux problèmes ouverts en TALN.

Cette thèse se compose de deux parties: l'analyse de modèle et le traitement de nouveaux problèmes ouverts. Dans la première partie, nous explorons deux aspects importants des RNN: l'architecture de leurs connexions et les opérations de base dans leurs fonctions de transition. Plus précisément, dans le premier article, nous définissons plusieurs mesures rigoureuses pour évaluer la complexité architecturale de toute architecture récurrente donnée, quelle que soit la topologie du réseau. Des expériences approfondies sur ces mesures démontrent à la fois la validité théorique de celles-ci, et l'importance de guider la conception des architectures RNN. Dans le deuxième article, nous proposons un nouveau module permettant de combiner plusieurs flux d'informations de manière multiplicative dans les fonctions de transition de base des RNN. Il a été démontré empiriquement que les RNN équipés du nouveau module possédaient de meilleures propriétés de gradient et des capacités de généralisation plus grandes sans coûts de calcul et de mémoire supplémentaires.

La deuxième partie se concentre sur deux problèmes non résolus de la TALN: comment effectuer un raisonnement avancé à sauts multiples en compréhension de texte machine, et comment incorporer des traits de personnalité dans des systèmes conversationnels. Nous recueillons deux ensembles de données à grande échelle, dans le but de motiver les progrès méthodologiques sur ces deux problèmes. Spécifiquement, dans le troisième article, nous introduisons l'ensemble de données sc HotpotQA qui contient plus de 113 000 paires question-réponse basées sur Wikipedia. La plupart des questions de HotpotQA ne peuvent résolues que par un raisonnement multi-saut précis sur plusieurs documents. Les faits à l'appui nécessaires au raisonnement sont également fournis pour aider le modèle à établir des prédictions explicables. Le quatrième article aborde le problème du manque de personnalité des chatbots. Le jeu de données persona-chat que nous proposons encourage des conversations plus engageantes et cohérentes en conditionnant la personnalité des membres en conversation sur des personnages spécifiques. Nous montrons des modèles de base entraînés sur PERSONA-CHAT sont capables d'exprimer des personnalités cohérentes et de réagir de manière plus captivante en se concentrant sur leurs propres personnages ainsi que ceux de leurs interlocuteurs.

Mots clés: réseaux de neurones récurrents, apprentissage profond, traitement automatique du langage naturel, compréhension en lecture, système de dialogue

Summary

The recurrent neural network (RNN) is one of the most powerful machine learning models specialized in capturing temporal variations and dependencies of sequential data. Thanks to the resurgence of deep learning during the past decade, we have witnessed plenty of novel RNN structures being invented and applied to various practical problems especially in the field of natural language processing (NLP). This thesis follows a similar direction, in which we offer new insights about RNNs' structural properties and how the recently proposed RNN models may stimulate the formation of new open problems in NLP.

The scope of this thesis is divided into two parts: model analysis and new open problems. In the first part, we explore two important aspects of RNNs: their connecting architectures and basic operations in their transition functions. Specifically, in the first article, we define several rigorous measurements for evaluating the architectural complexity of any given recurrent architecture with arbitrary network topology. Thoroughgoing experiments on these measurements demonstrate their theoretical validity and utility of guiding the RNN architecture design. In the second article, we propose a novel module to combine different information flows multiplicatively in RNNs' basic transition functions. RNNs equipped with the new module are empirically showed to have better gradient properties and stronger generalization capacities without extra computational and memory cost.

The second part focuses on two open problems in NLP: how to perform advanced multi-hop reasoning in machine reading comprehension and how to encode personalities into chitchat dialogue systems. We collect two different large scale datasets aiming to motivate the methodological progress on these two problems. Particularly, in the third article we introduce HotpotQA dataset containing over 113k Wikipedia based question-answer pairs. Most of the questions in HotpotQA are answerable only through accurate multi-hop reasoning over multiple documents. Supporting facts required for reasoning are also provided to help the model to make explainable predictions. The fourth article tackles the problem of the lack of personality in chatbots. The proposed PERSONA-CHAT dataset encourages more engaging and consistent conversations by forcing dialog partners conditioning on given personas. We show that baseline models trained on PERSONA-CHAT are able to express consistent personalities and to respond in more captivating ways by concentrating on personas of both themselves and other interlocutors.

Keywords: recurrent neural networks, deep learning, natural language pro-

cessing, reading comprehension, dialogue system

Contents

Résumé . . . . **ii**

Summary . . . . **iv**

Contents . . . . **vi**

List of Figures . . . . **x**

List of Tables . . . . **xii**

Acknowledgments . . . . **xvi**

1 Introduction **1**
- 1.1 Architectural Analysis and New Structure Design of RNNs 2
- 1.2 Open Problems in Machine Reading Comprehension and Dialogue System 4

2 Background and Related Work **7**
- 2.1 From Artificial Neural Networks to Deep Learning 7
- 2.2 Recurrent Neural Networks 10
  - 2.2.1 Vanilla Recurrent Neural Network 12
  - 2.2.2 Backpropagation Through Time (BPTT) 12
  - 2.2.3 Gradient Vanishing/Exploding Problems 13
  - 2.2.4 Gates and Memory Cells 13
  - 2.2.5 Sequence-to-sequence Models 16
  - 2.2.6 Attention Mechanism 17
  - 2.2.7 External Memory 18
  - 2.2.8 Making RNNs deeper 21
- 2.3 Learning Neural Natural Language Representations 21
  - 2.3.1 Neural Language Model 23
  - 2.3.2 Word Embedding and Beyond 25
  - 2.3.3 Neural Reading Comprehension 27

    2.3.4 Neural Dialogue System 32

3 Prologue to First Article 36
  3.1 Article Detail 36
  3.2 Context 36
  3.3 Contributions 37

4 Architectural Complexity Measures of Recurrent Neural Networks 38
  4.1 Introduction 38
  4.2 General RNN 39
    4.2.1 The Connecting Architecture 39
    4.2.2 A General Definition of RNN 42
  4.3 Measures of Architectural Complexity 42
    4.3.1 Recurrent Depth 42
    4.3.2 Feedforward Depth 44
    4.3.3 Recurrent Skip Coefficient 46
  4.4 Experiments and Results 47
    4.4.1 Tasks and Training Settings 47
    4.4.2 Recurrent Depth is Non-trivial 48
    4.4.3 Comparing Depths 50
    4.4.4 Recurrent Skip Coefficients 51
    4.4.5 Recurrent Skip Coefficients vs. Skip Connections 53
  4.5 Conclusion 54
  4.6 Proofs 55

5 Prologue to Second Article 61
  5.1 Article Detail 61
  5.2 Context 61
  5.3 Contributions 62

6 On Multiplicative Integration with Recurrent Neural Networks 63
  6.1 Introduction 63
  6.2 Structure Description and Analysis 65
    6.2.1 General Formulation of Multiplicative Integration 65
    6.2.2 Gradient Properties 65
  6.3 Experiments 66
    6.3.1 Exploratory Experiments 66
    6.3.2 Character Level Language Modeling 70
    6.3.3 Speech Recognition 71
    6.3.4 Learning Skip-Thought Vectors 72
    6.3.5 Teaching Machines to Read and Comprehend 74
  6.4 Relationship to Previous Models 75

6.4.1 Relationship to Hidden Markov Models 75
6.4.2 Relations to Second Order RNNs and Multiplicative RNNs . 76
6.4.3 General Multiplicative Integration 77
6.5 Conclusion 77

7 Prologue to Third Article 79
7.1 Article Detail 79
7.2 Context 79
7.3 Contributions 80

8 HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering 81
8.1 Introduction 81
8.2 Data Collection 83
8.2.1 Pipeline 83
8.2.2 Implementation Details 86
8.3 Processing and Benchmark Settings 87
8.4 Dataset Analysis 90
8.5 Experiments 94
8.5.1 Model Architecture and Training 94
8.5.2 Results 95
8.5.3 Establishing Human Performance 99
8.6 Related Work 100
8.7 Conclusions 101

9 Prologue to Fourth Article 102
9.1 Article Detail 102
9.2 Context 102
9.3 Contributions 103

10 Personalizing Dialogue Agents: I have a dog, do you have pets too? 104
10.1 Introduction 104
10.2 Related Work 106
10.3 The PERSONA-CHAT Dataset 107
10.3.1 Personas 109
10.3.2 Revised Personas 109
10.3.3 Persona Chat 110
10.3.4 Evaluation 110
10.4 Models 112
10.4.1 Baseline ranking models 112
10.4.2 Ranking Profile Memory Network 112

10.4.3 Key-Value Profile Memory Network 113
10.4.4 Seq2Seq 114
10.4.5 Generative Profile Memory Network 114
10.5 Experiments 115
10.5.1 Automated metrics 115
10.5.2 Human Evaluation 117
10.5.3 Profile Prediction 119
10.6 Conclusion & Discussion 121
10.7 Dialogues between Humans and Models 122

11 Conclusion 127

Bibliography 128

List of Figures

4.1 (a) An example of an RNN's $\mathcal{G}_c$ and $\mathcal{G}_{\text{un}}$. V_{in} is denoted by square, V_{hid} is denoted by circle and V_{out} is denoted by diamond. In $\mathcal{G}_c$, the number on each edge is its corresponding σ. The longest path is colored in red. The longest input-output path is colored in yellow and the shortest path is colored blue. The value of three measures are $d_r = \frac{3}{2}$, $d_f = 3.5$ and $s = 2$. (b) 5 more examples. (1) and (2) have $d_r = 2, \frac{3}{2}$, (3) has $d_f = 5$, (4) and (5) has $s = 2, \frac{3}{2}$. 44

4.2 (a) The architectures for *sh*, *st*, *bu* and *td*, with their (d_r, d_f) equal to $(1, 2)$, $(1, 3)$, $(1, 3)$ and $(2, 3)$, respectively. The longest path in *td* are colored in red. (b) The 9 architectures denoted by their (d_f, d_r) with $d_r = 1, 2, 3$ and $d_f = 2, 3, 4$. We only plot the hidden states within 1 time step (which also have a period of 1) in both (a) and (b). 49

4.3 (a) Various architectures that we consider in Section 4.4.4. From top to bottom are baseline $s = 1$, and $s = 2$, $s = 3$. (b) Proposed architectures that we consider in Section 4.4.5 where we take $k = 3$ as an example. The shortest paths in (a) and (b) that correspond to the recurrent skip coefficients are colored in blue. 52

6.1 (a) Curves of log-L2-norm of gradients for lin-RNN (blue) and lin-MI-RNN (orange). Time gradually changes from $\{1,\ 5,\ 10\}$. (b) Validation BPC curves for vanilla-RNN, MI-RNN-simple using Eq. 6.2, and MI-RNN-general using Eq. 6.4. (c) Histogram of vanilla-RNN's hidden activations over the validation set, most activations are saturated. (d) Histogram of MI-RNN's hidden activations over the validation set, most activations are **not** saturated. 68

6.2 (a) MSE curves of uni-skip (ours) and MI-uni-skip (ours) on semantic relatedness task on SICK dataset. MI-uni-skip significantly outperforms baseline uni-skip. (b) Validation error curves on attentive reader models. There is a clear margin between models with and without MI. 74

8.1 An example of the multi-hop questions in HOTPOTQA. We also highlight the supporting facts in *blue italics*, which are also part of the dataset. 82

8.2 Screenshot of our worker interface on Amazon Mechanical Turk. . . 87

8.3 Types of questions covered in HotpotQA. Question types are extracted heuristically, starting at question words or prepositions preceding them. Empty colored blocks indicate suffixes that are too rare to show individually. See main text for more details. 91
8.4 Our model architecture. Strong supervision over supporting facts is used in a multi-task setting. 95

List of Tables

4.1 Test BPCs of *sh*, *st*, *bu*, *td* for *tanh* RNNs and LSTMs. 49
4.2 Test BPCs of *tanh* RNNs with recurrent depth $d_r = 1, 2, 3$ and feedforward depth $d_f = 2, 3, 4$ respectively. 50
4.3 Test accuracies with different s for *tanh* RNN and LSTM in MNIST/pMNIST. 52
4.4 our best model compared to previous results on MNIST/pMNIST. . . . 53
4.5 Test accuracies for architectures (1), (2), (3) and (4) for *tanh* RNN on MNIST/pMNIST. 54

6.1 Test BPCs and the standard deviation of models with different scales of weight initializations. 69
6.2 Top: test BPCs on character level Penn-Treebank dataset. Middle: test BPCs on character level text8 dataset. Bottom: test BPCs on character level Hutter Prize Wikipedia dataset. 70
6.3 Test CERs and WERs on WSJ corpus. 71
6.4 Skip-thought+MI on Semantic-Relatedness task (top), Paraphrase Detection task (middle) and four different classification tasks (bottom). . . 73
6.5 Multiplicative Integration (with batch normalization) on Teaching Machines to Read and Comprehend task. 75

8.1 Data split. The splits *train-easy*, *train-medium*, and *train-hard* are combined for training. The distractor and full wiki settings use different test sets so that the gold paragraphs in the full wiki test set remain unknown to any models. 88
8.2 Retrieval performance comparison on full wiki setting for *train-medium*, *dev* and *test* with 1,000 random samples each. MAP and are in %. Mean Rank averages over retrieval ranks of two gold paragraphs. CorAns Rank refers to the rank of the gold paragraph containing the answer. 90
8.3 Types of answers in HotpotQA. 92

8.4 Types of multi-hop reasoning required to answer questions in the HotpotQA dev and test sets. We show in ***orange bold italics*** bridge entities if applicable, *blue italics* supporting facts from the paragraphs that connect directly to the question, and **green bold** the answer in the paragraph or following the question. The remaining 8% are single-hop (6%) or unanswerable questions (2%) by our judgement. 93
8.5 Retrieval performance in the full wiki setting. Mean Rank is averaged over the ranks of two gold paragraphs. 97
8.6 Main results: the performance of question answering and supporting fact prediction in the two benchmark settings. We encourage researchers to report these metrics when evaluating their methods. . 97
8.7 Performance breakdown over different question types on the dev set in the distractor setting. "Br" denotes questions collected using bridge entities, and "Cp" denotes comparison questions. 98
8.8 Ablation study of question answering performance on the dev set in the distractor setting. "– sup fact" means removing strong supervision over supporting facts from our model. "– train-easy" and "– train-medium" means discarding the according data splits from training. "gold only" and "sup fact only" refer to using the gold paragraphs or the supporting facts as the only context input to the model. 99
8.9 Comparing baseline model performance with human performance on 1,000 random samples. "Human UB" stands for the upper bound on annotator performance on HotpotQA. For details please refer to the main body. 100

10.1 Example Personas (left) and their revised versions (right) from the persona-chat dataset. The revised versions are designed to be characteristics that the same persona might have, which could be rephrases, generalizations or specializations. 108
10.2 Example dialog from the persona-chat dataset. Person 1 is given their own persona (top left) at the beginning of the chat, but does not know the persona of Person 2, and vice-versa. They have to get to know each other during the conversation. 111
10.3 **Evaluation of dialog utterance prediction with various models** in three settings: without conditioning on a persona, conditioned on the speakers given persona ("Original Persona"), or a revised persona that does not have word overlap. 116

10.4 **Evaluation of dialog utterance prediction with generative models** in four settings: conditioned on the speakers persona (“self persona”), the dialogue partner’s persona (“their persona”), both or none. The personas are either the original source given to Turkers to condition the dialogue, or the revised personas that do not have word overlap. In the “no persona” setting, the models are equivalent, so we only report once. 116
10.5 **Evaluation of dialog utterance prediction with ranking models** using hits@1 in four settings: conditioned on the speakers persona (”self persona”), the dialogue partner’s persona (”their persona”), both or none. The personas are either the original source given to Turkers to condition the dialogue, or the rewritten personas that do not have word overlap, explaining the poor performance of IR in that case. 117
10.6 **Human Evaluation** of various PERSONA-CHAT models, along with a comparison to human performance, and Twitter and OpenSubtitles based models (last 4 rows), standard deviation in parenthesis. 118
10.7 **Profile Prediction.** Error rates are given for predicting either the persona of speaker 0 (Profile 0) or of speaker 1 (Profile 1) given the dialogue utterances of speaker 0 (PERSON 0) or speaker 1 (PERSON 1). This is shown for dialogues between humans (PERSON 0) and either the KV Profile Memory model (“KV Profile”) which conditions on its own profile, or the KV Memory model (“KV w/o Profile”) which does not. 120
10.8 **Profile Prediction By Dialog Length.** Error rates are given for predicting either the persona of speaker 0 (Profile 0) or of speaker 1 (Profile 1) given the dialogue utterances of speaker 0 (PERSON 0) or speaker 1 (PERSON 1). This is shown for dialogues between humans (PERSON 0) and the KV Profile Memory model averaged over the first N dialogue utterances from 100 conversations (where N is the “Dialogue Length”). The results show the accuracy of predicting the persona improves in all cases as dialogue length increases. 120
10.9 Example dialog between a human (Person 1) and the OpenSubtitles KV Memory Network model (Person 2). 122
10.10 Example dialog between a human (Person 1) and the Seq2Seq model (Person 2). 123
10.11 Example dialog between a human (Person 1) and the Key-Value Profile Memory Network with Self Persona. 124
10.12 Example dialog between a human (Person 1) and the Generative Profile Memory Network with Self Persona. 125

10.13 Example dialog between a human (Person 1) and the Language Model trained on the OpenSubtitles 2018 dataset (does not use persona). 125
10.14 Example dialog between a human (Person 1) and the Language Model trained on the Twitter dataset (does not use persona). 126

Acknowledgments

When I was reading Prof. Yoshua Bengio's famous denoising autoencoder paper during my first undergraduate research project in 2011, I never thought that I will become his student in the following years. That paper strongly motivated me to find my research path towards neural networks (later well-known as deep learning) and finally drove me to pursue the Ph.D. under Yoshua's supervision. As a researcher, I am impressed by Yoshua's humility and curiosity to the unknown, although he is already a widely respected researcher in the field. As Yoshua's student, I am very grateful for his insightful advices on my research, and more importantly, the freedom and patience given to me that allow me to explore my own research interests. A famous Chinese proverb says, "Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime". I would like to thank Yoshua to be my guide during the past five years, from whom the most precious thing I learned is not the machine learning knowledge, but how to be enthusiastic and stay brave when facing challenges.

I would like to thank my friend and co-author Yuhuai Wu from University of Toronto, with whom I wrote my first two NIPS papers which became the first part of this thesis. The days and nights spent with him chatting about research and life will become one of my most memorable experiences. I also would like to thank my friend Jake Zhao from New York University, from who I learned how to become ambitious and steadfast towards the goal that I am pursuing.

As a member of the big Mila family, I would like to thank all the friends I met there with whom I had a wonderful and unforgettable time: Zhouhan Lin, Yikang Shen, Harm de Vries, Laurent Dinh, Pengfei Liu, Mohammad Pezeshki, Min Lin, Jie Fu, Kyunghyun Cho, Alexandre de Brébisson, Sandeep Subramanian, Alex Lamb, Caglar Gulcehre, Orhan Firat, Taesup Kim, Anirudh Goyal, Benjamin Scellier, Asja Fischer, Junyoung Chung, Yuchen Lu, Guillaume Alain, Dong-hyun Lee, Chen Xing, Dmitriy Serdyuk, Li Yao, Gerry Che, Kyle Kastner, Iulian Vlad Serban, Jae Hyun Lim, Thomas Mesnard, Dzmitry Bahdanau, Akram Erraqabi, Nicolas Ballas, Prof. Aaron Courville and Prof. Roland Memisevic.

I am also very thankful to all my friends who are my collaborators outside Mila and offered me plentiful help in my research: Hongyuan Mei from John Hopkins University, Zhilin Yang and Prof. Ruslan Salakhutdinov from Carnegie Mellon University, Peng Qi from Stanford University, Jason Weston, Douwe Kiela, Arthur Szlam and Abdel-rahman Mohamed from Facebook AI Research, Xiang Zhang and Sungjin Ahn from Element AI, Xingdi Yuan, Tong Wang and Adam Trischler from Microsoft Research, Vinod Nair from Deepmind.

Finally, I would like to thank my parents and my wife Ying Zhang for their selfless support to my career. It was their love that helped me overcome all the obstacles during my Ph.D. life.

1 Introduction

With the resurgence of deep learning (DL) during the past decade, the recurrent neural network (RNN), as one of the critical members of neural network models, gradually drew attention from the machine learning research community because of its strong modeling capacity for sequential learning problems. Since 2014, RNNs soon took the center stage in a series of natural language processing (NLP) scenarios, including machine translation (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2014), dialogue system (Vinyals and Le, 2015; Sordoni et al., 2015; Serban et al., 2016b) and machine reading comprehension (Hermann et al., 2015; Rajpurkar et al., 2016; Devlin et al., 2018). At the time, although the empirical success of the RNN was so significant that it reshaped the problem formulation and methodology in several subfields of NLP, relatively fewer research was conducted on the systematic understanding of RNNs' architectural basics and computational fundamentals. This becomes the motivation of the first part of this thesis (Chapter 3 to Chapter 6 (Zhang et al., 2016; Wu et al., 2016)), in which we attempt to offer new insights on understanding the RNN's architectural properties in a generic perspective as well as exploring new functional designs inside the RNN's internal computational procedure. On the other hand, as the recently introduced advanced recurrent neural modules redefined the pipeline of NLP research, we are now able to explore new natural language scenarios which were infeasible before. The second part of this thesis (Chapter 7 to Chapter 10 (Yang et al., 2018a; Zhang et al., 2018)) is motivated by this trend, in which we dive deep into two open problems about multi-hop reasoning in machine reading comprehension and personality encoding in dialogue systems. For both problems we collect new datasets and build corresponding baselines, hoping that our proposed testbed can promote the further research development in these two directions.

1.1 Architectural Analysis and New Structure Design of RNNs

The original RNN (Rumelhart et al., 1985; Jordan, 1997; Elman, 1990), also known as “vanilla RNN”, has relatively simple structure in which a hidden state $\mathbf{h}_t$ at the current time step t is computed based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ via a set of linear transformations $\{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$ and elementwise nonlinearity σ:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}). \tag{1.1}$$

The vanilla RNN is also a *single hidden layer* RNN. In the following decades, various new connecting architectures were proposed based upon the vanilla RNN: Schmidhuber (1992); El Hihi and Bengio (1996) introduced different forms of *stacking* in RNNs where hidden layers are stacked on top of each other; Schuster and Paliwal (1997) proposed the *bidirectional* RNNs which conduct recurrent computation in both forward and backward direction over the input sequence; Raiko et al. (2012); Graves (2013); Lin et al. (1996); El Hihi and Bengio (1996); Sutskever and Hinton (2010); Koutnik et al. (2014) explored the use of skip connections (shortcuts) among different hidden states and Hermans and Schrauwen (2013) proposed deep RNNs which are stacked RNNs with skip connections; Pascanu et al. (2013a) suggested adding more nonlinearities in RNNs’ transition functions to make RNNs “deeper” in the recurrent direction. Despite all the empirical success achieved so far, few of these research attempted to analyze RNN connecting architectures in a systematic way with generic and rigorous formulations. This leads to several drawbacks: (1) It is hard to quantitatively measure the complexity of different connecting architectures: even the concept of “depth” is not clear with current RNNs. (2) The lack of correct definition of connection architectures limits previous researchers to only consider the simple “deep” RNNs while leaving vastly field of connecting variations unexplored. In the first paper (Chapter 3 and Chapter 4), we attempt to rigorously analyze the connecting architectures of RNNs by introducing a generic graph-theoretic formulation under which we could further evaluate the architectural complexity of an RNN via three novel measures: *recurrent depth*, *feedforward depth*, and *recurrent skip coefficient*. These measures reflect the fact that RNNs may undergoes multiple transformations not only feedforwardly (from input to output within a time step) but also recurrently (across multiple time steps) in sophisticated ways. Compared with previous research, the recurrent depth can be viewed as a general extension of the work of Pascanu et al. (2013a), the feedforward depth can be considered as an accurate measure of the input-output nonlinearness for different stacked RNNs, and the recurrent skip coefficient plays the role of quantifying the complexity of various skip connections in RNNs. Notably, the recurrent skip coefficient is strongly related to vanishing/exploding gradient issues and helps dealing with long term dependency problems. Our experimental results on five different datasets validate the usefulness of the proposed definitions and measurements where they help answering the question of "What is the depth of an RNN?" under the generic scenario and are able to provide guidance for the design and inspection of new connecting architectures for particular learning tasks.

Besides the connecting architecture, another crucial structural aspect of the RNN is its transition function which describes the computational procedure associated with each unit in the network. As we discussed before, the vanilla RNN adopts a very simple formulation in its transition function. Hochreiter and Schmidhuber (1997) invented the long short term memory (LSTM) in which they introduced the *gating mechanism* and *memory cells*, while Cho et al. (2014) further simplified the gating structures and proposed gated recurrent unit (GRU). Moreover, there is a recent resurgence of new structural designs for RNNs' transition functions (Chung et al., 2015; Kalchbrenner et al., 2015; Jozefowicz et al., 2015), most of which are derived from vanilla RNNs, LSTMs or GRUs. Despite of their structural differences, they share a common computational building block either in their hidden-to-hidden or gating computations, described by the following equation:

$$\phi(\mathbf{W}\mathbf{x} + \mathbf{U}\mathbf{z} + \mathbf{b}). \tag{1.2}$$

This computational building block serves as a combinator for integrating information flows from different sources $\mathbf{x}$ and $\mathbf{z}$ by an additive operation "+". Such additive operation are widely adopted in various state computations in RNNs (LSTMs and GRUs) including hidden state and gate/cell computations. In our second paper (Chapter 5 to Chapter 6), we consider an alternative in which the additive integration is replaced by a multiplicative one. Specifically, we propose to use the

Hadamard product “$\odot$” to fuse information from $\mathbf{x}$ and $\mathbf{z}$:

$$\phi(\mathbf{W}\boldsymbol{x} \odot \mathbf{U}\boldsymbol{z} + \mathbf{b}). \tag{1.3}$$

We name the above information integration design as *Multiplicative Integration* (MI). MI naturally derives the gating effect between $\mathbf{Wx}$ and $\mathbf{Uz}$ where they dynamically rescale each other. We empirically show that RNNs with MI enjoy better gradient properties due to such gating effect and most of the hidden units are non-saturated. From an engineering perspective, MI is one of the few improvements that directly touches the essential transition function of an RNN with (1) adaptability towards any recurrent models (e.g. LSTMs and GRUs), (2) no extra computational cost as it brings almost no extra parameters and (3) no extra engineering beyond implementing the RNN model itself. In Chapter 6 we will see that the general form of MI is by design performing at least as well as the standard RNN transition function. Our experimental results demonstrate that MI can provide a substantial performance boost over many of the existing RNN models.

1.2 Open Problems in Machine Reading Comprehension and Dialogue System

One of the extraordinary intelligent skills of human is the ability of reasoning and inference over abstract symbols, especially natural languages. Such ability is also considered as a crucial step towards artificial general intelligence (AGI). Machine reading comprehension (MRC) and question answering (QA) tasks provide measurable ways to verify the reasoning ability of intelligent machines, in which correctly answering the question requires the machine to perform sophisticated understanding and reasoning process over the given related natural language materials. Recently, the emergence of many large-scale QA datasets has sparked much progress in this direction. However, existing datasets have several limitations that hinder further advancements: (1) The most popular dataset SQuAD (Rajpurkar et al., 2016) only concentrates on testing single-step (or *single-hop*) reasoning ability where most of the questions can be addressed by matching the question with a single sentence in a single context paragraph. Machines trained on this dataset end up with some keyword-matching like mechanisms and can hardly reason over a larger context. TriviaQA (Joshi et al., 2017b) and SearchQA (Dunn et al., 2017) attempted to make the setting more challenging by collecting multiple documents as the context whilst most of the questions can still be answered by matching the question with a few nearby sentences in one single paragraph. (2) Datasets designed for multi-hop reasoning like WikiHop (Welbl et al., 2018a) and ComplexWebQ (Talmor and Berant, 2018) are constructed on existing knowledge bases (KBs), which implies that they are constrained by the schema of the KBs used and therefore have very limited question-answer diversity. (3) Given a question, all the existing datasets provide the answer as the only distant supervision while the machine has no idea what supporting facts lead to that answer. This makes it difficult for explaining the machine's prediction and tracing the underlying reasoning process. The HotpotQA[1] dataset introduced in the third paper (Chapter 7 to Chapter 8) tries to address the above challenges by forcing the question staying in natural language form, requiring reasoning over multiple facts in multiple documents (we name it as *multi-hop reasoning*), and not being constrained by existing KBs. It also explicitly highlights in-document supporting sentences for each question, which denote where the answer is actually derived, and thus help guiding the model to perform meaningful and explainable reasoning. Specifically, we collected the data via Amazon crowdsourcing service, where qualified workers are shown multiple context documents extracted from Wikipedia and asked explicitly to raise questions requiring reasoning over these documents. The entire data collection pipeline is carefully designed and we hope that this pipeline as a byproduct can also sheds light on future work in this direction. During the experiments, we show that the multi-hop reasoning questions in HotpotQA is challenging for the latest MRC systems and the supporting facts enable models to improve performance and make explainable predictions.

Question answering is in fact a single turn communication between the questioner and the respondent. A more generic scenario is a conversation (dialogue) in which interlocutors conduct multiple turns of communication which require more complicated context understanding over the entire dialogue history. Despite the past success in NLP research, the communication between a human and a machine

---

[1] The name comes from the first three authors' arriving at the main idea during a discussion at a hot pot restaurant.

is still in its infancy especially for the general chit-chat setting. Although the recently introduced neural based approaches have had sufficient capacity to generate meaningful responses with accessing sufficiently large datasets, these models' weaknesses are obvious after even a very short conversation with them (Serban et al., 2016a; Vinyals and Le, 2015). Chit-chat models are known to have several issues include: (1) Inconsistent personality problem where they are typically trained over dialogues each with different speakers; (2) Explicit long-term memorizing ability is absent as they are typically trained to respond given only the recent dialogue history (Vinyals and Le, 2015); (3) They often produce not very captivating "safe" answers which are non-specific, like "I don't know" (Li et al., 2015). As a result, these models produce an unsatisfying overall experience to human interlocutors. Despite the fact that a large amount of human dialogues concentrate on socialization, personal interests and chit-chat (Dunbar et al., 1997), the low quality of conversations makes the chit-chat setting often being ignored as an end-application comparing with task-oriented scenarios. We believe the above problems are partially due to there being no good publicly available dataset for general chit-chat model learning. In the fourth paper (Chapter 9 to Chapter 10), we make a step towards more engaging chit-chat dialogue agents by endowing them with a configurable but persistent persona (profile) which are multiple sentences of textual description. Comparing with persona-free settings, models with augmented memory can explicitly store these personas and use them to produce more personal, specific, consistent and engaging responses. We present the PERSONA-CHAT dataset to facilitate the training of such models. PERSONA-CHAT collects text based dialogues between different crowdworkers who were randomly paired, each asked to both mimic a given persona (randomly assigned, and created by another set of crowdworkers) and try to know each other during the conversation. Models trained on such utterances can leverage the personas from both speakers to perform better next utterance prediction. Our experiments show that, comparing with existing resources, PERSONA-CHAT enables models to express better consistency and more engagingness during a conversation. We hope that PERSONA-CHAT will aid training agents that can ask questions about users' profiles, remember the answers, and use them naturally in conversation.

2 Background and Related Work

2.1 From Artificial Neural Networks to Deep Learning

Modern Artificial Neural Networks (ANNs) are a huge class of parameterized function approximators which capture the underlying relations among inputs (and outputs). Originated from the Perceptron in 1950s (Rosenblatt, 1958), ANNs have experienced rapid evolution in the past few decades both in theory and in practice. Some remarkable events include Minsky's criticism towards the Perceptron (Minsky and Papert, 1969), the introduction of the backpropagation algorithm for training multi-layer ANNs (Werbos, 1974; Rumelhart et al., 1985) and the recent methodological renaissance of ANNs called "Deep Learning" (Bengio, 2009; LeCun et al., 2015).

Generally speaking, ANNs can have various computational dependencies describing different functional behaviors and internal logic. In supervised learning, an ANN can be defined by some parameterized function $F : \mathcal{X} \to \mathcal{Y}$ that explicitly models the dependencies between input $\mathbf{x} \in \mathcal{X}$ and output $\mathbf{y} \in \mathcal{Y}$:

$$\mathbf{y} = F(\mathbf{x}; \theta). \tag{2.1}$$

Here θ is a set of learnable parameters. In most ANNs, F adopts "linear transformation + elementwise nonlinearity" as the basic computational building block which we denote as f. In the simplest case (perceptron), the original F consists of only one such building block f and Eq. 2.1 becomes

$$\mathbf{y} = F(\mathbf{x}; \theta) = f(\mathbf{x}; \theta_f) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{2.2}$$

where matrix $\mathbf{W}$ describes the linear transformation, σ is the elementwise nonlinearity, $\theta_f = \{\mathbf{W}, \mathbf{b}\}$. Eq. 2.2 is equivalent to the scalar version that

$$y_i = \sigma(\sum_j w_{i,j} x_j + b_i), \tag{2.3}$$

where y_i and x_j are the i th and j th dimension of $\mathbf{y}$ and $\mathbf{x}$ respectively, $w_{i,j}$ is the (i, j) th element of $\mathbf{W}$. $w_{i,j} x_j$ and σ are analogical to the "connection (synapse)" and "neural activation" in biological neural networks, and Eq. 2.3 describes the entire connecting behaviors of "neuron" y_i to other "neurons" x_j.

An important property of f is that, under mild conditions, given enough number of different $\{f_i\}$, a linear combination of $\{f_i\}$ can approximate arbitrary complex functional dependencies with arbitrary precision, and this claim is also known as the "universal approximation theorem"(Hornik et al., 1989; Cybenko, 1989).

ANNs have rich topological architectures. Given the above f as the basic computational building block, the "topology" roughly includes two aspects: (1) How the relations between output y_j and input x_i are organized inside f, which we refer to as " intra-f topology". (2) How different fs (or "layers") connect to each other, which we refer to as "inter-f topology".

The simplest intra-f topology is "fully-connected" which is the matrix multiplication described in Eq. 2.2, where all dimensions of $\mathbf{y}$ depend on all dimensions of $\mathbf{x}$. A more interesting intra-f topology is "convolutional" (Fukushima, 1980; LeCun et al., 1998; Krizhevsky et al., 2012), in which $\mathbf{x}$ and $\mathbf{y}$ are reorganized so that they contain 2D (or higher dimensional) information and the linear transformation in f becomes the convolution operation ($\mathbf{W}$ then becomes convolutional kernels). The convolution operation forces y_i to only depend on a local input subset $\{x_{m,n}\}$ and such dependency is shared over different 2D (or higher dimensional) locations in $\mathbf{x}$-$\mathbf{y}$ space. Besides, the convolution operation is often combined with some "pooling" operation to further extract and summarize the local information. The resulting module is called convolutional neural network (CNN). CNNs are quite successful in natural image understanding problems as natural images have tons of repeated low-level patterns over the 2D space.

The simplest inter-f topology is "multi-layer feed-forward", in which the function approximator F is defined by stacking different building block f on top of each other:

$$\mathbf{y} = F(\mathbf{x}; \theta) = f_m \circ \cdots \circ f_1(x). \tag{2.4}$$

Here $\theta = \{\mathbf{W}_i, \mathbf{b}_i\}_{i=1}^m$, the intermediate results $\{\mathbf{h}_i | \mathbf{h}_i = f_i \circ \cdots \circ f_1(x),\ i = 1, \cdots, m\}$ are called "hidden layers" and m is the number of layers or the "depth" of the network. Obviously, the word "feed-forward" simply illustrates the unidirectional way of connecting different f_i which is from bottom (input) to top (output). An important property of multi-layer feed-forward neural networks is about the feature hierarchy (Bengio, 2009; Bengio et al., 2013): Different layers extract different levels of features from the inputs, the higher the layer, the more abstract/general the corresponding representations is. This property is widely examined in CNNs for image understanding problems where lower layers often extract low-level visual patterns such as edges and corners, and activations in higher layers often reflect high-level visual concepts such as different parts of an object (Zeiler and Fergus, 2014). Furthermore, it has been theoretically proved that the multi-layer network's expressive power significantly benefits from increasing the network depth (Montufar et al., 2014; Telgarsky). Another important inter-f topology is "recurrent" (Rumelhart et al., 1985; Jordan, 1997; Elman, 1990). Unlike the feed-forward case, the recurrent neural network (RNN) has an extra recurrent direction on which the same functional dependency of f is repeated iteratively. This recurrent direction is also denoted as the "time" direction because RNN is able to exhibit dynamic temporal behaviors and is usually used for modeling sequential data.

The computational and topological richness of ANNs may be a double-edged sword, because successfully learning/optimizing such complicated nonlinear models can be very costly and sometimes impossible. Unlike classic statistical leanring methods such as Support Vector Machines (SVMs) (Cortes and Vapnik, 1995; Vapnik) whose optimization procedure is convex with unique global optima, the loss surface of an ANN is highly non-convex and often has a lot of bad local extrema (e.g. minima or saddle points (Dauphin et al., 2014; Choromanska et al., 2015)). As a result, almost all the feasible optimization techniques for training ANNs are gradient-based, and there is no guarantee for these greedy techniques to find the global optima. Furthermore, in early days, the lack of enough training data makes ANNs easily overfit whilst the absence of high-performance computational resources prevents researchers from exploring and exploiting larger models. As a result, the potential capacity of "large" and "deep" ANNs on large scale problems are underestimated for decades before the "Deep Learning" resurgence.

The "Deep Learning" trend originated from the invention of "pretraining" techniques which significantly alleviated the difficulty of training/optimizing deep neural networks (Hinton and Salakhutdinov, 2006; Bengio et al., 2007). Since then, a huge amount of theoretical and empirical work drew the community's attention back to the buried treasure of deep neural networks. Specifically, those works include but are not limited to (1) several landmark models in image classification/detection like very deep CNNs (Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2015; He et al., 2015) and R-CNNs (Girshick et al., 2014); (2) hybird and end-to-end speech recognition systems (Graves et al., 2013; Hinton et al., 2012); (3) various neural-based approaches for natural language processing (NLP) problems such as word embedding and neural machine translation (Mikolov et al., 2013a; Sutskever et al., 2014; Bahdanau et al., 2014); (4) deep reinforcement learning (Mnih et al., 2015; Silver et al., 2016); (5) deep generative models such as variational autoencoders and generative adversarial networks (Kingma and Welling, 2013; Rezende et al., 2014; Goodfellow et al., 2014); (6) optimization/-training techniques such as dropout (Srivastava et al., 2014), batch normalization (Ioffe and Szegedy, 2015), ADAM (Kingma and Ba, 2014), etc. In sum, "Deep Learning" is not a brand new idea since its foundation is still based on the classic neural networks, but it is now pushing the neural network research a big step forward with not only deeper models but also deeper ideas.

2.2 Recurrent Neural Networks

As briefly discussed in the previous section, recurrent neural networks (RNNs) are a class of ANNs in which the same functional dependency is repeated iteratively on its recurrent direction, resulting in an inner representation (inner state) computation at each recurrent (time) step. More formally, given the input sequence $\{\mathbf{x}_t\}$, the inner representation $\mathbf{h}_t$ at recurrent step t is computed using some function f that

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta). \tag{2.5}$$

Eq. 2.5 implies several important properties about the RNN: (1) The RNN is able to model the temporal order of elements in the input sequence, because if the order of $\mathbf{x}_t$ in the input sequence is changed, the inner representation $\mathbf{h}_t$ is also changed correspondingly. (2) The RNN can handle input sequences with variable lengths as $f(\cdot)$ can repeat for arbitrary number of times. (3) At every recurrent step, the same functional dependency f is conducted implying that RNN can model the latent temporal structure of the input in a homogeneous and consistent way across time. (4) In the ideal case, $\mathbf{h}_t$ depends on all the past inputs $\{\mathbf{x}_k\}_{k=-\infty}^{t-1}$ and can memorize the information back to recurrent step $-\infty$.

RNN is "Turing-Complete" if f and θ are chosen properly in the sense that in theory it can simulate arbitrary complex computational programs (Siegelmann, 1999). This is analogical to the "universal approximation theorem" in the feed-forward network.

There is an interesting topological similarity between the RNN and the feed-forward network: if we start from $\mathbf{h}_0$, unroll Eq. 2.5 for m steps and consider $\mathbf{h}_t$ as the main variable while taking $\mathbf{x}_t$ as the parameters of f_k at recurrent step k, Eq. 2.5 then becomes

$$\mathbf{h}_t = f_m^* \circ \cdots \circ f_1^*(\mathbf{h}_0) \tag{2.6}$$

where $f_k^*(\cdot) = f(\;\cdot\;; \mathbf{x}_k, \theta)$. Eq. 2.6 implies that an RNN unrolled for m steps has exactly the same formulation as a m-layers feed-forward network described in Eq. 2.4. From this point of view, an RNN can be considered as a deep feed-forward network with (1) variable depth, (2) same functional dependency repeated at each layer and (3) extra input plugged-in at each layer.

There are at least two fundamental aspects related to an RNN's practical performance: (1) The ability of capturing nonlinear temporal dependencies of the input sequences. Actually, the complexity of temporal dependencies of an input sequence can be highly varied: the traveled distance of an object moving in constant speed linearly depends on the traveled time, while the price change in a stock market has complicated relations to its past history. In any case, the $f(\cdot;\theta)$ in an RNN is expected to be flexible enough so that there exists some parameter configuration θ^* that $f(\cdot;\theta^*)$ is "close enough" to the true underlying dependencies. (2) The ability of memorizing and processing information in various temporal scales. Dependencies inside the input sequence can be either short term or long term. For example, given a paragraph in char-level as the input sequence, characters within one word have very short term intra-word dependencies on each other (dependency length less than the length of the word) while characters in different words may have inter-word long term dependencies due to the word-level or phrase-level relations (dependency length is roughly the average length of words × length of sentences). An ideal f should have the property that even all the inputs $\{\mathbf{x}_t\}$ are mixed up implicitly inside the recurrent computation after many time steps, the network can still extract useful information from any past recurrent step. Unfortunately, current RNN models all suffer from the problem of learning long term dependencies known as "gradient vanishing/exploding", while there are still structural solutions to alleviate such forgetting effect.

2.2.1 Vanilla Recurrent Neural Network

The vanilla RNN is the most simple RNN structure. In the standard formulation of the vanilla RNN, Eq. 2.5 becomes the "linear transformation + elementwise nonlinearity" functional dependency:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{2.7}$$

where $\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$, $\mathbf{W}$ is the input-to-hidden matrix, $\mathbf{U}$ is the hidden-to-hidden matrix and $\mathbf{b}$ is the bias vector. Compared with the fully-connected feed-forward network in Eq. 2.2, the only difference in Eq. 2.7 is the additional matrix $\mathbf{U}$ which bridges the current inner state $\mathbf{h}_t$ to its previous state $\mathbf{h}_{t-1}$.

2.2.2 Backpropagation Through Time (BPTT)

Similar to the feed-forward network, the RNN can be optimized by gradient-based backpropagation algorithm which is called backpropagation through time (BPTT) (Rumelhart et al., 1985; Werbos, 1990). As its name implies, BPTT propagates the gradient signals through the recurrent (time) direction. Here we take the vanilla RNN as an example. In vanilla RNN, given the total loss $\Delta = \sum_1^T \delta_t$ where δ_t is the partial loss at recurrent step t and the input sequence $\{\mathbf{x}_k\}_{k=1}^T$, the full gradient $\{\frac{\partial \Delta}{\partial \mathbf{M}}\}_{\mathbf{M}=\mathbf{W},\mathbf{U},\mathbf{b}}$ with respect to any parameter $\mathbf{M}$ is calculated by

$$\frac{\partial \Delta}{\partial \mathbf{M}} = \sum_{1 \le t \le T} \frac{\partial \delta_t}{\partial \mathbf{M}}, \tag{2.8}$$

where each $\frac{\partial \delta_t}{\partial \mathbf{M}}$ involves the following Jacobian matrix that forms the gradient propagation path from the current recurrent state $\mathbf{h}_t$ to the previous recurrent state $\mathbf{h}_{t-n}$ for $1 \leq n \leq t-1$:

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-n}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \cdots \frac{\partial \mathbf{h}_{k=t-n+1}}{\partial \mathbf{h}_{t-n}} = \prod_{k=t-n+1}^{t} \mathbf{U}^T diag(\sigma'_k), \tag{2.9}$$

where $\sigma'_k = \sigma'(\mathbf{W}\mathbf{x_k} + \mathbf{U}\mathbf{h}_{k-1} + \mathbf{b})$ and $diag(\sigma'_k)$ is a diagonal matrix with the diagonal elements being elements in σ'_k. From Eq. 2.9 we can find that the gradients are propagated iteratively in the recurrent direction in the order of $\mathbf{h}_{t-1}$, $\mathbf{h}_{t-2}$, $\cdots$, $\mathbf{h}_{t-n}$.

2.2.3 Gradient Vanishing/Exploding Problems

The gradient vanishing/exploding problem is mainly about the extreme behaviors during the RNN training where gradient norms may exponentially decrease to zero or increase to infinity and prevent the RNN from further learning. (Hochreiter, 1991; Bengio et al., 1994; Pascanu et al., 2013b). Those behaviors can be clearly illustrated in Eq. 2.9 in which the gradient $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-n}}$ is computed by multiplying n matrices $\{\mathbf{U}^T diag(\sigma'_k)\}_{k=t-n+1}^{n}$ together. Assume that $|\sigma'_k|$ is bounded by some constant α, then the spectral norm $n_\sigma = ||diag(\sigma'_k)||$ satisfies $n_\sigma \leq \alpha$. Now if the spectral norm $n_\mathbf{U} = ||\mathbf{U}||$ satisfies $n_\mathbf{U} < \frac{1}{\alpha}$, we have

$$||\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-n}}|| = \prod_{k=t-n+1}^{t} ||\mathbf{U}^T|| \cdot ||diag(\sigma'_k)|| \leq (n_\alpha n_\mathbf{U})^n, \tag{2.10}$$

where $n_\alpha n_\mathbf{U} < 1$. Obviously, if n is large enough, the right side of the inequality will shrink to zero which implies that $||\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-n}}||$ also shrinks to zero. The above analysis reveals the gradient vanishing case while similar conclusion can be made for the exploding case also. In other words, although $\mathbf{h}_t$ functionally depends on $\mathbf{h}_{t-n}$ for any n, such dependency becomes untraceable when n is large.

2.2.4 Gates and Memory Cells

Although gradient vanishing/exploding is an inherent problem of RNN models, there exists several structural alternatives suffering much less from such negative impact compared with the vanilla RNN ((Hochreiter and Schmidhuber, 1997; Cho et al., 2014)). The main idea of these alternatives is to introduce (1) *gate* and (2) a naive memory structure called *memory cell* inside RNN's basic computational building block f.

A *gate* is a function $\mathbf{g}$ that consists of a linear transformation and an elementwise *sigmoid* function $\frac{1}{1+e^{-\mathbf{x}}}$ with output range in $[0, 1]$. A gate $\mathbf{g}$ is usually multiplied elementwise with another activation $\mathbf{f}$ to get $\mathbf{g} \odot \mathbf{f}$, so that it can rescale $\mathbf{f}$'s different dimensions $\{f_k\}$ by corresponding ratios $\{g_k\}$, where $g_k = 1$ means fully flowing in of f_k and $g_k = 0$ means fully blocking out of f_k. According to its functionality, the gate can dynamically control the information that flows through recurrent states and thus can further adjust the gradient propagation.

A *memory cell* is often made up of some extra states that can encode the past information and keep the encoded information unchanged for a controllable length of time. A simple example is the memory cell in long short term memory (LSTM).

LSTM is a variant of RNN which contains gates and memory cells (Hochreiter and Schmidhuber, 1997). The motivation behind LSTM is to use memory cells to store information through time and to use gates to control the information flows. An LSTM maintains three gates (an input gate $\mathbf{g}^i$, a forget gate $\mathbf{g}^f$ and an output gate $\mathbf{g}^o$), a memory cell $\mathbf{c}$, an input context vector $\mathbf{z}$ and a hidden state $\mathbf{h}$. Given an input sequence $\{\mathbf{x}_t\}$, the computational dependencies of LSTM at recurrent step t are defined as follows:

$$\mathbf{g}^i_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i), \tag{2.11}$$

$$\mathbf{g}^f_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f), \tag{2.12}$$

$$\mathbf{g}^o_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o), \tag{2.13}$$

$$\mathbf{z}_t = \phi(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}), \tag{2.14}$$

$$\mathbf{c}_t = \mathbf{g}^i_t \odot \mathbf{z}_t + \mathbf{g}^f_t \odot \mathbf{c}_{t-1}, \tag{2.15}$$

$$\mathbf{h}_t = \mathbf{g}^o_t \odot \varphi(\mathbf{c}_t). \tag{2.16}$$

Here $\phi(\cdot)$ and $\varphi(\cdot)$ are different elementwise nonlinearities. $\mathbf{z}_t = \phi(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b})$ is similar to the activation in the vanilla RNN while in LSTM it serves as the encoded input information for $\mathbf{c}_t$ to memorize. In Eq. 2.15, $\mathbf{g}^i_t$ controls the ratio of how much the current input information should be memorized by $\mathbf{c}_t$ and $\mathbf{g}^f_t$ controls the ratio of how much previous information should be forgotten. Specifically, if $\mathbf{g}^f_t = \mathbf{1}$, $\mathbf{c}_t$ will memorize all the information from its previous state $\mathbf{c}_{t-1}$, if $\mathbf{g}^f_t = \mathbf{0}$,

$\mathbf{c}_t$ will forget all the information from $\mathbf{c}_{t-1}$. We have the similar claim for $\mathbf{g}_t^i$ and the encoded input information. In fact, Eq. 2.15 implicitly illustrates LSTM's ability of alleviating gradient vanishing/exploding problem: Instead of directly computing the next hidden state by "linear transformation + elementwise nonlinearity" as in Eq .2.7, the encoded input context vector $\mathbf{z}_t$ is additively combined into the memory cell $\mathbf{c}_t$ and we have

$$\mathbf{c}_t = \sum_{k=-\infty}^{t} (\prod_{m=k+1}^{t} \mathbf{g}_m^f) \odot \mathbf{g}_k^i \odot \mathbf{z}_k, \tag{2.17}$$

which shows that encoded input information $\mathbf{z}_k$ at every recurrent step k has a non-ignorable additive contribution to the current $\mathbf{c}_t$. Moreover, if we look at the gradient flow $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-n}}$, a major part of it is (the full expression of $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-n}}$ is much more complicated)

$$\prod_{k=t-n+1}^{t} diag(\mathbf{g}_k^f). \tag{2.18}$$

The above term suffers much less from the vanishing problem if the forget gate $\mathbf{g}_i^f$ is close to $\mathbf{1}$. In practice, $\mathbf{g}_i^f$'s bias $\mathbf{b}_f$ is often initialized as some positive vector such as $\mathbf{1}$ so that the ratio value of $\mathbf{g}_i^f$ is relatively large at the beginning of training and the gradient vanishing effect is then diminished.

In addition to LSTM, there is another popular gated RNN model called gated recurrent unit (GRU) which has similar behaviors as LSTM but with less parameters (Cho et al., 2014). GRU can be considered as a simplified LSTM, which only maintains two gates (an update gate $\mathbf{g}^u$ and a reset gate $\mathbf{g}^r$). Given an input sequence $\{\mathbf{x}_t\}$, the computational dependencies of GRU at recurrent step t are defined as follows:

$$\mathbf{g}_t^u = \sigma(\mathbf{W}_u \mathbf{x}_t + \mathbf{U}_u \mathbf{h}_{t-1} + \mathbf{b}_u), \tag{2.19}$$

$$\mathbf{g}_t^r = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r), \tag{2.20}$$

$$\mathbf{z}_t = \phi(\mathbf{W} \mathbf{x}_t + \mathbf{U}(\mathbf{g}_t^r \odot \mathbf{h}_{t-1}) + \mathbf{b}), \tag{2.21}$$

$$\mathbf{h}_t = (1 - \mathbf{g}_t^u) \odot \mathbf{h}_{t-1} + \mathbf{g}_t^u \odot \mathbf{z}_t. \tag{2.22}$$

From Eq.(2.22) we can see that GRU's hidden state $\mathbf{h}$ plays similar role as the memory cell $\mathbf{c}$ in LSTM: the update gate $\mathbf{g}^u$ works as the input gate $\mathbf{g}^i$ in LSTM and $(1 - \mathbf{g}^u)$ works as the forget gate $\mathbf{g}^f$ in LSTM. However, LSTM has no restriction between $\mathbf{g}^i$ and $\mathbf{g}^f$ while GRU forces these two gates summing up to $\mathbf{1}$. The reset gate $\mathbf{g}^r$ is a new functionality compared with gates in LSTM. When $\mathbf{g}^r$ is close to $\mathbf{0}$, GRU is forced to drop all the historical information passed through the previous hidden state, which helps eliminating redundant information useless for the future prediction. As a whole, the reset gate is mainly responsible for capturing short-term dependencies in data distribution while the update gate tends to put more focus on long-term dependencies.

Chung et al. (2014) made empirical performance comparisons between LSTMs and GRUs on a wide range of tasks and they found GRUs to be comparable to LSTMs.

2.2.5 Sequence-to-sequence Models

A wide range of learning scenarios can be formed as a mapping problem that takes a sequence as the input and outputs another sequence. For instance, in a French-to-English translation task, a French sentence will be mapped (translated) to the corresponding English sentence, where the input is a sequence of French words and the output is a sequence of English words.

From an RNN perspective, such sequential learning scenario can be modeled by the so called *neural sequence-to-sequence model* introduced in neural machine translation (Cho et al., 2014; Sutskever et al., 2014). A neural sequence-to-sequence model consists of an encoder RNN f_{enc} and a decoder RNN f_{dec}. Given an input sequence $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$, the encoder RNN reads each x_t, perform recurrent computation iteratively and outputs a final encoder representation r summarizing the entire $\mathbf{x}$:

$$r = f_{enc}(\mathbf{x}) \tag{2.23}$$

r is usually the last hidden state of the encoder RNN. Given the output target $\mathbf{y} = \{y_1, y_2, \cdots, y_m\}$, the decoder RNN models a target distribution $P(\mathbf{y}|\mathbf{x})$ conditioned on $\mathbf{x}$ by taking r into consideration:

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= f_{dec}(\mathbf{y}, r) \\ &= \prod_{t=1}^{m} P(y_t|r, \mathbf{y}_{1:t-1}). \end{aligned} \tag{2.24}$$

Eq.2.24 shows the mechanism of f_{dec} where, in each recurrent step t, the decoder RNN generates a conditional distribution $P(y_t|r, \mathbf{y}_{1:t-1})$ to predict y_t.

In practice, the input sequence $\mathbf{x}$ can be very long, especially in tasks like machine translation and machine reading comprehension. In these tasks, the fixed encoder representation r can hardly capture all the proper information from $\mathbf{x}$. As a result, there exists a huge information loss of input context during the decoding process which leads to significant performance drop (Bahdanau et al., 2014) .

2.2.6 Attention Mechanism

One solution for the encoder information loss problem in sequence-to-sequence models is *attention mechanism* first proposed for neural machine translation (Bahdanau et al., 2014). The intuition behind is simple: to generate target symbols in different decoding steps, the decoder RNN's focus on input information should be different. Specifically, at each decoding step t, instead of conditioning on the fixed input representation r, the decoder first computes a group of attention weights $\{w_{t,i}\}$ over the entire input sequence $\{x_k\}$:

$$w_{t,i} = \frac{exp(e_{t,i})}{\sum_{k=1}^{n} exp(e_{t,k})}, \tag{2.25}$$

where $e_{t,i}$ depends on the last decoder hidden state h_{t-1}^{dec} and the encoder hidden state h_i^{enc} at step i:

$$e_{t,i} = \varphi(h_{t-1}^{dec}, h_i^{enc}). \tag{2.26}$$

Then we can obtain a time-varying context vector c_t by combining current attention weights and input hidden states as a weighted sum:

$$c_t = \sum_{i=1}^{n} w_{t,i} h_i^{enc}, \tag{2.27}$$

where c_t serves as the dynamic summarization over the entire input sequence at time t, now the target distribution $P(\mathbf{y}|\mathbf{x})$ becomes:

$$P(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{m} P(y_t|c_t, \mathbf{y}_{1:t-1}). \tag{2.28}$$

From the above equations we can clearly verify that, the attention mechanism allows flexible concentration on different part of input sequence when predicting different y_t and thus achieves higher input information utilization. Intuitively, the attention mechanism mimics the information flow of human attention: assume that we are translating French sentence "J'ai une pomme" ("I haven an apple") into English, when generating the word "apple", we search over {"J'ai", "une", "pomme"} and focus on the French word "pomme", while the rest of the input "J'ai une" are mostly neglected by our brain.

As a concise and effective approach of integrating different pieces of information, the original attention mechanism has been extended to different variants, including *bi-directional attention*(Seo et al., 2017), *self-attention* (Parikh et al., 2016; Lin et al., 2017) and a fully attentive model called *transformer* built upon self-attentive structures (Vaswani et al., 2017). Notably, transformers significantly outperform previous deep architectures in machine translation. Moreover, Devlin et al. (2018) recently introduced a language representation model called *BERT* which achieved state-of-the-art performances in almost all the standard natural language processing (NLP) tasks such as language inference and sentence classification. Surprisingly, BERT is a pre-trained deep bi-directional transformer merely adding an output layer without any task-specific structure design. All these examples demonstrate the strong vitality of the attention mechanism.

2.2.7 External Memory

In Section 2.2.4, we discussed the memory cell in LSTM/GRU designed for memorizing long-term information. However, its structural simplicity limits its performance on complicated memorizing tasks such as machine reading comprehension. In this section we consider two advanced memory structures: *end-to-end memory network (EMN)* (Sukhbaatar et al., 2015a) and *neural turing machine (NTM)* (Graves et al., 2014). Both of them have some external memories with read-and-write protocols and RNN based controllers. These newly introduced functionalities allow the recurrent model to perform more stable information storage with more flexible memory access.

End-to-end Memory Network

The end-to-end memory network (EMN) can be considered as an end-to-end continuous form of the original memory network (MN)(Weston et al., 2014). The idea behind MN and EMN is that, given any input set $\{x_i\}_{i=1}^N$ to be stored, each x_i is converted into a memory vector $\mathbf{m}_i$ with dimension d in continuous space and we obtain an external memory matrix $\mathbf{M}$ with size of $N \times d$. When reading, a query representation $\mathbf{q}$ is generated (by some query generation function f in controller) and a context vector $\mathbf{c}$ is then retrieved from $\mathbf{M}$ through either discrete (hard) retrieval in MN:

$$\mathbf{c} = \arg\max_i \; s(\mathbf{q}, \mathbf{m}_i) \tag{2.29}$$

where s is some scoring function, or continous (soft) retrieval in EMN:

$$\mathbf{c} = \sum_i w_i \mathbf{m}_i \tag{2.30}$$

where weights $\{w_i\}$ is computed by a *softmax* function:

$$w_i = \frac{\exp(\mathbf{q}^T \mathbf{m}_i)}{\sum_j \exp(\mathbf{q}^T \mathbf{m}_j)} \tag{2.31}$$

Some representation $\mathbf{o}$ is then computed based on $\mathbf{c}$ to produce the final output.

MN and EMN also allow multi-hop reasoning by iteratively generating multiple $(\mathbf{q}_k, \mathbf{c}_k)$ pairs in K (predefined) steps. In the k-th step, $\mathbf{q}_k$ and $\mathbf{c}_k$ are computed based on previous $\mathbf{q}_{k-1}$ and $\mathbf{c}_{k-1}$ via query generation function f in the controller

$$\mathbf{q}_k = f(\mathbf{q}_{k-1}, \mathbf{c}_{k-1}) \tag{2.32}$$

and performing Eq.2.29 to Eq.2.31. The final output is then generated based on the last content vector $\mathbf{c}_K$.

Compared with memory cell vector in LSTM/GRU which has fixed dimension, the external memory matrix $\mathbf{M}$ in MN/EMN is (1) extendable and flexible in memory size, (2) suitable for almost any type of input data, as one only needs an embedding function to convert input x into the corresponding memory vector $\mathbf{m}$, (3) more stable in the sense of input information storage, as $\mathbf{m}_i$ will not be modified unless some writing-mechanism is introduced, (4) an attentive memory access protocol (in EMN) which allows more efficient and explainable input information utilization.

Neural Turing Machine

The neural turing machine (NTM) is another advanced recurrent model. Like MNs/EMNs, the NTM also has an external memory matrix $\mathbf{M}$, but in this case $\mathbf{M}$ is not independent input embeddings outside the NTM's computational structure. Instead, $\mathbf{M}$ has fixed size and is accessible only internally by the NTM controller.

The NTM has both a *reading-mechanism* and a *writing-mechanism*. Suppose that $\mathbf{M}_t$ has size of $N \times d$ and its values may change over time. In the reading-mechanism, a reading weight vector $\mathbf{w}_t^r$ with size of N is emitted based on a generated reading head at time t, which satisfies: (1) each element $w_{i,t}^r$ is non-negative and (2) $||\mathbf{w}_t^r||_1 = 1$. The read context vector $\mathbf{r}_t$ is then computed as the weighted sum of all the memory slots $\mathbf{m}_{i,t}$ in current $\mathbf{M}_t$:

$$\mathbf{r}_t = \mathbf{M}_t^T \mathbf{w}_t^r = \sum_i w_{i,t}^r \mathbf{m}_{i,t}. \tag{2.33}$$

$\mathbf{r}_t$ is then sent to the controller for further processing.

The writing-mechanism in the NTM is inspired by the input gate and forget gate in the LSTM, where the two gates are now generalized as *erase* and *add* actions. Specifically, at time t, the controller emits a writing weight vector $\mathbf{w}_t^w$, an erase vector $\mathbf{e}_t$ with its elements ranging in $[0, 1]$ and an add vector $\mathbf{a}_t$, the memory slot $\mathbf{m}_{i,t-1}$ in $\mathbf{M}_{t-1}$ at time $t-1$ is then modified as:

$$\mathbf{m}_{i,t} = \mathbf{m}_{i,t-1} \odot [1 - w_{i,t}^w \mathbf{e}_t] + w_{i,t}^w \mathbf{a}_t \tag{2.34}$$

Eq.2.34 shows that, the weight $w_{i,t}^w$ controls how much the $\mathbf{m}_{i,t}$ is concentrated among all slots in $\mathbf{M}_t$, while $\mathbf{e}_t$ and $\mathbf{a}_t$ are fine-grained modifications on different dimensions of $\mathbf{m}_{i,t}$ once it is concentrated.

From both EMNs and NTMs, we can find that the attention mechanism plays a very important role in memory accessing. In fact, in the sequence-to-sequence model, the encoded input sequence can be considered as an external memory while the decoder serves as the controller which queries the input encoding at every time step t using an attention mechanism.

2.2.8 Making RNNs deeper

Pascanu et al. (2014) raised another important aspect of RNNs: the *depth* . Unlike the feed-forward network in which the "depth" is simply defined as the number of nonlinearities between the input and output, it is not very clear what the "depth" means in RNNs and how to make RNNs "deeper". The authors first tried to discuss the concept of "depth" for RNNs, they then introduced three different ways to extend the depth of RNNs: (1) by stacking, (2) by adding extra nonlinearities between hidden layer and output layer and (3) by adding extra nonlinearities inside RNNs' transition functions (between two consecutive hidden states). They empirically evaluated their proposed deeper RNNs on several sequential learning tasks and showed the effectiveness of increasing the depth of RNNs. However, RNN architectures discussed in this work are quite limited and the authors did not give a rigorous analysis of the depth in an RNN, since the RNN's topological connecting architecture can be arbitrary. In Section 4 we will give much more detailed analysis about RNN connecting architectures and the meaning of "depth" under the most general conditions.

2.3 Learning Neural Natural Language Representations

What is *natural language*? A language is said to be *natural* when it is evolved naturally through daily use in a human society, reflecting human's broad and complicated consciousness towards the real world. Unlike formal/constructed languages such as computer programming languages which have pre-defined rigorous rules, natural languages are less regular and more flexible.

Natural language processing (NLP) is mainly about how computers process and understand natural language. Because of the essence of natural language, the research scope of NLP also strongly overlaps with domains like linguistics and psychology. Obviously, natural language runs counter to the nature of regularness and well-ordering rooted in computers, and that makes NLP a quite challenging problem. In early days, researchers made efforts in building hand-crafted rule-based NLP systems and proposing complex, structured ontologies to encode natural language into computer-readable forms((Winograd, 1971; Schank and Abelson, 1977)). Many are optimistic that a broad set of NLP tasks, such as machine translation(Hutchins et al., 1955) and chatbot(Weizenbaum, 1966), can be solved to some extent, through careful design of a set of complete rules. Unfortunately, none of these early tries successfully fulfilled their expectations, since the researchers strongly overestimated the robustness of rule-based systems when facing real, complicated natural language scenarios.

During the late 1980s, statistical machine learning (ML) appeared as the game changer in this field (Manning et al., 1999). ML approaches provide probabilistic views of natural language data: dependencies between different language components are no more hard-coded but are modeled in a soft way with degree of uncertainty. The ability to model uncertainty makes ML approaches much more robust in complex NLP problems especially when non-standard expressions and errors are heavily involved in the input data. Moreover, Compared with rule-based systems built upon human experts' knowledge and heuristics, ML approaches enable fully automatic learning on the raw natural language data, which allows the system to keep improving itself as more data are provided, thus it can capture the true underlying data distribution more accurately and generalize better on unseen examples. During this period, methods such as the hidden Markov model (HMM) (Charniak et al.), the decision tree (Màrquez and Rodríguez, 1998) and different kinds of Bayesian models (McCallum et al., 1998; Stolcke and Omohundro, 1994) were proposed for solving various tasks including part-of-speech tagging, language modeling in speech recognition, parsing, document classification, machine translation and so on.

In the beginning of this thesis, we already discussed the recent resurgence of deep neural networks starting from the early 2010s. Not surprisingly, this deep learning trend also brought fresh blood to the NLP community, during which the *deep representation learning* and *end-to-end learning paradigm* dominated in the field and deep learning based models achieved state-of-the-art results in almost all the major NLP tasks. Generally speaking, deep representation learning aims to take advantage of the deep neural networks to automatically extract hierarchical representations from the raw natural language input. Such learning process can be done in either supervised or unsupervised way, and the learned rich representations are then fed into downstream tasks. Furthermore, the end-to-end learning paradigm strongly simplifies the problem formulation and modeling pipeline of the NLP tasks. It abandons all the intermediate feature and model engineering, allowing a more concise architecture as long as the input and output of the problem are clearly defined. In the following sections, we will examine how these deep learning techniques reshaped several major aspects of NLP.

2.3.1 Neural Language Model

As we discussed before, natural language data can be modeled in probabilistic ways, among which a simple setup is the *language model (LM)* (Manning et al., 1999). Given a piece of natural language content such as a sentence, an article or even the entire Wikipedia, one can form it as a sequences of words $\{w_1, w_2, \cdots, w_L\}$. An LM assigns a joint probability distribution $p(w_1, w_2, \cdots, w_L)$ over the entire sequence, this is achieved by decomposing $p(w_1, w_2, \cdots, w_L)$ into the product of all the successive conditional distributions:

$$p(w_1, w_2, \cdots, w_L) = \prod_{i=1}^{L} p(w_i | w_1, \cdots, w_{i-1}). \tag{2.35}$$

From another point of view, Eq.2.35 shows that LM should have the ability of predicting the current word based on all the preceding words.

N-gram Language Model

A classic statistical LM is the *N-gram language model* (N-gram LM). In N-gram LM, we make the Markov assumption that the current word only depends on previous $N-1$ words, and Eq.2.35 becomes:

$$p(w_1, w_2, \cdots, w_L) = \prod_{i=1}^{L} p(w_i | w_{i-N+1}, \cdots, w_{i-1}). \tag{2.36}$$

The conditional distribution $p(w_i | w_{i-N+1}, \cdots, w_{i-1})$ in N-gram LM is count-based, which is simply calculated from corresponding frequency counts in the original word sequence:

$$p(w_i|w_{i-N+1},\cdots,w_{i-1}) = \frac{N_{\{w_{i-N+1},\cdots,w_{i-1},w_i\}}}{N_{\{w_{i-N+1},\cdots,w_{i-1}\}}}, \tag{2.37}$$

where $N_{\{\cdots\}}$ denotes the count of subsequence $\{\cdots\}$ appearing in the original training sequence. Eq.2.37 implies an inherent problem of the naive count-based N-gram LM: it will assign the probability of zero to an unseen sequence, which is quite often the case. We call this problem the *data sparsity* problem. In practice, we also need to introduce some smoothing strategy to force the N-gram LM to assign non-zero probabilities over the entire space of all the possible sequences, so that the data sparsity problem is a bit alleviated (Büttcher et al., 2016).

However, there are several inherent drawbacks of N-gram LM: (1) It suffers from the *curse of dimensionality* that, the possible word sequences increase exponentially as the dependency length N increases linearly. This causes the data sparsity problem mentioned before and smoothing techniques have quite limited performance when the potential sequence space becomes huge. (2) It is unable to grasp various syntactic and semantic meanings behind words and sentences since only N-gram exact string match counts in Eq.2.37. (3) The Markov assumption is an inaccurate approximation which may result in failures of capturing long-term dependencies and is unable to discover complex non-linear sequential dependencies among words.

Neural Language Model

The *neural language model (NLM)* addresses all the above issues in N-gram LM. There are two different kinds of NLM: *feed-forward neural language model (feed-forward NLM)* (Bengio et al., 2003) and *recurrent neural language model (recurrent NLM)* (Mikolov et al., 2010). Feed-forward NLM has two significant structural differences compared with N-gram LM. Firstly, for each unique word w in the input vocabulary, a feed-forward NLM assigns a real-valued vector $\mathbf{e}_w$ to w. This vector is trainable, it works as a computable distributed feature representation of w, or in other words, the *embedding* of w. Secondly, the conditional distribution $p(w_i|w_{i-N+1},\cdots,w_{i-1})$ is modeled by a feed-forward NN $f_{NN}(\cdot)$, in which the current word w_i further depends on embeddings of previous $N-1$ words:

$$p(w_i|w_{i-N+1},\cdots,w_{i-1}) = f_{NN}(\mathbf{e}_{w_{i-N+1}},\cdots,\mathbf{e}_{w_{i-1}}). \tag{2.38}$$

During learning, both these embeddings and the feed-forward NN keep being improved to better fit the language distribution. The introduced distributed embeddings and NN based probabilistic function strongly alleviate the curse of dimensionality problem, enabling much better interpolation and extrapolation for unseen examples. Moreover, if two words are semantically similar, their embeddings are close in the feature space, illustrating that these embeddings can encode word semantics to some extent.

The recurrent NLM discards the explicit word-level Markov assumption in N-gram LM and feed-forward LM. In recurrent NLM, the recurrent transition function $f_{RNN}(\cdot)$ manages to iteratively encode every word in a sequences with arbitrary length. Given a sequence $\{w_1, \cdots, w_L\}$ and corresponding embedding vectors $\{\mathbf{e}_{w_1}, \cdots, \mathbf{e}_{w_L}\}$, the history up to step $t-1$ in that sequence is summarized in its hidden state $\mathbf{h}_{t-1}$:

$$\mathbf{h}_{t-1} = f_{RNN}(\mathbf{e}_{w_{t-1}}, \mathbf{h}_{t-2}). \tag{2.39}$$

The $p(w_i|w_{i-N+1}, \cdots, w_{i-1})$ is then computed based on $\mathbf{h}_{t-1}$. Here we should emphasize that, compared with the feed-forward NLM, the recurrent NLM is better at learning sequential data due to its recurrent nature.

To sum up, NLMs have strong capacities of modeling complex language dependencies with better generalization abilities. One should notice that the recently proposed killer model BERT discussed previously can be considered as an advanced variant of the original NLM, which are trained on significant larger corpus.

2.3.2 Word Embedding and Beyond

In this section, we will dive deep into the distributed word embedding vectors proposed but not fully examined in Bengio et al. (2003). Generally speaking, a word embedding is a representation technique that maps words into vectors with their semantic meanings encoded in the corresponding vector space. A word embedding model is usually learned through unsupervised language modeling on some corpus. Since the embedding vector is the compressed distributed representation of the word, its dimension is much smaller than the vocabulary size of the corpus. Nowadays, the trained word embedding serves as the standard building block of input representation for lots of downstream NLP tasks including parsing, classification, language generation and so on.

The *word2vec* model is the most popular word embedding model (Mikolov et al., 2013a,b). The motivation of word2vec is to reduce the computational complexity in standard NLMs by implementing simpler structure with less nonlinearities, at the same time preserving the quality of the learned distributed word representations. Specifically, word2vec adopts a log-linear function and the non-linear hidden layer in the original NLM is removed. There are two new architectures proposed in word2vec: *continuous skip-gram model (CSG)* and *continuous bag-of-words model (CBOW)* (Mikolov et al., 2013a).

In CSG, given the current word, the model predicts its neighbors. More formally, given the corpus $C = (w_1, w_2, \cdots, w_{N_C})$ with vocabulary V, CSG aims to maximize the log probability of observing w_t's surrounding words within a fixed context window size c given w_t:

$$\sum_{t=1}^{N_C} \sum_{-c \le k \le c, k \ne 0} \log p(w_{t+k}|w_t). \tag{2.40}$$

The conditional distribution $p(w_{t+k}|w_t)$ is defined as:

$$p(w_{t+k}|w_t) = \frac{\exp(\mathbf{e}_{w_t}^T \mathbf{o}_{w_{t+k}})}{\sum_{w \in V} \exp(\mathbf{e}_{w_t}^T \mathbf{o}_w)}. \tag{2.41}$$

$\mathbf{e}_w$ is the word embedding of w while $\mathbf{o}_w$ is the output word representation of w. Compared with the CSG model, the CBOW model has similar architecture and learning objective while it predicts the current word w_t given w_t's neighbors.

The trained word2vec representations have several interesting syntactic/semantic properties: (1) Naive similarity. Given two words w_1 and w_2 with similar meaning, their embedding vectors $\mathbf{e}(w_1)$ and $\mathbf{e}(w_2)$ are close in embedding space. For example, the cosine distance between $\mathbf{e}(cat)$ and $\mathbf{e}(dog)$ is much smaller than the cosine distance between $\mathbf{e}(cat)$ and $\mathbf{e}(car)$. (2) Subtractive similarity. Given two word pairs $(w_1, \tilde{w}_1)$ and $(w_2, \tilde{w}_2)$ where $\tilde{}$ implies some syntactic or semantic relation, we have that the subtraction $\mathbf{e}(\tilde{w}_1) - \mathbf{e}(w_1)$ is close to the subtraction $\mathbf{e}(\tilde{w}_2) - \mathbf{e}(w_2)$ in cosine distance, or in other words, $\mathbf{e}(\tilde{w}_2)$ is close to $\mathbf{e}(\tilde{w}_1) - \mathbf{e}(w_1) + \mathbf{e}(w_2)$. For example, we have $\mathbf{e}(slowly)$ is closest to $\mathbf{e}(quickly) - \mathbf{e}(quick) + \mathbf{e}(slow)$ where the adjective-adverb relation is captured. Another example is that $\mathbf{e}(queen)$ is closest to $\mathbf{e}(king) - \mathbf{e}(man) + \mathbf{e}(woman)$ where the title and gender information is encoded.

(3) Additive similarity. Given words w_1, w_2 and w_3, if the meaning of w_3 can be obtained by semantically combining w_1 and w_2, then in embedding space we often have that the closest embedding of $\mathbf{e}(w_1)+\mathbf{e}(w_2)$ is $\mathbf{e}(w_3)$. For example, $\mathbf{e}(Hanoi)$ is the closest embedding to $\mathbf{e}(Vietnam)+\mathbf{e}(capital)$. In short, the above properties all demonstrate word2vec's strong capacity of representing different aspects of word characteristics.

Inspired by word embedding models, researchers made further extensions and developed new embedding models for higher level language components. Le and Mikolov (2014) proposed the *paragraph vector*, which adopts the same model architecture as word2vec but used it for encoding paragraphs. Kiros et al. (2015) developed a sentence-level embedding called *skip-thought vector*, which is obtained by training an RNN encoder-decoder model. The idea is similar to the skip-gram model in word2vec but where the current sentence is encoded as a vector which is further used to predict surrounding sentences. Lin et al. (2017) presented a self-attentive sentence embedding model where the embedding becomes a matrix with each row representing an attention over a different part of the sentences. Moreover, from the embedding point of view, powerful models such as ELMO (Peters et al., 2018) and BERT (Devlin et al., 2018) can also be considered as embedding functions which take arbitrary word sequences as input and output hierarchical representations.

2.3.3 Neural Reading Comprehension

Given a piece of natural language article, how can we examine whether a person understands the article context? Because "understanding" is a subjective mental activity that cannot be explicitly investigated outside the person's own consciousness, some indirect measurement is required. One efficient way is to ask questions about the article to that person. Since the question can be at any level addressing any aspect of that article, the person is said to "understand the article" if he is able to answer all the related questions. The same claim holds for machines. In simpler scenarios such as syntactic parsing, machines can only "understand" the context up to a given task-specific level, which is the "word-level syntax" in this case. Only when the *machine reading comprehension* (MRC) task was introduced, one has the possibility of fully testing a machine's capacity for general understanding of natural language. On the other hand, MRC is quite challenging because it requires the machine to effectively integrate different levels of NLP skills in both syntactic and semantic aspects. In the following, we will briefly go over the history of MRC and discuss the recent progress in this field including new data sources and new models, especially those related to deep learning approaches.

History

As a subfield of NLP, the development pathway of MRC systems strongly overlaps with the historical progress of the broader NLP world discussed in the beginning of this section. Before advanced machine learning approaches become popular, MRC did not attract much attention from AI researchers and there were only a few attempts towards address this problem (Lehnert, 1977; Schank and Abelson, 1977). Unsurprisingly, most of these early tries were rule-based systems built upon simple hand-crafted syntactic/semantic features with very limited generalization capacity. One of the representative work during this period was from (Hirschman et al., 1999), where the authors built a bag-of-words pattern matching system comprising various linguistic processing. They also released the first MRC dataset, although the size of the dataset is very small (120 stories in total for validation and test set).

Things started to change in the early 2010s, when larger datasets with training examples were proposed and statistical machine learning models finally came to the stage of MRC. The most famous dataset during this period is MCTest (Richardson et al., 2013). MCTest has 660 children-level fictional stories as the reading context with more than 2000 related questions, and the answer is in a multi-choice form which offers a clear metric. Besides, MCTest is the first MRC dataset collected using a scalable crowd-sourcing method and it is a reference for data collecting methodology for later large-scale MRC datasets. MCTest inspired the design of bunch of statistical MRC models (Wang et al., 2015; Yin et al., 2016) which achieved considerable improvements over their rule-based predecessors. However, the drawbacks of these traditional statistical models are clear: the linguistic feature engineering is still a necessity to get meaningful input representations, while the existing off-the-shelf linguistic tools often have generalization problems on complicated MRC tasks (Chen, 2018).

MRC finally reached the era of *neural reading comprehension* (NRC) on 2015, when Hermann et al. (2015) proposed the *attentive reader* together with a large scale MRC dataset later referred to as the *CNN/DailyMail* dataset. The attentive reader was the first to adopt a neural attention based model with the end-to-end learning process (inspired by Bahdanau et al. (2014)) and it strongly outperformed the previous non-neural models. Besides, the CNN/DailyMail dataset embraced a tricky cloze-style[1] answer form allowing an extreme cheap data collection. However, the dataset has several inherent limitations making it a less satisfying testbed, as pointed out in (Chen et al., 2016; Chen, 2018). In 2016, Rajpurkar et al. (2016) built the *Stanford question answering dataset* (SQuAD) which became the most popular NRC dataset in the following two years. SQuAD gathered more than 100,000 question-answer pairs from Wikipedia articles enabling large-scale end-to-end training. Since its release, we witnessed a bloom of neural models quickly pushing the boundary of NRC close to human-level performance. In the following, we will further discuss in detail about this important dataset, the neural architectures inspired by it and the future of NRC. Here, we first give a formal description of the NRC problem. NRC aims to train a deep neural network model whose inputs are paragraph content c and a question q and output is an answer a. Different from tasks like open domain *question answering* (QA) where the question can be arbitrary, in NRC the q is required to be answerable solely depending on c. The capacity of the system is evaluated on how accurate the generate answer a is matched with the ground truth answer a^*. In detail, the evaluation depends on the form of the answer. In a common case, a^* is a span of words exactly matching with some subsequence in the given content. The system performance is then evaluated by exact match (EM) and F1 score. For other answer forms like multiple choices or cloze-style, the performance is exactly the accuracy of making the correct choice or filling in the correct words.

SQuAD: Dataset and Models

The Stanford question answering dataset (SQuAD) is without a doubt a milestone in the field of NRC. The original SQuAD contains more than 100,000 questions extracted from 23,215 paragraphs of 536 Wikipedia articles which has a broad coverage of different topics. It has several notable features: (1) It is the first large-scale MRC dataset with fully human-generated high quality data. It takes

---

[1] A cloze-style test is a language test that certain words in a sentence are removed and the participant is asked to fill in these the missing words.

advantage of crowd-sourcing techniques through launching a careful designed data collection pipeline on the Amazon Mechanical Turk platform. (2) It is the first MRC dataset where the question is in fluent natural language. (3) To offer a feasible evaluation metric, the answer adopts the form of the span of words in the corresponding paragraph. As we already discussed, this allows both the accuracy-style metric like exact match (EM) and softer measures like F1 score which is more suitable for long answers. SQuAD is also quite diverse in the sense of answer types and reasoning types: The answers vary from nouns like person/data/location to adjective/verb phrases and clauses. The reasoning required for answering questions also includes syntactic/lexical variation and multiple sentence reasoning. In a word, because of its scale, data quality and data diversity, SQuAD stood out from its competitors at the time it was released.

SQuAD is over-complicated for standard shallow models as already shown from the unsatisfying performance of the baseline model in its original paper. However, it soon became the hottest testbed for deep learning models and significantly drove ahead the deep architecture design for the MRC problem. Most of these architectures have a similar processing pipeline: First, a passage contextual information encoder $g^c_{enc}(\cdot)$ and a question encoder $g^q_{enc}(\cdot)$ are applied given the input passage c and question q. Then, some integrating mechanism $f_{int}(\cdot)$ combines the information flow from both c and q, conducts various multi-level neural operations and finally outputs the start and end positions of the answer (p_s, p_e) in c:

$$(p_s, p_e) = f_{int}(g^c_{enc}(c), g^q_{enc}(q)). \tag{2.42}$$

Tracing back to the evolution of these architectures, most efforts have been made on $f_{int}(\cdot)$. Wang and Jiang (2016) proposed *Match-LSTM + Answer-pointer* which was the first deep model that achieved significant progress on this task. Seo et al. (2017) improved the performance by a large margin through a bi-attentional structure called *BiDAF*, which was then widely used as a basic building block of many successors. Since BiDAF, many other researchers made further architectural extensions by developing more complicated attention mechanisms such as co-attention, self-attention and other variants (Xiong et al., 2016; Clark and Gardner, 2017; Huang et al., 2017). The *ensemble* approach and the involvement of *reinforcement learning* techniques also helped on pushing the performance boundary close to the human upper bound (Hu et al., 2017). Finally, the famous BERT architecture, as we mentioned several times before, unbelievably surpassed the human performance and thus declared the end of the glorious SQuAD era (Devlin et al., 2018).

Post-SQuAD Era: What is Next?

However, solving SQuAD is far from solving the general MRC problem. In fact, SQuAD has limitations in several respects: First of all, it is over-simplified that the answer is restricted in the span-of-words form. A more generic case should be free-text form which has no constraints on how the answer is phrased. More importantly, the majority of the questions can be answered solely depending on matching with one sentence in a single paragraph, which we refer to as *single-hop reasoning*. This limited the dataset's scope when evaluating more intricate reasoning skills.

To overcome the problems of SQuAD, many new datasets have been collected in 2018. Up to the date of writing this thesis, the notable datasets includes: *SQuAD 2.0*, which is an advanced version of the original SQuAD with extra newly collected 50,000 unanswerable examples (Rajpurkar et al., 2018); *TriviaQA*, which consists of question-answer pairs accompanied with multiple supporting documents, although often the information from one document is already enough to answer the given question (Joshi et al., 2017b); *QAngaroo*, which is the first dataset evaluating a system's multi-hop reasoning capacity (Welbl et al., 2018b). In other words, to answer the question correctly the system must combine evidence from at least two passages and perform multi-step reasoning over them. However, because the data collection in QAngaroo is based on pre-existing knowledge base (KB) schema, the question is not in natural language form and its type diversity is significantly restricted; *HotpotQA*, which also facilitates multi-hop reasoning like QAngaroo, but this time the question is pure natural language without any KB constraints and is accompanied by accurate supporting facts which enables the system making explainable predictions. As one of the major works of this thesis, we will give a thorough investigation of HotpotQA in Section 8.

2.3.4 Neural Dialogue System

Dialogue system has flexible definitions depending on the diverse scenarios considered. In the scope of this thesis, we consider a dialogue system as a machine learning model that can conduct conversations with humans. We only care about the natural language aspect where the input and output are textual responses, while ignoring modes such as speech recognition and synthesis, emotional expression and other human-computer interaction issues. This makes our discussion focusing on two hardest components of the dialogue system: *natural language understanding* (NLU) and *natural language generation* (NLG).

Dialogue systems hold a special and crucial status in the history of AI. In Alan Turing's 1950 paper "Computing Machinery and Intelligence" (Turing, 2009), the famous *Turing test* was proposed which is broadly considered as the first and the most influential approach to evaluate the intelligence of a non-human system. To answer the essential question "Can a machine think?", the Turing test suggests a simple idea: Let the machine being tested and a human participant both talk to a third human evaluator in a text-only conversational environment. Without knowing which is which, the evaluator is asked to judge the identity of the machine from these two conversational partners only based on the textual responses generated from them. The machine is said to behave *intelligently* if the human evaluator cannot tell the machine from the human participant. Apparently, Turing test is a subjective metric totally relying on the judgment of the human evaluator. However, because the concept of *intelligence* is too complicated to have an accurate and quantitative measurement, Turing test is so far the most feasible way of examining the general intelligence of a machine. Obviously, on can think the Turing test as a machine-human dialogue and the machine here exactly matches with all the characteristics of a dialogue system defined in the scope of this thesis. In other words, a successful dialogue system is at least a clear sign of approaching the final goal of *artificial general intelligence* (AGI).

History

A dialogue system is also referred to as a *chatbot*. We will use both names interchangeably in the rest of this thesis. The first chatbot was called *ELIZA* built in 1960s, originally aiming for illustrating how superficial the machine-human interaction can be (Weizenbaum, 1966). It consists of several simple executable scripts with mainly keyword matching and substitution tricks, pretending to focus and understand its dialogue partner's words but indeed not. Opposite to the expectation of its creator, many candidates talked to ELIZA really thought that it can indeed express its "personal feelings" and emotions. This phenomenon was even become a psychological term named the *ELIZA effect*, revealing the unpredictable social influence of chatbots on humans. The progress has been almost halted after then because of the AI winter of the 1970s. The practical success of expert systems in late 1980s pulled the interests back towards chatbots and one of the turning points was the launching of the *Loebner Prize* competition[2], which awards chatbots that are most human-like when conducting conversations. Many sophisticated chatbots came into view since then. One of the notable systems during this period is the *Alicebot* (Wallace, 2009), inspired by the original ELIZA. The Alicebot adopted *Artificial Intelligence Markup Language* (AIML) which is an XML based schema framework for encoding multi-level rules and heuristics. Alicebot won the Loebner Prize three times while AIML became a standard framework that many successors used it for constructing their own rule-based submodules. As happened in most NLP tasks already discussed, the dialogue system was also reformed by statistical data-driven approaches since late 1990s. Through this line of research we see Markov approaches which model dialogues as sequences of either words or decisions (Hutchens and Alder, 1998; Singh et al., 2000; Young et al., 2013), traditional ML methods that formulate different dialogue system components as classification problems (Gorin et al., 1997; Reithinger and Klesen, 1997), and most importantly, the union with neural end-to-end learning methodology which empower superior model construction both in scale and in quality (Vinyals and Le, 2015; Sordoni et al., 2015; Serban et al., 2016b).

Neural Dialogue Models for General Chitchat

As introduced earlier, the standard text-based dialogue system consists of two parts, the NLU part which processes and converts the current dialogue input into internal representations via dialogue state tracking and action modeling, and the NLG part which outputs responses in either generative or retrieval ways. In a *neural dialogue system*, NLU, NLG and even their sub-components (for example,

---

[2] http://www.aisb.org.uk/events/loebner-prize

the dialogue state tracker in (Mrkšić et al., 2015)) can be parameterized as neural networks. In most situations however, the entire system is modeled as a variant of the simple neural sequence-to-sequence model ready for end-to-end training (Cho et al., 2014; Vinyals and Le, 2015). In detail, given the current dialog history $H_{t-1} = (x_1, r_1, ..., x_{t-1}, x_{t-1})$ where r_i and x_i stand for responses of the model and the other dialogue partner respectively, an encoder RNN $f_{enc}(\cdot)$ first converts the current response x_t into some intermediate representation. It is then merged with the last inner dialogue state s_{t-1} (summarizing the previous dialogue history) by a dialog state tracking network $f_H(\cdot)$ (usually another RNN or more advanced external memories) to produce the current state inner s_t. The latest model response r_t is then generated by another RNN decoder $f_{dec}(\cdot)$ based on s_t. Formally, the whole process in the t-th round can be formularized using the following equation:

$$r_t = f_{dec}(f_H(f_{enc}(x_t), s_{t-1})). \tag{2.43}$$

Depending on the practical purpose, chatbots can be categorized as *goal-oriented* systems and *non-goal-oriented* ones which are also called *chitchat bots*. As its name denotes, the goal-oriented system is designed to help the human partner attaining some specific goal such as flight/hotel booking and product technical support (Aust et al., 1995; Young et al., 2013; Wen et al., 2016). Existing goal-oriented chatbots are often hybrid ones incorporating discrete predefined slot/action space while utilizing statistical/neural procedures modeling the intermediate features for user intention and responding policy. Because the goal-oriented system's application scenarios are often controlled environments with clear boundaries and limited outliers, its practical implementation and commercialization are rather feasible. On the other hand, chitchat systems are facing the open domain understanding problem as the user responses can cover arbitrary topics with arbitrary phrasings during a casual conversation. Early systems like Alicebot (Wallace, 2009) attempted to manually cover a wide range of topics and daily conversation tricks by hand-crafted AIML scripts which is hardly extendable. Thanks to the explosion of public information posted in social media, forums and video websites, researchers collected bunch of large scale chitchat data sources including *Twitter Corpus* (Ritter et al., 2010), *Reddit Corpus* (Schrading et al., 2015), *Ubuntu Dialogue Corpus* (Lowe et al., 2015), *OpenSubtitles* (Tiedemann, 2012) and so on. Taking advantages from both corpora richness and the representative capacity of neural architectures, a new trend of end-to-end chitchat models quickly dominated the field (Hutchens and Alder, 1998; Singh et al., 2000; Young et al., 2013). Distinct from its counterparts like retrieval or rule-based models, end-to-end models can generate more tailored and coherent responses by projecting the dialogue history into comprehensive continuous feature space. However, the performance evaluation of a chitchat bot is quite challenging compared with its goal-oriented cousin which can at least be measured by whether it completes the goal through the conversation. For chitchat bot, automatic metrics in other NLP domains like BLEU in machine translation or ROUGE in document summarization are almost unacceptable, as empirically proved in (Liu et al., 2016). This is mainly because the language diversity and context variation are so severe in a conversation that there even does not exist a ground truth response. The alternative is to take human-feedback based subjective evaluations derived from the Turing test methodology, which may be expensive and not quite scalable (Serban et al., 2015). Another open problem for end-to-end chitchat bots is about how to introduce and encode long term external information (knowledge) beyond the current dialogue history to improve the conversation consistency and engagingness. One remarkable idea is to assign personalities to chatbots so that they can better mimic manlike emotions and reactions and thus can smooth the conversation. Along this direction, researchers explored implicit ways of embedding personalities through learning from dialogues (Li et al., 2016b; Joshi et al., 2017a). In Section 10, we will investigate a recent dataset called *persona-chat* proposed by the author, allowing end-to-end models to explicitly model personalities in natural language form and legitimate a more engaging dialogue. Besides, an efficient crowd-sourcing based dialogue evaluation pipeline will be further examined on this dataset.

3 Prologue to First Article

3.1 Article Detail

Architectural Complexity Measures of Recurrent Neural Networks. Saizheng Zhang[1], Yuhuai Wu[1], Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan Salakhutdinov, Yoshua Bengio. Advances in Neural Information Processing Systems (NIPS), 2016.

Personal Contribution. The idea of exploring RNN connecting architectures came from several discussions among Yuhuai Wu, Zhouhan Lin and me. Yuhuai Wu and I further developed the rigorous graph-theoretic framework together with three architectural complexity measures. I proposed the original proofs of two main theories while Tong Che made several crucial polishments on them and finished the remaining proofs. I conducted the experiments on language modeling datasets and sequential MNIST dataset, Yuhuai Wu and Zhouhan lin contributed to the rest. Roland Memisevic, Ruslan Salakhutdinov and Yoshua Bengio offered several critical suggestions about the general framework formulation and experiments.

3.2 Context

This work systematically analyzes the connecting architectures of recurrent neural networks (RNNs). Specifically, we first present a rigorous graph-theoretical framework describing the connecting architectures of RNNs in general. We then propose three architecture complexity measures of RNNs: (a) the *recurrent depth*, which captures the RNN's over-time nonlinear complexity, (b) the *feedforward depth*, which captures the local input-output nonlinearity (similar to the "depth" in feedforward neural networks (FNNs)), and (c) the *recurrent skip coefficient* which

---

[1] Equal contribution.

captures how rapidly the information propagates over time. Our experimental results show that RNNs might benefit from larger recurrent depth and feedforward depth. We further demonstrate that increasing recurrent skip coefficient offers performance boosts on long term dependency problems.

In this work, we purely concentrate on the graph aspect of RNN connecting architectures, where we assume that all the hidden nodes share the same formulation of the transition function. This homogeneous assumption aims to disentangle other performance related factors. While there do exist other critical aspects that could influence the performance of an RNN, especially the computational operations in its transition function: Gating mechanism introduced in LSTMs and GRUs can dynamically control the information that flows through recurrent states and thus can further adjust the gradient propagation (Hochreiter and Schmidhuber, 1997; Cho et al., 2014); The initialization of hidden-to-hidden transition matrix also have significant impact on the RNN learning process (Le et al., 2015; Arjovsky et al., 2015). Besides, the recently introduced normalization techniques such as recurrent batch-normalization (Cooijmans et al., 2016) and layer-normalization (Ba et al., 2016) also play important roles in boosting the performance of RNNs.

3.3 Contributions

This work is the very first attempt to rigorously analyze the connecting architecture of RNNs. It introduced a general formulation of RNN architectures under which we could further evaluate the architectural complexity of an RNN via three novel measures: recurrent depth, feedforward depth, and recurrent skip coefficients. It also answered the question of “What is the depth of an RNN?” under the most general scenario. Empirical evidence from our experiments implied that these measures are able to provide guidance for the design and inspection of new recurrent architectures for a particular learning task.

4 Architectural Complexity Measures of Recurrent Neural Networks

4.1 Introduction

We focus on an important theoretical aspect of recurrent neural networks (RNNs): the connecting architecture. Ever since Schmidhuber (1992); El Hihi and Bengio (1996) introduced different forms of "stacked RNNs", researchers have taken architecture design for granted and have paid less attention to the exploration of other connecting architectures. Some examples include Raiko et al. (2012); Graves (2013) who explored the use of skip connections; Hermans and Schrauwen (2013) who proposed the deep RNNs which are stacked RNNs with skip connections; Pascanu et al. (2013a) who pointed out the distinction of constructing a "deep" RNN from the view of the recurrent paths and the view of the input-to-hidden and hidden-to-output maps. However, they did not rigorously formalize the notion of "depth" and its implications in "deep" RNNs. Besides "deep" RNNs, there still remains a vastly unexplored field of connecting architectures. We argue that one barrier for better understanding the architectural complexity is the lack of a general definition of the connecting architecture. This forced previous researchers to mostly consider the simple cases while neglecting other possible connecting variations. Another barrier is the lack of quantitative measurements of the complexity of different RNN connecting architectures: even the concept of "depth" is not clear with current RNNs.

In this work, we try to address these two barriers. We first introduce a general definition of a recurrent neural network, where we divide an RNN into two basic ingredients: a well-defined graph representation of the connecting architecture, and a set of transition functions describing the computational process associated with each unit in the network. Observing that the RNN undergoes multiple transformations not only feedforwardly (from input to output within a time step) but also recurrently (across multiple time steps), we carry out a quantitative analysis of the number of transformations in these two orthogonal directions, which results in the definitions of *recurrent depth* and *feedforward depth*. These two depths can be viewed as general extensions of the work of Pascanu et al. (2013a). We also explore a quantity called the *recurrent skip coefficient* which measures how quickly information propagates over time. This quantity is strongly related to vanishing/exploding gradient issues, and helps deal with long term dependency problems. Skip connections crossing different timescales have also been studied by Lin et al. (1996); El Hihi and Bengio (1996); Sutskever and Hinton (2010); Koutnik et al. (2014). Instead of specific architecture design, we focus on analyzing the graph-theoretic properties of recurrent skip coefficients, revealing the fundamental difference between the regular skip connections and the ones which truly increase the recurrent skip coefficients.

We empirically evaluate models with different recurrent/feedforward depths and recurrent skip coefficients on language modelling and sequential MNIST tasks. We also show that our experimental results further validate the usefulness of the proposed definitions.

4.2 General RNN

RNNs are learning machines that recursively compute new states by applying transition functions to previous states and inputs. It has two ingredients: the connecting architecture describing how information flows between different nodes and the transition function describing the nonlinear transformation at each node. The connecting architecture is usually illustrated informally by an infinite directed acyclic graph, which in turn can be viewed as a finite directed cyclic graph that is unfolded through time. In this section, we first introduce a general definition of the connecting architecture and its underlying computation, followed by a general definition of an RNN.

4.2.1 The Connecting Architecture

We formalize the concept of the connecting architecture by extending the traditional graph-based illustration to a more general definition with a *finite directed multigraph* and its *unfolded* version. Let us first define the notion of the *RNN* *cyclic graph* $\mathcal{G}_c$ that can be viewed as a cyclic graphical representation of RNNs. We attach "weights" to the edges in the cyclic graph $\mathcal{G}_c$ that represent time delay differences between the source and destination node in the unfolded graph representation.

Definition 4.2.1.1. *Let* $\mathcal{G}_c = (V_c, E_c)$ *be a weighted directed multigraph*[1], *in which* $V_c = V_{\text{in}} \cup V_{\text{out}} \cup V_{\text{hid}}$ *is a finite set of nodes, and* $V_{\text{in}}, V_{\text{out}}, V_{\text{hid}}$ *are not empty.* $E_c \subset V_c \times V_c \times \mathbb{Z}$ *is a finite set of directed edges. Each* $e = (u, v, \sigma) \in E_c$ *denotes a directed weighted edge pointing from node* u *to node* v *with an integer weight* σ. *Each node* $v \in V_c$ *is labelled by an integer tuple* (i, p). $i \in \{0, 2, \cdots m-1\}$ *denotes the time index of the given node, where* m *is the* ***period number*** *of the RNN, and* $p \in S$, *where* S *is a finite set of node labels. We call the weighted directed multigraph* $\mathcal{G}_c = (V_c, E_c)$ *an RNN cyclic graph, if*

(1) For every edge $e = (u, v, \sigma) \in E_c$, *let* i_u *and* i_v *denote the time index of node* u *and* v, *then* $\sigma = i_v - i_u + k \cdot m$ *for some* $k \in \mathbb{Z}$.

(2) There exists at least one directed cycle[2] *in* $\mathcal{G}_c$.

(3) For any closed walk ω, *the sum of all the* σ *along* ω *is not zero.*

(4) There are no incoming edges to nodes in V_{in}, *and no outgoing edges from nodes in* V_{out}. *There are both incoming edges and outgoing edges for nodes in* V_{hid}.

Condition (1) assures that we can get a periodic graph (repeating pattern) when unfolding the RNN through time. Condition (2) excludes feedforward neural networks in the definition by forcing to have at least one cycle in the cyclic graph. Condition (3) simply avoids cycles after unfolding.

The cyclic representation can be seen as a time folded representation of RNNs, as shown in Figure 4.1 (a). Given an RNN cyclic graph $\mathcal{G}_c$, we unfold $\mathcal{G}_c$ over time $t \in \mathbb{Z}$ by the following procedure:

Definition 4.2.1.2 (Unfolding). *Given an RNN cyclic graph* $\mathcal{G}_c = (V_c, E_c, \sigma)$, *we define a new infinite set of nodes* $V_{un} = \{(i + km, p) | (i, p) \in V, k \in \mathbb{Z}\}$. *The new set of edges* $E_{un} \in V_{un} \times V_{un}$ *is constructed as follows:* $((t, p), (t', p')) \in E_{un}$ *if and only if there is an edge* $e = ((i, p), (i', p'), \sigma) \in E$ *such that* $t' - t = \sigma$, *and* $t \equiv i (\operatorname{mod} m)$. *The new directed graph* $\mathcal{G}_{\text{un}} = (V_{un}, E_{un})$ *is called the unfolding of*

---

[1] A directed multigraph is a directed graph that allows multiple directed edges connecting two nodes.

[2] A directed cycle is a closed walk with no repetitions of edges.

$\mathcal{G}_c$. Any infinite directed graph that can be constructed from an RNN cyclic graph through unfolding is called an RNN unfolded graph.

Lemma 4.2.1.1. *The unfolding $\mathcal{G}_{\text{un}}$ of any RNN cyclic graph $\mathcal{G}_c$ is a directed acyclic graph (DAG).*

Figure 4.1(a) shows an example of two graph representations $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$ of a given RNN. Consider the edge from node $(1, 7)$ going to node $(0, 3)$ in $\mathcal{G}_c$. The fact that it has weight 1 indicates that the corresponding edge in $\mathcal{G}_{\text{un}}$ travels one time step, $((t + 1, 7), (t + 2, 3))$. Note that node $(0, 3)$ also has a loop with weight 2. This loop corresponds to the edge $((t, 3), (t + 2, 3))$.

The two kinds of graph representations we presented above have a one-to-one correspondence. Also, any graph structure θ on $\mathcal{G}_{\text{un}}$ is naturally mapped into a graph structure $\bar{\theta}$ on $\mathcal{G}_c$. Given an edge tuple $\bar{e} = (u, v, \sigma)$ in $\mathcal{G}_c$, σ stands for the number of time steps crossed by $\bar{e}$'s covering edges in E_{un}, i.e., for every corresponding edge $e \in \mathcal{G}_{\text{un}}$, e must start from some time index t to $t + \sigma$. Hence σ corresponds to the "time delay" associated with e.

In addition, the *period number* m in Definition 4.2.1.1 can be interpreted as the time length of the entire non-repeated recurrent structure in its unfolded RNN graph $\mathcal{G}_{\text{un}}$. Strictly speaking, m has the following properties in $\mathcal{G}_{\text{un}}$: $\forall k \in \mathbb{Z}$, if $\exists v = (t, p) \in V_{un}$, then $\exists v' = (t + km, p) \in V_{un}$; if $\exists e = ((t, p), (t', p')) \in E_{un}$, then $\exists e' = ((t + km, p), (t' + km, p')) \in E_{un}$. In other words, shifting the $\mathcal{G}_{\text{un}}$ through time by km time steps will result in a DAG which is identical to $\mathcal{G}_{\text{un}}$, and m is the smallest number that has such property for $\mathcal{G}_{\text{un}}$. Most traditional RNNs have $m = 1$, while some special structures like *hierarchical* or *clockwork RNN* (El Hihi and Bengio, 1996; Koutnik et al., 2014) have $m > 1$. For example, Figure 4.1(a) (unfolded graph representation $\mathcal{G}_{\text{un}}$) shows that the period number of this specific RNN is 2. It is clear that if there exists a directed cycle ϑ in $\mathcal{G}_c$, and the sum of σ along ϑ is positive (or negative), then there is a path which is the pre-image of ϑ in $\mathcal{G}_{\text{un}}$ whose length (summing the edge σ's) approaches $+\infty$ (or $-\infty$). This fact naturally induces the general definition of unidirectionality and bidirectionality of RNNs as follows:

Definition 4.2.1.3. *An RNN is called unidirectional if its cyclic graph representation $\mathcal{G}_c$ has the property that the sums of σ along all the directed cycles ϑ in*

$\mathcal{G}_c$ share the same sign, i.e., either all positive or all negative. An RNN is called bidirectional if it is not unidirectional.

4.2.2 A General Definition of RNN

The connecting architecture in Sec. 4.2.1 describes how information flows among RNN units. Assume $\bar{v} \in V_c$ is a node in $\mathcal{G}_c$, let $\mathrm{In}(\bar{v})$ denotes the set of incoming nodes of $\bar{v}$, $\mathrm{In}(\bar{v}) = \{\bar{u} | (\bar{u}, \bar{v}) \in E_c\}$. In the forward pass of the RNN, the transition function $F_{\bar{v}}$ takes outputs of nodes $\mathrm{In}(\bar{v})$ as inputs and computes a new output. For example, vanilla RNNs units with different activation functions, LSTMs and GRUs can all be viewed as units with specific transition functions. We now give the general definition of an RNN:

Definition 4.2.2.1. *An RNN is a tuple $(\mathcal{G}_c, \mathcal{G}_{\mathrm{un}}, \{F_{\bar{v}}\}_{\bar{v} \in V_c})$, in which $\mathcal{G}_{\mathrm{un}} = (V_{un}, E_{un})$ is the unfolding of RNN cyclic graph $\mathcal{G}_c$, and $\{F_{\bar{v}}\}_{\bar{v} \in V_c}$ is the set of transition functions. In the forward pass, for each hidden and output node $v \in V_{un}$, the transition function $F_{\bar{v}}$ takes all incoming nodes of v as the input to compute the output.*

An RNN is *homogeneous* if all the hidden nodes share the same form of the transition function.

4.3 Measures of Architectural Complexity

In this section, we develop different measures of RNNs' architectural complexity, focusing mostly on the graph-theoretic properties of RNNs. To analyze an RNN solely from its architectural aspect, we make the mild assumption that the RNN is homogeneous. We further assume the RNN to be unidirectional. For a bidirectional RNN, it is more natural to measure the complexities of its unidirectional components.

4.3.1 Recurrent Depth

Unlike feedforward models where computations are done within one time frame, RNNs map inputs to outputs over multiple time steps. In some sense, an RNN undergoes transformations along both feedforward and recurrent dimensions. This fact suggests that we should investigate its architectural complexity from these two different perspectives. We first consider the recurrent perspective.

The conventional definition of depth is the *maximum* number of nonlinear transformations from inputs to outputs. Observe that a directed path in an unfolded graph representation G_{un} corresponds to a sequence of nonlinear transformations. Given an unfolded RNN graph G_{un}, $\forall i, n \in \mathbb{Z}$, let $\mathfrak{D}_i(n)$ be the length of the *longest* path from any node at starting time i to any node at time $i+n$.

From the recurrent perspective, it is natural to investigate how $\mathfrak{D}_i(n)$ changes over time. Generally speaking, $\mathfrak{D}_i(n)$ increases as n increases for all i. Such increase is caused by the recurrent structure of the RNN which keeps adding new nonlinearities over time. Since $\mathfrak{D}_i(n)$ approaches ∞ as n approaches ∞,[3] to measure the complexity of $\mathfrak{D}_i(n)$, we consider its asymptotic behaviour, i.e., the limit of $\frac{\mathfrak{D}_i(n)}{n}$ as $n \to \infty$. Under a mild assumption, this limit exists. To perform a practical calculation of this limit, the next theorem relies on $\mathcal{G}_{\text{un}}$'s cyclic counterpart $\mathcal{G}_c$, where the computation is much easier:

Theorem 4.3.1.1 (Recurrent Depth). *Given an RNN and its two graph representation $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$, we denote $C(\mathcal{G}_c)$ to be the set of directed cycles in $\mathcal{G}_c$. For $\vartheta \in C(\mathcal{G}_c)$, let $l(\vartheta)$ denote the length of ϑ and $\sigma_s(\vartheta)$ denote the sum of edge weights σ along ϑ. Under a mild assumption*[4]*,*

$$d_r = \lim_{n\to+\infty} \frac{\mathfrak{D}_i(n)}{n} = \max_{\vartheta\in C(\mathcal{G}_c)} \frac{l(\vartheta)}{\sigma_s(\vartheta)}. \tag{4.1}$$

Thus, d_r is a positive rational number.

More intuitively, d_r is a measure of the average maximum number of nonlinear transformations per time step as n gets large. Thus, we call it *recurrent depth*:

Definition 4.3.1.1 (Recurrent Depth). *Given an RNN and its two graph representations $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$, we call d_r, defined in Eq.(4.1), the recurrent depth of the RNN.*

In Figure 4.1(a), one can easily verify that $\mathfrak{D}_t(1) = 5$, $\mathfrak{D}_t(2) = 6$, $\mathfrak{D}_t(3) = 8$, $\mathfrak{D}_t(4) = 9 \ldots$ Thus $\frac{\mathfrak{D}_t(1)}{1} = 5$, $\frac{\mathfrak{D}_t(2)}{2} = 3$, $\frac{\mathfrak{D}_t(3)}{3} = \frac{8}{3}$, $\frac{\mathfrak{D}_t(4)}{4} = \frac{9}{4}$ …., which eventually

---

[3] Without loss of generality, we assume the unidirectional RNN approaches positive infinity.

[4] See a full treatment of the limit in general cases in Theorem 4.6.0.1 and Proposition 4.6.0.1 in Section 4.6.

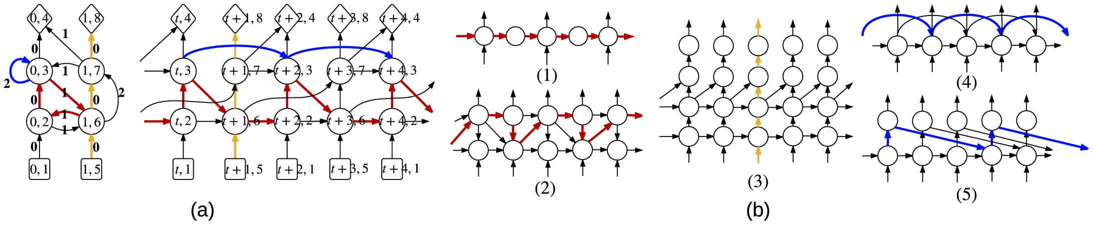


Figure 4.1 – (a) An example of an RNN's $\mathcal{G}_c$ and $\mathcal{G}_{\text{un}}$. V_{in} is denoted by square, V_{hid} is denoted by circle and V_{out} is denoted by diamond. In $\mathcal{G}_c$, the number on each edge is its corresponding σ. The longest path is colored in red. The longest input-output path is colored in yellow and the shortest path is colored blue. The value of three measures are $d_r = \frac{3}{2}$, $d_f = 3.5$ and $s = 2$. (b) 5 more examples. (1) and (2) have $d_r = 2, \frac{3}{2}$, (3) has $d_f = 5$, (4) and (5) has $s = 2, \frac{3}{2}$.

converges to $\frac{3}{2}$ as $n \to \infty$. As n increases, most parts of the longest path coincides with the path colored in red. As a result, d_r coincides with the number of nodes the red path goes through per time step. Similarly in $\mathcal{G}_c$, observe that the red cycle achieves the maximum ($\frac{3}{2}$) in Eq.(4.1). Usually, one can directly calculate d_r from $\mathcal{G}_{\text{un}}$.

It is easy to verify that *simple RNNs* and *stacked RNNs* share the same recurrent depth which is equal to 1. This reveals the fact that their nonlinearities increase at the same rate, which suggests that they will behave similarly in the long run. This fact is often neglected, since one would typically consider the number of layers as a measure of depth, and think of stacked RNNs as "deep" and simple RNNs as "shallow", even though their discrepancies are not due to recurrent depth (which regards time) but due to feedforward depth, defined next.

4.3.2 Feedforward Depth

Recurrent depth does not fully characterize the nature of nonlinearity of an RNN. As previous work suggests (Sutskever et al., 2014), stacked RNNs do outperform shallow ones with the same hidden size on problems where a more immediate input and output process is modeled. This is not surprising, since the growth rate of $\mathfrak{D}_i(n)$ only captures the number of nonlinear transformations in the time direction, not in the feedforward direction.

The perspective of feedforward computation puts more emphasis on the specific paths connecting inputs to outputs. Given an RNN unfolded graph G_{un}, let $\mathfrak{D}_i^*(n)$ be the length of the longest path from any input node at time step i to any output node at time step $i+n$. Clearly, when n is small, the recurrent depth cannot serve as a good description for $\mathfrak{D}_i^*(n)$. In fact. it heavily depends on another quantity which we call *feedforward depth.* The following proposition guarantees the existence of such a quantity and demonstrates the role of both measures in quantifying the nonlinearity of an RNN.

Proposition 4.3.2.1 (**Input-Output Length Least Upper Bound**). Given an RNN with recurrent depth d_r, we denote

$$d_f = \sup_{i,n \in \mathbb{Z}} \mathfrak{D}_i^*(n) - n \cdot d_r. \tag{4.2}$$

The supremum d_f exists and thus we have the following upper bound for $\mathfrak{D}_i^*(n)$

$$\mathfrak{D}_i^*(n) \leq n \cdot d_r + d_f.$$

The above upper bound explicitly shows the interplay between recurrent depth and feedforward depth: when n is small, $\mathfrak{D}_i^*(n)$ is largely bounded by d_f; when n is large, d_r captures the nature of the bound ($\approx n \cdot d_r$). These two measures are equally important, as they separately capture the maximum number of nonlinear transformations of an RNN in the long run and in the short run.

Definition 4.3.2.1. (**Feedforward Depth**) *Given an RNN with recurrent depth* d_r *and its two graph representations* $\mathcal{G}_{\text{un}}$ *and* $\mathcal{G}_c$*, we call* d_f*, defined in Eq.(4.2), the feedforward depth*[5] *of the RNN.*

To calculate d_f in practice, we introduce the following theorem:

Theorem 4.3.2.1 (**Feedforward Depth**). *Given an RNN and its two graph representations* $\mathcal{G}_{\text{un}}$ *and* $\mathcal{G}_c$*, we denote* $\xi(\mathcal{G}_c)$ *the set of directed paths that start at an input node and end at an output node in* $\mathcal{G}_c$*. For* $\gamma \in \xi(\mathcal{G}_c)$*, denote* $l(\gamma)$ *the length and* $\sigma_s(\gamma)$ *the sum of* σ *along* γ*. Then we have:*

$$d_f = \sup_{i,n \in \mathbb{Z}} \mathfrak{D}_i^*(n) - n \cdot d_r = \max_{\gamma \in \xi(\mathcal{G}_c)} l(\gamma) - \sigma_s(\gamma) \cdot d_r,$$

---

[5] Conventionally, an architecture with depth 1 is a three-layer architecture containing one hidden layer. But in our definition, since it goes through two transformations, we count the depth as 2 instead of 1. This should be particularly noted with the concept of feedforward depth, which can be thought as the conventional depth plus 1.

where m is the period number and d_r is the recurrent depth of the RNN. Thus, d_f is a postive rational number.

For example, in Figure 4.1(a), one can easily verify that $d_f = \mathfrak{D}_t^*(0) = 3$. Most commonly, d_f is the same as $\mathfrak{D}_t^*(0)$, i.e., the maximum length from an input to its current output.

4.3.3 Recurrent Skip Coefficient

Depth provides a measure of the complexity of the model. But such a measure is not sufficient to characterize behavior on long-term dependency tasks. In particular, since models with large recurrent depths have more nonlinearities through time, gradients can explode or vanish more easily. On the other hand, it is known that adding skip connections across multiple time steps may help improve the performance on long-term dependency problems (Lin et al. (1996); Sutskever and Hinton (2010)). To measure such a "skipping" effect, we should instead pay attention to the length of the *shortest path* from time i to time $i+n$. In G_{un}, $\forall i, n \in \mathbb{Z}$, let $\mathfrak{d}_i(n)$ be the length of the shortest path. Similar to the recurrent depth, we consider the growth rate of $\mathfrak{d}_i(n)$.

Theorem 4.3.3.1 (Recurrent Skip Coefficient). *Given an RNN and its two graph representations $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$, under mild assumptions*[6]

$$j = \lim_{n\to+\infty} \frac{\mathfrak{d}_i(n)}{n} = \min_{\vartheta\in C(\mathcal{G}_c)} \frac{l(\vartheta)}{\sigma_s(\vartheta)}. \tag{4.3}$$

Thus, j is a positive rational number.

Since it is often the case that j is smaller or equal to 1, it is more intuitive to consider its reciprocal.

Definition 4.3.3.1. (Recurrent Skip Coefficient)[7]. *Given an RNN and its two graph representations $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$, we define $s = \frac{1}{j}$, whose reciprocal is defined in Eq.(4.3), as the recurrent skip coefficient of the RNN.*

---

[6] See Proposition 4.6.0.3 in Section 4.6.

[7] One would find this definition very similar to the definition of the recurrent depth. Therefore, we refer readers to examples in Figure 4.1 for some illustrations.

With a larger recurrent skip coefficient, the number of transformations per time step is smaller. As a result, the nodes in the RNN are more capable of "skipping" across the network, allowing unimpeded information flow across multiple time steps, thus alleviating the problem of learning long term dependencies. In particular, such effect is more prominent in the long run, due to the network's recurrent structure. Also note that not all types of skip connections can increase the recurrent skip coefficient. We will consider specific examples in our experimental results section.

4.4 Experiments and Results

In this section we conduct a series of experiments to investigate the following questions: (1) Is recurrent depth a trivial measure? (2) Can increasing depth yield performance improvements? (3) Can increasing the recurrent skip coefficient improve the performance on long term dependency tasks? (4) Does the recurrent skip coefficient suggest something more compared to simply adding skip connections? We first show evaluations on RNNs with *tanh* nonlinearities, and then present similar results for LSTMs.

4.4.1 Tasks and Training Settings

PennTreebank dataset: We evaluate our models on character level language modelling using the PennTreebank dataset Marcus et al. (1993). It contains 5059k characters for training, 396k for validation and 446k for test, and has a alphabet size of 50. We set each training sequence to have the length of 50. Quality of fit is evaluated by the bits-per-character (BPC) metric, which is $\log_2$ of perplexity.

text8 dataset: Another dataset used for character level language modelling is the text8 dataset[8], which contains $100M$ characters from Wikipedia with an alphabet size of 27. We follow the setting from Mikolov et al. (2012) and each training sequence has length of 180.

adding problem: The adding problem (and the following copying memory problem) was introduced in Hochreiter and Schmidhuber (1997). For the adding problem, each input has two sequences with length of T where the first sequence

---

[8] http://mattmahoney.net/dc/textdata.

are numbers sampled from uniform[0, 1] and the second sequence are all zeros except two elements which indicates the position of the two elements in the first sequence that should be summed together. The output is the sum. We follow the most recent results and experimental settings in Arjovsky et al. (2015) (same for copying memory).

copying memory problem: Each input sequence has length of $T + 20$, where the first 10 values are random integers between 1 to 8. The model should remember them after T steps. The rest of the sequence are all zeros, except for the last 11 entries in the sequence, which starts with 9 as a marker indicating that the model should begin to output its memorized values. The model is expected to give zero outputs at every time step except the last 10 entries, where it should generate (copy) the 10 values in the same order as it has seen at the beginning of the sequence. The goal is to minimize the average cross entropy of category predictions at each time step.

sequential MNIST dataset: Each MNIST image data is reshaped into a 784×1 sequence, turning the digit classification task into a sequence classification one with long-term dependencies (Le et al., 2015; Arjovsky et al., 2015). A slight modification of the dataset is to permute the image sequences by a fixed random order beforehand (permuted MNIST). Results in Le et al. (2015) have shown that both *tanh* RNNs and LSTMs did not achieve satisfying performance, which also highlights the difficulty of this task.

For all of our experiments we use *Adam* (Kingma and Ba, 2014) for optimization, and conduct a grid search on the learning rate in $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. For *tanh* RNNs, the parameters are initialized with samples from a uniform distribution. For LSTM networks we adopt a similar initialization scheme, while the forget gate biases are chosen by the grid search on $\{-5, -3, -1, 0, 1, 3, 5\}$. We employ early stopping and the batch size was set to 50.

4.4.2 Recurrent Depth is Non-trivial

To investigate the first question, we compare 4 similar connecting architectures: 1-layer (shallow) "*sh*", 2-layers stacked "*st*", 2-layers stacked with an extra bottom-up connection "*bu*", and 2-layers stacked with an extra top-down connection "*td*", as shown in Figure 4.2 (a). Although the four architectures look quite similar, they

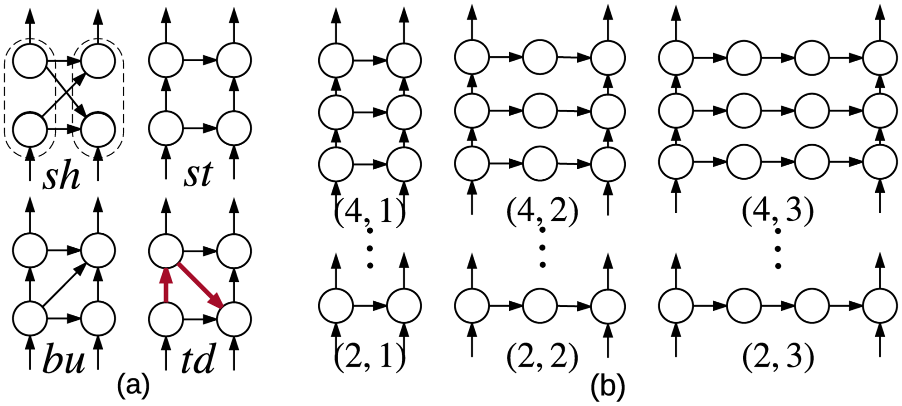


Figure 4.2 – (a) The architectures for *sh*, *st*, *bu* and *td*, with their (d_r, d_f) equal to $(1, 2)$, $(1, 3)$, $(1, 3)$ and $(2, 3)$, respectively. The longest path in *td* are colored in red. (b) The 9 architectures denoted by their (d_f, d_r) with $d_r = 1, 2, 3$ and $d_f = 2, 3, 4$. We only plot the hidden states within 1 time step (which also have a period of 1) in both (a) and (b).

have different recurrent depths: *sh*, *st* and *bu* have $d_r = 1$, while *td* has $d_r = 2$. Note that the specific construction of the extra nonlinear transformations in *td* is not conventional. Instead of simply adding intermediate layers in hidden-to-hidden connection, as reported in Pascanu et al. (2013a), more nonlinearities are gained by a recurrent flow from the first layer to the second layer and then back to the first layer at each time step (see the red path in Figure 4.2 (a)).

We first evaluate our architectures using *tanh* RNN on PennTreebank, where *sh* has hidden-layer size of 1600. Next, we evaluate four different models for text8 which are *tanh* RNN-small, *tanh* RNN-large, LSTM-small, LSTM large, where the model's *sh* architecture has hidden-layer size of 512, 2048, 512, 1024 respectively. Given the architecture of the *sh* model, we set the remaining three architectures to have the same number of parameters.

Table 4.1 shows that the *td* architecture outperforms all the other architectures

DATASET	MODELS, ARCHS	*sh*	*st*	*bu*	*td*
PennTreebank	*tanh* RNN	1.54	1.59	1.54	**1.49**
text8	*tanh* RNN-SMALL	1.80	1.82	1.80	**1.77**
	tanh RNN-LARGE	1.69	1.67	1.64	**1.59**
	LSTM-SMALL	1.65	1.66	1.65	**1.63**
	LSTM-LARGE	1.52	1.53	1.52	**1.49**

Table 4.1 – Test BPCs of *sh*, *st*, *bu*, *td* for *tanh* RNNs and LSTMs.

for all the different models. Specifically, *td* in *tanh* RNN achieves a test BPC of 1.49, which is comparable to the BPC of 1.48 reported in Krueger and Memisevic (2015) using stabilization techniques. Similar improvements are shown for LSTMs, where *td* architecture in LSTM-large achieves BPC of 1.49, outperforming the BPC of 1.54 reported in Mikolov et al. (2012) with MRNN (Multiplicative RNN).

It is also interesting to note the improvement we obtain when switching from *bu* to *td*. The only difference between these two architectures lies in changing the direction of one connection (see Figure 4.2 (a)), which also increases the recurrent depth. Such a fundamental difference is by no means self-evident, but this result highlights the necessity of the concept of recurrent depth.

4.4.3 Comparing Depths

From the previous experiment, we found some evidence that with larger recurrent depth, the performance might improve. To further investigate various implications of depths, we carry out a systematic analysis for both recurrent depth d_r and feedforward depth d_f on *text8* and sequential MNIST datasets. We build 9 models in total with $d_r = 1, 2, 3$ and $d_f = 2, 3, 4$, respectively (as shown in Figure 4.2 (b)). We ensure that all the models have roughly the same number of parameters (e.g., the model with $d_r = 1$ and $d_f = 2$ has a hidden-layer size of 360).

Table 4.2 displays results on the *text*8 dataset. We observed that for a fixed feedforward depth d_f, increasing the recurrent depth d_r does improve the model performance, and the best test BPC is achieved by the architecture with $d_f = 2$ and $d_r = 3$. This suggests that the increase of d_r can aid in better capturing the over-time nonlinearity of the input sequence. However, for a fixed d_r, increasing d_f only helps when $d_r = 1$. For a recurrent depth of $d_r = 3$, increasing d_f only hurts models performance. This can potentially be attributed to the optimization issues

d_f, d_r	$d_r = 1$	$d_r = 2$	$d_r = 3$
$d_f = 2$	1.88	1.84	**1.83**
$d_f = 3$	1.86	1.84	1.85
$d_f = 4$	1.94	1.89	1.88

Table 4.2 – Test BPCs of *tanh* RNNs with recurrent depth $d_r = 1, 2, 3$ and feedforward depth $d_f = 2, 3, 4$ respectively.

when modelling large input-to-output dependencies.

With sequential MNIST dataset, we next examined the effects of d_f and d_r when modelling long term dependencies. In particular, we observed that increasing d_f does not bring any improvement to the model performance, and increasing d_r might even be detrimental for training. Indeed, it appears that d_f only captures the local nonlinearity and has less effect on the long term prediction. This result seems to contradict previous claims (Hermans and Schrauwen, 2013) that stacked RNNs ($d_f > 1$, $d_r = 1$) could capture information in different time scales and would thus be more capable of dealing with learning long-term dependencies. On the other hand, a large d_r indicates multiple transformations per time step, resulting in greater gradient vanishing/exploding issues Pascanu et al. (2013a), which suggests that d_r should be neither too small nor too large.

4.4.4 Recurrent Skip Coefficients

To investigate whether increasing a recurrent skip coefficient s improves model performance on long term dependency tasks, we compare models with increasing s on the adding problem, the copying memory problem and the sequential MNIST problem (without/with permutation, denoted as MNIST and pMNIST). Our baseline model is the shallow architecture proposed in Le et al. (2015). To increase the recurrent skip coefficient s, we add connections from time step t to time step $t+k$ for some fixed integer k, shown in Figure 4.3 (a), right panel. By using this specific construction, the recurrent skip coefficient increases from 1 (i.e., baseline) to k and the new model with extra connection has 2 hidden matrices (one from t to $t+1$ and the other from t to $t+k$).

For the adding problem, we follow the same setting as in Arjovsky et al. (2015). We evaluate the baseline LSTM with 128 hidden units and an LSTM with $s = 30$ and 90 hidden units (roughly the same number of parameters as the baseline). The results are quite encouraging: as suggested in Arjovsky et al. (2015) baseline LSTM works well for input sequence lengths $T = 100, 200, 400$ but fails when $T = 750$. On the other hand, we observe that the LSTM with $s = 30$ learns perfectly when $T = 750$, and even if we increase T to 1000, LSTM with $s = 30$ still works well and the loss reaches to zero.

For the copying memory problem, we use a single layer RNN with 724 hidden units as our basic model, and 512 hidden units with skip connections. So they

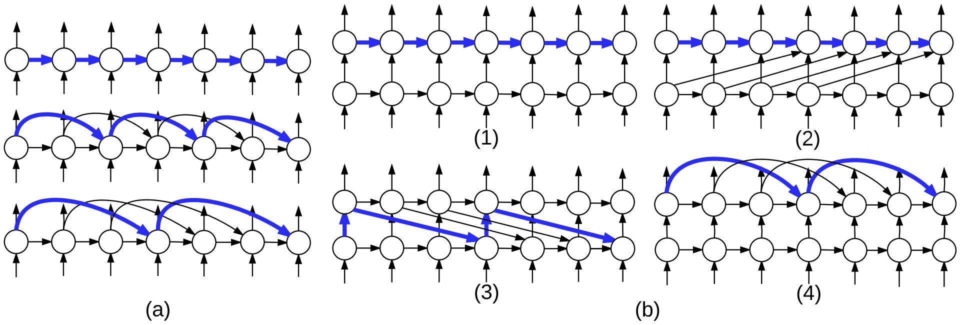


Figure 4.3 – (a) Various architectures that we consider in Section 4.4.4. From top to bottom are baseline $s = 1$, and $s = 2$, $s = 3$. (b) Proposed architectures that we consider in Section 4.4.5 where we take $k = 3$ as an example. The shortest paths in (a) and (b) that correspond to the recurrent skip coefficients are colored in blue.

have roughly the same number of parameters. Models with a higher recurrent skip coefficient outperform those without skip connections by a large margin. When $T = 200$, test set cross entropy (CE) of a basic model only yields 0.2409, but with $s = 40$ it is able to reach a test set cross entropy of 0.0975. When $T = 300$, a model with $s = 30$ yields a test set CE of 0.1328, while its baseline could only reach 0.2025. We varied the sequence length (T) and recurrent skip coefficient (s) in a wide range (where T varies from 100 up to 300, and s from 10 up to 50), and found that this kind of improvement persists.

For the sequential MNIST problem, the hidden-layer size of the baseline model is set to 90 and models with $s > 1$ have hidden-layer sizes of 64.

Results in Table 4.3 show that *tanh* RNNs with recurrent skip coefficient s larger than 1 could improve the model performance dramatically. Within a reasonable range of s, test accuracy increases quickly as s becomes larger. We note that our model is the first *tanh* RNN model that achieves good performance on this task,

RNN(*tanh*)	s = 1	s = 5	s = 9	s = 13	s = 21
MNIST	34.9	46.9	74.9	85.4	**87.8**
RNN(*tanh*)	s = 1	s = 3	s = 5	s = 7	s = 9
*p*MNIST	49.8	79.1	84.3	**88.9**	88.0
LSTM	s = 1	s = 3	s = 5	s = 7	s = 9
MNIST	56.2	**87.2**	86.4	86.4	84.8
LSTM	s = 1	s = 3	s = 4	s = 5	s = 6
*p*MNIST	28.5	25.0	60.8	62.2	**65.9**

Table 4.3 – Test accuracies with different s for *tanh* RNN and LSTM in MNIST/*p*MNIST.

Model	MNIST	pMNIST
iRNNLe et al. (2015)	97.0	≈82.0
uRNNArjovsky et al. (2015)	95.1	91.4
LSTMArjovsky et al. (2015)	**98.2**	88.0
RNN($tanh$)Le et al. (2015)	≈35.0	≈35.0
$s$tanh(s = 21, 11)	98.1	**94.0**

Table 4.4 – our best model compared to previous results on MNIST/pMNIST.

even improving upon the method proposed in Le et al. (2015), see Table 4.4. In addition, we also formally compare with the previous results reported in Le et al. (2015); Arjovsky et al. (2015), where our model (referred to as $s$tanh) has a hidden-layer size of 95, which is about the same number of parameters as in the $tanh$ model of Arjovsky et al. (2015). Table 4.4 shows that our simple architecture improves upon the uRNN by 2.6% on pMNIST, and achieves almost the same performance as LSTM on the MNIST dataset with only 25% number of parameters (Arjovsky et al., 2015). Note that obtaining good performance on sequential MNIST requires a larger s than that for pMNIST. LSTMs also showed performance boost and much faster convergence speed when using larger s, as displayed in Table 4.3, top panel. LSTM with $s = 3$ already performs quite well and increasing s did not result in any significant improvement, while in pMNIST, the performance gradually improves as s increases from 4 to 6. We also observed that the LSTM network performed worse on permuted MNIST compared to a $tanh$ RNN. Similar result was also reported in Le et al. (2015).

4.4.5 Recurrent Skip Coefficients vs. Skip Connections

We also investigated whether the recurrent skip coefficient can suggest something more than simply adding skip connections. We design 4 specific architectures shown in Figure 4.3 (b). (1) is the baseline model with a 2-layer stacked architecture, while the other three models add extra skip connections in different ways. Note that these extra skip connections all cross the same time length k. In particular, (2) and (3) share quite similar architectures. However, the way in which the skip connections are allocated make a big difference on their recurrent skip coefficients: (2) has $s = 1$, (3) has $s = \frac{k}{2}$ and (4) has $s = k$. Therefore, even though (2), (3) and (4) all add extra skip connections, the fact that their recurrent skip

Architecture, s		(1), 1	(2), 1	(3), $\frac{k}{2}$	(4), k
MNIST	k = 17	39.5	39.4	54.2	**77.8**
	k = 21	39.5	39.9	69.6	**71.8**
*p*MNIST	k = 5	55.5	66.6	74.7	**81.2**
	k = 9	55.5	71.1	78.6	**86.9**

Table 4.5 – Test accuracies for architectures (1), (2), (3) and (4) for *tanh* RNN on MNIST/*p*MNIST.

coefficients are different might result in different performance.

We evaluated these architectures on the sequential MNIST and *p*MNIST datasets. The results show that differences in s indeed cause big performance gaps regardless of the fact that they all have skip connections, see Table 4.5. Given the same k, the model with a larger s performs better. In particular, model (3) is better than model (2) even though they only differ in the direction of the skip connections. It is interesting to see that for MNIST (unpermuted), the extra skip connection in model (2) (which does not really increase the recurrent skip coefficient) brings almost no benefits, as model (2) and model (1) have almost the same results. This observation highlights the following point: when addressing the long term dependency problems using skip connections, instead of only considering the time intervals crossed by the skip connection, one should also consider the model's recurrent skip coefficient, which can serve as a guide for introducing more powerful skip connections.

4.5 Conclusion

In this work, we first introduced a general formulation of RNN architectures, which allows one to construct more general RNNs, and provides a solid framework for the architectural complexity analysis. We then proposed three architectural complexity measures: recurrent depth, feedforward depth, and recurrent skip coefficients, each capturing the complexity in the long term, complexity in the short term and the speed of information flow. We also find empirical evidence that increasing recurrent depth might yield performance improvements, increasing feedforward depth might not help on long term dependency tasks, while increasing the recurrent skip coefficient can largely improve performance on long term dependency tasks. These measures and results can provide guidance for the design of new recurrent architectures for a particular learning task. Future work could involve more comprehensive studies (e.g., providing analysis on more datasets, using different architectures with various transition functions) to investigate the effectiveness of the proposed measures.

4.6 Proofs

To show theorem 4.3.1.1, we first consider the most general case in which d_r is defined (Theorem 4.6.0.1). Then we discuss the mild assumptions under which we can reduce to the original limit (Proposition 4.6.0.1). Additionally, we introduce some notations that will be used throughout the proof. If $v = (t, p) \in \mathcal{G}_{\text{un}}$ is a node in the unfolded graph, it has a corresponding node in the folded graph, which is denoted by $\bar{v} = (\bar{t}, p)$.

Theorem 4.6.0.1. *Given an RNN cyclic graph and its unfolded representation* $(\mathcal{G}_c, \mathcal{G}_{\text{un}})$*, we denote* $C(\mathcal{G}_c)$ *the set of directed cycles in* $\mathcal{G}_c$*. For* $\vartheta \in C(\mathcal{G}_c)$*, denote* $l(\vartheta)$ *the length of* ϑ *and* $\sigma_s(\vartheta)$ *the sum of* σ *along* ϑ*. Write* $d_i = \limsup_{k\to\infty} \frac{\mathfrak{D}_i(n)}{n}$.[9] *we have :*

- *The quantity* d_i *is periodic, in the sense that* $d_{i+m} = d_i, \forall i \in \mathbb{N}$.
- *Let* $d_r = \max_i d_i$*, then*

$$d_r = \max_{\vartheta \in C(\mathcal{G}_c)} \frac{l(\vartheta)}{\sigma_s(\vartheta)} \tag{4.4}$$

Proof. The first statement is easy to prove. Because of the periodicity of the graph, any path from time step i to $i + n$ corresponds to an isomorphic path from time step $i + m$ to $i + m + n$. Passing to limit, and we can deduce the first statement.

Now we prove the second statement. Write $\vartheta_0 = \text{argmax}_\vartheta \frac{l(\vartheta)}{\sigma_s(\vartheta)}$. First we prove that $d \geq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)}$. Let $c_1 = (t_1, p_1) \in \mathcal{G}_{\text{un}}$ be a node such that if we denote $\overline{c_1} = (\overline{t_1}, p_1)$ the image of c_1 on the cyclic graph, we have $\overline{c_1} \in \vartheta_0$. Consider the subsequence $S_0 = \left\{\frac{\mathfrak{D}_{\overline{t_1}}(k\sigma_s(\vartheta_0))}{k\sigma_s(\vartheta_0)}\right\}_{k=1}^{\infty}$ of $\left\{\frac{\mathfrak{D}_{\overline{t_1}}(n)}{n}\right\}_{n=1}^{\infty}$. From the definition of $\mathfrak{D}$ and the fact that

---

[9] $\mathfrak{D}_i(n)$ is not defined when there does not exist a path from time i to time $i+n$. We simply omit undefined cases when we consider the limsup. In a more rigorous sense, it is the limsup of a subsequence of $\{\mathfrak{D}_i(n)\}_{n=1}^{\infty}$, where $\mathfrak{D}_i(n)$ is defined.

ϑ_0 is a directed circle, we have $\mathfrak{D}_{\overline{t_1}}(k\sigma_s(\vartheta_0)) \geq kl(\vartheta_0)$, by considering the path on $\mathcal{G}_{\text{un}}$ corresponding to following ϑ_0 k -times. So we have

$$d_r \geq \limsup_{k\to+\infty} \frac{\mathfrak{D}_i(n)}{n} \geq \limsup_{k\to+\infty} \frac{\mathfrak{D}_{\overline{t_1}}(k\sigma_s(\vartheta_0))}{k\sigma_s(\vartheta_0)} \geq \frac{kl(\vartheta_0)}{k\sigma_s(\vartheta_0)} = \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)}$$

Next we prove $d_r \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)}$. It suffices to prove that, for any $\epsilon \geq 0$, there exists $N > 0$, such that for any path $\gamma : \{(t_0, p_0), (t_1, p_1), \cdots, (t_{n_\gamma}, p_{n_\gamma})\}$ with $t_{n_\gamma} - t_1 > N$, we have $\frac{n_\gamma}{t_{n_\gamma} - t_1} \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)} + \epsilon$. We denote $\bar{\gamma}$ as the image of γ on the cyclic graph. $\bar{\gamma}$ is a walk with repeated nodes and edges. Also, we assume there are in total Γ nodes in cyclic graph $\mathcal{G}_c$.

We first decompose $\bar{\gamma}$ into a path and a set of directed cycles. More precisely, there is a path γ_0 and a sequence of directed cycles $C = C_1(\gamma), C_2(\gamma), \cdots, C_w(\gamma)$ on $\mathcal{G}_c$ such that:

— The starting and end nodes of γ_0 is the same as γ. (If γ starts and ends at the same node, take γ_0 as empty.)

— The catenation of the sequences of directed edges $E(\gamma_0), E(C_1(\gamma)), E(C_2(\gamma)), \cdots, E(C_w(\gamma))$ is a permutation of the sequence of edges of $E(\gamma)$.

The existence of such a decomposition can be proved iteratively by removing directed cycles from γ. Namely, if γ is not a paths, there must be some directed cycles C' on γ. Removing C' from γ, we can get a new walk γ'. Inductively apply this removal, we will finally get a (possibly empty) path and a sequence of directed cycles. For a directed path or loop γ, we write $D(\gamma)$ the distance between the ending node and starting node when travel through γ once. We have

$$D(\gamma_0) := \overline{t_{n_\gamma}} - \overline{t_0} + \sum_{i=1}^{|\gamma_0|} \sigma(e_i)$$

where $e_i, i \in \{1, 2, \cdots, |\gamma_0|\}$ is all the edges of γ_0. $\bar{t}$ denotes the module of t: $t \equiv \bar{t} (\operatorname{mod} m)$.

So we have:

$$|D(\gamma_0)| \leq m + \Gamma \cdot \max_{e\in\mathcal{G}_c} \sigma(e) = M$$

For convenience, we denote $l_0, l_1, \cdots, l_w$ to be the length of path γ_0 and directed cycles $C_1(\gamma), C_2(\gamma), \cdots, C_w(\gamma)$. Obviously we have:

$$n_\gamma = \sum_{i=0}^{w} l_i$$

And also, we have

$$t_{n_\gamma} - t_1 = \sum_{i=1}^{w} \sigma_s(C_i) + D(\gamma_0)$$

So we have:

$$\frac{n_\gamma}{t_{n_\gamma} - t_1} = \frac{l_0}{t_{n_\gamma} - t_1} + \sum_{i=1}^{w} \frac{l_i}{t_{n_\gamma} - t_1} \leq \frac{\Gamma}{N} + \sum_{i=1}^{w} \frac{l_i}{t_{n_\gamma} - t_1}$$

In which we have for all $i \in \{1, 2, \cdots, w\}$:

$$\frac{l_i}{t_{n_\gamma} - t_1} = \frac{l_i}{\sigma_s(C_i)} \cdot \frac{\sigma_s(C_i)}{t_{n_\gamma} - t_1} \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)} \frac{\sigma_s(C_i)}{t_{n_\gamma} - t_1}$$

So we have:

$$\sum_{i=1}^{w} \frac{l_i}{t_{n_\gamma} - t_1} \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)} \left[1 - \frac{D(\gamma_0)}{t_{n_\gamma} - t_1}\right] \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)} + \frac{M'}{N}$$

in which M' and Γ are constants depending only on the RNN $\mathcal{G}_c$.

Finally we have:

$$\frac{n_\gamma}{t_{n_\gamma} - t_1} \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)} + \frac{M' + \Gamma}{N}$$

take $N > \frac{M'+\Gamma}{\epsilon}$, we can prove the fact that $d_r \leq \frac{l(\vartheta_0)}{\sigma_s(\vartheta_0)}$. □

Proposition 4.6.0.1. *Given an RNN and its two graph representations $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$, if $\exists \vartheta \in C(\mathcal{G}_c)$ such that* (1) *ϑ achieves the maximum in Eq.(4.4)* and (2) *$U(\vartheta)$ in $\mathcal{G}_{\text{un}}$ visits nodes at every time step, then we have*

$$d_r = \max_{i \in \mathbb{Z}} \left(\limsup_{n \to +\infty} \frac{\mathfrak{D}_i(n)}{n} \right) = \lim_{n \to +\infty} \frac{\mathfrak{D}_i(n)}{n}$$

Proof. We only need to prove, in such a graph, for all $i \in \mathbb{Z}$ we have

$$\liminf_{n \to +\infty} \frac{\mathfrak{D}_i(n)}{n} \geq \max_{i \in \mathbb{Z}} \left(\limsup_{n \to +\infty} \frac{\mathfrak{D}_i(n)}{n} \right) = d_r$$

Because it is obvious that

$$\text{liminf}_{n\to+\infty}\frac{\mathfrak{D}_i(n)}{n}\leq d_r$$

Namely, it suffice to prove, for all $i\in\mathbb{Z}$, for all $\epsilon>0$, there is an $N_\epsilon>0$, such that when $n>N_\epsilon$, we have $\frac{\mathfrak{D}_i(n)}{n}\geq d_r-\epsilon$. On the other hand, for $k\in\mathbb{N}$, if we assume $(k+1)\sigma_s(\vartheta)+i>n\geq i+k\cdot\sigma_s(\vartheta)$, then according to condition (2) we have

$$\frac{\mathfrak{D}_i(n)}{n}\geq\frac{k\cdot l(\vartheta)}{(k+1)\sigma_s(\vartheta)}=\frac{l(\vartheta)}{\sigma_s(\vartheta)}-\frac{l(\vartheta)}{\sigma_s(\vartheta)}\frac{1}{k+1}$$

We can see that if we set $k>\frac{\sigma_s(\vartheta)}{l(\vartheta)\epsilon}$, the inequality we wanted to prove. □

We now prove Proposition 4.3.2.1 and Theorem 4.3.2.1 as follows.

Proposition 4.6.0.2. *Given an RNN with recurrent depth d_r, we denote*

$$d_f=\sup_{i,n\in\mathbb{Z}}\mathfrak{D}_i^*(n)-n\cdot d_r.$$

The supremum d_f exists and we have the following least upper bound:

$$\mathfrak{D}_i^*(n)\leq n\cdot d_r+d_f.$$

Proof. We first prove that $d_f<+\infty$. Write $d_f(i)=\sup_{n\in\mathbb{Z}}\mathfrak{D}_i^*(n)-n\cdot d_r$. It is easy to verify $d_f(\cdot)$ is $m-$periodic, so it suffices to prove for each $i\in\mathbb{N}$, $d_f(i)<+\infty$. Hence it suffices to prove

$$\lim_{n\to\infty}\sup(\mathfrak{D}_i^*(n)-n\cdot d_r)<+\infty.$$

From the definition, we have $\mathfrak{D}_i(n)\geq\mathfrak{D}_i^*(n)$. So we have

$$\mathfrak{D}_i^*(n)-n\cdot d_r\leq\mathfrak{D}_i(n)-n\cdot d_r.$$

From the proof of Theorem 4.6.0.1, there exists two constants M' and Γ depending only on the RNN $\mathcal{G}_c$, such that

$$\frac{\mathfrak{D}_i(n)}{n}\leq d_r+\frac{M'+\Gamma}{n}.$$

So we have

$$\limsup_{n\to\infty}(\mathfrak{D}_i^*(n) - n \cdot d_r) \leq \limsup_{n\to\infty}(\mathfrak{D}_i(n) - n \cdot d_r) \leq M' + \Gamma.$$

Also, we have $d_f = \sup_{i,n\in\mathbb{Z}} \mathfrak{D}_i^*(n) - n \cdot d_r$, so for any $i, n \in \mathbb{Z}$,

$$d_f \geq \mathfrak{D}_i^*(n) - n \cdot d_r.$$

□

Theorem 4.6.0.2. *Given an RNN and its two graph representations $\mathcal{G}_{\text{un}}$ and $\mathcal{G}_c$, we denote $\xi(\mathcal{G}_c)$ the set of directed path that starts at an input node and ends at an output node in $\mathcal{G}_c$. For $\gamma \in \xi(\mathcal{G}_c)$, denote $l(\gamma)$ the length and $\sigma_s(\gamma)$ the sum of σ along γ. Then we have:*

$$d_f = \sup_{i,n\in\mathbb{Z}} \mathfrak{D}_i^*(n) - n \cdot d_r = \max_{\gamma\in\xi(\mathcal{G}_c)} l(\gamma) - \sigma_s(\gamma) \cdot d_r.$$

Proof. Let $\gamma : \{(t_0, 0), (t_1, p_1), \cdots, (t_{n_\gamma}, p)\}$ be a path in $\mathcal{G}_{\text{un}}$ from an input node $(t_0, 0)$ to an output node (t_{n_γ}, p), where $t_0 = i$ and $t_{n_\gamma} = i + n$. We denote $\bar{\gamma}$ as the image of γ on the cyclic graph. From the proof of Theorem 4.6.0.1, for each $\bar{\gamma}$ in $\mathcal{G}_c$, we can decompose it into a path γ_0 and a sequence of directed cycles $C = C_1(\gamma), C_2(\gamma), \cdots, C_w(\gamma)$ on $\mathcal{G}_c$ satisfying those properties listed in Theorem 4.6.0.1. We denote $l_0, l_1, \cdots, l_w$ to be the length of path γ_0 and directed cycles $C_1(\gamma), C_2(\gamma), \cdots, C_w(\gamma)$. We know $\frac{l_k}{\sigma_s(C_k)} \leq d_r$ for all $k = 1, 2, \ldots, w$ by definition. Thus,

$$l_k \leq d_r \cdot \sigma_s(C_k)$$
$$\sum_{k=1}^{w} l_k \leq d_r \cdot \sum_{k=1}^{w} \sigma_s(C_k)$$

Note that $n = \sigma_s(\gamma_0) + \sum_{k=1}^{w} \sigma_s(C_k)$. Therefore,

$$\begin{aligned} l(\gamma) - n \cdot d_r &= l_0 + \sum_{k=1}^{w} l_k - n \cdot d_r \\ &\leq l_0 + d_r \cdot (\sum_{k=1}^{w} \sigma_s(C_k) - n) \\ &= l_0 - d_r \cdot \sigma_s(\gamma_0) \end{aligned}$$

for all time step i and all integer n. The above inequality suggests that in order to take the supremum over all paths in $\mathcal{G}_{\text{un}}$, it suffices to take the maximum over a directed path in $\mathcal{G}_c$. On the other hand, the equality can be achieved simply by choosing the corresponding path of γ_0 in $\mathcal{G}_{\text{un}}$. The desired conclusion then follows immediately. ☐

Lastly, we show Theorem 4.3.3.1.

Theorem 4.6.0.3. *Given an RNN cyclic graph and its unfolded representation* $(\mathcal{G}_c, \mathcal{G}_{\text{un}})$*, we denote* $C(\mathcal{G}_c)$ *the set of directed cycles in* $\mathcal{G}_c$*. For* $\vartheta \in C(\mathcal{G}_c)$*, denote* $l(\vartheta)$ *the length of* ϑ *and* $\sigma_s(\vartheta)$ *the sum of* σ *along* ϑ*. Write* $s_i = \liminf_{k\to\infty} \frac{\mathfrak{d}_i(n)}{n}$*. We have :*

- *The quantity* s_i *is periodic, in the sense that* $s_{i+m} = s_i, \forall i \in \mathbb{N}$*.*
- *Let* $s = \min_i s_i$*, then*

$$d_r = \min_{\vartheta \in C(\mathcal{G}_c)} \frac{l(\vartheta)}{\sigma_s(\vartheta)}. \tag{4.5}$$

Proof. The proof is essentially the same as the proof of the first theorem. So we omit it here. ☐

Proposition 4.6.0.3. *Given an RNN and its two graph representations* $\mathcal{G}_{\text{un}}$ *and* $\mathcal{G}_c$*, if* $\exists \vartheta \in C(\mathcal{G}_c)$ *such that* (1) ϑ *achieves the minimum in Eq.(4.5) and* (2) $U(\vartheta)$ *in* $\mathcal{G}_{\text{un}}$ *visits nodes at every time step, then we have*

$$s = \min_{i\in\mathbb{Z}} \left(\liminf_{n\to+\infty} \frac{\mathfrak{d}_i(n)}{n} \right) = \lim_{n\to+\infty} \frac{\mathfrak{d}_i(n)}{n}.$$

Proof. The proof is essentially the same as the proof of the Proposition 4.6.0.1. So we omit it here. ☐

5 Prologue to Second Article

5.1 Article Detail

On Multiplicative Integration with Recurrent Neural Networks. Yuhuai Wu [1], Saizheng Zhang [1] , Ying Zhang, Yoshua Bengio, Ruslan Salakhutdinov. Advances in Neural Information Processing Systems (NeurIPS), 2016.

Personal Contribution. I proposed the original idea of multiplying two different information flows in the transition function of RNN. Yuhuai Wu and I further developed the general form of the multiplicative integration and designed exploratory experiments for understanding its functionality. I conducted experiments on language modeling datasets, Ying Zhang was responsible for examining the module efficiency on speech recognition problems and Yuhuai Wu performed evaluation on training skip-thought vectors and reading comprehension tasks. Yoshua Bengio and Ruslan Salakhutdinov provided valuable advice on model evaluation.

5.2 Context

This work introduces a general and simple structural design called "Multiplicative Integration" (MI) to improve recurrent neural networks (RNNs). As a simple Hadamard product operation, MI changes the way in which information from difference sources flows and is integrated in the computational building block of an RNN, while introducing almost no extra parameters. The new structure can be easily embedded into many popular RNN models, including LSTMs and GRUs. We empirically analyze its learning behaviour and conduct evaluations on several

---

[1] Equal contribution.

tasks using different RNN models. Our experimental results demonstrate that Multiplicative Integration can provide a substantial performance boost over many of the existing RNN models.

5.3 Contributions

The introduced MI mechanism is one of the few improvements that directly touches the essential transition function of an RNN with (1) strong adaptability towards any recurrent models, (2) no extra computational cost as it brings almost no extra parameters and (3) no extra engineering beyond implementing the RNN model itself. In addition, the general form of MI is by design performing at least as well as the standard RNN transition function.

6 On Multiplicative Integration with Recurrent Neural Networks

6.1 Introduction

Recently there has been a resurgence of new structural designs for recurrent neural networks (RNNs) Chung et al. (2015); Kalchbrenner et al. (2015); Jozefowicz et al. (2015). Most of these designs are derived from popular structures including vanilla RNNs, Long Short Term Memory networks (LSTMs) Hochreiter and Schmidhuber (1997) and Gated Recurrent Units (GRUs) Cho et al. (2014). Despite of their varying characteristics, most of them share a common computational building block, described by the following equation:

$$\phi(\mathbf{W}\boldsymbol{x} + \mathbf{U}\boldsymbol{z} + \mathbf{b}), \tag{6.1}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{z} \in \mathbb{R}^m$ are state vectors coming from different information sources, $\mathbf{W} \in \mathbb{R}^{d\times n}$ and $\mathbf{U} \in \mathbb{R}^{d\times m}$ are state-to-state transition matrices, and $\boldsymbol{b}$ is a bias vector. This computational building block serves as a combinator for integrating information flow from the $\boldsymbol{x}$ and $\boldsymbol{z}$ by a sum operation "+", followed by a nonlinearity ϕ. We refer to it as the *additive building block*. Additive building blocks are widely implemented in various state computations in RNNs (e.g. hidden state computations for vanilla-RNNs, gate/cell computations of LSTMs and GRUs.

In this work, we propose an alternative design for constructing the computational building block by changing the procedure of information integration. Specifically, instead of utilizing sum operation "+", we propose to use the Hadamard product "$\odot$" to fuse $\mathbf{W}\boldsymbol{x}$ and $\mathbf{U}\boldsymbol{z}$:

$$\phi(\mathbf{W}\boldsymbol{x} \odot \mathbf{U}\boldsymbol{z} + \mathbf{b}) \tag{6.2}$$

The result of this modification changes the RNN from first order to second order Goudreau et al. (1994), while introducing no extra parameters. We call this kind of information integration design a form of *Multiplicative Integration*. The effect of multiplication naturally results in a gating type structure, in which $\mathbf{W}\boldsymbol{x}$ and $\mathbf{U}\boldsymbol{z}$ are the gates of each other. More specifically, one can think of the state-to-state computation $\mathbf{U}\boldsymbol{z}$ (where for example $\boldsymbol{z}$ represents the previous state) as dynamically rescaled by $\mathbf{W}\boldsymbol{x}$ (where for example $\boldsymbol{x}$ represents the input). Such rescaling does not exist in the additive building block, in which $\mathbf{U}\boldsymbol{z}$ is independent of $\boldsymbol{x}$. This relatively simple modification brings about advantages over the additive building block as it alters RNN's gradient properties, which we discuss in detail in the next section, as well as verify through extensive experiments.

In the following sections, we first introduce a general formulation of Multiplicative Integration. We then compare it to the additive building block on several sequence learning tasks, including character level language modelling, speech recognition, large scale sentence representation learning using a Skip-Thought model, and teaching a machine to read and comprehend for a question answering task. The experimental results (together with several existing state-of-the-art models) show that various RNN structures (including vanilla RNNs, LSTMs, and GRUs) equipped with Multiplicative Integration provide better generalization and easier optimization. Its main advantages include: (1) it enjoys better gradient properties due to the gating effect. Most of the hidden units are non-saturated; (2) the general formulation of Multiplicative Integration naturally includes the regular additive building block as a special case, and introduces almost no extra parameters compared to the additive building block; and (3) it is a drop-in replacement for the additive building block in most of the popular RNN models, including LSTMs and GRUs. It can also be combined with other RNN training techniques such as Recurrent Batch Normalization Cooijmans et al. (2016). We further discuss its relationship to existing models, including Hidden Markov Models (HMMs) Baum and Eagon (1967), second order RNNs Giles et al. (1991); Goudreau et al. (1994) and Multiplicative RNNs Sutskever et al. (2011).

6.2 Structure Description and Analysis

6.2.1 General Formulation of Multiplicative Integration

The key idea behind Multiplicative Integration is to integrate different information flows $\mathbf{W}\boldsymbol{x}$ and $\mathbf{U}\boldsymbol{z}$, by the Hadamard product "$\odot$". A more general formulation of Multiplicative Integration includes two more bias vectors $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ added to $\mathbf{W}\boldsymbol{x}$ and $\mathbf{U}\boldsymbol{z}$:

$$\phi((\mathbf{W}\boldsymbol{x} + \boldsymbol{\beta}_1) \odot (\mathbf{U}\boldsymbol{z} + \boldsymbol{\beta}_2) + \boldsymbol{b}) \tag{6.3}$$

where $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in \mathbb{R}^d$ are bias vectors. Notice that such formulation contains the first order terms as in a additive building block, i.e., $\boldsymbol{\beta}_1 \odot \mathbf{U}\boldsymbol{h}_{t-1} + \boldsymbol{\beta}_2 \odot \mathbf{W}\boldsymbol{x}_t$. In order to make the Multiplicative Integration more flexible, we introduce another bias vector $\boldsymbol{\alpha} \in \mathbb{R}^d$ to gate[1] the term $\mathbf{W}\boldsymbol{x} \odot \mathbf{U}\boldsymbol{z}$, obtaining the following formulation:

$$\phi(\boldsymbol{\alpha} \odot \mathbf{W}\boldsymbol{x} \odot \mathbf{U}\boldsymbol{z} + \boldsymbol{\beta}_1 \odot \mathbf{U}\boldsymbol{z} + \boldsymbol{\beta}_2 \odot \mathbf{W}\boldsymbol{x} + \boldsymbol{b}), \tag{6.4}$$

Note that the number of parameters of the Multiplicative Integration is about the same as that of the additive building block, since the number of new parameters ($\boldsymbol{\alpha}$, $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$) are negligible compared to total number of parameters. Also, Multiplicative Integration can be easily extended to LSTMs and GRUs[2], that adopt vanilla building blocks for computing gates and output states, where one can directly replace them with the Multiplicative Integration. More generally, in any kind of structure where k information flows ($k \geq 2$) are involved (e.g. RNN with multiple skip connections Zhang et al. (2016) or in feedforward models like residual networks He et al. (2015)), one can implement pairwise Multiplicative Integration for integrating all k information sources.

6.2.2 Gradient Properties

The Multiplicative Integration has different gradient properties compared to the additive building block. For clarity of presentation, we first look at vanilla-RNN and RNN with Multiplicative Integration embedded, referred to as *MI-RNN*. That is, $\boldsymbol{h}_t = \phi(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{b})$ versus $\boldsymbol{h}_t = \phi(\mathbf{W}\boldsymbol{x}_t \odot \mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{b})$. In a vanilla-RNN,

---

[1] If $\boldsymbol{\alpha} = \mathbf{0}$, the Multiplicative Integration will degenerate to the vanilla additive building block.

[2] See exact formulations in the Appendix.

the gradient $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}}$ can be computed as follows:

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}} = \prod_{k=t-n+1}^{t} \mathbf{U}^T \text{diag}(\phi'_k), \tag{6.5}$$

where $\phi'_k = \phi'(\mathbf{W}\boldsymbol{x}_k + \mathbf{U}\boldsymbol{h}_{k-1} + \mathbf{b})$. The equation above shows that the gradient flow through time heavily depends on the hidden-to-hidden matrix $\mathbf{U}$, but $\mathbf{W}$ and $\boldsymbol{x}_k$ appear to play a limited role: they only come in the derivative of ϕ' mixed with $\mathbf{U}\boldsymbol{h}_{k-1}$. On the other hand, the gradient $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}}$ of a MI-RNN is[3]:

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}} = \prod_{k=t-n+1}^{t} \mathbf{U}^T \text{diag}(\mathbf{W}\boldsymbol{x}_k)\text{diag}(\phi'_k), \tag{6.6}$$

where $\phi'_k = \phi'(\mathbf{W}\boldsymbol{x}_k \odot \mathbf{U}\boldsymbol{h}_{k-1} + \mathbf{b})$. By looking at the gradient, we see that the matrix $\mathbf{W}$ and the current input $\boldsymbol{x}_k$ is directly involved in the gradient computation by gating the matrix $\mathbf{U}$, hence more capable of altering the updates of the learning system. As we show in our experiments, with $\mathbf{W}\boldsymbol{x}_k$ directly gating the gradient, the vanishing/exploding problem is alleviated: $\mathbf{W}\boldsymbol{x}_k$ dynamically reconciles $\mathbf{U}$, making the gradient propagation easier compared to the regular RNNs. For LSTMs and GRUs with Multiplicative Integration, the gradient propagation properties are more complicated. But in principle, the benefits of the gating effect also persists in these models.

6.3 Experiments

In all of our experiments, we use the general form of Multiplicative Integration (Eq. 6.4) for any hidden state/gate computations, unless otherwise specified.

6.3.1 Exploratory Experiments

To further understand the functionality of Multiplicative Integration, we take a simple RNN for illustration, and perform several exploratory experiments on the character level language modeling task using Penn-Treebank dataset Marcus et al. (1993), following the data partition in Mikolov et al. (2012). The length of the

---

[3] Here we adopt the simplest formulation of Multiplicative Integration for illustration. In the more general case (Eq. 6.4), diag($\mathbf{W}\boldsymbol{x}_k$) in Eq. 6.6 will become diag($\boldsymbol{\alpha} \odot \mathbf{W}\boldsymbol{x}_k + \boldsymbol{\beta}_1$).

training sequence is 50. All models have a single hidden layer of size 2048, and we use Adam optimization algorithm Kingma and Ba (2014) with learning rate $1e^{-4}$. Weights are initialized to samples drawn from uniform$[-0.02, 0.02]$. Performance is evaluated by the bits-per-character (BPC) metric, which is $\log_2$ of perplexity.

Gradient Properties

To analyze the gradient flow of the model, we divide the gradient in Eq. 6.6 into two parts: 1. the gated matrix products: $\mathbf{U}^T \mathrm{diag}(\mathbf{W}\boldsymbol{x}_k)$, and 2. the derivative of the nonlinearity ϕ', We separately analyze the properties of each term compared to the additive building block. We first focus on the gating effect brought by $\mathrm{diag}(\mathbf{W}\boldsymbol{x}_k)$. In order to separate out the effect of nonlinearity, we chose $\boldsymbol{\phi}$ to be the identity map, hence both vanilla-RNN and MI-RNN reduce to linear models, referred to as *lin-RNN* and *lin-MI-RNN*.

For each model we monitor the log-L2-norm of the gradient $log||\partial C/\partial \boldsymbol{h}_t||_2$ (averaged over the training set) after every training epoch, where $\boldsymbol{h}_t$ is the hidden state at time step t, and C is the negative log-likelihood of the single character prediction at the final time step ($t = 50$). Figure. 6.1 (a) shows the evolution of the gradient norms for small t, i.e., $0, 5, 10$, as they better reflect the gradient propagation behaviour. Observe that the norms of lin-MI-RNN (orange) increase rapidly and soon exceed the corresponding norms of lin-RNN by a large margin. The norms of lin-RNN stay close to zero ($\approx 10^{-4}$) and their changes over time are almost negligible. This observation implies that with the help of $\mathrm{diag}(\mathbf{W}\boldsymbol{x}_k)$ term, the gradient vanishing of lin-MI-RNN can be alleviated compared to lin-RNN. The final test BPC (bits-per-character) of lin-MI-RNN is 1.48, which is comparable to a vanilla-RNN with stabilizing regularizer Krueger and Memisevic (2015), while lin-RNN performs rather poorly, achieving a test BPC of over 2.

Next we look into the nonlinearity ϕ. We chose $\phi = \tanh$ for both vanilla-RNN and MI-RNN. Figure 6.1 (c) and (d) shows a comparison of histograms of hidden activations over all time steps on the validation set after training. Interestingly, in (c) for vanilla-RNN, most activations are saturated with values around ± 1, whereas in (d) for MI-RNN, most activations are non-saturated with values around 0. This has a direct consequence in gradient propagation: non-saturated activations imply that $\mathrm{diag}(\phi'_k) \approx 1$ for $\phi = tanh$, which can help gradients propagate, whereas saturated activations imply that $\mathrm{diag}(\phi'_k) \approx 0$, resulting in gradients vanishing.

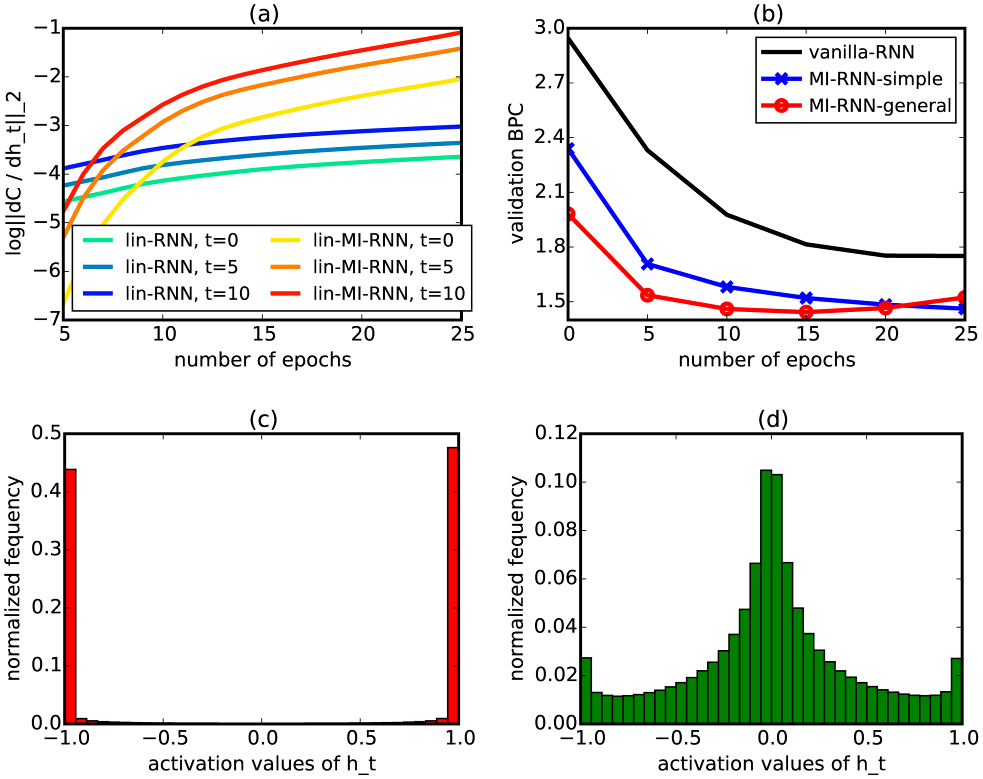


Figure 6.1 – (a) Curves of log-L2-norm of gradients for lin-RNN (blue) and lin-MI-RNN (orange). Time gradually changes from $\{1, 5, 10\}$. (b) Validation BPC curves for vanilla-RNN, MI-RNN-simple using Eq. 6.2, and MI-RNN-general using Eq. 6.4. (c) Histogram of vanilla-RNN's hidden activations over the validation set, most activations are saturated. (d) Histogram of MI-RNN's hidden activations over the validation set, most activations are **not** saturated.

Scaling Problem

When adding two numbers at different order of magnitude, the smaller one might be negligible for the sum. However, when multiplying two numbers, the value of the product depends on both regardless of the scales. This principle also applies when comparing Multiplicative Integration to the additive building blocks. In this experiment, we test whether Multiplicative Integration is more robust to the scales of weight values. Following the same models as in Section 6.3.1, we first calculated the norms of $\mathbf{W}\boldsymbol{x}_k$ and $\mathbf{U}\boldsymbol{h}_{k-1}$ for both vanilla-RNN and MI-RNN for different k after training. We found that in both structures, $\mathbf{W}\boldsymbol{x}_k$ is a lot smaller than $\mathbf{U}\boldsymbol{h}_{k-1}$ in magnitude. This might be due to the fact that $\boldsymbol{x}_k$ is a one-hot

$r_{\mathbf{W}} =$	0.02	0.1	0.3	0.6	std
RNN	1.69	1.65	1.57	1.54	0.06
MI-RNN	**1.39**	1.40	1.40	1.41	0.008

Table 6.1 – Test BPCs and the standard deviation of models with different scales of weight initializations.

vector, making the number of updates for (columns of) $\mathbf{W}$ be smaller than $\mathbf{U}$. As a result, in vanilla-RNN, the pre-activation term $\mathbf{W}\boldsymbol{x}_k + \mathbf{U}\boldsymbol{h}_{k-1}$ is largely controlled by the value of $\mathbf{U}\boldsymbol{h}_{k-1}$, while $\mathbf{W}\boldsymbol{x}_k$ becomes rather small. In MI-RNN, on the other hand, the pre-activation term $\mathbf{W}\boldsymbol{x}_k \odot \mathbf{U}\boldsymbol{h}_{k-1}$ still depends on the values of both $\mathbf{W}\boldsymbol{x}_k$ and $\mathbf{U}\boldsymbol{h}_{k-1}$, due to multiplication.

We next tried different initialization of $\mathbf{W}$ and $\mathbf{U}$ to test their sensitivities to the scaling. For each model, we fix the initialization of $\mathbf{U}$ to uniform$[-0.02, 0.02]$ and initialize $\mathbf{W}$ to uniform$[-r_{\mathbf{W}}, r_{\mathbf{W}}]$ where $r_{\mathbf{W}}$ varies in $\{0.02, 0.1, 0.3, 0.6\}$. Table 6.1 shows results. As we increase the scale of $\mathbf{W}$, performance of the vanilla-RNN improves, suggesting that the model is able to better utilize the input information. On the other hand, MI-RNN is much more robust to different initializations, where the scaling has almost no effect on the final performance.

On different choices of the formulation

In our third experiment, we evaluated the performance of different computational building blocks, which are Eq. 6.1 (*vanilla-RNN*), Eq. 6.2 (*MI-RNN-simple*) and Eq. 6.4 (*MI-RNN-general*)[4]. From the validation curves in Figure 6.1 (b), we see that both MI-RNN, simple and MI-RNN-general yield much better performance compared to vanilla-RNN, and MI-RNN-general has a faster convergence speed compared to MI-RNN-simple. We also compared our results to the previously published models in Table 6.2, top panel, where MI-RNN-general achieves a test BPC of 1.39, which is to our knowledge the best result for RNNs on this task without complex gating/cell mechanisms.

---

[4] We perform hyper-parameter search for the initialization of $\{\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \mathbf{b}\}$ in MI-RNN-general.

Penn-Treebank	BPC
RNN Mikolov et al. (2012)	1.42
HF-MRNN Mikolov et al. (2012)	1.41
RNN+stabalization Krueger and Memisevic (2015)	1.48
MI-RNN (ours)	**1.39**
linear MI-RNN (ours)	1.48
text8	
RNN+smoothReLu Pachitariu and Sahani (2013)	1.55
HF-MRNN Mikolov et al. (2012)	1.54
MI-RNN (ours)	**1.52**
LSTM (ours)	1.51
MI-LSTM(ours)	**1.44**
HutterWikipedia	
stacked-LSTM Graves (2013)	1.67
GF-LSTM Chung et al. (2015)	1.58
grid-LSTM Kalchbrenner et al. (2015)	1.47
MI-LSTM (ours)	**1.44**

Table 6.2 – Top: test BPCs on character level Penn-Treebank dataset. Middle: test BPCs on character level text8 dataset. Bottom: test BPCs on character level Hutter Prize Wikipedia dataset.

6.3.2 Character Level Language Modeling

In addition to the Penn-Treebank dataset, we also perform character level language modeling on two larger datasets: *text8*[5] and *Hutter Challenge Wikipedia*[6]. Both of them contain 100M characters from Wikipedia while *text8* has an alphabet size of 27 and *Hutter Challenge Wikipedia* has an alphabet size of 205. For both datasets, we follow the training protocols in Mikolov et al. (2012) and Chung et al. (2015) respectively. We use Adam for optimization with the starting learning rate grid-searched in $\{0.002, 0.001, 0.0005\}$. If the validation BPC (bits-per-character) does not decrease for 2 epochs, we half the learning rate.

We implemented Multiplicative Integration on both vanilla-RNN and LSTM, referred to as *MI-RNN* and *MI-LSTM*. The results for the *text*8 dataset are shown in Table 6.2, middle panel. All five models, including some of the previously published models, have the same number of parameters (≈4M). For RNNs without complex gating/cell mechanisms (the first three results), our MI-RNN (with $\{\boldsymbol{\alpha}, \boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \mathbf{b}\}$ initialized as $\{2, 0.5, 0.5, 0\}$) performs the best, our MI-LSTM (with $\{\boldsymbol{\alpha}, \boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \mathbf{b}\}$

---

[5] `http://mattmahoney.net/dc/textdata`

[6] `http://prize.hutter1.net/`

WSJ Corpus	CER	WER
DRNN+CTCbeamsearch Hannun et al. (2014)	10.0	14.1
Encoder-Decoder Bahdanau et al. (2015)	6.4	9.3
LSTM+CTCbeamsearch Graves and Jaitly (2014)	9.2	8.7
Eesen Miao et al. (2015)	-	**7.3**
LSTM+CTC+WFST (ours)	6.5	8.7
MI-LSTM+CTC+WFST (ours)	**6.0**	8.2

Table 6.3 – Test CERs and WERs on WSJ corpus.

initialized as $\{1, 0.5, 0.5, 0\}$) outperforms all other models by a large margin[7].

On *Hutter Challenge Wikipedia* dataset, we compare our MI-LSTM (single layer with 2048 unit, ≈17M, with $\{\boldsymbol{\alpha}, \boldsymbol{\beta_1}, \boldsymbol{\beta_2}, \mathbf{b}\}$ initialized as $\{1, 1, 1, 0\}$) to the previous stacked LSTM (7 layers, ≈27M) Graves (2013), GF-LSTM (5 layers, ≈20M) Chung et al. (2015), and grid-LSTM (6 layers, ≈17M) Kalchbrenner et al. (2015). Table 6.2, bottom panel, shows results. Despite the simple structure compared to the sophisticated connection designs in GF-LSTM and grid-LSTM, our MI-LSTM outperforms all other models and achieves the new state-of-the-art on this task.

6.3.3 Speech Recognition

We next evaluate our models on Wall Street Journal (WSJ) corpus (available as LDC corpus LDC93S6B and LDC94S13B), where we use the full 81 hour set "si284" for training, set "dev93" for validation and set "eval92" for test. We follow the same data preparation process and model setting as in Miao et al. (2015), and we use 59 characters as the targets for the acoustic modelling. Decoding is done with the CTC Graves et al. (2006) based weighted finite-state transducers (WFSTs) Mohri et al. (2002) as proposed by Miao et al. (2015).

Our model (referred to as *MI-LSTM+CTC+WFST*) consists of 4 bidirectional MI-LSTM layers, each with 320 units for each direction. CTC is performed on top to resolve the alignment issue in speech transcription. For comparison, we also train a baseline model (referred to as *LSTM+CTC+WFST*) with the same size but using vanilla LSTM. Adam with learning rate 0.0001 is used for optimization and Gaussian weight noise with zero mean and 0.05 standard deviation is injected for

---

[7] Cooijmans et al. (2016) reports better results but they use much larger models (≈16M) which is not directly comparable.

regularization. We evaluate our models on the character error rate (CER) without language model and the word error rate (WER) with extended trigram language model.

Table 6.2, top right panel, shows that MI-LSTM+CTC+WFST achieves quite good results on both CER and WER compared to recent works, and it has a clear improvement over the baseline model. Note that we did not conduct a careful hyper-parameter search on this task, hence one could potentially obtain better results with better decoding schemes and regularization techniques.

6.3.4 Learning Skip-Thought Vectors

Next, we evaluate our Multiplicative Integration on the Skip-Thought model of Kiros et al. (2015). Skip-Thought is an encoder-decoder model that attempts to learn generic, distributed sentence representations. The model produces sentence representation that are robust and perform well in practice, as it achieves excellent results across many different NLP tasks. The model was trained on the BookCorpus dataset that consists of 11,038 books with 74,004,228 sentences. Not surprisingly, a single pass through the training data can take up to a week on a high-end GPU (as reported in Kiros et al. (2015)). Such training speed largely limits one to perform careful hyper-parameter search. However, with Multiplicative Integration, not only the training time is shortened by a factor of two, but the final performance is also significantly improved.

We exactly follow the authors' Theano implementation of the skip-thought model[8]: Encoder and decoder are single-layer GRUs with hidden-layer size of 2400; all recurrent matrices adopt orthogonal initialization while non-recurrent weights are initialized from uniform distribution. Adam is used for optimization. We implemented Multiplicative Integration only for the encoder GRU (embedding MI into decoder did not provide any substantial gains). We refer our model as *MI-uni-skip*, with $\{\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \mathbf{b}\}$ initialized as $\{1, 1, 1, 0\}$. We also train a baseline model with the same size, referred to as *uni-skip(ours)*, which essentially reproduces the original model of Kiros et al. (2015).

During the course of training, we evaluated the skip-thought vectors on the semantic relatedness task, using SICK dataset, every 2500 updates for both MI-

---

[8] `https://github.com/ryankiros/skip-thoughts`

Semantic-Relatedness	r	ρ	**MSE**
uni-skip Kiros et al. (2015)	0.8477	0.7780	0.2872
bi-skip Kiros et al. (2015)	0.8405	0.7696	0.2995
combine-skip Kiros et al. (2015)	0.8584	0.7916	0.2687
uni-skip (ours)	0.8436	0.7735	0.2946
MI-uni-skip (ours)	**0.8588**	**0.7952**	**0.2679**

Paraphrase detection	**Acc**	**F1**
uni-skip Kiros et al. (2015)	73.0	81.9
bi-skip Kiros et al. (2015)	71.2	81.2
combine-skip Kiros et al. (2015)	73.0	82.0
uni-skip (ours)	**74.0**	81.9
MI-uni-skip (ours)	**74.0**	**82.1**

Classification	MR	CR	SUBJ	MPQA
uni-skip Kiros et al. (2015)	75.5	79.3	92.1	86.9
bi-skip Kiros et al. (2015)	73.9	77.9	92.5	83.3
combine-skip Kiros et al. (2015)	76.5	80.1	**93.6**	87.1
uni-skip (ours)	75.9	80.1	93.0	87.0
MI-uni-skip (ours)	**77.9**	**82.3**	93.3	**88.1**

Table 6.4 – Skip-thought+MI on Semantic-Relatedness task (top), Paraphrase Detection task (middle) and four different classification tasks (bottom).

uni-skip and the baseline model (each iteration processes a mini-batch of size 64). The results are shown in Figure 6.2 (a). Note that MI-uni-skip significantly outperforms the baseline, not only in terms of speed of convergence, but also in terms of final performance. At around 125k updates, MI-uni-skip already exceeds the best performance achieved by the baseline, which takes about twice the number of updates.

We also evaluated both models after one week of training, with the best results being reported on six out of eight tasks reported in Kiros et al. (2015): semantic relatedness task on SICK dataset, paraphrase detection task on Microsoft Research Paraphrase Corpus, and four classification benchmarks: movie review sentiment (MR), customer product reviews (CR), subjectivity/objectivity classification (SUBJ), and opinion polarity (MPQA). We also compared our results with the results reported on three models in the original skip-thought paper: *uni-skip*, *bi-skip*, *combine-skip*. Uni-skip is the same model as our baseline, bi-skip is a bidirectional model of the same size, and combine-skip takes the concatenation of

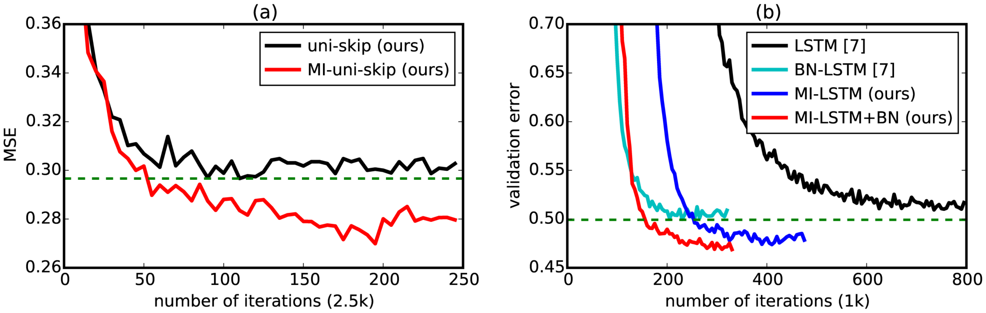


Figure 6.2 – (a) MSE curves of uni-skip (ours) and MI-uni-skip (ours) on semantic relatedness task on SICK dataset. MI-uni-skip significantly outperforms baseline uni-skip. (b) Validation error curves on attentive reader models. There is a clear margin between models with and without MI.

the vectors from uni-skip and bi-skip to form a 4800 dimension vector for task evaluation. Table 6.4 shows that MI-uni-skip dominates across all the tasks. Not only it achieves higher performance than the baseline model, but in many cases, it also outperforms the combine-skip model, which has twice the number of dimensions. Clearly, Multiplicative Integration provides a faster and better way to train a large-scale Skip-Thought model.

6.3.5 Teaching Machines to Read and Comprehend

In our last experiment, we show that the use of Multiplicative Integration can be combined with other techniques for training RNNs, and the advantages of using MI still persist. Recently, Cooijmans et al. (2016) introduced Batch-Normalization Ioffe and Szegedy (2015) for RNNs. They evaluated their proposed technique on a uni-directional Attentive Reader Model Hermann et al. (2015) for the question answering task using the CNN corpus [9]. To test our approach, we evaluated the following four models: 1. A vanilla LSTM attentive reader model with a single hidden layer size 240 (same as Cooijmans et al. (2016)) as our baseline, referred to as *LSTM (ours)*, 2. A multiplicative integration LSTM with a single hidden size 240, referred to as *MI-LSTM*, 3. MI-LSTM with Batch-Norm, referred to as *MI-LSTM+BN*, 4. MI-LSTM with Batch-Norm everywhere (as detailed in Cooijmans et al. (2016)), referred to as *MI-LSTM+BN-everywhere*. We compared

---

[9] Note that Cooijmans et al. (2016) used a truncated version of the original dataset in order to save computation.

Attentive Reader	Val. Err.
LSTM Cooijmans et al. (2016)	0.5033
BN-LSTM Cooijmans et al. (2016)	0.4951
BN-everywhere Cooijmans et al. (2016)	0.5000
LSTM (ours)	0.5053
MI-LSTM (ours)	0.4721
MI-LSTM+BN (ours)	0.4685
MI-LSTM+BN-everywhere (ours)	**0.4644**

Table 6.5 – Multiplicative Integration (with batch normalization) on Teaching Machines to Read and Comprehend task.

our models to results reported in Cooijmans et al. (2016) (referred to as LSTM, BN-LSTM and BN-LSTM everywhere)[10].

For all MI models, $\{\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \mathbf{b}\}$ were initialized to $\{1, 1, 1, 0\}$. We follow the experimental protocol of Cooijmans et al. (2016)[11] and use exactly the same settings as theirs, except we remove the gradient clipping for MI-LSTMs. Figure. 6.2 (b) shows validation curves of the baseline (LSTM), MI-LSTM, BN-LSTM, and MI-LSTM+BN, and the final validation errors of all models are reported in Table 6.5, bottom right panel. Clearly, using Multiplicative Integration results in improved model performance regardless of whether Batch-Norm is used. However, the combination of MI and Batch-Norm provides the best performance and the fastest speed of convergence. This shows the general applicability of Multiplication Integration when combining it with other optimization tricks.

6.4 Relationship to Previous Models

6.4.1 Relationship to Hidden Markov Models

One can show that under certain constraints, MI-RNN is effectively implementing the forward algorithm of the Hidden Markov Model(HMM). A direct mapping can be constructed as follows (see Wessels and Omlin (2000) for a similar derivation). Let $\mathbf{U} \in \mathbb{R}^{m \times m}$ be the state transition probability matrix with

---

[10] Learning curves and the final result number are obtained by emails correspondence with authors of Cooijmans et al. (2016).

[11] `https://github.com/cooijmanstim/recurrent-batch-normalization.git`.

$\mathbf{U}_{ij} = \Pr[h_{t+1} = i | h_t = j]$, $\mathbf{W} \in \mathbb{R}^{m \times n}$ be the observation probability matrix with $\mathbf{W}_{ij} = \Pr[x_t = i | h_t = j]$. When $\boldsymbol{x}_t$ is a one-hot vector (e.g., in many of the language modelling tasks), multiplying it by $\mathbf{W}$ is effectively choosing a column of the observation matrix. Namely, if the jth entry of $\boldsymbol{x}_t$ is one, then $\mathbf{W}\boldsymbol{x}_t = \Pr[x_t | h_t = j]$. Let $\boldsymbol{h}_0$ be the initial state distribution with $\boldsymbol{h}_0 = \Pr[h_0]$ and $\{\boldsymbol{h}_t\}_{t \geq 1}$ be the alpha values in the forward algorithm of HMM, i.e., $\boldsymbol{h}_t = \Pr[x_1, ..., x_t, h_t]$. Then $\mathbf{U}\boldsymbol{h}_t = \Pr[x_1, ..., x_t, h_{t+1}]$. Thus $\boldsymbol{h}_{t+1} = \mathbf{W}\boldsymbol{x}_{t+1} \odot \mathbf{U}\boldsymbol{h}_t = \Pr[x_{t+1} | h_{t+1}] \cdot \Pr[x_1, ..., x_t, h_{t+1}] = \Pr[x_1, ..., x_{t+1}, h_{t+1}]$. To exactly implement the forward algorithm using Multiplicative Integration, the matrices $\mathbf{W}$ and $\mathbf{U}$ have to be probability matrices, and $\boldsymbol{x}_t$ needs to be a one-hot vector. The function ϕ needs to be linear, and we drop all the bias terms. Therefore, RNN with Multiplicative Integration can be seen as a nonlinear extension of HMMs. The extra freedom in parameter values and nonlinearity makes the model more flexible compared to HMMs.

6.4.2 Relations to Second Order RNNs and Multiplicative RNNs

MI-RNN is related to the second order RNN Giles et al. (1991); Goudreau et al. (1994) and the multiplicative RNN (MRNN) Sutskever et al. (2011). We first describe the similarities with these two models:

The second order RNN involves a second order term $\boldsymbol{s}_t$ in a vanilla-RNN, where the ith element $\boldsymbol{s}_{t,i}$ is computed by the bilinear form: $\boldsymbol{s}_{t,i} = \boldsymbol{x}_t^T \mathcal{T}^{(i)} \boldsymbol{h}_{t-1}$, where $\mathcal{T}^{(i)} \in \mathbb{R}^{n \times m} (1 \leq i \leq m)$ is the ith slice of a tensor $\mathcal{T} \in \mathbb{R}^{m \times n \times m}$. Multiplicative Integration also involve a second order term $\boldsymbol{s}_t = \boldsymbol{\alpha} \odot \mathbf{W}\boldsymbol{x}_t \odot \mathbf{U}\boldsymbol{h}_{t-1}$, but in our case $\boldsymbol{s}_{t,i} = \alpha_i(\mathbf{w}_i \cdot \boldsymbol{x}_t)(\mathbf{u}_i \cdot \boldsymbol{h}_{t-1}) = \boldsymbol{x}_t^T(\alpha \mathbf{w}_i \otimes \mathbf{u}_i)\boldsymbol{h}_{t-1}$, where $\mathbf{w}_i$ and $\mathbf{u}_i$ are ith row in $\mathbf{W}$ and $\mathbf{U}$, and α_i is the ith element of $\boldsymbol{\alpha}$. Note that the outer product $\alpha_i \mathbf{w}_i \otimes \mathbf{u}_i$ is a rank-1 matrix. The Multiplicative RNN is also a second order RNN, but which approximates $\mathcal{T}$ by a tensor decomposition $\sum \boldsymbol{x}_t^{(i)} \mathcal{T}^{(i)} = \mathbf{P}\text{diag}(\mathbf{V}\boldsymbol{x}_t)\mathbf{Q}$. For MI-RNN, we can also think of the second order term as a tensor decomposition: $\boldsymbol{\alpha} \odot \mathbf{W}\boldsymbol{x}_t \odot \mathbf{U}\boldsymbol{h}_{t-1} = \mathbf{U}(\boldsymbol{x}_t)\boldsymbol{h}_{t-1} = [\text{diag}(\boldsymbol{\alpha})\text{diag}(\mathbf{W}\boldsymbol{x}_t)\mathbf{U}]\boldsymbol{h}_{t-1}$.

There are however several differences that make MI a favourable model: (1) Simpler Parametrization: MI uses a rank-1 approximation compared to the second order RNNs, and a diagonal approximation compared to Multiplicative RNN. Moreover, MI-RNN shares parameters across the first and second order terms, whereas the other two models do not. As a result, the number of parameters are largely reduced, which makes our model more practical for large scale problems, while avoiding overfitting. (2) Easier Optimization: In tensor decomposition methods, the products of three different (low-rank) matrices generally makes it hard to optimize Sutskever et al. (2011). However, the optimization problem becomes easier in MI, as discussed in section 2 and 3. (3) General structural design vs. vanilla-RNN design: Multiplicative Integration can be easily embedded in many other RNN structures, e.g. LSTMs and GRUs, whereas the second order RNN and MRNN present a very specific design for modifying vanilla-RNNs.

Moreover, we also compared MI-RNN's performance to the previous HF-MRNN's results (Multiplicative RNN trained by Hessian-free method) in Table 6.2, bottom left and bottom middle panels, on Penn-Treebank and text8 datasets. One can see that MI-RNN outperforms HF-MRNN on both tasks.

6.4.3 General Multiplicative Integration

Multiplicative Integration can be viewed as a general way of combining information flows from two different sources. In particular, Rasmus et al. (2015) proposed the ladder network that achieves promising results on semi-supervised learning. In their model, they combine the lateral connections and the backward connections via the "combinator" function by a Hadamard product. The performance would severely degrade without this product as empirically shown by Pezeshki et al. (2015). Yang et al. (2014) explored neural embedding approaches in knowledge bases by formulating relations as bilinear and/or linear mapping functions, and compared a variety of embedding models on the link prediction task. Surprisingly, the best results among all bilinear functions is the simple weighted Hadamard product. They further carefully compare the multiplicative and additive interactions and show that the multiplicative interaction dominates the additive one.

6.5 Conclusion

In this work we proposed to use Multiplicative Integration (MI), a simple Hadamard product to combine information flow in recurrent neural networks. MI can be easily integrated into many popular RNN models, including LSTMs and GRUs, while introducing almost no extra parameters. Indeed, the implementation of MI requires almost no extra work beyond implementing RNN models. We also show that MI achieves state-of-the-art performance on four different tasks or 11 datasets of varying sizes and scales. We believe that the Multiplicative Integration can become a default building block for training various types of RNN models.

7 Prologue to Third Article

7.1 Article Detail

HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. Zhilin Yang[1], Peng Qi[1], Saizheng Zhang[1], Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, Christopher D. Manning. Conference on Empirical Methods in Natural Language Processing (EMMLP), 2018.

Personal Contribution. The idea of examining existing models' multi-hop reasoning abilities came from brainstormings among the first three authors. For the data collection, Zhilin Yang and Peng Qi built the hyperlink graph framework for generating candidate Wikipedia paragraphs while I implemented the preliminary version of the crowdsourcing interface. We three further made significant improvements on the data collection pipeline in which I was mainly responsible for the hit reviewing script and worker quality evaluation. For the experiments, I prepared the distractor setting and built the retrieval baseline for full-wiki setting. Zhilin Yang proposed the model based data split and conducted the benchmark setting experiments and ablation studies with strong end-to-end neural baseline models. Peng Qi gave a thorough data analysis of the collected dataset and established the human performance evaluation. The other co-authors provided several critical advice on data collection and experiments.

7.2 Context

Existing question answering (QA) datasets fail to train QA systems to perform complex reasoning and provide explanations for answers. We introduce HOTPOTQA, a new dataset with 113k Wikipedia-based question-answer pairs with

---

[1] Equal contribution.

four key features: (1) the questions require finding and reasoning over multiple supporting documents to obtain an answer; (2) the questions are diverse and not constrained to any pre-existing knowledge bases or knowledge schemas; (3) we provide sentence-level supporting facts required for reasoning, allowing QA systems to reason with strong supervision and explain the predictions; (4) we offer a new type of factoid comparison questions to test QA systems' ability to extract relevant facts and perform necessary comparison. We show that HOTPOTQA is challenging for the latest QA systems, and the supporting facts enable models to improve performance and make explainable predictions.

7.3 Contributions

The HotpotQA dataset introduced in this work is the very first large scale question answering dataset aiming at facilitating the development of QA systems capable of performing explainable, multi-hop reasoning. Compared to previous existing resources, HotpotQA dataset ensures the requirement of true multi-hop reasoning, enables explainable model prediction by introducing support facts and forces the question-answer pairs being in diversified natural language form. A byproduct is the proposed high-quality multi-hop question collection pipeline which may shed light on future work in this direction.

8 HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering

8.1 Introduction

The ability to perform reasoning and inference over natural language is an important aspect of intelligence. The task of question answering (QA) provides a quantifiable and objective way to test the reasoning ability of intelligent systems. To this end, a few large-scale QA datasets have been proposed, which sparked significant progress in this direction. However, existing datasets have limitations that hinder further advancements of machine reasoning over natural language, especially in testing QA systems' ability to perform *multi-hop reasoning*, where the system has to reason with substantial information from more than one document to arrive at the answer.

First, some datasets mainly focus on testing the ability of reasoning within a single paragraph or document, or *single-hop reasoning*. For example, SQuAD (Rajpurkar et al., 2016) questions are designed to be answered given a single paragraph as the context, and most of the questions can in fact be addressed by matching the question with a single sentence in that paragraph. As a result, it has fallen short at testing systems' ability to reason over a larger context. TriviaQA (Joshi et al., 2017b) and SearchQA (Dunn et al., 2017) create a more challenging setting by using information retrieval to collect multiple documents to form the context given existing question-answer pairs. Nevertheless, most of the questions can be answered by matching the question with a few nearby sentences in one single paragraph, which is limited as it does not require more complex reasoning (e.g., over multiple paragraphs).

Second, existing datasets that target multi-hop reasoning, such as WikiHop (Welbl et al., 2018a) and ComplexWebQ (Talmor and Berant, 2018), are constructed on top of existing knowledge bases (KBs). As a result, these datasets are constrained by the schema of the KBs they use, and therefore the diversity of

Paragraph A, Return to Olympus:
[1] *Return to Olympus is the only album by the alternative rock band Malfunkshun.* [2] *It was released after the band had broken up and after lead singer Andrew Wood (later of Mother Love Bone) had died of a drug overdose in 1990.* [3] Stone Gossard, of Pearl Jam, had compiled the songs and released the album on his label, Loosegroove Records.

Paragraph B, Mother Love Bone:
[4] *Mother Love Bone was an American rock band that formed in Seattle, Washington in 1987.* [5] The band was active from 1987 to 1990. [6] *Frontman Andrew Wood's personality and compositions helped to catapult the group to the top of the burgeoning late 1980s/early 1990s Seattle music scene.* [7] *Wood died only days before the scheduled release of the band's debut album, "Apple", thus ending the group's hopes of success.* [8] The album was finally released a few months later.

Q: What was the former band of the member of Mother Love Bone who died just before the release of "Apple"?
A: Malfunkshun
Supporting facts: 1, 2, 4, 6, 7

Figure 8.1 – An example of the multi-hop questions in HotpotQA. We also highlight the supporting facts in *blue italics*, which are also part of the dataset.

questions and answers is inherently limited.

Third, all of the above datasets only provide distant supervision; i.e., the systems only know what the answer is, but do not know what supporting facts lead to it. This makes it difficult for models to learn about the underlying reasoning process, as well as to make explainable predictions.

To address the above challenges, we aim at creating a QA dataset that requires reasoning over multiple documents, and does so in natural language, without constraining itself to an existing knowledge base or knowledge schema. We also want it to provide the system with strong supervision about what text the answer is actually derived from, to help guide systems to perform meaningful and explainable reasoning.

We present HotpotQA[1], a large-scale dataset that satisfies these desiderata. HotpotQA is collected by crowdsourcing based on Wikipedia articles, where crowd workers are shown multiple supporting context documents and asked explicitly to come up with questions requiring reasoning about all of the documents. This

---

[1] The name comes from the first three authors' arriving at the main idea during a discussion at a hot pot restaurant.

ensures it covers multi-hop questions that are more natural, and are not designed with any pre-existing knowledge base schema in mind. Moreover, we also ask the crowd workers to provide the supporting facts they use to answer the question, which we also provide as part of the dataset (see Figure 8.1 for an example). We have carefully designed a data collection pipeline for HotpotQA, since the collection of high-quality multi-hop questions is non-trivial. We hope that this pipeline also sheds light on future work in this direction. Finally, we also collected a novel type of questions—comparison questions—as part of HotpotQA, in which we require systems to compare two entities on some shared properties to test their understanding of both language and common concepts such as numerical magnitude. We make HotpotQA publicly available at **https://HotpotQA.github.io**.

8.2 Data Collection

The main goal of our work is to collect a diverse and explainable question answering dataset that requires multi-hop reasoning. One way to do so is to define reasoning chains based on a knowledge base (Welbl et al., 2018a; Talmor and Berant, 2018). However, the resulting datasets are limited by the incompleteness of entity relations and the lack of diversity in the question types. Instead, in this work, we focus on text-based question answering in order to diversify the questions and answers. The overall setting is that given some context paragraphs (e.g., a few paragraphs, or the entire Web) and a question, a QA system answers the question by extracting a span of text from the context, similar to Rajpurkar et al. (2016). We additionally ensure that it is necessary to perform multi-hop reasoning to correctly answer the question.

8.2.1 Pipeline

It is non-trivial to collect text-based multi-hop questions. In our pilot studies, we found that simply giving an arbitrary set of paragraphs to crowd workers is counterproductive, because for most paragraph sets, it is difficult to ask a meaningful multi-hop question. To address this challenge, we carefully design a pipeline to collect text-based multi-hop questions. Next, we will highlight the key design choices in our pipeline.

Building a Wikipedia Hyperlink Graph. We use the entire English Wikipedia dump as our corpus.[2] In this corpus, we make two observations: (1) hyper-links in the Wikipedia articles often naturally entail a relation between two (already disambiguated) entities in the context, which could potentially be used to facilitate multi-hop reasoning; (2) the first paragraph of each article often contains the most information that could be queried in a meaningful way. Based on these observations, we extract all the hyperlinks from the first paragraphs of all Wikipedia articles. With these hyperlinks, we build a directed graph G, where each edge (a, b) indicates there is a hyperlink from the first paragraph of article a to article b.

Generating Candidate Paragraph Pairs. To generate meaningful pairs of paragraphs for multi-hop question answering with G, we start by considering an example question "when was of the singer and songwriter of Radiohead born?" To answer this question, one would need to first reason that the "singer and songwriter of Radiohead" is "Thom Yorke", and then figure out his birthday in the text. We call "Thom Yorke" a *bridge entity* in this example. Given an edge (a, b) in the hyperlink graph G, the entity of b can usually be viewed as a bridge entity that connects a and b. As we observe article b's usually determine the theme of the shared context between a and b, but not all entity b's are suitable for collecting multi-hop questions. For example, entities like countries are frequently referred to in Wikipedia, but don't necessarily have much in common with all incoming links. It is also difficult, for instance, for the crowd workers to ask meaningful multi-hop questions about highly technical entities like the IPv4 protocol. To alleviate this issue, we constrain the bridge entities to a set of curated pages in Wikipedia, see Section 8.2.2. After manually curating a set of pages B, we create candidate paragraph pairs by sampling edges (a, b) from the hyperlink graph such that $b \in B$.

Comparison Questions. In addition to questions collected using bridge entities, we also collect another type of multi-hop questions—comparison questions. The main idea is that comparing two entities from the same category usually results

---

[2] https://dumps.wikimedia.org/

Algorithm 1 Overall data collection procedure

```
Input: question type ratio $r_1 = 0.75$, yes/no ratio $r_2 = 0.5$
while not finished do
  if random() $< r_1$ then
    Uniformly sample an entity $b \in B$
    Uniformly sample an edge $(a, b)$
    Workers ask a question about paragraphs $a$ and $b$
  else
    Sample a list from $L$, with probabilities weighted by list sizes
    Uniformly sample two entities $(a, b)$ from the list
    if random() $< r_2$ then
      Workers ask a yes/no question to compare $a$ and $b$
    else
      Workers ask a question with a span answer to compare $a$ and $b$
    end if
  end if
  Workers provide the supporting facts
end while
```

in interesting multi-hop questions, e.g., "Who has played for more NBA teams, Michael Jordan or Kobe Bryant?" To facilitate collecting this type of question, we manually curate 42 lists of similar entities (denoted as L) from Wikipedia.[3] To generate candidate paragraph pairs, we randomly sample two paragraphs from the same list and present them to the crowd worker.

To increase the diversity of multi-hop questions, we also introduce a subset of yes/no questions in comparison questions. This complements the original scope of comparison questions by offering new ways to require systems to reason over both paragraphs. For example, consider the entities Iron Maiden (from the UK) and AC/DC (from Australia). Questions like "Is Iron Maiden or AC/DC from the UK?" are less ideal, because one would deduce the answer is "Iron Maiden" even if one only had access to that article. With yes/no questions, one may ask "Are Iron Maiden and AC/DC from the same country?", which requires reasoning over both paragraphs.

To the best of our knowledge, text-based comparison questions are a novel type of questions that have not been considered by previous datasets. More importantly, answering these questions usually requires arithmetic comparison, such as

---

[3] This is achieved by manually curating lists from the Wikipedia "List of lists of lists" (`https://wiki.sh/y8qv`). One example is "Highest Mountains on Earth".

comparing ages given birth dates, which presents a new challenge for future model development.

Collecting Supporting Facts. To enhance the explainability of question answering systems, we want them to output a set of *supporting facts* necessary to arrive at the answer, when the answer is generated. To this end, we also collect the sentences that determine the answers from crowd workers. These supporting facts can serve as strong supervision for what sentences to pay attention to. Moreover, we can now test the explainability of a model by comparing the predicted supporting facts to the ground truth ones.

The overall procedure of data collection is illustrated in Algorithm 1.

8.2.2 Implementation Details

Wikipedia Preprocessing. We downloaded the dump of English Wikipedia of October 1, 2017, and extracted text and hyperlinks with WikiExtractor.[4] We use Stanford CoreNLP 3.8.0 (Manning et al., 2014) for word and sentence tokenization. We use the resulting sentence boundaries for collection of supporting facts, and use token boundaries to check whether Turkers are providing answers that cover spans of entire tokens to avoid nonsensical partial-word answers.

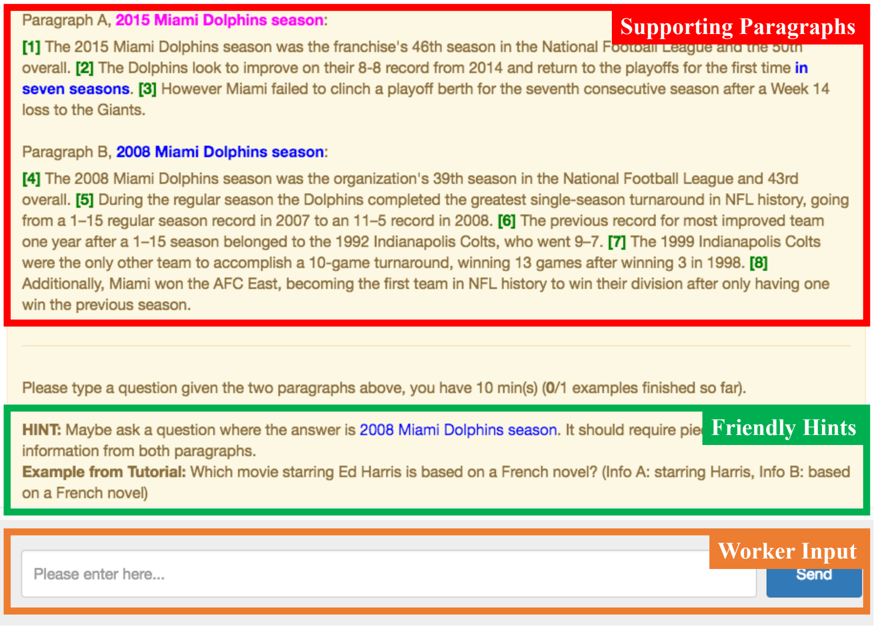
Curating Wikipedia Pages. To make sure the sampled candidate paragraph pairs are intuitive for crowd workers to ask high-quality multi-hop questions about, we manually curate 591 categories from the lists of popular pages by WikiProject.[5] For each category, we sample (a, b) pairs from the graph G where b is in the considered category, and manually check whether a multi-hop question can be asked given the pair (a, b). Those categories with a high probability of permitting multi-hop questions are selected.

Bonus Structures. To incentivize crowd workers to produce higher-quality data more efficiently, we follow Yang et al. (2018b), and employ bonus structures. We mix two settings in our data collection process. In the first setting, we reward the top (in terms of numbers of examples) workers every 200 examples. In the

---

[4] `https://github.com/attardi/wikiextractor`

[5] `https://wiki.sh/y8qu`



Paragraph A, 2015 Miami Dolphins season:

[1] The 2015 Miami Dolphins season was the franchise's 46th season in the National Football League and the 50th overall. [2] The Dolphins look to improve on their 8-8 record from 2014 and return to the playoffs for the first time in seven seasons. [3] However Miami failed to clinch a playoff berth for the seventh consecutive season after a Week 14 loss to the Giants.

Paragraph B, 2008 Miami Dolphins season:

[4] The 2008 Miami Dolphins season was the organization's 39th season in the National Football League and 43rd overall. [5] During the regular season the Dolphins completed the greatest single-season turnaround in NFL history, going from a 1–15 regular season record in 2007 to an 11–5 record in 2008. [6] The previous record for most improved team one year after a 1–15 season belonged to the 1992 Indianapolis Colts, who went 9–7. [7] The 1999 Indianapolis Colts were the only other team to accomplish a 10-game turnaround, winning 13 games after winning 3 in 1998. [8] Additionally, Miami won the AFC East, becoming the first team in NFL history to win their division after only having one win the previous season.

Please type a question given the two paragraphs above, you have 10 min(s) (**0**/1 examples finished so far).

HINT: Maybe ask a question where the answer is 2008 Miami Dolphins season. It should require pie information from both paragraphs.
Example from Tutorial: Which movie starring Ed Harris is based on a French novel? (Info A: starring Harris, Info B: based on a French novel)

Figure 8.2 – Screenshot of our worker interface on Amazon Mechanical Turk.

second setting, the workers get bonuses based on their productivity (measured as the number of examples per hour).

Crowd Worker Interface. Our crowd worker interface is based on Amazon Mechanical Turk[6] using the ParlAI (Miller et al., 2017), an open-source project that facilitates the development of dialog systems and data collection with a dialog interface. We adapt ParlAI for collecting question answer pairs by converting the collection workflow into a system-oriented dialog. This allows us to have more control over the turkers input, as well as provide turkers with in-the-loop feedbacks or helpful hints to help Turkers finish the task, and therefore speed up the collection process. See Figure 8.2 for an example of the worker interface during data collection.

8.3 Processing and Benchmark Settings

We collected 112,779 valid data examples in total .We first split out a subset of data called *train-easy*. Specifically, we randomly sampled questions (∼3–10 per

---

[6] `https://www.mturk.com/`

Name	Desc.	Usage	# Examples
train-easy	single-hop	training	18,089
train-medium	multi-hop	training	56,814
train-hard	hard multi-hop	training	15,661
dev	hard multi-hop	dev	7,405
test-distractor	hard multi-hop	test	7,405
test-fullwiki	hard multi-hop	test	7,405
Total			112,779

Table 8.1 – Data split. The splits *train-easy*, *train-medium*, and *train-hard* are combined for training. The distractor and full wiki settings use different test sets so that the gold paragraphs in the full wiki test set remain unknown to any models.

Turker) from top-contributing turkers, and categorized all their questions into the *train-easy* set if an overwhelming percentage in the sample only require reasoning over one of the paragraphs. We only sampled these turkers because they contributed more than 70% of our data. This *train-easy* set contains 18,089 mostly single-hop examples.

We implemented a question answering model based on the current state-of-the-art architectures, which we discuss in detail in Section 8.5.1. Based on this model, we performed a three-fold cross validation on the remaining multi-hop examples. Among these examples, the models were able to correctly answer 60% of the questions with high confidence (determined by thresholding the model loss). These correctly-answered questions (56,814 in total, 60% of the multi-hop examples) are split out and marked as the *train-medium* subset, which will also be used as part of our training set.

After splitting out *train-easy* and *train-medium*, we are left with hard examples. As our ultimate goal is to solve multi-hop question answering, we focus on questions that the latest modeling techniques are not able to answer. Thus we constrain our dev and test sets to be hard examples. Specifically, we randomly divide the hard examples into four subsets, *train-hard*, *dev*, *test-distractor*, and *test-fullwiki*. Statistics about the data split can be found in Table 8.1. In Section 8.5, we will show that combining *train-easy*, *train-medium*, and *train-hard* to train models yields the best performance, so we use the combined set as our default training set. The two test sets *test-distractor* and *test-fullwiki* are used in two different benchmark settings, which we introduce next.

We create two benchmark settings. In the first setting, to challenge the model to find the true supporting facts in the presence of noise, for each example we employ bigram tf-idf (Chen et al., 2017) to retrieve 8 paragraphs from Wikipedia as *distractors*, using the question as the query. We mix them with the 2 gold paragraphs (the ones used to collect the question and answer) to construct the **distractor** setting. The 2 gold paragraphs and the 8 distractors are shuffled before they are fed to the model. In the second setting, we fully test the model's ability to locate relevant facts as well as reasoning about them by requiring it to answer the question given the first paragraphs of all Wikipedia articles without the gold paragraphs specified. This **full wiki** setting truly tests the performance of the systems' ability at multi-hop reasoning in the wild.[7] The two settings present different levels of difficulty, and would require techniques ranging from reading comprehension to information retrieval. As shown in Table 8.1, we use separate test sets for the two settings to avoid leaking information, because the gold paragraphs are available to a model in the distractor setting, but should not be accessible in the full wiki setting.

We also try to understand the model's good performance on the *train-medium* split. Manual analysis shows that the ratio of multi-hop questions in *train-medium* is similar to that of the hard examples (93.3% in *train-medium* vs. 92.0% in *dev*), but one of the question types appears more frequently in *train-medium* compared to the hard splits (Type II: 32.0% in *train-medium* vs. 15.0% in *dev*, see Section 8.4 for the definition of Type II questions). These observations demonstrate that given enough training data, existing neural architectures can be trained to answer certain types and certain subsets of the multi-hop questions. However, *train-medium* remains challenging when not just the gold paragraphs are present:Table 8.2 shows the comparison between *train-medium* split and hard examples like *dev* and *test* under retrieval metrics in full wiki setting. As we can see, the performance gap between *train-medium* split and its *dev*/*test* is close, which implies that *train-medium* split has a similar level of difficulty as hard examples under the full wiki setting in which a retrieval model is necessary as the first processing step.

---

[7] As we required the crowd workers to use complete entity names in the question, the majority of the questions are unambiguous in the full wiki setting.

Set	MAP	Mean Rank	CorAns Rank
train-medium	41.89	288.19	82.76
dev	42.79	304.30	97.93
test	45.92	286.20	74.85

Table 8.2 – Retrieval performance comparison on full wiki setting for *train-medium*, *dev* and *test* with 1,000 random samples each. MAP and are in %. Mean Rank averages over retrieval ranks of two gold paragraphs. CorAns Rank refers to the rank of the gold paragraph containing the answer.

8.4 Dataset Analysis

In this section, we analyze the types of questions, types of answers, and types of multi-hop reasoning covered in the dataset.

Question Types. We heuristically identified question types for each collected question. To identify the question type, we first locate the *central question word* (CQW) in the question. Since HotpotQA contains comparison questions and yes/no questions, we consider as *question words* WH-words, copulas ("is", "are"), and auxiliary verbs ("does", "did"). Because questions often involve relative clauses beginning with WH-words, we define the CQW as the first question word in the question if it can be found in the first three tokens, or the last question word otherwise. Then, we determine question type by extracting words up to 2 tokens away to the right of the CQW, along with the token to the left if it is one of a few common prepositions (e.g., in the cases of "in which" and "by whom").

We visualize the distribution of question types in Figure 8.3, and label the ones shared among more than 250 questions. As is shown, our dataset covers a diverse variety of questions centered around entities, locations, events, dates, and numbers, as well as yes/no questions directed at comparing two entities ("Are both A and B ...?"), to name a few.

Answer Types. We further sample 100 examples from the dataset, and present the types of answers in Table 8.3. As can be seen, HotpotQA covers a broad range of answer types, which matches our initial analysis of question types. We

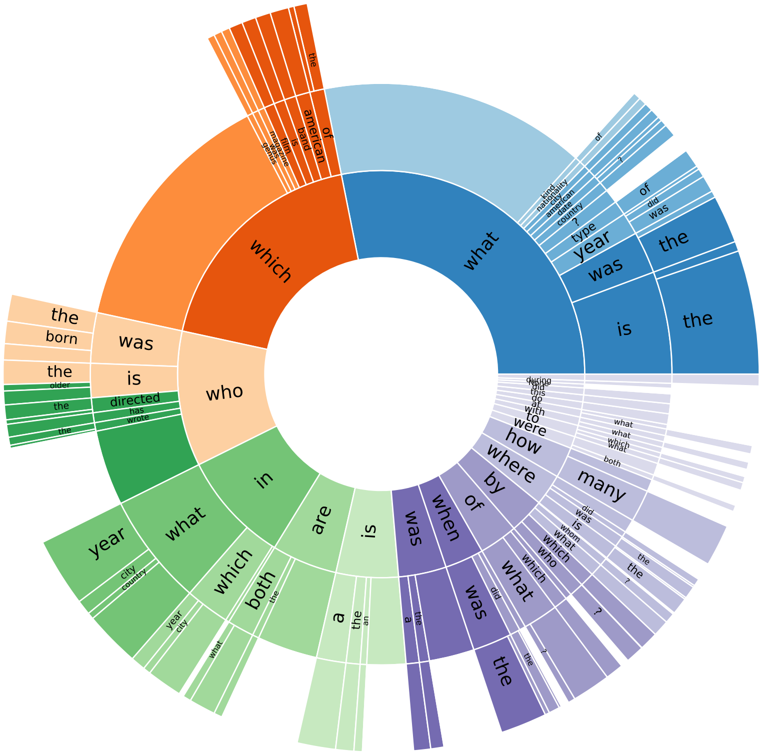


Figure 8.3 – Types of questions covered in HotpotQA. Question types are extracted heuristically, starting at question words or prepositions preceding them. Empty colored blocks indicate suffixes that are too rare to show individually. See main text for more details.

find that a majority of the questions are about entities in the articles (68%), and a non-negligible amount of questions also ask about various properties like date (9%) and other descriptive properties such as numbers (8%) and adjectives (4%).

Multi-hop Reasoning Types. We also sampled 100 examples from the dev and test sets and manually classified the types of reasoning required to answer each question. Besides comparing two entities, there are three main types of multi-hop reasoning required to answer these questions, which we show in Table 8.4 accompanied with examples.

Most of the questions require at least one supporting fact from each paragraph to answer. A majority of sampled questions (42%) require chain reasoning (Type I in the table), where the reader must first identify a bridge entity before the second hop can be answered by filling in the bridge. One strategy to answer these questions would be to decompose them into consecutive single-hop questions. The

Answer Type	%	Example(s)
Person	30	King Edward II, Rihanna
Group / Org	13	Cartoonito, Apalachee
Location	10	Fort Richardson, California
Date	9	10th or even 13th century
Number	8	79.92 million, 17
Artwork	8	Die schweigsame Frau
Yes/No	6	-
Adjective	4	conservative
Event	1	Prix Benois de la Danse
Other proper noun	6	Cold War, Laban Movement Analysis
Common noun	5	comedy, both men and women

Table 8.3 – Types of answers in HotpotQA.

bridge entity could also be used implicitly to help infer properties of other entities related to it. In some questions (Type III), the entity in question shares certain properties with a bridge entity (e.g., they are collocated), and we can infer its properties through the bridge entity. Another type of question involves locating the answer entity by satisfying multiple properties simultaneously (Type II). Here, to answer the question, one could find the set of all entities that satisfy each of the properties mentioned, and take an intersection to arrive at the final answer. Questions comparing two entities (Comparison) also require the system to understand the properties in question about the two entities (e.g., nationality), and sometimes require arithmetic such as counting (as seen in the table) or comparing numerical values ("Who is older, A or B?"). Finally, we find that sometimes the questions require more than two supporting facts to answer (Other). In our analysis, we also find that for all of the examples shown in the table, the supporting facts provided by the Turkers match exactly with the limited context shown here, showing that the supporting facts collected are of high quality.

Aside from the reasoning types mentioned above, we also estimate that about 6% of the sampled questions can be answered with one of the two paragraphs, and 2% of them unanswerable. We also randomly sampled 100 examples from *train-medium* and *train-hard* combined, and the proportions of reasoning types are: Type I 38%, Type II 29%, Comparison 20%, Other 7%, Type III 2%, single-hop 2%, and unanswerable 2%.

Reasoning Type	%	Example(s)
Inferring the ***bridge entity*** to complete the 2nd-hop question (Type I)	42	**Paragraph A:** The 2015 Diamond Head Classic was a college basketball tournament ... ***Buddy Hield*** *was named the tournament's MVP*. **Paragraph B:** ***Chavano Rainier ”Buddy” Hield*** is a Bahamian professional basketball player for the **Sacramento Kings** of the NBA... **Q:** Which team does the player named 2015 Diamond Head Classic's MVP play for?
Comparing two entities (Comparison)	27	**Paragraph A:** LostAlone were a British rock band ... consisted of *Steven Battelle, Alan Williamson, and Mark Gibson*... **Paragraph B:** Guster is an American alternative rock band ... Founding members *Adam Gardner, Ryan Miller, and Brian Rosenworcel* began... **Q:** Did LostAlone and Guster have the same number of members? (**yes**)
Locating the **answer entity** by checking multiple properties (Type II)	15	**Paragraph A:** Several *current and former members of the Pittsburgh Pirates* – ... John Milner, **Dave Parker**, and Rod Scurry... **Paragraph B:** **David Gene Parker**, *nicknamed ”The Cobra”*, is an American former player in Major League Baseball... **Q:** Which former member of the Pittsburgh Pirates was nicknamed ”The Cobra”?
Inferring about the property of an entity in question through a ***bridge entity*** (Type III)	6	**Paragraph A:** *Marine Tactical Air Command Squadron 28* is a United States Marine Corps aviation command and control unit based at ***Marine Corps Air Station Cherry Point***... **Paragraph B:** ***Marine Corps Air Station Cherry Point*** ... is a United States Marine Corps airfield located in **Havelock, North Carolina**, USA ... **Q:** What city is the Marine Air Control Group 28 located in?
Other types of reasoning that require more than two supporting facts (Other)	2	**Paragraph A:** ... the towns of Yodobashi, **Okubo, Totsuka, and Ochiai town** *were merged into Yodobashi ward*. ... ***Yodobashi Camera*** *is a store with its name taken from the town and ward*. **Paragraph B**: ***Yodobashi Camera*** Co., Ltd. is *a major Japanese retail chain specializing in electronics, PCs, cameras and photographic equipment*. **Q:** Aside from Yodobashi, what other towns were merged into the ward which gave the major Japanese retail chain specializing in electronics, PCs, cameras, and photographic equipment it's name?

Table 8.4 – Types of multi-hop reasoning required to answer questions in the HotpotQA dev and test sets. We show in ***orange bold italics*** bridge entities if applicable, *blue italics* supporting facts from the paragraphs that connect directly to the question, and **green bold** the answer in the paragraph or following the question. The remaining 8% are single-hop (6%) or unanswerable questions (2%) by our judgement.

8.5 Experiments

8.5.1 Model Architecture and Training

To test the performance of leading QA systems on our data, we reimplemented the architecture described in Clark and Gardner (2017) as our baseline model. We note that our implementation without weight averaging achieves performance very close to what the authors reported on SQuAD (about 1 point worse in F1). Our implemented model subsumes the latest technical advances on question answering, including character-level models, self-attention (Wang et al., 2017), and bi-attention Seo et al. (2017). Combining these three key components is becoming standard practice, and various state-of-the-art or competitive architectures (Liu et al., 2018; Clark and Gardner, 2017; Wang et al., 2017; Seo et al., 2017; Pan et al., 2017; Salant and Berant, 2018; Xiong et al., 2018) on SQuAD can be viewed as similar to our implemented model. To accommodate yes/no questions, we also add a 3-way classifier after the last recurrent layer to produce the probabilities of "yes", "no", and span-based answers. During decoding, we first use the 3-way output to determine whether the answer is "yes", "no", or a text span. If it is a text span, we further search for the most probable span.

Supporting Facts as Strong Supervision. To evaluate the baseline model's performance in predicting explainable supporting facts, as well as how much they improve QA performance, we additionally design a component to incorporate such strong supervision into our model. For each sentence, we concatenate the output of the self-attention layer at the first and last positions, and use a binary linear classifier to predict the probability that the current sentence is a supporting fact. We minimize a binary cross entropy loss for this classifier. This objective is jointly optimized with the normal question answering objective in a multi-task learning setting, and they share the same low-level representations. With this classifier, the model can also be evaluated on the task of supporting fact prediction to gauge its explainability. Our overall architecture is illustrated in Figure 8.4. Though it is possible to build a pipeline system, in this work we focus on an end-to-end one, which is easier to tune and faster to train.

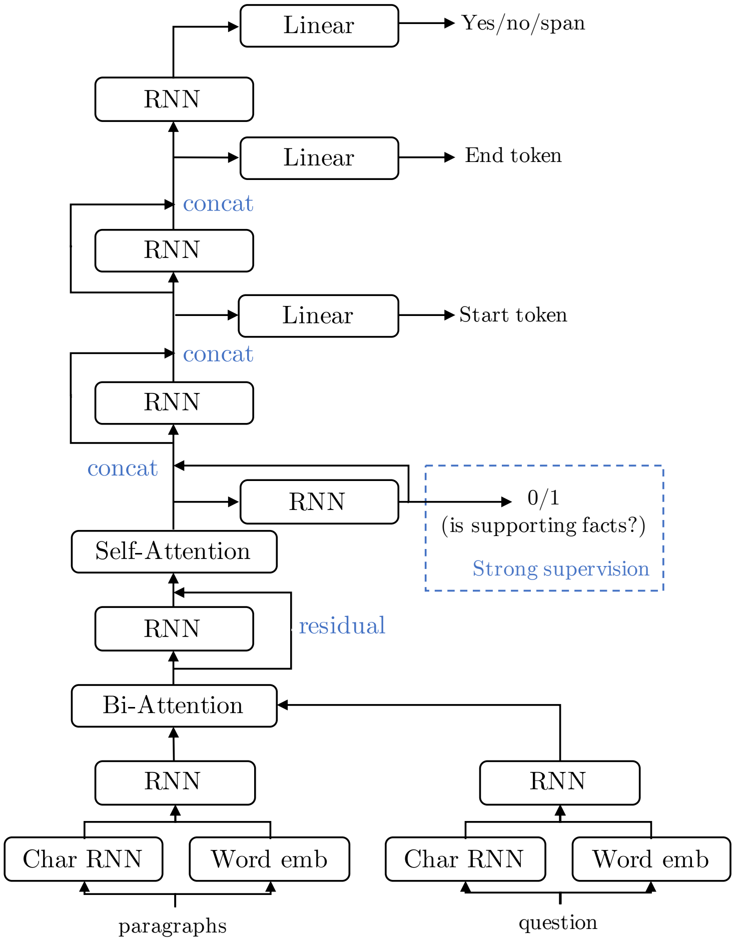


Figure 8.4 – Our model architecture. Strong supervision over supporting facts is used in a multi-task setting.

8.5.2 Results

We evaluate our model in the two benchmark settings. In the full wiki setting, to enable efficient tf-idf retrieval among 5,000,000+ wiki paragraphs, given a question we first return a candidate context pool of at most 5,000 paragraphs using an inverted index based filtering strategy (see Algorithm 2) and then select the top 10 paragraphs in the pool by standard bigram tf-idf retrieval to gather final candidates.[8] Retrieval performance is shown in Table 8.5. After retrieving these 10 paragraphs, we then use the model trained in the distractor setting to evaluate its performance on these final candidate paragraphs.

---

[8] We choose the number of final candidates as 10 to stay consistent with the distractor setting where candidates are 2 gold paragraphs plus 8 distractors.

Algorithm 2 Inverted Index Filtering Strategy

```
Input: question text $q$, control threshold $N$, ngram-to-Wikidoc inverted index $\mathcal{D}$
Inintialize:
Extract unigram + bigram set $r_q$ from $q$
$N_{cand} = +\infty$
$C_{gram} = 0$
while $N_{cands} > N$ do
    $C_{gram} = C_{gram} + 1$
    Set $S_{overlap}$ to be an empty dictionary
    for $w \in r_q$ do
        for $d \in \mathcal{D}[w]$ do
            if $d$ not in $S_{overlap}$ then
                $S_{overlap}[d] = 1$
            else
                $S_{overlap}[d] = S_{overlap}[d] + 1$
            end if
        end for
    end for
    $S_{cand} = \emptyset$
    for $d$ in $S_{overlap}$ do
        if $S_{overlap}[d] \geq C_{gram}$ then
            $S_{cand} = S_{cand} \cup \{d\}$
        end if
    end for
    $N_{cands} = |S_{cand}|$
end while
return $S_{cand}$
```

Following previous work (Rajpurkar et al., 2016), we use exact match (EM) and F1 as two evaluation metrics. To assess the explainability of the models, we further introduce two sets of metrics involving the supporting facts. The first set focuses on evaluating the supporting facts directly, namely EM and F1 on the set of supporting fact sentences as compared to the gold set. The second set features joint metrics that combine the evaluation of answer spans and supporting facts as follows. For each example, given its precision and recall on the answer span $(P^{(\mathrm{ans})}, R^{(\mathrm{ans})})$ and the supporting facts $(P^{(\mathrm{sup})}, R^{(\mathrm{sup})})$, respectively, we calculate joint F1 as

$$P^{(\mathrm{joint})} = P^{(\mathrm{ans})} P^{(\mathrm{sup})}, \quad R^{(\mathrm{joint})} = R^{(\mathrm{ans})} R^{(\mathrm{sup})},$$

Set	MAP	Mean Rank	Hits@2	Hits@10
dev	43.93	314.71	39.43	56.06
test	43.21	314.05	38.67	55.88

Table 8.5 – Retrieval performance in the full wiki setting. Mean Rank is averaged over the ranks of two gold paragraphs.

Setting	Split	Answer		Sup Fact		Joint	
		EM	F1	EM	F1	EM	F1
distractor	dev	44.44	58.28	21.95	66.66	11.56	40.86
distractor	test	45.46	58.99	22.24	66.62	12.04	41.37
full wiki	dev	24.68	34.36	5.28	40.98	2.54	17.73
full wiki	test	25.23	34.40	5.07	40.69	2.63	17.85

Table 8.6 – Main results: the performance of question answering and supporting fact prediction in the two benchmark settings. We encourage researchers to report these metrics when evaluating their methods.

$$\text{Joint F1} = \frac{2P^{(\text{joint})}R^{(\text{joint})}}{P^{(\text{joint})} + R^{(\text{joint})}}.$$

Joint EM is 1 only if both tasks achieve an exact match and otherwise 0. Intuitively, these metrics penalize systems that perform poorly on either task. All metrics are evaluated example-by-example and then averaged over the evaluation set.

The performance of our model on the benchmark settings is reported in Table 8.6, where all numbers are obtained with strong supervision over supporting facts. From the distractor setting to the full wiki setting, expanding the scope of the context increases the difficulty of question answering. The performance in the full wiki setting is substantially lower, which poses a challenge to existing techniques on retrieval-based question answering. Overall, model performance in all settings is significantly lower than human performance as shown in Section 8.5.3, which indicates that more technical advancements are needed in future work.

We also investigate the explainability of our model by measuring supporting fact prediction performance. Our model achieves 60+ supporting fact prediction F1 and ∼40 joint F1, which indicates there is room for further improvement in terms of explainability.

In Table 8.7, we break down the performance on different question types. In

Setting	Br EM	Br F1	Cp EM	Cp F1
distractor	43.41	59.09	48.55	55.05
full wiki	19.76	30.42	43.87	50.70

Table 8.7 – Performance breakdown over different question types on the dev set in the distractor setting. "Br" denotes questions collected using bridge entities, and "Cp" denotes comparison questions.

the distractor setting, comparison questions are more challenging than questions involving bridge entities (as defined in Section 8.2), which indicates that this novel question type might not be well-modeled by existing neural architectures. In the full wiki setting, the performance of bridge entity questions drops significantly while that of comparison questions decreases only marginally. This is because both entities usually appear in the comparison questions, and thus reduces the difficulty of retrieval. Combined with the retrieval performance in Table 8.5, we believe that the deterioration in the full wiki setting in Table 8.6 is largely due to the difficulty of retrieving both entities.

We perform an ablation study in the distractor setting, and report the results in Table 8.8. Both self-attention and character-level models contribute notably to the final performance, which is consistent with prior work. This means that techniques targeted at single-hop QA are still somewhat effective in our setting. Moreover, removing strong supervision over supporting facts decreases performance, which demonstrates the effectiveness of our approach and the usefulness of the supporting facts. We establish an estimate of the upper bound of strong supervision by only considering the supporting facts as the oracle context input to our model, which achieves a 10+ F1 improvement over not using the supporting facts. Compared with the gain of strong supervision in our model ($\sim$2 points in F1), our proposed method of incorporating supporting facts supervision is most likely suboptimal, and we leave the challenge of better modeling to future work. At last, we show that combining all data splits (*train-easy*, *train-medium*, and *train-hard*) yields the best performance, which is adopted as the default setting.

Setting	EM	F1
our model	44.44	58.28
– sup fact	42.79	56.19
– sup fact, self attention	41.59	55.19
– sup fact, char model	41.66	55.25
– sup fact, train-easy	41.61	55.12
– sup fact, train-easy, train-medium	31.07	43.61
gold only	48.38	63.58
sup fact only	51.95	66.98

Table 8.8 – Ablation study of question answering performance on the dev set in the distractor setting. "– sup fact" means removing strong supervision over supporting facts from our model. "– train-easy" and "– train-medium" means discarding the according data splits from training. "gold only" and "sup fact only" refer to using the gold paragraphs or the supporting facts as the only context input to the model.

8.5.3 Establishing Human Performance

To establish human performance on our dataset, we randomly sampled 1,000 examples from the dev and test sets, and had at least three additional Turkers provide answers and supporting facts for these examples.

As a baseline, we treat the original Turker during data collection as the prediction, and the newly collected answers and supporting facts as references, to evaluate human performance. For each example, we choose the answer and supporting fact reference that maximize the F1 score to report the final metrics to reduce the effect of ambiguity (Rajpurkar et al., 2016).

As can be seen in Table 8.9, the original crowd worker archives very high performance in both finding supporting facts, and answering the question correctly. If the baseline model were provided with the correct supporting paragraphs to begin with, it achieves parity with the crowd worker in finding supporting facts, but still falls short at finding the actual answer. When distractor paragraphs are present, the performance gap between the baseline model and the crowd worker on both tasks is enlarged to ∼30% for both EM and F1.

We further establish the upper bound of human performance in HotpotQA, by taking the maximum EM and F1 for each example. Here, we use each Turker's answer in turn as the prediction, and evaluate it against all other workers' answers.

Setting	Answer		Sp Fact		Joint	
	EM	F1	EM	F1	EM	F1
gold only	65.87	74.67	59.76	90.41	41.54	68.15
distractor	60.88	68.99	30.99	74.67	20.06	52.37
Human	83.60	91.40	61.50	90.04	52.30	82.55
Human UB	96.80	98.77	87.40	97.56	84.60	96.37

Table 8.9 – Comparing baseline model performance with human performance on 1,000 random samples. "Human UB" stands for the upper bound on annotator performance on HotpotQA. For details please refer to the main body.

As can be seen in Table 8.9, most of the metrics are close to 100%, illustrating that on most examples, at least a subset of Turkers agree with each other, showing high inter-annotator agreement. We also note that crowd workers agree less on supporting facts, which could reflect that this task is inherently more subjective than answering the question.

8.6 Related Work

Various recently-proposed large-scale datasets have been constructed in recent years. We categorize them in four categories in our discussion.

Single-document datasets. SQuAD (Rajpurkar et al., 2016, 2018) questions that are relatively simple because they usually require no more than one sentence in the paragraph to answer.

Multi-document datasets. TriviaQA Joshi et al. (2017b) and SearchQA Dunn et al. (2017) contain question answer pairs that are accompanied with more than one document as the context. This further challenges QA systems' ability to accommodate longer contexts. However, since the supporting documents are collected after the question answer pairs with information retrieval, the questions are not guaranteed to involve interesting reasoning between multiple documents.

KB-based multi-hop datasets. Recent datasets like WikiHop (Welbl et al., 2018a) and ComplexWebQ (Talmor and Berant, 2018) explore different approaches of using pre-existing knowledge bases (KB) with pre-defined logic rules to generate valid QA pairs, to test QA models' capability of performing multi-hop reasoning. The diversity of questions and answers is largely limited by the fixed KB schemas or logical forms. Furthermore, some of the questions might be answerable by one text sentence due to the incompleteness of KBs.

Free-form answer-generation datasets. MS MARCO (Nguyen et al., 2016) contains 100k user queries from Bing Search with human generated answers. Systems generate free-form answers and are evaluated by automatic metrics such as ROUGE-L and BLEU-1. However, the reliability of these metrics is questionable because they have been shown to correlate poorly with human judgement (Novikova et al., 2017).

8.7 Conclusions

We present HotpotQA, a large-scale question answering dataset aimed at facilitating the development of QA systems capable of performing explainable, multi-hop reasoning over diverse natural language. We also offer a new type of factoid comparison questions to test systems' ability to extract and compare various entity properties in text.

9 Prologue to Fourth Article

9.1 Article Detail

Personalizing Dialogue Agents: I have a dog, do you have pets too? Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, Jason Weston. Annual Meeting of the Association for Computational Linguistics (ACL), 2018.

Personal Contribution. The original idea of persona based chitchat was derived from a series of discussions among Jason Weston, Arthur Szlam, Douwe Kiela and me. With the help and suggestions from all the co-authors, I implemented the pipeline for collecting personas and dialogues via crowd sourcing, performed experiments on end-to-end neural models and conducted the original human evaluation. Jason Weston evaluated the performance of ranking based models and further proposed the persona prediction task. Emily Dinan rewrote the entire framework and made critical improvements on both experiments and human evaluation. Jack Urbanek provided solid engineering support through the entire project.

9.2 Context

Chit-chat models are known to have several problems: they lack specificity, do not display a consistent personality and are often not very captivating. In this work we present the task of making chit-chat more engaging by conditioning on profile information. We collect data and train models to (1) condition on their given profile information; and (2) information about the person they are talking to, resulting in improved dialogues, as measured by next utterance prediction. Since (2) is initially unknown, our model is trained to engage its partner with personal topics, and we show the resulting dialogue can be used to predict profile information about the interlocutors.

9.3 Contributions

The PERSONA-CHAT dataset in this work is the very first large scale dataset that allows explicitly modeling chatbot personalities using natural language profiles under a chit-chat environment. Compared with existing resources, models trained on PERSONA-CHAT expressed better consistency and more engagingness during a conversation. Furthermore, the proposed profile prediction task moves chit-chat in the direction of goal-directed dialogue which has metrics for success. Indeed, the original and rephrased profiles collected are also interesting as a semantic similarity dataset in their own right. We hope that the dataset will aid training agents that can ask questions about users' profiles, remember the answers, and use them naturally in conversation.

10 Personalizing Dialogue Agents: I have a dog, do you have pets too?

10.1 Introduction

Despite much recent success in natural language processing and dialogue research, communication between a human and a machine is still in its infancy. It is only recently that neural models have had sufficient capacity and access to sufficiently large datasets that they appear to generate meaningful responses in a chit-chat setting. Still, conversing with such generic chit-chat models for even a short amount of time quickly exposes their weaknesses (Serban et al., 2016a; Vinyals and Le, 2015).

Common issues with chit-chat models include: (i) the lack of a consistent personality (Li et al., 2016a) as they are typically trained over many dialogs each with different speakers, (ii) the lack of an explicit long-term memory as they are typically trained to produce an utterance given only the recent dialogue history (Vinyals and Le, 2015); and (iii) a tendency to produce non-specific answers like "I don't know" (Li et al., 2015). Those three problems combine to produce an unsatisfying overall experience for a human to engage with. We believe some of those problems are due to there being no good publicly available dataset for general chit-chat.[1].

Because of the low quality of current conversational models, and because of the difficulty in evaluating these models, chit-chat is often ignored as an end-application. Instead, the research community has focused on task-oriented communication, such as airline or restaurant booking (Bordes and Weston, 2016), or else single-turn information seeking, i.e. question answering (Rajpurkar et al., 2016). Despite the success of the latter, simpler, domain, it is well-known that a large quantity of human dialogue centers on socialization, personal interests and chit-chat (Dunbar et al., 1997). For example, less than 5% of posts on Twitter are

---

[1] For example, currently the most general chit-chat dataset available in `http://parl.ai` a large repository of dialogue datasets is probably OpenSubtitles, which is based on movie scripts, not natural conversations.

questions, whereas around 80% are about personal emotional state, thoughts or activities, authored by so called “Meformers” (Naaman et al., 2010).

In this work we make a step towards more engaging chit-chat dialogue agents by endowing them with a configurable, but persistent persona, encoded by multiple sentences of textual description, termed a profile. This profile can be stored in a memory-augmented neural network and then used to produce more personal, specific, consistent and engaging responses than a persona-free model, thus alleviating some of the common issues in chit-chat models. Using the same mechanism, any existing information about the persona of the dialogue partner can also be used in the same way. Our models are thus trained to both ask and answer questions about personal topics, and the resulting dialogue can be used to build a model of the persona of the speaking partner.

To support the training of such models, we present the PERSONA-CHAT dataset, a new dialogue dataset consisting of 162,064 utterances between crowdworkers who were randomly paired and each asked to act the part of a given provided persona (randomly assigned, and created by another set of crowdworkers). The paired workers were asked to chat naturally and to get to know each other during the conversation. This produces interesting and engaging conversations that our agents can try to learn to mimic.

Studying the next utterance prediction task during dialogue, we compare a range of models: both generative and ranking models, including Seq2Seq models and Memory Networks (Sukhbaatar et al., 2015b) as well as other standard retrieval baselines. We show experimentally that in either the generative or ranking case conditioning the agent with persona information gives improved prediction of the next dialogue utterance. The PERSONA-CHAT dataset is designed to facilitate research into alleviating some of the issues that traditional chit-chat models face, and with the aim of making such models more consistent and engaging, by endowing them with a persona. By comparing against chit-chat models built using the OpenSubtitles and Twitter datasets, human evaluations show that our dataset provides more engaging models, that are simultaneously capable of being fluent and consistent via conditioning on a persistent, recognizable profile.

10.2 Related Work

Traditional dialogue systems consist of building blocks, such as dialogue state tracking components and response generators, and have typically been applied to tasks with labeled internal dialogue state and precisely defined user intent (i.e., goal-oriented dialogue), see e.g. Young (2000). The most successful goal-oriented dialogue systems model conversation as partially observable Markov decision processes (POMDPs) (Young et al., 2013). All those methods typically do not consider the chit-chat setting and are more concerned with achieving functional goals (e.g. booking an airline flight) than displaying a personality. In particular, many of the tasks and datasets available are constrained to narrow domains (Serban et al., 2015).

Non-goal driven dialogue systems go back to Weizenbaum's famous program ELIZA (Weizenbaum, 1966), and hand-coded systems have continued to be used in applications to this day. For example, modern solutions that build an open-ended dialogue system to the Alexa challenge combine hand-coded and machine-learned elements (Serban et al., 2017a). Amongst the simplest of statistical systems that can be used in this domain, that are based on data rather than hand-coding, are information retrieval models (Sordoni et al., 2015), which retrieve and rank responses based on their matching score with the recent dialogue history. We use IR systems as a baseline in this work.

End-to-end neural approaches are a class of models which have seen growing recent interest. A popular class of methods are generative recurrent systems like seq2seq applied to dialogue (Sutskever et al., 2014; Vinyals and Le, 2015; Sordoni et al., 2015; Li et al., 2016b; Serban et al., 2017b). Their strengths are that (i) they are not constrained by hard-code rules or explicit internal states that may work well in a narrow domain, but are too restrictive for more open dialogue such as chit-chat, and (ii) being based on architectures rooted in language modeling and machine translation, they excel at generating syntactically coherent language, and can generate entirely novel responses. Their deficiencies are that they typically need a large amount of data to be trained, and the vanilla approach generates responses given only the recent dialogue history without using other external memory. The latter issue makes neural models hence typically lack both domain knowledge in the domain being discussed, and a persistent personality during discussions.

A promising direction, that is still in its infancy, to fix this issue is to use a memory-augmented network instead (Sukhbaatar et al., 2015b; Dodge et al., 2015) by providing or learning appropriate memories. A related class of neural methods is to retrieve and rank candidates rather than generate words, similarly to IR methods, but using memory-augmented networks to score the candidates instead. We compare the generative and ranking approaches to each other in this work.

Serban et al. (2015) list available corpora for training dialogue systems. Perhaps the most relevant to learning chit-chat models are ones based on movie scripts such as OpenSubtitles and Cornell Movie-Dialogue Corpus, and dialogue from web platforms such as Reddit and Twitter, all of which have been used for training neural approaches (Vinyals and Le, 2015; Dodge et al., 2015; Li et al., 2016b; Serban et al., 2017b). Naively training on these datasets leads to models with the lack of a consistent personality as they will learn a model averaged over many different speakers. Moreover, the data does little to encourage the model to engage in understanding and maintaining knowledge of the dialogue partner's personality and topic interests.

According to the survey from Serban et al. (2015), personalization of dialogue systems is "an important task, which so far has not received much attention". In the case of goal-oriented dialogue some work has focused on the agent being aware of the human's profile and adjusting the dialogue accordingly, but without a personality to the agent itself (Lucas et al., 2009; Joshi et al., 2017a). For the chit-chat setting, the most relevant work is (Li et al., 2016a). For each user in the Twitter corpus, personas were captured via distributed embeddings (one per speaker) to encapsulate individual characteristics such as background information and speaking style, and they then showed using those vectors improved the output of their seq2seq model for the same speaker. Their work does not focus on attempting to engage the other speaker by getting to know them, as we do here. For that reason, our focus is on explicit profile information, not hard-to-interpret latent variables.

10.3 The PERSONA-CHAT Dataset

The aim of this work is to facilitate more engaging and more personal chit-chat dialogue. The PERSONA-CHAT dataset is a crowd-sourced dataset, collected

Original Persona	**Revised Persona**
I love the beach. My dad has a car dealership I just got my nails done I am on a diet now Horses are my favorite animal.	To me, there is nothing like a day at the seashore. My father sales vehicles for a living. I love to pamper myself on a regular basis. I need to lose weight. I am into equestrian sports.
I am an eccentric hair stylist for dogs My favorite past time is collecting Civil War antiques. I fake a British accent to seem more attractive. I have been married four times and widowed three. I have an allergy to mangoes	I work with animals. I like finding or buying historical artifacts. I heard girls liked foreigners. I have a lot of experience with marriage I have reactions to certain fruits.
I play a lot of fantasy videogames. I have a computer science degree. My mother is a medical doctor I am very shy. I like to build model spaceships.	RPGs are my favorite genre. I also went to school to work with technology. The woman who gave birth to me is a physician. I am not a social person. I enjoy working with my hands.

Table 10.1 – Example Personas (left) and their revised versions (right) from the PERSONA-CHAT dataset. The revised versions are designed to be characteristics that the same persona might have, which could be rephrases, generalizations or specializations.

via Amazon Mechanical Turk, where each of the pair of speakers condition their dialogue on a given profile, which is provided.

The data collection consists of three stages:

(i) Personas: we crowdsource a set of 1155 possible personas, each consisting of at least 5 profile sentences, setting aside 100 never seen before personas for validation, and 100 for test.

(ii) Revised personas: to avoid modeling that takes advantage of trivial word overlap, we crowdsource additional rewritten sets of the same 1155 personas, with related sentences that are rephrases, generalizations or specializations, rendering the task much more challenging.

(iii) Persona chat: we pair two Turkers and assign them each a random (original) persona from the pool, and ask them to chat. This resulted in a dataset of 162,064 utterances over 10,907 dialogs, 15,602 utterances (1000 dialogs) of which are set aside for validation, and 15,024 utterances (968 dialogs) for test.

The final dataset and its corresponding data collection source code, as well as models trained on the data, are all available open source in ParlAI[2].

In the following, we describe each data collection stage and the resulting tasks in more detail.

---

[2] https://github.com/facebookresearch/ParlAI/tree/master/projects/personachat

10.3.1 Personas

We asked the crowdsourced workers to create a character (persona) description using 5 sentences, providing them only a single example:

"I am a vegetarian. I like swimming. My father used to work for Ford. My favorite band is Maroon5. I got a new job last month, which is about advertising design."

Our aim was to create profiles that are natural and descriptive, and contain typical topics of human interest that the speaker can bring up in conversation. Because the personas are not the real profiles of the Turkers, the dataset does not contain personal information (and they are told specifically not to use any). We asked the workers to make each sentence short, with a maximum of 15 words per sentence. This is advantageous both for humans and machines: if they are too long, crowdsourced workers are likely to lose interest, and for machines the task could become more difficult.

Some examples of the personas collected are given in Table 10.1 (left).

10.3.2 Revised Personas

A difficulty when constructing dialogue datasets, or text datasets in general, is that in order to encourage research progress, the task must be carefully constructed so that is neither too easy nor too difficult for the current technology (Voorhees et al., 1999). One issue with conditioning on textual personas is that there is a danger that humans will, even if asked not to, unwittingly repeat profile information either verbatim or with significant word overlap. This may make any subsequent machine learning tasks less challenging, and the solutions will not generalize to more difficult tasks. This has been a problem in some recent datasets: for example, the dataset curation technique used for the well-known SQuAD dataset suffers from this word overlap problem to a certain extent (Chen et al., 2017).

To alleviate this problem, we presented the original personas we collected to a new set of crowdworkers and asked them to rewrite the sentences so that a new sentence is about *"a related characteristic that the same person may have"*, hence the revisions could be rephrases, generalizations or specializations. For example *"I like basketball"* can be revised as *"I am a big fan of Michael Jordan"* not because they mean the same thing but because the same persona could contain both.

In the revision task, workers are instructed not to trivially rephrase the sentence by copying the original words. However, during the entry stage if a non-stop word is copied we issue a warning, and ask them to rephrase, guaranteeing that the instructions are followed. For example, *"My father worked for Ford."* can be revised to *"My dad worked in the car industry"*, but not *"My dad was employed by Ford."* due to word overlap.

Finally, we encourage the construction of natural sentences. In earlier versions of the task we noticed that the word overlap constraint caused unwanted unnatural constructions such as *"I like eating pretzels"* revised as *"I like to chew and swallow twisted bread with salt"*. Giving explicit instructions about this seemed to help, where we prefer a revision like *"I enjoy beers and beer snacks"*.

Some examples of the revised personas collected are given in Table 10.1 (right).

10.3.3 Persona Chat

After collecting personas, we then collected the dialogues conditioned on the personas. For each dialogue, we paired two random crowdworkers, and gave them the instruction that they will chit-chat with another worker, while playing the part of a given character. We then provide them with a randomly chosen persona from our pool, different to their partners. The instructions are on purpose quite terse and simply ask them to "chat with the other person naturally and try to get to know each other". In an early study we noticed the crowdworkers tending to talk about themselves (their own persona) too much, so we also added the instructions "both ask questions and answer questions of your chat partner" which seemed to help. We also gave a bonus for high quality dialogs. The dialog is turn-based, with a maximum of 15 words per message. We again gave instructions to not trivially copy the character descriptions into the messages, but also wrote explicit code sending them an error if they tried to do so, using simple string matching. We define a minimum dialogue length which is randomly between 6 and 8 turns each for each dialogue. An example dialogue from the dataset is given in Table 10.2.

10.3.4 Evaluation

We focus on the standard dialogue task of predicting the next utterance given the dialogue history, but consider this task both with and without the profile in-

Persona 1	**Persona 2**
I like to ski	I am an artist
My wife does not like me anymore	I have four children
I have went to Mexico 4 times this year	I recently got a cat
I hate Mexican food	I enjoy walking for exercise
I like to eat cheetos	I love watching Game of Thrones

[PERSON 1:] Hi
[PERSON 2:] Hello ! How are you today ?
[PERSON 1:] I am good thank you , how are you.
[PERSON 2:] Great, thanks ! My children and I were just about to watch Game of Thrones.
[PERSON 1:] Nice ! How old are your children?
[PERSON 2:] I have four that range in age from 10 to 21. You?
[PERSON 1:] I do not have children at the moment.
[PERSON 2:] That just means you get to keep all the popcorn for yourself.
[PERSON 1:] And Cheetos at the moment!
[PERSON 2:] Good choice. Do you watch Game of Thrones?
[PERSON 1:] No, I do not have much time for TV.
[PERSON 2:] I usually spend my time painting: but, I love the show.

Table 10.2 – Example dialog from the PERSONA-CHAT dataset. Person 1 is given their own persona (top left) at the beginning of the chat, but does not know the persona of Person 2, and vice-versa. They have to get to know each other during the conversation.

formation being given to the learning agent. Our goal is to enable interesting directions for future research, where chatbots can for instance have personalities, or imputed personas could be used to make dialogue more engaging to the user.

We consider this in four possible scenarios: conditioning on no persona, your own persona, their persona, or both. These scenarios can be tried using either the original personas, or the revised ones. We then evaluate the task using three metrics: (i) the log likelihood of the correct sequence, measured via perplexity, (ii) F1 score, and (iii) next utterance classification loss, following Lowe et al. (2015). The latter consists of choosing N random distractor responses from other dialogues (in our setting, N=19) and the model selecting the best response among them, resulting in a score of one if the model chooses the correct response, and zero otherwise (called hits@1 in the experiments).

10.4 Models

We consider two classes of model for next utterance prediction: ranking models and generative models. Ranking models produce a next utterance by considering any utterance in the training set as a possible candidate reply. Generative models generate novel sentences by conditioning on the dialogue history (and possibly, the persona), and then generating the response word-by-word. Note one can still evaluate the latter as ranking models by computing the probability of generating a given candidate, and ranking candidates by those scores.

10.4.1 Baseline ranking models

We first consider two baseline models, an IR baseline Sordoni et al. (2015) and a supervised embedding model, Starspace Wu et al. (2017) [3]. While there are many IR variants, we adopt the simplest one: find the most similar message in the (training) dataset and output the response from that exchange. Similarity is measured by the tf-idf weighted cosine similarity between the bags of words. Starspace is a recent model that also performs information retrieval but by learning the similarity between the dialog and the next utterance by optimizing the embeddings directly for that task using the margin ranking loss and k-negative sampling. The similarity function $sim(q, c')$ is the cosine similarity of the sum of word embeddings of the query q and candidate c'. Denoting the dictionary of $\mathcal{D}$ word embeddings as W which is a $\mathcal{D} \times d$ matrix, where W_i indexes the ith word (row), yielding its d-dimensional embedding, it embeds the sequences q and c'.

In both methods, IR and StarSpace, to incorporate the profile we simply concatenate it to the query vector bag of words.

10.4.2 Ranking Profile Memory Network

Both the previous models use the profile information by combining it with the dialogue history, which means those models cannot differentiate between the two when deciding on the next utterance. In this model we instead use a memory network with the dialogue history as input, which then performs attention over the profile to find relevant lines from the profile to combine with the input, and

---

[3] `github.com/facebookresearch/StarSpace`

then finally predicts the next utterance. We use the same representation and loss as in the Starspace model, so without the profile, the two models are identical. When the profile is available attention is performed by computing the similarity of the input q with the profile sentences p_i, computing the softmax, and taking the weighted sum:

$$q^+ = q + \sum s_i p_i, \qquad s_i = \text{Softmax}(sim(q, p_i))$$

where $\text{Softmax}(z_i) = e^{z_i} / \sum_j e^{z_j}$. One can then rank the candidates c' using $sim(q^+, c')$. One can also perform multiple "hops" of attention over the profile rather than one, as shown here, although that did not bring significant gains in our parameter sweeps.

10.4.3 Key-Value Profile Memory Network

The key-value (KV) memory network Miller et al. (2016) was proposed as an improvement to the memory network by performing attention over keys and outputting the values (instead of the same keys as in the original), which can outperform memory networks dependent on the task and definition of the key-value pairs. Here, we apply this model to dialogue, and consider the keys as dialog histories (from the training set), and the values as the next dialogue utterances, i.e., the replies from the speaking partner. This allows the model to have a memory of past dialogues that it can directly use to help influence its prediction for the current conversation. The model we choose is identical to the profile memory network just described in the first hop over profiles, while in the second hop, q^+ is used to attend over the keys and output a weighted sum of values as before, producing q^{++}. This is then used to rank the candidates c' using $sim(q^{++}, c')$ as before. As the set of (key-value) pairs is large this would make training very slow. In our experiments we simply trained the profile memory network and used the same weights from that model and applied this architecture at test time instead. Training the model directly would presumably give better results, however this heuristic already proved beneficial compared to the original network.

10.4.4 Seq2Seq

The input sequence x is encoded by applying $h_t^e = LSTM_{enc}(x_t \mid h_{t-1}^e)$. We use GloVe (Pennington et al., 2014) for our word embeddings. The final hidden state, h_t^e, is fed into the decoder $LSTM_{dec}$ as the initial state h_0^d. For each time step t, the decoder then produces the probability of a word j occurring in that place via the softmax, i.e.,

$$p(y_{t,j} = 1 \mid y_{t-1}, \ldots, y_1) = \frac{\exp(w_j h_t^d)}{\sum_{j'=1}^{K} \exp(w_{j'} h_t^d)}.$$

The model is trained via negative log likelihood. The basic model can be extended to include persona information, in which case we simply prepend it to the input sequence x, i.e., $x = \forall p \in P \mid\mid x$, where $\mid\mid$ denotes concatenation. For the OpenSubtitles and Twitter datasets trained in Section 10.5.2 we found training a language model (LM), essentially just the decoder part of this model, worked better and we report that instead.

10.4.5 Generative Profile Memory Network

Finally, we introduce a generative model that encodes each of the profile entries as individual memory representations in a memory network. As before, the dialogue history is encoded via $LSTM_{enc}$, the final state of which is used as the initial hidden state of the decoder. Each entry $p_i = \langle p_{i,1}, \ldots, p_{i,n} \rangle \in P$ is then encoded via $f(p_i) = \sum_j^{|p_i|} \alpha_i p_{i,j}$. That is, we weight words by their inverse term frequency: $\alpha_i = 1/(1 + log(1 + \text{tf}))$ where tf is computed from the GloVe index via Zipf's law [4]. Let F be the set of encoded memories. The decoder now attends over the encoded profile entries, i.e., we compute the mask a_t, context c_t and next input $\hat{x}_t$ as:

$$a_t = softmax(FW_a h_t^d),$$
$$c_t = a_t^{\intercal} F; \quad \hat{x}_t = tanh(W_c[c_{t-1}, x_t]).$$

If the model has no profile information, and hence no memory, it becomes equivalent to the Seq2Seq model.

---

[4] $\text{tf} = 1e6 * 1/(idx^{1.07})$

10.5 Experiments

We first report results using automated evaluation metrics, and subsequently perform an extrinsic evaluation where crowdsourced workers perform a human evaluation of our models.

10.5.1 Automated metrics

The main results are reported in Table 10.3. Overall, the results show the following key points:

Persona Conditioning Most models improve significantly when conditioning prediction on their own persona at least for the original (non-revised) versions, which is an easier task than the revised ones which have no word overlap. For example, the Profile Memory generation model has improved perplexity and hits@1 compared to Seq2Seq, and all the ranking algorithms (IR baseline, Starspace and Profile Memory Networks) obtain improved hits@1.

Ranking vs. Generative. Ranking models are far better than generative models at ranking. This is perhaps obvious as that is the metric they are optimizing, but still the performance difference is quite stark. It may be that the word-based probability which generative models use works well, but is not calibrated well enough to give a sentence-based probability which ranking requires. Human evaluation is also used to compare these methods, which we perform in Section 10.5.2.

Ranking Models. For the ranking models, the IR baseline is outperformed by Starspace due to its learnt similarity metric, which in turn is outperformed by Profile Memory networks due to the attention mechanism over the profiles (as all other parts of the models are the same). Finally KV Profile Memory networks outperform Profile Memory Networks in the no persona case due to the ability to consider neighboring dialogue history and next utterance pairs in the training set that are similar to the current dialogue, however when using persona information the performance is similar.

Revised Personas. Revised personas are much harder to use. We do however still see some gain for the Profile Memory networks compared to none (0.354 vs. 0.318 hits@1). We also tried two variants of training: with the original personas in the training set or the revised ones, a comparison of which is shown in Table

Method	**No Persona**		**Original Persona**		**Revised Persona**	
	ppl	**hits@1**	**ppl**	**hits@1**	**ppl**	**hits@1**
Generative Models						
Seq2Seq	38.08	0.092	40.53	0.084	40.65	0.082
Profile Memory	38.08	0.092	34.54	0.125	38.21	0.108
Ranking Models						
IR baseline	-	0.214	-	0.410	-	0.207
Starspace	-	0.318	-	0.491	-	0.322
Profile Memory	-	0.318	-	0.509	-	0.354
KV Profile Memory	-	0.349	-	0.511	-	0.351

Table 10.3 – Evaluation of dialog utterance prediction with various models in three settings: without conditioning on a persona, conditioned on the speakers given persona ("Original Persona"), or a revised persona that does not have word overlap.

Persona	**Method**	**Original**			**Revised**		
		ppl	**hits@1**	**F1**	**ppl**	**hits@1**	**F1**
No Persona		38.08	0.092	0.168	38.08	0.092	0.168
Self Persona	Seq2Seq	40.53	0.084	**0.172**	40.65	0.082	**0.171**
	Profile Memory	**34.54**	**0.125**	**0.172**	38.21	**0.108**	0.170
Their Persona	Seq2Seq	41.48	0.075	0.168	41.95	0.074	0.168
	Profile Memory	36.42	0.105	0.167	**37.75**	0.103	0.167
Both Personas	Seq2Seq	40.14	0.084	0.169	40.53	0.082	0.166
	Profile Memory	35.27	0.115	0.171	38.48	0.106	0.168

Table 10.4 – Evaluation of dialog utterance prediction with generative models in four settings: conditioned on the speakers persona ("self persona"), the dialogue partner's persona ("their persona"), both or none. The personas are either the original source given to Turkers to condition the dialogue, or the revised personas that do not have word overlap. In the "no persona" setting, the models are equivalent, so we only report once.

10.5. *Training* on revised personas helps, both for test examples that are in original form or revised form, likely due to the model be forced to learn more than simple word overlap, forcing the model to generalize more (i.e., learn semantic similarity of differing phrases).

Their Persona. We can also condition a model on the other speaker's persona, or both personas at once, the results of which are in Tables 10.4 and 10.5. Using "Their persona" has less impact on this dataset. We believe this is because most speakers tend to focus on themselves when it comes to their interests. It would be interesting how often this is the case in other datasets. Certainly this is skewed

Method	No Persona Orig	 Rewrite	Self Persona Orig	 Rewrite	Their Persona Orig	 Rewrite	Both Personas Orig	 Rewrite
IR baseline	0.214	0.214	0.410	0.207	0.181	0.181	0.382	0.188
Training on original personas								
Starspace	0.318	0.318	0.481	0.295	0.245	0.235	0.429	0.258
Profile Memory	0.318	0.318	0.473	0.302	0.283	0.267	0.438	0.266
Training on revised personas								
Starspace	0.318	0.318	0.491	0.322	0.271	0.261	0.432	0.288
Profile Memory	0.318	0.318	0.509	0.354	0.299	0.294	0.467	0.331
KV Profile Memory	0.349	0.349	0.511	0.351	0.291	0.289	0.467	0.330

Table 10.5 – Evaluation of dialog utterance prediction with ranking models using hits@1 in four settings: conditioned on the speakers persona (”self persona”), the dialogue partner's persona (”their persona”), both or none. The personas are either the original source given to Turkers to condition the dialogue, or the rewritten personas that do not have word overlap, explaining the poor performance of IR in that case.

by the particular instructions one could give to the crowdworkers. For example if we gave the instructions “try not to talk about yourself, but about the other's interests' likely these metrics would change.

10.5.2 Human Evaluation

As automated metrics are notoriously poor for evaluating dialogue (Liu et al., 2016) we also perform human evaluation using crowdsourced workers. The procedure is as follows. We perform almost exactly the same setup as in the dataset collection process itself as in Section 10.3.3. In that setup, we paired two Turkers and assigned them each a random (original) persona from the collected pool, and asked them to chat. Here, from the Turker's point of view everything looks the same except instead of being paired with a Turker they are paired with one of our models instead (they do not know this). In this setting, for both the Turker and the model, the personas come from the test set pool.

After the dialogue, we then ask the Turker some additional questions in order to evaluate the quality of the model.

They are, in order:

— **Fluency**: We ask them to judge the fluency of the other speaker as a score from 1 to 5, where 1 is “not fluent at all”, 5 is “extremely fluent”, and 3 is “OK”.

Method Model	 Profile	 **Fluency**	 **Engagingness**	 **Consistency**	**Persona** **Detection**
Human	Self	4.31(1.07)	4.25(1.06)	4.36(0.92)	0.95(0.22)
Generative PersonaChat Models					
Seq2Seq	None	3.17(1.10)	3.18(1.41)	2.98(1.45)	0.51(0.50)
Profile Memory	Self	3.08(1.40)	3.13(1.39)	3.14(1.26)	0.72(0.45)
Ranking PersonaChat Models					
KV Memory	None	3.81(1.14)	3.88(0.98)	3.36(1.37)	0.59(0.49)
KV Profile Memory	Self	3.97(0.94)	3.50(1.17)	3.44(1.30)	0.81(0.39)
Twitter LM	None	2.22(1.17)	1.68(0.89)	1.82(0.95)	0.55(0.50)
OpenSubtitles 2018 LM	None	2.46(1.28)	2.0(1.09)	1.98(1.15)	0.52(0.50)
OpenSubtitles 2009 LM	None	2.74(1.39)	1.74(1.02)	1.8(1.17)	0.47(0.50)
OpenSubtitles 2009 KV Memory	None	2.14(1.20)	2.22(1.22)	2.06(1.29)	0.42(0.49)

Table 10.6 – Human Evaluation of various PERSONA-CHAT models, along with a comparison to human performance, and Twitter and OpenSubtitles based models (last 4 rows), standard deviation in parenthesis.

- **Engagingness**: We ask them to judge the engagingness of the other speaker *disregarding fluency* from 1-5, where 1 is "not engaging at all", 5 is "extremely engaging", and 3 is "OK".
- **Consistency**: We ask them to judge the consistency of the persona of the other speaker, where we give the example that "I have a dog" followed by "I have no pets" is not consistent. The score is again from 1-5.
- **Profile Detection**: Finally, we display two possible profiles, and ask which is more likely to be the profile of the person the Turker just spoke to. One profile is chosen at random, and the other is the true persona given to the model.

The results are reported in Table 10.6 for the best performing generative and ranking models, in both the No Persona and Self Persona categories, 100 dialogues each. We also evaluate the scores of human performance by replacing the chatbot with a human (another Turker). This effectively gives us upper bound scores which we can aim for with our models. Finally, and importantly, we compare our models trained on PERSONA-CHAT with chit-chat models trained with the Twitter and OpenSubtitles datasets (2009 and 2018 versions) instead, following Vinyals and Le (2015). Example chats from a few of the models are shown in the Appendix in Tables 10.9, 10.10, 10.11, 10.12, 10.13 and 10.14.

Firstly, we see a difference in fluency, engagingness and consistency between all PERSONA-CHAT models and the models trained on OpenSubtitles and Twitter.

PERSONA-CHAT is a resource that is particularly strong at providing training data for the beginning of conversations, when the two speakers do not know each other, focusing on asking and answering questions, in contrast to other resources. We also see suggestions of more subtle differences between the models, although these differences are obscured by the high variance of the human raters' evaluations. For example, in both the generative and ranking model cases, models endowed with a persona can be detected by the human conversation partner, as evidenced by the persona detection accuracies, whilst maintaining fluency and consistency compared to their non-persona driven counterparts.

Finding the balance between fluency, engagement, consistency, and a persistent persona remains a strong challenge for future research.

10.5.3 Profile Prediction

While the main study of this work is the ability to improve next utterance classification by conditioning on a persona, one could naturally consider two tasks: (1) next utterance prediction during dialogue, and (2) profile prediction given dialogue history. In the main paper we show that Task 1 can be improved by using profile information. Task 2, however, can be used to extract such information.

In this section we conduct a preliminary study of the ability to predict the persona of a speaker given a set of dialogue utterances. We consider the dialogues between humans (PERSON 0) and our best performing model, the retrieval-based Key-Value Profile Memory Network (PERSON 1) from Section 10.5.2. We tested the ability to predict the profile information of the two speakers from the dialogue utterances of each speaker, considering all four combinations. We employ the same IR baseline model used in the main paper to predict profiles: it ranks profile candidates, either at the entire profile level (considering all the sentences that make up the profile as a bag) or at the sentence level (each sentence individually). We consider 100 negative profile candidates for each positive profile, and compute the error rate of predicting the true profile averaged over all dialogues and candidates. The results are given in Table 10.7, both for the model conditioned on profile information, and the same KV Memory model that is not. The results indicate the following:

— It is possible to predict the humans profile from their dialogue utterances

		Profile Level		**Sentence Level**	
Speaker	**Profile**	KV Profile	KV w/o Profile	KV Profile	KV w/o Profile
PERSON 0	Profile 0	0.057	0.017	0.173	0.141
PERSON 0	Profile 1	0.234	0.491	0.431	0.518
PERSON 1	Profile 0	0.254	0.112	0.431	0.349
PERSON 1	Profile 1	0.011	0.512	0.246	0.530

Table 10.7 – Profile Prediction. Error rates are given for predicting either the persona of speaker 0 (Profile 0) or of speaker 1 (Profile 1) given the dialogue utterances of speaker 0 (PERSON 0) or speaker 1 (PERSON 1). This is shown for dialogues between humans (PERSON 0) and either the KV Profile Memory model ("KV Profile") which conditions on its own profile, or the KV Memory model ("KV w/o Profile") which does not.

Speaker	**Profile**	**Dialogue Length**							
		1	2	3	4	5	6	7	8
PERSON 0	Profile 0	0.76	0.47	0.35	0.29	0.23	0.19	0.17	0.17
PERSON 0	Profile 1	0.51	0.39	0.32	0.29	0.27	0.27	0.25	0.25
PERSON 1	Profile 0	0.57	0.52	0.48	0.46	0.45	0.43	0.43	0.43
PERSON 1	Profile 1	0.81	0.58	0.48	0.47	0.45	0.44	0.43	0.43

Table 10.8 – Profile Prediction By Dialog Length. Error rates are given for predicting either the persona of speaker 0 (Profile 0) or of speaker 1 (Profile 1) given the dialogue utterances of speaker 0 (PERSON 0) or speaker 1 (PERSON 1). This is shown for dialogues between humans (PERSON 0) and the KV Profile Memory model averaged over the first N dialogue utterances from 100 conversations (where N is the "Dialogue Length"). The results show the accuracy of predicting the persona improves in all cases as dialogue length increases.

(PERSON 0, Profile 0) with high accuracy at both the profile and sentence level, independent of the model they speaking to.

- Similarly the model's profile can be predicted with high accuracy from its utterances (PERSON 1, Profile 1) when it is conditioned on the profile, otherwise this is chance level (w/o Profile).
- It is possible to predict the model's profile from the human's dialogue, but with a lower accuracy (PERSON 0, Profile 1) as long as the model is conditioned on its own profile. This indicates the human responds to the model's utterances and pays attention to the model's interests.
- Similarly, the human's profile can be predicted from the model's dialogue, but with lower accuracy. Interestingly, the model without profile conditioning is better at this, perhaps because it does not concentrate on talking about itself, and pays more attention to responding to the human's interests.

There appears to be a tradeoff that needs to be explored and understood here.

We also study the performance of profile prediction as the dialogue progresses, by computing error rates for dialogue lengths 1 to 8 (the longest length we consider in this work). The results, given in Table 10.8, show the error rate of predicting the persona decreases in all cases as dialogue length increases.

Overall, the results in this section show that it is plausible to predict profiles given dialogue utterances, which is an important extraction task. Note that better results could likely be achieved with more sophisticated models.

10.6 Conclusion & Discussion

In this work we have introduced the PERSONA-CHAT dataset, which consists of crowd-sourced dialogues where each participant plays the part of an assigned persona; and each (crowd-sourced) persona has a word-distinct paraphrase. We test various baseline models on this dataset, and show that models that have access to their own personas in addition to the state of the dialogue are scored as more consistent by annotators, although not more engaging. On the other hand, we show that models trained on PERSONA-CHAT (with or without personas) are more engaging than models trained on dialogue from other resources (movies, Twitter).

We believe PERSONA-CHAT will be a useful resource for training components of future dialogue systems. Because we have paired human generated profiles and conversations, the data aids the construction of agents that have consistent personalities and viewpoints. Furthermore, predicting the profiles from a conversation moves chit-chat tasks in the direction of goal-directed dialogue, which has metrics for success. Because we collect paraphrases of the profiles, they cannot be trivially matched; indeed, we believe the original and rephrased profiles are interesting as a semantic similarity dataset in their own right. We hope that the data will aid training agents that can ask questions about users' profiles, remember the answers, and use them naturally in conversation.

10.7 Dialogues between Humans and Models

In Tables 10.9, 10.10, 10.11, 10.13, 10.14 we show example dialogues between Turkers and various models collected as part of the human evaluation of Section 10.5.2.

Persona 1	**Persona 2 (Not used by model)**
I am a doctor	I eat sardines for breakfast daily
I have one brother	I just had surgery
My mom is a secretary	I love playing Frisbee
I like to eat a lot of meat	I have two large dogs
I am a bodybuilder	I never finished high school

[PERSON 1:] Hi I am Dr. Jenson. What is your name?
[PERSON 2:] Miss Dawes.
[PERSON 1:] Hi Miss Dawes, tell me about your family!
[PERSON 2:] Miss Baggoli, I know it must show on your records that my name is Mary, but everyone has referred to me as Lola since I was a squealing infant in my mother' s arms.
[PERSON 1:] fascinating. My secretary mother would have made note of all of that.
[PERSON 2:] In addition to the remainder, it writes my autobiography.
[PERSON 1:] Oh yeah? well I enjoy a nice steak. how about you?
[PERSON 2:] Father always used to say that if we ever had the money you have... we' d have steak and ice cream three times every day!
[PERSON 1:] I eat all the steak and chicken too, even bacon
[PERSON 2:] Father always used to say that if we ever had the money you have... we' d have steak and ice cream three times every day!
[PERSON 1:] yeah you said that already
[PERSON 2:] Dynamic structure can absorb shock.
[PERSON 1:] yes it can. I know about that as a bodybuilder
[PERSON 2:] Ladies and gentlemen... the strongest man in the world is blowing up a hot water bottle.

Table 10.9 – Example dialog between a human (Person 1) and the OpenSubtitles KV Memory Network model (Person 2).

Persona 1	**Persona 2 (Not used by model)**
I am afraid of dogs	I am a carpenter
I used to work at a carnival	I used to drink but I stopped five years ago
I like to drink scotch to relax	I have a wife and three kids
I write short stories for a living	I like playing poker
I like dark superhero movies	I dont have many friends

[PERSON 1:] Hi are you relaxing on this fine evening?
[PERSON 2:] I am doing good and you?
[PERSON 1:] I am well, just drinking some scotch and working on my writing
[PERSON 2:] I love to cook and bake but I rarely make time for it
[PERSON 1:] what do you usually bake?
[PERSON 2:] I do not cook, I bake cakes at a local restaurant. how about you?
[PERSON 1:] I used to be a juggler at a carnival but now I am an author
[PERSON 2:] I am a dancer, what do you do?
[PERSON 1:] I just told you. Do you have any pets?
[PERSON 2:] no, I am allergic to pets unfortunately.
[PERSON 1:] I understand. I am not allergic but I am terrified of dogs!
[PERSON 2:] I have a cat. she has been with me through three wives, haha.
[PERSON 1:] Funny that you just told me you have no pets
[PERSON 2:] I do not have any animals.

Table 10.10 – Example dialog between a human (Person 1) and the Seq2Seq model (Person 2).

Persona 1	**Persona 2**
I am engaged to be married	I am a writer
I used to play sports in high school	I write romance novels
My favorite sport is soccer	I live on a faraway island
I am a college graduate	I am an introvert
I love watching sports on TV	I have two cats

[PERSON 1:] I just got done playing soccer. What have you been up to?
[PERSON 2:] I do not have pets. I spend my free time writing. what about you?
[PERSON 1:] Just graduated from college. Looking for work.
[PERSON 2:] yes I graduated, in writing and poetry
[PERSON 1:] Have you had anything published?
[PERSON 2:] I mainly write fiction novels.
[PERSON 1:] Do you like to watch sports?
[PERSON 2:] do you like kings of leon my favorite by them is use somebody
[PERSON 1:] Are you married? I will be married soon.
[PERSON 2:] haha, no time. I have got a novel to finish.
[PERSON 1:] What is your novel about?
[PERSON 2:] it is a love story. I really enjoy writing. do you write or read?
[PERSON 1:] Like to read. Spend most time watching football.
[PERSON 2:] yes I do I am and author for a living

Table 10.11 – Example dialog between a human (Person 1) and the Key-Value Profile Memory Network with Self Persona.

Persona 1	**Persona 2**
I dont like exercise or physical activity	My favorite type of food is Italian
My favorite food is ice cream	I have a big pet dog
I am a homebody	I want to travel to Europe
I have a husband who I love very much	I love reading books about travel
I love watching movies and TV	I like to walk long distances

[PERSON 1:] Hello, how are you doing today?
[PERSON 2:] I am good. I just got finished walking my dog.
[PERSON 1:] Oh cool! I am just sitting watching a movie.
[PERSON 2:] I have been watching that one before. I love it.
[PERSON 1:] Me too. I don't like going out much so I enjoy movies and TV.
[PERSON 2:] that is a great idea. I am going to read a book on writing.
[PERSON 1:] Oh, are you a writer?
[PERSON 2:] yes, I love to read. I am a great author.
[PERSON 1:] What sorts of topics do you write about?
[PERSON 2:] I could read all of those. I am sure you enjoy reading.
[PERSON 1:] I do, but I prefer movies.
[PERSON 2:] I have not, I can not do that. I have to go to the east coast.
[PERSON 1:] Why can't you watch movies?
[PERSON 2:] I have not. I have always had that done.

Table 10.12 – Example dialog between a human (Person 1) and the Generative Profile Memory Network with Self Persona.

Persona 1	**Persona 2 (Not used by model)**
I am a single mom of two boys	I am an omnivore
I live in Ohio	I love the band Rush
I work as an accountant	I work in sales
I like to go hiking in my spare time	Last week I was at a friend's wedding
I drive a Honda Civic	My hobby is graphic design

[PERSON 1:] Hello are you from Ohio too?
[PERSON 2:] I'm from the village.
[PERSON 1:] Thats nice I am in accountant what work do you do?
[PERSON 2:] I work for the company.
[PERSON 1:] That sounds classified. Do you have any children I am a single mother of 2 boys.
[PERSON 2:] I'm not going to lose.
[PERSON 1:] That's how I see hiking in my spare time I will not lose
[PERSON 2:] I'm not afraid of you
[PERSON 1:] thats usually peoples mistake.
[PERSON 2:] I'm sure you think you're a bit of a disappointment.

Table 10.13 – Example dialog between a human (Person 1) and the Language Model trained on the OpenSubtitles 2018 dataset (does not use persona).

Persona 1	**Persona 2 (Not used by model)**
I have a golden retriever named Ginger	I share an appartment in Seattle with two roomates
I just became a first time father	I just graduated from college two years ago
I studied abroad for a year	I gam a kindergarten teacher
I have seen the movie Titanic ten times	I can barely pay my bills every month

[PERSON 1:] Hello. Want to see a picture of my new baby?
[PERSON 2:] I don't think so. I have to be honest, but I just like the way it was.
[PERSON 1:] I understand. I have seen the titanic movie 10 times. The shows I like things the same.
[PERSON 2:] I just got anew iphone 7 plus
[PERSON 1:] That is very cool. I used one of those when I studied abroad.
[PERSON 2:] I think it was a great example of the.
[PERSON 1:] I eventually have to com back and start a family
[PERSON 2:] I know I have to be a friend.
[PERSON 1:] My dog, Ginger, is my best friend.
[PERSON 2:] I love you so much
[PERSON 1:] Too much too soon sorry.
[PERSON 2:] I know. I don't know if I had to.

Table 10.14 – Example dialog between a human (Person 1) and the Language Model trained on the Twitter dataset (does not use persona).

11 Conclusion

RNNs regained the attention from the community in 2014, due to a series of research advances in neural machine translation (Sutskever et al., 2014; Bahdanau et al., 2014) in which improved recurrent models with gating/memorizing mechanisms were widely adopted. Since then, more recurrent models with novel connecting architectures and functionalities were developed and their effectiveness were also proved empirically in a wide range of NLP tasks. As recurrent models become increasingly complicated and powerful, many challenging problems almost untouched before are now feasible, whilst deeper understanding towards various fundamental aspects of RNN becomes imperative.

This thesis provides several new perspectives on both RNN's structural properties and its potentials in recently proposed challenging NLP problems. The first two articles explore the basic architectural and functional properties of RNN under universal condition and propose several principles of how to construct an RNN with better gradient flows and the ability of capturing multi-scale nonlinear dependencies. The third and the fourth article turn to two of the most crucial NLP tasks which are machine reading comprehension and dialogue systems respectively. Specifically, we build two large scale datasets attempting to motivate model/algorithm innovations in problems of advanced multi-step reasoning on natural language documents and integrating external personality information to conduct a more captivating machine-human conversation.

During the time of writing this thesis, we have seen the rise of a new structural design methodology heavily embracing the attention mechanism derived from the original RNN (Vaswani et al., 2017; Devlin et al., 2018). The proposed modules are newly crowned in NLP by leveraging unsupervised pretraining on huge corpus, demonstrating their potentials as general building blocks for understanding natural language. In the coming decade, we expect that RNN and its inheritors will address more essential intelligence problems such as understanding common sense, complex reasoning and flexible utilization of the world knowledge.

Bibliography

Arjovsky, M., Shah, A., and Bengio, Y. (2015). Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*.

Aust, H., Oerder, M., Seide, F., and Steinbiss, V. (1995). The philips automatic train timetable information system. *Speech Communication*, 17(3-4):249–262.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2015). End-to-end attention-based large vocabulary speech recognition. *arXiv preprint arXiv:1508.04395*.

Baum, L. and Eagon, J. (1967). An inequality with application to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of the American Mathematical Society*, 73:360–363.

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166.

Bordes, A. and Weston, J. (2016). Learning end-to-end goal-oriented dialog. *arXiv preprint arXiv:1605.07683*.

Büttcher, S., Clarke, C. L., and Cormack, G. V. (2016). *Information retrieval: Implementing and evaluating search engines*.

Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M. Equations for part-of-speech tagging.

Chen, D. (2018). *Neural Reading Comprehension and Beyond*. PhD thesis, Stanford University.

Chen, D., Bolton, J., and Manning, C. D. (2016). A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858*.

Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading Wikipedia to answer open-domain questions. In *Association for Computational Linguistics (ACL)*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015). The loss surfaces of multilayer networks.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2015). Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*.

Clark, C. and Gardner, M. (2017). Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association of Computational Linguistics*.

Cooijmans, T., Ballas, N., Laurent, C., and Courville, A. (2016). Recurrent batch normalization. *http://arxiv.org/pdf/1603.09025v4.pdf.*

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805.*

Dodge, J., Gane, A., Zhang, X., Bordes, A., Chopra, S., Miller, A., Szlam, A., and Weston, J. (2015). Evaluating prerequisite qualities for learning end-to-end dialog systems. *arXiv preprint arXiv:1511.06931.*

Dunbar, R. I., Marriott, A., and Duncan, N. D. (1997). Human conversational behavior. *Human nature*, 8(3):231–246.

Dunn, M., Sagun, L., Higgins, M., Guney, U., Cirik, V., and Cho, K. (2017). SearchQA: A new Q&A dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179.*

El Hihi, S. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems*, pages 493–499.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G., and Lee, Y. (1991). Second-order recurrent neural networks for grammatical inference. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 273–281. IEEE.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Gorin, A. L., Riccardi, G., and Wright, J. H. (1997). How may i help you? *Speech communication*, 23(1-2):113–127.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., and Chen, D. (1994). First-order versus second-order single-layer recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(3):511–513.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Hannun, A. Y., Maas, A. L., Jurafsky, D., and Ng, A. Y. (2014). First-pass large vocabulary continuous speech recognition using bi-directional recurrent dnns. *arXiv preprint arXiv:1408.2873*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692.

Hermans, M. and Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 190–198.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hirschman, L., Light, M., Breck, E., and Burger, J. D. (1999). Deep read: A reading comprehension system. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 325–332. Association for Computational Linguistics.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Hu, M., Peng, Y., Huang, Z., Qiu, X., Wei, F., and Zhou, M. (2017). Reinforced mnemonic reader for machine reading comprehension. *arXiv preprint arXiv:1705.02798*.

Huang, H.-Y., Zhu, C., Shen, Y., and Chen, W. (2017). Fusionnet: Fusing via fully-aware attention with application to machine comprehension. *arXiv preprint arXiv:1711.07341*.

Hutchens, J. L. and Alder, M. D. (1998). Introducing megahal. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, pages 271–274. Association for Computational Linguistics.

Hutchins, W. J., Dostert, L., and Garvin, P. (1955). The georgetown-ibm experiment. In *In*. Citeseer.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 448–456.

Jordan, M. I. (1997). Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495.

Joshi, C. K., Mi, F., and Faltings, B. (2017a). Personalization in goal-oriented dialog. *arXiv preprint arXiv:1706.07503*.

Joshi, M., Choi, E., Weld, D., and Zettlemoyer, L. (2017b). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1601–1611.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350.

Kalchbrenner, N., Danihelka, I., and Graves, A. (2015). Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in Neural Information Processing Systems*, pages 3276–3284.

Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A clockwork rnn. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1863–1871.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Krueger, D. and Memisevic, R. (2015). Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lehnert, W. G. (1977). The process of question answering. Technical report, YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE.

Li, J., Galley, M., Brockett, C., Gao, J., and Dolan, B. (2015). A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*.

Li, J., Galley, M., Brockett, C., Spithourakis, G. P., Gao, J., and Dolan, B. (2016a). A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*.

Li, J., Monroe, W., Ritter, A., Galley, M., Gao, J., and Jurafsky, D. (2016b). Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.

Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1996). Learning long-term dependencies is not as difficult with NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338.

Lin, Z., Feng, M., Santos, C. N. d., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

Liu, C.-W., Lowe, R., Serban, I. V., Noseworthy, M., Charlin, L., and Pineau, J. (2016). How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation. *arXiv preprint arXiv:1603.08023*.

Liu, X., Shen, Y., Duh, K., and Gao, J. (2018). Stochastic answer networks for machine reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Lowe, R., Pow, N., Serban, I., and Pineau, J. (2015). The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *arXiv preprint arXiv:1506.08909*.

Lucas, J., Fernández, F., Salazar, J., Ferreiros, J., and San Segundo, R. (2009). Managing speaker identity and user profiles in a spoken dialogue system. *Procesamiento del Lenguaje Natural*, 43:77–84.

Manning, C. D., Manning, C. D., and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993). Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

Màrquez, L. and Rodríguez, H. (1998). Part-of-speech tagging using decision trees. In *European Conference on Machine Learning*, pages 25–36. Springer.

McCallum, A., Nigam, K., et al. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

Miao, Y., Gowayyed, M., and Metze, F. (2015). Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding. *arXiv preprint arXiv:1507.08240*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mikolov, T., Sutskever, I., Deoras, A., Le, H.-S., and Kombrink, S. (2012). Subword language modeling with neural networks. *preprint, (http://www.fit.vutbr.cz/imikolov/rnnlm/char.pdf)*.

Miller, A., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016). Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*.

Miller, A. H., Feng, W., Fisch, A., Lu, J., Batra, D., Bordes, A., Parikh, D., and Weston, J. (2017). ParlAI: A dialog research software platform. *arXiv preprint arXiv:1705.06476*.

Minsky, M. and Papert, S. (1969). Perceptrons.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932.

Mrkšić, N., Séaghdha, D. Ó., Thomson, B., Gasic, M., Su, P.-H., Vandyke, D., Wen, T.-H., and Young, S. (2015). Multi-domain dialog state tracking using recurrent 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 (Volume 2: Short Papers)*, volume 2, pages 794–799.

Naaman, M., Boase, J., and Lai, C.-H. (2010). Is it really about me?: message content in social awareness streams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 189–192. ACM.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NIPS)*.

Novikova, J., Dušek, O., Curry, A. C., and Rieser, V. (2017). Why we need new evaluation metrics for NLG. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Pachitariu, M. and Sahani, M. (2013). Regularization and nonlinearities for neural language models: when are they needed? *arXiv preprint arXiv:1301.5650*.

Pan, B., Li, H., Zhao, Z., Cao, B., Cai, D., and He, X. (2017). Memen: Multi-layer embedding with memory networks for machine comprehension. *arXiv preprint arXiv:1707.09098*.

Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013a). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

Pascanu, R., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). How to construct deep recurrent neural networks. In *International Conference on Learning Representations 2014(Conference Track)*.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013b). On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1310–1318.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pezeshki, M., Fan, L., Brakel, P., Courville, A., and Bengio, Y. (2015). Deconstructing the ladder network architecture. *arXiv preprint arXiv:1511.06430*.

Raiko, T., Valpola, H., and LeCun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In *International Conference on Artificial Intelligence and Statistics*, pages 924–932.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 784–789.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Rasmus, A., Valpola, H., Honkala, M., Berglund, M., and Raiko, T. (2015). Semi-supervised learning with ladder network. *arXiv preprint arXiv:1507.02672*.

Reithinger, N. and Klesen, M. (1997). Dialogue act classification using language models. In *Fifth European Conference on Speech Communication and Technology*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Richardson, M., Burges, C. J., and Renshaw, E. (2013). Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings* *of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203.

Ritter, A., Cherry, C., and Dolan, B. (2010). Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 172–180. Association for Computational Linguistics.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Salant, S. and Berant, J. (2018). Contextualized word representations for reading comprehension. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Schank, R. C. and Abelson, R. P. (1977). Scripts, plans, goals and understanding: An inquiry into human knowledge structures.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Schrading, N., Alm, C. O., Ptucha, R., and Homan, C. (2015). An analysis of domestic abuse discourse on reddit. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2577–2583.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Seo, M., Kembhavi, A., Farhadi, A., and Hajishirzi, H. (2017). Bidirectional attention flow for machine comprehension. In *Proceedings of the International Conference on Learning Representations*.

Serban, I. V., Lowe, R., Charlin, L., and Pineau, J. (2016a). Generative deep neural networks for dialogue: A short review. *arXiv preprint arXiv:1611.06216*.

Serban, I. V., Lowe, R., Henderson, P., Charlin, L., and Pineau, J. (2015). A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*.

Serban, I. V., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., Kim, T., Pieper, M., Chandar, S., Ke, N. R., et al. (2017a). A deep reinforcement learning chatbot. *arXiv preprint arXiv:1709.02349*.

Serban, I. V., Sordoni, A., Bengio, Y., Courville, A., and Pineau, J. (2016b). Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Serban, I. V., Sordoni, A., Lowe, R., Charlin, L., Pineau, J., Courville, A. C., and Bengio, Y. (2017b). A hierarchical latent variable encoder-decoder model for generating dialogues.

Siegelmann, H. T. (1999). Computation beyond the turing limit. In *Neural Networks and Analog Computation*, pages 153–164. Springer.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Singh, S. P., Kearns, M. J., Litman, D. J., and Walker, M. A. (2000). Reinforcement learning for spoken dialogue systems. In *Advances in Neural Information Processing Systems*, pages 956–962.

Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, B. (2015). A neural network approach to context-sensitive generation of conversational responses. *arXiv preprint arXiv:1506.06714*.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Stolcke, A. and Omohundro, S. (1994). Inducing probabilistic grammars by bayesian model merging. In *International Colloquium on Grammatical Inference*, pages 106–118. Springer.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015a). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015b). End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Sutskever, I. and Hinton, G. (2010). Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Talmor, A. and Berant, J. (2018). The web as a knowledge-base for answering complex questions. In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics.*

Telgarsky, M. Benefits of depth in neural networks.

Tiedemann, J. (2012). Parallel data, tools and interfaces in opus. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012).*

Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer.

Vapnik, V. N. *Statistical learning theory*, volume 1.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Vinyals, O. and Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Voorhees, E. M. et al. (1999). The trec-8 question answering track report. In *Trec*, volume 99, pages 77–82.

Wallace, R. S. (2009). The anatomy of alice. In *Parsing the Turing Test*, pages 181–210. Springer.

Wang, H., Bansal, M., Gimpel, K., and McAllester, D. (2015). Machine comprehension with syntax, frames, and semantics. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 700–706.

Wang, S. and Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*.

Wang, W., Yang, N., Wei, F., Chang, B., and Zhou, M. (2017). Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198.

Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

Welbl, J., Stenetorp, P., and Riedel, S. (2018a). Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association of Computational Linguistics*.

Welbl, J., Stenetorp, P., and Riedel, S. (2018b). Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association of Computational Linguistics*, 6:287–302.

Wen, T.-H., Vandyke, D., Mrksic, N., Gasic, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., and Young, S. (2016). A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*.

Werbos, P. (1974). Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Wessels, T. and Omlin, C. W. (2000). Refining hidden markov models with recurrent neural networks. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 2, pages 271–276 vol.2.

Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

Winograd, T. (1971). Procedures as a representation for data in a computer program for understanding natural language. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE PROJECT MAC.

Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., and Weston, J. (2017). Starspace: Embed all the things! *arXiv preprint arXiv:1709.03856*.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. R. (2016). On multiplicative integration with recurrent neural networks. In *Advances in neural information processing systems*, pages 2856–2864.

Xiong, C., Zhong, V., and Socher, R. (2016). Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*.

Xiong, C., Zhong, V., and Socher, R. (2018). DCN+: Mixed objective and deep residual coattention for question answering. In *Proceedings of the International Conference on Learning Representations*.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. (2018a). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Yang, Z., Zhang, S., Urbanek, J., Feng, W., Miller, A. H., Szlam, A., Kiela, D., and Weston, J. (2018b). Mastering the dungeon: Grounded language learning by mechanical turker descent. In *Proceedings of the International Conference on Learning Representations*.

Yin, W., Ebert, S., and Schütze, H. (2016). Attention-based convolutional neural network for machine comprehension. *arXiv preprint arXiv:1602.04341*.

Young, S., Gašić, M., Thomson, B., and Williams, J. D. (2013). Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179.

Young, S. J. (2000). Probabilistic methods in spoken–dialogue systems. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 358(1769):1389–1402.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer.

Zhang, S., Dinan, E., Urbanek, J., Szlam, A., Kiela, D., and Weston, J. (2018). Personalizing dialogue agents: I have a dog, do you have pets too? In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2204–2213.

Zhang, S., Wu, Y., Che, T., Lin, Z., Memisevic, R., Salakhutdinov, R. R., and Bengio, Y. (2016). Architectural complexity measures of recurrent neural networks. In *Advances in neural information processing systems*, pages 1822–1830.