Extended QA System on SQuAD 2.0

Stanford CS224N Default Project

Lingyan Hao
Department of Computer Science
Stanford University
lingyanh@stanford.edu

Yi Wei
Department of Computer Science
Stanford University
yvetteww@stanford.edu

Jialin Zhu
Department of Computer Science
Stanford University
ejlzhu@stanford.edu

Abstract

Our motivation is to build a Question Answering (QA) system that gives answers as specific and as accurate to queries, which is in itself an art but based on the science of Natural Language Processing (NLP). Some efforts have been made to either increase the scale of the dataset (semi-annotation, SQuAD), or to improve the NLP structure (Statistical QA, Neural QA). The main goal of our project is to produce a QA system that works well on SQuAD 2.0 dataset that performs better than the baseline Bidirectional Attention Flow (BiDAF) model. To better capture the context from a more expressive set of answers and understand the interactions between the question and the document, we utilized the coattention mechanism by encoding the two-way attention outputs together through a bidirectional reccurrent neural network (RNN). For our main finding, we aim to implement the mechanisms that would improve upon the baseline method under the Exact Match (EM) and the F1 scores. In the experiment, our best performing single model obtained a F1 score of 63.40 and an EM score of 59.87 which both achieved better results than the baseline. To further improve the performance of our model, we built ensemble model that achieved EM of 60.81 and F1 of 64.21.

1 Introduction

Question Answering (QA) has been one of the most studied fundamental problems in Natural Language Processing (NLP). The machine comprehension of natural language is the ability to process text and translate sentences into internal representations so that the system can generate relevant answers to the questions. A variety of recent researches has achieved near human performance in various open domain QA tasks including choosing from multiple possible answers or span predictions. One of the most popular span prediction datasets for these tasks is the Stanford Question Answering Dataset (SQuAD). In this paper, we focus on SQuAD 2.0 dataset, which includes over 50,000 unanswerable questions and 100,000 answerable questions extracted from Wikipedia pages.

The problems that underlie the traditional approaches are that sometimes the models would attend to irrelevant parts of the context and treat answers as single tokens. With the goal of achieving above the vanilla Bidirectional Attention Flow (BiDAF) model performance, here we experimented with the coattention network that attends to the question and context simultaneously, the character-level embedding that conditions on internal word structures, and the answer pointer that predicts the end location of the answer conditioning on the start location probability distribution. We also used a mix strategy that combined two of our best performing models: the character embeddings and the conditioning end prediction on start prediction answer pointer model. Our findings showed meaningful improvement over the baseline BiDAF model and we compared the individual and

combined performances of the strategies to better understand how they work and how we can improve in the future.

2 Related Work

The baseline model for the default project was based on Bidirectional Attention Flow (BiDAF) [1], a hierarchical multi-layer architecture that adds a context-query attention layer after the embedding and encoding layer. It has word-level and contextual embeddings, but not the character-level embedding layer which was used in the original BiDAF model [2]. Running the BiDAF baseline model achieved EM/F1 of 57.5/61.0. In Dynamic Coattention Networks for Question Answering [3], the authors implemented the encoding layer in a similar fashion to BiDAF except that they added a second-level attention of the Context-to-Question (C2Q) distribution over the Question-to-Context (Q2C) attention outputs. They also introduced dynamic decoder that can find the global optimal span of text and other nuances to their DCN model which achieves higher performance with EM/F1 of 66.2/75.9. Instead of the dynamic decoder approach, we referred to the Answer Pointer component of the 'Match-LSTM with Answer Pointer' model [4]. On each timestep, the Answer Pointer outputs a probability distribution over locations in the context, and p^{end} depends on p^{start} over two timesteps. Inspired by the works listed above, we implemented character-level embeddings in addition to the existing word-level embeddings, added temporal information to the coattention context in our attention layer, and built a new output layer that conditions end location prediction on start location prediction.

3 Approach

3.1 Baseline BiDAF model

Our baseline model is the standard model based on the Bidirectional Attention Flow (BiDAF) which uses word-level embeddings. It consists of five layers as follows:

- Embedding layer: generates an embedding look-up from word indices in both the context and the question;
- Encoder layer: a one-layer, bidirectional LSTM that incorporates temporal dependencies between timesteps of the embeddings;
- Attention layer: a two-way attention flow between the context and the question with a static attention encoding output;
- Modeling layer: a two-layer, bidirectional LSTM that refines the output sequence of the attention layer;
- Output layer: takes the outputs from both the attention layer and the modeling layer as the input and produces the probability distributions of the start and end location of a predicted answer span.

The implementation details can be found in Section 4 in the Default Final Project: Building a QA system (IID SQuAD track) handout [1].

3.2 Embedding Layer with Character Embeddings

First we define character-level embeddings from pretrained Glove vector embeddings and learn them through a 1D CNN with 64 output channels and a 3x3 filter followed by max pooling to get a new embedding matrix with the same size of the word-level embeddings. We then concatenate the word-level and character-level embeddings and refine them using a two-layer highway encoder [5]. The final output of this layer consists of an embedding vector with a feature size twice as large for each word.

3.3 Attention Layer with Coattention from DCN

In our model, we replace the attention layer with that from the dynamic coattention networks. Simlar to BiDAF, it involves a two-way attention between the question and the context. In more details, question and context are first encoded by feeding input into a single LSTM and added sentinels for

both which allows the model to not attend to the context when necessary. To allow for variation between the question and context encoding space, additional non-linear projection layer is introduced on top of question encoding. We denote the question and context encodings as Q and C. These encodings are multiplied to acquire the affinity matrix $L \in \mathbb{R}^{(N+1)\times(M+1)}$ between the question and the context. Then L is softmax normalized row-wise and column-wise to produce attention weights a and b for the C2Q and Q2C attention, respectively.

$$\alpha^i = \text{softmax}(L_{i,:}) \in \mathbb{R}^{M+1} \tag{1}$$

$$a_i = \sum_{j=1}^{M+1} \alpha_j^i q_j \in \mathbb{R}^l \tag{2}$$

$$\beta^j = \text{softmax}(L_{:,j}) \in \mathbb{R}^{N+1} \tag{3}$$

$$b_j = \sum_{i=1}^{N+1} \beta_j^i c_i \in \mathbb{R}^l \tag{4}$$

We then get a second-level attention s_i by a weighted sum of the Q2C attention b_j over the C2Q attention distributions α^i.

$$s_i = \sum_{j=1}^{M+1} \alpha_j^i b_j \in \mathbb{R}^l \qquad \forall i \in \{1, ..., N\} \tag{5}$$

One implementation detail is that we remove the context sentinel from α as the sentinel represents a hidden state that does not correspond to any content of the context, so it is meaningless to let it attend to the question. Finally, the second-level attention s_i is concatenated with first-level C2Q attention outputs a_i and fed into a bidirectional LSTM which functions as adding temporal information to the final step of encoding.

$$\{u_1, ..., u_N\} = biLSTM([s_1; a_1], ..., [s_N; a_N]) \tag{6}$$

The resulting hidden states u_i are the *coattention encoding* adapted from DCN.

3.4 Output Layer Conditioning End Location on Start Location

We use a Boundary Model in the Answer Pointer layer, which replaces the original output layer in the baseline model [4]. Specifically, it predicts the probability of the start and the end location by conditioning the end location on the start location using a two-timestep, unidirectinoal LSTM. In the first timestep, the LSTM hidden state attends to the input hidden states $H^r \in \mathbb{R}^{l\times N}$,

$$F_1 = v^T \tanh(VH^r + (Wh_0 + b)) \tag{7}$$

where h_0 is the initial LSTM hidden state and F_1 is the attention score. Taking a softmax of the attention score produces an attention distribution β_s, the probability distribution for the start location. Then the product of β_s and H^r is fed into the LSTM for a time step to produce a new hidden state,

$$h_1 = \overrightarrow{LSTM}(H^r \beta_s^\intercal, h_0) \tag{8}$$

In the second timestep, the h_1 substitutes the h_0 in Eq. (7) and calculates the attention score over H^r again and produces a new attention distribution β_e, which is the probability for the end location.

3.5 Dropout Probability Tuning

To minimize overfitting, we tune the dropout probability by gradually increasing it from the default value of 0.2 to 0.4 in intervals of 0.1. We also try to find out the critical dropout value above which will no longer reduce overfitting but make the context representations uninformative [6].

3.6 Final Ensemble Model

We obtain the final ensemble model by incorporating all the implemented features described above into the baseline BiDAF model. Since the DCN layer slightly underperforms over the baseline two-way attention layer (as shown in the result section below), we choose to keep the baseline attention layer. In addition, we incorporate the character-level embeddings and Answer Pointer output conditioning and finetune the dropout hyperparameter. This ensembel model is expected to have the best performance in this research.

4 Experiments

4.1 Data

In our project, we used the latest version of the Stanford Question Answering (SQuAD 2.0) dataset. We conducted experiments on the official SQuAD 2.0 train set (all of the 129,941 examples), the dev set (half of the official dev set randomly selected, 6078 examples), and the test set (the remaining examples from the official dev set plus hand-labeled examples, 5915 examples).

4.2 Evaluation method

We evaluated the results using AvNA (Answer vs. No Answer), F1 and EM (Exact Match) scores, while F1 and EM scores were used as main metrics used to rank the leaderboard. We also inspected the example outputs to capture a better understanding of the model qualitatively. The Answer vs No Answer (AvNA) is the measure that finds the classification accuracy when only considering the answer (any span predicted) vs. no-answer predictions of the model. The EM score is calculated as:

$$\text{EM} = \frac{\text{exact matches}}{\text{total number of questions}}$$

The F1 score is calculated as:

$$\text{F1} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where the precision and recall are:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}, \ \text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negatives}}$$

4.3 Experimental details

For the experiment, we trained the baseline model using the following hyperparameters: learning rate = 0.5, dropout rate (the probability of zeroing out an activation in dropout layers) = 0.2, batch size = 64. The model took 10 hours and 33 minutes on the Azure virtual machine to complete the 30-epoch training. It reached the best performance value at 3.852M steps.

4.3.1 Coattention

We added the coattention layer to the model and trained with the same hyperparameters as the baseline model: learning rate = 0.5, dropout rate (the probability of zeroing out an activation in dropout layers) = 0.2, batch size = 64 and hidden size = 200 and the Adadelta optimizer. The model took 11 hours and 47 minutes on the Azure virtual machine to complete the 30-epoch training. It reached the best performance value at 3.752M steps.

4.3.2 Character-level Embeddings

We used the default parameters for the character-level embeddings with the following hyperparameters: we used the Adadelta optimizer at learning rate = 0.5, dropout rate = 0.2, batch size = 64 and hidden size = 100. The GloVe embedding has size of 300 and our final input embedding size is 200. The model took 15 hours and 15 minutes and reached the best performance value at 3.5M steps. The addition of character-level embedding has great improvement on the results, as shown in table 1.

4.3.3 Conditioning End Prediction on Start Prediction (Answer Pointer)

For the implementation of the Answer Pointer RNN, we used a single LSTM cell in which each forward() call proceeds exactly one timestep. We further refined the inputs to the LSTM cell by summing up the linear projection of the attention layer outputs and the linear projection of the modeling layer outputs. As before, we used the default hyperparameters: learning rate = 0.5, dropout rate = 0.2, batch size = 64 and hidden size = 100. The model took 9 hours and 20 minutes on the Azure virtual machine to complete the 30-epoch training. It reached the best performance value at 2.752M steps.

4.3.4 Ensemble

Here we combined two of our top performing models: Character Embedding and the Answer Pointer. For the first trial we used the same hyperparameters as before (learning rate = 0.5, dropout rate = 0.2, batch size = 64 and hidden size = 100). The model took 13 hours and 21 minutes on the Azure virtual machine to complete the 30-epoch training. It reached the best performance value at 3.3M steps. We noticed the model was overfitting at around 1.5M steps, so we experimented with various dropout rates (dropout rate = 0.2, 0.3, 0.4) on the ensemble model. The performance of different parameter combinations are demonstrated in table 2.

4.4 Results

4.4.1 Overall Results

We recorded the best performing parameters on the dev set during the training. The dev set evaluation results are shown in the following table:

	F1 Score	EM Score	AvNA
Baseline	59.93	56.53	66.96
Coattention	60.56	57.81	67.14
Char-Embedding	63.40	59.87	70.04
Answer Pointer	61.77	58.81	68.88
Ensemble	**64.21**	**60.81**	**70.91**

Table 1: Performance Comparison of Different Methods

From the table above, it is clear that we obtained the best result with the ensemble model with significant improvements in both EM and F1 scores. For the final ensemble test set result submitted on the leaderboard, we achieved a F1 score of 63.325 and an EM score of 59.814.

4.4.2 Other Performance Results

For the final results on the ensemble model in table 1, we have tried several parameter combinations as shown below.

	F1 Score	EM Score	AvNA
Ensemble(1), dp = 0.2	**64.21**	**60.81**	**70.91**
Ensemble(2), dp = 0.3	63.65	60.53	70.02
Ensemble(3), dp = 0.4	57.01	54.01	64.51

Table 2: Performance Comparison of Different Dropout Rates

We observed an overfitting pattern in Ensemble(1) with the default parameters as the loss started to increase after 1.5M steps. To prevent overfitting, we tried to tune the drop out probability to 0.3 and 0.4. The best performance is achieved with Ensemble(1) at the drop rate of 0.20. This is not as expected but still reasonable because when we increase the drop out rate, the model become more general with more variance in some layers and thus results in lower accuracy; so more epochs should be considered for higher dropout rates (further discussed in the Analysis section).

5 Analysis

5.1 Analysis on Results

5.1.1 Dynamic Coattention vs. Static Coattention

As shown on Fig.1, the dynamic coattention layer using a bidirectional LSTM for final output encoding underperforms over the the static cottention layer in the baseline model. One possible reason is because the sentinel vectors in the similarity matrix $L \in \mathbb{R}^{(N+1)\times(M+1)}$ do not succeed in predicting rare or unseen words by either reproducing a word from the recent context or producing a

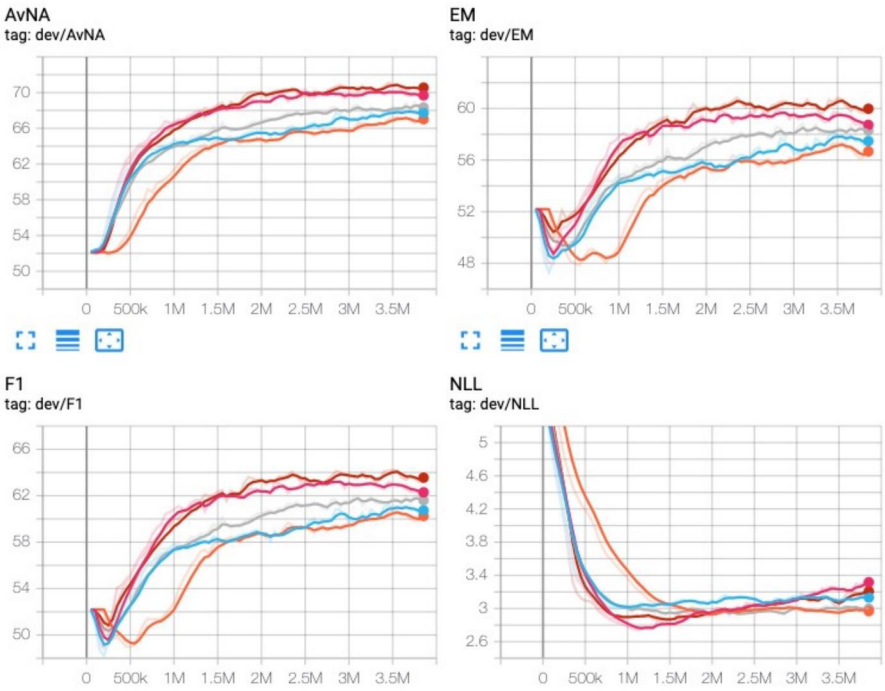

Figure 1: Training progress on the models: the AvNA, EM, F1 and NLL scores of Baseline Model (Blue), Coattention Model (Orange), Character-level Embedding Model (Pink), Answer Pointer Model (Grey), and the Ensemble Model (Red).

word from a standard softmax classifier [7]. Another reason for the underperformance of DCN may be that the second-level attention output $s_i \in \mathbb{R}^l$ is not fully representative of the co-dependenece between the context and the question thereby increasing extra but unproductive complexity in the model.

5.1.2 Benefits of Character-level Embeddings

Fig.1 exhibits considerable improvement by incorporating character-level word embeddings with the original word-level embeddings in baseline model. This improvement can be attributed to the ability of character-level embeddings to delve into the internal structure of words by splitting them into shorter sub-words. This significantly helps identify complex word structures (or morphology) as well as reproducing rare/unseen words in the context.

5.1.3 Enhancement by Conditional Answer Pointer

As shown in Fig.1, the Answer Pointer layer conditioning the end location probability on the start location probability also outperforms the baseline output layer, which independently predicts the start and the end location. This connection between the start and end location not only allows them to focus on places in the context that are closest to the ground-truth answer, but also helps the answer generation to stop at a reasonable location in the context.

5.1.4 Overfitting Reduction by Dropout Probability Tuning

Fig.2 shows the effects of tuning the dropout probability to avoid overfitting, thereby minimizing the negative log-likelihood (NLL). Specifically, by increasing the dropout probability (meaning a higher probability to zero out a word), feature co-adaptation is reduced as each feature can be evaluated more independently without overly binding to other features. We also discover the critical dropout probability of 0.3 because a too high dropout probability will produce uninformative context representations, which lead to inefficient and even erroneous training results. Even though the model with 0.3 dropout minimizes the loss, it does not achieve the highest score mainly due to insufficient

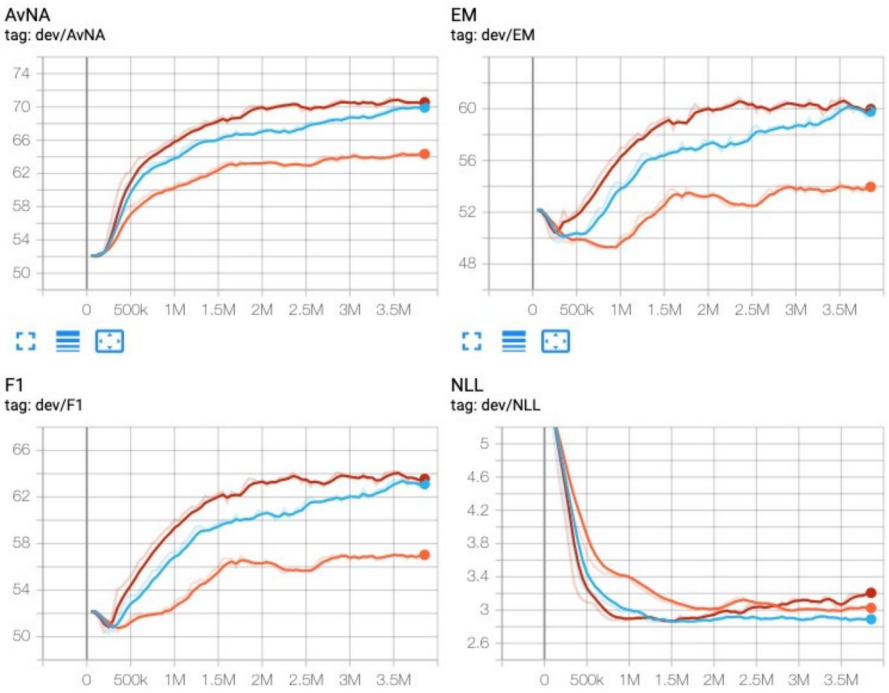

Figure 2: Training progress on different dropout rates on the Ensemble model: the AvNA, EM, F1 and NLL scores of dropout rate = 0.2 (Red), dropout rate = 0.3 (Blue), dropout rate = 0.4 (Orange).

training time as it learns from the context more slowly. It should outperform our current best model if it were trained for a longer time.

5.2 Analysis through Examples

We inspect the answers generated by our best performing model to understand qualitatively when our model worked and when it did not work. We examine the performance of our model by looking at specific samples in the validation set. This section demonstrates one example of improvement from the baseline, and two other major errors where our model fails to produce the correct answers.

5.2.1 Improvement from the Baseline

Question: Who designed the garden for the University Library?
Context: "Another important library – the University Library, founded in 1816, is home to over two million items. The building was designed by architects Marek Budzyński and Zbigniew Badowski and opened on 15 December 1999. It is surrounded by green. The University Library garden, designed by Irena Bajerska, was opened on 12 June 2002. "
Answer: Irena Bajerska
Baseline Prediction: Marek Budzyński and Zbigniew Badowski
Our Prediction: Irena Bajerska
In this example, it was Irena Bajerska who designed the garden, while “Marek Budzyński and Zbigniew Badowski" who designed the library is a piece of distracting information in the question. This is a good example where the baseline model was distracted, but our model was able to correctly produce the answer because it attended to the right answer span.

5.2.2 False Positive

Question: Of Poland’s inhabitants in 1901, what percentage was Catholic?
Context: "Throughout its existence, Warsaw has been a multi-cultural city. According to the 1901 census, out of 711,988 inhabitants 56.2% were Catholics, 35.7% Jews, 5% Greek orthodox Christians and 2.8% Protestants. Eight years later, in 1909, there were 281,754 Jews (36.9%), 18,189 Protestants

(2.4%) and 2,818 Mariavites (0.4%). This led to construction of hundreds of places of religious worship in all parts of the town."
Answer: N/A
Prediction: 56.2%
Here the model is confused about whether the question has an answer. This is tricky as the context describes the Warsaw city but the question asks about inhabitants in Poland. Our model identifies a potentially correct answer span but ignores the assumption in the question.

5.2.3 Semantics Ambiguity

Question: What European event caused the Huguenots to abandon Charlesfort?
Context: "French Huguenots made two attempts to establish a haven in North America. In 1562, naval officer Jean Ribault led an expedition that explored Florida and the present-day Southeastern U.S., and founded the outpost of Charlesfort on Parris Island, South Carolina. The Wars of Religion precluded a return voyage, and the outpost was abandoned. In 1564, Ribault's former lieutenant René Goulaine de Laudonnière launched a second voyage to build a colony; he established Fort Caroline in what is now Jacksonville, Florida."
Answer: The Wars of Religion
Our Prediction: Wars of Religion
Our prediction and the answer here only differ by an article "The", which does not impact the overall meaning. Although the model performs well, the EM score will be impacted.

6 Conclusion

In this project, we implemented and finetuned several models based off BiDAF to perform automated question answering for SQuAD 2.0 dataset. We experimented on improving embedding layer with character embeddings, attention layer with dynamic coattention, output layer with answer pointer, and finetuning on hyperparameters. In our final ensemble model, we achieved our best EM/F1 of 60.81/64.21.

Adding character embeddings to the existing word embeddings gave us a successful performance boost compared with the BiDAF baseline model. On the other hand, dynamic coattention from DCN did not beat the attention and modeling layer combined in the baseline model but was worth trying. Then we tried a decoder that uses the Answer Pointer, which aims to improve the accuracy of the answer span. Finally, we trained our ensemble model based on character embedding and conditional answer pointer over hyperparameter engineering.

Given several failed attempts with trying to improve the attention layer of DCN, we were limited in GPU hours to experiment with other attention models, such as self-attention and stacked attention layers. However, our success with adding character embedding indicates possible usefulness of using pre-training with external QA-datasets or data augmentation to enrich the information contained in embeddings. Last but not least, ensemble models that comprise best ideas from each layer of neural networks almost always help enhance performance.

References

[1] CS 224N Default Final Project:Building a QA system (IID SQuAD track), 2021. http://web.stanford.edu/class/cs224n/project/default-final-project-handout-squad-track.pdf

[2] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603,* 2016.

[3] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604,* 2016.

[4] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905,* 2016.

[5] Rupesh Kumar Srivastava, Klaus Greff and Jürgen Schmidhuber. Highway Networks. *arXiv preprint arXiv:1505.00387,* 2015.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsk, Ilya Sutskever and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1929-1958, 2014.

[7] Stephen Merity, Caiming Xiong, James Bradbury and Richard Socher. Pointer Sentinel Mixture Models. *arXiv preprint arXiv:1609.07843,* 2016.