Augmenting BiDAF with Per-Token Features

Stanford CS224N Default Project

Viet Nguyen
Department of Computer Science
Stanford University
vin041@stanford.edu

Abstract

The DrQA document reader showed that adding per-token features (e.g. part-of speech and named entity recognition tags) to a question answering model significantly improves performance on the SQuAD benchmark [1]. I add six features to a baseline BiDAF model and explore the benefit of applying attention to not only LSTM hidden state, but also these per-token features. I verify the benefit of applying self-attention to these features and find that the augmented model significantly improves upon the baseline in terms of metrics and train time. My best model achieves a test score of **(62.06 EM, 64.89 F1)** compared to a baseline of **(59.33, 62.09)**, reaching an optimal model in half the training steps.

1 Collaborators

- Mentor: Chris Waites

2 Introduction

The SQuAD 2.0 benchmark [2] is a question answering (QA) benchmark that provides a paragraph (context) and a question, asking a model to highlight the answer of the question in the context, or respond with "No Answer" if no answer exists. A question's answer is represented as a span of consecutive words in the context. The DrQA model adds six per-token features (e.g. part-of speech (POS) and named entity recognition (NER) tags) to a baseline LSTM-based question answering model to improve performance on the SQuAD 1.1 benchmark, a previous version of SQuAD where every question has an answer [1]. In this project, I further explore the use of per-token features, adding these six features to a baseline BiDAF inspired model and exploring the benefit of applying various forms of attention to not only the LSTM hidden state, but also these features[3].

I find that augmenting BiDAF with per-token features improves model performance significantly in terms of metrics and train time. In addition, self-attention in combination with these features further contribute to model performance.

This form of model augmentation may not benefit large pre-trained models such as BERT. Tenney, Das, and Pavlick show that classical NLP tags such as POS and NER can be easily retrieved from lower layers of BERT via probing, making these features redundant [4]. However, providing additional features to LSTM or smaller transformer models has the potential to improve performance and/or training time. Furthermore, augmenting models with features that cannot be probed from the model's word representations could provide a benefit in performance.

3 Related Work

The ideas and models referenced in this paper fall into the domain and category of pre-BERT QA. My proposed model(s) use the same per-token features as in the DrQA model [1]. To my knowledge, this is the only published model that augments a QA model with additional features.

Model	EM	F1
BiDAF (no char embeddings)	65.0	75.4
BiDAF	67.7	77.3
DrQA	70.0	79.0
R-Net	76.46	84.27
BiDAF++	77.57	84.86
QANet	80.93	87.77

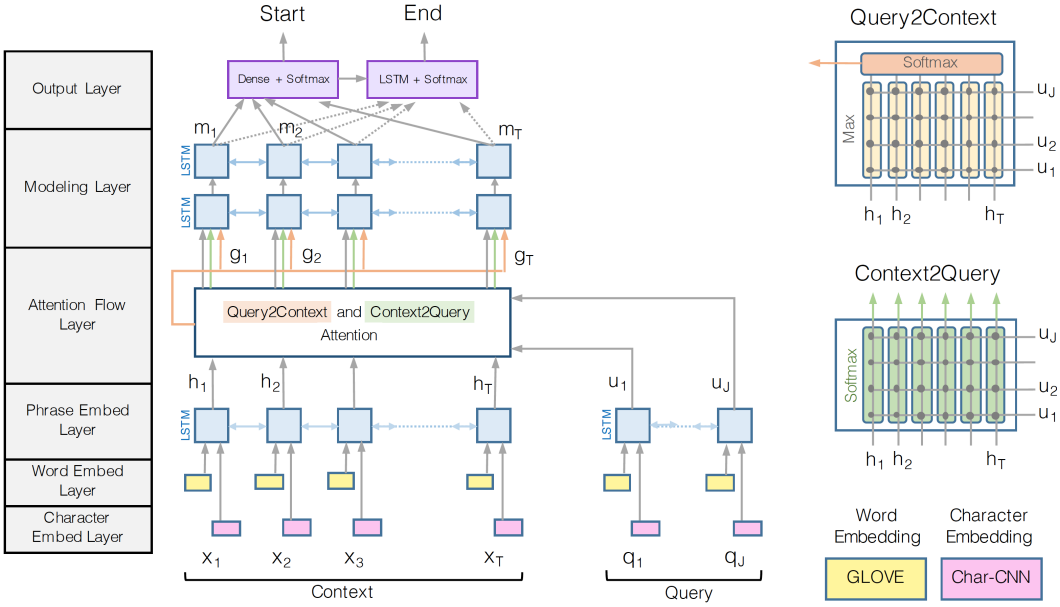
Table 1: SQuAD 1.1 test set scores of selected pre-BERT models

All presented models use BiDAF as a baseline[3]. BiDAF's name stems from its use of bidirectional LSTM's and attention. BiDAF's main building blocks include word and character embeddings, bidirectional LSTMs, context-to-query (C2Q) attention, and query-to-context attention (Q2C).

This paper takes inspiration from a variety of other pre-BERT QA models. For example, R-NET uses the same building blocks as BiDAF but with GRU's instead of LSTMs, and a form of additive self-attention [5]. BiDAF++ is a model that extends BiDAF by adding a form of self-attention based off of BiDAF's context-to-query attention [6]. The transformer encoder uses a more expressive and powerful form of self-attention than R-Net or BiDAF++ [7]. QaNet applies a variant on the transformer encoder with convolutions to the SQuAD 1.1 question answering task [8].

4 Approach

Figure 1: BiDAF model architecture [3]



This paper uses a variant of BiDAF as a baseline [3]. BiDAF outputs two probability distributions $\mathbf{p_{start}}$ and $\mathbf{p_{end}}$ over each of N context words, selecting the answer span $(\mathbf{i}, \mathbf{j})$ that maximizes $\mathbf{p_{start}}(\mathbf{i}) * \mathbf{p_{end}}(\mathbf{j}) \ \ \forall(\mathbf{i}, \mathbf{j}) \in \{\mathbf{1}, ..., \mathbf{N}\}$ with $\mathbf{0} <= \mathbf{j} - \mathbf{i} < \mathbf{L}$. L is a hyperparameter that limits the model from predicting long answer spans, and is set to 15 in the baseline. BiDAF was tested on SQuAD 1.1 which did not include examples where no answer could be found in the context. To allow the model to predict "No Answer", we prepend an out of vocabulary token (OOV) to the context, and if the model predicts (OOV) as the start and end token of the answer, the model returns "No Answer".

The other difference between the baseline and BiDAF is the form of Q2C attention and the attention layer output (the "Attention Flow Layer" in Figure 1). Let N be the context length and M the query length. Let H be the hidden size. Let $\mathbf{c_1}, ..., \mathbf{c_N} \in \mathbb{R}^{\mathbf{2H}}$ represent the context words coming into the attention layer and $\mathbf{q_1}, ..., \mathbf{q_M} \in \mathbb{R}^{\mathbf{2H}}$ represent the question words coming into the attention layer.

Let $\mathbf{C} \in \mathbb{R}^{\mathbf{N \times 2H}}$ and $\mathbf{Q} \in \mathbb{R}^{\mathbf{M \times 2H}}$ be concatenations of the context and question words respectively. Let $\mathbf{S} \in \mathbb{R}^{\mathbf{N \times M}}$ be the similarity matrix between the context and query, and $\mathbf{S^1} \in \mathbb{R}^{\mathbb{N} \times \mathbb{M}}$ be the C2Q weight matrix as described in the BiDAF paper[3]. The Q2C weight matrix $\mathbf{S^2} \in \mathbb{R}^{\mathbb{N} \times \mathbb{N}}$ is calculated as follows:

$$\mathbf{S^{int}_{:j}} = \mathbf{softmax}(\mathbf{S^1_{:j}}) \in \mathbb{R}^{\mathbf{N}} \quad \forall \mathbf{i} \in \{\mathbf{1}, ..., \mathbf{M}\}$$

$$\mathbf{S^2} = \mathbf{S^1}(\mathbf{S^{int}})^{\mathbf{T}} \in \mathbb{R}^{\mathbf{N \times N}}$$

Finally, the output of the attention layer is

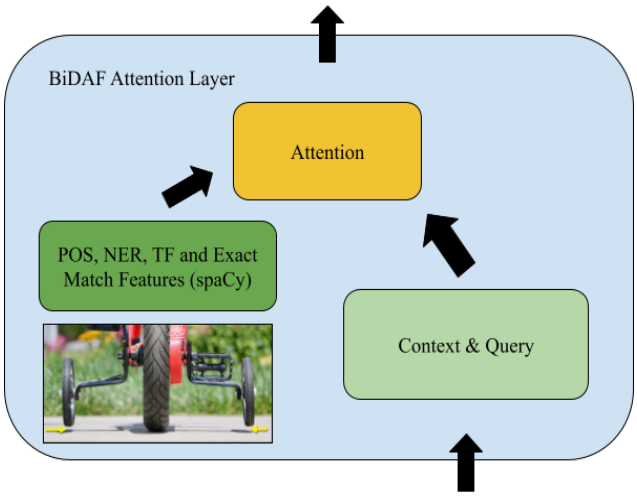
$$\mathbf{att} = [\mathbf{C}; \mathbf{A}; \mathbf{C} \odot \mathbf{A}; \mathbf{C} \odot \mathbf{B}] \in \mathbb{R}^{\mathbf{N \times 8H}}$$

where

$$\mathbf{A} = \mathbf{S^1 Q} \in \mathbb{R}^{\mathbf{N \times 2H}}, \quad \mathbf{B} = \mathbf{S^2 C} \in \mathbb{R}^{\mathbf{N \times 2H}}$$

A brief test comparing this baseline to BiDAF with its original version of Q2C attention finds that this substitution improves performance by roughly 1.5 F1 points on the SQuAD 2.0 dev set.

Figure 2: Proposed BiDAF modification



Augmenting this baseline model, I use a spaCy pipeline to perform tokenization, lemmatization, part-of-speech tagging, and named entity recognition [9]. Using this pipeline, I construct the six features used in the Dr.QA document reader [1]. These features are processed for each word in the context paragraph that SQuAD provides.

These features can be divided into two classes:

- f_{token} includes a part-of-speech (POS) tag, a named entity tag (NER), and a normalized term frequency (TF). The normalized term frequency is the number of occurrences of the token in the context divided by the total number of tokens in the context.
- $f_{exactMatch}$ includes three binary features. Given a context token, the binary features are true if and only if the token can be exactly matched to a token in the question in original, lowercase, or lemma form.

Of these six features, TF is represented as a floating point number, and the other five categorical features are represented by a one-hot encoding.

I modify the aforementioned baseline by concatenating the per-token features (in their original form and multiplied by the Q2C matrix) to the original attention output:

$$\textbf{Let } \mathbf{att2} = [\mathbf{att}, \mathbf{f_{token}}; \mathbf{f_{exactMatch}}; \mathbf{S^2} \times \mathbf{f_{token}}; \mathbf{S^2} \times \mathbf{f_{exactMatch}}] \in \mathbb{R}^{\mathbf{N} \times (\mathbf{8H+2X})}$$

where X represents the size of the feature encodings for a single word. Finally, I use the form of self attention described in the BiDAF++ model to build $\mathbf{att2}$ [6]:

$$\textbf{Let } \mathbf{S_{ij}^{self}} = \mathbf{w_{self}^T}[\mathbf{c_i}, \mathbf{c_j}; \mathbf{c_i} \odot \mathbf{c_j}] \in \mathbb{R} \textbf{ if } \mathbf{i \neq j} \textbf{ else } \mathbf{S_{ij}^{self}} = -\mathbf{inf}$$

where $\mathbf{w_{self}} \in \mathbb{R}^{\mathbf{6H}}$ is a weight vector parameter.

$$\textbf{Let } \mathbf{S^3_{i,:}} = \mathbf{softmax}(\mathbf{S^{self}_{i,:}}) \ \forall \mathbf{i} \in \{\mathbf{1}, ..., \mathbf{N}\}, \ \mathbf{S^3} \in \mathbb{R}^{\mathbf{N} \times \mathbf{N}}$$

$$\textbf{Let } \mathbf{attFinal} = [\mathbf{att2}; \mathbf{S^3C}; \mathbf{C} \odot (\mathbf{S^3C}); \mathbf{S^3} \times \mathbf{f_{token}}; \mathbf{S^3} \times \mathbf{f_{exactMatch}}] \in \mathbb{R}^{\mathbf{N} \times (\mathbf{12H+3X})}$$

$\mathbf{attFinal}$ is then linearly projected to a size of $\mathbb{R}^{\mathbf{N} \times \mathbf{512}}$ before being forwarded to the modeling and output layers as described in the BiDAF paper [3]. I also test a variant of this model where after the linear projection, a transformer encoder block is applied to this projection, and the transformer enconder's output is forwarded to the modeling and output layers [7].

5 Experiments

5.1 Data

I use the provided train, dev, and test datasets for SQuAD 2.0 [2]. The SQuAD dataset consists of 100,000 answerable question answer pairs and 50,000 unanswerable question answer pairs. According to the SQuAD website, "(SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable". Note that 3 ground truth answers are provided for a given answerable question answer pair in the dev and test sets [10]. The dev and test datasets are a 50-50 split of the official dev dataset for SQuAD 2.0.

5.2 Evaluation method

My main evaluation metrics on the SQuAD 2.0 dev and test sets are exact match (EM) and F1 scores. EM measures if the model predicts an answer span that matches exactly with one of the ground truth answers. For a given ground truth answer, F1 score measures the harmonic mean of precision and recall. The maximum F1 score over the ground truth answers is measured. I also perform an ablation analysis of two models to determine the significance of the per-token features and attention. Ablation is performed by setting specific features to 0 and evaluating on the dev set. In the ablation analysis, I additionally report the classification accuracy of answer vs no answer predictions (AvNA).

5.3 Experimental details

The models use pre-trained GloVe word and character embeddings. The word embeddings have size 300, and the character embeddings have size 64. Following BiDAF, the character convolutions use a kernel size of 5 and 100 output channels [3]. A dropout rate of .2 applied to the initial embeddings, after each LSTM layer, and in the transformer encoder block. The models train with a batch size of 64 and a hidden size of 100 for the LSTM sub-components. Each model uses the AdaDelta optimizer with a learning rate of 0.5 and no L2 weight decay. Each model is trained for 30 epochs and the model that achieves the highest dev-set F1 score during train time (evaluated every 50,000 steps) is saved. If a model's dev negative log likelihood fails to reach a new minimum over 1M steps (roughly 8 epochs) and the F1 score falls by 1.0 below the peak F1 score, the model is stopped early.

5.4 Results

The baseline model with character embeddings is represented by (*). The baseline with the token features and self-attention as described in Section 4 is represented by (**). The (**) model with the transformer encoder layer is represented by (***).

Model	train steps	EM	F1
Baseline w/out char embeddings	3.30M	56.70	60.13
(*) Baseline w/ char embeddings	3.65M	59.94	63.41
BiDAF++ re-implement	2.95M	62.88	65.87
(**) (* + tokenFeats + attention)	2.75M	62.88	65.81
(***) (** + transformer encoder)	1.50M	64.01	66.85

Table 2: SQuAD 2.0 dev set scores

I report the EM and F1 dev-set SQuAD 2.0 scores of each model I train in Table 2. Character embeddings contribute 3.28 F1 points when compared to a baseline without them, comparable to the difference between the original BiDAF model evaluated on SQuAD 1.1 with and without character embeddings (1.9 points). Adding per-token features to the baseline model improves the F1 score by 2.40 F1 points. Adding a transformer encoder further improves the F1 score by 1.04 F1 points.

Table 1 (see the Related Work section) shows that the self attention used by models such as R-Net[5] and BiDAF++[6] significantly improves SQuAD 1.1 performance compared to previous models. For example, BiDAF++ outperforms BiDAF by 7.5 F1 points on SQuAD 1.1. This result is not replicated in my experiments, with my BiDAF++ re-implementation outperforming the baseline by only 2.46 F1 points (adding 1.5 points for the baseline's improvement over BiDAF, this totals ≈ 4.0 F1 points).

Model	EM	F1
BiDAF (no-answer)	59.17	62.09
BiDAF + self-attention	59.33	62.31
(**) (baseline + tokenFeats + attention)	61.49	65.00
(***) (** + transformer)	62.06	64.89
BiDAF++	65.65	68.87

Table 3: SQuAD 2.0 test set scores of selected models

* BiDAF (no-answer) is roughly equivalent to my baseline model

Under the premise that a poor objective function could shrink the difference between two architectures, this gap could be explained by cross-entropy loss being a sub-optimal objective function for the task. My BiDAF++ re-implementation with token features (dev set) underperforms BiDAF++ (test set) by 3.0 F1 points. In addition to applying self-attention to BiDAF, BiDAF++ also uses a different loss function for non-answerable questions, which I did not implement. This alternate loss function seems to significantly improve performance compared to applying the same cross-entropy loss function to QA pairs with and without answers.

Besides improving performance, adding token features greatly reduced the amount of training time required to achieve the best model across the 30 epochs. (***) takes 1.50M steps compared to the baselines 3.65M steps. Although from Table 2 (**) appears to take about as long as the baseline to train, it reaches a local peak of 65.36 F1 after just 750K steps. This suggests that the per-token features serve to "kickstart" the model, providing valuable representations from the start, rather than having the model learn these representations on its own.

As a general observation, test set scores tended to be lower than dev set scores. (***) drops by 1.96 F1 points and (**) drops by 0.81 F1 points. This phenomenon is most likely a combination of selection bias and differences in how the test, dev, and train set are structured. Dev set F1 scores are noisy, and choosing the model that maximizes F1 score may not maximize test set F1 score. On a smaller note, the train set only contains contexts of length 400 or less, while the test set contains longer contexts. However, this has a negligible effect since the test set contains exactly 1 out of 5915 examples where this is the case.

Features	EM	F1	AvNA
Full	62.88	65.81	71.85
No $f_{exactMatch}$	31.17	26.89	54.95
No f_{token}	55.84	60.68	69.35
No per-token Q2C attention	61.44	64.53	70.58
No per-token self attention	59.57	62.88	69.53
No per-token attention	57.65	61.05	68.16
Only per-token attention	20.62	25.25	52.68

Table 4: Dev-set ablation analysis on (**) (baseline + tokenFeats + attention)

Features	EM	F1	AvNA
Full	64.01	66.85	72.21
No $f_{exactMatch}$	43.77	46.09	60.24
No f_{token}	60.70	64.08	70.17
No per-token Q2C attention	64.09	66.69	71.65
No per-token self attention	61.91	64.83	70.64
No per-token attention	62.21	64.83	70.48
Only per-token attention	41.71	44.57	59.65

Table 5: Dev-set ablation analysis on (***) (** + transformer)

As was found in the DrQA model, the ablation analysis for (**) and (***) found $f_{exactMatch}$ to be a more useful feature than f_{token} [1]. Notably, removing EM features from (***) drops the AvNA by 11.97 points, suggesting that EM is heavily used to predict answer vs no answer. On first principles, the model probably chooses an answer more often if relevant context words match question words. Considering that removing EM features drops the model's F1 score by 20.76 points, roughly twice the drop in AvNA, EM features are also heavily used to determine the answer spans themselves.

The other (***) features are marginally helpful. (***) prefers applying self attention (-1.57 F1) rather than Q2C attention (-.56 F1) to tokens. This preference is expected since the more useful exact match features are redundant with Q2C attention. (***) preferred using the unprocessed exact match and token features to their attention counterparts, perhaps preferring to use the transformer's multi-headed-attention to process them.

(**)'s ablation results are similar to those of (***). The major difference between the two is that upon removing the EM features, (**)'s F1 score drops by almost 40 points, showing that (**) is even more reliant on exact match as a feature. Despite (**) performing marginally better on the test set, I would consider (***) to be the stronger model due the more equal contribution of its features and quicker train time.

6 Analysis

Research has shown that BERT is able to encode accessible POS and NER information about a context[4]. One possible hypothesis as to why the model finds f_{token} less useful than $f_{exactMatch}$ is that LSTMs are also capable of encoding token features in their word representations, making the token features more redundant than the exact match features. This observation could also explain why adding these features greatly decreases the training time needed to reach an optimal model on the dev-set (when comparing (**) to (*)): features that are usually learned by the LSTMs are instead provided from the start.

On the other hand, inspecting a few incorrect (***) predictions suggests some flaws in heavily using the EM feature:

- **Question**: What kingdom annexed Duchy in 1796?
- **Context**: Warsaw remained the capital of the Polish–Lithuanian Commonwealth until 1796, when it was annexed by the Kingdom of Prussia to become the capital of the province of South Prussia. Liberated by Napoleon's army in 1806, Warsaw was made the capital of the newly created Duchy of Warsaw. Following the Congress of Vienna of 1815, Warsaw

became the centre of the Congress Poland, a constitutional monarchy under a personal union with Imperial Russia. The Royal University of Warsaw was established in 1816.

- **Answer**: N/A
- **Prediction**: Kingdom of Prussia

In this example, the model's NER and EM features probably suggest that "Kingdom of Prussia" is the answer. The entity "Kingdom of Prussia" is in the correct form if an answer existed, and the word "kingdom" is found in the question, activating the EM feature. However, the model misses that "annex" is the opposite of "liberate".

- **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. Most of them were destroyed in the aftermath of the Warsaw Uprising of 1944. After the war, the new communist authorities of Poland discouraged church construction and only a small number were rebuilt.
- **Answer**: N/A
- **Prediction**: 56.2%

Once again, the model responds in the correct answer form if one existed, and observes that "Catholics" are described by "56.2%". It also observes that Poland is found in the context and the passage. However, it fails to recognize that the census data applied to Warsaw rather than Poland.

Although the EM feature improves SQuAD F1 score overall, these counterexamples underscore that (***) performance may not entail actual "understanding". Most likely a stronger end-to-end baseline is needed to answer these types of questions correctly, and feature engineering should be viewed as a way of augmenting an existing end-to-end model's performance and/or train time.

7 Conclusion

In summary, I find that augmenting BiDAF's attention layer with token and exact match features significantly improves both metrics (F1, EM), and training time. Applying query-to-context and self attention to these tokens provides marginal benefits, with self attention being more important. Further adding a transformer encoder layer causes the model to become less reliant on the EM feature, though it still uses it significantly.

Future work could experiment with applying per-token features to transformer based models. Though my highest performing model includes one transformer encoder block, it still relies on several LSTM layers to make its predictions. These per-token features may interact differently with different types of models, and perhaps novel features could work best with different model architectures.

References

[1] Jason Weston Danqi Chen, Adam Fisch and Antoine Bordes. Reading wikipedia to answer open-domain questions. arXiv preprint arXiv:1704.00051, 2017.

[2] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

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

[4] Ellie Pavlick Ian Tenney, Dipanjan Das. Bert rediscovers the classical nlp pipeline. https://arxiv.org/pdf/1905.05950.pdf, 2019.

[5] Furu Wei Baobao Chang Wenhui Wang, Nan Yang and Ming Zhou. R-net: Machine reading comprehension with self-matching networks. Natural Language Computing Group, Microsoft Research Asia, 2017.

[6] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension, 2017.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[8] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

[9] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[10] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.