Intermediate Pretrained Contextual Embedding Models With Applications in Question Answering

Stanford CS224N Default Project

Ryan Silva
Department of Computer Science
Stanford University
rdsilva@stanford.edu

Abstract

The goal of this research is to explore deep learning techniques for question answering on the Stanford Question Answering Dataset (SQuAD 2.0). This project builds on current state-of-the-art techniques using pretrained contextual embeddings (PCEs) from ALBERT, which is a deep transformer-based Language Model. Most fine-tuned models for downstream tasks such as question answering use the final layer of language models as PCEs [1]. This paper presents intermediate models that take as input all hidden layer representations of deep transformer LMs and produce a combined token representation as the output. The focus of this paper is to describe how these intermediate PCE models affect performance on the question answering SQuAD 2.0 task. All models are trained and tested on a custom subset of the official SQuAD 2.0 data. We find that adding an intermediate Dot Product Attention model leads to a small increase in performance over the last-layer-only baseline, improving F1 by 0.32% in a model fine-tuned for 2 epochs, and 0.66% in a model fine-tuned for 5 epochs.

1 Introduction

Question answering is an important task in Natural Language Processing because it requires both understanding of natural language and knowledge about the world [2]. Accurate question answering systems have the potential to greatly improve user interaction with technology, such as enabling dialog systems to provide concise and accurate answers to everyday questions. In the question answering setting, a model is given a context paragraph and a question about that paragraph as input. The label for each input is a pair of start and end tokens identifying a continuous sequence in the context that correctly answers the question. The goal of the model is to output the correct positions of the start and end tokens that identify the answer in the paragraph. In other words, the model does not need to generate new text to answer the question, but rather selects a passage from the context as an answer. SQuAD 2.0 introduces questions to which there may be no answer in the provided context. In these cases, the model must predict that there is no answer possible.

As with many other NLP tasks, the state of the art in question answering has vastly improved by introducing Pretrained Contextual Embeddings (PCEs) [1] [3] into models. PCEs are token embeddings that represent a word based on the other words and phrases that surround it, rather than having a fixed representation of a word, such as in embeddings like word2vec. PCEs are powerful abstractions because they capture the semantics of the word itself, as well as context around the word at an arbitrary distance, within the embedding.

This paper approaches question answering by building on previous work on PCEs, most significantly the work from Lan, et al.[4] which presented ALBERT, a deep pretrained language model. ALBERT is a successor to BERT, both of which are deep transformer Language Models (LMs). In contrast to recurrent architectures such as an LSTM used in previous LMs, a transformer is a feedforward model. This has several advantages including avoiding locality bias inherent in recurrent architectures, as well as having a fixed length computation path, as opposed to paths that increase linearly with input in LSTMs. The latter advantage mitigates the vanishing gradient problem, and speeds up training and inference of the model, which enable the use of much deeper architectures.

Most research has focused on using the final layer of models such as ALBERT [1]. The focus of this paper is to develop intermediate models that take as input all hidden layer representations of deep transformer LMs and produce a combined token representation as the output. Additionally, this paper experiments with how these intermediate models affect performance on the question answering SQuAD 2.0 task.

2 Related Work

In the work by Peters et al. [3], which developed the seminal ELMO model, the authors show that "exposing the deep internals of the pre-trained network is crucial, allowing downstream models to mix different types of semi supervision signals." The authors found that a PCE that combines all hidden layers of a Language Model performs better than only using the last hidden layer. In particular, for the question answering task on SQuAD 1.0 data, models utilizing combined embeddings improved F1 by 0.3 to 0.5 absolute percentage points over the same models that only used the last layer. By examining the weights given to each layer, the authors found that different task models preferred different hidden layers in their pretrained ELMO model. The authors concluded that "the biLM layers encode different types of syntactic and semantic information about words-in-context, and that using all layers improves overall task performance".

Can these same methodologies be applied to more recent transformer language models such as BERT [1] and ALBERT [4], which are much deeper and trained on much more data? Transformer networks contain encoder blocks that utilize a self attention mechanism to produce Pretrained Contextual Embeddings [5]. The PCEs can be used in a wide range of natural language tasks by using them as input to a down stream task-specific model, and fine tuning the entire network for a specific NLP task. BERT lead to new state-of-the-art results on eleven NLP tasks. In particular, the authors improve the SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

Since modern language models such as BERT and ALBERT are deep, it is theorized that the hidden layers represent different and successively abstract representations for tokens. In an overview study of BERT findings, Rogers et al. [6] conclude that the lower layers have the most linear word order information, while semantic information is spread across the entire model. Syntactic information is most prominent in the middle BERT layers, as evidenced by Hewitt and Manning (2019) [7], which show that reconstructing syntactic tree information from BERT embeddings is most successful at the middle BERT layers. Clark et al. show that with BERT, only "certain attention heads correspond well to linguistic notions of syntax and coreference". Corefernce is thought to be an important skill in the question answering task, and thus even though the last layers of models like BERT and ALBERT are the most task specific, since different information is distributed throughout BERT layers, it may be beneficial to directly encode all layer representations for down stream tasks.

Finally, this research explores how attention mechanisms can be used to combine hidden embeddings. Attention, in a general sense, requires triples of a query, key and value. The values are combined linearly by assigning them a weight computed from the query and the key. Attention mechanisms have been successfully applied in Neural Machine Translation [8], where a decoder network produces a translation in a target language by attending to different states of an encoded source input at each step. One key advantage of attention is that different attention weights are computed for each query and key without explicitly allocating parameters for each combination. This research introduces intermediate models that use attention weights to combine the hidden layers of ALBERT. The attention weights can be interpreted as the importance of different layers for each token in terms of how useful they are for the down stream task. In the case of question answering, these weights indicate how important a certain layer is for determining the start and end of the answer in the context paragraph.

3 Approach

3.1 The Question Answering Model

This research builds on the question answering model proposed by Devlin et al. [1]. The input question and context are packed together into a single sequence and fed to the pretrained language model to embed the tokens as PCEs. A model with two vectors of learnable parameters S and E is used to process these PCEs and determine the start and end of the answer within the context respectively. The model produces logits for each token in the context by computing the dot product between the PCEs, denoted as T_k, and either S or E, as in the following equations:

$$s_k = \frac{\exp S^T T_k}{\sum_j \exp S^T T_j} \qquad e_k = \frac{\exp E^T T_k}{\sum_j \exp E^T T_j}$$

where $S, E, T_k \in \mathbb{R}^H$. The score of a span from token i to token j is given by $s_i + e_j$, and the maximum scoring span with $j \geq i$ is used for the prediction. The training objective is the sum of the cross entropy of the start and end logits, averaged over all training examples:

$$Loss = -\frac{1}{2N} \sum_{i}^{N} \log s_i + \log e_i$$

where s_i and e_i are the probabilities predicted by the model for the true labeled start and end tokens in the ith training example. S, E, and the deep language model which produces T_k are jointly finetuned on training data using this objective. For SQuAD 2.0 questions where there is no answer, a no-answer span is computed using the special CLS token embedding C: $s_{null} = S^T C + E^T C$. If this score is higher than any other span's score in the context, the model predicts no-answer.

3.2 Intermediate PCE Models

The main contribution of this paper is experimenting with question answering specific models that make use of representations at every layer of ALBERT. The approach in this paper is to learn a function that maps every hidden embedding for a token to a single vector representation. This function should generate a combined PCE which is more suited to the task of question answering than only using the last embedding layer of deep pretrained models, which is currently the most common practice [1] [3]. This function is provably at least as powerful and expressive as simply using the last layer embedding, since the function can trivially return the last layer. In general, this function is:

$$T'_k = f(h_i; \theta), 1 \leq i \leq L$$

Where $h_i \in \mathbb{R}^H$ is the hidden embedding at the output of an encoder block in layer i of a transformer language model (LM).

3.2.1 Parameterized Model

The first approach is inspired by the work from Peters et al. [3], which allocates learnable parameters for weighting each hidden layer, and then linearly combining the hidden layers by multiplying them by the learned weights w_i. The model also introduces a parameter γ for scaling the resulting combined vector. Separate weights are used for learning token representation for the start and end logits. The resulting token is expressed as:

$$T'_k = \gamma \sum_{i=1}^{L} w_i h_i$$

3.3 Attention Models

The following models all make use of an attention based mechanism to construct the weights w_i for each layer. Unlike the weighted parameter approach, attention allows the model to determine different layer weights for each token without explicitly allocating weights for them. As discussed in the introduction section, attention mechanisms operate on pairs of queries, keys and values. Thus the following intermediate models are adapted to take in an additional query input from the task specific model, which can be viewed as just another parameter in the model, however it can also be represented explicitly as a parameter q:

$$T'_k = f(h_i; \theta, q), 1 \leq i \leq L$$

For the specific task of question answering, the query parameter would be either $q = S$ to compute a start-of-question task-specific token, or $q = E$ for an end token, where S and E come from the downstream question answering model described above.

3.3.1 Dot-Product Attention Model

The second model uses a dot product attention mechanism. This form of attention directly compares how similar a layer embedding is to the query parameter to determine the layer weights. Mathematically, this model computes a linear combination of the hidden layers, where

$$w_i = \frac{\exp{(q^T h_i)}}{\sum_{j=1}^{L} \exp{q^T h_j}}$$

3.3.2 Weighted Attention Model

The third model is similar to the second except it uses a multiplicative attention mechanism. This model introduces its own parameters specifically for learning attention weights. In this model the layer weights are as follows:

$$w_i = \frac{\exp{(q^T W h_i)}}{\sum_{j=1}^{L} \exp{q^T W h_j}}$$

where $W \in \mathbb{R}^{HxH}$ is a learned parameter in the model.

3.3.3 Encoder Model

The final approach makes use of a transformer encoder block as the model for producing combined PCEs, as presented in Vaswani et al. [5]. However in this model, instead of using self attention, the encoder block computes attention weights using the input query parameter. This model introduces many more learned parameters, including weights for producing embedded queries keys and values, as well as additional fully connected layers. See Wolf et al. [9] for a detailed encoder description and implementation. The model can be abstracted as

$$T'_k = Encoder(h_i; \theta, q), 1 \leq i \leq L$$

4 Experiments

4.1 Data

All models are trained and tested on a custom subset of the official SQuAD 2.0 data [10]. Each example in the data contains a question and a context paragraph, which are both simply sequences of text, with the context usually being about a paragraph in length. The objective is to identify the correct answer to the question somewhere in the paragraph if it exists, or predict a no-answer if an answer does not exist in the provided context. Each question that has an answer has three separate solutions gathered by crowd sourcing answers, which may or may not be the same.

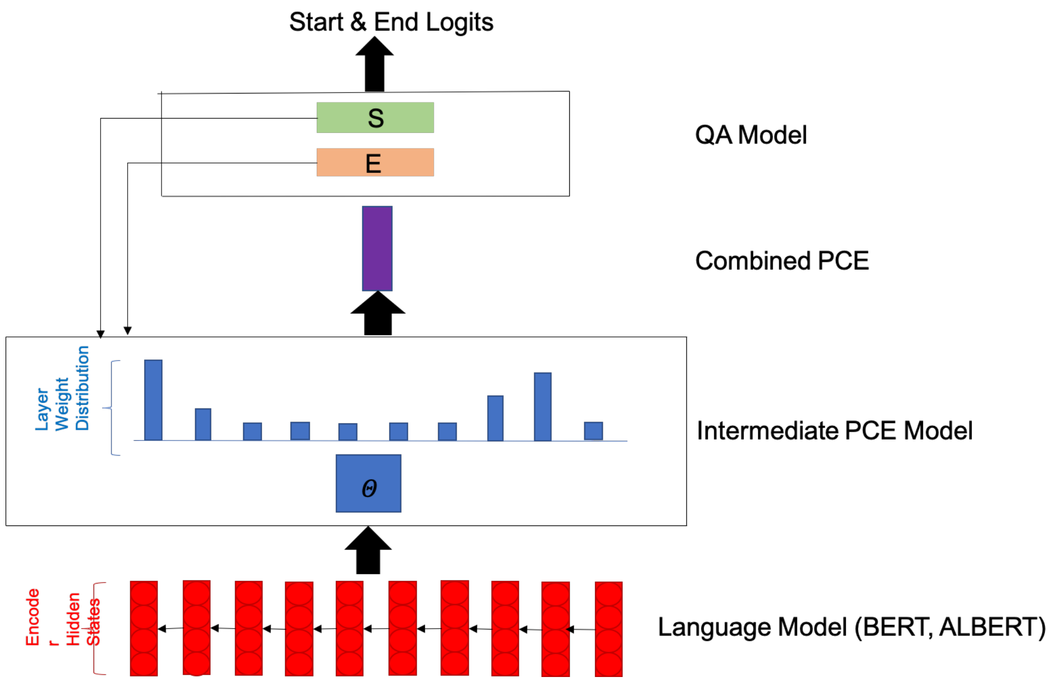

Figure 1: Diagram showing how an Intermediate PCE model can be used to combine hidden layers from a deep language model for a downstream task such as question answering.

4.2 Evaluation method

To identify the answer a model must predict the start and end tokens of a contiguous sequence within the paragraph that corresponds to one of the provided solutions. Performance is measured via two metrics: Exact Match (EM) score and F1 score. Exact Match is a binary measure (i.e. true/false) of whether the system output matches the ground truth answer exactly. F1 is the harmonic mean of precision and recall. In this context, precision is the fraction of words predicted that are also in the solution, and recall is the fraction of solution words that are also found in the predicted words.

When a question has no answer, both the F1 and EM score are 1 if the model predicts no-answer, and 0 otherwise. For questions that do have answers, the maximum F1 and EM score across the three human-provided answers is used. Finally, the EM and F1 scores are averaged across the entire evaluation dataset to get the final reported scores.

4.3 Experimental details

Experiments were run on an Azure cloud resource with GPU [3]. The Transformers library from HuggingFace [9] was used to download pretrained models used in this research. The baseline models using BERT and ALBERT were trained using existing scripts from the library. PyTorch was used to build the intermediate PCE models presented in this research.

Two experiments were done, one over 2 epochs of training and one over 5 epochs. In each experiment, models were fine-tuned for the same number of epochs over the entire SQuAD 2.0 training set. The learning rate for each model was set to 3e-5, a batch size of 8 for BERT models and 12 for ALBERT models, a max sequence length input of 384, a and document stride of 128 was used for all experiments. A linearly decreasing learning rate schedule was used for all fine-tuning runs. Additionally, the same random seed was used within each experiment.

4.4 Results

Surprisingly on the Test set, the Dot Attention Model fine-tuned on only 2 epochs of the training data performed best. This model also performed the best on the Dev set. However the Encoder model trained for 5 epochs was only .2% F1 behind the Dot-Attention model.

Model	Test F1	Test EM
Dot Attention 2 Epochs	**78.370**	**74.844**
Dot Attention 5 Epochs	77.143	72.781
Encoder Attention	78.159	74.303

Table 1: Results on the test set as determined by the 2020 CS224n class leader board

4.4.1 Experiment 1: 2 Epochs

These are the Dev set results of running the baselines, as well as the Weighted and Dot Attention models for 2 Epochs over the training data. While all fine-tuned models have similar training losses, models that use ALBERT rather than BERT PCEs do much better on the Dev set. This is somewhat expected, as one of the key innovations of ALBERT is fewer weights, and because of this ALBERT is less prone to overfitting the data. Another interesting observation is that the Weighted Intermediate PCE Model performs much worse compared to ALBERT and the Dot Attention intermediate PCE Models. This is contrary to what we expected after Peters et al. [3] showed that a weighted model had a positive effect in model performance. Finally, the Dot Attention Model proves to have a small positive impact on the resulting downstream task, increasing F1 over the baseline ALBERT by 0.32% and EM by 0.35%.

Model	DEV F1	DEV EM	DEV Loss
BERT-Base-no-ft	52.133	52.122	-
BERT-Base	75.037	71.832	0.634
ALBERT	79.636	76.538	0.648
Weighted	78.144	75.156	0.697
Dot Attention	**79.953**	**76.883**	**0.634**

Table 2: Results after 2 epochs of fine-tuning on the SQuAD 2.0 training set for select baselines and models. The Dot Product Attention model performs the best on all metrics.

4.4.2 Experiment 2: 5 Epochs

Only the dot product attention model outperforms the ALBERT baseline, increasing F1 performance by 0.66% and Em by 0.66%. Surprisingly, this simple model beats the more sophisticated attention models that introduce their own parameters, which were hypothesized to perform better in the experiments.

The results also show that while the loss of all models trained for 5 epochs is significantly lower than the models trained for 2 epochs, the Development set metrics are slightly worse. See the analysis section for details.

Model	DEV F1	DEV EM	DEV Loss
ALBERT	77.740	73.922	**0.235**
Dot Attention	**78.401**	**74.580**	0.240
Weighted Attention	76.817	72.918	0.239
Encoder	76.57	72.639	0.258

Table 3: Results after 5 epochs of fine-tuneing on the SQuAD 2.0 training set for ALBERT and attention models. The Dot Product Attention model performs the best on F1 and EM.

5 Analysis

While fine-tuning the models for 5 epochs rather than just 2 over the training data significantly decreased the loss, this surprisingly did not improve performance on the Dev an Test sets. It is unclear from looking at the validation performance of whether the issue here is due to overfitting the training data. A common symptom of overfitting shows the validation performance decreasing as training performance improves, however in these experiments, the validation performance rises to a certain threshold and then stagnates and hovers around 77-78 F1.

Upon inspection, one issue during training could perhaps be the linearly decreasing learning rate schedule. A linear schedule decreases the learning rate from the initial value down to 0 at a constant rate over the number of steps (calculated from the size from the number of epochs, batch size, and training set size). Further research would look into how a linear training warm up, as well as a cosine learning rate schedule with hard resets could improve Dev and Test set performance.

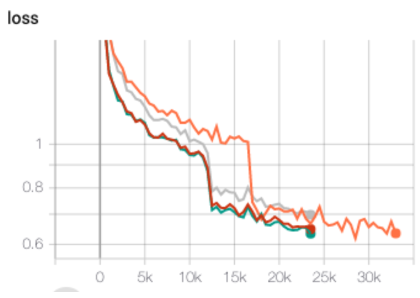

Figure 2: Train loss plot for models fine-tuned over 2 epochs

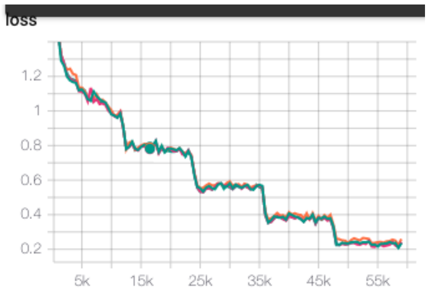

Figure 3: Train loss plot for models fine-tuned over 5 epochs

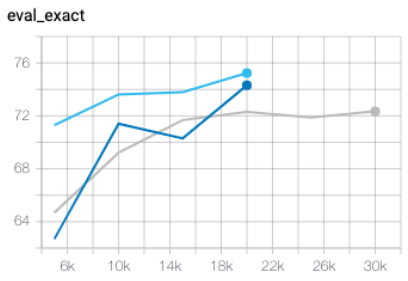

Figure 4: Dev EM plot for models fine-tuned over 2 epochs. Light Blue: Dot Attention, Blue: Weight Attention, Grey: BERT

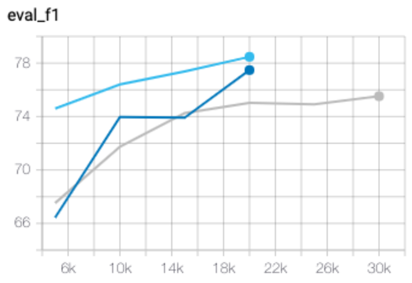

Figure 5: Dev F1 plot for models fine-tuned over 2 epochs. Light Blue: Dot Attention, Blue: Weight Attention, Grey: BERT

6 Conclusion

The purpose of this research was to explore different approaches for creating an improved PCE by using an intermediate model to directly combine all hidden layers of ALBERT. After several experiments, it is clear that only the Dot Attention Model outperforms the baseline (fine-tuning a model using only the last layer of ALBERT). This is an unexpected result, as this is the only model which does not introduce its own parameters for calculating weights for the hidden layers.

The test leader board results show that the Encoder Model was not far behind the best Dot Attention model on the test set. These two approaches are vastly different in terms of the number of parameters and layers of computation they introduce in order to produce an intermediate PCE. This result may show that there is not much to gain in the hidden layers of ALBERT for the down stream task of question answering, since the simple Dot Product model achieves equal or better performance as an Encoder model that applies many computations on each hidden layer to produce a combined PCE.

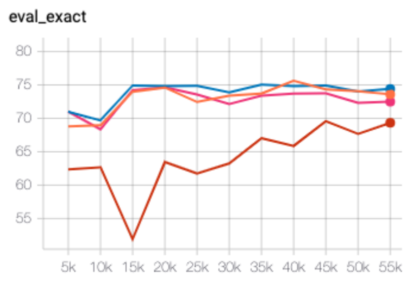

Figure 6: Dev EM plot for models fine-tuned over 5 epochs. Blue: Dot Attention, Red: Weight Attention, Pink: Encoder Attention, Orange: ALBERT

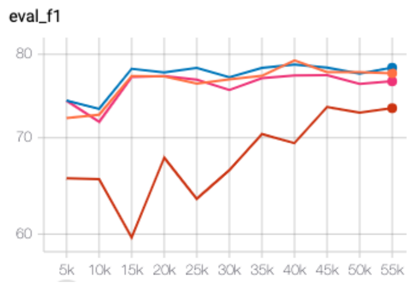

Figure 7: Dev F1 plot for models fine-tuned over 5 epochs. Blue: Dot Attention, Red: Weight Attention, Pink: Encoder Attention, Orange: ALBERT

These results are in some ways encouraging for future research, as the results show that a small increase in performance can be gained by using an intermediate Dot Product Attention PCE Model. This model introduces no parameters of its own and is an extra computation that can be done efficiently for little cost. Further research can explore how this intermediate model affects other downstream tasks where a query can be provided by the fine-tuned model in order to produce intermediate PCEs. It would be an interesting and important result to show whether the intermediate Dot Product Attention Model can lead to a small increase in performance across a range of NLP tasks, or whether it is only applicable to the Question Answering task.

References

[1] Jacob Devlin, Ming-Wei Chang, Lee Kenton, and Toutanova Kristina. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv*, 2018.

[2] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[3] ME Peters, M Neumann, M Iyyer, M Gardner, C Clark, K Lee, and L Zettlemoyer. Deep contextualized word representations. arxiv 2018. *arXiv preprint arXiv:1802.05365*, 1802.

[4] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[6] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *arXiv preprint arXiv:2002.12327*, 2020.

[7] John Hewitt and Christopher D Manning. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, 2019.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

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

7 Appendix

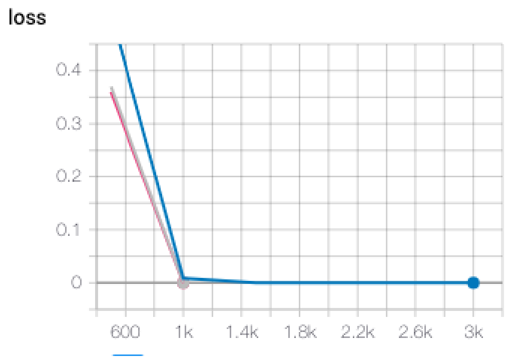

Figure 8: Loss plot showing that custom models can overfit a small amount of example training questions