# Transfer Learning from Pre-trained BERT to Question Answering Task on SQuAD2.0

Dorin Keshales (ID. 313298424)

Submitted as final project report for Practical Topics in Machine Learning, BIU, 2020

## 1 Introduction

Over the past few years, Transfer Learning has led to a new wave of state-of-the-art results in Natural Language Processing (NLP). Transfer Learning's effectiveness comes from pre-training a model on abundantly-available unlabeled text data with a self-supervised task, such as language modeling or filling in missing words. After that, the model can be fine-tuned on smaller labeled datasets, often resulting in (far) better performance than training on the labeled data alone. The recent success of Transfer Learning was ignited in 2018 by GPT [1], ULMFiT [2], ELMo [3], and BERT [4]. BERT, or Bidirectional Encoder Representations from Transformers, is one of the most popular transformer-based models, as for it obtains state-of-the-art results on a wide array of NLP tasks, and is considered a major breakthrough in NLP. BERT is a deeply bidirectional, unsupervised language representation, pre-trained using a combination of Masked Language Modeling objective and Next Sentence Prediction task on a large corpus comprising the Toronto Book Corpus and Wikipedia. It consists of deep Transformer layers, enabling it to capture dependencies across long sequences as well as sentence-level understanding. Its bidirectional representation is conditioned on both left and right context at the same time in all layers. Those features make it an excellent model to produce contextual representations for texts. The pre-training version of BERT (that is, the weights obtained from the Masked Language Modeling and Next Sentence Prediction training routines outlined above) are used as starting weights for a supervised learning phase. During a fine-tuning phase, the previously learned representations are used as a baseline for a problem-specific training. Fine-tuning of BERT is always associated with a particular practical task such as for example Question Answering.

Machine Reading Comprehension (MRC) and Question Answering (QA) is one of the most challenging tasks in NLP, where the machine tries to comprehend the given context paragraph and to provide the correct answer to the question. To measure the performance of the various efforts toward the task, in 2016 Stanford NLP group released Stanford Question Answering Dataset (SQuAD1.1), that is

a reading comprehension dataset which approximates the real reading comprehension circumstances. It consists of 100,000 question-answer pairs each with a given context paragraph and it soon becomes a standard test for the reading comprehension task with public leaderboard available. In 2018, the team further released SQuAD2.0 with over 50,000 unanswerable questions that post a much harder requirement on model development. It largely facilitates the progress in the machine comprehension field and has become one of the most important benchmarks in NLP research. The dataset now have more than 150,000 questions. Each question is either unanswerable with the given context paragraph or can be answered with a continuous span of text in the context paragraph.

In this project, I chose to experiment with the Transfer Learning technique by using BERT model and the reading comprehension question answering task, that is I picked up pre-trained BERT language model and tried to fine-tune it, with two different fine-tuning techniques, on the Question Answering task using SQuAD2.0 dataset. The reason for choosing this idea to be the theme of my project is that in the last year, I have taken several Machine Learning and Deep Learning courses, where I have learned a lot about the Transfer Learning approach at the theoretical level. Since then, I have had the desire to also experience with practical application of the Transfer Learning approach by applying it using one of the popular pre-trained models known today. As I was looking for a downstream task to fine-tune the model on, I came across with the Question Answering task that I found very interesting and challenging, since it requires a comprehensive understanding of natural languages and the ability to do further inference and reasoning. From 'BERT for Question Answering on SQuAD 2.0' paper [5], I got inspired to pick BERT model and the Stanford Question Answering Dataset (SQuAD2.0) as the pre-trained model and the dataset for this work.

So far, my best model has achieved an F1 score of 74.008 and EM score of 69.434 on the dev set.

## 1.1 Related Works

Traditionally, previous state-of-the-art QA systems (now called non-PCE models) relied on pre-trained word vectors and a variety of high-performing networks, including Bidirectional Attention Flow (BiDAF) [6] and Dynamic Coattention Network (DCN) [7]. These systems all performed extremely well on SQuAD1.1 but far worse on SQuAD2.0, which contains over 50,000 adversarial questions that could not be answered. Then, in October 2018, BERT [4] was released and achieved state-of-the-art results for several different NLP tasks, including QA. This is due to the fact that, among other additions, BERT uses Pre-trained Contextual Embeddings (PCE), where embeddings for every token are learned specifically within the context of their surrounding words. This encodes much more locale-specific information than traditional pre-trained vectors that are more general and not tied to a particular context. In recent years, Pre-trained Contextual Embeddings (PCE) language models, such as ELMo[3] or BERT [4],

are widely used on question answering tasks, and the modifications of BERT reach the state-of-art performance on SQuAD2.0.

Although, as described, non-PCE models didn't work so well as the PCE ones, there was an attempt presented in 'Question Answering on SQuAD2.0' paper [8] to build a non-PCE Question Answering model on SQuAD2.0. They used the non-PCE method, BiDAF as their baseline, where BiDAF is a bi-directional network for question answering that represents the interactions of query and context at different levels of granularity. BiDAF mainly consists of 6 layers: word embedding layers; a contextual embedding layer, a bi-directional LSTM on both query and context; an attention flow layer that compute the flow from context to query and vice versa; a modeling layer, a bi-directional LSTM that encodes context words; and an output layer. In their experiment, they made several modifications to the original BiDAF model [6] (the baseline) including adding character-level embeddings and word features in the embedding layer, applying self-attention layer, different optimizers, drop-out rates, hidden layer dimensions and alternate types of RNN, to improve its performance on SQuAD2.0. Their model achieved significant improvement compared with the original BiDAF model.

According to the paper [4], the pre-trained BERT (PCE model) representations can be fine-tuned with additional architectures to succeed in specific tasks. Therefore, as part of the SquAD2.0 challenge, many participants tried to build their own output network on top of pre-trained BERT model to improve performance on SQuAD2.0. For example, the authors of 'BERT for Question Answering on SQuAD 2.0' paper [5] designed several modules on top of BERT model as the task-specific output layers, according to insights they gained from other networks. Their intention was to build a subsequent encoder-decoder architecture as post-processing to improve BERT model's performance on the SQUAD2.0 challenge. For their main encoder-decoder architecture, they adapted RNN-based bi-directional long short-term memory layer (LSTM) [9] and gated recurrent units (GRU) [10] as the encoder and decoder, which are commonly used in sequence to sequence [11] translation task. They also tried a CNN-based encoder block, which is implemented in QANet [12] and CharCNN [13] networks. For multi-layer state transitions in their recurrent neural networks, they used highway network [14] to adaptively copy or transform representations. And for the final output layer, they compared the original linear layer and the QA output layer from Bi-Directional Attention Flow paper [6]. They also tried to ensemble their model with BERT-Large-Cased models for better performance. To sum up, by replacing the linear BERT output layer with an encoder-decoder architecture, they successfully implemented the task-specific layers that can deal with the SQuAD2.0 problems quite well. Their best-proposed single model built an LSTM encoder-decoder structure followed by a highway network as the output layers, on top of the BERT-base-uncased model and achieved an F1 score of 77.96 on the dev set. The ensemble version achieved 79.44 on the Dev Set and 77.827 on the Test Set.

In this project, I simply adapted pre-trained BERT-Base model (described in paper [4]) with minimal task-specific output layers and fine-tuned it on SQuAD2.0 with two different fine-tuning techniques.

# 2 Solution

## 2.1 General approach

The task of Fine-Tuning a network is to tweak the parameters of an already trained network so that it adapts to the new task at hand. There are different fine-tuning techniques. In this project, I chose to examine and compare two fine-tuning techniques on BERT-Base-Uncased model, released by Google research in 2018, with the QA task using the SQuAD2.0 dataset. The first technique, which is applied in Model_1, is to train the entire architecture. That is, to further train the entire pre-trained model together with additional task-specific layers on the SQuAD2.0 dataset and feed the final output to a softmax layer. In this technique, the error is back-propagated through the entire architecture and the pre-trained weights of the model are updated based on the new dataset (SQuAD2.0). The second technique, which is applied in Model_2, is to train some layers while freezing others. That is, to train it partially - to keep the weights of the initial layers of the model frozen while retraining only the higher layers. This technique became popular, among other reasons, following observations that have been made in NLP literature regarding pre-trained language models. For example, Clark et al. (2019) [15] analyzed BERT's attention and observed that the bottom layers attend broadly, while the top layers capture linguistic syntax. Kovaleva et al. (2019) [16] found that the last few layers of BERT change the most after task-specific fine-tuning. So for the matter of this work, In Model_2, only the 4 last layers of BERT were retrained while the other 8 layers were frozen in training.

In this project, I picked up the BERT-Base-Uncased model as my starting point [17, 18] for both models, Model_1 and Model_2. The pre-trained BERT-Base model can be easily adapted to span prediction tasks, i.e. question answering on SQuAD2.0, as in Figure 1. To achieve this, the input question and paragraph pairs are represented as a single packed sequence, with the question using the A segmentation embeddings and the paragraph using the B segmentation embeddings. The pairs are separated by a special token ([SEP]). In the original approach, the predicted answer span is learned by introducing two new learned vector parameters for the fine tuning: a start vector $S \in R^h$ and end vector $E \in R^h$. If we consider the final hidden vector from BERT for the i'th input token as $Ti \in R^h$, we can calculate the probability of word i being the start of the answer span as a dot product between Ti and S followed by a softmax over all of the words in the paragraph. And respectively perform dot product between Ti and E in order to calculate the probability of word i being the end

of the answer span. Due to runtime constraint those two learned vectors - S and E - were replaced with two fully connected linear layers $W, U \in R^{h \times 1}$ (one for outputting the start logits and the other one for outputting the end logits). Specifically for SQuAD2.0, I used the special token [CLS], which is added to the head of every input text sequence, as the ground truth start and end positions, for unanswerable questions. Therefore, probabilities corresponding the [CLS] token give the probability of the input paragraph-question being unanswerable. As a result, the predicted answer's start position and end position are equal to 0.

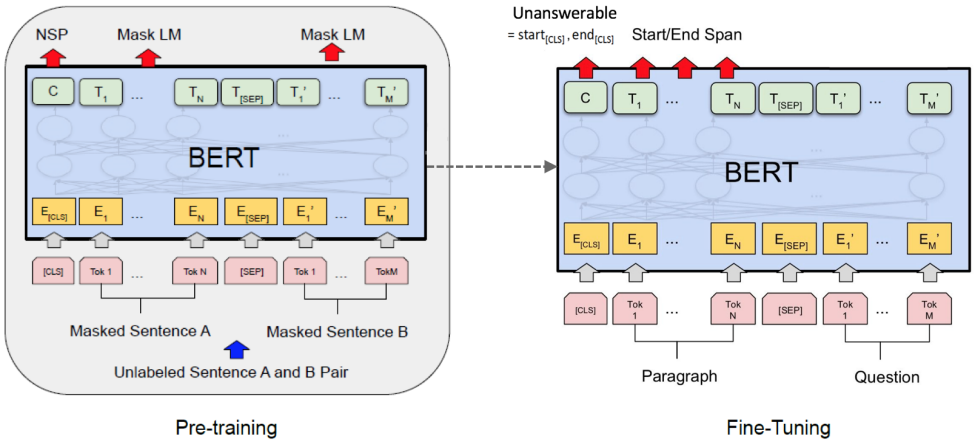


Figure 1: Figure representing the BERT pre-training and fine-tuning flow and network architecture. I used the BERT pre-trained model for question answering task on SQUAD2.0 dataset and fine-tuned the weights in two different fine-tuning techniques to improve performance.

If due to GPU memory constraint the BERT-Base-Uncased model couldn't fit within my GPU memory, then I would use the DistilBERT-Base-Uncased model as an alternative pre-trained model. DistilBERT [19] is a small, fast, cheap and light Transformer model trained by distilling Bert base that has 40% less parameters than Bert-Base-Uncased and runs 60% faster while preserving over 95% of Bert's performances. Luckily, the BERT- Base-Uncased model was able to fit within my GPU memory.

## 2.2 Design

The code for both models was written in Python using PyTorch framework. I used Jupyter notebooks of Google Colab platform for executing the experiments and the PyCharm IDE for debugging. The pre-trained BERT-Base model was taken from HuggingFace's PyTorch implementation of BERT [18].

As for the training time, for both models, I started with initial performance evaluation on the dev set before the training process begins, for later comparison between their pre-training performance on the dev set to their post-training performance i.e. understanding of how well the models are learning and generalizing on the dev set. Both models were trained for 2-3 epochs. One epoch is composed of training epoch i.e. performing one full pass over the training set, and then evaluation of model's performance on the dev set. Each epoch in Model_1 takes approximately 2.1 hours, where the training epoch takes around 2 hours and the performance evaluation takes around 4 minutes. The total training process of Model_1 takes approximately 6.3 hours. In Model_2 each epoch takes approximately 1.7 hours, where the training epoch takes around 1.6 hours and the performance evaluation takes around 4 minutes. The total training process of Model_2 takes approximately 5.1 hours.

I had two technical challenges during the work on the project. The main technical challenge was to deal with the Bert-Base model being a very large model with around 110M parameters. Therefore, I had to find a GPU platform (since training on CPU would have taken a non-reasonable amount of time) that: (1) Can supply a GPU memory large enough to contain the whole model (Bert-Base model + task specific output layers) within it. (2) Its session lifetime is long enough for the model to complete its run which contains several epochs, since as described earlier even with GPU usage the training process took hours upon hours. Luckily, I managed to work with Google Colab platform after conducting a few experiments to learn the Google Colab's GPU runtime limitations, and using some online tricks that helped me to keep the session alive long enough as I needed. Of course, I had to do some small adjustments in the code in order to shorten the runtime so it could meet the Google Colab's runtime limitations, for example replacing the two learned vector parameters with two fully connected linear layers. The second technical challenge I encountered while working on the project and was much easier to overcome, was the need to understand and learn how to work with the HuggingFace library. That is, understand how to load a pre-trained model and how to work with it, how the tokenized data should look, how to use the tokenizer correctly to tokenize all the data at once and not per sentence, what input Bert-Base model expects to receive and in what form, and how and why I should use the AdamW optimizer (Which is part of the HuggingFace library) as an optimizer for updating the model's weights during the training.

# 3 Experiments

## 3.1 Data

Stanford Question Answering Dataset (SQuAD) is one of the most widely-used reading comprehension benchmarks. In this project, I used SQuAD2.0 to train and evaluate the models. SQuAD2.0 is a reading comprehension dataset on a set of Wikipedia articles with more than 150,000 questions. Samples in this dataset include (question, answer, context paragraph) tuples. Compared with SQuAD1.1, SQuAD2.0 introduces over 50,000 unanswerable questions written by crowdworkers similar to the answerable ones. This requires the system not only to answer reading comprehension questions, but also to differentiate when there exists no answer to the question. However, if a question is answerable, the answer is guaranteed to be a continuous span in the context paragraph. The original SQuAD2.0 dataset has three splits, train, dev and test, with the first two publicly accessible and the last one held privately. The training set has more than 130,000 questions and the dev set has around 12,000 questions.

## 3.2 Measurement metric

In order to evaluate Model_1 and Model_2, I used the evaluation script made by the 'SQuAD2.0 Challenge' team, for official evaluation. In this evaluation script two metrics are applied to measure the performance of the model: Exact Match (EM) score and F1 score.

**EM** is a binary measurement (true/false) of whether the percentage of output from a system exactly matches the ground truth answer. It's the proportion of questions that are answered in exact same words as the ground truth. This is a fairly strict metric.

**F1 score** is a less strict metric, it is the harmonic mean of precision and recall. For each question, precision is calculated as the number of correctly predicted words divided by the total words in the predicted answer. Recall is the number of correctly predicted words divided by the number of words in the ground truth answer. $F1 = \frac{2 \times precision \times recall}{precision + recall}$. The system would have 100% precision if its answer is a subset of the ground truth answer and 50% recall if it only includes one out of the two words in the ground truth output.

When a question has no answer, both the F1 and EM score are 1 if the model predicts no-answer, and 0 otherwise.

## 3.3 Experimental details

I used HuggingFace's PyTorch implementation of BERT [18]. The pre-trained BERT model used is BERT-base-uncased, which contains around 110M parameters. In Model_1, all BERT parameters (in all layers) are not frozen in training

but fine-tuned together with the additional linear layers, while in Model_2 only the parameters of the last 4 layers of BERT are not frozen in training. The models are trained with AdamW optimizer included in HuggingFace's implementation, which is an optimizer with weight decay fixed that can be used to fine-tuned models. The seed value is set to 42 and the max sequence length is set to 512 as the Bert-Base's default definition. The conducted experiments included tuning of the following hyperparameters: learning rate, batch size, number of training epochs, dropout, weight decay and learning rate decay factor. Due to GPU memory constraint, the batch sizes that were tested were 6 and 10. In concept, the use of weight decay, dropout and learning rate decay came to reduce overfitting on the training data and as a result help the models to better generalize on the dev set. I ran every experiment on a single GPU. Due to the long runtime (of both models), the experiments were conducted in groups of 2-5 experiments that ran in parallel, each from a different Google Colab's account. The results of the experiments performed on Model_1 and Model_2 are summarized in Table 1 and Table 2, respectively.

| ID | Tunable Parameters | | | | | | Score | |
|---|---|---|---|---|---|---|---|---|
| | Learning Rate | Batch Size | No. epochs | Learning Rate Decay | Weight Decay | Dropout | EM | F1 |
| 1 | 2e-5 | 10 | 3 | - | - | - | 68.213 | 72.597 |
| 2 | 2e-5 | 10 | 3 | - | - | 0.1 | 66.562 | 70.969 |
| 3 | 2e-5 | 6 | 3 | - | - | - | 66.091 | 70.584 |
| 4 | 2e-5 | 10 | 3 | 0.5 (every epoch) | - | - | 69.173 | 73.551 |
| 5 | 2e-5 | 6 | 3 | 0.5 (every epoch) | - | - | 67.775 | 72.27 |
| 6 | 2e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | - | **69.434** | **74.013** |
| 7 | 2e-5 | 10 | 3 | 0.5 (every 2 epochs) | 1e-4 | - | 69.434 | 74.013 |
| 8 | 2e-5 | 10 | 3 | 0.5 (every 2 epochs) | 1e-2 | - | 68.466 | 72.937 |
| 9 | 2e-5 | 10 | 3 | 0.5 (every 2 epochs) | 0.1 | - | 69.249 | 73.843 |
| 10 | 2e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | - | 67.211 | 71.735 |
| 11 | 1e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | - | 67.716 | 72.111 |
| 12 | 1e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | - | 68.137 | 72.535 |
| 13 | 9e-6 | 10 | 3 | - | - | - | 67.615 | 71.944 |
| 14 | 9e-6 | 6 | 3 | - | - | - | 69.476 | 73.379 |
| 15 | 3e-5 | 10 | 3 | 0.5 (every epoch) | - | - | 68.289 | 72.743 |
| 16 | 3e-5 | 6 | 3 | 0.5 (every epoch) | - | - | 67.211 | 71.412 |
| 17 | 3e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | - | 67.986 | 72.31 |
| 18 | 3e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | - | 64.97 | 69.404 |
| 19 | 4e-5 | 10 | 3 | 0.5 (every epoch) | - | - | 68.230 | 72.737 |
| 20 | 4e-5 | 6 | 3 | 0.5 (every epoch) | - | - | 65.627 | 69.875 |
| 21 | 4e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | - | 66.503 | 71.172 |
| 22 | 4e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | - | 62.629 | 66.841 |
| 23 | 5e-5 | 10 | 3 | 0.05 (every 10000 train examples) | - | - | 66.554 | 71.059 |

Table 1: Model_1 performance on SQuAD2.0 dataset in conducted experiments with different hyperparameters.

## 3.4 Experimental results

Table 3 shows the best scores of the models explored in this project. The scores are broken down to all questions, the answerable questions and the unanswerable

| ID | Tunable Parameters | | | | | Score | |
|---|---|---|---|---|---|---|---|
| | **Learning Rate** | **Batch Size** | **No. epochs** | **Learning Rate Decay** | **Weight Decay** | **EM** | **F1** |
| 1 | 2e-5 | 10 | 3 | - | - | 64.347 | 68.475 |
| 2 | 2e-5 | 6 | 3 | - | - | 64.718 | 68.78 |
| 3 | 2e-5 | 10 | 3 | 0.5 (every epoch) | - | 62.065 | 66.423 |
| 4 | 2e-5 | 6 | 3 | 0.5 (every epoch) | - | 63.362 | 67.56 |
| 5 | 2e-5 | 10, | 3 | 0.5 (every 2 epochs) | - | 63.951 | 68.324 |
| 6 | 2e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | 64.516 | 68.592 |
| 7 | 3e-5 | 10 | 3 | - | - | 65.29 | 69.361 |
| 8 | 3e-5 | 6 | 3 | - | - | 66.166 | 70.061 |
| 9 | 4e-5 | 10 | 3 | - | - | 66.689 | 70.681 |
| 10 | 4e-5 | 6 | 3 | - | - | 66.47 | 70.459 |
| 11 | 4e-5 | 10 | 3 | 0.5 (every epoch) | - | 64.726 | 69.087 |
| 12 | 4e-5 | 6 | 3 | 0.5 (every epoch) | - | 65.636 | 69.987 |
| 13 | 4e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | 66.099 | 70.243 |
| 14 | 4e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | 66.823 | 70.978 |
| 15 | 4e-5 | 6 | 3 | 0.5 (every 2 epochs) | 0.1 | 66.983 | 70.968 |
| 16 | 5e-5 | 10 | 3 | - | - | 66.554 | 70.619 |
| 17 | 5e-5 | 6 | 3 | - | - | 66.166 | 70.121 |
| 18 | 5e-5 | 10 | 3 | 0.5 (every epoch) | - | 65.459 | 69.771 |
| 19 | 5e-5 | 6 | 3 | 0.5 (every epoch) | - | 66.049 | 70.383 |
| 20 | 5e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | 66.554 | 70.751 |
| 21 | 5e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | 65.459 | 69.739 |
| 22 | 6e-5 | 10 | 2 | - | - | 66.798 | 70.511 |
| 23 | 6e-5 | 10 | 3 | - | 1e-2 | 66.04 | 70.227 |
| 24 | 6e-5 | 6 | 3 | - | - | 66.36 | 70.466 |
| 25 | 6e-5 | 10 | 3 | 0.5 (every epoch) | - | 66.276 | 70.557 |
| 26 | 6e-5 | 6 | 3 | 0.5 (every epoch) | - | 65.99 | 70.267 |
| 27 | 6e-5 | 10 | 3 | 0.5 (every 2 epochs) | - | **67.354** | **71.761** |
| 28 | 6e-5 | 10 | 3 | 0.5 (every 2 epochs) | 1e-2 | 66.242 | 70.524 |
| 29 | 6e-5 | 10 | 3 | 0.5 (every 2 epochs) | 0.1 | 67.186 | 71.421 |
| 30 | 6e-5 | 6 | 3 | 0.5 (every 2 epochs) | - | 66.461 | 70.622 |
| 31 | 7e-5 | 10 | 3 | - | - | 66.647 | 70.856 |
| 32 | 7e-5 | 10 | 3 | 0.05 (every 10000 train examples) | - | 65.316 | 69.662 |

Table 2: Model_2 performance on SQuAD2.0 dataset in conducted experiments with different hyperparameters.

questions. The initial evaluation score of the models (same initial score for both) also appears in Table 3 as kind of a baseline model describes the performance of Bert-Base on the dev set of SQuAD2.0, without any training or fine-tuning process. In the initial evaluation, The score is 22.451 F1 and 20.567 EM. As for the models scores, the best score Model_1 gets on the dev set is 74.013 F1 and 69.434 EM, and the best score Model_2 gets on the dev set is 71.761 F1 and 67.354 EM. If we compare the initial evaluation score obtained before training to the scores of Model_1 and Model_2 post-training, it is easy to see that in both models a successful learning process was performed - both in terms of predicting the correct answer locations for answerable questions and in terms of detecting whether the question is answerable or unanswerable. It can be inferred from Table 3, that both models are a bit better in detecting unanswerable questions than in predicting the correct answer when the questions are answerable.

| Model | Overall | | Answerable | | Unanswerable | |
|---|---|---|---|---|---|---|
| | **EM** | **F1** | **EM** | **F1** | **EM** | **F1** |
| Baseline (Initial Validation) | 20.567 | 22.451 | 0.033 | 3.806 | 41.042 | 41.042 |
| Model_1 | 69.434 | 74.013 | 63.225 | 72.395 | 75.626 | 75.626 |
| Model_2 | 64.153 | 68.514 | 57.675 | 66.409 | 70.613 | 70.613 |

Table 3: Has-Answer and No-Answer F1 and EM scores for the models explored in this project.

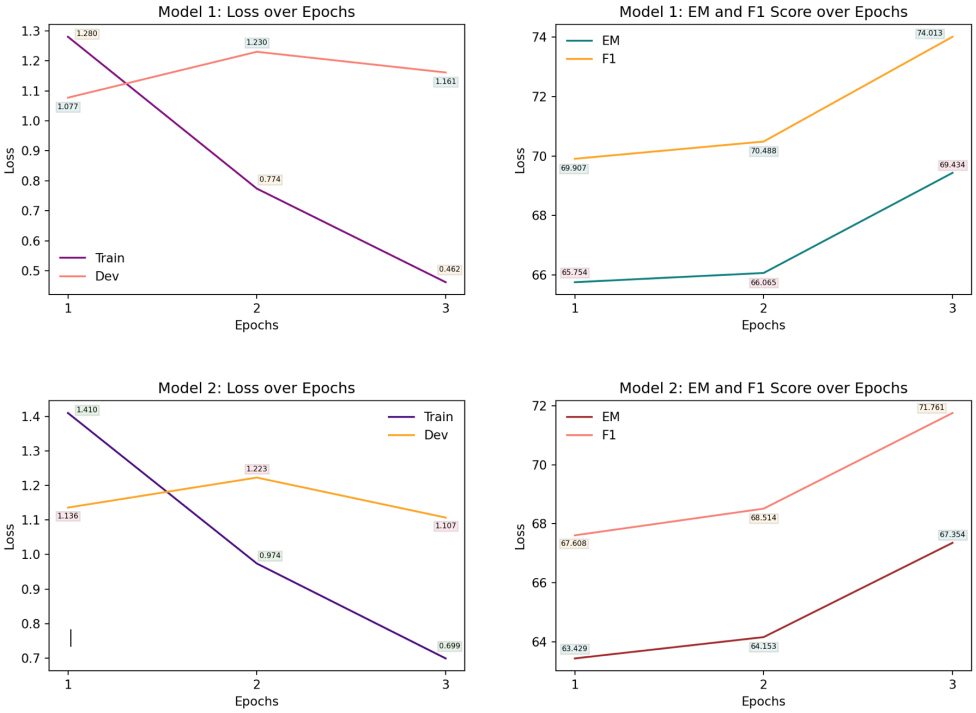


Figure 2: The upper graphs represent the averaged training loss and the averaged dev loss over epochs in Model_1 and Model_2, respectively. The lower graphs represent the EM and F1 score over epochs in Model_1 and Model_2, respectively.

# 4 Discussion

For both modes, I have tried a lot of different parameter combinations to achieve final performance. An unexpected result was finding out that BERT-Base model that was fine-tuned with it's entire architecture (Model_1), i.e. all BERT layers were fine-tuned together with the additional task-specific output layers, outper-

forms a BERT-Base model that only 4 out of its 12 layers were retrained. My initial thought was that retraining only the last few layers of BERT-Base while freezing the others, will reduce and even prevent overfitting on the training data and as a result, will cause to better generalization on the dev set and to better performance on SQuAD2.0. If we compare the two left graphs in Figure 2, which describe the loss per epoch in Model_1 and Model_2 respectivly, we can see that the overfitting was indeed reduced in Model_2 but not quite disappeared as I first thought. Another interesting observation that can be seen in the two right -graphs in Figure 2, is that the trend of EM score and F1 score looks identical in both models and wasn't changed due to the reduction of overfitting in Model_2, except Model_2's lower performance. It seems that although the training data is overfitted in Model_1, with each additional epoch there is an improvement with the EM and F1 score, which makes me think that in this case overfitting on the training data does not prevent from the model to generalize well on the dev set.

## 4.1 Qualitative analysis

### 4.1.1 Examples

| Question | Answer | Model_1 | Model_2 |
|---|---|---|---|
| Other than the steamboat, what modern form of travel brought visitors to Florida? | railroad | railroad | railroad |
| What impact does higher worker productivity and leveled pay have on lower earners? | No answer | relatively stagnant wages | No answer |
| What index is an indicator of the effects of taxes applied to social spending? | the Gini index | gini index | No answer |
| What does Vriscovci mean in Polish? | No answer | belonging to warsz | belonging to warsz |
| What did creating highways in the Amazon rainforest lead to? | increased settlement and deforestation | increased settlement and deforestation | increased settlement and deforestation |
| Charleston settler Elie Prioleau was from what French town? | Pons | pons | pons |

Table 4: Examples

### 4.1.2 Analysis

#### 4.1.2.1 Paraphrases

In some error cases, the core concept the question cares about is expressed in a different way in the context paragraph. When those paraphrases differ in wording, the models will fail to recognize that they refer to the same thing.

**Question:** What is the seldom used force unit equal to one thousand newtons?

**Context paragraph:** Other arcane units of force include the sthène, which is equivalent to 1000 N, and the kip, which is equivalent to 1000 lbf.

**Answers:** sthène
**Prediction of Model_1:** No answer
**Prediction of Model_2:** No answer

#### 4.1.2.2 Logical reasoning

Sometimes guiding words in the question are not directly mentioned in the context paragraph, making it difficult for the models to notice the existence of the answer in the context paragraph.

**Question:** What was the Italian title of Polo's book?

**Context paragraph:** The first recorded travels by Europeans to China and back date from this time. The most famous traveler of the period was the Venetian Marco Polo, whose account of his trip to C̈ambaluc,̈the capital of the Great Khan, and of life there astounded the people of Europe. The account of his travels, Il milione (or, The Million, known in English as the Travels of Marco Polo), appeared about the year 1299. Some argue over the accuracy of Marco Polo's accounts due to the lack of mentioning the Great Wall of China, tea houses, which would have been a prominent sight since Europeans had yet to adopt a tea culture, as well the practice of foot binding by the women in capital of the Great Khan. Some suggest that Marco Polo acquired much of his knowledge through contact with Persian traders since many of the places he named were in Persian.

**Answers:** Il milione
**Prediction of Model_1:** No answer
**Prediction of Model_2:** No answer

In this example, the keywords "Italian title" and "book" are not directly mentioned in the context. To correctly generate the answer requires logical reasoning like human beings, which my models cannot achieve currently.

#### 4.1.2.3 Partial match

Sometimes, the models tend to find answer in the context paragraph that partially matches the concept talked about in the question, but fails to notice the important words that further describe that concept. For example, in some cases, what question asks about is opposite to what is described in the context paragraph. The model may focus on the similarity in phrases but fail to capture the difference in the direction of meaning.

**Question:** How many Protestant Walloons fled to England before the Foreign Protestants Naturalization Act was passed?

**Context paragraph:** Both before and after the 1708 passage of the Foreign Protestants Naturalization Act, an estimated 50,000 Protestant Walloons and Huguenots fled to England, with many moving on to Ireland and elsewhere. In relative terms, this was one of the largest waves of immigration ever of a single ethnic community to Britain. Andrew Lortie (born André Lortie), a leading Huguenot theologian and writer who led the exiled community in London, became known for articulating their criticism of the Pope and the doctrine of transubstantiation during Mass.

**Answers:** No answer
**Prediction of Model_1:** 50,000
**Prediction of Model_2:** 50,000

Those three types of errors show that what the models are doing is still kind of text matching. The models' ability in reasoning still need improvement.

# 5 Code

Click here to Model_1 notebook.
Click here to Model_2 notebook.
Click here to SQuAD2.0 data.

# References


[1] Tim Salimans Ilya Sutskever Alec Radford, Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. s3-us-west-2.amazonaws.com.

[2] Sebastian Ruder Jeremy Howard. Universal language model fine-tuning for text classification. 2018. arXiv:1801.06146.

[3] Mohit Iyyer Matt Gardner Christopher Clark Kenton Lee Luke Zettlemoyer Matthew E. Peters, Mark Neumann. Deep contextualized word representations. 2018. arXiv:1802.05365.

[4] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018. arXiv:1810.04805.

[5] Zhaozhuo Xu uwen Zhang. Bert for question answering on squad 2.0. https://stanford.io/37CpL3P.

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

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

[8] Qiwen Wang Mengyu Li, Boyao Sun. Question answering on squad2.0. https://web.stanford.edu/class/cs224n//reports/default/15816213.pdf.

[9] Mirella Lapata Jianpeng Cheng, Li Dong. Long short-term memory-networks for machine reading. 2016. arXiv:1601.06733.

[10] Caglar Gulcehre Dzmitry Bahdanau Fethi Bougares Holger Schwenk Yoshua Bengio Kyunghyun Cho, Bart Van Merriënboer. Learning phrase representations using rnn encoderdecoder for statistical machine translation. 2014. arXiv:1406.1078.

[11] Sean Robertson. Knuth: Computers and typesetting, 2018. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html.

[12] Minh-Thang Luong Rui Zhao Kai Chen Mohammad Norouzi Quoc V Le Adams Wei Yu, David Dohan. Qanet: Combining local convolution with global self-attention for reading comprehension. 2018. arXiv:1804.09541.

[13] Yann LeCun Xiang Zhang, Junbo Zhao. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, pages 649—-657, 2015.

[14] Jürgen Schmidhuber Rupesh Kumar Srivastava, Klaus Greff. Highway networks. 2015. arXiv:1505.00387.

[15] Omer Levy Christopher D. Manning Kevin Clark, Urvashi Khandelwal. What does bert look at? an analysis of bert's attention. 2019. arXiv:1906.04341.

[16] Anna Rogers Anna Rumshisky Olga Kovaleva, Alexey Romanov. Revealing the dark secrets of bert. 2019. arXiv:1908.08593.

[17] Github, 2018. https://github.com/google-research/bert.

[18] Github, 2018. https://github.com/huggingface/pytorch-pretrained-BERT.

[19] Julien Chaumond Thomas Wolf Victor Sanh, Lysandre Debut. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. 2019. arXiv:1910.01108.