Advanced NLP: Major Project

WikiQA-based Open Domain Question Answering

Final Report

Sagar Joshi (2020701007) | Abhishek Tyagi (20173067) | Gadela Kesav (2018101079)

Team 33: ChatBot | Mentor: Priyanka Ravva | Github Link | Presentation Link

Table of Contents

Introduction

The WikiQA dataset was introduced for the problem of open domain question answering in order to provide a challenging dataset for the task. The dataset has been used as a testbed for sophisticated deep learning algorithms to determine answers to a given question by making use of information more involved than simpler lexical matching based inferences. The specific task in question answering dealt with in this dataset is that of answer sentence selection. This task can be formulated as below:
Given a question and a set of candidate sentences - one of which could be the correct answer to the question - select the appropriate sentence.

In the creation of this dataset, the authors also introduced another task of answer triggering. This task involves determining whether either of the candidates answers the given question when provided the input as that for answer selection. The problem was introduced to move towards more realistic settings, where the retrieved sentences may not really contain the answer. Such question - candidate sets having no answer present in them were introduced to make about 2/3rd of the dataset size.

In our project, we focus on the answer selection problem. While determining the scope, we went through a number of benchmarks established on the dataset since its introduction before narrowing down on the models we finalized to implement as part of the project. This included using attentive pooling networks using BiLSTMs as a formidable baseline and the TANDA finetuning strategy as our final model. In the process, we also implemented a vanilla BERT-finetuned model as a second baseline check for TANDA performance.

The subsequent report includes a detailed description of these implementations, the preprocessing, and important considerations made through the process, along with the results obtained and our analysis of them.

Datasets & Preprocessing

WikiQA was the primary dataset for the answer selection task, going by the project scope. However, implementing the TANDA approach also required making use of the Answer-Sentence Natural Questions (ASNQ) dataset.

As a common preprocessing for both the datasets, regex-based cleaning to remove punctuations and lowercasing was done before feeding them to the tokenization module. The maximum length for truncation was set based on the distribution of lengths in both the datasets, the specifics of which are provided in the forthcoming sections for the respective dataset. The answer selection task was solved from the point of view of BERT-based encoder architectures as a sentence pair classification task. So, the tokenization scheme followed was as below:

[CLS] tokens of question [SEP] tokens of sentence [SEP]

For the first baseline where BiLSTMs and CNNs were used for modeling the problem, a much simpler space-based tokenization was done before passing them for retrieving their GloVe embeddings.

1. WikiQA Dataset [3]

The WikiQA Dataset was created from Bing query logs, with questions mapped to the introduction sections of a Wikipedia page retrieved on search, which may or may not contain the answer to the question. The statistics of the WikiQA dataset are shown below.

	Train	Dev	Test	Total
# of ques.	2,118	296	633	3,047
# of sent.	20,360	2,733	6,165	29,258
# of ans.	1,040	140	293	1,473
Avg. len. of ques.	7.16	7.23	7.26	7.18
Avg. len. of sent.	25.29	24.59	24.95	25.15
# of ques. w/o ans.	1,245	170	390	1,805

Since we are dealing with the answer selection task in this project, only the subset of the dataset having at least one sentence as the answer to the question was used, which had, in the train split, 873 questions mapped to 8,672 sentences, with the binary labels indicating whether the sentence contains the answer to the question.

From the train split of the WikiQA Dataset, the distribution of token lengths from a BERT tokenizer is as shown below. The scheme for this tokenization is as mentioned before.

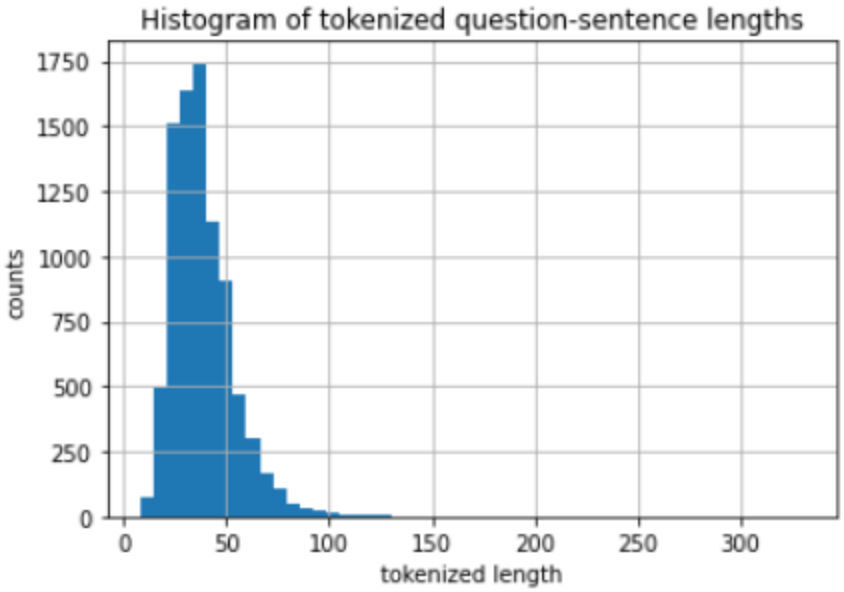


Fig 1. Distribution of token lengths from the WikiQA dataset

Based on these visible statistics, the maximum length for the tokenized question-sentence pair was capped to 64.

2. Answer-Sentence Natural Questions (ASNQ) Dataset [4]

The ASNQ dataset was created from the Google Natural Question dataset which was created for the reading comprehension task. The original dataset is composed of long paragraphs corresponding to a question, which may contain phrases annotated as the short answer and contains 57,242 unique questions in the train split. The ASNQ dataset was intended to be a large-scale generic dataset for the answer selection task, thus the natural questions dataset was re-arranged in the form of annotated question-answer pairs to introduce the ASNQ dataset. In the creation of the dataset, sentences that were contained in the long answer and had the short answer contained in them were labeled as positive samples. The rest of the samples were treated as negative samples. Statistics of the proportion of the different types of positive and negative samples are shown in the table below.

Label	$S \in LA$	$SA \in S$	# Train	# Dev
1	No	No	19,446,120	870,404
2	No	Yes	428,122	25,814
3	Yes	No	442,140	29,558
4	Yes	Yes	61,186	4,286

Here, S indicates the sentence, LA and SA indicate Long Answer and Short Answer respectively. Thus, there is a large (about 19.5 million) number of negative samples such that they do not belong to the long answer, nor have short answer contained in them, constituting type 1 negative samples. While the presence of a large number of negative samples is argued to be important for bringing in robustness to the model, an interesting insight was obtained from the ablations performed by the authors in applying the

TANDA approach. The authors applied the strategy (described later) on different combinations of negative sample types, to find that the performance on using only sample types 2 & 3 as negative samples while keeping type 4 samples as positive gave a performance almost as good as that on using all 1, 2, 3 sample types as negative samples. The result of this ablation study is shown below.

Model	WikiQA		TREC-QA	
	MAP	MRR	MAP	MRR
Neg: 1 Pos: 4	0.870	0.880	0.808	0.847
Neg: 2 Pos: 4	0.751	0.763	0.662	0.751
Neg: 3 Pos: 4	0.881	0.895	0.821	0.869
Neg: 2,3 Pos: 4	0.883	0.898	0.823	0.871
Neg: 1,2,3 Pos: 4	0.884	0.898	0.823	0.872

This study gave an important decision for us in filtering the data since the unfiltered train size of over 20 million question-sentence pairs was too computationally heavy to deal with. We filtered and stored an intermediate form of the ASNQ dataset by treating sample types 2 & 3 as negative samples and sample type 4 as positive samples. Thus, our filtered dataset size was 931,448 samples in train and 59,658 samples in dev splits. This was a much more manageable but also substantial sized data for fine-tuning a transformer model.

The distribution of token lengths (with the question and sentence tokens concatenated) from 10,000 random samples from the train split with the maximum length capped to 256 is as shown below.

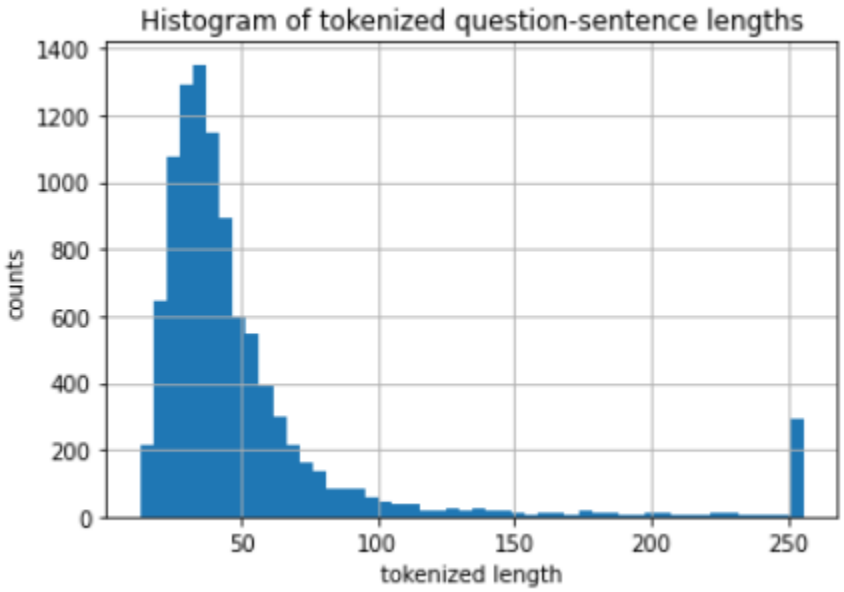


Fig 2. Distribution of token lengths from the ASNQ dataset

Based on these statistics, the maximum length of the tokenization output was capped to 128, and longer sequences were truncated.

Evaluation

The MAP and MRR metrics have been traditionally used in literature for evaluating the performance of the QA systems on answer selection.

Mean Average Precision (MAP): This metric summarizes the precision-recall curve of the system, taking the mean of the precision of the results achieved at each threshold for answer selection. It is a popular metric used in information retrieval for search result ranking evaluation

Mean Reciprocal Rank (MRR): It is the average of the reciprocals of the rank given to the correct answer by the system. If the sentence containing the answer is ranked first, a full score will be rewarded for the performance at that instance, else the score awarded by the metric will decrease inversely with the assignment of lower rank to the correct answer. For a set of Q questions, with each instance i having its candidate sentence ranked at position rank i , the metric can be formulated as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

In addition, **accuracy** was also used to measure in transformer encoder-based finetuning as an added metric for evaluation, especially since it involved treating the answer selection task as a sentence pair classification task.

Baseline: Attentive Pooling Networks [1]

1. Introduction

Attentive Pooling (AP): A two-way attention mechanism for discriminative model training. In the context of pair-wise ranking or classification with neural networks, AP enables the pooling layer to be aware of the current input pair, in a way that information from the two input items can directly influence the computation of each other's representations.

Along with the above representations of the paired inputs, AP jointly learns a similarity measure over projected segments (e.g. trigrams) of the pair, and subsequently, derives the corresponding attention vector for each input to guide the pooling.

The two-way attention mechanism is independent of the underlying representation learning and is applicable to both CNNs and RNNs.

2. Neural Networks for Answer Selection

Given a pair (q, a) consisting of a question q and a candidate answer a, both networks score the pair by first computing fixed-length independent continuous vector representations r^q and r^a , and then computing the cosine similarity between these two vectors.

The first layer in both QA-CNN and QA-biLSTM, transforms each input word w into a fixed-size real-valued word embedding $r^w \in R^d$.
Word embeddings (WEs) are encoded by column vectors in an embedding matrix $W^0 \in R^{d \times |V|}$
Word Embeddings used were 300-dimensional GloVe.

V = fixed-sized vocabulary
d = dimension of the word embeddings
Given input pair (q, a),
    question **q** contains M tokens
    candidate answer **a** contains L tokens
the output of the first layer consists of two sequences of word embeddings $\mathbf{q}^{\mathbf{emb}} = \{r^{w1}, ..., r^{wM}\}$ and $\mathbf{a}^{\mathbf{emb}} = \{r^{w1}, ..., r^{wL}\}$

2.1 Convolution

For $\mathbf{q}^{\mathbf{emb}} = \{r^{w1}, ..., r^{wM}\}$, we define:
    matrix $Z^q = [z_1, ..., z_M]$ as a matrix where each column contains a vector $z_m \in R^{dk}$
    that is the concatenation of a sequence of k word embeddings centralized in the m-th word of the question.

The output of the convolution with c filters over the question q is computed as follows:

$Q = W^1 Z^q + b^1$

Where each column m in $Q \in R^{cxM}$ contains features extracted in a context window around the m-th word of q.

Learn → W^1 , b^1

Hyperparams → number of convolutional filters c,
and the size of the word context window k

In a similar manner, and using the same NN parameters W^1 & b^1 , we compute $A \in R^{cxL}$, the output of the convolution over the candidate answer a.

$A = W^1 Z^a + b^1$

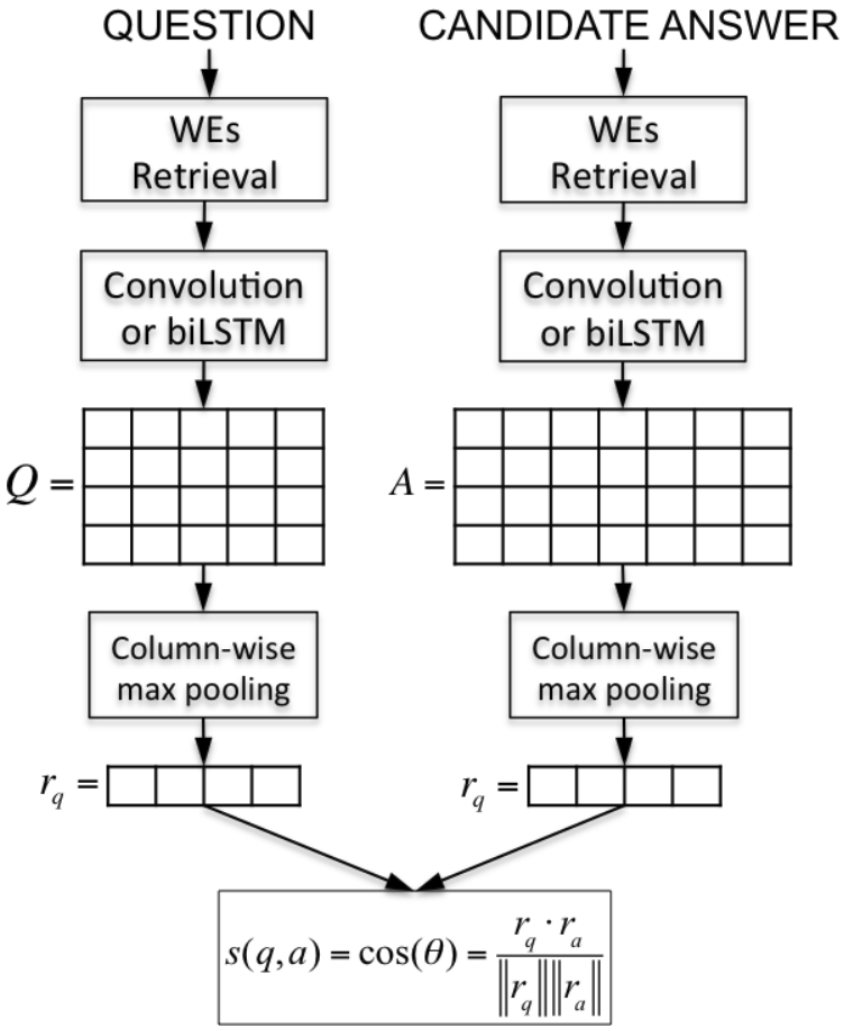


Fig 3. Joint illustration of QA-CNN and QA-biLSTM.

2.2 LSTM

Given $\mathbf{q}^{emb} = \{r^{w1}, ..., r^{wM}\}$, the hidden vector h(t) (with size H) at the time step t is updated as follows:

In the LSTM architecture, there are three gates (input i, forget f and output o), and a cell memory vector c. σ is the sigmoid function. The input gate can determine how incoming vectors r^{wt} alter the state of the

memory cell. The output gate can allow the memory cell to have an effect on the outputs. Finally, the forget gate allows the cell to remember or forget its previous state. $W \in R^{H \times d}$, $U \in R^{H \times H}$ and $b \in R^{H \times 1}$ are the network parameters.

$$
\begin{aligned}
i_t &= \sigma(\mathbf{W}_i r^{w_t} + \mathbf{U}_i \mathbf{h}(t-1) + \mathbf{b}_i) \\
f_t &= \sigma(\mathbf{W}_f r^{w_t} + \mathbf{U}_f \mathbf{h}(t-1) + \mathbf{b}_f) \\
o_t &= \sigma(\mathbf{W}_o r^{w_t} + \mathbf{U}_o \mathbf{h}(t-1) + \mathbf{b}_o) \\
\tilde{C}_t &= \tanh(\mathbf{W}_m r^{w_t} + \mathbf{U}_m \mathbf{h}(t-1) + \mathbf{b}_m) \\
C_t &= i_t * \tilde{C}_t + f_t * C_{t-1} \\
\mathbf{h}_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Single direction LSTMs suffer from the weakness of not utilizing the contextual information from the future tokens. Bidirectional LSTM utilizes both the previous and future context by processing the sequence in two directions and generating two independent sequences of LSTM output vectors. One processes the input sequence in the forward direction, while the other processes the input in the reverse direction.

The output at each time step is the concatenation of the two output vectors from both directions. We define $c = 2 \times H$ for the notation consistency with the previous subsection. After computing the hidden state h_t for each time step t, we generate the matrices $Q \in R^{c \times M}$ and $A \in R^{c \times L}$, where the j-th column in Q (A) corresponds to j-th hidden state h_j that is computed by the biLSTM when processing *q(a)*. The same network parameters are used to process both questions and candidate answers.

2.3 Scoring and Training Procedure

Given the matrices Q and A, we compute the vector representations $r^q \in R^c$ and $r^a \in R^c$ by applying a column-wise max-pooling over Q and A, followed by a non-linearity.
Formally, the j-th elements of the vectors r^q and r^a are computed as follows:

$$[r^q]_j = \tanh\left(\max_{1<m<M} [Q_{j,m}]\right)$$

$$[r^a]_j = \tanh\left(\max_{1<l<L} [A_{j,l}]\right)$$

The last layer in QA-CNN and QA-biLSTM scores the input pair (q,a) by computing the cosine similarity between the two representations:

$$s(q, a) = r^q . r^a / ||r^q|| \; ||r^a||$$

Both networks are trained by minimizing a pairwise ranking loss function over the training set D. The input in each round is two pairs (q, a^+) and (q, a^-), where a^+ is a ground truth answer for q, and a^- is an incorrect answer.

Define the training objective as a hinge loss:

$$L = \max\{0, m - s_\theta(q, a^+) + s_\theta(q, a^-)\}$$

where m is constant margin, $s_\theta(q, a^+)$ and $s_\theta(q, a^-)$ are scores generated by the network with parameter set θ. During training, for each question we randomly sample 50 negative answers from the entire answer set, but only use the one with the highest score to update the model.
We use stochastic gradient descent (SGD) to minimize the loss function with respect to θ.

3. Attentive Pooling Networks for Answer Selection

Attentive pooling is an approach that enables the pooling layer to be aware of the current input pair, in a way that information from the question q can directly influence the computation of the answer representation r^a , and vice versa. The main idea consists of learning a similarity measure over the projected segments in the input pairs, and uses the similarity scores between the segments to compute attention vectors. When AP is applied to CNN, which we call AP-CNN, the network learns the similarity measure over the convolved input sequences. When AP is applied to biLSTM, which we call AP-biLSTM, the network learns the similarity measure over the hidden states produced by the biLSTM when processing the two input sequences. We use a similarity measure that has a bilinear form but followed by a non-linearity.

Application of AP over the output of the convolution or the biLSTM to construct the representations r^q & r^a:
Consider the input pair (q, a) where the question has size M and the answer has size L.
After we compute the matrices $Q \in R^{c \times M}$ and $A \in R^{c \times L}$, either by convolution or biLSTM,

we compute a new matrix $G \in R^{M \times L}$ as follows:

$$G = \tanh(Q^T U A)$$

where $U \in R^{c \times c}$ is a matrix of parameters to be learned by the NN.

When the convolution is used to compute Q and A, the matrix G contains the scores of a soft alignment between the convolved k-size context windows of q and a.

When the biLSTM is used to compute Q and A, the matrix G contains the scores of a soft alignment between the hidden vectors of each token in q and a.

Next, we apply column-wise and row-wise max-poolings over G to generate the vectors $g^q \in R^M$ and $g^a \in R^L$, respectively. Formally, the j-th elements of the vectors g^q and g^a are computed as follows:

$$[g^q]_j = \max_{1<m<M} [G_{j,m}]$$

$$[g^a]_j = \max_{1<l<L} [G_{l,j}]$$

We can interpret each element j of the vector g^a as an importance score for the context around the j-th word in the candidate answer a with regard to the question q. Likewise, each element j of the vector g^q can be interpreted as the importance score for the context around the j-th word in the question q with regard to the candidate answer a.

Next, we apply the softmax function to the vectors g^q and g^a to create attention vectors σ^q and σ^a . For instance, the j-th element of the vector σ^q is computed as follows:

$$[\sigma^q]_j = \frac{e^{[g^q]_j}}{\sum_{1<l<M} e^{[g^q]_l}}$$

Finally, the representations r^q and r^a are computed as the dot product between the attention vectors σ^q and σ^a and the output of the convolution (or biLSTM) over q and a, respectively:

$r^q = Q\sigma^q$ & $r^a = A\sigma^a$

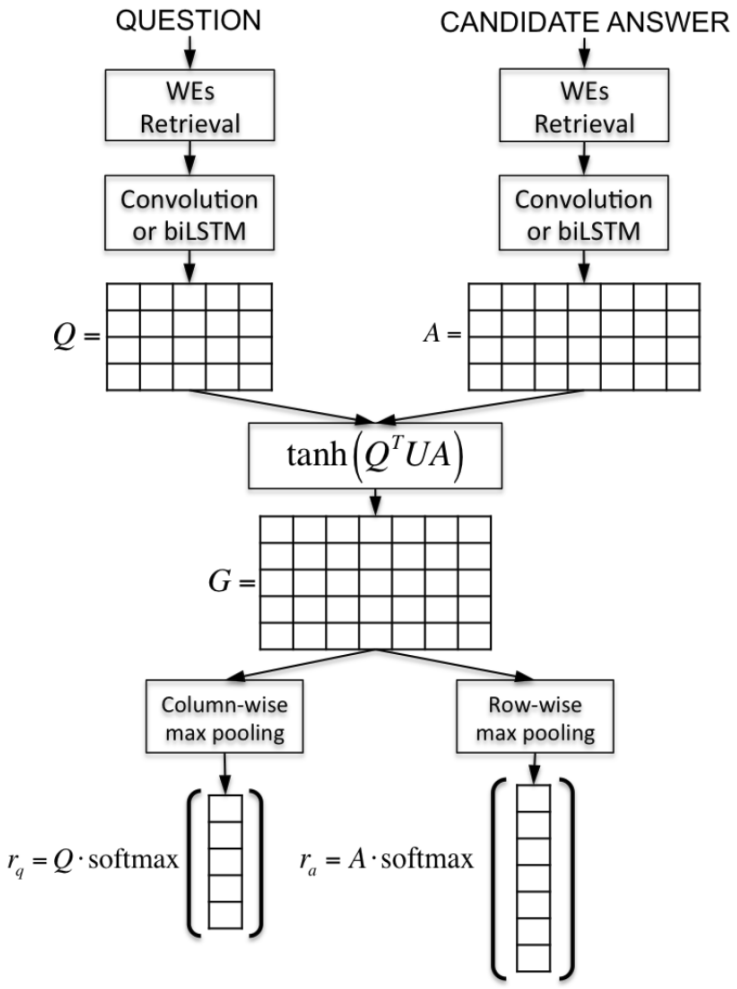


Fig 4. Attentive Pooling Networks for Answer Selection.

Like in QA-CNN and QA-biLSTM, the final score is also computed using the cosine similarity between r^q and r^a . We use SGD to train AP-CNN and AP-biLSTM by minimizing the same pairwise loss function used in QA-CNN and QA-biLSTM.

4. Experimental Setup

We wrote code written in Pytorch 1.8 and trained each model for 20 Epochs with early stopping.

4.1 Hyperparameters

Hyperparameter	AP-CNN	QA-CNN	AP-BiLSTM	QA-BiLSTM
WORD EMB. SIZE	300	300	300	300
CONV. FILTERS / HID. VEC. SIZE	400	400	150X2	150X2
CONTEXT WINDOW SIZE	3	2	1	1
MINIBATCH SIZE	20	20	20	20
INIT. LEARNING RATE	1.1	0.05	1.1	1.1

4.2 Learning Rate

We used a learning rate schedule that decreases the learning rate λ according to the training epoch t. Following [2], we set the learning rate for epoch t, λ_t , using the equation:

$$\lambda_t = \lambda / t$$

5. Results

Model	MAP	MRR
QA- BiLSTM	0.6287	0.6421
AP- BiLSTM	0.6482	0.6571
Improvement	*0.0195*	*0.0150*
QA-CNN	0.6420	0.6524
AP-CNN	**0.6593**	**0.6671**
Improvement	*0.0173*	*0.0147*

These results demonstrate the improvement of 1.5-2.0% in both CNN and BiLSTM models with Attention Pooling mechanism.

Baseline+: Encoder Finetuning

1. Architecture for encoder-based approach

A transformer encoder model with no additional head outputs a token embedding corresponding to each of the input tokens. In the paradigm of transformer finetuning, the task of answer selection was framed as a binary sentence pair classification task. For classification, we added two linear layers on top of the encoder output. The input to the first layer was a mean-pooled embedding of all the token embeddings output by the encoder. The attention mask was used to eliminate the padded token embeddings from the computation. Following the computation, we have a single 768-dimensional representation for the question-sentence pair.

The first layer provides a projection from the 768 dimensions to 512, having dropout connections of 25%. LeakyReLU activations were applied to the output of this layer with a negative slope of 0.2. The output of this layer is fed to the final linear layer for classification, which outputs 2-way classification scores. A dropout of 15% was applied to this layer. While evaluation, sigmoid activation was used for determining the score given to the question-sentence pair.

2. Experimental setup

The specifications of the encoder models in experimentation and the hyperparameter choices are given in the table below.

Dataset	WikiQA
Encoder models used	BERT[5], RoBERTa[6]
Model size	base (for both the models)
Loss	Binary Cross Entropy with Logits This implementation of the cross-entropy loss gives a numerically stable implementation of applying cross-entropy over sigmoid activation in a unified computation.
Weights for binary classes	The dataset was unbalanced for the classification task, with the positive samples being 1/8th of the total. Hence, they were upweighted by a factor of 8 while computing the loss.
Batch size	16
Learning rate	5e-6
Optimizer	Adam [betas=(0.9, 0.999), epsilon=1e-7]
No. of epochs	10
Strategy for selecting the best checkpoint	MRR on dev split

The training was carried out over a single GeForce GTX 1080 Ti GPU.

3. Results

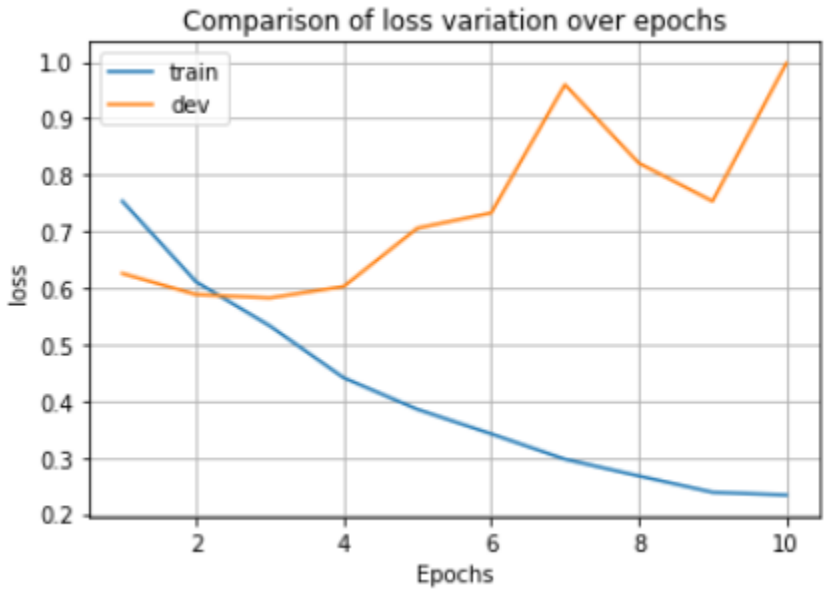
The evaluation was performed using the classification accuracy score in addition to MAP and MRR as planned.

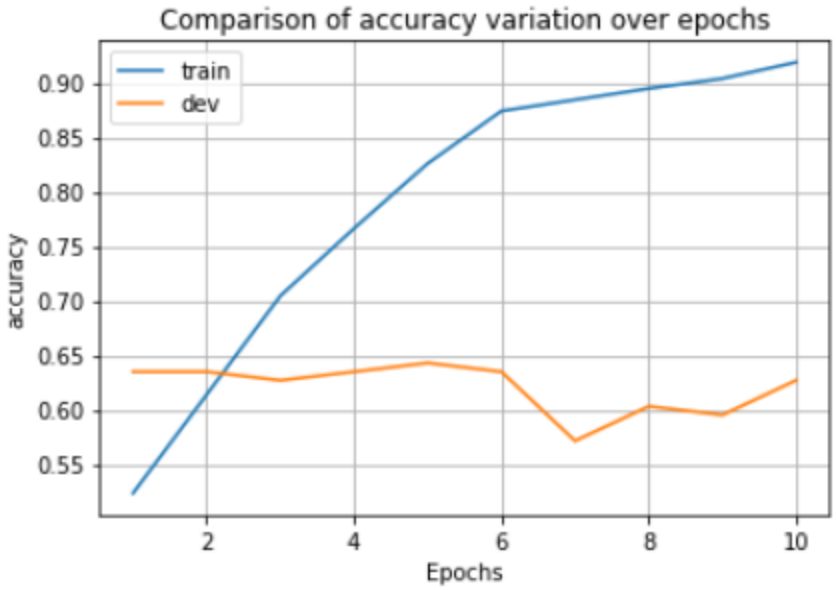
3.1 BERT-base[5]

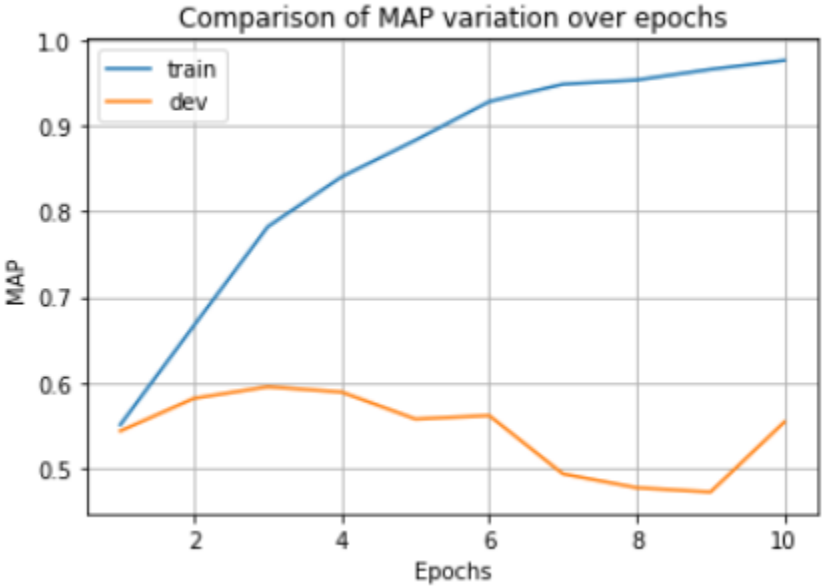
For BERT-base, the best checkpoint was obtained at the 5th epoch based on the MRR performance. The resultant metrics for this checkpoint are shown in the table below.

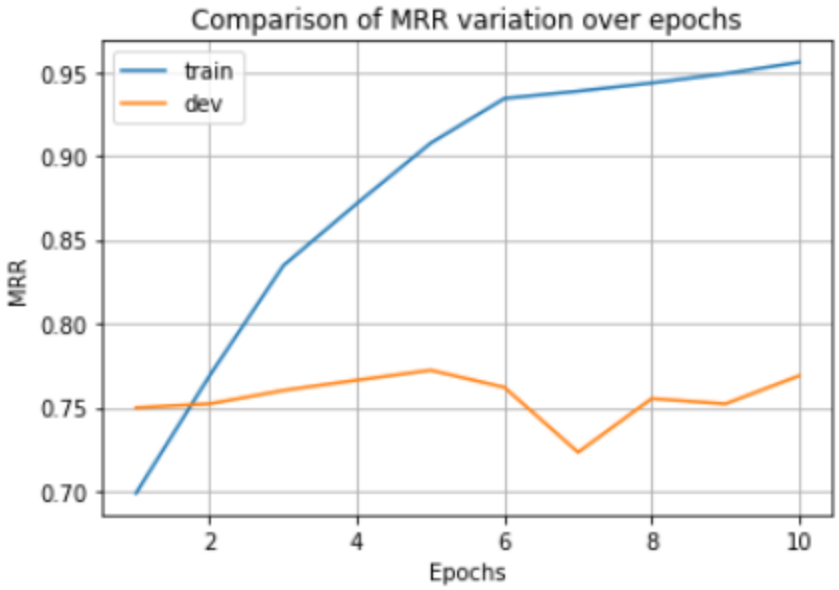
Split	Accuracy	MAP	MRR
train	76.63	84.03	90.76
dev	64.28	55.75	77.21
test	58.02	52.63	74.06

The variation in the cross-entropy loss and calculated metrics over epochs for train and dev splits is shown in the figures below.




Comparison of accuracy variation over epochs
train
dev
accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
2
4
6
8
10
Epochs


Comparison of MAP variation over epochs
train
dev
MAP
1.0
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10
Epochs

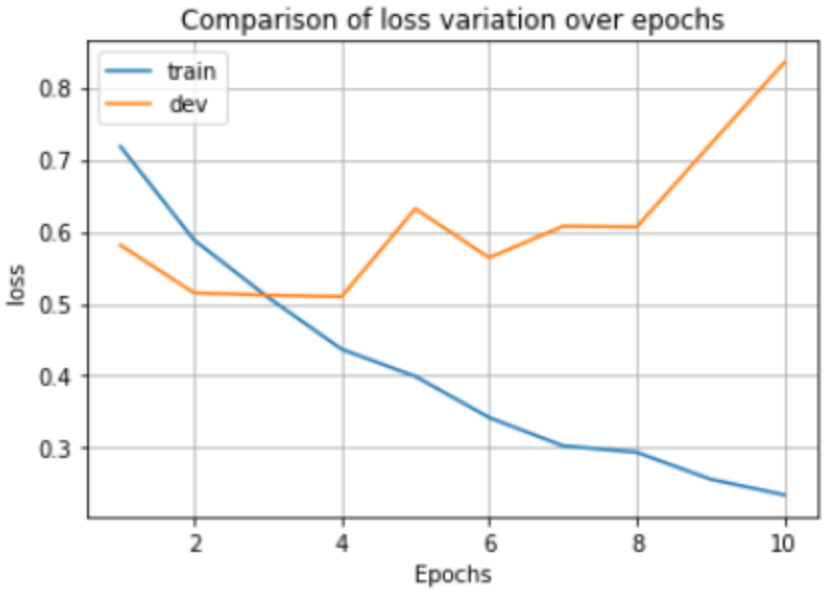


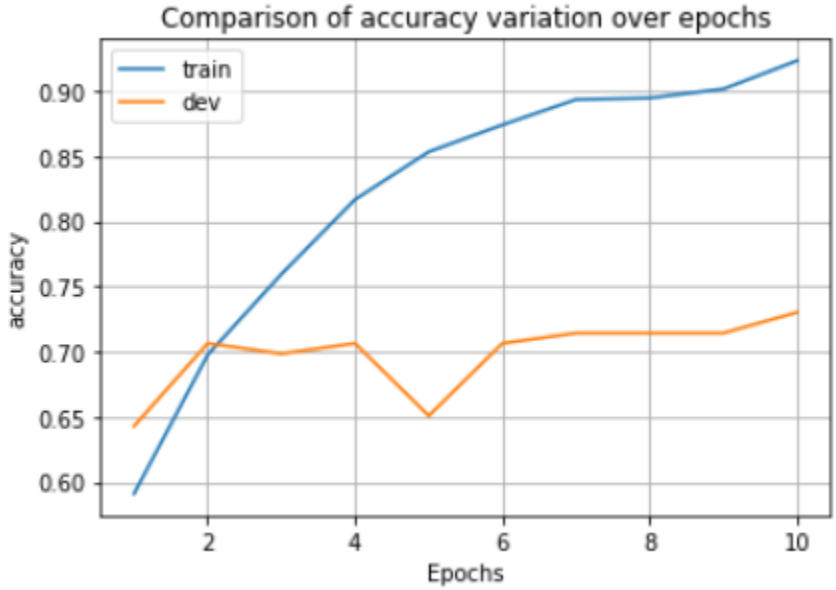
3.2 RoBERTa-base[6]

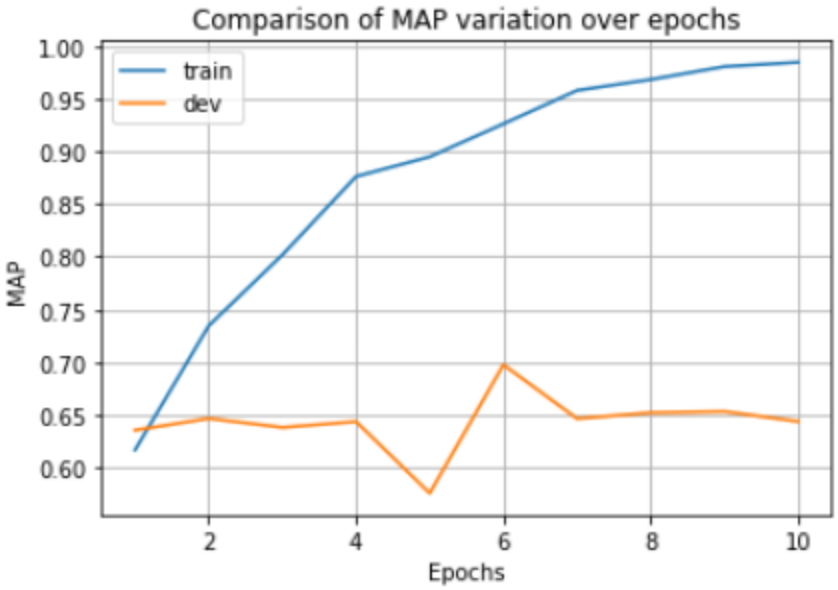
Based on MRR, the parameters of the model after the 10th epoch were obtained as the best checkpoint. The resultant metrics for this checkpoint are shown in the table below.

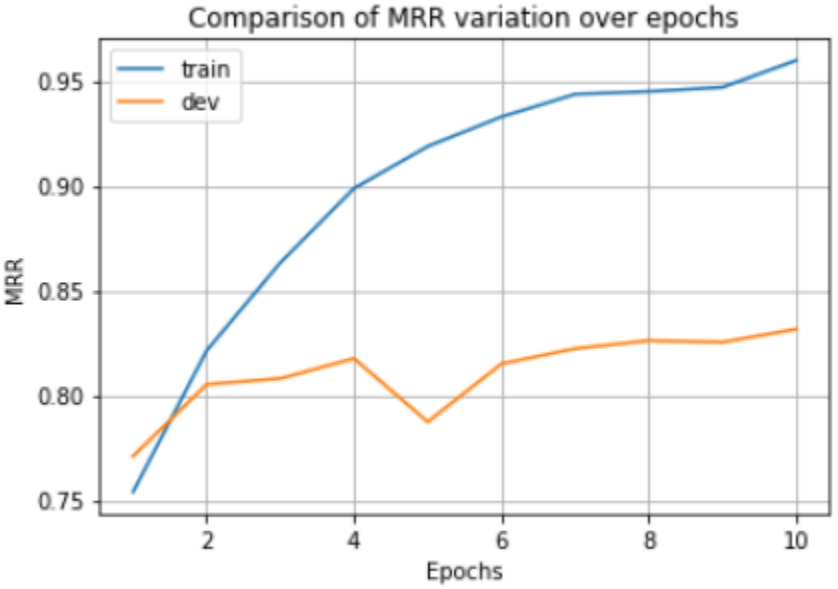
Split	Accuracy	MAP	MRR
train	92.32	98.44	95.98
dev	73.02	64.40	83.15
test	72.02	66.29	83.08

The variation in the cross-entropy loss and calculated metrics over epochs for train and dev splits is shown in the figures below.


Comparison of loss variation over epochs
train
dev
loss
Epochs
0.8
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10


Comparison of accuracy variation over epochs
train
dev
accuracy
Epochs
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2
4
6
8
10


Comparison of MAP variation over epochs
train
dev
MAP
Epochs
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2
4
6
8
10


Comparison of MRR variation over epochs
train
dev
MRR
Epochs
0.95
0.90
0.85
0.80
0.75
2
4
6
8
10

Baseline++: TANDA[4]

1. Idea Behind TANDA

While pretrained transformer-based models give a good performance on finetuning on the domain-specific data for answer selection, the performance obtained is lesser than optimal with several drawbacks owing to the usually low size of the individual datasets available for answer selection task. The WikiQA dataset at hand itself has 2,351 question-sentence pairs (after filtering for answer selection), which is much less considering the no. of parameters in a BERT-based architecture. This can easily run in overfitting on the data even with a few steps of training. The authors of TANDA point out other drawbacks present including instability in training, an on-off effect where the model tends to predict the same label for all the examples. The model performance was also observed to be exhibiting a large variance over different finetuning attempts.

In order to mitigate these issues observed, an additional ‘transfer’ step was proposed before ‘adapting’ of finetuning the model on the task-specific dataset, thus dividing the finetuning process into two stages. Since collecting large domain-specific data for different domain-specific answer selection tasks would be difficult, the availability of a single, large dataset for answer selection to transfer the capability of a pretrained encoder architecture to the task of answer selection would significantly improve the capability of the model before being fine-tuned on the target dataset. In essence, the pretrained transformer model, which has already acquired a good amount of knowledge of the general language dependencies would be first made better at solving the general task of answer selection. These additional capabilities will help the model easily adapt to the target domain even with a small amount of data being available since the model now has to learn only some domain-specific aspects rather than the broader task.

An intuitive flow diagram indicating the knowledge gained by the model through the processes of pretraining followed by the transfer and adapt steps introduced in TANDA is shown in the figure below.

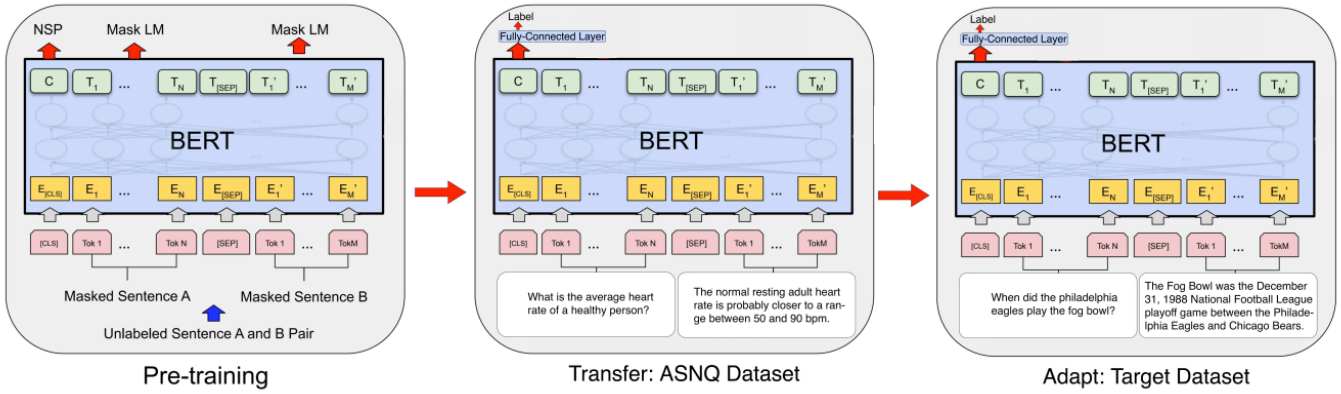


Figure 5. Model training stages with TANDA

2. Architecture for TANDA approach

Following the TANDA finetuning strategy would involve making use of a pretrained encoder as the base, training it separately for the transfer and adapt steps, making use of the refined parameters from the former step in the latter. For each of the steps, we added separate, different linear layers for the classification part of answer selection. The reasoning behind using different linear layers for both the steps was based on the general understanding of the type of representations learned in the layers of a model. While the initial few layers learn low-level, general representations, the last few layers are more specific, learning higher-level representations specific to the task.

For the transfer step, a single linear layer was applied to the output of the encoder architecture. The output of the encoder was computed by mean-pooling the token embeddings. The token embeddings were multiplied by the attention mask for eliminating the padded token embeddings from the computation. The output of the linear layer was a 2-way classification output. A dropout of 25% was added to the linear layer for robustness in training.

3. Experimental setup

The specifications of the hyperparameter choices and the model versions used for the transfer step are provided in the table below.

Dataset	ASNQ
Encoder models used	BERT, RoBERTa
Model size	base (for both the models)
Loss	Binary Cross Entropy with Logits This implementation of the cross-entropy loss gives a numerically stable implementation of applying cross-entropy over sigmoid activation in a unified computation.
Weights for binary classes	In the ASNQ dataset, the positive samples were found to be 1/16th of the total. Hence, they were upweighted by a factor of 16 while computing the loss.
Batch size	48
Learning rate	1e-6
Optimizer	Adam [betas=(0.9, 0.999), epsilon=1e-7]
No. of epochs	2
Eval steps	10000 After every 10000 steps, dev split was used for checking for the best checkpoint
Strategy for selecting the best checkpoint	BCE loss on dev split of ASNQ

The transfer process of finetuning was carried out over 3 GeForce GTX 2080 Ti GPUs. For the adapt process, the experimental setup remains the same as that followed for encoder finetuning before.

4. Results

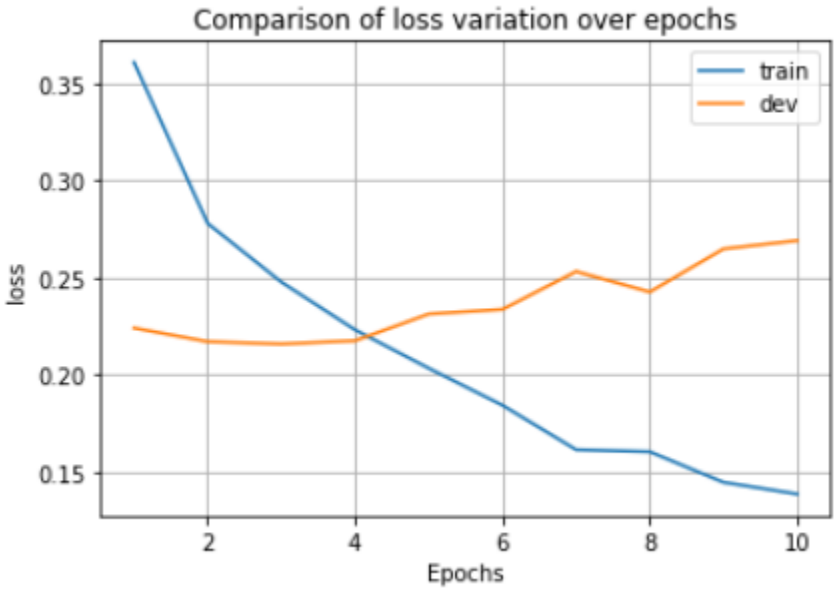
The saved encoder after completion of the transfer step was used as the starting point for the adapt step. After training for 10 epochs, the results of the evaluation performed are shown below.

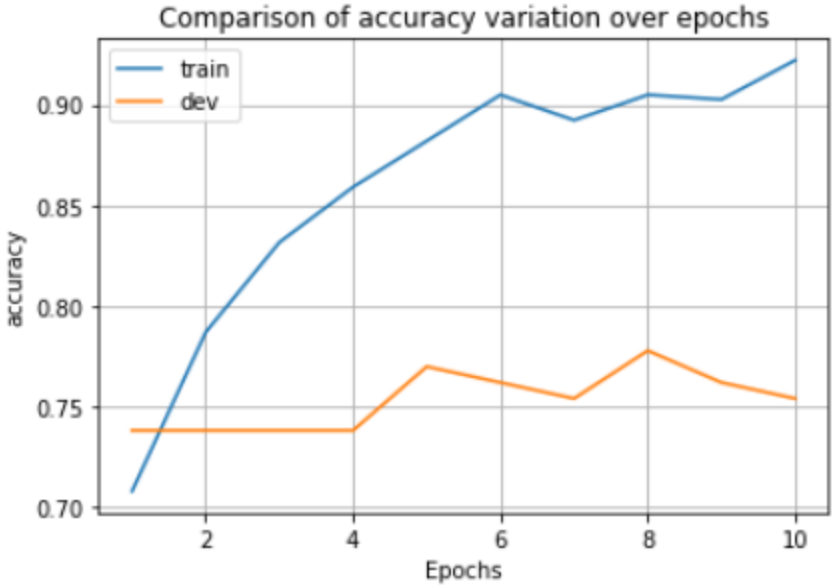
3.1 BERT-base

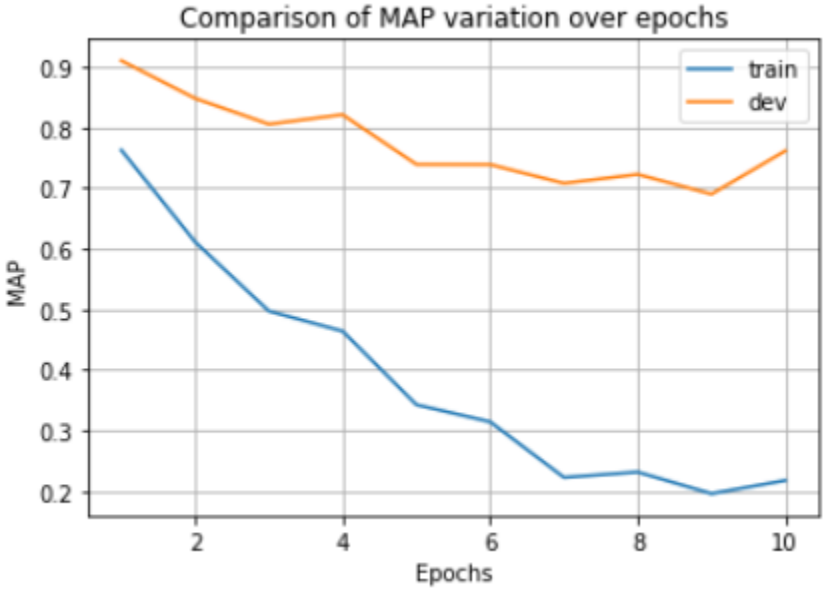
For BERT-base, the best checkpoint in the adapt stage was obtained at the 8th epoch based on the MRR performance on the dev split. The resultant metrics for this checkpoint are shown in the table below.

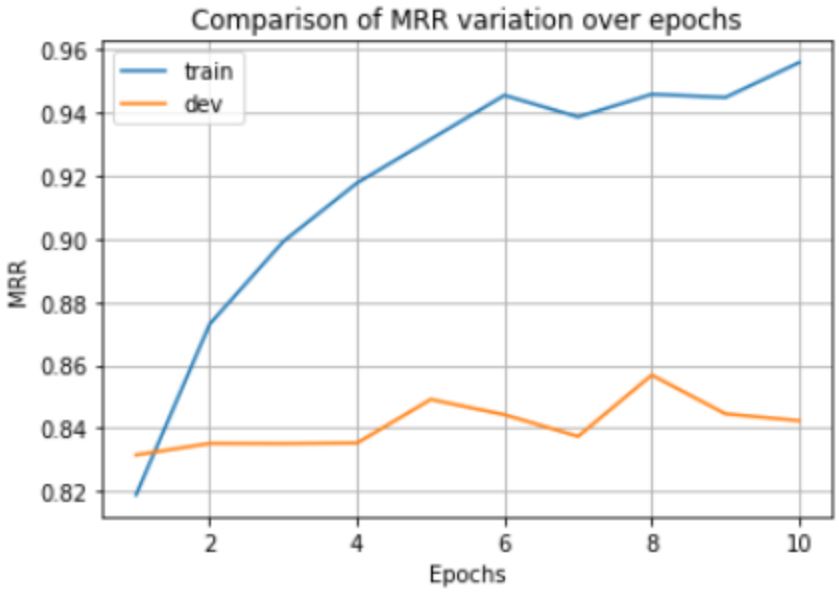
Split	Accuracy	MAP	MRR
train	90.49	23.11	94.59
dev	77.78	72.29	85.69
test	70.56	70.55	84.25

The variation in the cross-entropy loss and calculated metrics over the epochs for train and dev splits in the adapt stage is shown in the figures below.




Comparison of accuracy variation over epochs
train
dev
accuracy
0.90
0.85
0.80
0.75
0.70
2
4
6
8
10
Epochs


Comparison of MAP variation over epochs
train
dev
MAP
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
2
4
6
8
10
Epochs

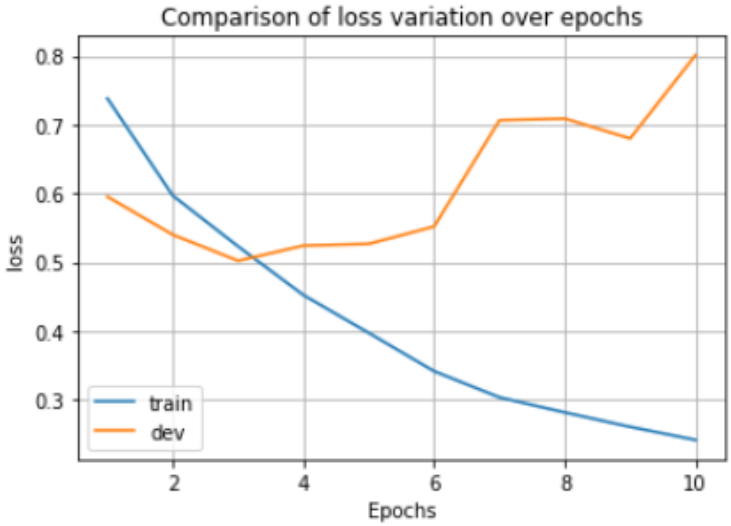


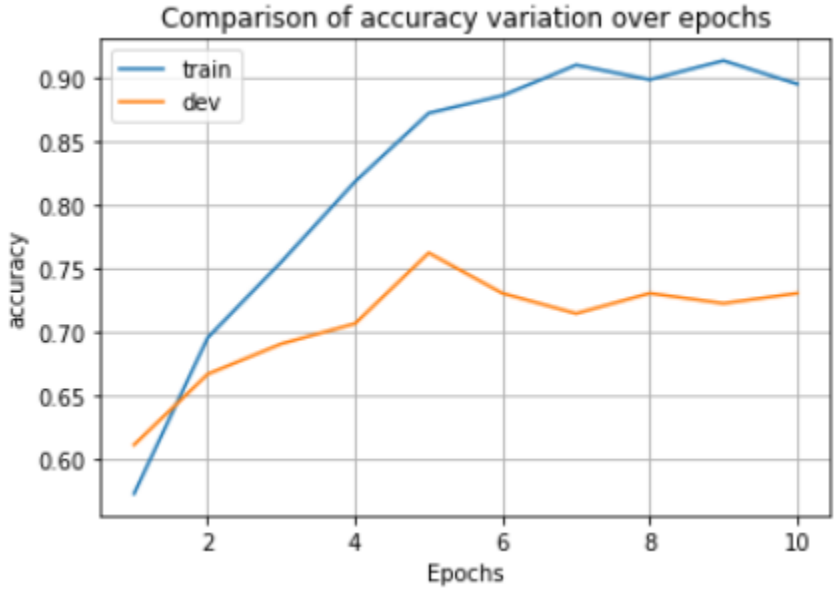
3.2 RoBERTa-base

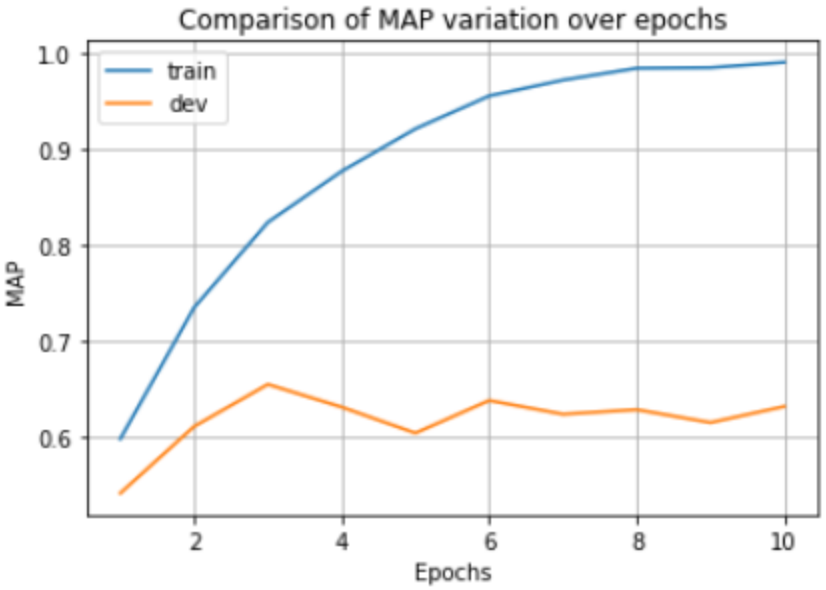
For RoBERTa-base, the best checkpoint in the adapt stage was obtained at the 5th epoch based on the MRR performance on the dev split. The resultant metrics for this checkpoint are shown in the table below.

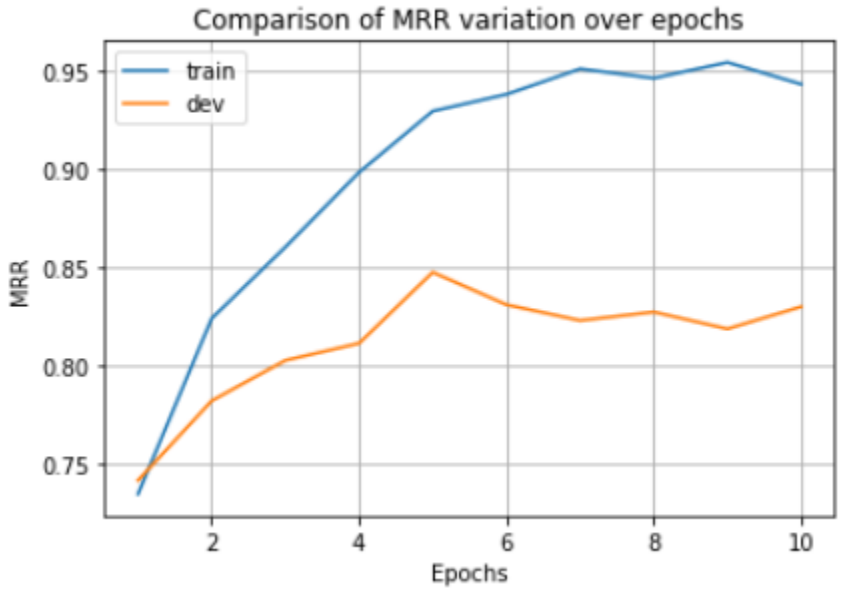
Split	Accuracy	MAP	MRR
train	87.17	92.06	98.99
dev	76.19	60.36	84.79
test	76.13	66.15	85.48

The variation in the cross-entropy loss and calculated metrics over the epochs for train and dev splits in the adapt stage is shown in the figures below.


Comparison of loss variation over epochs
loss
Epochs
train
dev
0.8
0.7
0.6
0.5
0.4
0.3
2
4
6
8
10


Comparison of accuracy variation over epochs
accuracy
Epochs
train
dev
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2
4
6
8
10


Comparison of MAP variation over epochs
train
dev
MAP
Epochs
1.0
0.9
0.8
0.7
0.6
2
4
6
8
10


Comparison of MRR variation over epochs
train
dev
MRR
Epochs
0.95
0.90
0.85
0.80
0.75
2
4
6
8
10

Analysis

Here, we present our noted observations and analysis we gained from all the experimentations we performed and the results we documented.

1. Attention Pooling
   Two models were tried out in attention pooling - LSTMs and CNNs, with an added ablation by removing the dual attention mechanism used. The dual attention helped improve the scores by a margin of up to 1.5% in MAP and MRR, however, the values attained by both the models were in the range of 60s, indicating a not-so-impressive performance.

2. Encoder Finetuning
   In this approach, a very simple strategy was followed, taking a pretrained encoder's weights and applying regular finetuning to them. Even in this simple setting, the performance improved by a margin of about at least 10 points in MRR. This indicates the large gains that can be obtained just from having a large no. of pretrained parameters in the model.

3. TANDA Finetuning
   On following the TANDA procedure in encoder finetuning, the results were pushed much further, especially in the case of BERT, by a margin of about 7-8 points. The gains that can be obtained by leveraging a large task-specific dataset to refine the pretrained weights of the model for the task were obvious.

4. BERT v/s RoBERTa performance
   a) The performance of RoBERTa was superior to BERT in the basic finetuning approach, while it was competitive to BERT with TANDA. RoBERTa, which essentially is a more robust version of BERT by use of additional pretraining data and more efficient, intensive pretraining, has proven its superiority over basic BERT in many tasks. The same was observed in our case as well.
   b) Comparing the performance curves over epochs, it appears that those of RoBERTa showed an upward trend towards the 10th epoch of the training, while those of BERT were flat, or on a downwards trend. This indicates the possibility of getting better results with RoBERTa with further finetuning.

5. Observations in a better training process due to TANDA
   We plotted the losses and performance on the accuracy, MAP & MRR metrics as they varied over 10 epochs of finetuning on the target data. As observed in the TANDA paper as well, there was a spiking effect in the variation of the values over the epochs in regular finetuning. With TANDA, the spiking effect was smoothed out to some extent as compared to the previous case. Although it was not completely mitigated, we expect it to get much better with an increase in the number of epochs in the transfer stage. We had limited the no. of epochs to 2 due to computational time limitations.

Conclusion

Through this project, we dived into the area of question answering by exploring the answer selection problem by making use of the WikiQA dataset. We studied the dataset and various implementations of algorithms in the pre-transformer and post-transformer era, shortlisted 3 approaches for implementation, and successfully implemented them. The results we achieved were close to that achieved by the authors of the papers we implemented. The performance can be taken further to match with the benchmarks by training for more epochs or additional hyperparameter tuning, however, we restricted ourselves based on the compute available while adhering to the project timeline.

The major insight out of the project was the effectiveness of large task-specific data for obtaining an improvement in low-resource domains. As an immediate future scope, we believe that the TANDA approach can be tried out with better encoder architectures instead of restricting to BERT or RoBERTa. For instance, SentenceBERT [7] architecture provides good quality sentence representations in a Siamese network. The advantage of this architecture can be leveraged in the TANDA strategy for furthering the benchmark on WikiQA.

References

1. C Santos, M Tan, B Xiang, B Zhou, “Attentive Pooling Networks” arXiv preprint arXiv:1602.03609, 2016

2. dos Santos, Cicero Nogueira and Zadrozny, Bianca. Learning character-level representations for part-of-speech tagging. In Proceedings of the 31st International Conference on Machine Learning, JMLR: W&CP volume 32, Beijing, China, 2014.

3. Y. Yi, Y. Wen-tau, M. Christopher, “WikiQA: A Challenge Dataset for Open-Domain Question Answering”, EMNLP 2015

4. S. Garg, T. Vu, A. Moschitti, “TANDA: Transfer and Adapt Pre-Trained Transformer Models for Answer Sentence Selection”, AAAI 2020

5. J. Devlin, M. Chang, K. Lee, K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, ArXiv abs/1810.04805 (2019): n. Pag.

6. Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, “RoBERTa: A Robustly Optimized BERT Pretraining Approach”, ArXiv abs/1907.11692 (2019): n. Pag.

7. N. Reimers, I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”, EMNLP 2019