Assignment 6: Adapting Transformers Language Models for Your Problems

1 Concepts, intuitions, and big picture

1. How does the BERT model expect a pair of sentences to be processed?

   A. Tokens-of-sentence-1 [SEP] Tokens-of-sentence-2

   B. [CLS] Tokens-of-sentence-1 Tokens-of-sentence-2

   C. [CLS] Tokens-of-sentence-1 [SEP] Tokens-of-sentence-2

   Answer: C

2. When should you train a new tokenizer?

   A. When your dataset is similar to that used by an existing pretrained model, and you want to pretrain a new model

   B. When your dataset is similar to that used by an existing pretrained model, and you want to fine-tune a new model using this pretrained model

   C. When your dataset is different from the one used by an existing pretrained model, and you want to pretrain a new model

   Answer: C

3. Select the sentences that apply to the BPE model of tokenization.

   A. BPE is a subword tokenization algorithm that starts with a small vocabulary and learns merge rules.

   B. BPE is a subword tokenization algorithm that starts with a big vocabulary and progressively removes tokens from it.

   C. BPE tokenizers learn merge rules by merging the pair of tokens that is the most frequent.

   D. A BPE tokenizer learns a merge rule by merging the pair of tokens that maximizes a score that privileges frequent pairs with less frequent individual parts.

   E. BPE tokenizes words into subwords by splitting them into characters and then applying the merge rules.

   F. BPE tokenizes words into subwords by finding the longest subword starting from the beginning that is in the vocabulary, then repeating the process for the rest of

the text.

Answer: A, C, E

4. What are the labels in a masked language modeling problem?

   A. Some of the tokens in the input sentence are randomly masked and the labels are the original input tokens.

   B. Some of the tokens in the input sentence are randomly masked and the labels are the original input tokens, shifted to the left.

   C. Some of the tokens in the input sentence are randomly masked, and the label is whether the sentence is positive or negative.

   D. Some of the tokens in the two input sentences are randomly masked, and the label is whether the two sentences are similar or not.

   Answer: A

5. When should you pretrain a new model?

   A. When there is no pretrained model available for your specific language

   B. When you have lots of data available, even if there is a pretrained model that could work on it

   C. When you have concerns about the bias of the pretrained model you are using

   Answer: A, C

6. Is there something wrong with the following code?

```
from transformers import AutoTokenizer , AutoModel

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
model = AutoModel.from_pretrained("gpt2")

encoded = tokenizer("Hey!" , return_ tensors="pt")
result = model(**encoded)
```

   A. No, it seems correct.

   B. The tokenizer and model should always be from the same checkpoint.

   C. It's good practice to pad and truncate with the tokenizer as every input is a batch.

   Answer: C

2 Getting Your Attention with Self-Attention

Recall that the transformer architecture uses scaled dot-product attention to compute attention weights:

$$\alpha^{(t)} = \text{softmax}\left(\frac{q_t K^\top}{\sqrt{h}}\right) \in [0,1]^n$$

The resulting embedding in the output of attention at position t are:

$$\text{Attention}(Q,K,V)_t = \sum_{t'=1}^{n} \alpha_{t'}^{(t)} v_{t'} \in \mathbb{R}^{1\times h},$$

where $\alpha^{(t)} = [\alpha_0^{(t)}, \ldots, \alpha_n^{(t)}]$. The same idea can be stated in a matrix form,

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{h}}\right) V \in \mathbb{R}^{n\times h}$$

In the above equations:

- h is the hidden state dimension and n is the input sequence length;
- $X \in \mathbb{R}^{n\times h}$ is the input to the attention;
- $x_t \in \mathbb{R}^{1\times h}$ is the slice of X at position t, i.e. vector representation (embedding) of the input token at position t;
- $W_q, W_k, W_v \in \mathbb{R}^{h\times h}$ are the projection matrices to build query, key and value representations;
- $Q = XW_q \in \mathbb{R}^{n\times h}$, $K = XW_k \in \mathbb{R}^{n\times h}$, $V = XW_v \in \mathbb{R}^{n\times h}$ are the query, key and value representations;
- $q_t = x_t W_q \in \mathbb{R}^{1\times h}$ is the slice of Q at position t. Similarly, $k_t = x_t W_k \in \mathbb{R}^{1\times h}$ and $v_t = x_t W_v \in \mathbb{R}^{1\times h}$.

2.1 Forget 'Softmax', 'Argmax' is the New Boss in Town!

Recall from lectures that we can think about attention as being queryable softmax pooling. In this problem, we ask you to consider a hypothetical 'argmax' version of attention where it returns exactly the value corresponding to the key that is most similar to the query, where similarity is measured using the traditional inner-product. In other words, here use a version of Attention that instead of using softmax we use argmax to generate outputs from the attention layer. For example, $\text{softmax}([1,3,2])$ becomes $\text{argmax}([1,3,2]) = [0,1,0]$.

1. Perform argmax attention with the following keys and values:

$$\text{Keys} = \left\{ \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}, \begin{bmatrix} -1 \\ 1 \\ -2 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ -2 \end{bmatrix} \right\}, \quad \text{Values} = \left\{ \begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix}, \begin{bmatrix} 3 \\ -2 \\ 4 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} \right\}$$

using the following query:

$$q = \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

What would be the output of the attention layer for this query? Remember, to simplify calculations, use an argmax instead of softmax. Note about notation: each vector in 'Keys = {.}' and 'Values{.}' are key/value vectors corresponding to four, non-interchangeable positions (i.e., ordering of these vectors matter).

Answer:

The dot products of the query with the keys are:

- $q \cdot k_1 = (-2)(0) + (1)(1) + (1)(2) = 3$
- $q \cdot k_2 = (-2)(1) + (1)(1) + (1)(2) = 1$
- $q \cdot k_3 = (-2)(-1) + (1)(1) + (1)(-2) = 1$
- $q \cdot k_4 = (-2)(0) + (1)(-1) + (1)(-2) = -3$

The maximum dot product is 3, which corresponds to the first key. The output of the attention layer is the first value vector: $\begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix}$.

2. How does this design choice affect our ability to usefully train models involving attention? (**Hint**: think about how the gradients flow through the network in the backward pass. Can we learn to improve our queries or

keys during the training process?)

Answer: Using argmax instead of softmax in the attention mechanism effectively prevents the model from learning how to adjust the queries and keys to improve the attention distribution. This is because argamx essentially results in a one-hot attention weight vector, where the weights are non-differentiable with respect to the query and key vectors. As such, during backpropagation, the gradients of the loss function with respect to the query and key projections will mostly be zero. In effect, the attention mechanism will be untrainable.

2.2 Superposition of Information in Self-Attention

We will show that the attention mechanism is able to copy the information from its input, whenever needed.

1. **Copying mechanism in self-attention:** We'll first show that it is particularly simple for attention to "copy" a value vector to the output. Describe (in 1-2 sentences) what properties of the inputs to the attention operation would result in the output $\text{Attention}(Q, K, V)_t$ being approximately equal to v_j for some $j \in [0 \dots n]$.

   Hint: Show that there can exist $j \in [0 \dots n]$ such that $\alpha_j^{(t)} \gg \sum_{i \neq j} \alpha_i^{(t)}$, if certain

property holds for the query q_t, and the keys $\{k_0 \dots k_n\}$.

Answer: The output $\text{Attention}(Q, K, V)_t$ will be approximately equal to v_j if the query vector q_t has a significantly higher dot product with the key vector k_j than with any other key vector k_i (for $i \neq j$), such that the softmax function will output an attention weight $\alpha_j^{(t)}$ close to 1 for $i = j$ and close to 0 for for $i \neq j$.

2. **Extracting signals after averaging them:** Instead of focusing on just one vector v_j, the attention mechanism might want to incorporate information from multiple source vectors. Consider the case where we instead want to incorporate information from two vectors v_a and v_b, with corresponding key vectors k_a and k_b. How should we combine information from two value vectors v_a and v_b? A common way to combine value vectors is to average them: $\bar{v} = \frac{1}{2}(v_a + v_b)$. However, after such averaging it is not quite clear how to tease apart the original information in value vectors v_a and v_b. Unless ... some special properties hold!

   Suppose that although we don't know v_a or v_b, we do know that v_a lies in a subspace A formed by the m basis vectors $\{a_1, a_2, \dots, a_m\}$, while v_b lies in a subspace B formed by the p basis vectors $b_1, b_2, \dots, b_p$. This means that any v_a can be expressed as a linear combination of its basis vectors, as can v_b. All basis vectors have norm 1 and are orthogonal to each other. Additionally, suppose that the two subspaces are orthogonal; i.e., $a_j^\top b_k = 0$ for all j, k. Using the basis vectors $\{a_1, a_2, \dots, a_m\}$, construct a matrix M such that for arbitrary vectors $v_a \in A$ and $v_b \in B$, we can use M to extract v_a from the average vector $\bar{v}$. In other words, we want to construct M such that for any v_a, v_b, $M\bar{v} = v_a$. Show that $M\bar{v} = v_a$ holds for your M.

   Answer:

   Since $\{a_1, \dots, a_m\}$ is an orthonormal basis for subspace A, we can form a matrix $A_{basis} = [\, a_1 \quad a_2 \quad \dots \quad a_m \,]$, such that the projection matrix onto subspace A is $P_A = A_{basis} A_{basis}^\top$.

   The goal is to find a matrix M such that $M\bar{v} = v_a$, where $\bar{v} = \frac{1}{2}(v_a + v_b)$.

   With $M = 2P_A = 2A_{basis} A_{basis}^\top$, check if $M\bar{v} = v_a$:

$$M\bar{v} = 2A_{basis} A_{basis}^\top \left(\frac{1}{2}(v_a + v_b) \right)$$

$$M\bar{v} = A_{basis} A_{basis}^\top (v_a + v_b)$$

$$M\bar{v} = A_{basis} A_{basis}^\top v_a + A_{basis} A_{basis}^\top v_b$$

   Since $v_a \in A$ and $A_{basis} A_{basis}^\top$ is the projection onto A, $A_{basis} A_{basis}^\top v_a = v_a$.

   Since $v_b \in B$ and subspaces A and B are orthogonal, the projection of v_b onto A is the zero vector, i.e., $A_{basis} A_{basis}^\top v_b = 0$.

Hence,

$$M\bar{v} = v_a + 0 = v_a$$

Thus, the matrix $M = 2A_{basis}A_{basis}^\top$ satisfies the condition.

3. Averaging pairs of representations: As before, let v_a and v_b be two value vectors corresponding to key vectors k_a and k_b, respectively. Assume that (1) all key vectors are orthogonal, so $k_i^\top k_j = 0$ for all $i \neq j$; and (2) all key vectors have norm 1. Find an expression for a query vector q_t such that $\text{Attention}(Q, K, V)_t = \frac{1}{2}(v_a + v_b)$ and justify your answer.

   Answer:

   Consider the expression $q_t = k_a + k_b$.

   Due to the orthonormality of the key vectors, the dot products are $q_t k_a^\top = 1$, $q_t k_b^\top = 1$, and $q_t k_i^\top = 0$ for all other keys k_i where $i \neq a, b$.

   The softmax function will assign equal attention weights to v_a and v_b and zero weights to other value vectors when the scaled dot products for k_a and k_b are equal and much larger than others.

   With $q_t = k_a + k_b$, the dot products are equal for k_a and k_b and zero for others, resulting in the attention weights $\alpha_a^{(t)}$ and $\alpha_b^{(t)}$ approaching $\frac{1}{2}$ as the scaling factor $\frac{1}{\sqrt{h}}$ or the magnitude of $k_a + k_b$ increases, and $\alpha_i^{(t)}$ approaching 0 for $i \neq a, b$.

   Hence, $q_t = k_a + k_b$ is a suitable expression for the query vector.

Programming

3.1 Finetuning Basics

Figure for question 1

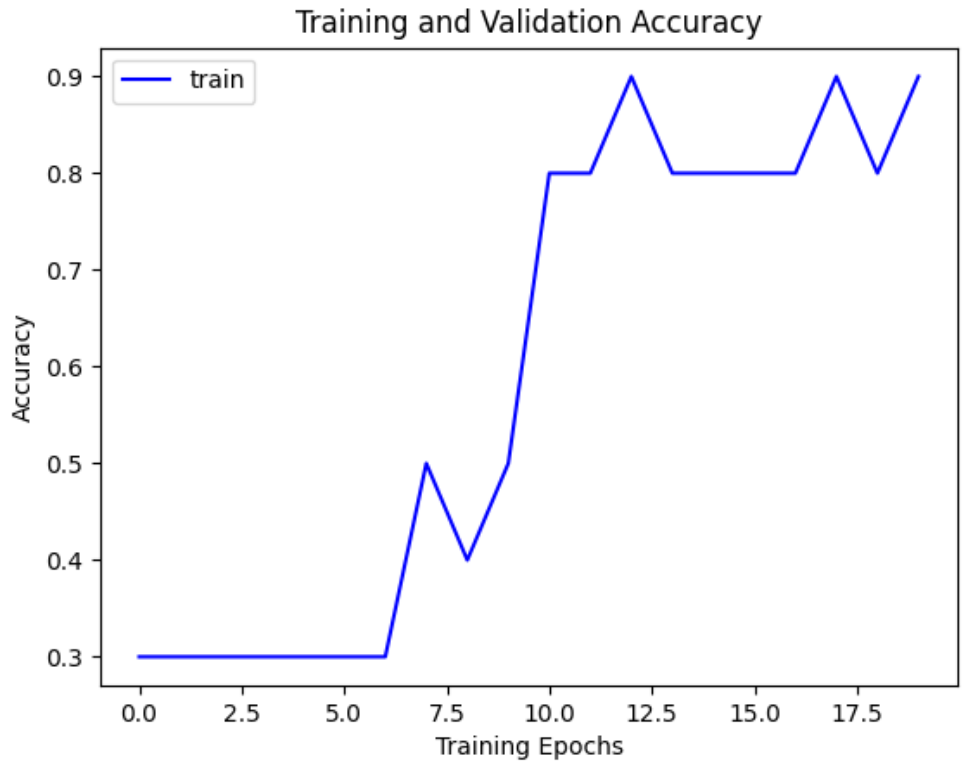
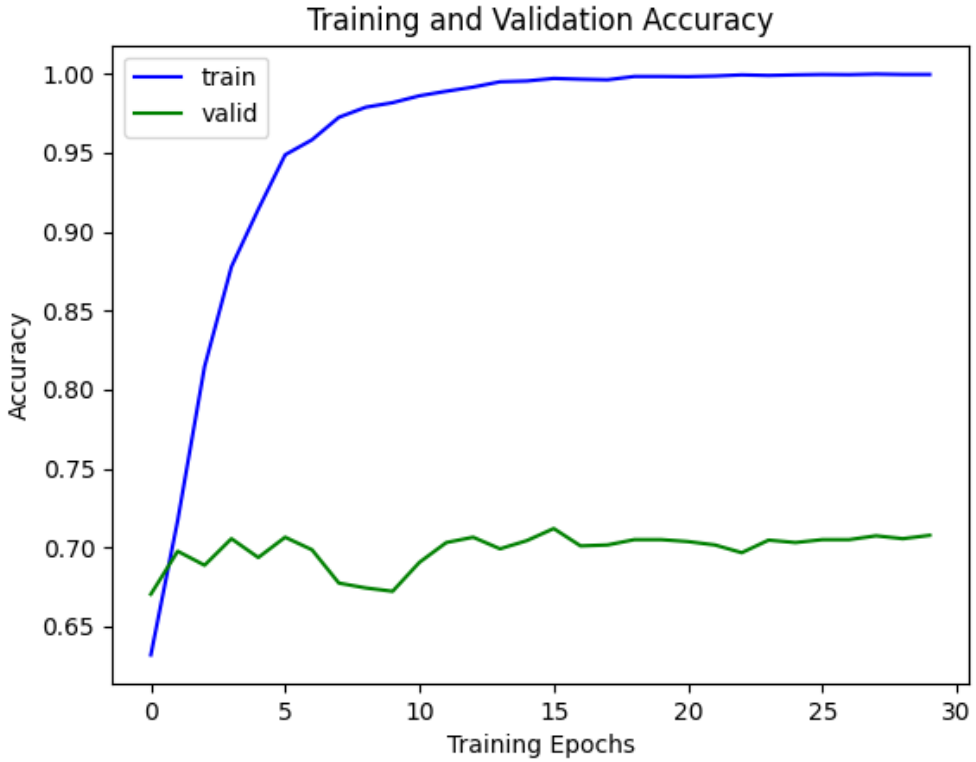
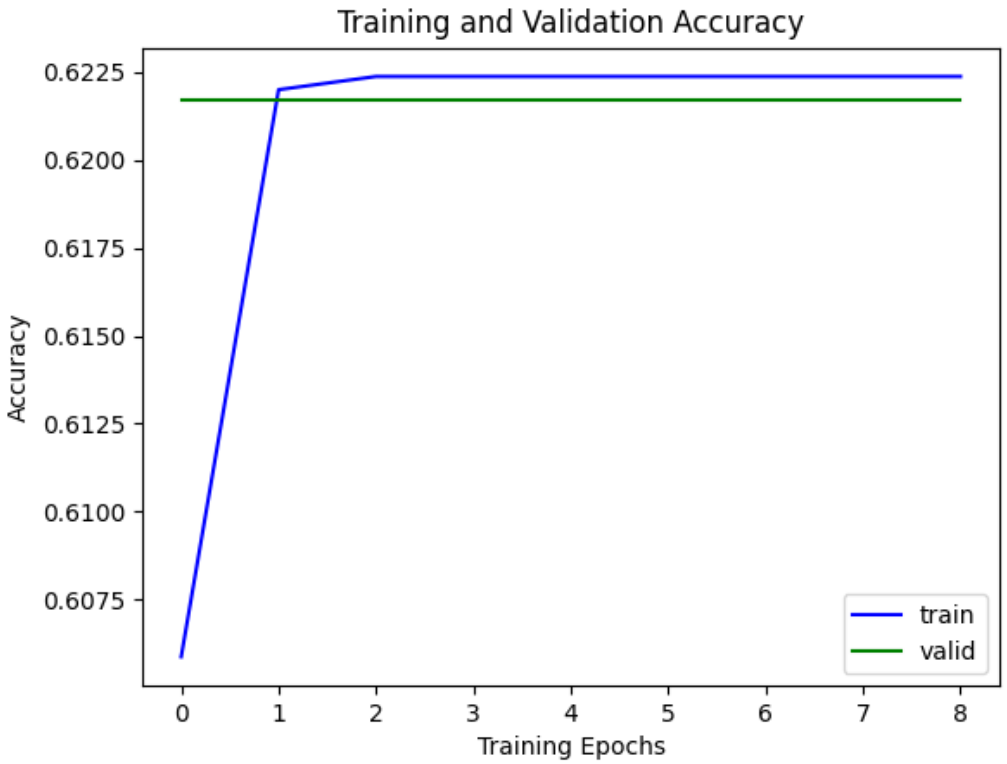


Figure for question 2



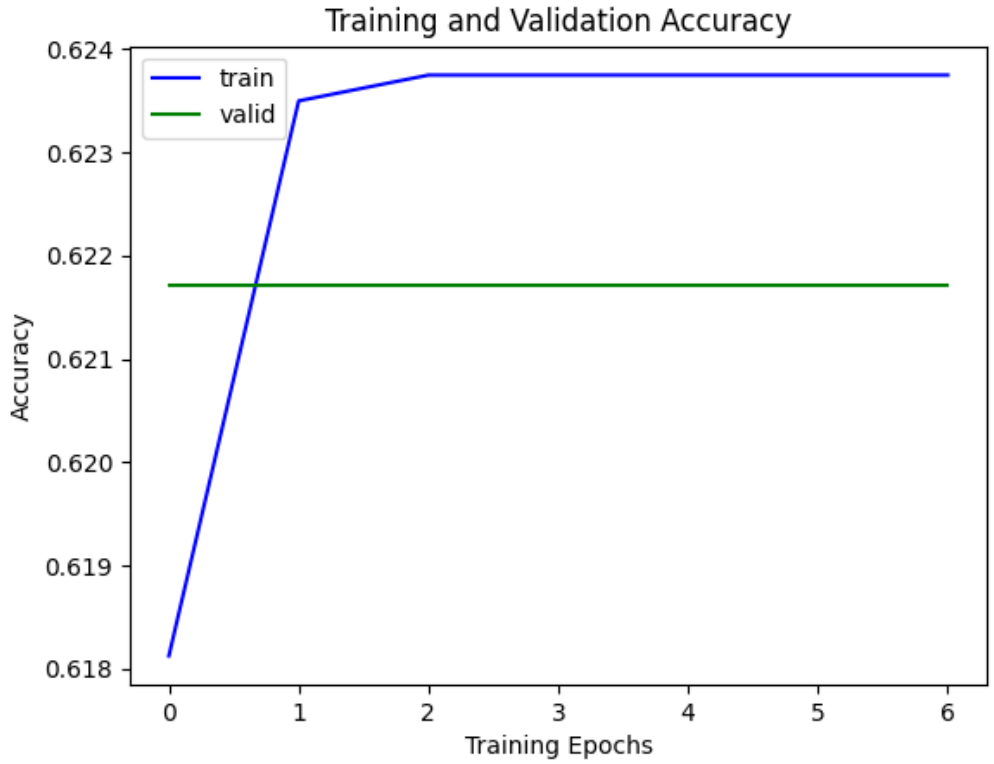
Figures for question 3

Test 1 Accuracy (lr=1e-3, num_epochs=9):



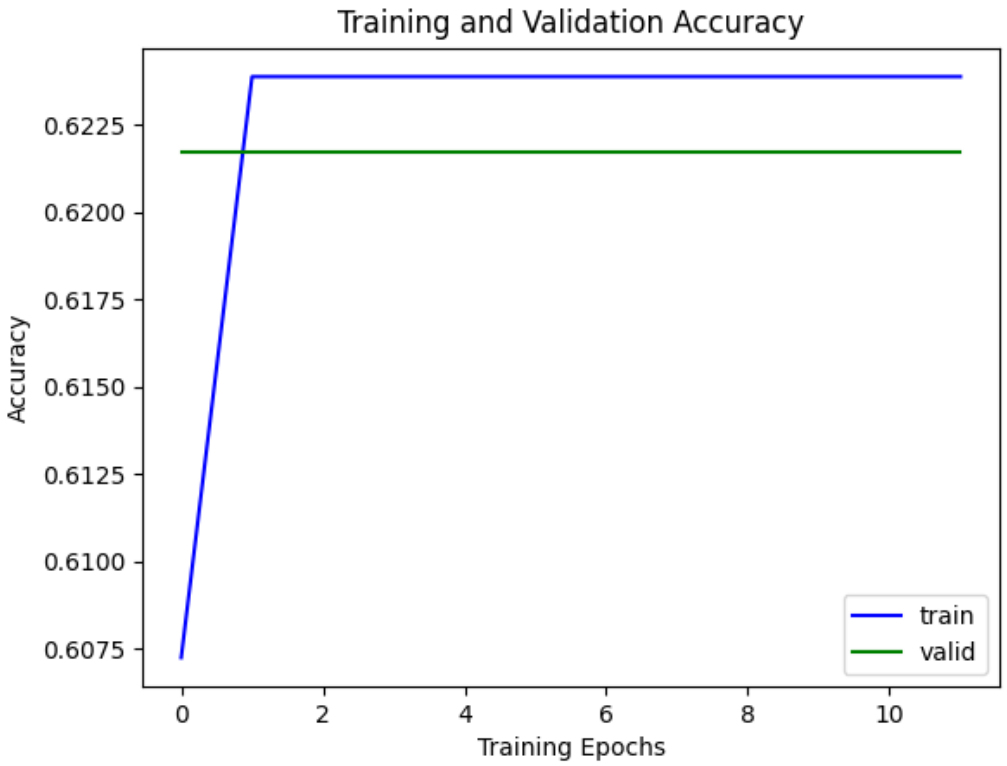
- Average DEV metrics: accuracy={'accuracy': 0.6217125382262997}
- Average TEST metrics: accuracy={'accuracy': 0.6271899088997898}

Test 2 Accuracy (lr=5e-4, num_epochs=7):



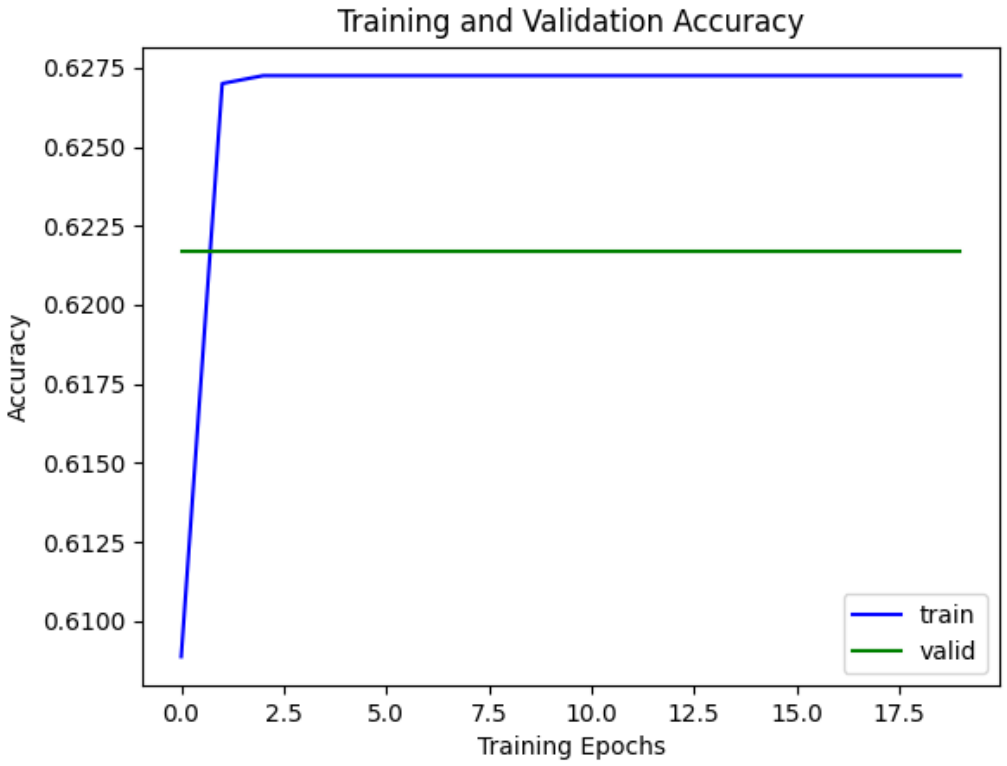
- Average DEV metrics: accuracy={'accuracy': 0.6217125382262997}
- Average TEST metrics: accuracy={'accuracy': 0.6194814295725298}

Test 3 Accuracy (lr=125e-4, num_epochs=12):



- Average DEV metrics: accuracy={'accuracy': 0.6217125382262997}
- Average TEST metrics: accuracy={'accuracy': 0.6187806587245971}

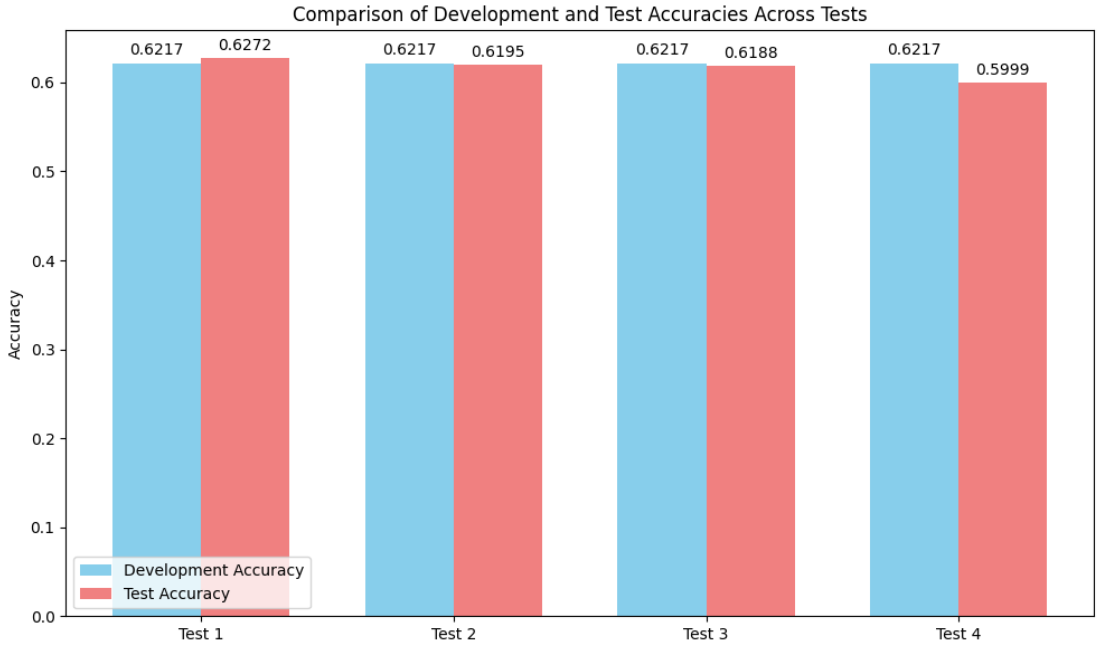
Test 4 Accuracy (lr=125e-4, num_epochs=20):



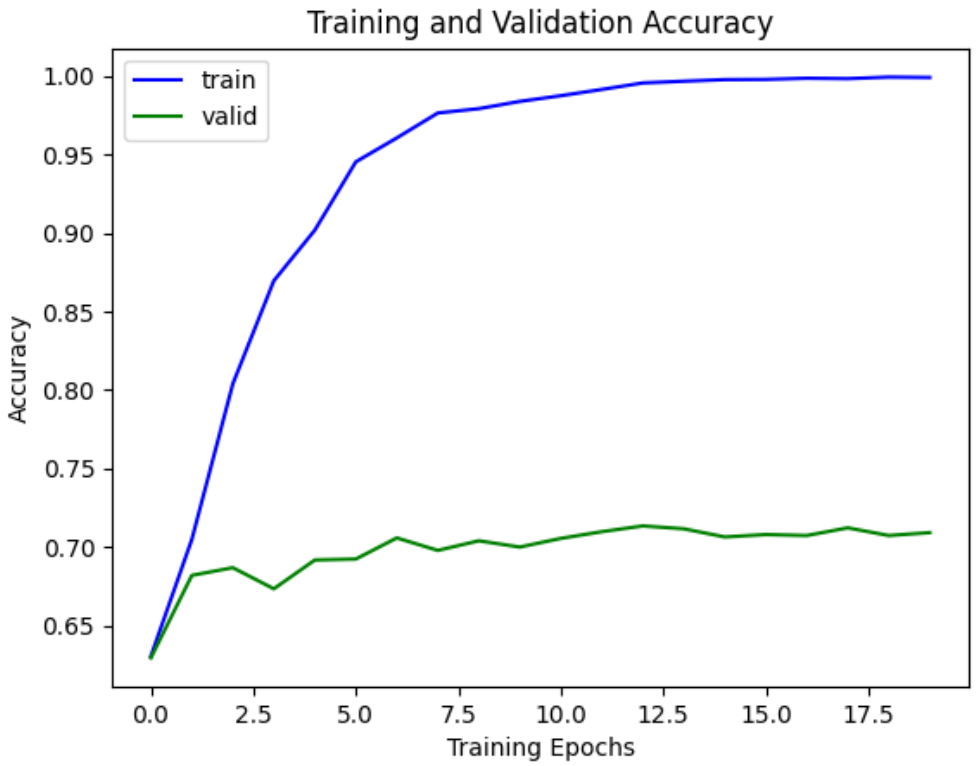
- Average DEV metrics: accuracy={'accuracy': 0.6217125382262997}

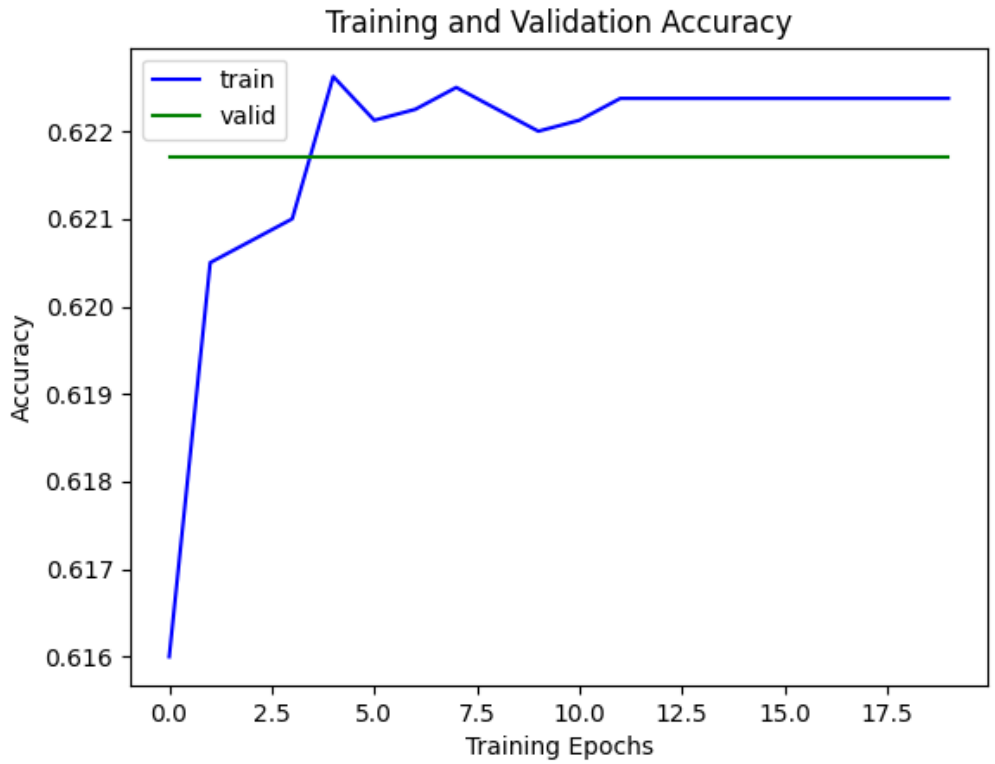
- Average TEST metrics: accuracy={'accuracy': 0.5998598458304134}

All tests compared:

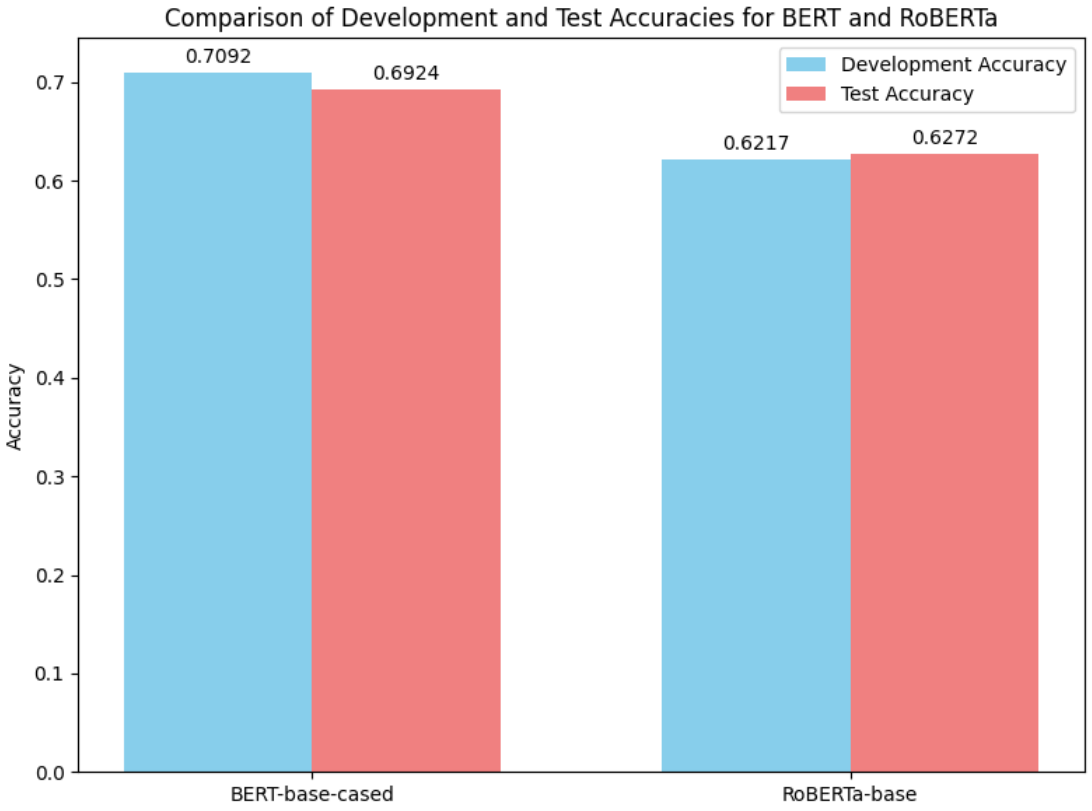


Figures for question 4





Accuracy:



BERT-base-cased

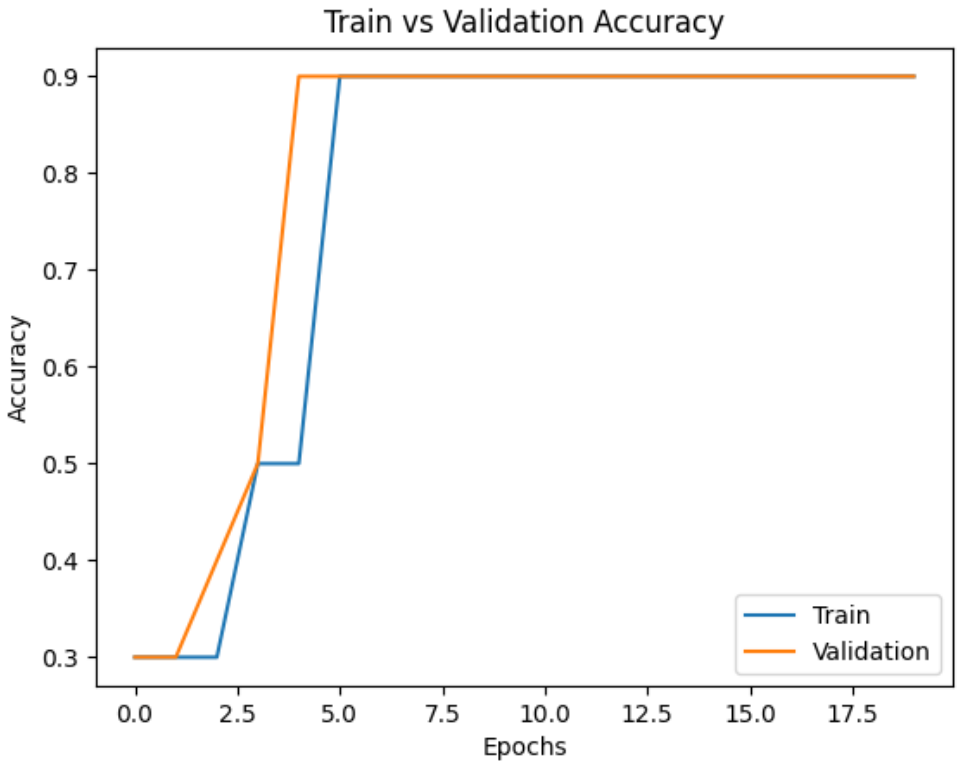
- Average DEV metrics: accuracy={'accuracy': 0.7091743119266055}
- Average TEST metrics: accuracy={'accuracy': 0.6923615977575333}

RoBERTa-base

- Average DEV metrics: accuracy={'accuracy': 0.6217125382262997}
- Average TEST metrics: accuracy={'accuracy': 0.6271899088997898}

3.2 Parameter-Efficient Fine-Tuning

For Question 1



Hyperparameters for full finetuneing:

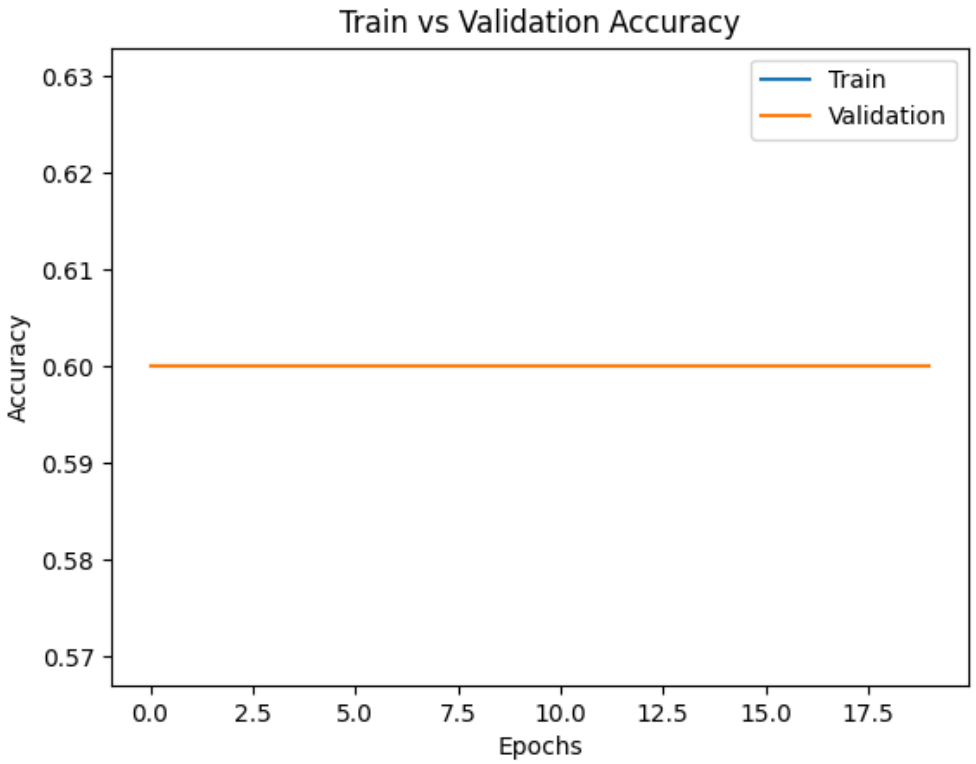
lr=1e-4, num_epochs=20, batch_size=64

Accuracy:

- Average DEV metrics: accuracy={'accuracy': 0.9}
- Average TEST metrics: accuracy={'accuracy': 0.9}

For Question 2

The number of tuned paramters is **2d + 2** (2d for number of labels and +2 for bias terms)



Hyperparameters for head finetuneing:

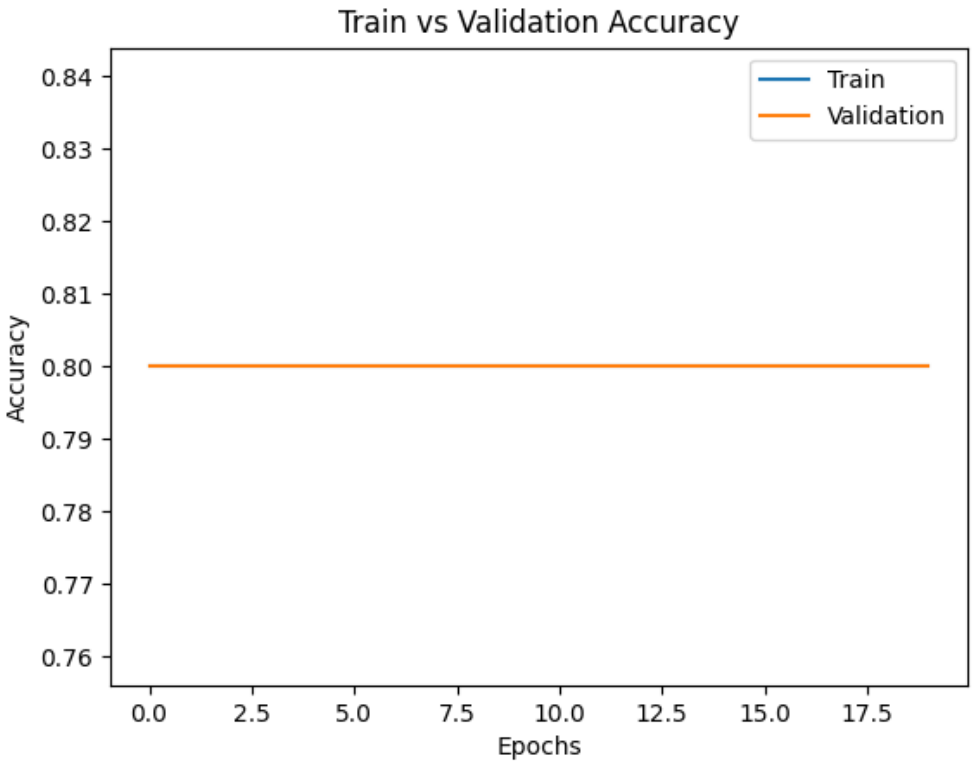
lr=1e-4, num_epochs=20, batch_size=64

Accuracy:

- Average DEV metrics: accuracy={'accuracy': 0.6}
- Average TEST metrics: accuracy={'accuracy': 0.6}

For Question 3

The number of tuned parameters for our prefix-tuning on BoolQ that appends trainable prefix embeddings of length 128, given the model hidden size is d is **130d + 2**



Hyperparameters for prefix finetuneing:

lr=1e-4, num_epochs=20, batch_size=64

Accuracy:

- Average DEV metrics: accuracy={'accuracy': 0.8}
- Average TEST metrics: accuracy={'accuracy': 0.8}