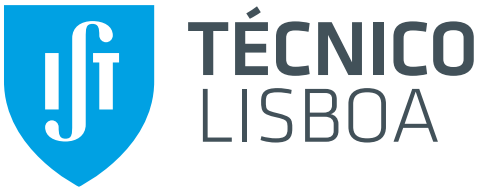

Electrical and Computer Engineering

Deep Learning

Homework 2

Authors:

Rodrigo Coimbra : 100078

Rodrigo Pereira : 100080

January 2024

Index

Contribution of each member to the project

For this project, we both addressed question 2 together. Then, we divided question 3. Therefore, Rodrigo Coimbra (100078) addressed Questions 3.1 and 3.3, while Rodrigo Pereira (100080) addressed Questions 3.2 and 3.4. For the first group, as these are theoretical questions, we decided that both of us would individually contemplate how to deduce the required answers and then come together to discuss, so we did the question together.

1 Question 1

1.1 Question 1.1

We were tasked to calculate the computational complexity of a softmax function with two multiplied matrices as input. Both matrices have a size of L by D as is showed in the following equations.

$$\mathbf{Q} \in \mathbb{R}^{(L\times D)} \wedge \mathbf{K} \in \mathbb{R}^{(L\times D)} \wedge \mathbf{V} \in \mathbb{R}^{(L\times D)} \tag{1}$$

We can then divide this problem in three parts, the first matrix multiplication, the application of softmax and lastly the final matrix multiplication. To compute the computational complexity we will use the big O notation.

Looking at the matrices operation, using the regular way of multiplication, and without using any sort of optimization algorithm, we will have to calculate each element individually. To calculate what is supposed to be a single element, we will have to do D multiplications between each element. Let's call the output matrix of the multiplication of A. For the first element of the matrix we get:

$$\mathbf{A_{11}} = \mathbf{Q}_{11} \cdot \mathbf{K}_{11} + \mathbf{Q}_{12} \cdot \mathbf{K}_{21} + ... + \mathbf{Q}_{1D} \cdot \mathbf{K}_{D1}$$

$$\mathbf{A_{li}} = \sum_{k=1}^{D} \mathbf{Q}_{lk} \cdot \mathbf{K}_{ki}$$

Then the matrix obtained can be written as:

$$\mathbf{A} = \begin{bmatrix} \sum_{k=1}^{D} \mathbf{Q}_{1k} \cdot \mathbf{K}_{k1} & \cdots & \sum_{k=1}^{D} \mathbf{Q}_{1k} \cdot \mathbf{K}_{kL} \\ \vdots & \vdots & \vdots \\ \sum_{k=1}^{D} \mathbf{Q}_{Lk} \cdot \mathbf{K}_{k1} & \cdots & \sum_{k=1}^{D} \mathbf{Q}_{Lk} \cdot \mathbf{K}_{kL} \end{bmatrix} \tag{2}$$

When trying to develop this formula in any programming language*, we obtain a for loop with size D. We then have to fill the obtained matrix which has a size of L by L, then we will have another two loops for that, making this block of code have a complexity of $\mathbf{O}(L^2 \times D)$.

For the softmax of the matrix obtained, that has a size of L by L, we also have to divide it into different parts. Firstly, we have to calculate the sum of the exponentials of one row, then we divide each member of that same row by the sum obtained. For the first element of the matrix we get:

$$\frac{e^{x_{11}}}{\sum_{i=1}^{L} e^{x_{1i}}}, \quad x_{ji} \in \mathbf{A} \tag{3}$$

When we do this sum of all members in a row and then this division for every column of that same row, it makes it $\mathbf{O}(L + L)$, however we have to do this for each row resulting in $\mathbf{O}(L \times (L + L))$. We will call the matrix obtained $\mathbf{B}$ that has a size of L by L.

*Algorithm for matrix multiplication without optimization

Finally, we multiply the matrix $\mathbf{B}$. obtained from the softmax, with $\mathbf{V}$. Once again we are multiplying matrices so we will obtain, once again, a computational complexity of $\mathbf{O}(L^2 \times D)$.

Combining every complexity obtained we will have $\mathbf{O}(L^2 \times D + L \times (2L) + L^2 \times D) = \mathbf{O}(2L^2 \times D + L \times (2L))$ and knowing that $L >> D \geq 2$ we end with $\mathbf{O}(L^2)$.

1.2 Question 1.2

Our objective is to obtain a feature map that for the $exp(\mathbf{q}^T\mathbf{k}) \approx \phi(\mathbf{q})^T\phi(\mathbf{k})$. Knowing that $\mathbf{q}$ and $\mathbf{k}$ are vectors $\in \mathbb{R}^D$, we can represent them as $\mathbf{q} = (q_1, q_2, \cdots, q_D)$ and $\mathbf{k} = (k_1, k_2, \cdots, k_D)$. Substituting that in the first three terms of the McLaurin series expansion for the exponential function, we obtain the following:

$$exp(t) \approx 1 + t + \frac{t^2}{2} \xrightarrow{t=\mathbf{q}^T\mathbf{k}}$$
$$\Rightarrow 1 + q_1k_1 + q_2k_2 + \cdots + q_Dk_D + \frac{q_1^2k_1^2 + q_2^2k_2^2 + \cdots + q_D^2k_D^2}{2} +$$
$$+ q_1q_2k_1k_2 + \cdots + q_1q_Dk_1k_D + q_2q_Dk_2k_D$$

The expression $\phi(\mathbf{q})^T\phi(\mathbf{k})$ represents the dot product of a row vector and a column vector. This results in a single value obtained by multiplying the corresponding components of the two vectors and then summing them. Therefore, each of the sums is one component, resulting in the following:

$$\phi(\mathbf{q})^T = \begin{bmatrix} 1 & q_1 & q_2 & \cdots & q_D & \frac{q_1^2}{\sqrt{2}} & \frac{q_2^2}{\sqrt{2}} & \cdots & \frac{q_D^2}{\sqrt{2}} & q_1q_2 & \cdots & q_1q_D & q_2q_D \end{bmatrix} \tag{4}$$

$$\phi(\mathbf{k})^T = \begin{bmatrix} 1 & k_1 & k_2 & \cdots & k_D & \frac{k_1^2}{\sqrt{2}} & \frac{k_2^2}{\sqrt{2}} & \cdots & \frac{k_D^2}{\sqrt{2}} & k_1k_2 & \cdots & k_1k_D & k_2k_D \end{bmatrix} \tag{5}$$

Examining the vectors (4) and (5), we can observe that the dimensionality M depends of D. Taking $\phi(\mathbf{q})$ as an example, M has 1 dimension from the first component, then D dimensions from the first $\mathbf{q}$ components that represent t. Lastly, we need to consider the combinations of all pairs of $\mathbf{q}$ for the last components that represent t^2. This number can be viewed as the number of multinomial coefficients of the sum of all $\mathbf{q}$ components squared, i.e., $(q_1 + q_2 + q_3)^2$. For that we have the following expression [†]:

$$\#_{n,m} = \binom{n+m-1}{m-1} \tag{6}$$

Here, m represents the number of terms being summed up, and n represents the power to which the sum is raised. Therefore, in our case we have $n = 2$ and $m = D$ is $\binom{2+D-1}{D-1} = \binom{D+1}{D-1}$. As a result, for the first three terms in this series, M can be describe as $1 + D + \binom{D+1}{D-1}$. Furthermore, to generalize for any number of terms, we observe that the last term is always raised to $k-1$, decreasing until it reaches the first number raised to 0. Therefore, by using the formula 6 we can sum all the terms from 0 to $k-1$, resulting in the following expression:

$$M = \sum_{x=0}^{k-1} \binom{x+D-1}{D-1} \tag{7}$$

1.3 Question 1.3

We know from the previous question that $exp(\mathbf{q}^T\mathbf{k}) \approx \phi(\mathbf{q})^T\phi(\mathbf{k})$, where $\phi(\mathbf{q})$ and $\phi(\mathbf{k})$ are both column vectors of size M. However, now we have two matrices, $\Phi(\mathbf{Q})$ and $\Phi(\mathbf{K})$, both of size $L \times M$. Therefore, each of the $\phi(\mathbf{q}_i)$ and $\phi(\mathbf{k}_i)$ represents a row vector equal to the transpose of the column vector $\phi(\mathbf{q})$.

[†] Multinomial theorem - Number of multinomial coefficients

$$\mathbf{Z} \approx \mathbf{D}^{-1}\Phi(\mathbf{Q})\Phi(\mathbf{K})^T\mathbf{V} \Leftrightarrow$$
$$\Leftrightarrow softmax(\mathbf{Q}\mathbf{K}^T)\mathbf{V}\mathbf{V}^{-1} \approx \mathbf{D}^{-1}\Phi(\mathbf{Q})\Phi(\mathbf{K})^T \Leftrightarrow$$
$$\Leftrightarrow softmax(\mathbf{Q}\mathbf{K}^T) \approx \mathbf{Diag}(\Phi(\mathbf{Q})\Phi(\mathbf{K})^T \cdot \mathbf{1}_L)^{-1}\Phi(\mathbf{Q})\Phi(\mathbf{K})^T$$

Looking now at the expression inside the Diag function, we have the following:

$$\Phi(\mathbf{Q})\Phi(\mathbf{K})^T \cdot \mathbf{1}_L = \\ = \begin{bmatrix} \phi(\mathbf{q}_1)\phi(\mathbf{k}_1) & \phi(\mathbf{q}_1)\phi(\mathbf{k}_2) & \cdots & \phi(\mathbf{q}_1)\phi(\mathbf{k}_L) \\ \phi(\mathbf{q}_2)\phi(\mathbf{k}_1) & \phi(\mathbf{q}_2)\phi(\mathbf{k}_2) & \cdots & \phi(\mathbf{q}_2)\phi(\mathbf{k}_L) \\ \vdots & & & \vdots \\ \phi(\mathbf{q}_L)\phi(\mathbf{k}_1) & \phi(\mathbf{q}_L)\phi(\mathbf{k}_2) & \cdots & \phi(\mathbf{q}_L)\phi(\mathbf{k}_L) \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{8}$$

We know that $\phi(\mathbf{q}_i)\phi(\mathbf{k}_i) \approx exp(\mathbf{q}_i\mathbf{k}_i)$, where $\mathbf{k}_i$ is now a column vector due to the transposition of $\Phi(\mathbf{K})$. This approximation is necessary to achieve the softmax function and, therefore, if applied the approximation from eq. 9 to our main expression, it results in an equality to Z.

$$\begin{bmatrix} \phi(\mathbf{q}_1)\phi(\mathbf{k}_1) & \phi(\mathbf{q}_1)\phi(\mathbf{k}_2) & \cdots & \phi(\mathbf{q}_1)\phi(\mathbf{k}_L) \\ \phi(\mathbf{q}_2)\phi(\mathbf{k}_1) & \phi(\mathbf{q}_2)\phi(\mathbf{k}_2) & \cdots & \phi(\mathbf{q}_2)\phi(\mathbf{k}_L) \\ \vdots & & & \vdots \\ \phi(\mathbf{q}_L)\phi(\mathbf{k}_1) & \phi(\mathbf{q}_L)\phi(\mathbf{k}_2) & \cdots & \phi(\mathbf{q}_L)\phi(\mathbf{k}_L) \end{bmatrix} \approx \begin{bmatrix} e^{\mathbf{q}_1\mathbf{k}_1} & e^{\mathbf{q}_1\mathbf{k}_2} & \ldots & e^{\mathbf{q}_1\mathbf{k}_L} \\ e^{\mathbf{q}_2\mathbf{k}_1} & e^{\mathbf{q}_2\mathbf{k}_2} & \ldots & e^{\mathbf{q}_2\mathbf{k}_L} \\ \vdots & & & \vdots \\ e^{\mathbf{q}_L\mathbf{k}_1} & e^{\mathbf{q}_L\mathbf{k}_2} & \ldots & e^{\mathbf{q}_L\mathbf{k}_L} \end{bmatrix} \tag{9}$$

Therefore, by substituting the approximation from Equation 9 into Equation 8, we obtain the following :

$$\begin{bmatrix} e^{\mathbf{q}_1\mathbf{k}_1} & e^{\mathbf{q}_1\mathbf{k}_2} & \ldots & e^{\mathbf{q}_1\mathbf{k}_L} \\ e^{\mathbf{q}_2\mathbf{k}_1} & e^{\mathbf{q}_2\mathbf{k}_2} & \ldots & e^{\mathbf{q}_2\mathbf{k}_L} \\ \vdots & & & \vdots \\ e^{\mathbf{q}_L\mathbf{k}_1} & e^{\mathbf{q}_L\mathbf{k}_2} & \ldots & e^{\mathbf{q}_L\mathbf{k}_L} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{L} e^{\mathbf{q}_1\mathbf{k}_n} \\ \sum_{n=1}^{L} e^{\mathbf{q}_2\mathbf{k}_n} \\ \vdots \\ \sum_{n=1}^{L} e^{\mathbf{q}_L\mathbf{k}_n} \end{bmatrix}$$

Given that the inverse of a diagonal matrix is given by replacing the main diagonal elements of the matrix with their reciprocals, we obtain the following:

$$\mathbf{Diag}(\Phi(\mathbf{Q})\Phi(\mathbf{K})^T \cdot \mathbf{1}_L)^{-1}\Phi(\mathbf{Q})\Phi(\mathbf{K})^T \approx$$
$$\begin{bmatrix} \frac{1}{\sum_{n=1}^{L} e^{\mathbf{q}_1\mathbf{k}_n}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sum_{n=1}^{L} e^{\mathbf{q}_2\mathbf{k}_n}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sum_{n=1}^{L} e^{\mathbf{q}_L\mathbf{k}_n}} \end{bmatrix} \begin{bmatrix} e^{\mathbf{q}_1\mathbf{k}_1} & e^{\mathbf{q}_1\mathbf{k}_2} & \ldots & e^{\mathbf{q}_1\mathbf{k}_L} \\ e^{\mathbf{q}_2\mathbf{k}_1} & e^{\mathbf{q}_2\mathbf{k}_2} & \ldots & e^{\mathbf{q}_2\mathbf{k}_L} \\ \vdots & & & \vdots \\ e^{\mathbf{q}_L\mathbf{k}_1} & e^{\mathbf{q}_L\mathbf{k}_2} & \ldots & e^{\mathbf{q}_L\mathbf{k}_L} \end{bmatrix} =$$
$$\begin{bmatrix} \frac{e^{\mathbf{q}_1\mathbf{k}_1}}{\sum_{n=1}^{L} e^{\mathbf{q}_1\mathbf{k}_n}} & \frac{e^{\mathbf{q}_1\mathbf{k}_2}}{\sum_{n=1}^{L} e^{\mathbf{q}_1\mathbf{k}_n}} & \cdots & \frac{e^{\mathbf{q}_1\mathbf{k}_L}}{\sum_{n=1}^{L} e^{\mathbf{q}_1\mathbf{k}_n}} \\ \frac{e^{\mathbf{q}_2\mathbf{k}_1}}{\sum_{n=1}^{L} e^{\mathbf{q}_2\mathbf{k}_n}} & \frac{e^{\mathbf{q}_2\mathbf{k}_2}}{\sum_{n=1}^{L} e^{\mathbf{q}_2\mathbf{k}_n}} & \cdots & \frac{e^{\mathbf{q}_2\mathbf{k}_L}}{\sum_{n=1}^{L} e^{\mathbf{q}_2\mathbf{k}_n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{e^{\mathbf{q}_L\mathbf{k}_1}}{\sum_{n=1}^{L} e^{\mathbf{q}_L\mathbf{k}_n}} & \frac{e^{\mathbf{q}_L\mathbf{k}_2}}{\sum_{n=1}^{L} e^{\mathbf{q}_L\mathbf{k}_n}} & \cdots & \frac{e^{\mathbf{q}_L\mathbf{k}_L}}{\sum_{n=1}^{L} e^{\mathbf{q}_L\mathbf{k}_n}} \end{bmatrix} = \mathbf{Z}$$

1.4 Question 1.4

Let's start by looking at the sizes of the matrices. We have that $\mathbf{V}$ is of size $L \times D$, $\Phi(\mathbf{Q})$ is of size $L \times M$, $\Phi(\mathbf{K})^T$ is of size $M \times L$ and $\mathbf{D}^{-1}$ is of size $L \times L$.

Starting with the multiplication of $\Phi(\mathbf{K})^T$ and $\mathbf{V}$, we obtain a complexity of $\mathbf{O}(M \times L \times D)$, and a resulting matrix of size $M \times D$.

Next, we perform the multiplication of $\Phi(\mathbf{Q})$ with the previously obtained matrix. This operation has a complexity of $\mathbf{O}(L \times M \times D)$, resulting in a matrix of size $L \times D$. Let's call it $\mathbf{P}$ for simplification.

Taking into account the complexity of inverting the $\mathbf{D}$, which is a diagonal matrix, the inversion operation has a complexity of $\mathbf{O}(L)$.

Finally, for the last operation, since one of the matrices is diagonal, the complexity of the multiplication can be expressed through two sums as follows:

$$\sum_{i=1}^{L} \sum_{j=1}^{D} \mathbf{D}_{ii}^{-1} \mathbf{P}_{ij}$$

This translates to a complexity of $\mathbf{O}(L \times D)$. Summing up all the operations, the overall complexity is given by $\mathbf{O}(M \times L \times D + L \times M \times D + L + L \times D) = \mathbf{O}(M \times L \times D)$.

2 Question 2

For this Question, we were tasked with the development of a Pytorch convulutional neural network, using a skeleton code provided by the professors.

2.1 Question 2.1

We implemented in pytorch a CNN, using the layers and hyperparameters provided by the professors.We then run it through 15 epochs while changing the learning rate between 0.1, 0.01, 0.001.

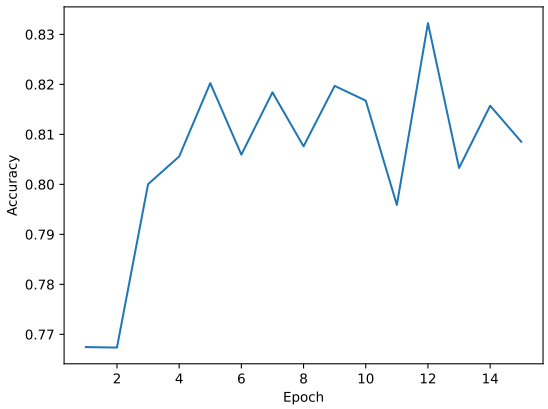

Fig. 1: Validation Accuracy with Lr=0.1

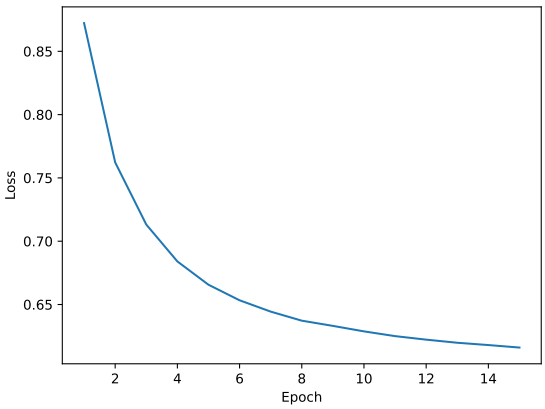

Fig. 2: Train Loss with Lr=0.1

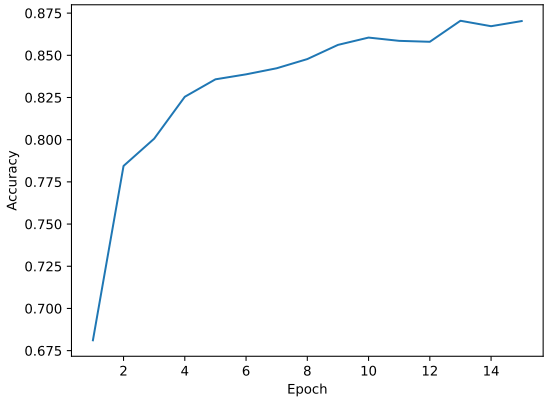

Fig. 3: Validation Accuracy with Lr=0.01

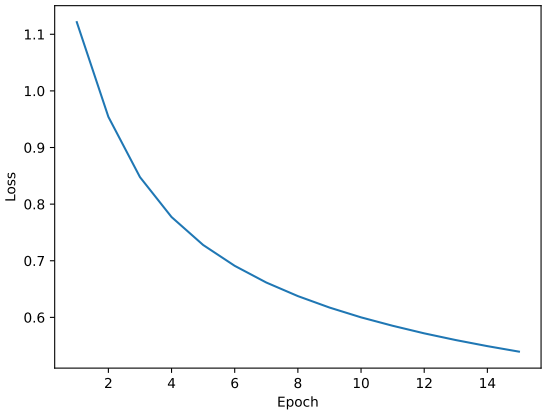

Fig. 4: Train Loss with Lr=0.01

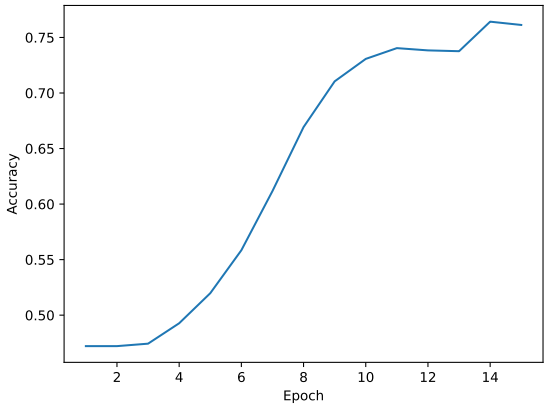

Fig. 5: Validation Accuracy with Lr=0.001

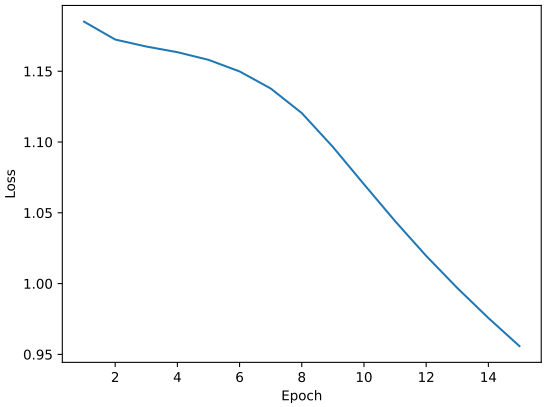

Fig. 6: Train Loss with Lr=0.001

Learning rate	0.001	0.01	0.1
Train Loss	0.9558	0.5395	0.6160
Validation Acc	0.7613	0.8703	0.8085
Test Acc	0.7391	0.8261	0.7807

Tab. 1

As can be seen in Fig.1, the model with a learning rate of 0.1 exhibits a significantly unstable trend, suggesting the need to consider lowering the learning rate. Additionally, we can observe that the test accuracy is the highest when the learning rate parameter is set to 0.01 and the lowest when set to 0.001. Furthermore, the model with $lr = 0.01$ was also the one with the lowest train loss and best validation accuracy.

2.2 Question 2.2

Now we removed the Max Pooling layers and changed the convolutional layers so the torches keep the same size in the end. Once again, we run it throught 15 epochs while changing between different learning rates.

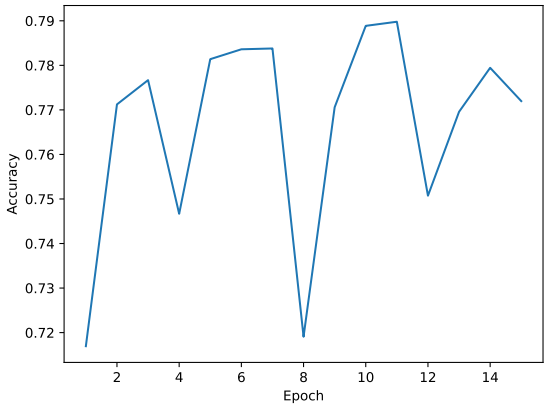

Fig. 7: Validation Accuracy with Lr=0.1

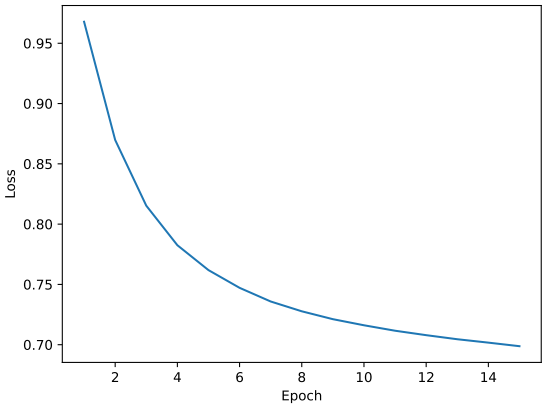

Fig. 8: Train Loss with Lr=0.1

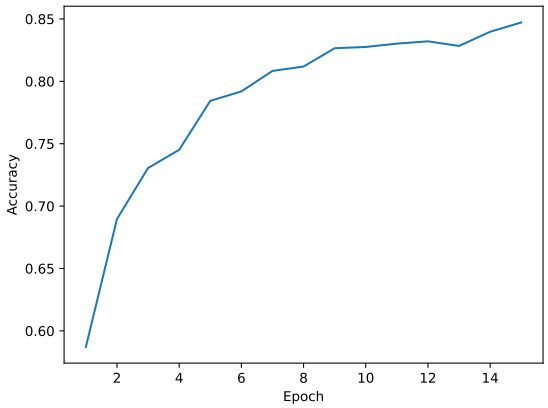

Fig. 9: Validation Accuracy with Lr=0.01

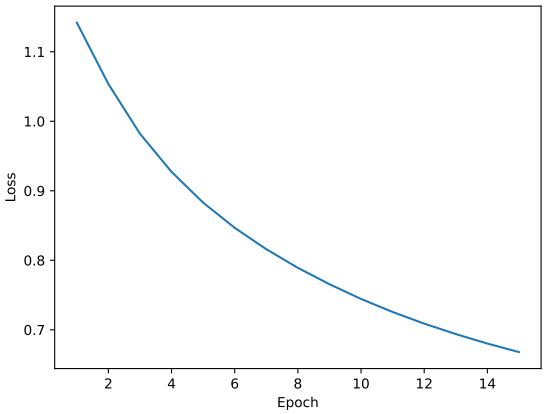

Fig. 10: Train Loss with Lr=0.01

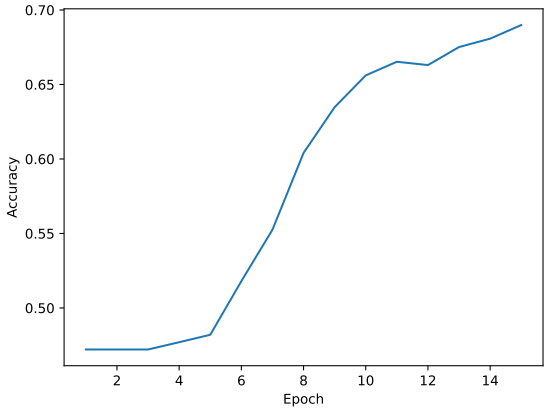

Fig. 11: Validation Accuracy with Lr=0.001

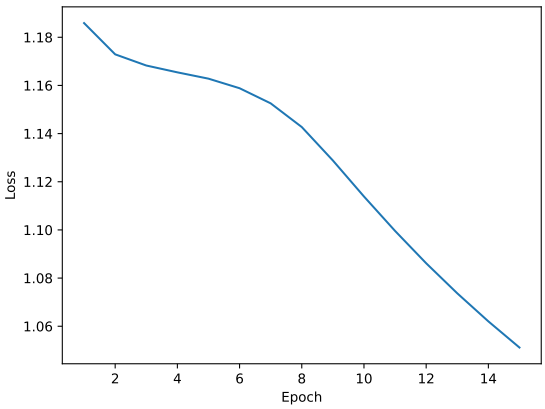

Fig. 12: Train Loss with Lr=0.001

Learning rate	0.001	0.01	0.1
Train Loss	1.0512	0.6679	0.6988
Validation Acc	0.6899	0.8472	0.7720
Test Acc	0.7146	0.8147	0.7524

Tab. 2

As observed, the model with a learning rate of 0.1 remained too unstable to be considered a good option. However, it wasn't the worst anymore, being the one with $lr = 0.001$ the worst. As before, the model with $lr = 0.01$ outperformed the others, having the best overall metrics.

2.3 Question 2.3

After implementing the function get_number_trainable_params in our code, we obtained 224892 for both models. This similarity arises from the fact that, in the first model, we used MaxPooling with a stride of 2 to halve our input, while in the second model, we directly use convolutions with a stride of 2.This results in an equal number of parameters, as both operations lead to a similar reduction during the MaxPooling and convolution operations. Additionally, this is only true because the MaxPooling operation doesn't involve weights. Furthermore, when comparing both models, we see that the second one performed slightly worse than the other. Since all these models depend on initialization, i.e., random values, it's possible that this specific seed favored the other model. Another possible explanation, is that the model adapts better to this type of data. However, given that the difference in the test accuracy is only approximately 0.01, it's insufficient to make a definitive conclusion.

3 Question 3

3.1 Question 3.1

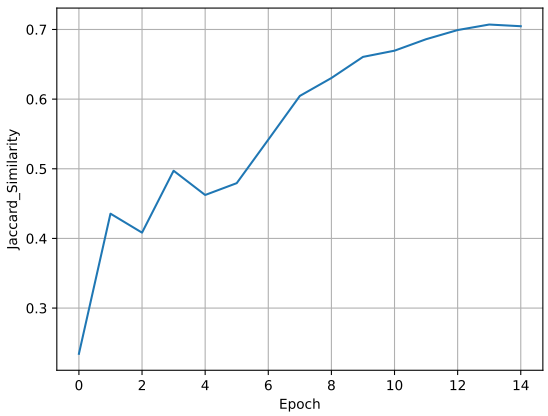

Fig. 13: Jaccard similarity

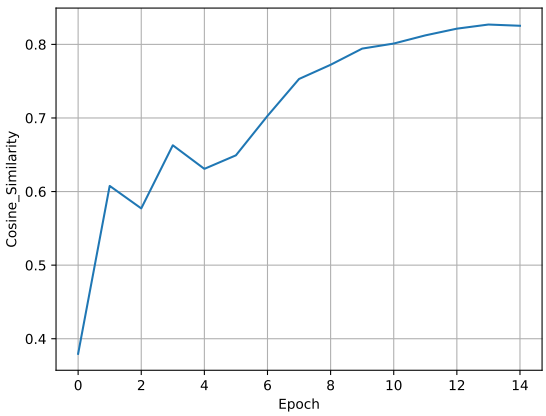

Fig. 14: Cosine similarity

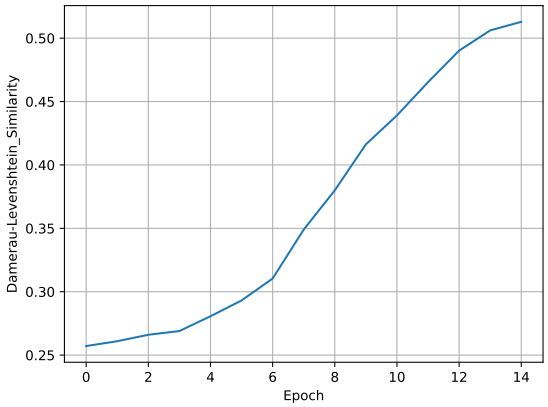

Fig. 15: Damerau-levenshtein similarity

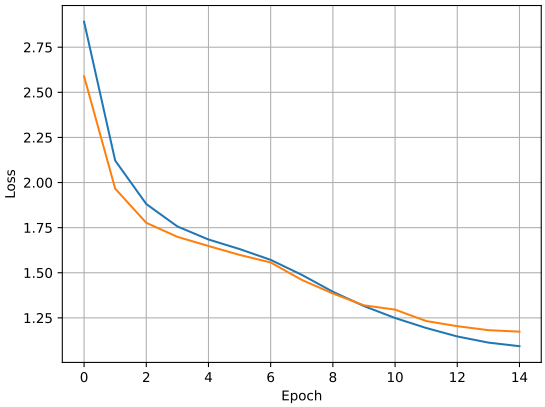

Fig. 16: Loss of both train(blue) and validation(orange) sets

As it can be seen, the model that has the decoder that employs a recurrent-based approach to process the input shows good results when we score it according to cosine similarity with 0.8324. However, looking now at Damerau-levenshtein similarity we observe a much smaller similarity of 0.5087, showing that the model is not very good when using this criterion. This similarities look to different things, that will be explained in 3.4.

Jaccard similarity	0.7149
Cosine similarity	0.8324
Damerau-levenshtein similarity	0.5087
Loss	1.1828

Tab. 3: Scores obtained on the test set

3.2 Question 3.2

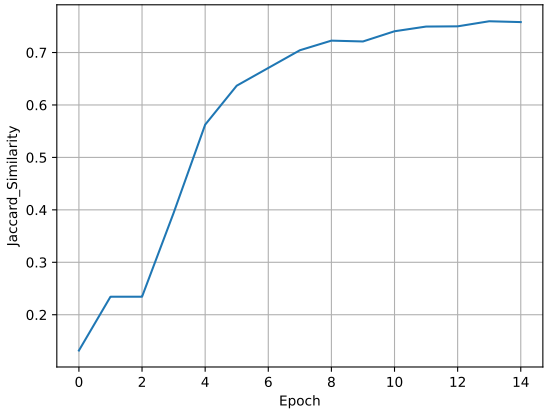

Fig. 17: Jaccard similarity

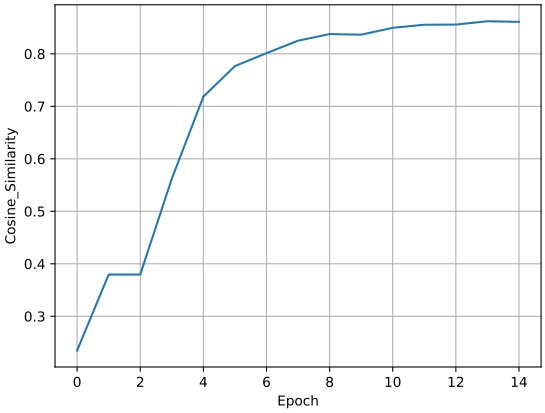

Fig. 18: Cosine similarity

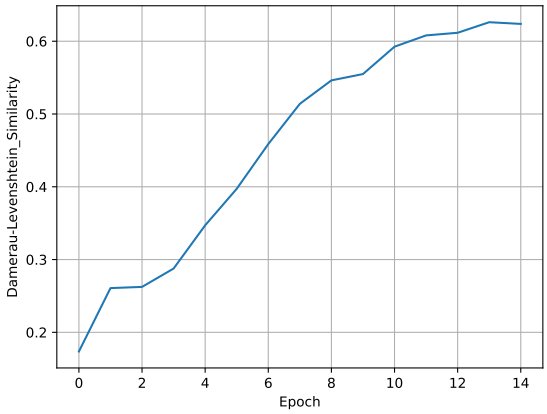

Fig. 19: Damerau-levenshtein similarity

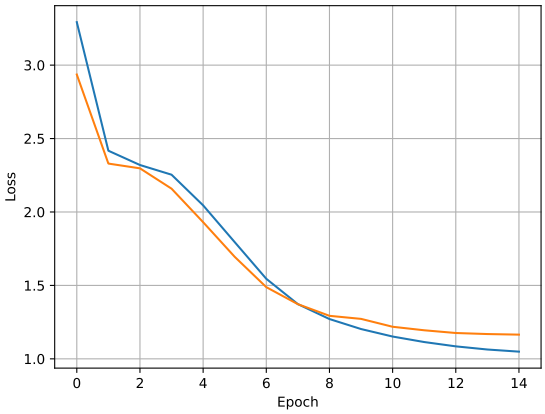

Fig. 20: Loss of both train(blue) and validation(orange) sets

Lastly, in this scenario, the decoder employing an attention-based mechanism yields favorable results when assessed using cosine similarity, achieving a score of 0.8651. However, a different picture emerges when evaluating the model using Damerau-Levenshtein similarity, where the observed similarity drops significantly to 0.6341. This discrepancy indicates that, according to the Damerau-Levenshtein criterion, the model's performance is comparatively less satisfactory.

Jaccard similarity	0.7644
Cosine similarity	0.8651
Damerau-levenshtein similarity	0.6341
Loss	1.1606

Tab. 4: Scores obtained on the test set

3.3 Question 3.3

3.3.1 LSTM

LSTM was developed to solve a problem in RNN which is the vanishing/exploding gradient. The vanishing/exploding gradient appears when we want to use a RNN where the inputs are too many, by multiplying the output with the weights, to get to the next iteration and use it's memory, we could get something way too big if that weight is greater than 1 (exploding gradient) or something too small if the weight is less than 1(vanishing gradient).

The way LSTM solves this problem is by creating a long term memory and a short term memory, where the short term memory is used almost similar to the RNN but we also have a long term memory where will take effect on more "distant" iterations.

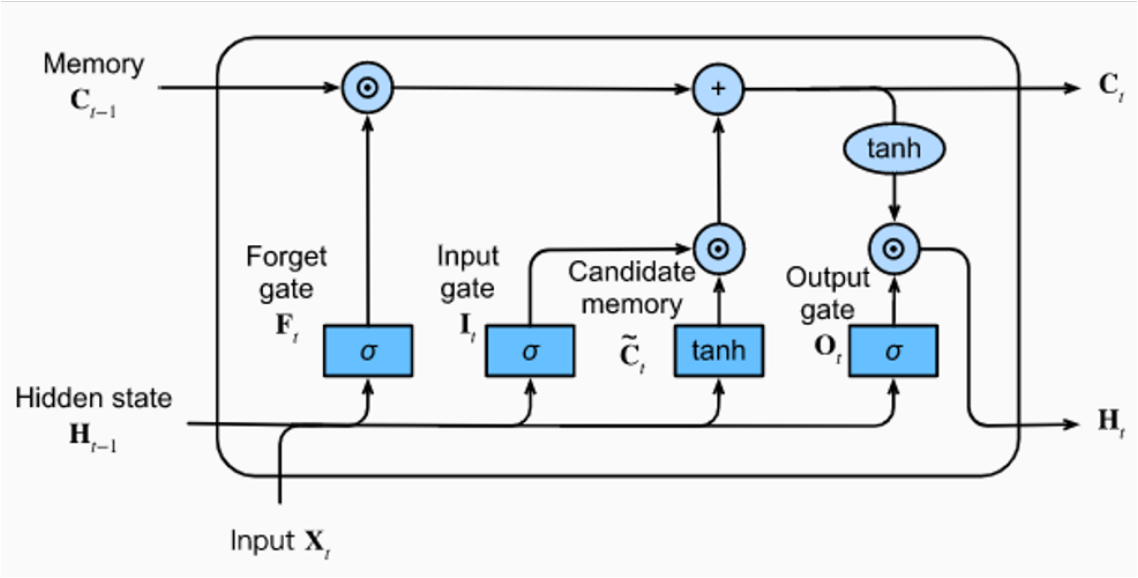

Fig. 21: LSTM

In Figure 21[‡], we have the basic structure of a LSTM for the first input. Hidden state will be our short term memory and memory is our long term memory. The forget gate works as a way to define how much of our long term memory we want to keep. We apply weights to our short-term memory and input, and introduce a bias. Then, we pass the results through a sigmoid, which returns a value between 0 and 1.

Now our input gate will work alongside the candidate memory, where our candidate memory is the potential memory to add to our long term memory and the input gate memory gives the percentage of that potential memory that we want to keep. Once again we apply our weights and biases for each gate, and then multiply the output of them to add to our long term memory.

That gives our long term memory to keep for the next input, but now we want to calculate our short term memory for the next iteration, for that we use an output gate that gives, once again, the percentage of short term memory that we want to keep. After calculating that percentage we apply a tanh on our long term memory, that gives us the potential short term memory. As we did with our long term memory, we do the same multiplication to our short term memory.

3.3.2 Attention Mechanism

While the LSTM is an architecture of RNNs to solve a specific problem intrinsic to it, the Attention Mechanism is, as the name says, a mechanism used by the model to be able to know context from previous inputs to decide a solution. For that, it uses three matrices of values, named Value, Keys and Query. He also learn those weights, in order to understand better the context of the phrases, because it relate the actual word to all of the previous.

While an LSTM can lose track of information from distant parts of the input sequence, which might be important for providing context to the current word, attention mechanisms establish connections to each of the previous inputs. This allows the prediction of the next word to be conditioned by the most important words related to the current word. This observation is supported by examining all our scores, where each score indicates a higher value for the second model. This suggests that the attention mechanism performs better than LSTMs alone.

[‡]Long Short-Term Memory (LSTM)

3.4 Question 3.4

To evaluate the results obtained, we use 3 different string similarity scores.

3.4.1 Jaccard Similarity

This type of score uses a more direct approach where it divides each string it wants to compare in different groups, let's call them group **A** for the target answer and **B** for the predicted string. To get the groups, we will create a smaller groups of n letters, for example, the word "profunda" in groups of 2 letters will result in {pr,ro,of,fu,un,nd,da} for **A** and then let's say our output obtained was "prefounda" which will result in {pr,re,ef,fo,ou,un,nd,da}. Then what the Jaccard Similarity does to calculate the score is:

$$Score = \frac{(\mathbf{A} \cap \mathbf{B})}{(\mathbf{A} \cup \mathbf{B})}$$

$$(\mathbf{A} \cap \mathbf{B}) = \{pr, un, nd, da\} = 4$$

$$(\mathbf{A} \cup \mathbf{B}) = \{pr, ro, of, fu, re, ef, fo, ou, un, nd, da\} = 11$$

Thus resulting in a Score of $\frac{4}{11}$ for this example.

3.4.2 Damerau-Levenshtein Similarity

Damerau-Levenshtein Similarity is way simpler since it just compares what is different between each strings and says the *distance* between them. we only need to observe the differences between both strings. If we need to add, erase, or change a character to make the strings appear more similar, we increase the distance by the count of those changes. We also need to pay attention as it needs to be the lowest number possible of changes. Using the same strings as before, "profunda" and "prefounda", the *distance* between them will be 2 since by changing "e" in "pre" to "o" and removing the "o" in "fou". To calculate the score we just do:

$$Score = 1 - \frac{Distance}{Max(strlen(A), strlen(B))}$$

3.4.3 Cosine Similarity

Lastly, Cosine Similarity uses dot product to obtain its score. We start by obtaining the $cos(\phi)$ through the dot product formula:

$$cos(\phi) = \frac{\overrightarrow{\mathbf{a}} \cdot \overrightarrow{\mathbf{b}}}{|\mathbf{a}||\mathbf{b}|}$$

Because we are using dot product, we also need to define the vectors $\overrightarrow{\mathbf{a}}$ and $\overrightarrow{\mathbf{b}}$. To get the arrays, we will draw all the different words from both sentences, and then turn each of those different words in features. Then we just set each feature to 1 or 0 if those words are present or not in the sentence. For example, if we have the sentences "The black cat eats mouses" and "The white cat runs from mouses", we can turn this words into features such as ("white","black","cat","eats","runs","from","mouses").We also ignore stop words and punctuation to create those features. Then we get for each vector:

$$\overrightarrow{\mathbf{a}} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \overrightarrow{\mathbf{b}} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Applying the dot product to calculate $cos(\phi)$, we will obtain a value between -1 and 1 where the closer to one, the more identical they are, the closer to -1 the more different they are and if they are close to 0 then those sentences doesn't have that much in common.

Basically, they work in very different ways, one compares character per character having even their order in mind(Damerau-Levenshtein Similarity), while Cosine Similarity and Jaccard divide the sentence into groups and compare them, using different notations to calculate the score, which justifies why the scores obtained differ.