Deep Learning - Homework 2

99222 - Frederico Silva, 99326 - Sebastião Carvalho

January 6, 2024

Contents

1 Question 1

1.1 Question 1.1

Using a single attention head, the output Z is given by $Z = \text{softmax}\left(QK^T\right)V$.

The complexity of computing this Z, is the complexity of computing all the matrix multiplications and the softmax.

Starting with the dimensions of each matrix, we have: $Q \in \mathbb{R}^{L\times D}, K \in \mathbb{R}^{L\times D} and V \in \mathbb{R}^{L\times D}$.

The complexity of computing a matrix product AB, where $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$, is $O(mnp)$. We can prove this by taking into account that, to compute the element c_{ij} of the matrix $C = AB$, we need to compute the dot product between the i-th row of A and the j-th column of B, and this dot product has complexity of $O(n)$, because it is the sum of n products of real numbers. Since we need to compute mp elements of C, the complexity of computing C is $O(mnp)$.

Let's now consider $P = softmax\left(QK^T\right)$. With this P, we can compute $Z = PV$.

In our case, we first need to compute QK^T and since $Q \in \mathbb{R}^{L\times D}$ and $K^T \in \mathbb{R}^{D\times L}$, the complexity of this operation is $O(L^2D)$.

Finally, we need to compute the softmax of each row of the matrix QK^T, which has complexity of $O(L^2)$. The complexity of computing P is $O(L^2D + L^2 + L^2) = O(L^2D)$. Since $P = softmax\left(QK^T\right)$, $P \in \mathbb{R}^{L \times L}$.

Lastly, we need to compute the matrix product PV. Since $P \in \mathbb{R}^{L \times L}$ and $V \in \mathbb{R}^{L \times D}$, the complexity of this operation is $O(L^2D)$.

With this, the final complexity of computing Z is $O(L^2D)$, since the complexity of computing P is $O(L^2D)$ and the complexity of computing PV is $O(L^2D)$.

Let's consider the number of hidden units (D) is fixed, so the complexity of computing Z is $O(L^2)$.

This may cause a problem for long sequences of text, since the complexity of computing Z is $O(L^2)$, where L is the length of the sequence. This means that the complexity of computing Z is quadratic in the length of the sequence, which is not good for long sequence inputs.

1.2 Question 1.2

For this exercise, we will use the McLaurin series expansion of the exponential function, to approximate the softmax and reduce the computational complexity. The McLaurin series expansion of the exponential function is given by:

$exp(t) = \sum_{n=0}^{\infty} \frac{t^n}{n!}$

Firstly, considering $exp(t) \approx 1 + t + \frac{t^2}{2}$, we want to create a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^M$ such that, for an arbitrary $q \in \mathbb{R}^D$ and $k \in \mathbb{R}^D$ we have $exp(q^T k) \approx \phi(q)^T \phi(k)$.

With this, we want to find a mapping ϕ such that: $\phi(q)^T \phi(k) = 1 + q^T k + \frac{(q^T k)^2}{2}$.

The first two terms of the series are trivial, since we can define $\phi(q) = [1, q_1, \ldots, q_n]$. For the third term, we need to decompose the square of the dot product of q and k, into a sum of products of the elements of q and k.

For vectors x and z with the same dimension, n, we have that $x^T z = \sum_{i=1}^{D} x_i z_i$. Now for the square of the doct product we have:

$(x^T z)^2 = (x_1 z_1 + \cdots + x_n)(x_1 z_1 + \cdots + x_n z_n) = (x_1 z_1)^2 + 2x_1 z_1 x_2 z_2 + \cdots + 2x_1 z_1 x_n z_n +$
$+ (x_2 z_2)^2 + \cdots + x_2 z_2 x_n z_n + \cdots + (x_n z_n)^2 = \sum_{i=1}^{n} (x_i)^2 (z_i)^2 + 2 \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_i z_i x_j z_j$

This way, we can define $\phi(x) = [1, x_1, \ldots, x_n, \frac{1}{\sqrt{2}} x_1^2, x_1 x_2, \ldots, x_1 x_n, \frac{1}{\sqrt{2}} x_2^2, \ldots, x_2 x_n, \ldots, x_{n-1} x_n, \frac{1}{\sqrt{2}} x_n^2]$.

With this, $exp(q^T k) \approx \phi(q)^T \phi(k)$, and we can use this to approximate the softmax function.

In terms of dimensionality, if the vector x has dimension D, the first two terms of the series will have dimension $D + 1$. From the third term, we can see that the number of terms will be $\sum_{i=1}^{D} i$, and this sum has a known closed form, which is $\frac{D(D+1)}{2}$. With this, and considering K as the degree of the polynomial, for $K = 2$, the vector $\phi(x)$ will have dimension $M = 1 + D + \frac{D(D+1)}{2}$.

Now, we want to assess what would be the dimensionality of the feature space M, if we used the McLaurin series with $K \geq 3$ terms. For this, we have to assess the dimensionality of each term.

According to the multinomial theorem:

$(x_1 + x_2 + \cdots + x_D)^K = \sum_{k_1 + k_2 + \cdots + k_D = K, k_1, k_2, \ldots, k_D \geq 0} \binom{n}{k_1, k_2, \ldots, k_D} \prod_{t=1}^{D} x_t^{k_t}$,
where $\binom{n}{k_1, k_2, \ldots, k_D} = \frac{n!}{k_1! k_2! \ldots k_D!}$.

To prove that the number of terms in the expansion for the K-th term is $\binom{K+D-1}{D-1}$, we will use the stars and bars method.

We know that from combinatorics, for any pair of positive integers n and k, the number of k-tuples of non-negative integers whose sum is n, is equal to the number of multisets of cardinality n taken from a set of size k, or equivalently, the number of multisets of cardinality $k-1$ taken from a set of size $n+1$.

Using the stars and bars methods, to show that there are $\binom{K+D-1}{D-1}$ possible arrangements, observe that any arrangement of stars and bars consists of a total of $D+K-1$ objects, K of which are stars and $D-1$ of which are bars. We only need to pick $D-1$ of the $K+D-1$ positions to be bars.

Since $k_1 + k_2 + \cdots + k_D = K$, we can apply this combinatorics theorem and know that the number of k-tuples of non-negative integers whose sum is K, is equal to $\binom{K+D-1}{D-1}$, and thus, the number of terms in the expansion for the K-th term is $\binom{K+D-1}{D-1}$.

With this, for $K \geq 3$, the dimensionality of the feature space will be: $M = \sum \binom{K+D-1}{D-1}$.

1.3 Question 1.3

In the previous exercise, we defined the feature map $\phi : \mathbb{R}^{\mathbb{D}} \to \mathbb{R}^{\mathbb{M}}$, such that $exp(q^T k) \approx \phi(q)^T \phi(k)$. Now let's consider the mapping Φ where $\Phi(X)$, which results in a matrix whose rows are $\phi(x_i)$, where x_i is the i-th row of X.

Our goal is to show that the self-attention operation can be approximated as $Z \approx D^{-1}\Phi(Q)\Phi(K)^T V$, where $D = Diag(\Phi(Q)\Phi(K)^T 1_L)$.

Looking at the original self-attention operation, we can see that the difference, is that now we want to approximate $softmax(QK^T)$ as $D^{-1}\Phi(Q)\Phi(K)^T$.

Considering $softmax(QK^T)_{ij} = \frac{exp(q_i^T k_j)}{\sum_{l=1}^{L} exp(q_i^T k_l)}$, since we want $softmax(QK^T) \approx D^{-1}\Phi(Q)\Phi(K)^T$, we can see that D^{-1} will correspond to the denominator of the softmax operation, and $\Phi(Q)\Phi(K)^T$ will correspond to the numerator.

Let's focus on the $\Phi(Q)\Phi(K)^T$ part. We know that this is the same as:

$$\Phi(Q)\Phi(K)^T = \begin{bmatrix} - - - \phi(q_1) - - - \\ - - - \phi(q_2) - - - \\ \vdots \\ - - - \phi(q_L) - - - \end{bmatrix} \begin{bmatrix} | & | & \cdots & | \\ \phi(k_1) & \phi(k_2) & \cdots & \phi(k_L) \\ | & | & \cdots & | \end{bmatrix}$$

Since $\phi(q_i)$ is spread along the i-th row, for the (i,j)-th element of the matrix $\Phi(Q)\Phi(K)^T$ we have that the cell i,j is equal to $phi(q_i)_1\phi(k_j)_1 + phi(q_i)_2\phi(k_j)_2 + \cdots + phi(q_i)_M\phi(k_j)_M = \phi(q_i)^T\phi(k_j)$.

With this, we can see that $\Phi(Q)\Phi(K)^T$ is a matrix whose (i,j)-th element, is equal to $\phi(q_i)^T\phi(k_j)$.

We know that $\phi(q)^T\phi(k) \approx exp(q^T k)$, so we can see that $\Phi(Q)\Phi(K)^T$ is a matrix whose (i,j)-th element is an approximation of $exp(q_i^T k_j)$.

Now let's look at D. We know that $D = Diag(\Phi(Q)\Phi(K)^T 1_L)$. Since $\Phi(Q)\Phi(K)^T$ is a matrix whose (i,j)-th element is an approximation $exp(q_i^T k_j)$, and 1_L is a vector of ones, the product $\Phi(Q)\Phi(K)^T 1_L$ will be a vector whose i-th element is an approximation of $\sum_{j=1}^{L} exp(q_i^T k_j)$, given it is the sum of the i-th row of $\Phi(Q)\Phi(K)^T$.

With this, we can see that $D \in \mathbb{R}^{L \times L}$ is a diagonal matrix whose (i,i)-th element is an approximation of $\sum_{j=1}^{L} exp(q_i^T k_j)$. Since this is a diagonal matrix, and the inverse of a diagonal matrix is a diagonal matrix whose (i,i)-th element is the inverse of the (i,i)-th element, we can see that D^{-1} is a diagonal matrix whose (i,i)-th element is an approximation of $\frac{1}{\sum_{j=1}^{L} exp(q_i^T k_j)}$.

Now that we have D^{-1} and $\Phi(Q)\Phi(K)^T$, we can see that $D^{-1}\Phi(Q)\Phi(K)^T$ is a matrix whose (i,j)-th element is an approximation of the softmax operation, which is what we wanted to show. With this, we can see that $Z = softmax(QK^T)V \approx D^{-1}\Phi(Q)\Phi(K)^T V$.

1.4 Question 1.4

Now we wish to use the above approximation to obtain a computational complexity that is linear in L. First, let us compute the computational complexity of the approximation. For that, we need the complexity of calculating $\Phi(Q)$ and $\Phi(K)$.

Since to compute each row of $\Phi(Q)$ and $\Phi(K)$ we need to compute $\phi(q_i)$ and $\phi(k_i)$, and the complexity of this is $O(M)$, the complexity of calculating $\Phi(Q)$ and $\Phi(K)$ is $O(LM)$.

Now, we want the complexity of calculating D^{-1}. This is a diagonal matrix, of the inverse of the sum of the rows of $\Phi(Q)\Phi(K)^T$. Because of that, the cost of computing D^{-1} is the cost of computing the sum of the rows of $\Phi(Q)\Phi(K)^T$. We know that $(\Phi(Q)\Phi(K)^T)_{i,j} = \phi(q_i)^T\phi(k_j)$, so the cell (i,i) of D is $\sum_{j=1}^{L}\phi(q_i)^T\phi(k_j) = \phi(q_i)^T\sum_{j=1}^{L}\phi(k_j)$. With this, we can compute $\sum_{j=1}^{L}\phi(k_j)$ independently and store it. Having done that, we then compute $\phi(q_i)^T\sum_{j=1}^{L}\phi(k_j)$ for each row i of $\Phi(Q)$, which has complexity $O(LM)$. Since the last thing we need to do is to invert the diagonal matrix, and the complexity of this is $O(L)$, the complexity of calculating D^{-1} is $O(LM)$.

Now, to avoid a complexity that is squared in L, we need to calculate the matrix products in a different order. We know that the dimensions of each matrix in the approximation are: $L \times L$ for D^{-1}, $L \times M$ for $\Phi(Q)$, $M \times L$ for $\Phi(K)^T$ and $L \times D$ for V. Since we want a complexity that is linear in L, we need to do the matrix products in the order: $\Phi(K)^T \times V$, $\Phi(Q) \times (\Phi(K)^T \times V)$ and $D^{-1} \times (\Phi(Q) \times (\Phi(K)^T \times V))$. Let's compute the complexity of each matrix product.

Since $\Phi(K)^T$ is $M \times L$ and V is $L \times D$, the complexity of $\Phi(K)^T \times V$ is $O(MLD)$. The result of this product is a matrix of size $M \times D$.

The second product has $\Phi(Q)$, which is $L \times M$, and the result of the previous product, which is $M \times D$. So the complexity of this product is $O(LMD)$. The result of this product is a matrix of size $L \times D$.

The last product has D^{-1}, which is $L \times L$, and the result of the previous product, which is $L \times D$ and so, the complexity of this product is $O(L^2D)$. To avoid a complexity that is squared in L, we cannot do this product. Since D^{-1} is a diagonal matrix, in reality, we don't need to do this product, but only need to divide each row of the previous product by the corresponding cell of D^{-1}. Due to that, the complexity of this operation is $O(LD)$, since the division is $O(1)$ and there are $L \times D$ cells to divide.

Finally, the overall complexity of the approximation is $O(LM)+O(MLD)+O(LMD)+O(LD) = O(LMD)$, which is linear in L.

2 Question 2

2.1 Question 2.1

After running the code, the best configuration was for the learning rate of 0.01. The following plots were generated:

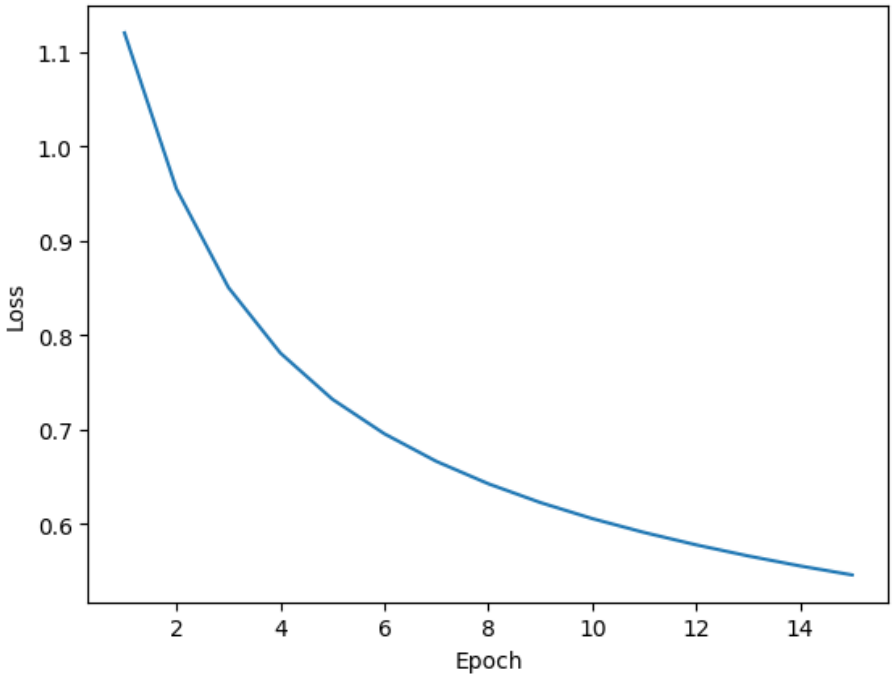

Figure 1: Training loss for $\eta = 0.01$

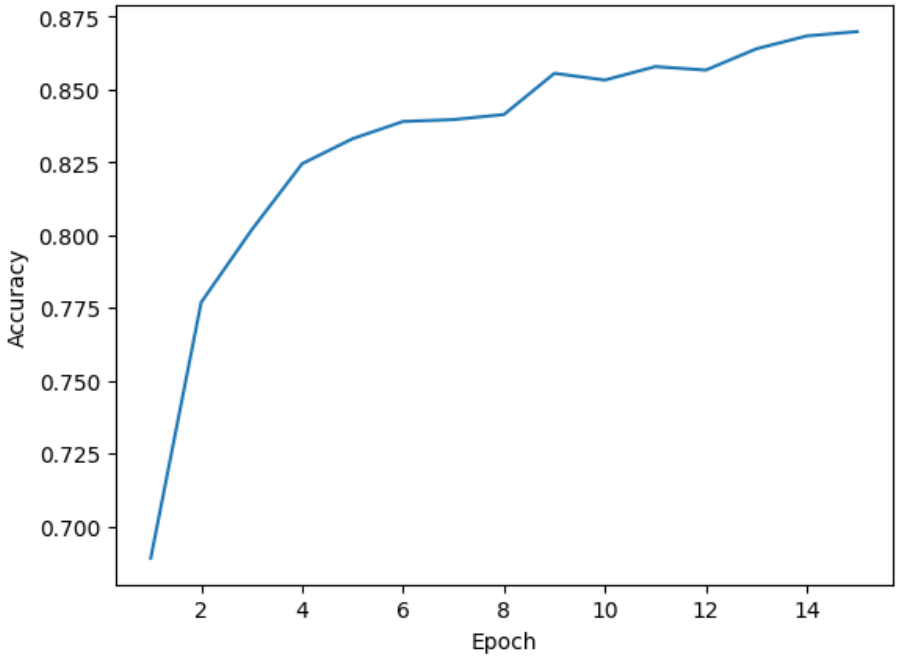

Figure 2: Validation accuracy for $\eta = 0.01$

The final test accuracy was 0.8280.

2.2 Question 2.2

Using the best learning rate from the previous question ($\eta = 0.01$), the performance of this network was slightly worse, having achieved a final test accuracy of 0.8147.

2.3 Question 2.3

Both networks present the same number of parameters, 224892. The difference in performance between the two networks, resides in the use of max pooling layers. Max pooling can help the network focusing on the most important features, making the network more robust to small changes in the input. Furthermore, max pooling can also help with overfitting. In our case, the use of max pooling layers helped the network to achieve a better test accuracy results.

3 Question 3

3.1 Question 3.1

After running the code, the following plots were generated:

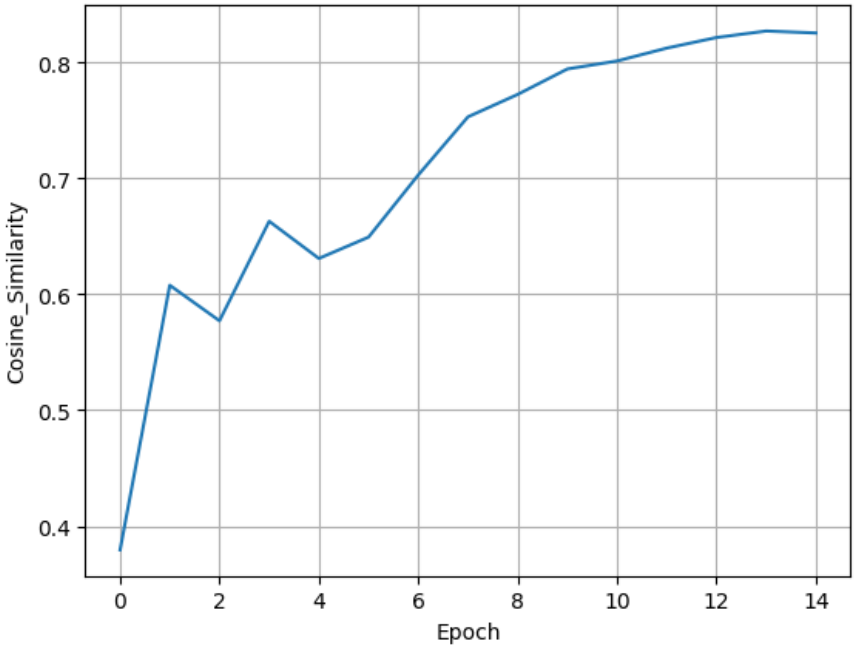

Figure 3: Cosine similarity

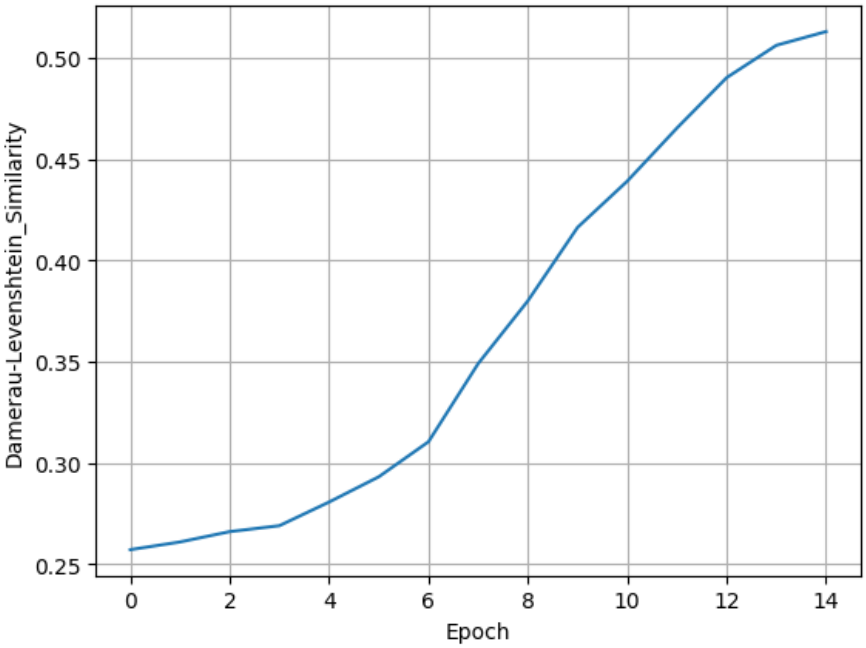

Figure 4: Damereau-Levenshtein similarity

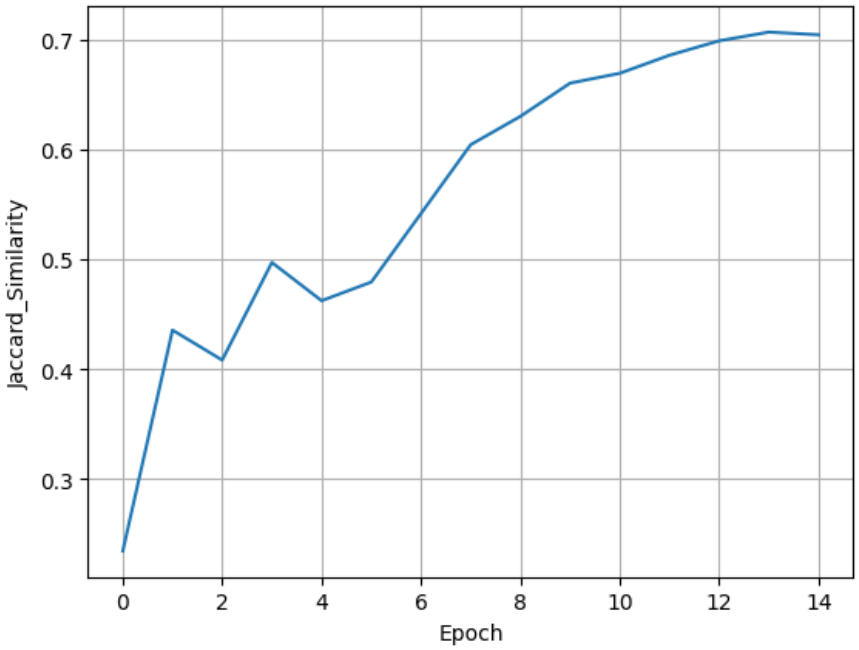

Figure 5: Jaccard similarity

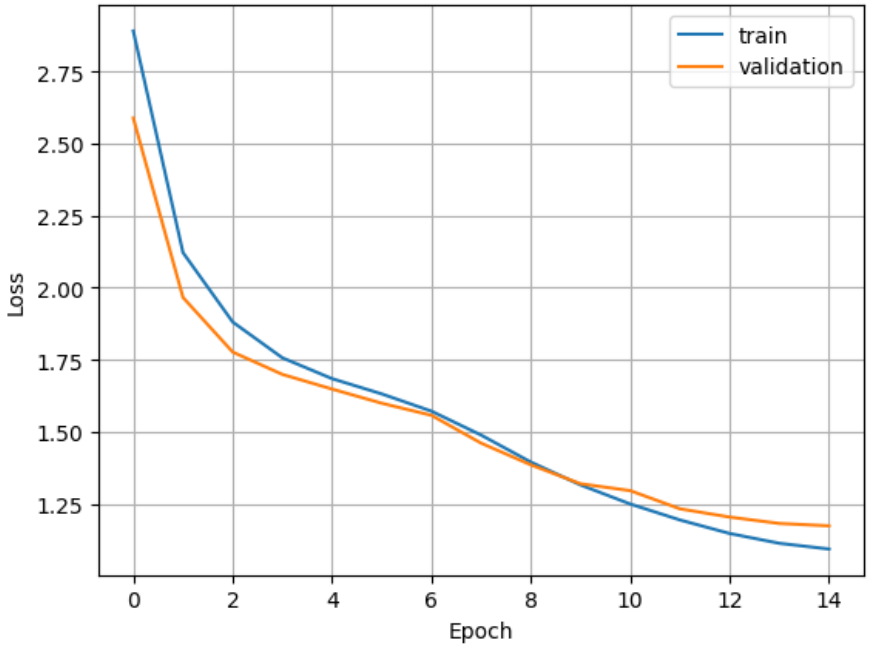

Figure 6: Validation and training loss

In the test set, the jaccard similarity was 0.715, the cosine similarity was 0.832 and the damereau-levenshtein similarity was 0.509. The final test loss was 1.183.

3.2 Question 3.2

After running the code, the following plots were generated:

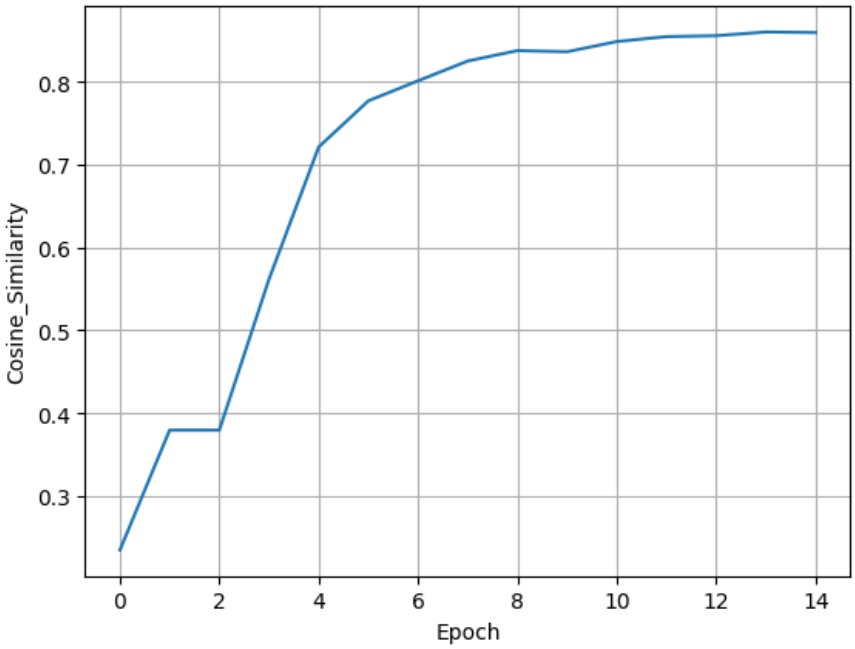

Figure 7: Cosine similarity

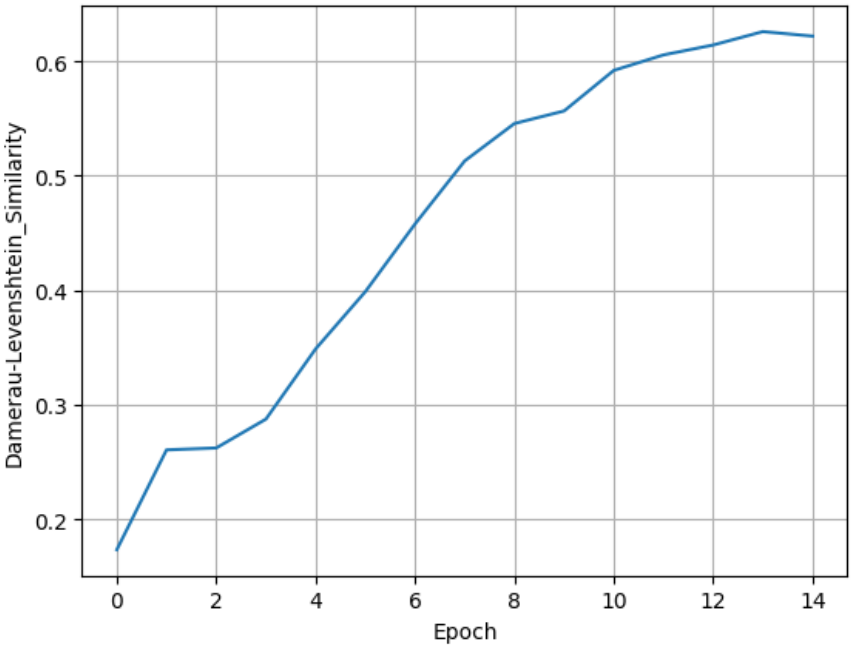

Figure 8: Damereau-Levenshtein similarity

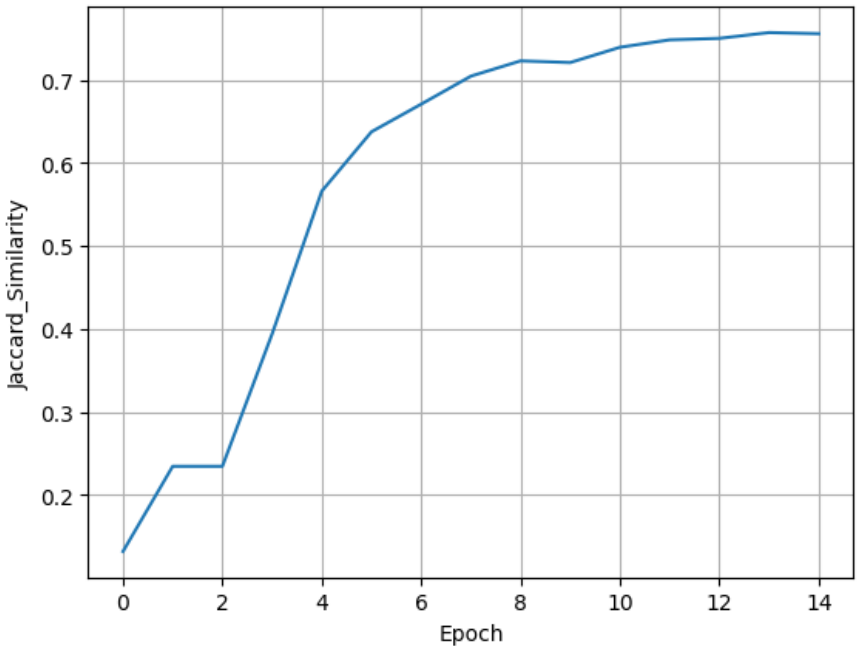

Figure 9: Jaccard similarity

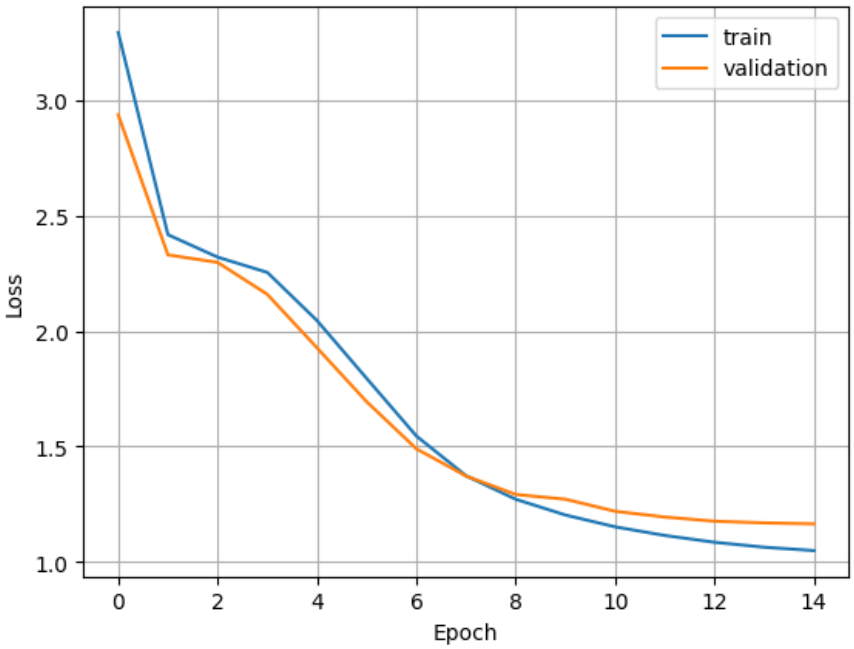

Figure 10: Validation and training loss

In the test set, the jaccard similarity was 0.764, the cosine similarity was 0.865 and the damereau-levenshtein similarity was 0.632. The final test loss was 1.161.

3.3 Question 3.3

To compare the performance of our two models, it's important to understand their distinct approaches in processing text input. The LSTM model, processes the text input token by token, maintaining a hidden state that captures and carries relevant information, so that the model can keep track of long term dependencies within the sequence. The model with the attention mechanism, on the other hand, processes the text input as a whole, assigning weights to different parts of the text, which allows the model to prioritize specific segments of the input.

After examining the performance of our models, we see that the LSTM model starts with a lower initial loss, indicating a better initial understanding of the sequential data. In contrast, the model with the attention mechanism, starts with a higher loss and has its decrease accentuated, after the third epoch, where the difference between the losses of both models reaches its max value. While the LSTM model has a more stable decrease in loss, the model with the attention mechanism has a more accentuated decrease in loss, which might indicate that, after the third epoch, the model with the attention mechanism has assigned appropriately the weights and so, it achieves a better understanding of the input sequence.

Lastly, we can also look at the predicted output of both models, to see how they performed. The LSTM model, generates predictions where individual words appear more coherent in isolation. In contrast, the model with the attention mechanism, generates word predictions that are more similar to the target, but don't make sense individually. This is likely due to the attention mechanism's capacity to focus on different parts of the input sequence and draw correlations that, in this case, similarity prioritize similarity over standalone word coherence.

3.4 Question 3.4

In our program, we analyse three types of similarity metrics: Jaccard, cosine, and Damerau-Levenshtein. The Jaccard similarity, evaluates proportion of unique words shared between the

two texts, relative to the total unique words in both. The cosine similarity, on the other hand, measures the similarity in the orientation of the word frequency vectors. Lastly, the Damerau-Levenshtein similarity calculates the number of operations (insertions, deletions, substitutions, and transpositions) needed to change one string into another.

Comparing both models, we can verify that the model with the attention mechanism, had greater performance in all three metrics, when compared to the LSTM model. For the two models, we can verify that both the Jaccard and cosine similarities had great increases in performance, when compared to their baseline scores. The Damerau-Levenshtein similarity, also had an increase in performance, but not as significant as the other two.

The difference in Damerau-Levenshtein similarity, between the two models, can be explained by the fact that, while the LSTM model, generates predictions that are more coherent in isolation, the model with the attention mechanism, generates predictions that are more similar to the target, but don't make sense individually. Overall, the fact that the this metric is not as high as the other two, indicates that the models still struggle with character level predictions.

4 Credits

Each member contributed to the following questions:

- Sebastião Carvalho
  - Question 1.1, 1.2, 1.3 and 1.4
  - Question 2.1
- Frederico Silva
  - Question 2.2 and 2.3
  - Question 3.1, 3.2, 3.3 and 3.4

The divison was done to ensure that each member contributed equally to the project.

5 Sources

- Multinomial theorem
- Stars and bars (combinatorics)
- What is a max pooling layer in CNN?
- What exactly is the difference between LSTM and attention in neural networks?
- A simple overview of RNN, LSTM and Attention Mechanism
- What are the pros and cons of using cosine similarity vs. Jaccard similarity for text analysis?
- The Damerau-Levenshtein Distance: a Powerful Measure for String Similarity