INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Informática

Deep Learning (Dei)

MEIC 2023-2024 – 2nd Period

Homework 2

Pedro Mateus 99306; Guilherme Leitão 99951

Pedro Mateus	***Q1 - 3, 4; Q3 - 1, 2, 3, 4***
Guilherme Leitão	***Q1 - 1, 2, 4; Q2 - 1, 2, 3***

Question 1

1.

The computation $\boldsymbol{Z=Softmax(QK^T)V}$ in the context of a self-attention layer for a transformer model involves three steps:

1. Computing the product $\boldsymbol{QK^T}$, which results in an $\boldsymbol{L\times L}$ matrix;
    - The complexity is $\boldsymbol{O(L^2D)}$ - for each of the $\boldsymbol{L\times L}$ entries in the resulting matrix, we compute a product of vectors of dimension $\boldsymbol{D}$.
2. Applying the *Softmax* function to each row of the resulting $\boldsymbol{L\times L}$ matrix;
    - Here the computational complexity is $\boldsymbol{O(L^2)}$ - since the *Softmax* function is applied to $\boldsymbol{L}$ entries for each of the $\boldsymbol{L}$ rows.
3. Multiplying the Softmax results by $\boldsymbol{V}$, which is an $\boldsymbol{L\times D}$ matrix.
    - The complexity here is $\boldsymbol{O(L^2D)}$ - as it is a simple matrix multiplication between an $\boldsymbol{L\times L}$ and an $\boldsymbol{L\times D}$ matrices.

Therefore, the total complexity for computing $\boldsymbol{Z}$ is $\boldsymbol{O(L^2D + L^2 + L^2D) = O(L^2D) = O(L^2)}$ (since D is a constant).

This becomes problematic for long sequences because the number of computations grows quadratically with the sequence length $\boldsymbol{L}$. This means that for very long sequences, the amount of computation (in terms of memory and time) can become impractically large.

2.

For a feature map $\phi: R^D \rightarrow R^M$, if we apply this approximation, then for any vectors $q, k \in R^D$, the feature map of the product $q^T k$ is approximated as:

$$exp(q^T k) \approx 1 + q^T k + \frac{1}{2}(q^T k)^2$$

- The 1st term ***1*** contributes as a constant;
- The 2nd term $q^T k$ is linear (since they both $\in R^D$) and requires ***D*** dimensions (***D*** multiplications);
- Finally, the 3rd term $(q^T k)^2$ is quadratic , which would imply a dimensionality of $\frac{D(D+1)}{2}$:
    - the resulting matrix formed by all possible products $q_i k_j$ for $i, j = 1, \ldots, D$ is symmetric since $q_i k_j = q_j k_i$;
    - therefore, you only need to calculate the number of unique non-diagonal elements for one side (due to symmetry), resulting in a total of $\frac{D(D-1)}{2}$ multiplications;
    - when you add the diagonal multiplications ***D*** (since there are ***D*** elements in the diagonal), you get $D + \frac{D(D-1)}{2} = \frac{2D + D(D-1)}{2} = \frac{2D - D + D^2}{2} = \frac{D(D+1)}{2}$.

So the total dimensionality ***M*** for the feature space would be:

$$M = 1 + D + \frac{D(D+1)}{2}$$

If we were to use $K = 4$ terms in the McLaurin series expansion, the fourth term $\frac{t^3}{6}$ would introduce a cubic dimensionality, further increasing it. We would calculate the dimensionality via an identical thought process as the aforementioned one, reaching the following:

$$M = 1 + D + \frac{D(D+1)}{2} + \frac{D(D+1)(D+2)}{6}$$

As you can see, the dimensionality of the feature space follows a combinatorial pattern based on the number of terms ***K*** used in the series expansion.

As such, the dimensionality if we were to use $K \geq 3$ is given by the expression:

$$M = 1 + D + \sum_{i=3}^{K} \binom{D+K-2}{K-1}$$

3.

Q1-3 Q - query vector
k - key vector
v - value vector

A self-attention operation is given by $Z = \text{softmax}(Q \cdot K^T) \cdot V$, therefore we need to prove that $D^{-1}\phi(Q) \cdot \phi(K^T) \approx \text{softmax}(Q \cdot K^T)$

$$\phi(Q) = \begin{bmatrix} \phi(q_1)- \\ \phi(q_2)- \\ \vdots \\ \phi(q_L)- \end{bmatrix}, \quad \phi(K) = \begin{bmatrix} \phi(k_1)- \\ \phi(k_2)- \\ \vdots \\ \phi(k_L)- \end{bmatrix}; \quad 1_L = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$\phi(Q) \cdot \phi(K)^T = \begin{bmatrix} \sum_{i,j} \phi(q_1)_i \phi(k_1)_j & \sum \phi(q_1)_i \phi(k_2)_j & \cdots & \sum \phi(q_1)_i \phi(k_L)_j \\ \vdots & & & \vdots \\ \sum_i \phi(q_L)_i \phi(k_1)_i & \sum_i \phi(q_L)_i \phi(k_2)_i & \cdots & \sum_i \phi(q_L)_i \phi(k_L)_i \end{bmatrix} =$$

$$= \begin{bmatrix} \phi(q_1)^T \cdot \phi(k_1) & \phi(q_1)^T \phi(k_2) & \cdots & \phi(q_1)^T \cdot \phi(k_L) \\ \vdots & & & \vdots \\ \phi(q_L)^T \phi(k_1) & \phi(q_2)^T \cdot \phi(k_2) & \cdots & \phi(q_L)^T \cdot \phi(k_L) \end{bmatrix} \approx$$

$$\approx \begin{bmatrix} \exp(q_1^T \cdot k_1) & \exp(q_1^T \cdot k_2) & \cdots & \exp(q_1^T \cdot k_L) \\ \vdots & & & \vdots \\ \exp(q_L^T \cdot k_1) & \exp(q_L^T \cdot k_2) & \cdots & \exp(q_L^T \cdot k_L) \end{bmatrix}$$

$$\phi(Q) \cdot \phi(K)^T \cdot 1_L = \begin{bmatrix} \sum_j^L \exp(q_1^T \cdot k_j) \\ \sum_j^L \exp(q_2^T \cdot k_j) \\ \vdots \\ \sum_j^L \exp(q_L^T \cdot k_j) \end{bmatrix}$$

$$D = \begin{bmatrix} \sum_j^L \exp(q_1^T \cdot k_j) & & & \\ & \sum_j^L \exp(q_2^T \cdot k_j) & & \\ & & \ddots & \\ & & & \sum_j^L \exp(q_L^T \cdot k_j) \end{bmatrix}$$

$$D^{-1} = \begin{bmatrix} \frac{1}{\sum_j^L \exp(q_1^T \cdot k_j)} & & & \\ & \frac{1}{\sum_j^L \exp(q_2^T \cdot k_j)} & & \\ & & \ddots & \\ & & & \frac{1}{\sum_j^L \exp(q_L^T \cdot k_j)} \end{bmatrix}$$

$$D^{-1}\cdot\phi(Q)\cdot\phi(K)^T = \begin{bmatrix} \frac{\exp(q_1^T\cdot k_1)}{\sum_j \exp(q_1^T\cdot k_j)} & \frac{\exp(q_1^T\cdot k_2)}{\sum_j \exp(q_1^T\cdot k_j)} & \cdots & \frac{\exp(q_1^T\cdot k_L)}{\sum_j \exp(q_1^T\cdot k_j)} \\ \vdots & & & \vdots \\ \frac{\exp(q_L^T\cdot k_1)}{\sum_j \exp(q_L^T\cdot k_j)} & \frac{\exp(q_L^T\cdot k_2)}{\sum_j \exp(q_L^T\cdot k_j)} & \cdots & \frac{\exp(q_L^T\cdot k_L)}{\sum_j \exp(q_L^T\cdot k_j)} \end{bmatrix}$$

$$Q = \begin{bmatrix} q_1 - \\ q_2 - \\ \vdots \\ q_L - \end{bmatrix} \; ; \; K = \begin{bmatrix} k_1 - \\ k_2 - \\ \vdots \\ k_L - \end{bmatrix}$$

$$Q\cdot K^T = \begin{bmatrix} \sum_i q_{1i}k_{1i} & \sum_i q_{1i}k_{2i} & \cdots & \sum_i q_{1i}k_{Li} \\ \vdots & & & \vdots \\ \sum_i q_{Li}k_{1i} & \cdots & & \sum_i q_{Li}k_{Li} \end{bmatrix} = \begin{bmatrix} q_1^T\cdot k_1 & q_1^T\cdot k_2 & \cdots & q_1^T\cdot k_L \\ \vdots & & & \vdots \\ q_L^T\cdot k_1 & \cdots & & q_L^T\cdot k_L \end{bmatrix}$$

$$\text{softmax}(Q\cdot K^T) = \begin{bmatrix} \frac{\exp(q_1^T\cdot k_1)}{\sum_j \exp(q_1^T\cdot k_j)} & \frac{\exp(q_1^T\cdot k_2)}{\sum_j \exp(q_1^T\cdot k_j)} & \cdots & \frac{\exp(q_1^T\cdot k_L)}{\sum_j \exp(q_1^T\cdot k_j)} \\ \vdots & & & \vdots \\ \frac{\exp(q_L^T\cdot k_1)}{\sum_j \exp(q_L^T\cdot k_j)} & \frac{\exp(q_L^T\cdot k_2)}{\sum_j \exp(q_L^T\cdot k_j)} & \cdots & \frac{\exp(q_L^T\cdot k_L)}{\sum_j \exp(q_L^T\cdot k_j)} \end{bmatrix} \approx$$

$$\approx D^{-1}\cdot\phi(Q)\cdot\phi(K)^T$$

We can conclude that $\text{softmax}(Q\cdot K^T) \approx D^{-1}\phi(Q)\phi(K)^T$, so we can approximate the self-attention operation $Z \approx D^{-1}\phi(Q)\phi(K)^T\cdot V$

4.

The following problem is an optimization problem concerning the most efficient way to multiply a given sequence of matrices, by deciding the sequence of the matrix multiplications involved.

Given $\phi(Q) \in R^{L \times M}$, $\phi(K) \in R^{L \times M}$ and $V \in R^{L \times D}$: $Z \approx D^{-1}\phi(Q)\phi(K)^T V$, where $D = Diag(\phi(Q)\phi(K)^T 1_L)$, we can see that by applying the matrix multiplication from the right to the left we can reduce the amount of steps necessary to compute Z.

- We can compute $\phi(Q)$ and $\phi(K)$ by applying $\phi()$ to each element of Q and k, this process will have a total complexity of $O(LM + LM) = O(2LM) = O(LM)$;
- By first calculating $\phi(K)^T V$ we can obtain a total complexity of $O(LM + LMD) = O(LMD)$;
- The second step is to calculate $\phi(Q)\phi(K)^T V$ this multiplication will have a total complexity of $O(LMD + LM + LMD) = O(2LMD) = O(LMD)$;
- For the term D^{-1}:
    - We can compute $\phi(K)^T 1_L$ with a total complexity of $O(LM + LM) = O(2LM) = O(LM)$;
    - $\phi(Q)\phi(K)^T 1_L$ will have a total complexity of $O(LM + LM) = O(LM)$;
    - Calculating the $Diag(\phi(Q)\phi(K)^T 1_L)$ will have a total cost of $O(LM + L) = O(LM)$;
    - Since the matrix is diagonal, calculating its inverse is done just by inverting the value of each in the main diagonal so it has a total complexity of $O(L + LM) = O(LM)$;
- The final product $D^{-1}\phi(Q)\phi(K)^T V$ will have a total complexity that is equal to the sum of the complexities of both D^{-1} and $\phi(Q)\phi(K)^T V$ so it's $O(LM + LMD) = O(LMD)$.

As we can see, it also depends linearly on M and D.

Question 2

1.

The model with a lower *η=0.001* had its validation accuracies ranging between *0.4721* and *0.6799* with a final test accuracy of **0.7183**.

The model with *η=0.01* resulted in its validation accuracies ranging between *0.4917* and *0.8792*, with an improved final test accuracy of **0.8431**.

The model with a higher *η=0.1* resulted in its validation accuracies ranging between *0.7572* and *0.8308*, with an improved final test accuracy of **0.8280**.

We can also see from Figures 4 - 6, although not very conclusively, that the training loss for the three rates are somewhat similar, with the *η=0.01* being perhaps a bit smoother and more consistent.

Therefore, we can safely assume that the model with the learning rate of ***η=0.01*** has achieved a better performance.

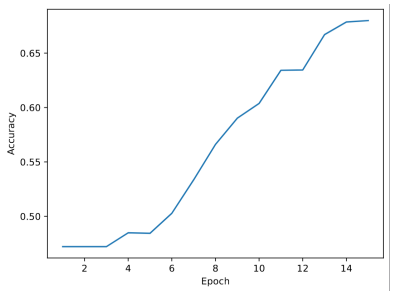


Figure 1:
CNN-accuracy-2-1-0.001.pdf

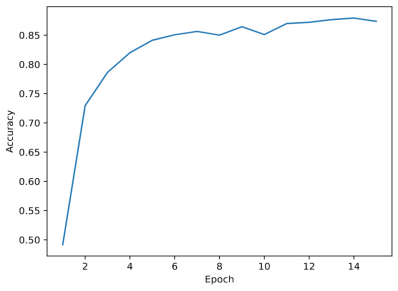


Figure 2:
CNN-accuracy-2-1-0.01.pdf

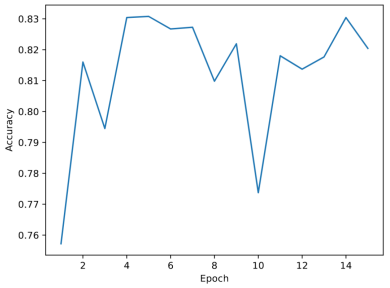


Figure 3:
CNN-accuracy-2-1-0.1.pdf

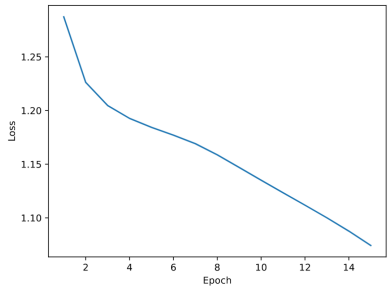


Figure 4:
CNN-loss-2-1-0.001.pdf

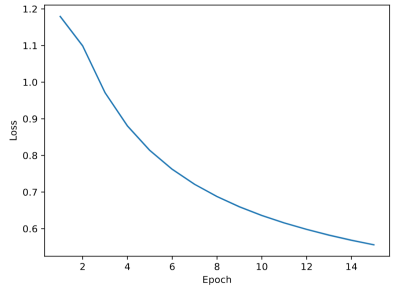


Figure 5:
CNN-loss-2-1-0.01.pdf

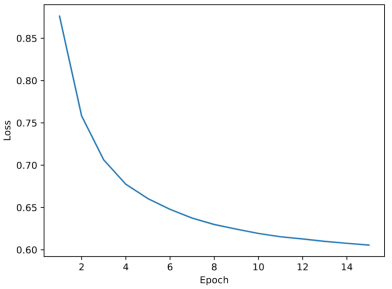


Figure 6:
CNN-loss-2-1-0.1.pdf

2.

With a learning rate *η=0.001*, the validation accuracies ranged between *0.4721* and *0.6713* as shown in Figure 7, with a final test accuracy of **0.6862**. This indicates a moderate level of performance, though less stable compared to the model with max-pooling layers in the previous question.

The model with *η=0.01*, the model achieved better validation accuracies ranging from *0.4786* to *0.8507* as you can see in Figure 8, with a noticeable increase in the final test accuracy to exactly **0.8336**. This result suggests that the model without max-pooling benefits from a slightly higher learning rate in terms of accuracy.

With a learning rate of *η=0.1*, where the validation accuracies were more volatile, with a range from *0.7312* to *0.8123* as depicted in Figure 9 and a final test accuracy of **0.8015**.

By examining Figures 10 - 12, we can see that the training loss for all three learning rates decreases over time. However, the model with a learning rate of *η=0.01* presents a very slightly smoother and more consistent decline in loss.

Consequently, it can be inferred that the model without max-pooling and with an adjusted convolution layer structure performs optimally at a learning rate of *η=0.01*, as it yields a good balance between validation accuracy and training loss consistency. However, it is noteworthy to mention that the previous model with max-pooling layers had an overall higher final test accuracy, suggesting that it might be more suitable.

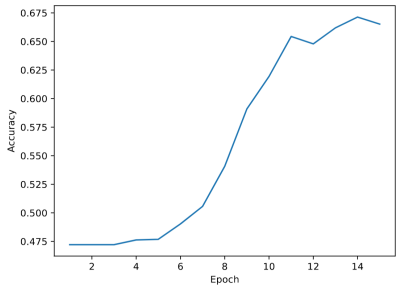


Figure 7:
CNN-accuracy-2-2-0.001.pdf

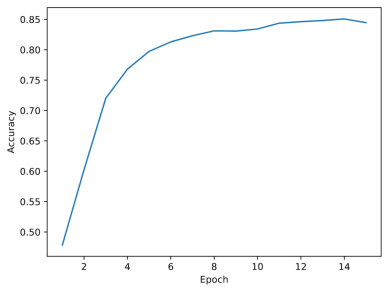


Figure 8:
CNN-accuracy-2-2-0.01.pdf

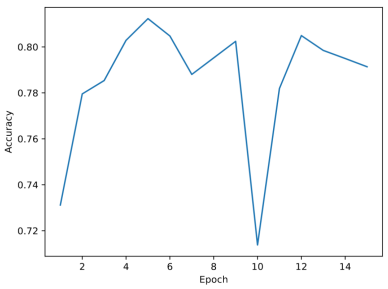


Figure 9:
CNN-accuracy-2-2-0.1.pdf

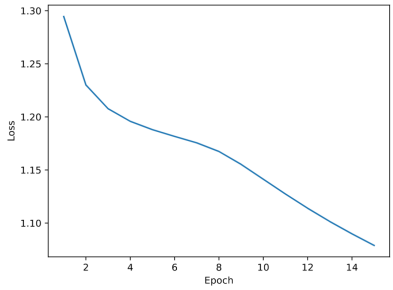


Figure 10:
CNN-loss-2-2-0.001.pdf

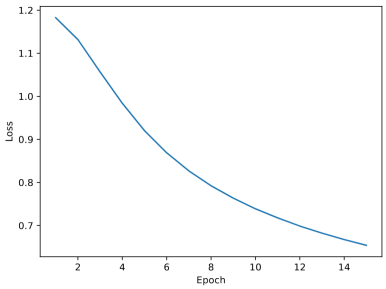


Figure 11:
CNN-loss-2-2-0.01.pdf

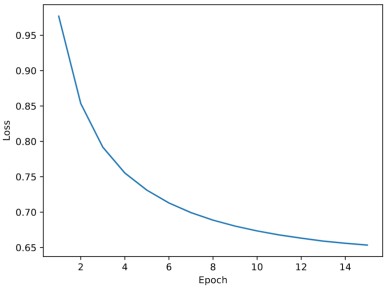


Figure 12:
CNN-loss-2-2-0.1.pdf

3.

The two models, each with 225,618 trainable parameters, demonstrate that replacing max-pooling with a stride-2 convolution, in this case, doesn't impact the parameter count. This is because these particular changes maintain the number of trainable elements.

Despite the same number of parameters, the model with max-pooling layers demonstrated (slightly) superior test accuracy, validating it in improving model performance.

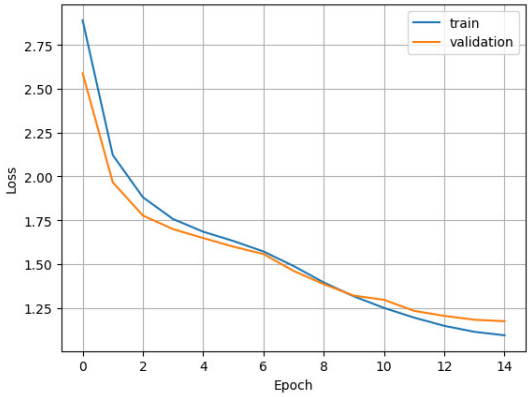
This makes sense as it reduces overfitting by condensing the input's feature representation, which helps the network focus on more relevant features and be less sensitive to small variations.

This process not only simplifies the model but also increases its ability to generalize well to new data.

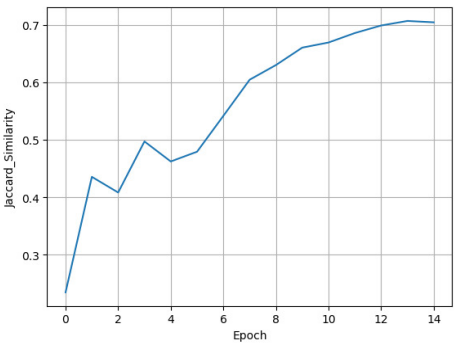
Question 3

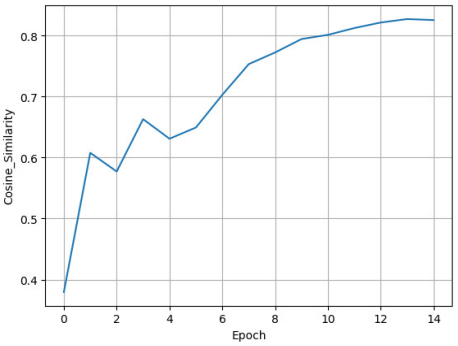
1.

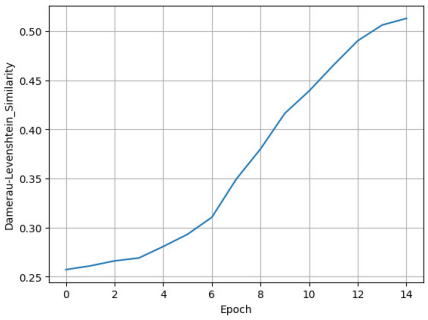
Training and validation loss over time:



String similarity scores over time:







Final test loss and the string similarity scores:

jaccard similarity: 0.7149178437842456,

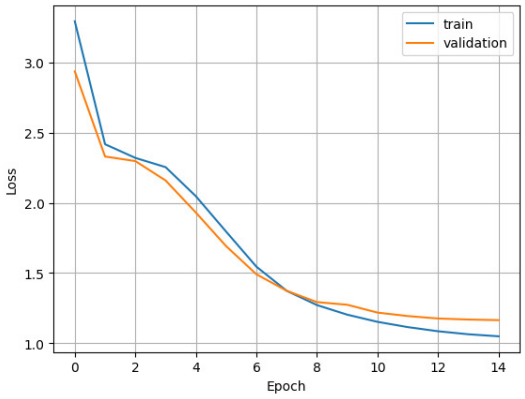
cosine similarity: 0.8324116011552667,

damerau-levenshtein similarity: 0.5087067982887039,

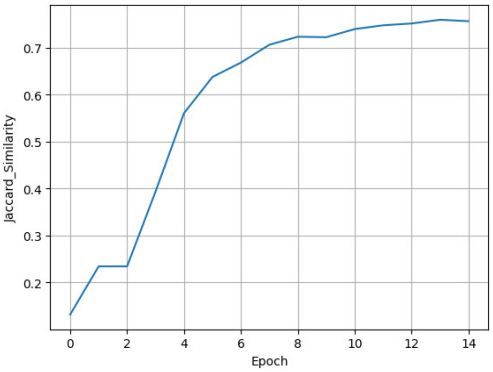
loss: 1.1828282017533371

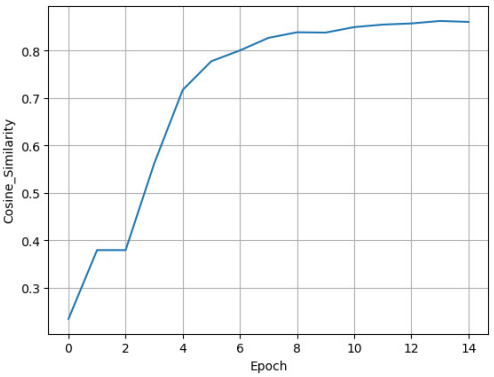
2.

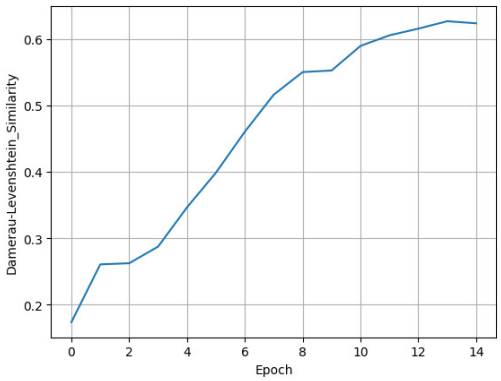
Training and validation loss over time:



String similarity scores over time:







Final test loss and the string similarity scores:

jaccard similarity: 0.7646119664184675,

cosine similarity: 0.8653416548246561,

damerau-levenshtein similarity: 0.6334206621177492,

loss: 1.16039157931397

3.

Both the LSTM and the attention mechanism are sequence-to-sequence translation models, however there are some differences in the way they process text.

Starting by the way the encoder works, the LSTM model converts a variable size text sequence into a fixed size vector, whereas in the attention model the text sequence is stored in a matrix with fixed number of rows and variable size of columns. In addition, in the attention model it's also calculated the input conditioning state (**c**), which is basically a parameter that gives different weights to the features, allowing for the model to focus more on a subset of the features.

In the decoder the LSTM is a simple RNN that calculates the translation of the next word based on the previous result. In the attention model the decoder is also a variant of an RNN as the LSTM, however the next word is translated based on the translation of the previous and on the conditioning state.

Therefore when comparing both the LSTM and the attention model, we can conclude that the attention mechanism is able to focus on special words within a sentence and focus more on them in order to build a sentence.

By looking at the graphs and final results, we can see that both of the methods implemented are similar in terms of final performance, achieving similar final test and validation loss values, however the attention model has a steeper curve than the LSTM model, which may indicate that it starts to converge sooner than the LSTM.

It's worth mentioning that since we are dealing with small audio clips the advantage of the attention model in using a matrix to store the text sequence is not noticeable.

4.

Each of the similarity values differ in the way they are calculated, which explains why they reach different values.

The Jaccard coefficient measures similarity between the model output and the expected, by dividing the size of the intersection of both sets by the size of the union.

Considering both the model output and the expected output as vectors, the cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths.

Damerau-Levenshtein distance between two sentences is the minimum number of operations (consisting of insertions, deletions or substitutions of a single character, or transposition of two adjacent characters) required to change one sentence into the other, in our case it's the minimum number of operations required to change the sentence the model has produced into the expected value.

The cosine similarity takes more into consideration the overall similarity of the documents, whereas in the Jaccard similarity focus more in the similarity of the words present in each sentence, the Damerau-Levenshtein is similar to the cosine similarity, however it also takes into account the order of the words in the sentence.

In both of the models we can see that the similarity scores are ordered the same way

(1 - cosine, 2 - jaccard, 3 - Damerau-Levenshtein).