Deep Learning 2023

Homework 2 – Group 48

Miguel Vale (99113) Pedro Lobo (99115)

Contribution

We started by dividing the coding part of this homework by the two of us. Questions 2 was done by Pedro while question 3 was done by Miguel. Although we distributed this questions between the both of us, we double-checked each other's implementation.

Question 1 was tackled by the two of us.

The report was written by the two of us, with each of us writing the answers to the questions that we implemented and then reviewed by the other.

Question 1

1. *Consider the self-attention layer of a transformer with a single attention head, which performs the computation* $\mathbf{Z} = Softmax(\mathbf{QK^T})\mathbf{V}$*, where* $\mathbf{Q} \in \mathbb{R}^{L\times D}$*,* $\mathbf{K} \in \mathbb{R}^{L\times D}$*,* $\mathbf{V} \in \mathbb{R}^{L\times D}$*, for a long sequence length* L *and hidden size* D*. What is the computation complexity, in terms of* L*, of computing* $\mathbf{Z}$*? Briefly explain why this could be problematic for long sequences.*

   Each line of the matrix $\mathbf{Q}$ is multiplied by each line of the matrix $\mathbf{K}$, with each multiplication being a operation of size D. So, the total cost of the matrix multiplication is $O(L \times L \times D)$.

   Then, the softmax function is applied to each element of the resulting matrix, which has size $L \times L$.

   Finally, the matrix $\mathbf{V} \in \mathbb{R}^{L\times D}$ is multiplied by the resulting matrix $\in \mathbb{R}^{L\times D}$, which as a cost of $O(L^2 D)$.

   So, the total cost of computing $\mathbb{Z}$ is $O(L^2 D)$.

   This is problematic for long sequences because the cost of computing $\mathbb{Z}$ grows quadratically with the length of the sequence, which means that the time and space required to compute $\mathbb{Z}$ for longer sequences grows very quickly.

2. *We will show that by suitably approximating the softmax transformation the computational complexity above can be reduced. Consider the McLaurin series expansion for the exponential function,*

$$exp(t) = \sum_{n=0}^{\infty} \frac{t^n}{n!} = 1 + t + \frac{t^2}{2!} + \frac{t^3}{6} + \dots$$

Using only the first three terms in this series, we obtain $exp(t) \approx 1 + t + \frac{t^2}{2}$. *Use this approximation to obtain a feature map* $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ *such that, for arbitrary* $\boldsymbol{q} \in \mathbb{R}^D$ *and,* $\boldsymbol{k} \in \mathbb{R}^D$, *we have* $exp(\boldsymbol{q}^T\boldsymbol{k}) \approx \phi(\boldsymbol{q})^T\phi(\boldsymbol{k})$. *Express the dimensionality of the feature space* M *as a function of* D. *What would be the dimensionality if you used* $K \geq 3$ *terms in the McLaurin series expansion (as a function of* D *and* K*)?*

$M = 1 + D + \frac{D\times(D-1)}{2}$, where 1 is a constant term, D is linear to the number of features and $\frac{D\times(D-1)}{2}$ is the number of possible combinations of two features, only excluding the combinations of a feature with itself.

If $K \geq 3$, t^{k-1} will include all the features up to $k-1$, so

$$M = \sum_{i}^{k-1} \binom{D+k-1}{k} \tag{1}$$

Therefore,

$$M = \sum_{i}^{k-1} \frac{(D+i-1)!}{i!(D-1)!} = 1 + \frac{D!}{(D-1)!} \tag{2}$$

3. *Using the approximation* $exp(\boldsymbol{q}^T k) \approx \phi(\boldsymbol{q})^T\phi(\boldsymbol{k})$, *and denoting by* $\Phi(\boldsymbol{Q}) \in \mathbb{R}^{L\times M}$ *and* $\Phi(\boldsymbol{K}) \in \mathbb{R}^{L\times M}$ *the matrices whose rows are (respectively)* $\phi(\boldsymbol{k}_i)$, *where* $\boldsymbol{q}_i$ *and* $\boldsymbol{k}_i$ *denote (also respectively) the* i^{th} *rows of the original matrices* $\boldsymbol{Q}$ *and* $\boldsymbol{K}$, *show that the self-attention operation can be approximated as* $\mathbf{Z} \approx \mathbf{D}^{-1}\Phi(\boldsymbol{Q})\Phi(\boldsymbol{K})^T\mathbf{V}$, *where* $\mathbf{D} = \boldsymbol{Diag}(\Phi(\boldsymbol{Q})\Phi(\boldsymbol{K})^T\mathbf{1}_L)$ *(here,* $\boldsymbol{Diag}(v)$ *denotes a matrix with the entries of vector* v *in the diagonal, and* $\mathbf{1}_L$ *denotes a vector of of ones with size of length* L*).*

As vector v is the product between $\Phi(\mathbf{Q})$ and $\Phi(\mathbf{K})^T$, it is a vector where each entry is the sum of the correspondent line in $\Phi(\mathbf{Q})\Phi(\mathbf{K})$.

The diagonal matrix $\mathbf{Diag}(v) \in \mathbb{R}^{L\times L}$ is a matrix with the vector v in the diagonal. As the matrix is diagonal, its inverse can be calculated by taking the inverse of each of the elements in the diagonal. So, $D^{-1}\Phi(\mathbf{Q})\Phi(\mathbf{K})^T$ will work as a softmax function, as the

inverse of the sum of the elements in each line of the matrix will be multiplied by the correspondent line of $\Phi(\mathbf{Q})\Phi(\mathbf{K})^T$.

$$\text{Softmax}(\mathbf{QK}^T)\mathbf{V} = \frac{e^{\mathbf{QK}^T}}{\sum e^{\mathbf{QK}^T}} \cdot \mathbf{V} \approx D^{-1}\Phi(\mathbf{Q})\Phi(\mathbf{K})^T\mathbf{V} \tag{3}$$

4. *Show how we can exploit the above approximation to obtain a computational complexity which is linear in L. How does the computational complexity depend on M and D?*

   $\mathbf{Q} \in \mathbb{R}^{L\times D}$ and $\phi : \mathbb{R}^D \to \mathbb{R}^M$, so $\phi(\mathbf{q}_i) \mapsto \Phi(\mathbf{Q}) \in \mathbb{R}^{L\times M}$, which means that the computational complexity is in the order of $O(L \times D \times M)$, which is linear in L.

   As $M << D$, the computational cost of $\mathbb{O}(L^2M)$ is much smaller than $\mathbb{O}(L^2D)$.

Question 2

Image classification with CNNs

1. *Implement a simple convolutional network. Train your model for 15 epochs using SGD tuning only the learning rate on your validation data, using the following values:* 0.1, 0.01, 0.001. *Report the learning rate of best configuration and plot two things: the training loss and the validation accuracy, both as a function of the epoch number.*

   The best configuration, in terms of test accuracy, has a learning rate of $\eta = 0.01$, as shown in Table 1.

Learning Rate	Test Accuracy
0.1	0.7864
0.01	0.8318
0.001	0.7864

Table 1: Final CNN test accuracies for each learning rate.

   The training loss and validation accuracy for the best configuration are plotted in Figures 1 and 2, respectively.

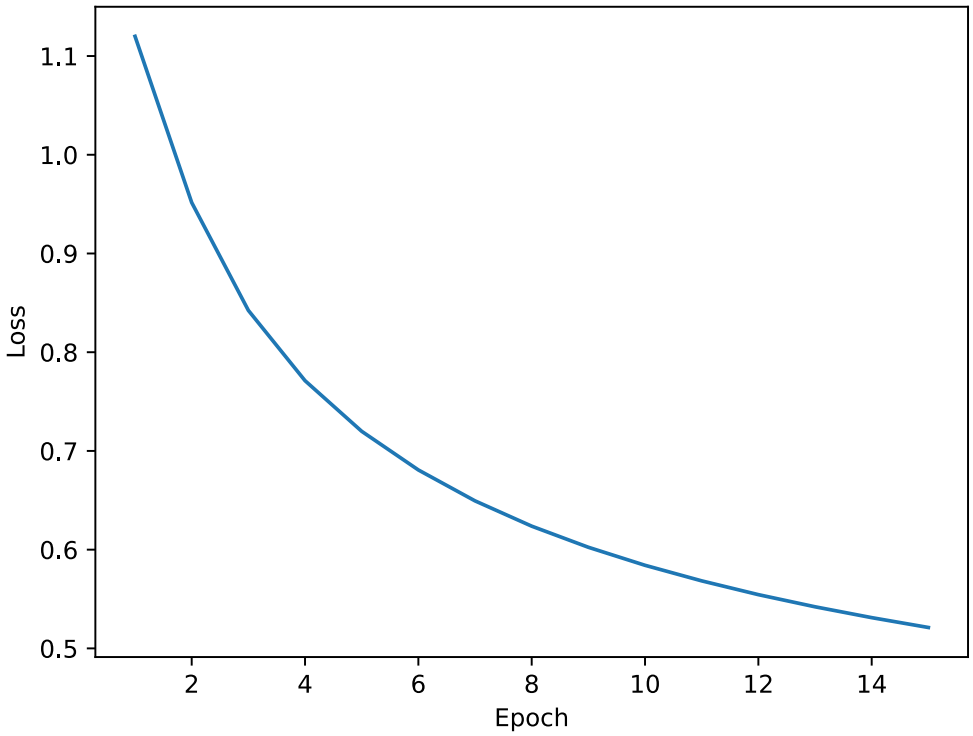

Figure 1: Training loss as a function of the epoch number for $\eta = 0.01$.

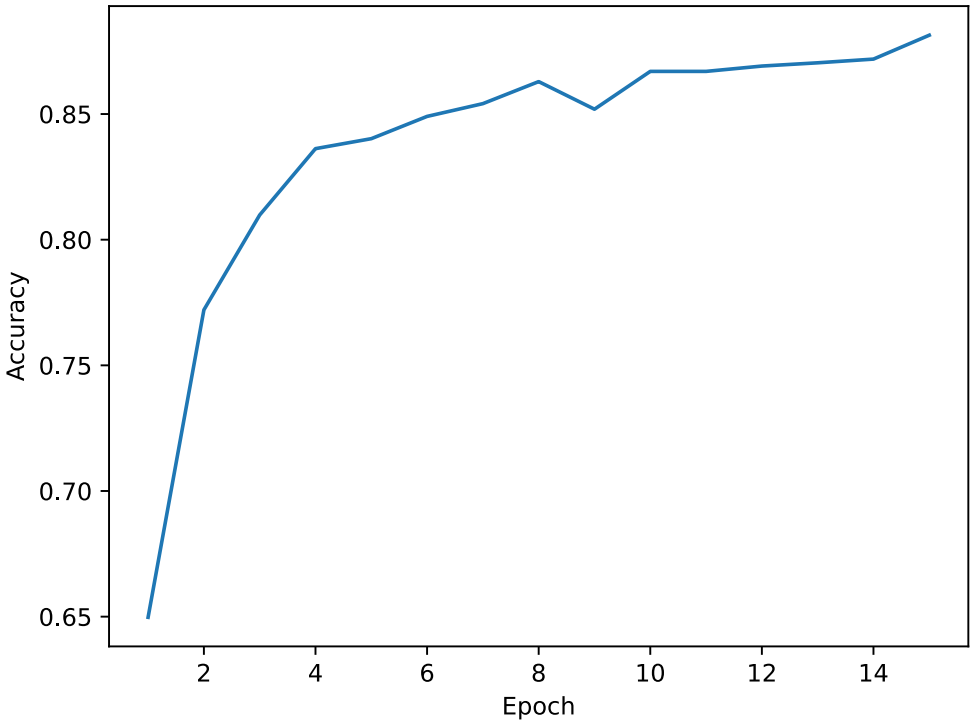

Figure 2: Validation accuracy as a function of the epoch number for $\eta = 0.01$.

2. *Implement and asses a similar network but where the max-pooling layers were re-*

moved and `self.conv1` *and* `self.conv2` *are different.* `self.conv1` *is a convolution layer with 8 output channels, a kernel of size 3x3, padding of 1 and stride of 2, and* `self.conv2` *is a convolution layer with 16 output channels, a kernel of size 3x3, padding of 0, and stride of 2. Modify the* `__init__` *and the forward functions to use the* `no_maxpool` *variable and ensure the ability to switch between current and previous definitions of the* `self.conv1` *and* `self.conv2` *layers and application or not of the max-pooling layer. Report the performance of this network using the optimal hyper-parameters defined in the previous question.*

The best configuration, with the pooling layers removed, in terms of test accuracy, has a learning rate of $\eta = 0.01$, as shown in Table 2.

Learning Rate	**Test Accuracy**
0.1	0.7580
0.01	0.7958
0.001	0.7127

Table 2: Final CNN (with no pooling layers) test accuracies for each learning rate.

The training loss and validation accuracy for the best configuration without pooling layers are plotted in Figures 3 and 4, respectively.

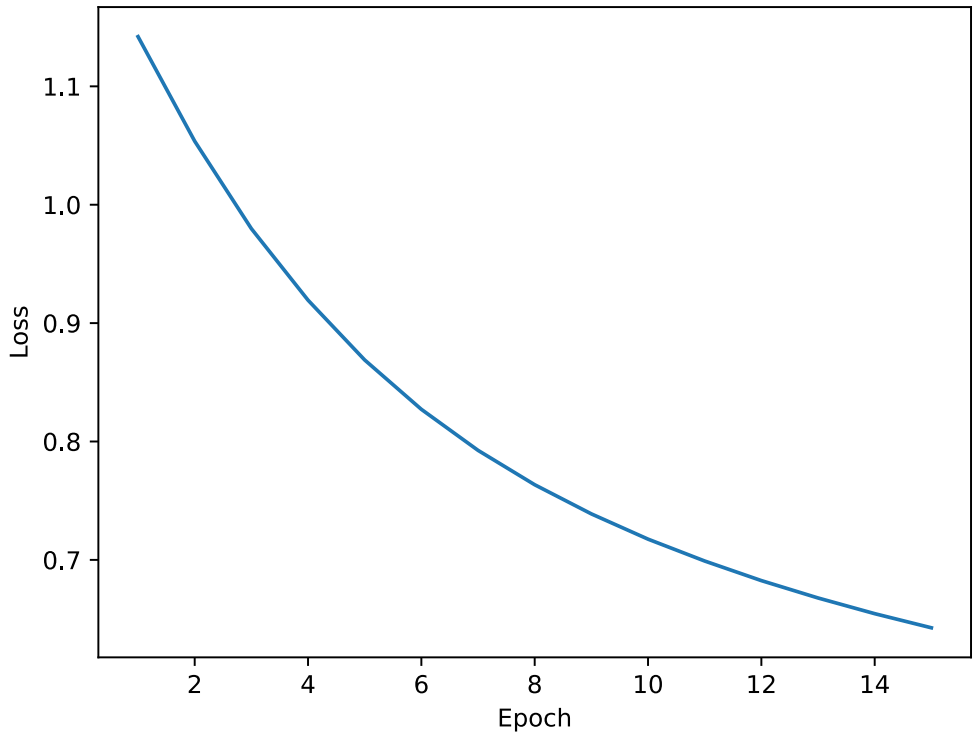

Figure 3: Training loss as a function of the epoch number for $\eta = 0.01$.

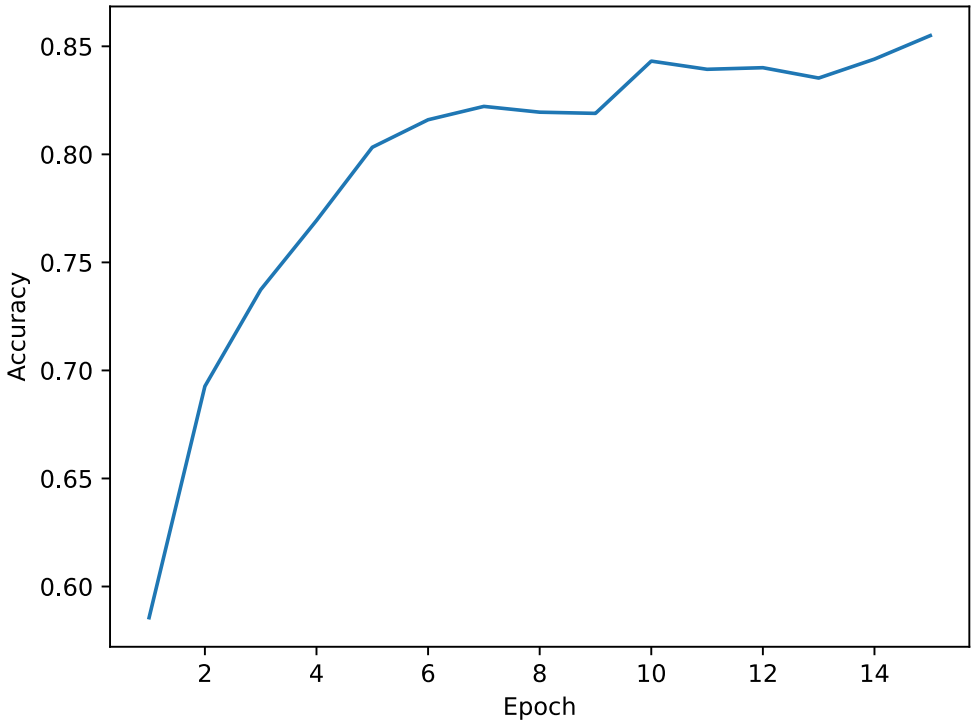

Figure 4: Validation accuracy as a function of the epoch number for $\eta = 0.01$.

3. *Implement the function* `get_number_trainable_params` *to determine the number of trainable parameters of CNNs from the two previous questions. What justifies the difference in terms of performance between the networks?*

   The number of trainable parameters for both the CNNs with the max pooling layres and the CNNs without the max pooling layers is 224892.

   The difference in performance between the two network may be justified by the fact that max pooling layers aggregate the information from the previous convolutional layer in a way that is invariant to small translations, rotation and scales of the input. This means that the network with the max pooling layers is more robust to small changes to the input, which is useful for image classification tasks, like the one at hand, conferring the models with pooling layers a better performance.

Question 3

Automatic Speech Recognition

1. *For the LSTM, plot the comparison of the training and validation loss over epochs and the string similarity scores obtained in the validation set. Also, report the final test loss and the string similarity scores obtained with the test set.*

The comparison of the training and validation losses over epochs are plotted in Figure 5 while the Jaccard, cosine and Damerau-Levenshtein string similarity scores obtained in the validation set are represented, respectively, in Figures 6, 7 and 8.

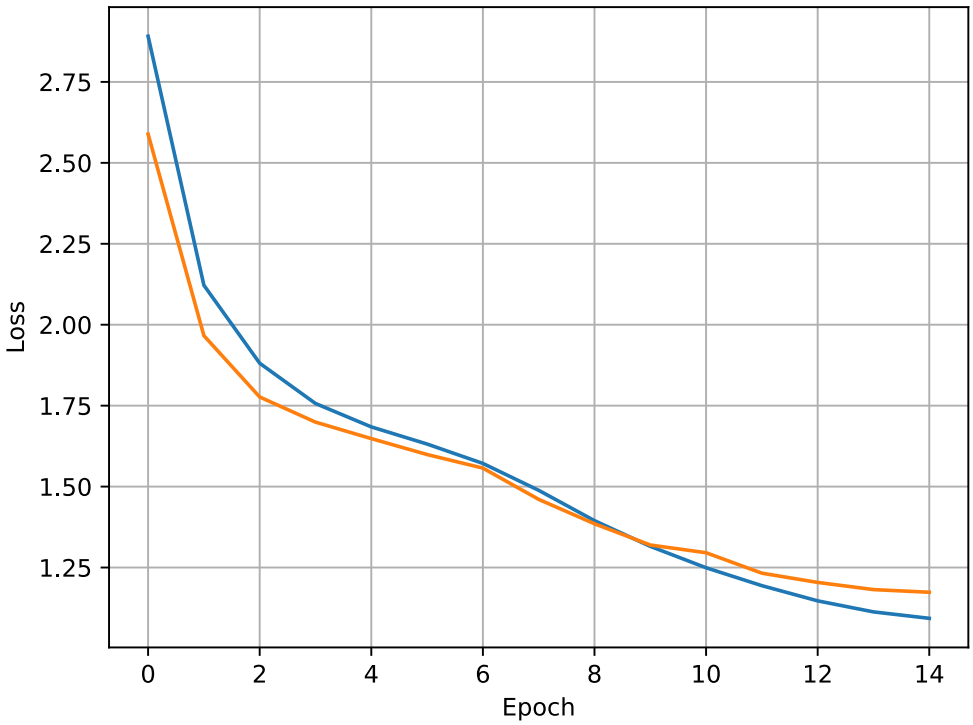

Figure 5: Training and validation losses over epochs of the LSTM.

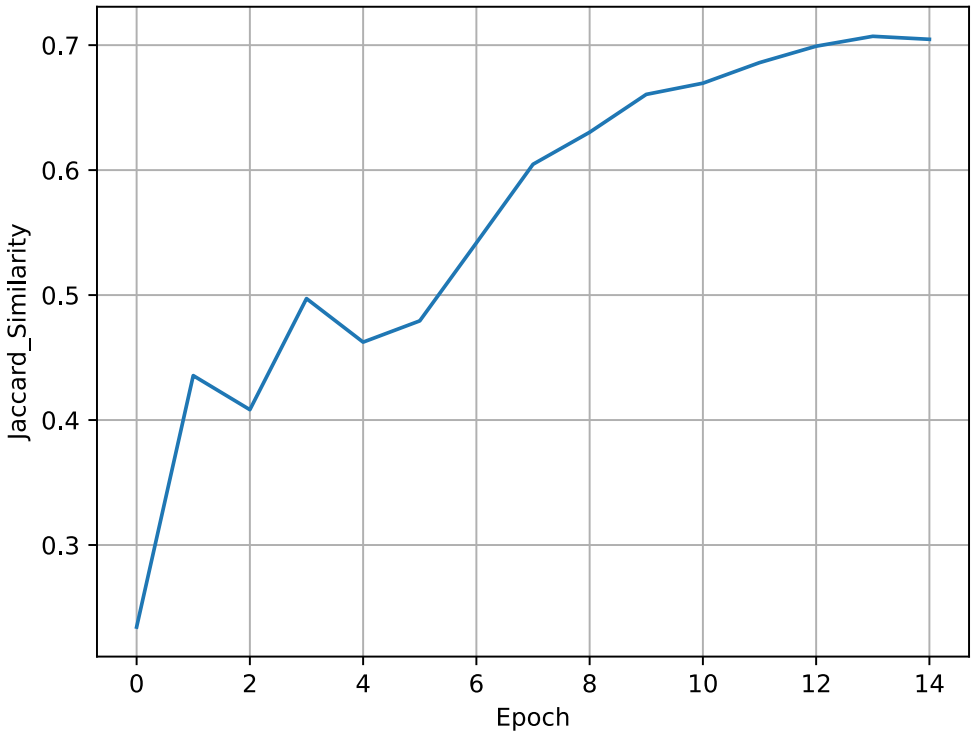

Figure 6: Jaccard similarity scores in the validation set over epochs of the LSTM.

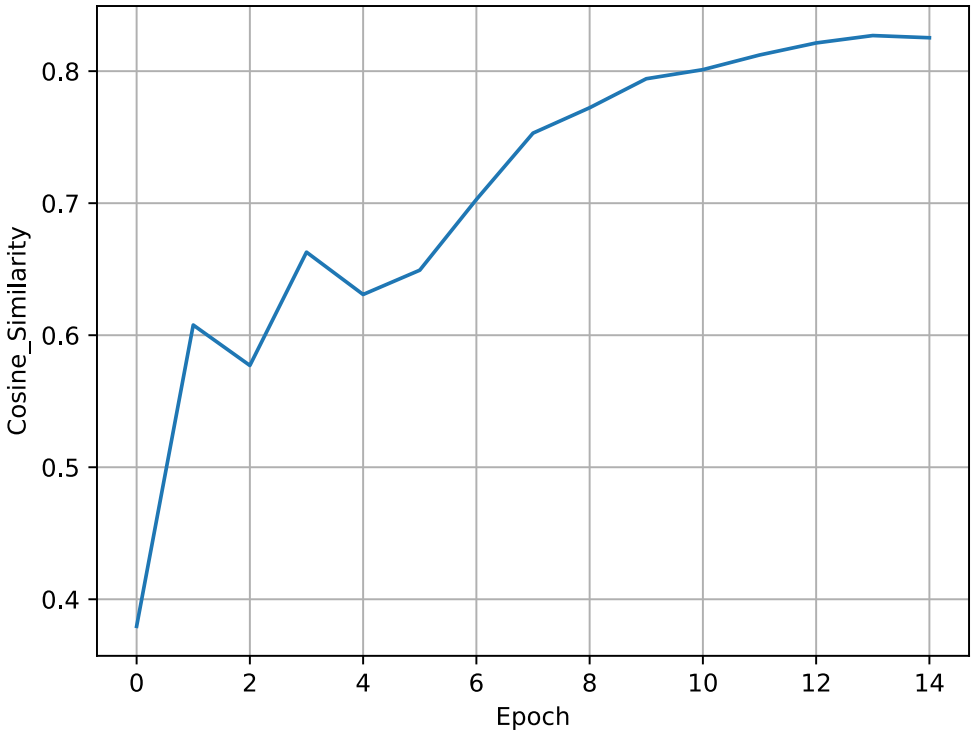

Figure 7: Cosine similarity scores in the validation set over epochs of the LSTM.

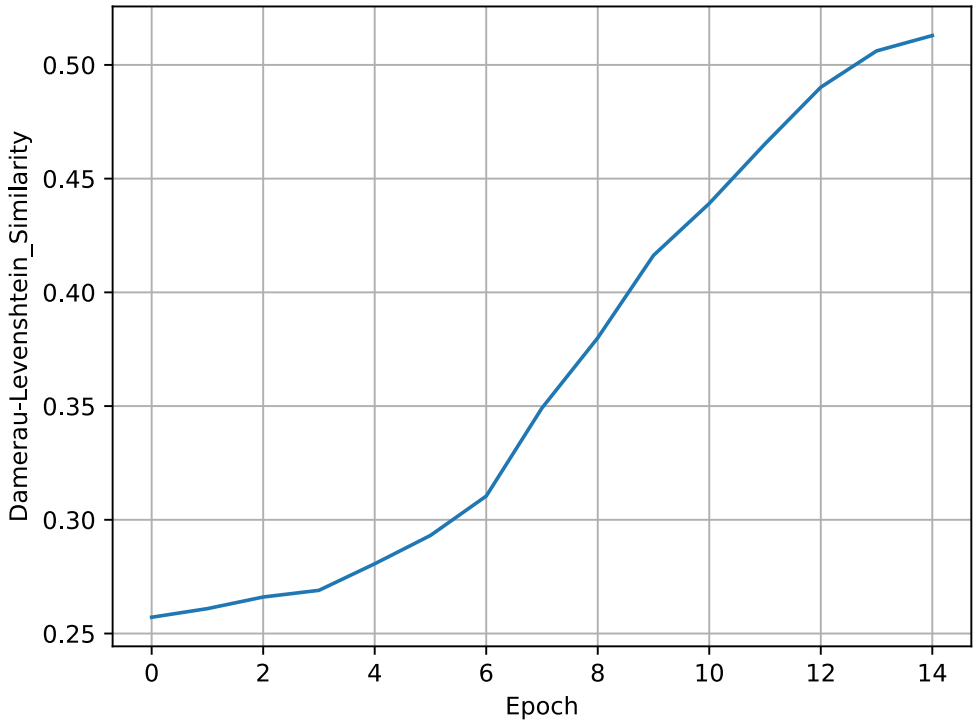

Figure 8: Damerau-Levenshtein similarity scores in the validation set over epochs of the LSTM.

The final test loss is 1.1828277518109578. The string similarity scores obtained with the test set are depicted in Table 3.

String Similarity Metric	**Score**
Jaccard	0.714888785108462
Cosine	0.8323868545583469
Damerau-Levenshtein	0.5086983165244969

Table 3: String similarity score of the LSTM.

2. *For the attention mechanism, plot the comparison of the training and validation loss over epochs and the string similarity scores obtained in the validation set. Also, report the final test loss and the string similarity scores obtained with the test set.*

   The comparison of the training and validation losses over epochs are plotted in Figure 9 while the Jaccard, cosine and Damerau-Levenshtein string similarity scores obtained in the validation set are represented, respectively, in Figures 10, 11 and 12.

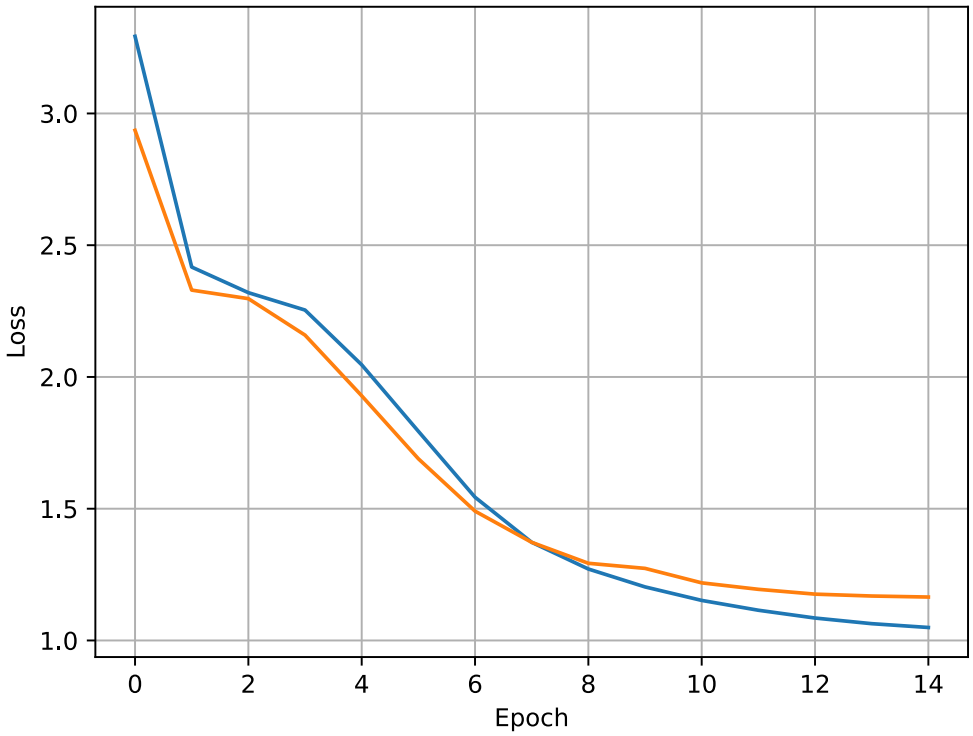

Figure 9: Training and validation losses over epochs of the attention mechanism.

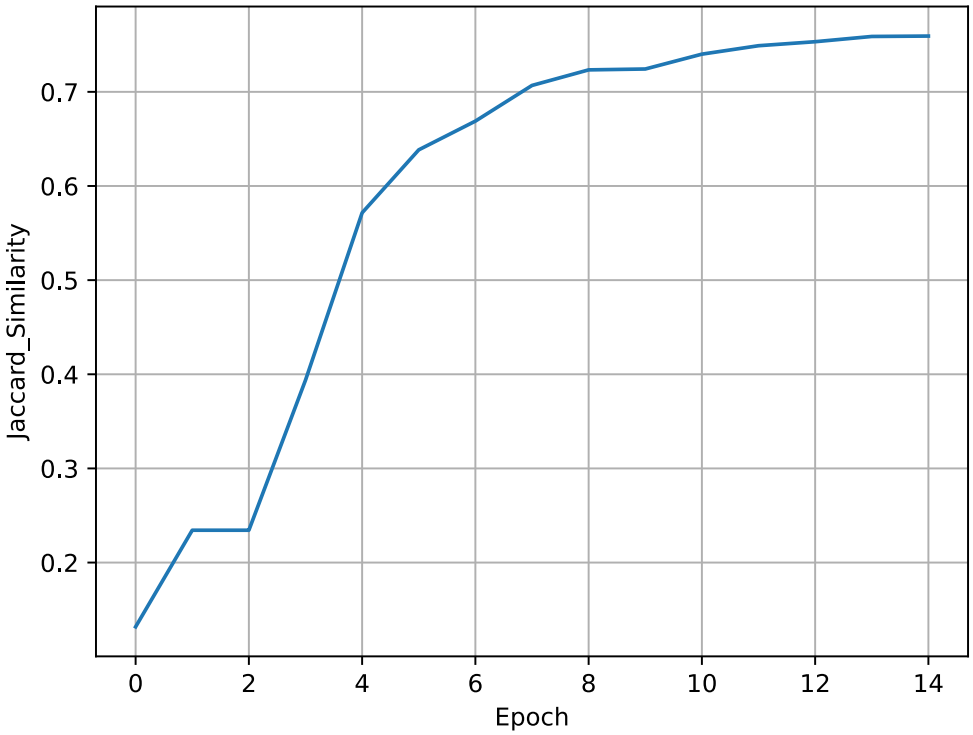

Figure 10: Jaccard similarity scores in the validation set over epochs of the attention mechanism.

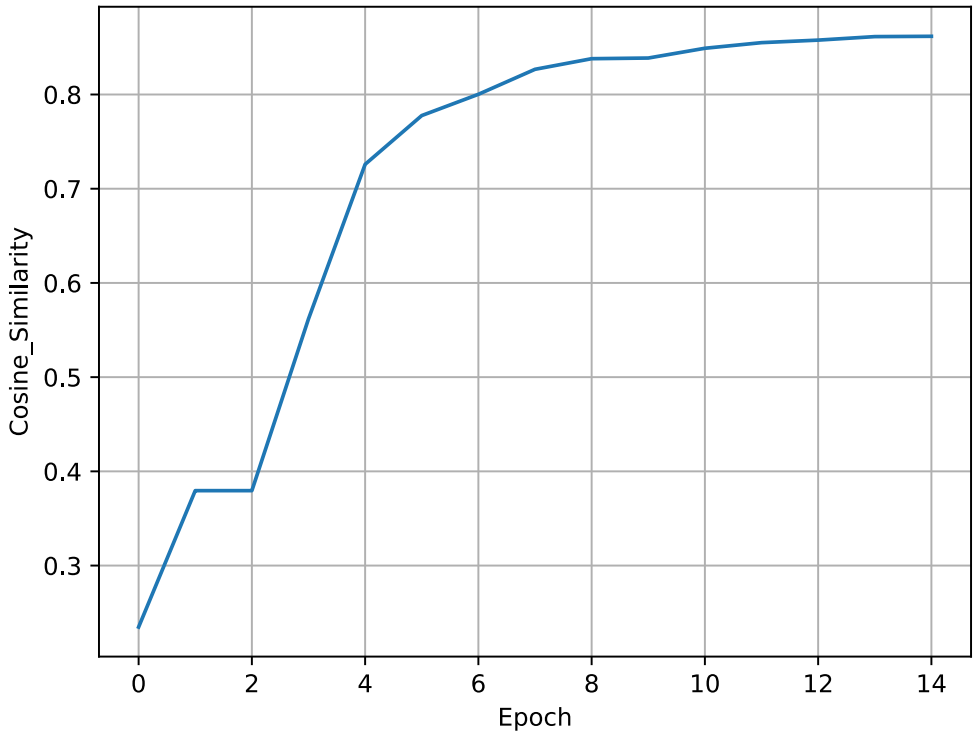

Figure 11: Cosine similarity scores in the validation set over epochs of the attention mechanism.

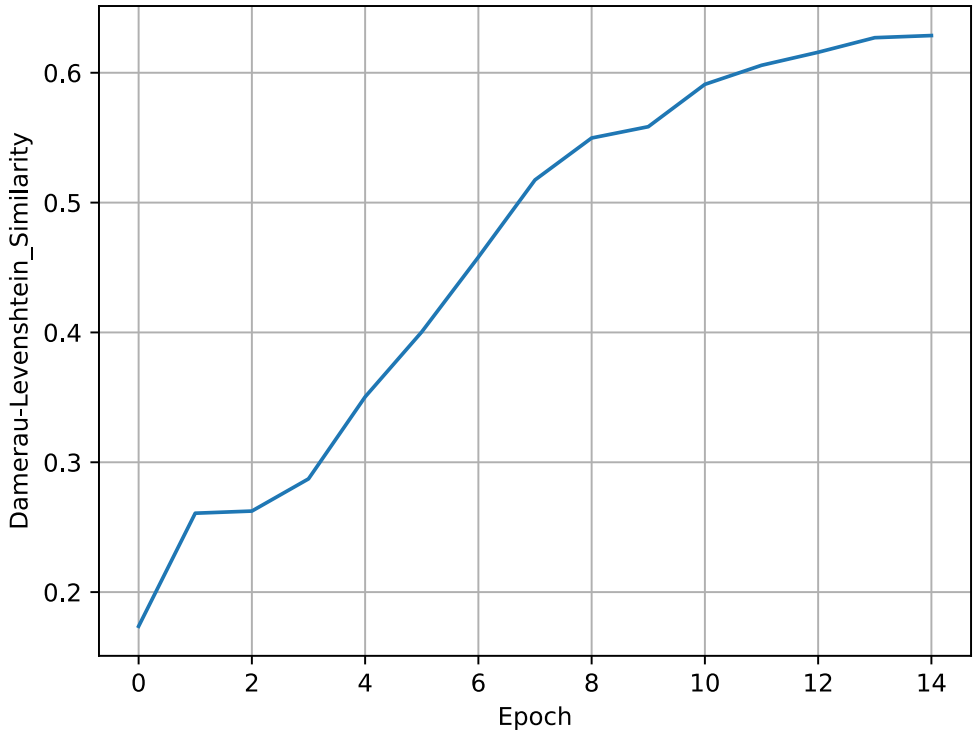

Figure 12: Damerau-Levenshtein similarity scores in the validation set over epochs of the attention mechanism.

The final test loss is 1.1603884776917899. The string similarity scores obtained with the test set are depicted in Table 4.

String Similarity Metric	Score
Jaccard	0.7644286566881149
Cosine	0.865211054644348
Damerau-Levenshtein	0.6329973770804788

Table 4: String similarity score of the attention mechanism.

3. *Comment on the differences in the test results obtained using the decoder architectures in questions 1 and 2. Note: start by illustrating how the LSTM (in question 1) and the attention mechanism (in question 2) process the text input.*

   LSTMs have a memory cell that allows them to capture long-term dependencies in the input sequence. They use gates to control the flow of information into and out of the cell, making them capable of learning and retaining information over extended periods.

   The attention mechanism allows the model to selectively focus on different parts of the input sequence when generating each element of the output sequence. Attention helps the model selectively attend to relevant information, providing a dynamic way to weigh the importance of different parts of the input sequence at each step of decoding.

   The attention mechanism outperforms the LSTM as it is better processing long sequences and capturing textual dependencies. On the other hand, the LSTM needs to *remember* past information, which is not as efficient as the attention mechanism.

4. *Comment on the score values obtained by each string similarity score used. Why each score has different values based on what each similarity tries to evaluate?*

   The different score values arise from the distinct characteristics and underlying principles of each similarity metric. The more appropriate metric depends on the use case.

   Jaccard similarity compares the intersection of two sets to their union. The resulting value, ranging from 0 to 1, represents the proportion of shared elements between the sets.

   Cosine similarity measures the cosine of the angle between two vectors that represent the two strings.

   Damerau-Levenshtein similarity quantifies the similarity between two strings based

on the minimum number of edit operations (insertions, deletions, substitutions, and transpositions) needed to transform one string into the other.