TÉCNICO LISBOA

Deep Learning

Homework 2

Authors:	**Student number:**
Vito Rozman	108734
Tomer Klein	108547

Group 91

1st Semester 2023/2024

Contents

1 Question

1.1 Solution 1

Let's break down the computation:

- QK^Tinvolves a matrix multiplication between $Q \in \mathbf{R}^{L\times D}$ and the transpose of $K^T \in \mathbf{R}^{D\times L}$, resulting in a matrix of size $L \times L$. This takes complexity of $O(D \cdot L^2)$.
- Softmax(QK^T) computes the softmax along the rows of the matrix obtained from the previous step, which takes complexity of $O(L^2)$.
- The final multiplication $V(Softmax(QK^T))$ involves multiplying the Softmax-weighted matrix by $V \in \mathbf{R}^{LxD}$, resulting in a matrix of size LxD and again takes complexity of $O(D \cdot L^2)$.

So we have: $O(D \cdot L^2) + O(L^2) + O(D \cdot L^2) = O(D \cdot L^2)$

This complexity becomes problematic for long sequences due to its quadratic dependence on the sequence length (L). As the length of the sequence increases, the computational cost grows quadratically. This can lead to significant computational demands and memory requirements, making it challenging to handle very long sequences efficiently.

1.2 Solution 2

For arbitrary vectors $q \in \mathbb{R}^D$ and $k \in \mathbb{R}^D$, using the McLaurin series approximation $\exp(t) \approx 1 + t + \frac{t^2}{2}$, we define the feature map $\phi(q)$ as $\phi : \mathbb{R}^D \to \mathbb{R}^M$ such that $\exp(q^T k) \approx 1 + q^T k + \frac{1}{2}(q^T k)^2 = \phi(q)^T \phi(k)$.

Let $\phi(q) = [1, q, \frac{q^2}{\sqrt{2}}]$ and $\phi(k) = [1, k, \frac{k^2}{\sqrt{2}}]$, and let's calculate the inner product:

$$\phi(q)^T \phi(k) = 1 + q^T k + \frac{1}{2}(q^T k)^2$$

As observed, this is equivalent to the three-term McLaurin series approximation. Consequently, this feature map provides a reliable approximation of $\exp(q^T k)$.

The dimensionality of the feature space M is determined by the dimensionality of $\phi(q)$, which is 3. Except for the constant term, the other two terms have the same dimensionality which is D. Therefore, we have $M = 2D + 1$, and in this case, $M = 7$.

If we look at at for general K, the idea is the same. For each additional term added, there is another term in the feature map with dimensionality D: $M = (K - 1)D + 1$.

1.3 Solution 3

The self-attention operation is defined as:

$$Z = \text{Softmax}(QK^T)V.$$

Using given approximation $\exp(q^T k) \approx \phi(q)^T \phi(k)$, where $\phi(x) = [1, x, \frac{x^2}{2}]$, we can rewrite the self-attention operation as:

$$Z \approx \text{Softmax}(\phi(Q)\phi(K)^T)V. \tag{1}$$

If we define $D = \text{Diag}(\phi(Q)\phi(K)^T \mathbf{1}_L)$, we can se that the equation (1) can rewrite:

$$Z \approx D^{-1}(\phi(Q)\phi(K)^T)V.$$

1.4 Solution 4

To calculate this approximation for linear computational complexity, we can compute $\Phi(Q)\Phi(K)^T$ efficiently. The computational complexity is now dominated by the matrix multiplication $\Phi(Q)\Phi(K)^T$, which, using standard algorithms, has a complexity proportional to $L \cdot M \cdot D$.

Therefore, the overall computational complexity is linear in L and depends on M and D through the product $L \cdot M \cdot D$.

2 Question

2.1 Solution 1

We implemented simple convolutional network and trained with 15 epochs with SDG. To find the best tuning of learning rates we checked $\eta \in \{0.1, 0.01, 0.001\}$. The best results were computed with $\eta = 0.01$ which resulted final test accuracy 83.55%. In Figure 1 and 2 are presented the training loss and the validation accuracy, both as a function of the epoch number.

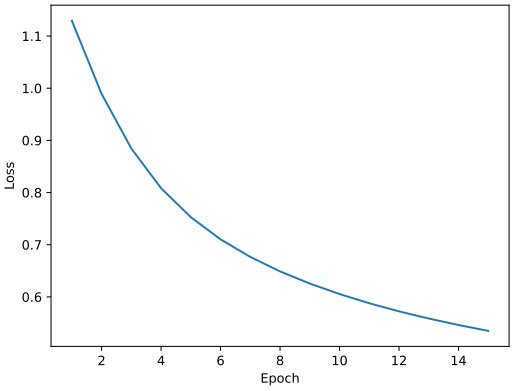

Figure 1: *Training loss ($\eta = 0.01$)*

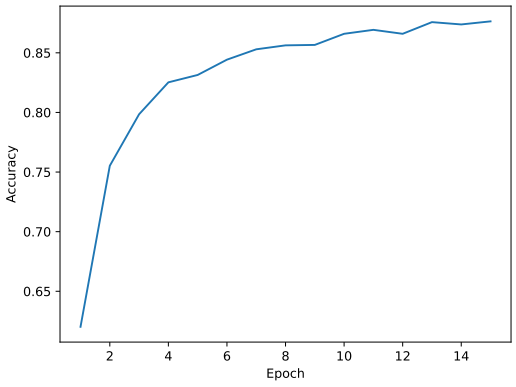

Figure 2: *Training accuracy ($\eta = 0.01$)*

2.2 Solution 2

Here, we aimed to examine the influence of max-pooling. In this architecture, there are no max-pooling layers after the convolutional layer. Once again, we experimented with different learning rate parameters to find the best fit. It is observed that the highest accuracy decreased from 83.55% to 82.42%. Results are on Figures 3 and 4.

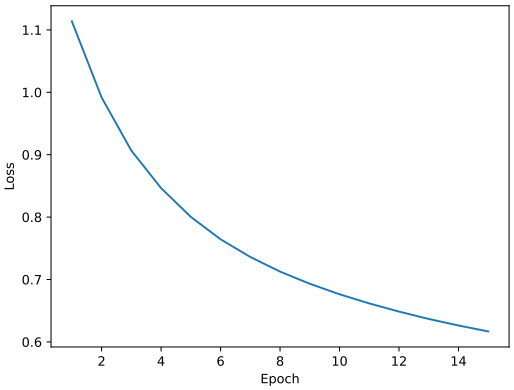

Figure 3: *Training loss ($\eta = 0.01$)*

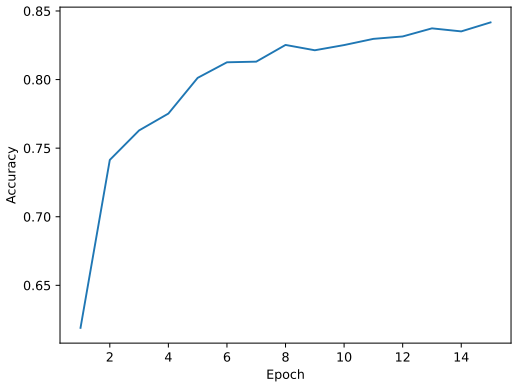

Figure 4: *Training accuracy ($\eta = 0.01$)*

2.3 Solution 3

The comparison between models with and without max-pooling reveals differences in the number of parameters and accuracy seen in Table 1.

Table 1: *Comparison of Models with and without Max-pooling*

	max-pooling	no max-poling
# parameters	291452	224892
acc ($\eta = 0.1$)	0.7769	0.7788
acc ($\eta = 0.01$)	0.8355	0.8242
acc ($\eta = 0.001$)	0.7732	0.7599

Max-pooling introduces additional parameters, often associated with the pooling layers. These parameters contribute to the increased total count. The pooling operation involves selecting the maximum value from a set of values, and this requires parameters for each pooling window. Without max-pooling layers, the model has fewer parameters, which might lead to a more parameter-efficient architecture.

Max-pooling layers are often used to downsample and retain the most important features. This can help the model focus on the most salient information, contributing to better overall performance. It's also possible that the increased capacity (due to more parameters) allows the model to learn more complex patterns in the data. The model without max-pooling achieves slightly lower accuracy and could potentially lead to less effective capturing of essential features and spatial hierarchies in the data.

Max-pooling can contribute to better performance by downsampling and focusing on essential features. However, it also increases the number of parameters, which may lead to overfitting if not carefully managed.

3 Question

3.1 Solution 1

In Figures 5, 6, 7, 8, the loss and three different metrics for comparing texts are presented. The final results on the test set are shown in the table 2.

Table 2: *Results on test set with RNN*

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

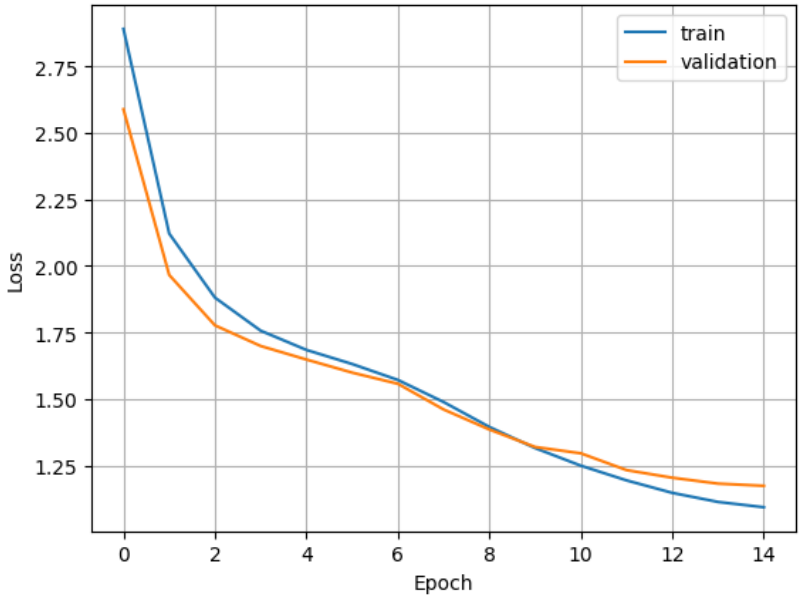

Figure 5: *Training and validation loss with RNN*

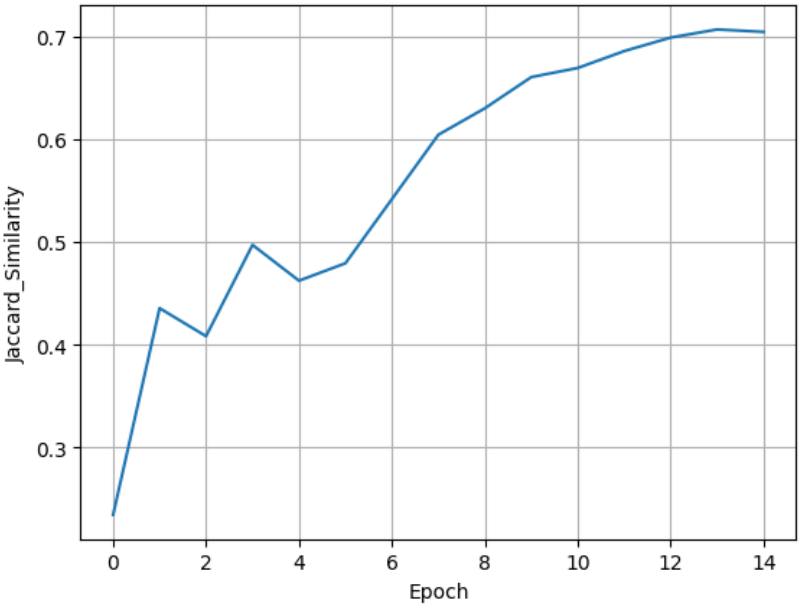

Figure 6: *Jaccard similarity with RNN*

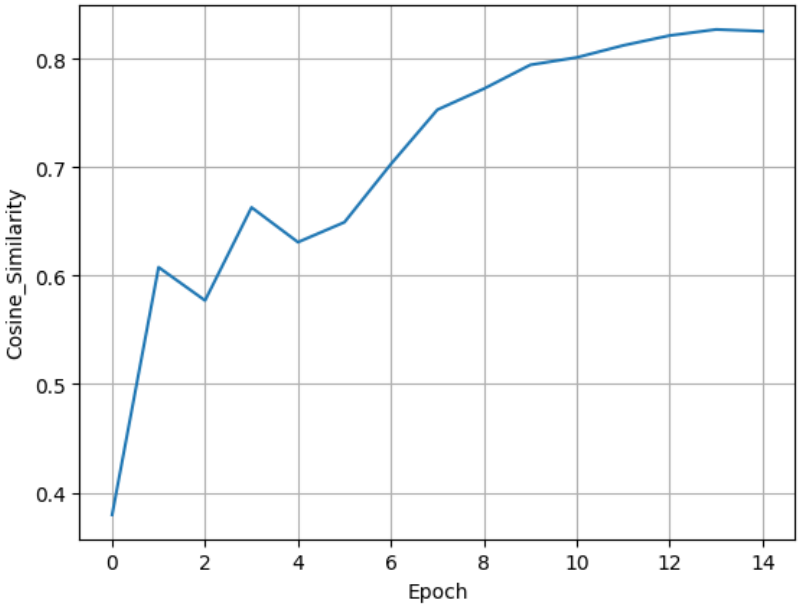

Figure 7: *Cosine similarity with RNN*

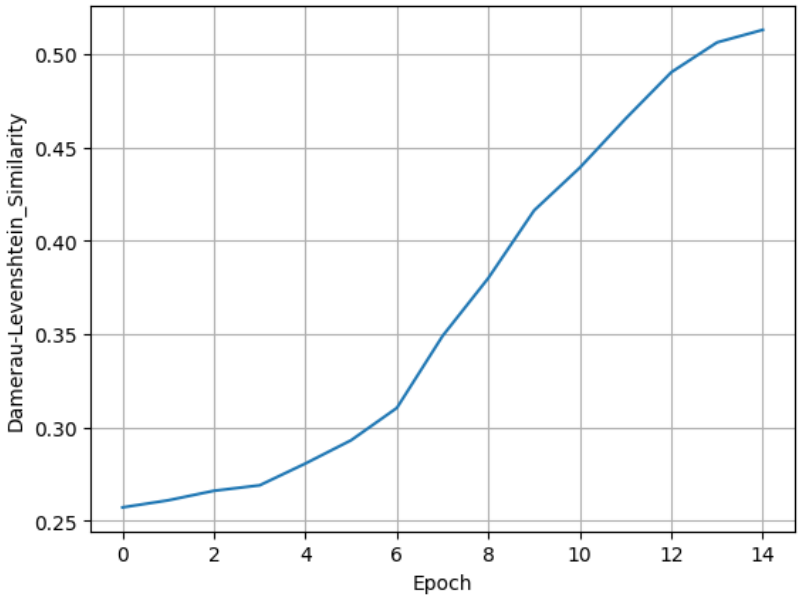

Figure 8: *Damerau-Levenshtein similarity with RNN*

3.2 Solution 2

In Figures 9, 10, 11, 12, the loss and three different metrics for comparing texts are presented. The final results on the test set are shown in the table 3.

Table 3: *Results on test set with Transformer*

Loss	1.1608
Jaccard similarity	0.7635
Cosine similarity	0.8645
Damerau-Levenshtein similarity	0.6312

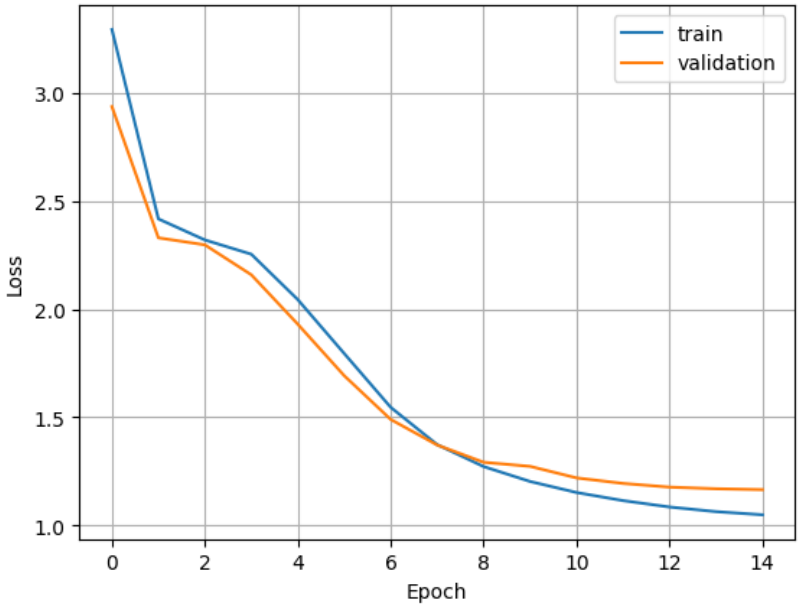

Figure 9: *Training and validation loss with Transformer*

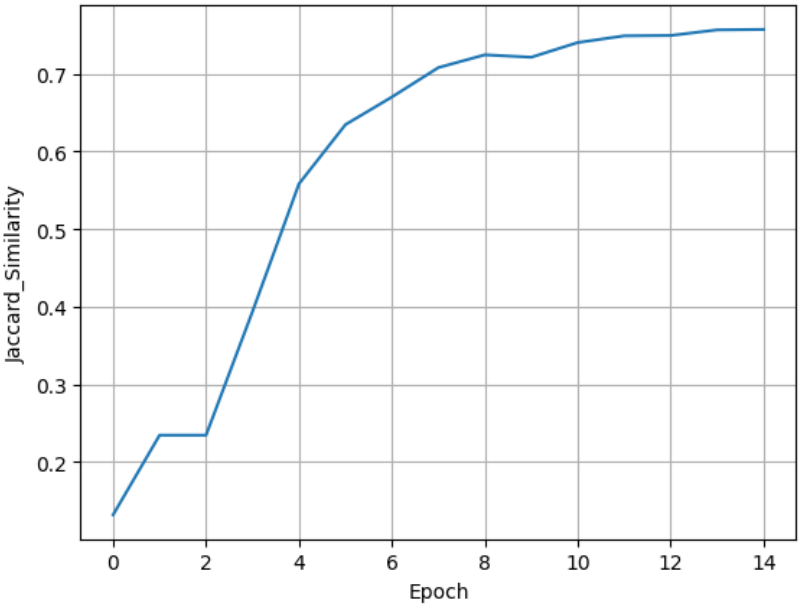

Figure 10: *Jaccard similarity with Transformer*

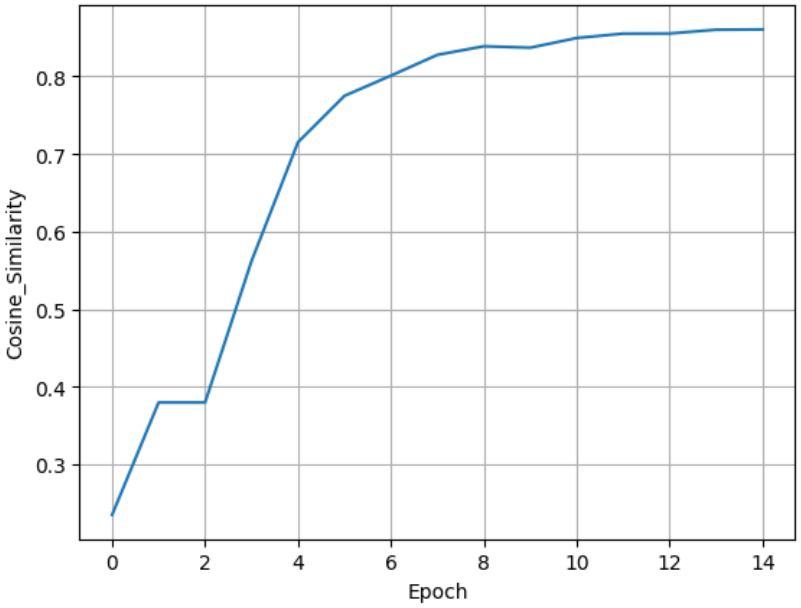

Figure 11: *Cosine similarity with Transformer*

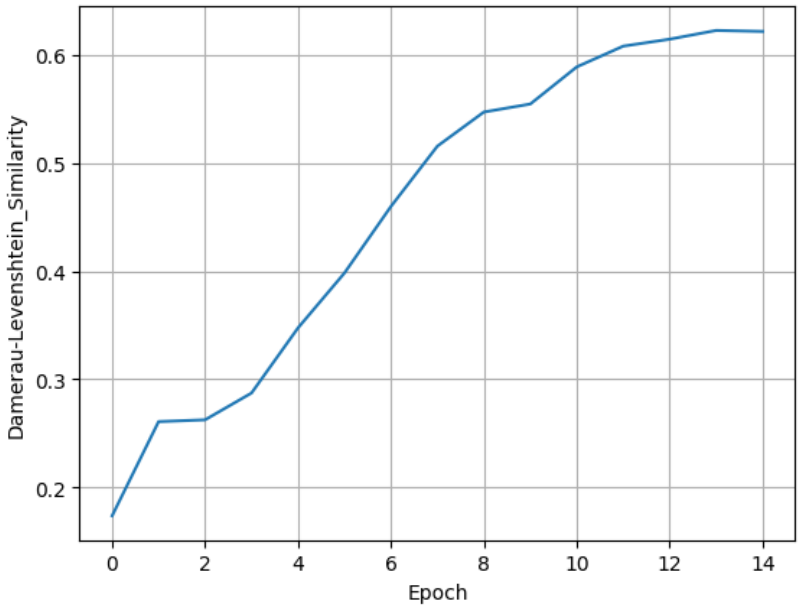

Figure 12: *Damerau-Levenshtein similarity with Transformer*

3.3 Solution 3

Transformers using attention-based mechanism and LSTMs (Long Short-Term Memory networks) are both types of architectures used for processing sequential data, such as text. While LSTMs are a type of recurrent neural network (RNN) designed to capture long-range dependencies in sequential data, the other rely on a self-attention mechanism to process input sequences in parallel, making them more efficient for long-range dependencies.

Results have shown us that the decoder with self-attention mechanisms have better performance compared to RNN. The parallel processing nature of transformers with self-attention enables them to handle sequences more efficiently, which is advantageous for tasks where capturing global context is crucial.

3.4 Solution 4

Jaccard similarity measures the intersection over the union of sets.

- Recurrent Network Score (0.7149): In the context of your recurrent network, it seems that the similarity is around 71%, indicating that there is a considerable overlap between the sets of elements in the compared strings.
- Transformer Attention Mechanism Score (0.7635): The increased score here might suggest that the transformer attention mechanism is better at capturing overlapping elements or commonalities between strings.

Cosine similarity is based on the cosine of the angle between two vectors.

- Recurrent Network Score (0.8324): In this context, a score of 83% indicates a relatively high degree of alignment or similarity between the vector representations of the strings in the recurrent network.
- Transformer Attention Mechanism Score (0.8645): The transformer attention mechanism achieves a slightly higher score, suggesting that it may be more effective in capturing the overall direction or similarity of the vector representations.

Damerau-Levenshtein similarity measures the similarity based on the minimum number of operations required to transform one string into the other.

- Recurrent Network Score (0.5087):The lower score may indicate that the recurrent network is not as effective in capturing the similarity based on these edit operations.
- Transformer Attention Mechanism Score (0.6312): The transformer attention mechanism achieves a higher score, suggesting that it may better capture the structural or sequential similarity between the strings.

4 Contribution of each member

In our project, each team member contributed equally for the project. We maintained a collaborative spirit throughout, working closely together and meeting regularly to ensure shared decision-making. This balanced approach allowed us to leverage individual strengths, resulting in a successful project outcome.