Deep Learning - Homework 2

Luís Xu 99100 and Jaime Gosai 99239 - G03

06 January 2024

Work contribution

The whole Question 1 was done by together, the Question 2 was done by Jaime Gosai and the Question 3 was done by Luís Xu.

Question 1

Q1

1) The complexity is $O(L^2 D)$. For long sequences the computation will be expensive, since we depend quadratically on the input.

2) $\exp(q^T k) = 1 + q^T k + \frac{(q^T k)^2}{2!} = 1 + \sum_{i=1}^{D} q_i k_i + \frac{\left(\sum_{i=1}^{D} q_i k_i\right)^2}{2!}$

$= \phi(q)^T \phi(k)$

$\phi(v) = \left(1, v_1, \dots, v_D, \frac{v_1 v_1}{\sqrt{2!}}, \dots, \frac{v_1 v_D}{\sqrt{2!}}, \dots, \frac{v_D v_D}{\sqrt{2!}}\right)$, $\phi : \mathbb{R}^D \to \mathbb{R}^M$

$M = 1 + D + D^2$

$\left(\sum_{i=1}^{k} x_i\right)^2 = (x_1 + \dots x_k)^2 = x_1 x_1 + x_1 x_2 + \dots + x_k x_{k-1} + x_k x_k$

$= \sum_{i=1}^{k} \sum_{j=1}^{k} x_i x_j$

$\left(\sum_{i=1}^{D} q_i k_i\right)^2 = \sum_{i=1}^{D} \sum_{j=1}^{D} q_i k_i q_j k_j = \sum_{i=1}^{D} \sum_{j=1}^{D} q_i q_j k_i k_j$

For $n \geq 3$ $\exp(q^T k) = 1 + q^T k + \dots + \frac{(q^T k)^n}{n!}$

$= 1 + \sum_{i=1}^{D} q_i k_i + \dots + \frac{\left(\sum_{i=1}^{D} q_i k_i\right)^n}{n!}$

$\left(\sum_{i=1}^{k} x_i\right)^n = \underbrace{\sum_{i=1}^{k} \cdots \sum_{n=1}^{k}}_{n \text{ summations}} x_i \dots x_n$

, with this we can use the same steps to create the ϕ function.

$M = \sum_{i=0}^{n} D^i = \frac{D^{n+1} - 1}{D - 1}$

$\sum_{k=0}^{n} r^k = \frac{r^{n+1} - 1}{r - 1}$

3) $z \simeq \hat{D}^{-1}\hat{\Phi}(Q)\hat{\Phi}(K)^T V \Leftrightarrow \mathrm{softmax}(QK^T) \simeq \hat{D}^{-1}\hat{\Phi}(Q)\hat{\Phi}(K)^T$

$$\mathrm{softmax}(QK^T) = \mathrm{softmax}\left(\begin{bmatrix} q_1 k_1^T & \cdots & q_1 k_L^T \\ \vdots & \ddots & \vdots \\ q_L k_1^T & \cdots & q_L k_L^T \end{bmatrix}\right)$$

$$\simeq \begin{bmatrix} \frac{\phi(q_1)\phi(k_1)^T}{\sum_{i=1}^{L}\phi(q_1)\phi(k_i)^T} & \cdots & \frac{\phi(q_1)\phi(k_L)^T}{\sum_{i=1}^{L}\phi(q_1)\phi(k_i)^T} \\ \vdots & \ddots & \vdots \\ \frac{\phi(q_L)\phi(k_1)^T}{\sum_{i=1}^{L}\phi(q_L)\phi(k_i)^T} & \cdots & \frac{\phi(q_L)\phi(k_L)^T}{\sum_{i=1}^{L}\phi(q_L)\phi(k_i)^T} \end{bmatrix}$$

$$= \begin{bmatrix} D_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & D_L \end{bmatrix} \begin{bmatrix} \phi(q_1)\phi(k_1)^T & \cdots & \phi(q_1)\phi(k_L)^T \\ \vdots & & \vdots \\ \phi(q_L)\phi(k_1)^T & \cdots & \phi(q_L)\phi(k_L)^T \end{bmatrix}, \quad D_i = \phi(q_i)\sum_{j=1}^{L}\phi(k_j)^T$$

$$= \mathrm{Diag}\left(\hat{\Phi}(Q)\hat{\Phi}(K)^T 1_L\right)^{-1} \begin{bmatrix} \phi(q_1) \\ \vdots \\ \phi(q_L) \end{bmatrix} \begin{bmatrix} \phi(k_1) \\ \vdots \\ \phi(k_L) \end{bmatrix}^T$$

4) Determating the complexity of matrix multiplications:

	1	2	3	4
1	0	LM	L^2M	MLD
2	-	0	L^2M	2MLD
3	——	.	0	MLD
4	——	——		0

A_1: $Diag(\Phi(Q)\Phi(K)^T 1_L)^{-1} \in \mathbb{R}^{L\times L}$

A_2: $\Phi(Q) \in \mathbb{R}^{L\times M}$

A_3: $\Phi(K)^T \in \mathbb{R}^{M\times L}$

A_4: $V \in \mathbb{R}^{L\times D}$

- Notice that the $A_1 B$ is equal to:

$$\mathbb{R}^{L\times K} \ni \begin{bmatrix} a_1 & \cdots & a_1 \\ \vdots & & \\ a_L & \cdots & a_L \end{bmatrix} \odot B \in \mathbb{R}^{L\times K}$$

, and the complexity $O(LK)$

a_i is the diagonal element of A_1

- $H_{1,2} = A_1 A_2 = LM$ $\quad H_{2,3} = A_2 A_3 = L^2M$ $\quad H_{3,4} = MLD$

- $H_{1,3} = \min(H_{1,2} A_3, A_1 H_{2,3}) = \min(LM + L^2M, L^2 + L^2M)$

$= L^2M$

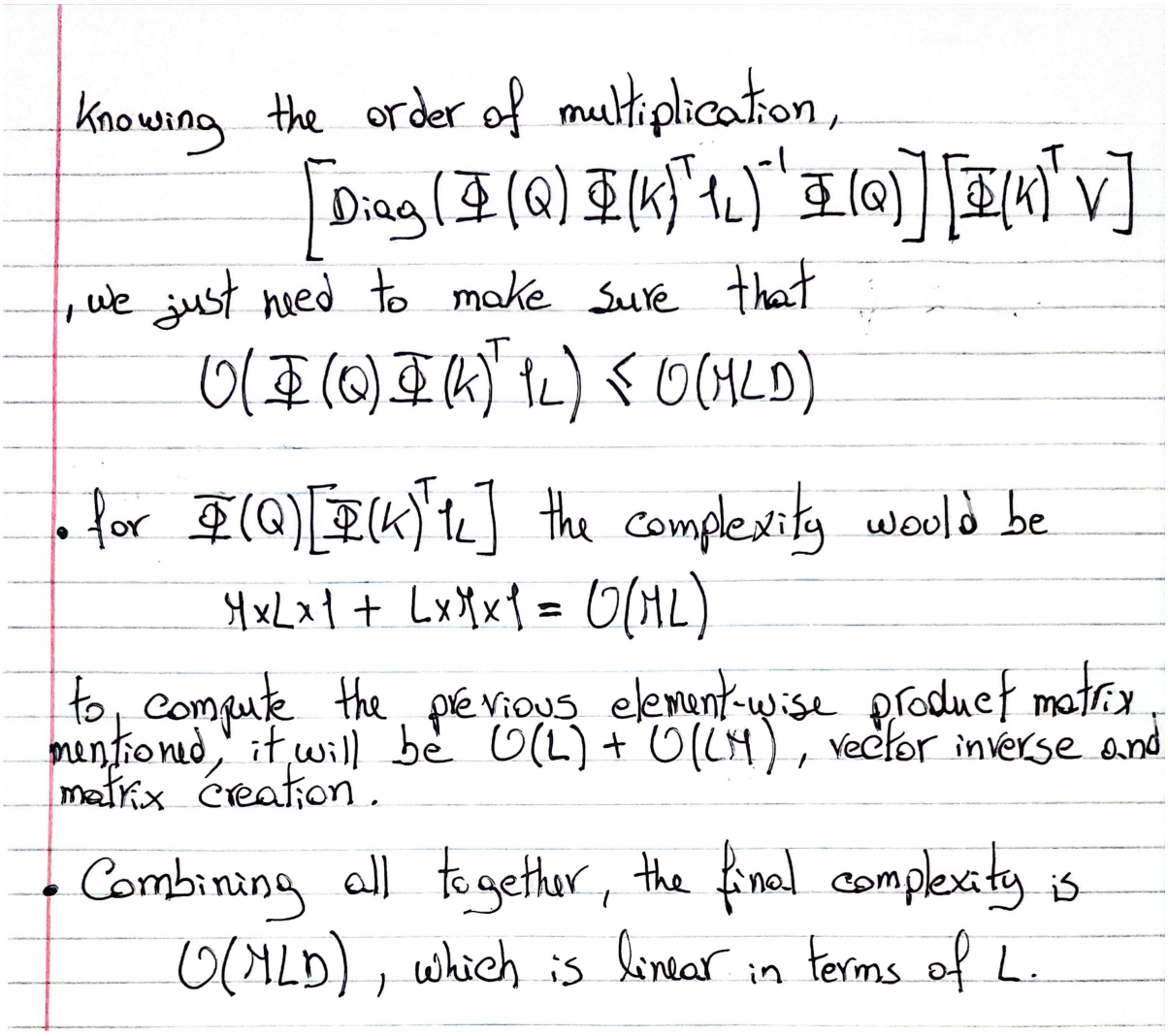
$H_{2,4} = \min(H_{2,3} A_4, A_2 H_{3,4}) = \min(L^2M + L^2D, MLD + MLD)$

$= 2MLD$

- $H_{1,4} = \min(H_{1,3} A_4, H_{1,2} H_{3,4}, A_1 \times H_{2,4})$

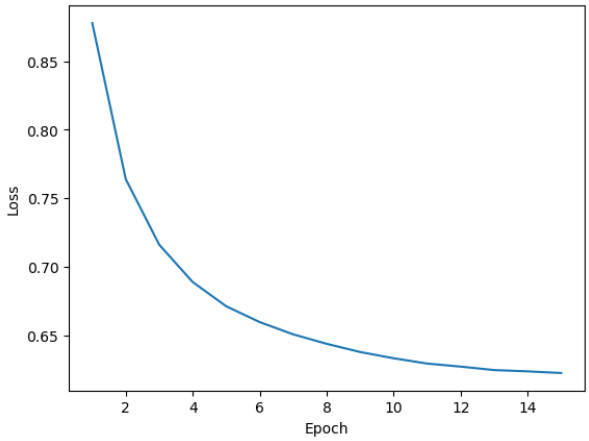
$= \min(L^2M + L^2D, LM + MLD + LMD, LD + 2MLD)$

$= LD + 2MLD$, $O(MLD)$

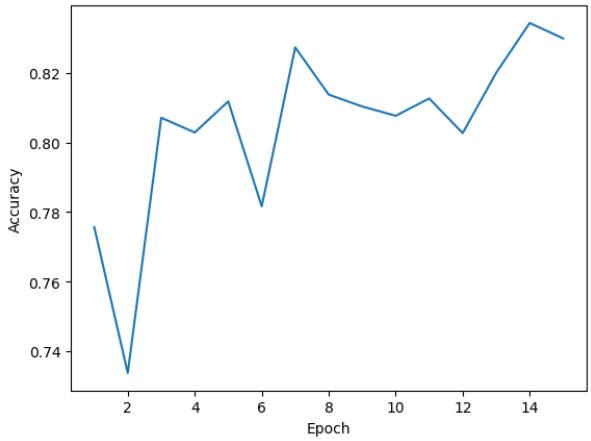


Question 2

1. The best configuration is for a learning rate of 0.01, presenting a validation accuracy of 0.8730 and a final test accuracy of 0.8223. It is also better converging than the other models.

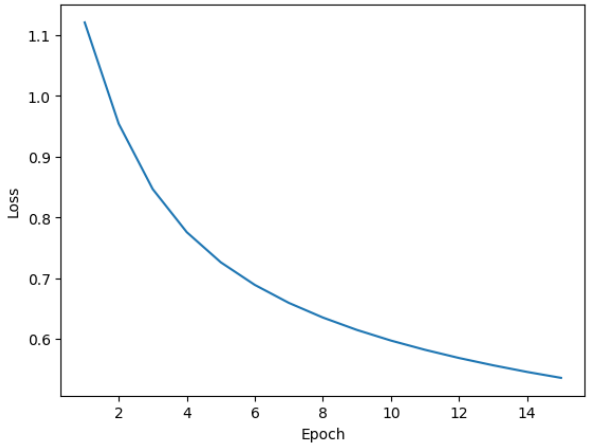


(a) training-loss

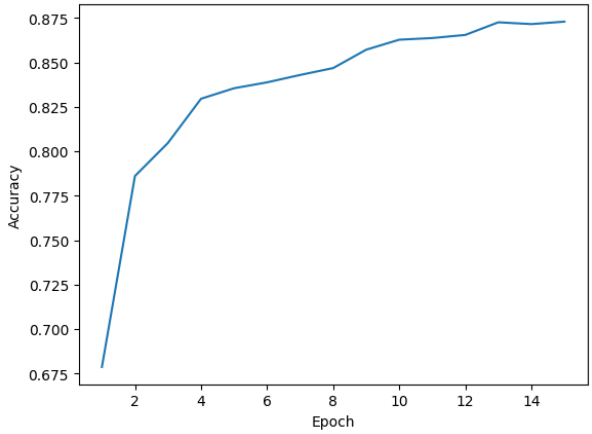


(b) validation-accuracy

Figure 1: Question 2.1, learning rate of 0.1 with pooling

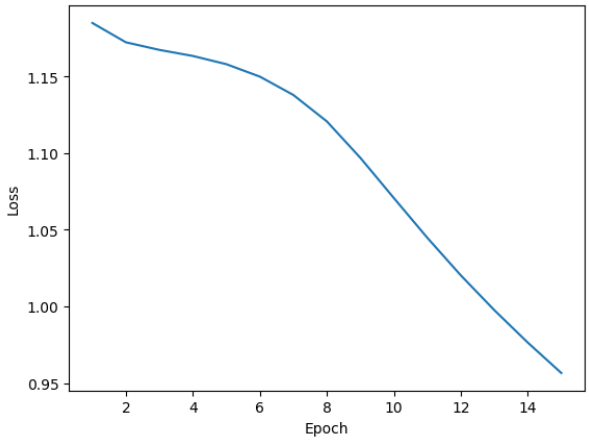


(a) training-loss

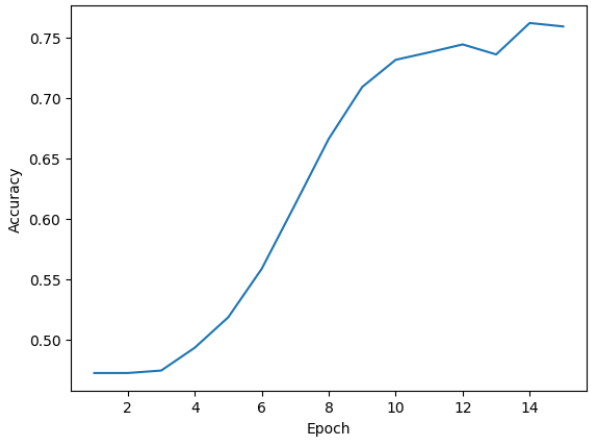


(b) validation-accuracy

Figure 2: Question 2.1, learning rate of 0.01 with pooling



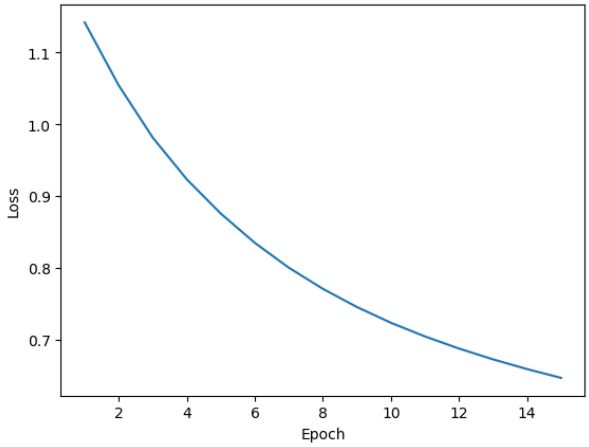
(a) training-loss



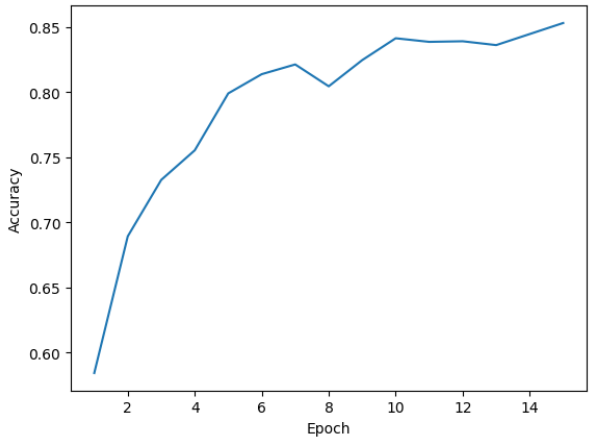
(b) validation-accuracy

Figure 3: Question 2.1, learning rate of 0.001 with pooling

2. The model using the optimal hyper-parameters defined in the previous question, which is 0.01, gives a validation accuracy of 0.8532 and a final Test accuracy of 0.7996.



(a) training-loss



(b) validation-accuracy

Figure 4: Question 2.2, learning rate of 0.01 without pooling

3. The number of trainable parameters was 224892 in all models (equal reduction), meaning that the down sampling achieved by the max pooling layers was able to better preserve the important details of the input, leading to a higher final accuracy. The higher stride convolution layers, although more time efficient, failed to preserve as many key details whilst down sampling the input.

Question 3

1. The results (in terms of final test) are:

- Jaccard similarity: 0.71489
- Cosine similarity: 0.83239
- Damerau-levenshtein similarity: 0.50870
- Loss: 1.18283

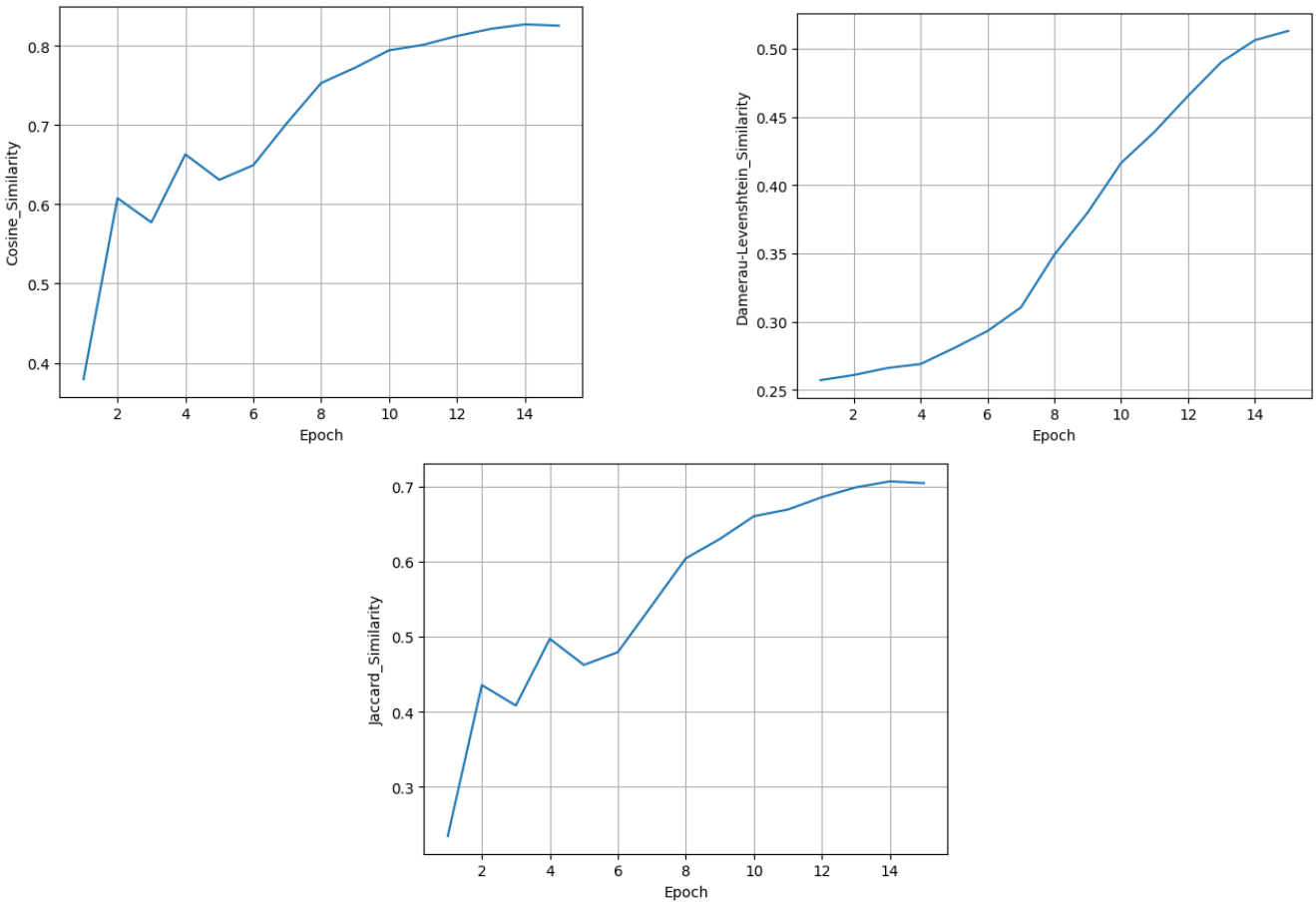


Figure 5: Question 3.1, validation set.

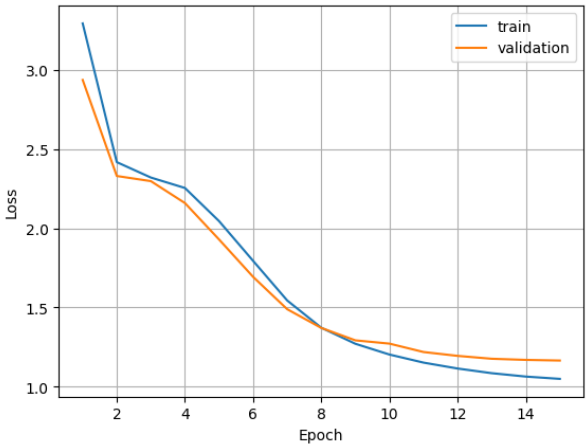
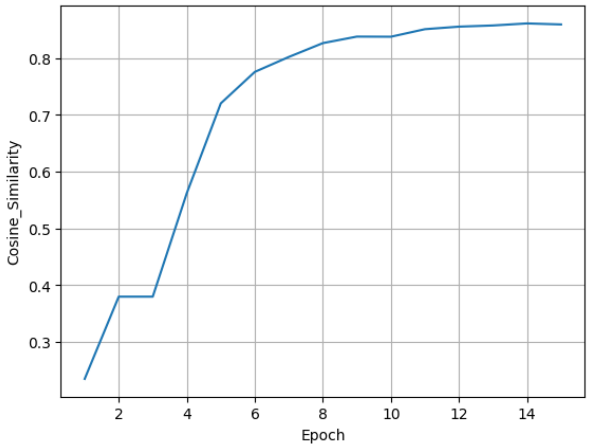


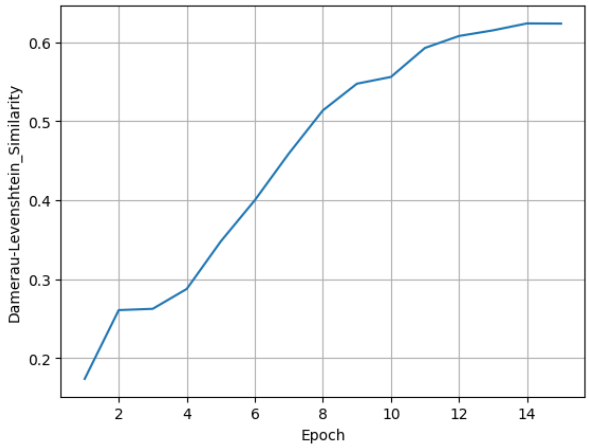
Figure 6: Question 3.1, TextDecoderRecurrent.

2. The results (in terms of final test) are:

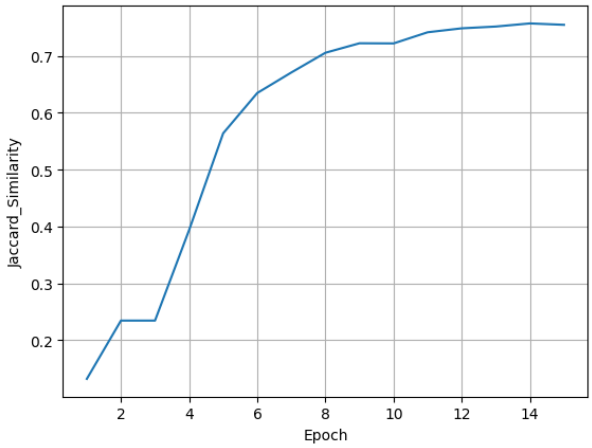
   - Jaccard similarity: 0.76541
   - Cosine similarity: 0.86592
   - Damerau-levenshtein similarity: 0.63409
   - Loss: 1.16045



(a) Cosine similarity



(b) Damerau-levenshtein similarity



(c) Jaccard similarity

Figure 7: Question 3.2, validation set.

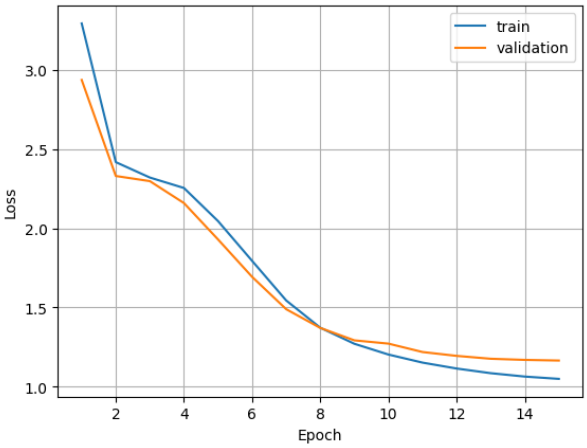


Figure 8: Question 3.2, TextDecoderTransformer.

3. The LSTM decoder processes the input sequentially only maintains the context of a fixed size previous inputs. The generations of the next output is based on the context memory.

   The attention decoder uses transformer to process the whole input in parallel, and generate the weight vector of the input.

   The second decoder perform better than the first, since it generates the output based on the whole text context.

4. - The **Jaccard similarity measures** the similarity between two sets by comparing the intersection of the sets. It is commonly used in set theory and is useful when the order of elements is not important.
   - The **Cosine similarity** measures the cosine of the angle between two vectors, which each vector represent term frequency distributions. It doesn't take into account the frequency of elements, so it may not reflect the importance of rare or common terms. It also doesn't consider word order.
   - The **Damerau–Levenshtein** similarity measures the distance between two strings, which takes into account the number of insertion, deletion, substitution, and transposition operations needed to transform one string into the other. However, the Damerau–Levenshtein distance can be computationally expensive, especially for long strings or large datasets.

   We can see that the Jaccard and Cosine are much higher than Damerau–Levenshtein, since they are simpler measures. The Cosine in greater than Jaccard, because it doesn't take into account the frequency of elements.