Deep Learning – Homework 2

Group 40
99238 – Inês Ji | 99314 – Raquel Cardoso

Contribution: Both members of the group took part in the resolution of all questions in the homework.

Question 1

1.

The computational complexity associated with computing $\boldsymbol{Z}$, in terms of sequence length $\boldsymbol{L}$, is the following:

- $O(L^2 \cdot D)$ to compute the dot product of $\boldsymbol{Q}$ and $\boldsymbol{K^T}$. Since we are multiplying matrices of dimensions R^{LxD} and R^{DxL} we will have a resulting matrix of dimension R^{LxL}, which means we will have D multiplications for each element of the resulting matrix throughout L^2 entries and therefore the above-mentioned complexity.
- $O(L^2)$ to compute the $\boldsymbol{Softmax}$ function applied to the resulting matrix of dimension R^{LxL} described above. The $\boldsymbol{Softmax}$ function is applied element to element therefore, we will apply it L^2 times, resulting in the complexity written.
- $O(L^2 \cdot D)$ to compute the multiplication between the result of applying the $\boldsymbol{Softmax}$ function (matrix of dimension R^{LxL}) and $\boldsymbol{V}$, which is a matrix of dimension R^{LxD} ($L \cdot D$ entries). This dot product results in a matrix of dimension R^{LxD}, therefore we will have L multiplications for each element of the resulting matrix throughout $L \cdot D$ entries, resulting in the described complexity.

The computational complexity is dominated by $O(L^2 \cdot D)$, which involves multiplication of matrices. The quadratic scaling of sequence length can be problematic for long sequences, as it makes the self-attention mechanism computationally expensive on extended sequences, leading to longer training times and an increased demand for computational resources.

2.

considering K the number of terms in McLaurin series expansion, $K=3: \exp(t) \approx 1 + t + \frac{t^2}{2}$

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

$$\exp(q^T k) \approx \phi(q)^T \cdot \phi(k)$$

$$q^T k = q_1 k_1 + q_2 k_2 + \dots q_D k_D = \sum_{i=1}^{D} q_i k_i$$

$$\begin{aligned}(q^T k)^2 &= (q_1 k_1 + q_2 k_2 + \dots + q_D k_D) \cdot (q_1 k_1 + q_2 k_2 + \dots + q_D k_D) \\ &= q_1^2 k_1^2 + q_2^2 k_2^2 + \dots + q_D^2 k_D^2 + 2 \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} q_i k_i \, q_j k_j \\ &= \sum_{i=1}^{D} q_i^2 k_i^2 + 2 \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} q_i q_j k_i k_j\end{aligned}$$

$$\frac{(q^T k)^2}{2} = \sum_{i=1}^{D} \frac{q_i^2}{\sqrt{2}} \frac{k_i^2}{\sqrt{2}} + \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} q_i q_j \, k_i k_j$$

$$\phi(q)^T \cdot \phi(k) \approx 1 + q^T k + \frac{(q^T k)^2}{2} \approx 1 + \sum_{i=1}^{D} q_i k_i + \sum_{i=1}^{D} \frac{q_i^2}{\sqrt{2}} \frac{k_i^2}{\sqrt{2}} + \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} q_i q_j \, k_i k_j$$

if we split each term of $1 + \sum_{i=1}^{D} q_i k_i + \sum_{i=1}^{D} \frac{q_i^2}{\sqrt{2}} \frac{k_i^2}{\sqrt{2}} + \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} q_i q_j \, k_i k_j$ into a multiplication of q and k we get:

$$1 \times 1 + \sum_{i=1}^{D} q_i \times \sum_{i=1}^{D} k_i + \sum_{i=1}^{D} \frac{q_i^2}{\sqrt{2}} \times \sum_{i=1}^{D} \frac{k_i^2}{\sqrt{2}} + \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} q_i q_j \times \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} k_i k_j$$

since this equals $\phi(q)^T \cdot \phi(k)$, for an arbitrary $\phi(x)$ we would have:

$$\phi(x) = \begin{bmatrix} 1 \\ \sum_{i=1}^{D} x_i \\ \sum_{i=1}^{D} \frac{x_i^2}{\sqrt{2}} \\ \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} x_i x_j \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \dots x_D \\ \frac{x_1^2}{\sqrt{2}} \dots \frac{x_D^2}{\sqrt{2}} \\ x_1 x_2 \dots x_1 x_D \\ (\dots) \\ x_{D-1} x_D \end{bmatrix}$$

$$\phi(q)^T\phi(k) = \begin{bmatrix} 1 & \sum_{i=1}^{D} q_i & \sum_{i=1}^{D} \frac{q_i^2}{\sqrt{2}} & \sum_{i=1}^{D-1}\sum_{j=i+1}^{D} q_i q_j \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \sum_{i=1}^{D} k_i \\ \sum_{i=1}^{D} \frac{k_i^2}{\sqrt{2}} \\ \sum_{i=1}^{D-1}\sum_{j=i+1}^{D} k_i k_j \end{bmatrix} =$$

$$= 1 + \sum_{i=1}^{D} q_i k_i + \sum_{i=1}^{D} \frac{q_i^2}{\sqrt{2}} \frac{k_i^2}{\sqrt{2}} + \sum_{i=1}^{D-1}\sum_{j=i+1}^{D} q_i q_j k_i k_j = 1 + q^T k + \frac{(q^T k)^2}{2}$$

this way we know $\phi(x)$ is correct.

to calculate M (dimensionality of $\phi(x)$ in order of D):

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \dots x_D \\ \frac{x_1^2}{\sqrt{2}} \dots \frac{x_D^2}{\sqrt{2}} \\ x_1x_2 \dots x_1x_D \\ (\dots) \\ x_{D-1}x_D \end{bmatrix}$$

$1 \rightarrow 1$

$x_1 \dots x_D \rightarrow D$

$\frac{x_1^2}{\sqrt{2}} \dots \frac{x_D^2}{\sqrt{2}} \rightarrow D$

$x_1x_2 \dots x_1x_D \rightarrow D-2+1 = D-1$ (we went from x_1x_2 to x_1x_D inclusively so we have $D-2+1$ terms)

$x_{D-1}x_D \rightarrow D-(D-1) = 1$ (*)

(*) for all terms of $\sum_{i=1}^{D-1}\sum_{j=i+1}^{D} x_i x_j$: $(D-1) + (D-2) + (D-3) + \dots + 1$
this can be expressed as $\sum_{i=1}^{D-1} D-i$

the closed form of $\sum_{i=1}^{D-1} D-i = \frac{D(D+1)}{2} - D$

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

for terms $k \geq 3$, M is calculated the same way except it follows a different approximation of the Mclaurin series by expanding it further until K terms.

if $\exp(t) \approx 1 + t + \frac{t^2}{2} + \frac{t^3}{6}$ then $M > 1 + D + \frac{D(D+1)}{2}$

we need to turn $\frac{(q^T k)^3}{6}$ into a sum of terms that can be seperated into q and k like we did above and then add new rows to $\phi(x)$ that describe them. afterwards, we would compute M just like we did before

to generalize for K terms we would have to do this for every single term of $\exp(q^T k)$ up until K, since the dimensionality for a given term K is the sum of all dimensions from 0 to $k-1$ plus it's own (K).
to make this process more feasable we can apply the Multinomial theorem:

$$(x_1 + x_2 + \ldots + x_m)^n = \sum_{k_1 + k_2 + \ldots + k_m = n;\ k_1, k_2, \ldots, k_m \geq 0} \binom{n}{k_1, k_2, \ldots, k_m} \prod_{t=1}^{m} x_t^{k_t},$$

$$\text{where } \binom{n}{k_1, k_2, \ldots, k_m} = \frac{n!}{k_1! k_2! \cdots k_m!}$$

since every term of $\exp(q^T k)$ can be represented by sums this theorem describes the value of $\frac{(q^T k)^n}{n!}$ for a specific value $n \in [0, k-1]$

$$\frac{(q^T k)^n}{n!} = \frac{1}{n!} \sum_{\substack{x_1 + x_2 + \ldots + x_D = n \\ x_1 + \ldots x_D \geq 0}} \left(\frac{n!}{x_1! x_2! \cdots x_D!} \right) \prod_{t=1}^{D} (q_t \cdot k_t)^{x_t}$$

to calculate the value of $\exp(q^T k)$ extended until a certain value K we can use:

$$\sum_{n=0}^{K-1} \frac{(q^T k)^n}{n!} = \sum_{n=0}^{K-1} \frac{1}{n!} \left(\sum_{\substack{x_1 + x_2 + \dots + x_D = n \\ x_1 \dots x_D \geq 0}} \left(\frac{n!}{x_1! x_2! \dots x_D!} \right) \prod_{t=1}^{D} (q_t k_t)^{x_t} \right)$$

the dimensionality of $\sum_{n=0}^{K-1} \frac{(q^T k)^n}{n!}$ corresponds to the total of sums made, and in Multinomial Theorem that is the sum from $n=0$ until $K-1$ of the number of multinomial coefficients, and can be represented as

$$\sum_{n=0}^{K-1} \binom{n+D-1}{D-1} = \sum_{n=0}^{K-1} \frac{(n+D-1)!}{(D-1)!\, n!}$$

The Multinomial Theorem[1] was used in the resolution of this exercise.

[1] Multinomial Theorem, retrieved on 29th December 2023, by Wikipedia contributors at *Wikipedia, The Free Encyclopedia*. Last Revision on 3rd November 2023. https://en.wikipedia.org/wiki/Multinomial_theorem

3.

$$Z \approx D^{-1}\phi(Q)\phi(K)^T V \qquad Z = \text{Softmax}(QK^T)V$$

$$\exp(q^T k) \approx \phi(q)^T \cdot \phi(k)$$

$$\text{Softmax}(XY) = \frac{\exp(XY)}{\sum_{i=1}^{n} \exp(x_i y_j)}$$

$$\text{Softmax}(QK^T)V \approx D^{-1}\phi(Q)\phi(K)^T V \iff$$

$$\iff \text{Softmax}(QK^T) \approx D^{-1}\phi(Q)\phi(K)^T \iff$$

$$\iff \frac{\exp(QK^T)}{\sum_{j=1}^{L} \exp(Q_i k_j)} \approx \frac{\phi(Q)\phi(K)^T}{D} \iff D^{-1} = \frac{1}{D} \text{ because } D \text{ is a diagonal matrix}$$

$$\iff \exp(QK^T) \approx \phi(Q)\phi(K)^T \wedge \sum_{j=1}^{L} \exp(Q_i k_j) \approx D$$

$$\iff \text{''} \wedge \sum_{j=1}^{n} \exp(Q_i k_j) \approx \text{Diag}(\phi(Q)\phi(K)1_L)$$

for $\exp(QK^T) \approx \phi(Q)\phi(K)^T$:

- $q_1 \dots q_L$ and $k_1 \dots k_L \in \mathbb{R}^D$
- $\phi(q_1) \dots \phi(q_L)$ and $\phi(k_1) \dots \phi(k_L) \in \mathbb{R}^M$

$$\underset{(\mathbb{R}^{L\times M})}{\phi(Q)} = \begin{bmatrix} \phi(q_1)_1 & \phi(q_1)_2 & \dots & \phi(q_1)_M \\ \phi(q_2)_1 & \phi(q_2)_2 & \dots & \phi(q_2)_M \\ & & \dots & \\ \phi(q_L)_1 & \phi(q_L)_2 & \dots & \phi(q_L)_M \end{bmatrix}$$

$$\underset{(\mathbb{R}^{M\times L})}{\phi(K)^T} = \begin{bmatrix} \phi(k_1)_1 & \phi(k_2)_1 & \dots & \phi(k_L)_1 \\ \phi(k_1)_2 & \phi(k_2)_2 & \dots & \phi(k_L)_2 \\ & & \dots & \\ \phi(k_1)_M & \phi(k_2)_M & \dots & \phi(k_L)_M \end{bmatrix}$$

$$\underset{(\mathbb{R}^{L\times L})}{\phi(Q)\cdot\phi(K)^T} = \begin{bmatrix} \phi(q_1)_1\phi(k_1)_1 + \ldots + \phi(q_1)_M\phi(k_1)_M & (\ldots) & \phi(q_1)_1\phi(k_L)_1 + \ldots + \phi(q_1)_M\phi(k_L)_M \\ \phi(q_2)_1\phi(k_1)_1 + \ldots + \phi(q_2)_M\phi(k_1)_M & (\ldots) & \phi(q_2)_1\phi(k_L)_1 + \ldots + \phi(q_2)_M\phi(k_L)_M \\ & (\ldots) & \\ \phi(q_L)_1\phi(k_1)_1 + \ldots + (q_L)_M\phi(k_1)_M & (\ldots) & \phi(q_L)_1\phi(k_L)_1 + \ldots + \phi(q_L)_M\phi(k_L)_M \end{bmatrix} =$$

$$= \begin{bmatrix} \sum_{\omega=1}^{M}\phi(q_1)_\omega\phi(k_1)_\omega & \sum_{\omega=1}^{M}\phi(q_1)_\omega\phi(k_2)_\omega & (\ldots) & \sum_{\omega=1}^{M}\phi(q_1)_\omega\phi(k_L)_\omega \\ \sum_{\omega=1}^{M}\phi(q_L)_\omega\phi(k_1)_\omega & \sum_{\omega=1}^{M}\phi(q_L)_\omega\phi(k_2)_\omega & (\ldots) & \sum_{\omega=1}^{M}\phi(q_L)_\omega\phi(k_L)_\omega \end{bmatrix} =$$

$$= \begin{bmatrix} \phi(q_1)\cdot\phi(k_1) & \phi(q_1)\cdot\phi(k_2) & (\ldots) & \phi(q_1)\cdot\phi(k_L) \\ & & (\ldots) & \\ \phi(q_L)\cdot\phi(k_1) & \phi(q_L)\cdot\phi(k_2) & (\ldots) & \phi(q_L)\cdot\phi(k_L) \end{bmatrix}$$

$\phi(Q)\cdot\phi(K)^T$ is a $\mathbb{R}^{L\times L}$ matrix where each entry is $\phi(q_i)\cdot\phi(k_j)$ being i and j the #row and #column, respectively.
We know $\exp(q^Tk) \approx \phi(q)^T\phi(k)$ and with the demonstration above we can also conclude that $\exp(QK^T) \approx \phi(Q)\cdot\phi(K)^T$

for $\sum_{j=1}^{L}\exp(Q_i K_j) \approx \text{Diag}\left(\phi(Q)\,\phi(K)^T 1_L\right)$:

from the above calculations $\text{Diag}\left(\phi(Q)\,\phi(K)^T 1_L\right) \approx \text{Diag}\left(\exp(QK^T)1_L\right)$

$\phi(Q)\phi(K)\cdot 1_L$ results in a $\mathbb{R}^L$ column in which each entry$_i$ is the sum of all entries of the row$_i$ from $\phi(Q)\,\phi(K)$: $\sum_{j=1}^{L}\phi(q_i)\cdot\phi(k_j)^T$

$$\phi(Q)\phi(K)1_L = \begin{bmatrix} \sum_{j=1}^{L}\phi(q_1)\cdot\phi(k_j) \\ (\ldots) \\ \sum_{j=1}^{L}\phi(q_L)\cdot\phi(k_j) \end{bmatrix}$$

$$\underset{(\mathbb{R}^{L\times L})}{\text{Diag}}\left(\phi(Q)\phi(K)1_L\right) = \begin{bmatrix} \sum_{j=1}^{L}\phi(q_1)\cdot\phi(k_j)^T & (\ldots) & 0 \\ & (\ldots) & \\ 0 & (\ldots) & \sum_{j=1}^{L}\phi(q_L)\cdot\phi(k_j)^T \end{bmatrix}$$

$\text{Diag}(\phi(Q)\phi(K)1_L)$ allows each entry of $D^{-1}\cdot\phi(Q)\cdot\phi(K)^T$ to be $\frac{\phi(q_i)\cdot\phi(k_j)^T}{\sum_{j=1}^{L}\phi(q_i)\cdot\phi(k_j)^T}$, which is the Softmax formula $\frac{\exp(Q_i K^T)}{\sum_{j=1}^{L}\exp(Q_i K_j)}$

afterwards we multiply $D^{-1}\phi(Q)\phi(K)^T$ by V and obtain Z

$$\text{Diag}(\phi(Q)\,\phi(K)\,1_L) \approx \sum_{j=1}^{L}\exp(Q_i K_j),$$

$$\text{Softmax}(QK^T) \approx D^{-1}\phi(Q)\phi(K)^T,$$

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

4.

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

- $D^{-1} \in \mathbb{R}^{L \times L}$
- $\phi(Q) \in \mathbb{R}^{L \times M}$
- $\phi(K)^T \in \mathbb{R}^{M \times L}$
- $V \in \mathbb{R}^{L \times D}$

- we can use the rule of associativity to multiply the matrices in a order that's most convenient, so we'll start by $\phi(K)^T \cdot V$

- the result of $\phi(K)^T \cdot V$ is a $\mathbb{R}^{M \times D}$ matrix. to compute this we have a complexity of $O(LMD)$ because each entry is made by L multiplications and there are $M \cdot D$ entries

- afterwards we can proceed to $\phi(Q) \cdot (\phi(K)^T \cdot V)$ which will output a matrix $\in \mathbb{R}^{L \times D}$. the complexity will be $O(MLD)$ because each entry is made by M multiplications and there are $L \cdot D$ entries

- if we were to multiply $D^{-1} \cdot (\phi(Q) \cdot \phi(K)^T \cdot V)$ we would get an $O(L^2 M)$ complexity even though in D^{-1} we only have L non-zero elements in a total of L^2 elements.

since D is the diagonalization of a column $\mathbb{R}^L$ we can invert the values in the column and have the non-zero values of D^{-1} (for simplicity purposes let's name it S)

$$\underset{(\mathbb{R}^L)}{S} = \begin{bmatrix} \frac{1}{\sum_{j=1}^{L} \phi(q_1) \cdot \phi(k_j)} \\ (\dots) \\ \frac{1}{\sum_{j=1}^{L} \phi(q_L) \cdot \phi(k_j)} \end{bmatrix}$$

since we can't multiply $\mathbb{R}^L$ by $\mathbb{R}^{L\times D}$ and multiplying $\mathbb{R}^{L\times L}$ by $\mathbb{R}^{L\times D}$ gives us a complexity of $O(L^2 \cdot D)$ we can exploit this complexity by using Hadamard between a certain matrix and $\phi(Q)\phi(K)^T V$ in a way that the output is the same as multiplying D^{-1} by $\phi(Q)\phi(K)^T V$

that certain matrix will be S but in $\mathbb{R}^{L\times D}$ dimension, in which all M columns are the same. (for simplicity let's name it H).

$$\underset{(\mathbb{R}^{L\times D})}{H} = \begin{bmatrix} \frac{1}{\sum_{j=i}^{L} \phi(q_i)\cdot\phi(k_j)} & (\cdots) & \frac{1}{\sum_{j=i}^{L} \phi(q_i)\cdot\phi(k_j)} \\ & (\cdots) & \\ \frac{1}{\sum_{j=1}^{L} \phi(q_L)\cdot\phi(k_j)} & (\cdots) & \frac{1}{\sum_{j=1}^{L} \phi(q_L)\cdot\phi(k_j)} \end{bmatrix}$$

we do this because multiplying a diagonal matrix (A) by any other matrix (B) is basically multiplying the non-zero value of a certain row$_i$ of A by all entries of the row$_i$ of B.

therefore by doing $H \odot (\phi(Q)\phi(K)^T \cdot V)$ we will get the same output of $D^{-1} \cdot (\phi(Q)\phi(K)^T V)$ without compromising complexity being quadratic in L since the output of $\mathbb{R}^{L\times D} \odot \mathbb{R}^{L\times D}$ has a complexity of $O(LD)$ and therefore is linear in L

- even though they will not dominate the computation complexity of Z we should calculate the complexity of auxiliary values and steps:

- computing $\phi(Q)$ and $\phi(K)$ are $O(LM)$ since we have to compute phi(q_i) and phi(k_i) which are $\mathbb{R}^M$ for all L rows

- computing $\phi(K)^T 1_L$ is $O(LM)$ since it will result in a vector $\in \mathbb{R}^M$ where each entry is made by L multiplications

- computing $\phi(Q)(\phi(K)^T 1_L)$ is $O(ML)$ since it will result in vector $\in \mathbb{R}^L$ where each entry is made by M multiplications

- computing S is $O(L)$ because we are inverting L values since $\phi(Q)\phi(K)^T 1_L \in \mathbb{R}^L$

- computing H is $O(D)$ since we are creating a matrix $\in \mathbb{R}^{L \times D}$ composed of D copies of the same vector

- the overall complexity of Z will be:

$$O(LMD) + O(MLD) + O(LD) + O(LM) + O(LM) + O(LM) + O(ML) + O(L) + O(D) =$$

$$= O(LMD) + O(LD) + O(LM) + O(L) + O(D)$$

which will be dominated by $O(LMD)$ and is linear in L

Question 2

1.

- Learning rate (η): 0.1

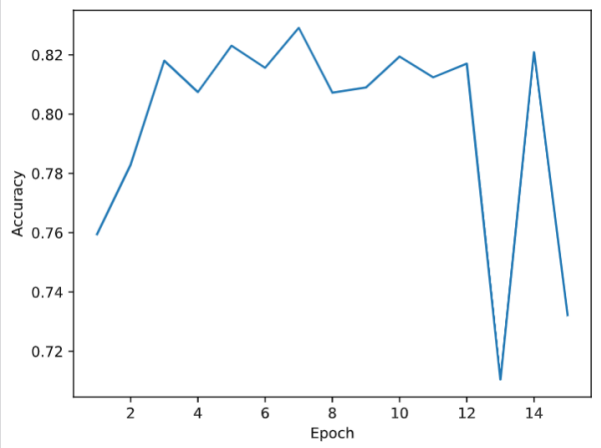

Figure 1 – CNN with max pooling layers and with learning rate 0.1: validation accuracy as a function of the epoch number.

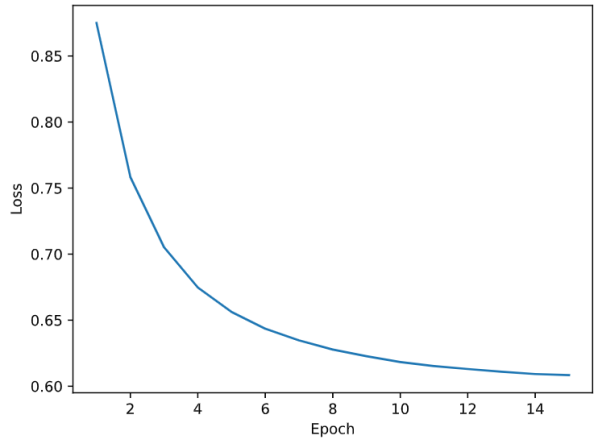

Figure 2 – CNN with max pooling layers and with learning rate 0.1: training loss.as a function of the epoch number.

The performance with learning rate 0.1 was the following: The final validation accuracy was 0.7322 with a final training loss of 0.6084 and with a final test accuracy of 0.7013.

- Leaning rate ((η): 0.01

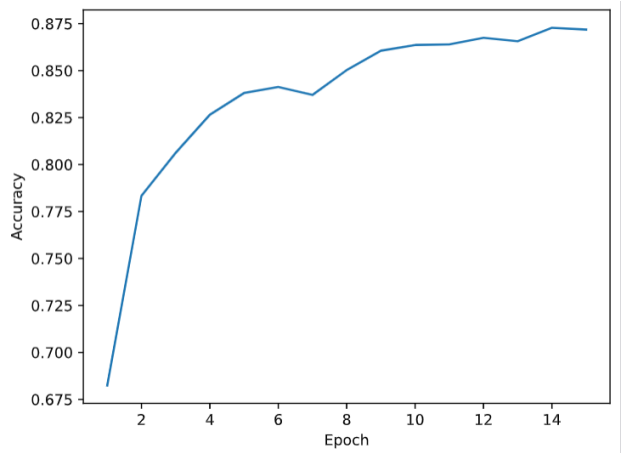

Figure 3 – CNN with max pooling layers and with learning rate 0.01: validation accuracy as a function of the epoch number.

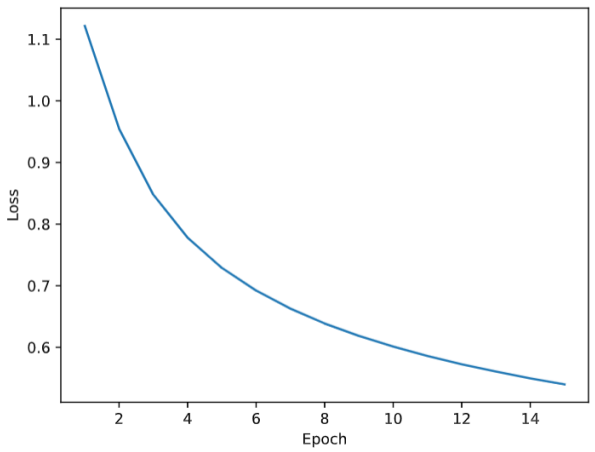

Figure 4 – CNN with max pooling layers and with learning rate 0.01: training loss.as a function of the epoch number.

The performance with learning rate 0.01 was the following: The final validation accuracy was 0.8719 with a final training loss of 0.5400 and with a final test accuracy of 0.8280.

- Learning rate ((η): 0.001

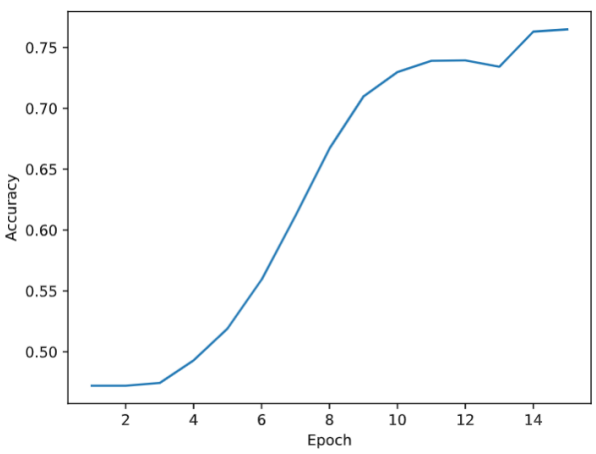

Figure 5 – CNN with max pooling layers and with learning rate 0.001: validation accuracy as a function of the epoch number.

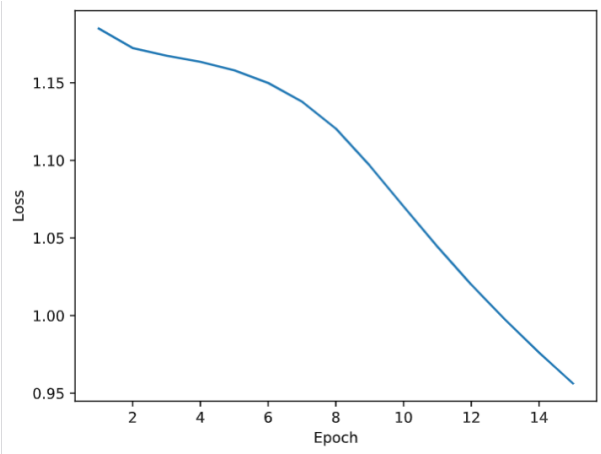

Figure 6 – CNN with max pooling layers and with learning rate 0.001: training loss.as a function of the epoch number.

The performance with learning rate 0.001 was the following: The final validation accuracy was 0.7649 with a final training loss of 0.9563 and with a final test accuracy of 0.7505.

After training the model for 15 epochs using SGD with different values of learning rate: 0.1, 0.01 and 0.001, by looking at the plots and the final test accuracy of 0.8280 (0.01) compared to 0.7013 (0.1) and 0.7505 (0.001), the learning rate of the best configuration was 0.01.

2.

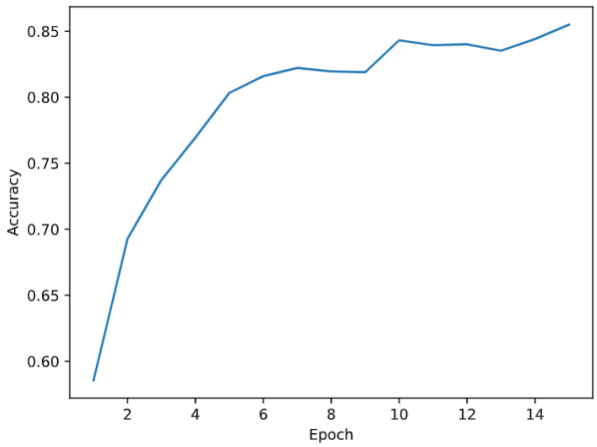

Figure 7 – CNN without max pooling layers and with learning rate 0.01: validation accuracy as a function of the epoch number.

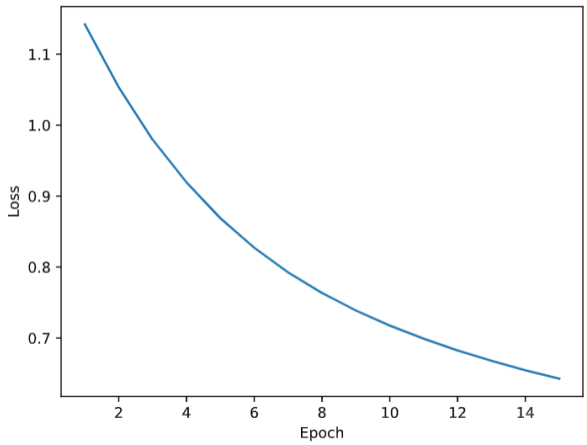

Figure 8 – CNN without max pooling layers and with learning rate 0.01: training loss.as a function of the epoch number.

The optimal hyper-parameters defined in the previous question was with learning rate of 0.01. The performance of this network was the following: The final validation accuracy was 0.8550 with a final training loss of 0.6426 and with a final test accuracy of 0.7958.

3.

Using the function get number trainable parameters, we can observe that the number of trainable parameters from the previous two questions were the exact same, 224892 parameters. The network 1 uses max pooling layers (2x2, stride 2) after each convolutional layer while network 2 doesn't have any. However, even though network 1's and network 2's two convolutional layers share the same output channels, kernel size and padding, they have a different stride in its convolutional layers, with network 1 having stride 1 and network 2 having stride 2. Both convolution stride and max pooling lead to reducing the spatial dimensions but in different ways.

By using stride 2 or max pooling (2x2, stride 2) there was a difference in performance between the two of them. With the same hyper-parameters, network 1 performs better than network 2. The reason being that features tend to encode spatial presence of some pattern or concept over the different tiles of the feature map, and it's more informative to look at the most dominant features while discarding the less relevant ones (suppressing the noise in the input data) which can aid in better generalization and thus better performance (max pooling) than at their general spatial representation (convolution stride) because convolution stride skips pixels without explicit feature selection. Additionally, by capturing the presence of dominant patterns across the feature maps by selecting the maximum value in each pooling window, max pooling helps the model to became invariant to the location and orientation of the features, which means that the network can recognize an object in an image no matter where it is located, while convolution stride directly changes the spatial resolution within convolution layers without feature selection which may lead to less relevant features not being discarded.

Question 3

1.

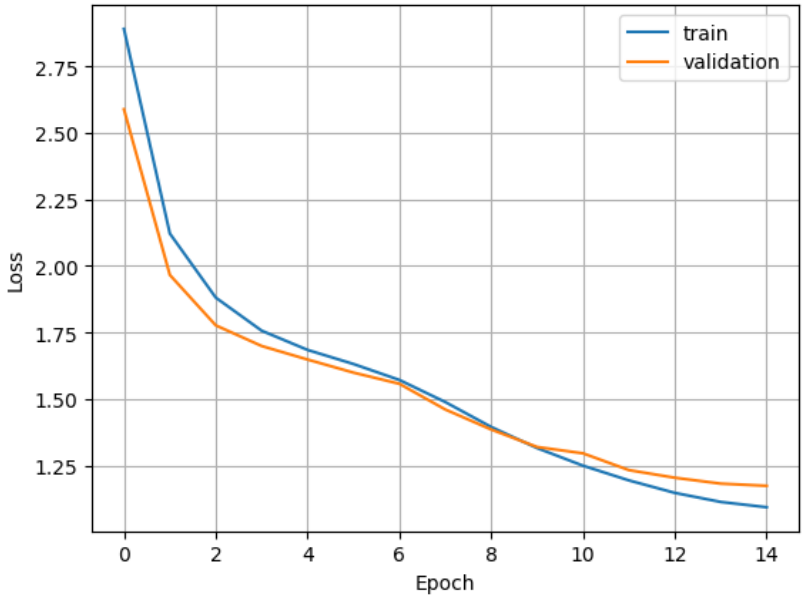

Figure 9 – ASR with recurrent-based approach: training and validation loss over epochs.

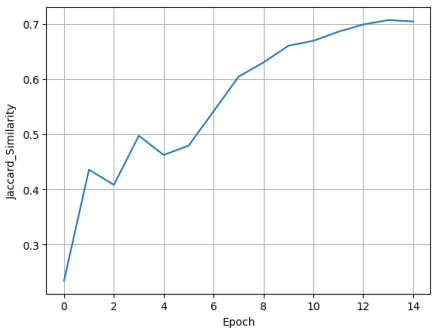

Figure 10 – ASR with recurrent-based approach: Jaccard Similarity obtained in the validation set over epochs.

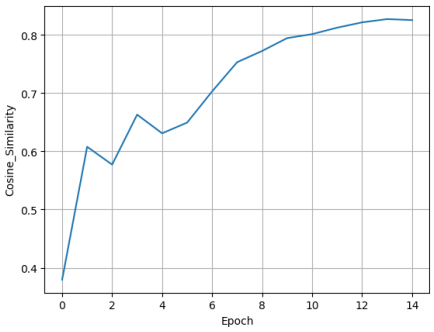

Figure 11 – ASR with recurrent-based approach: Cosine Similarity obtained in the validation set over epochs.

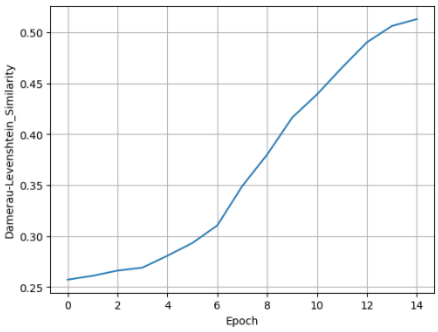

Figure 12 – ASR with recurrent-based approach: Damerau-Levenstein Similarity obtained in the validation set over epochs.

The final test loss was 1.1828280331158056 and the string similarity scores obtained with the test set were:

- Jaccard Similarity: 0.714888785108462
- Cosine Similarity: 0.8323868545583469
- Damerau-Levenshtein Similarity: 0.5086983165244969

2.

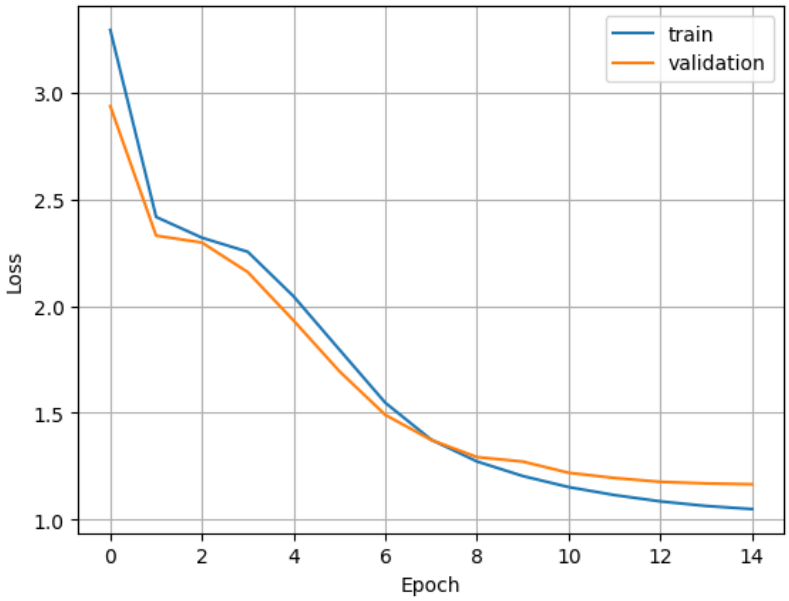

Figure 13 – ASR with attention-based mechanism: training and validation loss over epochs.

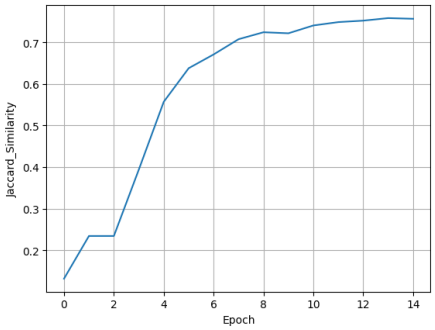

Figure 14 – ASR with attention-based mechanism: Jaccard Similarity obtained in the validation set over epochs.

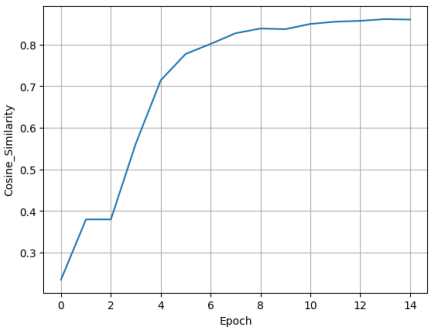

Figure 15 – ASR with attention-based mechanism: Cosine Similarity obtained in the validation set over epochs.

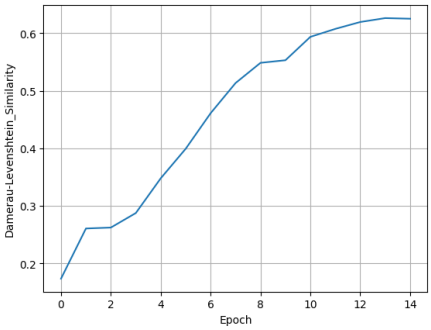

Figure 16 – ASR with attention-based mechanism: Damerau-Levenstein Similarity obtained in the validation set over epochs.

The final test loss was 1.1603623446894855 and the string similarity scores obtained with the test set were:

- Jaccard Similarity: 0.7656637423081285
- Cosine Similarity: 0.8661194847755104
- Damerau-Levenshtein Similarity: 0.6343659405293635

3.

The decoder structure works in the same way in both questions except in the point 2 (question 1) and 3 (question 2) which is where the token embeddings are processed. In question 1, it passes through a normalization layer followed by an LSTM, while in the question 2, it passes through an attention layer after a position embeddings matrix is added to the token embeddings.

LSTM is a type of recurrent neural network (RNN) designed to capture long-range dependencies and handle the vanishing gradient problem present in traditional RNNs. In the LSTM layer, it processes these embeddings sequentially, maintaining an internal state that captures information from previous tokens. It generates output at each step and passes it to the next step, allowing it to learn complex temporal dependencies.

Attention mechanism allows the model to focus on different parts of the input sequence when generating each part of the output sentence. In the attention layer, it processes the token embeddings along with position embeddings. It attends to different parts of the input sequence in parallel, leveraging attention weights that represent the importance of each token relative to the current token and incorporates a masking matrix to prevent attending to future tokens. This allows the model to focus on relevant information while decoding.

By using different mechanisms to process the input in the decoder, the test results were different. Since it utilizes scaled dot-product attention with multiple heads, allowing the model to selectively focus on different parts of the input sequence in parallel, the training was faster using attention mechanism than using LSTM.

The attention-based decoder performed better than LSTM in the testing, by having a smaller loss and scoring higher when comparing Jaccard, Cosine and Damerau-Levenshtein Similarity scores from both implementations, due to its enhanced ability to capture complex dependencies and context in the data through the parallelized and refined attention mechanisms, leveraging information more effectively by exploiting the context from the encoder compared to the sequential processing of tokens of the LSTM-based decoder.

4.

Each string similarity metric used has their own evaluation criteria to measure the similarity between strings which leads to their respective scores, since they capture distinct aspects of the alignment between the generated output and the reference transcription.

In both implementations (question 1 and 2), Cosine Similarity has a higher value, followed by Jaccard and then Damerau-Levenshtein. All values of each string similarity metric are higher with attention-based mechanism than with recurrent-based approach implementation.

Cosine Similarity: Calculates the similarity based on the cosine angle between two vectors, part of the same inner product space, representing the strings in a vector space model and determines whether two vectors are pointing in roughly the same direction. This makes it effective for measuring similarity in terms of orientation or direction rather than magnitude or scale. Cosine Similarity Measure is widely used in Sentiment Analysis tasks within Natural Language Processing (NLP), due to its effectiveness in measuring semantic similarity between several words or sentences. In our application, the encoder's input is an audio signal in the form of a mel-scale conversion of a spectrogram. The mel-scale features not only accommodate variations in speaking styles but also reduce the dimensionality of the input. This aligns well with a Cosine Similarity strength: the ability to measure the similarity of vectors' directions in a vector space, and not their magnitudes. Cosine Similarity is also the only measure among the three we evaluated that inherently captures semantic analysis, an extremely important aspect of Automatic Speech Recognition. Due to these considerations, Cosine Similarity placed first among the measures evaluated, in both implementations.

Jaccard Similarity: Calculates the similarity of two sets of data to see which members are shared and distinct. The calculation is done by dividing the number of observations in both sets by the number of observations in either set. This makes it particularly sensitive to the overlap elements and it is great for when order doesn't matter. It is not commonly employed in Automatic Speech Recognition due to the inherent sequential nature of spoken words, which makes this measure less suitable for this application. Jaccard Similarity is frequently applied in Natural Language Processing tasks, such as determining the similarity between two text documents. This involves comparing sets of words from both documents, which is beneficial for practical tasks such as document clustering. Jaccard Similarity obtained acceptable scores in both implementations and placed second among the similarity measures evaluated.

Damerau-Levenshtein Similarity: Calculates the similarity by counting the minimum number of operations (insertions, deletions, substitutions, transpositions) needed to transform one string to another. This makes it useful to capture the structural similarity between strings and it is sensitive to the sequence of characters. It stands out in practical applications such as "autocorrection", due to its ability to primarily capture the structural similarities among strings or words. It can, for instance, be valuable in the context of comparing a misspelled word with a structurally similar correct word, facilitating the correct replacement of the wrong word with the correct one. In the context of an Automatic Speech Recognition application like this exercise, the structural similarities captured by Damerau-Levenshtein Similarity may be less relevant and effective in correctly identifying the words said. This is emphasized by its average score in both implementations, placing the Damerau-Levenshtein Similarity measure in third among the similarity measures evaluated.