# Deep Learning (IST, 2024-25)

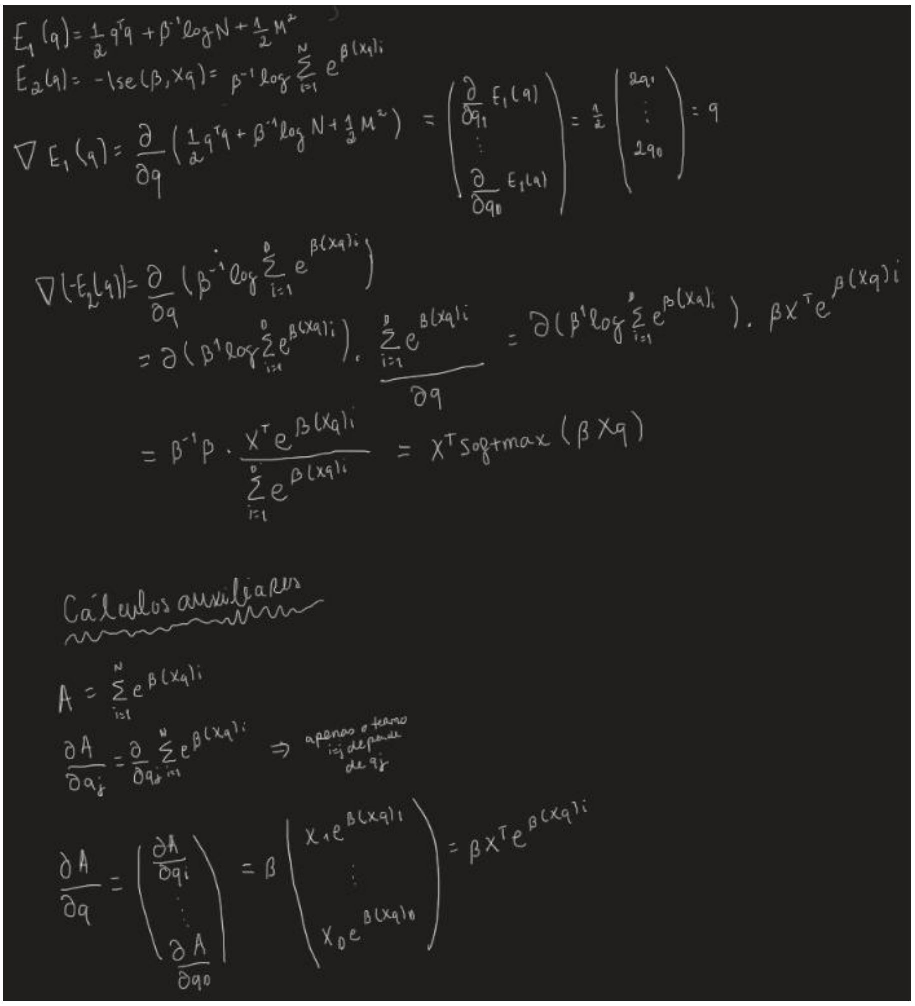
# Homework 2

Diogo Ribeiro
ist1102484

Vasco Paisana
ist1102533

## Question 1

**1.**



$$E_1(q) = \frac{1}{2}q^Tq + \beta^{-1}\log N + \frac{1}{2}M^2$$

$$E_2(q) = -\text{lse}(\beta, Xq) = \beta^{-1}\log\sum_{i=1}^{N} e^{\beta(Xq)_i}$$

$$\nabla E_1(q) = \frac{\partial}{\partial q}\left(\frac{1}{2}q^Tq + \beta^{-1}\log N + \frac{1}{2}M^2\right) = \begin{pmatrix} \frac{\partial}{\partial q_1}E_1(q) \\ \vdots \\ \frac{\partial}{\partial q_D}E_1(q) \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 2q_1 \\ \vdots \\ 2q_D \end{pmatrix} = q$$

$$\nabla(-E_2(q)) = \frac{\partial}{\partial q}\left(\beta^{-1}\log\sum_{i=1}^{D} e^{\beta(Xq)_i}\right)$$

$$= \partial\left(\beta^{-1}\log\sum_{i=1}^{D} e^{\beta(Xq)_i}\right) \cdot \frac{\sum_{i=1}^{D} e^{\beta(Xq)_i}}{\partial q} = \partial\left(\beta^{-1}\log\sum_{i=1}^{D} e^{\beta(Xq)_i}\right) \cdot \beta X^T e^{\beta(Xq)_i}$$

$$= \beta^{-1}\beta \cdot \frac{X^T e^{\beta(Xq)_i}}{\sum_{i=1}^{D} e^{\beta(Xq)_i}} = X^T \text{softmax}(\beta Xq)$$

Cálculos auxiliares

$$A = \sum_{i=1}^{N} e^{\beta(Xq)_i}$$

$$\frac{\partial A}{\partial q_j} = \frac{\partial}{\partial q_j}\sum_{i=1}^{N} e^{\beta(Xq)_i} \Rightarrow \text{apenas o termo } i{=}j \text{ depende de } q_j$$

$$\frac{\partial A}{\partial q} = \begin{pmatrix} \frac{\partial A}{\partial q_1} \\ \vdots \\ \frac{\partial A}{\partial q_D} \end{pmatrix} = \beta\begin{pmatrix} X_1 e^{\beta(Xq)_1} \\ \vdots \\ X_D e^{\beta(Xq)_D} \end{pmatrix} = \beta X^T e^{\beta(Xq)_i}$$

$$H\,E_1(q) = \begin{bmatrix} \frac{\partial^2}{\partial q_1^2} E_1(q) & \cdots & \frac{\partial^2}{\partial q_1 \partial_D} E_1(q) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial q_D q_1} E_1(q) & \cdots & \frac{\partial^2}{\partial q_D^2} E_1(q) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial q_1} \cdot \frac{\partial}{q_1} E_1(q) & \cdots & \frac{\partial}{q_1} \frac{\partial}{q_D} E_1(q) \\ & \ddots & \\ \frac{\partial}{\partial q_D} \frac{\partial}{\partial q_1} E_1(q) & \cdots & \frac{\partial}{\partial q_D} \frac{\partial}{q_D} E_1(q) \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ 0 & 1 & & & \vdots \\ \vdots & & \ddots & & \vdots \\ & & & 1 & 0 \\ 0 & \cdots & & 0 & 1 \end{bmatrix} = I$$ , that is positive semi-definite

$$H \cdot E_2(q) = \begin{bmatrix} \frac{\partial^2}{\partial q_1^2} \cdot E_2(q) & \cdots & \frac{\partial^2}{\partial q_1 \partial_D} \cdot E_2(q) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial q_D q_1} \cdot E_2(q) & \cdots & \frac{\partial^2}{\partial q_D^2} \cdot E_2(q) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial}{\partial q_1} \cdot \frac{\partial}{q_1} \cdot E_2(q) & \cdots & \frac{\partial}{q_1} \frac{\partial}{q_D} \cdot E_2(q) \\ & \ddots & \\ \frac{\partial}{\partial q_D} \frac{\partial}{\partial q_1} \cdot E_2(q) & \cdots & \frac{\partial}{\partial q_D} \frac{\partial}{q_D} \cdot E_2(q) \end{bmatrix}$$

$$\frac{\partial}{\partial q_j} \underbrace{\frac{X^T e^{\beta (Xq)_i}}{\sum_{i=1}^{D} e^{\beta (Xq)_i}}}_{\Sigma} = \begin{cases} \dfrac{\beta X_i^2 e^{\beta (Xq)_i}\, \Sigma - \beta X_i e^{\beta (Xq)_i} \cdot X_i e^{\beta (Xq)_i}}{\Sigma^2}, & i = j \\[2ex] \dfrac{0 - \beta X_j e^{\beta (Xq)_j} \cdot X_i e^{\beta (Xq)_i}}{\Sigma^2}, & i \neq j \end{cases}$$

$$= \begin{cases} \dfrac{\beta X_i^2 e^{\beta (Xq)_i} \left( \Sigma - e^{\beta (Xq)_i} \right)}{\Sigma^2}, & i = j \\[2ex] \dfrac{-\beta X_j e^{\beta (Xq)_j} \cdot X_i e^{\beta (Xq)_i}}{\Sigma^2}, & i \neq j \end{cases}$$

$$= \begin{cases} \beta X_i^2 \operatorname{softmax}(\beta (Xq)_i) \left(1 - \operatorname{softmax}(\beta (Xq)_i)\right), & i = j \\[2ex] \beta X_i X_j \left(-\operatorname{softmax}(\beta (Xq)_j) \operatorname{softmax}(\beta (Xq)_i)\right), & i \neq j \end{cases} = \begin{cases} \beta X_i^2 p_i (1 - p_i), & i = j \\ \beta X_i X_j (-p_j p_i), & i \neq j \end{cases}$$

As a square matrix A is said to be diagonally dominant if $|A_{ii}| \geq \sum_{j \neq i} |A_{ij}|$, for any i, and if A is symmetric and diagonally dominant with non negative diagonal elements is positive semi-definite then we just need to prove that $H - E_2(q)$ has these properties

$H = \beta X^T \underbrace{(\mathrm{diag}(p) - pp^T)}_{S} X$ , being p the softmax probabilities

$$\sum_{j \neq i} S_{ij} = \sum_{j \neq i} p_j p_i = p_i \sum_{j \neq i} p_j$$

as $\sum p = 1$

$$\sum_{j \neq i} p_j = 1 - p_i \quad , \quad \sum_{j \neq i} S_{ij} = p_i (1 - p_i)$$

Thus, $\sum_{j \neq i} S_{ij} = S_{ii}$ , satisfying diagonally dominance

S is a symmetric matrix
S has non-negative diagonal values $(0 \leq p_i \leq 1)$
∴ S is positive semi-definite

A matrix A is PSD if for any non-zero vector $v$

$$v^T A v \geq 0$$

So we just need to make $u = Xv$

Having $H = u^T S u$ , since S is psd for any vector u we have

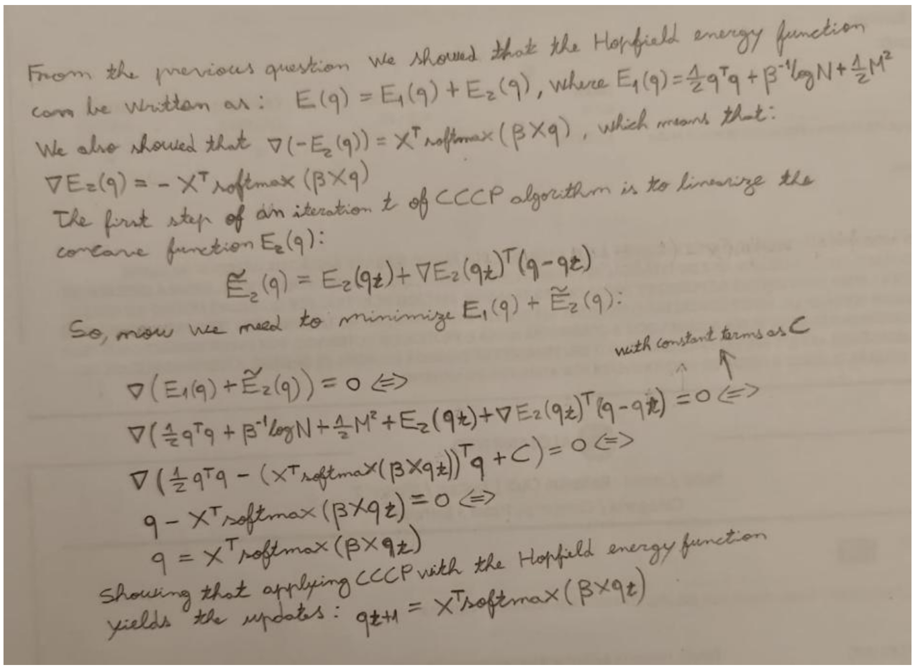
$$u^T S u \geq 0$$

which implies that

$$v^T X^T S X v \geq 0$$

Because $\beta > 0$ , $H - E_2(q)$ is psd

**2.**



From the previous question we showed that the Hopfield energy function can be written as: $E(q) = E_1(q) + E_2(q)$, where $E_1(q) = \frac{1}{2}q^Tq + \beta^{-1}\log N + \frac{1}{2}M^2$

We also showed that $\nabla(-E_2(q)) = X^T \text{softmax}(\beta X q)$, which means that:

$\nabla E_2(q) = -X^T \text{softmax}(\beta X q)$

The first step of an iteration $t$ of CCCP algorithm is to linearize the concave function $E_2(q)$:

$$\tilde{E}_2(q) = E_2(q_t) + \nabla E_2(q_t)^T(q - q_t)$$

So, now we need to minimize $E_1(q) + \tilde{E}_2(q)$:

with constant terms as $C$

$$\nabla\left(E_1(q) + \tilde{E}_2(q)\right) = 0 \iff$$

$$\nabla\left(\tfrac{1}{2}q^Tq + \beta^{-1}\log N + \tfrac{1}{2}M^2 + E_2(q_t) + \nabla E_2(q_t)^T(q - q_t)\right) = 0 \iff$$

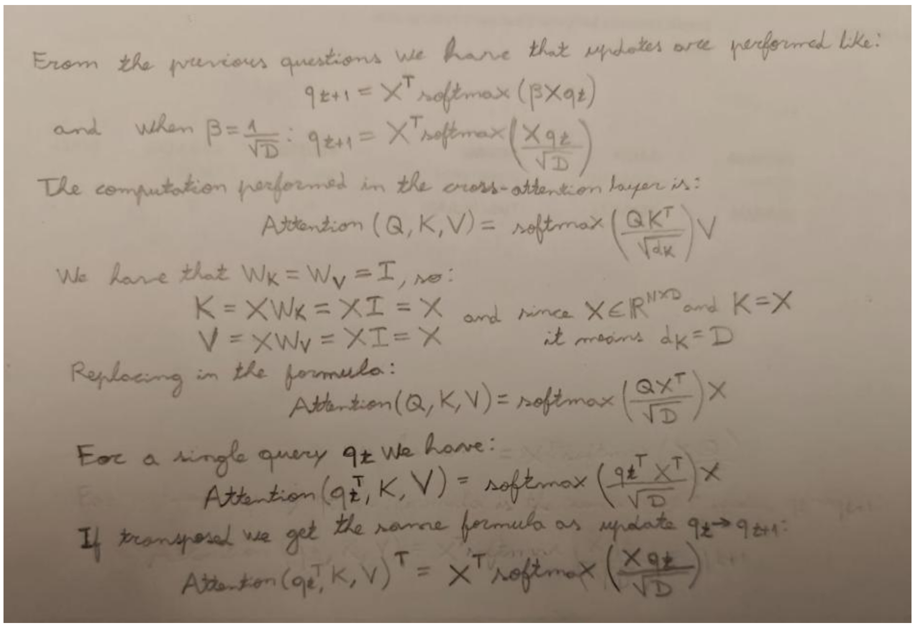
$$\nabla\left(\tfrac{1}{2}q^Tq - (X^T\text{softmax}(\beta X q_t))^T q + C\right) = 0 \iff$$

$$q - X^T\text{softmax}(\beta X q_t) = 0 \iff$$

$$q = X^T\text{softmax}(\beta X q_t)$$

Showing that applying CCCP with the Hopfield energy function yields the updates: $q_{t+1} = X^T\text{softmax}(\beta X q_t)$

**3.**



From the previous questions we have that updates are performed like:

$$q_{t+1} = X^T\text{softmax}(\beta X q_t)$$

and when $\beta = \frac{1}{\sqrt{D}}$: $q_{t+1} = X^T\text{softmax}\left(\frac{Xq_t}{\sqrt{D}}\right)$

The computation performed in the cross-attention layer is:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

We have that $W_K = W_V = I$, so:

$K = XW_K = XI = X$ and since $X \in \mathbb{R}^{N \times D}$ and $K = X$

$V = XW_V = XI = X$ it means $d_K = D$

Replacing in the formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QX^T}{\sqrt{D}}\right)X$$

For a single query $q_t$ we have:

$$\text{Attention}(q_t^T, K, V) = \text{softmax}\left(\frac{q_t^T X^T}{\sqrt{D}}\right)X$$

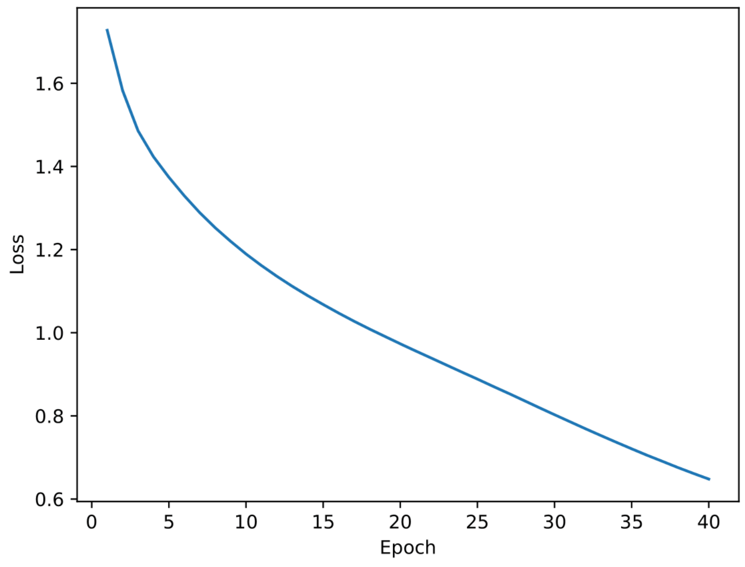
If transposed we get the same formula as update $q_t \to q_{t+1}$:

$$\text{Attention}(q_t^T, K, V)^T = X^T\text{softmax}\left(\frac{Xq_t}{\sqrt{D}}\right)$$

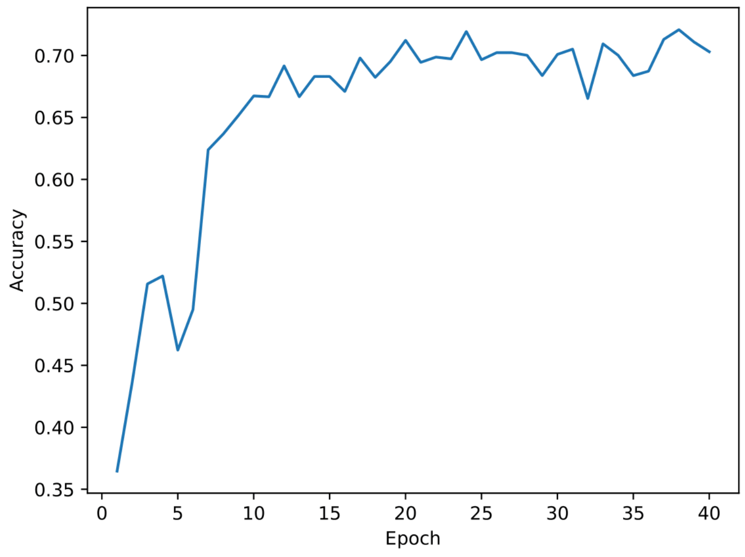
## Question 2

**1.** After training the model for 40 epochs using 3 different values for learning rate (0.1; 0.01; 0.001), we obtained that the learning rate of the best configuration is 0.01, with a final test accuracy of 0.6900, compared to lower final test accuracies for learning rates 0.1 (0.5543) and 0.001 (0.6057).

**Training loss as function of epoch number for the best configuration**

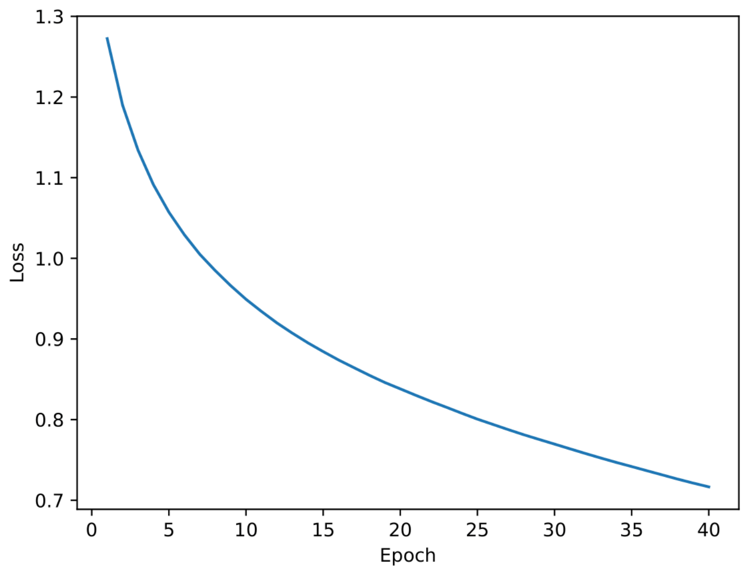


**Validation accuracy as function of epoch number for the best configuration**

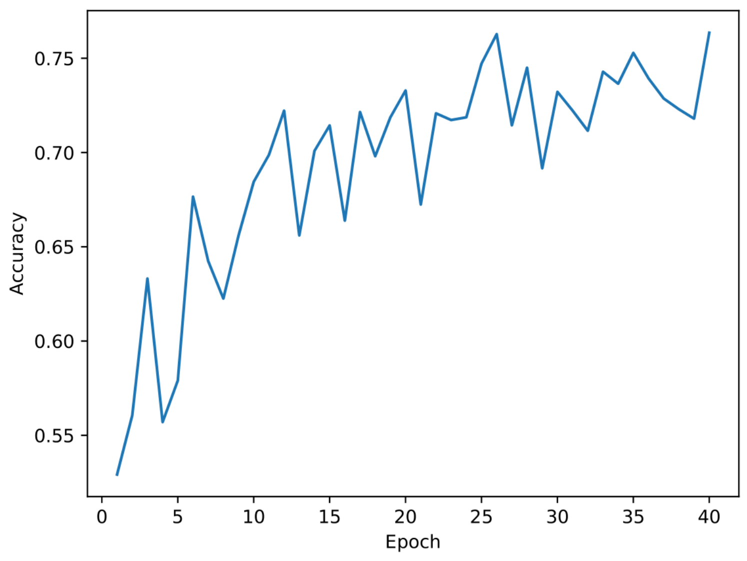


**2.** Using the optimal configuration from the previous exercise i.e. learning rate 0.01, and training the model for 40 epochs we obtained a final test accuracy of 0.7623.

**Training loss as function of epoch number for the best configuration**



**Validation accuracy as function of epoch number for the best configuration**



**3.**

We obtained 5340742 trainable parameters for exercise 1. While for exercise 2 we got 755718 trainable parameters, which is considerably lower (around 85% less).

By replacing the transformation of flattening the output of the convolutional blocks with a global average pooling layer, there is a reduction on the number of parameters, given that this comes as another pooling layer.

It is important to highlight that, in addition to the reduction of computational costs, it helps in retaining the most essential features, allowing the model to generalize better - confirmed by the better validation and test accuracies. Batch normalization stabilizes the gradients during training, thus accelerating convergence and also contributing to this phenomenon; furthermore, it reduces sensitivity to initial weights and learning rate, shown by the performance plots, starting with accuracies and losses only attained after some epochs then when it is not used.

**4.**

Using small kernels reduces the number of parameters compared to kernel of (width x height), which brings numerous advantages. Given less parameters, we are able to generalize and prevent overfitting. Moreover, training and inference require fewer computation steps, reducing the time and the power consumption to train the model.

Smaller kernels are more effective at capturing local features because excessively large kernels can lead to over smoothing and the loss of fine-grained details in the input data. By stacking multiple small kernels, we can achieve an equivalent receptive field to a larger kernel while maintaining a more granular focus on local patterns, since each small kernel is followed by an activation function that introduces non-linear transformations between layers. This allows the network to learn more complex and abstract features than otherwise.

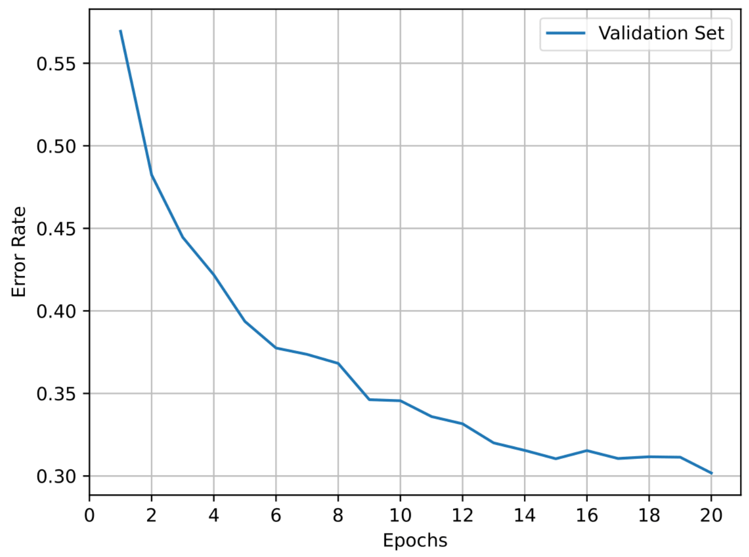
Pooling layers simplify feature maps by reducing their size, which lowers computational costs by decreasing the number of calculations and memory required for subsequent layers. They also highlight important features by emphasizing the most significant activations in a region, and help prevent overfitting by reducing the number of learnable parameters, making the network more focused on general patterns rather than noise or fine-grained details.

## Question 3

**1.**

**a)** After training we obtained the following plot:

**Validation CER at the end of each epoch**

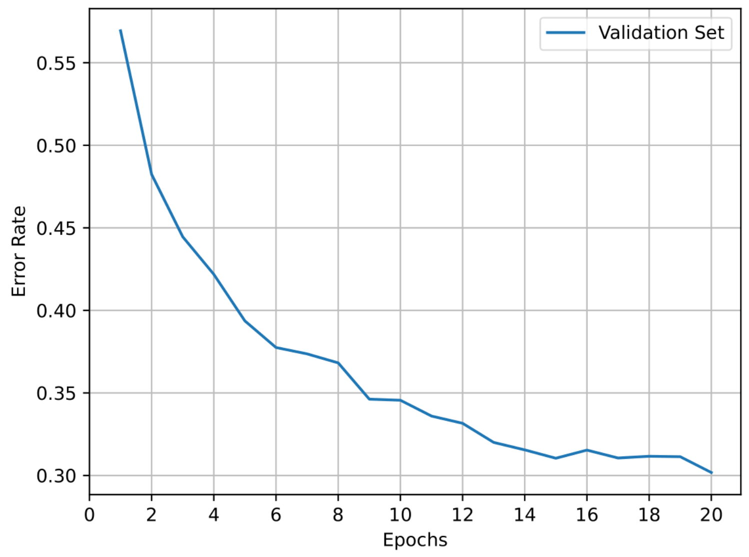


Then, running the checkpoint (minimum CER) on the test set:

- **CER –** 0.3006
- **WER –** 0.7980

**b)** After training, now with attention mechanism, we obtained the following plot:

**Validation CER at the end of each epoch**



Then, running the checkpoint (minimum CER) on the test set:

- **CER –** 0.2041
- **WER –** 0.7190

**c)** Using the same checkpoint as b) on the test set but now with nucleus sampling instead of greedy decoding:

- **CER – ** 0.2269
- **WER – ** 0.7630
- **WER@3 –** 0.6500

## Contribution of each member & bibliography

Questions 2 & 3 were made by both group elements using VSCode Live Share extension (https://marketplace.visualstudio.com/items?itemName=ms-vsliveshare.vsliveshare).

Question 1 was split: Diogo made 1.2 and 1.3 and Vasco made 1.1.

To help solve the exercises, we looked for information in theorical and practical classes, and from the following pages:

- https://www.deeplearningbook.org/ (linear algebra section)
- https://www.geeksforgeeks.org/how-to-choose-kernel-size-in-cnn/
- https://data-ai.theodo.com/blog-technique/2019-10-31-convolutional-layer-convolution-kernel
- https://viso.ai/deep-learning/batch-normalization/
- https://pytorch.org/docs/stable/nn.html

ChatGPT was used in Question 3 to understand cases of mismatching dimensions i.e. why we need to change the input sequence passed to decoder to tgt[:, :-1] instead of tgt.