****

**Deep Learning (IST, 2024-25)**

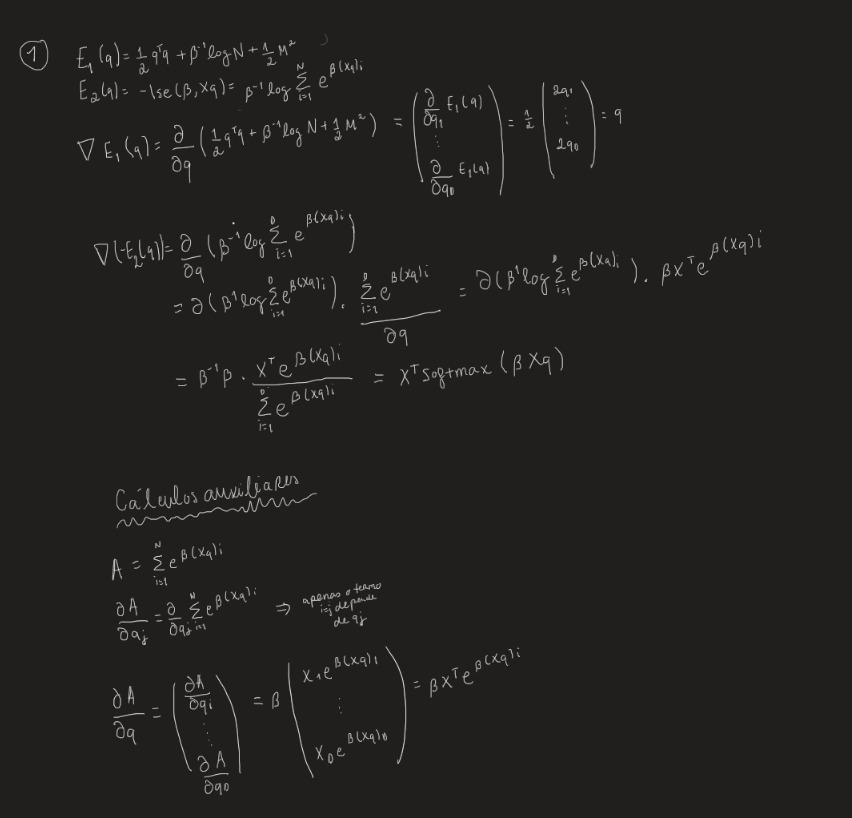
**Homework 2**

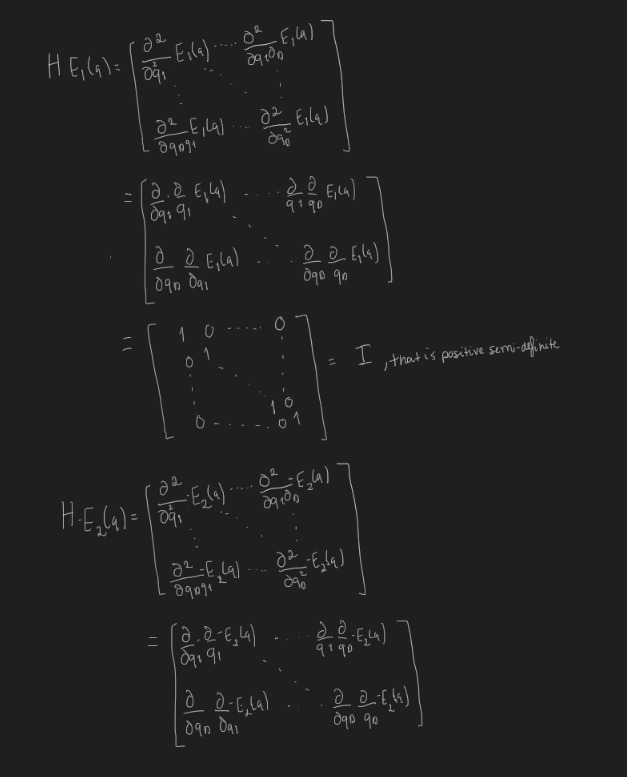
Diogo Ribeiro Vasco Paisana

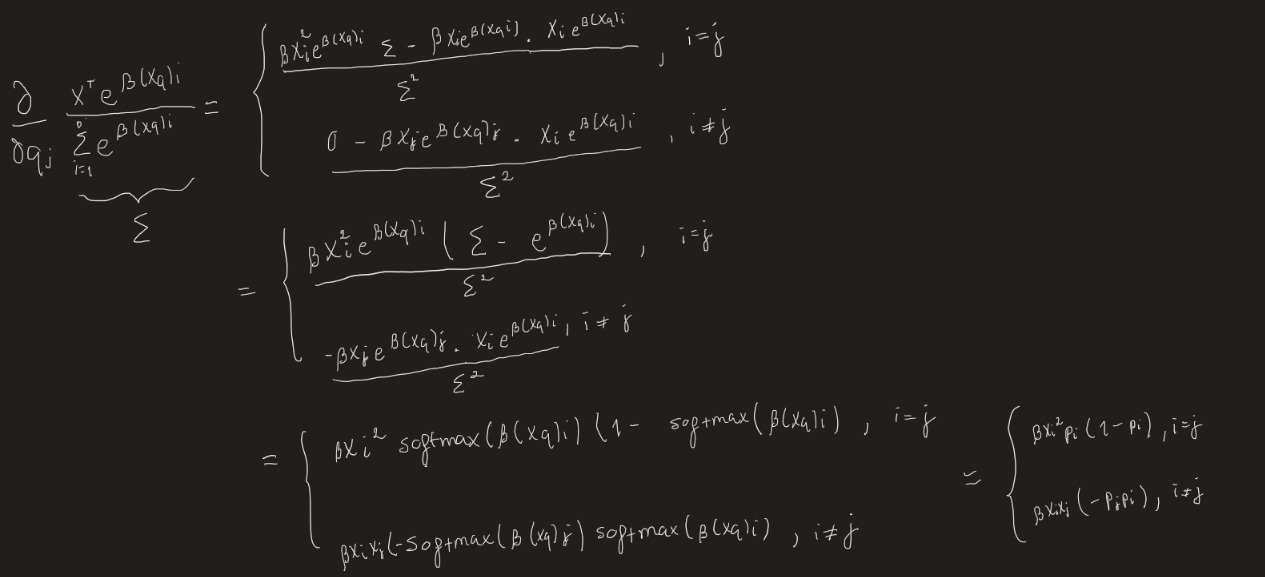
ist1102484 ist1102533

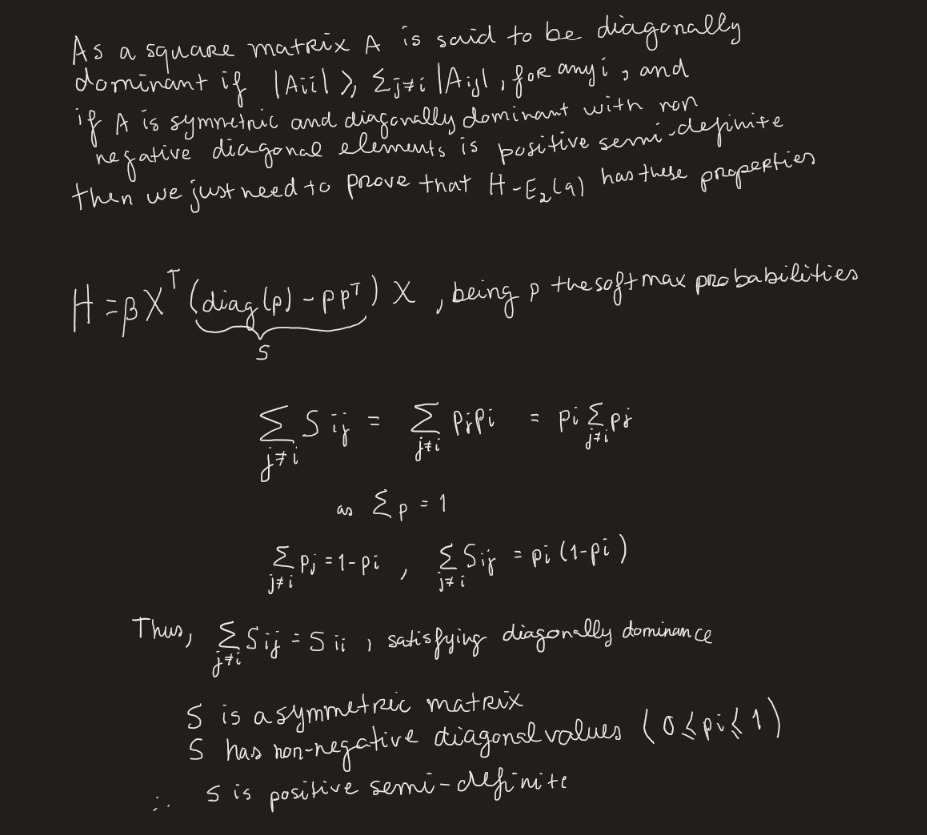
**Question 1**

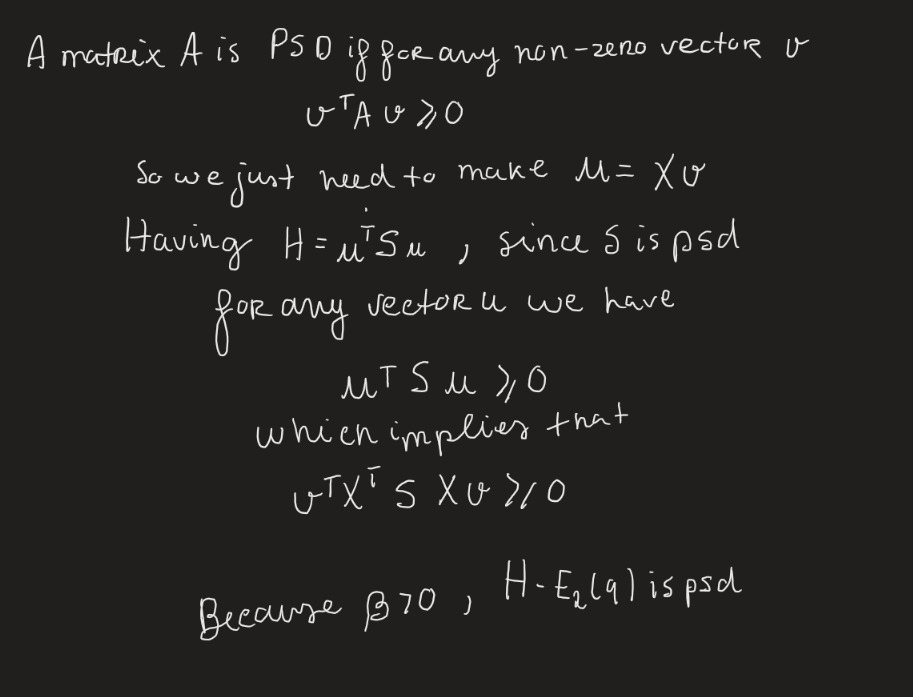
**1.**

****

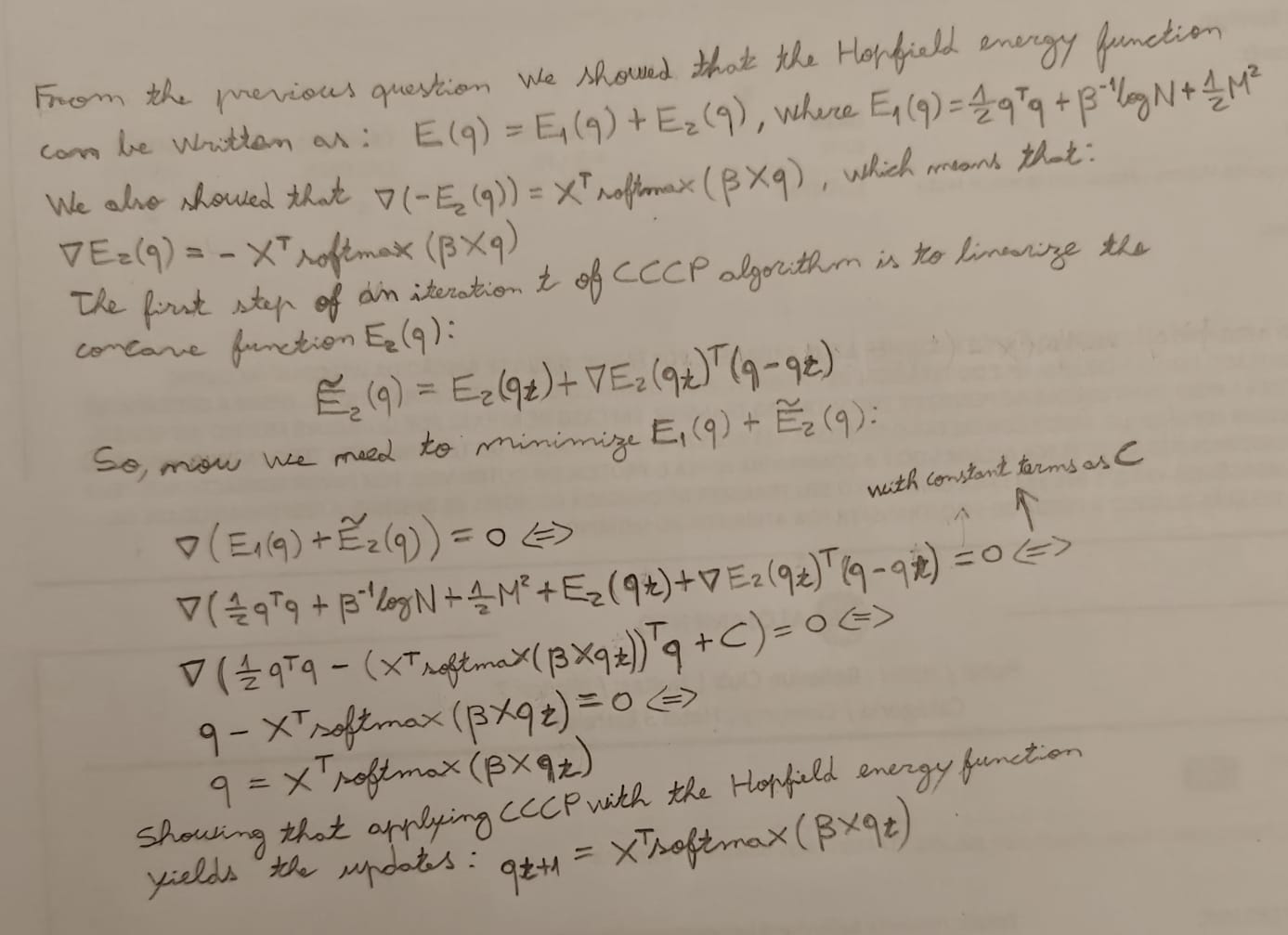
****

****

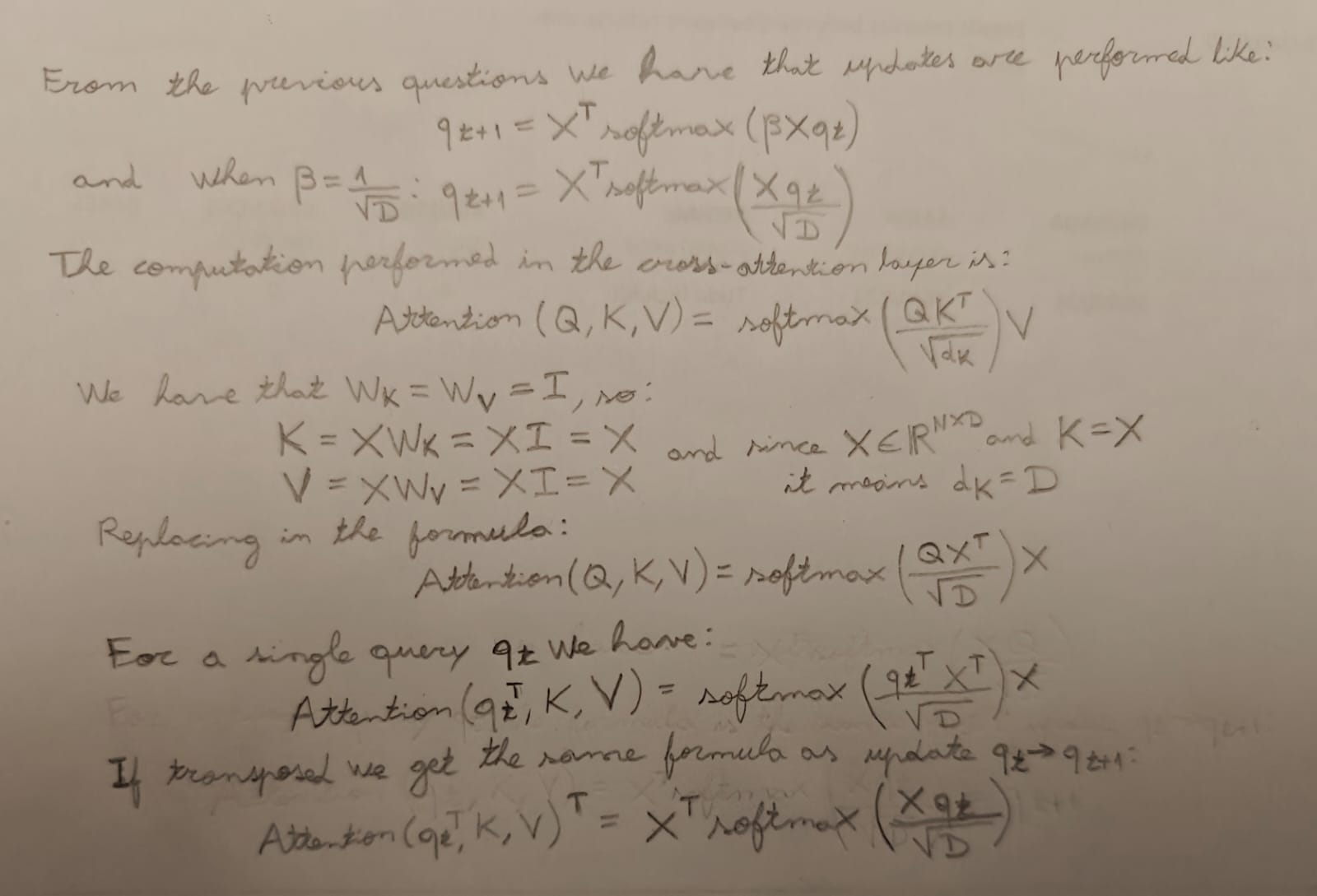
****

****

**2.**

****

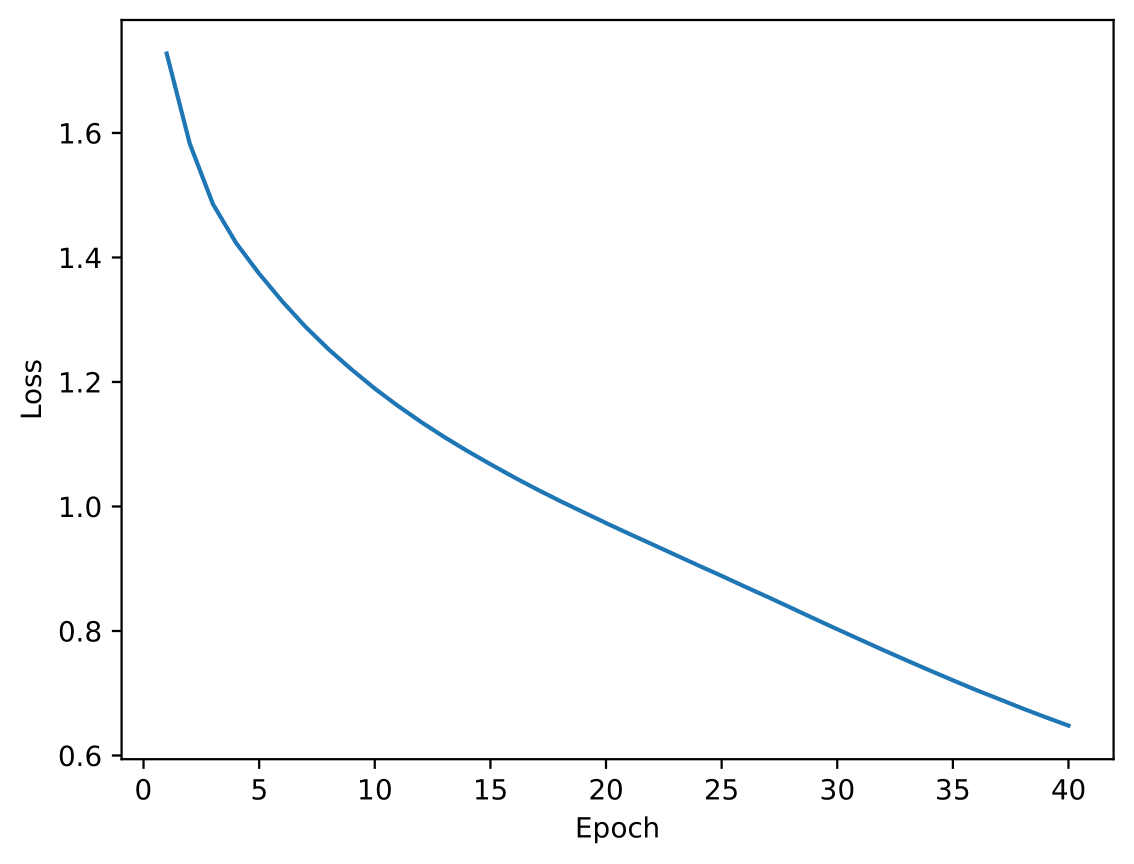
**3.**

****

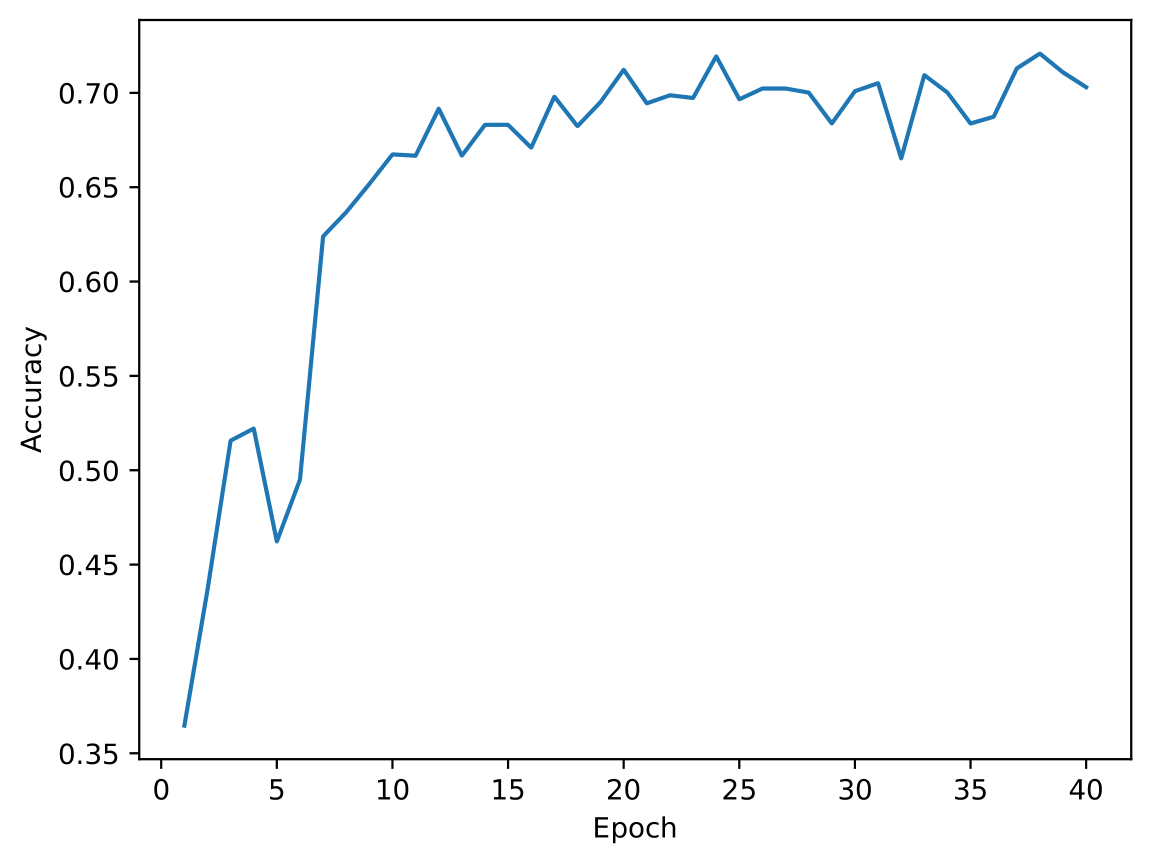
**Question 2**

**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**

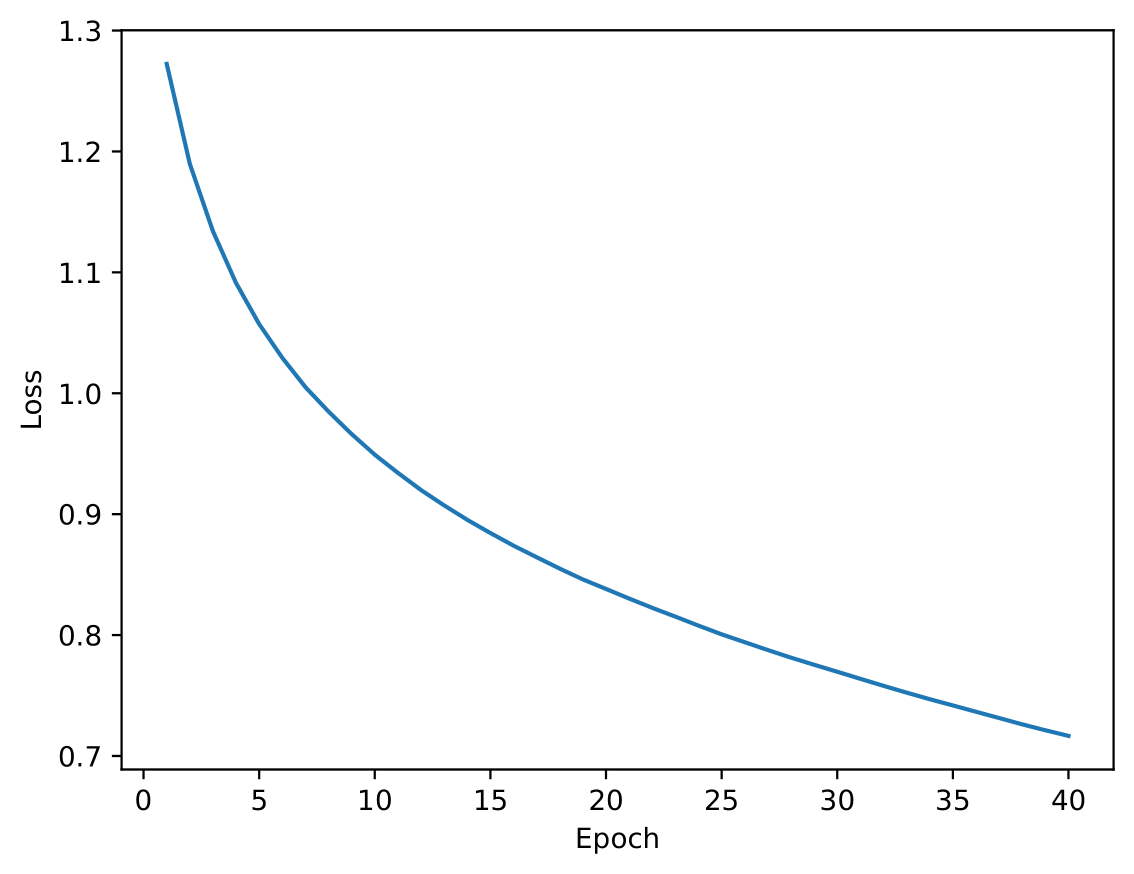
****

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

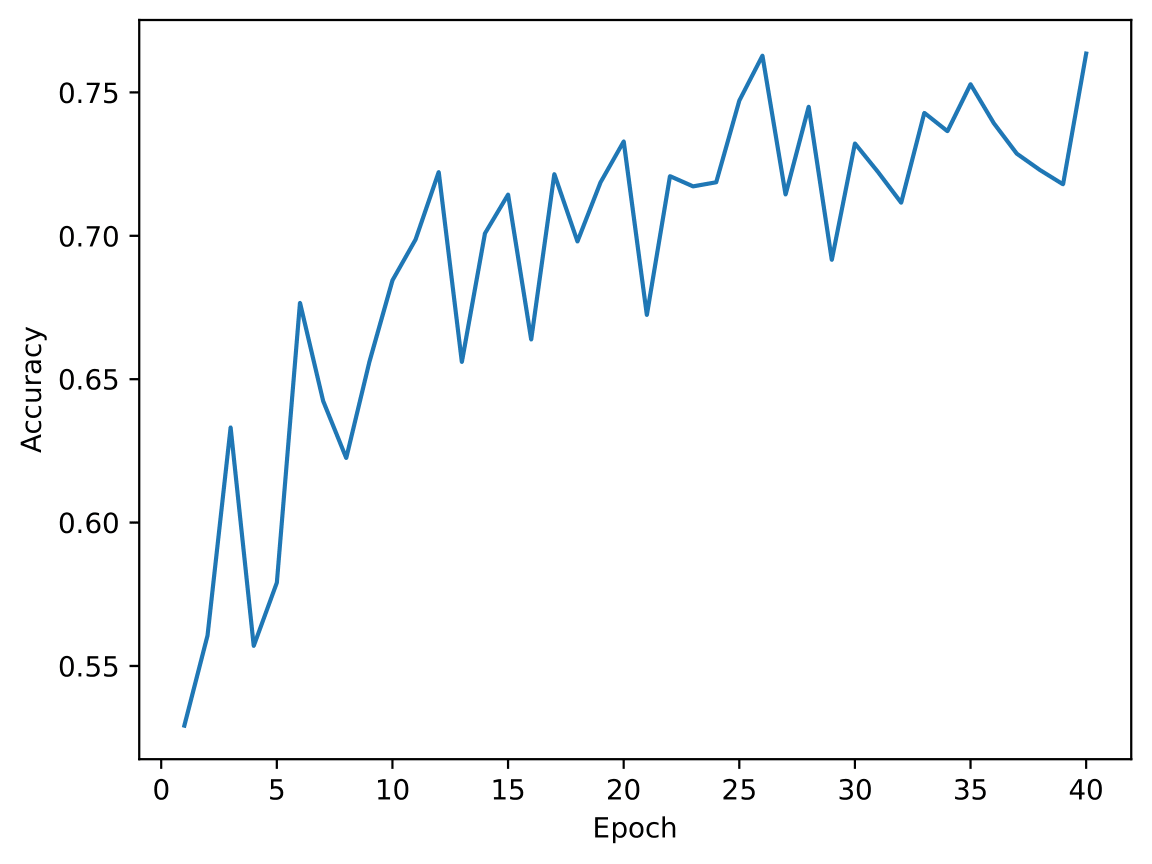
****

**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.

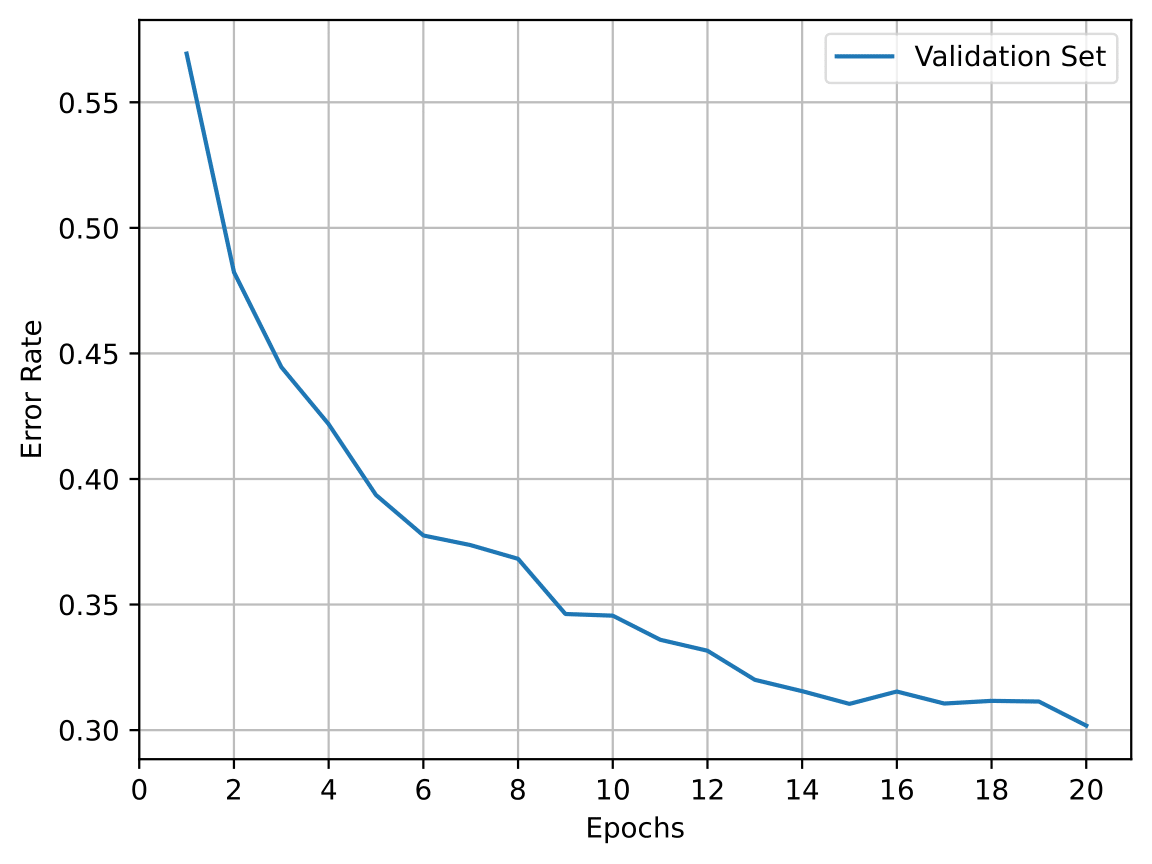
Pooling layers simplify feature maps by reducing their size, which lowers computational costs by decreasing the number of calculations and memory required for subsequent layers. They also highlight important features by emphasizing the most significant activations in a region, and help prevent overfitting 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**

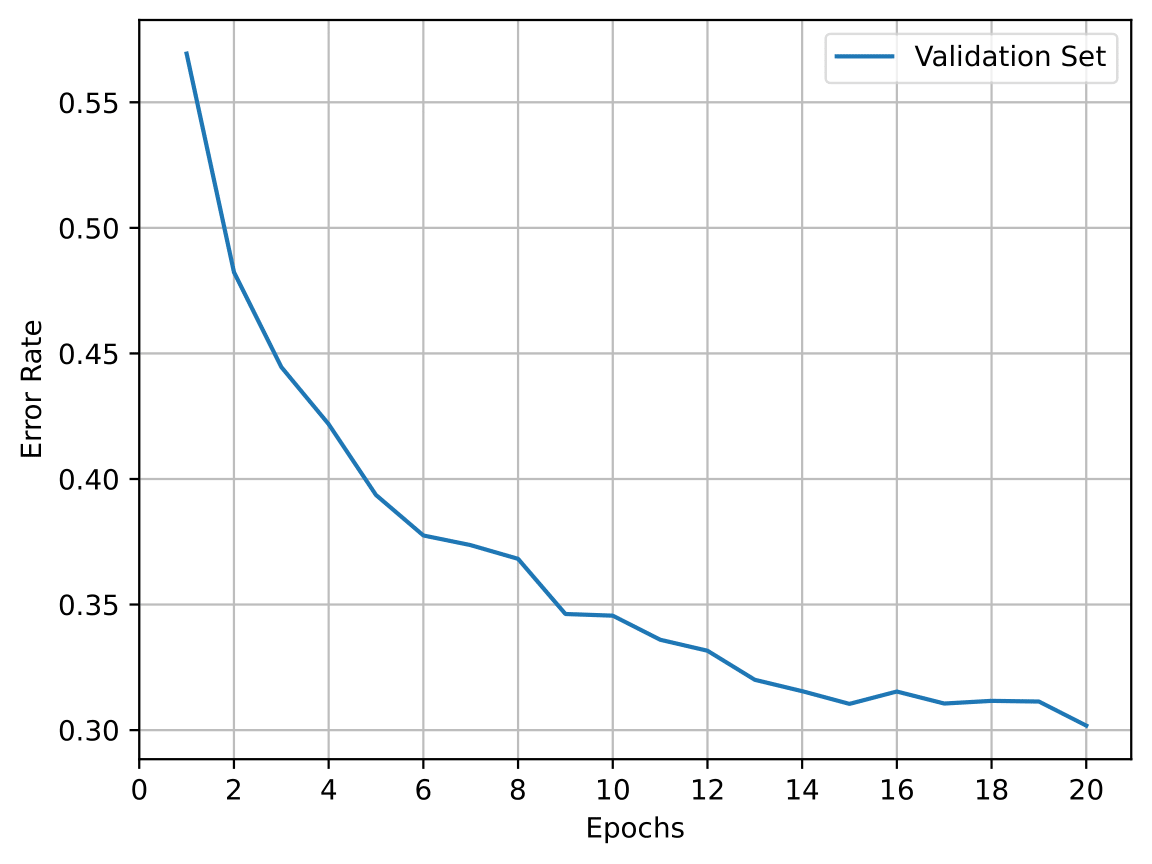
****

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.