TÉCNICO LISBOA

Deep Learning

MSc Electrical and Computer Engineering

Homework 2 - Report

Authors:

António Jotta (99893) antonio.jotta@tecnico.ulisboa.pt
António Vasco Morais de Carvalho (102643) antonio.v.morais.c@tecnico.ulisboa.pt

Group 17

Group contribution:
António Jotta: Question 1.1, 1.3; Question 2.
António Morais: Question 1.2; Question 3.
Both: Report (respective questions with review on both sides).

2024/2025 – 1st Semester, P2

Contents

Question 1

1

The task is to show that the Hopfield energy function

$$E(\mathbf{q}) = -\mathrm{lse}(\beta, \mathbf{X}\mathbf{q}) + \beta^{-1}\log N + \frac{1}{2}\mathbf{q}^\top\mathbf{q} + \frac{1}{2}M^2, \tag{1}$$

can be written as

$$\begin{aligned} E(\mathbf{q}) &= \underbrace{-\mathrm{lse}(\beta, \mathbf{X}\mathbf{q})}_{E_2(\mathbf{q})} + \underbrace{\beta^{-1}\log N + \frac{1}{2}\mathbf{q}^\top\mathbf{q} + \frac{1}{2}M^2}_{E_1(\mathbf{q})} = \\ &= E_1(\mathbf{q}) + E_2(\mathbf{q}), \end{aligned} \tag{2}$$

where E_1 is a convex function and E_2 is a concave function, and

$$\mathrm{lse}(\beta, \mathbf{z}) = \beta^{-1}\log\sum_{i=1}^{N} e^{\beta z_i}, \quad M = \max_i \|\mathbf{x}_i\|.$$

Then show that the gradients of E_1 and $-E_2$ are, respectively,

$$\nabla E_1(\mathbf{q}) = \mathbf{q}, \quad \nabla(-E_2(\mathbf{q})) = \mathbf{X}^\top\mathrm{softmax}(\beta\mathbf{X}\mathbf{q}),$$

compute the Hessians of E_1 and $-E_2$, and show that they are positive semi-definite (psd).

Firstly, computing the gradient and Hessian of E_1 gives (let $C \in \mathbb{R}$ and $\mathbf{I}$ be the identity matrix):

$$\begin{aligned} E_1(\mathbf{q}) &= \underbrace{\beta^{-1}\log N}_{C} + \frac{1}{2}\mathbf{q}^\top\mathbf{q} + \underbrace{\frac{1}{2}M^2}_{C} = \frac{1}{2}\mathbf{q}^\top\mathbf{q} + C \Leftrightarrow \\ &\Leftrightarrow \nabla E_1(\mathbf{q}) = \mathbf{q} \Leftrightarrow \\ &\Leftrightarrow \nabla^2 E_1(\mathbf{q}) = \mathbf{I}. \end{aligned}$$

Now for $-E_2$, computing its gradient gives:

$$\begin{aligned} -E_2(\mathbf{q}) &= \mathrm{lse}(\beta, \mathbf{X}\mathbf{q}) = \beta^{-1}\log\underbrace{\left(\sum_{i=1}^{N} e^{\beta(\mathbf{X}\mathbf{q})_i}\right)}_{S} \Leftrightarrow \\ \Leftrightarrow \nabla(-E_2(\mathbf{q})) &= \frac{1}{\beta S}\nabla_{\mathbf{q}} S = \\ &= \frac{1}{\beta S}\sum_{i=1}^{N} \beta e^{\beta(\mathbf{X}\mathbf{q})_i}\mathbf{X}_i^\top = \\ &= \frac{1}{S}\sum_{i=1}^{N} e^{\beta(\mathbf{X}\mathbf{q})_i}\mathbf{X}_i^\top. \end{aligned}$$

Recognizing that

$$\frac{e^{\beta(\mathbf{Xq})i}}{S} = \frac{e^{\beta(\mathbf{Xq})i}}{\sum_{j=1}^{N} e^{\beta(\mathbf{Xq})_j}} = \mathrm{softmax}_i(\beta\mathbf{Xq}),$$

the gradient of $-E_2$ is given by

$$\begin{aligned}\nabla(-E_2(\mathbf{q})) &= \sum_{i=1}^{N} \mathrm{softmax}_i(\beta\mathbf{Xq})\mathbf{X}_i^\top = \\ &= \mathbf{X}^\top \mathrm{softmax}(\beta\mathbf{Xq}).\end{aligned}$$

Consider $\mathbf{p} = \mathrm{softmax}(\beta\, X\mathbf{q})$, the Hessian of $-E_2(\mathbf{q})$ is

$$\nabla^2(-E_2(\mathbf{q})) = \beta\, X^\top \big[\mathrm{diag}(\mathbf{p}) - \mathbf{p}\,\mathbf{p}^\top\big] X,$$

where the matrix $\mathrm{diag}(\mathbf{p}) - \mathbf{p}\,\mathbf{p}^\top$ is psd, and multiplying by $\beta > 0$, $\mathbf{X}^\top$, and $\mathbf{X}$ preserves psd. Therefore $\nabla^2(-E_2(\mathbf{q})) \succeq 0$, implying $-E_2(\mathbf{q})$ is convex, and thus $E_2(\mathbf{q})$ is concave, which concludes the given proof task.

2

The Convex-Concave Procedure (CCCP) is an iterative algorithm designed to find local minima of functions that can be expressed as the sum of a convex and a concave component, under certain mild conditions. The algorithm proceeds works as follows: at iteration t,

- Step 1: Linearize the concave function E_2 using a first-order Taylor approximation around $\mathbf{q}_t$,

$$E_2(\mathbf{q}) \approx \tilde{E}_2(\mathbf{q}) := E_2(\mathbf{q}_t) + (\nabla E_2(\mathbf{q}_t))^\top(\mathbf{q} - \mathbf{q}_t);$$

- Step 2: Compute a new iterate by solving the convex optimization problem

$$\mathbf{q}_{t+1} = \arg\min_{\mathbf{q}} E_1(\mathbf{q}) + \tilde{E}_2(\mathbf{q}).$$

The task is to show that applying the CCCP algorithm with the Hopfield energy function (1) yields the updates

$$\mathbf{q}_{t+1} = \mathbf{X}^\top \mathrm{softmax}(\beta\mathbf{X}\mathbf{q}_t). \tag{3}$$

The CCCP decomposes the energy function $E(\mathbf{q})$ into the sum of a convex function $E_1(\mathbf{q})$ and a concave function $E_2(\mathbf{q})$. As shown previously, the energy function can be decomposed as (2), where $E_1(\mathbf{q})$ is convex and $E_2(\mathbf{q})$ is concave. To address the concave component, $E_2(\mathbf{q})$ is linearized using a first-order Taylor expansion around $\mathbf{q}_t$ (as mentioned in Step 1:):

$$E_2(\mathbf{q}) \approx \tilde{E}_2(\mathbf{q}) = E_2(\mathbf{q}_t) + (\nabla E_2(\mathbf{q}_t))^\top(\mathbf{q} - \mathbf{q}_t),$$

where $\nabla E_2(\mathbf{q}_t) = -\mathbf{X}^\top \mathrm{softmax}(\beta\mathbf{Xq})$.

Now onto Step 2, which involves solving the convex optimization problem obtained by replacing $E_2(\mathbf{q})$ with $\tilde{E}_2(\mathbf{q})$, yielding the update (3). The linearized energy function (2) is rewritten as (let $C \in \mathbb{R}$):

$$
\begin{aligned}
E(\mathbf{q}) &\approx E_1(\mathbf{q}) + \tilde{E}_2(\mathbf{q}) = \\
&= \frac{1}{2}\mathbf{q}^\top\mathbf{q} + \underbrace{\beta^{-1}\log N + \frac{1}{2}M^2 + E_2(\mathbf{q}_t)}_{C} + (\nabla E_2(\mathbf{q}_t))^\top\mathbf{q} - \underbrace{E_2(\mathbf{q}_t))^\top\mathbf{q}_t}_{C} = \\
&= \frac{1}{2}\mathbf{q}^\top\mathbf{q} + (\nabla E_2(\mathbf{q}_t))^\top\mathbf{q} + C = \\
&= \frac{1}{2}\mathbf{q}^\top\mathbf{q} + \left(-\mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q})\right)^\top\mathbf{q} + C = \\
&= \frac{1}{2}\mathbf{q}^\top\mathbf{q} - \left(\mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q})\right)^\top\mathbf{q} + C.
\end{aligned}
$$

Minimizing the above expression with respect to $\mathbf{q}$ yields:

$$\mathbf{q}_{t+1} = \arg\min_{\mathbf{q}}\left[\frac{1}{2}\mathbf{q}^\top\mathbf{q} - \left(\mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q}_t)\right)^\top\mathbf{q}\right].$$

Taking the derivative with respect to $\mathbf{q}$ and setting $\nabla_{\mathbf{q}}E(\mathbf{q}) = 0$ gives:

$$
\begin{aligned}
&\nabla_{\mathbf{q}}E(\mathbf{q}) = \mathbf{q} - \mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q}_t) \Leftrightarrow \\
&\overset{\nabla_{\mathbf{q}}E(\mathbf{q})=0}{\Leftrightarrow} \mathbf{q} - \mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q}_t) = 0 \Leftrightarrow \\
&\Leftrightarrow \mathbf{q} = \mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q}_t).
\end{aligned}
$$

This demonstrates that applying the CCCP algorithm to the Hopfield energy function produces the iterative update (3).

3

Considering $\beta = \frac{1}{\sqrt{D}}$, the task is to compare the first update $\mathbf{q}_t \mapsto \mathbf{q}_{t+1}$ with the computation performed in the cross-attention layer of a Transformer with a single attention head. The identity projection matrices are defined as $\mathbf{W}_K = \mathbf{W}_V = \mathbf{I}$, and the input is $\mathbf{X} \in \mathbb{R}^{N\times D}$. The query matrix $\mathbf{Q} = [\mathbf{q}_t^{(1)}, \ldots, \mathbf{q}_t^{(N)}]^\top \in \mathbb{R}^{N\times D}$ is assumed to be precomputed.

The CCCP update (3), where $\mathbf{q}_t \in \mathbb{R}^D$ represents the current query vector, can be expanded as follows:

$$\text{softmax}(\beta\mathbf{X}\mathbf{q}_t)_i = \frac{e^{\beta(\mathbf{X}\mathbf{q}_t)_i}}{\sum_{j=1}^{N} e^{\beta(\mathbf{X}\mathbf{q}_t)_j}}.$$

The expanded form demonstrates that the update computes a weighted sum of the rows of $\mathbf{X}$, where the weights are determined by the softmax distribution over the similarity scores $(\mathbf{X}\mathbf{q}_t)_i = \mathbf{x}_i^\top\mathbf{q}_t$. Substituting these weights into the update rule gives:

$$\mathbf{q}_{t+1} = \mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q}_t) = \sum_{i=1}^{N}\text{softmax}(\beta\mathbf{X}\mathbf{q}_t)_i\mathbf{x}_i. \tag{4}$$

The cross-attention mechanism of a transformer with a single attention head, using the identity projection matrices $\mathbf{W}_K = \mathbf{W}_V = \mathbf{I}$, and the input $\mathbf{X} \in \mathbb{R}^{N\times D}$ yield $K = V = X$. The attention scores in this mechanism are computed using the scaled dot-product attention:

$$\text{Attention}(\mathbf{q}_t, \mathbf{X}) = \frac{1}{\sqrt{D}}\mathbf{X}\mathbf{q}_t.$$

Applying the softmax function row-wise to the attention scores produces the attention probabilities:

$$P_i = \text{softmax}\left(\frac{1}{\sqrt{D}}\mathbf{X}\mathbf{q}_t\right)_i = \frac{e^{\frac{1}{\sqrt{D}}(\mathbf{X}\mathbf{q}_t)_i}}{\sum_{j=1}^{N} e^{\frac{1}{\sqrt{D}}(\mathbf{X}\mathbf{q}_t)_j}}.$$

Using these attention probabilities, the output of the attention mechanism is expressed as a weighted sum of the rows of $\mathbf{X}$:

$$Z(\mathbf{q}_t, \mathbf{X}) = \sum_{i=1}^{N} \text{softmax}\left(\frac{1}{\sqrt{D}}\mathbf{X}\mathbf{q}_t\right)_i \mathbf{x}_i. \tag{5}$$

Lastly, a comparison of (4) and (5) shows that the computation performed in the cross-attention layer of a transformer with a single attention head is identical to the update performed by the CCCP algorithm when $\beta = \frac{1}{\sqrt{D}}$, thus completing the proof.

Question 2

1

A convolutional network was implemented using stochastic gradient descent for training and Pytorch for automatic differentiation. The model consists of three convolutional blocks with output channel sizes of 32, 64 and 128. Each block includes a convolutional layer (3×3 kernel, stride 1, padding 1), ReLU activation, max pooling (2×2) and dropout (0.1). The output of the convolutional blocks is flattened and passed through a multilayer perceptron (MLP) with layers of size 1024 and 512, followed by an output layer with a Log-Softmax activation. The model was trained for 40 `epochs` with a `batch_size` of 8, and the `learning_rate` (η) was tuned over the following values: $\{0.1, 0.01, 0.001\}$. Training loss and validation accuracy were monitored to determine the best configuration.

The best `learning_rate` found was **0.01**. The results are presented below:

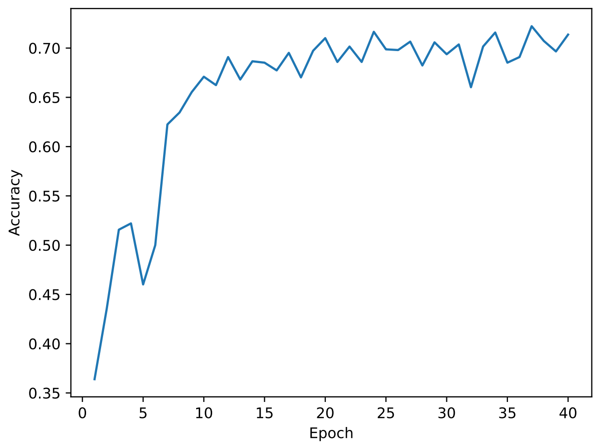


Figure 1: Validation accuracy for $\eta = 0.01$ as a function of the number of epochs.

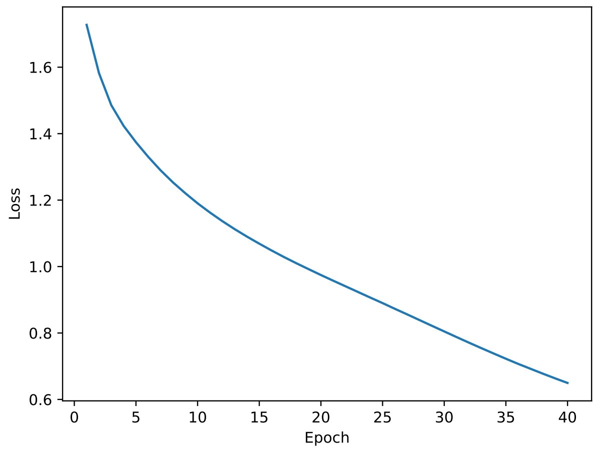


Figure 2: Training loss for $\eta = 0.01$ as a function of the number of epochs.

- **Validation accuracy:** 0.7137.
- **Test accuracy:** 0.6937.

2

The convolutional network implemented was trained with modifications to the architecture to enhance performance. Batch normalization was incorporated into each convolutional block using `nn.BatchNorm2d`, replacing the direct normalization of the convolutional outputs. Additionally, the flattening transformation was replaced with a global average pooling layer, implemented using `nn.AdaptiveAvgPool2d` with a kernel size of $(1, 1)$. In the MLP block, batch normalization was also added before the dropout layers, utilizing `nn.BatchNorm1d`. The performance of the network was evaluated using the optimal configuration defined in the previous question.

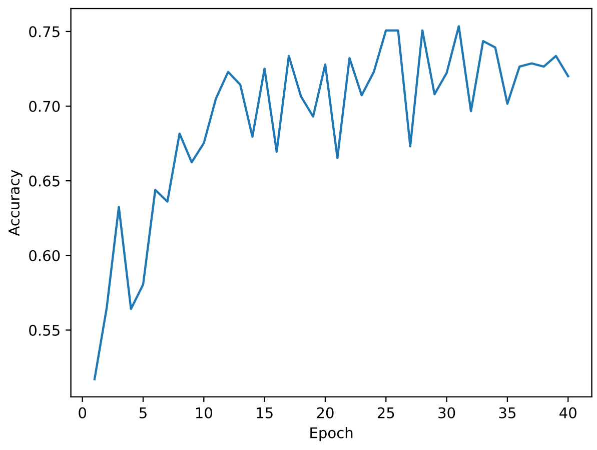


Figure 3: Validation accuracy for $\eta = 0.01$ as a function of the number of epochs.

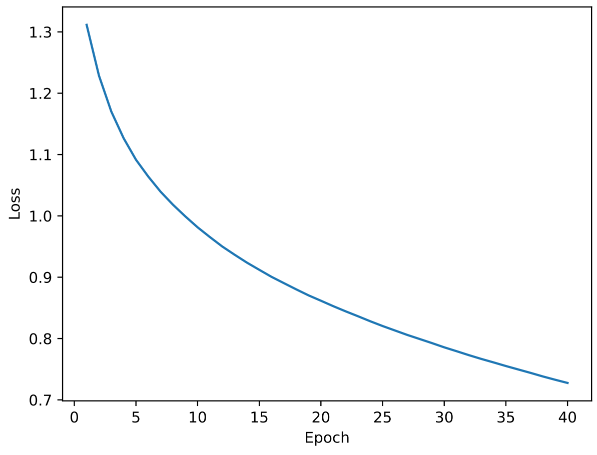


Figure 4: Training loss for $\eta = 0.01$ as a function of the number of epochs.

- **Validation accuracy:** 0.7201.
- **Test accuracy:** 0.7403.

3

In this question, the function `get_number_trainable_params` was implemented to determine the number of trainable parameters of CNN's from the two previous questions.

Model Version	**Number of Trainable Parameters**
Q2.1	5340742
Q2.2	756742

Table 1: Number of trainable parameters for Q2.1 and Q2.2.

The difference in the number of parameters is due to applying Global Average Pooling (GAP) to the output of the third convolutional block instead of flattening it. The output of the third convolutional block has dimensions $[8, 128, 6, 6]$. Flattening this output results in a tensor of dimensions $[8, 4608]$ (where $4608 = 128 \times 6 \times 6$). In contrast, applying GAP reduces the output to dimensions $[8, 128]$. As a result, without GAP, the first layer of the MLP would require 4608 input features, while with GAP, it only requires 128.

The performance improvement is due to the addition of batch normalization in both the convolutional blocks and the MLP. Batch normalization improves training stability by addressing exploding or vanishing gradients, reduces dependency on weight initialization, and serves as a regularizer by introducing noise through mini-batch statistics, which helps prevent overfitting. Moreover, GAP enhances accuracy by reducing the number of parameters. This process focuses on the most significant features in each channel, improving generalization and lowering the risk of overfitting.

4

Small convolution kernels, such as 3×3, are preferred over a single large kernel that spans the entire input dimension due to their ability to significantly reduce the number of parameters, making the model more efficient. Stacking small kernels in deeper networks enables the receptive field to expand, covering larger areas without increasing the parameter count excessively. This approach supports the hierarchical extraction of local features and increasingly abstract representations.

Pooling layers, such as max pooling with a 2×2 kernel, perform spatial downsampling by halving the height and width of the feature maps. This operation reduces the number of parameters and computational requirements in subsequent layers, while also introducing a degree of translation invariance. By focusing on the most relevant features in each region, pooling helps control overfitting and accelerates training.

Question 3

1

In this section, character-level recurrent sequence-to-sequence model was implemented, that can solve the Phoneme-to-Grapheme Conversion task for a small dataset of English pronunciation extracted from WikiPron . This task uses two error metrics:

- Character Error Rate (CER) is the levenshtein distance between the predicted se- quences and the true spellings, divided by the length of the true spellings.
- Word Error Rate (WER) is the percentage of examples for which the predicted sequence does not exactly match the true spelling.

a)

A vanilla character-level encoder-decoder model was implemented, consisting of a bidirectional LSTM encoder and an autoregressive LSTM decoder. The implementation specifically involved completing the `forward()` method for both the `Encoder` and `Decoder` classes in the provided `models.py` file. After implementing the required methods, the model was trained for 20 `epochs` using the following hyperparameters: a learning rate of 0.003, a `dropout` rate of 0.3, a `hidden_size` of 128, and a `batch_size` of 64. The dropout layer was applied exclusively to the embeddings and the final outputs in both the encoder and the decoder.

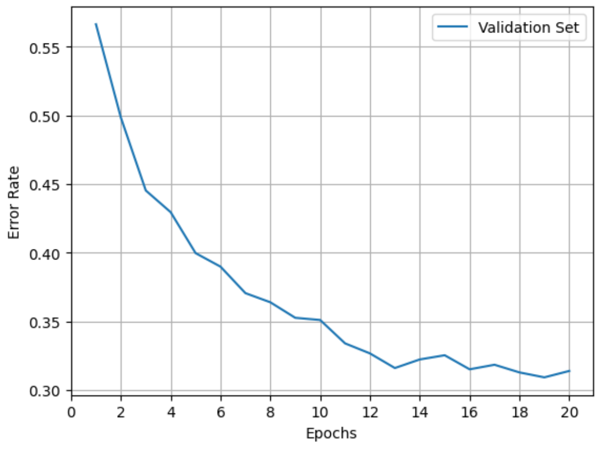


Figure 5: CER of the encoder-decoder model (without attention) as a function of the number of epochs.

- **Test CER:** 0.3046
- **Test WER:** 0.8020

b)

In sequence-to-sequence models, the Bahdanau Attention mechanism enables the decoder to selectively focus on specific parts of the input sequence during decoding. This is achieved by computing a weighted sum of the encoder outputs, where the weights represent the relevance of each encoder hidden state to the current decoding step.

The tasks involved implementing the `forward()` method of the `BahdanauAttention` class in the `models.py` file and modifying the `Decoder` implementation to incorporate the attention mechanism. Once the implementation was completed, the model was trained and tested using the same hyperparameters as in the previous exercise.

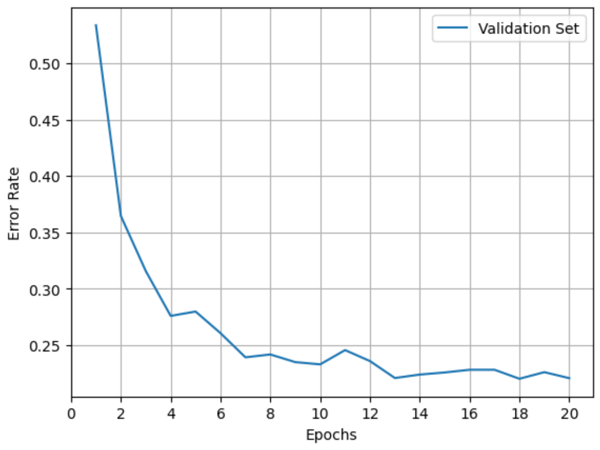


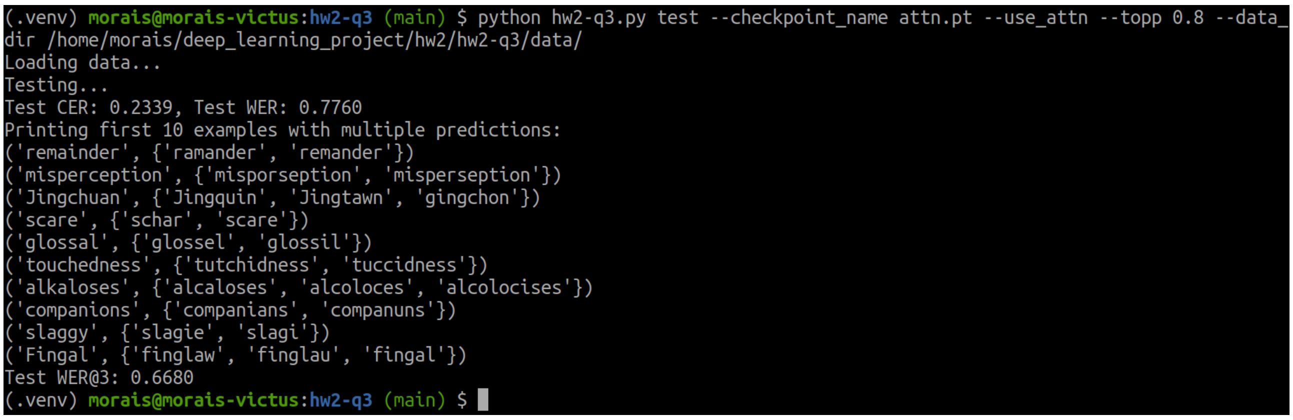
Figure 6: CER of the encoder-decoder model (with attention) as a function of the number of epochs.

- **Test CER:** 0.2168
- **Test WER:** 0.7460

c)

Up to this point, greedy decoding has been used to generate hypotheses from the models. In this approach, at each time step, the next token is selected as the one with the highest probability, conditioned on the source and previously generated tokens. While this procedure is not guaranteed to find the globally most likely sequence, it is simple to implement and highly effective in practice. However, in this exercise, nucleus sampling was implemented as an alternative decoding technique.

More specifically, the `nucleus_sampling` function in `hw2-q3.py` was implemented. After completing this implementation, the model with attention from the last exercise was reused to generate predictions on the test set.



```
(.venv) morais@morais-victus:hw2-q3 (main) $ python hw2-q3.py test --checkpoint_name attn.pt --use_attn --topp 0.8 --data_
dir /home/morais/deep_learning_project/hw2/hw2-q3/data/
Loading data...
Testing...
Test CER: 0.2339, Test WER: 0.7760
Printing first 10 examples with multiple predictions:
('remainder', {'ramander', 'remander'})
('misperception', {'misporseption', 'misperseption'})
('Jingchuan', {'Jingquin', 'Jingtawn', 'gingchon'})
('scare', {'schar', 'scare'})
('glossal', {'glossel', 'glossil'})
('touchedness', {'tutchidness', 'tuccidness'})
('alkaloses', {'alcaloses', 'alcoloces', 'alcolocises'})
('companions', {'companians', 'companuns'})
('slaggy', {'slagie', 'slagi'})
('Fingal', {'finglaw', 'finglau', 'fingal'})
Test WER@3: 0.6680
(.venv) morais@morais-victus:hw2-q3 (main) $ 
```

Figure 7: Terminal output after reusing the model with attention, and `nucleus_sampling` to generate predictions from the test set.

- **Test CER:** 0.2339

- **Test WER:** 0.7760
- **Test WER@3:** 0.6680