Deep Learning - DEI

Group 30: Alice Mota ist1102500 | Francisco Leitão ist103898

Contribution:

Question 1 was solved by both members, with Francisco having done exercises 1.1 and 1.3, and Alice doing 1.2. Question 2 was done by Francisco, while Question 3 was done by Alice.

Homework 2:

Question 1

The answers to question 1 of this homework are at the end of the document. The start of each exercise is marked with the question number at the top of the page.

Question 2

1. The following table shows the final validation and test accuracies for each learning rate.

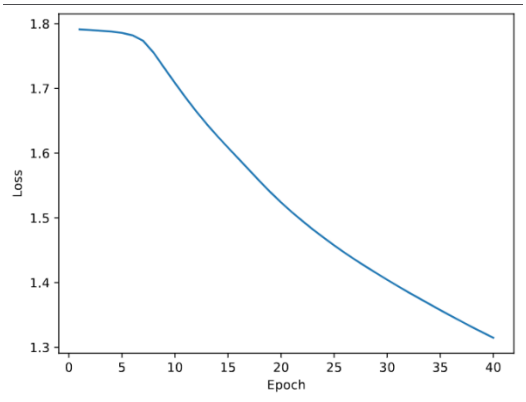
Learning Rate	Validation Accuracy	Test Accuracy
0.001	60.87%	60.57%
0.01	70.30%	69.00%
0.1	53.77%	55.43%

Table 1 - Final validation and test accuracies for each learning rate

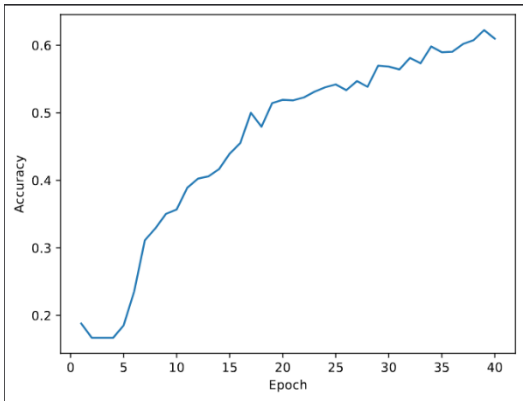
The best configuration was the one with a learning rate of 0.01, yielding a validation accuracy of 70.30% and a test accuracy of 69.00%.

The plots below show the validation accuracy and training loss as functions of the epoch number for each learning rate:

a) $\eta = 0.001$

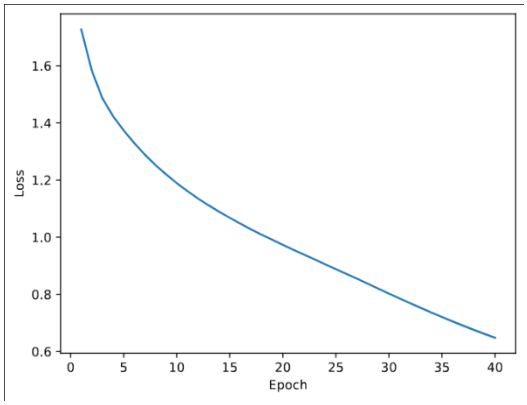


Plot 1 (η = 0.001)– Training loss in function of the epoch number

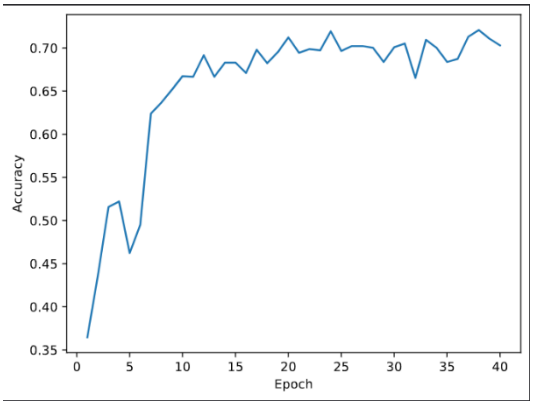


Plot 2 (η = 0.001)– Validation accuracy in function of the epoch number

b) $\eta = 0.01$

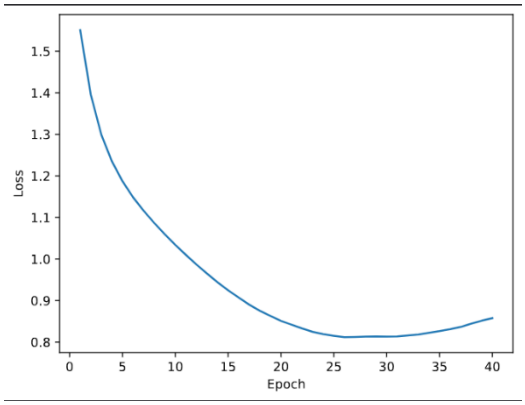


Plot 3 (η = 0.01)– Training loss in function of the epoch number

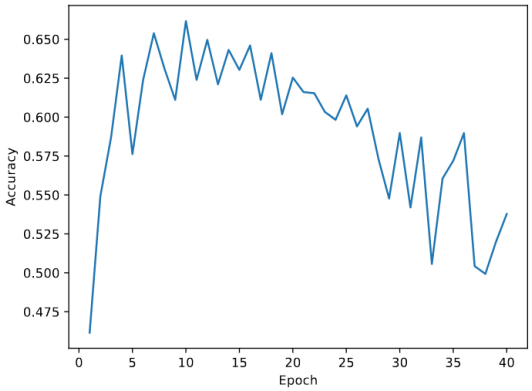


Plot 4 (η = 0.01)– Validation accuracy in function of the epoch number

c) $\eta = 0.1$

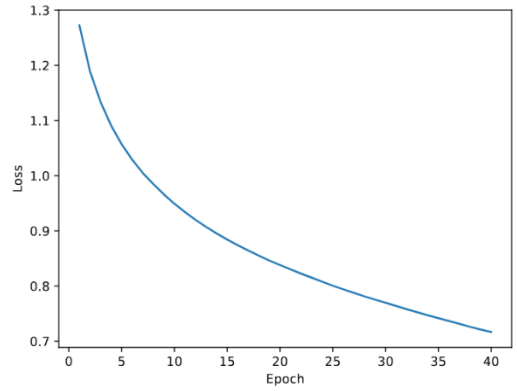


Plot 5 (η = 0.1)– Training loss in function of the epoch number

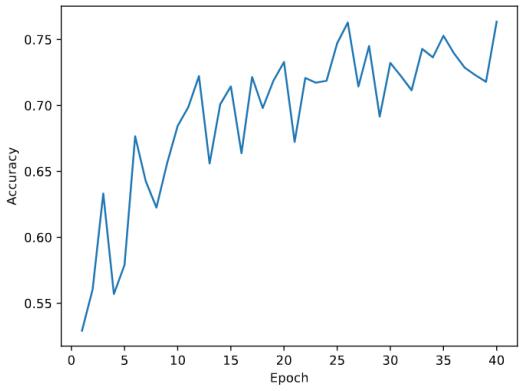


Plot 6 (η = 0.1)– Validation accuracy in function of the epoch number

2. Since the best configuration from the previous question was the one with a learning rate of 0.01, it was chosen for this question. The resulting network, with batch normalization, achieved a validation accuracy of 76.35% and a test accuracy of 76.23%, representing a considerable improvement in performance compared to the previous model.
   The plots below show the validation accuracy and training loss as functions of the epoch number of the new network:



Plot 7 (η = 0.01)– Training loss in function of the epoch number



Plot 8 (η = 0.01)– Training loss in function of the epoch number

3. The differences in the number of trainable parameters and performance arise from how Batch Normalization (BatchNorm) affects the architecture and training dynamics. With BatchNorm, we have a total of 755,718 trainable parameters, while without it we have almost 10 times more – 5,340,742 trainable parameters.

   The biggest reason for this parameter difference is the fact that with batch norm we have a global average pooling layer that reduces the feature map size to 1 x 1, significantly reducing the input size to the fully connected layers. We can verify this in line 85 of *hw2-q2.py* where the input size is equal to 128 with BatchNorm and 128 * 6 * 6 = 4,608 without Batchnorm. It is worth noting that BatchNorm replaces the bias terms with two new trainable parameters (γ and β) for each feature map, which is a minimal overhead compared to the last reason.

   Regarding the performance difference, it is clear from the results from the previous question that the network with BatchNorm performs better. This difference in performance can be explained by several factors:

   a) **Internal Covariate Shift Reduction:** with BatchNorm, the intermediate feature maps are normalized, reducing the internal covariate shift;

   b) Regularization Effect: with BatchNorm, a regularization effect is introduced in the network by compressing the feature map to a smaller size. This smaller network size can improve generalization and help reduce overfitting;

   c) Enabling Deeper Networks: with BatchNorm, by normalizing the activations it prevents the gradients from becoming too small or too large (vanishing gradient or gradient explosion)

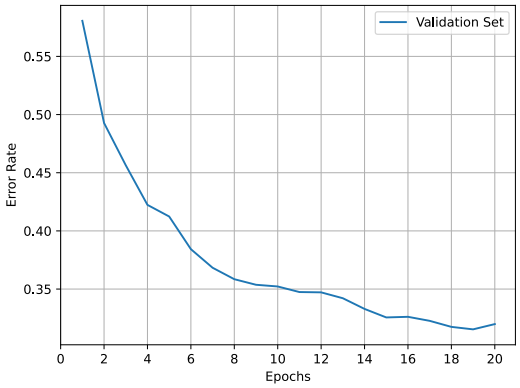
4. Small kernels (such as the 3 × 3 used in this exercise) reduce the number of parameters and computational cost compared to large kernels (e.g. $w \times h$), making them more efficient. Additionally, using multiple layers of small kernels can achieve the same effective receptive field as a single large kernel but with fewer parameters and more non-linearities (due to the activation functions between layers). Finally, smaller kernels capture local patterns more effectively, enabling better feature hierarchies and generalization in deeper networks.

   Regarding the effect of pooling layers, they reduce the spatial dimensions (width and height) of feature maps, thereby lowering computational complexity in subsequent layers and reducing the risk of overfitting. Furthermore, pooling layers emphasize the most significant features by selecting the maximum value (in the case of max pooling) within a region, making the network invariant to small translations or distortions in the input. Finally, pooling increases the receptive field of neurons in subsequent layers, allowing them to capture larger-scale patterns in the data.

Question 3

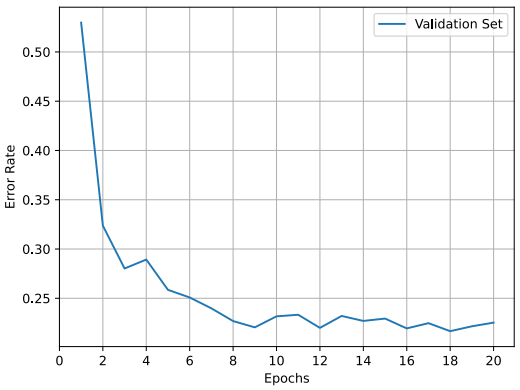
1.

a) In the training set (Plot 1) the performance, with error rate as the chosen metric was of 0.3154. In the test set, the vanilla character-level encoder-decoder displayed a Character Error Rate (CER) of 0.3087 and a Word Error Rate (WER) of 0.8100. The small gap between validation and test CER indicates strong generalization. However, the high test WER highlights the model's difficulty in achieving exact word-level matches despite accurate character-level predictions, suggesting the need for more sophisticated mechanisms.



Plot 1 – Vanilla character-level encoder-decoder error rate in function of the epoch number

b) In the training set (Plot 2) the performance, with error rate as the chosen metric was of 0.2166. In the test set, the vanilla character-level encoder-decoder displayed a Character Error Rate (CER) of 0.2132 and a Word Error Rate (WER) of 0.7190. While CER is close to the best validation rate, WER can likely be improved through further optimization or fine-tuning.



Plot 2 – Vanilla character-level encoder-decoder error rate in function of the epoch number

c) The results obtained after running the implemented function and evaluating the model on the test set are as follows:|

   a. Character Error Rate (CER): 0.2265
   b. Word Error Rate (WER): 0.7400
   c. WER@3: 0.6510, which means that approximately 65% of examples lack a correct prediction among the 3 samples, highlighting potential areas for model improvement.

Nucleous sampling enhanced text generation by balancing quality and diversity. It introduced variability in predictions as well, as illustrated by the obtained examples:

Ground Truth	Predictions
remainder	remander, romander, remandere
Jingchuan	Jingquine, Zinghthwan, Jingtuan
scare	scar, skare
glossal	glossel, glossal
touchedness	tutchidness, tutchedness, tutchidenis
alkaloses	alcolousis, alcolocise, alcolosise
companions	companniens, compannians, Kempanians
Fingal	fingaall, fingaal
grievances	greevences, greavences, greevencis
Bronx	Branks, Branx, brongs

Although predictions vary, some examples demonstrate challenges in producing accurate outputs, particularly for less common words (e.g., "Jingchuan").
The results indicate a trade-off between generating varied predictions and maintaining accuracy. Further refinement of the sampling strategy could include adjusting the pp or combining beam search.

References:

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., et al. (2020). Hopfield networks is all you need. arXiv preprint, arXiv:2008.02217. Retrieved from https://arxiv.org/abs/2008.02217

Yuille, A. L., & Rangarajan, A. (2003). The concave-convex procedure. Neural Computation, 15(4), 915–936. https://doi.org/10.1162/08997660360581958

Hopfield Layers. Retrieved from https://ml-jku.github.io/hopfield-layers/

Question 1

1.

We want to show that the energy function $E(q)$ can be written as:

$$E(q) = E_1(q) + E_2(q)$$

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

First, we can define E_1 and E_2:

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

$$E_2(q) = -\text{lse}(\beta, Xq)$$

In this exercise we will:

1. Compute the gradients $\nabla E_1(q)$ and $\nabla(-E_2(q))$
2. Compute the hessians of $E_1(q)$ and $-E_2(q)$ and show that they are both positive semi-definite

(9)

Gradient of $E_1(q)$:

$$\nabla\left(\frac{1}{2}q^Tq + \beta^{-1}\log N + \frac{1}{2}M^2\right)$$

$= \nabla\left(\frac{1}{2}q^Tq\right)$, since the terms $(\beta^{-1}\log N)$ and $\left(\frac{1}{2}M^2\right)$ are constants with respects to q, their gradients are 0

$= q$

So $\underline{\nabla E_1(q) = q}$

Gradient of $-E_2(q)$:

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

Using the chain rule:

$\nabla\text{lse}(\beta, Xq) = X^T\nabla\text{lse}(\beta, Xq) = X^T\text{Softmax}(\beta Xq)$, this way

$\underline{\nabla(-E_2(q)) = \nabla\text{lse}(\beta, Xq) = X^T\text{Softmax}(\beta Xq)}$

(2)

Hessian of $E_1(q)$

$$H_{E_1} = \nabla^2 E_1(q) = \nabla(q) = I_D, \quad I_D \text{ is a } D \times D \text{ identity matrix}$$

To prove I_D is positive semi-definite, we can use the suggested diagonal dominance fact:

1. I_D is symmetric since $I_D = I_D^T$
2. Diagonal elements are all 1 ($I_{ii} = 1, \forall i$)
3. The off diagonal elements are all 0 ($I_{ij} = 0, i \neq j$)

This way, I_D satisfies the diagonal dominance condition:

$$|I_{ii}| \geq \sum_{j \neq i} |I_{ij}| \quad (1 \geq 0)$$

Since I_D is a symmetric diagonally dominant matrix with non-negative diagonal elements we can conclude that $H_{E_1} = I_D$ is positive semi-definite.

Hessian of $-E_2(q)$

$$H_{-E_2} = \nabla^2(-E_2(q)) = \nabla(X^T \text{softmax}(\beta X q)) \text{, using the chain rule}$$

$$= \beta X^T (\text{diag}(p) - pp^T) X, \quad p = \text{softmax}(\beta X q)$$

Using the proof from Lemma A22 from the "Hopfields Networks is All You Need" paper:

Lemma A22:

for an arbitrary z:

$$z^T(\text{diag}(p) - pp^T)z = \sum_i p_i z_i^2 - \left(\sum_i p_i z_i\right)^2$$

$$= \left(\sum_i p_i z_i^2\right)\left(\sum_i p_i\right) - \left(\sum_i p_i z_i\right)^2 \geq 0$$

"The last inequality holds true because the Cauchy-Schwartz inequality says $(a^Ta)(b^Tb) \geq (a^Tb)^2$, which is the last inequality with $a_i = z_i\sqrt{p_i}$ and $b_i = \sqrt{p_i}$. Consequently $(\text{diag}(p) - pp^T)$ is positive semi-definite."

Since $H_{-E_2} = \beta X^T(\text{diag}(p) - pp^T)X$, the positive semi-definiteness from $(\text{diag}(p) - pp^T)$ is preserved, making H_{-E_2} positive semi-definite.

Since both H_{E_1} and H_{-E_2} are positive semi-definite, both E_1 and $-E_2$ are convex, therefore $E_2 = -(-E_2)$ is concave, reaching the wanted conclusion that:

$E = E_1(q) + E_2(q)$, where E_1 is convex and E_2 is concave

2.

Energy function:

$$E(q) = -lse(\beta, Xq) + \beta^{-1}\log N + \frac{1}{2}q^Tq + \frac{1}{2}M^2$$

where:

→ $M = \max_i \|x_i\|$

→ $lse(\beta, z)$ is the log-sum-exp function with "temperature" β^{-1}

$$lse(\beta, z) = \beta^{-1}\log \sum_{i=1}^{N} \exp(\beta z_i)$$

As seen in 1., $E(q)$ can be written as:

$$E(q) = E_1(q) + E_2(q)$$

where

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

→ concave function: $E_2(q) = -lse(\beta, Xq)$

And $\nabla E_1(q) = q$; $\nabla(-E_2(q)) = X^T softmax(\beta Xq)$

Thus:

$$\nabla E_2(q) = -X^T softmax(\beta Xq)$$

The linearized approximation of $E_2(q)$ around q_t is:

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

Now, substituting $\nabla E_2(q_t) = -X^T softmax(\beta Xq_t)$

$$\tilde{E}_2(q) = -lse(\beta, Xq_t) - [X^T softmax(\beta Xq_t)]^T(q - q_t)$$

The optimization problem becomes:

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

Substituting $E_1(q) = \frac{1}{2} q^T q + \beta^{-1} \log N + \frac{1}{2} M^2$ and $\tilde{E}_2(q)$

$$q^{t+1} = \arg\min \left[\frac{1}{2} q^T q - \left[X^T \text{softmax}(\beta X q^t)^T q \right] \right]$$

The CCCP applied to E is therefore:

$$\nabla E_1(q^{t+1}) = -\nabla E_2(q^t)$$

$$\nabla \left(\frac{1}{2} q^t q + C \right) q^{t+1} = \nabla \text{lse}(\beta, X^T q^t)$$

where $\nabla \text{lse}(\beta, X^T q) = X \text{softmax}(\beta X^T q)$

The update rule for a state pattern q therefore reads:

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

3.

From the previous question we have

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

Let's compare the first update with the computation performed in the cross-attention layer of a transformer with a single attention head, $W_K = W_V = I$ with input $X \in \mathbb{R}^{N \times D}$, assuming a pre-computed query matrix $Q = [q_t^{(1)}, \dots q_t^{(N)}]^T$. From page 5 of the paper "Hopfields Networks is All You Need" we have a section "Hopfield update rule is attention of the transformer" we have:

$$Z = \text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V \quad \text{where} \quad D = d_K$$

$K = X^T$ and $V = X^T W_V$, with $K, V \in \mathbb{R}^{N \times D}$ are the key and value matrices, both equal to X in this case due to the identity projections, so we can simplify to:

$q_t^{(1)} \rightarrow$ first line of matrix Q

$$Z = \text{softmax}(\beta Q X) X^T, \text{ if we consider the first update } q_t \rightarrow q_{t+1}, \text{ this update is mathematically equivalent to the Hopfield update.}$$

Concluding, the first update in the Hopfield rule corresponds exactly to the computation performed in the cross-attention layer of a transformer under the given the assumptions.