TÉCNICO LISBOA

Deep Learning

Masters Degree in Mechanical Engineering

Homework 2 [EN]

Authors:

Francisco Pinto (ISTID 089888) francisco.v.pinto@tecnico.ulisboa.pt
Alexandre Gonçalves (ISTID 100121) alexandre.n.goncalves@tecnico.ulisboa.pt

Group 81

2024/2025 – 2nd Quarter

Introduction

This report contains our solutions to Homework 2 for the Deep Learning course offered at IST during the 2024-25 academic year. .

Contribution Statement:

- **Alexandre Gonçalves:** Conducted theoretical derivations and implemented the convex-concave procedure for modern Hopfield networks (Question 1). Designed the architecture for the convolutional neural network (CNN) and implemented its batch normalization enhancements (Question 2).
- **Francisco Pinto:** Implemented and analyzed the convolutional neural network (CNN) performance, including optimization and evaluation tasks (Question 2). Focused on the implementation and evaluation of sequence-to-sequence models for phoneme-to-grapheme conversion, integrated attention mechanisms, and explored nucleus sampling for varied outputs (Question 3).

We certify that the work presented here reflects the collective efforts of all team members.

Exercise 1

Problem statement:

Modern Hopfield networks are a type of network that exhibit associative memory capabilities Hopfield, 1982. Let

$$X \in \mathbb{R}^{N \times D}$$

be a matrix whose rows hold a set of N memory patterns (or feature vectors), and let

$$q \in \mathbb{R}^{D}$$

be a query vector (called a "state pattern"). The modern Hopfield network iteratively updates q_t according to

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

and under certain conditions, these dynamical trajectories converge to a fixed point. This update rule can be seen as the minimization of an "energy" function of the form:

$$E(q) = \underbrace{\tfrac{1}{2}\|q\|^2}_{E_1(q)} - \underbrace{\text{lse}(\beta, Xq)}_{E_2(q)},$$

where $\text{lse}(\beta, z) = \beta^{-1} \log\left(\sum_{i=1}^{N} \exp(\beta z_i)\right)$ is the log-sum-exp function with "temperature" β^{-1} and $z = Xq \in \mathbb{R}^N$. Note that as $\beta \to \infty$, $\text{lse}(\beta, z)$ approaches $\max_i z_i$ plus a constant.

1.1) Decomposition of the energy and analysis of gradients and Hessians

Considering $E(q) = -lse(\beta, Xq) + \frac{1}{2}q^T q + \beta^{-1} \log N + \frac{1}{2}M^2$ allows us to decompose our energy function in the following way:

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

$$E_1 = \tfrac{1}{2}q^T q + \beta^{-1} \log N + \tfrac{1}{2}M^2, \quad E_2 = -lse(\beta, Xq)$$

The gradients of E_1 and E_2 can be calculated in the following way:

- $E_1 = \underbrace{\frac{1}{2}q^T q}_{q-dependent} + \underbrace{\beta^{-1} \log N + \frac{1}{2}M^2}_{q-independent} \implies \nabla_q E_1 = \nabla_q(\frac{1}{2}q^T q) = \nabla_q \frac{1}{2} \sum_{i=1}^{D} q_i^2 = q$
- Using the chain rule:

$$\begin{aligned}
\nabla_q(-E_2) = -\nabla_{Xq} E_2 . \nabla_q(Xq) = \nabla_{Xq}(\beta^{-1} \log(\textstyle\sum_{i=1}^{D} exp(\beta[Xq]_i))) . \nabla_q(Xq) \\
= X^T \beta^{-1} \nabla_{Xq}(\log(\textstyle\sum_{i=1}^{D} exp(\beta[Xq]_i))) = X^T \beta^{-1} \frac{1}{\sum_{i=1}^{D} exp(\beta[Xq]_i)} \nabla_{Xq}(\textstyle\sum_{i=1}^{D} exp(\beta[Xq]_i)) \\
= X^T \beta^{-1} \frac{1}{\sum_{i=1}^{D} exp(\beta[Xq]_i)} [\beta exp(\beta[Xq]_1), \beta exp(\beta[Xq]_2), ..., \beta exp(\beta[Xq]_D)] \\
= X^T \frac{1}{\sum_{i=1}^{D} exp(\beta[Xq]_i)} [exp(\beta[Xq]_1), exp(\beta[Xq]_2), ..., exp(\beta[Xq]_D)] \\
= X^T softmax(\beta Xq)
\end{aligned}$$

The Hessians of E_1 and E_2 can be calculated in the following way:

- $H_{E_1} = \nabla_q(\nabla_q E_1) = \nabla_q\ q = I$
- $\frac{d}{dq_j} softmax(\beta Xq)_i = \beta(X^T softmax(\beta Xq)_i \delta_{ij} - (X^T softmax(\beta Xq))_i (X^T softmax(\beta Xq))_j)$
- $H_{-E_2} = -\nabla_q(\nabla_q E_2) = \nabla_q(X^T softmax(\beta Xq)) = X^T \nabla_{Xq} softmax(\beta Xq)$
  $= \beta X^T \Big(diag(X^T softmax(\beta Xq)) - X^T softmax(\beta Xq) \big(X^T softmax(\beta Xq)\big)^T \Big) X$

Finally, we can confirm if the obtained matrices are positive semi-definite:

- $v^T H_{E_1} v = v^T I v = v^T v = ||v||^2 > 0, \forall v \neq 0$
  therefore, H_{E_1} is s.p.d.
- $v^T H_{E_2} v = \beta v^T X^T \Big(diag(X^T softmax(\beta Xq)) - X^T softmax(\beta Xq) \big(X^T softmax(\beta Xq)\big)^T \Big) Xv$
  $= (Xv)^T \beta \Big(diag(X^T softmax(\beta Xq)) - X^T softmax(\beta Xq) \big(X^T softmax(\beta Xq)\big)^T \Big) (Xv)$

  Therefore, proving that H_{E_2} is s.p.d is equivalent to proving that :
  $v^T (diag(X^T softmax(\beta Xq)) - X^T softmax(\beta Xq) \big(X^T softmax(\beta Xq)\big)^T) v > 0, \forall v \neq 0$

  $v^T (diag(X^T softmax(\beta Xq)) - X^T softmax(\beta Xq) \big(X^T softmax(\beta Xq)\big)^T) v$
  $= \sum_i (X^T softmax(\beta Xq))_i z_i^2 - \big(\sum_i (X^T softmax(\beta Xq))_i z_i \big)^2$

  And, since, acording to the first fact provided on the footnotes:
  $(X^T softmax(\beta Xq))_i z_i^2 \geq \big(\sum_i (X^T softmax(\beta Xq))_i z_i \big)^2$

  Hence, we can conclude that H_{E_2} is s.p.d.

1.2) The Convex-Concave Procedure

- $\nabla_q \tilde{E}_2 = \nabla_q \big(E_2(q_t) + (\nabla_{q_t} E_2(q_t))^T (q - q_t) \big) = \nabla_{q_t} E_2(q_t) = -X^T softmax(\beta X q_t)$
- $q_{t+1} = arg\ min_q \big(E_1(q) + \tilde{E}_2(q) \big) \implies \nabla_{q_{t+1}} \big(E_1(q_{t+1}) + \tilde{E}_2(q_{t+1}) \big) = 0$
  $\iff \nabla_{q_{t+1}} E_1(q_{t+1}) + \nabla_{q_{t+1}} \tilde{E}_2(q_{t+1}) = 0 \iff q_{t+1} - X^T softmax(\beta X q_t) = 0$
  $\iff q_{t+1} = X^T softmax(\beta X q_t)$

1.3) Update rule comparison

Considering the computation performed in the cross-attention layer of a transformer with a single attention head, and matrices as described:

$$Z = softmax(\frac{QK^T}{\sqrt{D}})V = softmax(\frac{QXI}{\sqrt{D}})XI = softmax(\beta QX)X$$

Taking into account that this definition uses Z and Q as line vectors, instead of row vectors:

$$Z^T = softmax(\beta Q^T X)X = softmax(\beta Q^T X)X = softmax(\beta (XQ)^T)X = (X^T softmax(\beta XQ))^T$$

Therefore: $Z = X^T softmax(\beta XQ)$, which is the same update rule as the one obtained previously.

Question 2: Image Classification with CNNs

Q2.1: Baseline Convolutional Neural Network (CNN)

Model Architecture

The baseline CNN was designed to classify images using the Intel Image Classification dataset. The model architecture was implemented as required in the exercise:

- **Three Convolutional Blocks:**
  - **Output Channels:** 32, 64, and 128.
  - Each block consists of:
    - A 3×3 convolutional layer (`nn.Conv2d`) with stride 1 and padding 1.
    - ReLU activation (`nn.ReLU`).
    - Max pooling (`nn.MaxPool2d`) with 2×2 kernel and stride 2.
    - Dropout (`nn.Dropout`) with probability 0.1.
- **Multilayer Perceptron (MLP):**
  - Flatten the output from the convolutional layers.
  - Linear layer with 1024 units + ReLU + Dropout(0.1).
  - Linear layer with 512 units + ReLU.
  - Final linear layer projecting to output classes + Log-Softmax.

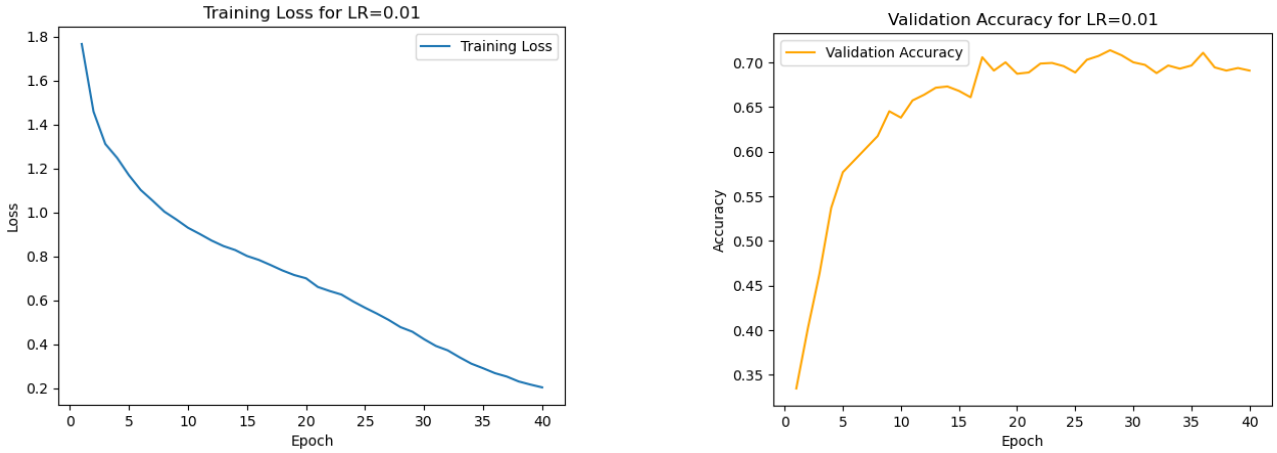
Training

The model was trained as requested and the results obtained are:

Best Learning Rate

Optimal Learning Rate: 0.01

Results



(a) Training Loss vs. Epochs **(b)** Validation Accuracy vs. Epochs

Figure 1: Training Loss and Validation Accuracy for Baseline CNN (LR: 0.01)

Q2.2: CNN with Batch Normalization

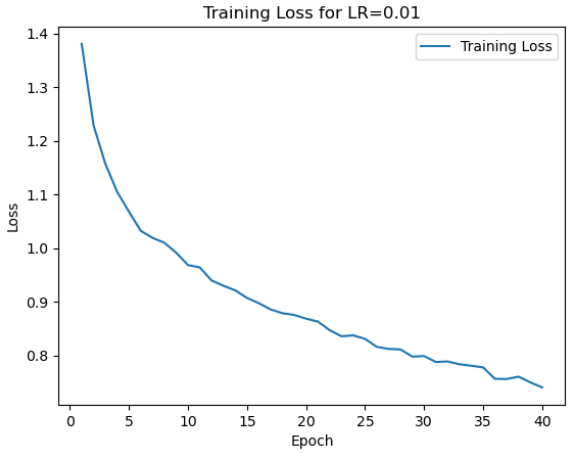
Model Modifications

The model was upgraded as required:

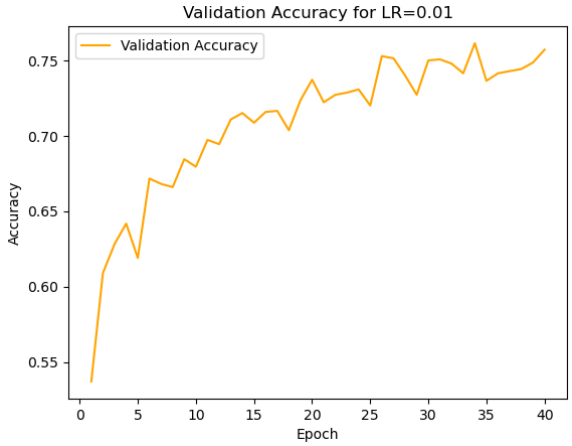
- Added `nn.BatchNorm2d` after each convolution and before activation.
- Applied `nn.AdaptiveAvgPool2d`.
- Incorporated `nn.BatchNorm1d` before dropout in the MLP.

Results

The results obtained with the best model from the previous exercise are shown on Figure 2.



(a) Training Loss with Batch Normalization



(b) Validation Accuracy with Batch Normalization

Figure 2: Training Loss and Validation Accuracy with Batch Normalization (LR: 0.01)

Performance Analysis:

Batch normalization resulted in faster and more stable training compared to the baseline model. By standardizing the inputs to each layer, the model had smoother and more efficient gradient updates, which reduced the impact of weight initialization and supported the use of higher learning rates. This stability allowed the model to train more quickly and better handle new, unseen data.

However, while batch normalization helped the training process run more smoothly, it did not lead to higher final validation accuracy compared to the baseline. The combination of batch normalization and dropout helped reduce overfitting, as shown by smoother training loss curves and steady validation accuracy. Overall, batch normalization improved the training dynamics but did not significantly improved the model's peak performance.

Exercise 2.3: Number of Trainable Parameters

The number of trainable parameters was calculated for both models:

- **Baseline CNN:** 5,340,742 trainable parameters.
- **CNN with Batch Normalization and Global Average Pooling:** 755,718 trainable parameters.

Discussion

- **Fewer Parameters:** Using Global Average Pooling (GAP) greatly reduces the number of trainable parameters by replacing large fully connected layers. Batch Normalization adds a small number of parameters for scaling (γ) and shifting (β), but this increase is very small.
- **More Stable Training:** Batch Normalization makes training faster and more stable by keeping the inputs to each layer well-behaved. GAP also makes the model simpler and helps prevent overfitting.

- **Less Overfitting:** Batch Normalization helps the model generalize better and reduces the need for dropout. Together with GAP, it leads to smoother learning and better performance on new data.
- **Faster Learning, Slightly Better Accuracy:** The model with Batch Normalization and GAP learns faster and is more stable during training. Additionally, the final validation accuracy is slightly better compared to the baseline model, improving from 0.7137 to 0.7614, making the learning process more efficient.

Exercise 2.4: Conceptual Questions

Small convolutional kernels, like 3×3, are better than large kernels for several reasons. They reduce the number of parameters, making the model faster and cheaper to train while lowering the risk of overfitting. Stacking small kernels allows the model to gradually learn complex patterns—early layers detect simple edges, while deeper layers recognize shapes and objects. More layers also mean more activation functions, which help the model learn more complex features. In contrast, a large kernel captures overall patterns in one step but can't learn detailed features and is more expensive to compute.

Pooling layers, like max pooling, help improve performance and efficiency by shrinking feature maps. This reduces the number of parameters and speeds up training while helping prevent overfitting by focusing on important features. Pooling also makes the model more resistant to small shifts or noise in the input. However, too much pooling can remove important details, which can hurt performance in tasks that need precise information. Overall, pooling helps build faster, stronger, and more generalizable models.

Exercise 3: Phoneme-to-Grapheme Conversion (P2G)

Exercise 3.1: Vanilla Encoder-Decoder Model

Model Implementation

The model was modified according to the exercise requirements for the Phoneme-to-Grapheme task with the following changes:

- **Bidirectional LSTM Encoder:** The `Encoder` class's `forward()` method in `models.py` was updated to use a bidirectional LSTM over the phoneme input sequence. Dropout was applied to both the input embeddings and the encoder's final outputs to reduce overfitting.
- **Autoregressive LSTM Decoder:** The `Decoder` class was adjusted to implement an autoregressive LSTM that generates grapheme sequences one token at a time. Dropout was also applied to the decoder's embeddings and outputs. An attention mechanism was not included in this version of the model.

Training Configuration

The model was trained using the hyperparameters specified in the statement.

Validation Performance

The Character Error Rate (CER) was evaluated at the end of each epoch to monitor performance (Figure 3).

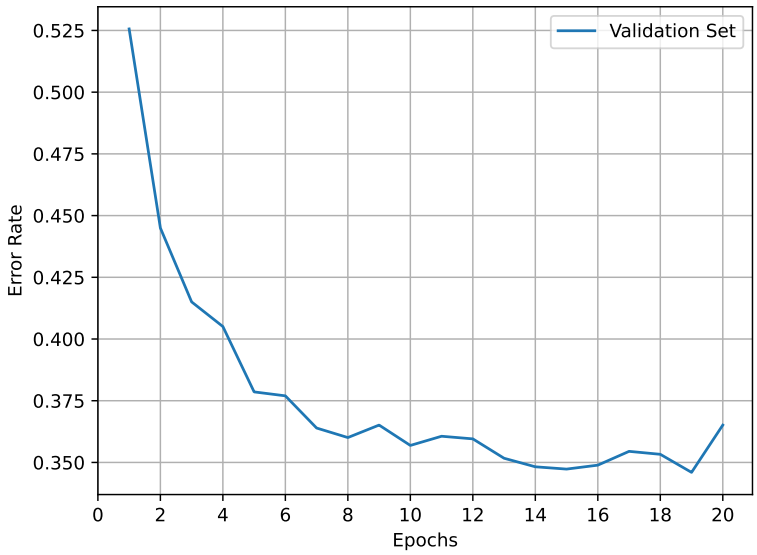


Figure 3: Validation CER across training epochs without attention.

Testing Results

The model was evaluated on the test set using the best-performing checkpoint yielding the following results:

Final Results

- **Character Error Rate (CER):** 0.3402
- **Word Error Rate (WER):** 0.8300

Performance Analysis

The vanilla encoder-decoder model achieved moderate performance, with a CER of **0.3402** and a WER of **0.8300**. While the model learned basic phoneme-to-grapheme mappings, its lack of attention limited its ability to focus on relevant parts of the input sequence, leading to frequent prediction errors.

Exercise 3.2: Encoder-Decoder Model with Attention

Model Implementation:

The model was enhanced by integrating the Bahdanau Attention mechanism to improve alignment between the encoder and decoder outputs:

- **Decoder Modification:** The `BahdanauAttention` class's `forward()` method was implemented to compute context vectors by weighing encoder outputs based on their relevance to each decoding step. The decoder was modified to dynamically combine this context vector with its LSTM output, enabling more accurate token prediction.
- **Attention Activation:** Attention was enabled by adding the `--use_attn` flag during training, allowing the decoder to dynamically focus on the most relevant parts of the input sequence.

Validation Performance

For training, the same hyperparameters from Exercise 3.1 were used, with the addition of attention. The validation CER over epochs is shown below:

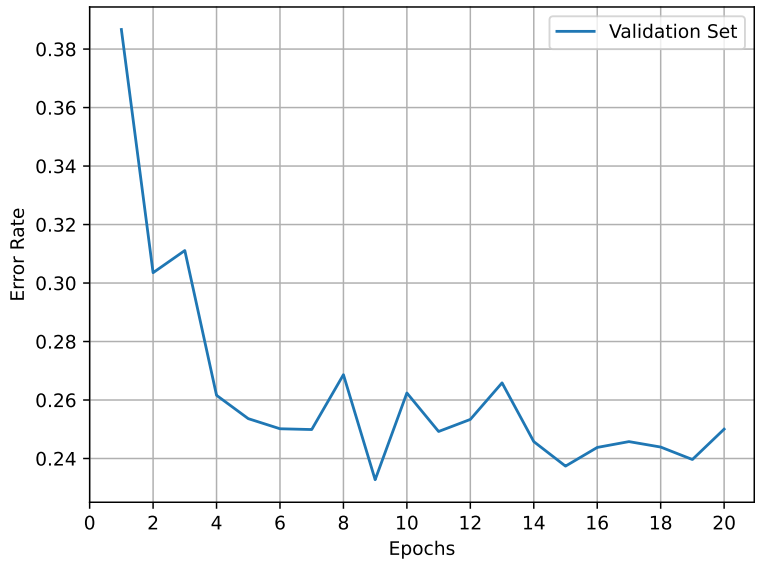


Figure 4: Validation CER across training epochs with Bahdanau Attention.

Final Results (With Attention)

- **Character Error Rate (CER):** 0.2171
- **Word Error Rate (WER):** 0.7270

Performance Analysis

The integration of Bahdanau Attention significantly improved performance, reducing CER by over 35% and WER by over 10% compared to the baseline model. By allowing the decoder to focus on the most relevant parts of the input sequence, the model better captured the relationships between phonemes and graphemes, leading to more accurate predictions.

Exercise 3.3: Nucleus Sampling for Diverse Predictions

Nucleus sampling (top-p sampling) was implemented to generate more diverse predictions. This method allows the model to sample outputs based on a cumulative probability distribution, introducing variability into the generated sequences.

Testing

The model with Bahdanau Attention was tested using nucleus sampling with $p = 0.8$.

Final Results

- Character Error Rate (CER): 0.2259
- Word Error Rate (WER): 0.7630
- Top-3 Word Error Rate (WER@3): 0.7370

Performance Analysis

Nucleus sampling introduced diversity into the output predictions but slightly degraded performance compared to greedy decoding. The CER increased to **0.2259**, and WER rose to **0.7630**. The **WER@3** score of **0.7370** indicates that while the model generated varied outputs, it was less precise in producing exact matches.

Analysis and Conclusion

- **Greedy Decoding:** Provided the best results for this task, achieving the lowest Character Error Rate (CER) of **0.2171** and Word Error Rate (WER) of **0.7270** when combined with the attention mechanism. Its step-by-step selection of the most probable token ensures precise and accurate outputs, making it well-suited for deterministic tasks like Phoneme-to-Grapheme (P2G) conversion where accuracy is critical.
- **Nucleus Sampling:** Introduced output diversity by sampling from the most probable tokens, but this came at the cost of slightly higher CER (**0.2259**) and WER (**0.7630**). The **Top-3 WER (0.7370)** suggests that while the model generated more varied outputs, it was less precise in producing exact matches. This method is better suited for creative tasks rather than structured ones like P2G, where exact outputs are necessary.
- **Impact of Attention Mechanism:** Incorporating the Bahdanau Attention mechanism significantly improved model performance by reducing CER by over **35%** and WER by over **10%** compared to the baseline. The attention mechanism allowed the decoder to focus on the most relevant parts of the input sequence, improving alignment between phonemes and graphemes and resulting in more accurate predictions.
- **Model Efficiency:** The use of a bidirectional LSTM encoder and dropout regularization led to stable and efficient training. These design choices helped the model generalize better to unseen data while effectively managing overfitting risks. However, the vanilla model without attention struggled to capture complex relationships, limiting its performance.

GitHub Link

You can find the code on GitHub at:
https://github.com/AlexGoncalves21/DeepLearning

References

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, *79*(8), 2554–2558.

Krotov, D., & Hopfield, J. J. (2016). Dense associative memory for pattern recognition. *Advances in Neural Information Processing Systems (NIPS)*.

Ramsauer, H., Schäfl, B., Lehner, S., Seidl, P., Widrich, M., Adler, T., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., Kreil, D. P., Kopp-Scheinpflug, C., Klambauer, G., & Hochreiter, S. (2020). Hopfield networks is all you need.

Yuille, A. L., & Rangarajan, A. (2001). The concave-convex procedure (cccp). *Advances in Neural Information Processing Systems (NIPS)*.