INSTITUTO SUPERIOR TÉCNICO
IST-2024 - DEEP LEARNING

Report - Deep Learning - HW2

Students:
BRUNO Elie
GAUGES Maira

Professor:
FIGUEIREDO Mario

January 10, 2025

Contents

1 Question I

1.1 Part 1

The energy equation $E(\mathbf{q})$ is given as

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

and can be re-written in the form

$$E(\mathbf{q}) = E_1(\mathbf{q}) + E_2(\mathbf{q}) \quad (2)$$

since $E(\mathbf{q})$ is the sum of $E_1(\mathbf{q})$ and $E_2(\mathbf{q})$ if these are defined as:

$$E_1(\mathbf{q}) = \frac{1}{2}\mathbf{q}^\top\mathbf{q} + \beta^{-1}\log N + \frac{1}{2}M^2 \quad (3)$$

$$E_2(\mathbf{q}) = -\text{lse}(\beta, \mathbf{Xq}). \quad (4)$$

The gradient of $E_1(\mathbf{q})$ can be derived in the following way:

In the calculation of the derivative only the first term $\frac{1}{2}\mathbf{q}^\top\mathbf{q}$ plays a role, since all others are independent of $\mathbf{q}$. Hence

$$\nabla E_1(\mathbf{q}) = \nabla\frac{1}{2}\mathbf{q}^\top\mathbf{q} = \nabla\frac{1}{2}(q_1^2 + q_2^2 + ... + q_D^2) = \mathbf{q} \quad (5)$$

The gradient of $-E_2(\mathbf{q})$ can be derived in the following way:

$$-E_2(\mathbf{q}) = \beta^{-1}\log\left(\sum_{i=1}^{N}\exp\left(\beta(\mathbf{Xq})_i\right)\right) \quad (6)$$

$$-\nabla E_2 = \beta^{-1}\cdot\frac{1}{\sum_{i=1}^{N}\exp\left(\beta(\mathbf{Xq})_i\right)}\nabla\left(\sum_{i=1}^{N}\exp\left(\beta(\mathbf{Xq})_i\right)\right) \quad (7)$$

where

$$\nabla\left(\sum_{i=1}^{N}\exp\left(\beta(\mathbf{Xq})_i\right)\right) \quad (8)$$

$$= \nabla\left(\exp\left(\beta(\mathbf{Xq})_1\right) + ... + \exp\left(\beta(\mathbf{Xq})_N\right)\right)$$

$$= \exp\left(\beta(\mathbf{Xq})\right)\cdot\beta X^\top$$

Therefore

$$-\nabla E_2 = \beta^{-1}\cdot\frac{1}{\sum_{i=1}^{N}\exp\left(\beta(\mathbf{Xq})_i\right)}\cdot\exp\left(\beta(\mathbf{Xq})\right)\cdot\beta\mathbf{X} \quad (9)$$

$$= \mathbf{X}^T\,\text{softmax}(\beta\mathbf{Xq})$$

as the softmax is defined as

$$\text{softmax}(\beta\mathbf{Xq}) = \frac{\exp\left(\beta\mathbf{Xq}\right)}{\sum_{i=1}^{N}\exp\left(\beta(\mathbf{Xq})_i\right)} \quad (10)$$

The next step is to calculate the Hessian of $E_1(\mathbf{q})$ and $-E_2(\mathbf{q})$. For $E_1(\mathbf{q})$ the Hessian is simply the identity matrix as

$$\nabla\mathbf{q} = \mathbf{I} \quad (11)$$

which is positive semi-definite (psd) as all diagonal elements are one and hence non-negative, it is a symmetric matrix matrix and diagonally dominant as only entries exist on the diagonal.

The Hessian of $-E_2(\mathbf{q})$ can be calculated using the rule of derivatives given as

$$\left(\frac{g}{h}\right)' = \frac{g'h - h'g}{h^2} \quad (12)$$

where we see g_i and h_i as

$$g_i = \exp(z_i) \quad h_i = \left(\sum_{i=k}^{N}\exp\left(\beta(\mathbf{Xq})_k\right)\right) \quad (13)$$

furthermore for simplification we define the function f_i as

$$f_i(z) = \text{softmax}(z)_i \quad (14)$$

A case differentiation is done for calculating the derivatives.

For the case $j = i$

$$\frac{\partial g_i}{\partial z_j} = \frac{\partial}{\partial z_j}\exp(z_i) = \exp(z_j) \quad (15)$$

$$\frac{\partial h_i}{\partial z_j} = \exp(z_j) \quad (16)$$

therefore

$$\frac{\partial f_i}{\partial z_j} = \frac{e^{z_i}\cdot\sum - e^{z_i}\cdot e^{z_i}}{\left(\sum\right)^2} = f_i(1 - f_i) \quad (17)$$

where $\sum$ is an abreviation of $\sum = \left(\sum_{i=k}^{N}\exp\left(\beta(\mathbf{Xq})_k\right)\right)$.

For the case $j \neq i$

$$\frac{\partial g_i}{\partial z_j} = 0 \quad \frac{\partial h_i}{\partial z_j} = \exp(z_j) \quad (18)$$

and therefore

$$\frac{\partial f_i}{\partial z_j} = \frac{0 - e^{z_i}e^{z_j}}{\left(\sum\right)^2} = -f_if_j \quad (19)$$

Summarizing both cases this gives

$$\frac{\partial f_i}{\partial z_j} = \begin{cases} f_i(1 - f_i) & \text{if } i = j, \\ -f_if_j & \text{if } i \neq j. \end{cases} \quad (20)$$

Furthermore $\frac{\partial z_i}{\partial q_j}$ must be considered:

$$\frac{\partial z_i}{\partial q_j} = \beta\mathbf{X} \quad (21)$$

The overall derivative can be written in matrix form as

$$-\nabla^2 E_2 = \beta\,\mathbf{X}^\top\left(\text{diag}(\mathbf{f}) - \mathbf{ff}^\top\right)\mathbf{X} \quad (22)$$

This furthermore can be re-written as

$$-\nabla^2 E_2 = \beta\,\mathbf{X}^\top\mathbf{MX} \quad (23)$$

To check if $\mathbf{M}$ is a psd matrix the following inequality must hold for any vector $\mathbf{v}$

$$\mathbf{v}^\top\mathbf{Mv} \geq 0 \quad (24)$$

For the given M it can be written as

$$\mathbf{v}^\top M\mathbf{v} = \mathbf{v}^\top \left(\text{diag}(\mathbf{f}) - \mathbf{f}\mathbf{f}^\top\right)\mathbf{v} \tag{25}$$

$$= \mathbf{v}^\top \text{diag}(\mathbf{f})\mathbf{v} - \mathbf{v}^\top\mathbf{f}\mathbf{f}^\top\mathbf{v}$$

$$= \sum_i v_i^2 f_i - \left(\sum_i v_i f_i\right)^2 \tag{26}$$

We can show that this term is greater or equal to zero using the Jensen equality since $\sum_i f_i = 1$.

The second term can be written as

$$\left(\sum_i v_i f_i\right)^2 = g\left(\sum_i f_i v_i\right) \tag{27}$$

and the first term as

$$\sum_i v_i^2 f_i = \sum_i g(v_i) f_i \tag{28}$$

when $g(x) = x^2$. According to the Jensen inequality therefore

$$\sum_i v_i^2 f_i \geq \left(\sum_i v_i f_i\right)^2 \tag{29}$$

and therefore the inequality

$$= \sum_i v_i^2 f_i - \left(\sum_i v_i f_i\right)^2 \geq 0 \tag{30}$$

holds true and we can conclude that $\mathbf{M}$ is psd. From this we know that $\beta\,\mathbf{X}^\top\mathbf{M}\mathbf{X}$ is also psd since β is a scalar and

$$\mathbf{v}^\top\mathbf{X}^\top\mathbf{M}\mathbf{v}\mathbf{X} = (\mathbf{X}\mathbf{v})^\top \mathbf{M}\,(\mathbf{v}\mathbf{X}) = \mathbf{u}^\top\mathbf{M}\mathbf{u} \geq 0 \tag{31}$$

where $\mathbf{u} = \mathbf{v}\mathbf{X}$.

Following this it is shown that $-\nabla^2 E_2$ is psd.

1.2 Part 2

First E_2 is linearized around $\mathbf{q_t}$ by

$$E_2(q) \approx \tilde{E}_2(q) := E_2(q_t) + (\nabla E_2(q_t))^T (q - q_t) \tag{32}$$

which yields

$$\tilde{E}_2(q) = -\text{lse}(\beta\mathbf{X}\mathbf{q}) - \text{softmax}(\beta\mathbf{X}\mathbf{q})^\top\mathbf{X}(\mathbf{q} - \mathbf{q}_t) \tag{33}$$

The next iterative $\mathbf{q_{t+1}}$ is computed as

$$\mathbf{q}_{t+1} = \arg\min_{\mathbf{q}} \left(E_1(\mathbf{q}) + \tilde{E}_2(\mathbf{q})\right) \tag{34}$$

Hence the one finds the derivative of $E_1(\mathbf{q}) + \tilde{E}_2(\mathbf{q})$ and sets this to zero to solve for $\mathbf{q}$:

$$\nabla\left(E_1(\mathbf{q}) + E_2(\mathbf{q}_t) + (\nabla E_2(\mathbf{q}_t))^\top\mathbf{q} - (\nabla E_2(\mathbf{q}_t))^\top\mathbf{q}_t\right)$$

$$= \nabla_q E_1(\mathbf{q}) + \nabla_q(\nabla E_2(\mathbf{q}_t)^\top\mathbf{q})$$

Setting this to zero

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

and solving for $\mathbf{q}$

$$\mathbf{q} = \mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q_t}) \tag{35}$$

hence

$$\mathbf{q}_{t+1} = \mathbf{X}^\top\text{softmax}(\beta\mathbf{X}\mathbf{q_t}) \tag{36}$$

1.3 Part 3

The computation performed in the cross-attention layer of a transformer with a single attention head is given as

$$Z = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_K}}\right)\mathbf{V} \tag{37}$$

When $\mathbf{W}_K = \mathbf{I}$ and $\mathbf{W}_V = \mathbf{I}$ $\mathbf{K}$ and $\mathbf{V}$ simplify to

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K = \mathbf{X} \tag{38}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_V = \mathbf{X} \tag{39}$$

and $d_k = D$. Then

$$Z = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{X}^\top}{\sqrt{D}}\right)\mathbf{X} = \text{softmax}\left(\frac{(\mathbf{X}\mathbf{Q}^\top)^\top}{\sqrt{D}}\right)\mathbf{X}$$

$$Z^\top = \mathbf{X}^\top\text{softmax}\left(\frac{(\mathbf{X}\mathbf{Q}^\top)}{\sqrt{D}}\right)$$

The calculation of $\mathbf{q}_{t+1}$ is a part of the transformer calculation and corresponds to row t in $Z^\top$.

2 Question 2: Image Classification with CNNs

2.1 2.1 Basic CNN Implementation

We implemented a convolutional neural network for the Intel Image Classification dataset with three convolutional blocks followed by an MLP classifier. The results of our training with different learning rates are shown in figs. 3 and 4. The initial architecture consisted of:

- Three convolutional blocks with output channels (32, 64, 128)
- Each block containing:
  - 3x3 convolution with stride 1 and padding 1
  - ReLU activation
  - 2x2 max pooling with stride 2
  - Dropout (p = 0.1)
- MLP block with:
  - Flattened convolutional output
  - Dense layer (1024 units)
  - ReLU activation
  - Dropout (p = 0.1)
  - Dense layer (512 units)
  - ReLU activation
  - Output layer (6 units) with LogSoftmax

As shown in figs. 3 and 4, after testing different learning rates (0.1, 0.01, 0.001), we found that lr = 0.01 provided the best performance. This configuration achieved a test accuracy of 70.33% with 5,340,742 trainable parameters. The model demonstrated stable learning behavior, though there was room for improvement in terms of both accuracy and efficiency.

2.2 2.2 Enhanced CNN with Batch Normalization

We then modified the architecture to include batch normalization and global average pooling, with results shown in fig. 5:

- Added batch normalization after each convolution layer
- Replaced flattening with global average pooling (1x1 kernel)
- Added batch normalization in the MLP block before dropout

This enhanced version showed significant improvements, as evident in fig. 5:

- Increased test accuracy to 75.23% (+4.9% absolute improvement)
- Reduced model parameters to 756,742 (85.8% reduction)
- Better training stability with:
  - Training loss: 0.5421
  - Validation loss: 0.6795
  - Validation accuracy: 75.43%

2.3 Analysis of Improvements

The significant improvements can be attributed to several factors, which are reflected in the training metrics shown in fig. 5:

2.3.1 1. Batch Normalization Effects

The addition of batch normalization layers provided several benefits:

- Reduced internal covariate shift
- Enabled faster training with more stable gradients
- Acted as a regularizer, improving generalization
- Allowed for better gradient flow through the network

2.3.2 2. Global Average Pooling

Replacing the flattening operation with global average pooling:

- Drastically reduced the number of parameters (from 5.3M to 756K)
- Enforced stronger spatial feature learning
- Improved generalization by reducing overfitting
- Made the model more robust to spatial translations

2.4 Analysis of Parameter Count Differences

We implemented the `get_number_trainable_params` function to analyze the number of trainable parameters in both networks. The significant difference in parameter count (5,340,742 vs 756,742) can be attributed to several key factors:

1. **Flattening vs Global Average Pooling**: The most substantial reduction comes from replacing the flattening operation with global average pooling:
   - In the original network, flattening created a large feature vector (128 * width * height) that connected to the first dense layer (1024 units)
   - Global average pooling reduces each feature map to a single value, resulting in just 128 features connecting to the dense layer
   - This change alone eliminated millions of parameters in the first dense layer connections
2. **Preserved Information Flow**: Despite the massive reduction in parameters, the network maintains (and even improves) its performance because:
   - Global average pooling maintains the spatial relationship of features
   - Batch normalization helps in better gradient flow and feature normalization
   - The reduced parameter count acts as a form of regularization, preventing overfitting

The improved performance with fewer parameters demonstrates that the original architecture was overparameterized, and the modifications helped create a more efficient and effective model.

2.5 Kernel Size and Pooling Effects

The choice of small kernels (3x3) instead of larger kernels (width $\times$ height) and the use of pooling layers are fundamental design decisions in modern CNNs:

2.5.1 Small Kernel Benefits

- **Parameter Efficiency**: Small kernels require significantly fewer parameters. A 3x3 kernel needs only 9 parameters per channel, while a full width$\times$height kernel would need width$\times$height parameters.
- **Hierarchical Feature Learning**: Stacking multiple layers with small kernels allows the network to:
  - Build increasingly complex features layer by layer
  - Capture the same receptive field as larger kernels with fewer parameters
  - Learn more discriminative features through multiple non-linear transformations
- **Better Generalization**: Small kernels force the network to learn local patterns first before combining them into more complex features, leading to better generalization.

2.5.2 Pooling Layer Effects

The pooling layers (2x2 max pooling in our implementation) serve several crucial purposes:

- **Dimensionality Reduction**: Each pooling layer reduces spatial dimensions by half, making computation more efficient.
- **Translation Invariance**: Max pooling helps the network become more robust to small translations in the input, as it preserves the strongest feature activations in each region.
- **Hierarchical Processing**: Combined with the convolution layers, pooling helps create a hierarchy where:
  - Early layers detect fine details (edges, textures)
  - Middle layers combine these into parts
  - Later layers recognize higher-level concepts
- **Increased Receptive Field**: Each neuron in deeper layers can "see" a larger portion of the input image, allowing the network to capture more context while maintaining parameter efficiency.

This combination of small kernels and pooling layers creates a powerful architecture that can efficiently learn hierarchical representations while maintaining computational feasibility and good generalization properties.

2.6 Future Considerations

Potential areas for further improvement could include:

- Experimenting with different learning rate schedules
- Investigating other normalization techniques (e.g., Layer Normalization)
- Testing different dropout rates

2.7 Conclusion

The enhanced CNN architecture demonstrated substantial improvements over the baseline model in multiple aspects, as evidenced by the comparison between figs. 3 and 4 and fig. 5:

1. **Efficiency**: The 85.8% reduction in parameters while improving accuracy suggests a more efficient use of model capacity.

2. **Performance**: The 4.9% absolute improvement in test accuracy indicates better generalization.

3. **Training Stability**: The combination of batch normalization and global average pooling led to more stable training dynamics.

These results highlight the importance of modern architectural choices in CNN design, particularly the synergistic effects of batch normalization and global average pooling. The improvements came not from adding complexity, but from making the architecture more efficient and trainable.

3 Question III

3.0.1 a)

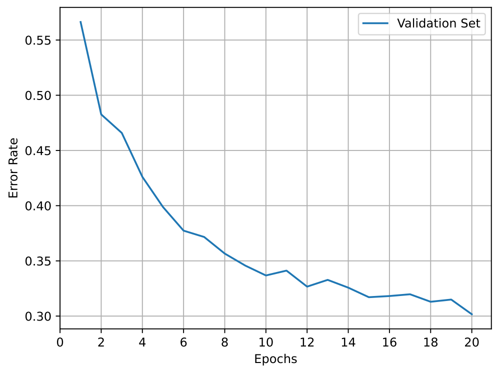


Figure 1: CER at the end of each epoch (no attention)

Test CER: 0.2926, Test WER: 0.7950

3.1 b) Bahdanau Attention

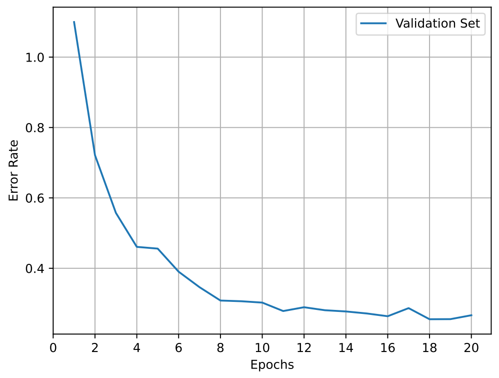


Figure 2: CER at the end of each epoch (including attention)

Test CER: 0.2410, Test WER: 0.7840

3.2 c)

Test CER: 0.2594, Test WER: 0.8170
Test WER@3: 0.7390

4 Project Workflow

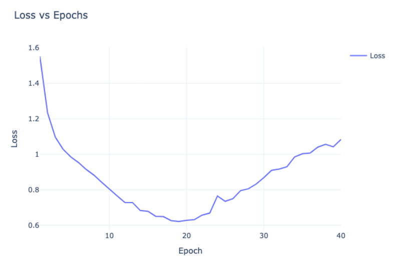
4.1 Task Division

When working on this project we split up the tasks between the two of us. Results to question 3 and question 1 were provided by Maira Gauges and while Elie Bruno worked on question 2. Each wrote the code to obtain the results and included results in the report. The first question was mainly done by Maira Gauges and Elie reviewed it.

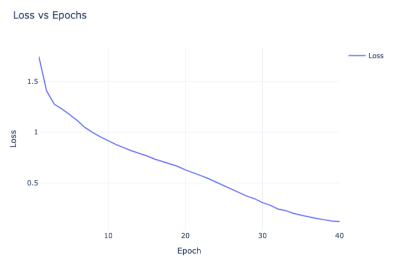
4.2 Use of Chat GPT / AI Tools

AI tools were used to help with the review of the code and to provide suggestions for improving the efficiency and readability of the code. Additionally, they were utilized to assist in the latex report to properly add the images and graphs. It was furthermore used to understand how certain pytorch functions work and how they can be used. In particular for working with tensors (reshaping, slicing, ...) AI gave helpful tipps.

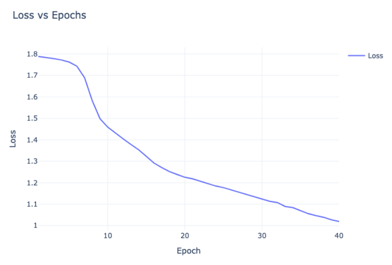
5 Appendix: Graphs and Visualizations



(a) LR = 0.1

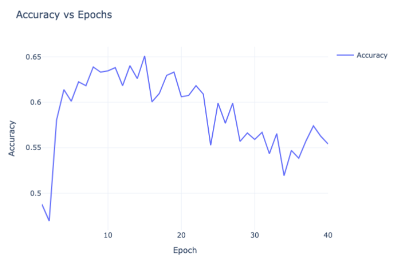


(b) LR = 0.01

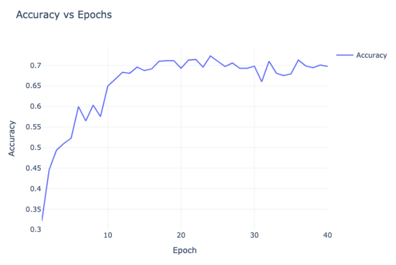


(c) LR = 0.001

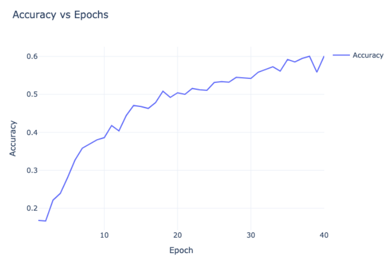
Figure 3: Training Loss for different Learning Rates without Batch Normalization



(a) LR = 0.1

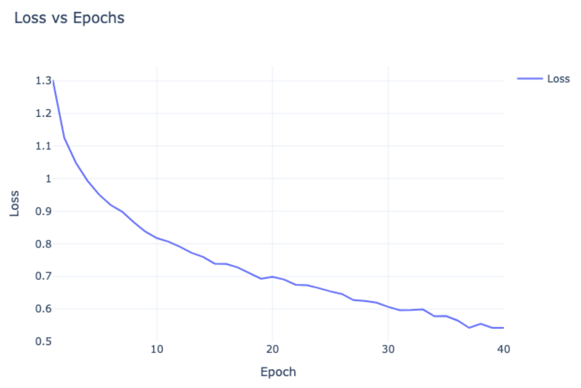


(b) LR = 0.01

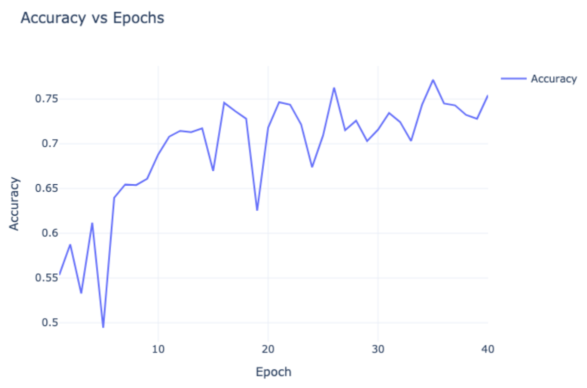


(c) LR = 0.001

Figure 4: Validation Accuracy for different Learning Rates without Batch Normalization



(a) Training Loss



(b) Validation Accuracy

Figure 5: Training Metrics with Batch Normalization (LR = 0.01)