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

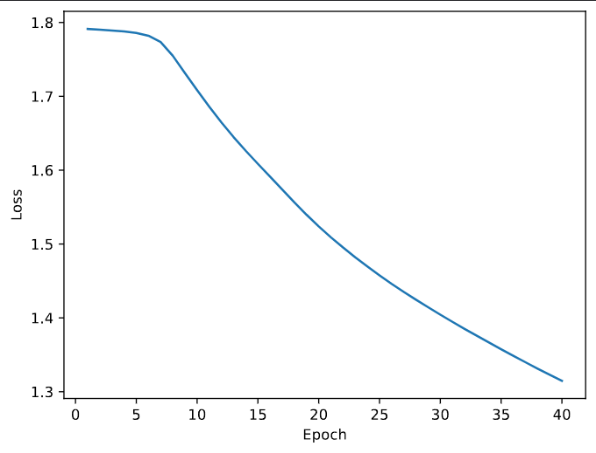
1. The following table shows the final validation and test accuracies for each learning rate.

|  |  |  |
| --- | --- | --- |
| **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

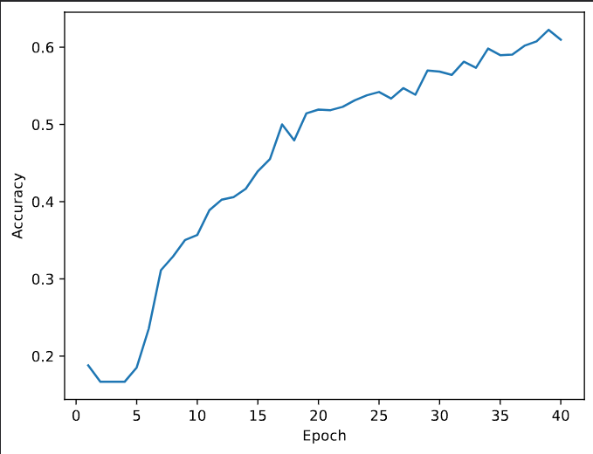
The best configuration was the one with a learning rate of 0.01, yielding a validation accuracy of 70.30% and a test accuracy of 69.00%.

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



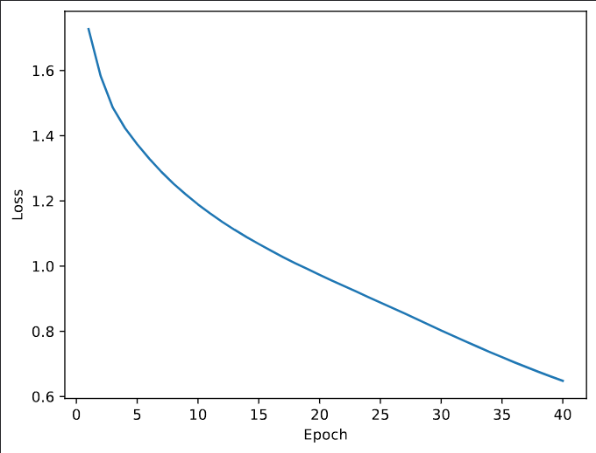
Plot 1 (η = 0.001)– Training loss in function of the epoch number

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

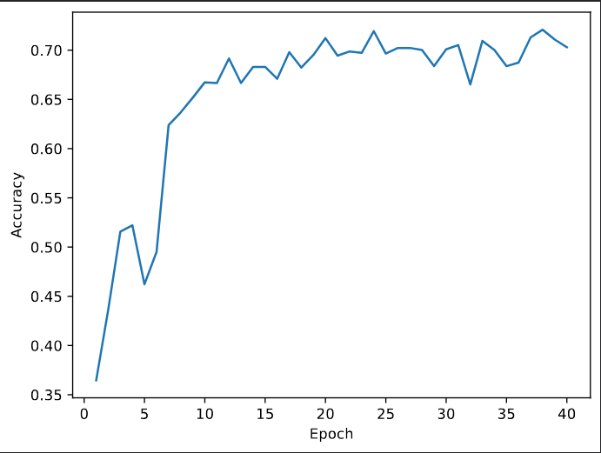








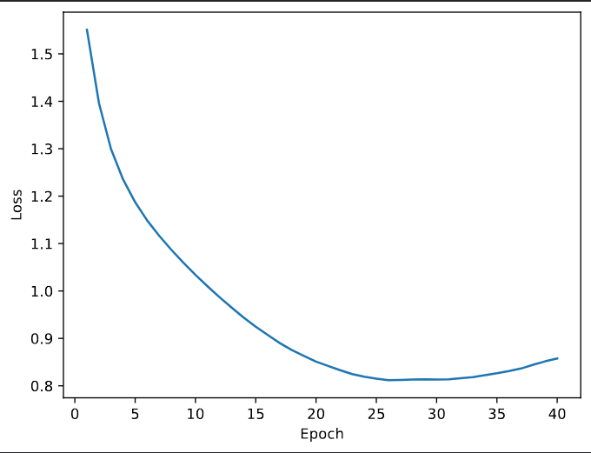
Plot 4 (η = 0.01)– Validation accuracy in function of the epoch number



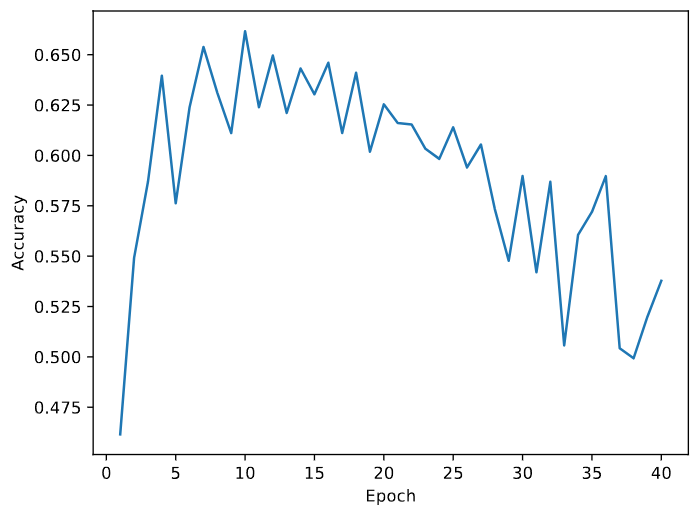
Plot 3 (η = 0.01)– Training loss in function of the epoch number



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



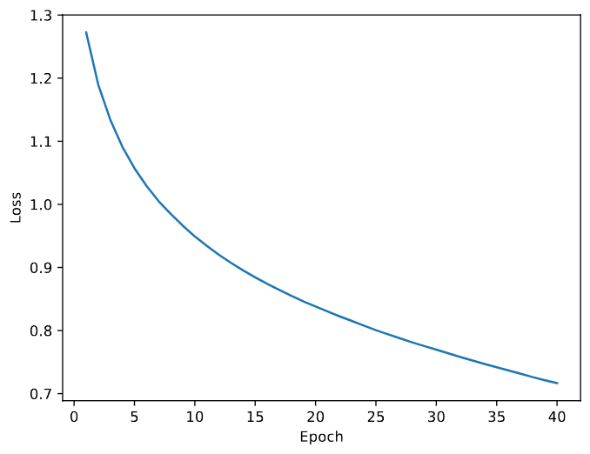
Plot 6 (η = 0.1)– Validation accuracy in function of the epoch number



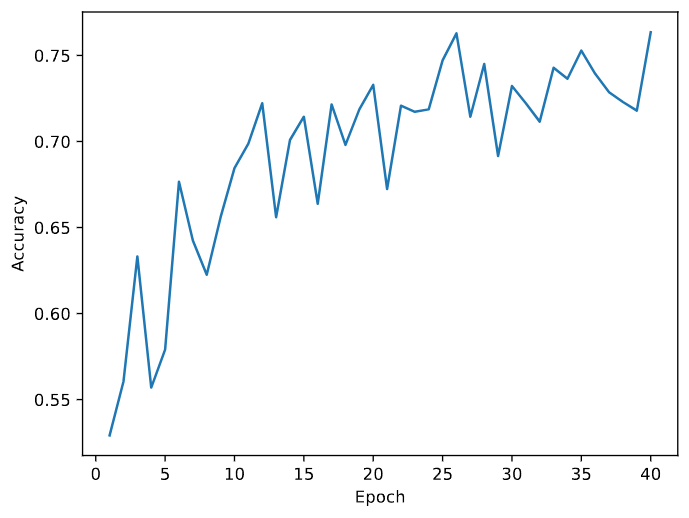
1. Since the best configuration from the previous question was the one with a learning rate of 0.01, it was chosen for this question. The resulting network, with batch normalization, achieved a validation accuracy of 76.35% and a test accuracy of 76.23%, representing a considerable improvement in performance 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



1. 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:

1. **Internal Covariate Shift Reduction:** with BatchNorm, the intermediate feature maps are normalized, reducing the internal covariate shift;
2. **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;
3. **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 × 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. 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

1. 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

1. The results obtained after running the implemented function and evaluating the model on the test set are as follows:|
   1. Character Error Rate (CER): 0.2265
   2. Word Error Rate (WER): 0.7400
   3. 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/