Deep Learning – Homework 1

Group 40

99238 – Inês Ji | 99314 – Raquel Cardoso

**Contribution:** both members of the group were involved in the resolution of all questions in the homework.

## **Question 1**

### A graph with orange and blue lines Description automatically generated1.a)

Figure 1 – Percepton: train and validation accuracies as a function of the epoch number

The performances with accuracy as the chosen metric were the following: 0.4654 on the training set, 0.4610 on the validation set and 0.3422 on the test set.

### 1.b)

* A graph of a graph

  Description automatically generated with medium confidenceLearning rate (η) = 0.01

Figure 2 – Logistic Regression: train and validation accuracies as a function of the epoch number (η = 0.01)

The performances with accuracy as the chosen metric were the following: 0.6609 on the training set, 0.6568 on the validation set and 0.5784 on the test set.

* A graph with blue and orange lines

  Description automatically generatedLearning rate (η) = 0.001

Figure 3 - Logistic Regression: train and validation accuracies as a function of the epoch number (η = 0.01)

The performances with accuracy as the chosen metric were the following: 0.6625 on the training set, 0.6639 on the validation set and 0.5936 on the test set.

Based on the plots obtained with two different learning rates (η = 0.01 and η = 0.001) we can determine that the model with η = 0.001 has better final results and has a more consistent learning process than the one with η = 0.01. This may happen because η = 0.01 may be too big, which can cause the algorithm to overshoot the optimal weights and potentially miss the convergence to the minimum.

### 2.a)

The claim is true. A logistic regression model calculates the sigmoid function of a linear combination (b0+b1X1+b2X2+…+bkXk) involving the input features (X), which in this case represent pixel values, and learned coefficients (b) during training. This linear nature signifies that it inherently assumes a linear relationship between the pixel values and the output. Moreover, logistic regression has the advantage of being a convex optimization problem, which means it has an unique global minimum during training.

A multi-layer perceptron using ReLU activations – which is a non-linear activation function – possesses the capacity to learn a broader spectrum of complex, non-linear relationships within input features, such as pixel values, capturing possible intricate patterns in the image data. If the activation function was linear, we wouldn’t be able to learn those same patterns. However, the expressive power of an MLP with ReLU activations comes at the cost of a non-convex optimization problem, since the ReLU function doesn’t have a unique global minimum.

The presence of multiple local minima is an added challenge to the training process, since it necessitates the use of optimization algorithms like stochastic gradient descent to navigate the parameter space in search of a local minimum. This difficulty in training contributed to the limited popularity of Neural Networks in machine learning from 1990 to 2010.

In summary, an MLP with ReLU activations, especially in the context of image data represented by pixel values, demonstrates greater expressiveness than a logistic regression model, as it can adeptly capture intricate patterns. Nonetheless, training logistic regression remains comparatively easier due to its convex optimization nature, offering a unique global minimum during the optimization process.

### A graph of a line Description automatically generated with medium confidence2.b)

Figure 4 - MLP: train and validation accuracies as a function of the epoch number

Figure 5 - MLP: train loss as a function of the epoch number

The performances with accuracy as the chosen metric were the following: 0.7975 on the training set, 0.7865 on the validation set and 0.7524 on the test set. The loss value was 58093.7315.

## **Question 2**

### 1.

* A graph with a line and a line

  Description automatically generatedA graph with a line

  Description automatically generatedLearning rate (η) = 0.1

Figure 6 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number (η = 0.1)

Figure 7 - PyTorch Logistic Regression: train and valid loss as a function of the epoch number (η = 0.1)

The final validation accuracy was 0.6224 and the final accuracy on the test set was 0.5577. The training loss value was 0.9687.

* Learning rate (η) = 0.01

A graph with a line

Description automatically generated A graph of a line

Description automatically generated with medium confidence

Figure 9 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number (η = 0.01)

Figure 8 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number (η = 0.01)

The final validation accuracy was 0.6535 and the final accuracy on the test set was 0.6200. The training loss value was 0.9370.

* Learning rate (η) = 0.001

A graph with a line

Description automatically generatedA graph of a line

Description automatically generated

Figure 11 – PyTorch Logistic Regression: train and valid loss as a function of the epoch number (η = 0.001)

Figure 10 – PyTorch Logistic Regression: validation accuracy as a function of the epoch number (η = 0.001)

The final validation accuracy was 0.6163 and the final accuracy on the test set was 0.6503. The training loss value was 1.0145.

In terms of higher final validation accuracy, the best configuration is the one with learning rate = 0.01 with final accuracy on the test set = 0.6200.

### 2.a)

* Batch size = 16

A line graph with numbers and a blue line

Description automatically generatedA graph of a line

Description automatically generated with medium confidence

Figure 13 – PyTorch MLP: train and valid loss as a function of the epoch number (batch size = 16)

Figure 12 – PyTorch MLP: validation accuracy as a function of the epoch number (batch size = 16)

The final validation accuracy was 0.8253 and the final accuracy on the test set was 0.7505. The training loss value was 0.4358. Real time = 1m50.673s.

* Batch size = 1024

A line graph with numbers and symbols

Description automatically generated A graph of a line

Description automatically generated with medium confidence

Figure 14 – PyTorch MLP: train and valid loss as a function of the epoch number (batch size = 1024)

Figure 14 – PyTorch MLP: validation accuracy as a function of the epoch number (batch size = 1024)

The final validation accuracy was 0.6953 and the final accuracy on the test set was 0.7316. The training loss value was 0.8706. Real time = 22.545s.

The best test accuracy is 0.7505 on the configuration with batch size = 16. The performance is better in all parameters in this configuration and in terms of time, the difference between the two is 1m28.128s, being the batch size = 16 the slower one. This is in concordance to the general rule that a higher batch size leads to faster training because of modern hardware vectorization and parallelization capacities. About the performance, smaller batch sizes such as 16 have some advantages that can justify its better performance: they generalize better because they introduce some noise to the training data which prevents overfitting, they also converge faster during training, which makes the model adapt more quickly.

### 2.b)

* Learning rate (η) = 1

A graph with blue lines

Description automatically generated A graph with numbers and lines

Description automatically generated

Figure 17 – PyTorch MLP: train and valid loss as a function of the epoch number (η = 1)

Figure 16 – PyTorch MLP: validation accuracy as a function of the epoch number (η = 1)

The best validation accuracy was 0.4898 (epoch number 2), the final validation accuracy was 0.4721 and the final accuracy on the test set was 0.4726. The training loss value was 1.1714.

* Learning rate (η) = 0.1

A line graph with numbers and a blue line

Description automatically generatedA graph of a line

Description automatically generated

Figure 18 – PyTorch MLP: validation accuracy as a function of the epoch number (η = 0.1)

Figure 19 – PyTorch MLP: train and valid loss as a function of the epoch number (η = 0.1)

The best validation accuracy was 0.8258 (epoch number 17), the final validation accuracy was 0.8253 and the final accuracy on the test set was 0.7505. The training loss value was 0.4358.

* Learning rate (η) = 0.01

A graph with a line

Description automatically generatedA graph of a line

Description automatically generated

Figure 21 – PyTorch MLP: train and valid loss as a function of the epoch number (η = 0.01)

Figure 20 – PyTorch MLP: validation accuracy as a function of the epoch number (η = 0.01)

The final (and best) validation accuracy was 0.8143 and the final accuracy on the test set was 0.7637. The training loss value was 0.4848.

* Learning rate (η) = 0.001

A line graph with numbers and a line

Description automatically generated with medium confidenceA graph of a line

Description automatically generated with medium confidence

Figure 22 – PyTorch MLP: validation accuracy as a function of the epoch number (η = 0.001)

Figure 23 – PyTorch MLP: train and valid loss as a function of the epoch number (η = 0.001)

The final (and best) validation accuracy was 0.6917 and the final accuracy on the test set was 0.7146. The training loss value was 0.8421.

The best test accuracy is 0.7637, in the configuration with learning rate = 0.01.

The best configuration in terms of best validation accuracy, is the one with learning rate = 0.1.

The worst configuration in terms of best validation accuracy, is the one with learning rate = 1.

The difference in performance is extremely noticeable between the configuration with learning rate = 1 and all the other ones, being the first one the worst configuration among the four, as mentioned above. The configurations with learning rate = 0.1 and learning rate = 0.01 have extremely similar performance, the first one has the best validation accuracy and the second one has the best test accuracy. The configuration with learning rate = 0.001 has worse performance than the ones with learning rate = 0.1 and learning rate = 0.01, especially in terms of validation accuracy.

The performance is directly affected by the learning rates. One possible reason for the performance to be bad when the learning rate = 1 is because this particular learning rate is just too big for this model, which causes the algorithm to overshoot the optimal weights and diverge. For learning rates 0.1 or 0.01 the model seems to be a good fit, as the loss plots decrease then begin to stabilize, and they have the best values of validation accuracy and test accuracy. When the learning rate = 0.001 it’s possible to notice from the loss plots that the model is clearly underfit: this may be because the network is converging too slowly.

### 2.c)

* Batch size = 256 | Epochs = 150

A graph with blue lines

Description automatically generatedA graph of a line graph

Description automatically generated with medium confidence

Figure 25 – PyTorch MLP: train and valid loss as a function of the epoch number

Figure 24 – PyTorch MLP: validation accuracy as a function of the epoch number

The best validation accuracy was 0.8620 (epoch number 147), the final validation accuracy was 0.8593 and the final accuracy on the test set was 0.7637. The training loss value was 0.2166.

Overfitting happens when the model learns the statistical noise or random flunctuations and the significant data (signal), instead of only the significant data. In this model, there is overfitting because the plot of the training loss keeps decreasing, while the plot of validation loss keeps decreasing and increasing instead of decreasing to a point of stability. The gap between the two plots is also significant and it should have been smaller to be considered a good fit.

* Batch size = 256 | Epochs = 150 | L2 regularization = 0.0001

A graph with blue lines

Description automatically generated A graph of a line

Description automatically generated with medium confidence

Figure 27 – PyTorch MLP: train and valid loss as a function of the epoch number

Figure 26 – PyTorch MLP: validation accuracy as a function of the epoch number

The best validation accuracy was 0.8595 (epoch number 147), the final validation accuracy was 0.8458, and the final accuracy on the test set was 0.7694. The training loss value was 0.2742.

* Batch size = 256 | Epochs = 150 | Dropout = 0.2

A graph with a line

Description automatically generated A graph of a line

Description automatically generated

Figure 29 – PyTorch MLP: train and valid loss as a function of the epoch number

Figure 28 – PyTorch MLP: validation accuracy as a function of the epoch number

The best validation accuracy was 0.8588 (epoch number 146), the final validation accuracy was 0.8563 and the final accuracy on the test set was 0.7845. The training loss value was 0.3666.

The best configuration in terms of best validation accuracy, is the one without any technique.

The worst configuration in terms of best validation accuracy, is the one with dropout = 0.2.

The best test accuracy is 0.7845 in the configuration with dropout = 0.2.

Both L2 Regularization and Dropout probability are both techniques used to prevent overfitting. Although they are mutually exclusive in this exercise they can also be used together. L2 Regularization imposes a penalty on large weights, so the neural network uses all features in the training data. Dropout probability, as the name suggests, is a technique in which there’s a chance that random weights are set to 0. This prevents overfitting by introducing some randomness in the learning process.

## **Question 3**

### a)

A close-up of a paper

Description automatically generated

### b)

A close-up of a paper

Description automatically generated

### A close-up of a white paper Description automatically generated

### A close-up of a paper with mathematical equations Description automatically generatedc)

