Deep Learning (IST, 2024-25)  
Homework 1

Group 23

113191 – Miguel Miranda | 102787 – Ricardo Santos

**Question 1.**

1. **(a)**

A graph of blue and orange lines

Description automatically generated

Figure 1 - Training and Validation accuracies as a function of the epoch number for the Perceptron

By the end of the 100 epochs the performances with accuracy as the chosen metric were the following: 0.5598 on the training set; 0.3868 on the validation; 0.3743 on the testing set.

1. **(b)**

|  |  |
| --- | --- |
| Figure 2 - Training and Validation accuracies as a function of the epoch number for the non-regularized Logistic Regression | Figure 3 - Training and Validation accuracies as a function of the epoch number for Logistic Regression with L2 regularization (l2\_penalty = 0.01) |

By the end of the 100 epochs the non-regularized version of the Logistic Regression obtained the following results with accuracy as the chosen metric: 0.6694 on the training set; 0.4623 on the validation set; 0.4597 on the testing set.

By the end of the 100 epochs the Logistic Regression with L2 regularization obtained the following results with accuracy as the chosen metric: 0.5683 on the training set; 0.4972 on the validation set; 0.5053 on the testing set.

The non-regularized version of logistic regression achieves a higher training accuracy (0.6694) because it fits the training data closely, allowing weights to grow large. However, this leads to poor generalization, as seen in the much lower validation (0.4623) and testing (0.4597) accuracies. This discrepancy indicates the model is overfitting to the training set.

In contrast, with L2 regularization (l2\_penalty = 0.01), the model achieves a lower training accuracy (0.5683) due to the constraint on weight magnitudes. This prevents the model from fitting overly complex patterns. However, this trade-off results in better generalization, as indicated by the higher validation (0.4972) and testing (0.5053) accuracies. This suggests the regularized model is capturing more general patterns in the data, reducing overfitting and improving performance on unseen data.

1. **(c)**

|  |  |
| --- | --- |
| A graph with a blue line  Description automatically generated  Figure 4 - L2 norm of the weights as a function of the epoch number for the non-regularized Logistic Regression | Figure 5 - L2 norm of the weights as a function of the epoch number for the Logistic Regression with L2 Regularization |

The L2-norm of the weights in the non-regularized logistic regression increases steadily throughout the 100 epochs. This continuous growth indicates that the model is trying to fit the training data as closely as possible by increasing the weight’s magnitudes, which ultimately leads to overfitting. As the weights become large, the model captures specific details and noise in the training data, causing poor performance on unseen data.

In the L2-regularized version, the L2-norm of the weights stabilizes early (after about 20 epochs) and remains relatively constant. This suggests that the regularization term effectively penalizes large weights, preventing them from growing excessively. As a result, the model maintains smaller weights, which helps reduce overfitting and promotes better generalization. This behavior is evident from the improved validation and testing accuracies compared to the non-regularized version.

1. **(d)**

Despite both L1 and L2 regularization promoting smaller weights, L1 regularization uniquely promotes sparsity, meaning that some weight values can be set to exactly zero during the regularization process.

L1 regularization adds a penalty proportional to the absolute value of the weights (||). This penalty can force weights to shrink to zero when the benefit of a weight's contribution to reducing the loss is outweighed by the penalty.

L2 regularization, on the other hand, adds a penalty proportional to the square of the weights (). This encourages weights to become smaller but does not typically shrink them to zero.

Because L1 regularization can lead to some weights being set to zero, it results in a simpler, sparser model that uses fewer features. In comparison, L2 regularization tends to produce a model where all features still contribute, though with smaller weights.

**Question 2.**

**1.**

The learning rate that achieved the highest validation accuracy is lr = 0.001, with final test accuracy of 0.5247 and final validation accuracy of 0.5264.

|  |  |
| --- | --- |
| Figure 6 - Training and Validation Loss as a function of the epoch number for the Logistic Regression using PyTorch (lr = 0.001) | Figure 7 - Validation Accuracy as a function of the epoch number for the Logistic Regression using PyTorch (lr = 0.001) |
| Figure 8 - Training and Validation Loss as a function of the epoch number for the Logistic Regression using PyTorch (lr = 0.1) | Figure 9 - Training and Validation Loss as a function of the epoch number for the Logistic Regression using PyTorch (lr = 0.00001) |

From the figures above we can see how the different learning rate values impact the model’s performance.  
With Learning Rate = 0.1 we can see that the loss values are extremely high and fluctuate significantly, which indicates that the model is likely experiencing instability due to large weight updates.

For Learning Rate = 0.00001 although the loss values are relatively low, they remain higher than those observed for a learning rate of 0.001. This suggests that the learning rate is too small, which causes the model to make minimal progress during training.

Finally, for Learning Rate = 0.001, we see that this is the configuration that achieves the highest validation accuracy and the lowest validation loss, indicating that this value provides a good balance between convergence speed and stability.

1. **(a)**

|  |  |
| --- | --- |
| Figure 10 – Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 64) | Figure 11 - Validation Accuracy as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 64) |
| Figure 12 - Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 512) | Figure 13 - Validation Accuracy as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 512) |

For batch\_size = 64 the final results were: validation accuracy = 0.6061; test accuracy = 0.6093; validation loss = 1.0197; test loss = 0.7804 and training time 1 minute and 4 seconds.

For batch\_size = 512 the final results were: validation accuracy = 0.5028; test accuracy = 0.5190; validation loss = 1.2562; test loss = 1.2551 and training time 0 minutes and 40 seconds.

The model trained with a smaller batch size (64) achieved higher validation accuracy because it allows for more frequent weight updates, enabling the model to adapt more effectively to the training data. However, the increased number of iterations per epoch leads to longer training times. In contrast, a larger batch size (512) results in faster training due to fewer iterations per epoch but sacrifices performance in terms of accuracy and loss.

1. **(b)**

|  |  |  |
| --- | --- | --- |
| Figure 14 - Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (dropout = 0.01) | Figure 15 - Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (dropout = 0.25) | Figure 16 - Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (dropout = 0.5) |

For dropout = 0.01, the model showed relatively low training loss, indicating that this was the configuration that was better able to fit the training data. However, the higher validation loss combined with the lowest validation accuracy shows that this was the worst configuration in terms of generalizing unseen data.

With dropout = 0.25, the training loss increased slightly compared to the dropout rate of 0.01, reflecting the regularization effect of dropout. The validation accuracy reached 0.6054, and the test accuracy was 0.6040. These results were the highest among all configurations, indicating that this level of dropout helped prevent overfitting while still allowing the model to learn effectively. The validation loss was also the lowest at 1.0221, demonstrating that the model could better generalize new data compared to the other dropout configurations.

Finally, for dropout = 0.5, the model exhibited the highest training loss 0.9045, suggesting that the model struggled to learn adequately as half the neurons were being dropped out during training. The validation accuracy was 0.6019, and the test accuracy dropped to 0.5933, which was lower than those obtained with the dropout rate of 0.25.

From the analysis of the data we can conclude that a dropout rate of 0.25 strikes the best balance between all configurations as it achieved the highest validation and test accuracies.

These results highlight the importance of selecting an appropriate dropout rate to ensure that the model generalizes well to unseen data while retaining sufficient capacity to learn from the training data.

1. **(c)**

|  |  |
| --- | --- |
| Figure 17 - Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 1024; momentum = 0.0) | Figure 18 - Validation accuracy as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 1024; momentum = 0.0) |
| Figure 19 - Training and Validation Loss as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 1024; momentum = 0.9) | Figure 20 - Validation accuracy as a function of the epoch number for the FeedForward using PyTorch (batch\_size = 1024; momentum = 0.9) |

When momentum was set to 0.0, the model achieved a validation accuracy of 0.4701 and a test accuracy of 0.4887 and a final validation loss of 1.3492 and final train loss of 1.3539. The lack of momentum caused slower convergence, higher loss values and poorer generalization due to the model solely relying on the current gradient for updates.

In contrast, with a momentum set to 0.9, the model achieved a significantly higher validation accuracy of 0.5997 and a test accuracy of 0.6007 with a final validation loss of 1.0469 and a final train loss of 0.9192. The use of momentum allowed the model to leverage past gradients, resulting in smoother and more effective updates. This led to faster convergence, lower loss values and overall better performance. The results demonstrated that adding momentum helps the model overcome the challenges of infrequent weight updates cause by a large batch size, ultimately improving both accuracy and stability.