Homework Assignment 1

Deep Learning, 23/24

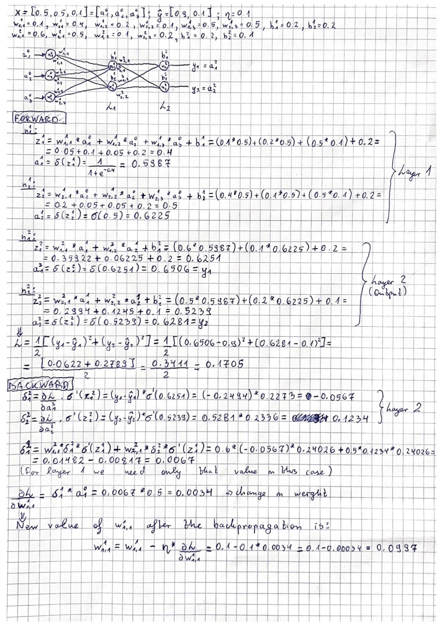
Nikolay Vasilev
(63190338)

1 Introduction

In this assignment, we will focus on training neural networks. In the first part, given a simple neural network, we will simulate forward and backward passes using pen and paper. After we make sure we understand how training the network works, we will implement it in Python and we will test it using different regularization parameters, learning rates, decay rates, optimizers, and other parameters. We aim to achieve classification accuracy of at least 42% with both Adam and SGD optimizers.

2 Backpropagation

In Figure 1 we can see the solution of our backpropagation exercise. As we can see, first we calculate the values of the neurons of each layer using a forward pass. The values at the


$y_1 = a_1^2$
$y_2 = a_2^2$
L_1
L_2

$x = [0.5, 0.5, 0.1] = [a_1^0, a_2^0, a_3^0]$; $\hat{y} = [0.9, 0.1]$; $\eta = 0.1$

$w_{1,1}^1 = 0.1$, $w_{2,1}^1 = 0.4$, $w_{1,2}^1 = 0.2$, $w_{2,2}^1 = 0.1$, $w_{1,3}^1 = 0.5$, $w_{2,3}^1 = 0.5$, $b_1^1 = 0.2$, $b_2^1 = 0.2$

$w_{1,1}^2 = 0.6$, $w_{2,1}^2 = 0.5$, $w_{1,2}^2 = 0.1$, $w_{2,2}^2 = 0.2$, $b_1^2 = 0.2$, $b_2^2 = 0.1$

$\boxed{\text{FORWARD}}$

$\underline{h_1^1}$:

$z_1^1 = w_{1,1}^1 * a_1^0 + w_{1,2}^1 * a_2^0 + w_{1,3}^1 * a_3^0 + b_1^1 = (0.1*0.5) + (0.2*0.5) + (0.5*0.1) + 0.2 =$
$= 0.05 + 0.1 + 0.05 + 0.2 = 0.4$
$a_1^1 = \sigma(z_1^1) = \frac{1}{1+e^{-0.4}} = 0.5987$

$\underline{h_2^1}$:

$z_2^1 = w_{2,1}^1 * a_1^0 + w_{2,2}^1 * a_2^0 + w_{2,3}^1 * a_3^0 + b_2^1 = (0.4*0.5) + (0.1*0.5) + (0.5*0.1) + 0.2 =$
$= 0.2 + 0.05 + 0.05 + 0.2 = 0.5$
$a_2^1 = \sigma(z_2^1) = \sigma(0.5) = 0.6225$

Layer 1

$\underline{h_1^2}$:

$z_1^2 = w_{1,1}^2 * a_1^1 + w_{1,2}^2 * a_2^1 + b_1^2 = (0.6*0.5987) + (0.1*0.6225) + 0.2 =$
$= 0.35922 + 0.06225 + 0.2 = 0.6251$
$a_1^2 = \sigma(z_1^2) = \sigma(0.6251) = 0.6506 = y_1$

$\underline{h_2^2}$:

$z_2^2 = w_{2,1}^2 * a_1^1 + w_{2,2}^2 * a_2^1 + b_2^2 = (0.5*0.5987) + (0.2*0.6225) + 0.1 =$
$= 0.2994 + 0.1245 + 0.1 = 0.5239$
$a_2^2 = \sigma(z_2^2) = \sigma(0.5239) = 0.6281 = y_2$

Layer 2 (Output)

$$\Downarrow$$

$L = \frac{1}{2}[(y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2] = \frac{1}{2}[(0.6506 - 0.9)^2 + (0.6281 - 0.1)^2] =$
$= \frac{[0.0622 + 0.2789]}{2} = \frac{0.3411}{2} = 0.1705$

$\boxed{\text{BACKWARD}}$

$\delta_1^2 = \frac{\partial L}{\partial a_1^2} \cdot \sigma'(z_1^2) = (y_1 - \hat{y}_1) * \sigma'(0.6251) = (-0.2494) * 0.2273 = -0.0567$
$\delta_2^2 = \frac{\partial L}{\partial a_2^2} \cdot \sigma'(z_2^2) = (y_2 - \hat{y}_2) * \sigma'(0.5239) = 0.5281 * 0.2336 = 0.1234$

Layer 2

$\delta_1^1 = w_{1,1}^2 * \delta_1^2 * \sigma'(z_1^1) + w_{2,1}^2 * \delta_2^2 * \sigma'(z_1^1) = 0.6 * (-0.0567) * 0.24026 + 0.5 * 0.1234 * 0.24026 =$
$= 0.01482 - 0.00817 = 0.0067$
(For layer 1 we need only that value in this case)

$\frac{\partial L}{\partial w_{1,1}^1} = \delta_1^1 * a_1^0 = 0.0067 * 0.5 = 0.0034 \Rightarrow$ change in weight

$$\Downarrow$$

New value of $w_{1,1}^1$ after the backpropagation is:

$w_{1,1}^1 = w_{1,1}^1 - \eta * \frac{\partial L}{\partial w_{1,1}^1} = 0.1 - 0.1 * 0.0034 = 0.1 - 0.00034 = 0.0997$

Figure 1: Forward and backward passes solution.

first layer (after weighted sum and sigmoid activation function) are 0.5987 for the first neuron and 0.6225 for the second neuron. Using these values and the corresponding weight and biases we also calculated the values at the second layer, which are also the outputs of the network:

$$y_1 = 0.6506$$

$$y_2 = 0.6281$$

Using these two values we calculate the loss, which is 0.1705.

Now, when we have calculated the loss we will use that in the backward pass to update the weight of the first neuron from the input layer connecting to the first neuron of the first layer ($w_{1,1}^1$). First, we will calculate the deltas (errors) on the second layer, which are the derivatives of the loss function over the corresponding neuron values (outputs). By calculating that we get -0.0567 for the first neuron of the last layer and 0.1234 for the second neuron of the last layer. Since we want to calculate only the new value of $w_{1,1}^1$ we can calculate just the delta of the first neuron from the first layer, which we derive using the deltas we calculated from the last layer and corresponding weights and output value. By doing so we can see that the value here is 0.0067. Using this delta and the input in the first neuron of the input layer, we get the weight change, which is 0.0034. Therefore the updated value at the end of the backward pass, using a learning rate of 0.1, will be:

$$w_{1,1}^1 = 0.0997$$

3 Network Implementation

After successfully implementing the neural network in Python, let's do some tests to try and find the best parameters. We first start by testing the SGD model. In Figure 2 we can see how the regularization parameter influences the SGD model's accuracies. As we expect, by increasing the regularization parameter the distance between the training and the validation accuracy becomes smaller, but so do the accuracies in the cases of too large value. By carefully analyzing the graphs, we can conclude that the value 0.03 seems to be the best for SGD's regularization parameters since it gives us pretty close training and validation accuracies, which are both still pretty high.

Using this value for the regularization rate will allow us to also find the best learning rate for this model. As we see in Figure 3, by increasing the learning rate, the loss increases, and even for some pretty big values we get chaotic

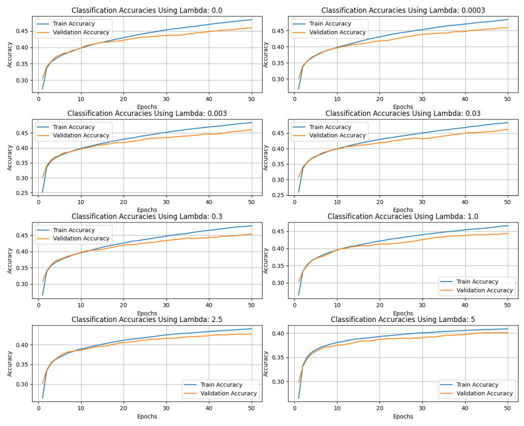


Figure 2: Influence of Regularization Parameter on the Accuracy of SGD.

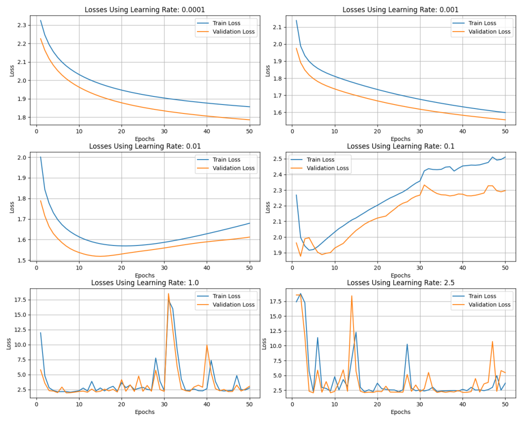


Figure 3: Influence of Learning Rate on the Loss of SGD.

loss. From these graphs, we can conclude that the value 0.001 gives the best performance. Anyway, when we apply learning rate scheduling, we see that this is not the case anymore. By trying different decay rates with different starting learning rates, we found out that the combination of the starting learning rate of 0.01 with scheduling outperforms the case of using the value 0.001 without decaying. Proof of that is Figure 4.

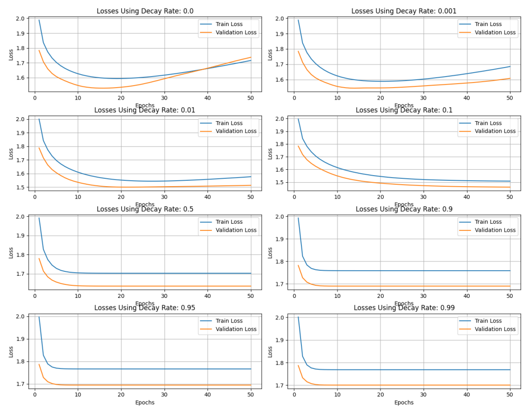


Figure 4: Influence of Decay Rate on the Loss of SGD.

We see that adapting the learning rate fixes the problem of increasing the loss and even helps us achieve the best values for the SGD model using a decay rate of 0.1.

A similar analysis can be done on the Adam model. In Figure 5, we see that we have different and in general lower accuracies than the ones we had using the SGD model.

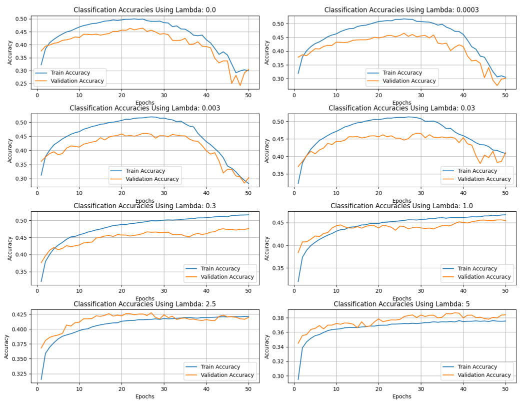


Figure 5: Influence of Regularization Parameter on the Accuracy of Adam.

In this case, the accuracy starts decreasing after the 30th epoch for the lower regularization parameters. We can see that using the regularization parameter 0.3 seems to give us the closest accuracies, while still maintaining high values. Let's see how will the learning rate influence the loss here using this regularization parameter.

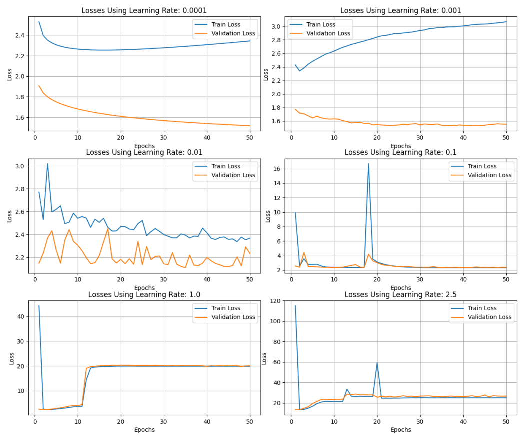


Figure 6: Influence of Learning Rate on the Loss of Adam.

In Figure 6 we see that for the Adam model, we don't have very smooth loss graphs and it looks like for value 0.01 we achieve the closest training and validation losses while maintaining low values. Lower values seem to be achieved for 0.0001, but we can't improve that with the learning rate decaying since the value is already pretty small. So let's try to improve the losses we get with a 0.01 additional learning rate by using scheduling.

Figure 7 proves that by increasing the decay rate the lines become smoother. Anyway, in the end, the lowest values we

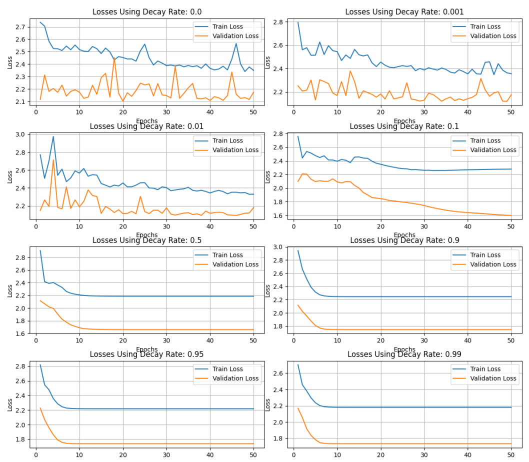


Figure 7: Influence of Decay Rate on the Loss of Adam Using Regularization Parameter 0.3.

get in this graph seem to be similar to the ones we would get using a learning rate of 0.0001. This means that the only thing that can help us improve the Adam model now is to try another regularization parameter. In Figure 5 we saw that using the parameter 0.03 also gives us good accuracies before they start decreasing. Even after that, they seem to be pretty close and still not so low. Let's see if the combination of learning rate 0.01, decaying, and this regularization parameter to try and fix this problem and therefore improve our Adam model.

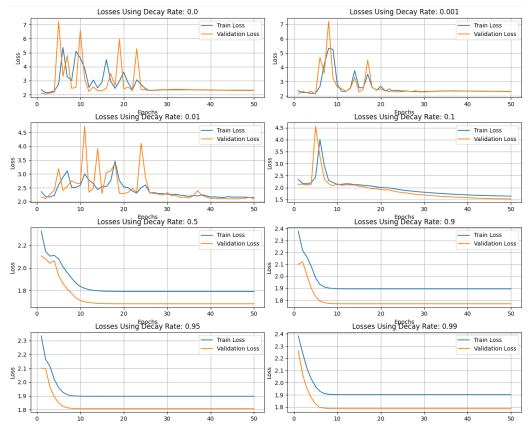


Figure 8: Influence of Decay Rate on the Loss of Adam Using Regularization Parameter 0.03.

In Figure 8 we see that using this regularization instead give us better improvement. In this case from the beginning both losses are close to each other and by increasing the decay rate we decrease their values. In this case we achieve better values and therefore decrease both losses, using the decay rate of 0.1 (same as for the SGD model).

We can further improve the models by changing the number of hidden layers and hidden neurons. After some further experimentation, the overall best parameters for both models proved to be the same and are visible in Table 1. Using these parameters gives us the Losses and Accuracies visible in Figure 9. As we can see in the graphs, even though the

Parameter	Value
Hidden Layers	1
Hidden Neurons	100
Batch size	64
Learning rate	0.01
Regularization parameter	0.03
Decay Rate	0.1

Table 1: Best parameters for both SGD and Adam models.

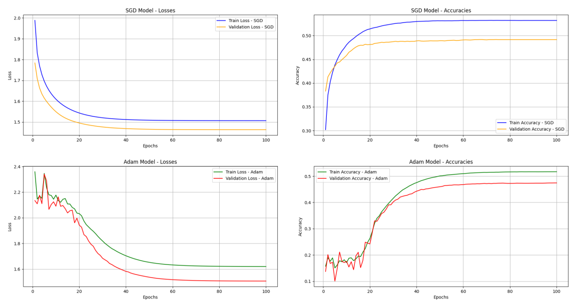


Figure 9: Comparison of both models using the best parameters.

Adam model seems to be more chaotic for the first epochs, both models achieve pretty good values at some point. For the SGD model, we can see that we faster achieve the end value. It takes around 50 epochs to stop decreasing its loss and therefore increasing its accuracy. We can also see that the SGD model can achieve our goal of 42% accuracy for less than 10 epochs, while the Adam model needs around 30. Another important thing visible in the graph is that the Adam model needs around 70 epochs to stop improving (decreasing loss and increasing accuracy). In general, SGD seems to have better performance than the Adam model using these parameters. The achieved results are also visible in Table 2.

Model	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy
SGD	1.51	1.46	53%	49%
Adam	1.62	1.51	52%	48%

Table 2: Final Performance of both models.

Even though the difference in the final accuracies and losses between both models is not so different, the SGD still seems to converge faster. The Adam model seems to need some time to find the right values, but after that, it also converges fast. This means that by implementing batch normalization for example we could take care of the noisy gradients and therefore further improve the models.