ACML-Assignment1

Yumao Gu i6225229
Andre Roelofs i6231145

November 2020

1 Introduction

This document describes the results of our custom backpropogation implementation in Python. Below you will find a description of the standard network to which most of our experiments are compared.

2 Model

The model of this assignment is a network with 3 layers: input - hidden - output. The input and output layers have 8 nodes, while the hidden layer has only 3 nodes (+biases). Both hidden and output layers utilize sigmoid as their activation function.

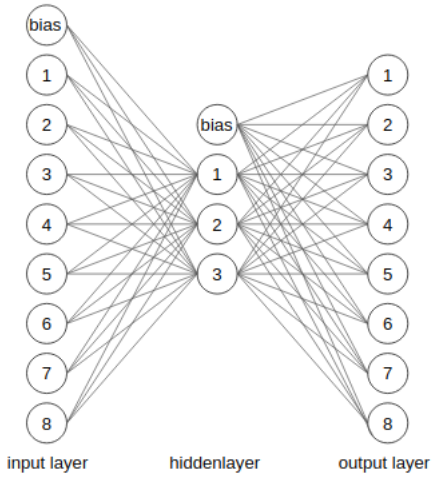

Figure 1: The Model of 9 * 4 * 8

3 Input and Output

The input classes are expressed as vectors containing seven 0 and a single 1. The position of the 1 in the vector is different for each one of the eight classes. The output vector of the network should be exactly the same as its input.

4 Weight Initialization

All weights are initialized to small values near zero using a normal distribution with a mean of 0 and variance of 0.01.

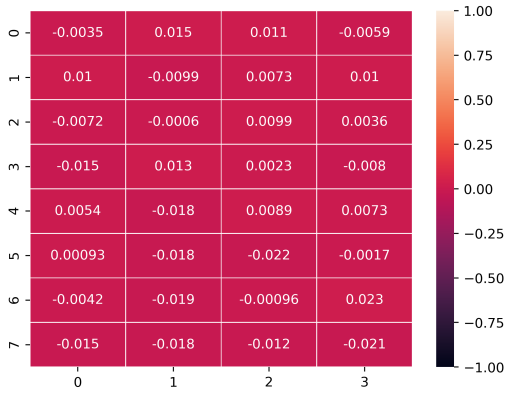

Figure 2: The initial weights from hidden layer to output layer

5 Loss Function

$$
\begin{aligned}
J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \qquad (8\\
&= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2
\end{aligned}
$$

Figure 3: The loss function we use

6 Weight Decay Effect

The model utilizes a fixed learning rate $\alpha = 0.01$ and only brings changes to the weight decay every iteration. Empirically, we had found that the smaller the λ the faster the model loss drops off. The loss does reach a notable plateau after

some iterations. The causes of this effect are discussed below. Nevertheless, a smaller λ also helps the loss function to overcome the plateau faster.

In this particular problem the implemented L2 regularization actually works against the model and prevents quick convergence. We assume that this is a direct result of L2 assuming that the distribution of weights should be Gaussian, which does not need to be the case for such a simple over fitting problem. As can be seen in Figure 4, higher λ values cause the model to get stuck in a loss function plateau for longer.

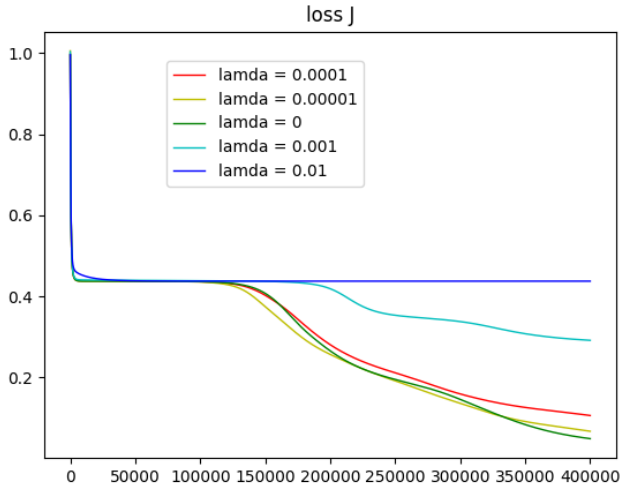

Figure 4: The evolution of the loss with different λ values.

7 Classes: k vs k-1

The network in this problem is heavily overparameterized: the number of parameters to be learned (59) is much higher than the number of the training samples (8). Therefore, we decided to not only employ the weight decay but to also try out the k-1 solution. So, instead of expressing each one of the classes by a vector containing 7 zeros and a single one, we instead use only 6 zeros and 1 one. Equation 1 shows an example of how the class labels changed. The last, 8th class, is expressed only using 7 as shown in Equation 2.

$$< 0,0,0,0,0,0,0,1 > \longrightarrow < 0,0,0,0,0,0,1 > \tag{1}$$

$$< 1,0,0,0,0,0,0,0 > \longrightarrow < 0,0,0,0,0,0,0 > \tag{2}$$

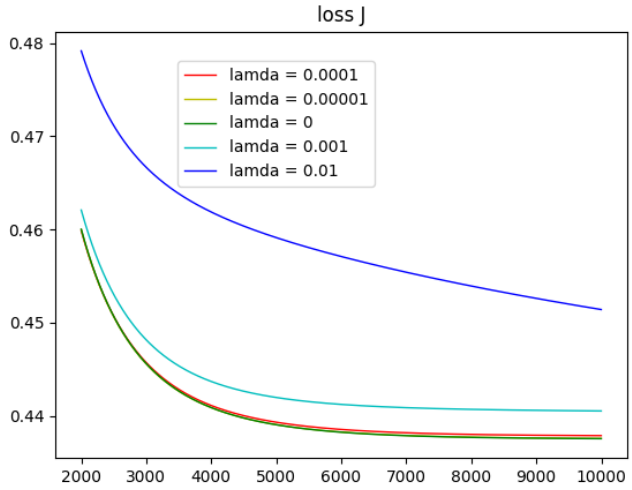

Figure 5: A closer look at early iterations from Figure 4

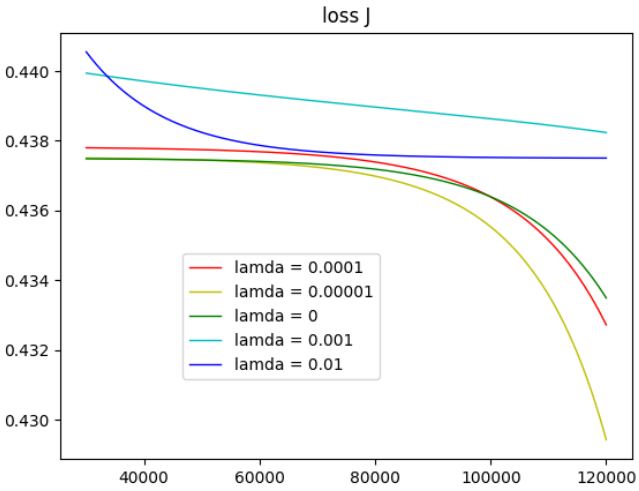

Figure 6: A closer look at later iterations from Figure 4

For the k-1 tests, first, the fastest converging model from Figure 4 was chosen with $\lambda = 0$. The learning rate was kept at the default value $\alpha = 0.01$. Figure 7 shows the amazing difference that the k-1 change creates, provided all other parameters are the same. The k-1 model has passed the plateau far faster than

the k counterpart. K-1 even converged to a lower overall loss after 400.000 iterations, perhaps due to only having to train 52 instead of 59 parameters.

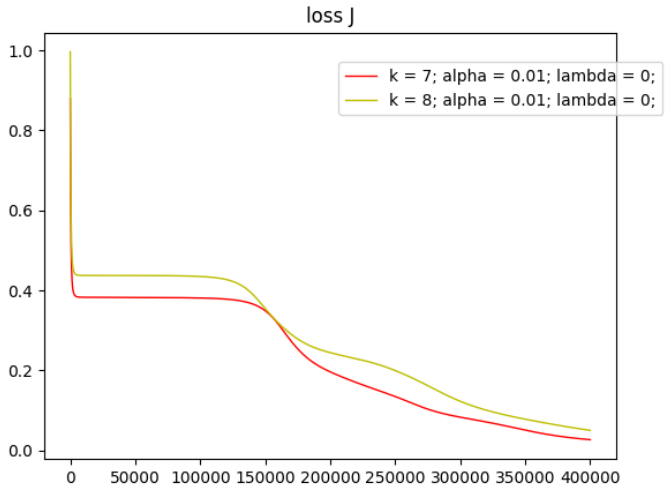

Figure 7: Comparison between the k and the k-1 model

8 Plateau Justification

The plateau as seen in Figure 4 commonly takes place in models with a low-learning rate and is caused by the gradient descent algorithm not being able to leave a sub-optimal local minimum. As expected, increasing the learning rate to sufficiently high values reduces or completely eliminates the existence of the plateau. Figure 8 shows the weights of the hidden layer during the plateau region. As can clearly be seen, bias weights have a significant impact on the outcome of the problem due to having much higher values. A small learning rate prevents gradient descent from significantly affecting the non-bias weights, causing it to be stuck at a wrong point.

9 Learning Rate Effect

During this experiments the weight decay was turned off $\lambda = 0$, only the α value was changed. Generally speaking, at least for this particular problem, the increase in the α value correlates heavily with the reduction or elimination of the plateau region. Learning rates as high as $\alpha = 5$ have been successful at finding an optimal local minima in our tests. The justification for such behaviour is

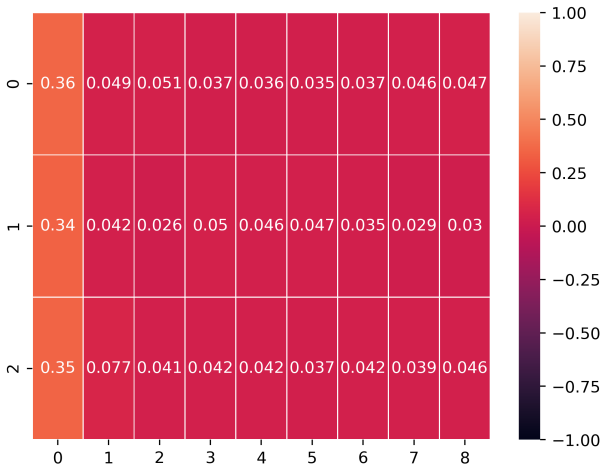

Figure 8: The weight in the plateau area

quite simple - gradient descent is able to use bigger steps to quickly get out of the sub-optimal local minima it had found.

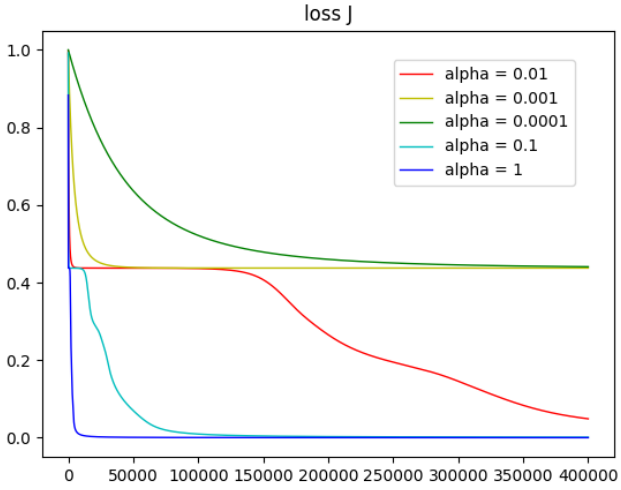

Figure 9: Effect of the learning rate on the convergence of the model

10 Batch Number Effect

Changing the batch size to just a single value has drastic effects on the convergence as seen in Figure 10. Setting batch size to 1 causes each individual class to have a sizeable effect on the weights during that iteration. In our experience, most machine learning models should be trained with the lowest possible batch size unless the number of samples per iteration is somehow involved in the loss function of a model.

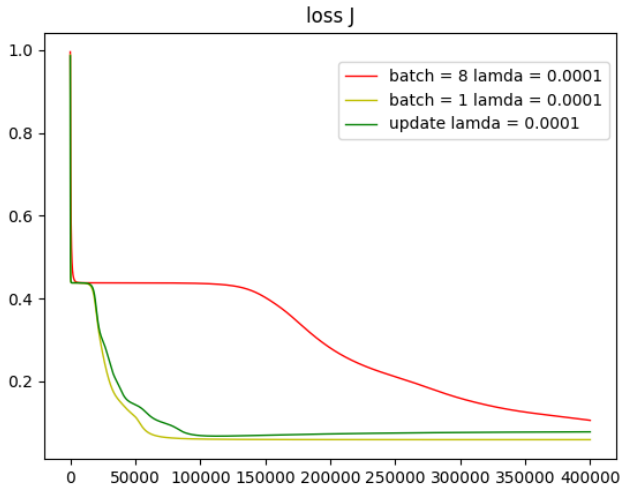

Figure 10: The effect of different batch sizes on the convergence of the model.

11 Weight Interpretation

First we decided to make a few assumptions on how the weights might look after the function has successfully converged. Our first assumption was that the bias weights would all be 0, or close to it. With the rationalization following the fact that bias adds no information to the network. However, as seen in Figures 12, 11 and 13, the bias generally has a very low, negative weight. More importantly, the weights from different training sessions of the same network, in which the weights were randomly initialized, produced completely different sets of weights. This indicates a high number of optimal local minima in the real function.

No clear pattern reoccurring pattern could be found in either hidden or output weights. Even color coding the values has produced no clear results. Therefore, no definite interpretation can be given.

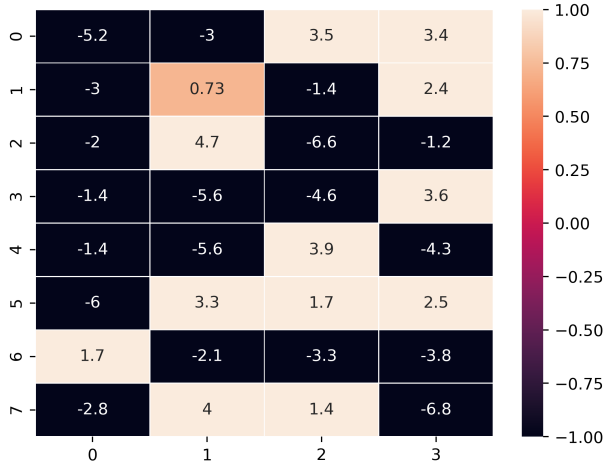

Figure 11: The 32 hidden weights found in a model after 350.000 iterations. $\alpha = 0.01$, $\lambda = 0.1$, batch size $= 8$

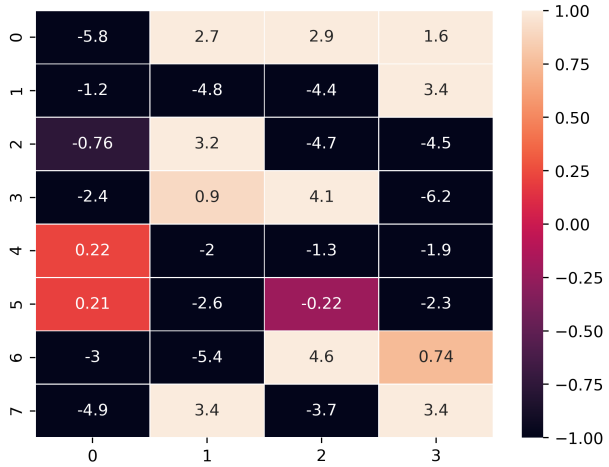

Figure 12: The 32 hidden weights found in a model after 350.000 iterations. $\alpha = 0.01$, $\lambda = 0.1$, batch size $= 8$

References

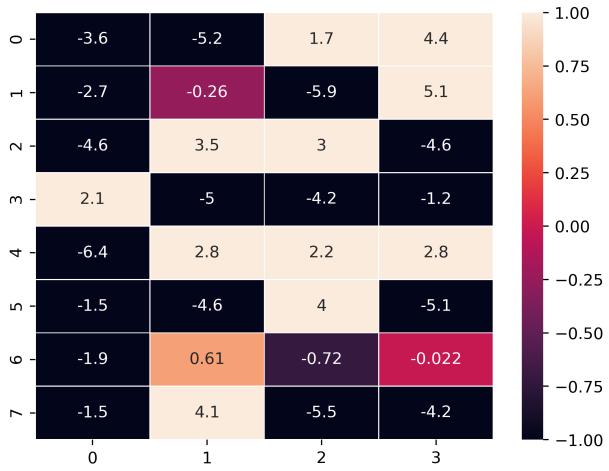

Figure 13: The 32 hidden weights found in a model after 350.000 iterations. $\alpha = 0.01$, $\lambda = 0.1$, batch size $= 8$