Lab report – TNM112

Deep learning for media technology, Lab 1

Gabriel Cederqvist(gabce093)

Lab partners:

Johan Bäcklund (johba008)

Johan Hedin (johed658)

Thursday 23rd November, 2023 (16:52)

Abstract

This is a lab report covering lab 1 in TNM112 - Deep Learning for Media Technology, at Linköping University. It consists of three tasks all focusing on different aspects of the Multi Layer Perceptron. The first task looks at how different parameters affect the neural network. The second task covers the implementation of a function that propagates data through the model and a function to evaluate the model. And the last task examines how weights and biases can be manually entered to create a linear decision border in a minimal neural network.

1 Introduction

This lab is divided into three tasks where different aspects of the Multi Layer Perceptron will be studied. The purpose of this lab is to get a deeper understanding of how the network is able to perform tasks such as classification with such high accuracy. It also aims to show how the different parameters that decides the structure of the network affects the training process and what changes can be made to improve it.

2 Background

2.1 Multi Layer Perceptron (MLP)

The Multi Layer Perceptron networks are made up out of so-called neurons, a function that takes several different inputs with values between 0 and 1 and in turn outputs a value between 0 and 1. For each connection to a neuron there is a weight, and each neuron also contains a bias. The neuron calculates its output by multiplying the weights with the inputs of the neuron and then sums it up together with the bias. This is then put through an activation function which will output a value between 0 and 1. In the network these neurons are the structured as layers where each neuron are outputs to all the neurons on the next layer. By optimizing the weights and biases of the neurons the network can be trained.[1]

2.2 Backpropagation and Stochastic Gradient Descent (SGD)

To the train an MLP, an algorithm called backpropagation is used. It is an algorithm that first initializes the weights and biases to a chosen value and then propagates the input values through the network. Then the error is calculated based on the output and the actual values. The error is then propagated back through the network where we can then calculate the gradients of the loss with respect to the weights and biases. The weights and biases are then updated based on the negative gradient, multiplied with a learning rate which decides the magnitude of the update.
When running backpropagation, you often do not run the entire training dataset through the network at once. Instead, you divide the dataset into batches and run the algorithm on each batch. Meaning that the gradients are calculated based on each batch rather than the entire dataset, this is what's called Stochastic Gradient Descent.[1]

2.3 Task 1

The purpose of the first task is to look at how different types of two-dimensional data affects the output of a Multi Layer Perceptron (MLP) as well as different hyper parameters of the network. For this task a MLP will be created and trained with the help of Keras, a library that supports the creation of neural networks in python. It makes it easy to change both the incoming data to the network and the hyper parameters of the network.

2.3.1 Task 1.1

First a linear two-dimensional dataset was created with 512 different training points and 2 classes. The network was created with no hidden layer meaning that the input was mapped directly to the output with only a single layer. Additionally, only a softmax function was used as the activation in this case since there was only one layer. The network was then trained twice with two different batch sizes, 512 and 16. It was trained for 4 epochs with a learning rate if 1.0 [2].

2.3.2 Task 1.2

For the second part of this task, a polar dataset was created. It consisted of 512 points and contained two classes, same as in task 1.1. However, this time the network was created with 1 hidden layer containing 5 neurons. It was trained for 20 epochs with a learning rate of 1.0 and a batch size of 16. These settings would stay constant and the network was trained for different activation functions, linear, sigmoid and ReLu.[2]

2.3.3 Task 1.3

For this task a polar dataset with 512 training points containing 5 classes with a standard deviation of $\sigma = 0.05$ was used. The network was built up of 10 hidden layers with 50 neurons in each layer. The activation function used was ReLU. The task was to then change the hyper parameters of the network to get the best classification accuracy possible. The parameters that were changed

was the mean and standard deviation of the normal initialization of the weights and biases, the learning rate, the batch size, and the number of epochs. An exponential decay of the learning rate was also tried.[2]

2.3.4 Task 1.4

For this task the built in initialization $glorot\_normal()$ function was used instead of the $RandomNormal()$ to initialize the weights and biases. The optimizer was also changed from $SGD()$ to $Adam()$.[2]

2.4 Task 2

This task consists of implementations for four different activation functions, a $setup\_model()$ function, a $feedforward()$ function and an $evaluate()$ function. The activation function that was implemented was linear, sigmoid, ReLU and softmax. The code for the activation functions can be seen here:

```
def activation(x, activation):
    if activation == 'linear':
        return x
    elif activation == 'ReLU':
        return np.maximum(np.zeros(x.shape), x)
    elif activation=='sigmoid':
        return 1/(1+np.exp(-x))
    elif activation=='softmax':
        s = np.exp(x) / sum(np.exp(x))
        return s
    else:
        raise Exception("Activation function is not valid",
    activation)
```

Next the $setup\_model()$ function was implemented which takes in a weights matrix, a bias matrix and a activation function which is then assigned to the model. The number of hidden layers was calculated by taking the length of the list of weights and subtracting one. The total number of weights and biases are also calculated by looping through both lists and adding up the size of each weight matrix.

Then the $feedforward()$ function was implemented. Where the first step was to define the size of the output matrix. It should contain a probability distribution for each point in the dataset that tells us which class it belongs to. This means the matrix will need to have as many rows as there are datapoints and as many columns as there are classes. Then it was time to implement the feedforward-loop. This was done according to the equation seen in figure 1, where the output of the previous layer is multiplied with the weight matrix of the current layer which is then added to the bias vector. This is then run through the activation function to get the output for the current layer. This will repeat until the last layer is reached, on which the softmax function that was implemented earlier is used. The softmax function turn our output into a probability distribution which can then be looked at to find the most probable class for the datapoint.

The last function to be implemented is the $evaluate()$ function. The function should output the accuracy and loss for both the training dataset and the test dataset. The first step of the implementation was to run the $feedforward()$ function to get the probability distribution for each datapoint from the neural

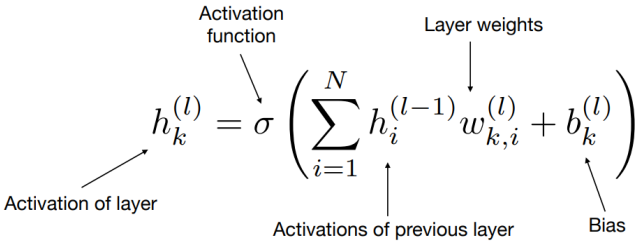

Figure 1: The equation used for the feedforward loop. (*Linköping University, TNM112-Deep Learning for Media Technology, 2023, Lecture 2 — The multi-layer perceptron, slide: 20*)

network for both data sets. Then the specific class was extracted from the prediction. This was done with the $argmax()$ function which takes in the vector with the probability distributions and outputs a vector containing the index to the highest value in each row. Mean squared error, seen in equation 1, was then used to calculate the loss for the datasets and the accuracy was calculated by dividing the amount of correct predictions with the total number of predictions.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} ||f(x_i) - y_i||_2^2 \tag{1}$$

3 Task 3

A minimal MLP was created with the input directly connected to the output. The dataset was a linear dataset and contained 2 classes. Since there was only one layer in the network only the softmax activation function will be used. The task was to then manually assign the weights and biases to draw a decision border at $x_2 = 1 - x_1$. To solve this, the matrix multiplication can first be written out as system of equations as seen in equation 2. Since the decision border lies where $y_1 = y_2$, the system in equation 2 can be extended to equation 3.

$$\begin{aligned} y_1 &= w_{1,1}x_1 + w_{1,2}x_2 + b_1 \\ y_2 &= w_{2,1}x_1 + w_{2,2}x_2 + b_2 \end{aligned} \tag{2}$$

$$(w_{1,1} - w_{2,1})x_1 + (w_{1,2} - w_{2,2})x_2 + (b_1 - b_2) = 0 \tag{3}$$

This means that in order to have a decision border at $x2 = 1 - x$, the weights $w_{1,1}$ and $w_{2,1}$ need to be set in way that the equation $w_{1,1} - w_{2,1} = 1$ and the same for the weights $w_{1,2}$ and $w_{2,2}$. The biases need to be set to that the equation $b_1 - b_2 = -1$. If the label of the area need to be switched for the classes it can be done by changing the sign of the weights and biases.

4 Results

4.1 Task 1

4.1.1 Task 1.1

It can be seen from the results in table (1) that we get a significantly more accurate result when using a smaller batch size during training. The reason for this is that the Stochastic Gradient Descent optimization will iterate over the data 32 times for every epoch since $512/16 = 32$. Because of this frequent adjustment of the weights, a network with higher accuracy is achieved.

Accuracy	Batch size
64.65	512
98.93	16

Table 1: Result of different batch sizes in Task 1.1

4.1.2 Task 1.2

The linear activation function does not give a result with high accuracy due to the fact that there is no linear separation in the data. Since the data is arranged in concentric circles we will never be able to reach a classification network with a linear activation function that averages a higher accuracy than 50% as seen in table (2).

The sigmoid function also does not manage to reach a higher accuracy than 50% as seen in 2. The network gets stuck in a local minima where it classifies all of the datapoint to the same class. This happens because of the vanishing gradient problem of the sigmoid activation function. The gradient becomes to small to be able to improve the network in a significant way.

ReLU achieves the best result out of the activation functions tested as seen in table (2). This is because it does not run into the vanishing gradient problem and is well suited for mapping non-linear data.

Linear	Sigmoid	ReLu
50%	50%	98.73%

Table 2: The accuracy of different activation functions in Task 1.2

4.1.3 Task 1.3

For a long time the network was getting stuck at a accuracy of 20%.What ended up having the biggest effect on the accuracy of the network was mean and standard deviation of the normal initialization of the weights and biases. With a standard deviation of 0.02 the network started to give good accuracies for the problem and an accuracy of 88% was able to be reached.

4.1.4 Task 1.4

With the initialization for the weights and biases changed to $glorot\_normal()$ and the optimizer changed to $Adam()$ a accuracy of 93.63% is achieved on the test data and is able to perform better than the result in task 1.3.

4.2 Task 2

The $activation$, $setup\_model()$, $feedforward()$ and $evaluate()$ functions were all implemented successfully and could accurately display the loss and accuracy of the test and training datasets.

4.3 Task 3

By rewriting the formula for the multiplication of the weights and addition of the biases, a way of determining the correct weights and biases to get a decision border at $x_2 = 1 - x_1$ was established. It was seen that the there is an endless amount of combinations of weights and biases that can create the decision border.

5 Conclusion

This lab looked at the different aspects of the Multi Layer Perceptron. First the Keras library was used to examine how different parameters affected the training of a model. Secondly a custom feedforward and evaluation function was implemented. By doing this you got a close look into how the data is propagated through the model. And lastly, to understand how decision borders are drawn in a classifier a way of manually determining the weights and biases of a minimal network was examined.

Use of generative AI

I have used generative AI to help me better understand some concepts, mainly backpropagation.

References

[1] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[2] TNM112. Lab 1 – the multilayer perceptron, 2023. Lab description – Deep learning for media technology.