Fundamentals of Artificial Intelligence

Assignment 3 machine learning ANN

Digit recognition

5DV124 HT20

William Danielsson || CS:id18wdn || CAS: Wida0006

E-mail: william99danielsson@hotmail.com

Course responsible

Ola Ringdahl: ringdahl@cs.umu.se

Other teachers

Lois Vanhee: loisv@cs.umu.se

Adam Dahlgren: dali@cs.umu.se

Timotheus Kampik: tkampik@cs.umu.se

Andrea Aler Tubella: aler@cs.umu.se

Lennart Steinvall: svartvit@cs.umu.se

Carl-Anton Anserud: anserud@cs.umu.se

Disclaimer: *My group partner quit the course after one week so that is why I am doing the assignments solo*

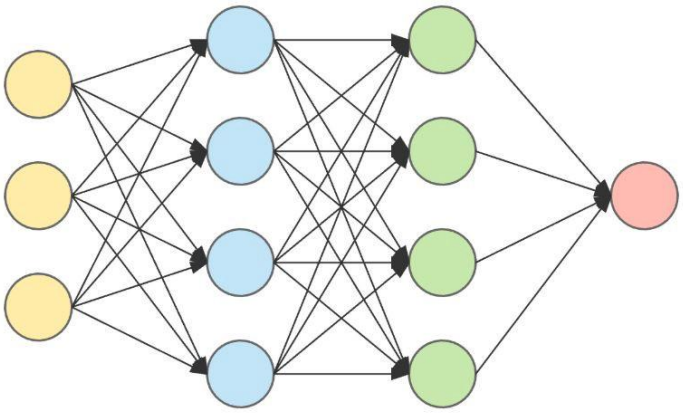
Introduction and Overview

The goal with this exercise was to make a neural network from scratch that could learn to correctly identify handwritten numbers. Specifically the numbers “4” “7” “8” & “9”. By using 4 different networks one for each number, each tasked with giving a value between -1 and 1 of how sure it is that a specific image contains the number that it is trained to look for.

By using a sample of 1000 images the data should be divided into a test-set and a training-set that the AI can train and then test on. When the result of the testing is good enough the program shall stop training and then go on to test the validation images to determine its overall performance.

Implementation

Neural Network explained



A neural network is a network/web of nodes called “neurons” that are all connected with each other. That depending on the strength of the connections to the previous layer determines the output of the next layer. Made to simulate how the brain works. In our brain we have neurons that can “fire” or “not fire” if one specific neuron fires then that can cause other neurons that

it is connected with to fire as well, and when you have billions of neurons all connected in a specific pattern you can start to get some intelligent behaviour.

A neural network is just that, a simulation of the human brain (in a very basic form of course). The principal is that the input layer (the yellow nodes in the picture above) gets some data and then depending on its weight either shrinks or amplifies the value of that data, then all the data from each neuron gets sent to every neuron in the next layer that then gets a new value depending on the combined value of the neurons in the previous layer. This process then repeats until it reaches the final layer (output) layer.

You can see how this process becomes a chain of events, almost a butterfly effect where the states of the next layer depends on the state of the previous layers. Now how do we get this to network to behave intelligently?

I said before that each neuron has a specific weight associated with them, this weight determines whether the data that that specific neuron holds become valuable or not. If we somehow can change the values of the weights in each layer so that they together can recognize patterns such as curves, edges or circles then we can use the different layers to recognize more complex structures. For example if the blue nodes could keep track of edges and then the green nodes can check if the edges form a pattern such as curves or circles then the output layer can piece these patterns together and see what number it is that is represented. A 9 for example is just a line with a circle on top of it, if the blue layer can see the edges and then the green layer can piece those edges into circles and lines then the output layer can see that it is a 9 that is drawn on the image.

Now imagine doing this for hand, choosing a value for every weight so that the correct neurons gets a high value that then can get sent down to the next layers to correctly guess that it is a 9. It would be almost impossible, instead we use something called learning. Where we let the AI guess what numbers it thinks it is and then depending on the error calculate a new value for each weight so that it overtime becomes the right weights to correctly recognize those numbers. That is the task at hand and that I have implemented.

It is stated in the specifications that we are not allowed to use “hidden layers”, hidden layers are the layers in between the input and output layers (the blue and green nodes in the picture). This means that my implementation will only have 2 layers, one input and one output.

Neuron

The first thing was to create a neuron, a neuron is supposed to hold a value, a weight and also its activation function value that is later used in the learning process. Most neural networks also use a bias for each neuron that is supposed to keep a neuron from “firing” if the value isn't bigger than the bias. But I figured that it wasn’t needed in my implementation so I later removed that attribute from the datatype.

Network

A network is another class that I created that creates a bunch of neurons and connects them with each other to form a network. Since my implementation is only allowed to have 2 layers I made the first layer as big as each pixel in each image. Every image is 784 pixels big so I created a list of 784 neurons each that gets the pixel value of its specific pixel. Then I created the output layer, which in this case is only one neuron big since we used 4 networks each to detect one specific number.

I then take all the values multiplied by its weight to get the sum of the whole first layer and then squish that number so that it fits between the range of -1 to 1with a tanH function to get a new value to the next layer (output layer). This output value is how much the network thinks that a specific image is it's number. So it if is very sure then the output should be very close to 1, and if it thinks that it isn't it's number then it should give a output closer to -1

Learning & Decision Making

I have a class called Intelligence where most of the math is done and where the networks are created and where they work together to train and to decide what number is drawn on each image. What's happening is that I created a training-set from the image sample that consists of 800 images. I then present each image once to each network and they all respond by giving out its output number from -1 to 1 of how sure they are that the image contains their respective number. Then depending on how wrong they are I calculate the error with backpropagation to change each neuron's weight in each layer to better be available to recognize patterns on that network's specific number that it looks for.

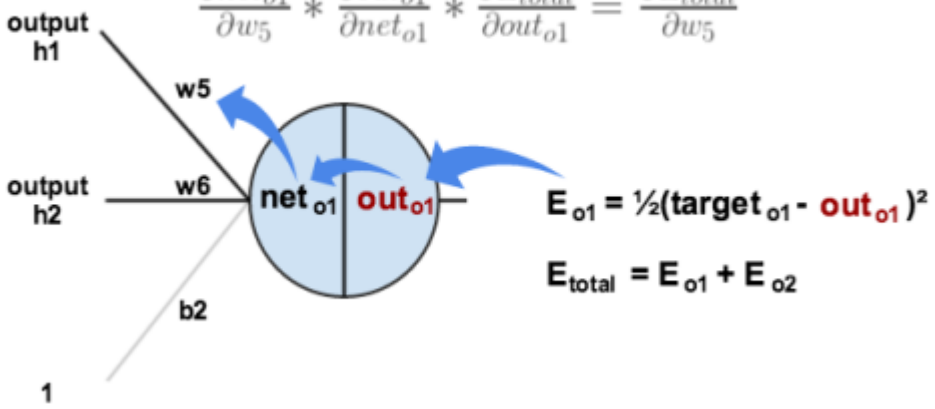
After the networks have trained and corrected their weights on all 800 images I then give them the remaining 200 images from the sample to test on. These images the network hasn't seen yet in its training so they will be brand new for this cycle. It then gives each image in the test-set to each network and they all give their output value. I then take the network with the highest output value and use that as the answer. So if the network that is specialized in finding the number 4 has the highest value then the AI says that 4 is the correct answer. It then checks with the label for the image and sees if it was correct. If it was correct it gets a point, then when it has guessed on each image in the test-set I divide the correct guesses (points) with the 200 images to calculate the accuracy. If the accuracy isn't high enough then repeat the whole training and testing cycle. Until the testing accuracy is high enough that I consider the AI to be fully learned.

Now to make it stay more accurate and to make sure that it doesn't just memorize the numbers I switch the images in the test-test and training-set each time. So of all the 1000 images there are always 800 random training-images and random 200 test-images so an image can be in the training-set in one cycle and then be in the test-set the next.

Backpropagation

So far I have been very abstract in how the AI learns. All I have said is that the weights change after each training image to better be available to recognize patterns. But how does this process go? How can the AI know exactly what weights should be changed and how much they should be changed? Well to be honest I don't really know that either, all I know is that it can know this by the wonder of backpropagation which is a complicated algorithm that can take a function and figure out how a single variable in that algorithm changes the whole algorithm and how much to change that variable so that the whole function becomes more efficient.

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

This right here is the equation that calculates the value of how much one specific weight should change with. It basically boils down to multi variable calculus that I haven't studied yet so I am in no way guilty of not understanding the math down to its root. But with this I can calculate the error for each weight and then add it to its current weight.

In my code I have implemented it so that if the network guesses on an image that have the same number that it is supposed to be good at recognizing then we say that its optimal output value should be 1 since 1 means that it is really sure that it is it's number, then we take the actual output that we got and subtract it from the optimal value. That then is the total error of the network, now to know what to change to get that error closer to 0 we use derivatives of the 3 parameters in the equation above to get the totalError divided by a specific weight. That is then our deltaWeight that we then can subtract from our original weight to get a better weight that can be seen in the equation below.

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

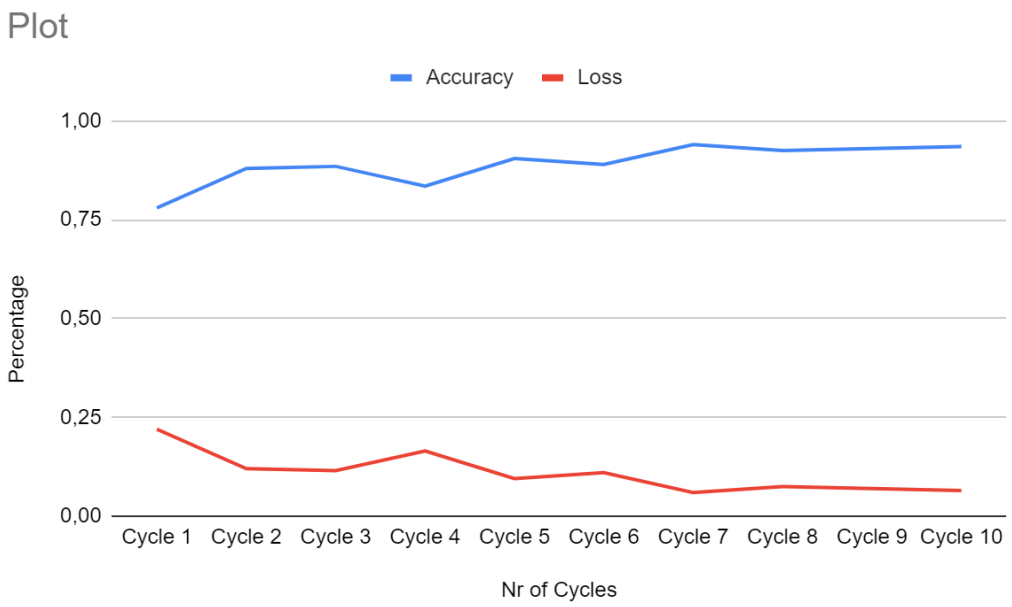
η is the learning rate at which it learns, this is to make sure that it doesn't overshoot or do too big jumps in its change of the weights.

Results

```
Accuracy after training cycle 1: 0.835
Accuracy after training cycle 2: 0.85
Accuracy after training cycle 3: 0.915
Accuracy after training cycle 4: 0.845
Accuracy after training cycle 5: 0.85
Accuracy after training cycle 6: 0.945
Accuracy after training cycle 7: 0.91
Accuracy after training cycle 8: 0.93
Accuracy after training cycle 9: 0.895
Accuracy after training cycle 10: 0.94
Ai deemed good enough, training stopped

Process finished with exit code 0
```

Testing 1: trains until testing accuracy surpasses 90% 5 times



Testing 1: Plot over the accuracy and loss, as you can see the graph starts to slow down at around 90%

This is how many cycles it takes until I deem the AI to be good enough. A cycle is one loop through the training images and one loop through the test images. This takes about 0.3 seconds to compute, I then decided that when it reaches 90% or higher accuracy 5 times on the test-set then it is done. The reason I choose that number is because it still gets about 90% correct answers on the validation images and it goes very fast to get this good. If I decide to keep going I can get it to average about 97-99% accuracy on the test-images as can be seen in the picture below.

```
Accuracy after training cycle 150: 0.98
Accuracy after training cycle 151: 0.975
Accuracy after training cycle 152: 0.98
Accuracy after training cycle 153: 0.985
Accuracy after training cycle 154: 0.97
Accuracy after training cycle 155: 0.985
Accuracy after training cycle 156: 0.975
Accuracy after training cycle 157: 0.96
Accuracy after training cycle 158: 0.97
Accuracy after training cycle 159: 0.97
Accuracy after training cycle 160: 0.98
Accuracy after training cycle 161: 0.97
Accuracy after training cycle 162: 0.985
Accuracy after training cycle 163: 0.975
Accuracy after training cycle 164: 0.985
Accuracy after training cycle 165: 0.965
Accuracy after training cycle 166: 0.995
Ai deemed good enough, training stopped
```

Testing 2: trains until testing accuracy surpasses 99% 5 times

Here I changed so that the AI is done when it gets 99 or more percent accuracy 5 times. This seems to be the upper limit though, it never quite makes it to 100%. This test run was basically just to find out how good it can be, in the final program I use the settings that can be seen in ***'testing 1'***.

Problems and changes

During my battle with this enormous task I stumbled upon quite a few problems. The biggest of which was that I had missed that it specified in the assignment slides that we were not allowed to use hidden layers. I had already implemented those and had to remove everything and scale down my code alot. Also I realized very late that there were supposed to be 4 networks. I imagined 1 network with 4 output neurons that each represented a number. So that I had to change as well. Probably the biggest task was the actual training, I only looked at the slides at first where it briefly explained the learning algorithm and I tried to implement that. But then realized that that example was a very abstract and not complete example. So I had to read up on backpropagation on my own and learn how to change a specific weight. This took me a few days but as soon as I found a source that explained it in a very in-depth way and when I finally found an actual equation that I could solve I managed to implement it right on my first try.

After I got my code to work I tried experimenting with different values to see how I could maximize the speed and accuracy of the AI. I realized that having a learnigrate $\eta = 0.01$ was the most optimal value. If I got higher it would get smart very quick but then stop learning at around 80% and if I took a too small value then it would get smarter but it would take much longer to get up in the high 90's percentile. I started with having all weights be a random number from -1 to 1 but then saw that sometimes that would make some of the more important weights get really bad values and it would take a long time until those weights got changed to their optimal value. So instead I decided that all weights start at 0.

I also played around with the distribution of the training/test size. I saw that a bigger training size resulted in faster learning, the test size is only supposed to check how good the AI is anyway. So I think that any test size over 100 is good enough to get a good overview of the performance. I ended up at 80% of the data as training-set, that leaves 200 images in the test-set which still should be high enough to not cause underfitting.

Summary

I personally think this has been the most interesting exercises I have been tasked with yet. I loved the fact that no code was given so I had to implement everything from scratch and think of all the problems and issues that could occur and what classes I needed and so on. The only thing I felt that was a bit frustrating was that there was no explanation of how the actual learning algorithm should work so I had to research that myself and I didn't expect it to be as complicated as it was.
But all in all this assignment got me really interested in neural networks and machine learning and I even feel a bit tempted to try to make an actual real AI with only 1 network, hidden layers and 10 output layers to recognize all 10 numbers from 0-9 as that is the real non-scaled AI problem that is so popular among the neural network community. I almost feel like doing it the way we did with only 4 numbers and 4 networks was almost more complicated and harder to understand.
I will take with me a lot of knowledge from this assignment that I hopefully can apply later on the exam as well!