# Introduction:

An example of a simple Gradient descent AI model:

Input layer

Hidden layer 1

Hidden layer 2

Output  
layer

In this model, there are 2 input neurons in the input layer, 4 neurons in the hidden layers, and 2 output neurons in the output layer.

Data will be passed from the input layer, to the hidden layer 1, then to the hidden later 2, then to the output layer.

Simple example:

Height

Weight

40lbs

2ft 2in

Species 1

Species 2

One example of this could be trying to predict whether an animal is species 1 or species 2 by using only height and weight inputs.

# Forward pass

Normalizing the height and weight, we can see that the values are as follows: 0.8, 0.5

Height

Weight

0.5

0.8

Species 1

Species 2

The next important step is to randomize the weights and bias of the neurons of the artificial network. The below diagram depicts the top half of the graph, showing the weights and biases:

0.8

W1

W2

W3

W4

W5

W6

W8

W7

B1

B2

B3

B4

B5

Do not mistake the biases as the neuron’s data, these are different. Randomizing these values gives you:

0.8

0.2

0.4

0.5

0.3

0.1

0.2

0.4

0.7

0.8

0.2

0.2

0.1

0.1

0.4

0.1

0.5

0.4

0.1

0.1

0.9

0.1

0.6

0.3

0.4

0.6

0.3

0.4

…

Now we are ready to do our first forward pass. During a forward pass, the neuron data is passed from the input neuron(s) to the output neuron(s), passing through many different mathematical manipulations.

The first stage of this forward pass is to calculate the value of the 2nd layer, given the value of the normalized values from the input layer (height of 0.8, as seen above, and weight of 0.5, as seen previously)

The 2nd layer neuron values are equal to the sum of the previous nodes, multiplied by the weight factor, adding a bias. This value is then passed through an activation function. For simplicity, we will be using ReLU for the activation function, which is shown below:

Where represents the number of neurons in net layer , and represents the weights of net layer

Visually, an example of neuron is shown below in the second layer:

0.8

0.2

0.1

0.1

0.5

…

0.31

i

j

For the first node in the 2nd layer above, the mathematical calculation for this is shown below:

After computing the first neuron of layer 2, we then go on to calculate all others. Then, similar to what is being shown, the neurons in layer 3 and layer 4 are calculated.

The final calculation for the forward pass is using the neuron values in the layer 4 to determine the output. There are 2 options as the theoretical answer: Species 1 and Species 2. The species that the AI determines as the right answer will be the highest value. So at the end, if the neuron values are 0.3 and 0.9, then the species will be determined to be species 2. This is mostly just simple math that occurs, and in the backward pass this math will be “edited” to make sure that the weights and biases are corrected to be in favor of selecting the correct answers

# Backward pass

The backwards pass is a bit more complicated. Essentially, we are going to take the actual resulting value, and then calculate how much the weights at each stage have an effect on the output, and then try to correct that value. For simplicity, we will continue to use the model above but pretend that we are at the end of the forward pass:

0.5

0.3

0.4

0.7

0.8

0.1

0.9

0.1

0.6

0.3

0.4

0.6

0.3

0.4

…

First it will be important to note how we got these numbers. Initially, the weights and biases are randomly generated, so that’s where they come from, which are at the bottom of each node (bias) and along the lines (weights). The numbers in the center of the circles are the node values, which are generated during the forward pass. For simplicity we will just be solving the top, last, node in backward pass. To be specific, the value of the top last node is solved using the forward pass talked about previously, which solves as follows:

Now, after a forward pass, we calculate the reverse/backward pass by undoing the forward pass. We compare the desired output to the actual output in the last layer, and get the derivative (difference) between them. If the output is 1.548 in the first node, and the real value of the data is 1.6, where Z represents the node values, and Y represents the actual data:

Then, we reverse the weights by finding where these represent the weight change and bias given the ideal data:

This calculates the weights and bias effect that it has on the actual output of the data. Now that we have the values of weights and bias that have had an effect on the output and how much they change, we can correct this value using a learning rate (alpha value), which determines how much to change these by. We then find the new weights and bias by doing the following:

The math for intermediate nodes is a bit different, so see the source code in the DenseLayer.py file for the exact math under the function, backward\_pass.

## Additional characteristics

One important thing to note is that the forward passes are done on a subset of the input data, and then a backward pass is performed all at once. This allows the network to calculate the changes in weights/bias needed for a subset of the data, rather than for each data point. This is called the batch size. If there is a batch size of 100, then 100 data points will go into the forward pass then a backward pass occurs to correct values. Another thing is the epoch amount. Epoch count is how many times the entire input set passes through the AI. So if the entire input size is 1000, and you set batch size as 500, and 2 epochs, then a total of 4 batches will pass through the network, (total of 2000 inputs). The learning rate (alpha value) determines how fast the network learns, but does not necessarily mean you should set it as high as possible. As noted in the other documentation, setting the learning rate to a lower value such as 0.01 or 0.05 is desirable depending on the batch size.

## Conclusion

There are many factors that can affect a gradient descent AI model, some of which we spoke about here. These include the process of forward pass and backward pass as well as the other characteristics of the AI such as learning rate (alpha value), batch size, epochs, etc. The above processes are how I implemented TinyAI so I just wanted to give some explanation to the process for if anyone is looking to learn how, and also follow along with my code.