Layout

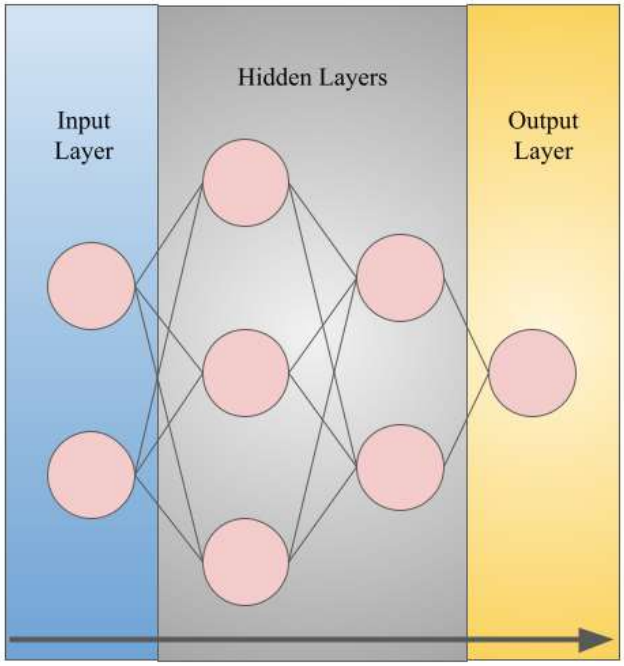
Note: Skip to "Summary (for implementation)" if you already understand neural networks on a fundamental level and are looking for a clean overview on how to implement one in a program.

Intro

Can a machine learn? Well, the obvious answer is yes. But can it learn anything outside the dataset provided by the programmer? Normally, no, since many decision-making programs require all possible (or at least probable) cases to be defined before determining which specific case it finds itself in, before deciding the best plan of action. However, with the help of neural networks, knowing all cases is no longer a requirement. The program only needs a random sample of cases to work off. Neural Networks are a great introduction to the world of machine learning. With accelerating technological advancements and increasing ambitions for projects that require machine learning (such as self-driving cars), neural networks will become a huge part of our lives.

Structure

Whenever you think of a neural network, this is the general structure that should come across your mind:



Each circle is a neuron, which holds a value. The inputs are put into the input layer, propagated through the hidden layers, and ultimately result in the outputs, which are taken from the output layer once the network has finished propagating.

Each line is a weight that connects two neurons in consecutive layers. Every neuron (except the input neurons) has a bias. Each and every neuron (except the input neurons) uses the weights that connect it to the neurons in the previous layer, the values of the neurons in the previous layer, and its bias in order to calculate a value that will be passed onto the next layer (input neurons just pass their value onto the next layer). This will be explained in greater detail in the Math section.

Note: There isn't a limit to the number of neurons allowed in a layer. The number of neurons in the input layer should equal the number of inputs in a training sample. The number of neurons in the output layer should equal the number of targets in a training sample.

Theory

Neural Networks (NN) are a universal function approximator. In other words, if there's a function we don't know, but we know which inputs correspond to which outputs (known outputs are called targets), we train the NN with those, and the NN approximates a function for them. Then you can feed basically any other input and it will spit out an output according to the function it approximated. A set of inputs and targets is called a training sample.

Neural Networks are able to approximate a function through the means of gradient descent. When a new network is created, the weights and biases aren't initially optimized, so the network will output an incorrect result. That incorrect result is compared to the target to form the cost (error) function. The goal of the network is to minimize this function based on its gradient with respect to the weights and biases of the network. You could think of weights and biases as variables in the cost function.

Math

Before we begin, we must: standardize the inputs to values between 0 and 1, and choose an activation function σ.

A good way to standardize each input is

$$input = \frac{input}{maxInput}$$

Some examples of good activation functions are tanh and ReLu, but the one you use may depend on what you're training your network on.

Now we can get behind the math. Here are some general rules to assume:

Superscripts will ***always*** be used to indicate the index of the layer.

Subscripts will be used to indicate the index of the neuron ***except*** for weights, in which case subscripts will be used to indicate the index of the weight in the weight array w. When a subscript is written as a pair, it will be used to indicate both the index of the neuron ***and*** the index of the weight.

a_i^L is read as "the activation value of neuron i of layer L"
w_i is read as "the weight at index i of array w of the current neuron"
$w_{(nIdx,\, i)}^L$ is read as "the weight at index i of array w of neuron $nIdx$ of layer L"

If there is not a superscript or subscript present, assume that we are talking about the current neuron in the current layer

Forward Propagation

We iterate through each layer from left to right

Key

$L \rightarrow$ current layer

$a \rightarrow$ activation value

$\sigma \rightarrow$ activation function

$w \rightarrow$ weights array (of current neuron)

$b \rightarrow$ bias (of current neuron)

$n \rightarrow$ neuron value (of current neuron)

$nIdx \rightarrow$ index of current neuron in current layer

For the first layer, iterate through each neuron and set its activation value to the inputs

$$a_{nIdx}^{L} = inputs_{nIdx}$$

Then, for the following layers, iterate through each neuron and calculate its neuron value and activation value

$$n = \left(\sum_{i=0}^{\#weights} w_i * a_i^{L-1} \right) + b$$

$$a = \sigma(n)$$

Backward Propagation

Key

$C \rightarrow$ cost function

$t \rightarrow$ targets

$L \rightarrow$ current layer

$a \rightarrow$ activation value

$\sigma \rightarrow$ activation function

$w \rightarrow$ weights array (of current neuron)

$b \rightarrow$ bias (of current neuron)

$n \rightarrow$ neuron value (of current neuron)

$nIdx \rightarrow$ index of current neuron in current layer

First, we forward propagate a training sample consisting of inputs and targets. Then we iterate through each layer from right to left, stopping at the first layer (since it doesn't have weights or biases).

<u>For the rightmost (output) layer</u>
Iterate through each neuron (in other words, "for each neuron in this layer")

We can calculate each output neuron's cost (error) using this formula

$$C_{nIdx} = \left(a^{L}_{nIdx} - t_{nIdx}\right)^2$$

where i is the index of the output neuron in layer L

Note: Taking the average of the costs of all the output neurons gives us the mean squared error (MSE), which allows us to evaluate the efficiency and progress of the network.

Keep in mind the formulas we used for forward propagation

$$n = \left(\sum_{i=0}^{\#weights} w_i * a_i^{L-1}\right) + b$$

$$a = \sigma(n)$$

Calculate the neuron's weight and bias gradients using the calculus chain rule

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n}\frac{\delta n}{\delta w_i}$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n}\frac{\delta n}{\delta b}$$

We calculate the individual partial derivatives

$$\frac{\delta C}{\delta a} = 2(a - t)$$ ← Note: This is the output neuron's activation gradient

$$\frac{\delta a}{\delta n} = \sigma'(n)$$

$$\frac{\delta n}{\delta w_i} = a_i^{L-1}$$

$$\frac{\delta n}{\delta b} = 1$$

Substituting the values of the partials into the formula to calculate the gradients

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n}\frac{\delta n}{\delta w_i} = 2(a-t) * \sigma'(n) * a_i^{L-1}$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n}\frac{\delta n}{\delta b} = 2(a-t) * \sigma'(n)$$

Save the neuron gradient, as we will need it for gradient calculation in the hidden layers

$$\frac{\delta C}{\delta n} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n} = 2(a-t) * \sigma'(n)$$

In terms of the neuron gradient

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta n}\frac{\delta n}{\delta w_i} = \frac{\delta C}{\delta n} * a_i^{L-1}$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta n}\frac{\delta n}{\delta b} = \frac{\delta C}{\delta n}$$

<u>For hidden layers</u>

Iterate through each neuron (in other words, "for each neuron in this layer")

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta a^{L+1}}\frac{\delta a^{L+1}}{\delta n^{L+1}}\frac{\delta n^{L+1}}{\delta a}\frac{\delta a}{\delta n}\frac{\delta n}{\delta w_i}$$

We're going to break the calculation into two parts: Calculating the activation gradient, then using that to calculate the weight gradients. First, calculate the activation gradient for this neuron.

$$\frac{\delta C}{\delta a} = \frac{\delta C}{\delta a^{L+1}}\frac{\delta a^{L+1}}{\delta n^{L+1}}\frac{\delta n^{L+1}}{\delta a}$$

Note how the activation value directly impacts each neuron in layer $L+1$ (which we'll call front neurons). In order to take into account all changes, we need to consider the gradient for all front neurons.

Note: $\#L+1$ is read as "the amount of neurons in layer L+1"

$$\frac{\delta C}{\delta a} = \frac{\delta C}{\delta a_0^{L+1}} \frac{\delta a_0^{L+1}}{\delta n_0^{L+1}} \frac{\delta n_0^{L+1}}{\delta a} + \frac{\delta C}{\delta a_1^{L+1}} \frac{\delta a_1^{L+1}}{\delta n_1^{L+1}} \frac{\delta n_1^{L+1}}{\delta a} + \ ... \ + \frac{\delta C}{\delta a_{\#L+1}^{L+1}} \frac{\delta a_{\#L+1}^{L+1}}{\delta n_{\#L+1}^{L+1}} \frac{\delta n_{\#L+1}^{L+1}}{\delta a}$$

$$\frac{\delta C}{\delta a} = \frac{\delta C}{\delta n_0^{L+1}} \frac{\delta n_0^{L+1}}{\delta a} + \frac{\delta C}{\delta n_1^{L+1}} \frac{\delta n_1^{L+1}}{\delta a} + \ ... \ + \frac{\delta C}{\delta n_{\#L+1}^{L+1}} \frac{\delta n_{\#L+1}^{L+1}}{\delta a}$$

$$\frac{\delta C}{\delta a} = \sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} \frac{\delta n_j^{L+1}}{\delta a}$$

We know that

$$n = \left(\sum_{i=0}^{\#weights} w_i * a_i^{L-1} \right) + b$$

holds true for all neurons, so we can assume that for all front neurons

$$n_j^{L+1} = \left(\sum_{i=0}^{\#weights\ of\ front\ neuron} w_{(j,\,i)}^{L+1} * a_i^L \right) + b_j^{L+1}$$

$$\frac{\delta n_j^{L+1}}{\delta a} = w_{(j,\,nIdx)}^{L+1}$$

where j is the index of the front neuron

Therefore:

$$\frac{\delta C}{\delta a} = \sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} * w_{(j,\,nIdx)}^{L+1}$$

the activation gradient is equal to the sum of: the neuron gradient of the front neuron multiplied by the weight at index $nIdx$ of the weight array w of the front neuron (the weight connecting that neuron and the current neuron), for all front neurons

Now that we have the activation gradient, we can calculate the weight and bias gradients for the current neuron.

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta a^{L+1}} \frac{\delta a^{L+1}}{\delta n^{L+1}} \frac{\delta n^{L+1}}{\delta a} \frac{\delta a}{\delta n} \frac{\delta n}{\delta w_i}$$

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta a} \frac{\delta a}{\delta n} \frac{\delta n}{\delta w_i}$$

Keep in mind the formulas we used for forward propagation and backpropagation at the rightmost layer

$$n = \left(\sum_{i=0}^{\#weights} w_i * a_i^{L-1} \right) + b$$

$$a = \sigma(n)$$

$$\frac{\delta a}{\delta n} = \sigma'(n)$$

$$\frac{\delta n}{\delta w_i} = a_i^{L-1}$$

Therefore

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta a} \frac{\delta a}{\delta n} \frac{\delta n}{\delta w_i} = \left(\sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} * w_{(j,\, nIdx)}^{L+1} \right) * \sigma'(n) * a_i^{L-1}$$

Do the same for the bias

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta a^{L+1}} \frac{\delta a^{L+1}}{\delta n^{L+1}} \frac{\delta n^{L+1}}{\delta a} \frac{\delta a}{\delta n} \frac{\delta n}{\delta b}$$

$$\frac{\delta n}{\delta b} = 1$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta a} \frac{\delta a}{\delta n} \frac{\delta n}{\delta b} = \left(\sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} * w_{(j,\, nIdx)}^{L+1} \right) * \sigma'(n)$$

In terms of the neuron gradient

$$\frac{\delta C}{\delta n} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n} = \left(\sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} * w_{(j,\, nIdx)}^{L+1}\right) * \sigma'(n)$$

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta n}\frac{\delta n}{\delta w_i} = \frac{\delta C}{\delta n} * a_i^{L-1}$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta a}\frac{\delta a}{\delta n}\frac{\delta n}{\delta b} = \frac{\delta C}{\delta n}$$

Save the neuron gradient, as we will need them for calculating gradients in the previous layers (as seen).

Updating every neuron's weights and bias

Now that we have calculated the formula to backpropagate, we need to backpropagate for all training samples. We are going to perform gradient descent over all training samples, meaning we need to take the average of the weight gradients and bias gradients of all the samples. Now we use those average gradients in our next and final calculation.

For each and every neuron

For each weight w

w = w - (averageWeightGradient * learningRate)

bias = bias - (averageBiasGradient * learningRate)

We subtract the gradient because we are trying to minimize the cost function. The gradient tells us which direction to add for maximum increase of the cost function, so the opposite of the gradient tells us which direction for a maximum decrease of the cost function.

We use a learning rate because we would like to take a "small step" into the direction opposite the gradient. If we take a "step" that is too big, the network could potentially "miss" the most optimal configuration of weights and biases. The learning rate is often between 0 and 1.

Congrats! You've officially completed one epoch of the training! In this case, an epoch is one iteration over all the training samples and backpropagating each training sample. Repeat this process for a large number of epochs, and watch as the MSE drops in value! For the final test, forward propage the inputs of the training sample, and the network should output values close to the target!

Summary (for implementation)

For each training sample, standardize your inputs (between 0 and 1). Pick an activation function. Pick a learning rate (between 0 and 1)

Create the forward propagation method

For the first layer, iterate through each neuron and set its activation value to the inputs

$$a^L_{nIdx} = inputs_{nIdx}$$

Iterate through the layers from left to right (in code, you would probably use an index). For every layer after the first layer, iterate through each neuron and calculate its neuron value and activation value

$$n = \left(\sum_{i=0}^{\#weights} w_i * a_i^{L-1} \right) + b$$

$$a = \sigma(n)$$

Create the backpropagation method

For the sample being backpropagated, put the inputs through a forward propagation and use the outputs and targets to calculate the output neurons' neuron gradients. Iterate through the layers from right to left.

For the rightmost (output) layer

Iterate through each neuron (in other words, "for each neuron in this layer")

Calculate the neuron gradient $\left(\frac{\delta C}{\delta n} \right)$ and use that to calculate each neuron's weights and bias. Save the neuron gradient into the neuron for calculating the weight and bias gradients for the hidden layers

$$\frac{\delta C}{\delta n} = 2(a - t) * \sigma'(n)$$

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta n} * a_i^{L-1}$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta n}$$

For hidden layers

Iterate through each neuron (in other words, "for each neuron in this layer")

Calculate the activation gradient

$$\frac{\delta C}{\delta a} = \sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} * w_{(j,\, nIdx)}^{L+1}$$

the activation gradient is equal to the sum of: the neuron gradient of the front neuron multiplied by the weight at index $nIdx$ of the the front neuron's weight array w (the weight connecting that neuron and the current neuron), for all front neurons.

Use the activation gradient to calculate the neuron gradient $\left(\frac{\delta C}{\delta n}\right)$ and use that to calculate each neuron's weights and bias. Save the neuron gradient into the neuron for calculating the weight and bias gradients for the hidden layers.

$$\frac{\delta C}{\delta n} = \left(\sum_{j=0}^{\#L+1} \frac{\delta C}{\delta n_j^{L+1}} * w_{(j,\, nIdx)}^{L+1}\right) * \sigma'(n)$$

$$\frac{\delta C}{\delta w_i} = \frac{\delta C}{\delta n} * a_i^{L-1}$$

$$\frac{\delta C}{\delta b} = \frac{\delta C}{\delta n}$$

Create an epoch

Backpropagate for every training sample, but make sure to continue to add (not set!) to the weight and bias gradients, as you will need to take the average. Once every sample has been backpropagated, take the averages of the weight and bias gradients and apply them using these formulas

For each weight w

w = w **-** (averageWeightGradient * learningRate)

bias = bias **-** (averageBiasGradient * learningRate)

Repeat epoch for a large number of times

The higher the epoch count, the more accurate the network will be (at least until the MSE converges)

Finishing up (Outro)

Thank you so much for reading my tutorial on creating a neural network with a simple gradient descent model! This is the first tutorial (or second, if you count the mess that was my implementation of the Iris Flower Identification dataset) I've ever made on this subject, so my apologies if it's a little bit rough. If you have any questions or critiques about the tutorial, feel free to send them my way! I would love to improve on my tutorials!

Feel free to check out my implementation for the XOR gate on GitHub
https://github.com/TacoL/Neural-Net