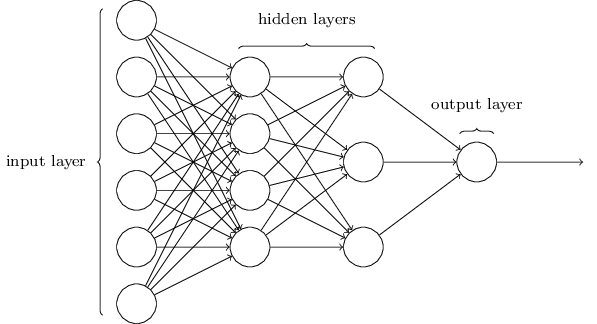
**Neural Networks: An Introduction**

The basic structure of the neural network is a collection of neurons that are created and connected together, allowing them to communicate and send messages to each other. Next, the network is tasked to solve a problem, which repeats the process in several iterations until it reaches a certain threshold, each time “nudging” the weight values and biases in attempt to increase the neural network’s performance at solving the problem. To train our network to estimate an unknown function, we give it a collection of data points – which we denote the “training set” – that the network will learn from and generalize on to make future inferences.

These collections of neurons are connected by synapses organized into three main layers: the input layer *x*, an arbitrary number of hidden layers, and the output layer *y*. You can see a simple neural network structure in the following diagram.



The number of neurons in the input layer is equal to the number of features (i.e data which we believe explains whatever process we are trying to model/explain) while the size of the output layer is equal to the number of classes to which our data points belong. The number of neurons in the hidden later is arbitrary and is usually dependent on the problem we are trying to solve.

The input layer communicates to one or more hidden layers where the actual processing is done via a system of weighted connections. You can think of weights as the “strength” of the connection between neurons. The hidden layer then produces an output through an activation function shown.

**Backpropagation Neural Network**

Backpropagation neural network is a supervised algorithm in which the error difference between the desired output and calculated output is back propagated. The procedure is repeated during learning to minimize the error by adjusting the weights through the back propagation of error.

The training of a neural network by backpropagation is broken down into the following parts:

1. Initialization of network
2. Forward propagation
3. Calculation and backpropagation of the associated error
4. Adjustments of weights

**Initialization of network**

We initialize the set of weights that need to be maintained for each neuron. One weight value for each input connection and an additional weight for the bias.

As we have mentioned above, the network is organized into layers. The input layer is the transposed row from the training dataset. Each node of the input layer is connected to each node of the hidden layer which is arbitrarily defined. The number of neurons of the output layer is the number of classes which the data points belong to.

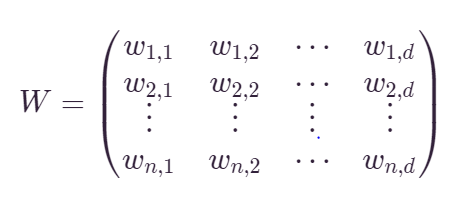
Each neuron of a layer holds a value representing an arbitrary combination of the nodes in the previous layer. For the first node in the layer we define this value as:

*z1 = w1x1 + w2x2 + …. + wnxn*

where n is the number of nodes in the input layer and x the value of each input node. This is just a linear combination of all the input values. In general, we have,

*z1 = w1,kx1 + w2,kx2 + …. + wn,kxn*

In order to efficiently execute all the necessary calculations, we weights are arranged into a weight matrix. We can define a matrix wnxd and use it to store all our weight values as follows:



The values for the weight matrices should be chosen randomly and not arbitrarily. A random normal distribution is ideal to avoid value that are bad for the learning process. The network weights and biases are random numbers with uniform distribution starting from -0.1 to 0.1. The following matrices were initialized:

(insert graph here of a random uniform distribution of values)

1. Input -> hidden layer 1 weights
2. Biases for hidden layer 1
3. Hidden Layer 1 -> hidden layer 2 weights
4. Biases for hidden layer 2
5. Hidden Layer 2 -> outputs weights
6. Biases for the output

For instance, when creating a weight matrix for input and hidden layer 1, it indicates the weights that are connecting the input and the hidden layer.

**Forward Propagation**

We start from the input have, we pass them through the network layer and calculate the actual output of the model straightforwardly. This step is called forward propagation because the calculation flow is going in the natural forward direction from the input to the neural network to the output.

This step generates predictions during training that will need to be corrected and the step needed after training to make predictions on the validation set.

Forward propagation is broken down into 2 parts:

1. Neuron activation
2. Neuron transfer

**Neuron activation**

Neuron activation is calculated as the weighted sum of the inputs.

v =

where w is a network weight, x is the transposed row from the dataset, and b is the bias, or the special weight that allows us to shift the function up and down and thus produces a better model for our data.

**Neuron transfer**

Once a neuron is activated, we need to transfer the activation to see what the neuron output actually is. We transform the activated neuron via a function, usually nonlinear into an activation level.

There are several transfer functions that we can use. The sigmoid function belongs to the most often used activation functions defined as:

The sigmoid activation function produces a number between 0 and 1 on an S-curve.

The network is forward propagated until the output is reached.

**Calculation and backpropagation of the associated error**

During this step, error is calculated by difference between the targeted output and actual output of each output unit. This error is back propagated to the previous layer that is the hidden layer. For each unit in the hidden layer N, error at the node is calculated. In the similar way, error at each node of previous hidden layer that is N-1 is calculated.

**Adjustments of Weights**

The calculated errors are used to correct the weights so that the error at each output unit is minimized.

Network weights are updated as follows:

*weight = weight + learning\_rate \*error \*input*

where weight is a given weight, learning\_rate is a parameter you must specify and controls how much change the weight to correct for the error, error is the error calculated by the backpropagation procedure for the neuron and input is the input value that caused the error.

The same procedure can be used for updating the bias weight, except there is no input term, or input is the fixed value of 1.0.

Each hidden unit updates its weights and bias. Each training cycle is called an epoch and the weights are updated in each cycle. It’s not analytically possible to determine where the global minimum is. Eventually the iteration stops until a certain threshold has been achieved.