**Neural networks**

**Softmax:**

Softmax is the function used to generate a probability function over all possible outputs in a classifier. For example, using the MNIST data, a Softmax output will have ten neurons each corresponding to the output number from 0 -9. The value of the neuron will be the probability that the input image is the corresponding number. This is done using the function

.

Here  is the output value of the th neuron in the Softmax output layer and is the sum of all the weights and bias input to the th neuron. It is clear to see from this equation that the sum over  of  will give a value of 1 like a probability distribution should.

**Cross Entropy Cost**

Cross entropy cost is one way in which a neural network can evaluate how bad the predicted probability distribution of a model  is compared to the true distribution . Using the MNIST example again,  will be the probability of the input image being the number  and  will be the one hot vector with a 1 at the index of the true value. The cross-entropy cost function is defined by the equation



Which will inform the network of how bad the prediction was and is the value the network tries to minimise during training. It is worth noting that if the network predicts an element to be zero, then the loss tends to infinity. It is also worth noting the minimum value of the loss is 0 so is always positive.

**Transfer Learning**

Transfer Learning in neural networks is where an existing network model is taken that is designed for a similar problem to the one you’re solving, and the later layers are retrained, sometimes only the final layer.

For example take a CNN that is trained to classify images of a large variety of objects (say 10000). The first layer will perform edge detection and the second layer will extract other features and so on until the bottleneck (second to last layer) which will provide a unique set of values for an image that the final layer will use to classify it. If we feed an image through the trained model of an object that is not part of the classifier, say a hairbrush, the output layer will be meaningless. However the network is still trained to extract useful features from an image so the bottleneck will still likely provide a set of distinguishing values for that image. This means that to classify images of hairbrushes as well, there is no need to retrain the whole model. Instead one can simply add images of hairbrushes to the training data and retrain just the final layer so that it has an extra output neuron corresponding to a hairbrush.

To do this one first takes a trained model. From this trained model, discard the final layer and replace it with an untrained output layer with the correct number of outputs for your problem. Training will involve several epochs whereby each image is fed through the entire network before the final layer weights are updated. This will happen several times and the feedforward step will take time. To shorten this process, one can feed the training data through the network once to calculate the bottleneck values for each image and store them on disk with the labels i.e. replace the images with the bottleneck values. The training then condenses into a case of training a simple fully connected network where we input the bottleneck values and train of the corresponding label.