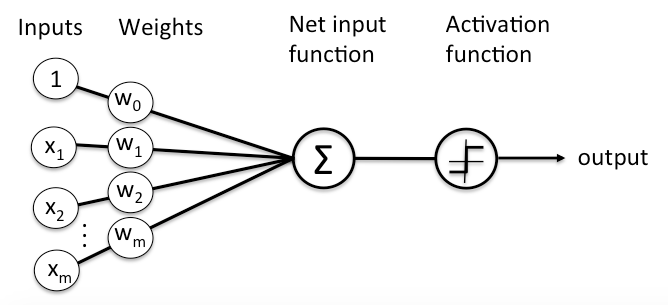
# Classifier Technology

* Neural Networks

Neural networks are one of the primary tools used in machine learning. The neural networks are the combination of a set of algorithms, and they are brain-inspired systems which are intended to replicate the way that human learns. They are excellent tools for finding patterns which are far too complicated or numerous for a human programmer to extract and teach the machine to learn. The neural network composed of several layers and each layer made of nodes. “A node combines input from the data with a set of coefficients(weights) that either amplify or dampen that input, thereby assigning significance to inputs for the task the algorithm is trying to learn”. (Introduction to Deep Neural Networks). All input-weight will sum and then passed through a node’s so-called activation function, to get what extent that signal progresses further through the network to affect the outcome.

Here is a diagram of what one node might looks like:

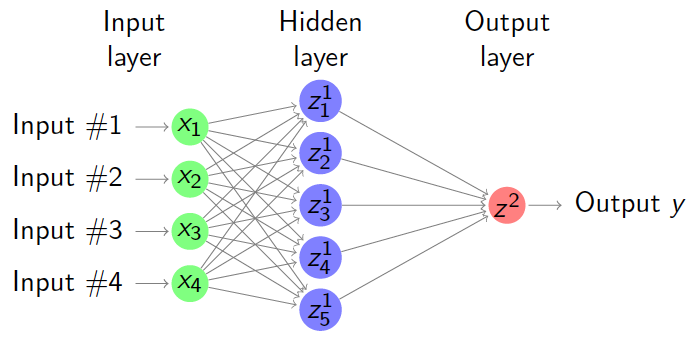


“A node layer is a row of those neuronlike switches that turn on or off as the input is fed through the net. Each layer’s output is simultaneously the subsequent layer’s input, starting from an initial input layer receiving your data.” (Neural Network Elements.)

Neural Network Elements. (n.d). In deeplearning4j.org. Retrieved from <https://deeplearning4j.org/neuralnet-overview#define>

For a slightly complex neural networks:

We suppose we have 4 inputs and weights, 5 hidden layer and 1bias:



Y = z2 = w1f(z11) + w2f(z21) + w3f(z31) + w4f(z41) + w5f(z51) + b =

Y =

In this report, neural network could be used in multi-classification problems. In order to train and optimize classifier, the concept below should be considered:

* Activation function:

We will use ReLU and softmax as activation function in this report.

ReLU activation function(Rectified linear units):

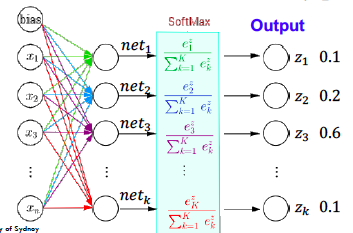
Most recent deep learning networks use for the hidden layers, a recitified linear unit has output 0 if the output is less than 0, and raw output otherwise. If the input is greater than 0, the output is equal to the input.

f(s) = max(0,z)

Softmax activation function:

The softmax function squashes the outputs of each unit to be between 0 and 1, just like a sigmoid function. But it also divides each output such that total sum of the outputs is equal to 1. The output of softmax function is equivalent to a categorical probability distribution, and it indicates the probability that of any of the classes are true:

= zk =



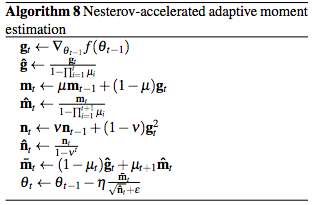
* Cost function:

In a neural network, cost functions are used to estimate the performance of a model. A cost function is a measure of how wrong the model is regarding its ability to estimate the relationship between X and y. To optimise classifier by using gradient descent method to minimise the cost function and reach a global minimum. The function used for softmax classifier below:

= -

* Gradient descent:

Gradient descent method is based on the cost function with parameter as mentioned. This method enables a model to learn the gradient or direction that the model should take in order to reduce errors.



http://cs229.stanford.edu/proj2015/054\_report.pdf

# Experiment and Discussion

The experiments are using third party library scikit-learn and keras

* Preprocessing Details

The previous methods are applied in our experiment. Before training the dataset, pre-processors are conducted to the datasets as following:

1. The dataset comprised of attributes with varying scales, the algorithm can benefit from rescaling the attributes to all have the same scale. This method is normalization and attributes rescaled into the range between 0 and 1 which is useful for optimization algorithms in gradient descent and weight inputs like neural networks. We rescale the data using scikit—learn using the MinMaxScaler class.
2. This experiment is mutil-class classifier, and in the last output layer is softmax. We can use LabelEncoder to normalize labels. We encode class values as integers and then convert integers to dummy variables (one hot encoding).

* Experiment Process

Neural network can be used as multi-class classifier by adding softmax in the last output layers initially. To apply neural network to this multi-classification problem, we need to set some parameters and adjust other parameters to optimize the performance.

1. More than one hidden layers:

In this assignment, we use a four-layer neural network, including two hidden layers, one input layer and one output layer. In each hidden layer, we use kernel\_initializer = ‘normal’ to define the way to set the initial random weights of keras layers. Each hidden layer has 64 neutrons, which can get the best performance. Besides, after the output layer we put a Softmax layer that can transfer the outputs to make sense of probability

1. Activation Function:

We put ReLU function in the two hidden layers and the softmax function after the output layer.

1. Nadam Algorithm:

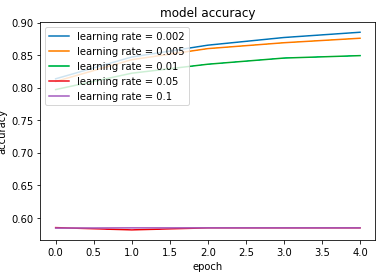
Nadam is a perfect optimizer to improve gradient descent efficiency for out multilayer neural network. The Nadam is like a combination of Adam algorithm RMSprop with Nesterov momentum. We set the learning rate is 0.005, and the other parameters like that: beta\_1 = 0.9, beta\_2 = 0.999, epsilon = None and schedule\_decay = 0.004.

1. Softmax and categorical\_coressentropy:

Softmax is very suitable for multi-class classificiation problems. The N-dimensional vector numbers will be transformed by softmax and then into another N-dimensional vector. The new vector’s number are all in range of 0 to 1.

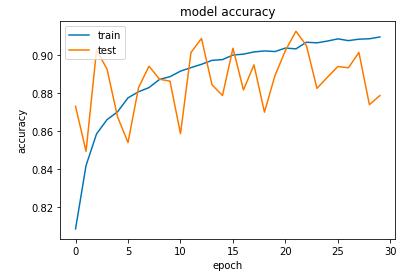
The categorical\_crossentropy should be implemented to provide a loss function for the neural network to optimize. This loss function indicates the distance between the predictions and the original label of the sample.

* + We are using the first 58101 samples as training data to train the model, and we set validation\_split is 0.1 which means the 90% data as training data and 10% data as testing data. We get the suitable parameters for this model.
    - For learning rate:



The 0.002 is the best learning rate.

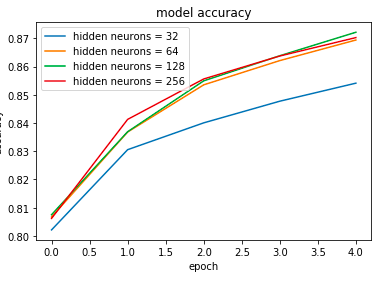
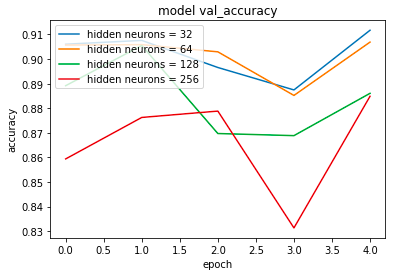
* + - For Epoch:



It is obviously that the accuracy in training data is rise but the accuracy in testing data is fluctuate with the increasing epoch. We set the epoch as 5 which is suitable.

* + - For hidden neurons:

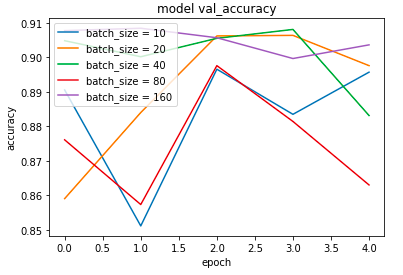
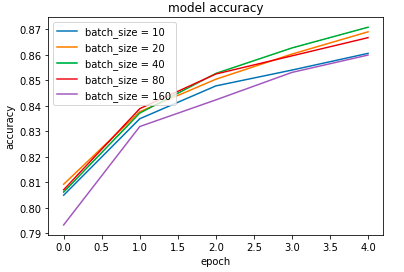
When trying to get the proper number for hidden neurons in each hidden layer, the result shown by the picture below. In this case, it indicates that when having too many hidden neurons, overfitting or other side-effects can damage the accuracy. So, the amount around 64 gets the best accuracy.

* + - For batch size:

After we get the best neurons in each hidden layer, we try to find the best batch size for each mini-batch, and the result shown by the picture below.

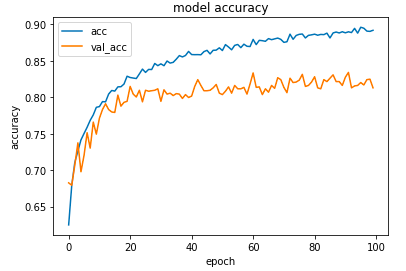
It is shown that the batch size of 20 gets the best accuracy of the five sizes.



* + For the rest of the raw data, we use the model above to train the data.

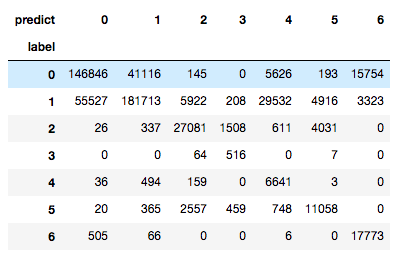
Unfortunately, the model does not perform well in testing data.

We increase the neutrons in the first hidden from 64 to 200, and in the two hidden layers, we set a Batch\_Normalization. The accuracy is around 89%, but the val\_accuracy is lower than accuracy.



For the testing data, the accuracy is 69%

The confusion matrix is below:



The classification report is below:

