1. Introduction
2. Exploratory Analysis
3. Predictive Modelling

One of the hypothesis involved was to check if a given set of certain parameters hold true, would the player make/miss the shot.

Predictive Modelling using Deep Learning

Neural networks generally perform better when the real-valued input and output variables are to be scaled to a sensible range. For this problem, each of the input variables and the target variable have a Gaussian distribution; therefore, standardizing the data in this case is desirable.

A small Multilayer Perceptron (MLP) model will be defined to address this problem and provide the basis for exploring different loss functions.

The model will expect 20 features as input as defined by the problem. The model will have one hidden layer with 25 nodes and will use the [rectified linear activation function (ReLU)](https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/). The output layer will have 1 node, given the one real-value to be predicted, and will use the linear activation function.

In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

In order to use stochastic gradient descent with [backpropagation of errors](https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/) to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

The function must also provide more sensitivity to the activation sum input and avoid easy saturation.

The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less. We can describe this function *g()* mathematically using the *max()* function over the set of 0.0 and the input *z*; for example:

g(z) = max{0, z}

The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are always output as zero.

By design, the output from ReLU is unbounded in the positive domain. This means that in some cases, the output can continue to grow in size. As such, it may be a good idea to use a form of weight regularization, such as an [L1 or L2 vector norm](https://machinelearningmastery.com/vector-norms-machine-learning/).

Training will be performed for 100 epochs and the test set will be evaluated at the end of each epoch so that we can [plot learning curves](https://machinelearningmastery.com/how-to-control-neural-network-model-capacity-with-nodes-and-layers/) at the end of the run.

Given the stochastic nature of the training algorithm, specific results may vary. In this case, we can see that the model learned the problem reasonably well, achieving about 83% accuracy on the training dataset and about 85% on the test dataset. The scores are reasonably close, suggesting the model is probably not over or underfit.