Our model is based on these steps:

* Getting historical data
* Create tech. analysis indicators
* Label the data to calibrate the model on what to look for
* Train the Neural Network
* Backtest
* Forecast

**Data**

* Get the data from yahoo finance
* We still with technical analysis data

**Data Labelling**

* We need to classify the data to tell the model what to look for
* The algo wont know what outcomes there are until the outcomes are defined.
  + We have binary labels: The stock price goes up or goes down.

**Neural Networks**

* We will be adding a neural network to the code which is an algo that can recognise underlying relationships in a set of data
* They tend to resemble the connections of neurons and synapse found in the brain

Neural networks have three layers:

* Input layer: All the training data we want to feed the model with
* Hidden layer: This layer manages the input data and processes it
  + For each input data, the hidden layer will create different weightings and formulae for it in order to create a formula for the output layer
* Output: Result, forecast etc.

Neural Network Training:

* Once we have all our data prepared, we will feed it into the model
* NN are simple and robust for trading

**Back testing**

* We need to check if the model is valid or not so we do this with back testing.

**Forecast**

* Generate a forecast with our model.

In a neural network, a neuron (also called a node or unit) is a single unit of computation that processes input data, applies a mathematical function (like the activation function), and passes the result to the next layer.

**Each neuron performs the following:**

* Receives Input: Each neuron gets inputs from the previous layer (or directly from the input features in the case of the first hidden layer).
* Weighted Sum: The inputs are multiplied by weights, and a bias is added to create a weighted sum.
* Activation Function: The neuron then applies an activation function (e.g., ReLU in your case) to introduce non-linearity, allowing the network to model complex patterns.
* Passes Output: The result is passed to the next layer or as the final output if it’s the last layer.

In our code, we have used 8 neurons in each hidden layer. This means that in each hidden layer, we have three hidden layers, has 8 neurons.

**Implications of 8 Neurons:**

* Learning Capacity:
  + The number of neurons in a layer directly affects how much information the network can “learn” from the input data.
  + More neurons allow the network to capture more complex patterns, but it also increases the complexity of the model.
  + 8 neurons are a moderate number. It provides the network with enough capacity to learn meaningful features without being too computationally expensive or prone to overfitting (memorizing the data rather than generalizing well).
* Balancing Underfitting and Overfitting:
  + If you have too few neurons, the model might underfit because it doesn’t have enough capacity to capture complex relationships in the data.
  + If you have too many neurons, the model might overfit, meaning it learns the noise and specific details of the training data instead of general patterns.
  + By using 8 neurons, you are aiming to balance between these two extremes: providing enough neurons to capture the important features without making the model too complex.
* Network Size and Efficiency:
  + More neurons increase the number of parameters (weights and biases) that the model has to learn, which can increase training time and computational cost.
  + Using 8 neurons per layer keeps the model relatively efficient and easier to train compared to using a much larger number, like 100 or more neurons.

*Intuition:*

*Imagine you’re training the model to identify patterns in stock prices based on technical indicators:*

*If you have just 1 or 2 neurons, the model might not have enough capacity to learn anything meaningful. So it’s brain isn’t big enough to learn.*

*If you have too many neurons, the model might memorize every tiny fluctuation in the training data, making it unable to generalize to new, unseen data.*

*8 neurons allow the model to have a reasonable level of complexity to learn useful patterns in the data without overfitting.*

**Why Multiple Layers and Neurons?**

Multiple Layers: Stacking multiple hidden layers (three in my case) helps the neural network learn more hierarchical features.

* The first layer may capture simple patterns.
* The second layer combines those simple patterns into more complex ones.
* And so on.

Multiple Neurons in Each Layer: Each neuron can learn a different feature or aspect of the data. With 8 neurons, the network has the ability to learn 8 different features at each level of abstraction.

**Results**

Shown below are the accuracy results of the NN on the training data and the testing data.

Firstly, what do the variables in the table tell us?

1. **Precision:**
   1. Precision tells you what percentage of the predicted positives were correct.
   2. TP (True Positives): Correctly predicted positive class.
   3. FP (False Positives): Incorrectly predicted positive class.
   4. A high precision means that when the model predicts a positive class, it is usually correct.
2. **Recall:**
   1. Recall tells you what percentage of actual positives were correctly identified by the model.
   2. FN (False Negatives): Actual positives that the model missed.
   3. High recall means that the model is good at identifying the positive class but may sometimes mislabel some negatives as positives.
3. **F1-Score:**
   1. The F1-score is the harmonic mean of precision and recall. It provides a balanced measure when you want to consider both precision and recall.
   2. The F1-score ranges from 0 to 1, where a higher score indicates better performance.
4. **Support**:
   1. Support refers to the number of actual instances of each class in the dataset. It tells you how many samples of class “0” and class “1” were available for evaluation.
5. **Accuracy**:
   1. Accuracy is the percentage of correct predictions out of all predictions made.
6. **Macro Avg:**
   1. This is the unweighted average of precision, recall, and F1-score across all classes. Each class contributes equally regardless of its support.
7. **Weighted Avg:**
   1. This is the weighted average of precision, recall, and F1-score across all classes, where each class contributes according to its support (i.e., the number of instances).

**Train Data Accuracy:**

* Class 0 (Predicted as “0”):
  + Precision: 0.55, meaning 55% of the time when the model predicted class 0, it was correct.
  + Recall: 0.27, meaning the model correctly identified only 27% of the actual class 0 instances.
  + F1-score: 0.36, a lower score indicating the model is not performing well on class 0.
  + Support: 1154, meaning there were 1154 actual occurrences of class 0 in the training data.
* Class 1 (Predicted as “1”):
  + Precision: 0.57, meaning 57% of the time the model was correct when it predicted class 1.
  + Recall: 0.81, meaning the model correctly identified 81% of the actual class 1 instances.
  + F1-score: 0.67, indicating the model performs reasonably well on class 1.
* Overall Accuracy: 0.56, which means the model is correct about 56% of the time on the training data.
* Macro Avg: Precision, recall, and F1-scores are roughly balanced across the two classes.
* Weighted Avg: The weighted average values are dominated by the class with more support (class 1).

**Test Data Accuracy:**

* Class 0:
  + Precision: 0.45, meaning 45% of the time when the model predicted class 0, it was correct.
  + Recall: 0.23, meaning the model correctly identified only 23% of actual class 0 instances.
  + F1-score: 0.30, indicating the model is not performing well on class 0 during testing.
  + Support: 506, meaning there were 506 actual occurrences of class 0 in the test data.
* Class 1:
  + Precision: 0.53, meaning 53% of the time the model predicted class 1, it was correct.
  + Recall: 0.75, meaning the model correctly identified 75% of the actual class 1 instances.
  + F1-score: 0.62, indicating the model performs decently on class 1 during testing.
* Overall Accuracy: 0.51, meaning the model is correct only 51% of the time on the test data.
* Macro Avg: The average precision, recall, and F1-scores are lower on the test data, especially for class 0.
* Weighted Avg: These values reflect the performance across both classes, again weighted by the number of instances.

**A screenshot of a data table

Description automatically generated with medium confidence**

**Findings.**

Class 1 Performs Better: In both training and testing, the model performs better on class 1 (price increase). It has higher precision, recall, and F1-scores for class 1 compared to class 0 (price decrease).

This is especially evident in the recall for class 1, where the model identifies around 75-81% of actual class 1 instances correctly.

Class 0 Performs Poorly: The model struggles with class 0 (price decrease). It has low recall and F1-scores for class 0, especially during testing, where it only identifies 23% of the actual class 0 instances.

This suggests that the model is biased toward predicting class 1, as it is better at identifying when prices will increase than when they will decrease.

Overfitting: The model’s performance on the training set (accuracy = 0.56) is slightly better than on the test set (accuracy = 0.51). This suggests the model may be overfitting to the training data, meaning it is learning patterns that are specific to the training data and not generalizing well to unseen data (test data).

Macro vs. Weighted Averages: The macro average F1-scores are lower than the weighted average F1-scores. This indicates that the model is biased towards class 1 because there are more instances of class 1, so the weighted average is higher due to the greater support for class 1.