This will make you think twice on putting money in a machine learning model…

Caleb Manske
Dec 17 · 6 min read

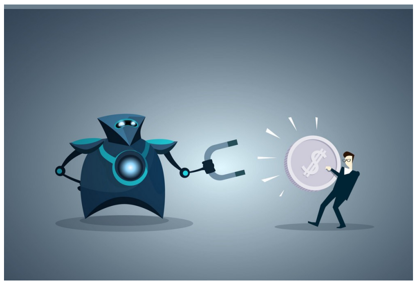
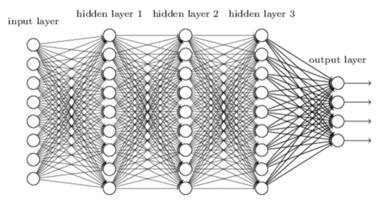

Source: Google images

Image you are in *one of two* scenarios:

> 1. An investor emails you, claiming his new stock algorithm for predicting can make you a lot of money. He thinks you should invest and badgers that you would be ridiculous not to invest.
>
> 2. You are a coder and you come across this cool algorithm that can predict your stock rising or falling in the future. You think it is so cool, and you are antsy to make money, so you copy the code for a different stock and plug in money.

I will explain the flaws in both of the scenarios, and I will also explain the opportunities for revision. I will explain through a framework with the use of three coded models I did with a type of machine learning called Deep Neural Networks (DNN).

But first, what is a Deep Neural Networks (DNN)? A DNN is an artificial neural network with multiple layers between the input and output layers. This synthetic process emulates the thinking process of the human brain. In coding, DNN contains hidden layers which help facilitate complex relationships to make a model. Here is a photo to illustrate:



A DNN can contain any number of hidden layers. The circles in the layers represent the nodes which are added to one another to make the next layer. (Source: Google images)

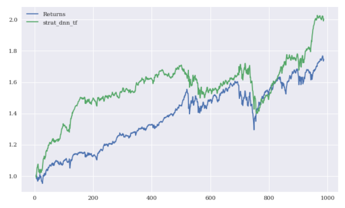
Let's move on, before I bore you with any more specifics…

In case you are confused, I am using a process of machine learning to learn the data relationships and to make a model from only part of the data while reserving the rest. The rest of the data will be tested against this model to see how well if performs.

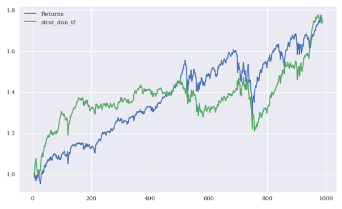
After that refresher, let's briefly point out the flaws and what can go wrong in scenario 1.

Flaw one, the model prediction could be way off, and you lose a lot of money:

As good as a model is, even with the exact parameters, the model can predict above or below a certain limit that is different from another instance of the same execution. For instance, when I refreshed my kernel (the thing that executes the code) on accident, numbers and the graph of the predication actually changed. No prediction is precise. Here are two graphs to illustrate that refreshed kernel:



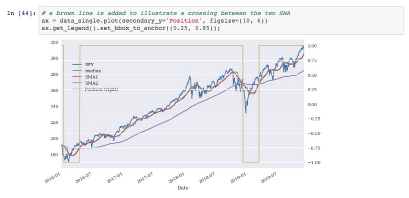
Execution 1 and same code.



Execution 2 and same code. See that the strategy (green line) is different from the graph above.

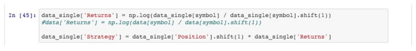
Flaw two the investor's model does not have score cards built in which can also lead for an opportunity in correction:

The basis of the model set up by scoring the performance does matter. For instance, is the algorithm merely looking at the rate of change between prices or does the model have financial definitions built into it for scoring — such as simple moving averages, returns, and standard deviations. For the model I built, I used a scorecard containing returns of the adjusted close and a strategy change from the short moving average crossing the long moving average.



```
In [44]: # a brown line is added to illustrate a crossing between the two SMA
         ax = data_single.plot(secondary_y='Position', figsize=(10, 6))
         ax.get_legend().set_bbox_to_anchor((0.25, 0.85));
```

Finding the crossing of the moving from simple moving average 1 to simple moving average 2. This crossing gets labeled as "Position".



```
In [45]: data_single['Returns'] = np.log(data_single[symbol] / data_single[symbol].shift(1))
         #data['Returns'] = np.log(data[symbol] / data[symbol].shift(1))

         data_single['Strategy'] = data_single['Position'].shift(1) * data_single['Returns']
```

Defining metrics, and using "Position" within the "Strategy" definition.

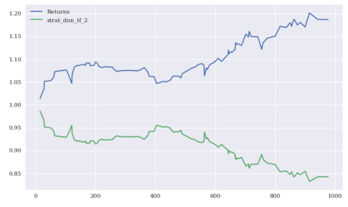
In addition, does the model also include other data to cross reference? Unfortunately for my small study, I did not have the data capabilities or time to add extra cross reference data for prediction.

Flaw and opportunity with bad stock price and good stock price:

The type of stock price that is imported into the model does matter. "Adjusted close" is different from "close". Adjusted close accounts for dividends and company stock splits. "Adjusted close" is also different from "high" and "low". This is obvious so I will not go into detail.

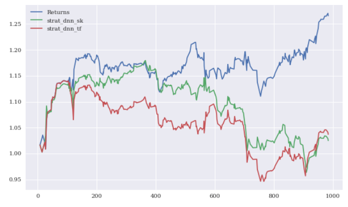
Flaw and opportunity with type of machine learning coding platform and package:

There are so many out there. I studied three DNN machine learners: MLPClassifier with Scikit-learn, and Google's TensorFlow DNNClassifier and TensorFlow's DNNRegressor. Let's hope he did not choose a DNNRegressor or you would be losing money as my study showed:



The strategy (green line) as a result of this DNNRegressor

But maybe if he chose DNNClassifier, so you might be okay:

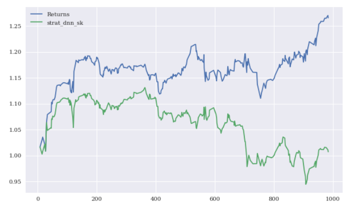


The strategy (green line) as a result of this DNNClassifier. The red line is the strategy of another model in made with Scikit-learn.

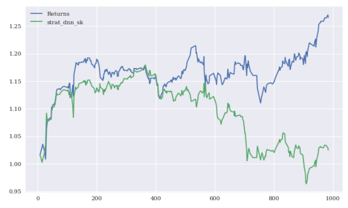
Let's briefly point out the flaws and what can go wrong in scenario 2.

Some of the flaws and opportunities are very similar to the past scenario so I will not go into them again. *But flaw one, parameters of a model do not perform equally across each stock:*

The model's number of hidden layers with tensors (look at the photo above for a quick explanation) reacts to different stocks differently. I was referencing a structure of code from a textbook and using it for a different stock. The number of tensors gave good performance for that stock in the textbook but not for the different stock in my project. I changed the number of tensors, and the stock actually performed better. Two graphs of one change:



The strategy (green line) is of the textbook code with a different stock.



Now the amount of nodes is changed in within the code and the graph reflects better change.

Secondly another flaw is that there is a bug in the code:

Oblivious or obvious but yet painful: *one must fix the bug*! Copying, pasting, and editing always encourage a bug to occur.

Another flaw and opportunity for you as a coder must fix the dataset to be properly formatted for input into the model:

This means the dataset should not have missing values (usually called NaNs or 0's in the profession of coding). Else, your data will be misrepresented, and will predict differently. However, in the case of 0 values for a stock, they would sometimes be okay since stocks go in a timeline. Sometimes filling with the average or another value is worse.

Also, the structure of code and data type does matter. When I was doing my project with using DNN with TensorFlow, I had to go try many structures of code just to import the data in the right format that would be accepted and liked by TensorFlow. This alone took me a day. The input for TensorFlow must be in a matrix/vector and has to have two inputs.

The moral of this article is to be careful with your money and your code. In sum, look at the framework, and the scoring metrics of the ML algorithm model. Look at the data type and input standard going into the model. And look at the type of coding package. Happy coding and happy earnings!

. . .

For more information on the code contain three DNN tested stock prediction models done by the author visit: https://github.com/studybug/DNN_ML_Stock_Predict

Caleb M is a graduate of Wheaton College, and a soon to be graduate of Udacity's Data Science Nanodegree program.

Machine Learning Investing Trading Coding Python