Chapter 12 : Part 4

**Deep Learning**

**RNN: Evaluating, Improving and Tuning the RNN**

Evaluation & Performance

**12.4.1 RMSE for to evaluate the model performance**

In the previous sections, the *RNN* we built was a *regressor*. Indeed, we were dealing with *Regression* because we were trying to *predict* a *continuous outcome* (the Google Stock Price). For *Regression*, the way to *evaluate* the *model performance* is with a *metric* called ***RMSE (Root Mean Squared Error)***. It is calculated as the root of the mean of the squared differences between the predictions and the real values.

* RMSE doesn't help here: However for our specific Stock Price Prediction problem, evaluating the model with the RMSE does not make much sense, since we are more *interested* in the *directions* taken by our *predictions*, rather than the *closeness* of their *values* to the *real stock price*. We want to check if our predictions follow the same directions as the *real stock* *price* and *we don’t really care whether our predictions are close the real stock price*. The *predictions* could indeed be close but often taking the opposite direction from the *real stock price*.
* Nevertheless if you are interested in the code that computes the ***RMSE*** for our ***Stock Price Prediction problem***, please find it just below:

**import** math

**from** sklearn.metrics **import** mean\_squared\_error

rmse = **math.sqrt**(**mean\_squared\_error**(real\_stock\_price, predicted\_stock\_price))

* Then consider *dividing* this *RMSE* by the *range* of the *Google Stock Price* values of *January 2017 (that is around 800)* to get a *relative error*, as opposed to an *absolute error*.
* It is more relevant since for example if you get an ***RMSE*** of ***50***, then this ***error*** would be ***very big*** if the ***stock price values*** ranged ***around*** ***100***, but it would be ***very small*** if the stock price values ranged around ***10000***.

**12.4.2 Different ways to improve RNN model**

1. Getting more Training Data: we trained our model on the past ***5*** *years* of the *Google Stock Price* but it would be even better to train it on the *past* ***10*** *years*.
2. Increasing the number of Timesteps: the model remembered the stock prices from the ***60*** *previous financial days* to predict the stock price of the next day. That’s because we chose a number of ***60*** *timesteps (****3*** *months)*. You could try to *increase* the number of *timesteps*, by choosing for example ***120*** *timesteps (****6*** *months)*.
3. Adding some other Indicators: if you have the *financial* instinct that the *stock price* of some other *companies* might be *correlated* to the one of *Google*, you could add this other stock price as a *new indicator* in the *training* *data*.
4. Adding more LSTM layers: we built a *RNN* with *four* *LSTM* *layers* but you could try with *even* *more*.
5. Adding more Neurones in the LSTM layers: we *highlighted* the fact that we needed a *high number of neurones* in the *LSTM* *layers* to respond better to the complexity of the problem and we chose to include *50 neurones* in each of our *4 LSTM layers*. You could try an *architecture* with even *more neurones* in each of the *4 (or more)* *LSTM* *layers*.

**12.4.3 Parameter tuning to improve RNN model**

You can do some *Parameter* *Tuning* on the *RNN* model we *implemented*. Remember, this time we are dealing with a *Regression* *problem* because we predict a *continuous* *outcome* (the Google Stock Price).

* *Parameter* *Tuning* for *Regression* is the same as *Parameter Tuning* for *Classification* which you learned in ANN and Chapter 11: Model Selection, the only difference is that you have to replace:

scoring = 'accuracy'

by:

scoring = 'neg\_mean\_squared\_error'

in the ***GridSearchCV*** class parameters.