MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROPOSAL

Andrei Anton
June 16, 2017

MACHINE LEARNING FOR PREDICTING THE **PRICE OF BITCOIN** AND THE **STOCK PRICES OF BITCOIN-INVOLVED COMPANIES** USING **LSTM RNNS**

DOMAIN BACKGROUND

Using **machine learning** for automated **prediction of time series data** has been a high interest topic for many years. The most well know problem in this field has been that of **stock price prediction**, but similar problems are those concerning **exchange rate prediction**. A (somewhat) recent surge in interest regarding blockchain based currencies (and other blockchain technologies) makes predicting the price of Bitcoin (the most popular blockchain currency) and that of the stock prices Bitcoin-involved companies an attractive problem.

There is also the high likelihood that successful approaches used to solve one type of time series data prediction problem could be generalized to other problems. In this way, work done on "not so socially useful" models like stock price prediction can find its way into models for predicting *electrical energy consumption*, *water consumption* or *food prices*, the successful prediction of which could contribute to efficient solving of important environmental, social and humanitarian problems. Also, changing our outlook towards a smaller scale, good predictions of energy usage patterns could be used to increase the autonomy/range of battery or solar powered devices and robots/drones.

Using **stock price prediction or similar financial-area problems seems like a good start for a novice interested in learning more about applying machine learning models to time series predictions**, since for these types of problems data is (a) readily and freely available, including historic data, (b) easy to interpret (there are no complicated corrections and normalizations that need to be applied, like when working with data from physical sensors), (c) informationally rich (economic data results from the actions of intelligent actors, so even when averaged out and compressed into a few "price" numbers, there is great potential for extracting useful patterns form this data, there is no "pure noise" data like what a faulty sensor could generate) and it's also (d) easy to tests "strategies" that represent practical uses of this models with the purpose of maximizing "profit" which is itself an easy to understand and interpret variable (in other applications like robotics "strategies" are harder to test, and it's also not just a few steps from prediction to action plan).

Of course, the obvious issue with approaching these as machine learning problems is the we are somewhat betting *against* the Efficient Market Hypothesis (EMH) whose consequences state that one cannot reliably predict the future price of a publicly traded asset. But we know that markets are not "perfect" and that most players do not efficiently exploit all the available information. And we also know that we're not the only ones assuming EMH to not be fully applicable to real world markets: investments funds and banks like JP Morgan and Goldman Sachs, and YouTube superstar Siraj Raval assume otherwise, so at least we're in good company for thinking this way.

The topic of **Bitcoin price prediction** has been previously explored in a certain degree, but most of the state-of-the-art models ale closely held trade secrets. At least the published models (Shah, Devavrat, Zhang 2014; Madan, Saluja, Zhao 2015; Sean McNally 2016 review) leave lots of approaches unexplored, from using potentially related data sources that could be used, to exploring prediction potential at different time level granularity and combining these approaches with the use of recurrent neural networks, more specifically LSTM models. Also, since at least some of the Bitcoin-involved companies are now publically listed, it's also interesting to explore the *reversed problem* of ***using Bitcoin price data to predict the stock price of Bitcoin-involved companies***.

The novelty of the current approaches I am investigating in this project consists ("novelty" at least in respect to *published* models) in:

- analysing predictive performance of similar models at different timescale granularities, ranging from 30 second to 24 hour intervals, and using (open, high, low, close, volume) data at *all* these granularities (most papers publish so far either use

only one time granularity setting, or some use only "average time period price" for prediction, which significantly reduces their potential for use as basis of real world automatic trading models)

- trying to incorporate features from other data sources that are conjectured to have some indirect influence on Bitcoin price, namely:
    - recent news headlines tone: to start with a very simplified model, we use recent (relative to each price data point) top news data whether it referenced bitcoin related topics positively (+1), negatively (-1) or not at all (0) (using Google Cloud Natural Language API to perform sentiment analysis on news headlines)
    - S&P 500: since we assume this indicator to be a measure of "health of US economy", and we assume there can be a correlation between this and Bitcoin price, we will derive from this a feature with a meaning similar to a "recent variation amplitude and magnitude average / moving average" that will be added to each Bitcoin price point – probably the moving average of the differenced closing price will work but I'll research a bit more before deciding
- trying to reverse the problem, and instead use the price of Bitcoin and its recent variation, to predict the price of the stock of a few particular Bitcoin-involved companies (Note: most Bitcoin-involved companies are privately owned, or with a different ownership structure but not publicly-listed stock, one reason being exactly that of avoiding the detrimental effects of extreme stock price variability driven by speculative trading using models like these).
- providing a basis for a public and open-source Bitcoin trading model that could save as a reference for other models (as mentioned below, it's hard/impossible to use any of the current models as a proper benchmark for a future model because they all seem to pursue very narrow/specific approaches that is not likely to coincide with any other future approaches so are hard to compare)

PROBLEM STATEMENT

This project aims to solve two related problems:

- Bitcoin price prediction – from (open, high, low, close, volume, day_of_week, news_tone, sp500_trend_derived_feature1) time series data, testing models with and without the last two features because I am not 100% sure they would have a positive contribution
- stock price prediction for Bitcoin-involved companies – from (open, high, low, adj_close, volume, anterior_bitcoin_price) data

A. Several models will be built that will predict:

- the next future price of Bitcoin in USD at the next time unit (with largest "time unit" being 24h, smallest 30s, and also at least one intermediate interval; more explicitly, we'll try to predict the *close price* as it's usual for simple stock price models)
- the sign of the future price variation (increase or decrease), derived from prediction described above – mostly because we want to know if at least we are predicting the direction of price change correctly, since this could still be useful for making trading decisions, even when actual errors are very high

based on:

- last data points in the time series, with data point features x_i (open, high, low, close, volume) + engineered features "day of week" (0 – 6) to also try and look for some day of week "seasonal" variation
- for 24h time granularity case only, the features mentioned at (1) to which we add the S&P 500 derived feature sp500_trend_derived_feature1 for the last P days (P will be between 1 and 30, chosen after performing experiments on a few sample from the dataset)
- for 24h time granularity case only, the features mentioned at (1) to which we add an engineered featured "news mentions in latest N days" with values of: 0 if there were no mentions of bitcoin related words in previous N days (or if the positive sentiment references were equal in number to the negative sentiment references), 1 if there were more "positive" than "negative" sentiment references, determined using Google Cloud Natural Language API for sentiment analysis (N will be determined after performing experiments on a few sample from the dataset)

B. A model will be built (of the same type as the most successful model from step A) that will predict:

- the future day value of the stocks of each of the 7 Bitcoin-involved companies
- the sign of the variation of the stock (increase or decrease), derived from prediction described above

based on:

- last data points in the time series, with data point features (open, high, low, adj. close, volume)
- previously mentioned features plus a similar length of bitcoin close price data

To summarize, the questions we seek answer to are:

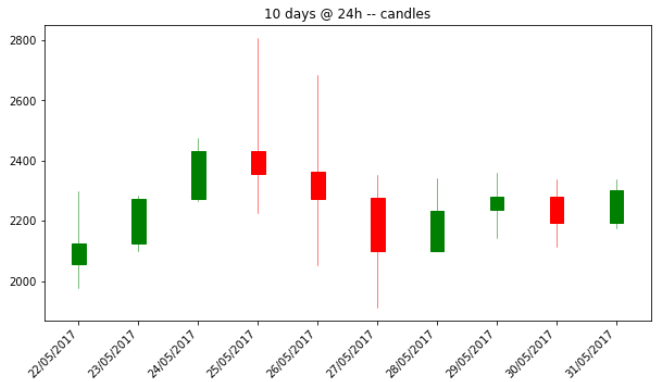
- Can Bitcoin price (or its variation direction) be predicted using LSTM RNN models, using OHLC pricing historical data (at different time granularities), recent news tone data, and recent S&P variation, with a good enough accuracy?
- Can the stock price of Bitcoin related companies (or its variation direction) be predicted using LSTM RNN models using stock price OHLC historical data and bitcoin price historical data, with a good enough accuracy? Is Bitcoin price of any help in predicting stock price of Bitcoin-involved companies?
- Do the LSTM RNN models actually perform any better than a multi-periodic linear regression model? (assuming this model even performs better than "predicting the price will stay the same")
- Do the LSTM RNN models perform better than an ARIMA model?

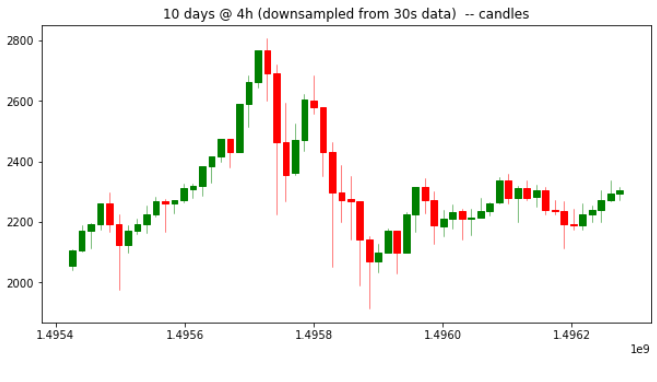
DATASETS AND INPUTS

The following bitcoin pricing datasets will be used (overlapping ranges from them will be clipped, some of the data may be discarded):

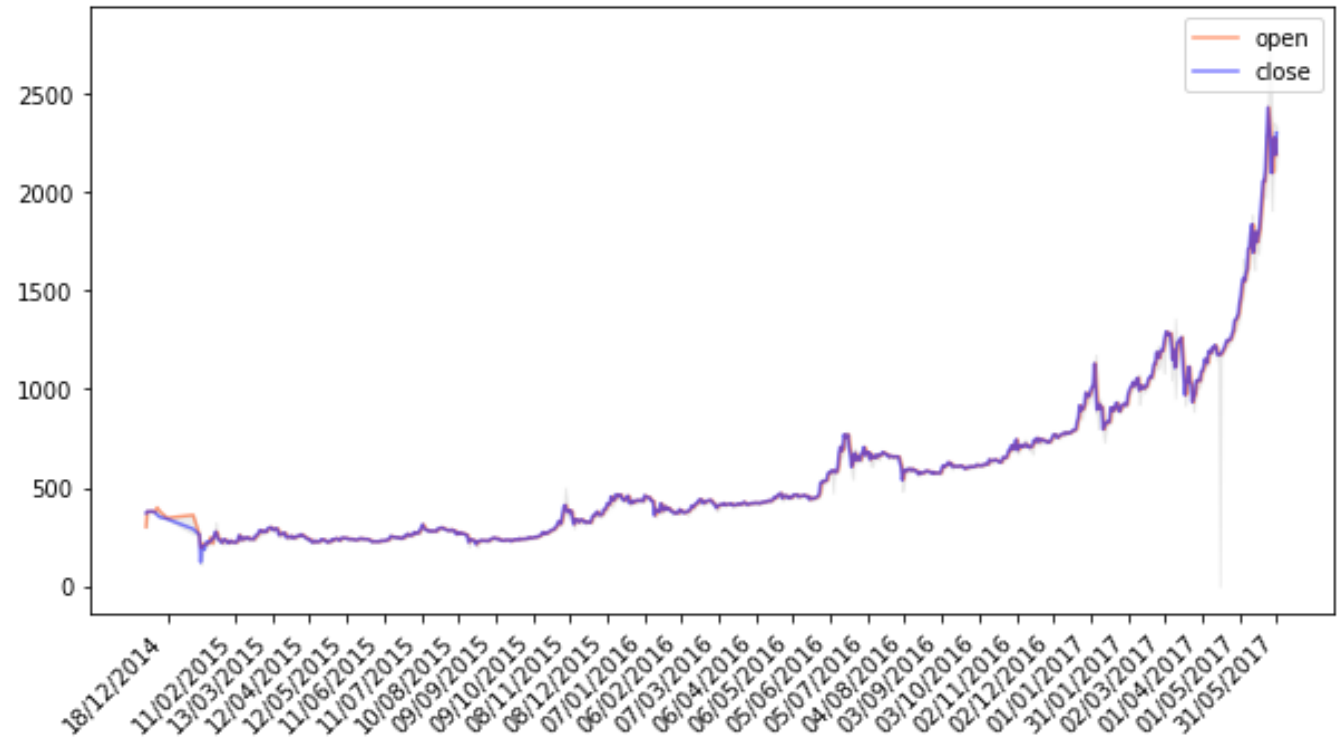
- BTC price in USD (low, high, open, close, volume) data (24h data is intended to be averaged from the 3 sources to compensate any inter-exchange variation, 30s data will only be used from GDAX – in general high frequency data is only easily available real-time so I would've needed to record it myself in advance from an API, or only available for small stretches of time)
    - BTC price in USD (low, high, open, close, volume) fetched from **GDAX** public API (historic data endpoint), *31/05/2017 - 01/12/2014*, saved in files: **`gdx_daily.csv`** for 24h interval data **(882 entries)** and **`gdx30s.csv`** for 30s interval data (**2.03 million data points** – when/if needed, intermediate time granularity data will be computed by averaging together multiple 30s data points)
    - BTC price in USD (low, high, open, close, volume) daily data from **Bitstamp** exchange, freely available at Quands, *13/9/2011 - 05/06/2017* **(2093 entries)**, saved in file **`BCHARTS-BITSTAMPUSD.csv`**
    - BTC price in USD (low, high, open, close, volume) daily data from **BTCE** exchange, freely available at Quands, *13/9/2011 - 05/06/2017* **(2123 entries)**, saved in file **`BCHARTS-BTCEUSD.csv`**

The most used visualization for this type of data are "candlestick charts", with a thick body between *open* and *close* price, a thin vertical line between *high* and *low* price, and colour (or fill) to denote direction of price movement. Shown below you can see this type of chart for a 10 days stretch of Bitcoin to USD price data, at 24h and 4h (downsampled from 30s data) time granularities (using actual data from the attached files):

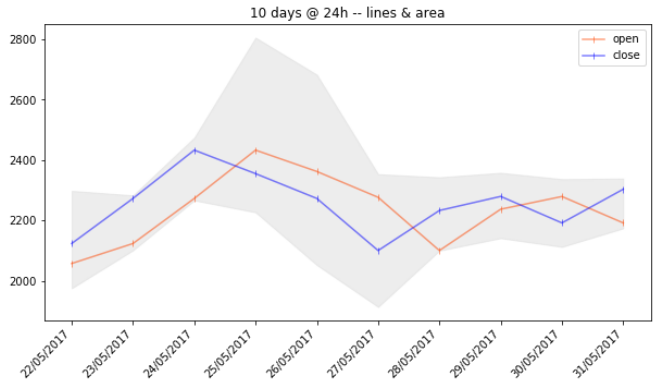


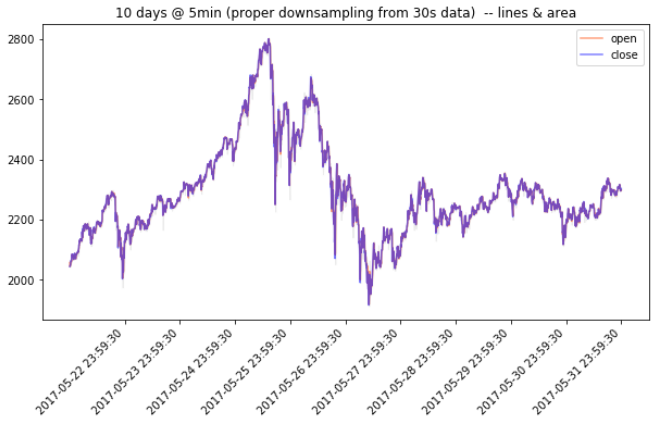


For better visualization of any number of data points (including a *very* large number), we prefer line and are charts like one below (the grey area denotes the *high-low* range and the orange and blue line denote the *open* and *close* prices), mainly because with these we can use the same visualization for vastly different time granularities.



(Full set of daily data – one can notice missing data for 2014 and an incorrectly recorded *low* of 0 which will have to be removed or "fixed" because it risks messing up regression models for that period)





(As we look at finer time granularities we can see an increased richness in information, but *also in noise*)

NOTE: We do *proper OLHC downsampling* (each period has the *open* of the first sub-period, the *close* of the last, the minimum *low*, the maximum *high,* the summed *volume*), not just "skipping steps", to make sure we don't throw away or corrupt data. (Nevertheless, visualizations like the 10days@5min above are almost indistinguishable visually between the "right" and "wrong" downsampling methods.) We are also aware that this kind of "proper downsampling" is still *wrong* from an economist's p.o.v. (despite the preserved information that still makes data useful for prediction), because a time interval's open price *does not have to* match the previous one's close (because pricing in a market is the consequence of discrete events, resulting in "illiquidity" etc.), and accordingly **we do not perform this operation for stock price data**, but for a high-liquidity asset like Bitcoin we can safely do this operation (previous-close/current-open gap is too small to matter). But it's still worth noting this because it explains that the small difference that we would obtain by, for example, comparing "actually collected 24h data" with "24h data produced by downsampling 30s data" – a very small difference wouldn't mean we have corrupted data, it's simply because of how a real market works.

S&P 500 Index value:

- daily (low, high, open, close, volume) data, 13/9/2011 - 05/06/2017, saved in file sp500-GSPC.csv

Stock prices of bitcoin involved companies, daily (low, high, open, close, volume):

- GBTC - Grayscale Bitcoin Investment Trust - bitcoin investor (for investors not willing to invest in BC directly) – in **`GBTC.csv`** *11/05/2015 - 05/06/2017* **(522 entries)**

https://www.google.com/finance?q=OTCMKTS%3AGBTC&ei=ojsoWaitMsHCUfH1pqAM
https://finance.yahoo.com/quote/GBTC?p=GBTC

- BTCS Inc. – in **`BTCS.csv`**
  15/11/2010 - 05/06/2017 **(1649 entries)**
  https://www.google.com/finance?q=OTCMKTS%3ABTCS&ei=_jwoWfmqOMKcUp_RuegM
- 1st Bitcoin Capital (BITCF) – in **`BITCF.csv`**
  21/04/2016 - 05/06/2017 **(283 entries)**
  https://www.google.com/finance?q=OTCMKTS%3ABITCF&ei=-0EoWdiPDcORUvyFi8AM
  https://finance.yahoo.com/quote/BITCF/?p=BITCF
- Overstock.com (ostk), Inc. - proeminent bitcoin accepting shop – in **`ostk.csv`**
  15/05/2014 - 05/06/2017 **(686 entries)**
  https://www.google.com/finance?q=NASDAQ%3AOSTK&ei=LD0oWfiZFMHCUfH1pqAM
  NOTE: missing adjusted close (only "close")
- WPCS International, owner of BTX Trader (https://btxtrader.com/) since 18 Jan 2013 – in **`WPCS.csv`**
  18/01/2013 - 05/06/2017 **(1102 entries)**
  NOTE: older than 18/01/2013 data is not considered because prior to this date the company was not a "Bitcoin-involved company"
  https://www.google.com/finance?q=NASDAQ%3AWPCS&ei=jT4oWfEukcRT-8OsuAs
- SmartTrans Holding (from 1 Jan 2014 is involved in bitcoin through Digi8 etc.) – in **`SMA.AX.csv`**
  01/01/2014 - 05/06/2017 **(868 entries)**
  https://www.google.com/finance?q=ASX%3ASMA&ei=iz8oWbFtzJVSko-csAw
- Bitcoin Group SE (ADE.F) – in **`ADE.F.csv`**
  05/10/2016 - 05/06/2017 **(170 entries)**
  https://finance.yahoo.com/quote/ADE.F/?p=ADE.F

Similar visualizations can be used for stock prices as we used for bitcoin prices.

SOLUTION STATEMENT

Two types of models will be explored for this prediction task.

First, we'll choose the simplest model that makes sense for this type of data, namely Stochastic Gradient Descent Linear Regression on 3 periods of K_1, K_2, and K_3 previous data points, then taking a linear combinations of the values predicted by these regressions), to serve as benchmark model to the more advanced Recurrent Neural Network (RNN) based model. An extra reason for choosing this model over other alternative simple models is that it's easy to adapt for on-line learning too, which would probably be the more realistic production use case, where the model would be updated with information from new data points as they are collected.

Seconds, we'll use a Recurrent Neural Network model, more exactly a LSTM model. Finding the hyperparameters like/and no. of layers for this model, as well as finding ways to select and preprocess training data for it is part of the problem we are solving. The aim will nevertheless be to achieve superior performance to the simple model mentioned above.

BENCHMARK MODEL(S)

Finding a proper benchmark model for bitcoin price prediction is a tricky issue. Basically every published paper uses a different model, on different data and reports different kinds of results. Some prominent papers (Shah, Devavrat, Zhang 2014) only report the performance (in term of ROI) of the simulated trading strategies which implement their predictive models. Most of the others report <10% better than random guess predictive performance (Madan, Saluja, Zhao 2015) for the sign of change (which can nevertheless be good enough to allow a trading strategy to extract some profit). Generally speaking, a ny way of trying to pick a benchmark model from the published ones leads quickly to an apples to oranges comparison.

I decided in the end to choose using an ARIMA model for comparison, for both Bitcoin prices and the stock prices of Bitcoin-involved companies.

For Bitcoin price alone, a Bayesian Linear Regression model might also be investigated, both the simple variant that is expected to perform similarly with SGD LR (our "simple benchmark" model), and the more advanced for that uses a linear combination of regression predictions for 3 different length time ranges. The problem is that implementing an "all the tricks included" version of this more advanced model (that would include things like time series clustering analysis for selection training time ranges etc.) would be very time consuming (more than I can allocate for this project now), while the "naïve" version would probably perform similar to SGD LR.

EVALUATION METRICS

Two evaluation metrics will be used to quantify the performance of the models:

Accuracy, computed as:

- $\frac{\#\ of\ time\ points\ with\ correctly\ classified\ direction\ of\ price\ change}{\#\ of\ total\ predicted\ time\ points}$, averaged for all test dataset points

- **NOTE:** In general, we are only interested in predicting one time point ahead, at least for larger than 10min time intervals (this is because we assume that in a real life online-trained trading model, after one time interval passed, we already have an additional data point that is immediately used to retrain the model before predicting next data point)

RMSE both in price unit and percentual:

- $\sqrt{\sum_{\substack{for\ each\\ predicted\ point}} \frac{(predictedValue - actualValue)^2}{\#\ of\ total\ predicted\ time\ points}}$, normalized by STDEV for percentul value (if computed from not previously normalized data)
- **NOTE:** again, this will also mainly be used to predict one point ahead

PROJECT DESIGN

DATA ANALYSIS AND PROCESSING

Data analysis start with an assessment of data quality and general characteristics, like:

- counting missing data points: both missing observations and observations with missing or clearly wrong value for some of the features – based on this we may choose to discard older data with more missing observations (probably we'll discard pre 2015-01-01 data because there are lots of missing data points, and it's also generally agreed that pre-2015 data for Bitcoin is very atypical since the market was still immature; downsampling will probably be made from 30s data with about 0.7% missing data to 10min data with no missing data points; other things will be determined after further analysis)
- general characteristics like global and monthly mean and std. deviation will be computed (for closing price), as well as moving average, moving standard deviation (multiple periods will be tried)
- visual inspection will be performed at different time granularities (with 24h dataset, and variously downsampled data from the 30s dataset including 10min, which is a realistic interval considering there are trading latencies from regular traders – we are not interested in truly-HF trading in this project), on candlestick/OHLC graphs with Bollinger Bands and volume fluctuation charts added, looking mainly for seasonal patterns or "traditional" trading technical analysis patterns ("head and shoulders", "cup and candle", "triangles" etc.)
- further analysis will also investigate how the differenced time series look, to try and see if making it stationary will help (unfortunately, this is not a simple question in the case of Bitcoin price: while this year's 2017 data will clearly benefit from de-trending, a first look at older data doesn't make it so clear as in that case useful patterns might be lost)

Data processing will start with a differencing for de-trending, if we choose to do this at all.

Next we plan to split data into 70% training and 30% test data (since this is time series data, order needs to be preserved, so regular k-fold crossvalidation is not an option, and other more complex alternative may introduce unneeded complexities). To account for the immaturity of the bitcoin market, we will also run an alternative scenario, where we split the ~2.5 years period

into 3 ~equal “eras”, and split each era into 70% training and 30% test data. This is because we can assume that some of the older patterns in bitcoin pricing might misinform trading on the present day market which might have changed character a bit.

Next we will proceed with scaling to [-1;1] (for Linear Regression model we might choose not to scale, or scaling to [0;1] instead, some further research is required here…), with min/max coefficients calculated from the training dataset (avoiding any contamination of model with information from test set).

EXPERIMENTAL TEST SETUP

We intend to pursue a walk-forward model validation strategy with a sliding window of N previous data points considered for prediction

CALCULATION SCORE FOR A REFERENCE “PERSISTENCE MODELS FORECAST”

The RMSE and accuracies will be calculated for a model that simply predicts the previous price or price variation direction, to have some sort of reference.

IMPLEMENTING A “SIMPLE” REGRESSION MODEL

This follows the technique described in *Shah, Devavrat and Zhang 2014,* but since this is not the actual investigated model, but a simple “benchmark model”, we implement the simplest possible variant of this (we don’t do time series clustering to determine best sub-sequences for learning etc.).

For each data point to be predicted, 3 models that performs linear regression on the observations from different periods of immediately anterior time, S_1, S_2, S_3 will be created and trained. Considering that we’ll simply be doing stochastic gradient descent linear regression on a limited number of data points, it is not that computationally expensive to do this before each prediction. A linear combination $w_0 + w_1 P(S_1) + w_2 P(S_2) + w_3 P(S_3)$ will be the actual delivered prediction. These “sub-model combination parameters” will also be determined by doing SGD LR, on the stored predictions for a large subset of the total training data (or the entire training data). When performing predictions on the test data, these same learned *w* parameters will be used.

This model’s performance will then be evaluated on the test dataset(s), namely RMSE (both procentual and in actual price units) for price prediction, and accuracy (percent of correctly predicted price variations) for price change direction change prediction.

The scikit-learn SGDRegressor implementation will be used. For the model hyperparameters, we’ll pick `squared_loss` loss function (ordinary least squares), `l2` regularization, and perform grid search to pick optimal `alpha` and `n_iter`. Other parameters will be set to the value that makes most sense for this use case (like shuffle=True etc.)

Note: We understand that this models is not a suitable model for time series prediction, even though the slightly improved “linear combination of linear models predictions” setup does capture *some* of the sequentiality characteristic of our time series data, but we use this because we want to benchmark the RNN model against something *better* than “persistence model forecast”.

IMPLEMENTING AN AUTOREGRESSIVE INTEGRATED MOVING AVERAGE (ARIMA) MODEL

The tsa.statespace.sarimax.SARIMAX model from the statsmodels Python package will be used. SARMAX stands for “Seasonal AutoRegressive Integrated Moving Averages with eXogenous regressors” but we will focus on the ARIMA components of this, choosing the simplest possible implementation.

Model hyperparameters will be set though a “grid search approach”.

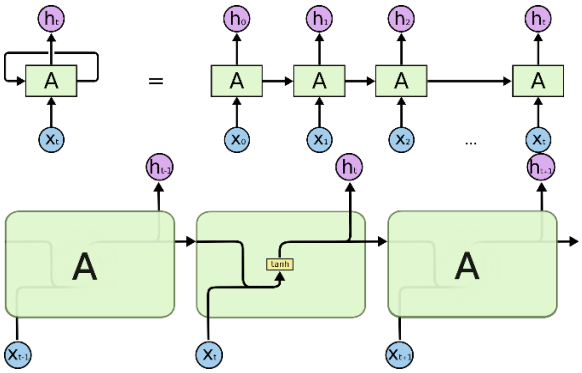
This model’s predictive performance will be assessed.

IMPLEMENTING A LSTM RNN MODEL

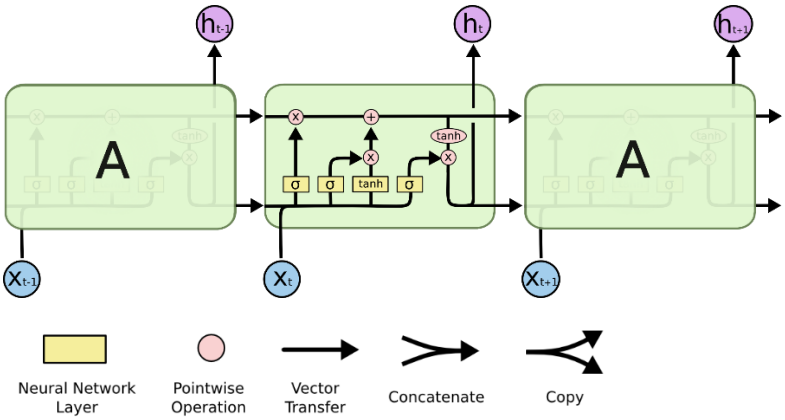
A recurrent neural network model is considered to be a good choice for predicting a sequence that is rich both in noise and (presumably) useful information like this one.

Recurrent Neural Networks started (re)gaining popularity in 2015, as best explained in [Andrej Karpathy's famous blog article] "The Unreasonable Effectiveness of Recurrent Neural Networks" which mainly served to draw attention to recent positive results with RNNs, mainly LSTMs (introduced by *Hochreiter & Schmidhuber 1997*).

Simple RNN:



LSTM RNN:



(**NOTE:** The actual submitted capstone project will ensure the reproducibility right for included graphics and might replace this figure with a different re-drawn one)

The practical difference between "classic" RNNs and LSTM networks is that LSTM "cells" (which can be seen like "machines made from multiple sub-NNs") solve the "forgetting" problem of simple RNNs that makes them practically useless for sequences with dependencies between patterns separated by large intervals (the problem is caused mainly by vanishing-gradient and exploding-gradient issues, though there are multiple mathematical details and proposed solutions here). Also, LSTMs have the practical advantage that the state they maintain on long term is actually accessible and transferable, allowing for much "tuning" of how the NN is to be used in a model.

Choice between "standard" LSTMs and other variants like GRUs was made by simple picking what has best support in preferred frameworks/libraries and best documentation.

Since the model has memory, we can simply train it using 1 previous data point and "walk-forward" though the training data. A variant with multiple previous data points will also be tried. (Of course, we'll need to keep the cell state with this approach.).

Basically the model variants that we will explore bill differ in:

- general training input configuration variations: input shape (depending on number of previous observations used, but also on how we treat the features of each data point), batch size (how many samples to be processed for an update of the weights of the network), number of epochs (times to pass through training data: based on literature and recommendations received so far we'll pick a few values between 100 and 3000)
- network configuration:
    - for the "1 previous data point input" configuration, the NN will maintain cell state between predictions (and start pre-initialized with the state it has after one walk-forward pass through the entire test dataset)
    - for the "N previous data points input" configurations, we will try both a stateful and a non-stateful configuration (the whole point of using a window of previous data points would be to be able to start a batch of predictions with no pre-learned state, but since RNN predictive performance behaviour is not so intuitive in practice, we will try both variations)
- network size: here we'll only vary the number of neurons in the LSTM hidden layer between 1 and 50 (only if we see an increasing trend we'll try more), testing on smaller sub-samples of the training data
    - for the best performing variant we'll also try adding another layer, moving towards something more akin to "deep learning"

The Keras LSTM implementation will be used (with Tensor Flow backend).

The Flask web framework will be used to create micro-service exposing a REST(-ish) API with the following endpoints (we may actually namespace then as **`/api/v1/model-types/lstm-1a/train`** etc. if we find more than one "good" model and we want to expose API endpoints for each):

- **`POST /api/v1/train`** *(this would be the "create" endpoint actually if we consider "trained models" to be the "resources" of our REST API)*
  returns: model_id
  parameters:
  - start_date
  - end_date
  - time_granularity – eg. `24h`, `1h`, `10m`
  - product – this will be either 'btc-usd' for Bitcoin USD price prediction or the symbol of the company for which stock price is to be predicted
  - use_bitcoin_price – set this to `true` if Bitcoin price is to be used as a feature for predicting stock price of Bitcoin-involved companies
  - note: the API assumes that for each combination of parameters, you will create a different model, therefore if you want to predict the price of multiple "products" (bitcoin price, stock prices") you will create multiple models
- **`POST /api/v1/models/<model_id>/predict`**
  return: array of predicted closing prices
  parameters: dates[]
  note 1: if, at the models set time_granularity, there are multiple intermediary data points between the last training data and the queried for data, the intermediate points are generated, one at a time, and each point included as "data" to be used for the prediction of the next one (obviously the error can grow exponentially the further in time we try to predict)
  note 2: after each call to this endpoint, model state will be reset
- **DELETE /api/v1/models/<model_id>** - we need this to clear up old models so we don't run out of memory

BUILDING A SIMPLE WEB FRONTEND GUI ON TOP OF THE API

A simple web frontend UI will be built, consisting of at least 3 screens:

- a "**Create and train model**" screen accessible at a **`/make-model`** URL, showing a form with the necessary inputs to satisfy the requirements of the **`POST /api/v1/train`** API endpoint

a "**Query model**" screen, accessible at **`/models/<model_id>`** URL, to which the user is redirected after the previous one and displays a form with the necessary inputs to satisfy the requirements of the **`POST /api/v1/models/<model_id>/predict`** API endpoint, and a "delete" button which calls the **`DELETE /api/v1/models/<model_id>`** API endpoint

- a "**Predictions**" screen accessible at **`/models/<model_id>/query?dates=<...>`** URL, to which the user is redirected after the previous one

Note: A password protected version of this web GUI will be available online, password protected, at an URL that will be shared with the Udacity capstone project reviewer (will be included in submitted project files together with the password). Instructions for running it locally/offline will also be provided.