An Analysis of Various Time-Series Forecasting Techniques Applied to Cryptocurrency Trading

Alex Guerra

November 1, 2018

Background

The problem of correctly predicting markets algorithmic-ally has long been a goal of traders globally.

Gradient Descent Algorithm for Approximating Prices

The goal of this model is to predict the next data point in a stable periodic market given previous prices. It is primitive, yet if a market is truly periodic, this model should produce better than random results.

If we have (for $\theta = (\theta_0, \ldots, \theta_3)$ and a scalar $x \in \mathbb{R}$ representing one timestep), our hypothesis function looks like:

$$h_{\theta(x)} = \theta_0 \sin(\theta_1 x + \theta_2) + \theta_3 + \theta_4 x$$

and our cost function look like (where $m \in \mathbb{Z}$ is the number of data points we evaluate this on)

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right)^2,$$

then gradient descent dictates that to update our θ_j, we have (with $\alpha \in \mathbb{R}$ as the learning rate)

$$\theta_{j+1} \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

What remains to do is to calculate the partials for each θ_j. Note that this is not a trivial task as our hypothesis is very much non-linear. We have

$$\begin{aligned}
\frac{\partial}{\partial \theta_0} J(\theta)(x_j) &= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) \sin\left(\theta_1 x^{(i)} + \theta_2 \right) \\
\frac{\partial}{\partial \theta_1} J(\theta)(x_j) &= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) \theta_0 x^{(i)} \cos\left(\theta_1 x^{(i)} + \theta_2 \right) \\
\frac{\partial}{\partial \theta_2} J(\theta)(x_j) &= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) \theta_0 \cos\left(\theta_1 x^{(i)} + \theta_2 \right) \\
\frac{\partial}{\partial \theta_3} J(\theta)(x_j) &= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) \\
\frac{\partial}{\partial \theta_4} J(\theta)(x_j) &= \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) x^{(i)}
\end{aligned}$$

Thus, we have a method that will, for a sufficiently small learning rate, locally minimize the cost function for a given hypothesis (note that a global minimum is far from guaranteed due to the non-convexness of the hypothesis function). If we have succeeded, we will have successfully fit a sine function to the data, from which we can move on to the next step. We will use this model to predict the next x, and due to the periodic nature of the true data, in the long run we will correctly predict the price more than we incorrectly do.

One important assumption of the previous paragraph was the convexity of the function we are trying to minimize. Unfortunately, in our model, this assumption if not met. There are computationally expensive ways of being relatively certain that a global minimum has been reached; however, in a market setting where these models need to be trained in a couple seconds, this will not suffice. Thus, to account for that, we must find a heuristic way of starting in the right "convex pool" so that gradient descent can work as intended. What we do is use heuristics to take good guesses for θ_0, θ_2, θ_3, and θ_4, and then run multiple threads in parallel with different starting values of θ_1 to converge in different convex pools, from which we select the one with the lowest error to maximize the probability that we have found a global minimum. This can be scaled well and can converge very quickly with proper implementation.