Predicting The Stock Market With Sentiment Analysis

Royi Rassin)

26.03.2021

Introduction

It is widely known that the stock market is unpredictable and while the jaded say it is no different than gambling, there are many communities that identify, share, and discuss stocks that are more likely to succeed or have a meaningful risk/reward ratio. In this work we explore the correlation between the sentiment of the thread-titles of the 'Stocks' subreddit community and the trend of the discussed stocks in the near future. While this is the big-picture goal, this project is part of an Optimization course, and the emphasis is on the optimization-technique I applied and why I think it is the most suitable one. In terms of the flow of the article: in the General Approach I will give a broad idea of what I did. In the Optimizers section, an overview of the different optimizers I tested and their big-picture idea is presented. Then, in the Experimental Results, I delve into the performance. And finally, I conclude the article with a discussion.

1 Related Works

Most of the notes from this paper are taken from Sebastian Ruder's excellent overview [Rud16] and the Adam-W paper [LH19].

2 General approach

I first tackled the problem by scraping threads that were posted from 18/09/2019 to 17/03/2021. The training-set is made of posts created up to 17/09/2020, and the development set from 18/09/2020. Labels were created by screening for keywords that are known to be positive or negative, and if a title contained both or neither, it was deemed neutral (keywords can be found in the appendix). Additionally, I made sure the class-split is completely balanced (for both configurations: negative/positive or negative/positive/neutral). Then, I trained a Bidirectional LSTM (BiLSTM) to classify the titles' sentiment: negative/positive, and measured the network's F1. Finally, I checked the development set for the most common sentiment for select stocks (they were only cherry-picked in the

sense that they are a frequent topic within the community), and compared it to that of the ground-truth. Furthermore, the sentiment was also compared to the change in percent during the time-span of the development set (how much the stock-price increased or decrease). Data pertaining to stocks was extracted from Yahoo Finance. However, the performance of the network can only be measured against the ground-truth of the dataset, if the network successfully predicts the development set, but fails to capture the trend of the stock-price, it is an issue with the data, not the model.

3 Optimizers

3.1 Gradient Descent

3.2 Batch Gradient Descent

Batch Gradient Descent (BGD) computes the gradients of the objective function J with respect to the parameters θ for the entire dataset:

$$\theta = \theta - \alpha * \nabla_\theta J(\theta)$$

Figure 1: For learning rate α, objective-function J, and parameters θ
Since every step iterates the entire dataset, the algorithm can be incredibly slow and memory-intensive.

3.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD), in a sense the opposite of BGD, updates the parameters for each example:

$$\theta = \theta - \alpha * \nabla_\theta J(\theta; x^i; y^i)$$

Figure 2: For learning rate α, objective-function J, parameters θ, and sample (x, y)
Since every update is performed after computing one example, the algorithm suffers from high variance which can lead to better (or worse) local minima.

3.3.1 Mini Batch Gradient Descent

Mini-Batch Gradient Descent attempts to capture the best of SGD and BGD. By computing the gradients for small batches of data, the variance is reduced and the memory requirements are relaxed (compared to BGD). In our work, we compared our results to this variant (with momentum):

$$\theta = \theta - \alpha * \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Figure 3: For learning rate α, objective-function J, parameters θ, and sample (x, y)

Unfortunately, Gradient Descent's performance highly depends on the learning-rate, it is not adaptive (more on this later), it is likely to fall into a suboptimal-local-minima, and it updates all parameters equally, even if they are not equally significant.

3.3.2 Momentum

SGD has trouble navigating areas where the surface curves more steeply in one dimension than in another, which is common around local optima. As a result, SGD tends to hover around the local optimum, but not actually reach it. Momentum helps accelerate SGD in the right direction and reduce the hovering. It does this by adding a fraction of the previous update vector to the current one. Mathematically:

$$v_t = \gamma v_{t-1} + \alpha * \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Figure 4: For learning rate α, objective-function J, parameters θ, momentum-parameter γ, and sample v_t

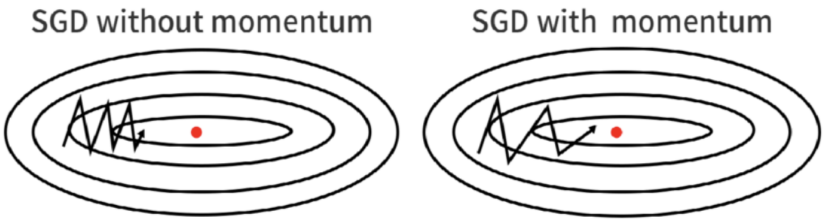

Figure 5: Left: SGD without momentum. Right:SGD with momentum.

3.4 Adaptive Optimizers

While SGD with momentum significantly sped up SGD, we are still stuck with updating all of the parameters in an identical manner. However, we would like to adapt our updates to each individual parameter and their importance; adaptive optimizers do just that. The following are some of the adaptive learners I tried, and what makes each of them unique.

3.4.1 AdaGrad

Remember how Gradient Descent updates all parameters in the same way? AdaGrad fixes that by assigning a learning-rate for each parameter independently and performs small updates for frequently occurring features and larger ones for infrequent ones. As a result, it should perform well on sparse data. Mathematically:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,ii} + \epsilon}} \nabla_\theta J(\theta_t)$$

Here, $G_t \in R^{dxd}$, is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t, while ϵ is a smoothing term that avoids division by zero.
AdaGrad's appeal is that it removes the need to manually tune the learning-rate, however, its main issue is its accumulating squared gradients in the denominator; it leads to a vanishing learning-rate which approaches to zero as the number of updates grows.

3.4.2 AdaDelta

AdaDelta addresses the vanishing learning-rate by applying a constant-size window w of past gradients and removing the learning-rate from the update rule, and as a result, the need to intelligently select one. The update rule is:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Where:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

And γ is a fraction (reminiscent of the Momentum term), $E[Delta\theta^2]_t$ is the past gradients average of size w.

3.4.3 RMSProp

RMSProp is very similar to AdaDelta, and was developed around the same time: it also solves the problem of AdaGrad's vanishing learning-rates, but here, we still need an initial learning-rate. Mathematically, the only difference is the learning rate:

$$\Delta\theta_t = -\frac{\alpha}{RMS[g]_t} \nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Where:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

3.4.4 Adam

Adaptive Moment Estimation (Adam) also computes adaptive learning-rates for each parameter and keeps an average of past gradients v_t (just like AdaDelta and RMSProp), it also keeps an average of past gradients with momentum m_t as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta_t)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta_t))^2$$

m_t is the estimate of the first moment (the mean), and v_t is the estimate of the second. To deal with the bias of the estimates towards zero, the originators adjusted the variables:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t}$$

And now, very similarly to AdaDelta and RMSProp, Adam's update rule is:

$$\theta_{t+1} = \theta_t + \frac{\alpha}{\sqrt{\hat{v_t}} + \epsilon}\hat{m_t}$$

3.4.5 Adam-W

In theory, networks with smaller weights (where all else is equal) are observed to overfit less. In this spirit, L2 regularization and weight decay are used to regularize networks. We can measure the importance of decreasing the loss by:

$$\theta_t = (1 - w)\theta_{t-1} - \alpha\nabla f[\theta_{t-1}]$$

The smaller w is, the more important it is to minimize the loss function, and the larger it is, the more emphasis put on finding small weights.
In general, weight decay and L2-regularization are equivalent.
Let us look at L2:

$$f_{reg}[\theta_{t-1}] = f[\theta_{t-1}] + \frac{w'}{2}\theta_{t-1}^2$$

And its gradient:

$$\nabla f_{reg}[\theta_{t-1}] = \nabla f[\theta_{t-1}] + w'\theta_{t-1}$$

and then update the weights:

$$\theta_t = \theta_{t-1} - \alpha\nabla f_{reg}[\theta_{t-1}]$$

$$\theta_t = \theta_{t-1} - \alpha\nabla f[\theta_{t-1}] - \alpha w'\theta_{t-1}$$

If you define $w' = \frac{w}{\alpha}$, you will see that we get weight decay.
However, the authors of Adam-W show us that for Adam, they are not equal!
The original variant of Adam applies L2-regularization in the following way:

$$\nabla_\theta J(\theta_t) = \nabla f_t(\theta_{t-1}) + w\theta_{t-1}$$

The regularization-term is added to the cost function, which is then derived to calculate the gradients. If we add the regularization-term at this point, the moving averages of the gradient and its square (m and v) also keep track of the gradients of it! Nevertheless, if we reformulate some of the algorithm to add the regularization-term:

$$\theta_t = \theta_{t-1} - \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)(\nabla f_t + w\theta_{t-1})}{\sqrt{v_t} + \epsilon}$$

Now, the regularization-term is normalized by $\sqrt{v}$ too. If the gradient of a weight is volatile, the corresponding v is volatile too, and the weight is regularized less compared to those with small and slowly changing gradients. The authors propose to decouple weight-decay and loss-based gradient updates, apply weight-decay, like in the equation above. This way, the regularization-term does not trickle into the moving averages and is thus only proportional to the weight itself.
You can also see the changes straight from the paper:

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow$ SelectBatch$(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1-\beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1-\beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow$ SetScheduleMultiplier(t) ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

4 Experimental results

Insofar, RMSprop, AdaDelta, and Adam are very similar algorithms that do well in similar circumstances. Yet, Adam's bias-correction motivated me to believe it will outperform existing methods. In that spirit, I read the Adam-W [LH19] and decided to apply an implementation I found, competing with the Keras library, and a highly optimized version of all the algorithms it implements. So I tested

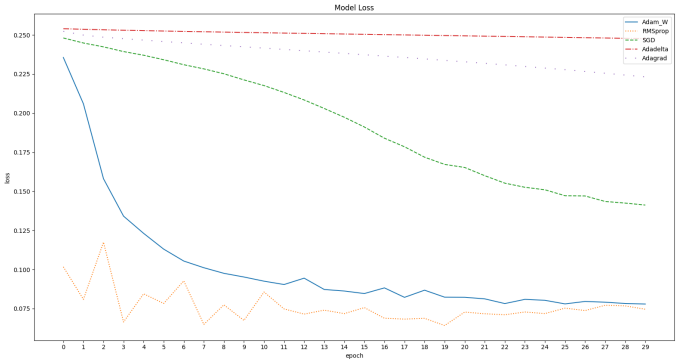

Figure 6: MSE as a loss-function

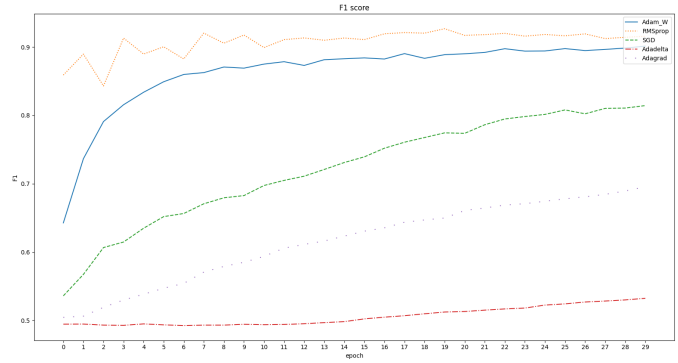

Figure 7: F1-scores

Keras' optimizers: SGD, AdaDelta, RMSProp and AdaGrad and compared it to the Adam-W I found.
In figure six and seven we can see that Adam-W nearly outperforms all Keras results, and more or less matches Keras' RMSProp. This is a great result, since Keras is a highly optimized library. Nevertheless, Keras' Adam outperforms the Adam-W (figure eight and nine), possibly because Keras does not implement the classic versions of the algorithms, but their most improved and robust versions with their tweaks.

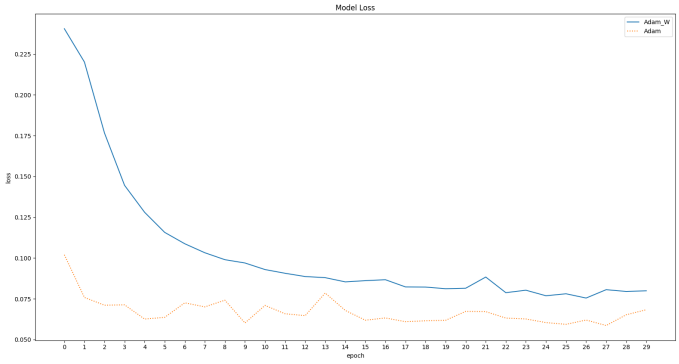

Figure 8: MSE as a loss-function

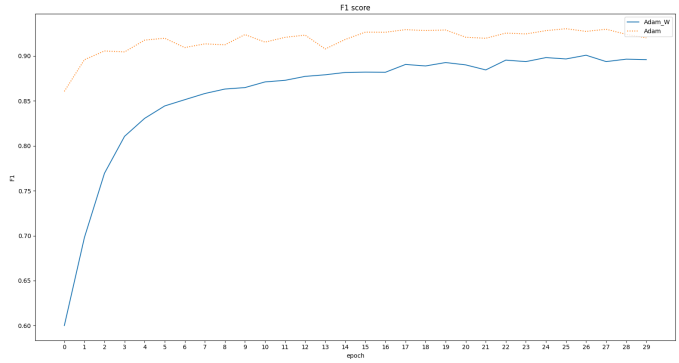

Figure 9: F1-scores

5 Discussion

While working on this project made me realize how crucial optimization techniques are, and how important it is to fit the appropriate technique to the problem at hand. The stock-market prediction is not the focus, nor the Sentiment Analysis, but I truly enjoyed working on areas I am already familiar with, but with a different perspective, one that emphasizes optimization.

Appendix

Table 1: Keywords

Bull	Bear
moon	crash
rocket	bear
diamond	red
up	down
all-in	dump
green	lose
bull	lost
rocket-emoji	loss
N/A	risk
N/A	hedge
N/A	hedgies
N/A	bad
N/A	life
N/A	false
N/A	inflation
N/A	porn loss
N/A	loss porn

Company	Text	Ground Truth	Prediction
TESLA	For those of you bullish on TSLA, are you worried about Volkswagen, GM, Hyundai going all-in on EVs in 2021?	Bear	Bear
TESLA	TSLA autonomous and robot taxi bear case	Bear	Bull
TESLA	TSLA, another bull case!	Bull	Bull
TESLA	I can't understand why apple and Tesla stock is going up wildly in recent months?? Regrets not buying it but I never know they will going up regret I don't buy the other stock either too because majority of them is going up because vaccine (crying alone in the corner of my room)	Bull	Bear

Table 2: Overall sentiment was 'Bull' and within that time-frame TESLA jumped 60%

References

[Rud16] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *ruder.io* (2016).

[LH19] Ilya Loshchilov and Frank Hutter. "DECOUPLED WEIGHT DECAY REGULARIZATION". In: *ICLR* (2019).