Can LSTM + WT + SAE get us to 82.5% annual returns on the Dow?

February 25th 2019

@janny_kul

Can a Wavelet Transform & Stacked Auto-Encoders help improve a Long Short Term Memory predictor for Equity Indices?

Photo by freestocks.org on Unsplash

I've spent a majority of my adult life in investing.

Recently I became more interested in approaching the topic from a quantitative angle. The promise of automating an investment approach whilst I sit on a beach sipping sangria's was all too compelling to ignore.

> *If I have seen further it is by* ***standing on the shoulders of Giants****.—* ***Isaac Newton***

With Sir Isaac's expression in my mind I thought what better place to start than existing research papers. I thought hopefully they'll give me some unique knowledge that I can build up on when I write my own strategies.

How wrong I was.

This is part of a multi-part series, links below:

1. *Using Artificial Intelligence to generate annual returns of 62.7%*
2. *Can LSTM + WSAE get us to 82.5% annual returns on the Dow?*

Around 6 months ago I stumbled across a research paper that on the face of it seemed very promising.

In short the technique goes something like this:

- Start with past prices for equity indices.
- Add some technical analysis.

Very familiar so far, but here's where it gets a bit fancy.

- Run this data through a Wavelet Transform.
- Then run the output of the WT through stacked auto-encoders.
- And out of this pops the magic.

The predictions are claimed to be more accurate than had you have not done any of the fancy stuff in the middle. And all we're using is past prices!

Skeptical, I delved deeper.

Wavelet Transform

The paper starts with a 2 level WT applied twice.

```
def lets_get_wavy(arr):
    level = 2
    haar = pywt.Wavelet("haar")
    coeffs = pywt.wavedec(arr, haar, level=level, mode="per")
    recomposed_return = pywt.waverec(coeffs, haar)
    sigma = mad(coeffs[-1],center=0)
    uthresh = sigma*np.sqrt(2*np.log(len(arr)))
```

```
        coeffs[1:] = ( pywt.threshold( i, value=uthresh, mode="soft" ) for i in coef
        y = pywt.waverec(coeffs, haar, mode="per" )
        return y
```

Now there isn't really a clear mention in the paper as to if a wavelet transform is applied to just the close price, or to every input time series separately. They use the phrase "*multivariate denoising using wavelet*" which I'd assume to mean it was applied to every time series. To be safe I tried both methods.

Thankfully the issue starts to become quite apparent from here.

I'm sure you've heard many times that whenever you're normalising a time series for a ML model to fit your normaliser on the *train set* first then apply it to the *test set*. The reason is quite simple, our ML model behaves like a mean reverter so if we normalise our entire dataset in one go we're basically giving our model the mean value it needs to revert to. I'll give you a little clue, if we knew the future mean value for a time series we wouldn't need machine learning to tell us what trades to do ;)

So back to our wavelet transform. Take a look at this line.

```
sigma = mad(coeffs[-1],center=0)
```

So we're calculating the mean absolute deviation across the noisy coefficient. Then..

```
(pywt.threshold( i, value=uthresh, mode="soft") for i in coeffs[1:])
```

We're thresholding the **entire time series** with *uthresh* derived from our sigma value.

Notice something a little bit wrong with this?

It’s basically the exact same issue as normalising your train and test set in one go. You’re leaking future information into each time step and not even in a small way. In fact you can run a little experiment yourself; the higher a level wavelet transform you apply, miraculously the more “accurate” your ML model’s output becomes.

Using a basic LSTM classification model without WT will get you directional accuracy numbers just over 50%, but applying a WT across the whole time series will erroneously give you accuracy numbers in the mid to high 60's.

I thought perhaps I’ve misinterpreted the paper. Perhaps what they did was apply the WT across each time step before feeding data into the LSTM.

So, I tried that.

Yep, accuracy dips below 50%.

We don’t even need to go as far as the auto-encoder part to figure out a pretty huge mistake that’s been made here.

We’re here though so we might as well finish up to be sure.

Stacked Auto Encoders

Stacked auto-encoders are intended to “denoise” our data with a higher level representation. The number of output nodes you give each level will force our data into a less dimensions, with some loss. I’m not entirely sure how this would ever help our LSTM make predictions; ultimately all we’re really doing here is just removing more data that might have been useful to uncover patterns.

Your results won’t vary much here if you roll with stacked auto-encoder’s with greedy layer-wise training or just multi-layer auto-encoder’s.

Going down the stacked auto-encoder route you can build each layer up like so:

```
def build_model(self,input_shape):
        input_layer   = Input(shape=(1, input_shape))
        encoded_layer = Dense(self.encoding_shape,
            activation="relu",
            activity_regularizer=regularizers.l2(0))(input_layer)
        encoded_layer_bn = BatchNormalization()(encoded_layer)
        output_layer  = Dense(input_shape,
            activation="sigmoid",
            activity_regularizer=regularizers.l2(0))(encoded_layer_bn)
        self.autoencoder = Model(inputs=input_layer,
            outputs=output_layer)
        self.encoder = Model(input_layer,
            encoded_layer_bn)
        self.autoencoder.compile(loss="mean_squared_error",
            optimizer="adam")
```

Then just fit and predict layer by layer until you're 5 layers deep like the paper suggests.

The accuracy suffers a bit when using our leaked WT as input but is still erroneously much better than using LSTM's alone, which would explain why the paper demonstrated such good results.

The end lesson here is clear, much like Frankenstein's monster, piecing together random pages from a stats text book isn't going to help us when we're still only passing in past price data. The old adage comes true once more—if it looks too good to be true, it probably is.

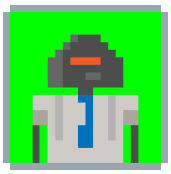
Side note, new project I'm working on:

🦉 **Stocktive | Stock analysis for the active investor**
*Stocktive gives you daily stock analysis for publicly traded companies for free.*www.stocktive.com

Disclaimer

This doesn't constitute as investment advice. Seek advice from an authorised financial advisor before making any investments. Past performance is not indicative of future returns.

Share this story



@janny_kul

Read my stories

Comments

What do you think?

0 Responses

Upvote LOL Love On fire Meh

Clap

0 Comments Hacker Noon Privacy Policy 1 Login

Recommend Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

Subscribe Do Not Sell My Data

Related

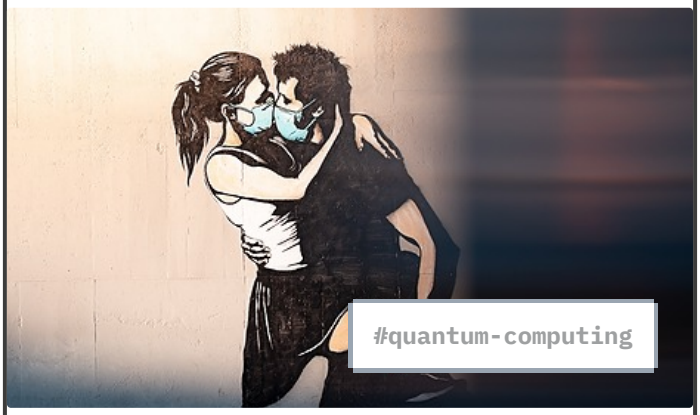
Deep Learning A-Z [Get 96% Off]

Udemy

Visit Udemy

promoted

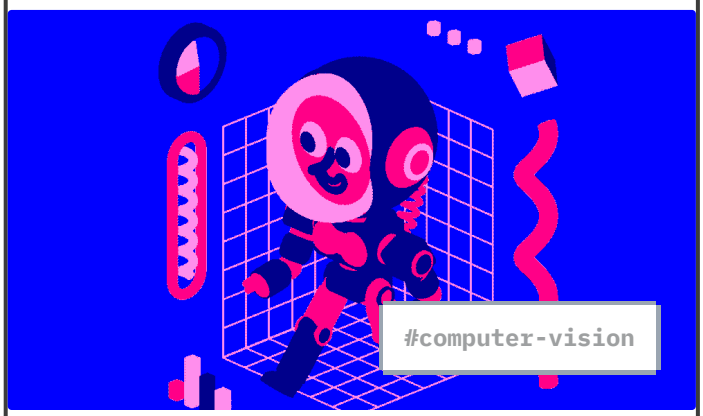
Pharmacelera's Quantum Drug Discovery Remedy



@jamesdargan
James Dargan

5min
04/06/20

Computer Vision Is Solving Problems That Weren't Even On Our List



@abhishek.deltech
Abhishek Kumar

5min
04/06/20

Tags

#machine-learning #investing #data-science #artificial-intelligence #stock-market

THE NOON NOTIFICATION

Subscribe to get your daily round-up of top tech stories!