Augmenting time series datasets via latent space sampling with applications in algorithmic trading

Andrei Bratu
Babeș-Bolyai University

WeADL 2021 Workshop

The workshop is organized under the umbrella of WeaMyL, project funded by the EEA and Norway Grants under the number RO-NO-2019-0133. Contract: No 26/2020.

Working together for a green, competitive and inclusive Europe

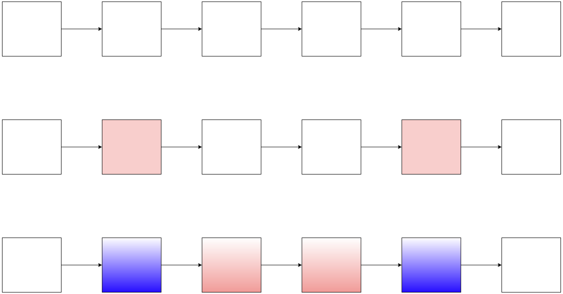
Outline

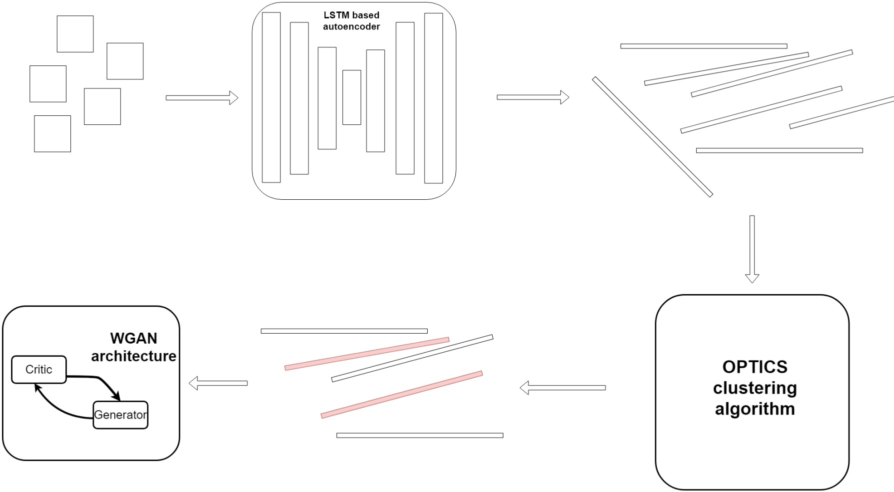
Background

- Securities trading (e.g. stocks, options, cryptocurrencies) is increasingly an automatic, algorithmic-driven field. Three out of four foreign currency exchange trades are automatic [1]
- There is a strong interest in applying deep learning and reinforcement learning to automatic trading, moving away from euristhics [2, 3]
- Eternal struggle of a data scientist: more quality data!

Approach

Autoencoder architecture is able to capture and model timeseries in latent representation. [6]




LSTM based
autoencoder
OPTICS
clustering
algorithm
WGAN
architecture
Critic
Generator

Exploring Data

We focus on the AAPL stock price, sampled at interval of 15 minutes

	date	time	open	high	low	close	volume
0	01/02/1998	09:30	13.6250	13.7500	13.5000	13.6875	202700
1	01/02/1998	09:45	13.6875	13.7500	13.5000	13.6250	334000
2	01/02/1998	10:00	13.6250	13.7500	13.5625	13.7500	299900
3	01/02/1998	10:15	13.7500	14.0000	13.6250	14.0000	430201
4	01/02/1998	10:30	13.9375	14.8125	13.7500	14.6250	944200
...	...	...	...	...	...	...	...
289482	03/12/2021	18:45	120.9900	121.1000	120.9700	121.0900	15752
289483	03/12/2021	19:00	121.0600	121.1000	120.9900	121.0000	19160
289484	03/12/2021	19:15	121.0000	121.0700	120.9900	121.0300	13815
289485	03/12/2021	19:30	121.0100	121.0300	121.0000	121.0300	6903
289486	03/12/2021	19:45	121.0300	121.1000	121.0200	121.0800	33259

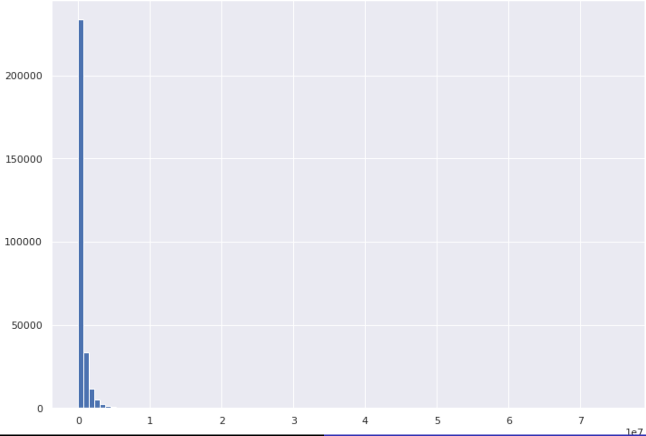
289487 rows × 7 columns

Exploring Data

	open	high	low	close	volume
count	289487.000000	289487.000000	289487.000000	289487.000000	2.894870e+05
mean	187.757776	188.038462	187.466230	187.758491	4.580510e+05
std	160.514466	160.689723	160.326415	160.513598	9.206810e+05
min	12.550000	12.950000	11.312500	12.850000	1.000000e+02
25%	78.750000	78.920000	78.500000	78.750000	5.900000e+03
50%	131.040000	131.330000	130.790000	131.040000	1.033630e+05
75%	248.160000	248.605000	247.625000	248.150000	5.635505e+05
max	704.800000	705.070000	704.530000	704.800000	7.514145e+07

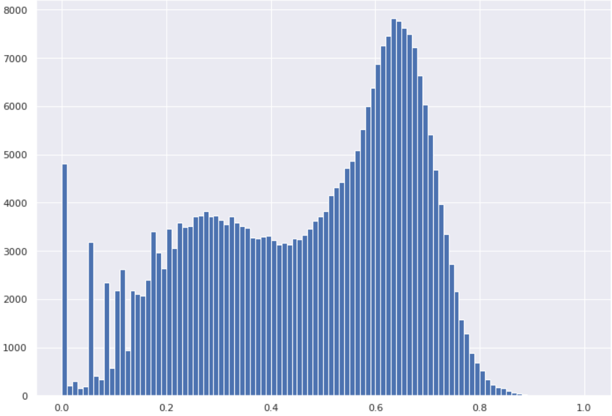
Exploring Data

'Volume' histogram before normalization



Exploring Data

'Volume' histogram after normalization



Exploring Data

- Split the dataset into TIME_STEPS x 5 chunks
- Two consecutive chunks will have TIME_STEPS_COMMON common time steps
- We obtain a dataset of roughly 25000 points

Autoencoder

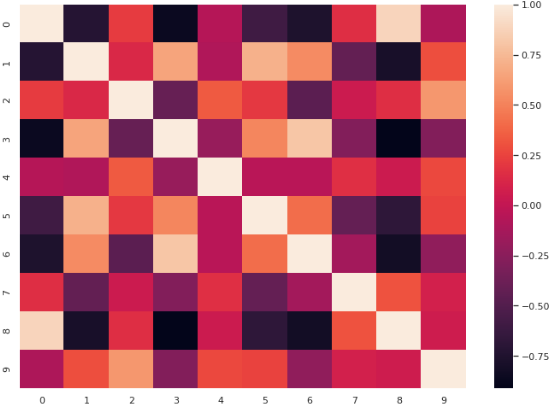
The autoencoder includes best practices such as Dropout layers and LeakyReLU activations [4, 5]

```
Model: "encoder"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_3 (InputLayer)        [(None, 20, 5)]           0
_________________________________________________________________
conv1d_3 (Conv1D)           (None, 20, 32)            672
_________________________________________________________________
leaky_re_lu_10 (LeakyReLU)  (None, 20, 32)            0
_________________________________________________________________
dropout_10 (Dropout)        (None, 20, 32)            0
_________________________________________________________________
conv1d_4 (Conv1D)           (None, 20, 64)            8256
_________________________________________________________________
leaky_re_lu_11 (LeakyReLU)  (None, 20, 64)            0
_________________________________________________________________
dropout_11 (Dropout)        (None, 20, 64)            0
_________________________________________________________________
conv1d_5 (Conv1D)           (None, 20, 128)           32896
_________________________________________________________________
leaky_re_lu_12 (LeakyReLU)  (None, 20, 128)           0
_________________________________________________________________
dropout_12 (Dropout)        (None, 20, 128)           0
_________________________________________________________________
lstm_5 (LSTM)               (None, 20, 128)           131584
_________________________________________________________________
leaky_re_lu_13 (LeakyReLU)  (None, 20, 128)           0
_________________________________________________________________
dropout_13 (Dropout)        (None, 20, 128)           0
_________________________________________________________________
lstm_6 (LSTM)               (None, 20, 64)            49408
_________________________________________________________________
leaky_re_lu_14 (LeakyReLU)  (None, 20, 64)            0
_________________________________________________________________
dropout_14 (Dropout)        (None, 20, 64)            0
_________________________________________________________________
lstm_7 (LSTM)               (None, 10)                3000
=================================================================
Total params: 225,816
Trainable params: 225,816
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "decoder"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
input_4 (InputLayer)        [(None, 10)]              0
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 20, 10)           0
_________________________________________________________________
lstm_8 (LSTM)               (None, 20, 64)            19200
_________________________________________________________________
leaky_re_lu_15 (LeakyReLU)  (None, 20, 64)            0
_________________________________________________________________
dropout_15 (Dropout)        (None, 20, 64)            0
_________________________________________________________________
lstm_9 (LSTM)               (None, 20, 128)           98816
_________________________________________________________________
leaky_re_lu_16 (LeakyReLU)  (None, 20, 128)           0
_________________________________________________________________
dropout_16 (Dropout)        (None, 20, 128)           0
_________________________________________________________________
conv1d_transpose_3 (Conv1DTr (None, 20, 128)          65664
_________________________________________________________________
leaky_re_lu_17 (LeakyReLU)  (None, 20, 128)           0
_________________________________________________________________
dropout_17 (Dropout)        (None, 20, 128)           0
_________________________________________________________________
conv1d_transpose_4 (Conv1DTr (None, 20, 64)           32832
_________________________________________________________________
leaky_re_lu_18 (LeakyReLU)  (None, 20, 64)            0
_________________________________________________________________
dropout_18 (Dropout)        (None, 20, 64)            0
_________________________________________________________________
conv1d_transpose_5 (Conv1DTr (None, 20, 32)           8224
_________________________________________________________________
leaky_re_lu_19 (LeakyReLU)  (None, 20, 32)            0
_________________________________________________________________
dropout_19 (Dropout)        (None, 20, 32)            0
_________________________________________________________________
time_distributed_2 (TimeDist (None, 20, 5)            165
_________________________________________________________________
time_distributed_3 (TimeDist (None, 20, 5)            0
=================================================================
Total params: 224,901
Trainable params: 224,901
Non-trainable params: 0
_________________________________________________________________
```

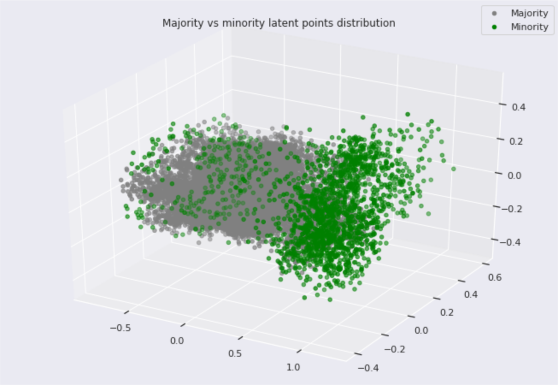
Latent Space

Hyperparameter tweaking concluded a latent space of dimension 10 offers the best loss - information tradeoff.



Latent Space

A "majority" cluster can be observed; 3-dim PCA visualization



WGAN

WGAN is an improvement over first-generation GAN architecture [7]

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 15, 1)]           0
_________________________________________________________________
dense_4 (Dense)              (None, 15, 20)            40
_________________________________________________________________
conv1d_9 (Conv1D)            (None, 8, 16)             1296
_________________________________________________________________
leaky_re_lu_23 (LeakyReLU)   (None, 8, 16)             0
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 4, 16)             1040
_________________________________________________________________
leaky_re_lu_24 (LeakyReLU)   (None, 4, 16)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 100)               6500
_________________________________________________________________
dense_6 (Dense)              (None, 100)               10100
_________________________________________________________________
dense_7 (Dense)              (None, 10)                1010
=================================================================
Total params: 19,986
Trainable params: 19,986
Non-trainable params: 0
_________________________________________________________________
```

```
Model: "critic"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 10, 1)]           0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 5, 16)             80
_________________________________________________________________
leaky_re_lu_20 (LeakyReLU)   (None, 5, 16)             0
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 3, 16)             1040
_________________________________________________________________
leaky_re_lu_21 (LeakyReLU)   (None, 3, 16)             0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 2, 16)             1040
_________________________________________________________________
leaky_re_lu_22 (LeakyReLU)   (None, 2, 16)             0
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               3300
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 5,561
Trainable params: 5,561
Non-trainable params: 0
_________________________________________________________________
```

Generated latent points

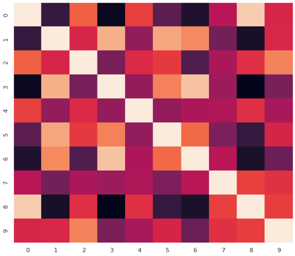
Figure: WGAN is able to generalize new interesting points in latent space

Generated latent points

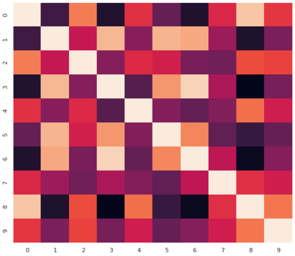
Multivariate Wilcoxon test [8] indicates that the generated points are actually different from non-majority points (p=0.0186)* Figure:

The correlation maps for minority and artificial are different:

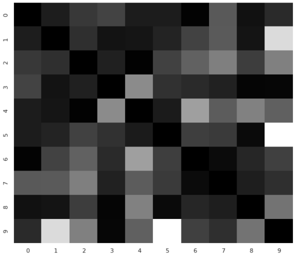
Minority



Artificial



Absolute difference



They're not even half bad!

	open	high	low	close	volume
0	15.243655	15.253026	15.194520	15.253026	120136.226562
1	15.228220	15.247979	15.180886	15.247979	99446.000000
2	15.276122	15.305717	15.231302	15.305717	66718.484375
3	15.385123	15.418838	15.341538	15.418838	54139.515625
4	15.219886	15.254024	15.176585	15.254024	57980.453125
5	14.824146	14.858824	14.782189	14.858824	53399.527344
6	14.866426	14.903136	14.825564	14.903136	43166.875000
7	14.899176	14.949899	14.864511	14.949899	12371.279297
8	14.692458	14.761474	14.666427	14.761474	2728.982178
9	14.716473	14.739958	14.672203	14.739958	73965.726562
10	14.880865	14.886304	14.829742	14.886304	198242.906250
11	14.941760	14.950253	14.890265	14.950253	230313.250000
12	14.908655	14.916911	14.857006	14.916911	241238.500000
13	14.926683	14.935029	14.875145	14.935029	235060.812500
14	14.872905	14.882419	14.822306	14.882419	203394.796875
15	14.815712	14.825365	14.765698	14.825365	182581.171875
16	14.743999	14.753530	14.694370	14.753530	170796.218750
17	15.097682	15.106466	15.046432	15.106466	193027.968750
18	15.156822	15.158875	15.101473	15.158875	382102.812500
19	15.353157	15.355431	15.299351	15.355431	394814.906250

	open	high	low	close	volume
0	328.285400	328.611328	327.923828	328.611328	99.999954
1	326.386292	326.535126	326.086853	326.535126	783.815796
2	330.967285	331.169830	330.576172	331.169830	1070.473511
3	332.969727	333.235931	332.445892	333.235931	1950.821655
4	333.611328	333.698517	333.305084	333.698517	468.448151
5	324.876434	325.506165	324.311035	325.506165	99.999954
6	329.570251	329.617706	329.417419	329.617706	302.992767
7	337.244415	337.603516	336.614471	337.603516	3137.767090
8	340.645233	341.047668	339.870667	341.047668	4611.264160
9	340.881500	341.277863	340.124542	341.277863	4799.050781
10	337.815125	338.226166	337.060516	338.226166	5621.359863
11	338.466064	338.895782	337.692413	338.895782	6761.752930
12	337.389343	337.804352	336.644196	337.804352	6170.877441
13	338.481079	338.867645	337.771973	338.867645	4593.961426
14	339.863068	340.254822	339.151520	340.254822	4806.296387
15	341.079346	341.704498	340.060669	341.704498	32221.625000
16	344.108734	344.690247	343.121704	344.690247	20956.394531
17	343.006866	343.571442	342.031799	343.571442	19307.960938
18	347.626007	348.201691	346.623596	348.201691	20125.775391
19	347.505188	348.110840	346.366669	348.110840	21908.580078

	open	high	low	close	volume
0	21.104191	21.112183	21.044468	21.112183	137670.421875
1	20.982815	20.992153	20.921492	20.992153	223587.109375
2	20.982824	20.994345	20.921881	20.994345	277790.656250
3	20.968788	20.981316	20.907495	20.981316	353173.406250
4	20.732935	20.743990	20.669832	20.743990	432019.812500
5	20.392403	20.404743	20.330650	20.404743	486898.906250
6	20.264952	20.278585	20.204550	20.278585	504092.031250
7	20.353367	20.368673	20.294113	20.368673	510079.812500
8	20.313545	20.329094	20.255043	20.329094	479180.218750
9	20.130693	20.150333	20.076599	20.150333	462485.562500
10	20.105383	20.132767	20.057438	20.132767	541669.125000
11	20.014164	20.022221	19.955524	20.022221	188173.484375
12	19.573996	19.652031	19.541817	19.652031	2278.294922
13	19.437897	19.477896	19.390755	19.477896	19736.904297
14	19.795088	19.859772	19.757679	19.859772	4509.490723
15	19.569996	19.711926	19.566645	19.711926	99.999954
16	19.494667	19.596134	19.474115	19.596134	520.183594
17	20.074871	20.155561	20.044037	20.155561	1793.617188
18	19.989128	20.065826	19.955566	20.065826	2648.292969
19	19.589943	19.685362	19.566496	19.685362	690.475647

Smoothing the points

- GAN network samples from poorly-represented regions since it maximizes critic's confusion. It is unable to represent constraints e.g. $high_t$ should be larger than col_t, $\forall col \in \{high, open, close, low\}$, $close_t == open_{t+1}$ for any timestep t
- We employ Bayesian search to find the closest latent point that satisfies the constraints. Formally for each tuple $(open_t, high_t, low_t, close_t) = ts_t$ of an arbitrary chunk we identify the maximum of a black box function f which is defined as: $\begin{cases} -\infty & invalid \\ \frac{1}{\|ts_t - ts_{gen}\|} & valid \end{cases}$, minimising the distance over all timesteps.

Integrating the points

- For each sample from the generated ones, ts_{fake} we find two consecutive time steps, $ts1$ and $ts2$, such that we minimize $\|\text{close}_{ts1} - open_{tsfake}\|, \|close_{tsfake} - open_{ts2}\|, \|\mu(volume_{ts1}) - \mu(volume_{tsfake})\|, \|\mu(volume_{tsfake}) - \mu(volume_{ts2})\|$
- Each integrated fake sample is assumed to be real after integration. Thus it is possible to have consecutive fake chunks

Results

- We employ a benchmark trading algorithm [9] based on reinforcement learning and apply it over both datasets
- We observe a 5-7% percent improvement of the algorithm's performace. These are preliminary results.
- We conclude that data augmentation is pheasible on timeseries datasets. We hypothesise that our approach can identify poorly represented intervals of a timeseries dataset.

QA time!

References

- Pratik Mulay, Nishant Poojary, Dr. Pravin Srinath *Automated Trading System - A Survey*. International Research Journal of Engineering and Technology, Aug 2016
- Yang, Hongyang, et al *Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy*. 3690996, Social Science Research Network, Sept 2020.
- Théate, Thibaut, and Damien Ernst *An Application of Deep Reinforcement Learning to Algorithmic Trading.* ArXiv:2004.06627, Oct 2020

References

- Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* arXiv:1207.0580
- Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li *Empirical Evaluation of Rectified Activations in Convolutional Network* arXiv:1505.00853
- Neda Tavakoli, Sima Siami-Namini, Mahdi Adl Khanghah, Fahimeh Mirza Soltani, Akbar Siami Namin *Clustering Time Series Data through Autoencoder-based Deep Learning Models* arXiv:2004.07296
- Martin Arjovsky, Soumith Chintala, Léon Bottou *Wasserstein GAN* arXiv:1701.07875
- Ching-Fan Sheu, Suzanne O'Curry *Implementation of*