UNIVERSITY OF TEHRAN

COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NEURAL NETWORK & DEEP LEARNING

STOCK MARKET PREDICTION USING RNN, LSTM, GRU

SIAVASH SHAMS

MOHAMMAD HEYDARI

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

UNIVERSITY OF TEHRAN

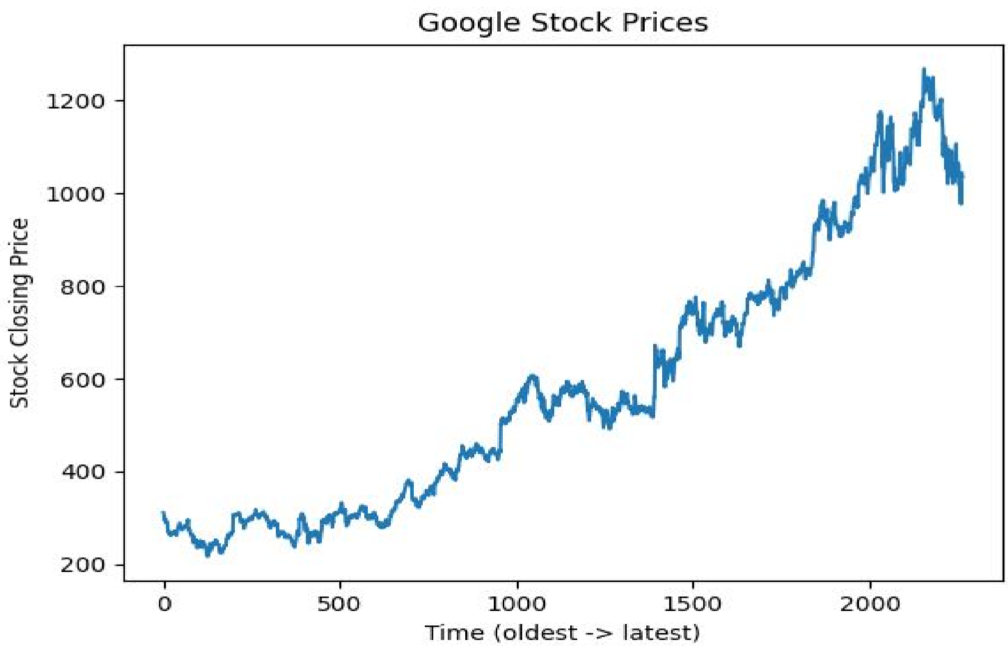
May. 2022

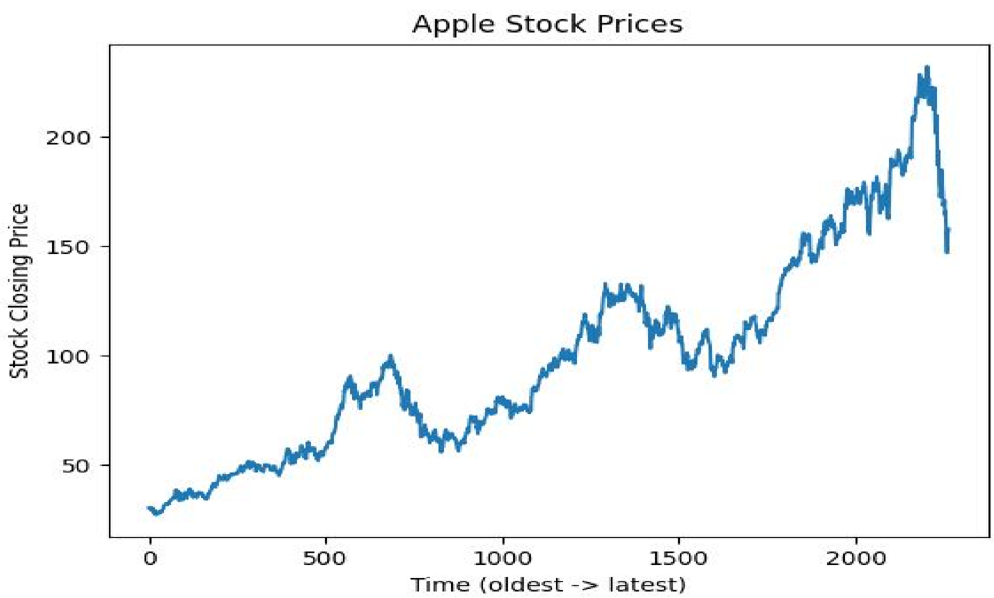
1 CONTENTS

STOCK MARKET PREDICTION

1.1

Below we can see Google and Apple, day to day stock closing prices from 2010 to 2018.



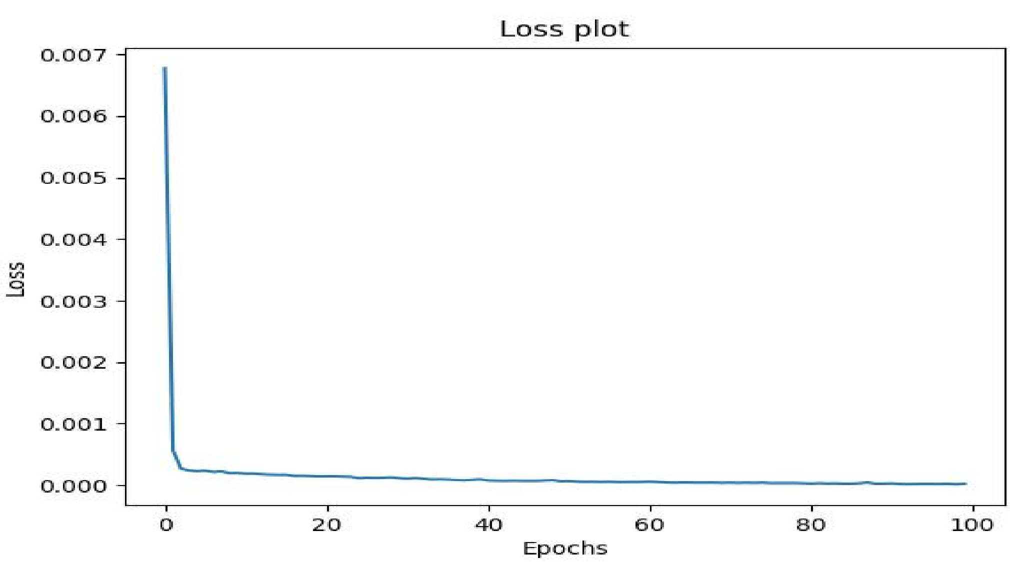


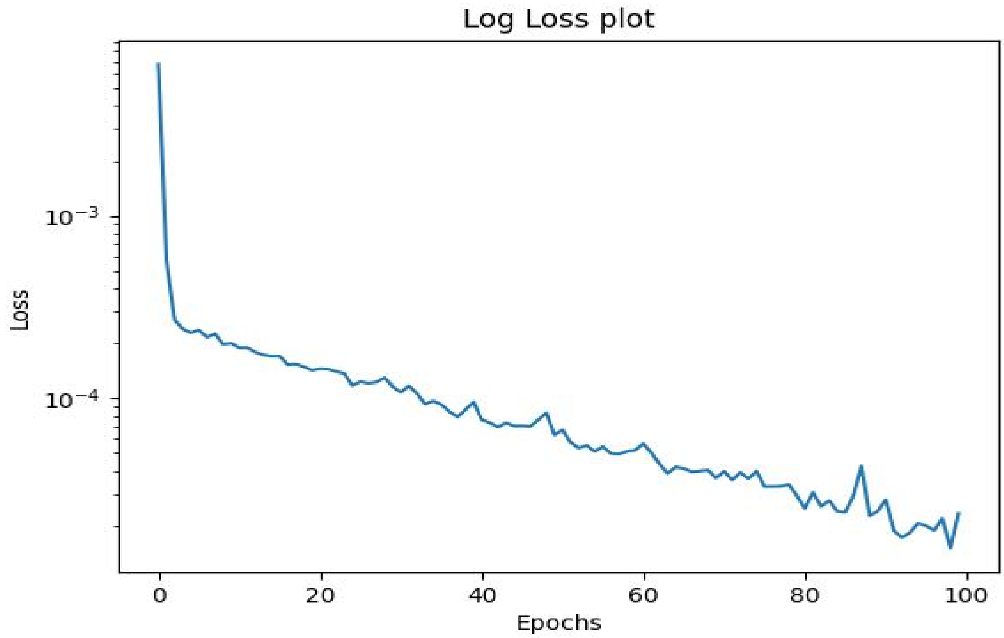
- **LSTM**

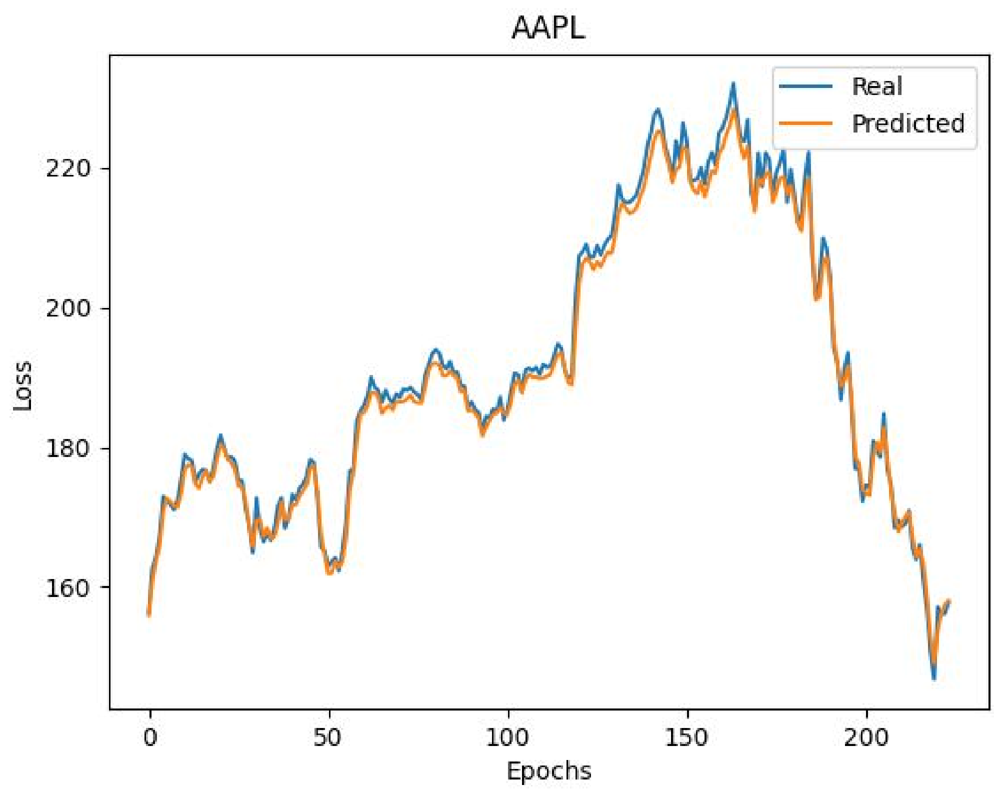
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 30, 64)            19712
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                33024
_________________________________________________________________
dense (Dense)                (None, 2)                 130
=================================================================
Total params: 52,866
Trainable params: 52,866
Non-trainable params: 0
```

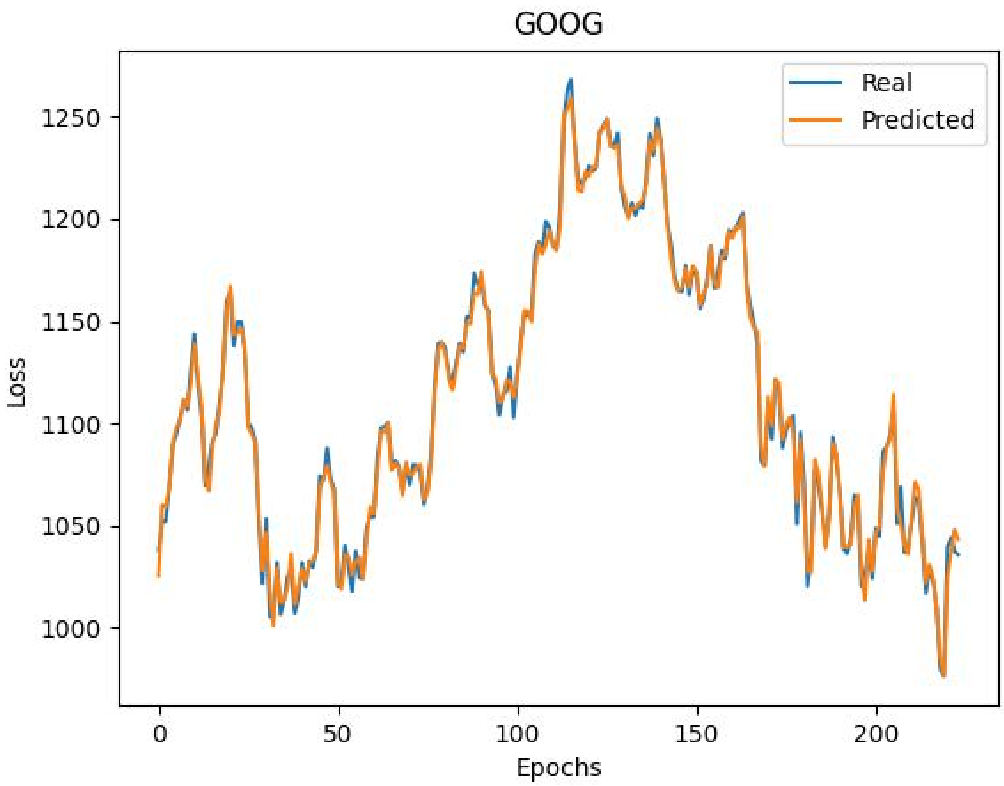
We will train our model using Adam optimizer with 100 epochs.

```
Total training time: 106.46464157104492 seconds
```




Log Loss plot
Loss
Epochs
10^{-3}
10^{-4}
0
20
40
60
80
100

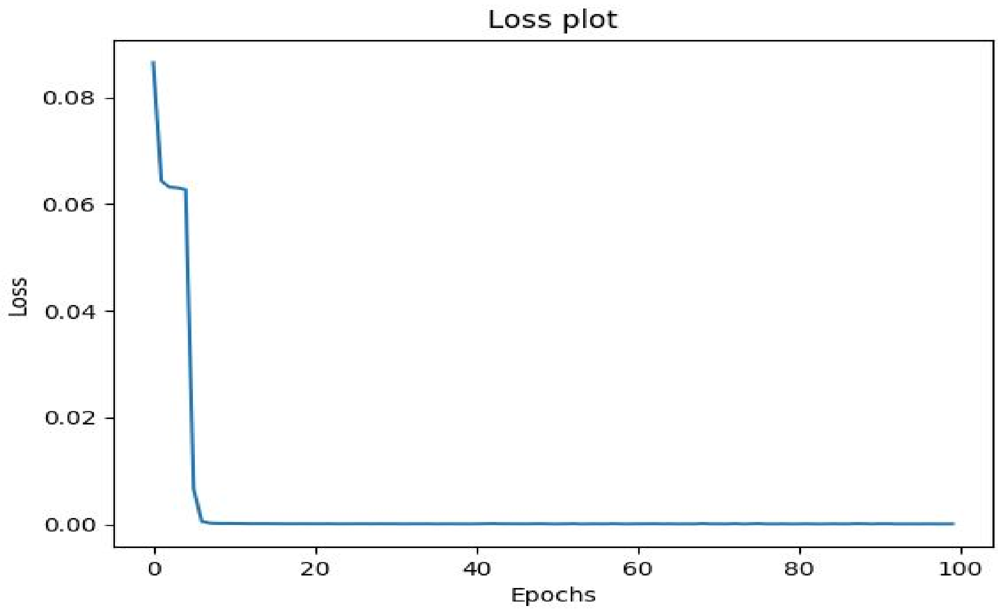

AAPL
Real
Predicted
Loss
Epochs
160
180
200
220
0
50
100
150
200

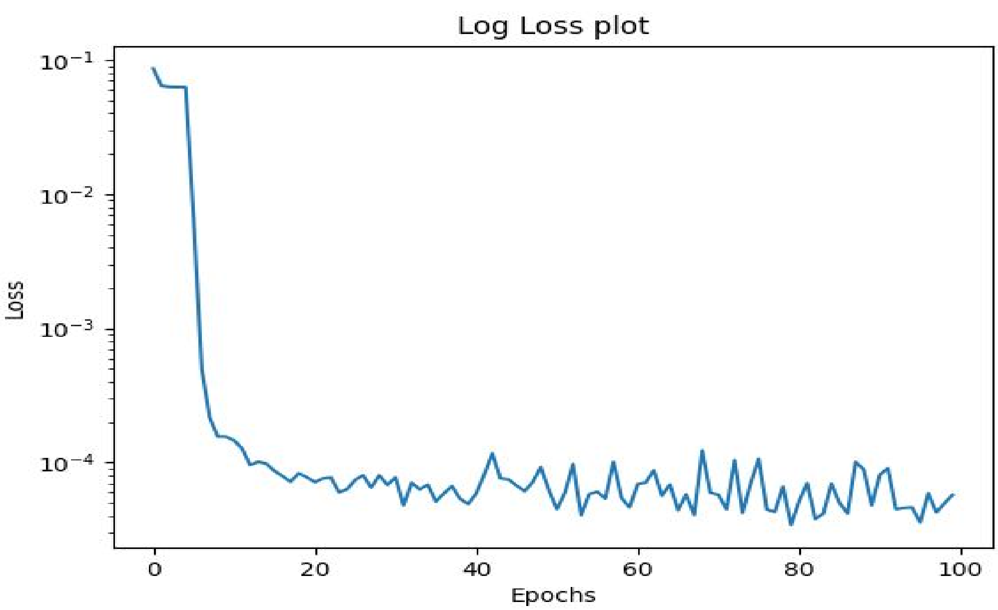


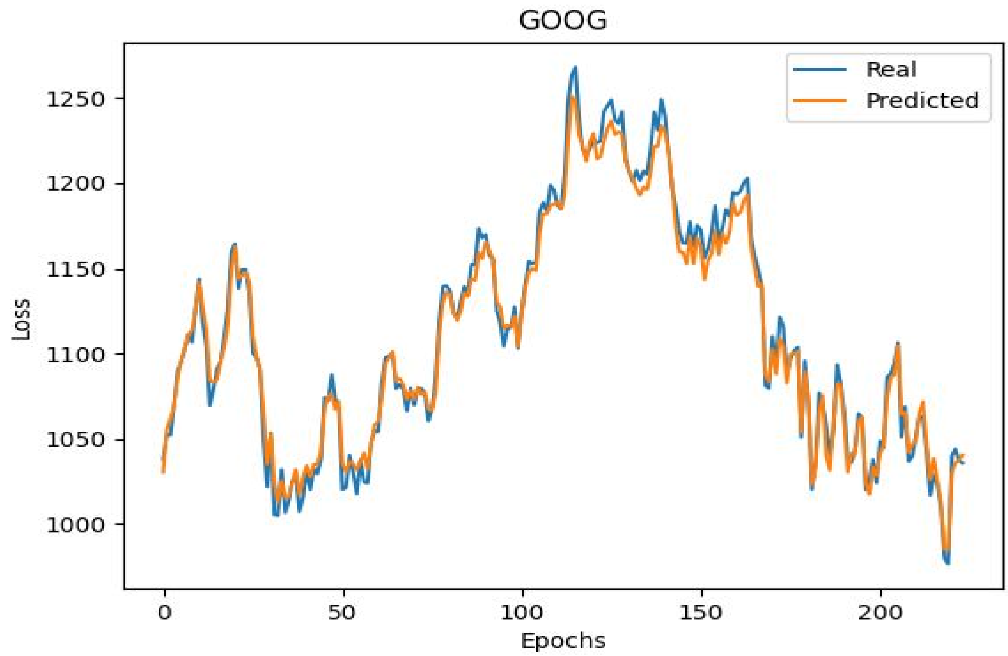
- **RNN**

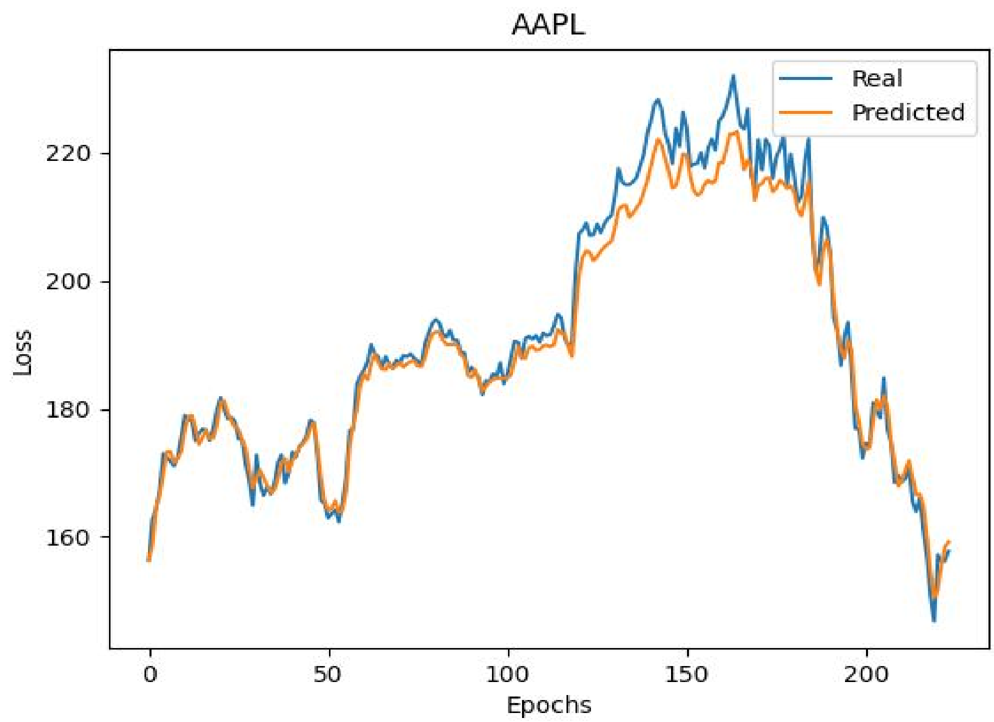
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 30, 64)            4928
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 13,314
Trainable params: 13,314
Non-trainable params: 0
```

```
Total training time: 26.431206941604614 seconds
```





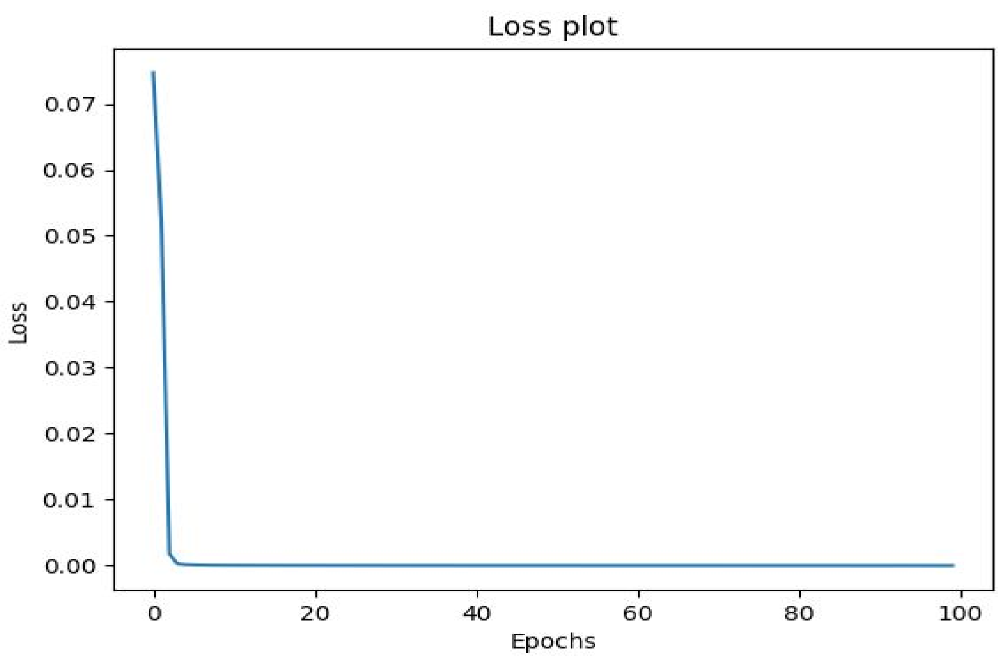

GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200

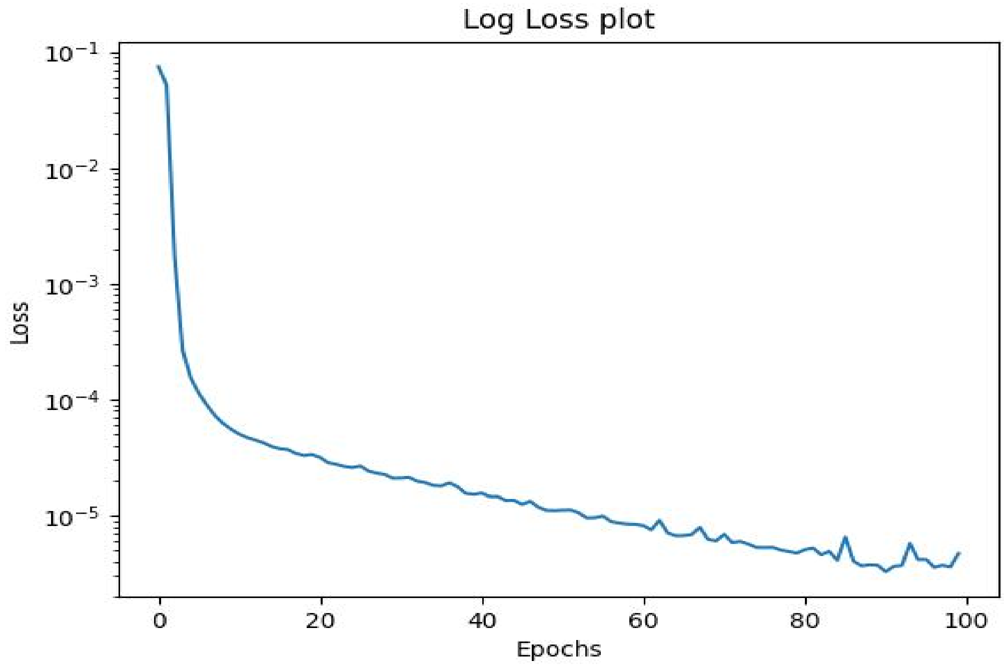

AAPL
Real
Predicted
Loss
Epochs
220
200
180
160
0
50
100
150
200

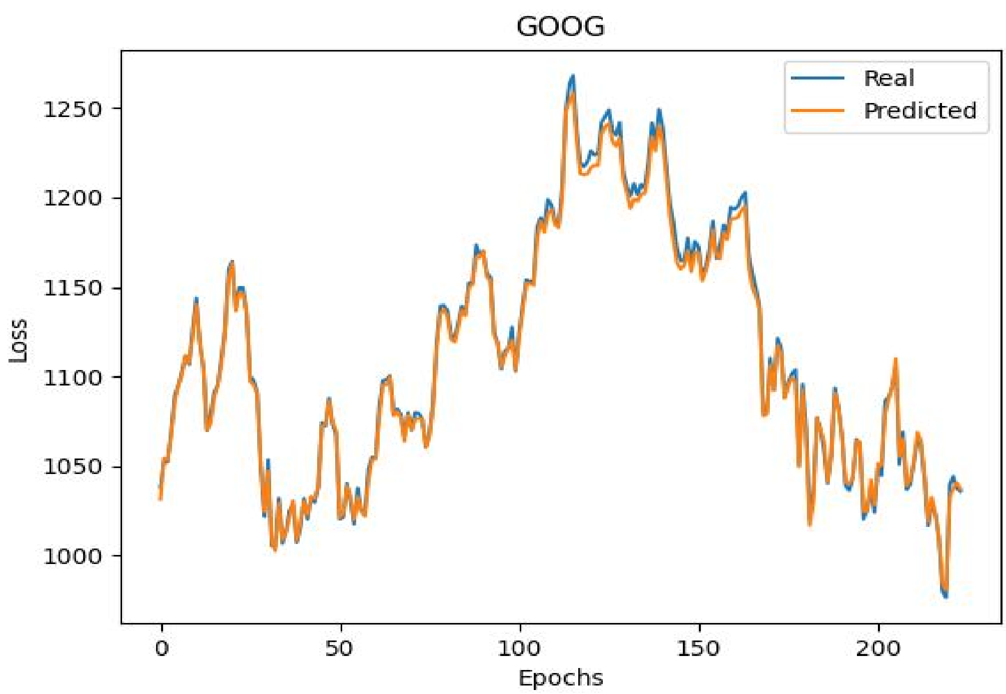
- **GRU**

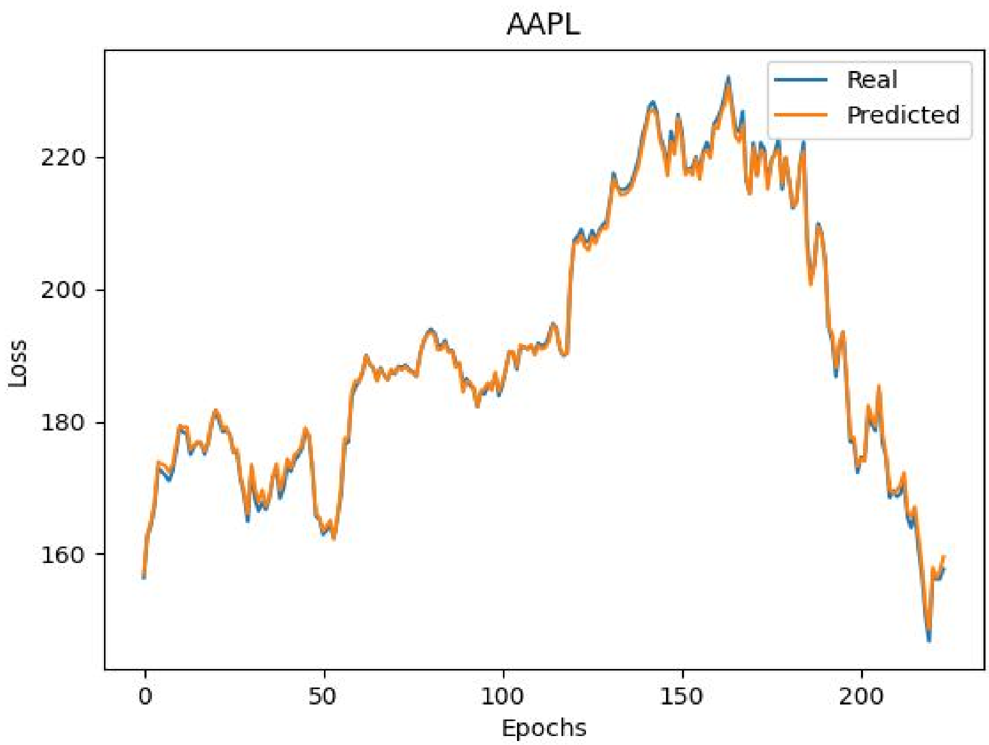
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 30, 64)            14976
_________________________________________________________________
gru_1 (GRU)                  (None, 64)                24960
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 40,066
Trainable params: 40,066
Non-trainable params: 0
```

```
Total training time: 81.92196726799011 seconds
```




Log Loss plot
Loss
Epochs
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
20
40
60
80
100


GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200



By looking at the results we conclude that GRU outperforms the other two memory cells

Performance: $GRU > LSTM > RNN$

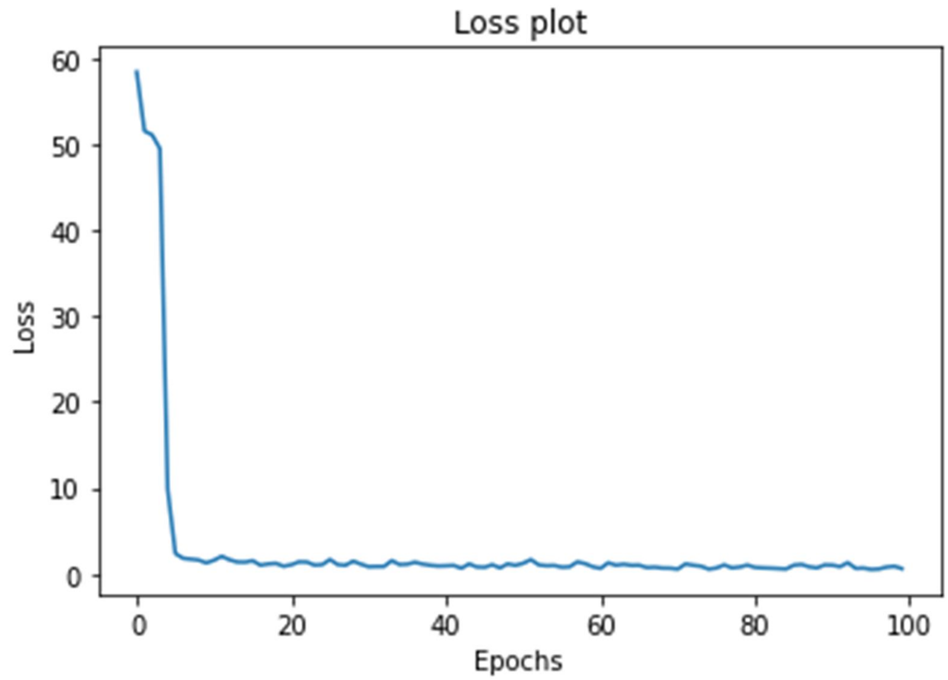
also training time is least for RNN and GRU comes second, and LSTM takes the longest. We can see that as the memory unit complexity (number of parameters) gets bigger, it requires more training time.

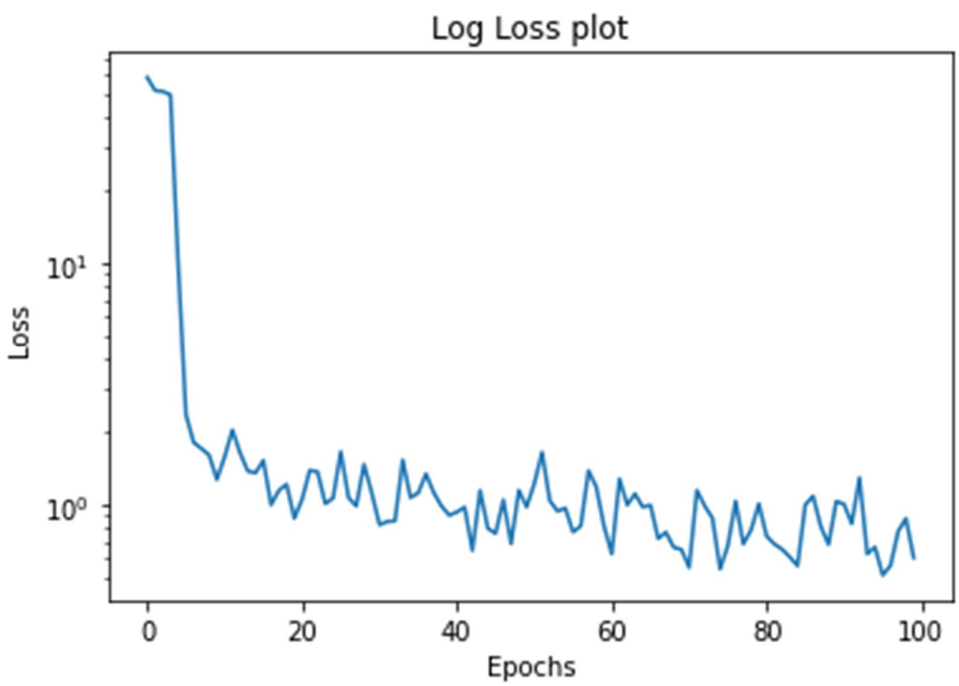
Training time: $LSTM > GRU > RNN$

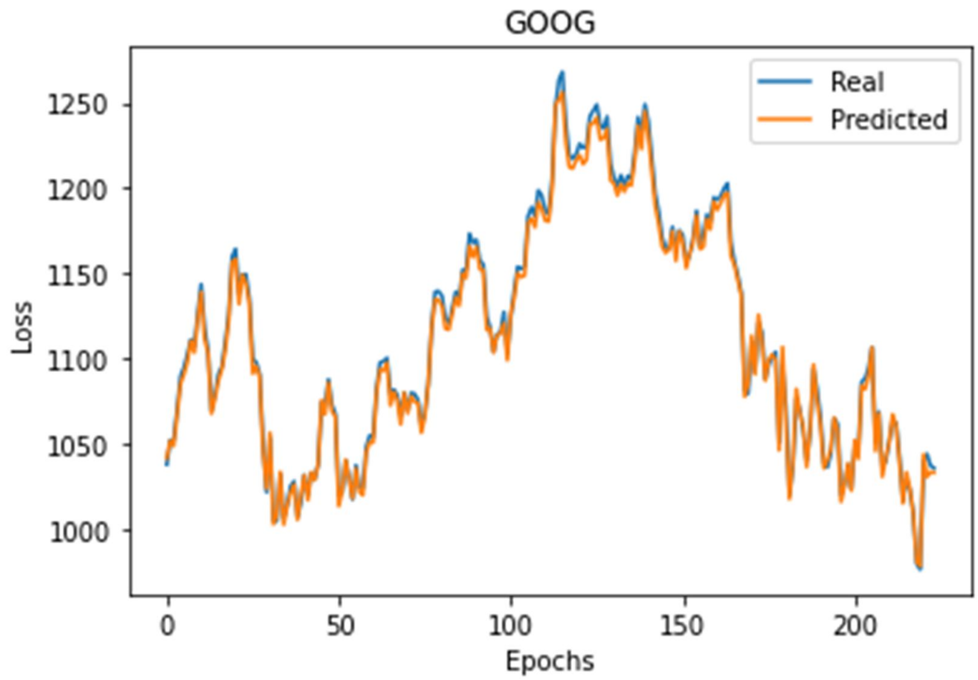
1.2

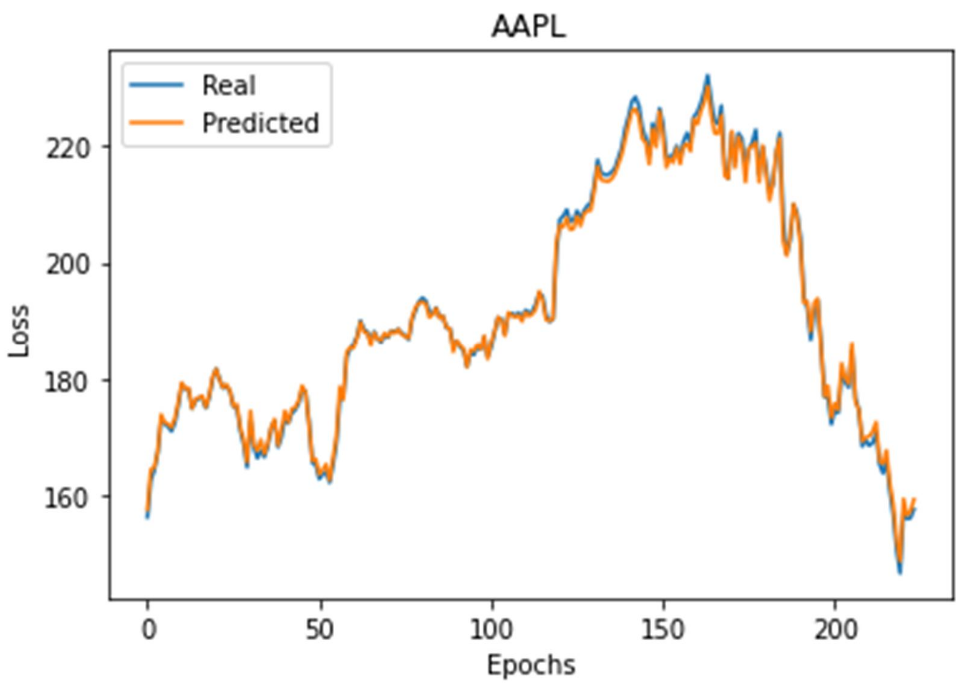
- GRU + MAPE

```
Total training time: 145.55148577690125 seconds
```


Loss plot
Loss
Epochs
0
10
20
30
40
50
60
0
20
40
60
80
100

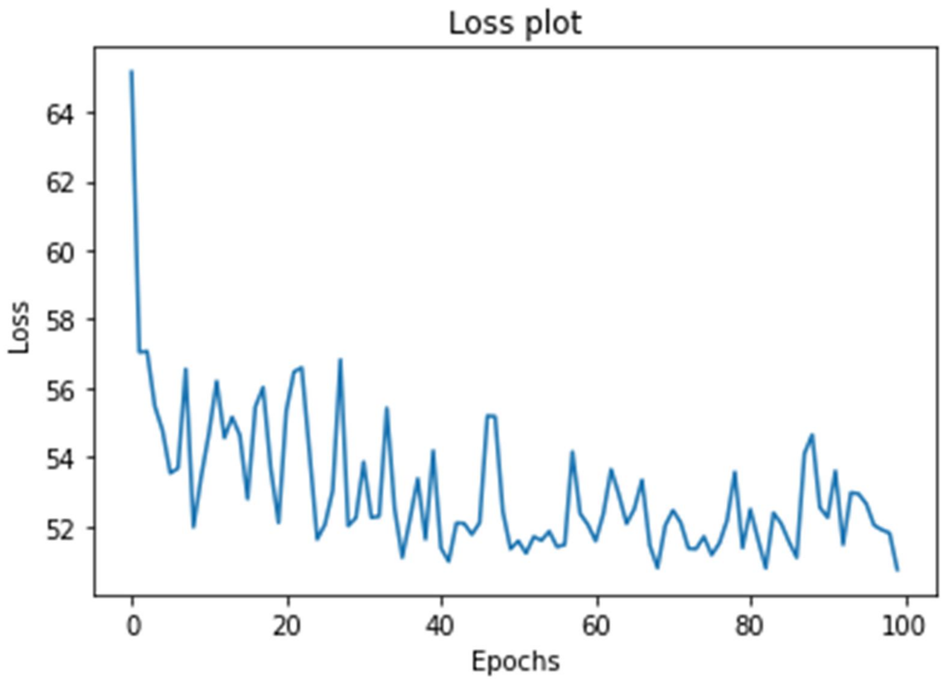

Log Loss plot
Loss
Epochs
10^1
10^0
0
20
40
60
80
100

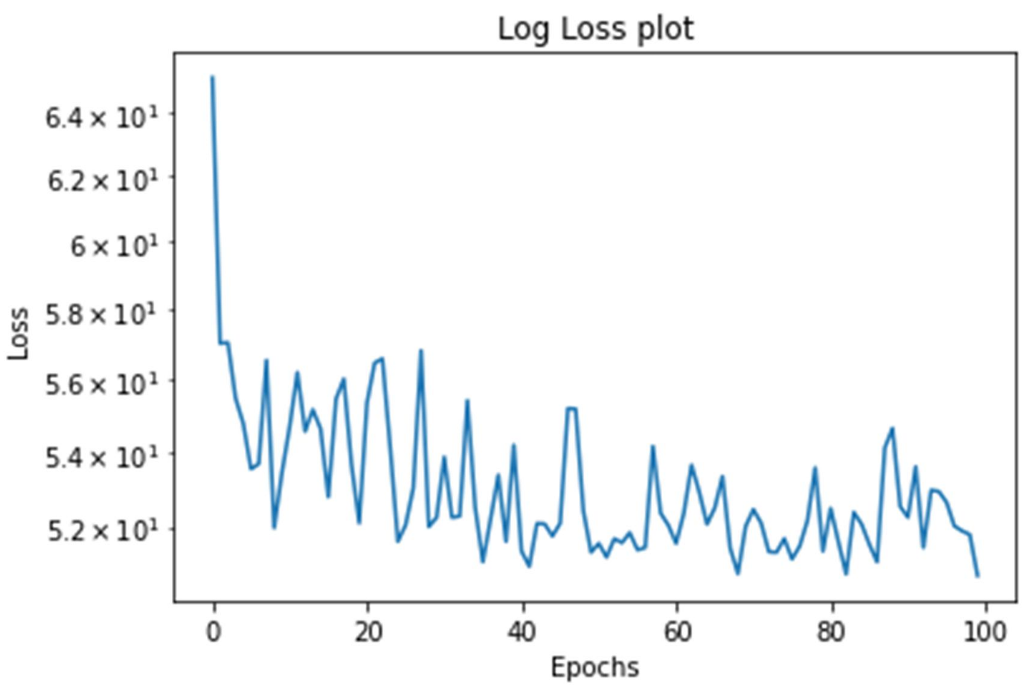

GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200

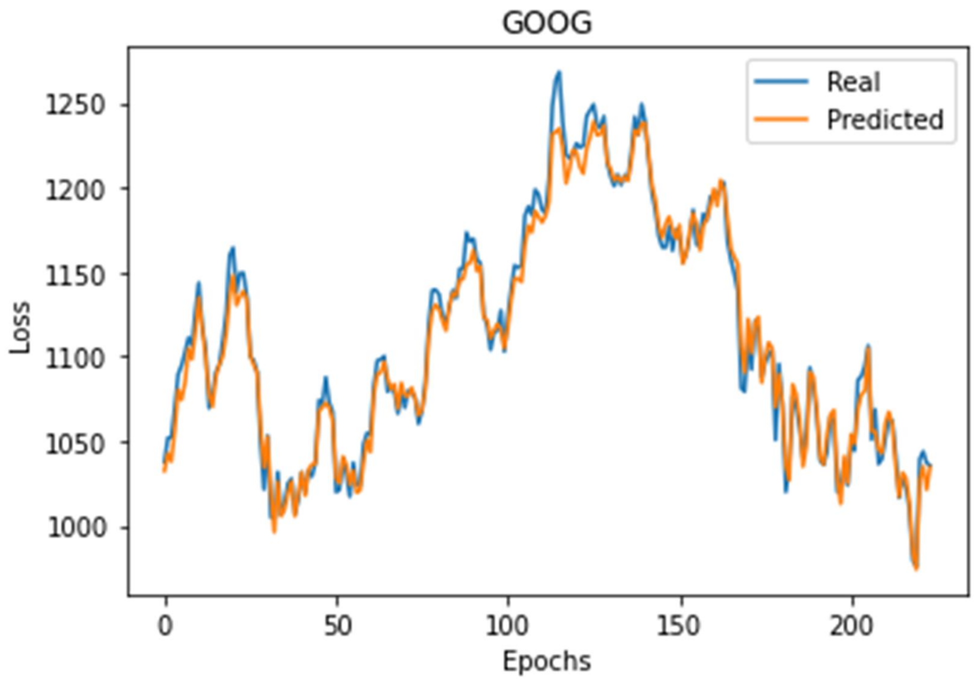

AAPL
Real
Predicted
Loss
Epochs
220
200
180
160
0
50
100
150
200

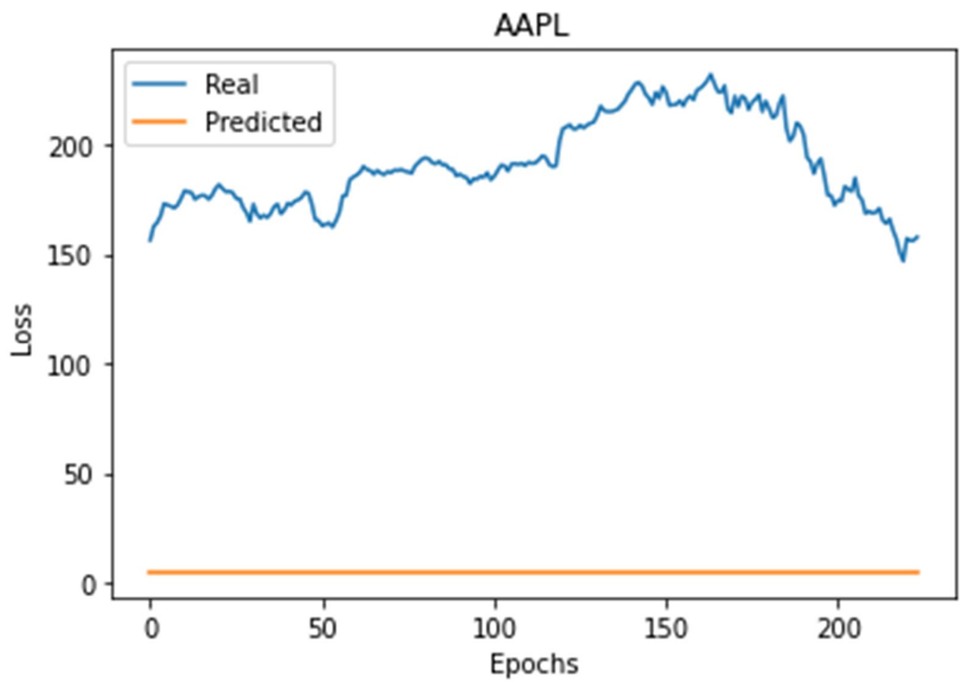
- MAPE + RNN

```
Total training time: 60.38407301902771 seconds
```

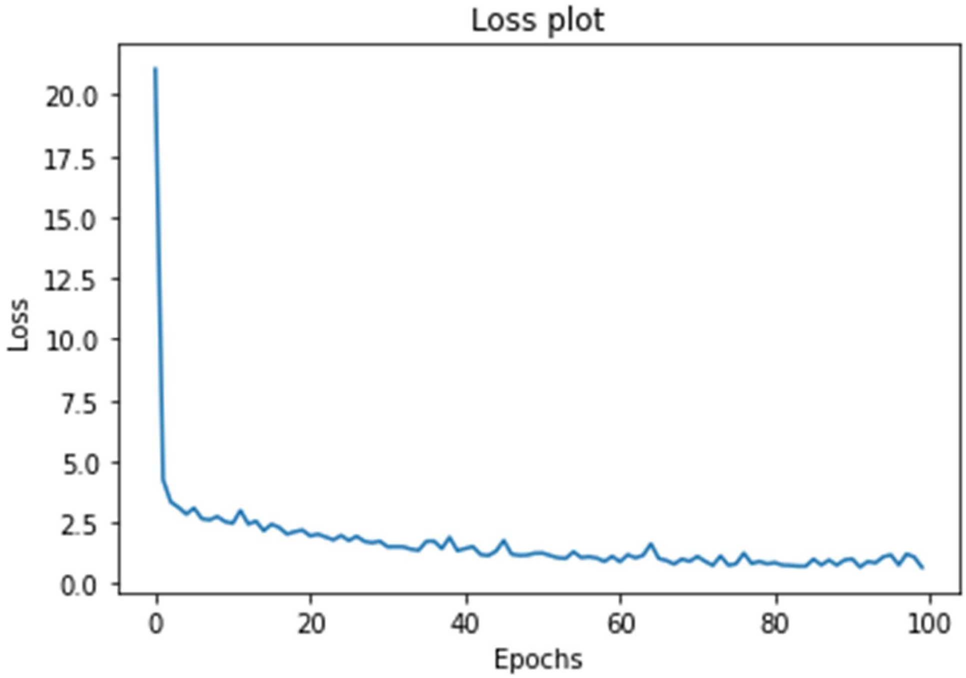


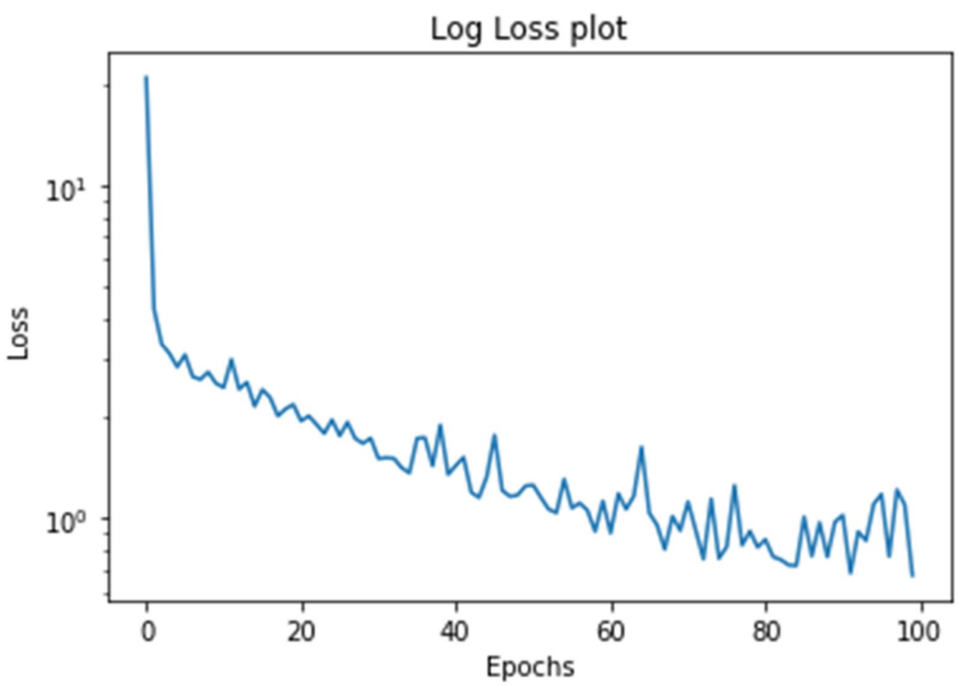


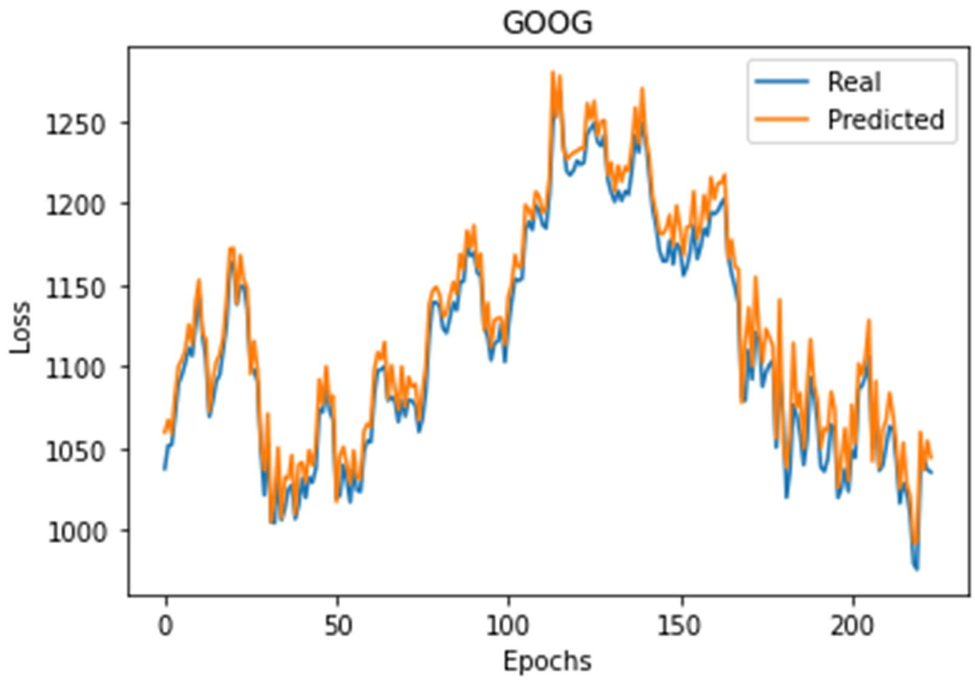

GOOG
Real
Predicted
1250
1200
1150
1100
1050
1000
Loss
0
50
100
150
200
Epochs

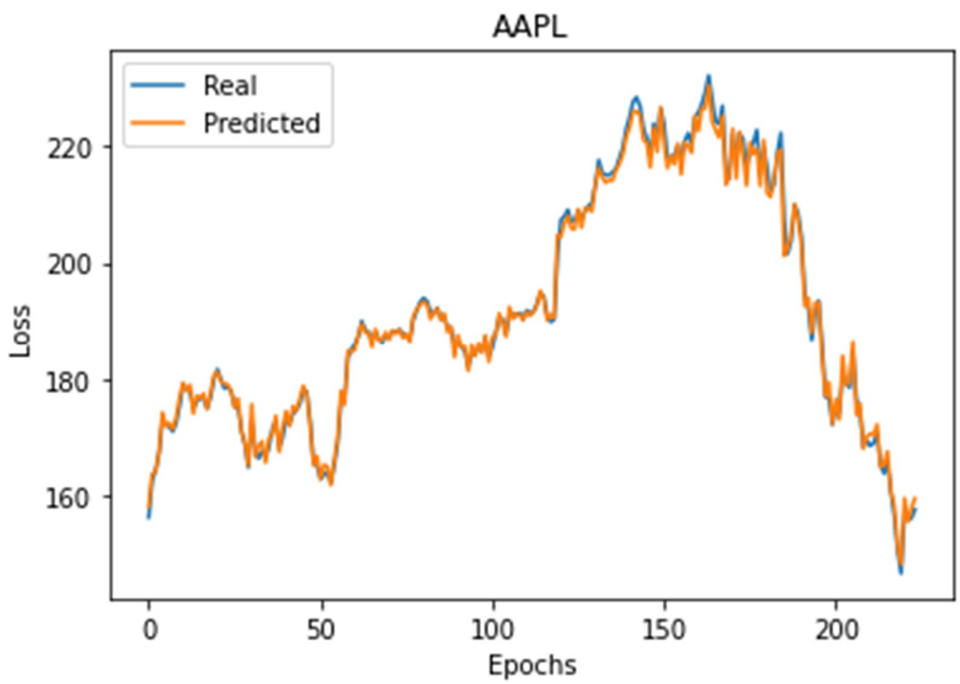

AAPL
Real
Predicted
200
150
100
50
0
Loss
0
50
100
150
200
Epochs

- MAPE + LSTM


Loss plot
Loss
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80
100
Epochs


Log Loss plot
Loss
10^1
10^0
0
20
40
60
80
100
Epochs



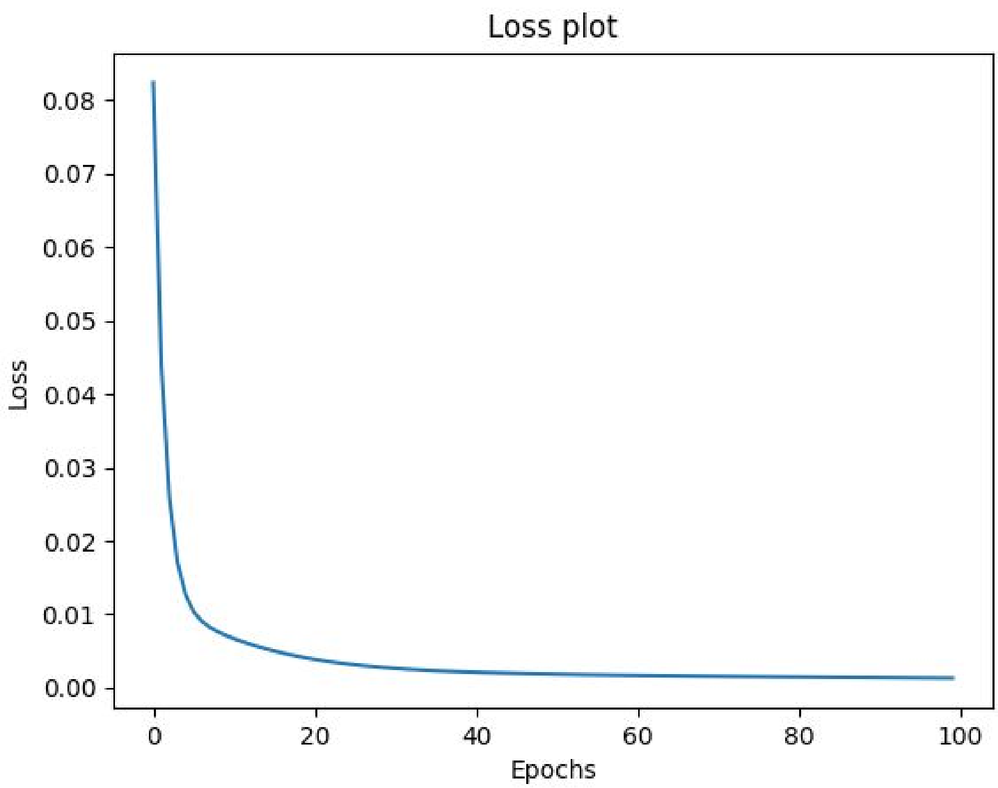


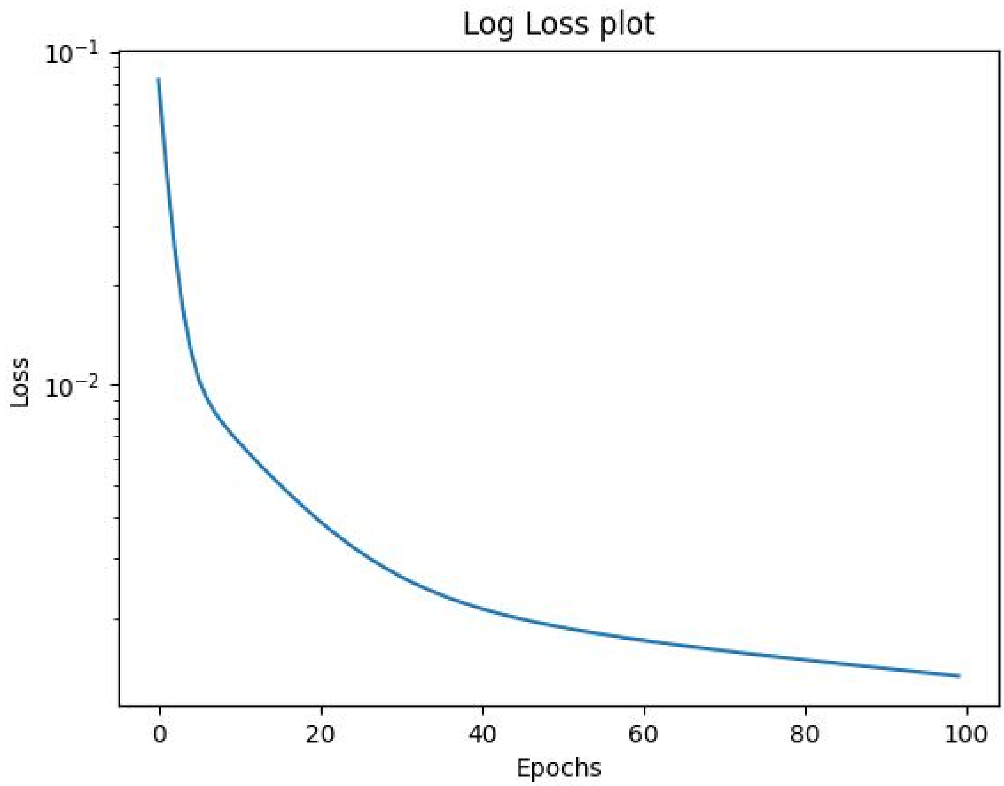
As we can see from the results, the MAPE loss function doesn't improve GRU and LSTM but reduces performance of RNN, as it has a constant gradient it suffers from vanishing gradient and thus the model is not trained well for RNN

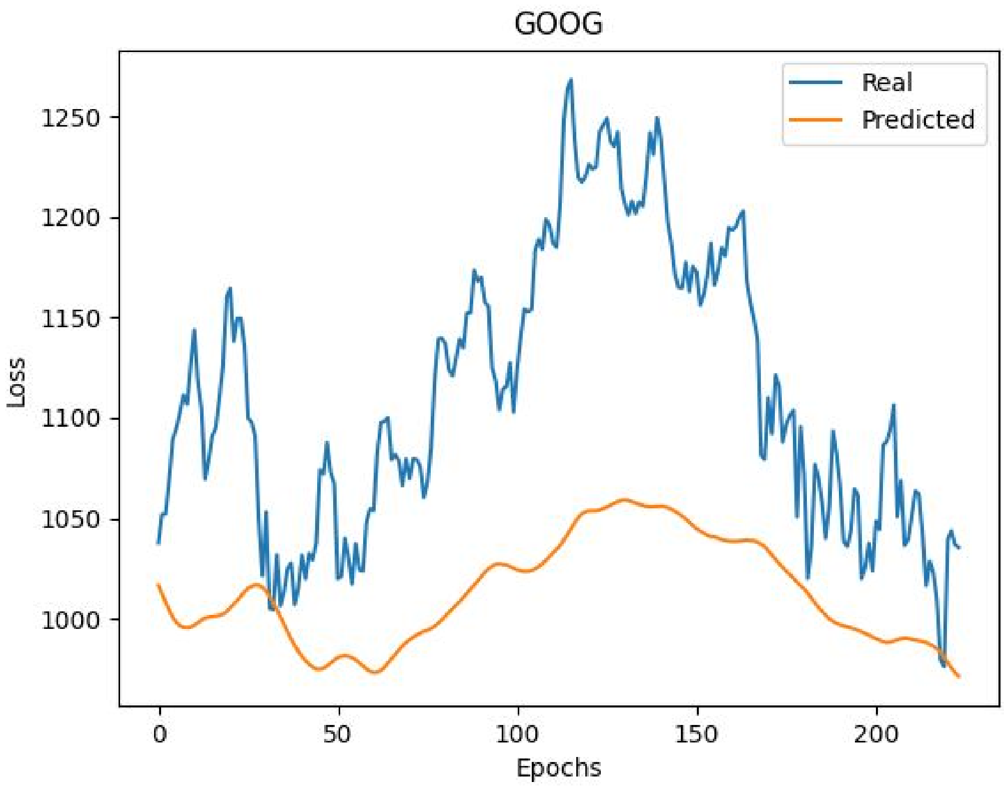
1.3

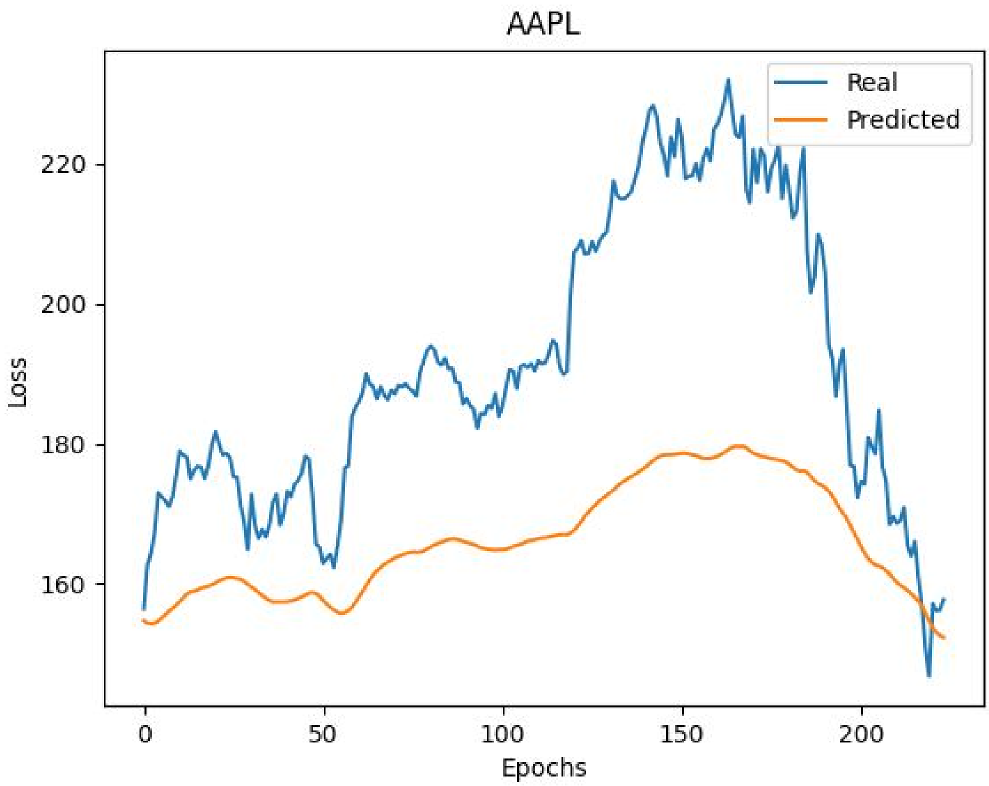
LSTM + Adagrad

```
Total training time: 149.08284449577332 seconds
```



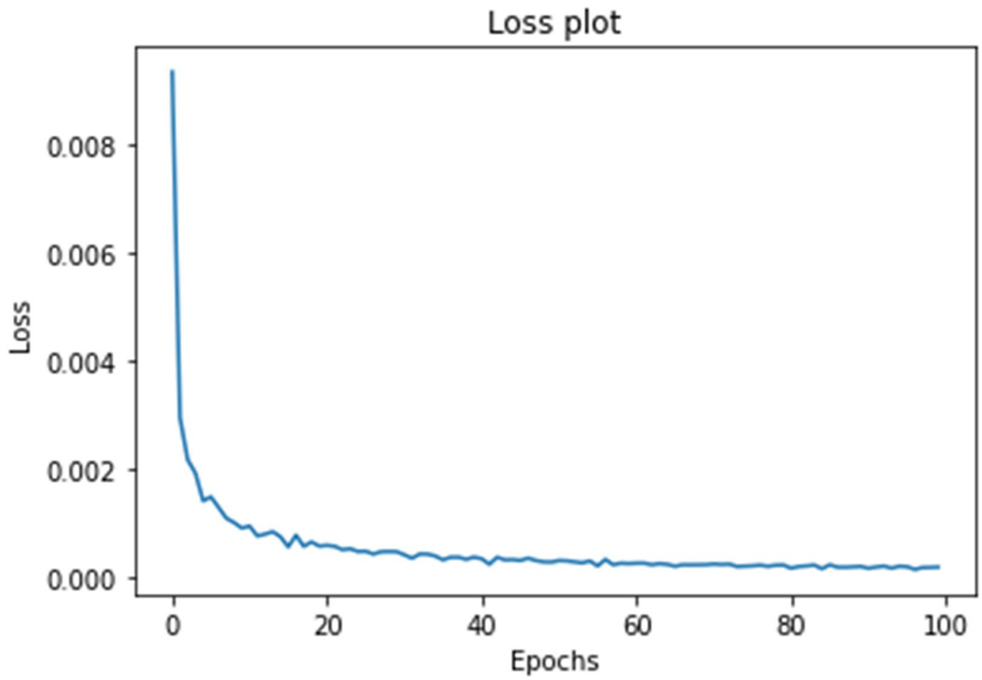

Log Loss plot
Loss
10^{-1}
10^{-2}
Epochs
0
20
40
60
80
100

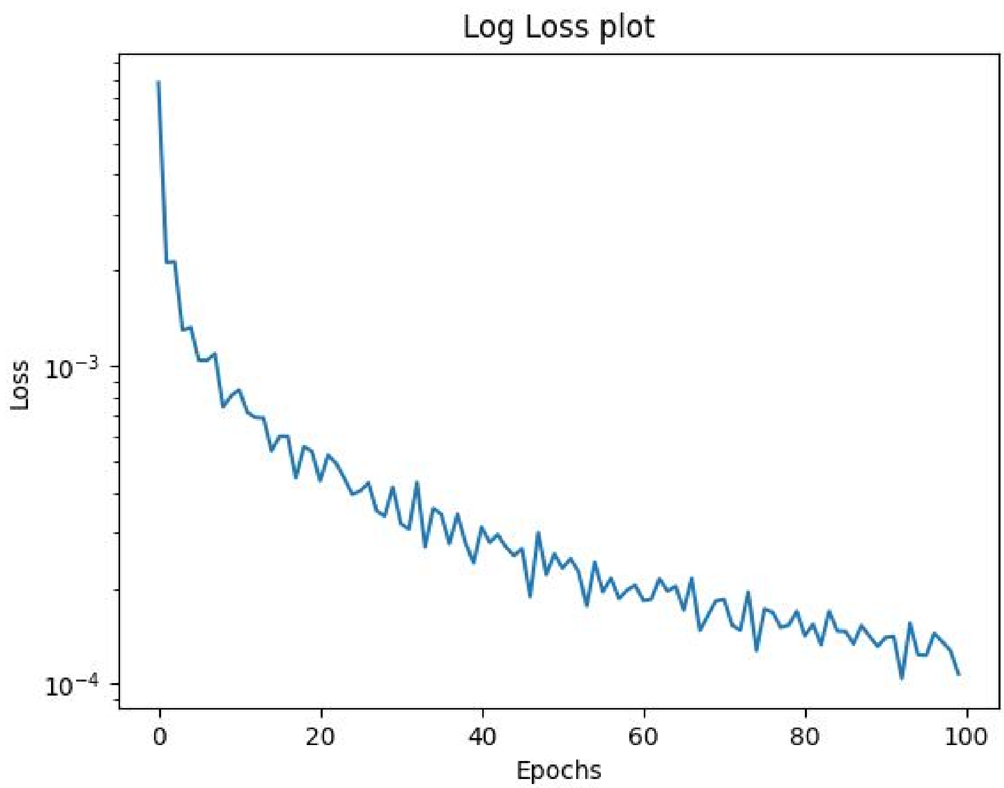

GOOG
Real
Predicted
Loss
Epochs
1000
1050
1100
1150
1200
1250
0
50
100
150
200

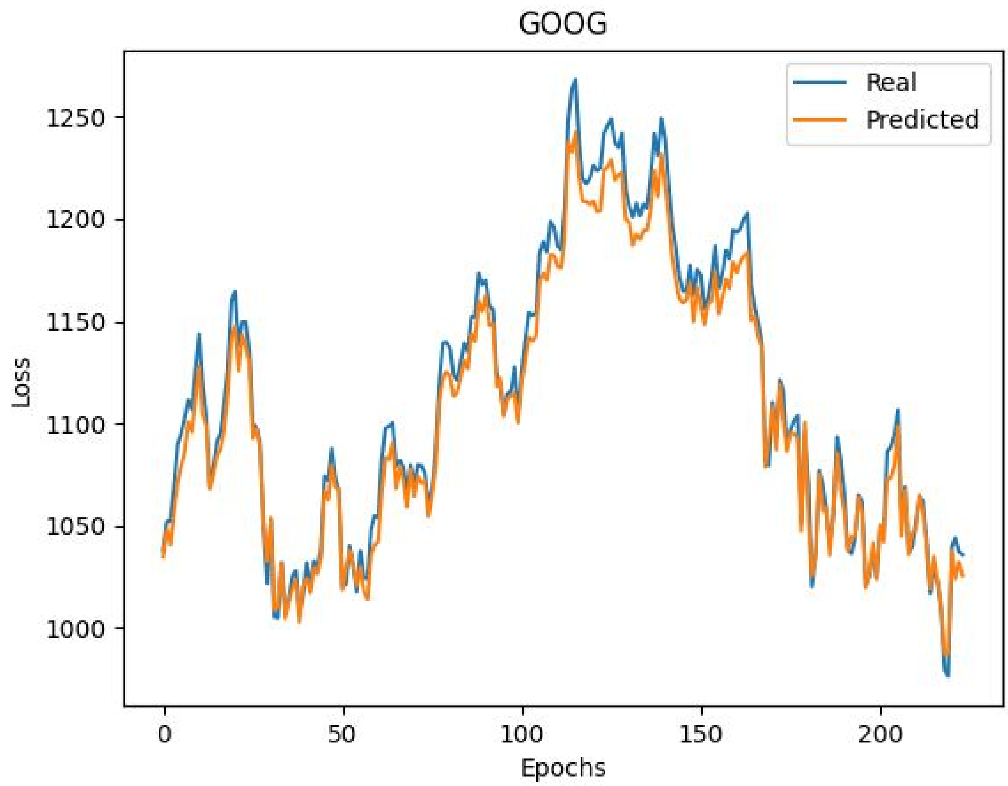


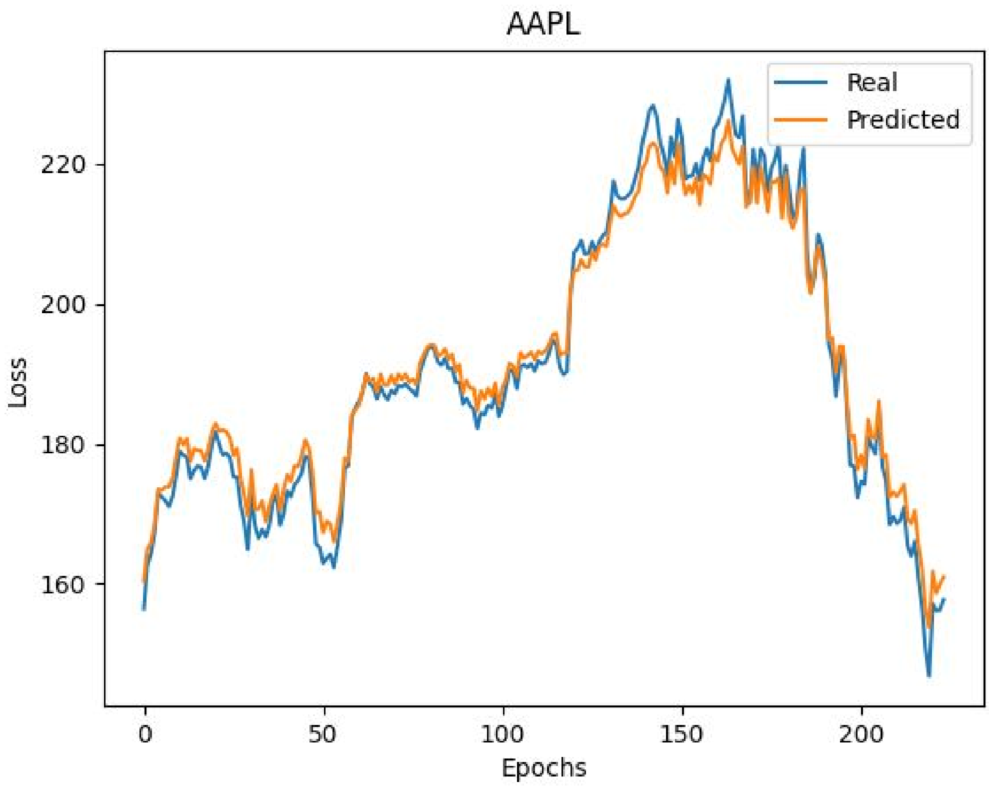
- LSTM + RMSprop

```
Total training time: 205.3320653438568 seconds
```


Loss plot
0.008
0.006
0.004
0.002
0.000
Loss
0
20
40
60
80
100
Epochs

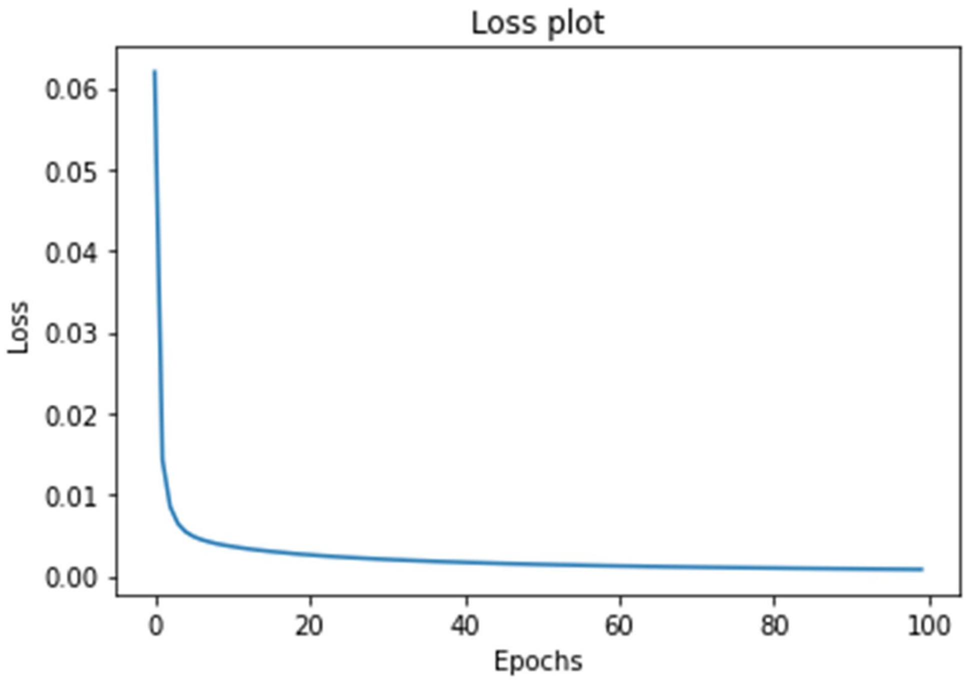

Log Loss plot
10^{-3}
10^{-4}
Loss
0
20
40
60
80
100
Epochs

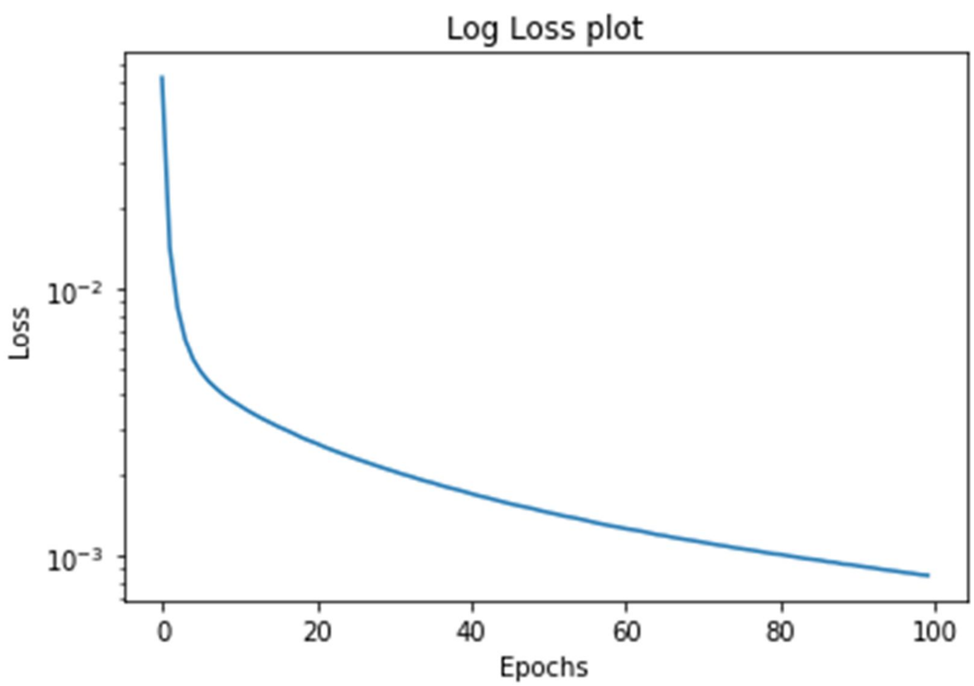

GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200

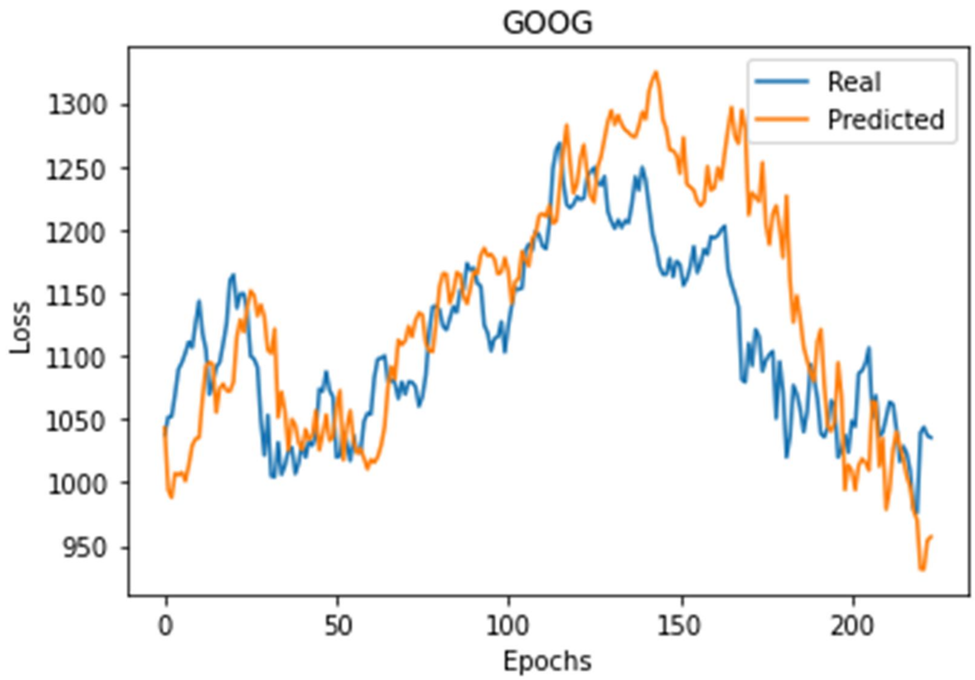


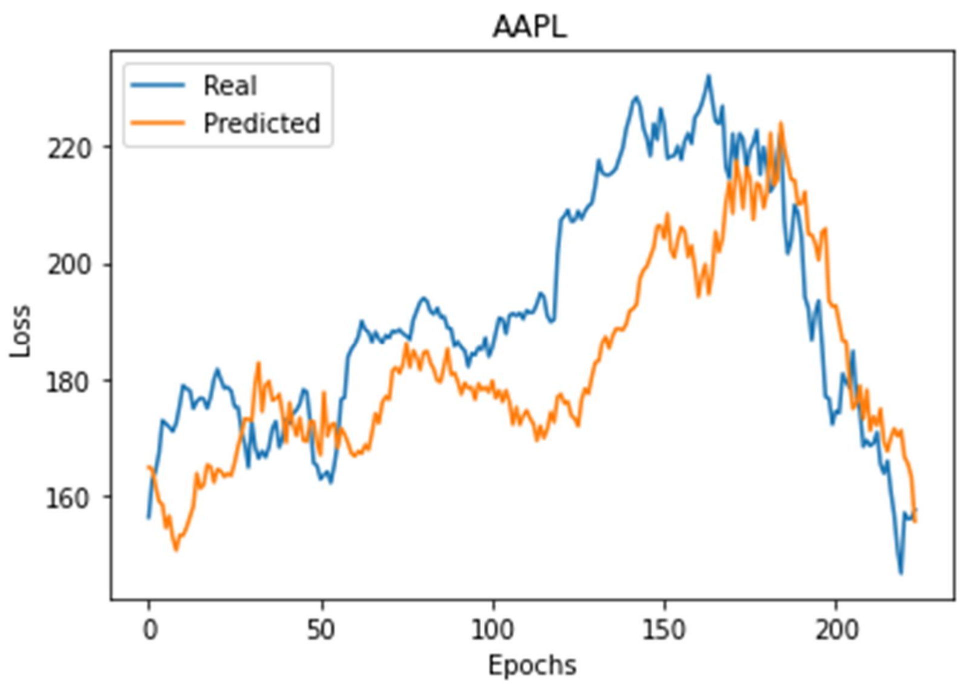
- ADAgrad + RNN

```
Total training time: 59.3671555519104 seconds
```


Loss plot
Loss
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100
Epochs

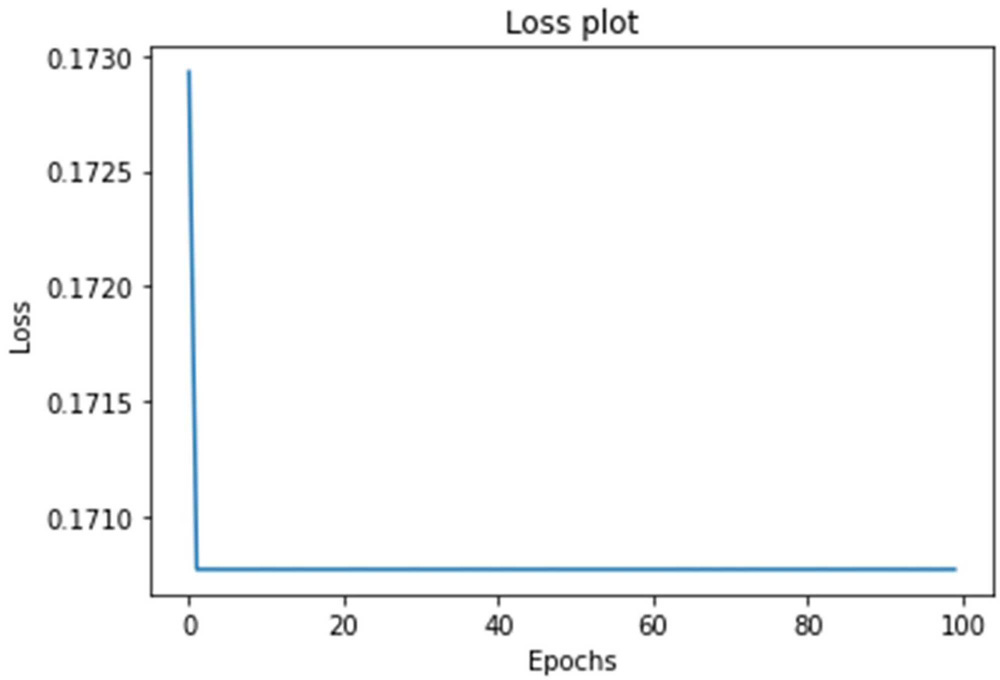

Log Loss plot
Loss
10^{-2}
10^{-3}
0
20
40
60
80
100
Epochs

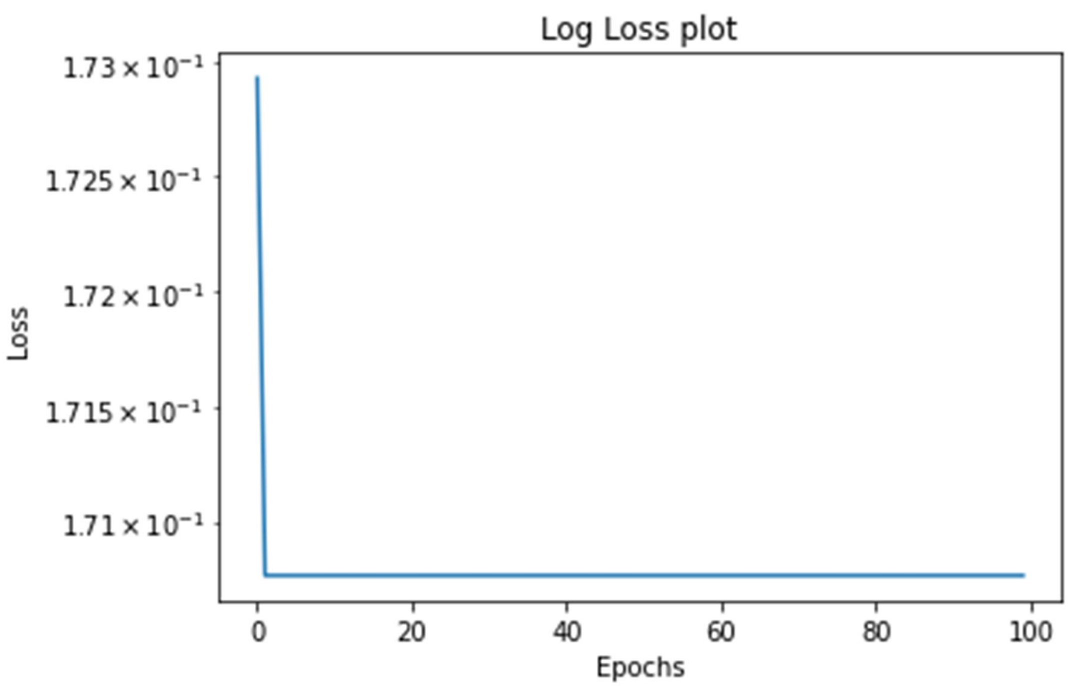


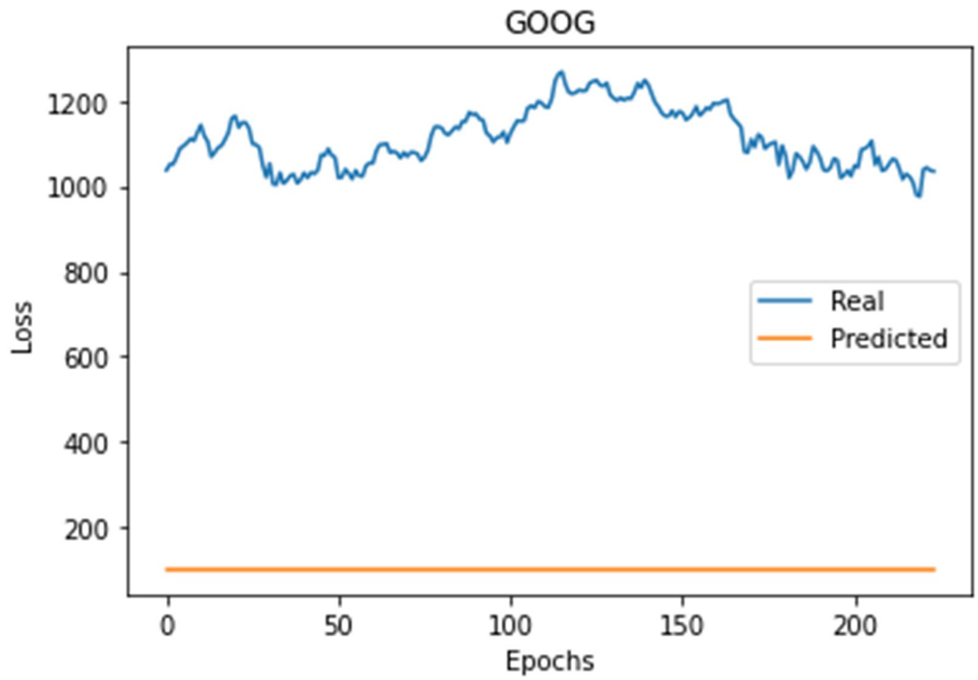


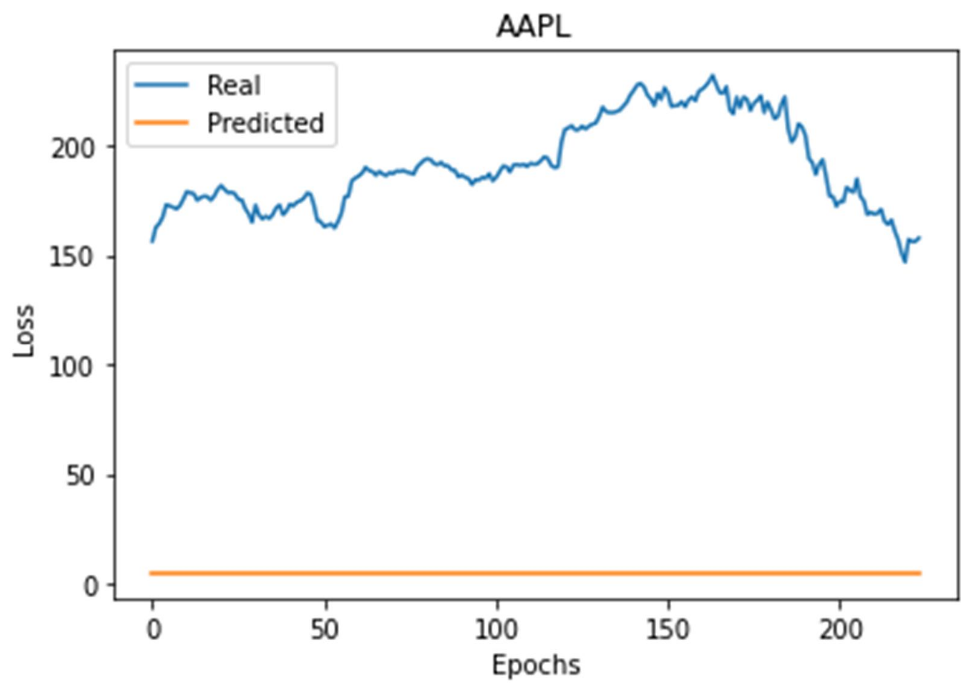
- RMSprop +RNN

```
Total training time: 63.164148807525635 seconds
```


Loss plot
0.1730
0.1725
0.1720
0.1715
0.1710
Loss
0
20
40
60
80
100
Epochs

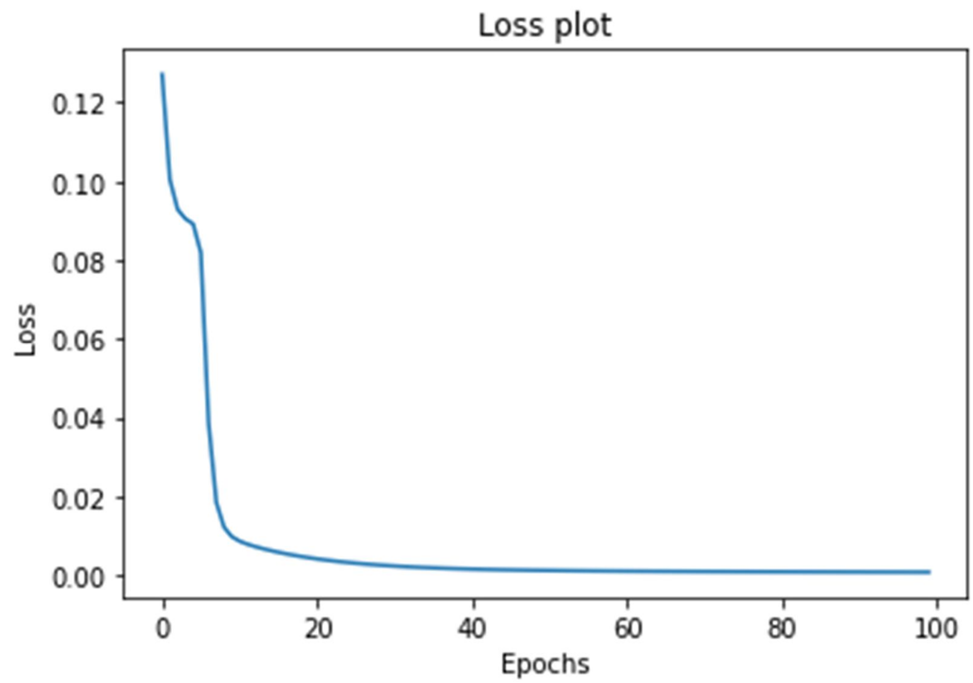

Log Loss plot
1.73×10^{-1}
1.725×10^{-1}
1.72×10^{-1}
1.715×10^{-1}
1.71×10^{-1}
Loss
0
20
40
60
80
100
Epochs

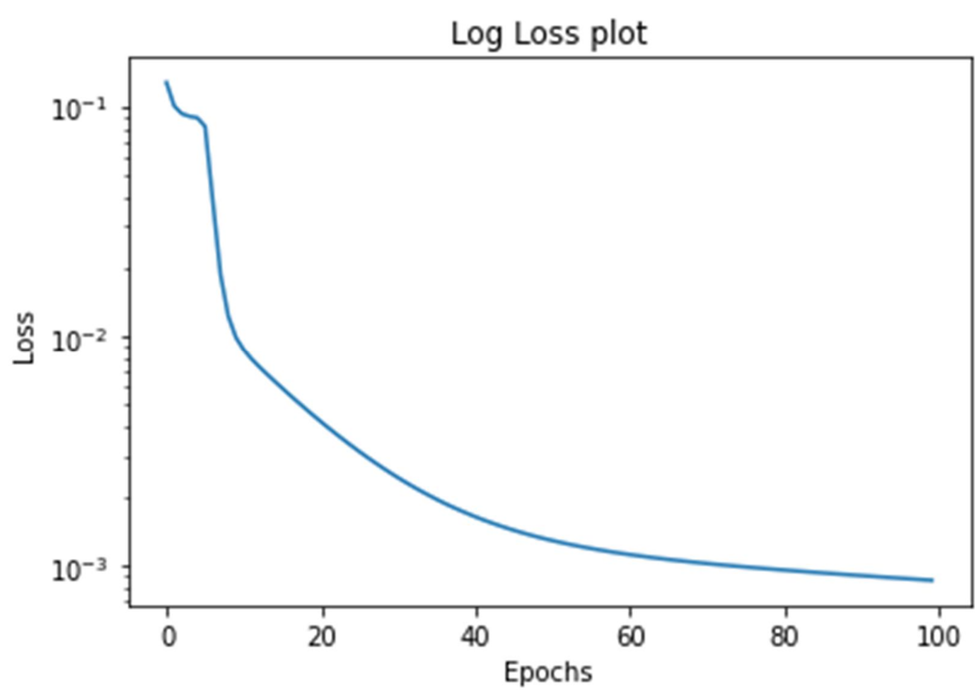

GOOG
1200
1000
800
600
400
200
Loss
Real
Predicted
0
50
100
150
200
Epochs

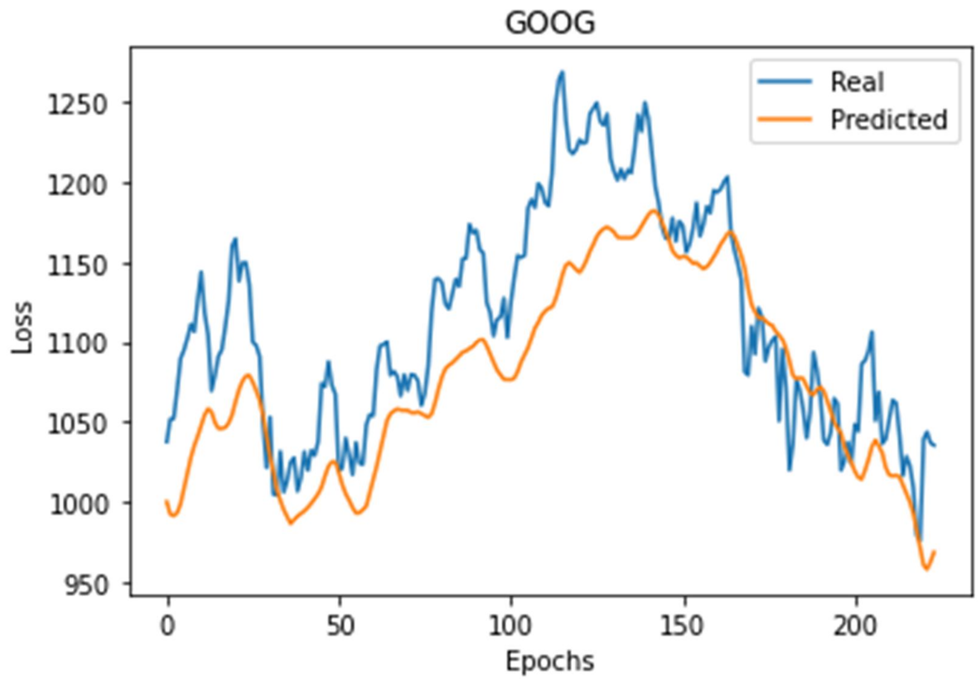

AAPL
Real
Predicted
200
150
100
50
0
Loss
0
50
100
150
200
Epochs

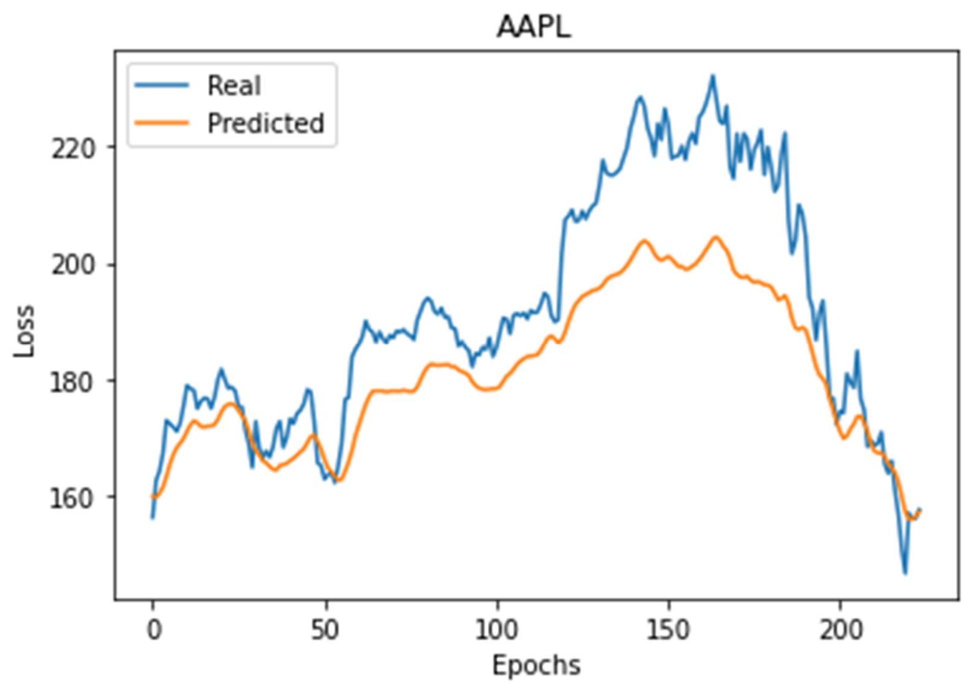
- ADAgrad + GRU

```
Total training time: 137.29109835624695 seconds
```



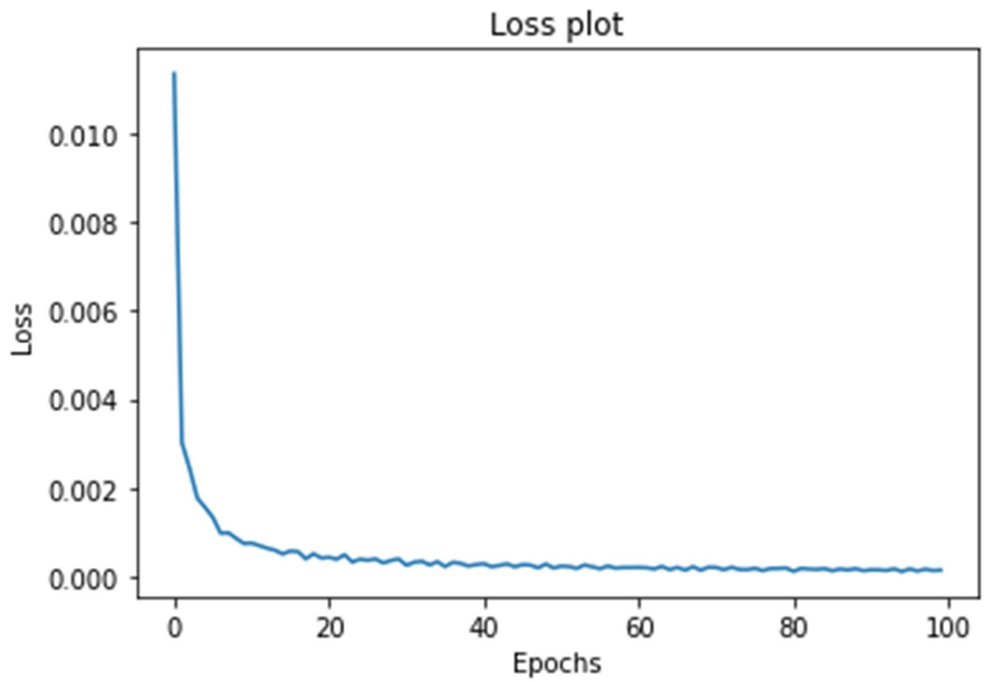


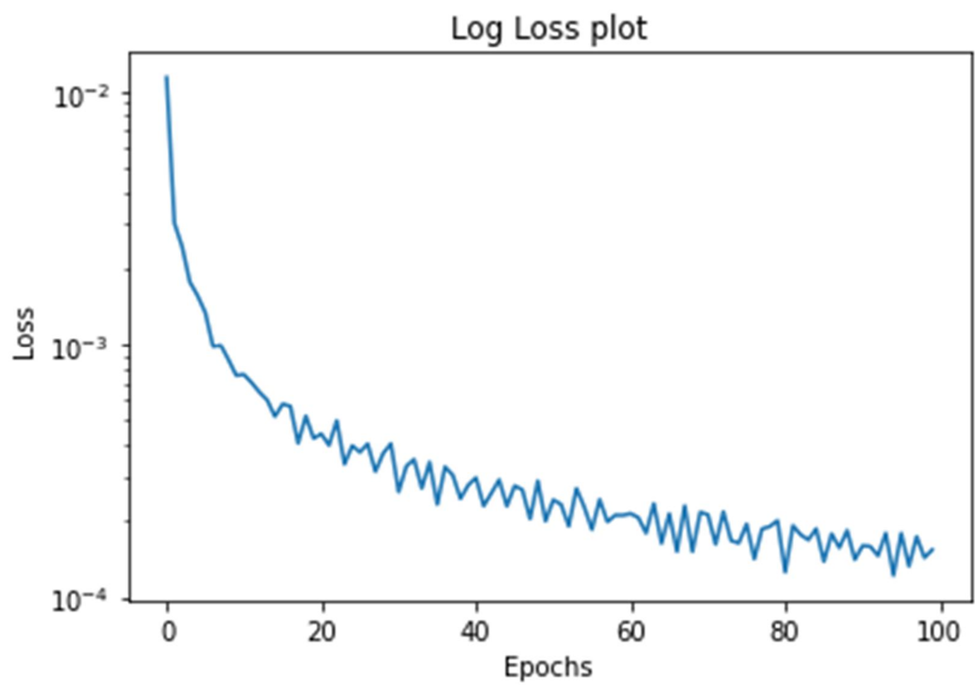

GOOG
Real
Predicted
1250
1200
1150
1100
1050
1000
950
Loss
0
50
100
150
200
Epochs

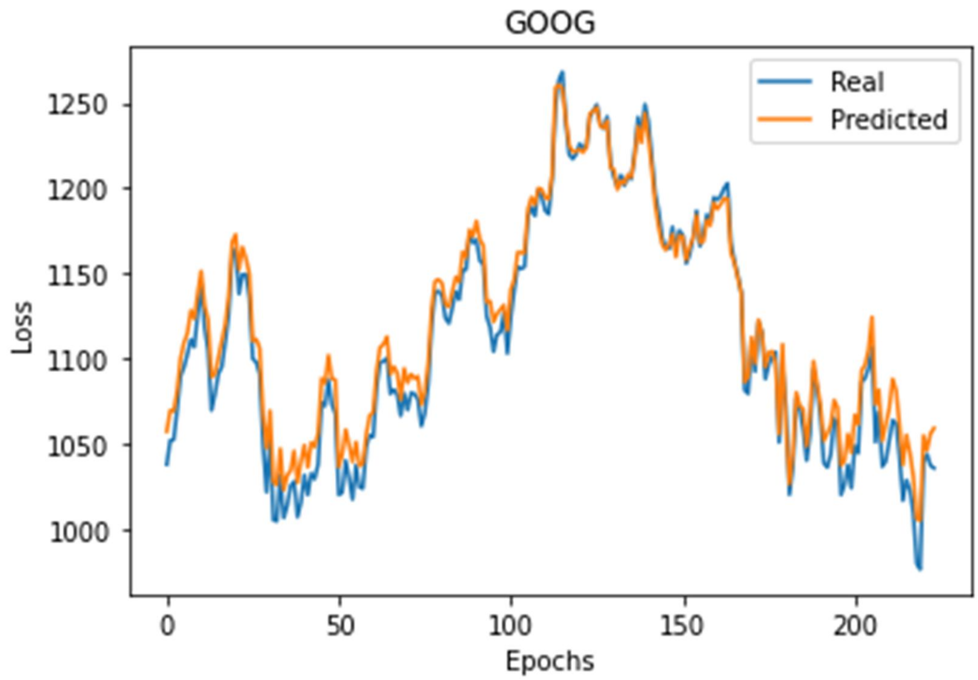

AAPL
Real
Predicted
220
200
180
160
Loss
0
50
100
150
200
Epochs

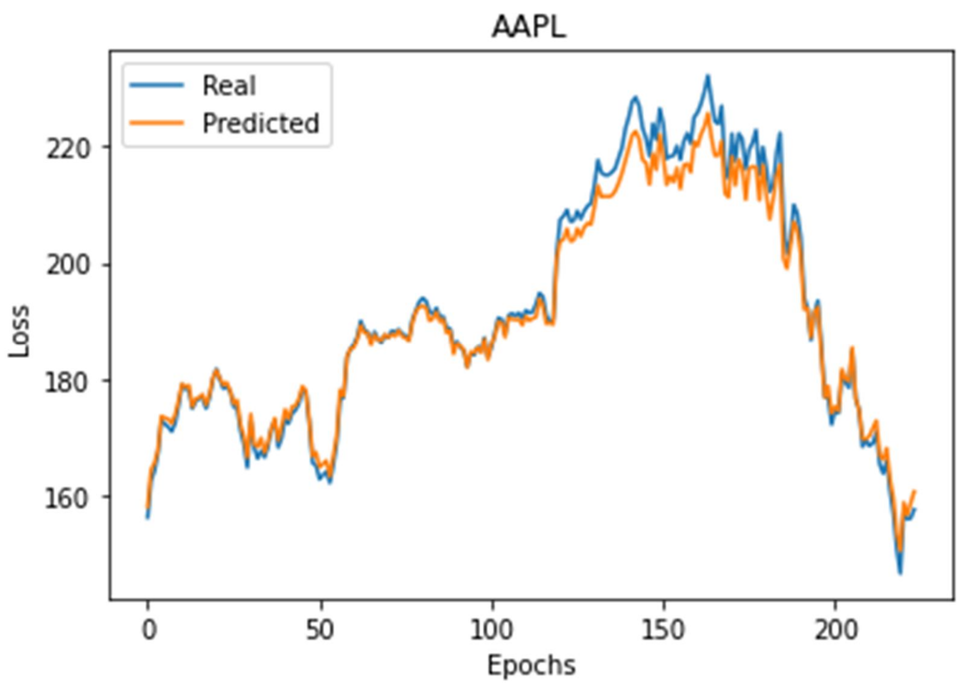
- RMSprop + GRU

```
Total training time: 137.48347330093384 seconds
```







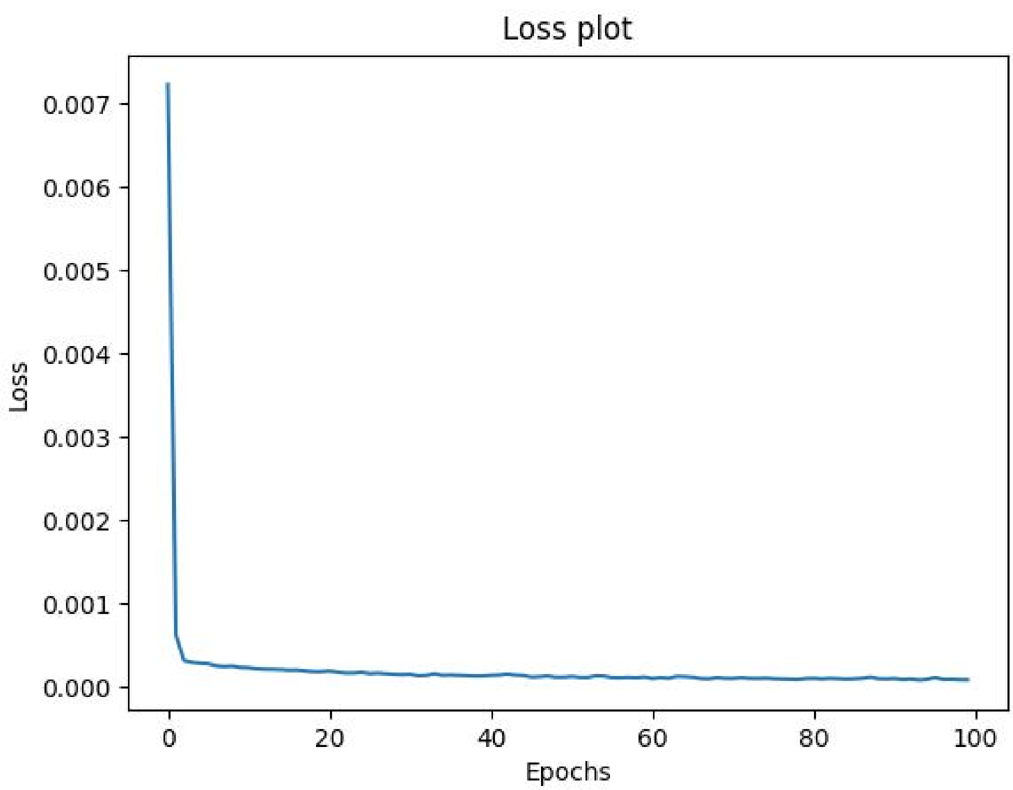


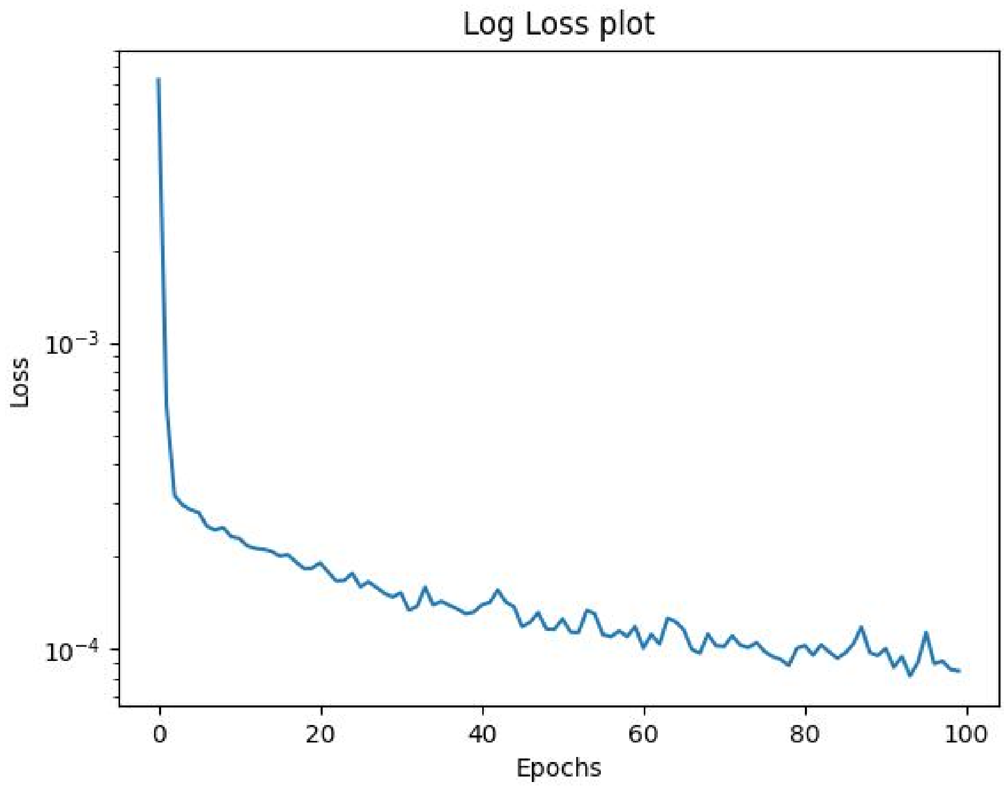
As we can see from the results both ADAgrad and RMSprop optimizers get outperformed by Adam, because Adam uses both first and second moments and benefits from advantages of two other optimizers, so it is generally the best choice.

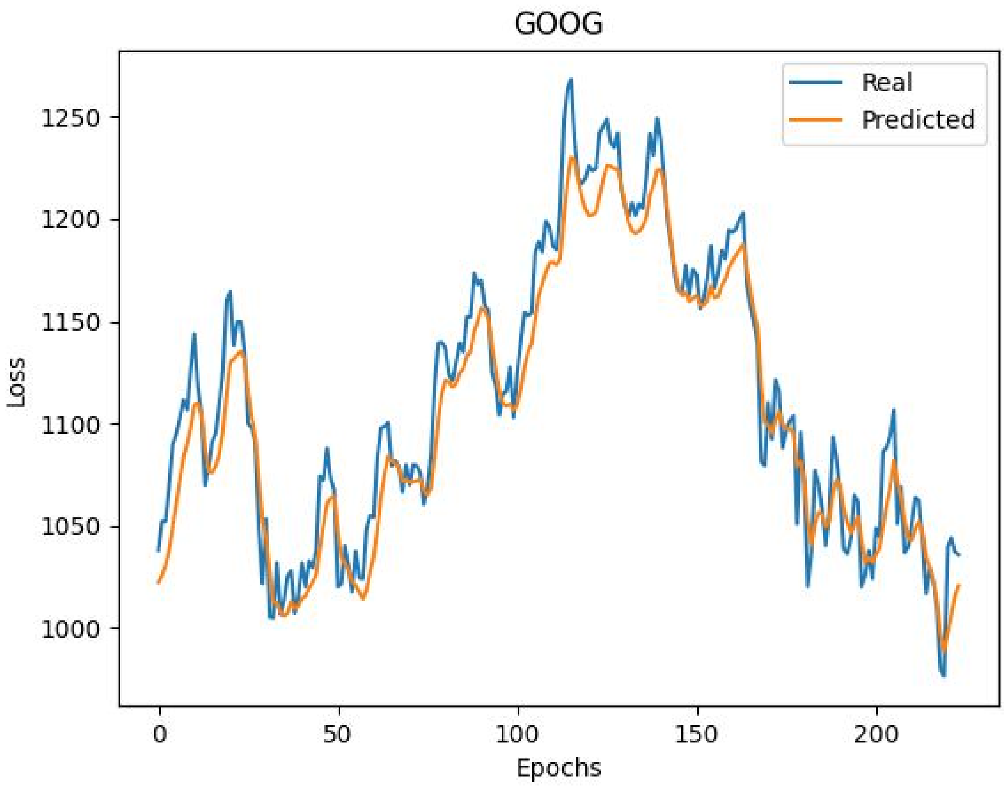
Also, Adam takes noticeably less time to train.

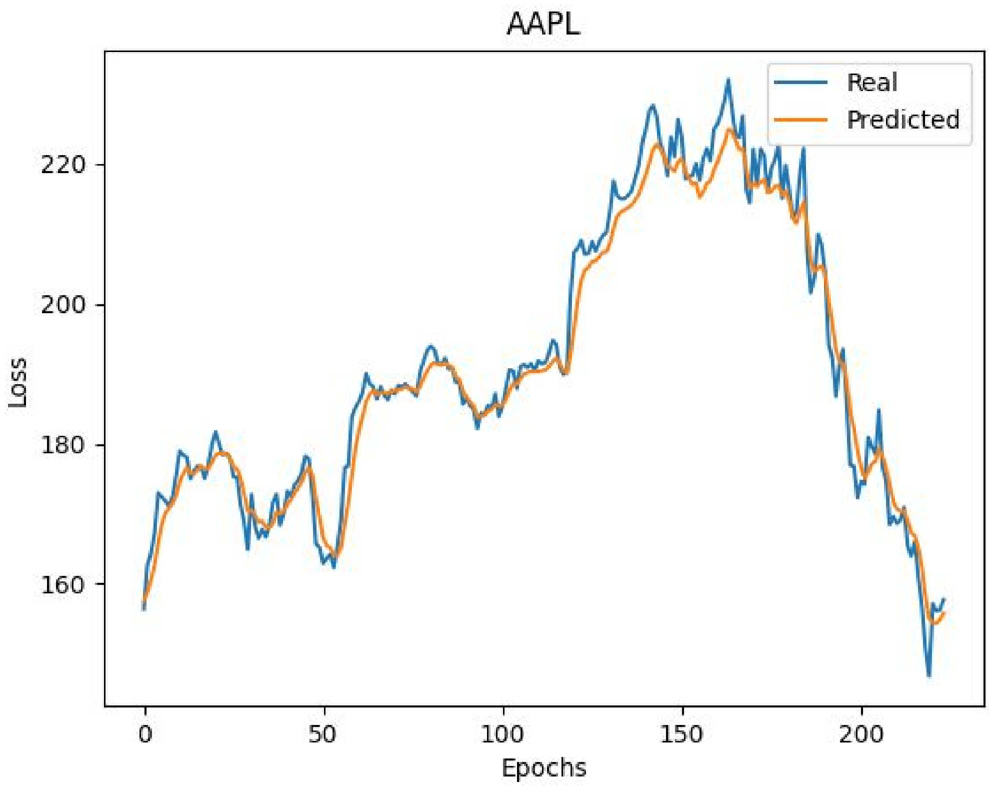
Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelta	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓

- LSTM + 0.1 Dropout

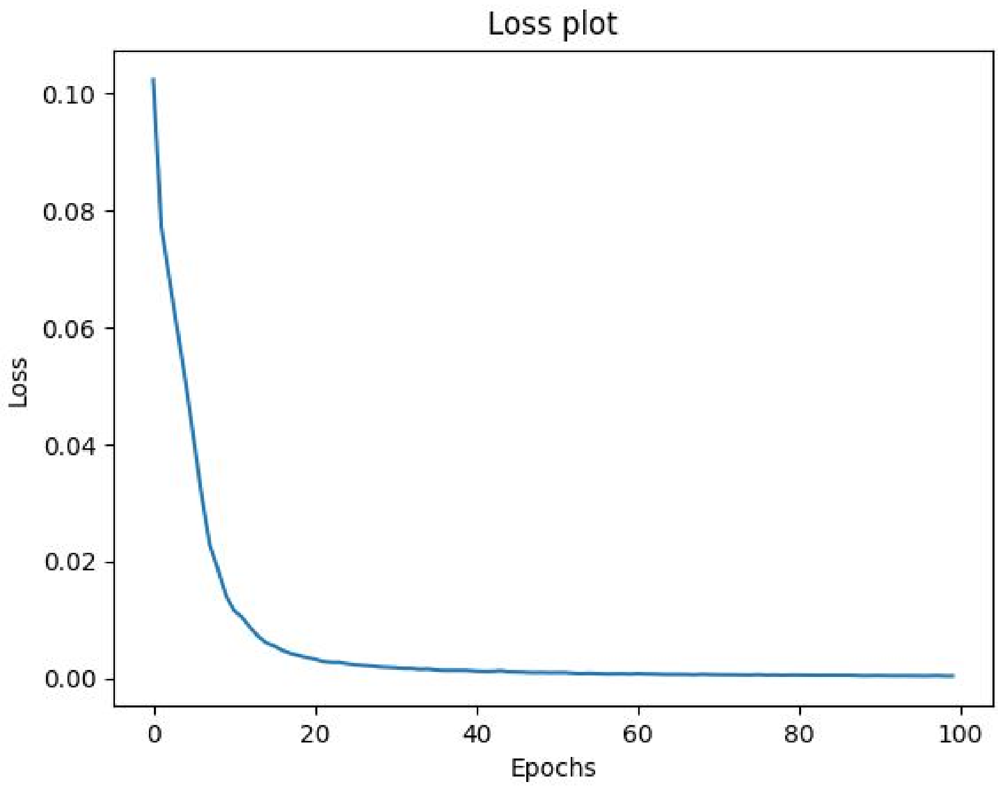


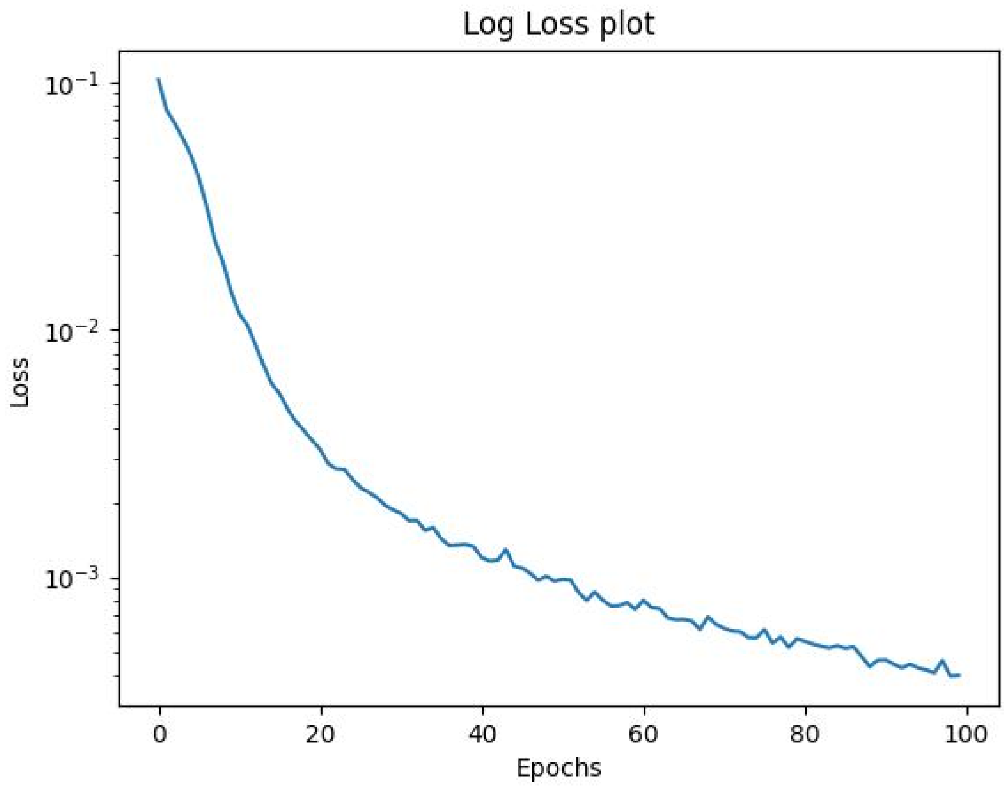

Log Loss plot
Loss
10^{-3}
10^{-4}
0
20
40
60
80
100
Epochs


GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200

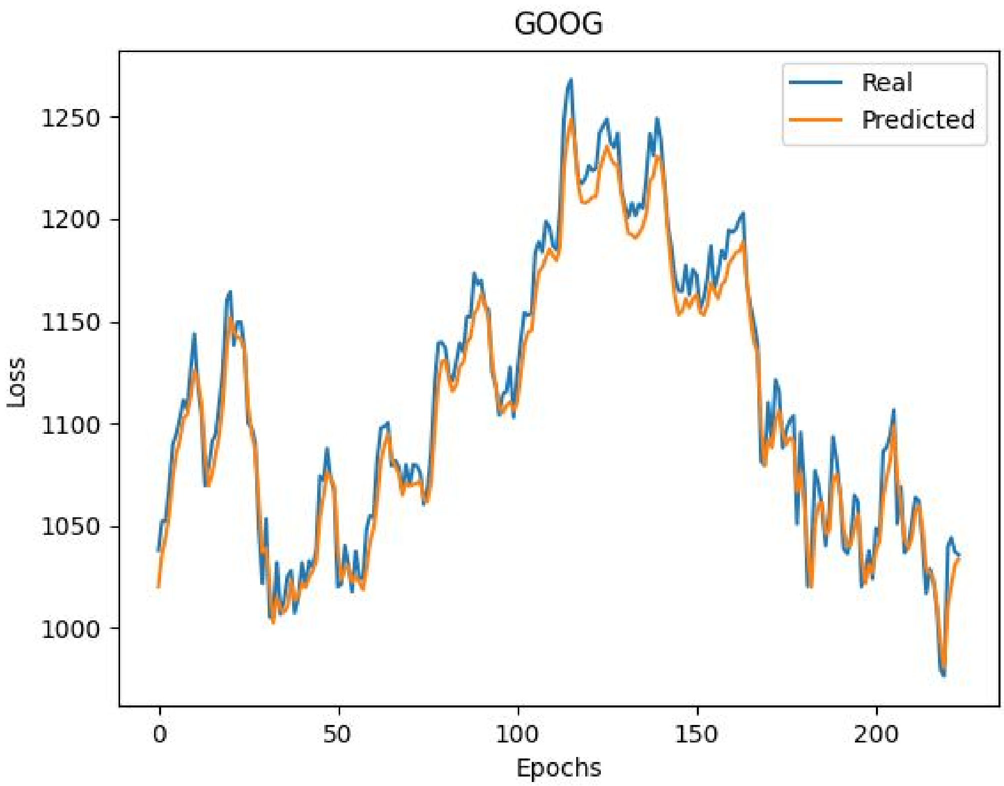


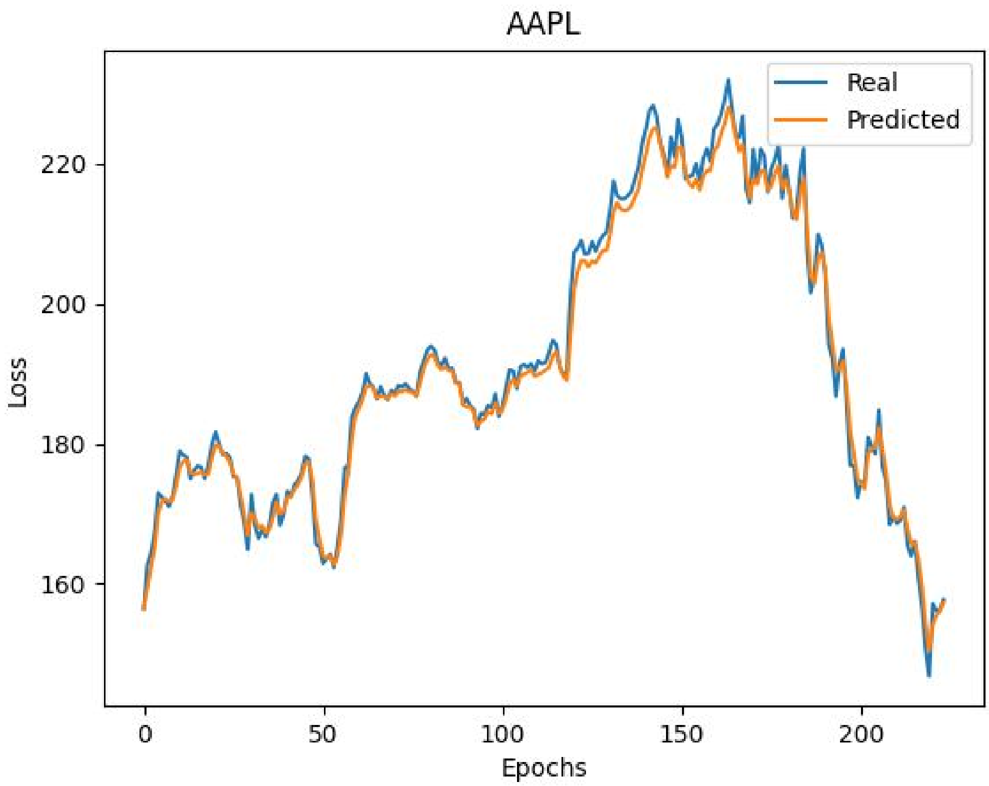
RNN + dropout 0.1


Loss plot
0.10
0.08
0.06
0.04
0.02
0.00
Loss
0
20
40
60
80
100
Epochs

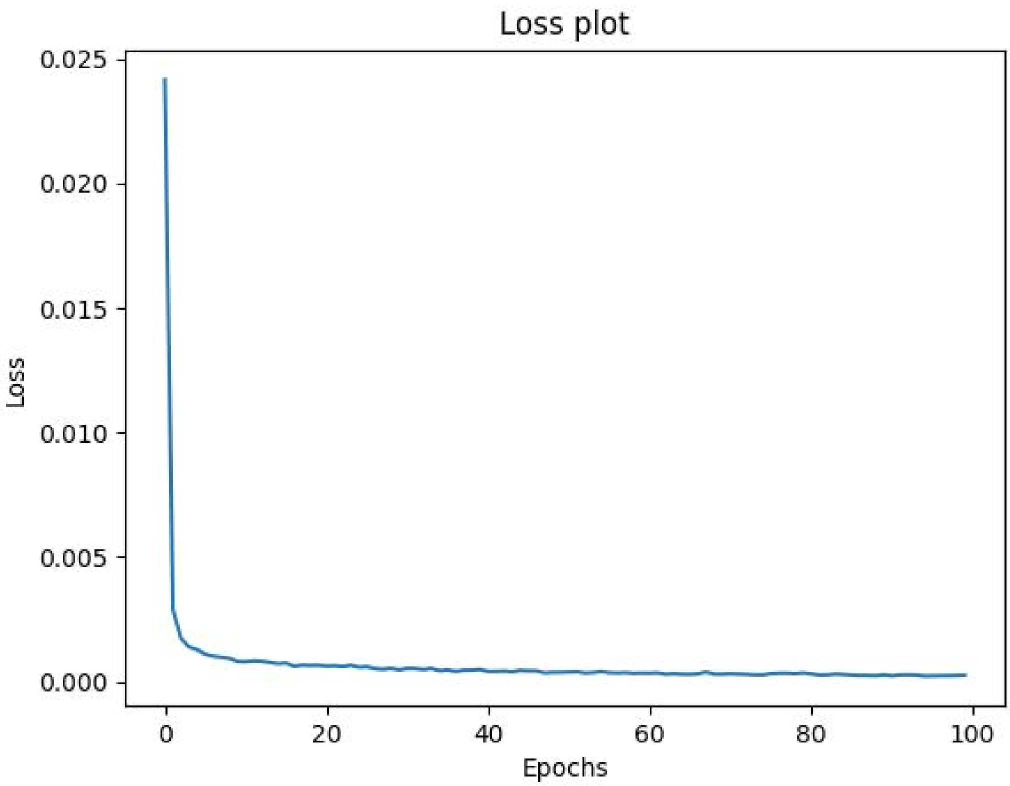


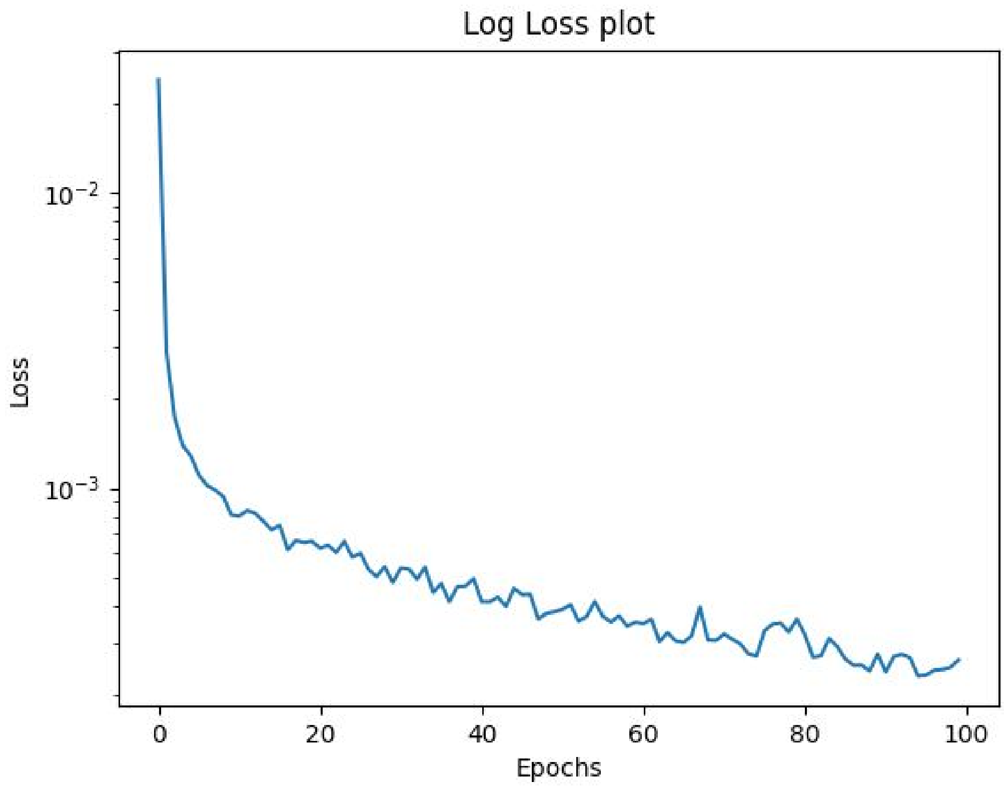
Log Loss plot
Loss
Epochs
10^{-1}
10^{-2}
10^{-3}
0
20
40
60
80
100

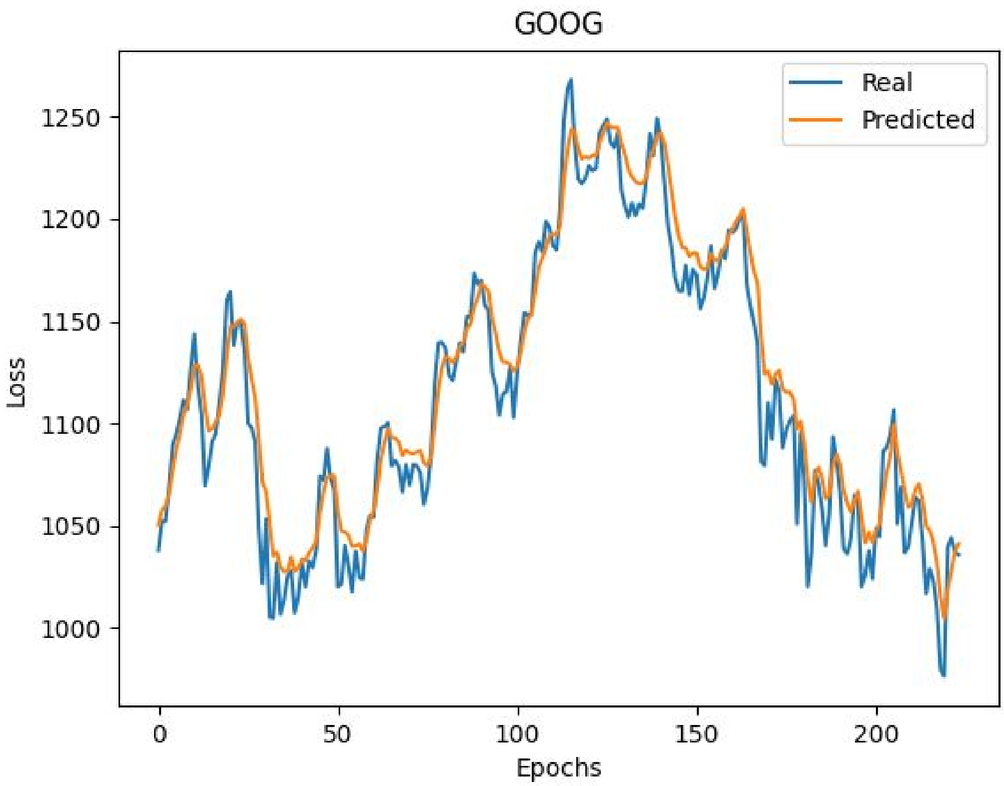

GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200

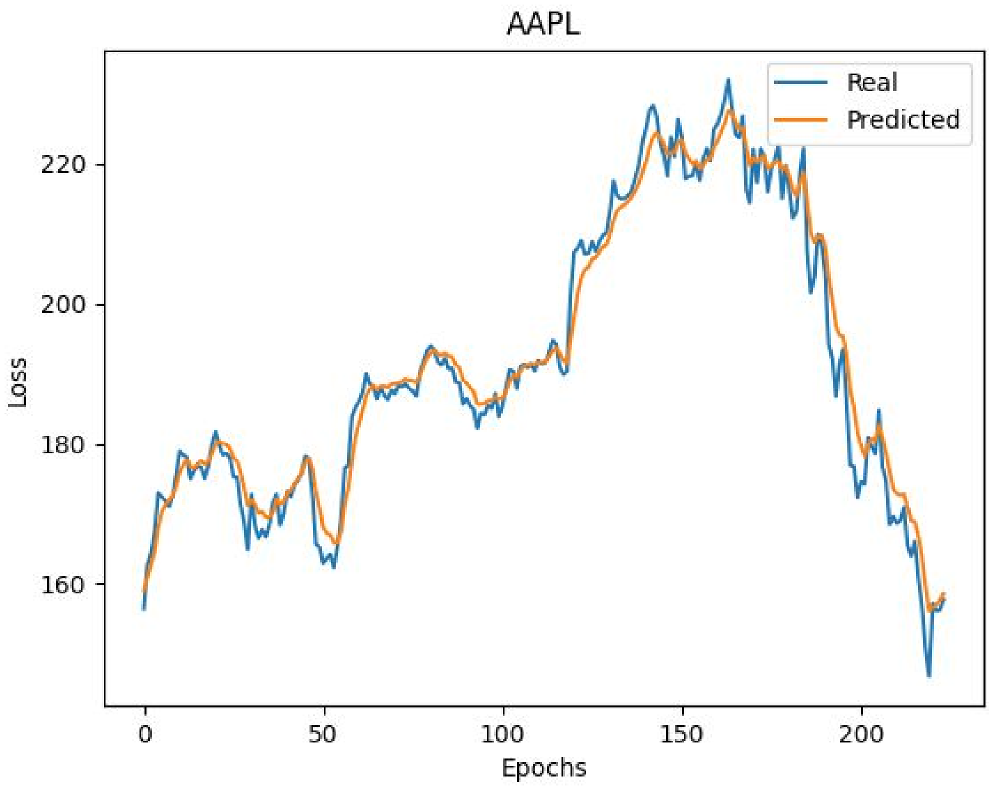


GRU + 0.1 dropout


Loss plot
0.025
0.020
0.015
0.010
0.005
0.000
Loss
0
20
40
60
80
100
Epochs


Log Loss plot
Loss
Epochs
10^{-2}
10^{-3}
0
20
40
60
80
100


GOOG
Real
Predicted
Loss
Epochs
1250
1200
1150
1100
1050
1000
0
50
100
150
200



By comparing the results to part 2 we can see that adding dropout decreased performance of LSTM and GRU units but improved RNN unit.

Dropout is a regularization technique, and is most effective at preventing overfitting. However, there are several places when dropout can hurt performance.

- when the network is small relative to the dataset, regularization is usually unnecessary. If the model capacity is already low, lowering it further by adding regularization will hurt performance.

- when training time is limited. Usually, dropout hurts performance at the start of training, but results in the final "converged" error being lower. Therefore, if you don't plan to train until convergence, you may not want to use dropout.