Machine Learning Models Comparison for Bitcoin Price Prediction

Thearasak Phaladisailoed
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
ph.thearasak@gmail.com

Thanisa Numnonda
Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
thanisa@it.kmitl.ac.th

***Abstract*—In recent years, Bitcoin is the most valuable in the cryptocurrency market. However, prices of Bitcoin have highly fluctuated which make them very difficult to predict. Hence, this research aims to discover the most efficient and highest accuracy model to predict Bitcoin prices from various machine learning algorithms. By using 1-minute interval trading data on the Bitcoin exchange website named bitstamp from January 1, 2012 to January 8, 2018, some different regression models with scikit-learn and Keras libraries had experimented. The best results showed that the Mean Squared Error (MSE) was as low as 0.00002 and the R-Square (R^2) was as high as 99.2%.**

Keywords—Bitcoin; cryptocurrency; machine learning

I. INTRODUCTION

In October 2008, Bitcoin was first introduced by Satoshi Nakamoto through his white paper entitled "Bitcoin: peer-to-peer Electronic Cash System" [1]. Bitcoin is the first decentralized cryptocurrency while other digital currencies (aka Altcoin or alternative virtual currencies) are created by cloning or adjusting the mechanism of Bitcoin [2]. All transactions controlled by cryptography make them secure, validated, and stored in "blockchain" by a decentralized network [3]. With the concept based on the new electronic cash system, online payment transactions can be done directly between any two willing parties without the need for a trusted third party such as a financial institution. Bitcoin was the largest and most popular in cryptocurrency market measured by market capitalization in March 2017. Bitcoin accounts occupied 72% of the total cryptocurrency in market and number of transactions were 286,419 in January – February 2017 which are more than all other cryptocurrencies [2]. In 2007, the price of Bitcoin was at 1,000 USD and went up to 16,000 USD in December 2017. This makes Bitcoin's prices extremely difficult to predict.

Therefore, this research aims to discover the most efficient and highest accuracy model to predict Bitcoin's prices from various machine learning algorithms. By using 1-hour interval exchange rate in USD from January 1, 2012 to January 8, 2018 via the Kaggle website, some different regression models with scikit-learn and Keras libraries had experimented.

In the rest of this paper, section 2 provides background work on scikit-learn, Tensorflow, and Keras. Related work is discussed in section 3. Implementation of various machine learning algorithms is demonstrated in section 4. Then, the results are shown in section 5 and conclusion will be discussed in the last section.

II. BACKGROUND WORK

This research uses two libraries; scikit-learn and Keras for analyzing data in order to create machine learning models. In this research, the Tensorflow library is also used to generate data flow graphs.

A. Scikit-learn

Scikit-learn is an open-source library for analyzing data mining. Python is used to analyze and create models from various machine learning algorithms, such as classification, regression, and clustering. Scikit-learn can also be used for preparing data in several ways: normalization, standardization, and cleaning outlier data or missing data [4].

B. Tensorflow

Tensorflow, created by Google, is an open-source deep learning framework. It can be used to train Neural Network (NN) models and to predict results by using much Graphical Processing Unit (GPU) to collaborate, therefore, powerful algorithms for deep learning and NN can be implemented. This framework can also be applied in several other areas such as speech recognition, computer vision, robotics, and so on. Tensorflow can generate data flow graphs for processing when graphs are composed of node groups. Fig 1 demonstrates an example of data flow processing [5].

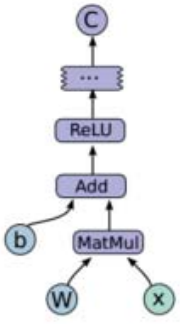

Fig. 1. Example of data flow processing in Tensorflow

C. Keras

Keras is an open-source library used for high-level NN. It provides API for NN programming written in Python. It can also be used with Tensorflow, CNTK, and Theano libraries [6]. Models of machine learning, NN, and deep learning can be created by using Keras. Dividing codes into parts make Keras easily to build and understand. The parts of generating models normally consist of neural layers, cost functions, optimizer, and activation functions. New defined functions or classes can also be easily developed by using Python.

III. Related Work

Shah et al [7] represent Bayesian regression algorithm for generating latent source models. The Bitcoin exchange datasets are taken from Okcoin in China. By collecting the data every 10 minutes to use in Bitcoin exchange, the result shows that, in 50 days, ROI is 89% with the sharp ratio at 4.10.

Madan et al [8] are also use the datasets from Okcoin, but separating the data into series of 30, 60, and 120 minutes. Binomial Logistic Regression, Support Vector Machine (SVM), and Random Forest are used to predict Bitcoin's prices with the accuracy at 97% and 55% for the next 10 minute's prices. However, there is no cross-validate in this research which might cause the obtained models to be overfitting.

Greaves et al [9] propose transaction graph data to predict the Bitcoin's prices. By collecting Bitcoin transactions, this research uses Linear Regression, Logistic Regression, SVM, Neural Network generating models to predict the prices. The result of accuracy is only 55% since the exchange behavior which directly affects the prices is not included in the transactions. Therefore, this research recommends including the exchange behavior into the transaction to increase the accuracy.

Almeida et al [10] propose artificial neural networks models to predict trends of tomorrow's Bitcoin prices. The models are generated by using the history open-source dataset from Quandl and Theano library from MATHLAB. In two years of exchange, the profit of 8000 USD from the models is gained.

McNally et al [11] present various modeling experiments on Long Short-Term Memory (LSTM), Recurrent Neural Network (RNN), and AutoRegressive Integrated Moving Average (ARIMA). The models are generated depending on the Open, High, Low, and Close data from CoinDesk and hash rate data inside Blockchain. The results show the highest accuracy at 52.78% and the Root Square Mean Error (RMSE) at 5.45%.

IV. Methodology

A. Data Collection

Machine learning models in this research use Bitcoin transaction data from the bitstamp website with publishing on the Kaggle website. However, in this research, 1-minute interval trading exchange data rate in USD from January 1, 2012 to January 8, 2018 is focused. The datasets are in CSV files.

B. Feature Selection

Features of the datasets from *A* are as follows:

Features	**Definition**
Close	latest trade
Open	opening trade
High	highest trade during day
Low	lowest trade during day
Weighted price	mean Bitcoin price
Volume_(BTC)	total trade volume of day in BTC
Volume_(Currency)	total trade volume of day in USD
Timestamp	data recorded time

In this research, the scikit-learn library is used to create models with only features Close, Open, High, and Low when the Weighted price is to predict.

From 1-minute interval trading exchange data in 3,161,057 rows, we adjust them into 2,195 rows of 1-day interval trading exchange data instead. Next, we divide them into a training set and a test set with the ratio of 70:30. Hence, the final training set is 1,756 and the test set is 439. From the above features, we calculated the correlation coefficient of all features and visualized graphs in Figure 2 and 3. From the graphs, all features are highly correlated, therefore we will create models with all features.

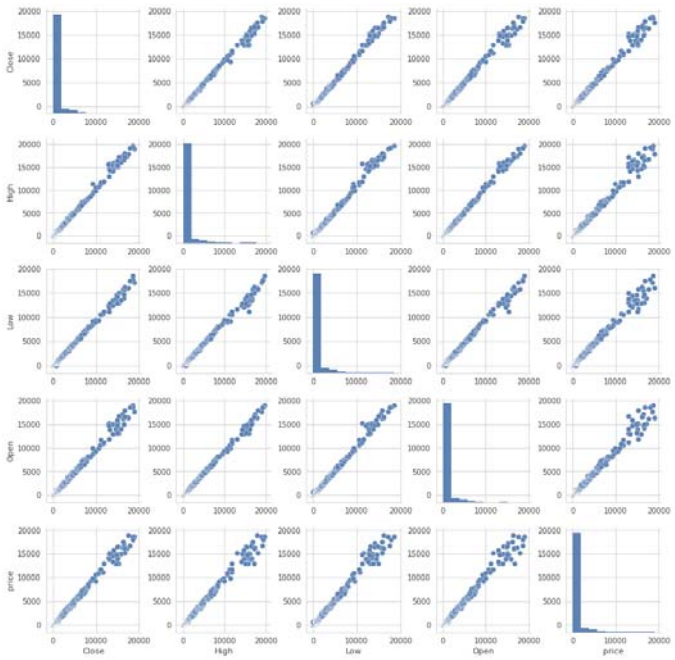

Fig. 2. Pair plot correlation coefficient for each parameter of features

Fig. 3. Heat map correlation coefficient values for each parameter of features

C. Data Preparation

Min-Max scaler is to adjust data into 0 to 1 by using min and max of the data [12]. MinMaxScaler in scikit-learn can use the equation 1 to transform data of selected features in 4.2.

$$x_{sc} = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (1)$$

Changing from 2 dimensions to 3 dimensions by adding time dimension is to be able to use the obtained data for LSTM and GRU modeling.

D. Modeling

In this research, we chose regression machine learning due to continuous values of Bitcoin price. With the scikit-learn library, the best two regression models; Theil-Sen Regression and Huber Regression were selected to compare. For deep learning based regression models, Keras library was used to create LSTM and GRU models.

[1] Theil-Sen Regression

Theil-Sen regression is the method which uses a median of the slope of all lines through pairs of data points. For this reason, the outliers can grow up to 29% for 2-dimensional data or pairs of data. However, if the dimension of data increases, the robust of outliers will be decreased [13].

Parameters of scikit-learn modeling are firstly set by Theil-Sen regression as follows whereas an example of Python code is shown in Figure 4.

Parameter	**Value**
C/Maximum step size (regularization)	1.0
loss/ loss function	epsilon insensitive
epsilon (Threshold)	0.1

```
ts = linear_model.TheilSenRegressor()
ts.fit(Xtrain_d,ytrain_d)

ytest_ = ts.predict(Xtest_d)

for i in range(len(ytest_)):
    label = price_scaler.inverse_transform(ytest_d[i].reshape(-1,1)).flatten()
    prediction = price_scaler.inverse_transform(ytest_[i].reshape(-1,1)).flatten()

yr_ = price_scaler.inverse_transform(ytest_.reshape(-1,1)).flatten()
yr = price_scaler.inverse_transform(ytest_d.reshape(-1,1)).flatten()

print(sqrt(mean_squared_error(y_pred=ytest_,y_true=ytest_d)))
print(mean_squared_error(y_pred=ytest_,y_true=ytest_d))
print(r2_score(ytest_d, ytest_))

plt.plot(yr,label="Real price",linestyle='--')
plt.plot(yr_,label="predict price",linestyle='-')
plt.xlabel('Time (hr)')
plt.ylabel('Price (USD)')
plt.title('TheilSenRegressor')
plt.legend(loc='best')
plt.show()
```

Fig. 4. Example code for Theil-Sen regression modeling

[2] Huber Regression

Huber regression uses a linear loss to separate between the outlier and inlier data. An outlier means its weight less than inlier's weight [14].

Parameters of scikit-learn modeling are firstly set by Huber regression as follows whereas an example of Python code is shown in Figure 5.

Parameter	**Value**
epsilon (Outliers)	1.35
alpha (regularization parameter)	0.0001

```
hr = linear_model.HuberRegressor()
hr.fit(Xtrain_d,ytrain_d)

ytest_ = hr.predict(Xtest_d)

for i in range(len(ytest_)):
    label = price_scaler.inverse_transform(ytest_d[i].reshape(-1,1)).flatten()
    prediction = price_scaler.inverse_transform(ytest_[i].reshape(-1,1)).flatten()

yr_ = price_scaler.inverse_transform(ytest_.reshape(-1,1)).flatten()
yr = price_scaler.inverse_transform(ytest_d.reshape(-1,1)).flatten()

print(sqrt(mean_squared_error(y_pred=ytest_,y_true=ytest_d)))
print(mean_squared_error(y_pred=ytest_,y_true=ytest_d))
print(r2_score(ytest_d, ytest_))

plt.plot(yr,label="Real price",linestyle='--')
plt.plot(yr_,label="predict price",linestyle='-')
plt.xlabel('Time (hr)')
plt.ylabel('Price (USD)')
plt.title('HuberRegressor')
plt.legend(loc='best')
plt.show()
```

Fig. 5. Example code for Huber regression modeling

RNN is a neural network algorithm suitable for time series modeling [15]. Normally it keeps details of state in a hidden state. The previously hidden state is used to calculate the current state and use the current hidden state to calculate the next state as shown in the equation 2 and 3.

$$h_t = \sigma(x_t W + h_{t-1} U) \quad (2)$$

$$o_t = \sigma(h_t V) \quad (3)$$

When t is time, h is a hidden state, x means input data, o means output data, σ is activation function, W, U, V is a weight matrix for calculating input, previously hidden state, and current hidden state respectively. The structure of RNN nodes can be described as shown in Fig 6.

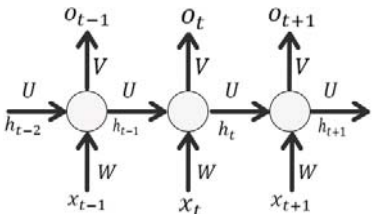

Fig. 6. Structure of nodes in RNN

Thus, RNN is suitable for sequence data or series time data, but it has a problem with long-term dependency data which cause vanishing gradient.

[3] Long short-term memory (LSTM)

LSTM is developed to solve the vanishing gradient problem in RNN [16]. The cell state and hidden state are used to collect and send data to be processed in the next state. Input, output, and forget gates are to define whether the data can pass through or not depending on data's priority. For these reasons, vanishing gradient can be protected as described in the equation 4 to 8.

$$i = \sigma(x_t W_i + h_{t-1} U_i) \quad (4)$$

$$f = \sigma(x_t W_f + h_{t-1} U_f) \quad (5)$$

$$o = \sigma(x_t W_o + h_{t-1} U_o) \quad (6)$$

$$c_t = (c_{t-1} \times f) + (i \times \sigma(x_t W_c + h_{t-1} U_c)) \quad (7)$$

$$h_t = \sigma(c_t) \times o \quad (8)$$

When i is input gate, f is forget gate, o is output gate, c is cell state, h is hidden state, σ is Activation function, W and U are weight matrix, and t is time. The structure of LSTM nodes can be described as shown in Fig 7.

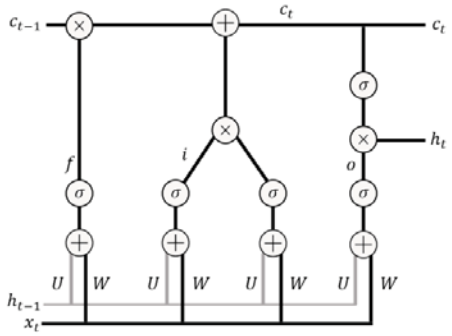

Fig. 7. Structure of nodes in LSTM

Parameters of Keras modeling are firstly set by LSTM as follows whereas an example of Python code is shown in Figure 8.

Parameter	Value
epochs	480
batch size	160
level	1
neurons	259
activation	relu
kernel initializer	glorot uniform

```
import math
from keras.initializers import lecun_uniform
NUM_EPOCHS = 480
BATCH_SIZE = 160
smodel = Sequential()
smodel.add(LSTM(259, input_shape=(1,4), go_backwards=True,
               activation='relu',return_sequences=False))
smodel.add(Dense(1))


smodel.compile(loss="mean_squared_error",optimizer="adam",
               metrics=["mean_squared_error"])

train_size = (Xtrain.shape[0]//BATCH_SIZE) * BATCH_SIZE
test_size = (Xtest.shape[0]//BATCH_SIZE) * BATCH_SIZE
Xtrain, Ytrain = Xtrain[0:train_size], Ytrain[0:train_size]
Xtest, Ytest = Xtest[0:test_size], Ytest[0:test_size]
print(Xtrain.shape, Xtest.shape, Ytrain.shape, Ytest.shape)
for i in range(NUM_EPOCHS):
    #print("Epoch {:d}/{:d}".format(i+1,NUM_EPOCHS))
    smodel.fit(Xtrain, Ytrain, batch_size=BATCH_SIZE, epochs=1,
               validation_data=(Xtest,Ytest), shuffle=False, verbose=0)
    #smodel.reset_states()
score,_ = smodel.evaluate(Xtest,Ytest, batch_size=BATCH_SIZE, verbose=0)
rmse = math.sqrt(score)
print("\n MSE: {:.3f}, RMSE: {:.3f}".format(score,rmse))
```

Fig. 8. Example code for LSTM modeling

[4] Gated Recurrent Unit (GRU)

In 2004, Kyunghyun Cho et al [17] presented GRU which is developed from LSTM. GRU has less complex structure than LSTM by adjusting gate in LSTM to reset gate and update gate. A reset gate is used to determine how much previous state data can be used with current input data whereas an update gate is to determine how much to collect the previous state. The structure of GRU nodes can be described as shown in Fig 9.

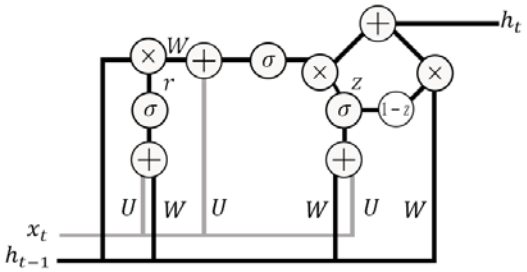

Fig. 9. Structure of nodes in GRU

Parameters of Keras modeling are firstly set by GRU as follows whereas an example of Python code is shown in Figure 10.

Parameter	Value
epochs	480
batch size	160
layer	1
neurons	19
activation	relu
kernel initializer	glorot uniform

```
import math
from keras.initializers import lecun_uniform
NUM_EPOCHS = 480
BATCH_SIZE = 160
smodel = Sequential()
smodel.add(GRU(231, input_shape=(1,4), go_backwards=True,
               activation='relu',return_sequences=False))
smodel.add(Dense(1))


smodel.compile(loss="mean_squared_error",optimizer="adam",
               metrics=["mean_squared_error"])

train_size = (Xtrain.shape[0]//BATCH_SIZE) * BATCH_SIZE
test_size = (Xtest.shape[0]//BATCH_SIZE) * BATCH_SIZE
Xtrain, Ytrain = Xtrain[0:train_size], Ytrain[0:train_size]
Xtest, Ytest = Xtest[0:test_size], Ytest[0:test_size]
print(Xtrain.shape, Xtest.shape, Ytrain.shape, Ytest.shape)
for i in range(NUM_EPOCHS):
    #print("Epoch {:d}/{:d}".format(i+1,NUM_EPOCHS))
    smodel.fit(Xtrain, Ytrain, batch_size=BATCH_SIZE, epochs=1,
               validation_data=(Xtest,Ytest), shuffle=False, verbose=0)
    #smodel.reset_states()
score,_ = smodel.evaluate(Xtest,Ytest, batch_size=BATCH_SIZE, verbose=0)
rmse = math.sqrt(score)
print("\n MSE: {:.3f}, RMSE: {:.3f}".format(score,rmse))
```

Fig. 10. Example code for GRU modeling

V. RESULTS

There are two of the most common metrics used to measure accuracy for continuous variables, Mean Squared Error (MSE) and R-Square (R^2). Table 1 shows the MSE and R^2 of all our implemented models while table 2 shows the calculated time of all our implemented models. The results show that deep learning based regression models: GRU and LSTM give the better result than Theil-Sen regression and Huber regression. GRU give the best results of MSE at 0.00002 and R^2 at 0.992 or 99.2%. Whereas, the calculated time that Huber regression use is much less than LSTM and GRU.

TABLE 1. MSE AND R^2 OF ALL IMPLEMENTED MODELS

Model	MSE	R^2
Theil-Sen Regression	0.000375	0.99176
Huber Regression	0.000373	0.99179
LSTM	0.000431	0.992
GRU	0.00002	0.992

TABLE 2. TIME OF ALL IMPLEMENTED MODELS

Model	Time (sec)
Theil-Sen Regression	0.9018
Huber Regression	0.0002
LSTM	111.0601
GRU	85.2718

Table 3 shows the comparison between bitcoin prices and predicted prices of all our implemented models from the first 20 days of test datasets.

TABLE 3. COMPARISON BETWEEN BITCOIN PRICES AND PREDICTED PRICES OF ALL IMPLEMENTED MODELS

Bitcoin Price (USD)	Theil-Sen Regression Predicted Price (USD)	Huber Regression Predicted Price (USD)	LSTM Predicted Price (USD)	GRU Predicted Price (USD)
672.625	678.239	676.281	677.108	**674.093**
682.195	699.690	695.557	690.657	**687.543**
684.804	709.765	706.897	704.314	**701.049**
712.237	710.672	709.035	**710.925**	707.554
701.076	741.917	736.290	730.109	**726.588**
704.956	**727.147**	728.359	733.379	729.774
727.068	732.725	730.324	**728.024**	724.518
726.058	756.211	752.010	747.952	**744.239**
700.377	753.370	752.170	754.145	**750.336**
692.705	**728.023**	732.672	739.584	735.932
704.181	719.905	720.058	720.418	**716.989**
705.016	730.748	728.050	728.981	**725.423**
705.256	731.022	**730.307**	734.268	730.643
707.527	730.220	**729.590**	733.603	729.986
724.068	733.377	732.123	735.318	**731.687**
709.290	751.523	749.190	750.929	**747.14**
715.907	**734.961**	736.695	743.72	739.993
702.076	742.430	740.617	743.391	**739.674**
698.656	**727.344**	729.007	736.267	732.612
700.143	725.598	724.678	725.288	**721.793**

VI. CONCLUSION

From the results of all our implemented GRU shows the best accuracy but takes calculated time more than Huber regression. However, setting parameters and the total number of datasets can affect the results. In addition, the selected features: Open, Close, High, and Low may not be enough to predict the Bitcoin prices since various factors, such as the reactions from social media, policies, and laws that each country announces to deal with digital currency can all contribute to the rise and fall of Bitcoin prices. Therefore, to the best results of all models, datasets should be always collected updated data and appended.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] G. Heleman, M. Rauchs, “Global cryptocurrency benchmarking study,” Cambridge Centre for Alternative Finance (2017).

[3] G. Neil, H. Halaburda. "Can we predict the winner in a market with network effects? Competition in cryptocurrency market," Games, vol.7 no.3 , 2016, pp. 16.

[4] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. Vanderplas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," arXiv preprint arXiv:1309.0238, 2013.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G S. Corrado, A. Davis, J. Dean, M. Devin, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv preprint arXiv:1603.04467, 2016.

[6] C. François, "Keras: Deep learning library for theano and tensorflow," URL: https://keras. io/k, 2015.

[7] D. Shah, K. Zhang, "Bayesian regression and Bitcoin," Communication, Control, and Computing (Allerton), 2014 52nd Annual Allerton Conference on. IEEE, 2014.

[8] I. Madan, S. Saluja and A. Zhao. "Automated Bitcoin Trading via Machine Learning Algorithms," 2015.

[9] A. Greaves, B. Au, "Using the Bitcoin Transaction Graph to Predict the Price of Bitcoin," 2015.

[10] J. Almeida, S. Tata, A. Moser and V. Smit, "Bitcoin prediciton using ANN," June 2015

[11] S. McNally, “Predicting the price of Bitcoin using Machine Learning” Diss. Dublin, National College of Ireland, 2016

[12] S. Patro, K. Sahu. "Normalization: A preprocessing stage," arXiv preprint arXiv:1503.06462, 2015.

[13] X. Dang, H. Peng, X. Wang and H. Zhang "Theil-Sen Estimators in a Multiple Linear Regression Model," Olemiss Edu, 2008.

[14] P. Huber. "Robust statistics." International Encyclopedia of Statistical Science. Springer Berlin Heidelberg, 2011, pp. 1248-1251.

[15] A. Bernal, S. Fok and R. Pidaparthi. "Financial Market Time Series Prediction with Recurrent Neural Networks." 2012.

[16] S. Hochreiter, J. Schmidhuber. "Long short-term memory." Neural computation vol.9 no.8, 1997, pp. 1735-1780.

[17] K. Cho, B. Van Merrienboer, C. Gulcehle, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078, June 2014.