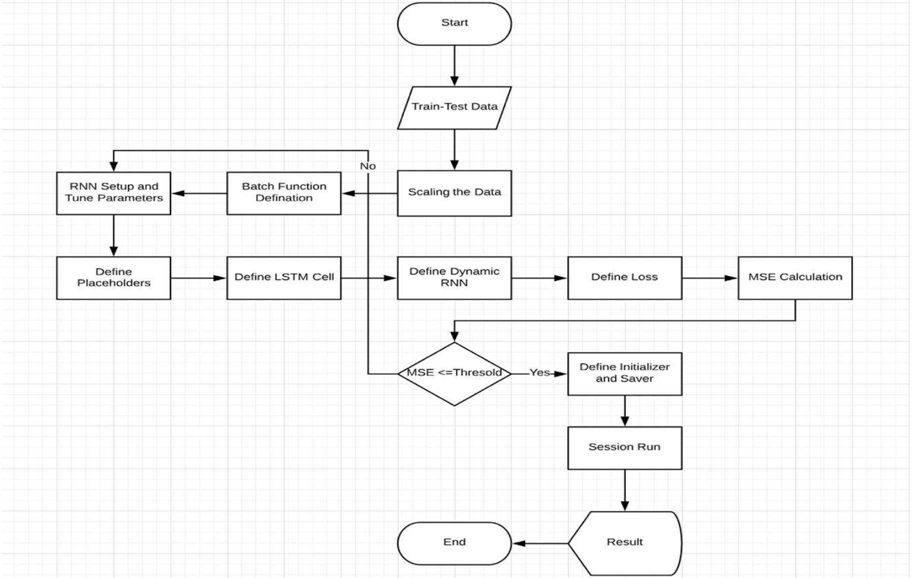
Presentation deck_NN test

Reproduce - Understand Article



Reproduce - Understand Article

Data: Stock price of five companies from "2013-01-01" to "2017-12-31"

> *"In particular, a total of 1259 days of data was collected. The first 1008 data points are for the first 4 years (2013 to 2016) and the last 251 data are for each day in 2017."*

Data Process: Scaling: Min-Max Scale

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

We will need to rescale data back after predicting.

Reproduce - Understand Article

Parameter

```
#Tuning parameters
num_inputs = 1
num_time_steps = 1
num_neurons = 500
num_outputs = 1
learning_rate = 0.002
num_train_iterations = 4000
```

LSTM Cell

```
#LSTM Cell
cell = tf.contrib.rnn.OutputProjectionWrapper(
    tf.contrib.rnn.BasicLSTMCell(num_units=num_neurons, activation=tf.nn.relu),
    output_size=num_outputs)
```

Loss

```
#Loss
loss = tf.reduce_mean(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train = optimizer.minimize(loss)
```

Reproduce - Understand Article

```
for i in range(1,251,+1):
    hold
    print("Pass ",i )

    train_set=stock[i:1008+i]

    #sklearn
    from sklearn.preprocessing import MinMaxScaler
    scaler = MinMaxScaler()
    train_scaled = scaler.fit_transform(train_set)
    #test_scaled = scaler.transform(test_set)
    #batch Defination
    #MSE CAlculation
    with tf.Session(config=tf.ConfigProto(gpu_options=gpu_options)) as sess:
        sess.run(init)

        for iteration in range(num_train_iterations):

            X_batch, y_batch = next_batch(train_scaled,batch_size,num_time_steps)
            sess.run(train, feed_dict={X: X_batch, y: y_batch})

            if iteration % 100 == 0:

                mse = loss.eval(feed_dict={X: X_batch, y: y_batch})
                print(iteration , "\tMSE:", mse)


        saver.save(sess, "./rnn_stock_time_series_model")
    #Session Run
    with tf.Session() as sess:

        saver.restore(sess, "./rnn_stock_time_series_model")

        train_seed = list(train_scaled[-num_time_steps:])
        for iteration in range(num_time_steps):
            X_batch = np.array(train_seed[-num_time_steps:]).reshape(1, num_time_steps, 1)
            y_pred = sess.run(outputs, feed_dict={X: X_batch})
            train_seed.append(y_pred[0, -1, 0])
    #Print Train_seed
    train_seed
    #results
    results = scaler.inverse_transform(np.array(train_seed[num_time_steps:]).reshape(num_time_steps,1))
    hold[i,1]=results
```

Figure 2: The Neural Network Loop

Reproduce - Data

```r
start_date = "2013-01-01"; end_date = "2017-12-31"

sp_prices <- tq_get("NFLX", get = "stock.prices",
                    from = start_date, to = end_date)

stock <- sp_prices |> arrange(date) |>
  select(date, adjusted) |> column_to_rownames('date')

head(stock, 5)
```

```
           adjusted
2013-01-02 13.14429
2013-01-03 13.79857
2013-01-04 13.71143
2013-01-07 14.17143
2013-01-08 13.88000
```

Reproduce - Train-Test Split and Scale

```r
# Max Min Scale
max_min_scale <- function(x, name = 'value') {
  df <- data.frame((x- min(x)) /(max(x)-min(x)))
  colnames(df) <- name; df}
max_min_scale_reverse <- function(y, x) {
  min(x) + y * (max(x)-min(x))}

train_set <- stock[1:1008,]
test_set <- stock[1009:nrow(stock),]
train_scaled <- train_set %>% max_min_scale
head(train_scaled, 3)
```

```
        value
1 0.000000000
2 0.005554876
3 0.004815041
```

Reproduce - Data Preperation for Fitting

```
data_prep <- function(scaled_data, prediction = 1, lag = 22){
  x_data <- t(sapply(1:(dim(scaled_data)[1] - lag - predictio
                   function(x) scaled_data[x: (x + lag - 1), 1]
  x_arr <- array(data = as.numeric(unlist(x_data)),
                  dim = c(nrow(x_data), lag, 1))
  y_data <- t(sapply((1 + lag):(dim(scaled_data)[1] - predicti
                      function(x) scaled_data[x: (x + predictio
  y_arr <- array(data = as.numeric(unlist(y_data)),
                  dim = c(length(y_data), prediction, 1))
  return(list(x = x_arr, y = y_arr))}

x_train = data_prep(train_scaled)$x
y_train = data_prep(train_scaled)$y
cat('x_dim: (', dim(x_train), ')|| y_dim: (', dim(y_train), ')
```

```
x_dim: ( 986 22 1 )|| y_dim: ( 986 1 1 )
```

Reproduce - LSTM Setup

```r
num_neurons = 50
learning_rate = 0.002
# Define LSTM
get_model <- function(){
  model <- keras_model_sequential() |>
    layer_lstm(units = num_neurons,
               batch_input_shape = c(1, 22, 1),
               activation = 'relu',
               stateful = TRUE) |>
    layer_dense(units = 1)

  model %>% compile(loss = 'mse', metrics = 'mae',
    optimizer=optimizer_adam(learning_rate=learning_rate))
}
```

Reproduce - LSTM Structure

```
lstm_model <- get_model()
summary(lstm_model)
```

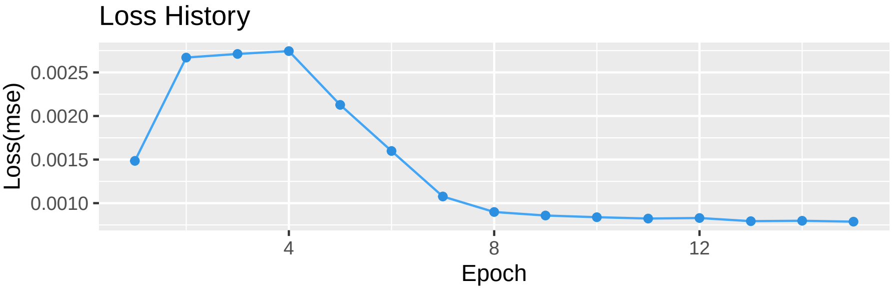
```
Model: "sequential"

--------------------------------------------------------------------
 Layer (type)                        Output Shape
====================================================================
 lstm (LSTM)                         (1, 50)
 dense (Dense)                       (1, 1)
====================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0

--------------------------------------------------------------------
```

Reproduce - LSTM Fitting

```r
set_random_seed(1209)
lstm_model <- get_model()
lstm_model %>% fit(x = x_train, y = y_train, batch_size = 1,
    epochs = 15, verbose = 2, shuffle = FALSE) -> history
```



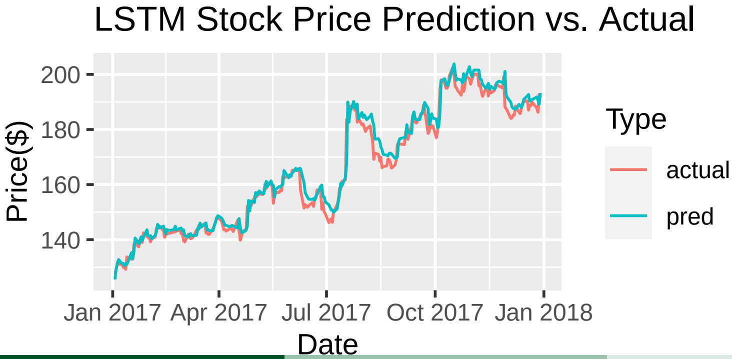
Reproduce - LSTM Result

```r
model_prediction <- function(test_len, whole_data, model){
  t <- test_len # test size
  Tt <- nrow(whole_data) # whole data size
  t_start <- Tt - t + 1 # test start date
  predictions <- vector(length = t)
  for (i in 1:t){
    n <- t_start - 1 + i
    test_set = pull(whole_data)[(n-22):n]
    test_scaled <- test_set %>% max_min_scale
    x_test = data_prep(test_scaled)$x
    y_pred_scaled <- predict(model, x_test, verbose = 0)
    y_pred <- max_min_scale_reverse(y_pred_scaled, test_set)
    predictions[i] <- y_pred
  }
  pred_df <- data.frame('date' = rownames(whole_data)[t_start:
    'pred' = predictions, 'actual' = pull(whole_data)[t_start:
```

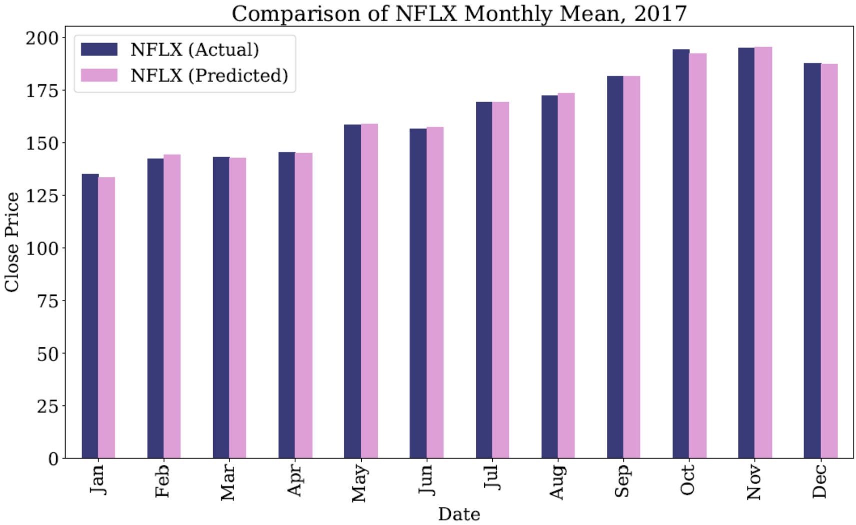
Reproduce - LSTM Result

```
model_prediction(length(test_set), stock, lstm_model) -> pred_
pred_df %>% head(2)
```

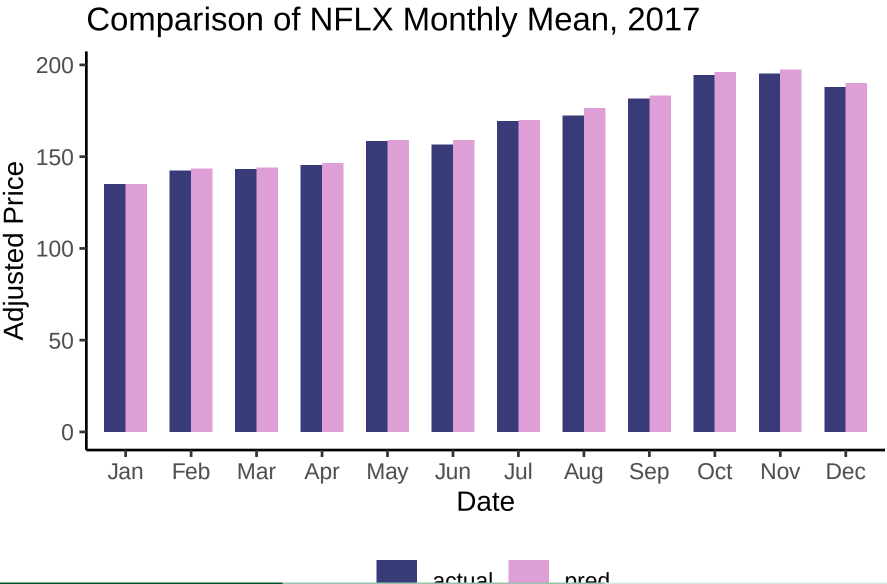
```
        date     pred actual
1 2017-01-03 125.5396 127.49
2 2017-01-04 129.4479 129.41
```



Result Comparison - Article Result



Result Comparison - Our Result



Result Comparison - |% of Error|

	|% of Error|	Article	Our Result
1	mean	2.0000	1.4553167271
2	std	1.5400	1.3480666801
3	variabce	2.3800	1.8172837741
4	min	0.0000	0.0002329987
5	X25_percentile	0.8000	0.5238703257
6	X50_percentile	1.6500	1.1002221473
7	X75_percentile	2.9600	1.9550019530
8	max	9.2500	8.8855777088
9	R_squared	0.9589	1.0556739845

Model Improving

Other Attempt - Variance Model

Other Attempt Movement Model

Model Comparison - ARIMA and LSTM

Thank Your

Any Questions?