ARIMA-Enriched LSTM Multistep Stock Time Series Forecasting with PCA-Tokenized News Articles

Senior Thesis, Statistics, Harvard University

Kenneth Chen

April 1, 2019

Abstract

As financial research has long sought to capture future stock market movements, quantitative metrics are often exploited as soon as they are discovered and arbitrage opportunities are scarce. As an additional means to predict market movements, text data from news articles offers qualitative information that may contain sentiments and investor biases not captured by quantitative metrics alone on which investors may exploit. The psychological nature of news articles has been examined and tokenized to a degree for daily stock market predictions but not extensively for multistep predictions into the future combined with quantitative metrics. Additionally, the statistics and machine learning communities have generally performed their analysis of stock markets using tools from their respective fields only. While machine learning black box algorithms generally are richer than statistical models, their lack of distributional assumptions can lead to poor fitting and numerical instability. In this paper, a novel foundational multistep time series model is built that leverages statistical ARIMA models to provide numerical stability while leveraging the richness of the LSTM machine learning model on stock time series and news articles data.

Contents

1 Introduction

1.1 Background

Under traditional financial market theory, any new information is incorporated quickly into market prices once and for all and arbitrage opportunities are eliminated. However, there are instances where stocks may move with no relation to reasonable quantitative and qualitative information, as seen in the Tesla's (TSLA) dropping stock prices merely due to a comment by Elon Musk that had nothing to do with his company itself. Tesla's history in particular shows a general short-sell sentiment by major financial institutions, and it is unreasonable to posit that new information is incorporated into the current stock price once and for all, without effects into the far future.

As such, there is a case to be made for psychological biases that persist through time in investors, due to sentiment generated by news articles. Investors are not completely rational nor are their decisions dominated completely by numerical analyses of financial markets and company fundamentals, or at the very least, investors are trading based on predicting the irrationality of other market players and competitors. News articles would especially affect large stock market players whom rely on both qualitative and quantitative measures, as opposed to smaller sized algorithmic trading firms that trade mainly on quantitative information and are less sensitive to sentiment in news articles. As such, the large sized market players will react more to news, and by virtue of their size, they will move stock market prices.

An examination is made for traditional theory holding in that news information should be completely incorporated into new stock prices, or if news information persists over long periods of time due to investor biases.

Multistep predictions of individual stocks will be performed, with different combinations of first order differenced historical price and tokenized news articles using (1) ARIMA, (2) LSTM with Stock Prices, (3) LSTM with Stock Prices and News Articles, and (4) ARIMA-Enriched LSTM with Stock Prices and News Articles.

1.2 Related Work

Recurrent Neural Networks (RNNs) are used extensively as models for prediction tasks where ordering of the inputs matters, unlike in traditional Neural Networks. This particular class of machine learning models applies well to time series, as observations are ordered. Previous research on news articles include Bag of Words models to tokenize news articles and, in combination with the previous day's stock price change, to predict same day percentage change for related stocks. [1]. Leveraging RNN-based machine learning, additional research on news articles has been done to predict daily binary outcomes on the Dow Jones Industrial Average increasing or decreasing [2].

While research on using news articles has been performed to predict daily stock market movements, there is a dearth of research on how news articles affects long-term stock market movements. Moreover, much of the relevant literature focuses on classification prediction tasks (stock rises or falls), and there is a little literature for regression prediction tasks for stock prices using RNNs, especially combining quantitative data (stock prices) and qualitative data (news articles) as the inputs to the RNN.

One particular form of a RNN, called Long Short Term Memory (LSTM), precludes vanishing and exploding gradient issues that arise when fitting the weights of the model in traditional RNNs. In the particular case of long historical time series, there is a good chance that propagating the hidden layer through many timesteps in a traditional RNN will result in vanishing or exploding gradients. As such, LSTM, which performs additive updates rather than multiplicative updates, helps eschew this issue and will be utilized as the baseline for the machine learning models in this paper.

2 Data Collection

2.1 Stock Data

Historical intraday stock time series data were obtained and persisted into a SQLite database through scraping Yahoo Finance, which gives date, opening, close, adjusted close, and volume on a daily basis. Only adjusted close prices are used, in order to account for dividends and any corporate actions that occurred prior to the next day's open. These adjusted prices give an accurate representation of the equity value of the firm and will be referred to as stock prices.

While data for the stocks from the S&P 500 were persisted into the database, only four stocks in the S&P 500 plus Tesla stock are examined, where the five companies together do not traditionally treat each other as competitors in their respective markets. This allows for generalizability of the results via fairer evaluation of different results of the models in this paper, so that results are not driven by the importance of news information in one particular industry. These stocks are AAPL (Apple), AMZN (Amazon), DIS (Disney), GS (Goldman Sachs), and TSLA (Tesla).

For sake of comparison, stock prices will begin cover an eight year span from June 29, 2010 to December 12, 2018, when all five companies have been publicly traded.

2.2 News Article Data

Historical news articles were obtained and persisted into pickled JSON objects through the New York Times (NYT) API. A single API call returns a nested JSON object that returns basic facts of all NYT articles in a given month for a given year. Pertinent facts include date, article URL, and keywords.

For each of the five companies, the wrangling code iteratively called the NYT API over June 29, 2010 to December 12, 2018. Within each API call, i.e. within each month for a given year, relevant articles to that company were found by finding a match between the company name and any one of the keyword tags associated with each article. If such a match was found, the wrangling code would then scrape all the article body text from the URL object for that article and pickle that data for later use in the analysis. As such, titles, headlines, descriptions to other articles, and captions to images were ignored, and the text scraped captures the true coherent message of the news article. Links to videos and images on NYT were also ignored.

3 Data Representation

3.1 Data Representation for Stock Prices

For clarity of notation, no indexing for the five stocks will be made for any formulas, but the formulas and variables of course apply to each of the five different stocks separately.

In order to avoid fitting to trends in the stock price time series rather than variations and to handle unit-root nonstationarity for ARIMA models, first order differences of lag 1 are used to eliminate clear trends and move the time series towards stationarity. That is,

$$\tilde{p}_t = p_{t+1} - p_t$$

where p_t is the untransformed stock price and $\tilde{p}_t$ is the first order differenced stock price, for $t \in \{1, \ldots, L\}$, and $L = 2142$ is the length of the times series data, or the number of trading days for the five stocks from June 29, 2010 to December 12, 2018

As will be detailed later in the Models section, LSTM models use tanh activation functions with outputs consequently restricted between -1 and 1, inclusive. As the outputs of the prediction task are thus the stock prices represented on a scale between -1 and 1, this requires us to rescale time

series $\tilde{\mathbf{p}} = \tilde{p}_1, ..., \tilde{p}_L$ to be within this range, which is done using the MinMaxScaler from sklearn library. Specifically, with $a = -1$ and $b = 1$, for each individual $\tilde{p}_t$,

$$\tilde{p}_{t,\text{std}} = (\tilde{p}_t - \min(\tilde{\mathbf{p}}))/(\max(\tilde{\mathbf{p}}) - \min(\tilde{\mathbf{p}}))$$
$$y_t = \tilde{p}_{t,\text{std}} * (b - a) + a$$

and y_t is bounded between a and b, since $0 \leq \tilde{p}_{t,\text{std}} \leq 1$. MinMaxScaler is used instead of standardization based on a Normal distribution, in order to bound all y_t to be between a and b, inclusive of the endpoints. The first ordered differenced, scaled stock prices will now be referred to as the transformed stock prices.

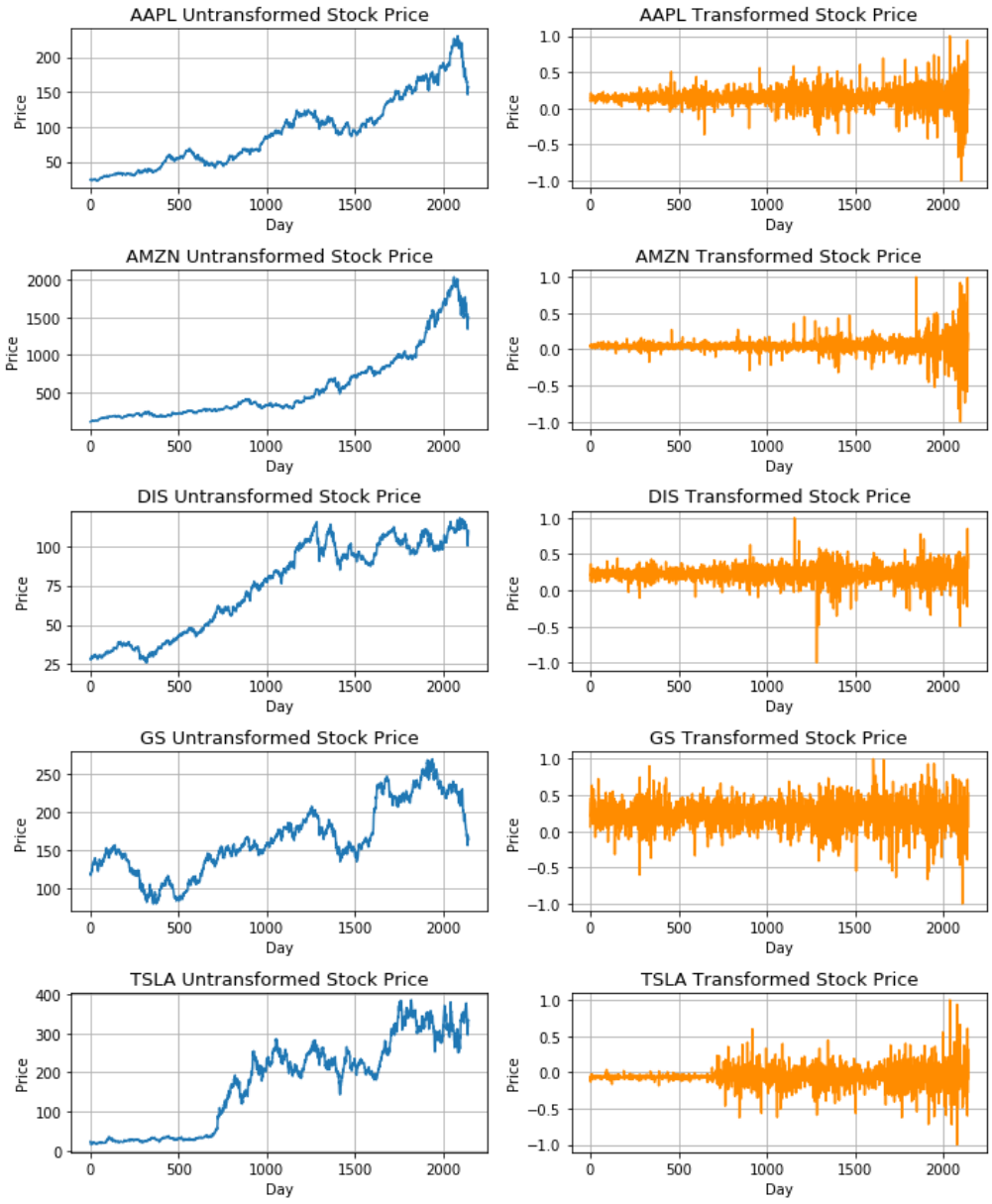


Figure 1: Stock Prices in Original & First Order Differenced and Scaled Spaces

While the transformed stock prices for the five stocks exhibit mean stationary, variance stationarity is not as apparent, and the robustness or sensitivity of the models will be considered in light of the lack of strong variance stationarity.

Stock prices from now on will refer to the first order differenced and scaled space, unless otherwise specified.

3.2 Data Representation for News Articles

A Bag of Words model with Principal Component Analysis was used to tokenize news articles as numerical objects. The Bag of Words model counts the number of occurrences of a word in a news article and acts as a proxy for the density of the word distribution in a given news article.

Using all words in the English language can be unwieldy in large matrix computations and memory allocation. As such, for each stock, a dictionary is first built over all the persisted words in its news articles from June 29, 2010 to December 12, 2018. Rather than using all words in the news articles and weighting the words using TF-IDF, or Term Frequency-Inverse Data Frequency, common words, such as "the", "a", "an", "in", are ignored by comparing potential words to add to each stock's dictionary via stopwords from NLTK stopwords, or Natural Language Toolkit. Using NLTK over TF-IDF helps save on computation time for the LSTM models involving news articles, while removing words that have little lexical context that fails to distinguish news articles from one another [3]. Punctuation and non-alpha characters were ignored, and all letters are lower-cased so that capitalization does not differentiate between words spelled in the same way otherwise.

The data representation $\mathbf{A}_{L\times K}$ for the news articles is created iteratively over the news articles for a particular stock, where $L = 2142$ and K is the number of unique words in the lexical corpus for the stock of interest, where $A_{l,k}$ corresponds to the l'th trading day having $A_{l,k}$ occurrences of the word k in news articles involving the company for that trading day.

$\mathbf{A}$ represents a distribution of the words over each day but is sparse. This can lead to optimization challenges for gradient descent steps for LSTMs since the matrix is not sufficiently smooth in a multidimensional space. Thus, PCA (Principal Component Analysis) decomposition is used to linearly reduce the dimensions of $\mathbf{A}$ via Singular Value Decomposition. To preserve over three quarters of the variance in the principal components, the number of principal components chosen was 200, and the exact full SVD solver is used to perform the PCA. Let the PCA reduced matrix be $\tilde{\mathbf{A}}_{L\times 200}$.

While the onegram Bag of Words model acts as a baseline, the disadvantage in counting individual words is that word semantics, i.e. the relationship between words, is lost. As a result, the meaning of words in a document may not accurately represent the numerical feature space generated from the steps above. To address this, in latter finer-tuned models, bigram and trigram Bag of Words models are also used, where the counts are of two word and three word sequences, respectively, excluding stopwords from NLK. Only the length(number of onegrams) most common bigrams and trigrams are considered, for fairness of comparison to the onegram tokenization in the models. PCA dimensionality reduction is performed in the exact same way as in the onegram case.

Table 1: Variance Explained By 200 PCA Components

Tokenization	AAPL	AMZN	DIS	GS	TSLA
Onegram	.798	.797	.872	.892	.965
Bigram	.700	.826	.641	.891	.939
Trigram	.865	.774	.919	.962	.969

Table 2: Dictionary Size

Metric	AAPL	AMZN	DIS	GS	TSLA
Number of Unique Words	35088	32566	29034	27980	16192

4 Mathematical Formulation of Prediction Task

4.1 Generating Train and Test Sets

Define n to be the number of data points available for training a particular sample. This will be called the training data lag length n. Define m to be the maximum number of steps to make the multistep prediction for into the future, from the last price in the training data. This will be called the prediction length m. The sample generation process will now be defined.

For a given company, the untransformed historical intraday stock prices have a length of L days. After first order time differencing and scaling, the time series is of length $L-1$. To create the train and test sets, all unique sub time series of length $n+m$ from the transformed time series are found. The first 70% of these samples are train set, and the last 30% of these samples are test set. Figure 2 below illustrates this process.

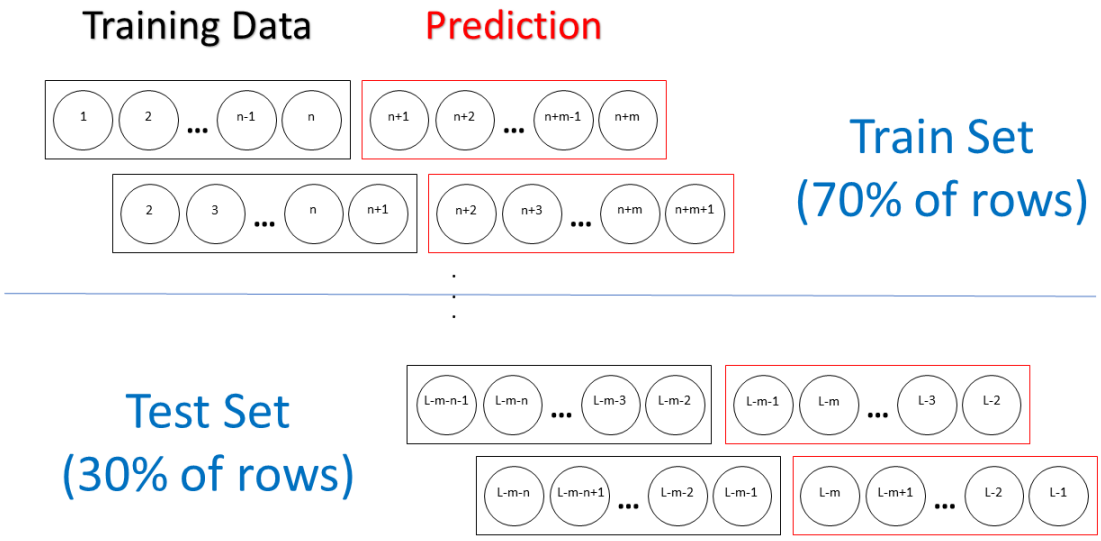


Figure 2: Creating Train and Test Sets

Each node represents the information of the stock at a point in time, which is stock price in the univariate case and stock price with news article representation at that point in time for the multivariate case. Each row represents a sample. Note the distinction between the definitions for training data and train set. The training data in the train set always begins at time points before training data in the test set, as it makes sense to only evaluate the models on samples beginning at time points strictly later than beginning time points in the samples used to train the models.

Models are trained using the train set, where for each sample, i.e. row in the train set, the training data of length n is used to make m predictions into the future that are evaluated against

the corresponding true m values while fitting the models. After training the models, each sample in the test set is used to perform and evaluate the loss of a multistep prediction of length m.

As the problem is a prediction task and not an inference task, the potential for serial correlations in the samples generated in this process is not relevant.

For simplicity of notation, equations and variables will now refer to each individual sample of length $n + m$ but of course apply to all samples generated from the above process.

4.2 Data Hyperparameters

Data hyperparameters include the training data lag length n, and prediction length into the future m.

Since the interest lies in long-term multistep forecasting, and not short-term forecasting, $n = 60$ and $m = \{5, 20, 60\}$. This corresponds to using information over the last 60 days, which corresponds to the length of one financial quarter, and performing multistep predictions for one week, one month, and one quarter into the future. In order to establish a baseline model, a simplifying assumption is made in that the location of the financial quarter reportings date in each sample does not eliminate or downweight the importance of data prior to that point.

A summary of the number of samples in the train and test sets is provided below.

Table 3: Number of Samples by Setup

n	m	Number of Train Samples	Number of Test Samples
60	5	1454	623
60	20	1444	618
60	60	1416	606

4.3 Retransformation of Predictions

Fitting and predicting in the transformed space represents a prediction task for first order differenced time series that are scaled. Let the predictions for one sample be $\hat{y}_{n+1}, ..., \hat{y}_{n+m}$. These predictions were then reverted to the original, untransformed space using the inverse transformations in order to obtain predictions for the original stock prices. Reverting back to the first order differenced space is calculated as

$$\hat{\tilde{p}}_{t'} = \frac{(\hat{y}_{t'} - a)(\max(\tilde{\mathbf{p}}) - \min(\tilde{\mathbf{p}}))}{b - a} + \min(\tilde{\mathbf{p}})$$

where $t' \in \{n + 1, ..., n + m\}$. $\tilde{\mathbf{p}}$, a, and b are defined as they are in Section 3.1.

Then, reverting to the original space, for the first prediction at $t' = n + 1$,

$$\hat{p}_{n+1} = \hat{\tilde{p}}_{n+1} + p_n$$

Then recursively, for the other $m - 1$ predictions where $t' \in \{n + 2, ..., n + m\}$,

$$\hat{p}_{t'} = \hat{\tilde{p}}_{t'} + \hat{p}_{t'-1}$$

4.4 MSE Train Evaluation

There is a prediction vector $\hat{\mathbf{p}}_{t'} = \hat{p}_{n+1}, ..., \hat{p}_{n+m}$ for each sample in the original space. With $t' \in \{n + 1, ..., n + m\}$, define the squared error for each sample $s \in S$ as,

$$\text{SE}_s = (\hat{\mathbf{p}}_{t'} - \mathbf{p}_{t'})^2$$

where the squaring operation is performed element-wise. Thus, SE_s will be a vector of length m that represents the squared error at each of the m timesteps into the future for one sample. When S is the set of samples in the train set, the MSE values are used to perform gradient steps during fitting. As will be discussed for LSTM models, it turns out that the sample size is 1 for each gradient step, so the scalar MSE_s for a sample is,

$$\text{MSE}_s = \frac{\sum \text{SE}_s}{m}$$

for $s \in S$. These MSE values represent how inaccurate the model is on average across each of the predictions of 1 to m timesteps into the future from the end of the training data in the same sample, with equal weighting across each future prediction timestep.

4.5 RMSE Test Evaluation

When S is the set of samples in the test set, the RMSE vector for each model is used to evaluate how well the models performed compared to each other. Thus, using all test samples in S, define the RMSE as,

$$\text{RMSE} = \sqrt{\frac{\sum_{s \in S} \text{SE}_s}{\text{length}(S)}}$$

where the summation is performed element-wise, then the division operation is performed element-wise, and then the square root operation operation is performed elementwise. Thus, RMSE will be vector of length m, corresponding to the error in the multistep forecasts from 1 to m steps into the future on average, across all test samples.

5 ARIMA Baseline Model

As stock prices are first order differences, ARIMA models with initial differencing step parameter equal to 1 will be used. No further differencing will be allowed, in order to have a fair comparison to LSTM models based on the first order differenced stock prices. Thus, the fitting of ARIMA models are focused on the most optimal p and q, the hyperparameters for the order to the autoregressive model and the order of the moving average model, respectively.

Only the test set in used for the ARIMA baseline. For each sample in the test set, the training data is used to fit an ARIMA$(p, 1, q)$ model while performing an AIC stepwise grid search over the hyperparameters p and q, as manual examination of the ACF and PACF plots is not practical over so many samples in the test set.

The best hyperparameters p^* and q^* for each sample were determined by the lowest AIC over a stepwise search. Lower model complexity is preferred, so the stepwise model begins at $p = 1$ and $q = 1$ (rather than high values of p and q, and the grid search will terminate once the AIC no longer decreases in any stepwise direction. The model using the best hyperparameters was used to perform predictions m steps into the future.

Predictions are calculated recursively one step at a time into the future. Formally, for $t \in \{n + 1, ..., n + m\}$,

$$\hat{y}_t = \phi_0 + \sum_{i=1}^{p^*} \phi_i y^*_{t-i} - \sum_{j=1}^{q^*} \theta_j y^*_{t-j}$$

where $y^*_z = y_z$ for $z \in \{1, ..., n\}$ and $y^*_z = \hat{y}_z$ for $z \in \{n+1, ..., n+m\}$. ϕ_0 is the fitted intercept, $\phi_1, ..., \phi_{p^*}$ are the fitted AR coefficients, and $\theta_1, ..., \theta_{q^*}$ are the fitted MA coefficients.

The RMSE evaluation for the m steps is then performed over all samples in the test set.

6 LSTM Overview

6.1 LSTM Unit

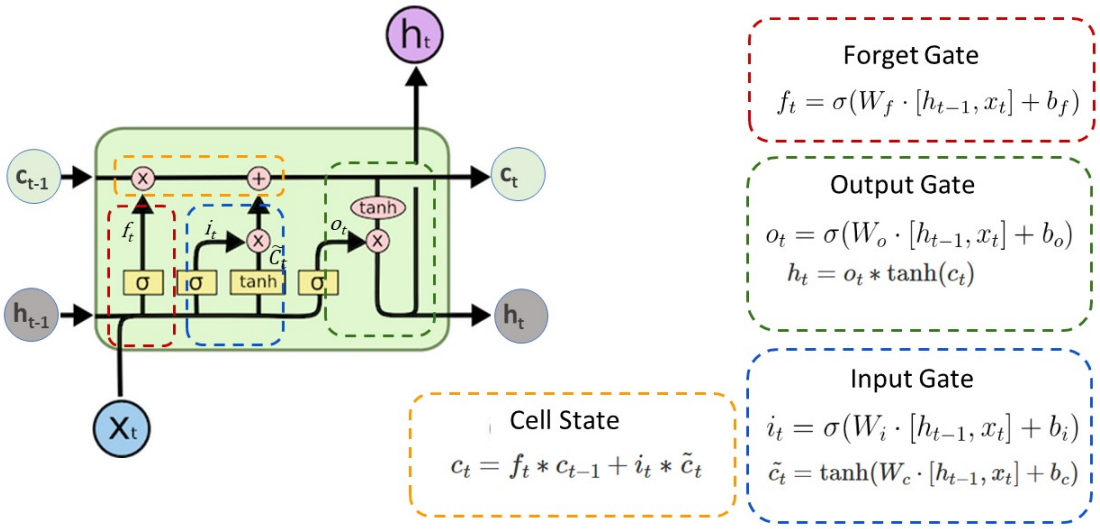


Figure 3: Diagram of a LSTM unit [5]

Before introducing the LSTM models in this paper, which mainly differ in the input x_t in Figure 3, an overview to the LSTM and the chosen hyperparameters is presented here.

LSTMs are RNNs with a cell and a hidden state, respectively denoted as c and h, indexed by timestep. Both are updated in each timestep and can be thought of as memories that are weighted by three gates. In particular, cell states c work as a long term memory and the updates depends on the relationship between the previous hidden state h_{t-1} and the input x_t.

Let $\sigma(z) = \frac{1}{1+e^{-z}}$ be the sigmoid function and $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ be the hyperbolic tangent function, for all $z \in \mathbb{R}$. Note that the sigmoid function output ranges from 0 to 1 and the tanh function output ranges from -1 and 1.

Let $\cdot$ be the matrix multiplication operator, $*$ be the elementwise multiplication operator, and [] be the concatenation operator along the axis of length 1 of its two arguments. Each part of a LSTM unit is detailed below, referencing the formulas in Figure 3 [6].

6.1.1 Forget Gate

The forget gate tries to estimate what features of the cell state, which captures long-term memory, should be forgotten.

The input x_t at a particular timestep is $d\times1$ vector, where d is the number of features at timestep t. The LSTM Models in this paper primarily vary in the structure of d and will be discussed later.

The previous hidden state h_{t-1} is a $u\times1$ vector, where u is a hyperparameter that represents the number of neurons, i.e the dimension of both the hidden state and cell state vectors. More neurons will allow for more model expressivity but runs the risk of overfitting the parameters on the train set.

The parameters include forget weights W_f of dimension $u\times(u+d)$. It also includes a bias term b_f of dimension $u\times1$. Then, noting that $[h_{t-1}, x_t]$ is of dimension $(u+d)\times1$,

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

f_t is then multiplied elementwise with c_{t-1}. Intuitively, f_t is choosing how much to remember in the cell state c_{t-1} (1 means remember all information in the neuron, 0 means forget all information in the neuron).

6.1.2 Input Gate

The input gate decides what new information will be stored in the long-term cell state.

As with the forget gate, in the input gate there are analogous parameters W_i of dimension $u\times(u+d)$ and b_i of dimension $u\times1$. Then,

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

The tanh portion also includes analogous parameters W_c of dimension $u\times(u+d)$ and b_c of dimension $u\times1$. Then,

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

i_t and $\tilde{c}_t$ are multiplied elementwise and are then added to the cell state. Intuitively, tanh is used to squash and standardize the new candidate values between -1 and 1 to avoid blowing up the proposed new candidate values. i_t is choosing how much of an update to apply from in weighing between 0 and 1 each neuron in the candidate new information $\tilde{c}_t$.

6.1.3 Cell State

Putting the forget and input gates together, the cell state update is,

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

6.1.4 Output Gate

As with the sigmoid portions of the forget and input gates, in the output gate there are analogous parameters W_o of dimension $u\times(u+d)$ and b_o of dimension $u\times1$. Then,

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

tanh is applied to the updated cell state c_t to push its values between -1 and 1, and the new hidden state is,

$$h_t = o_t * \tanh(c_t)$$

Intuitively, o_t is choosing how much of the cell state (long-term memory) is remembered in the new hidden state h_t (short-term memory) via weighing between 0 and 1 each neuron in the tanh'ed cell state c_t. This hidden state h_t is the output of the LSTM unit and is also passed onto the next LSTM unit.

6.2 Vanishing and Exploding Gradients

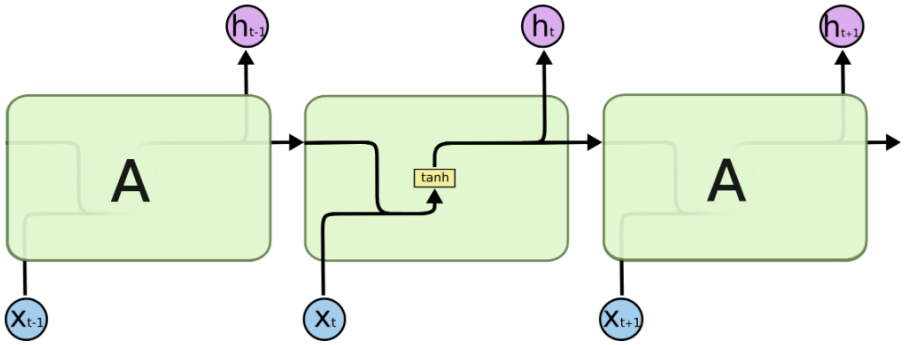


Figure 4: Diagram of a RNN [5]

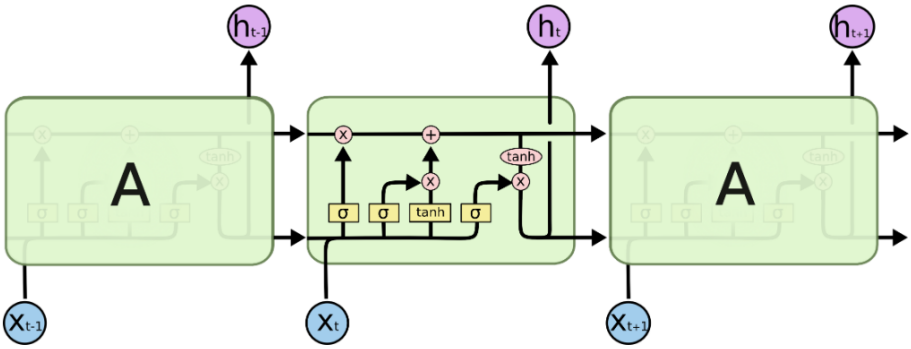


Figure 5: Diagram of a RNN with LSTM units [5]

As mentioned previously, the advantage of using LSTM units over standard RNN tanh units lies in the additive update for the cell state. When backpropagating to optimize the parameters, we do not run into vanishing nor exploding gradients as we do in standard RNN units. In standard RNN units, via chain rule, backpropagation involves taking the product through the derivatives of the tanh units and limits the standard RNN from having a numerically stable long-term dependency on earlier RNN units.

6.3 Final Dense Layer

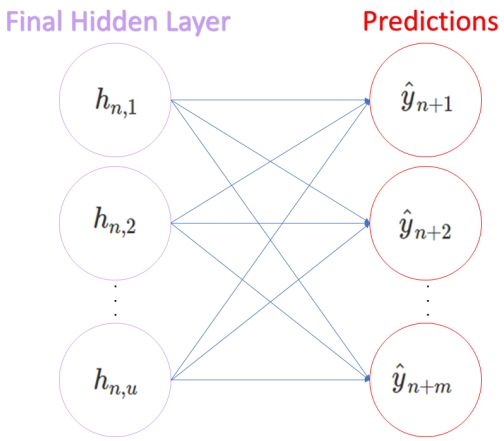


Figure 6: Fully connected hidden-to-dense layer

The output of each LSTM unit at time t is h_t. Only the last hidden layer in the training data, h_n, is used to map to predictions, as the final hidden layer incorporates information from deep learning performed on all of the previous observations in the training data. All of this information in each neuron of the final hidden layer is valuable to predicting each of the m future transformed stock prices from $n+1$ to $n+m$; hence, h_n is connected to a fully dense output of size m, which is the number of steps ahead for the multistep prediction of transformed stock prices. This is one step of a feed forward neural network, and this is how the predictions $\hat{y}_{n+1}, ..., \hat{y}_{m+n}$ are obtained.

Mathematically, each connection between the final hidden layer and the predictions consists of a multivariate linear regression with the bias term included. The parameters are $W_{p,m'}$ of dimension $1 \times u$ and $b_{p,m'}$ of dimension 1×1, for $m' \in \{n+1, ..., n+m\}$. Thus,

$$\hat{y}_{m'} = W_{p,m'} \cdot h_n + b_{p,m'}$$

6.4 Optimization and Prediction Overview

To optimize the LSTM parameters in this section, Keras, an open-source neural-network library written in Python, based on the TensorFlow backend was used. Keras performs the backpropagation necessary to optimize the parameters. Using the train set, his entails taking partial derivatives of the parameters through the linear and nonlinear functions, backwards from the final dense layer. A gradient step is made in the direction of steepest descent in MSE value to update all parameters. Making the gradient steps for all samples in the train set is an epoch, and 50 epochs are used for all LSTM models, for fairness of comparison. The efficient Adam optimizer was used in order to have a better chance of reaching converge quicker in the same number of epochs than the Stochastic Gradient Descent optimizer.

An AWS p2.xlarge instance was used with the Keras CuDNNLSTM layer to utilize a GPU for faster computing. Each model took about 9.5 minutes to fit, using the AWS GPU.

The fitted weights are used when evaluating inputs from the test set into the model, and the RMSE is calculated using the predictions and the true values in the m steps into the future for each sample.

6.5 Hyperparameter Choices and Optimization Details

The relevant hyperparameters for the LSTM layers are the number of epochs, batch size, shuffle, stateful, the number of neurons u [7].

The number of epochs is set to 50, as discussed above in the Optimization and Prediction section.

Batch size is 1. This is because we fit each sample in the train set sequentially, from the sample with the earliest starting point in time to the sample with the second earliest starting point in time, and so on so forth until the last sample in the train set. This corresponds to updating the model in light of a new observation of data at the next time point in the future, and it only makes sense to update our model sequentially in light of this additional future observation in a sliding window of length $n+m$, rather than in larger batches of many samples.

The shuffle parameter in the code for LSTMs is set to false to ensure samples from the train set are processed in the just mentioned sequential structure.

The stateful parameter is set to true, which means that the cell and hidden states computed for the sample in one batch will be reused as initial states for the sample in the next batch. This corresponds to maintaining our state representations from one sample to the next. Once a gradient step is made for each sample in the train set sequentially, the cell and hidden states are reset for the next epoch, where the process is repeated, and so on so forth for a total of 50 epochs.

Along with $m = \{5, 20, 60\}$, a grid analysis is performed over $u = \{10, 50, 100\}$ for completeness of comparison across the five stocks, in order to make sure overfitting or underfitting from the number of neurons is not a limiting factor of comparison.

7 LSTM Models

7.1 LSTM with Stock Prices

The LSTM with Stock Prices baseline model will have $d = 1$, since the input x_t to the LSTM model at each timestep t is just a single transformed scalar stock price y_t from the sample in the batch, where $t \in \{1, ..., n\}$. The prediction vector $\hat{\mathbf{y}}_{t'}$ of size m will be compared via MSE to $\mathbf{y}_{t'}$, for $t' \in \{n+1, ..., n+m\}$, to evaluate the gradient step.

The training data from the test set is fed into the model similarly to make predictions from each sample, which are in turn used to obtain RMSE.

7.2 LSTM with Stock Prices and News Articles

The LSTM with Stock Prices and News Articles model will have $d = 1 + 200 = 201$. $\tilde{\mathbf{A}}$, the onegram tokenization of news articles of dimension $L \times 200$, is concatenated column-wise with the transpose of the original untransformed stock price time series $\mathbf{p} = p_1, ..., p_L$. The combined $L \times 201$ feature matrix $\mathbf{F}$ is then processed by using first order differences and MinMixScalers independently for each column, in order to ensure similar scaling for all input variables, and then splitting the data into train and test sets. Each of the two sets is of dimension (number of samples)$\times(n+m)\times d$, or (number of samples)$\times$(lag length plus prediction length)$\times$(number of features), where the number of samples differs of course between the train and test sets based on Table 3. Notate the train and test matrices as $\tilde{\mathbf{F}}_{train}$ and $\tilde{\mathbf{F}}_{test}$, respectively.

For each sample $\mathbf{S}$ of dimension $(n+m)\times d$ in the batch trained in the LSTM model, the input $\mathbf{x}_t$ is the t'th row of $\mathbf{S}$, where $t \in \{1, ..., n\}$. That is, the t'th row that is fed in as $\mathbf{x}_t$ includes the transformed scalar stock price y_t and the transformed PCA representation of the news articles at time t. The prediction vector $\hat{\mathbf{y}}_{t'}$ of size m will be compared via MSE to $\mathbf{y}_{t'}$, for $t' \in \{n+1, ..., n+m\}$, to evaluate the gradient step, as in the LSTM baseline model.

The training data from the test set is fed into the model similarly to make predictions from each sample, which are in turn used to obtain RMSE.

7.3 ARIMA-Enriched LSTM with Stock Prices and News Articles

The ARIMA-Enriched model is considered only when $n = m = 60$, since a prerequisite to this model is that $n = m$. $d = 201 + 1 = 202$.

Compared to a LSTM with Stock Prices and News Articles, the only additional change is the concatenation of an additional column of a vector of ARIMA forecasts to both $\tilde{\mathbf{F}}_{train}$ and $\tilde{\mathbf{F}}_{test}$.

For each sample $\mathbf{S}$ in $\tilde{\mathbf{F}}_{train}$, isolate the column corresponding to the transformed stock prices $\mathbf{y}_t$, for $t \in \{1, ..., n, n+1, ..., n+m\}$. A stepwise ARIMA model with the same setup as the ARIMA baseline model is fit on the lag portion of $\mathbf{y}_t$, for $t \in \{1, ..., n\}$. The ARIMA model is then used to make m predictions $\hat{\mathbf{y}}_{t'}$, for $t' \in \{n+1, ..., n+m\}$, which will be used as inputs to our model. Pad $\hat{\mathbf{y}}_{t'}$ with m additional scalar values of any value, and call the new vector $\hat{\mathbf{y}}_{padded}$. Then concatenate $\hat{\mathbf{y}}_{padded}$ column-wise to $\mathbf{S}$.

The same operation is performed on each sample $\mathbf{S}$ in $\tilde{\mathbf{F}}_{test}$.

The additional padding of scalar values will not matter, since they will be dropped in MSE and RMSE evaluations, as only the transformed stock prices $\hat{\mathbf{y}}_{t'}$, for $t' \in \{n+1, ..., n+m\}$ will be used for calculating MSE and RMSE.

Due to the stationary assumptions of ARIMA models, the ARIMA predictions used as input features are thought of as adding data that are the future m predictions with mean stability to the LSTM with Stock Prices and News Articles model, which may help reduce high variance or overfitting in the LSTM.

Fitting the LSTM using the train set and evaluating the LSTM using the test set follows in the exact same way as for the LSTM with Stock Prices and News Articles.

8 Results

Each model will be presented with an overview, plots of the forecasts, plots of the train loss, and plots of the test loss. Note that for the plots of the forecasts, only the last third of the time series is plotted, as that is the portion of the time series for which predictions are made. For the plots of the test loss, for better visual comparison, only the first 20 timesteps into the future are plotted.

8.1 ARIMA

ARIMA models struggled to capture variance in the transformed space and thus in the original space. In particular, the predictions tend to focus on the fitting nearly linear trends to the training data of length n and does not adapt to unexpected fluctuations in the stock time series data. This is especially apparent as m increases. While the focus is on prediction and not inference, it does seem that the ARIMA models would likely fail certain stationary tests and that the transformed time series from Figure 1 do not follow a predictable pattern with constant variance.

Across all stocks, the test loss generally decreases as m increases. $n = 60$ for all setups, so all ARIMA models are trained on the same lag length for each sample. This means that the differences in test loss must be attributed the decreasing sample sizes in the test set as m increases. Thus, decreasing test loss as m increases is to be expected, as the model should not perform well on additional samples, due to its difficulty in capturing variance. The RMSE was lowest for DIS, moderate for AAPL and GS, and very high for AMZN and TSLA. From the transformed time series plots in Figure 1, it looks like higher test RMSE here corresponds to more heteroscedasticity in the transformed stock prices.

8.1.1 ARIMA: Forecasts

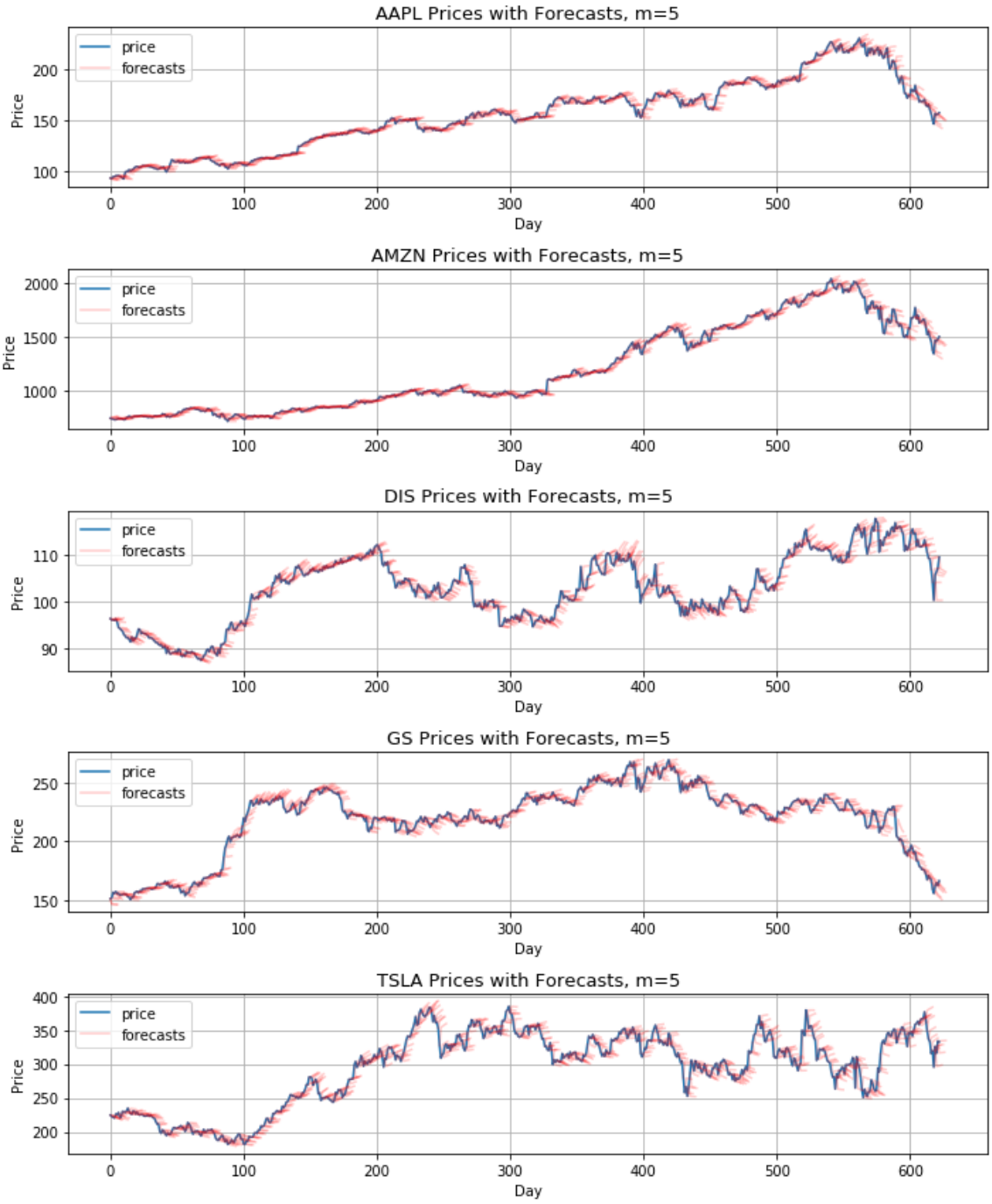


Figure 7: ARIMA Forecasts for 5 steps

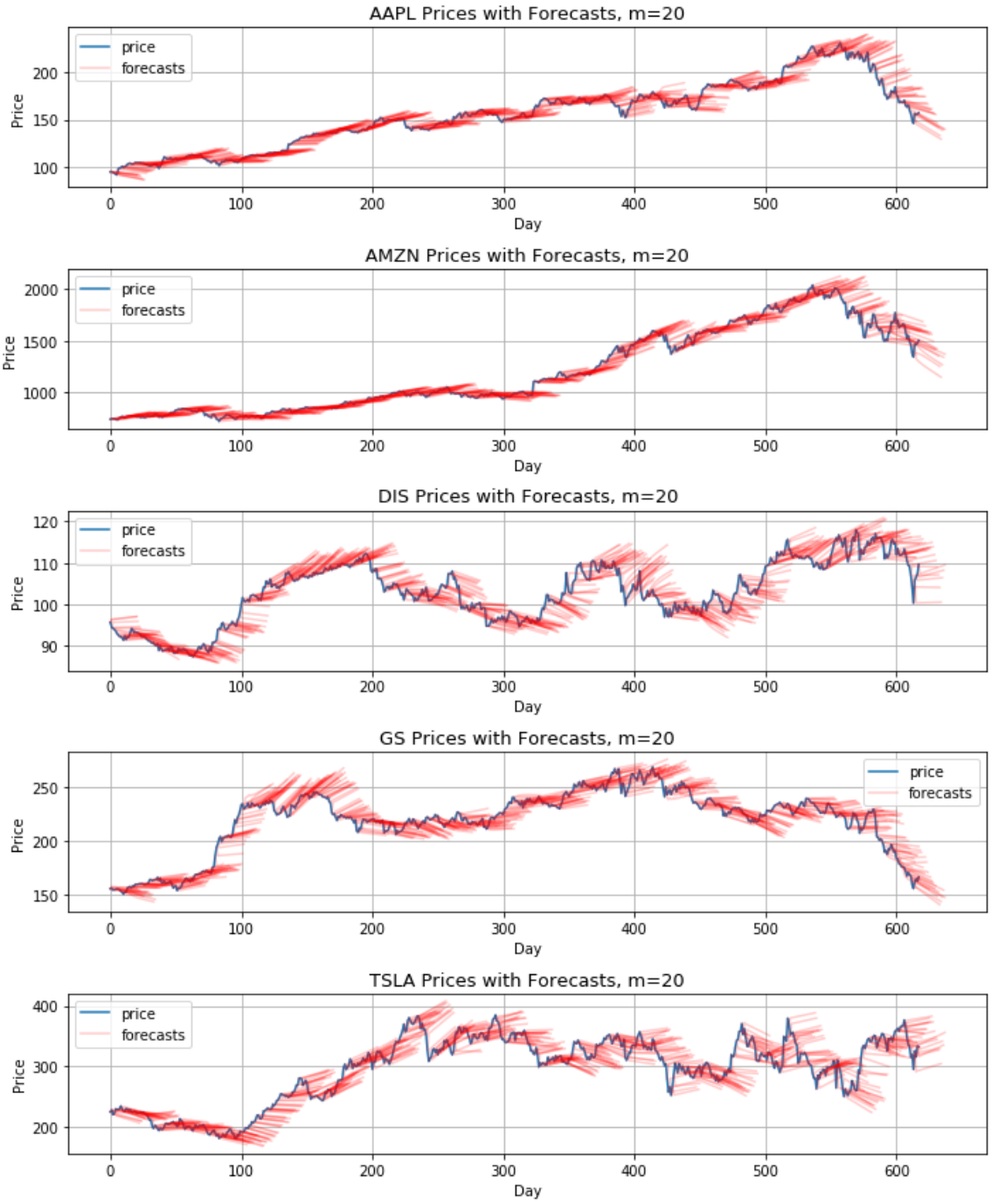


Figure 8: ARIMA Forecasts for 20 steps

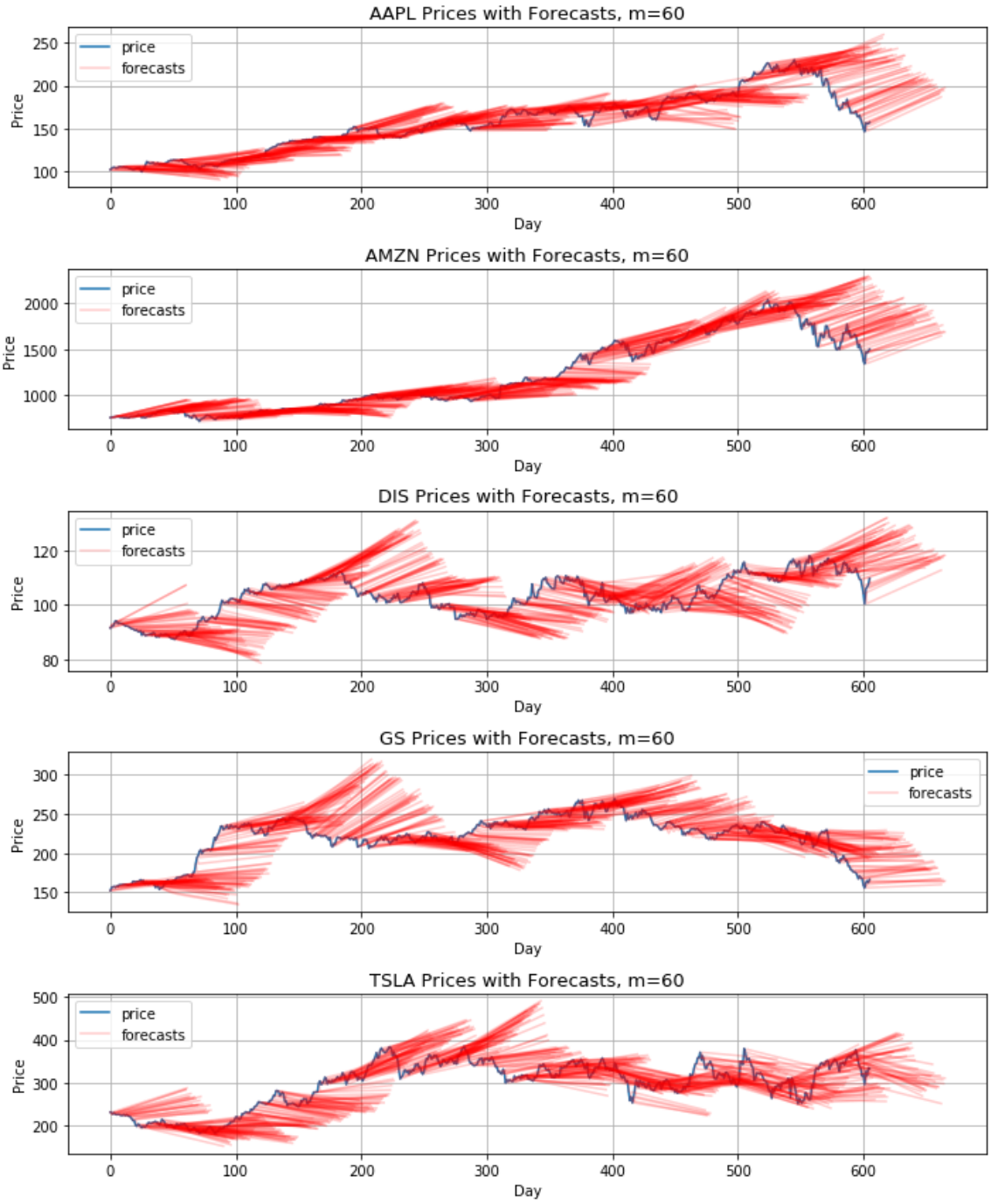
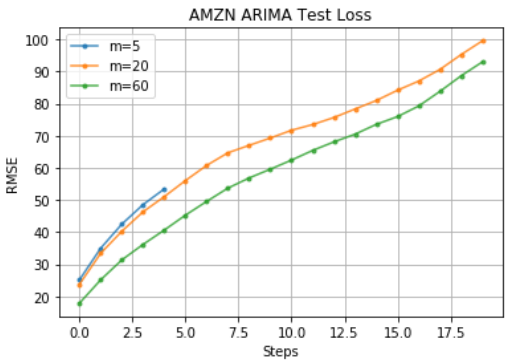
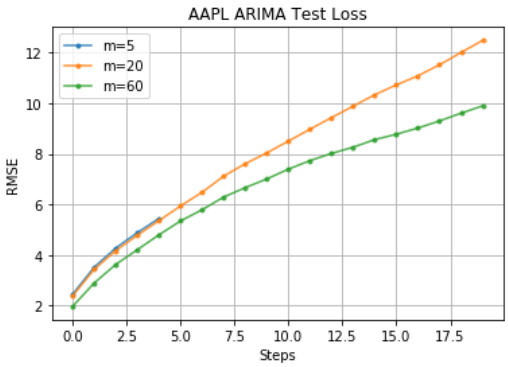
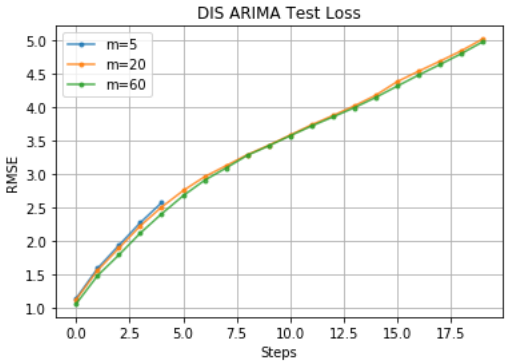


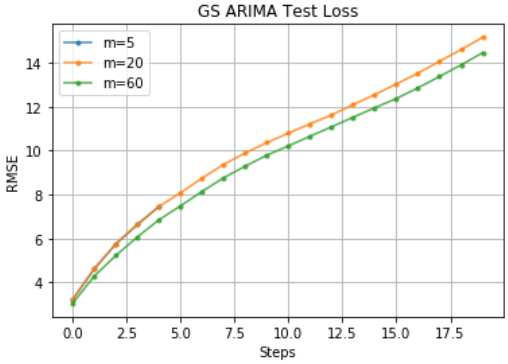
Figure 9: ARIMA Forecasts for 60 steps

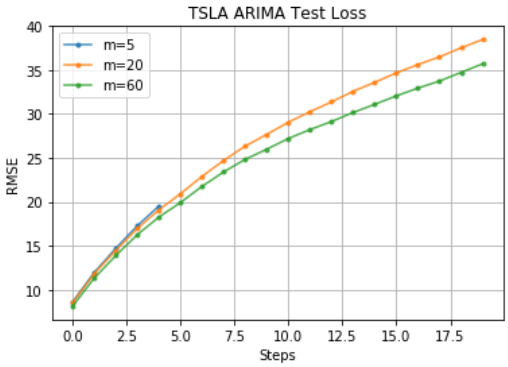
8.1.2 ARIMA: Test Loss











8.2 LSTM with Stock Prices

In general, the LSTM baseline model produced forecasts that tend to follow a linear trend but did a better job at predicting with variation, unlike the ARIMA baseline model that is not flexible enough to model time series that violate underlying stationarity assumptions.

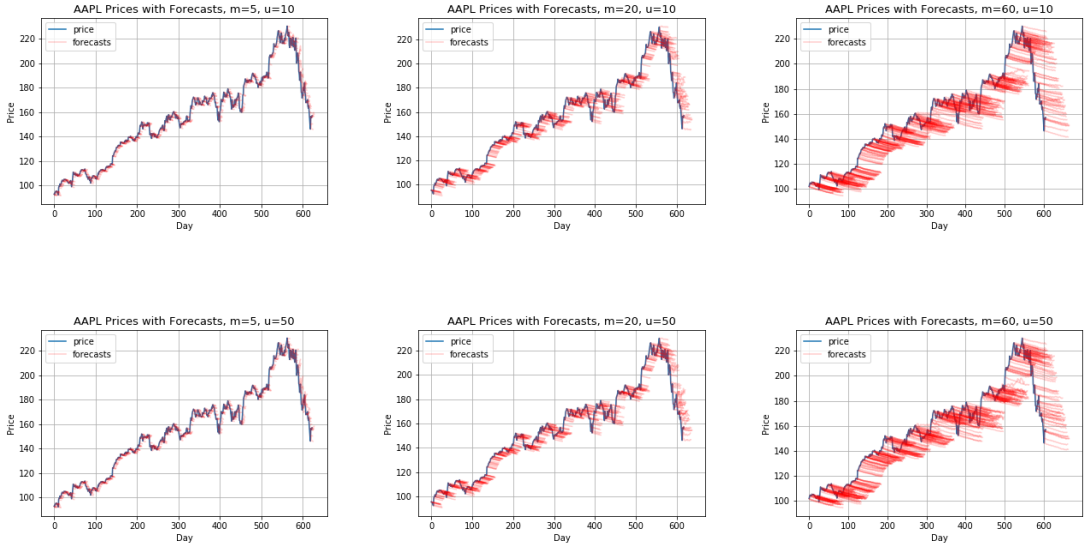
While MSE did generally decrease across all setups, model training struggled to converge smoothly, especially in the case of AMZN. This may be due to AMZN's high heteroscedasticity in the transformed stock price, compared to the other stocks. Thus, AMZN seemed especially overfit, as many of the predictions tended to be similar in predicting an upward trend with close to the same slope as each other.

In general, for a given choice of u, the test loss usually dropped as m increased, signaling that fewer samples from increasing m may go hand-in-hand with lower RMSE. Unlike the ARIMA baseline, where this would suggest that the model is biased and does not capture noise well in additional data samples, the LSTM model not only fits to the training data in each sample in the train set, but also adjusts its fit using the m steps ahead, unlike the ARIMA baseline, via gradient steps towards decreasing directions of MSE. Thus, here it suggests that increasing the number of timesteps m to make predictions allows for richer model expressivity by seeing more of the future during fitting, thus consequently lowering RMSE.

In general, for a given choice of m, the choice of u did not affect test loss much when $m = 20$ or $m = 60$, signaling that there is no strong reason to believe that the model is struggling with overfitting or underfitting in these cases, based on the number of neurons. When $m = 5$, however, higher u generally leads to higher RMSE values, suggesting that too many neurons to predict into the future for fewer timesteps can lead to overfitting during model training, with the exception to this observation being DIS.

As $m = 5$ does not seem stable across setups, focusing on $m = 20$ and $m = 60$, the test loss for AMZN was well above that of the other stocks. AMZN RMSE started from around 20 with the 1 timestep forward and ballooned to over 80 by 20 timesteps forward. DIS, which exhibits low heteroscedasticity in the transformed stock prices, performed the best with RMSE below 5 for the 20 timesteps forward.

8.2.1 LSTM with Stock Prices: Forecasts



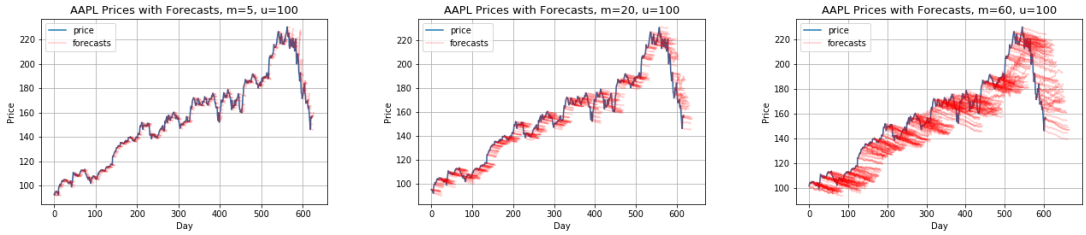


Figure 10: LSTM with Stock Prices, AAPL Forecasts

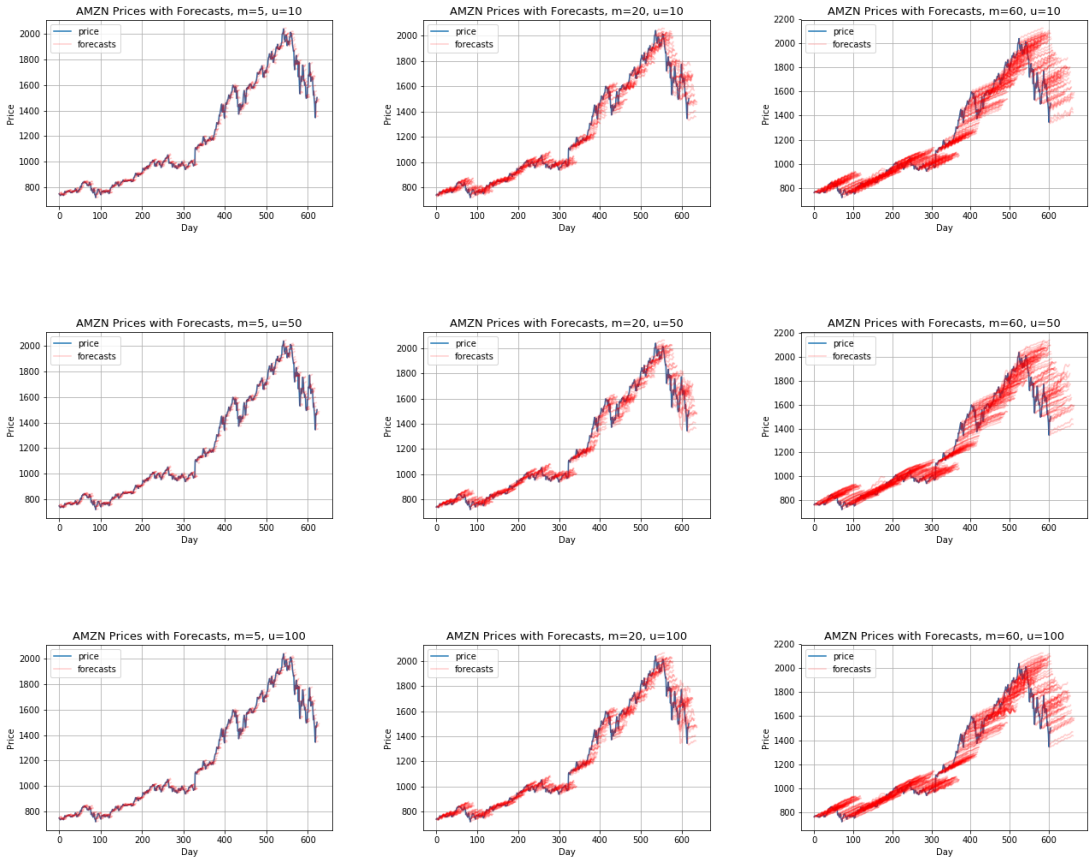
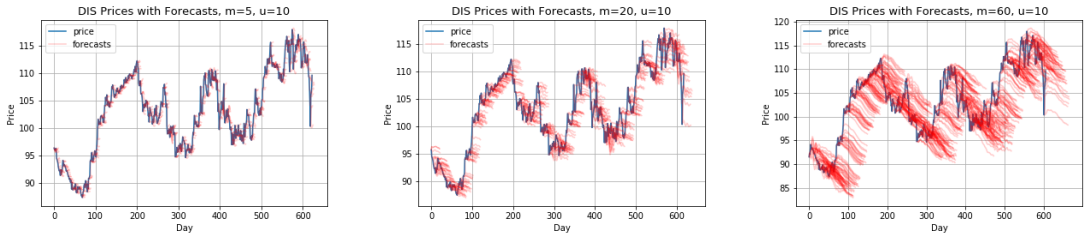


Figure 11: LSTM with Stock Prices, AMZN Forecasts



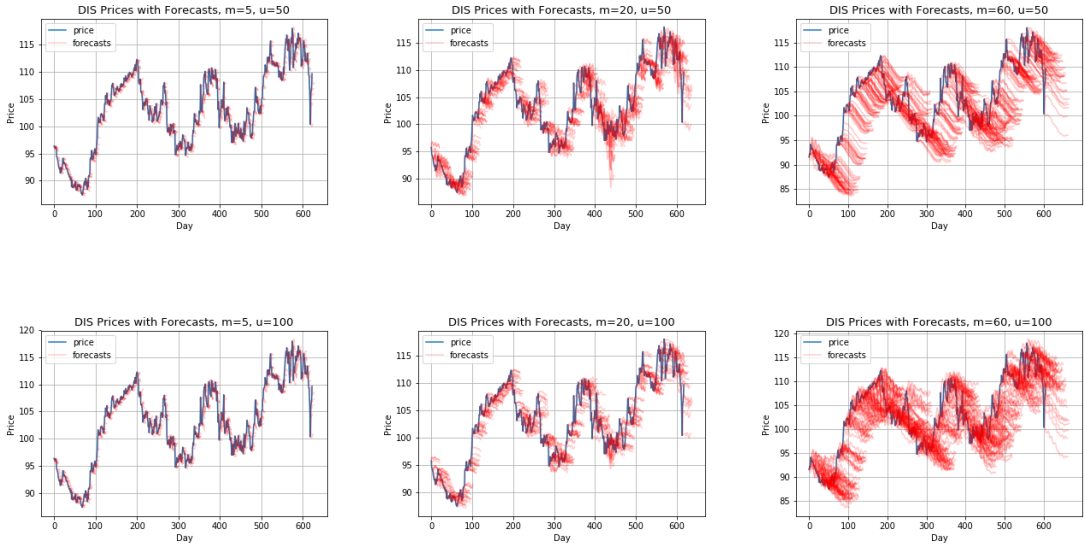


Figure 12: LSTM with Stock Prices, DIS Forecasts

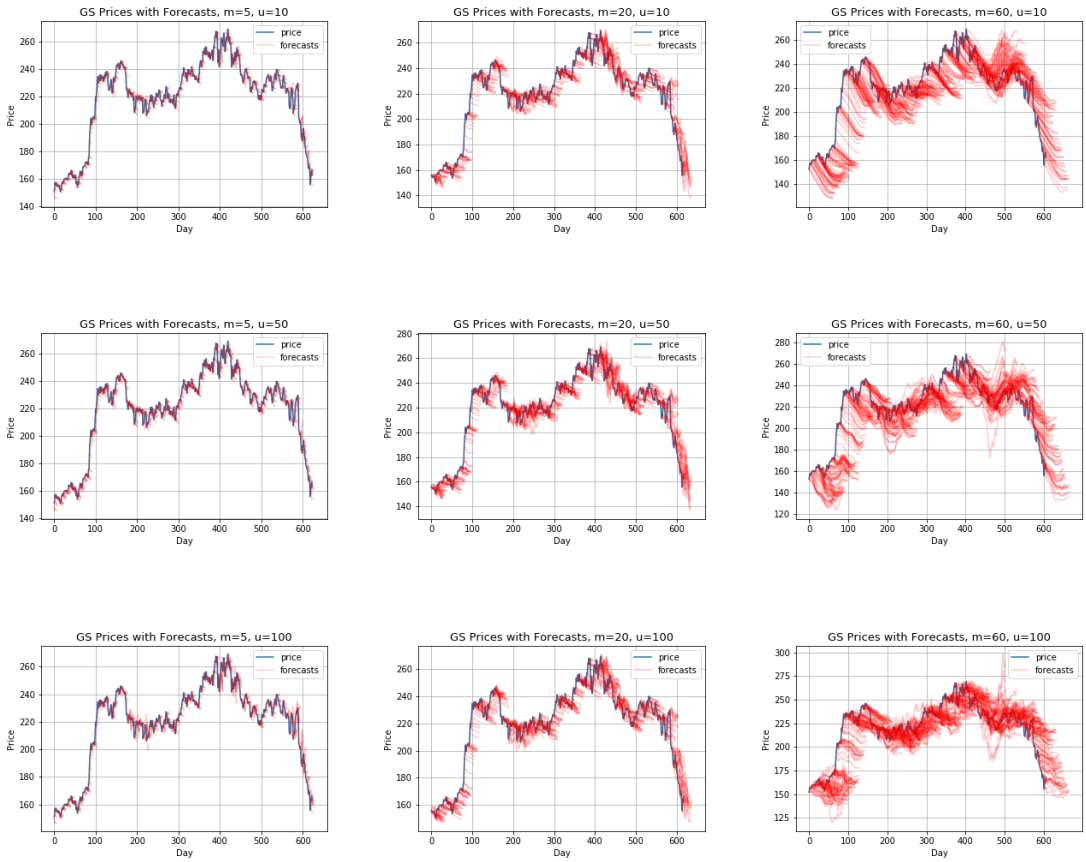


Figure 13: LSTM with Stock Prices, GS Forecasts

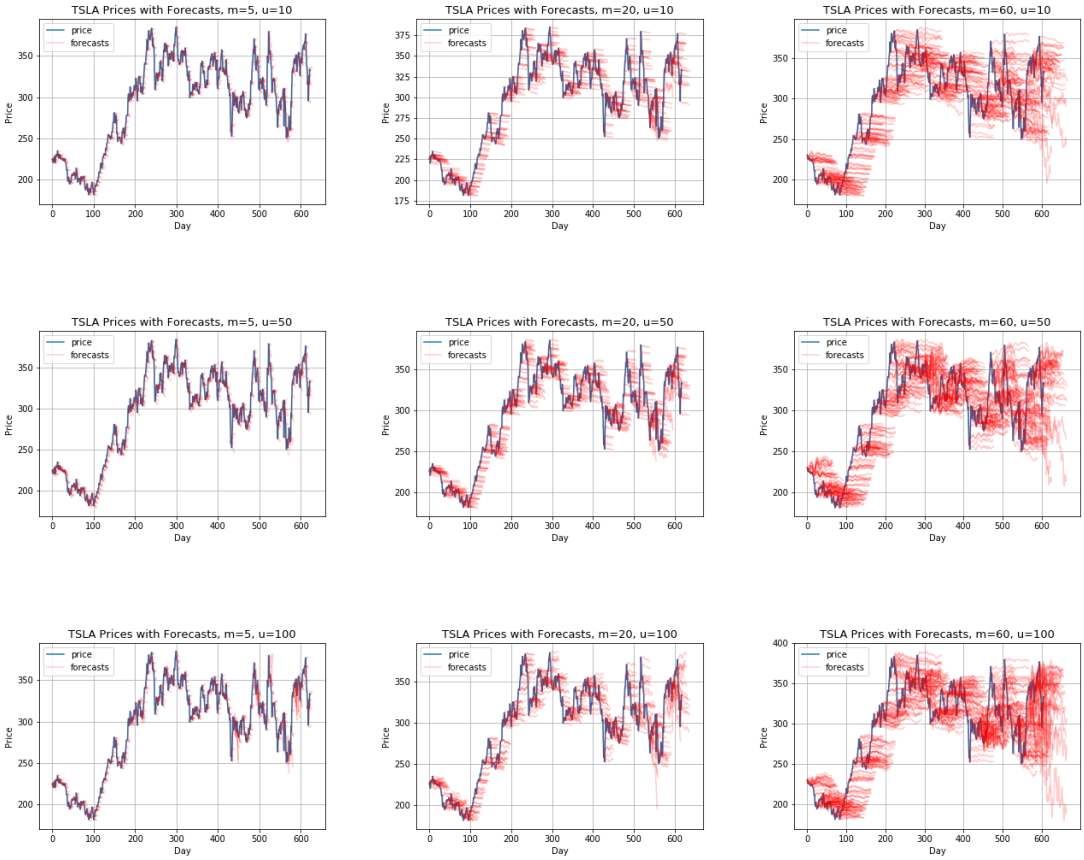


Figure 14: LSTM with Stock Prices, TSLA Forecasts

8.2.2 LSTM with Stock Prices: Train Loss

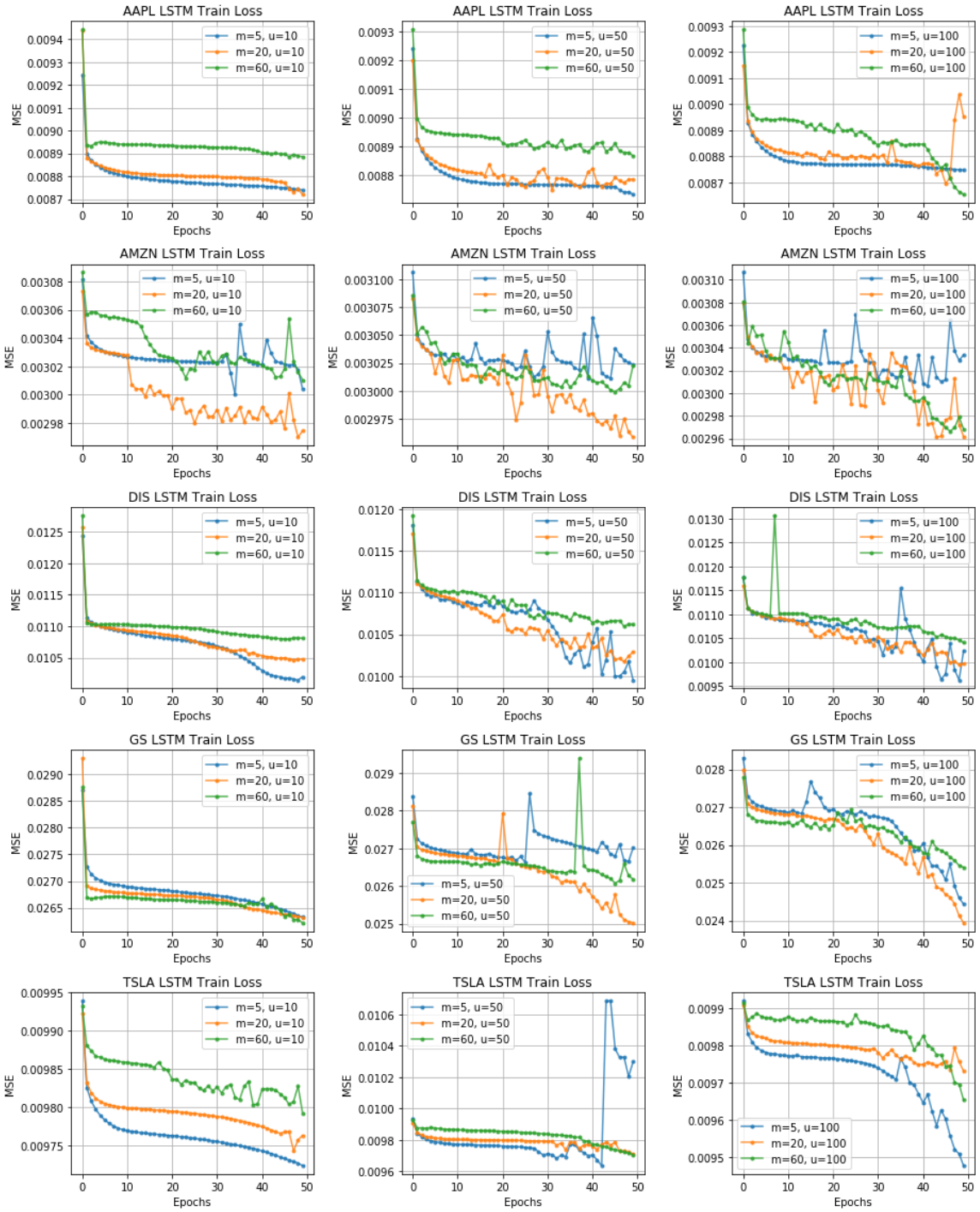


Figure 15: LSTM with Stock Prices, Train Loss

8.2.3 LSTM with Stock Prices: Test Loss grouped by Neurons

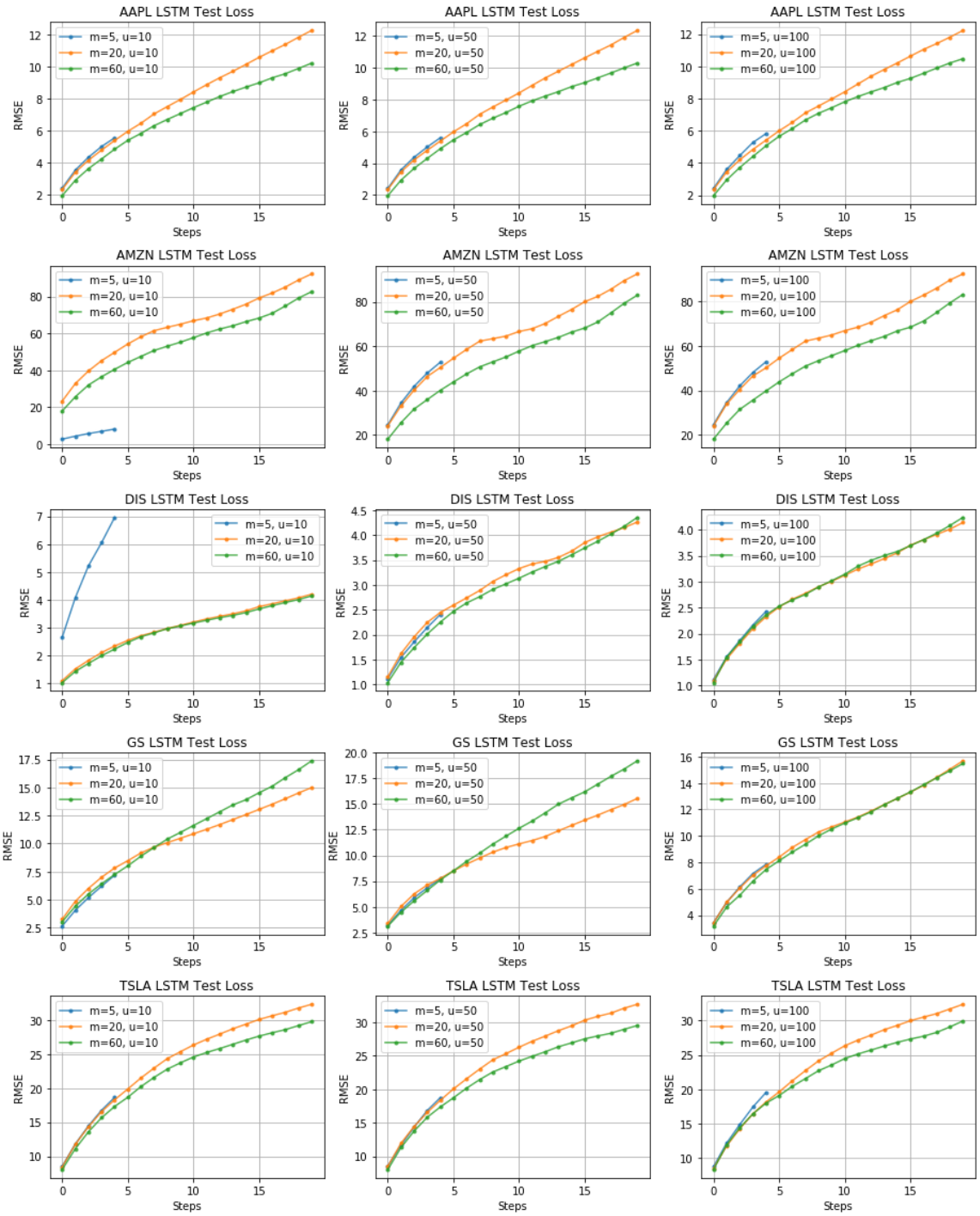


Figure 16: LSTM with Stock Prices, Test Loss by Neurons

8.2.4 LSTM with Stock Prices: Test Loss grouped by Steps

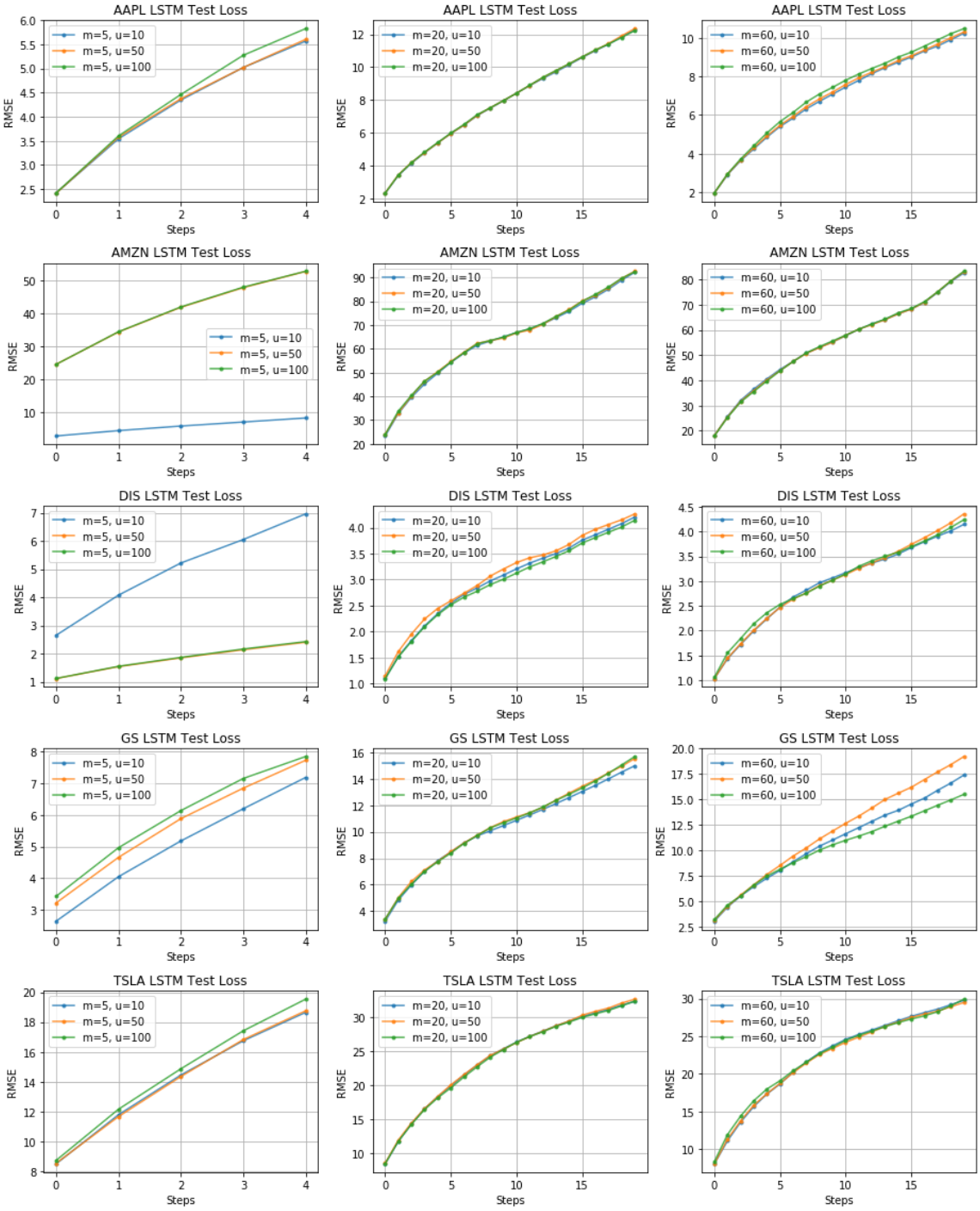


Figure 17: LSTM with Stock Prices, Test Loss by Steps

8.3 LSTM with Stock Prices and News Articles

In general, the LSTM with Stock Prices and News Articles models produced forecasts that do not tend to follow a linear trend and do a better job at predicting with variation compared to the LSTM baseline.

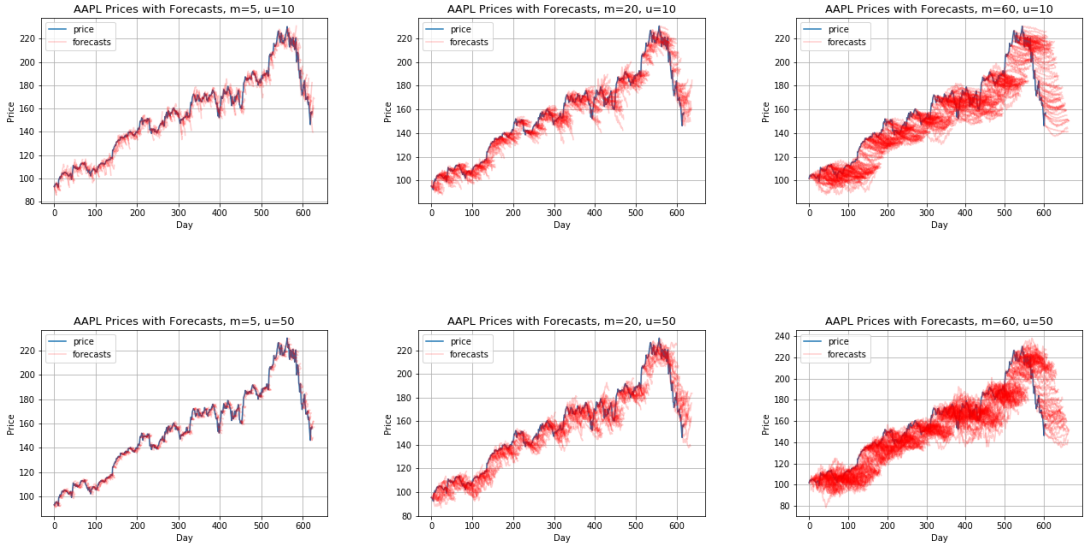
Model training is smoothly decreasing and shows convergence in all setups, unlike in the LSTM baseline. This is true even in the case of AMZN. A consistent pattern across all setups is also present, where the lower the value of m, the faster the convergence and the lower the MSE, which is expected as the number of timesteps m used to calculate MSE decreases. Furthermore, the MSE values over the epochs are lower for each setup here compared to those in the LSTM baseline model. The training loss exhibits healthy fitting behavior that is reflected in the better job this model does in forecasting with reasonable predictions that are not highly biased and still follow the general time series curve well.

In general, for a given choice of u, the test loss usually dropped as m increased. This drop does not happen in as many cases here as it does in the LSTM baseline, suggesting that while increasing the number of timesteps to make predictions allows for richer model expressivity by seeing more into the future during fitting, which consequently would lower RMSE, the model here may be struggling to display dropping RMSE in most cases as m increases because of high sensitivity to modeling variance in the time series.

In general, for a given choice of m, the choice of u affected RMSE more so than the LSTM baseline when $m = 20$ or $m = 60$, but no clear pattern can be noted. When $m = 5$, however, higher u generally leads to lower RMSE values, which is the opposite observation compared to the LSTM baseline. Here, this suggests that more neurons are valuable for predicting into the future, as underfitting due to sensitivity to variance will be mitigated as such. Whereas the LSTM baseline is highly biased, the model here has high variance, which makes sense in context of this particular case of $m = 5$ with increasing u.

Across the different setups, the test loss for AMZN was still well above that of the other stocks. AMZN RMSE started from around 20 with the 1 timestep forward and ballooned to over 100 by 20 timesteps forward. DIS, which exhibits low heteroscedasticity in the transformed stock prices, performed the best with RMSE below 8 across the different setups for 20 timesteps forward.

8.3.1 LSTM with Stock Prices and News Articles: Forecasts



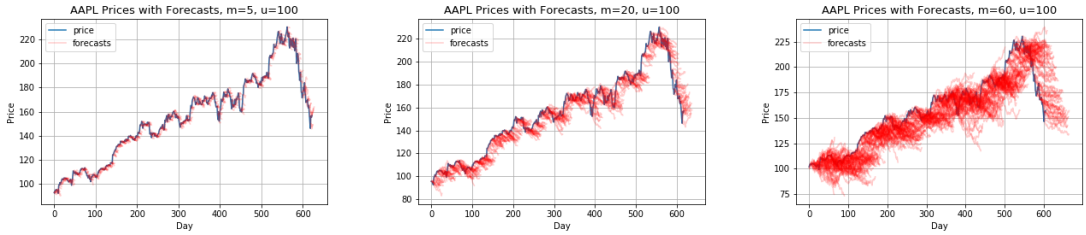


Figure 18: LSTM with Stock Prices and News Articles, AAPL Forecasts

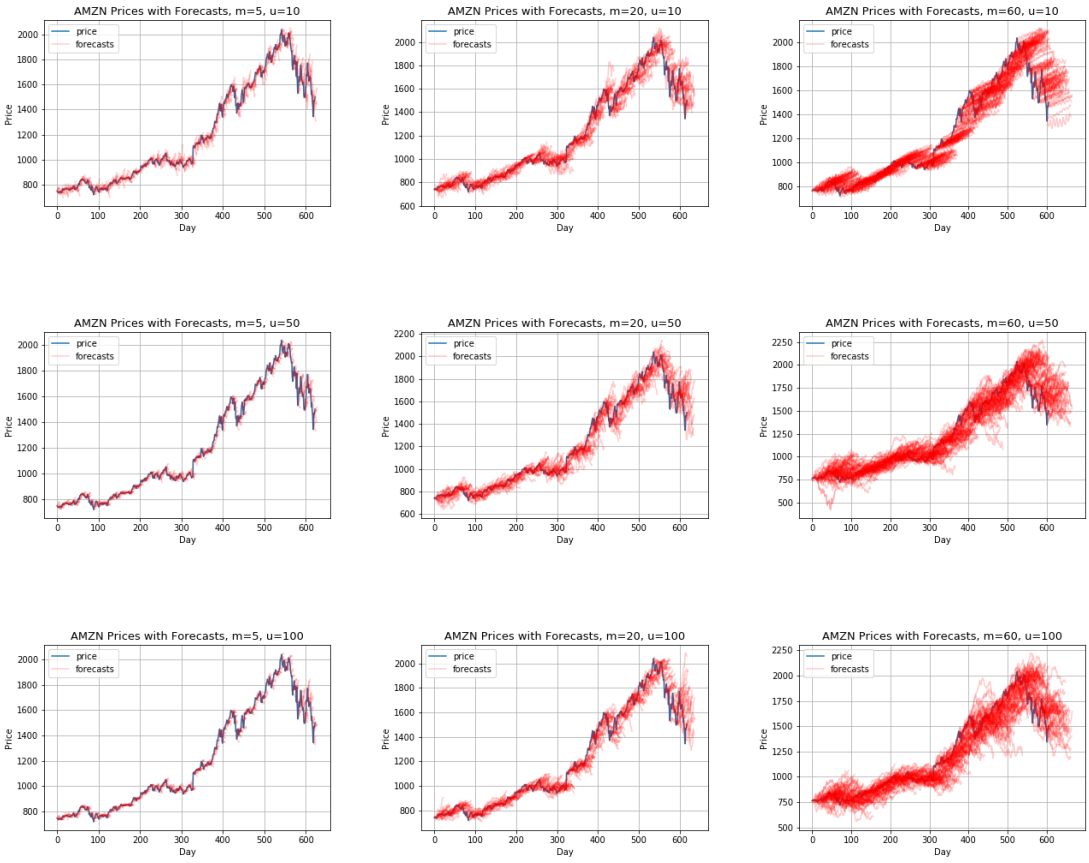
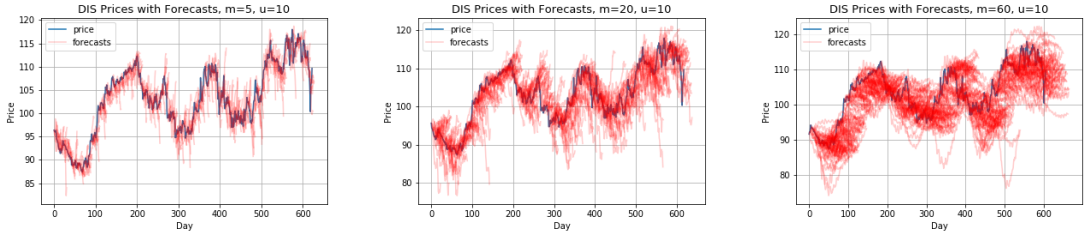


Figure 19: LSTM with Stock Prices and News Articles, AMZN Forecasts



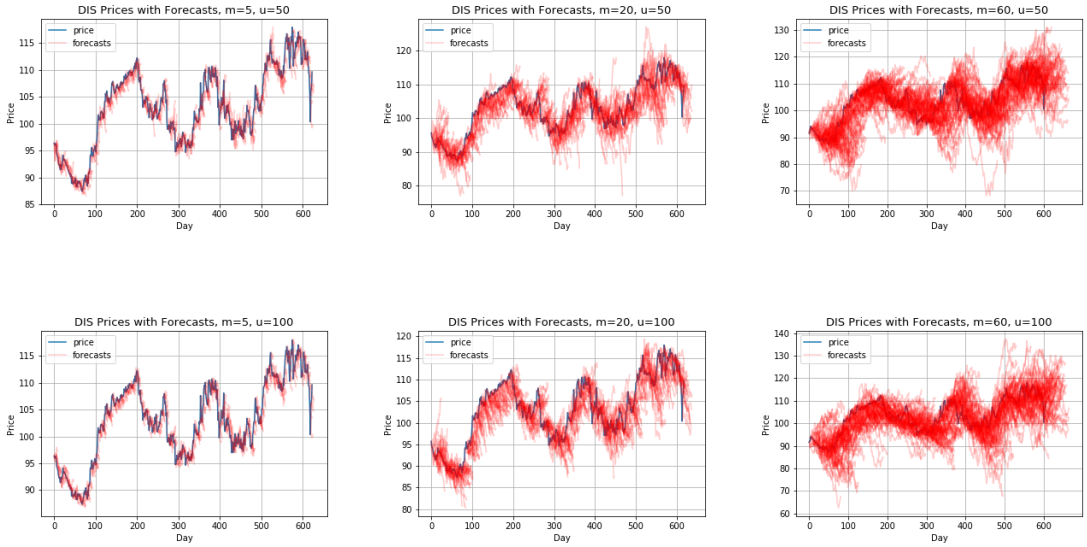


Figure 20: LSTM with Stock Prices and News Articles, DIS Forecasts

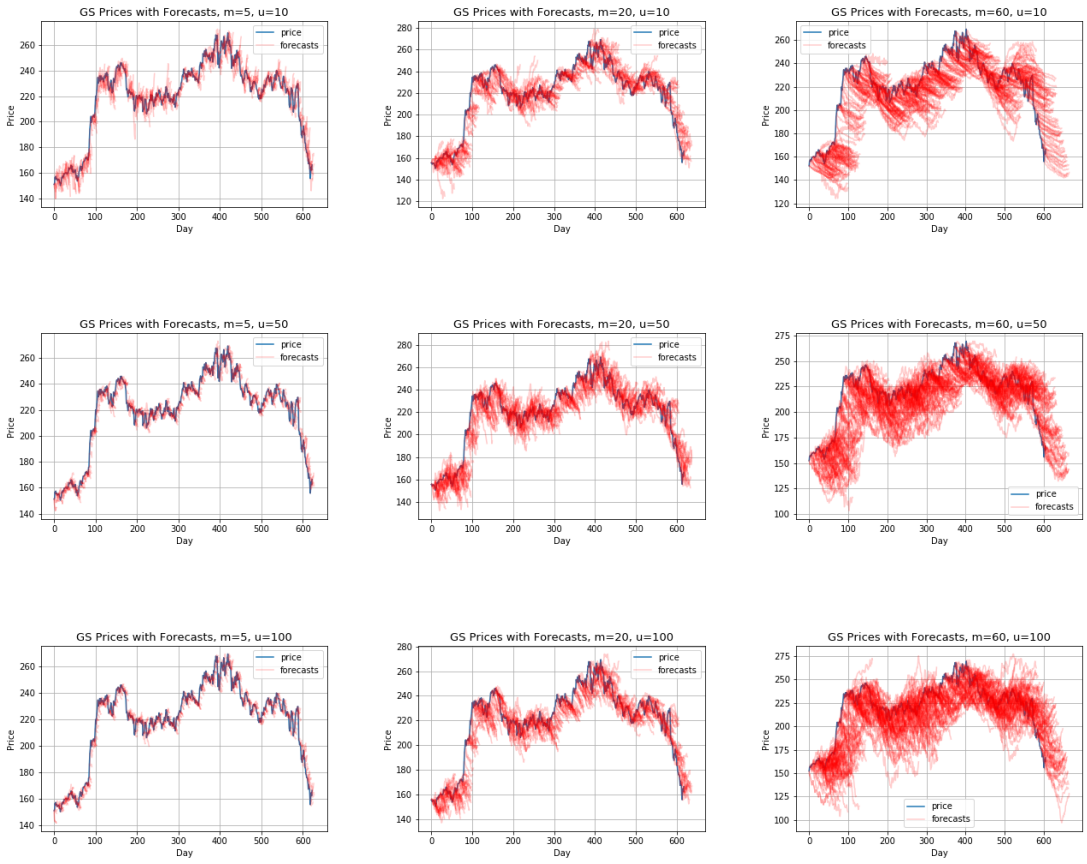


Figure 21: LSTM with Stock Prices and News Articles, GS Forecasts

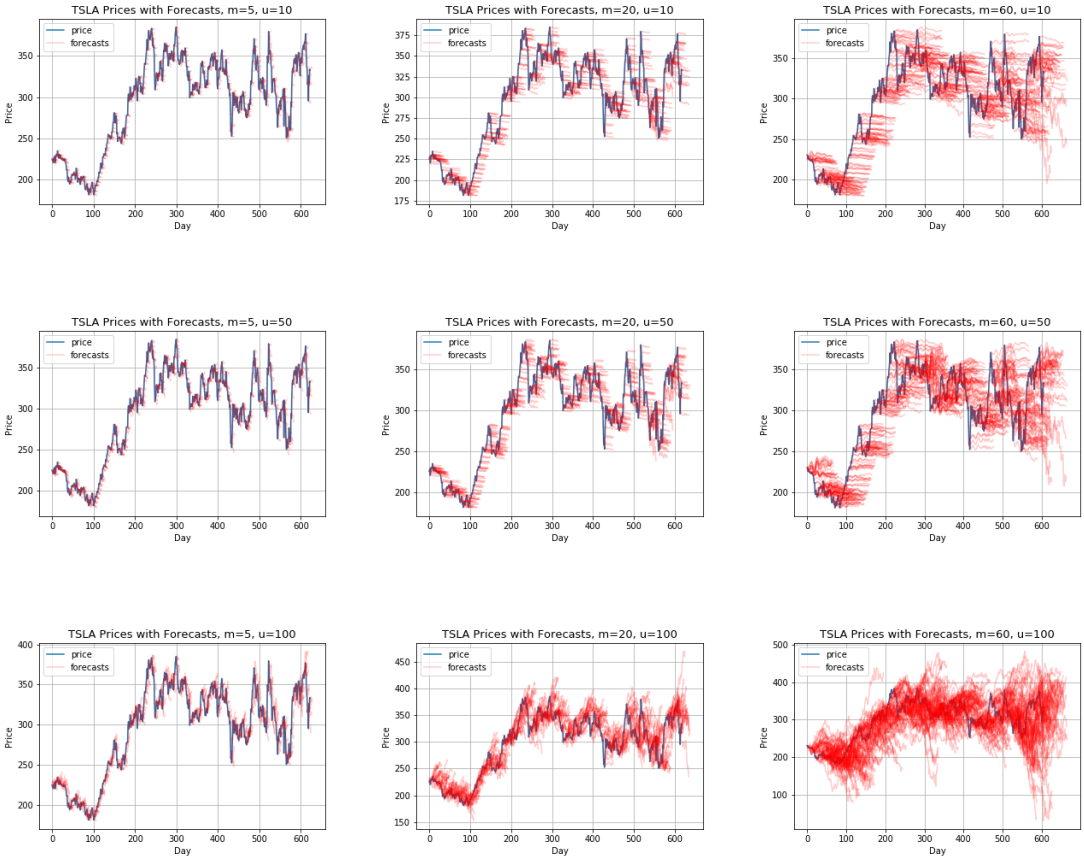


Figure 22: LSTM with Stock Prices and News Articles, TSLA Forecasts

8.3.2 LSTM with Stock Prices and News Articles: Train Loss

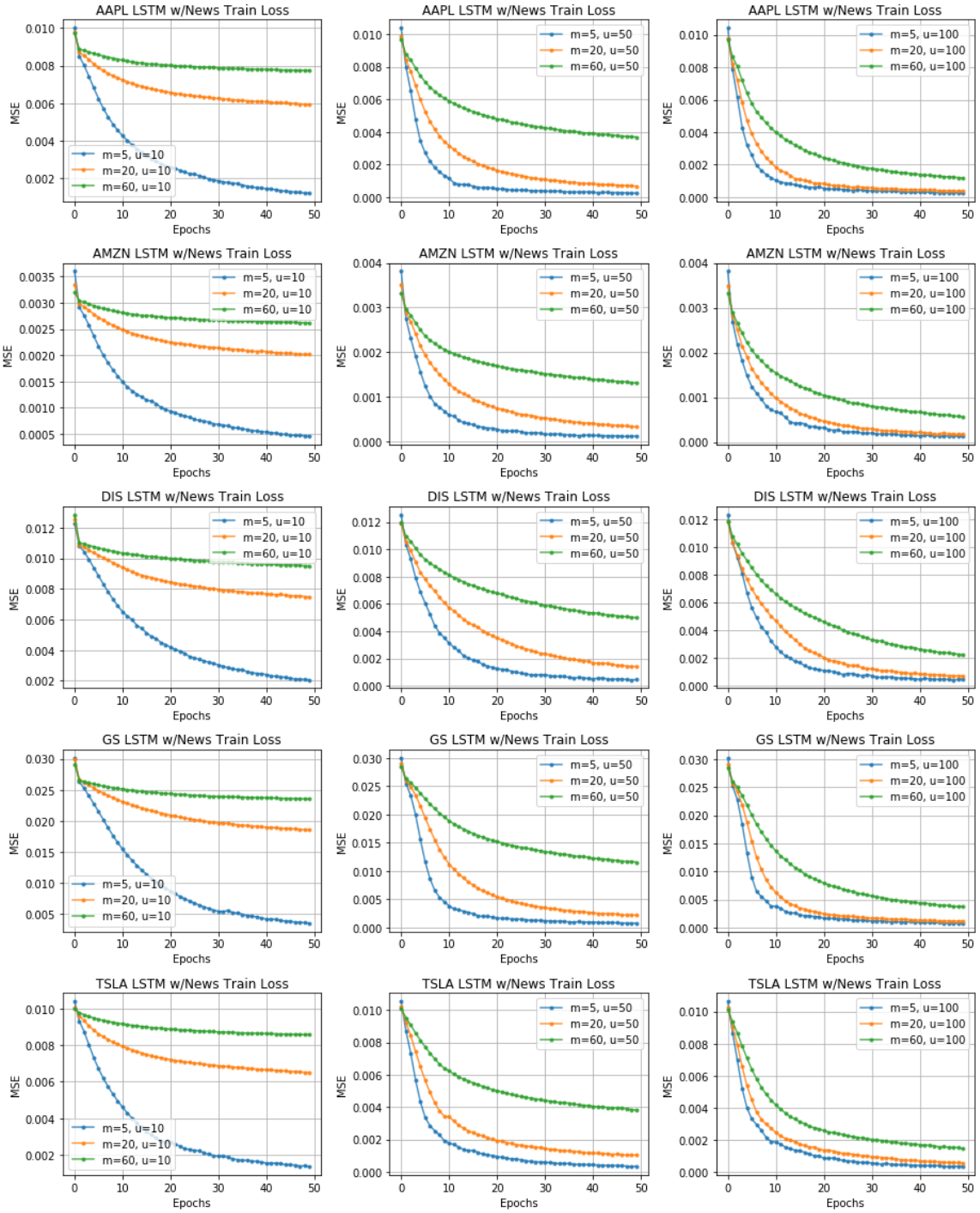


Figure 23: LSTM with Stock Prices and News Articles, Train Loss

8.3.3 LSTM with Stock Prices and News Articles: Test Loss grouped by Neurons

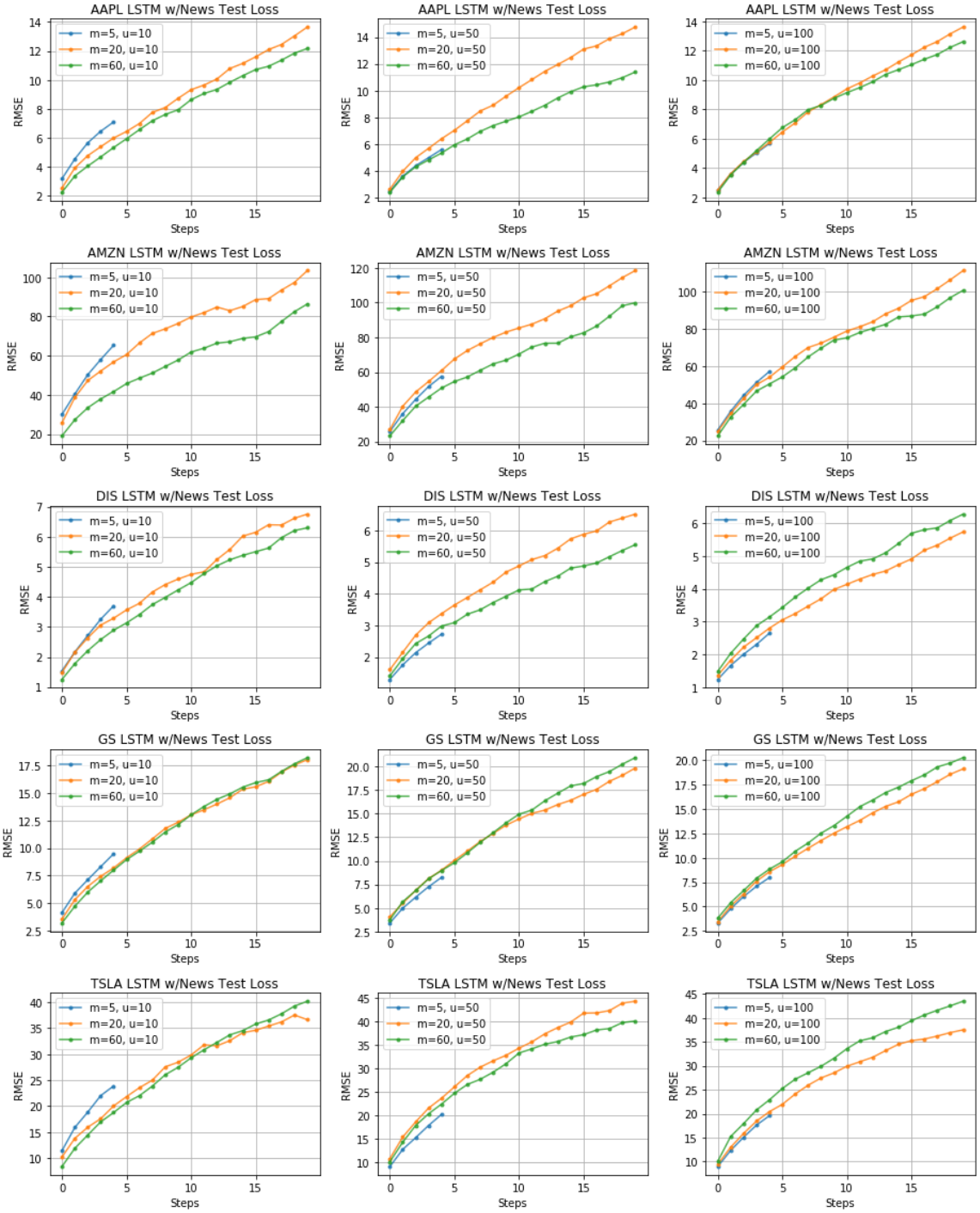


Figure 24: LSTM with Stock Prices and News Articles, Test Loss by Neurons

8.3.4 LSTM with Stock Prices and News Articles: Test Loss grouped by Steps

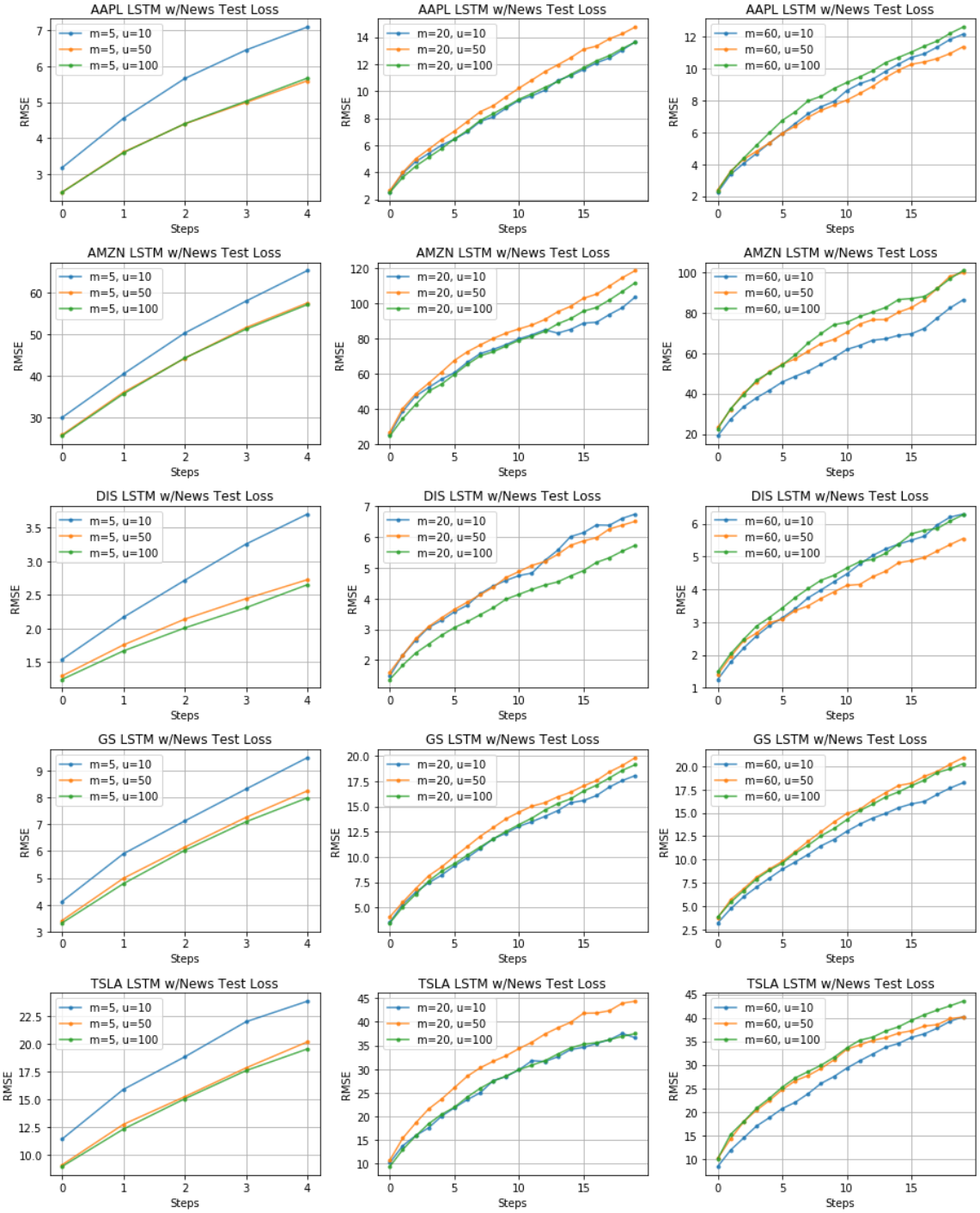


Figure 25: LSTM with Stock Prices and News Articles, Test Loss by Steps

8.4 Model Comparisons

A first interesting note is that for $m = 60$, there is asymptotic behavior in the two LSTM models in the growth of RMSE, particularly for DIS, GS, and TSLA. Comparing test RMSE across the three models thus far, the most commonly observed ordering is that the LSTM with news articles performs worse than the ARIMA baseline and the LSTM baseline. Even though the LSTM with news articles has the healthiest fitting and generalizes well to the overall trend in the time series, it is too noisy and lacks mean stability. As such, DIS, which performed the best of all stocks in terms of RMSE, will be examined further by different tokenizations of the news articles and ARIMA-Enrichment.

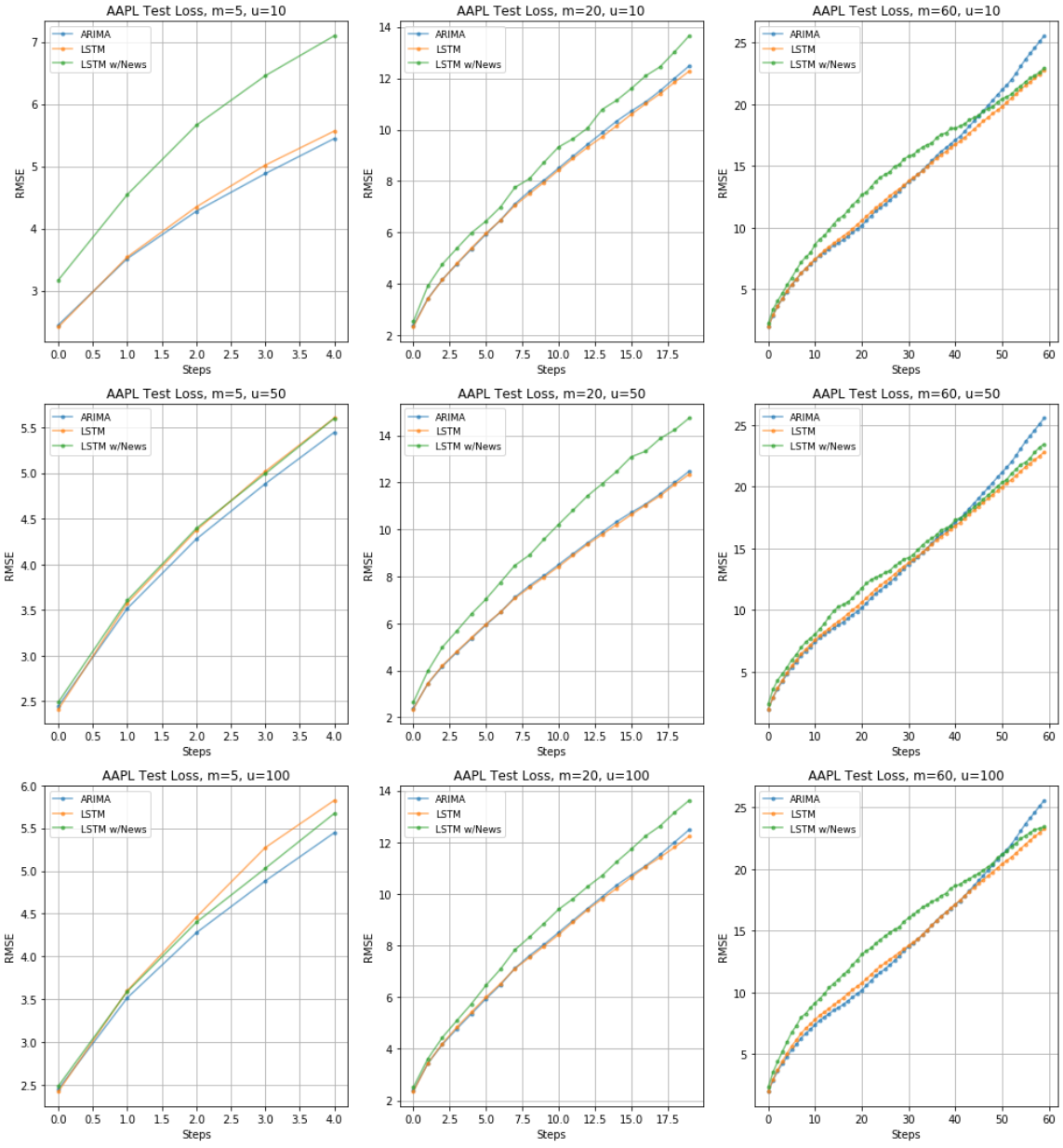


Figure 26: APPL Test Loss Comparison by Model

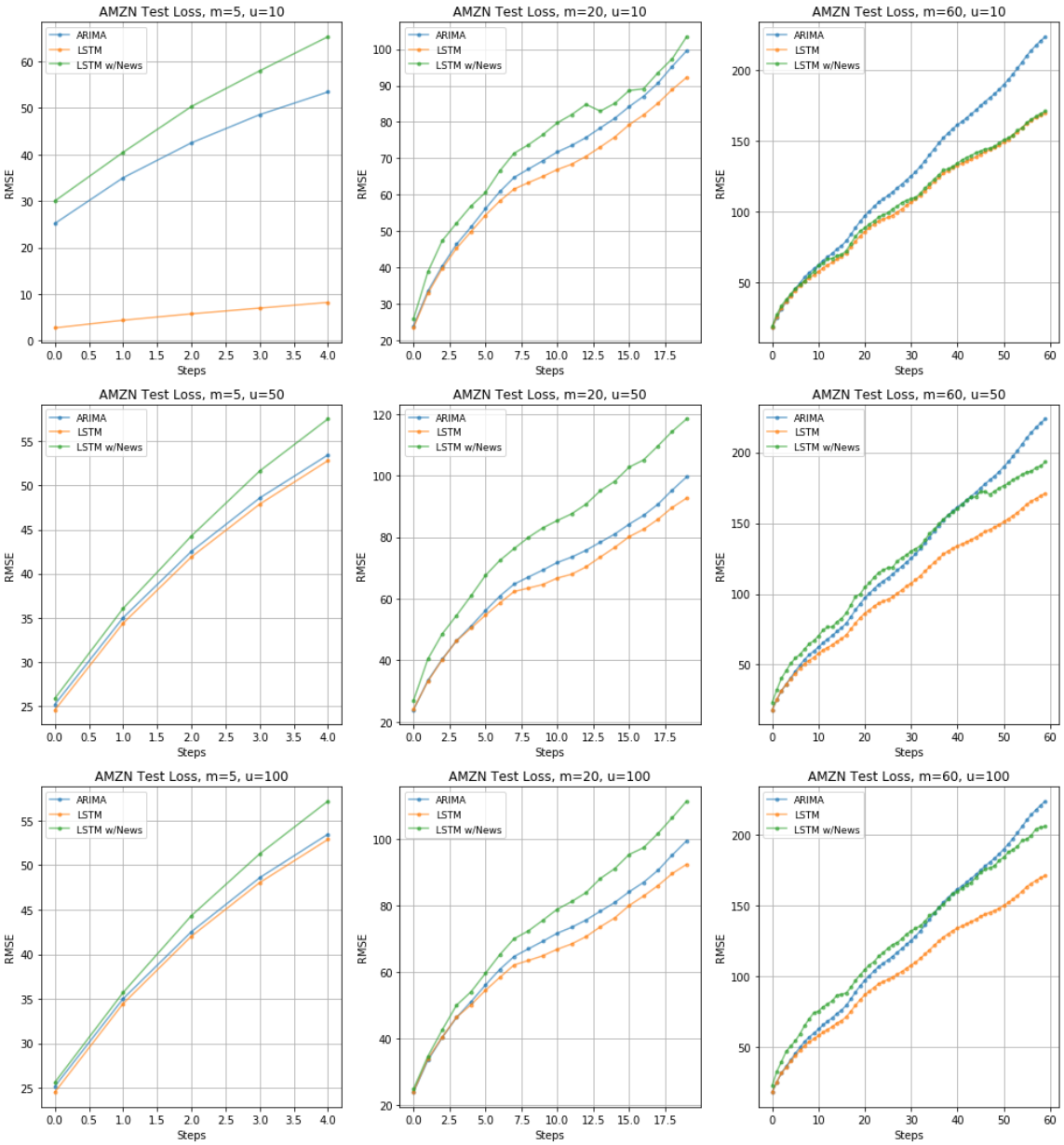


Figure 27: AMZN Test Loss Comparison by Model

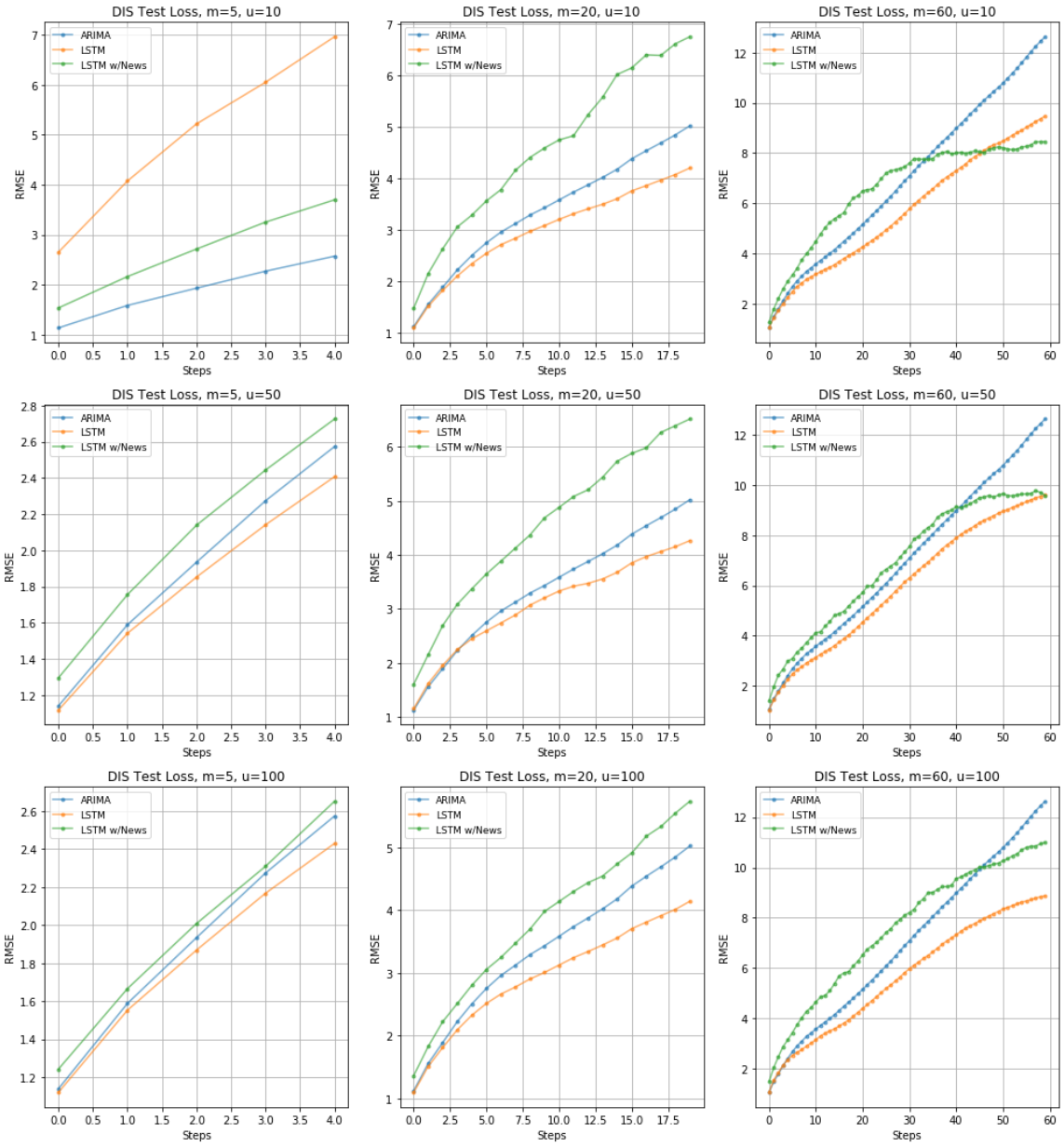


Figure 28: DIS Test Loss Comparison by Model

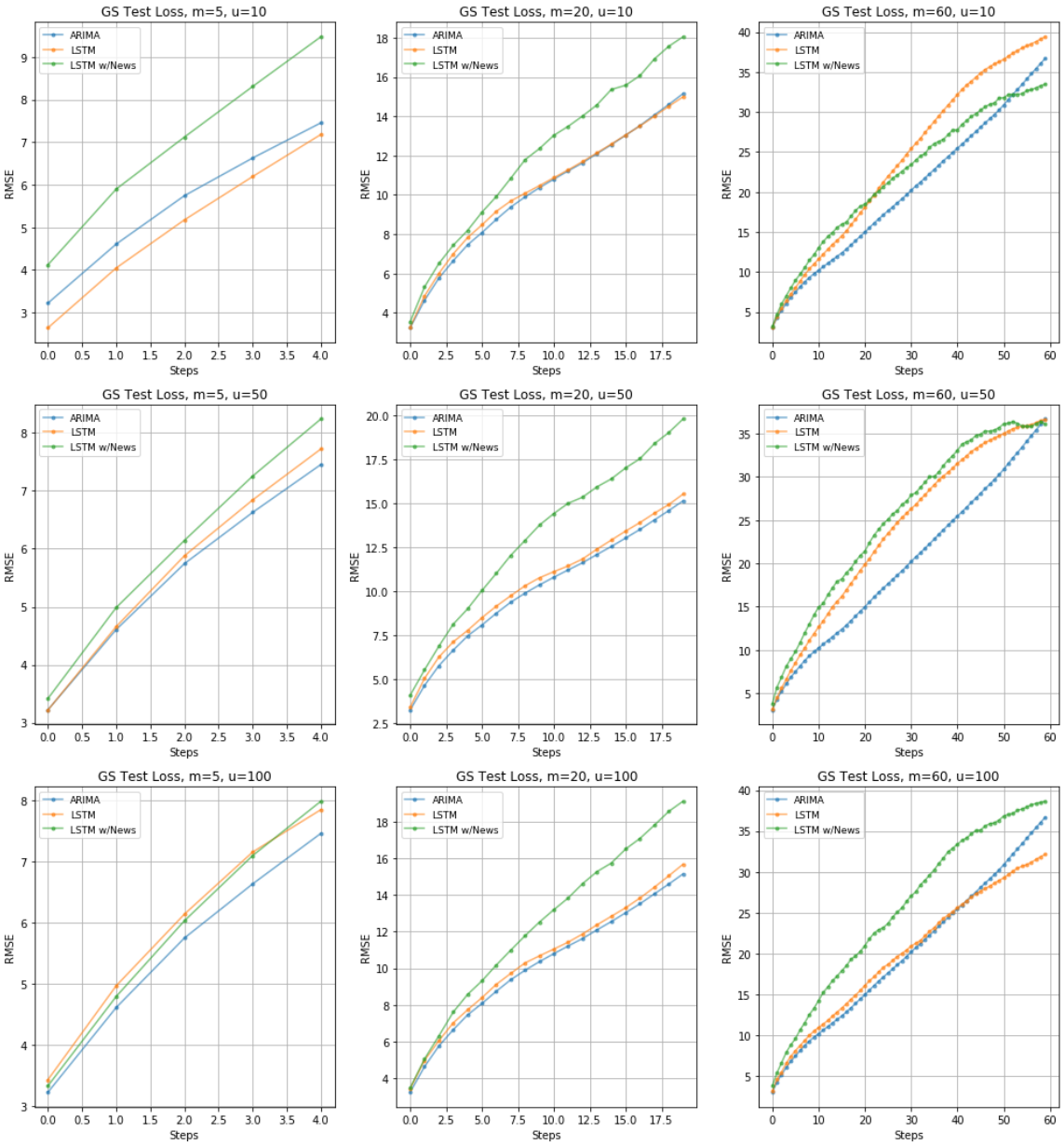


Figure 29: GS Test Loss Comparison by Model

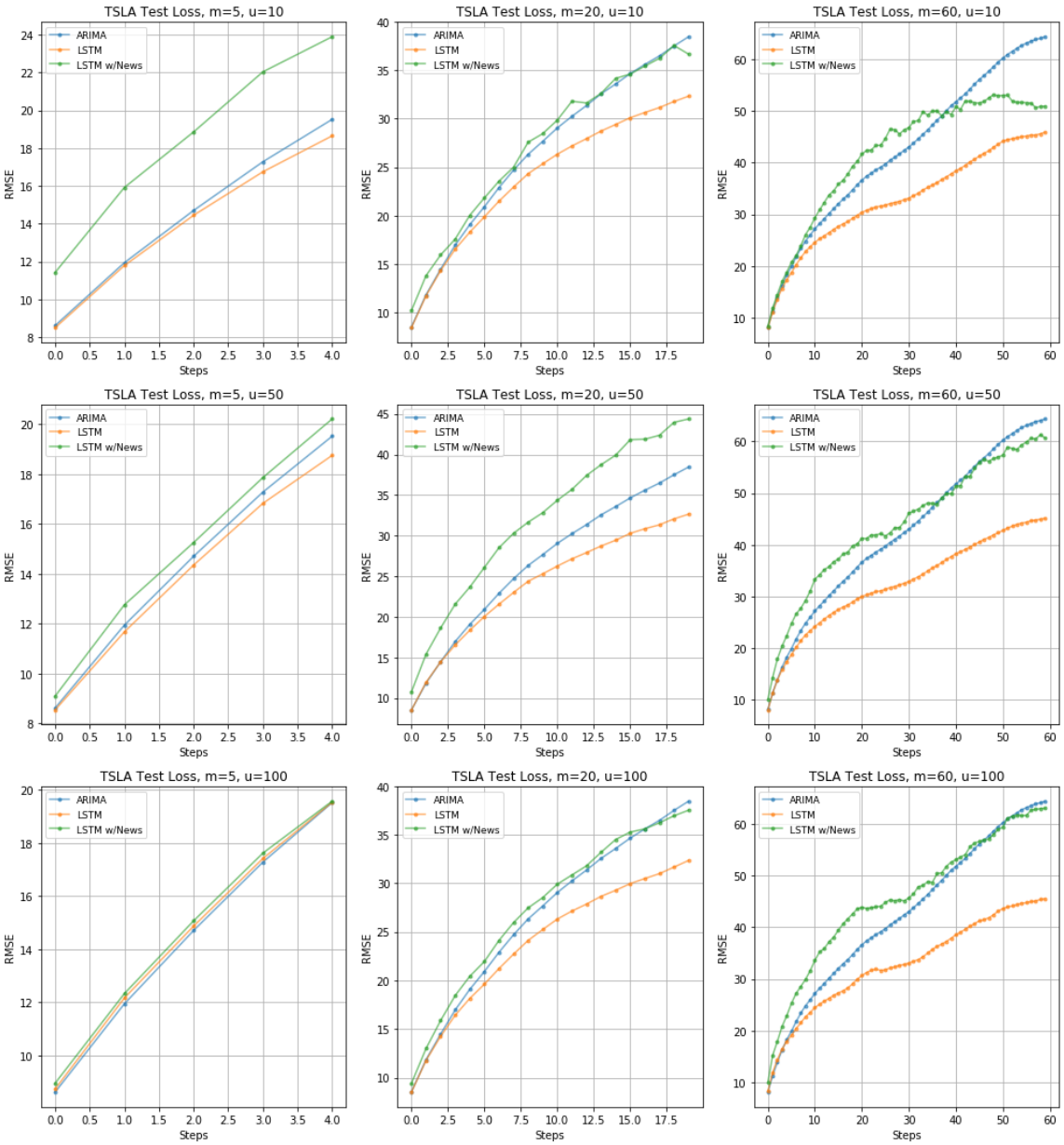


Figure 30: TSLA Test Loss Comparison by Model

8.5 Tokenizations Comparison

For DIS, the differences in performance across different tokenizations is minimal. From Table 1, it seems that an appreciable disadvantage would occur in model performance using bigrams, since the variance explained by the 200 principal components drops compared to that of onegrams, and that an appreciable advantage would occur in model performance using trigrams, since the variance explained by the 200 principal components rises compared to that of onegrams.

The onegram tokenization generally performs best on test loss RMSE, followed by bigrams, followed by trigrams. One likely reason is that, even though higher order gram representations capture more semantics in news articles, bigrams and trigrams do not occur very many times in the same news article. These higher order gram representations will tend to be less dense than, for example, onegram representations that will show less sparsity before a PCA transformation.

An interesting exception is that bigrams performs better than onegrams on RMSE for $u = 100$, and there is the interesting advantageous behavior of the trigram RMSE decreasing after roughly 35 steps into the future, when $u = 10$. Nothing definitive can be said about these aspects, so all three tokenizations will be treated as equals and will be used in the ARIMA-Enriched LSTM model.

8.5.1 Tokenization Comparison: DIS Forecasts

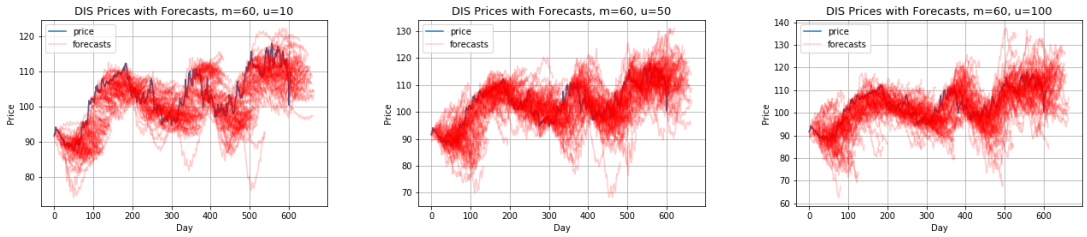


Figure 31: Tokenization Comparison, DIS Forecasts with Onegram Tokenization

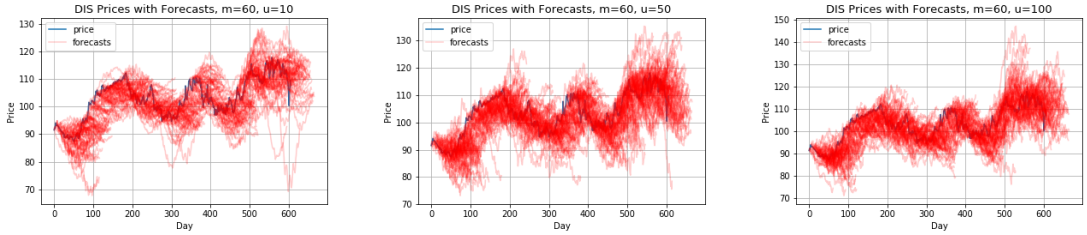


Figure 32: Tokenization Comparison, DIS Forecasts with Bigram Tokenization

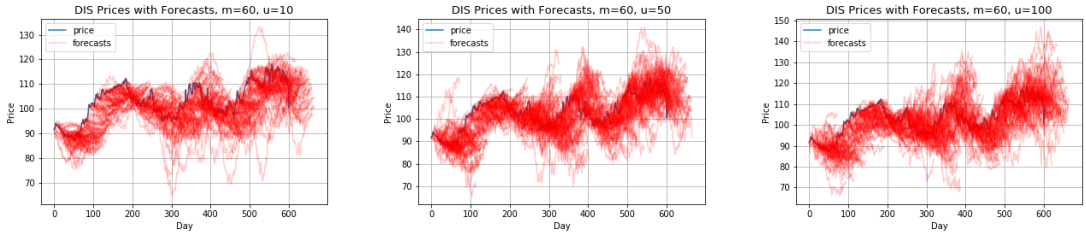


Figure 33: Tokenization Comparison, DIS Forecasts with Trigram Tokenization

8.5.2 Tokenization Comparison: DIS Train Loss

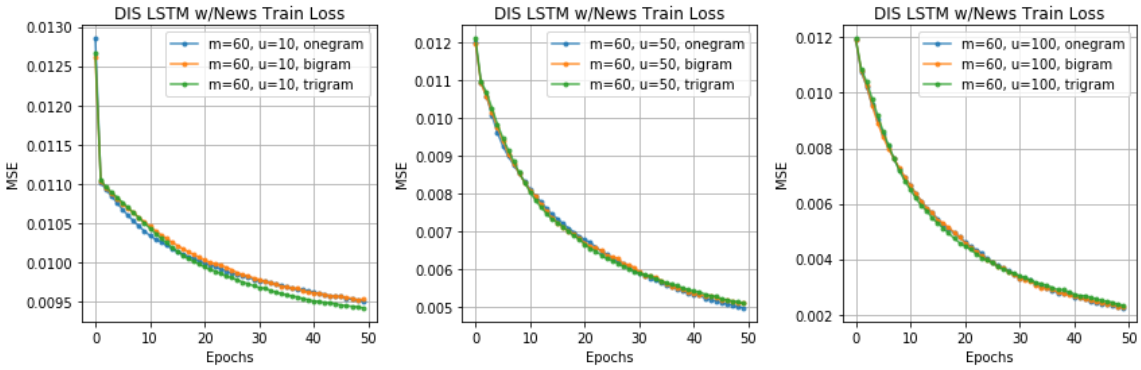


Figure 34: DIS Train Loss Comparison by Tokenization

8.5.3 Tokenization Comparison: DIS Test Loss

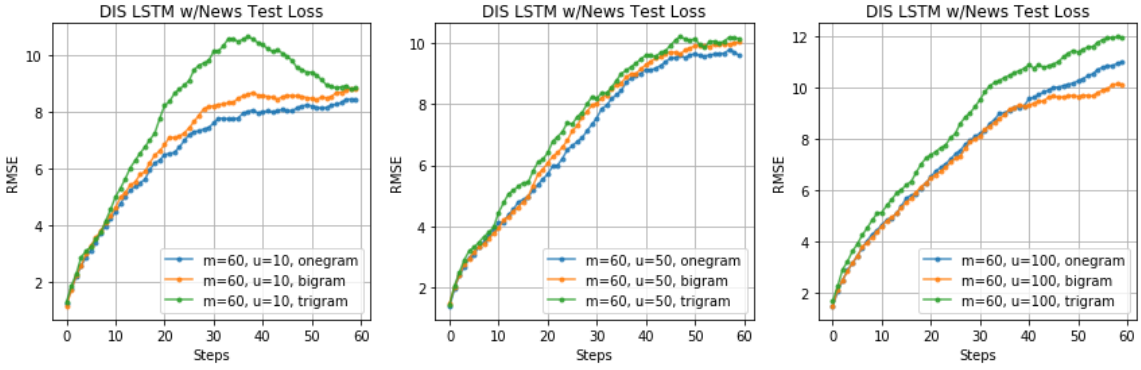


Figure 35: DIS Test Loss Comparison by Tokenization

8.6 ARIMA-Enriched LSTM with Stock Prices and News Articles

Recall that $m = n = 60$ for ARIMA-Enriched LSTMs and that all three tokenizations will be used.

The ARIMA-Enriched LSTM model for DIS exhibits the linear-like persistence from the ARIMA baseline but to a much lesser degree. Moreover, the linear-like persistence does have more curvature

to it than the ARIMA baseline, visually.

The ARIMA-Enriched version fits with incredibly low MSE values, as does the LSTM with Stock Prices and News Articles model, but the former does so almost immediately whereas the latter does not seem to converge completely even during the 50 epochs. Granted, however, the latter does converge more smoothly than the ARIMA-Enriched versions, but the MSEs are on such a small scale of .0001 that this is a nonproblem.

Despite the challenges from the ARIMA baseline returning, the ARIMA-Enriched model under the three different tokenizations generally performs better than the other models and strictly outperforms the other models at every timestep up to $m = 60$ timesteps into the future when $u = 50$ and $u = 100$. Even though much more tuning is required to address the challenges of ARIMA linear-like persistence, the improved performance of the ARIMA-Enriched model can be attributed to a combination of bias accurate ARIMA and variance accurate LSTM with Stock Prices and News Articles.

8.6.1 ARIMA-Enriched LSTM with Stock Prices and News Articles: Forecasts

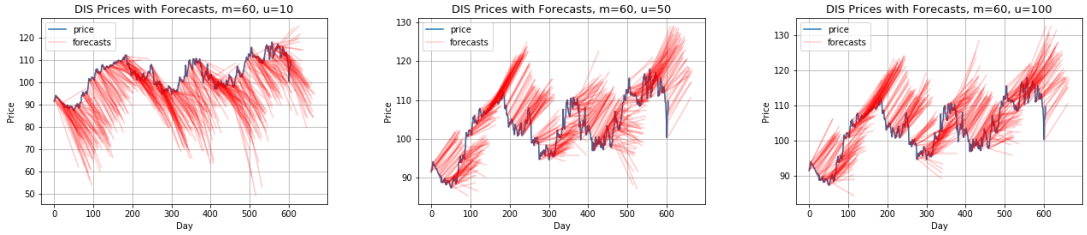


Figure 36: ARIMA-Enriched LSTM, DIS Forecasts with Onegram Tokenization

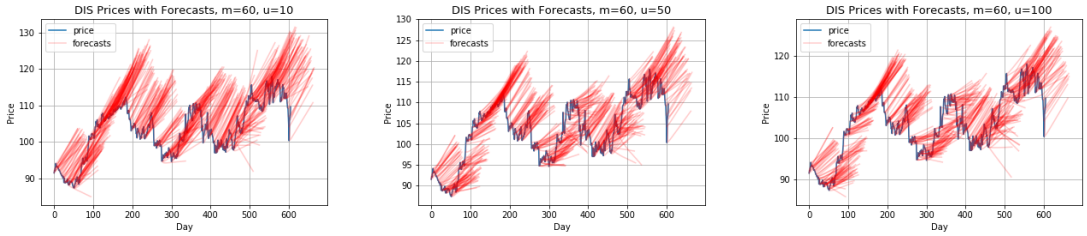


Figure 37: ARIMA-Enriched LSTM, DIS Forecasts with Bigram Tokenization

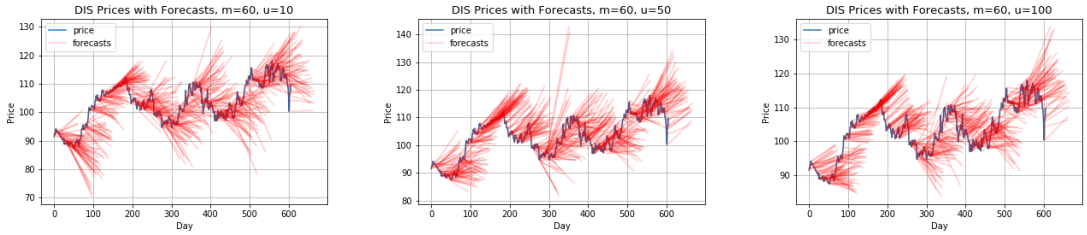


Figure 38: ARIMA-Enriched LSTM, DIS Forecasts with Trigram Tokenization

8.6.2 ARIMA-Enriched LSTM with Stock Prices and News Articles: Train Loss

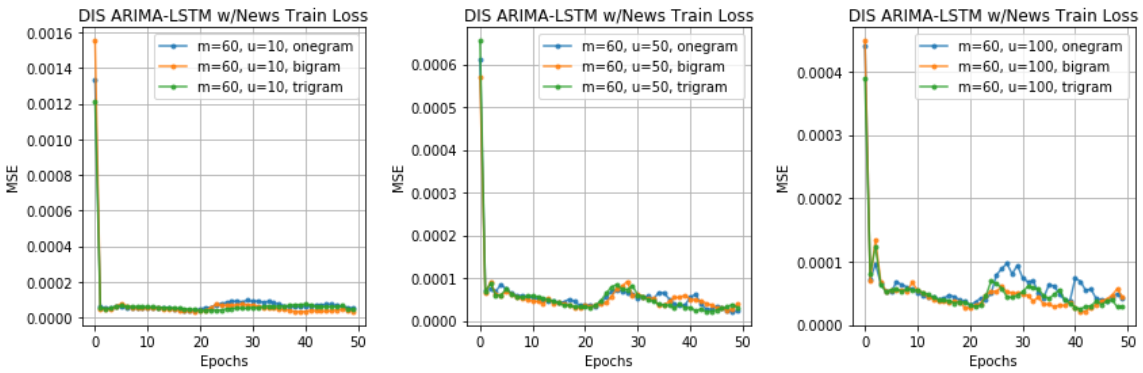


Figure 39: ARIMA-Enriched LSTM with Stock Prices and News Articles: DIS Train Loss

8.6.3 ARIMA-Enriched LSTM: Test Loss

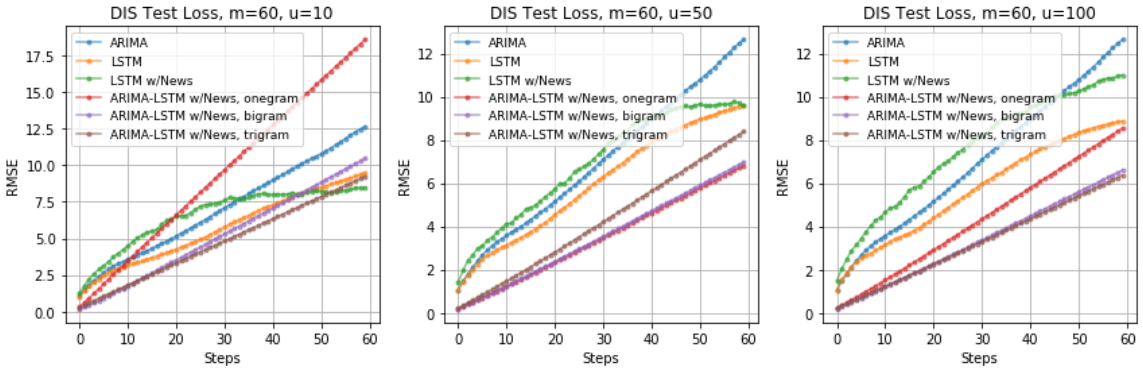


Figure 40: DIS Test Loss Comparison by Model with ARIMA-Enriched LSTM

9 Discussion

9.1 Summary

Qualitative data can be highly beneficial in long-term multistep stock price prediction tasks, as one-step statistical models may struggle in multistep prediction tasks due to an exploding uncertainty and confidence intervals for the predictions, which themselves must be used as inputs to the model. While qualitative data such as news articles can be rich in modeling stock prices, which is governed both by numerical metrics and non-numerical speculation, machine learning models such as the LSTM are highly sensitive to the quality of data available and may struggle to produce accurate predictions due to overfitting variance.

As the statistics and machine learning communities generally have tried to address underfitting and overfitting within their own communities, combining the benefits from highly parametric statistical models and constrastingly highly flexible machine learning models presents a novel opportunity for research. A model that leverages the stability of predictions from the ARIMA model as additional inputs into a LSTM was developed in this paper. Using a few of the popular data representation techniques from both fields, namely first order differencing and Bag of Words tokenization

with PCA, the ARIMA-Enriched LSTM performs in multistep forecasting better than either model alone, even without heavy transformations to the underlying time series or semantic representations for the news articles.

9.2 Future Work

To improve the ARIMA-Enriched LSTM, considerations should be made for both the time series and for the news articles.

On the statistical side, for the time series, further transformations of the time series should be made in order to better satisfy the stationary assumptions of the ARIMA model. Box Cox, log, and power transformations, as well as statistical tests for serial correlations such as Ljung-Box test and models such as SARIMA, Exponential Smoothing, and GARCH, are possible areas to explore. Proper weighting of the importance of certain points near financial reporting dates in the training data can be explored as well.

On the machine learning side, for the news articles, semantic representation of words using popular methods such as Word2Vec and Neural Word Embeddings to produce feature matrices for news can be explored. For the LSTM unit itself, different activation functions beyond the baseline of multivariate regression for the final dense layer can be considered. Variations on the LSTM unit, such as the computationally quicker GRU and the deeper Stacked LSTM, are other possibilities.

Additional data inputs can benefit the hybrid model explored in this paper. On the numerical side, additional predictions from other models or other financial metrics, such as volume, Beta, etc., may enrich the model. On the non-numerical side, NYC was a free API, but it would make more sense to use Bloomberg's API, resources permitting, to better capture news information that is readily used by investors who are potential market movers for public equities.

10 Code Files

The included zip file with the thesis includes only code files. Stored news data, databases, models, losses, images, etc., were removed so that it is more manageable to examine the code. Note that coding was done in Jupyter Notebook and Python, and the pythonic convention of having plots start from $t = 0$, which represents $t = 1$ in the mathmatical notation in the paper, was used throughout the plots in the paper.

A list of code files with descriptions are provided below, with the .py files in the python_scripts folder.

- **create_lightweight_database.ipynb** Creates a SQLite database for AAPL, AMZN, DIS, and GS by scraping Yahoo Finance
- **get_nyt_articles.ipynb** Calls the NYT API and scrapes NYT articles and stores each revelant article in raw text format as a pickled object
- **get_stock_prices.ipynb** Creates a SQLite database for all S&P 500 Companies by scraping Yahoo Finance
- **gpucheck.ipynb** Checks if GPUs are used for faster computing while on AWS
- **run_models.ipynb** Runs the models discussed in this paper and produces all plots, making calls to the .py files for defined functions
- **model_functions.py** Contains all functions to preprocess the data, run the models, and perform forecasts

- **save_load_functions.py** Contains all functions to save and load models, losses, plots, dataframes, etc.
- **tokenization_functions.py** Tokenizes the pickled raw news article data

11 Acknowledgements

I would like to acknowledge Professor Pierre Jacob, Assistant Professor of Statistics, at Harvard University, for his advice and guidance during this project.

References

[1] Lee, K., & Timmons, R. (2007). Predicting the Stock Market with News Articles. Retrieved from https://nlp.stanford.edu/courses/cs224n/2007/fp/timmonsr-kylee84.pdf

[2] Sareyan, A. (2018, June 10). Investing with NLP: Market Predictions. Retrieved from https://cs230.stanford.edu/projects_spring_2018/reports/8291206.pdf

[3] Bird S., Klein, E., & Loper, E. (2012, July 1). Accessing Text Corpora and Lexical Resources [Documentation]. Retrieved from https://www.nltk.org/book/ch02.html

[4] Sagar-Fenton, B., & McNeill, L. (2018, June 24). How many words do you need to speak a language? [BBC News]. Retrieved from https://www.bbc.com/news/world-44569277

[5] Olah, C. (2015, August 27). Understanding LSTM Networks [Blog post]. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] Torch Contributors (2018). TORCH.NN [Documentation]. Retrieved from https://pytorch.org/docs/stable/nn.html

[7] Multiple Contributors (2019). Recurrent Layers [Documentation]. Retrieved from https://keras.io/layers/recurrent/