Time-Series Force Cycle Prediction using Machine Learning (Deep Learning) and Statistical Techniques

Dhananjay Shrouty (UNI : ds3521)

23rd December 2017

Fall 2017

Goal

The goal of the project is to predict the behavior of the artificial muscle over time using statistical methods and machine learning

Abstract

The artificial muscles developed by Creative Machines lab holds a great promise in the world of Soft Robotics. It is inspired by natural muscle and is electrically driven. All our muscles change the way they respond to situations in real life. The artificial muscle shows this characteristic too. When it is subjected to cycles of force, the response it shows to external force stimuli changes over time. A great challenge is to predict how the muscle will behave in the future and how will it react to the external stimuli. In this project, I tell about how machine learning/deep learning and statistical techniques can help predict the behavior of the artificial muscle.

Literature review

Deep Learning

Deep learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised or unsupervised. Supervised learning comprises when the labels related to the data points are known and unsupervised learning constitutes the methods in which labels are not provided in the dataset and the algorithm has to figure out by itself to assign the data points into different meaningful groups (or clusters).
Deep learning uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input. For supervised learning tasks, deep learning methods obviate feature engineering, by translating the data into compact intermediate representations akin to principal components, and derive layered structures that remove redundancy in representation.

Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in

applications difficult to express with a traditional computer algorithm using rule-based programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to axons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream.

Typically, neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times. The distinction between a simple neural network and deep learning neural networks is, deep learning neural networks have much more layers compared to a simple neural network's few layers

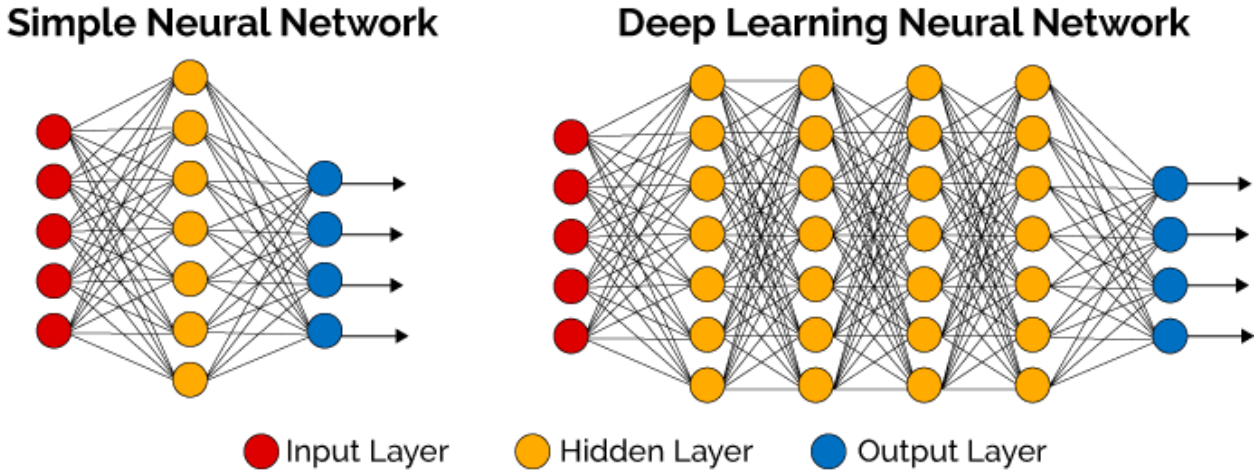

Figure 1. A simple neural network and a deep learning neural network

RNN Overview

Basic RNNs are a network of neuron-like nodes, each with a directed (one-way) connection to every other node. Each node (neuron) has a time-varying real-valued activation. Each connection (synapse) has a modifiable real-valued weight. Nodes are either input nodes (receiving data from outside the network), output nodes (yielding results), or hidden nodes (that modify the data en route from input to output).

For supervised learning in discrete time settings, sequences of real-valued input vectors arrive at the input nodes, one vector at a time. At any given time step, each non-input unit computes its

current activation (result) as a nonlinear function of the weighted sum of the activations of all units that connect to it. Supervisor-given target activations can be supplied for some output units at certain time steps. For example, if the input sequence is a speech signal corresponding to a spoken digit, the final target output at the end of the sequence may be a label classifying the digit.

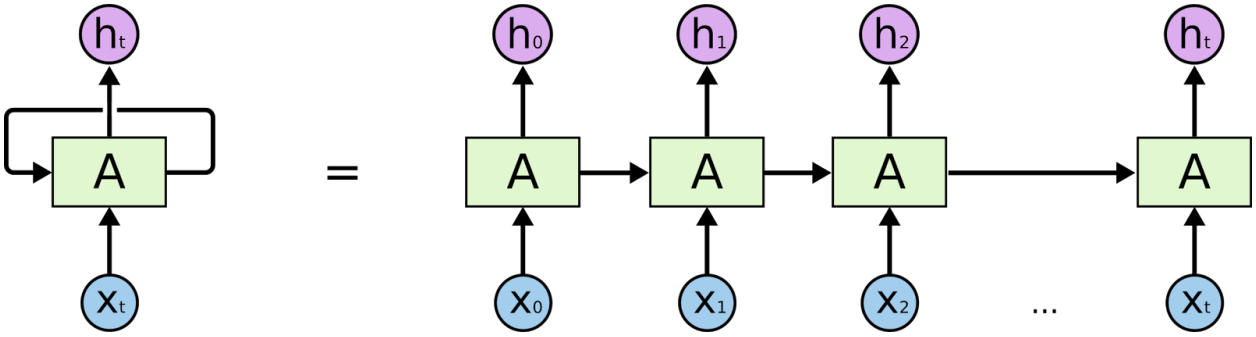

Figure 2. An unrolled recurrent neural network

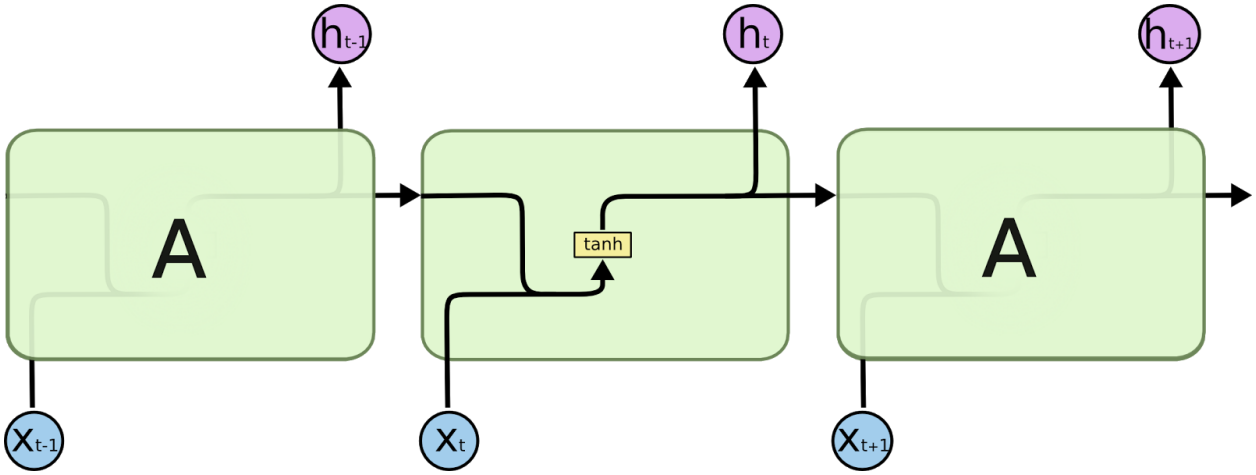

Figure 3. Inside a recurrent neural network

LSTM overview

Long short-term memory (LSTM) block or network is a recurrent neural network which can be used as a building component or block (of hidden layers) for an eventually bigger recurrent neural network. The LSTM block is itself a recurrent network because it contains recurrent connections similar to connections in a conventional recurrent neural network.

An LSTM block is composed of four main components: a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals; hence the word "memory" in LSTM. Each of the three *gates* can be thought of as a

"conventional" artificial neuron, as in a multi-layer (or feedforward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as *regulators* of the flow of values that goes through the connections of the LSTM; hence the denotation "gate". There are connections between these gates and the cell. Some of the connections are recurrent, some of them are not.

The expression *long short-term* refers to the fact that LSTM is a model for the *short-term memory* which can last for a *long* period of time. There are different types of LSTMs, which differ among them in the components or connections that they have.

An LSTM is well-suited to classify, process and predict time series given time lags of unknown size and duration between important events.

LSTMs were developed to deal with the exploding and vanishing gradient problem when training traditional RNNs. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs, hidden Markov models and other sequence learning methods in numerous applications.

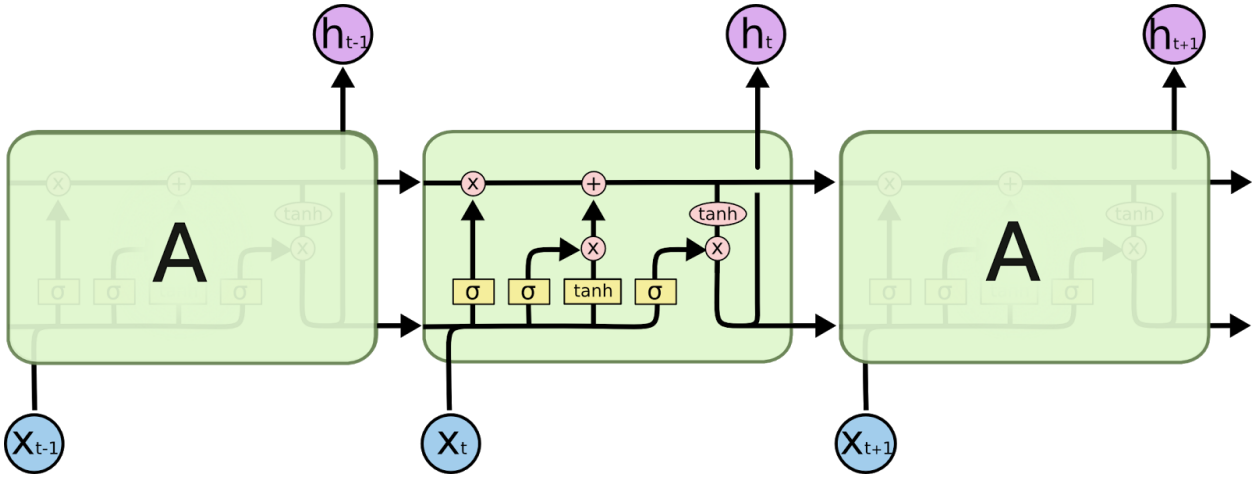

Figure 4. Inside a Long Short-term Memory (LSTM) Network

Sequence to Sequence using Encoder - decoder architecture

Sequences pose a challenge for DNNs because they require that the dimensionality of the inputs and outputs is known and fixed. In this paper, we show that a straightforward application of the Long Short-Term Memory (LSTM) architecture can solve general sequence to sequence problems. The idea is to use one LSTM to read the input sequence, one timestep at a time, to obtain large fixed dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector. The second LSTM is essentially a recurrent neural network language model except that it is conditioned on the input sequence. The LSTM's ability to

successfully learn on data with long range temporal dependencies makes it a natural choice for this application due to the considerable time lag between the inputs and their corresponding outputs. The encoder network encodes the entire entire input sequence into a vector and the decoder decodes the encoded input one entity at a time. The decoder outputs are determined by the encoded inputs and previous decoded outputs. Using of bi-directional LSTMs gives better prediction for machine translation purposes.

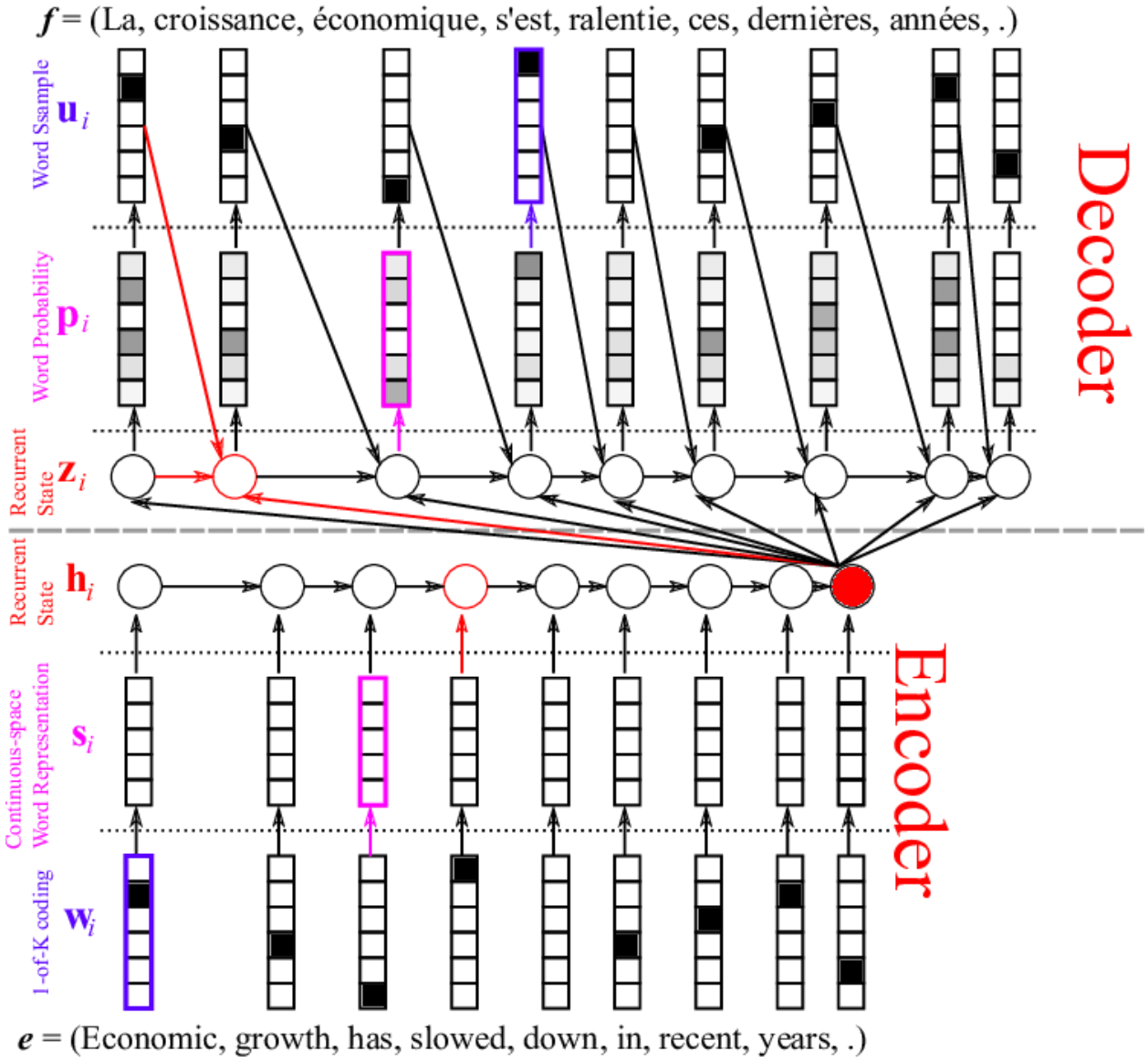

Figure 5. Using Encoder-Decoder architecture for Neural Machine Translation

ARIMA overview

ARIMA stands for Autoregressive Integrated Moving Average models. Univariate (single vector) ARIMA is a forecasting technique that projects the future values of a series based entirely on its own inertia. Its main application is in the area of short term forecasting requiring at least 40 historical data points. It works best when your data exhibits a stable or consistent pattern over time with a minimum amount of outliers. Sometimes called Box-Jenkins (after the original authors), ARIMA is usually superior to exponential smoothing techniques when the data is reasonably long and the correlation between past observations is stable

The first step in applying ARIMA methodology is to check for stationarity. "Stationarity" implies that the series remains at a fairly constant level over time. If a trend exists, as in most economic or business applications, then your data is NOT stationary. The data should also show a constant variance in its fluctuations over time. This is easily seen with a series that is heavily seasonal and growing at a faster rate. In such a case, the ups and downs in the seasonality will become more dramatic over time. Without these stationarity conditions being met, many of the calculations associated with the process cannot be computed.

If a graphical plot of the data indicates nonstationarity, then one should "difference" the series. Differencing is an excellent way of transforming a nonstationary series to a stationary one. This is done by subtracting the observation in the current period from the previous one. If this transformation is done only once to a series, one can say that the data has been "first differenced". This process essentially eliminates the trend if the series is growing at a fairly constant rate. If it is growing at an increasing rate, one can apply the same procedure and difference the data again. The data would then be "second differenced".

"Autocorrelations" are numerical values that indicate how a data series is related to itself over time. More precisely, it measures how strongly data values at a specified number of periods apart are correlated to each other over time. The number of periods apart is usually called the "lag". For example, an autocorrelation at lag 1 measures how values 1 period apart are correlated to one another throughout the series. An autocorrelation at lag 2 measures how the data two periods apart are correlated throughout the series. Autocorrelations may range from +1 to -1. A value close to +1 indicates a high positive correlation while a value close to -1 implies a high negative correlation.

ARIMA methodology attempts to describe the movements in a stationary time series as a function of what are called "autoregressive and moving average" parameters. These are referred

to as AR parameters (autoregessive) and MA parameters (moving averages). An AR model with only 1 parameter may be written as

$$X(t) = A(1) * X(t-1) + E(t)$$

where X(t) = time series under investigation

A(1) = the autoregressive parameter of order 1

X(t-1) = the time series lagged 1 period

E(t) = the error term of the model

This simply means that any given value X(t) can be explained by some function of its previous value, X(t-1), plus some unexplainable random error, E(t). If the estimated value of A(1) was .30, then the current value of the series would be related to 30% of its value 1 period ago. Of course, the series could be related to more than just one past value. For example,

$$X(t) = A(1) * X(t-1) + A(2) * X(t-2) + E(t)$$

This indicates that the current value of the series is a combination of the two immediately preceding values, X(t-1) and X(t-2), plus some random error E(t). Our model is now an autoregressive model of order 2.

A second type of Box-Jenkins model is called a "moving average" model. Although these models look very similar to the AR model, the concept behind them is quite different. Moving average parameters relate what happens in period t only to the random errors that occurred in past time periods, i.e. E(t-1), E(t-2), etc. rather than to X(t-1), X(t-2), (Xt-3) as in the autoregressive approaches. A moving average model with one MA term may be written as follows

$$X(t) = -B(1) * E(t-1) + E(t)$$

The term B(1) is called an MA of order 1. The negative sign in front of the parameter is used for convention only and is usually printed out automatically by most computer programs. The above model simply says that any given value of X(t) is directly related only to the random error in the previous period, E(t-1), and to the current error term, E(t). As in the case of autoregressive models, the moving average models can be extended to higher order structures covering different combinations and moving average lengths.

ARIMA methodology also allows models to be built that incorporate both autoregressive and moving average parameters together. These models are often referred to as "mixed models". Although this makes for a more complicated forecasting tool, the structure may indeed simulate

the series better and produce a more accurate forecast. Pure models imply that the structure consists only of AR or MA parameters - not both.

The models developed by this approach are usually called ARIMA models because they use a combination of autoregressive (AR), integration (I) - referring to the reverse process of differencing to produce the forecast, and moving average (MA) operations. An ARIMA model is usually stated as ARIMA(p,d,q). This represents the order of the autoregressive components (p), the number of differencing operators (d), and the highest order of the moving average term. For example, ARIMA(2,1,1) means that you have a second order autoregressive model with a first order moving average component whose series has been differenced once to induce stationarity.

Technical approach and Results

Data visualization

Figure 6 shows how the training data looks for our problem. We have multiple force cycles which are applied to the artificial muscle and we see a pattern that the behavior of the muscle response to the force changes with each cycle.

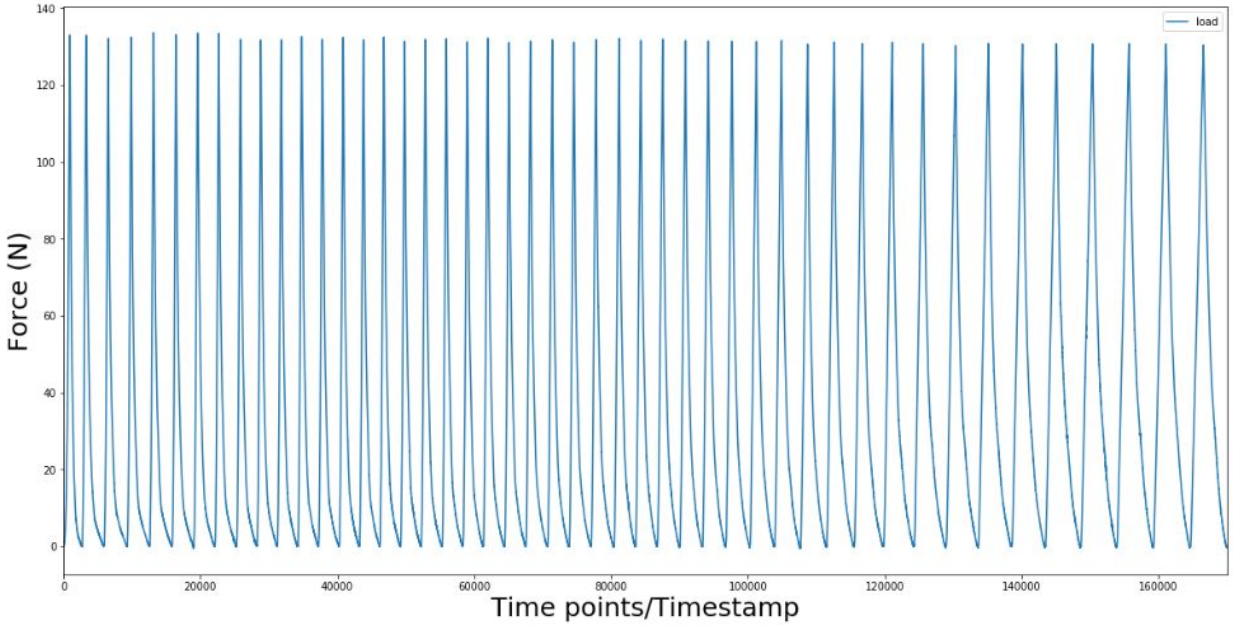

Figure 6. Training data for the problem

ARIMA Approach

ARIMA model was trained in a way so that after making a prediction it will take the entire training set and the newly predicted value to predict the next point and so on. The model was trained again and again on the updated training set and therefore this process take a lot of time in generating predictions. Figure 7 shows the results that were obtained using the ARIMA approach.

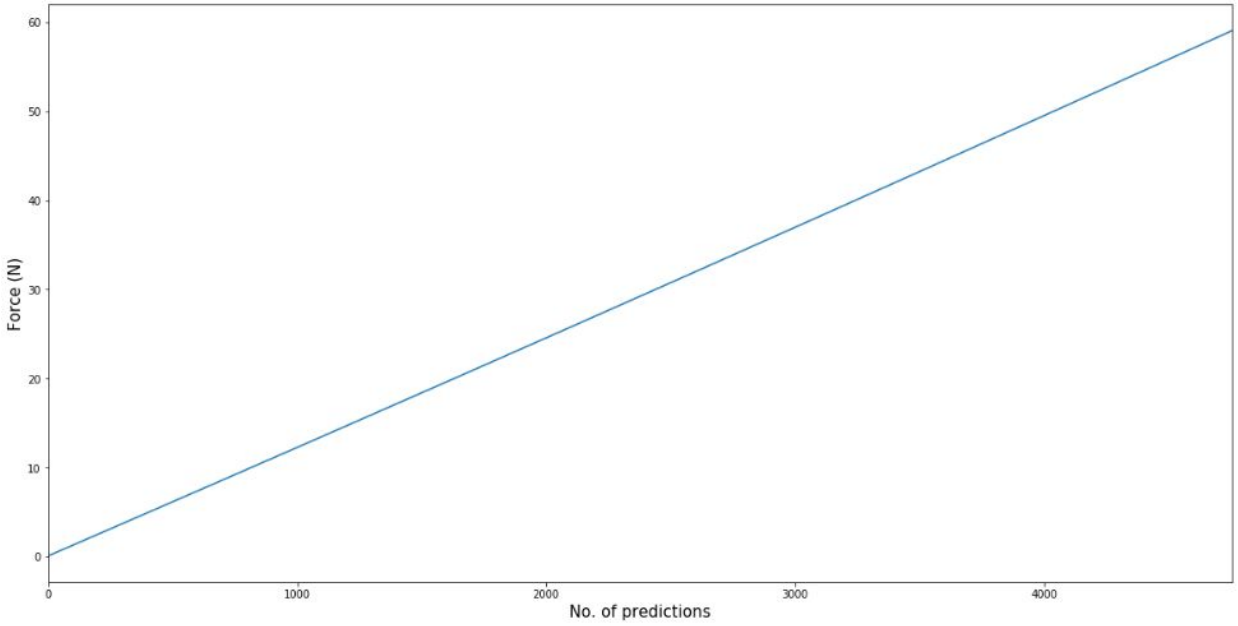

Figure 7. Predicted results from the ARIMA approach

LSTM approach

The next approach used a single LSTM and the time series data was given to it in a series approach to predict the next data point. The model was used trained on the entire training data to learn the weights and the predictions are shown in figure 8.

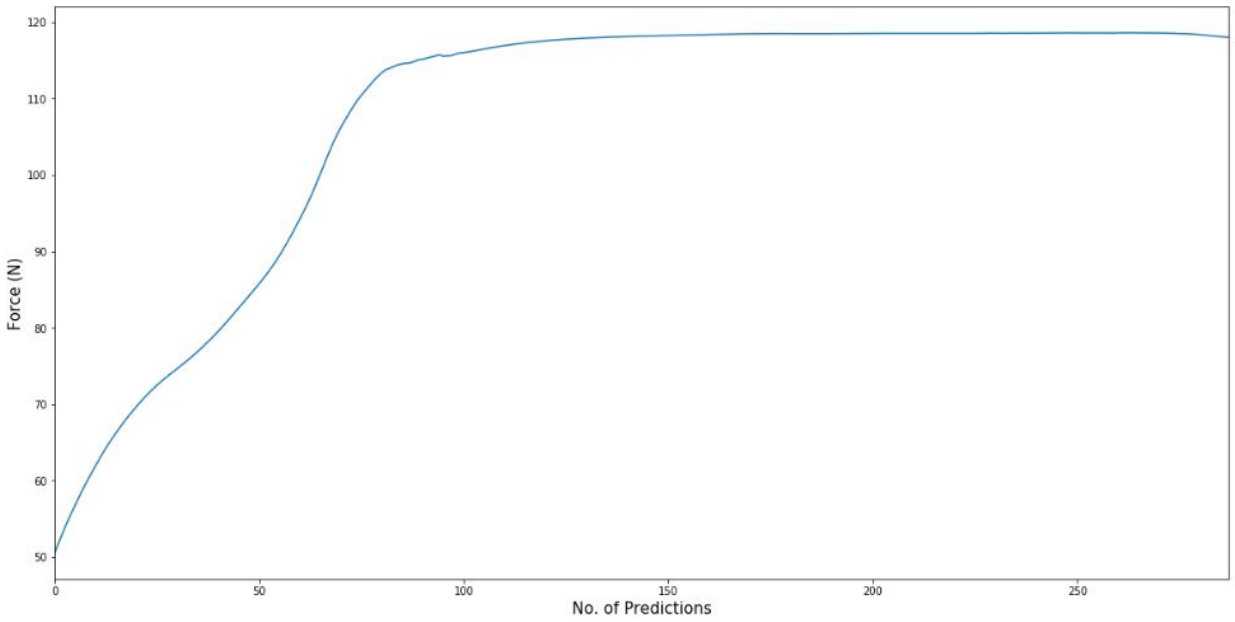

Figure 8. Predicted results from LSTM approach

The learning curves of the first approach can be found below

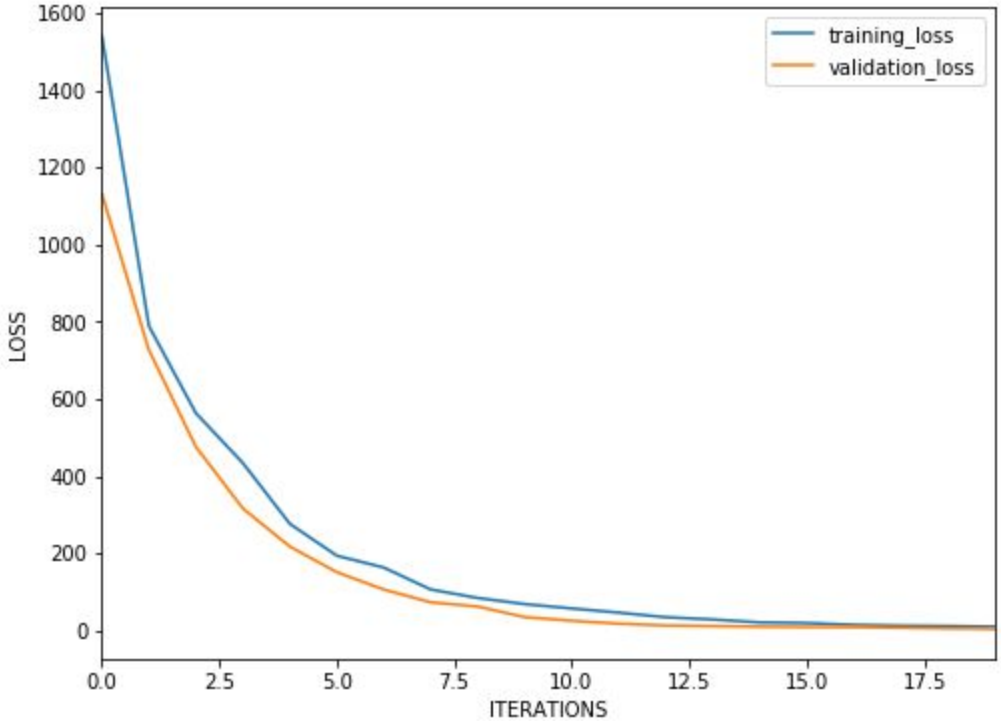

Figure 9. Learning curves for the LSTM approach

Encoder-Decoder Approach

The last approach used encoder decoder architecture to first encode 300 sequence long time series data, encode it and then try to decode it to get the prediction. The model was trained on the entire data to learn the weights of the encoder and decoder. The predictions of the encoder-decoder approach architecture are shown in figure 9.

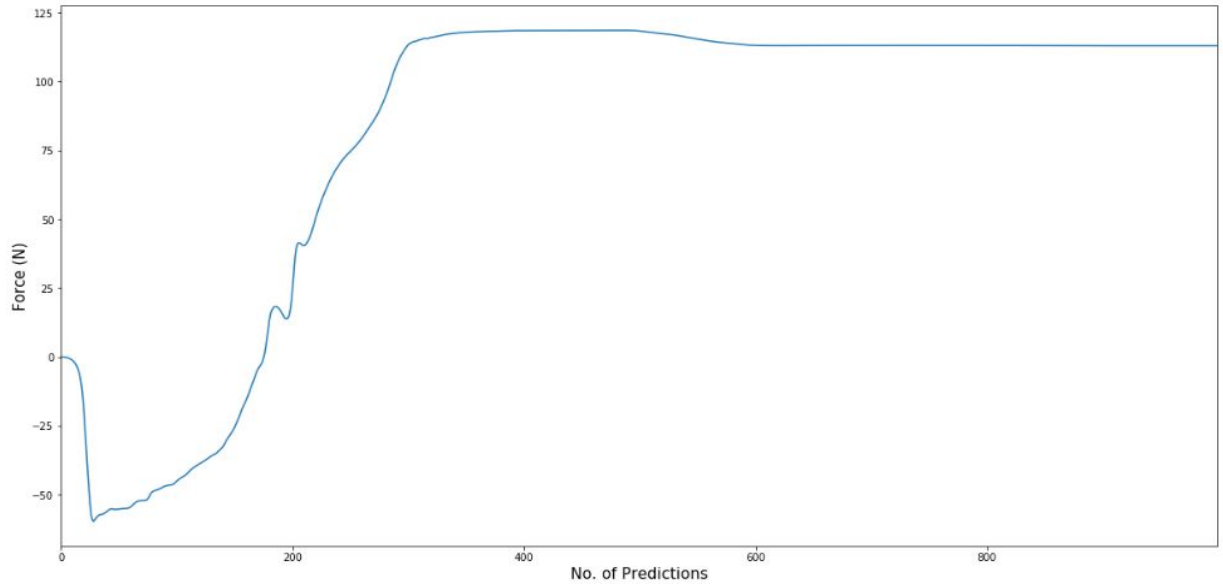

Figure 10. Predicted results from Encoder-Decoder approach

The learning curves of the encoder-decoder approach are shown in figure 11

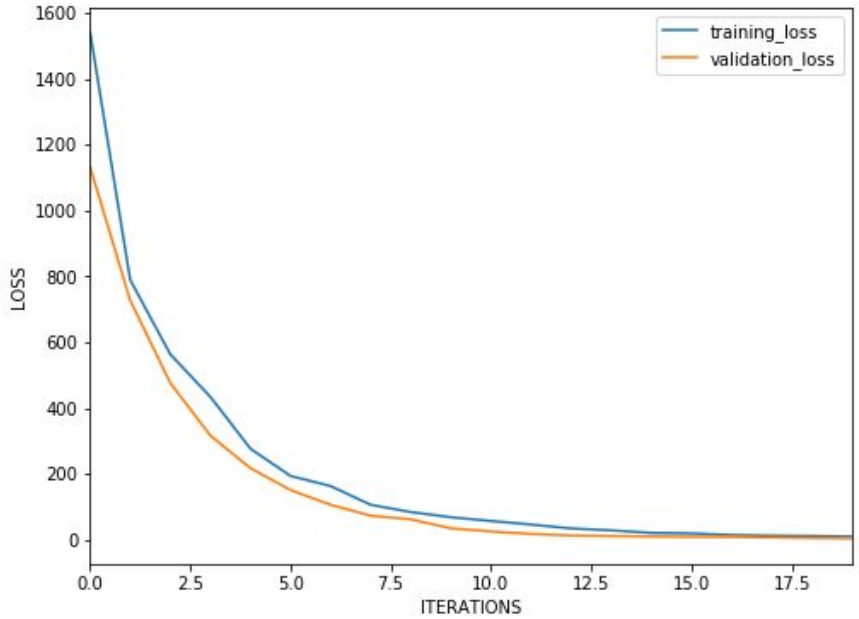

Figure 11. Learning curves for the Encoder-Decoder approach

Conclusions and Future work

Although none of the predict results were good, it comes to attention that multi-step ahead prediction (or generating predictions out of predictions) is not an easy task as it seems to be because predictions errors amplify and when we make predictions out of errored predictions they results will flatten out. The statistical algorithms and deep learning approaches used in the project were state of the art. A few methods of data preprocessing might further help in improving the results

- Undersampling - Choosing samples from dataset which effectively represents the entire dataset (with some approximation errors) and use the dataset on the approaches discussed in the report
- Multiple encoder-decoder stacks (although it will use a lot of computational strength)
- Teacher forcing or Professor forcing – Not correct as we should not tell the algorithm what the prediction should be to predict further predictions
- Taking an entire different approach and defining the characteristics of each cycle and trying to predict what would be the characteristics of the next cycle. This approach might not even require deep learning but since neural networks provide the best results, using deep learning is recommended.

Bibliography

1. Sutskever, Ilya et al, *Sequence to Sequence Learning with Neural Networks*, NIPS 2014
2. Bahdanau,Dzmitry et al, *Neural Machine Translation by Jointly Learning to Align and Translate*, ICLR 2015
3. Qin, Yao et al, *A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction*, arXiv 2014 https://arxiv.org/pdf/1704.02971.pdf
4. Hyland, Stephanie et al., *Real-Values (Medical) Time Series Generation with Recurrent Conditional GANs*, arXiv 2017 https://arxiv.org/pdf/1706.02633.pdf
5. http://pytorch.org/docs/master/torch.html?highlight=bmm#torch.bmm
6. https://github.com/buriburisuri/timeseries_gan
7. http://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
8. https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/viewFile/9692/9467
9. http://chandlerzuo.github.io/blog/2017/11/darnn
10. https://discuss.pytorch.org/t/lstm-time-sequence-generation/1916
11. http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction

Acknowledgments

I would like to thank Aslan Miriyev for all his time, discussions and support for the entire project. He is a great mentor and motivator and encourages people to try new approaches to tackle the problem in hand. I would also like to thank Prof. Hod Lipson for his insights upon how the preprocessing of the data should be done and how the algorithms can make better predictions in the future. I would also like to thank Chad DeChant for his helping me how deep learning methods can be modified for time series analysis.

Online materials

All code files can be found on my github with the link :
https://github.com/dhananjay014/Artificial-Muscle-Project