Modelling the behavior of a soft material-actuator

Aslan Miriyev aslan.miriyev@columbia.edu
Sam Cohen slc2206@columbia.edu
Neil Chen neil.chen@columbia.edu

***Abstract*— Self-contained electrically-driven soft actuators with high strain density, low cost, simple fabrication methods, and low current draw represent a key challenge in soft robotics. We present several methods for predictive analysis of the behavior of such an actuator, with a focus on time-series analysis of actuator load and wear.**

I. INTRODUCTION

A soft, robust, self-contained composite material-actuator exhibiting high actuation stress along with very high actuation strain resolves many of the issues confronted by traditional soft actuation solutions (FEAs, PAMs, DEAs, etc.). The complex electrical, chemical, and physical behavior of these actuators, however, presents multiple steady-state and transient response controls problems.

Here we approach one of these problems: we attempt to evaluate the load characteristic, and in turn the wear, of a single soft material-actuator sample over time.

We evaluate the performance of three separate approaches to modelling the behavior of this soft material-actuator:

1) Function approximation of actuator load cycle behavior via deep learning.
2) Regression analysis on derived features of actuator load cycle characteristic.
3) Fourier analysis/time-series autoregression methods.

Ultimately we find that more input features and/or samples are necessary to usefully model the behavior of such an actuator.

II. DATA

A. Experimental Setup

The experimental setup and data generation process consists of a new (or rejuvenated) actuator sample enclosed in an Instron load cell. The actuator is sufficiently constrained such that it can neither extend nor contract.

During measurement, a cycle begins whenever `Load` reaches $0N$. At the peak of a cycle, a `Load` threshold of roughly $135N$ has been reached, and current is no longer supplied to the actuator. Thus cycle `Load` minima and maxima remain roughly fixed between cycles. This behavior is visible in **Figure 1**.

B. Dataset

The dataset (`raw_cycles.csv`, which can be downloaded here) consists of 2 features and $170,103$ time-series samples measured at $0.100s$ intervals. Please preview the dataset online, or refer to **Table I** for more information.

The data represents 47 actuation *cycles*. A *cycle* consists of a *heating* phase and a cooling phase. During *heating*, the actuator exerts positive force, and thus `Load` is convex increasing with time. During *cooling*, the actuator gradually exerts less force, and thus `Load` is convex decreasing with time.

Feature	Unit	Format
`Time`	seconds	3 decimal places
`Load`	Newtons	5 decimal places

TABLE I: Unit and format of features in input dataset.

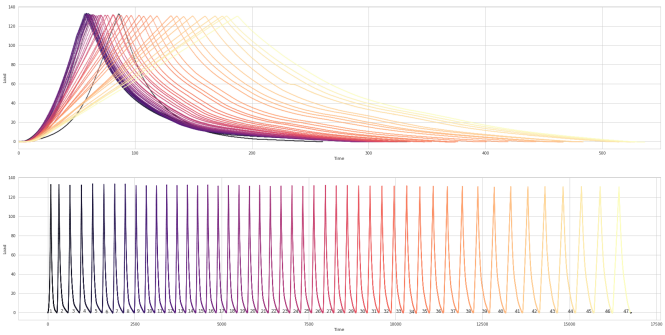

Fig. 1: Heating-cooling cycles over time.

Specified in this manner, the dataset consists of 47 complete cycles. Note that the local maximum and local minimum `Load` values of each cycle are roughly equal.

C. Exploratory Data Analysis

We begin by plotting and quantifying certain characteristics of the dataset.

We find that the average cycle heating and cooling times are 91.29 seconds and 265.32 seconds, respectively. We also note that cooling and heating durations appear to grow exponentially over the number of cycles. This trend is likely due to a variety of physical phenomena occurring within the actuator, such as gradual ethanol escape and internal heat build up.

III. DERIVED FEATURES

A. Motivation

We attempt to inject expert knowledge about the actuator system's behavior by constructing 11 derived features. These features encode approximately constant maximum/minimum-`Load` behavior, n^{th}-order moments of each cycle, cycle index, and more. Each feature is manually computed over the entire dataset; derived features are excluded from validation/testing sets.

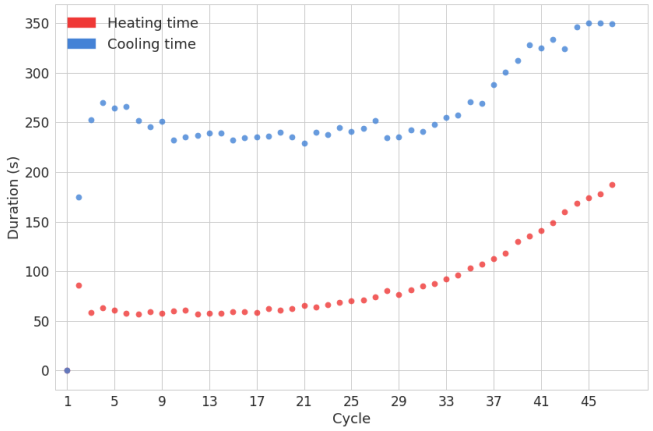

Fig. 2: Heating and cooling duration as a function of cycle number.

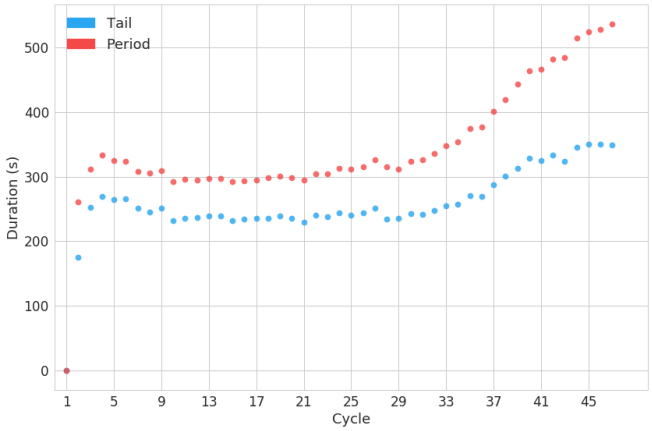

Fig. 3: Tail (time elapsed while `Load` $\leq$ (0.1×`Max Load`) and period duration as a function of cycle number.

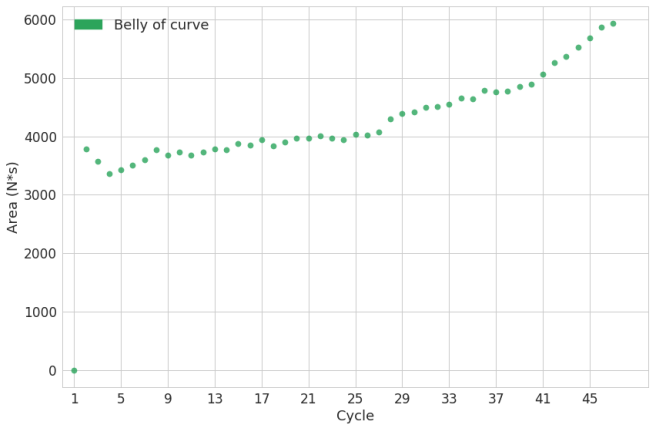

Fig. 4: Belly-of-cooling-curve area as a function of cycle number.

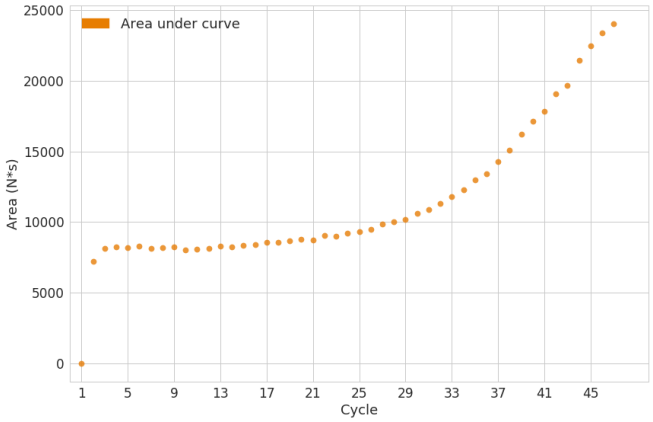

Fig. 5: Area under `Load` function as a function of cycle number.

B. Computation

Table II enumerates all derived features.

IV. Regression Analysis

We use the following approach to generate models and evaluate their performance on derived feature prediction:

1) Generate time-series cross-validation training-validation splits, delimiting data by each cycle.
2) Perform grid-search on regression parameters to identify models which best predict derived features

We then develop a regression model from these derived features:

1) Split cycles into four piecewise curves, as enumerated in *IV.A*.
2) Perform grid-search on regression parameters, minimizing loss (per piecewise curve) between predicted `Load` and ground truth `Load` as a function of `Time`.

A. Piecewise cycle splitting

We split curves into continuous piecewise sections using two simple methods: fixed boundaries and K-means clustering.

1) Fixed boundaries: We delimit curves at four points. Let $t_0 = 0$ denote the time at which a cycle begins; $t_{max} =$ the time at which a cycle reaches its maximum `Load`; and t_f denote the time at which a cycle finishes, such that $t_f =$ the `Period` of a cycle. Then:

$$t_1 = \frac{t_{max} - t_0}{2}$$
$$t_2 = t_{max}$$
$$t_3 = \frac{t_f - t_{max}}{2}$$

and we regress separately on the following subsections of a cycle's `Period`. Let $c_i =$ the set of observations in the i^{th} cycle of the dataset X. Then we define the following subsets of c_i:

$$p_1 = c_i[t_0 : t_1]$$
$$p_2 = c_i[t_1 : t_2]$$
$$p_3 = c_i[t_2 : t_3]$$
$$p_4 = c_i[t_3 : t_f]$$

Feature	Meaning
Time	Absolute time since $0.00s$
Load	Absolute load from $0N$
Min	One-hot encoding of local minima
Max	One-hot encoding of local maxima
Cycle	Index of cycle from 0
Area	Area under cycle curve, calculated as $\int_{t_{min}}^{t_{max}} \texttt{Load}\, dt_i$
Heating	Time elapsed while heating, calculated as $t_{max} - t_{min}$
Cooling	Time elapsed while cooling, calculated as $t_{end} - t_{max}$
HCprop	Proportion $\frac{\texttt{Heating}}{\texttt{Cooling}}$ of heating time to cooling time
Period	Period of cycle, equivalent to $\texttt{Heating} + \texttt{Cooling}$
Tail	Time elapsed while $\texttt{Load} \leq (0.1 \times \texttt{Max Load})$
Belly	Multiple integral between $\texttt{Cooling}$ curve and line tangent to both $\texttt{Max Load}$ and $\texttt{Load}_{t_{end}}$
Kurt	Kurtosis of curve (fourth standardized moment)
Skew	Skewness of curve

TABLE II: All derived features and calculations involved.

2) K-means clustering: We delimit curves based on the clusterings generated by K-means with $K = 2$. We run K-means, preserving no parameters, independently on the heating and cooling curves of each cycle. New (validation) points can be classified into cluster via the $1-$nearest-neighbors algorithm. K-means (as used here) computes the centroids of observations as

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

where $S_i =$ the set of observations x_j in cluster i. Since the load characteristic of the heating curve of each cycle is convex increasing, and we force $K = 2$, we can identify the elbow of each heating curve in an unsupervised manner. Similar reasoning applies to the cooling curve of each cycle. Then observations are once again clustered into one of p_1, p_2, p_3, p_4.

B. Piecewise regression performance

We identify a model which reports optimal validation accuracy via the following algorithm, using the definitions of t_i expressed in *IV.A*. r_i denotes a regression function using ridge (Tikhonov) regularization.

TRAIN(X, y)
1 $P_1, \ldots, P_4 = \{\}$
2 $\Gamma = [3.40, 10.0, 3.40, 5.60]$ // regularization penalties
3 $X = X \cup \bigcup_{i=2}^{5} i^{\text{th}}$-order features of X
4 **for** $c \in X$ // $c =$ a discrete cycle in X
5 **for** $i = 1$ **to** 4
6 $p_i = c[t_{i-1} : t_i]$ // $t_2 = t_{max}$ and $t_4 = t_f$
7 $P_i = P_i \cup p_i$
8 **for** $i = 1$ **to** 4
9 fit r_i on $(P_i, y[P_i])$, $\mathcal{L}_2$ parameter $\alpha_{r_i} = \Gamma_i$

Then new observations, with input features `Time` and `Cycle` (cycle index) only, map to predicted `Load` as follows. Let $X =$ the matrix of `Time` and `Cycle` observations for a single cycle.

PRED(X)
1 **for** $i = 1$ **to** 4
2 predict $y_i = r_i(X)$
3 **return** $y_1 + \cdots + y_4$

The performance of this algorithm is listed in **Table III**, where the best possible score is 1.000.

Feature	R^2 coefficient
Min	1.000
Max	1.000
Cycle	1.000
Area	0.953
Heating	0.934
Cooling	0.569
HCprop	0.733
Period	0.773
Tail	0.569
Belly	0.716
Kurt	0.854
Skew	0.479

TABLE III: Performance of piecewise regression algorithm.

Since the minimum and maximum `Load` of the experimental setup are constrained at roughly constant values, the `Period` of each cycle is the dominant predictor of cycle `Load` characteristic. Recursive feature elimination (RFE) confirms this.

The performance of our algorithm on predicting the `Area` feature, itself a function of `Period`, as a function of train-validation split ratio is depicted in **Figure 6**. Regression on `Load` values, given `Time` and `Cycle` index, is then fairly effective; see **Figure 7**.

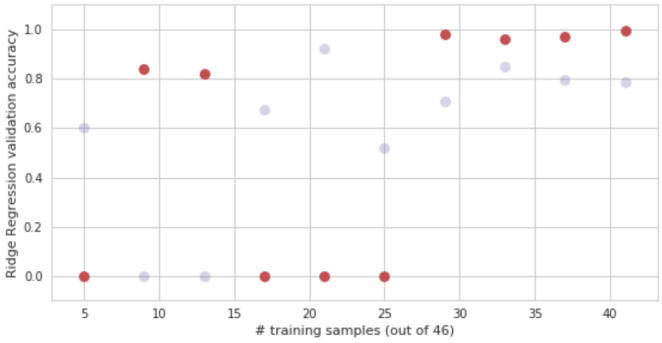

Fig. 6: Cross-validation performance on `Area` prediction via our ridge regression pipeline.

V. Learning Periodic Functions

We begin by fine-tuning a single model to learn simple periodic functions. As our eventual goal is to predict the behavior of a non-periodic Time series, we opt to similarly treat this task as a Time series forecasting problem in which our samples are not independent, but are rather related to one another across time. Our real data is fairly limited, and so we also attempt to simulate this constraint by only generating 20 cycles for each of the three synthetic datasets.

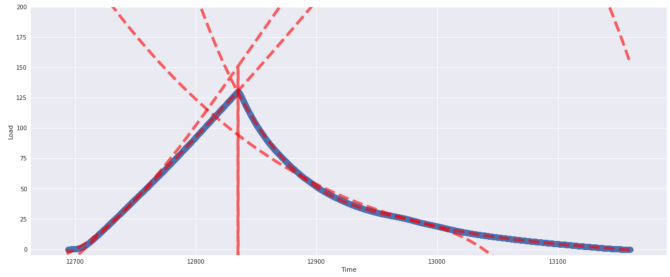

Fig. 7: Forecasting the 40^{th} cycle `Load` characteristic, with algorithm trained on 39 previous cycles. (Predicted piecewise `Load` curves in *red*; ground truth observations in *blue*).

A. Formatting Data for Supervised Learning

To build our set of examples, we first calculated the period P of a cycle. A single example is formatted such that it contains three full cycles worth of *input* points (*Px3*) and one full cycle worth of *label* points (P). We then utilize the "sliding window" method to step along our univariate time series one point at a time, collecting the proceeding *Px4* points to build an example. This process continues along the dataset until there aren't enough remaining points to produce a full *input* and *label* pairing.

The final result is stored in a Pandas Dataframe to be fed into the Neural Network.

B. Network Architecture

We apply Long short-term memory (LSTM) neural networks to best capture the time dependency of our data. The network's architecture and parameters are depicted in **Figure 8** and are further summarized below:

1) Input Layer (*Px3* data points)
2) LSTM Layer (120 memory gate neurons)
3) Dense Layer (100 neurons)
4) Output Layer (P predictions)

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (150, 200)                161600
_________________________________________________________________
dense_5 (Dense)              (150, 180)                36180
_________________________________________________________________
dense_6 (Dense)              (150, 62)                 11222
=================================================================
Total params: 209,002
Trainable params: 209,002
Non-trainable params: 0
```

Fig. 8: Periodic LSTM architecture parameters.

C. Prediction & Results

Below are our results on a sine wave, triangle wave, and square wave. Our plots are color coded as follows:

1) **Gray**: Training/Validation data
2) **Blue**: Test example input
3) **Green**: Test example label
4) **Red**: Test example forecast

Note that we can hardly see any green points in the figures– indicating accurate forecasting results.

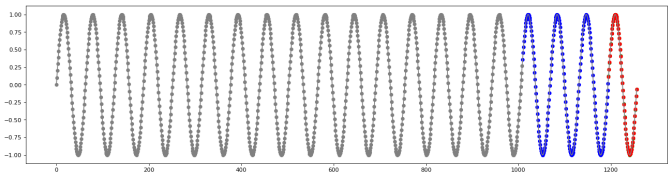

Fig. 9: Forecasting the sine wave with a LSTM network.

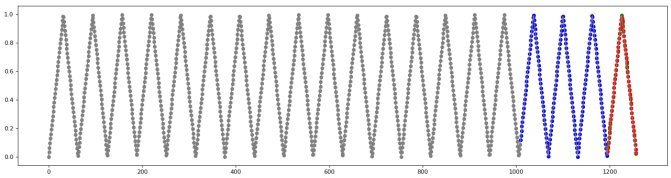

Fig. 10: Forecasting the Triangle wave with a LSTM network.

We quantify our prediction results by calculating the *Root Mean Squared Error* (RMSE) and the *Mean Absolute Error* (MAE) percentage for each wave forecast. To calculate the MAE %, we divide the MAE by the difference between a cycle's peak and trough. In this case, all cycles have the same peak and trough, so we simply subract the `min` from the `max` of the data set. By making the error relative to cycle height in order to produce a percentage, our results are more interpretable.

It appears as though smooth functions are far easier to learn than functions with cusps or discontinuities, as the model was able to achieve the best results on the `sine` wave with the least training. **Table IV** contains these results across all three periodic functions, where a *batch size* of 150 examples was used for each.

It is very interesting to note that the below results were achieved after 1500 epochs for both the square and triangle wave, but only after 100 epochs for the sine wave. It would appear that smooth functions are far easier to learn than ones with cusps or discontinuities. All three waves were trained using a *batch size* of 150 examples.

Function type	Epochs	MAE %	RMSE
`Sine`	100	0.111	0.003
`Triangle`	1500	0.439	0.005
`Square`	1500	1.19	0.061

TABLE IV: Periodic function forecasting results.

As these results show, using a relative MAE % metric, we are able to achieve 98+% accuracy on each of the three curves.

The training plots are provided below, where the validation set is comprised of the last 20% of the data (excluding the test set). We take the *Mean Squared Error* (MSE) as our error metric. Although we see a very sharp initial error drops and subsequent diminishing returns, 1500 epochs are needed to push past 1% error on the Triangle wave and 2% error on the Square wave.

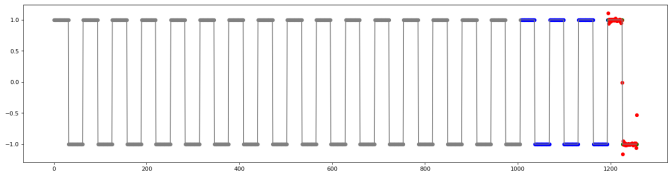

Fig. 11: Forecasting the Square wave with a LSTM network.

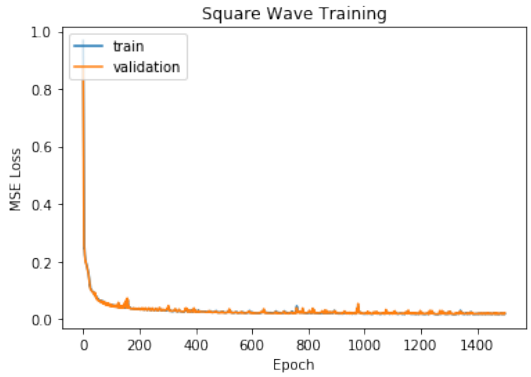

Fig. 12: Sine training plot.

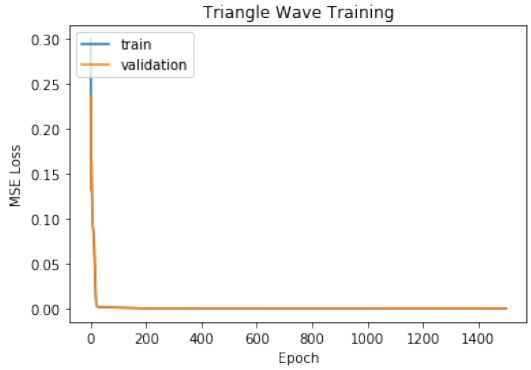

Fig. 13: Triangle training plot

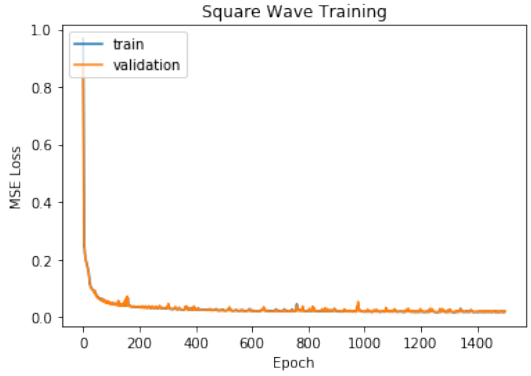

Fig. 14: Square training plot

VI. Learning Non-periodic Behavior

A. Data Formatting

Our real data is far more dense than our synthetic Sine, Triangle, and Square sets and so we subsample by 60 (take 1 point for every 60) so as to reduce the input dimension to our network. Additionally, the original data formatting phase no longer applies because cycle periods are no longer static and rather require more data points to capture as they grow. We address this by modifying this initial step to handle a variable number of input and output points.

We first perform a preprocessing step to calculate the period of each cycle based on it's peak and trough locations. We then use the same sliding window method that collects four cycles worth of points for each example. It's important to note that Keras LSTM networks must have a fixed size input and output, and so we store each example in a larger fixed-length array (a *"container"*). This length is preset and is always larger than an example worth of points, so as to not loose information. We represent *Null* values with -10. To help ensure that the network doesn't confuse these values with real data, we shift all data points until the smallest value is 0.

Upon the completion of this step, we have a Pandas data frame that effectively holds variable-length examples.

B. Network Architecture

Just as before, we use LSTM layer's to capture the time dependent nature of our data. However, we increase the complexity of our model as it now needs to learn substantially more complex behavior, including period growth and distortions to the curve's belly and back. We now apply two consecutive LSTM layers rather than one to create a more complex hierarchical feature representation of the input data. This is then feed into a dense hidden layer to extract one last feature vector to be fed into the output. The network's architecture and parameters are shown in **Figure 15** and are further summarized below:

1) Input Layer (*Px3* data points)
2) LSTM Layer (400 memory gate neurons)
3) LSTM Layer (256 memory gate neurons)
4) Dense Layer (128 neurons)
5) Output Layer (*P* predictions)

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_23 (LSTM)               (256, 381, 400)           643200
_________________________________________________________________
lstm_24 (LSTM)               (256, 256)                672768
_________________________________________________________________
dense_17 (Dense)             (256, 128)                32896
_________________________________________________________________
dense_18 (Dense)             (256, 127)                16383
=================================================================
Total params: 1,365,247
Trainable params: 1,365,247
Non-trainable params: 0
```

Fig. 15: Non-periodic LSTM architecture parameters.

Keras LSTM layers take in input and produce output in the form:(`batch_size`, `timesteps`, `input_dimension`) where `batch_size` is 256, `timesteps` is Px3, and `input_dimension` is 1 as our time series is of one variable.

C. Prediction & Results

In **Figure 16**, **Figure 17**, and **Figure 18** we show the entire training and test set (47 cycles) split into thirds stacked on

top of one another to best contrast the growth of each period over time. Just as before, we color code the plots as follows:

1) **Gray**: Training/Validation data
2) **Blue**: Test example input
3) **Green**: Test example label
4) **Red**: Test example forecast

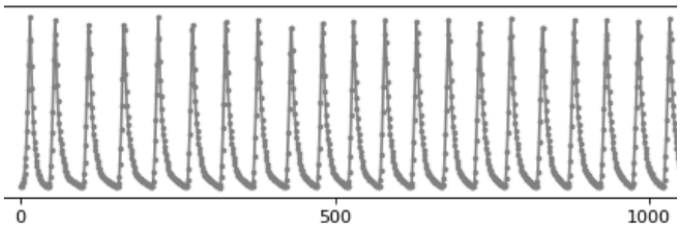

Fig. 16: First third of data set.

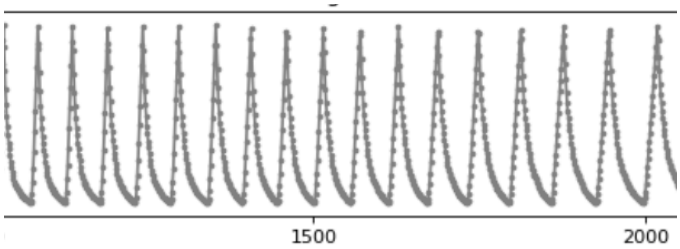

Fig. 17: Second third of data set.

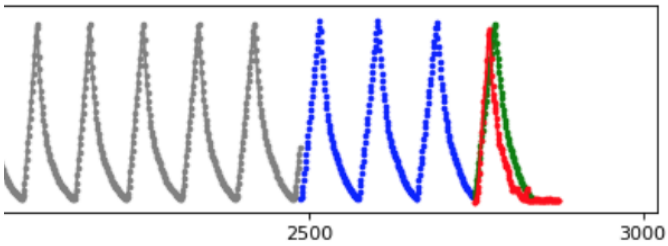

Fig. 18: Last part of data set and forecasts.

In **Figure 19** we plot the *MSE* network loss over epochs of training. It is clear that the network is appropriately tuned to avoid both under-fitting and over-fitting, as the training loss and validation loss are in near lock-step with one another.

Our best results running this experiment resulted in a Mean Absolute Error (MAE) percentage of *16.25*, and a Root Mean Square Error (RMSE) loss of *25.3*, and a predicted period of *72* while the true period was *88*. This still implies that the network learned to factor in period growth, as the initial training examples contained periods as low as *54*.

VII. Miscellaneous Approaches

A. Fourier analysis

Fourier analysis on cycle `Load` and `Time` behavior may reveal oscillatory components underlying the complex actuator `Load` characteristic. Applying a discrete Fourier transform (DFT) to raw cycle data results in a DFT magnitude

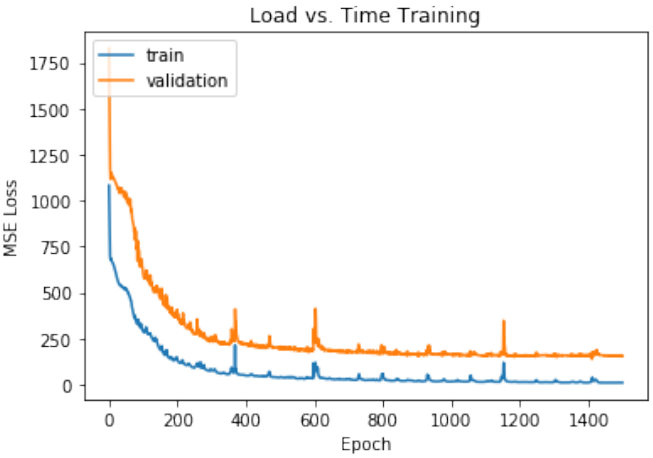

Fig. 19: Load vs. Time loss plot.

response with some easily-identifiable important frequencies, as seen in **Figure 20**.

While some frequencies clearly have stronger DFT magnitude than others, Fourier analysis does not effectively capture the varying `Period` phenomena that drives the `Load` characteristic of the actuator. Extrapolating the top-1000 frequencies from DFT analysis enables the 48^{th}-cycle prediction shown in **Figure 21**.

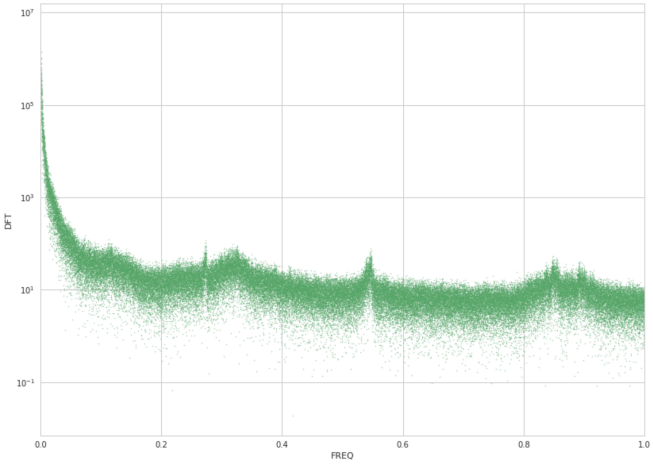

Fig. 20: Discrete Fourier transform (log-magnitude) of raw cycle data.

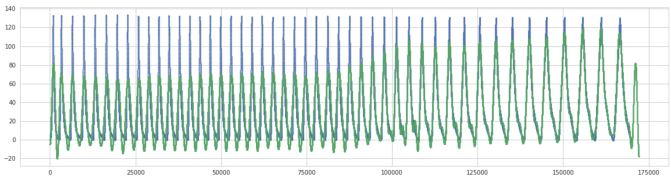

Fig. 21: Extrapolation of top-1000 frequencies from DFT analysis. Ground truth in *blue*; prediction in *green*.

B. *Time-series analysis*

Conventional time-series methods, including autoregressive integrated moving average (ARIMA) models tuned to take into account non-linear trends and non-constant periodicity, were attempted. These models performed poorly, but parts of our methodology in using them are included here for the benefit of future work.

Figure 22 depicts the autocorrelation behavior of the raw cycle signal, where *Lag* measures delay in units of cycle index. We observe little correlation between delayed windows of cycles, varying inversely with lag.

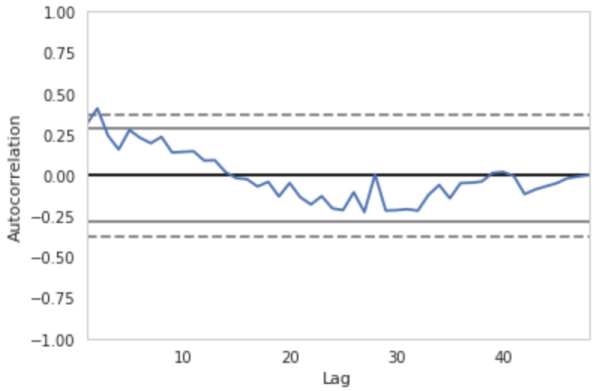

Fig. 22: (Serial) autocorrelation of cycles. Horizontal lines correspond to 95% and 99% confidence bands.

VIII. FUTURE WORK

Miriyev et al.[1] describe a closed-form approach to approximating the diffusion behavior of this actuator system. Future work may more accurately model the heating/cooling `Load` characteristic of the actuator using this additional feature.

ACKNOWLEDGMENT

We would like to acknowledge Aslan Miriyev (Ph.D.) and Professor Hod Lipson for their vision and expert mentorship over the course of this work, and for the providing the opportunity to work with unique and powerful data and hardware. We also thank Oscar Chang, Boyuan Chen, and Chad DeChant at the Creative Machines Lab, administered by Professor Lipson, for their patient and insightful guidance.

REFERENCES

[1] Miriyev, A., Trujillo, C., Caires, G., Lipson, H. (2018). Rejuvenation of soft materialactuator. MRS Communications, 1-6. doi:10.1557/mrc.2018.30