Final Project

Multi-step LSTM Autoencoder/Decoder for Multi-variate Time Series

Knapp, Stephen

CSCI E-89 Deep Learning, Spring 2020

Harvard University Extension School

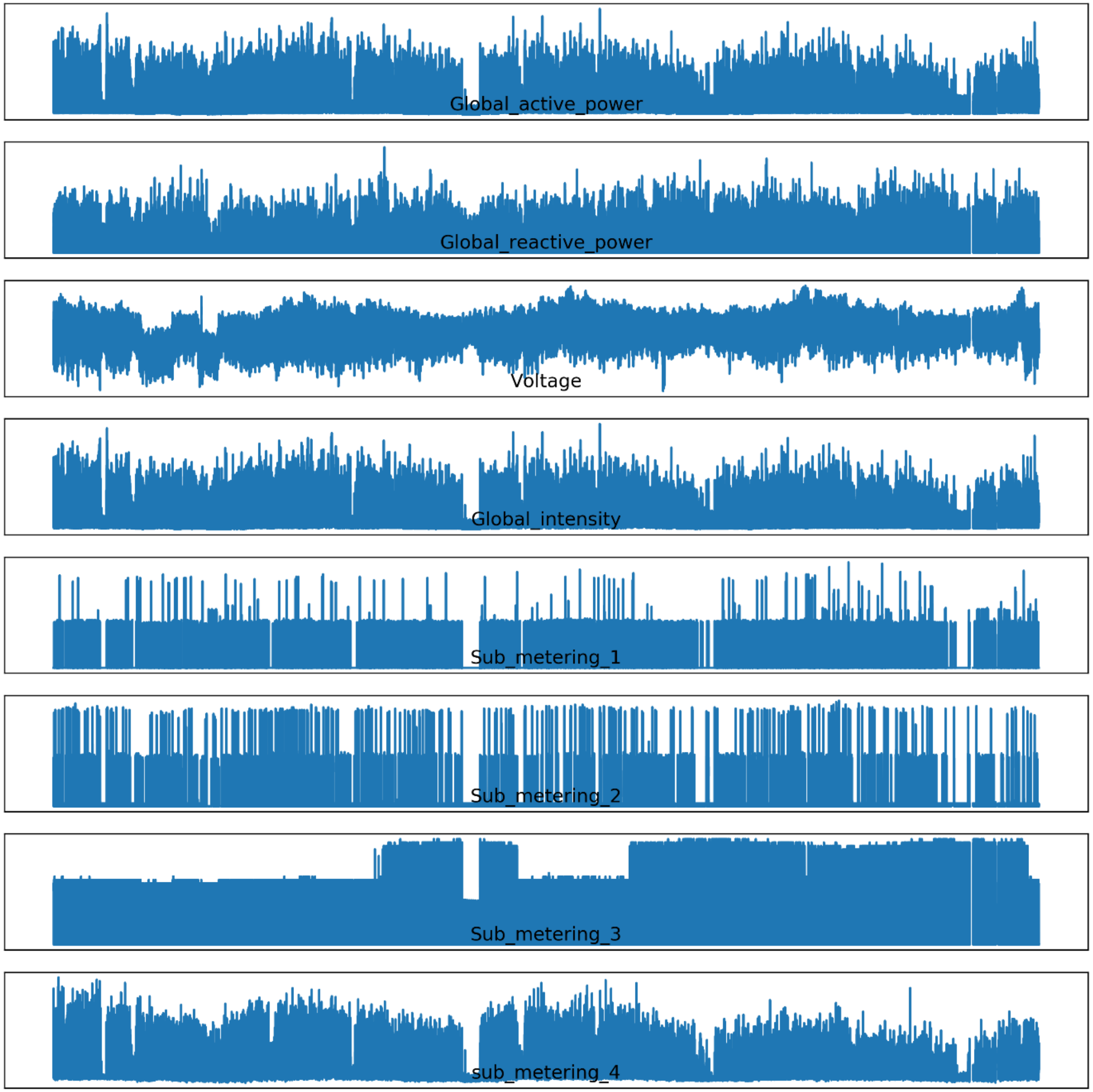
Prof. Zoran B. Djordjević

Introduction

- **Problem Statement:** Compress a multi-variate time series model into a lower dimensional space and tune the model to optimize stable and accurate predictions.
- This experiment uses a multi-step autoencoder/decoder LSTM network which receives a multi-variate time series input to predict future values of temporal slices (in this case temporal slides are weeks).
  - Keep it simple: the model will predict the next 46 weeks of energy consumption by day of the week based on the last four weeks of data from 8 time series inputs
- Identify the affect of different levels of tuning parameters:
  - # of epochs
  - # of neurons in the LSTM layers
  - # of dense layers in the decoder
  - Dropout
  - Kernel regularization
- Inspiration: Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python by Jason Brownlee; Section 20.8

Dataset: Household Power Consumption

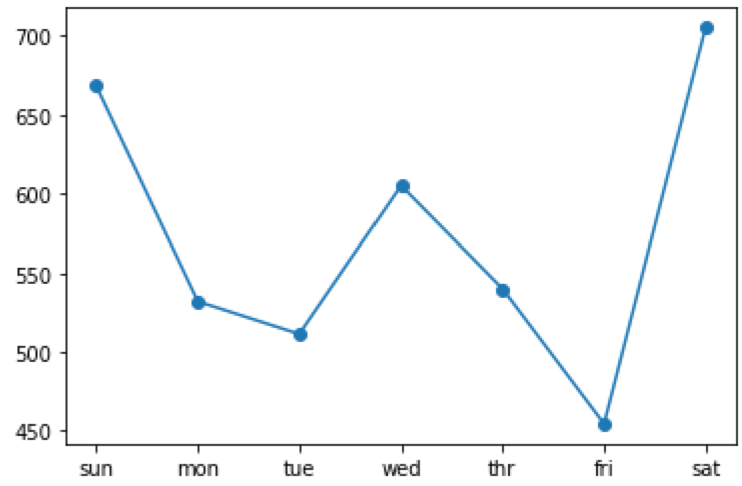
- Household Power Consumption: multi-variate time series that describes the electricity consumption for a single household for 4 years.
- Collection took place every minute, but this experiment looks at the daily and weekly aggregates.
- 7 variables originally, with the 8th calculated using the original 7
- Source: UCI machine learning repository
  - https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption



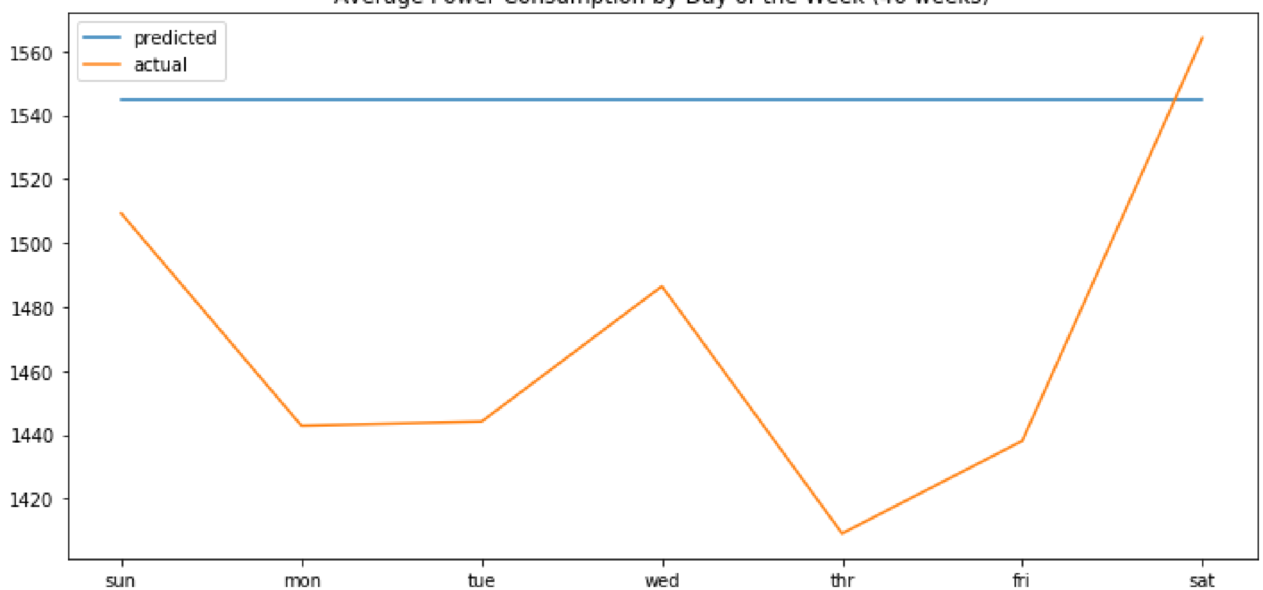
Baseline 1: Naïve Mode

Root Mean Square Error			
Min	**Max**	**Median**	**Range**
638.933	638.933	638.933	0

```
# daily persistence model
def daily_persistence(history):
	# get the data for the prior week
	last_week = history[-1]
	# get the total active power for the last day
	value = last_week[-1, 0]
	# prepare 7 day forecast
	forecast = [value for _ in range(7)]
	return forecast
```



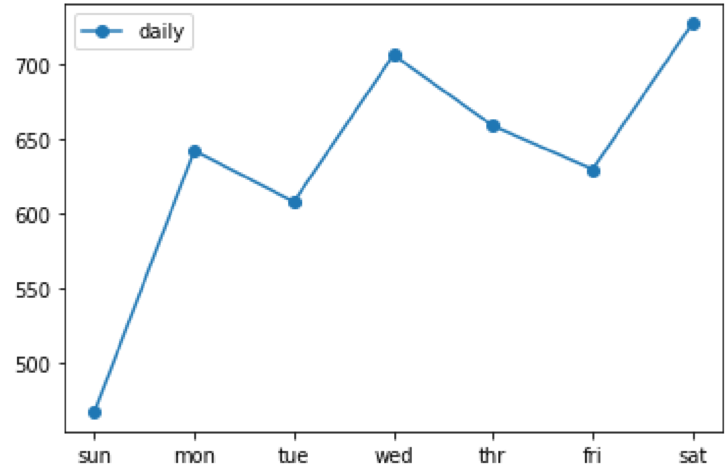
Average Power Consumption by Day of the Week (46 weeks)

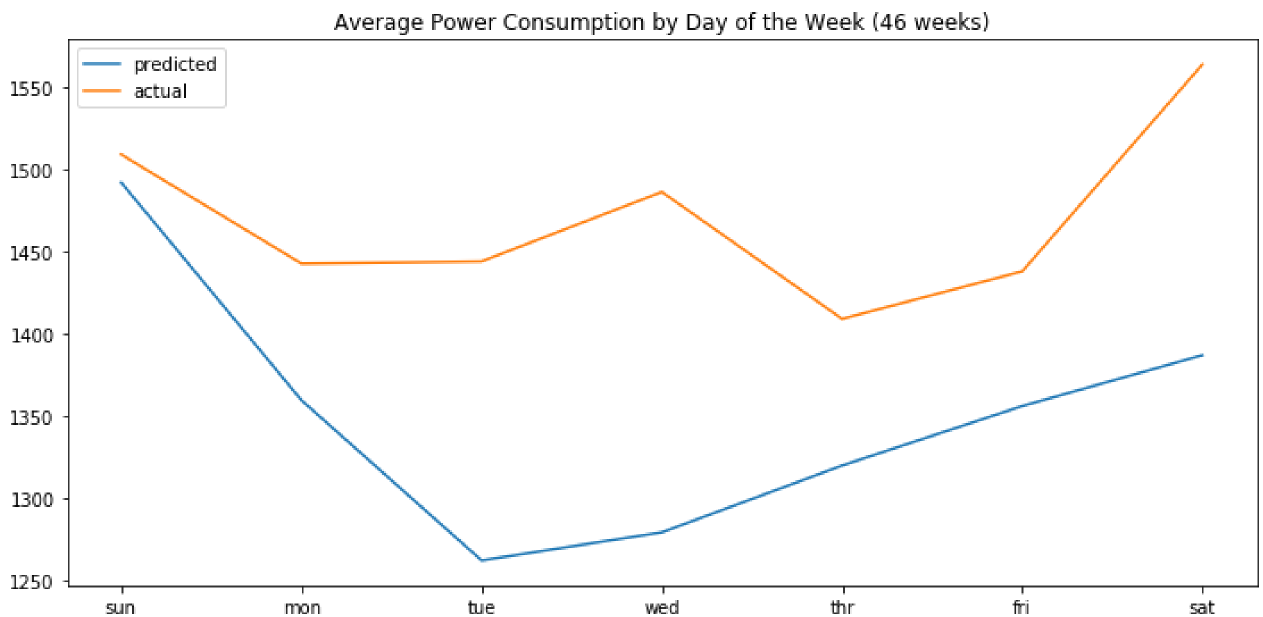


Baseline 2: Original encoder/decoder LSTM

Root Mean Square Error			
Min	**Max**	**Median**	**Range**
561.12	1223.676	598.41	662.556

```
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_timesteps, n_features)))
model.add(RepeatVector(n_outputs))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(100, activation='relu')))
model.add(TimeDistributed(Dense(1)))
model.compile(loss='mse', optimizer='adam')
```



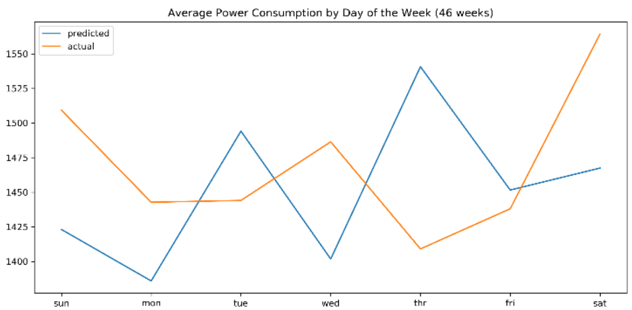


RepeatVector and TimeDistributed

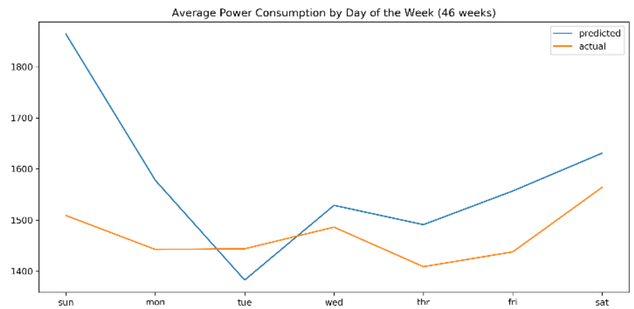
- These are both time series specific functions used with LSTM networks that work in tandem to decode the signal.
- RepeatVector
  - Connects the encoder and decoder layers.
  - Repeats the feature vector generated from the prior LSTM layer a specified number of times.
  - The number of times to repeat is simply the number of output values required.
  - In this case we need an output values for each of the 7 days of the week.
- TimeDistributed
  - Applies a layer (such as dense) to each temporal slice on an input.
  - The temporal slices in these experiment is weekly.

Optimal Number of Epochs

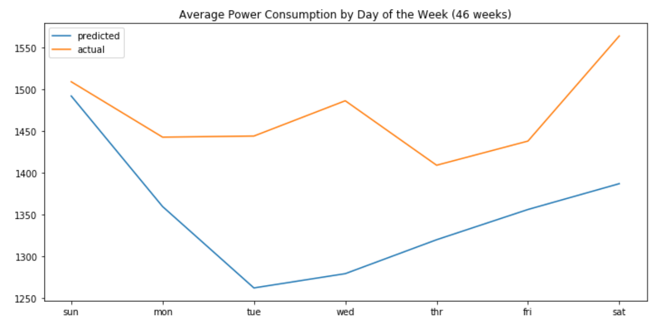
20 Epochs



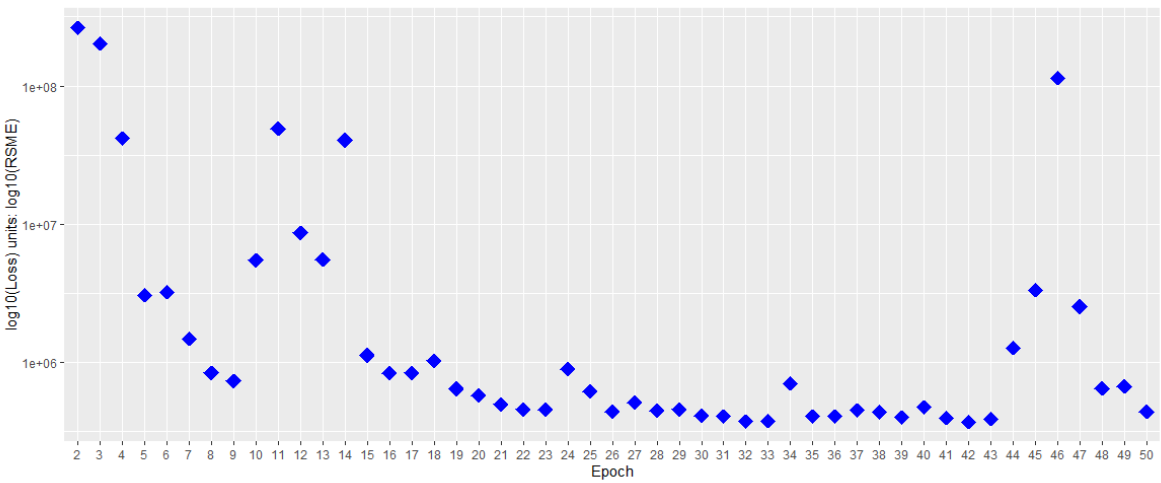
35 Epochs



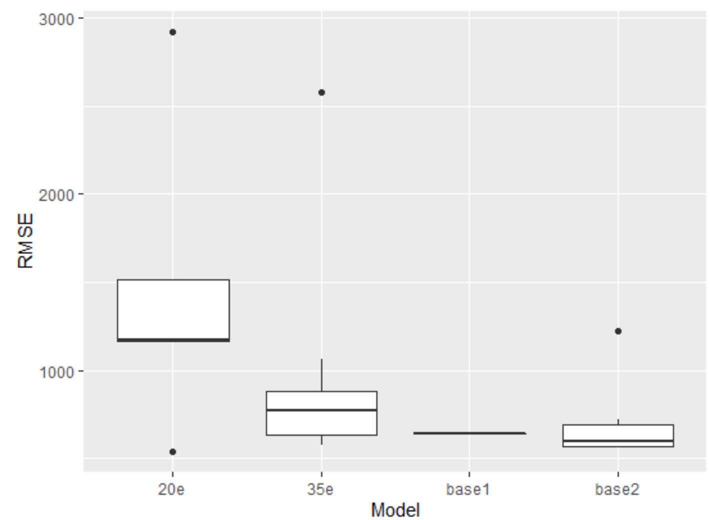
50 Epochs



Loss

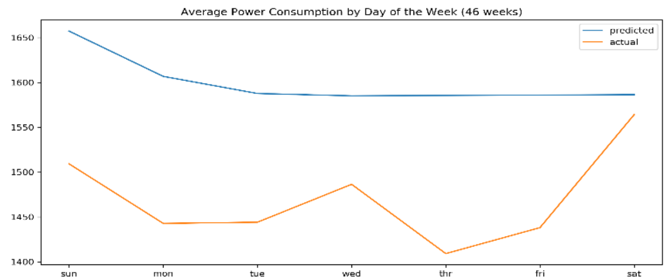


RSME

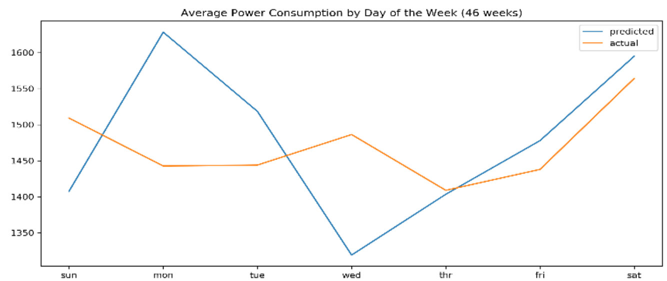


Tune Number of Layers and Layer Size

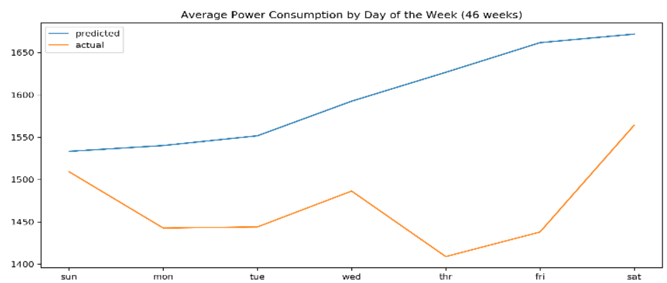
100 Neurons
2 Layer Decoder



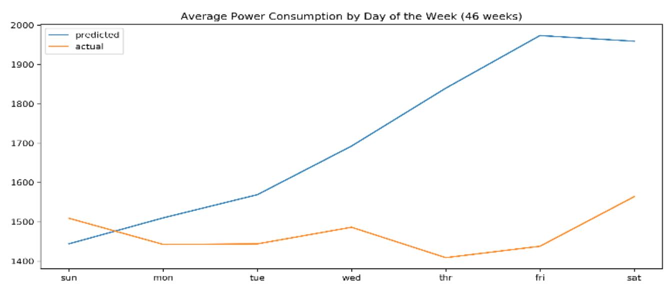
400 Neurons
3 Layer Decoder



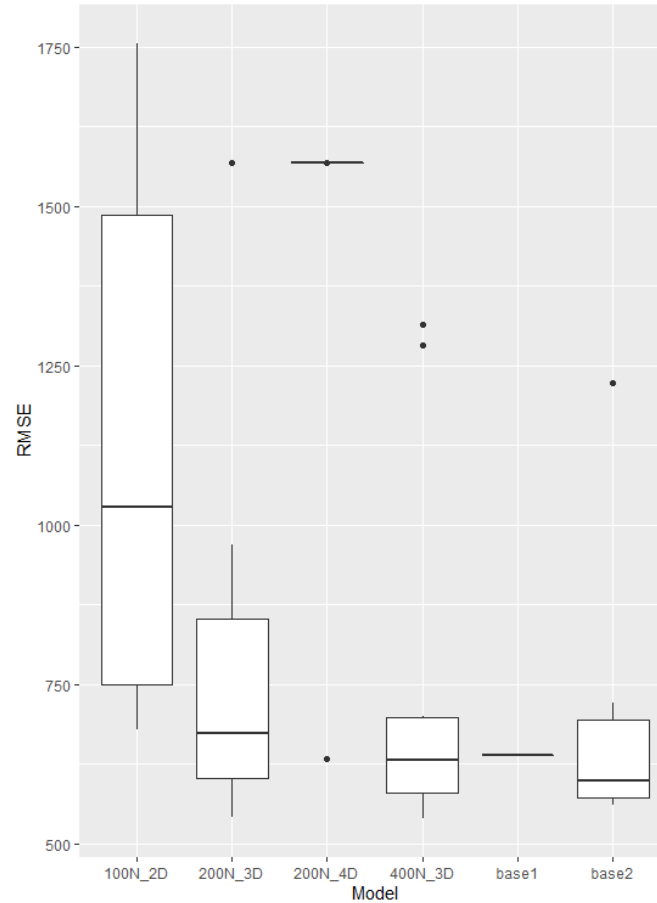
200 Neurons
3 Layer Decoder



200 Neurons
4 Layer Decoder



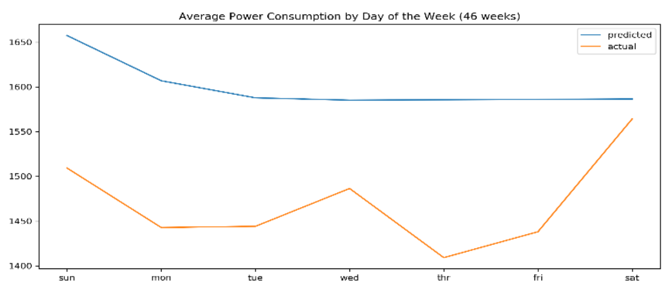

```
# define model
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_timesteps, n_features)))
model.add(RepeatVector(n_outputs))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(100, activation='relu')))
model.add(TimeDistributed(Dense(1)))
model.compile(loss='mse', optimizer='adam')
```



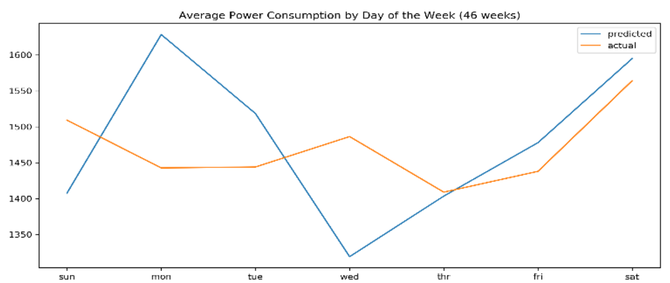
Test the Dropout Affect

```
# define model
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_timesteps, n_features), dropout=0.2))
model.add(RepeatVector(n_outputs))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(100, activation='relu')))
model.add(TimeDistributed(Dense(50, activation='relu')))
model.add(TimeDistributed(Dense(1)))
model.compile(loss='mse', optimizer='adam')
```

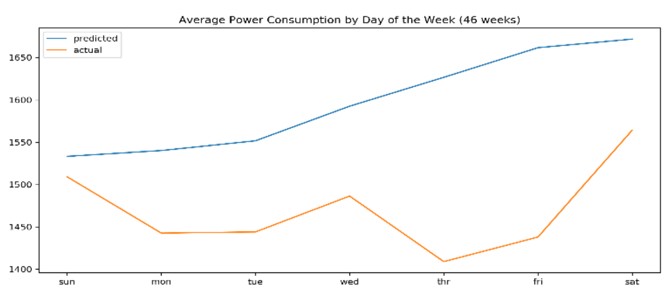
20% Dropout
35 Epochs

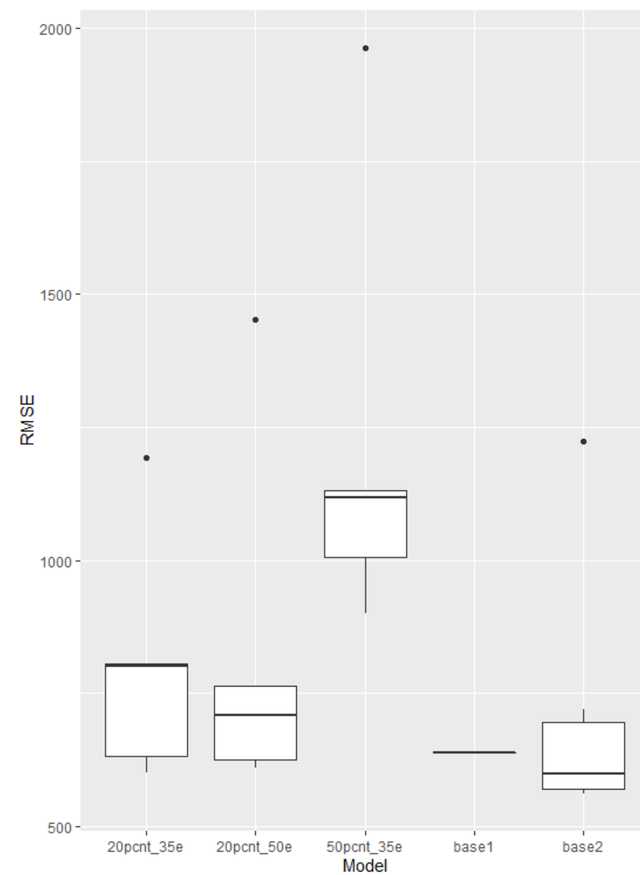


20% Dropout
50 Epochs



50% Dropout
35 Epochs

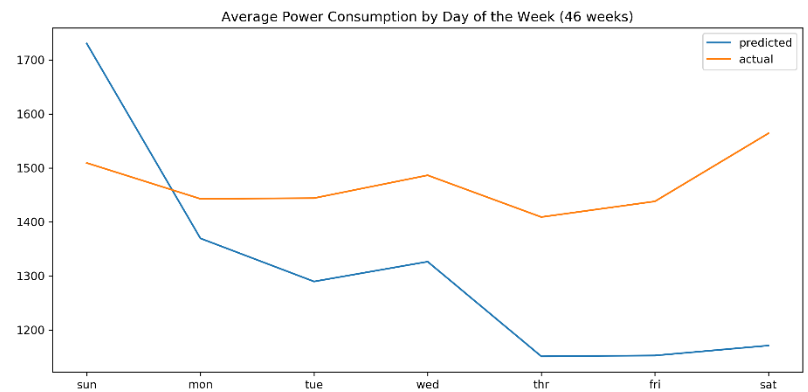




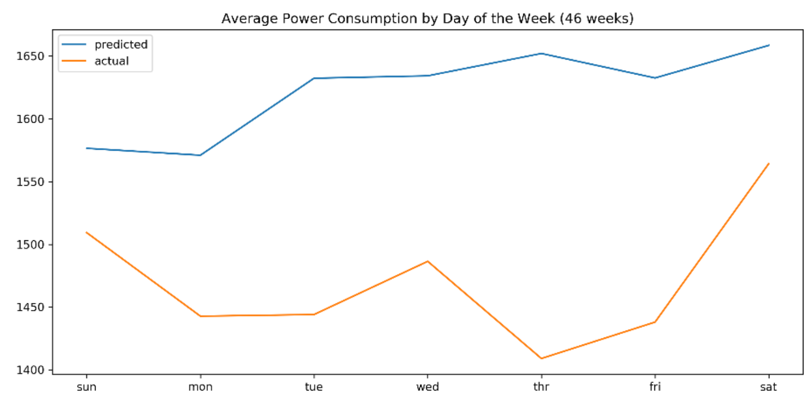
L1 and L2 Regularization (0.01)

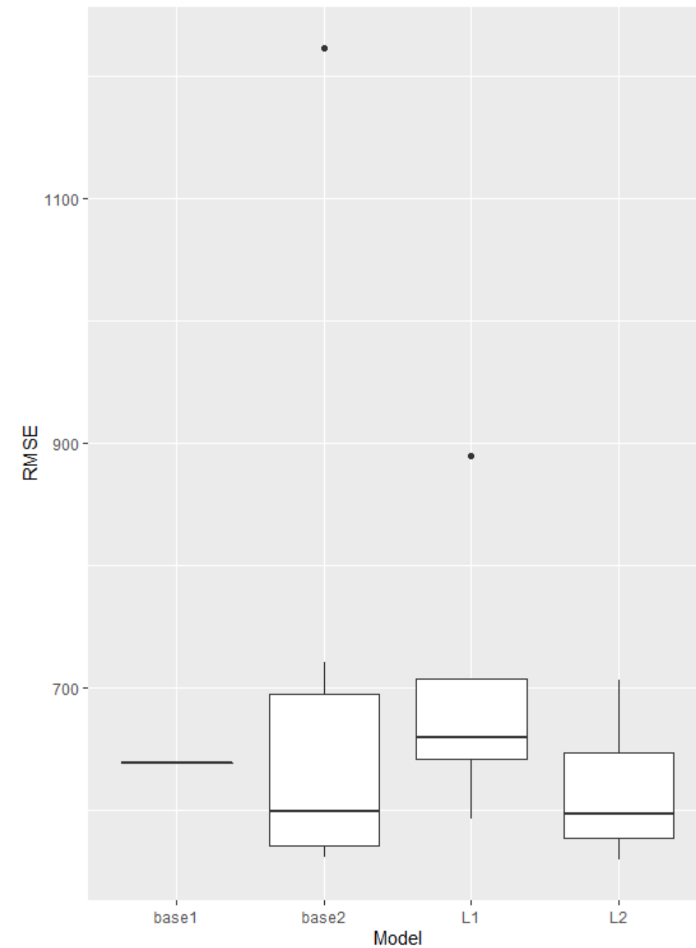
```python
# define model
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_timesteps, n_features), kernel_regularizer='l1'))
model.add(RepeatVector(n_outputs))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(100, activation='relu')))
model.add(TimeDistributed(Dense(50, activation='relu')))
model.add(TimeDistributed(Dense(1)))
model.compile(loss='mse', optimizer='adam')
```

L1

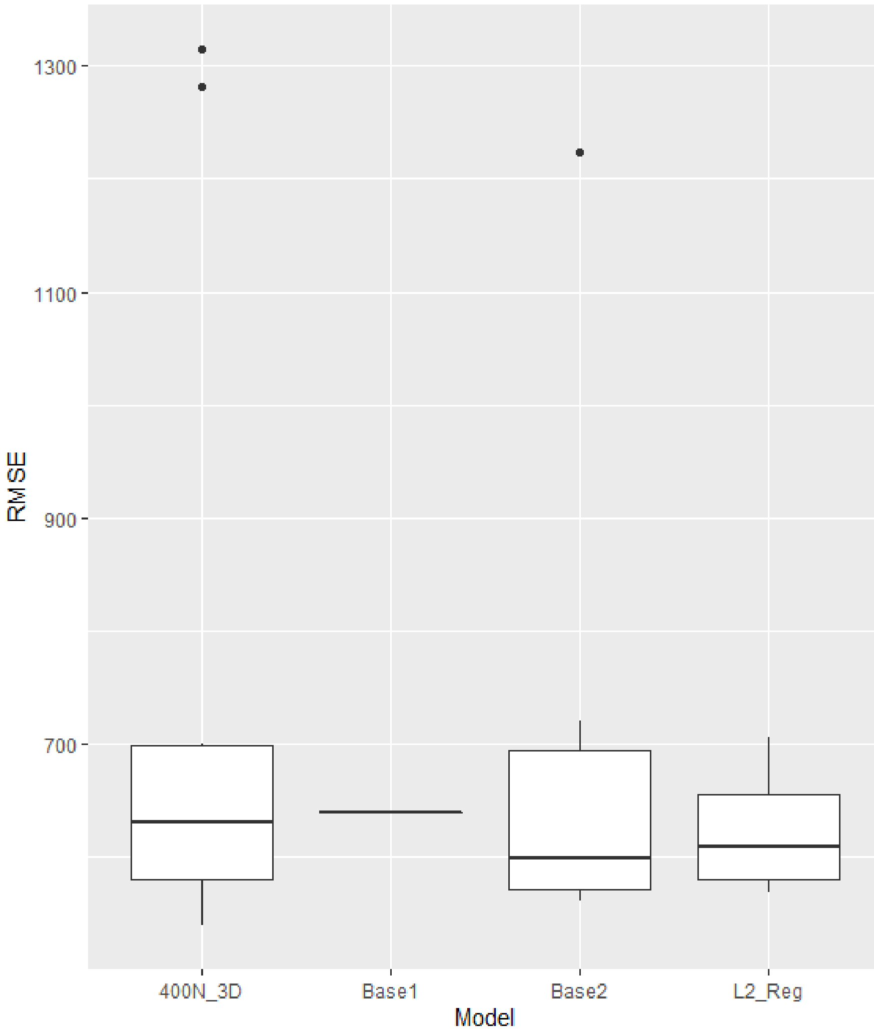


L2





Top Performers



- The model using L2 regularization (with a 3-layer decoder) is the overall best performer. Although the example model (base2) had a marginally lower median error value, the regularized model was more stable and had tighter quantiles.
- Only the example model (base2) and the three others show in the boxplot outperformed the naïve model.

Model	Min	Max	Median	Range
L2_Reg	559.196	706.191	596.64	146.995
Base2 (Example)	561.12	1223.676	598.41	662.556
400 Neuron/3 Layer Decoder	539.238	1315.565	630.249	776.327
Naïve	638.933	638.933	638.933	0

Going Forward

- Scale the model: input size is currently limited to about 4 weeks worth of data
- Reduce prediction of temporal slices to about 1/3 of the training data
- Optimize the L2 regularization value; currently using default value of 0.01
- Increase the number of run samples

Pain Points

- Unable to make the encoder/decoder symmetrical due to the RepeatVector layer
- Significant unused input data due to model's limited size.

YouTube URLs, Last Page

- Two minute (short): https://youtu.be/oFNoOs1J-30
- 15 minutes (long): https://youtu.be/FOwgHquJe2g