Evaluation of Deep Learning Architectures for Short Term Energy Forecasting

Christopher Castaldo

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

17 September 2020

Abstract:

Due to economic, political, and public pressures the traditional electricity grid model is undergoing large-scale restructuring. A key element in this restructuring is the integration of software to manage internet connected energy assets and to utilize data for pattern recognition in energy consumption behavior. One important mechanism to enable this transition is the development of forecasting models that can assist utilities in predicting geographical energy demand and renewable resource generation in real time. This has proven difficult using traditional machine learning models as the factors needed to predict demand are both dynamic and non-liner. Neural networks have been shown to overcome these shortcomings as they are able to automatically extract meaningful features form non-linear data. This is particularly useful in trying to forecast energy patterns during periods in which there is no historical precedent – such as the current Covid-19 pandemic or the recent wildfires in California and New South Wales. This report attempts to demonstrate the effectiveness of four types of neural networks: a multi-level perceptron, a long-short-term-memory recurrent network, a convolutional neural network, and a convolutional long-short-term-memory hybrid network. These models were tested using historical load, atmospheric, and temporal data to make three distinct forecasts – next hour energy demand for five North American cities, next hour renewable generation in three European countries, and next hour energy demand for three building lots in Manhattan. The discrepancy in locality is due to limitations in data access, however, this has no discernable impact on the underlying integrity of the models.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

Name: Chris Castaldo **Signature**: [illegible]

Acknowledgements

I would like to thank Dr. Michele Sevegnani for the immense help and patience throughout this project.

Table of Contents

Chapter 1 - Introduction:

This chapter gives a brief outline of the problem this project seeks to address as well as the projects motivation. Additionally, it will give a brief overview of the report structure and information about supplementary material.

1.1: Problem Outline and Motivation:

The United States electricity grid is a dynamic, complicated, quasi-governmental web of producers, purchasers, and consumers. However, it is not a self-correcting market and improper management can have disastrous spillover effects. This is demonstrably clear in California where a generous incentive market and governmental tunnel-vision has led to the proliferation of renewable assets [1]. This had a deflationary effect on the cost of electricity which lead to firm generation (natural gas and coal plants) being steadily replaced with intermittent generation (wind and solar) [1].

What was initially considered an environmental step-forward culminated into widespread disaster when an abnormal heat wave in August of 2020 lead to a drastic spike in energy consumption in. Compounded by a sudden drop in wind production due to abnormally clam conditions and the inability to import electricity from neighboring States whose grid was also under the strain of the heat wave, California experienced an electricity shortage and large-scale blackouts [2]. The California blackouts illustrate an often overlooked short-coming with renewable generation – the lack of dynamic scalability. Had the Californian government intervened and prevented the closure of their gas and coal plants, the reserved generational capacity could have been scaled up with minimal downtime and the crisis could have been averted [1]. Large-scale blackouts of this nature tend to have severe economic repercussions. For instance, the two-day August blackout in NYC was estimated to have caused nearly $7 billion in economic loss [3].

The costs of the California blackouts have yet to be quantified but given that the event occurred during the Covid-19 pandemic, they will likely be significant. In addition to faults in management and contingency planning, electrical grid disruptions have been occurring with greater and greater frequency due to increasingly unpredictable and destructive climate patterns. This has also been made demonstrably clear in California. The 2020 wildfire season has now become the largest in State history, officially surpassing the 2018 wildfire season which lead to the forced bankruptcy of the State's largest electric utility [4].

It has become increasingly evident that passivity towards grid decarbonization and resiliency building is no longer a viable option. This project proports that effective short-term forecasting is a necessary element in the path towards decarbonization and a more resilient electrical grid. The problems being experienced in California - and across the United States - stem from a litany of issues and effective short-term forecasting is by no means a cure all. However, it can play an integral part in the broader solution by allowing for effective real-time management of resources and smart grid infrastructure [1].

1.2: Project Objective:

The goal of short-term energy forecasting is to estimate the demand or generation of electricity for the next half hour up to the next two weeks [5]. Aggregated data, such as that on a city level, tends to be more consistent and easier to forecast while individualized data, such as that of a single residential household, tends to be more erratic and unpredictable [6]. Understanding the short-term consumption patterns on a large and granular level plays a significant role in the proper implementation of demand side response (DSR) programs. DSR is the dynamic shifting of energy consumption in real-time. For instance, if a spike in demand is predicted, utilities can signal to participants in the demand response program to scale back their consumption. Participation in DSR programs are generally incentivized by monthly retainer payments [1]. Effective short-term forecasting will help better inform DSR programs – hence lead to more efficient management of consumption behaviors and more informed decision making in regard to the integration of batteries, combined heat and power systems, photovoltaic panels, wind turbines, and other low carbon technologies [5].

Previous research on short-term energy forecasts has focused on single architectures dedicated towards forecasting single metrics. Given the dynamism of the energy pipeline, a responsive grid will need to process large amounts of data form various sources to make large-scale and granular forecasts while also being able to respond to changes in intermittent generation from wind and solar assets. This project attempts to fill that gap by creating a single platform in which various architectures can be compared for various forecasts. In this project four deep neural networks

(DNN) will be trained on temporal and meteorological data – these networks include a multi-layer perceptron (MLP) network, a long-short-term-memory (LSTM) network, a convolutional neural network (CNN), and a CNN-LSTM hybrid network. The results of these predictions will be displayed on a Graphical User interface (GUI) and connected to a PostgreSQL database. It should be noted that access to relevant data posed fairly significant limitations in this research which will be outlined and discussed in the subsequent chapters.

1.3: Project Structure and Supplementary Resources:

This report will be structured as follows: the second chapter will give a broad overview of the past literature and common methods used for short term energy forecasting and a more in depth overviewed of neural networks. The third chapter will discuss the data gathering and processing phase. The fourth chapter will discuss the design and implementation of the software. The fifth chapter will discuss the evaluation, testing, and results of the networks. And, lastly, the sixth chapter will discuss the limitations, lessons learned, and areas for future research.

The code for this project can be found at https://github.com/Castaldo/Energy-Time-Series-Forecasting

Chapter 2 – Literature Survey and Neural Network Overview:

Algorithms such as Linear Regression, Autoregressive Integrated Moving Average, Support Vector Machines, Neural Networks, and various hybrid models have all been proposed as methods for short-term energy forecasting. This section will provide a brief overview of those algorithms as well as a more in-depth overview of Artificial Neural Networks (ANN'S) - particularly on the ANN's used in this report.

2.1 – Linear Regression

Linear Regression is a popular algorithm due its simplicity and ease of implementation. Linear regression assumes a linear relationship between the predictors and the desired output variable. A multiple linear regression (MLR) model with **k** predictor variables can defined as:

[7]

$$\check{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \ldots + \beta_k x_k + \in$$

Where $\check{y}$ denotes the desired output, β denotes the regression coefficient, x denotes the **i-th** regression parameter, and $\in$ denotes the random error [7]. In most cases, MLR seeks to minimize the residual sum of squares (RSS) [7]. The RSS measures the differential between each prediction output and the actual fit of the line and is defined as:

[7]

$$RSS = \sum_{i=1}^{n} (y_i - \beta_0 + \beta_1 x_1)^2$$

In 1989, Mograhm and Rahman used MLR to predict next hour energy forecast with the predicators, dew point, temperature, and wind speed. The error of their predictions ranged from 0.20% to 10.02% [8]. Similarly, Kumar *et al.* achieved a mean average percentage error (MAPE) of 7.12% using MLR with similar atmospheric variables [9]. MLR, however, is limited in it capacity to generalize temporal values and discover non-linear correlations.

2.2 - Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) is a type of linear regression algorithm that takes into consideration the temporal aspects of the dataset. If past observations have a correlative significance to future observations, then ARIMA models can help capture those dependencies [5]. The ARIMA formula is a combination of two regressive formulas – autoregression and moving average regression. Autoregression can be denoted as:

[11]

$$y_t = c + a_1 y_{t-1} + \cdots + a_p y_{t-p} + u_t$$

Where a_p are the AR parameters, c is a constant, and u_t is the error term. Likewise, the moving average regression can be denoted as:

[11]

$$y_t = \mu + u_t + m_1 u_{t-1} + \cdots + m_p u_{t-p}$$

Where m_p are the MA parameters, u_t is the error term, and μ is the constant. Therefore, the full equation can be denoted as:

[11]

$$y_t = c + a_1 y_{t-1} + \cdots + a_p y_{t-p} + a_t + m_1 u_{t-1} + \cdots + m_q u_{t-q}$$

Where c is the intercept, and p and q denote the autoregressive and moving average terms.

Cao et al. proposed a hybrid ARIMA model to forecasts intraday load demand based on meteorological conditions and historic load data achieving a MAPE of 3.5% [10]. Similarly, Al Musaylh et al. used an ARIMA model to predict next hour demand in Queensland Australia, achieving a MAPE of 9.35% [12].

2.3 – Support Vector Machines

A Support Vector Machine is a type of supervised learning model that can map non-linear input data onto a hyperdimensional plane [14]. Broadly defined, an SVR formula can be expressed as:

[13]

$$y = w\psi(x) + b$$

Where $\psi(x)$ represents the non-linear mapping into input space $\mathbf{x}$ [14]. The details of SVM's are beyond the scope of this thesis but they can be thought of as hyperdimensional linear regression models that learn iteratively form their input sequences. Zhang proposed an SVM model based on meteorological conditions and historic load data for day ahead forecast, achieving a MAPE of 4.1% [13]. Similarly, Khan et al. trained an SVM model using historic load data, temporal data, and atmospheric data to predict next hour load demand in Kanpur India achieving a MAPE of just 1.90% [15].

2.4 - Neural Networks:

Due to the non-linearity and non-stationary nature of data relevant to energy forecasting, neural networks have gained more and more attention in recent research. Artificial Neural Networks are loosely inspired by the biological processes of the brain. In an ANN, nodes are connected to other nodes through weighted links - in the same way biological neurons are connected by a synapsis [16]. Nodes in the network take a real number as an input, process it through an activation function, multiply it by an associated weight, and pass the result to the neurons in the subsequent layer. The sum of the inputs is run through the same process, layer by layer, until an output layer is reached [16].

The weights $(\boldsymbol{W_m})$ associated with each connection are a measure of the correlative significance each input has to the output. For instance, previous hour energy demand will have a higher correlation to next hour energy demand than the direction the wind is blowing. Given those two inputs, a network would eventually 'learn' to assign more weight to the previous hour input than the direction of the wind input- thus giving it more of a determining influence on the networks final output. The network accomplishes this through a mathematical formula known as backpropagation

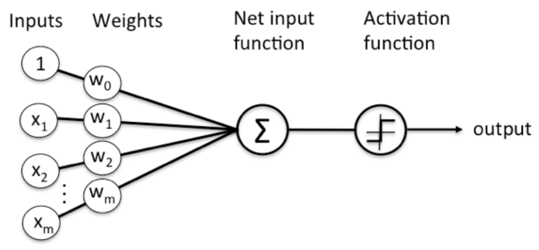


Figure 2.1: Simple Neural Network [1]

2.4.1 - Backpropagation:

Backpropagation utilizes gradient descent to map the correlative value of an input to an output in an ANN. Initially, weight values are chosen at random, the gradient then works backwards form the first output adjusting the weights of each network layer [16]. This requires the calculation of an **error function** *E (X, θ)* which measures the error between the desired output y_i and the predicted output $\hat{y}_i$. *X* represents the sum of input pairs (x_i, y_i) and *θ* represents a particular parameter value. The value of the gradient is calculated using the error function *E (X, θ)* with respect to weight value w_{ij}^k, bias b_i^k, and learning rate *a* [17]. The equation can be formally written as:

[18]

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial \mathrm{E}(\mathrm{X}, \theta^t)}{\partial \theta^t}$$

Where ∂ is the partial derivative. The error functions used in this project are the Mean Squared Error and the Mean Average Percent Error which are denoted as:

[19]

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \mathrm{MAPE} = \frac{100}{n}\sum\left|\frac{y - \hat{y}_i}{y}\right|$$

Where y_i is the target value for input pair (x_i, y_i) and $\hat{y}_i$ is the predicted output of the network. The error function is represented by the sum of the individual errors for each input-output pair in the network [17]. Therefore, using an MSE error function, we can interpret the function as:

[18]

$$\frac{\partial \mathrm{E}(\mathrm{X}, \theta)}{\partial w_{ij}^k} = \frac{1}{n}\sum_{d=1}^{n}\left(\frac{1}{2}(y_i - \hat{y}_i)^2\right)$$

The partial derivative of the error function with respect to each weight can be denoted as:

[18]

$$\frac{\partial \mathrm{E}}{\partial w_{ij}^k} = \delta_j^k o_j^{k-1}$$

Where δ_j^k is the partial derivative of the error function over the partial derivative of the activation function and o_j^{k-1} is the partial derivative of the sum of the weights multiplied by their respected output [17]. Those values are denoted as:

[18]

$$\delta_j^k = \frac{\partial \mathrm{E}}{\partial a_j^k} \qquad\qquad o_j^{k-1} = \frac{\partial}{\partial w_{ij}^k}\left(\sum_{l=0}^{r_{k-1}} w_{ij}^k \cdot o_j^{k-1}\right)$$

Through this formula a neural network works backwards to continuously update the weight values associated with each input output pair. The network will increase or decrease the value of each weight based on the results of the backpropagation algorithm with each iteration minimizing the error function [17]. The algorithm will take small 'steps', which can be visualized in figure 2.2, to try to find the global minima in a multi-dimensional plane. The size of the step is defined by the learning rate *a*. If set too high, the network will begin over correcting, resulting in a runaway gradient. If set to low, the network may get stuck in a local minimum which can be visualized in the bottom right corner of figure 2.2 [20]. Since the dimensional space that neural networks operate in are non-convex, random initializations of weight can lead to different outcomes in model accuracy [16].

Under the assumption that the inputs of the network have a correlative association with the desired output, backpropagation enables the network to adjust the weights of each input output pair to find relationships in the dataset. This is particularly helpful in data with a temporal element or data that isn't linearly separable.

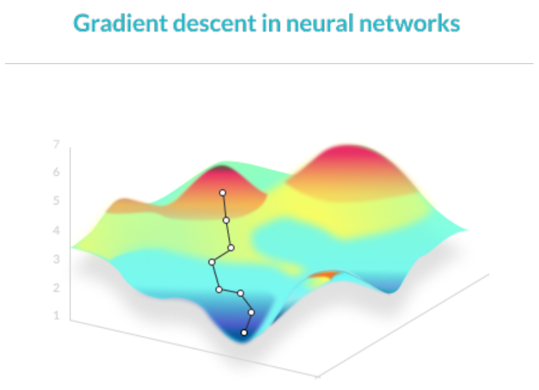


Figure 2.2: Stochastic Plain [2]

2.4.2 - Activation Functions:

The activation function is a function that squeezes the sum of each neurons input value into a defined range. There are many types of activation functions, but we will limit the explanation to the two used in this report - the **tanh** function and the **ReLU** function. The purpose of an activation function is to introduce non-linearity into the output of a neuron [21]. As stated previously, neural networks are very successful in making approximations for data that cannot fit neatly into a linear boundary.

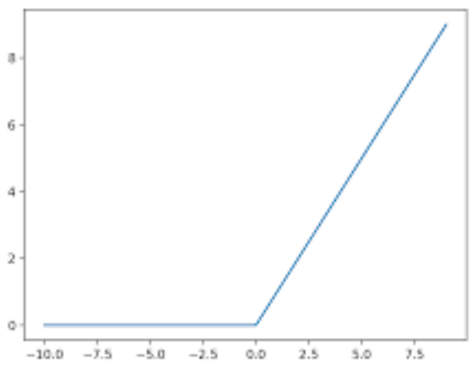


Figure 2.3: ReLU Activation [3]

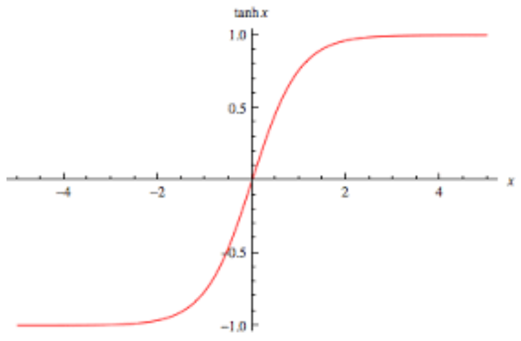


Figure 2.4: tanh Activation [3]

This capability is enabled by activation functions. Refer to the two plots above to see how each activation function transforms the input data. The ReLU function has a range of (0, ∞). If the neurons sum is negative the neuron is set to zero and no longer has influence over the network. If the sum is positive the neuron is unchanged. Due to this, ReLU results in a lighter and less expensive network with fewer neurons being activated at once [21]. This makes it less expensive to run, however it comes with a drawback. Any value that falls in the negative range will become disconnected from the network. This can help reduce over-fitting, but it can also limit the networks capacity to find more abstract correlations in the dataset. Alternatively, the tanh function has a value range of [-1, 1]. Tanh is computationally more expensive but is better at identifying abstract representations. However, it suffers from a similar problem as the ReLU function known as the vanishing gradient problem. If a value is placed far enough down the left side of the y-axis, the derivatives will continue to shrink after each iteration until their value flatlines. This results in increasingly less influential neurons that may eventual lose any correlative power in the network [21].

2.5- Multi-Layer Perceptron:

A Multi-layer Perceptron (MLP) is the most straightforward type of artificial neural networks - often colloquially referred to as 'vanilla' networks. An MLP consists of at least three layers of nodes – an input layer, a hidden layer, and an output layer. Through backpropagation, MLP networks are able to distinguish patterns in non-linear data.

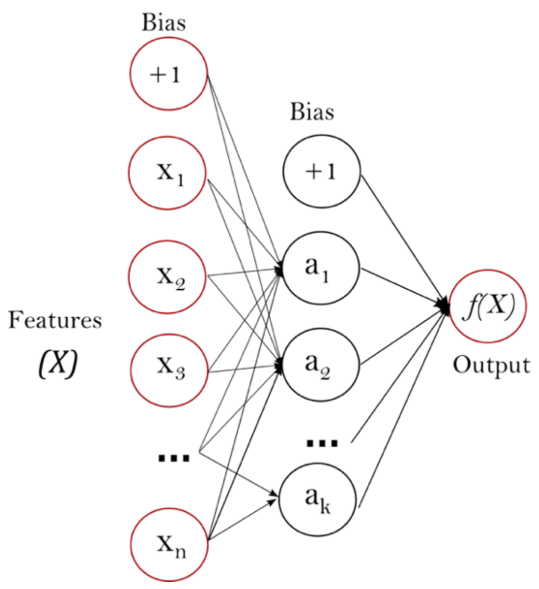


Figure 2.5: Simple Multi-Layer Perceptron [22]

Figure 2.5 demonstrate a simple single-layer MLP where X_i represents an input, a_i represents an activation function and $f(x)$ represents the output. An input layer (where the nodes correspond to the number of columns in the dataset) passes data to **n-** nodes in the first hidden layer. The sum is then put through an activation function which squishes the value into a defined range and determines whether or not the neuron activates. The new value of each activated neuron is then passed to **n-neurons** in the **n-th** hidden layer until the output layer is reached. The backpropagation formula works backwards through the network, updating each weight in a direction relative to the formulas output in order to minimize the gradient [23]. Eventually the network will recognize the correlative significance of each connection in each neuron and assigned it an appropriate weight to best predict the output.

Rodrigues et al. used an MLP network to forecast energy usage of residential household in Portugal achieving a MAPE of 4.12% [24]. Ruy et al. achieved a MAPE of 2.19% utilizing an MLP to forecast energy consumption of large industrial buildings [25]. Lastly, Dalto et al. used an MLP network, pretrained with a Stacked Auto-Regressor, to predict Croatian wind generation achieving an average error rate of just 0.55% [26].

2.6 - Recurrent Neural Networks (LSTM):

Recurrent Neural Networks (RNN) are able to capture relationships between a series of inputs (i.e. the outputs of previous inputs influence the outputs of the subsequent inputs). This capability makes RNN's particularly well-suited for data where the inputs and outputs are interdependent, such as temporal data [27].

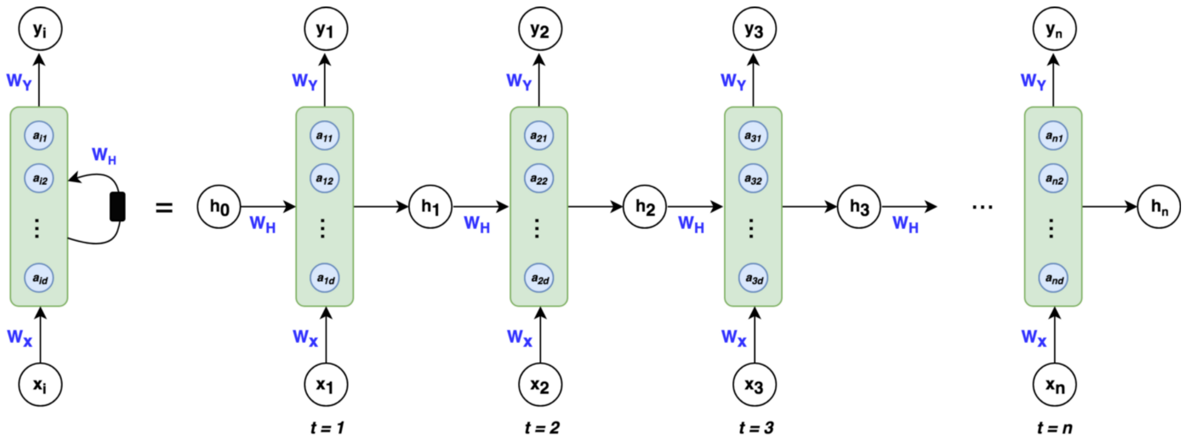


Figure 2.6: Unfolded RNN [4]

The diagram above displays an "**unfolded RNN". ** The green blocks represent the "**hidden states**" and the blue circles represent the "**hidden nodes**". Vector $\boldsymbol{h_i}$ represents the output after the activation function is applied to each node in the internal state. This output gets passed to the layer in the second state denoted as t [27]. In doing so, this architecture is able to take into account what happened at **t-1** by including **h** from the previous layer and applying it to the next input series. The weights are unchanging and are only updated after a pre-defined amount of 'steps' are taken [27]. The output at each hidden node is defined by the equation:

[28]

$$a_t = W_H h_{t-1} + W_X X_t$$

RNN's suffer from the vanishing gradient problem discussed early, where increasingly small gradients fail to update weights and earlier network layers stop contributing to the learning process. This prevents the networks from remembering information in longer sequences [29]. The solution to this issue is a type of neural network known as a Long-Short-Term-Memory Networks (LSTM).

LSTM networks have mechanisms known as gates that regulate the flow of information within the network. In an LSTM network, each node within the RNN's hidden layer is replaced with an LSTM cell shown in figure 2.7. An LSMT cell consists of three gates – an input gate, a forget gate, and an output gate – each transforming the data in a different way [30]. The first gate is the forget gate which looks at the information coming from h_{t-1} and X_t and runs it through a sigmoid function.

This creates a weighted value of the data's importance ranging from 0 (forget completely) to 1 (remember completely) [30]. The function can be written as:

[28]

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

The next step is two part. First the data is put through an input function which runs it through an additional sigmoid function. Then, separately, the data is passed through a tanh function which generates a new value denoted as $\tilde{C}_t$ [30]. These functions are defined as:

[28]

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

$$\tilde{C}_t = tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)$$

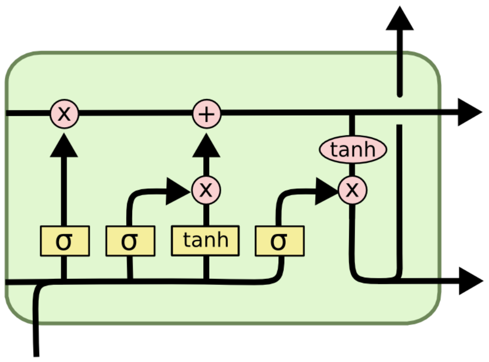


Figure 2.7: LSTM Cell [5]

The next step is to update the old cell state. This function is defined as:

[28]

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Lastly, the output layer is calculated. In this layer the input data is run through a sigmoid function and the cell state is run through another tanh function. These functions are defined as:

[28]

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

$$h_t = o_t * \tanh\left(C_t\right)$$

These values are then multiplied together, and the value represents the output of the cell [30]. Through the combination of these functions, LSMT networks are able to 'remember' an arbitrary amount of temporal or highly structured data allowing them to perform exceptionally well on time-series and textual inputs.

Muzaffar and Afshari used a Long-Short-Term-Memory (LSTM) model to forecast load demand using weather and historic load data, achieving a MAPE of 1.5% [31].

2.7 - Convolutional Neural Networks:

Convolutional neural networks (CNN's) are a type of regularized MLP and are typically used to classify image data but have proven to be successful with temporal data when implemented with a one-dimensional architecture [32]. In this type of network, a kernel of length **n** and width **k** (the width is determined by the number of variables in the data) will move across the dataset performing elementwise convolutions. The elementwise value of each product are summed together, and the result is run through an activation function [32]. This results in a tensor of values known as the convolutional layer. In the next step another sliding window approach is used which will take the max values of every **n-th** piece of data, reducing the output tensor further. Lastly the data is flattened, reducing its depth to one dimension and used as input to a fully connected MLP network with a pre-defined dropout rate (dropout determines what percentage of nodes are randomly dropped form the final tensor) [23]. This method drastically reduces the amount of data being handled by the network and is significantly quicker and more successful at preventing over-fitting than a non-regularized MLP.

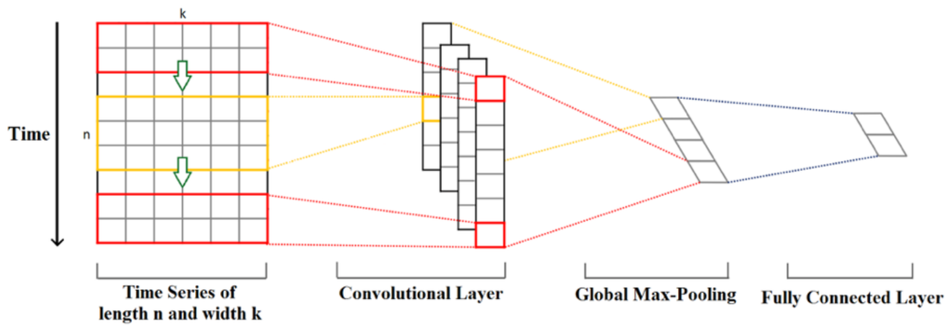


Figure 2.8: 1D Convolutional Neural Network [6]

Chapter 3 – Data Gathering and Processing:

This section will provide a brief background on the data used for each of the neural network models, including where it was sourced from and how it was processed and transformed.

3.1 – City Data

The dataset used for predicting aggregated next hour energy demand for the cities sampled in this project consisted of hourly weather data and hourly energy load data. The weather data was collected from the Modern-Era Retrospective Analysis for Research and Applications (MERRA) [33] database and the energy data was collected form each cities Independent Systems Operator [34][35][36][37]. The dataset used to train each model contained five years of hourly data. In the instance of San Francisco, estimates had to be made since the ISO that provides San Francisco electricity also provides electricity to over half of California. To account for this, the yearly energy usage for San Francisco (gathered from the California Energy Commission [38]) was taken as a percentage of the yearly usage for all regions in the ISO. This ratio came out to roughly .055%. That ratio was then mapped over the ISO's hourly dataset. For instance, if the hourly use for **date x** on **hour y** was 10,000 MwH, the assumption follows that San Francisco consumed approximately 550 MwH. The training dataset is demonstrated below:

City Model Features (Non-Weather)	
Date	MM/DD/YYY
Energy	Hourly Energy Usage in MwH
Previous Hour	Previous Hour Energy Use in MwH
Hour	Hour in 24-Hour Format
Day	Day of the Week (0 - 6)
Month	Month of the year (1 - 12)
Holiday	Holiday (Binary)
Season	Season (1 - 4)
Weekend	Weekend or Weekday (Binary)

City Models Features Map (Weather)	
Temperature	Temperature at Hour **x** (F)
Relative Humidity	Measure of Relative Humidity (%)
Short-wave irradiation	Measure of radient energy within defined wavelength
Snowfall	Measure of hourly snowfall (MM)
Snow Depth	Measure of snow depth (MM)
Rainfall	Measure of hourly rainfall (MM)
Wind direction	Direction of wind (0-360°)
Wind speed	Measure of Wind Speed (MPH)

Based on the geography of each city, the weather variables played different correlative roles in hourly energy demand. Therefore, the weather variables used in training each model varied based on locality. For instance, features such as '**snow depth'** had a much higher correlative significance for energy consumption in Vancouver than in New York City. All features that had an absolute correlative influence of more than 5% were chosen as features for each model. Additionally, the variables '**Holiday'**, '**Season'** and '**Weekend'** were added to the dataset to help identify additional non-linear and complex consumption patterns. The effects that weekend and season variables have in electricity consumption is demonstrated below.

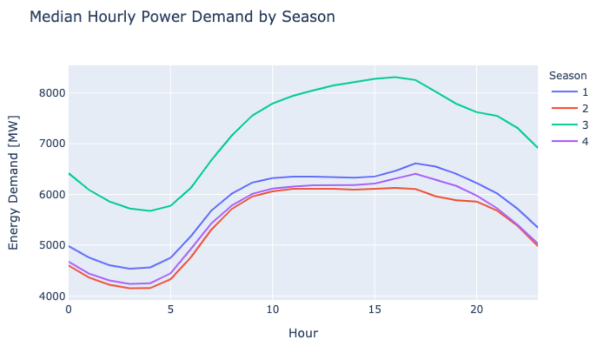


Figure 3.2: Seasonal Effect on Energy Consumption

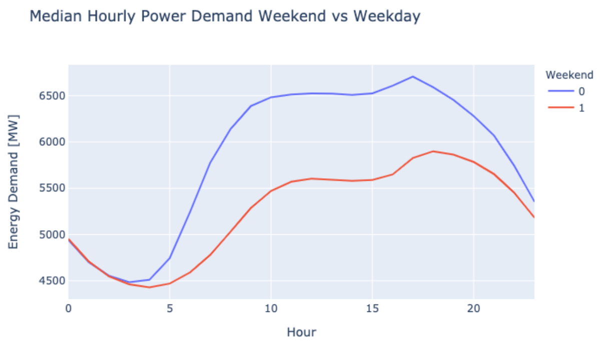
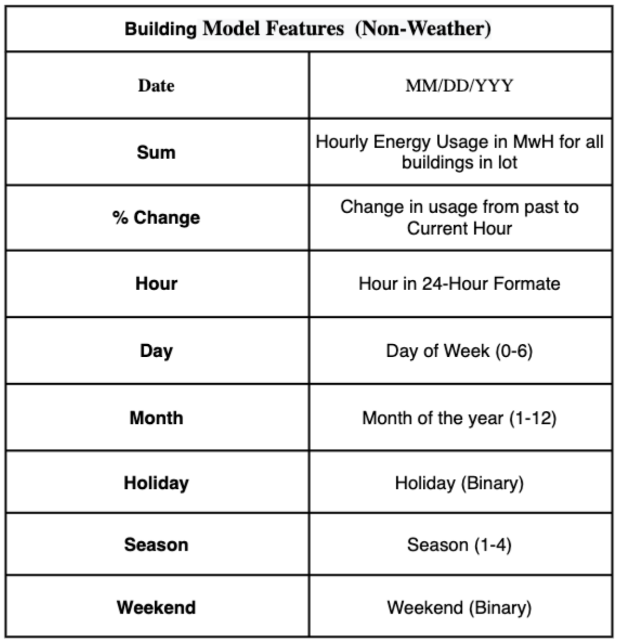


Figure 3.3: Weekend Effect on Energy Consumption

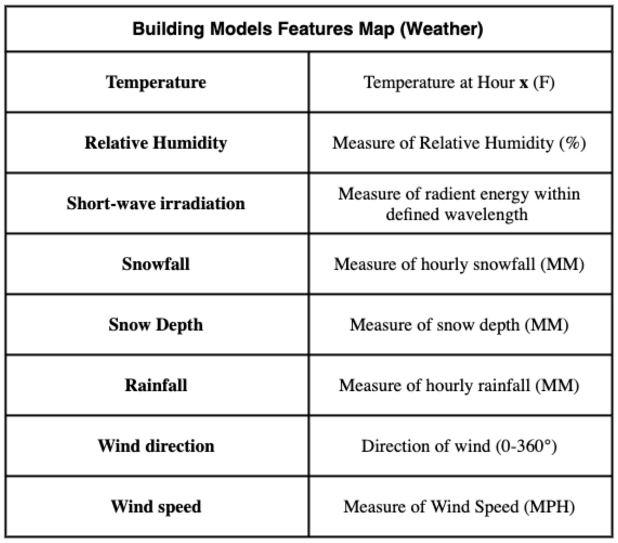
3.2 – Building Data

The dataset used for predicting next hour building usage consisted of hourly weather data and yearly energy data. The weather data was also collected from the MERRA database [33] and the energy data was collected form a public repository maintained by New York City [34]. The dataset contained one year of hourly data. Unfortunately, only yearly energy usage was provided for each building in the dataset. In order to estimate the hourly usage some assumptive transformations had to be made. In order to arrive at an estimate, the yearly consumption of each building was taken as a ratio of the yearly consumption of Manhattan as a whole. That ratio was then applied to the hourly consumption of Manhattan and used as a proxy of that building's hourly consumption. This is the same approach that was taken early with San Francisco's consumption. Consequently, this inhibited the model's ability to discover unique trends in each building's energy consumption profile due to the consumption pattern of each building being mapped over the consumption pattern of the whole city.

Just as in the city dataset, the variables '**Holiday'**, '**Season'** and '**Weekend'** were added to identify additional non-linear and complex consumption pattern. The training dataset is demonstrated below:



Building Model Features (Non-Weather)	
Date	MM/DD/YYY
Sum	Hourly Energy Usage in MwH for all buildings in lot
% Change	Change in usage from past to Current Hour
Hour	Hour in 24-Hour Formate
Day	Day of Week (0-6)
Month	Month of the year (1-12)
Holiday	Holiday (Binary)
Season	Season (1-4)
Weekend	Weekend (Binary)



Building Models Features Map (Weather)	
Temperature	Temperature at Hour **x** (F)
Relative Humidity	Measure of Relative Humidity (%)
Short-wave irradiation	Measure of radient energy within defined wavelength
Snowfall	Measure of hourly snowfall (MM)
Snow Depth	Measure of snow depth (MM)
Rainfall	Measure of hourly rainfall (MM)
Wind direction	Direction of wind (0-360°)
Wind speed	Measure of Wind Speed (MPH)

3.3 – Renewables

The dataset used for predicting next hour renewable generation consisted of hourly weather data and hourly generation data from three European Countries. The weather data was collected from the MERRA database [33] as well as a weather database maintained by the European Union [39]. The energy data was collected form a separate public database, also maintained by the European Union [39]. The dataset contained one year of hourly data. Unfortunately, the EU does not provide information about the geographic concentration of solar and wind assets throughout each member country. Consequently, it was impossible to completely map weather conditions with renewable output as the weather will vary in different geographic areas. To accommodate for this, weather data was gathered from various regions in each country and averaged as a mean. The mean outcome of each variable was then used to map weather conditions with renewable output. The regional areas in which the weather data was gathered are shown below. No additional transformations were made to the dataset and '**season'** was the only additional feature added. The training dataset is demonstrated below:

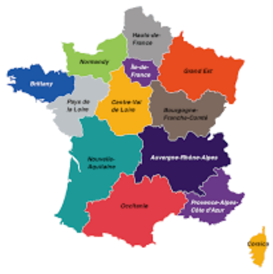


Figure 3.5: French Regions [7]

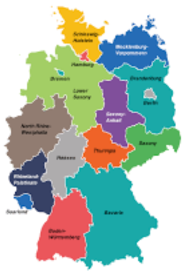


Figure 3.6: German Regions [7]

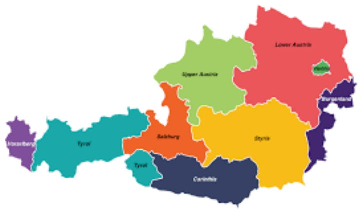


Figure 3.7: Austrian Regions [7]

Renewable Model Features (Non-Weather)	
Date	MM/DD/YYY
Solar	Hourly Solar Generation
Wind	Hourly Wind Generation
Previous Solar	Previous Hour Solar Generation
Previous Wind	Previous Hour Wind Generation
Month	Month of the year (1 - 12)
Hour	Hour of the day (24-H)

Renewable Models Features Map (Weather)	
Temperature	Temperature at Hour **x** (F)
Relative Humidity	Measure of Relative Humidity (%)
Short-wave irradiation	Measure of radient energy within defined wavelength
Snowfall	Measure of hourly snowfall (MM)
Snow Depth	Measure of snow depth (MM)
Rainfall	Measure of hourly rainfall (MM)
Wind direction	Direction of wind (0-360°)
Wind speed	Measure of Wind Speed (MPH)
Wind Velocity (10M)	Wind Speed 10 Meters Up
Atmospheric Horizontal Radiation	Measure of Atmospheric Solar irradiance
Atmospheric Ground Radiation	Measure of Solar irradiance at Ground Level

Chapter 4 – Requirements, Design, and Implementation:

This chapter will discuss the system architecture and as well as the user requirements. Consultation for system requirements was provided by Christopher Ruppel, Senior Vice President of energy development firm DCO Energy [41]. Mr. Ruppel's feedback were considered regularly during each iteration. Gathering and developing the requirements was based on an iterative approach in line with agile software methodology.

4.1 – System Architecture

The architecture can be viewed as consisting of two independent parts.

The first part handles the end-to-end neural network pipeline. Raw data is fed into the pre-processing modules which delete any anomalies, drops null values, concatenates the datasets, and adds relevant features. The processed dataset is then saved locally and passed to the model processing script which scales the values, selects the relevant features, and splits the data into a training, validation, and testing set using a 70/15/15 ratio. The scaled dataset is then passed to the model script which defines each architectures hyperparameters and runs the data through the network. If the network successfully meets the convergence metric, it is saved as JSON and written to a database for future analysis. MAPE/RMSE and loss curves are also created and saved which are network diagnostic tools that will be discussed in the subsequent chapter.

The second part handles the GUI which was built using a Dash and Flask framework. When launched, the Dash front end calls the loss, MAPE/RMSE, and graph scripts. The loss and MAPE/RMSE scripts retrieve the saved plots for each architecture and display them on the app's dashboard. The graph script loads a saved model based on user input and uses input test dataset as into the network. The network then makes its forecasts on the test data and plots the predicted values against the actual test set values. The user can dynamically cycle between each dataset for each metric and each model architecture. Figure 4.1 demonstrates the topology of the software's architecture.

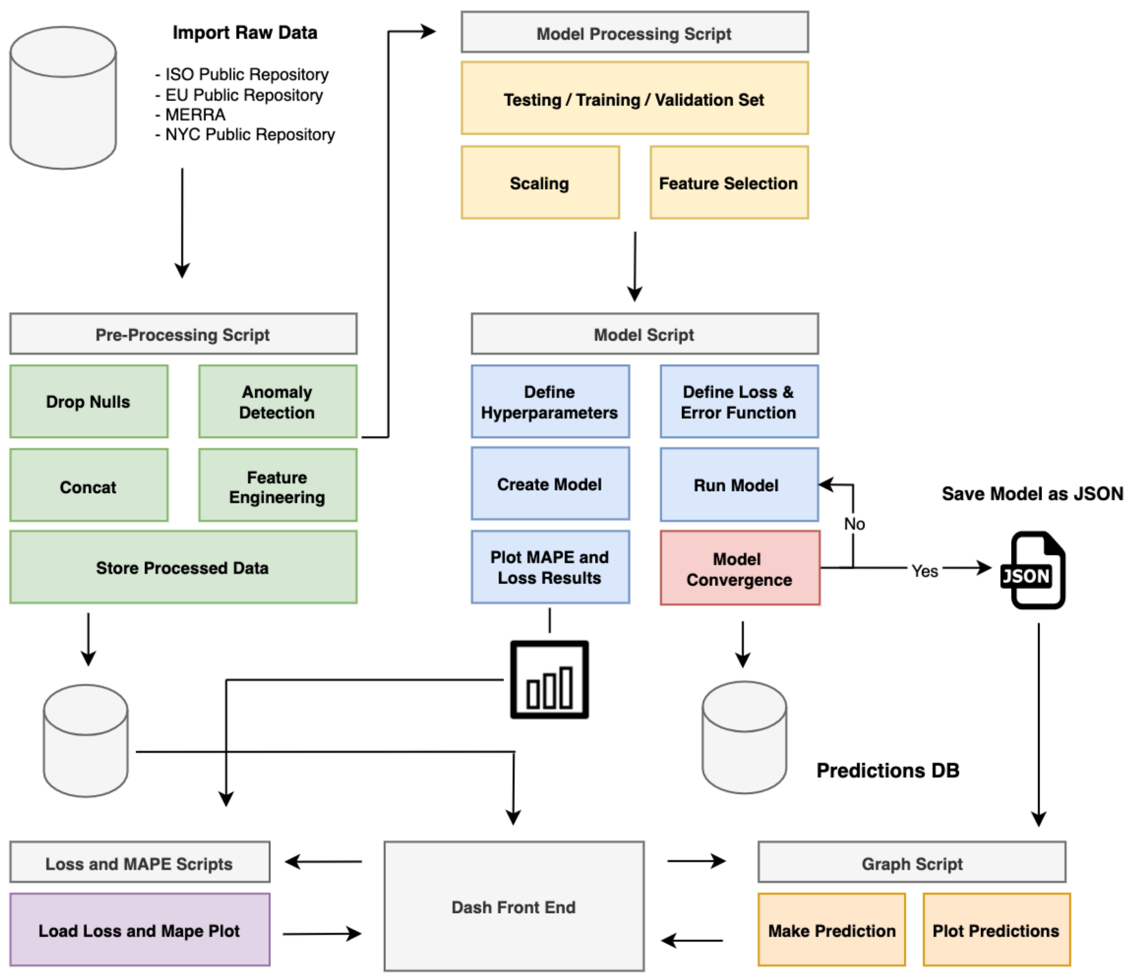


Figure 4.1: Software Architecture

4.2 – Development Processes

1. **Sprint 1:** Relevant time series data is gathered and processed

1.1. Must Haves

- Relevant time series data is collected form public repositories
- Processing module drops all null values
- Processing automatically detects and drops all anomalies within the dataset
- Processing module conducts the necessary transforms to approximate data

- Processing module successfully concatenates each dataset together
- Processing module successfully adds new features to each processed dataset

1.2 Could Haves

- Web crawler that automatically fetches data from relevant repositories

2. **Sprint 2**: Neural Network Design

2.1. Must Haves

- Naïve module for baseline forecasts estimations
- Multiple neural networks architectures that successfully analyze the input data
- Module that checks for a convergence metric and saves the model as JSON files if the metric is met. The module re-runs the network if the metric is not met.
- Module that saves the loss and MAPE/RMSE curves

2.2. Could Haves

- Hybrid networks
- XGBoost module that automates feature selection

3. **Sprint 3**: Dash Application

3.1. Must Haves

- Application displays a graph that charts the model predictions against the actual values for test set
- Application displays the MAPE/RMSE and loss curves for each model in the dataset
- Users can cycle between the predictions for each model architecture
- User can cycle between the datasets for each prediction

3.2. Could Haves

- Users can cycle between various test datasets
- Application displays various visualizations for each dataset

4. **Sprint 4**: Database and Deployment

4.1. Must Haves

- All model predictions are written to a PostgreSQL database

4.2. Should Have

- Users are able to make calls to the database from the GUI
- Containerization and Deployment on Docker

4.3 – Requirements Summary

Summarizing the requirements, the user should be able to cycle through each tested architecture and visualize the results. The visualizations should include a plot of the predicted values vs. the actual values for each architecture, a plot demonstrating the loss curve, and a plot demonstrating the MAPE/RMSE curve. Additionally, the user should be able to cycle through each dataset tested for each metric (i.e. New York City, Vancouver, San Diego, etc.). Lastly, the user should be able to choose from a number of test sets to fully evaluate the model's ability to approximate new data. The results of each model should be written to a PostgreSQL database. The final product should be compatible with Docker deployment.

In regard to data pre-processing, ideally the process would be fully automated, however there were limitations. Mainly, there is no federal standard in the way public databases are maintained. If every ISO was required to use the same naming conventions and database structure, then this process could theoretically be automated. However due to the discrepancies in formatting, each model required an individual script to process it.

4.4 – GUI

The image below shows the GUI in the dashboard for the models predicting next hour building consumption. Displayed is the result from CNN-LSTM network for Lot 835 in Manhattan, which includes the Empire State Building, The Magellan, and the Herald Towers. As shown, the user can choose between the four model architectures and between each building lot tested. Additionally, the MAPE/RMSE and Loss plots are displayed to help visuals the performance of each model.

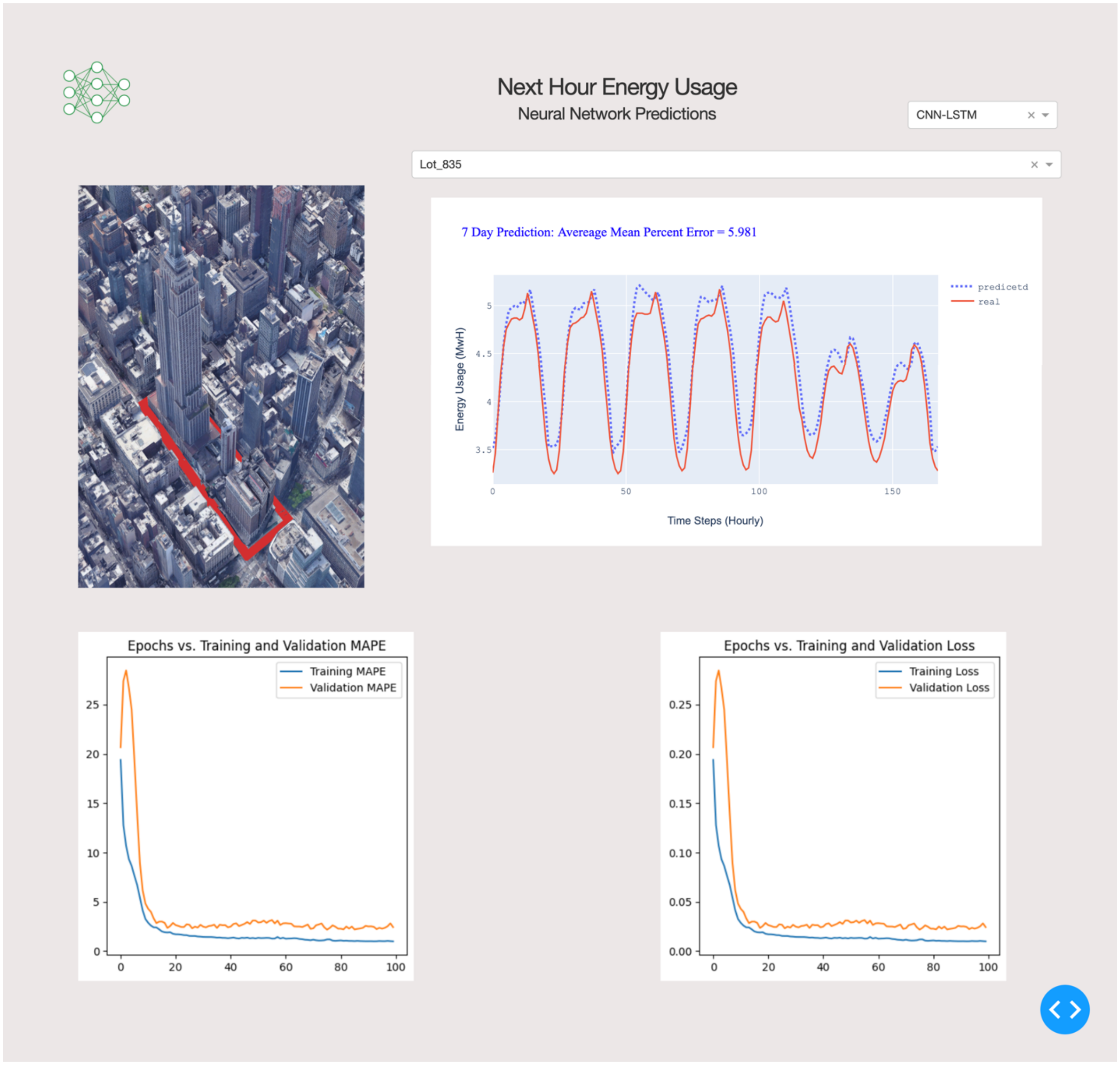

Next Hour Energy Usage
Neural Network Predictions
CNN-LSTM
Lot_835
7 Day Prediction: Avereage Mean Percent Error = 5.981
Energy Usage (MwH)
Time Steps (Hourly)
predicetd
real
Epochs vs. Training and Validation MAPE
Training MAPE
Validation MAPE
Epochs vs. Training and Validation Loss
Training Loss
Validation Loss

Figure 4.2: GUI

Chapter 5 – Model Results:

5.1 Evaluation and Testing:

Evaluating neural networks poses a few challenges. Most notably is that neural networks are black box systems (i.e. the parts of the program environment that are transforming the data are inaccessible and untestable [40]). Additionally, artificial neural networks are stochastic. When a network is initialized the initial weight values are randomly determined and the data is randomly shuffled [21]. This has a notable influence on the movement of the gradient and the model will produce different results each time it is ran - for instance, the gradient may get stuck in a local minima as shown in figure 2.2. Multiple measures were taken to ensure each model's robustness was tested as best as possible.

In order to deal with the stochastic nature of the networks each model was tested using walk-forward cross-validation. In walk-forward cross-validation the training and test data is split into **n folds** with each iteration being split further along the time sequence. There are numerous advantages to this approach. First, seasonal bias's will be eliminated in the test data. By testing at various points in the time sequence, we can measure the model's predictive merit during different periods of the year. Secondly, it prevents overgeneralization of the training data. By doing multiple tests in different parts the sequence we eliminate the likelihood that the test results are influenced by the model over-fitting the training data [21]. Lastly, it provides different results with different random initializations of the weights which allows for the measurement of the standard deviation of the error rate.

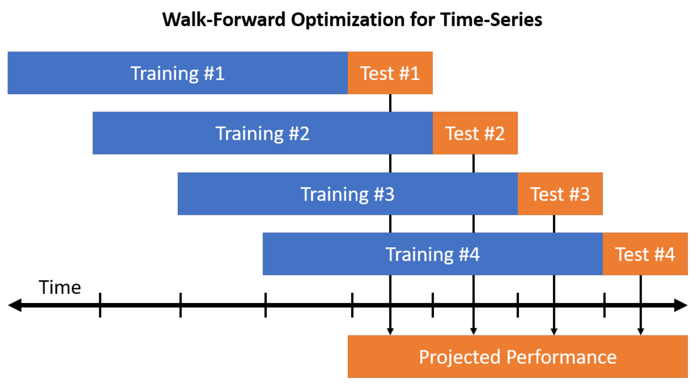


Figure 5.1: Walk-Forward Optimization [8]

Each model in this network was split into five folds with the result of each loop being stored in a NumPy array. The success of each model was determined by measuring the error rate and confidence interval for the mean score. Additionally, loss curves were used as diagnostic tools for network robustness. A loss curve is tool that measures the performance of learning networks that adjust their internal parameters incrementally. They can help visualize how a model is generalizing on the data. This is demonstrated by the charts shown below which have the model loss value on the x-axis and the amount of iterations (epochs) on the y-axis.

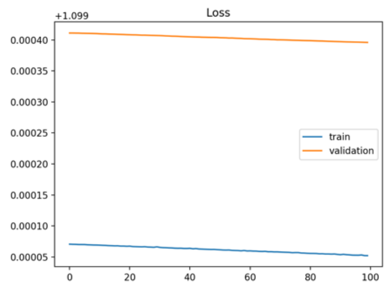


Figure 5.2: Underfitted model

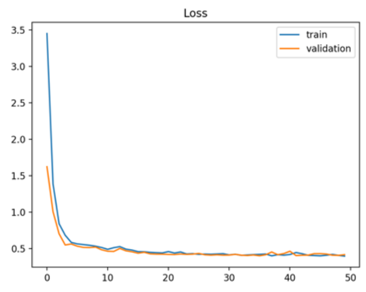


Figure 5.3: Well fit model

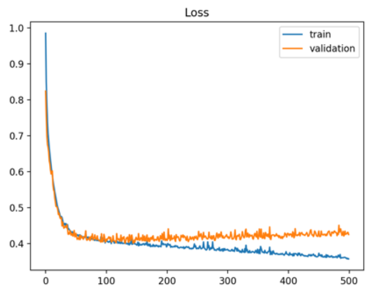


Figure 5.4: Overfitted model

The charts demonstrate an underfitted, well fitted, and overfitted model respectively. The blue line represents how well the model fit the **training** data and the orange line represents how well the model fit the **validation** data. The validation data is a part of the dataset that is held back from model training and is used as a measure of model performance on unseen data. In an underfit model, the validation line will be high relative to the training line with no convergence. This indicates that the model was completely unable to generalize the data [21]. A well fitted model will show a flat line for both the training and validation data with high levels of convergence. This indicates that the model was able to generalize the data very well and will be able to make consistent predictions on new data [21]. An overfitted model will show a validation line that progressively diverges away from the training line as the epoch number increases. This indicates that the model is over-generalizing the training data and finding trends and correlations unique to the training dataset [21]. The hyperparameters– such as layers, neurons, and batch size – were changed continuously with the resulting loss curve being used as an indicator of each parameters effect on the networks. This was an iterative process accomplished through complete trial and error. The learning rate was set using a build in keras callback function known as a scheduler. The scheduler is similar to the loss curve except it plots the loss value with dynamically adjusted learning rates as shown in figure 5.5.

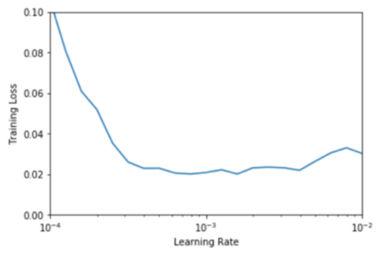


Figure 5.5: Learning Rate Loss

5.2: Model Results:

In total, this project compared four model architectures used on fourteen different datasets to make four distinct prediction. This resulted in a total of 56 trained networks. This section will provide the results for one model for each prediction being measured. For complete results, visit the GitHub repository linked in Chapter One. A naïve prediction is included in each results table - this prediction is the measure of the average error rate between each hourly index and will be used as a baseline to judge the model's performance. If the models error rate is lower than the naïve metric, then we can conclude that the model has discovered correlative patterns in the data and has predictive merit.

5.2.1: New York City:

The table below shows the results of the four networks on the New York City dataset. The **confidence** column indicates how confidant the model is that every subsequent iteration will fall within the error margin of the mean. Below the results table is the loss curve for **the last** fold in the model.

	Mean	**Low**	**High**	**Error Margin**	**Confidence**
Naïve	3.12%	3.12%	3.12%	N/A	N/A
MLP	1.97%	1.34%	2.43%	± 0.365	95%
LSTM	1.37%	1.12%	1.77%	± 0.196	95%
CNN	1.67%	1.26%	1.91%	± 0.247	95%
CNN-LSTM	1.19%	1.00%	1.43%	± 0.133	95%

Observing the loss curves below, we can see that the CNN-LSTM hybrid model is able to generalize the data better than the other models, albeit slightly overfit. The MLP and LSTM model have a lot of noise, indicating some difficulty generalizing the data. Conversely, the CNN model over-generalized the data, indicated by the disparity in the validation and training line. Additionally, we see that CNN-LSTM had the lowest MAPE value and the lowest error margin, indicating that it was the most consistent through each of the five training sets. These results was largely consistent in each model tested although there were instances where the CNN and LSTM networks slightly outperformed the CNN-LSTM network.

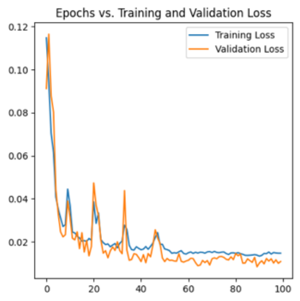


Figure 5.6: MLP Loss Curve

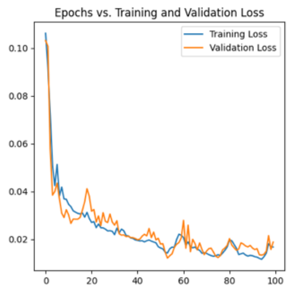


Figure 5.7: LSTM Loss Curve

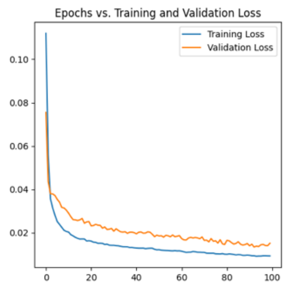


Figure 5.8: CNN Loss Curve

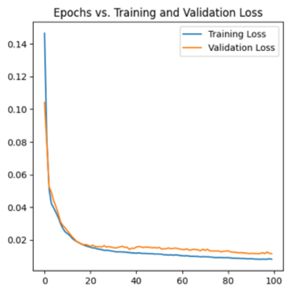


Figure 5.9: CNN-LSTM Loss Curve

Notably, each model trained on the NYC dataset outperformed the naïve prediction. This was consistent across the four other cities tested. This is an indication that energy consumption patterns in large metropolitan areas are highly consistent and have a high correlation with temporal and atmospheric conditions. This allowed the model to generalize the data successfully. Fault tolerance for the network is highly situational and it's difficult to give a blanket estimate, however the results show demonstrable evidence of the predictive merit of each network. The chart below displays the predicted values of the NYC's CNN-LSTM network plotted against the real values.

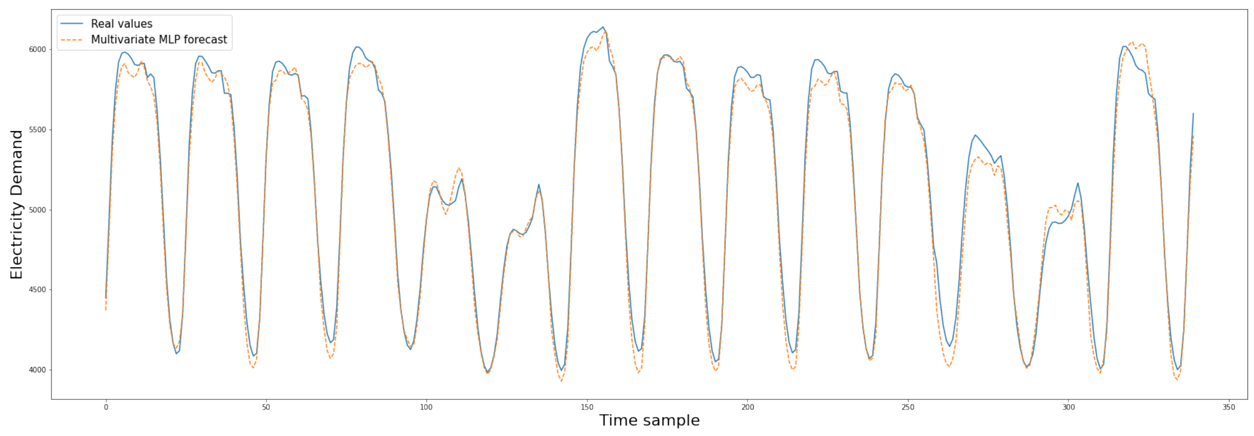


Figure 5.10: NYC Predicted vs Real Values

5.2.2: Lot 835:

The table below shows the results of Lot 835, which includes the Empire State Building, the Magellan, and the Herald Towers. As with the city dataset, the loss curves are also plotted below and depict **the last** fold in the model.

	Mean	Low	High	Error Margin	Confidence
Naïve	3.12%	3.12%	3.12%	N/A	N/A
MLP	4.81%	3.16%	8.54%	± 1.694	95%
LSTM	4.97%	2.77%	9.94%	± 2.241	95%
CNN	4.49 %	2.96%	5.91%	± 1.035	95%
CNN-LSTM	4.75%	3.43%	6.30%	± 0.859	95%

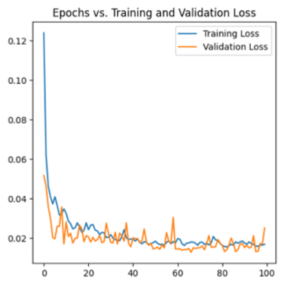


Figure 5.6: MLP Loss Curve

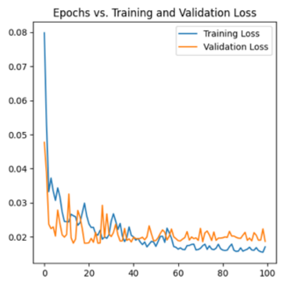


Figure 5.7: LSTM Loss Curve

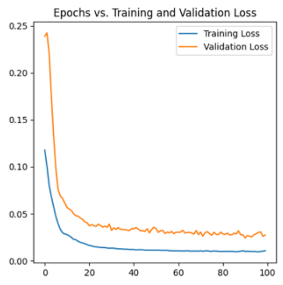


Figure 5.8: CNN Loss Curve

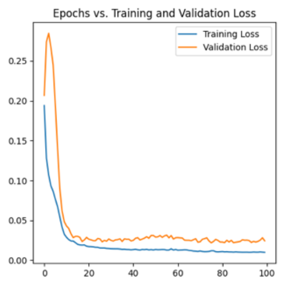


Figure 5.9: CNN-LSTM Loss Curve

Observing the charts above, we can see that the results largely emulate those seen in city models. The CNN-LSTM network is once again the best at generalizing the data. Additionally, just as in the city dataset, the MLP and LSTM models are quite noisy while the CNN model over-generalized. What differs from the city models, however, is that these models were only able to outperform the naïve metric roughly two-thirds of the time. In some instances, they significantly surpassed it. This was consistent in all three building lots tested. Diagnosing this shortcoming is difficult. By plotting the predicted values against the real values, we can get a better understating of the model's behavior.

The chart below displays the predicted values of the Lot 835's CNN-LSTM network plotted against the real values. In this iteration the model achieved a MAPE of 3.36%

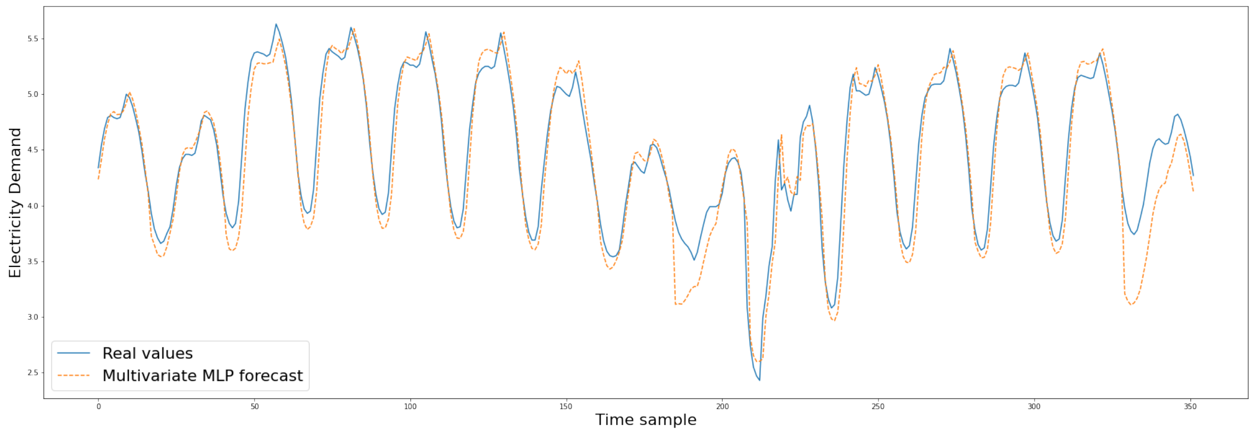


Figure 5.11: Lot 835 Predicted vs Real Values

What's notable is that the model generally maps the data well but suffers from an occasional and significant error. This is evident at time sample ~170 and ~330 where the MAPE hovers around 30%. Note that this plot shows the results of the CNN-LSTM model which had the best overall performance. The phenomena existed in the other models but to a much greater extend. This is demonstrated in figure 5.12 which shows an instance of the MLP network where the MAPE was roughly 75%. This measurement issue is consistent in each building dataset and with every model. There are numerous approaches that can be taken to try remedy this error. One would be training the model on a richer and more accurate dataset. As stated earlier, this model was limited in the fact that the hourly consumption measurements are estimates and may misrepresent the actual values. Additionally, the dataset only contained data for the year 2016. A more extensive dataset may help the model better recognize temporal trends that are difficult to extrapolate form a single year – such as seasonal variations. Conclusively, the results are promising but deficient. We see evidence of successful generalization, but

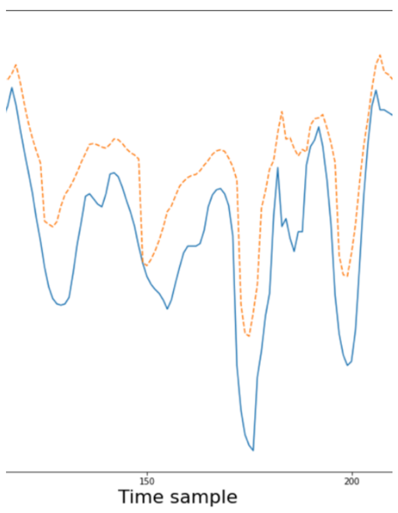


Figure 5.12: Section lot 835 MLP Plot of Actual vs Real Values

the occasional and large errors make the model unsuitable for reliable next hour predictions. Future research will need to obtain a dataset that contains the actual consumption patterns of the building within a larger time-frame.

5.2.3: France Solar:

The table below shows the prediction results for French solar generation. As with the other datasets, the loss curves below depict **the last** fold in the model. Note that these networks measured the root mean squared error (RMSE) not the mean average percent error. In this instance, the error rate represents how may megawatt hours the model was off by.

	Mean	Low	High	Error Margin	Confidence
Naïve	342.74	342.74	342.74	N/A	N/A
MLP	159.84	145.93	183.55	± 6.94	95%
LSTM	152.64	109.01	210.34	± 34.529	95%
CNN	372.22	259.67	476.99	± 65.573	95%
CNN-LSTM	307.61	227.41	412.01	± 57.234	95%

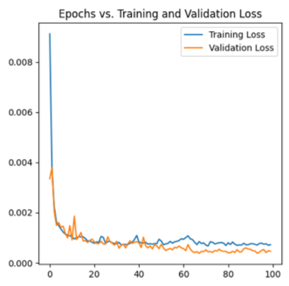


Figure 5.6: MLP Loss Curve

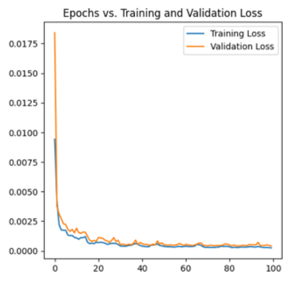


Figure 5.7: LSTM Loss Curve

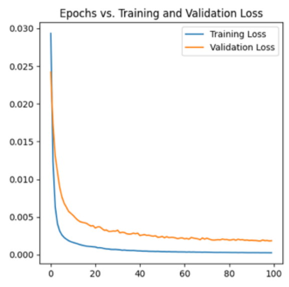


Figure 5.8: CNN Loss Curve

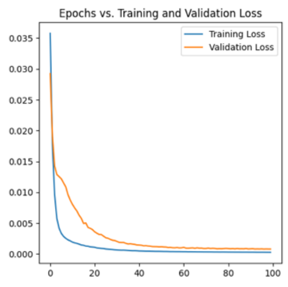


Figure 5.9: CNN-LSTM Loss Curve

The results of the solar model differ from those of the building and city datasets. We can see that the CNN has a smooth loss value but has also over-generalized the data as did the CNN-LSTM network. We can see that the consequences of over-fitting, even slightly, are severe as the CNN and CNN-LSTM network performed far worse than the other two datasets. Determining the best model in this instance is difficult as the LSTM model performed the best overall but the MLP had the least amount of variance. This trend held true even after running multiple subsequent tests. Interestingly, the trend does not hold true in the networks trained on the German or Austrian data. In the Austrian networks, the MLP and LSTM models performed equally well and in the German networks the LSTM model significantly outperformed the MLP model. However, in all cases, both the MLP and LSTM models outperformed the naïve metric demonstrating successful generalization of the data. The proper model choice will be dependent on the fault tolerance and the context of implementation. The chart below displays the MLP networks predicted values plotted against the real values. In this iteration, the MLP achieved an RMSE of 159.84.

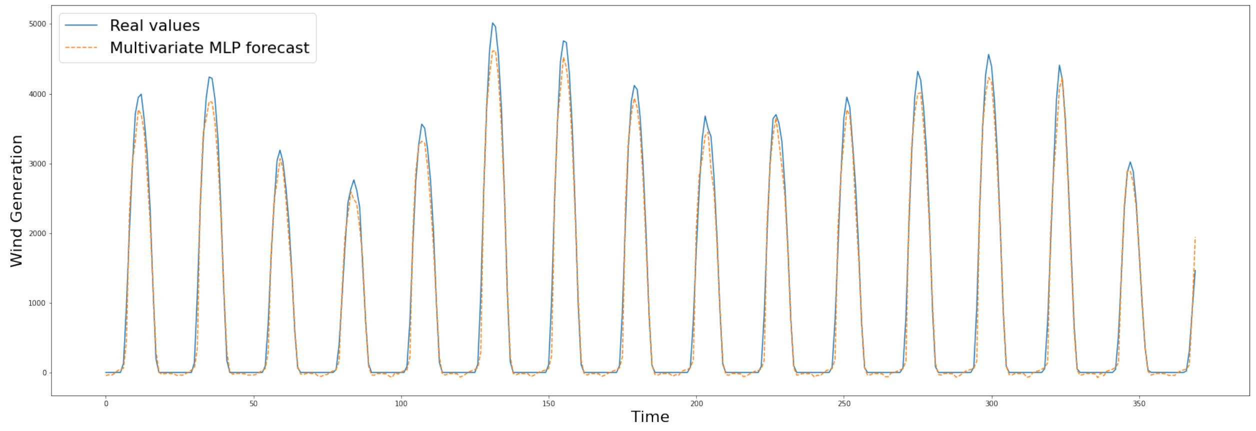


Figure 5.13: France Solar Predicted vs Real Values

5.2.4: France Wind:

The table below shows the prediction results for French wind generation. As with the city dataset, the loss curves below depict **the last** fold in the model. These networks also measured the root mean squared error and the error rate represents how may megawatt hours the model was off by.

	Mean	Low	High	Error Margin	Confidence
Naïve	220.47	220.47	220.47	N/A	N/A
MLP	663.22	561.88	916.95	± 115.081	95%
LSTM	679.00	405.07	1021.80	± 205.73	95%
CNN	719.88	573.64	941.99	± 124.32	95%
CNN-LSTM	543.02	427.73	731.15	± 91.84	95%

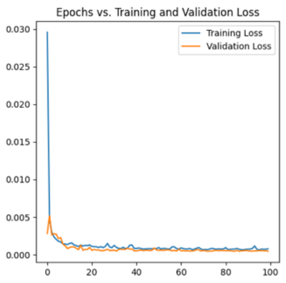


Figure 5.6: MLP Loss Curve

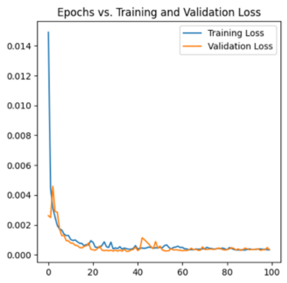


Figure 5.7: LSTM Loss Curve

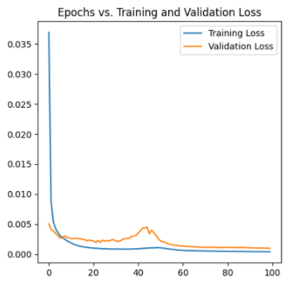


Figure 5.8: CNN Loss Curve

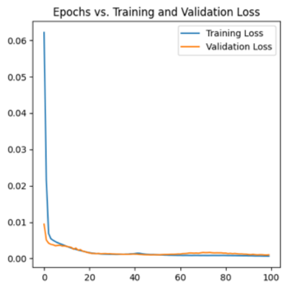


Figure 5.9: CNN-LSTM Loss Curve

Unsurprisingly the results of the wind data followed the same pattern as the solar, although with a much greater margin of error. No model performed better than the naïve metric and, in most cases, they performed significantly worse. This is due to inconsistencies in the model's ability to predict the variance of wind. Interestingly, the loss curves indicate that all the models were able to generalize the data well. The discrepancy between the loss curve and the RMSE is due to relatively infrequent but fairly significant errors in the model's predictions. The chart below displays the CNN-LSTM predictions plotted against the real values and demonstrates the variance in error rates. Note that this network achieved an RMSE of 412.42. The chart shows that the model is able to make fairly accurate predictions most of the time but underestimates peak generation, sometimes significantly. For instance, around time samples 300-310, it was off by nearly 2000 MwH.

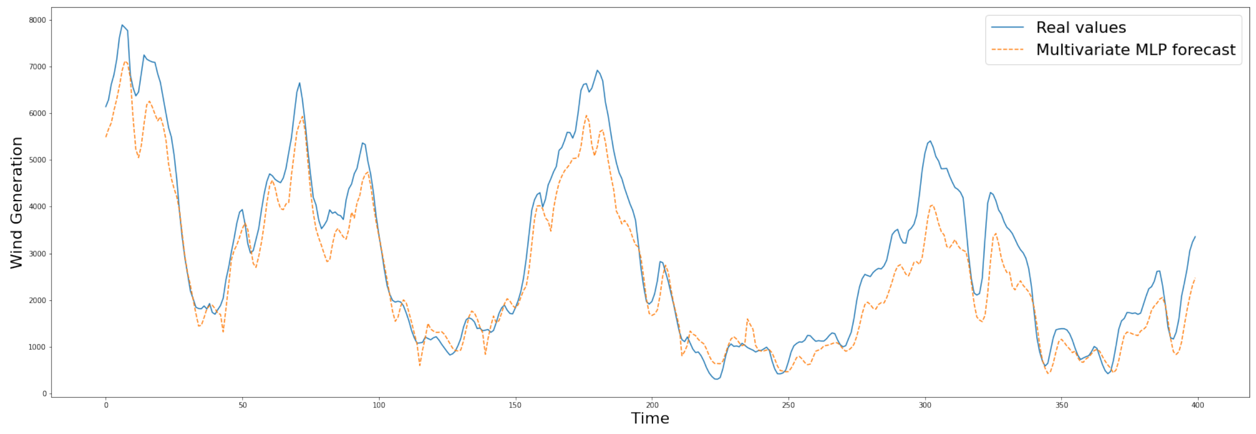


Figure 5.15: France Wind Predicted vs Real Values

Although the model generally performed well, these infrequent but significant errors can have drastic results on the final RMSE value. This same issue was present in the networks trained on the Austrian and German wind data. Just like the building networks, these networks were only trained on one year of data. A more extensive dataset may help the networks in extrapolating temporal data and predicting irregular peaks. Although the model is mostly accurate, forecasts errors of such inconsistency can be very problematic. Therefore, we have to deem all the models deficient.

Chapter 6 – Conclusion:

The objective of this report was to create an application that can comparatively analyze the success of various deep neural architectures on time series energy data. The final status of this software, at the end of this dissertation, is that it possesses all the core functionalities. The software can successfully run the neural network pipeline, end-to-end, from importation of unprocessed data to displaying the results on a Dashboard. Additionally, the the user can dynamically compare the results of the model and write the information to a database if they choose to do so. However, the software is far from production ready as numerous limitations presented themselves along the way. These limitations are discussed below:

6.1 Testability:

A fundamental limitation faced in this project was a lack of computational resources to properly and thoroughly test each network. Brownlee recommends that a neural network be tested at least 30 times to properly analyze its capabilities [21]. Given the scale of measurements attempted in this project, that would require 1560 individual iterations. Unfortunately, due to Covid-19, I was unable to access the universities resources and was limited by the capabilities of my personal laptop which would not be able to handle such stress. Because of this, each network was tested only five times. Consequently, the final results of the network may be a bit misrepresentative of the networks actually predictive merit. In order to fully evaluate each of the models and confidently report their capability, they will need to be trained on a more powerful computer for a longer period of time.

6.2 Adaptability and Extensibility:

The data used to test the networks came from a variety of sources maintained by both public and private entities. Consequently, there was no standardization in the way the data was published. As of right now, the software only works on the pre-selected datasets and would need to be reconfigured to work on new data. This is a limitation outside the control of development. In order for this software to be widely adaptable, there world need to be a government mandate on the way energy consumption and generation data is published. Barring such legislation, this software is limited to the extent that new data would need to be pre-processed manually before being fed into the network.

6.3 Quality:

The overall quality of the models was hindered by limitations in available data. As highlighted above, the building dataset only contained the estimated values of energy consumption based on city-wide patterns. This allowed for a model that could make rough estimations but fails to map the unique profiles of each building. Understanding these unique consumption patterns is integral in creating effective demand response programs as the contracts offered to buildings need to be custom tailored. Additionally, for a DSR program to know which building to target during **n** period in time, it needs as close as an approximation as possible of the building's future consumption profile. Furthermore, the building networks were only trained on one year of data. Providing a more comprehensive dataset would likely help each network extrapolate temporal or seasonal trends. It is likely, with these factors considered, that the results of the building networks are unrepresentative of the models actually capability to approximate the consumption patterns of individual buildings. Given that the approximations were made based off city-wide consumption, the models likely benefitted from the predictability of aggregated consumption. This is made even more evident given the plot of predicted values for each building which demonstrate a fairly consistent and predictable consumption pattern. Additionally, to a lesser extent, the renewable models may also have suffered from lack of temporal data. These models were also only trained on a year of data which may have hindered their ability to generalize seasonal variances.

Ultimately, the quality issues in this project can be rooted in a lack of data. Electricity data tends to be proprietary and utilities that exist as **'natural monopolies'** have vested interest in keeping it private [1].

6.4 Deployment:

I had hoped to deploy this software on a Docker container; however, I was prohibited by time constraints. A Docker container seems to be the most logical deployment method as it allows for a virtualized environment to be run on any OS under one system kernel. This would allow for high portability and ease of distribution to any consumer, generator, or supplier in the energy pipeline.

6.5 Further Work:

I plan to continuously work on this project into the foreseeable future as I have learned a great deal and very much enjoyed it. Thus far, this project has largely been a data science project focused on the end-to-end machine learning pipeline. As time passes, I hope to expand it into a more thorough piece of software. In future iterations I hope to add a more dynamic front-end where the networks can be tuned and trained directly form the GUI. Furthermore, I would like to add more visualization features so that the dataset can be better understood by the user. Additionally, I hope to eventually deploy the software on a docker container to improve its extensibility.

In terms of data access, the Joe Biden campaign is introducing numerous pieces of legislature regrading infrastructural reform. One of the most ambitious structural reformations is the transition to 100% clean energy by 2035 [41]. It seems likely that this would necessitate mass democratization of electricity data to help spur research and innovation in the private sector [1]. If these event come into fruition, then I hope to continually train and test these models as the data becomes more and more available.

6.6 Reflection:

This project has a been a very rewarding process through and through. It was one of the most significant learning processes I have undertaken. The final version of this project is a direct result of the steep learning curve undertaken. When I began this project three months ago, I had never written a line of code in Python and had only a vague understating of neural networks. As the project iteratively evolved from a foggy idea to a realized piece of software, I became increasingly confident in my Python and data science capabilities. I credit this project with spurring my interest in data science and machine learning and I plan to continuously work towards becoming more proficient in both. Additionally, it allowed me to incorporate elements of the energy sector, which is a filed I am very passion about and hope to pursue a career in. Ideally, this project can help catalyze my career, and I hope to find myself working at the intersection of energy and data science very soon.

Bibliography (Text):

[1] C. Ruppel. *Personal Interview.* Senior Vice President at DCO Energy, 2020.

[2] Skelton, G. (2020, August 31). *Column: In this year's atypical California fire season, politicians find the blame game won't work*. Los Angeles Times. https://www.latimes.com/california/story/2020-0831/george-skelton-fire-season-blame

[3] Electricity Consumers Resource Council (ELCON). (2004). The Economic Impacts of the August 2003 Blackout. https://elcon.org/wpcontent/uploads/Economic20Impacts20of20August20200320Blackout1.pdf

[4] Isidore, C. (2019, January 15). *PG&E, utility tied to wildfires, will file for bankruptcy.* CNN. https://www.cnn.com/2019/01/14/business/pge-bankruptcy-wildfires/index.html#:~:text=New%20York%20(CNN%20Business)%20Pacific,period%20required%20by%20California%20law

[5] Jacob, M., Neves, C., & Greetham, D. V. (2019). *Forecasting and assessing risk of individual electricity peaks.* Springer Nature.

[6] Singh, R. P., Gao, P. X., & Lizotte, D. J. (2012). On hourly home peak load prediction. *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*. https://doi.org/10.1109/smartgridcomm.2012.6485977

[7] Bremer, M. (2012, March). *Multiple Linear Regression.* Cornell University . http://mezeylab.cb.bscb.cornell.edu/labmembers/documents/supplement%205%20-%20multiple%20regression.pdf

[8] Moghram, I., & Rahman, S. (1989). Analysis and evaluation of five short-term load forecasting techniques. *IEEE Power Engineering Review*, *9*(11), 42-43. https://doi.org/10.1109/mper.1989.4310383

[9] Kumar, S., Mishra, S., & Gupta, S. (2016). Short term load forecasting using ANN and multiple linear regression. *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT)*. https://doi.org/10.1109/cict.2016.44

[10] Cao, X., Dong, S., Wu, Z., & Jing, Y. (2015). A data-driven hybrid optimization model for short term residential load forecasting. 2015 IEEE International Conference on Computer and

[11] Oracle. (n.d.). *ARIMA equations*. Oracle.com. https://docs.oracle.com/cd/E57185_01/CBREG/ch06s03s04s01.html

[12] Al-Musaylh, M. S., Deo, R. C., Adamowski, J. F., & Li, Y. (2018). Short-term electricity demand forecasting with MARS, SVR and ARIMA models using aggregated demand data in Queensland, Australia. *Advanced Engineering Informatics*, *35*, 1-16. https://doi.org/10.1016/j.aei.2017.11.002

[13] Fu, Y., Li, Z., Zhang, H., & Xu, P. (2015). Using support vector machine to predict next day electricity load of public buildings with sub-metering devices. *Procedia Engineering*, *121*, 1016-1022. https://doi.org/10.1016/j.proeng.2015.09.097

[14] Hong, W. (2009). Electric load forecasting by support vector model. *Applied Mathematical Modelling*, *33*(5), 2444-2454. https://doi.org/10.1016/j.apm.2008.07.010

[15] Khan, R. A., Dewangan, C. L., Srivastava, S. C., & Chakrabarti, S. (2018). Short term load forecasting using SVM models. *2018 IEEE 8th Power India International Conference (PIICON)*. https://doi.org/10.1109/poweri.2018.8704366

[16] Trask, Andrew W. *Grokking Deep Learning*. Manning, 2019.

[17] Li, F., Johnson, J., & Yeung, S. (2017, April 13). *Backpropagation and Neural Networks* [PowerPoint slides]. Stanford. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

[18] Brilliant. (n.d.). *Backpropagation*. brilliant.org. https://brilliant.org/wiki/backpropagation/#:~:text=Backpropagation%2C%20short%20for%20%22backward%20propagation,to%20the%20neural%20network's%20weights.

[19] Vandeput, N. (2020, September 10). *Forecast kpi: Rmse, Mae, MAPE & bias*. Medium. https://towardsdatascience.com/forecast-kpi-rmse-mae-mape-bias-cdc5703d242d

[20] Wang, X.g., et al. "An Improved Backpropagation Algorithm to Avoid the Local Minima Problem." *Neurocomputing*, vol. 56, 2004, pp. 455–460., doi:10.1016/j.neucom.2003.08.006.

[21] Hayou, Soufiane, et al. "On the Selection of Initialization and Activation Function for Deep Neural Networks." *ArXiv*, vol. 1, no. 2, 21 May 2018, doi:1805.08266 .

[22] Scikit-Learn. (n.d.). *1.17. Neural network models (supervised) — scikit-learn 0.23.2 documentation*. scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. Retrieved September 15, 2020, from https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[23] Brownlee, J. (2018). *Deep learning for time series forecasting: Predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

[24] Rodrigues, F., Cardeira, C., & Calado, J. (2014). The Daily and hourly energy consumption and load forecasting Using artificial neural network method: A case study using a set of 93 households in Portugal. Energy Procedia, 62,220-229. https://doi.org/10.1016/j.egypro.2014.12.383

[25] Seunghyoung Ryu, Jaekoo Noh, & Hongseok Kim. (2016). Deep neural network based demand side short term load forecasting. *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. https://doi.org/10.1109/smartgridcomm.2016.7778779

[26] Dalto, M., Matusko, J., & Vasak, M. (2015). Deep neural networks for ultra-short-term wind forecasting. *2015 IEEE International Conference on Industrial Technology (ICIT)*. https://doi.org/10.1109/icit.2015.7125335

[27] Chambers, J, and D Mandic. "Recurrent Neural Networks Architectures." *Wiley Series in Adaptive and Learning Systems for Signal Processing, Communications, and Control Recurrent Neural Networks for Prediction*, 2001, pp. 69–89., doi:10.1002/047084535x.ch5.

[28] Colah. (2015, August 27). Understanding LSTM networks. colah's blog. https://colah.github.io/posts/2015-08 Understanding-LSTMs/

[29] Narayan, A., & Hipel, K. W. (2017). Long short term memory networks for short-term electric load forecasting. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. https://doi.org/10.1109/smc.2017.8123012

[30] Hossain, Mohammad Safayet, and Hisham Mahmood. "Short-Term Photovoltaic Power Forecasting Using an LSTM Neural Network." *2020 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2020, doi:10.1109/isgt45199.2020.9087786.

[31] Muzaffar, S., & Afshari, A. (2019). Short-term load forecasts using LSTM networks. *Energy Procedia*, *158*, 2922-2927. https://doi.org/10.1016/j.egypro.2019.01.952

[32] Koprinska, Irena, et al. "Convolutional Neural Networks for Energy Time Series Forecasting." *2018 International Joint Conference on Neural Networks (IJCNN)*, 2018, doi:10.1109/ijcnn.2018.8489399.

[33] *Merra. (2017, April 10). GMAO - Global Modeling and Assimilation Office Research Site. https://gmao.gsfc.nasa.gov/reanalysis/MERRA/*

[34] *Load data - NYISO*. (n.d.). NYISO NYISO. https://www.nyiso.com/load-data

[35] *PJM - Data directory. (n.d.). PJM. https://www.pjm.com/markets-and-operations/data-dictionary.aspx*

[36] *Balancing authority load data*. (n.d.). BC Hydro - Power smart. https://www.bchydro.com/energy bc/operations/transmission/transmission-system/balancing-authority-load-data.html

[37] *California ISO. (n.d.). California ISO. https://www.caiso.com/Pages/default.aspx*

[38] *California Energy Commission. https://www.energy.ca.gov/*

[39] European Union. (September 14). *Home*. European Data Portal. https://www.europeandataportal.eu/en

[40] *Card, D. (2017, July 5). The "black box" metaphor in machine learning. Medium. https://towardsdatascience.com/the-black-box-metaphor-in-machine-learning-4e57a3a1d2b0*

[41] DCO Energy. (n.d.). Home | DCO. https://www.dcoenergy.com/

Bibliography (Figueres):

[1] Garbade, M. (2018, October). *How to create a simple neural network in Python*. KDnuggets. https://www.kdnuggets.com/2018/10/simple-neural-network-python.html

[2] Missinglink.a. (n.d.). *Backpropagation in neural networks: Process, example & code*. MissingLink.ai. https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/

[3] Brownlee, J. (2018). *Deep learning for time series forecasting: Predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

[4] Choi, S. (2018, April 23). Recurrent Neural Network and Their Applications [PowerPoint slides]. Slideshare. https://www.slideshare.net/samchoi7/rnnerica

[5] Colah. (2015, August 27). Understanding LSTM networks. colah's blog. https://colah.github.io/posts/2015-08 Understanding-LSTMs/

[6] Britz, D. (2016, January 10). Understanding Convolutional neural networks for NLP. WildML. https://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[7] (n.d.). Touropia. https://www.touropia.com/

[8] Nawara, S. (n.d.). *Avoiding data leakage in machine learning*. Conlan Scientific. https://conlanscientific.com/posts/category/blog/post/avoiding-data-leakage-machine-learning/