EECS 484 Project

Nikil Pancha

December 2017

1 Introduction

Short-term load forecasting is a task of importance in many sectors, from business to transit to energy. For this project, I have explored the area of energy, and in particular electricity on a hourly scale with a maximum prediction horizon of 24 hours. It is important for suppliers to have accurate forecasts so that they can determine how much electricity they need to produce ahead of time. If the supplier knows how much electricity will be demanded, they can adjust their output to minimize excess electricity production. If not enough energy is supplied, then the distributor will likely need to buy extra electricity on the spot market, where prices will be significantly higher than they are earlier. Although these exact concerns only apply to the energy industry, the problem of forecasting is relevant in other industries, wherever decisions must be made based on future projections. Electricity data in particular has fairly regular trends, and does not fluctuate wildly with season, in contrast with other forms of energy, such as natural gas, where there are large fluctuations. In practice, such a model will need to be trained for many different regions, so even if a method performs better than others, the training time could be prohibitively costly, so I limit training to 300 iterations.

In this project, I explore using LSTMs to forecast electricity demand. Due to computational time constraints, I limit myself to predicting 24 hours in the future. I first explore the simplest such model, where the prediction at time t is fed into the model as an input at time $t + 1$, which will be referred to as an autoregressive or iterative multistep method. This is the most popular way to use LSTMs for forecasting, and there has been significant work done in this area [1]. The next approach I evaluate is a sequence-to-sequence (seq2seq) model, which is popular in natural language processing tasks, and has not been used much for load forecasting, and the only example I found of it being used for forecasting was on data with very few attributes [2]. I also explore a modification to the standard sequence-to-sequence model which uses separate encoders for different timescales to try to capture longer dependencies in the data.

2 Related Work

Artificial neural networks (ANNs) have also successfully been used for time series forecasting [3]. Their incorporation of nonlinearities allows for more complicated trends to be captured than a linear regression model allows, and the increase of layer sizes and number of layers can further this ability [4]. One drawback of ANNs is their inability to model sequences directly, as they are designed to predict values based on a set number of inputs, with no ability to account for the temporal structure of the data. To resolve this, an ANN may be used autoregressively, using the previous output as an input at the next timestep, but this can cause predictions to severely decrease in accuracy as the prediction horizon increases [5, 6]. Instead, a common method is to use separate models to predict each timestep, which makes training expensive, especially when predictions for many horizons are needed [5].

Another type of Neural Network that has been used in time series prediction is the Convolutional Neural Network (CNN). 1-D convolutions have been used alone, as well as in conjunction with other methods [7].

When combined sequentially with an LSTM, I found that the model very quickly overfit, so I did not extensively explore the possibility of combining an LSTM and CNN.

Long short-term memory (LSTM) cells have been used in building a variety of neural networks, most commonly in the realm of natural language processing [8, 9]. LSTMs have also been used for the problem of short-term load forecasting, but they are most commonly used autoregressively, meaning that for multi-step predictions, they feed the prediction at time t as an input at time $t+1$. The other other popular architecture uses a separate LSTM on past data (called an encoder), and feeds the output state into another LSTM (called the decoder). This idea was first introduced in Sutskever et. al. [9], and has since become a popular model.

3 Models Used

The simplest type of Recurrent Neural Netowrk (RNN) is an Elman RNN, which has a state transition as follows [10]:

$$\begin{aligned} h_t &= g_h(W_h x_t + W_h h_{t-1} + b_h) \\ y_t &= g_y(W_y h_t + b_y) \end{aligned}$$

where h_t is the hidden state, x_t is the input, y_t is the prediction, g_h and g_y are nonlinear activation functions, and the Ws and bs are learnable parameter matrices and vectors. These networks, however, encounter the issue of vanishing gradients, as g_h is usually tanh. Becuase this W_h matrix may be applied many times, if even one of its eigenvalues is high, then when calculating gradients, the backwards signal will be amplified at every timestep. Similarly, if all of its eigenvalues are low, then the signal will exponentially decrease. This structure is popular because it allows for efficient processing of sequences, as the weights are invariant through time, rather than a feedforward architecture, where different weights would be applied to the sequence at different timesteps.

3.1 LSTM

Long short-term memory (LSTM) networks [11, 12] were developed to solve this issue of vanishing and exploding gradients in Elman RNNs, allowing for learning relationships in longer sequences. Instead of having a single operation to determine the output, an LSTM has several gates to control the flow of information. For an LSTM, the transition is as follows:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + U_c h_{t-1} + b_c) \\ h_t &= o_t \circ tanh(c_t) \end{aligned}$$

where $\circ$ denotes the Hadamard product, and σ denotes the sigmoid function [12]. Here, f_t, i_t, and o_t are gates that control which parts of the previous state to weight highly, which part of the input values to weight highly, and which part of the outputs to weight highly, respectively. c_t is the cell state, and h_t is the output, while W, U, and b are all learnable parameters.

Ideally this would start at $t = 0$ and the h_t and c_t would be built up from there, but due to computational constraints, typically this is truncated at some point, and starts off with $h_t = c_t = \mathbf{0}$. The idea of a seq2seq model is an encoder can learn an ideal representation of the previous $x_t, \ldots, x_{t-23}$ (in c_t and h_t), and then these c_t and h_t can be fed into another LSTM that starts with them as the initial state, rather than a 0 vector.

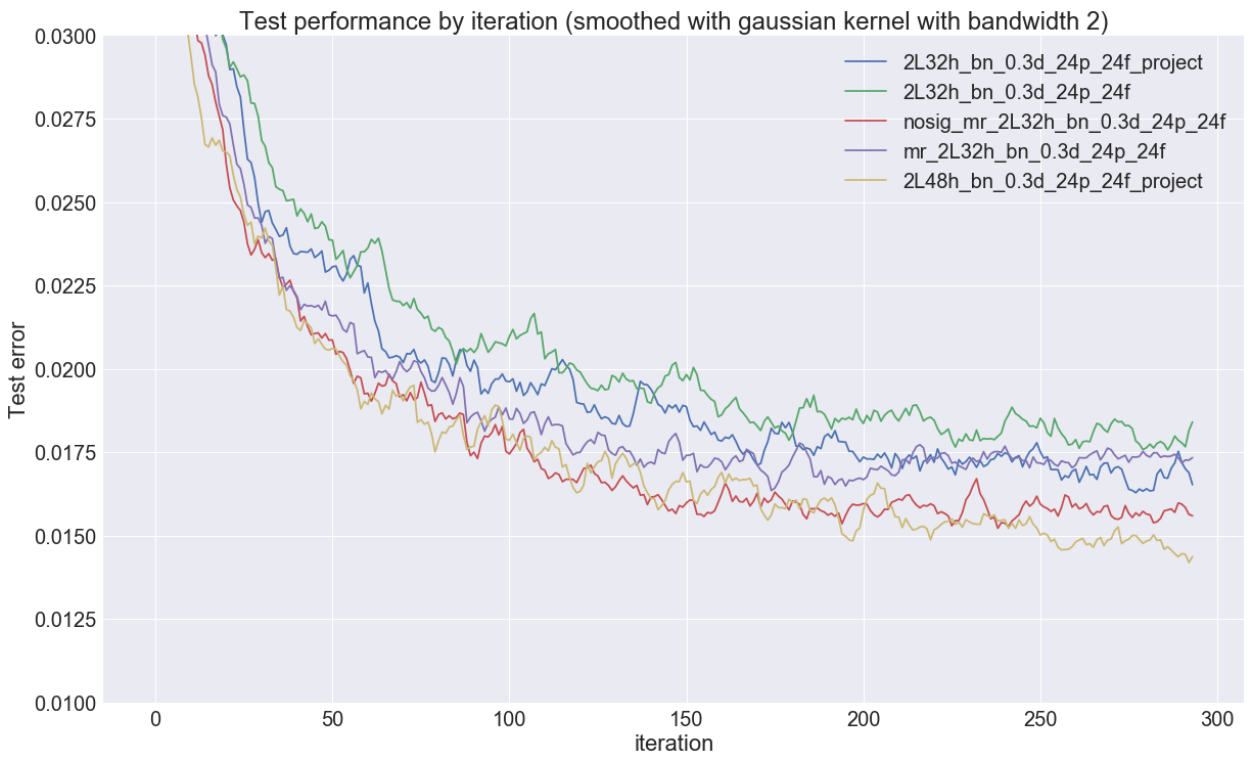

Figure 1: Plot of subset of experiments

3.2 Multiple Encoders

For dealing with time series on multiple scales, a couple traditional approaches are dilated convolutions and Clockwork RNNs. The downside to a Clockwork RNN is that it still needs to train on the whole length of the input, so even though it requires less computation than a standard LSTM, it requires enough that extracting information from 500 past points is too expensive. First, let y_t be the true load at time t, and let u_t be a feature vector with information at time t. Then, we can have two encoders, rather than just one. The first one will take an input $x_t = \begin{pmatrix} y_t \\ u_t \end{pmatrix}$, while the second one will take the input $\bar{x_t} = \begin{pmatrix} \frac{1}{24}\sum_{t-23}^{t} y_i \\ \frac{1}{24}\sum_{t-23}^{t} u_i \end{pmatrix}$. which is the average over 24 hours of the features. This will allow us to extract trends from much further in the past with little added cost; if we use 21 previous days, it only requires 21 steps, rather than $24 \cdot 21 = 504$ steps. If our decoder starts predictions at $t+1$ and ends at $t+24$, then we get an output state hd_t and cd_t. from this second encoder (using daily averages), and an output of hh_t and ch_t for the encoder operating on hourly data. Because we only have one decoder, we transform these with $h_{t+1} = W_{hh}hh_t + W_{hd}hd_t$ and $c_{t+1} = W_{ch}ch_t + W_{cd}cd_t$. I also experimented with choosing how much of the daily and hourly state to keep by outputting a single value from the RNN, and then using that output to decide an amount of each of the encoder states to use (using a sigmoid), with the idea that there might be certain cases where daily input is not useful, while in some cases, perhaps hourly information is less useful. I call this the multi-resolution model.

4 Experiments

For experiments, I used data from the GEFCom 2014 competition. This includes about 9 years of hourly electricity data with the attributes Date, Hour, and Temperature. All of these, as well as the output, were

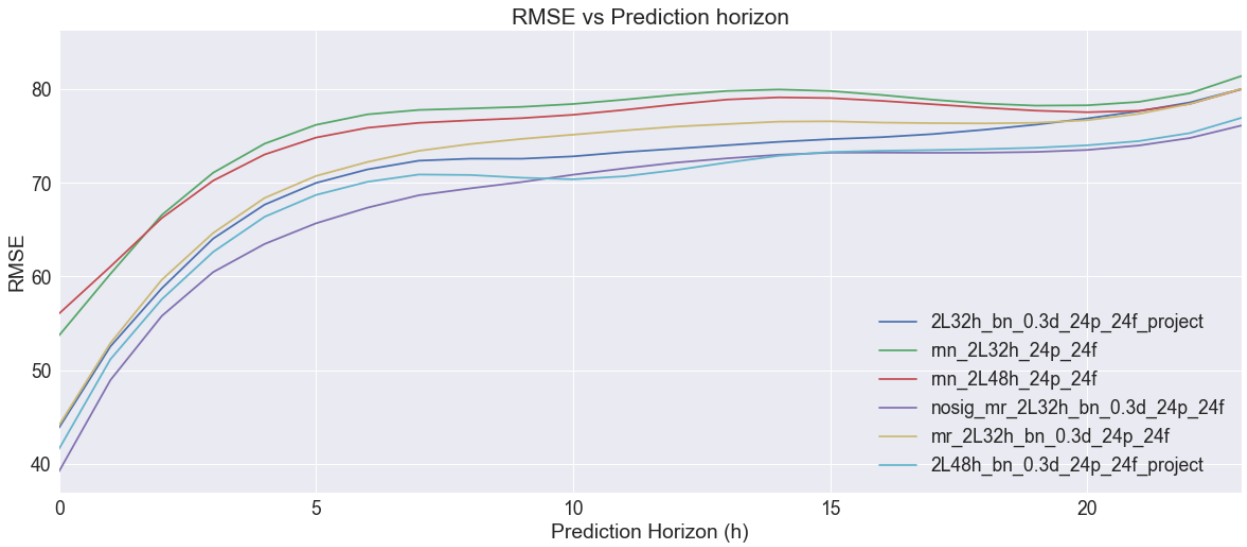

Figure 2: RMSE vs Prediction Horizon

standardized, and the hour and date were transformed into features as follows:

$$\text{hr}_t = \cos\left(\frac{2\pi}{24}\text{Hour}_t\right)$$
$$\text{day}_t = \cos\left(\frac{2\pi}{7}\text{DayOfWeek}_t\right)$$
$$\text{month}_t = \cos\left(\frac{2\pi}{12}\text{Month}_t\right)$$

In addition, I added an indicator variable for whether or not it is a holiday, as that might affect the daytime energy consumption. These features are then combined into the vector u_t. In the iterative multistep LSTM method, the input vector is $x_t = (y_{t-1}\ u_t)$, and once $t-1$ is in the future, this becomes $x_t = (\hat{y}_{t-1}\ u_t)$, allowing the input to retain its form. For the seq2seq models, $x_t = (y_t\ u_t)$ in the past, and $x_t = u_t$ for future steps, as y_t is then unknown. I use 24 hours of past data and 21 days of daily averages to predict demand at each of the next 24 hours.

A plot of all results can be seen in Figure 1. Several experiments are not plotted, but their results are in Table 1. These are smoothed to try to be able to better see relative trends. Many of these appear to not reach a minimum in the error, but due to the time required to train, they were not feasible to run for longer. The only experiment run without dropout achieves significantly worse performance than those using 0.3 dropout, even though it converges faster, so I decided to not further explore that, and stuck with 0.3 dropout. I did not explore more recent regularization strategies such as recurrent dropout and layer normalization due to time constraints and the issue that they significantly increase the running time of the model Both iterative multistep approaches seemed to perform comparably, although the one with more hidden units performs better initially, and then stops improving. In addition, when dropout is applied in the same manner as for the seq2seq model to the iterative LSTM, interestingly, it is not able to fit the data nearly as well. The suffix of "project" or prefix of "nosig" indicates a learned affine transformation between the encoder and decoder c_t and h_t.

We can see that the iterative multistep approaches seem to perform worse than the others, so it makes more sense to focus on the sequence to sequence models, seen in Figure 1. The green line is run using a seq2seq model with 2 layers and 32 hidden units each, with BatchNorm and Dropout, but no affine transform. The blue line is the exact same as the green, except it uses an affine transform. From this, we can see that it appears an that transforming the states is beneficial. Without an affine transform, we can see that using daily averages (purple line) significantly accelerates convergence, but might be slightly prone to overfitting.

Type	BatchNorm	Dropout	Hidden	Affine	Min avg MSE	Min MSE
Seq2seq	✓	0.3	32	✓	0.01629	0.01519
Seq2seq	✓	0.3	32		0.01757	0.01626
Basic		0.0	32	N/A	0.01762	0.01679
Basic		0.0	48	N/A	0.01785	0.01704
Multi	✓	0.3	32	✓	0.01522	0.01401
Seq2seq	✓	0.0	32		0.01715	0.01622
Multi	✓	0.3	32		0.01635	0.01565
Seq2seq	✓	0.3	48	✓	0.01420	0.01348
Basic		0.3	32	N/A	0.06498	0.06115

Table 1: Results: MSE values are scaled by the standard deviation, averaging with a Gaussian filter with $\sigma = 2$

In place of affine transforms, it uses learned sigmoid gates to control the information received from each encoder. This experiment's analogue is the green line, so we can see a significant improvement by including daily averages. Comparing the bottom two lines, we can see that when using an affine transformation with the multi-encoder model, we get much better performance than the comparable version without daily averages (blue line). Comparing this to a slightly larger network shows that the larger one might be better at capturing the patterns in the data, at the cost of a significantly increased runtime. In the future, it could be useful to run the multi-resolution version with more hidden units or layers to see if more information could be extracted.

Figure 2 shows the performance over different prediction horizons, and interestingly, there is not a steady increase in inaccuracy. In the beginning, errors increase rapidly, but after about a 5 hour horizon, it flattens out for the remaining time. This seems to indicate that it is equally easy to predict from 5 to 20 hours out, which is somewhat in line with intuition, as that all seems to be a medium-range target.

5 Conclusion

This paper investigated the relative performance of different varieties of LSTM architectures for time series forecasting, including standard iterative multistep, basic seq2seq, and multi-encoder models. As a baseline, the most common algorithm, the iterative multistep LSTM, achieves worse MSE than the other types of model, indicating that it might be appropriate to transition to an alternative one. Of the two types of seq2seq model, the one with multiple encoders achieves a lower error using the same parameters, but this might change if more hidden units or more timesteps back in the past are used. The goal of this was to improve performance without significantly increasing computational cost, so I somewhat succeeded in this aspect.

In contrast to [2], I evaluated the model on real-life data that included attributes beyond temporal ones (such as temperature), but still found results consistent with theirs. These models could be further explored in several ways, such as improving regularization through more modern methods, improving the combination of multiple encoders, and investigating attention mechanisms.

Code can be found at `https://github.com/cnapun/EECS-484`.

References

[1] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, "An overview and comparative analysis of recurrent neural networks for short term load forecasting," *arXiv preprint arXiv:1705.04378*, 2017.

[2] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using deep neural networks," in *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*. IEEE, 2016, pp. 7046–7051.

[3] A. Bakirtzis, V. Petridis, S. Kiartzis, M. Alexiadis, and A. Maissis, "A neural network short term load forecasting model for the greek power system," *IEEE Transactions on power systems*, vol. 11, no. 2, pp. 858–863, 1996.

[4] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[5] S. B. Taieb, R. J. Hyndman *et al.*, "Boosting multi-step autoregressive forecasts." in *ICML*, 2014, pp. 109–117.

[6] C. Hamzaçebi, D. Akay, and F. Kutay, "Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3839–3844, 2009.

[7] T. Lin, T. Guo, and K. Aberer, "Hybrid neural networks over time series for trend forecasting," 2017.

[8] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*. IEEE, 2013, pp. 273–278.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[10] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.