# Deep Learning for Time Series Modeling

CS 229 Final Project Report

Enzo Busseti, Ian Osband, Scott Wong

December 14th, 2012

## 1 Energy Load Forecasting

Demand forecasting is crucial to electricity providers because their ability to produce energy exceeds their ability to store it. Excess demand can cause "brown outs," while excess supply ends in waste. In an industry worth over $1 trillion in the U.S. alone [1], almost 9% of GDP [2], even marginal improvements can have a huge impact. Any plan toward energy efficiency should include enhanced utilization of existing production.

Energy loads provide an interesting topic for Machine Learning techniques due to the availability of large datasets that exhibit fundamental nonlinear patterns. Using data from the Kaggle competition "Global Energy Forecasting Competition 2012 - Load Forecasting" [3] we sought to use deep learning architectures to predict energy loads across different network grid areas, using only time and temperature data. Data included hourly demand for four and a half years from 20 different geographic regions, and similar hourly temperature readings from 11 zones. For most of our analysis we focused on short term load forecasting because this will aide online operational scheduling.

## 2 Deep Neural Networks

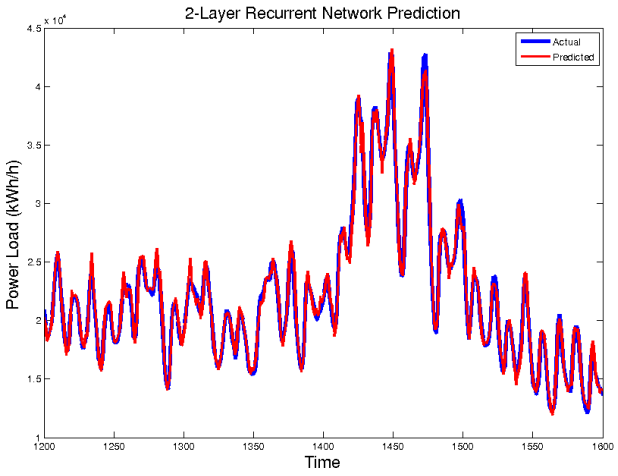
We successfully implemented deep learning architectures for forecasting power loads and found that this produced superior results to both linear and kernelized regression for our given data. We found that, due to the huge dataset we were able to implement complex nonlinear models without encountering much problem of over-fitting. Deep networks allowed us to add significant complexity to our model without specifying what forms the variation should take. Many papers extol the benefits of greedy layer-wise unsupervised training for deep network initialization [5] [6] [10] (i.e., stacked autoencoders and restricted boltzmann machines), which was the starting point for our work (see Appendix C). This is in contrast to the prohibitive scaling properties of either polynomial regression or even nonparametric Gaussian Processes, which scale $O(m^3)$ with data [4].

Our most successful iteration was a recurrent neural network with RMSE of 530 kWh/h and 99.6% correlation to test data(see Table 1). Figure 1 shows a sample comparison of actual load demand in blue (from the test data set) and the recurrent network's prediction in red. TESLA energy forecasting boasts mean absolute percentage error of 0.84-1.56% [7], suggesting our neural network implementations are beginning to approach the best of the private sector.

Table 1: Results for different learning models

| **Learning Method** | **RMSE** | **% RMSE** |
|---|---|---|
| Kernelized Regression | 1,540 | 8.3% |
| Frequency NN | 1,251 | 6.7% |
| Deep Feedforward NN | 1,103 | 5.9% |
| Deep Recurrent NN | 530 | 2.8% |

Figure 1: Load prediction with recurrent neural network




## 3 Review of Existing Techniques

There has been extensive research performed in the area of energy load forecasting [8], and in particular, the efficacy of neural networks in the field [9]. Additionally, numerous consultancies offer demand forecasting as a service to producers, while other producers have taken such analysis in-house. Difficulties in implementation and lack of transparency of results have been cited as their main weaknesses. Our hope was that through implementing a deep architecture as per the UFLDL tutorials we could successfully unearth complicated underlying features without specific oversight from the modelers perspective.

## 4 Step 1 - The Linear Model

### 4.1 Linear models as the degenerate NN

The underlying premise of neural networks is that multiple layers of generalized linear models can combine to produce non-linear outputs. Linear regression can therefore be thought of as a special case for a degenerate neural network with no hidden layers. Neural networks are complicated, with significant programming challenges and non-convex optimiza-

tions, so we first wanted to establish that they really do provide some worthwhile reward over simpler models.

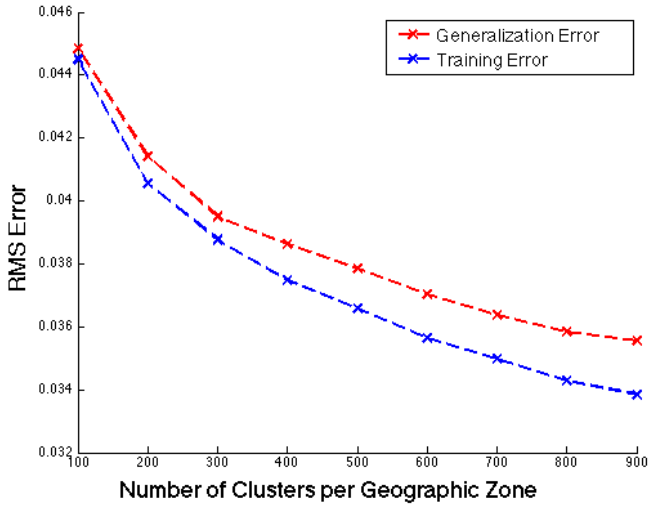
After considering some necessary state-space transformations to our data (Appendix A) we implemented least squares regressions over a simple linear model. Unsurprisingly, this suffered from severe high bias as our model was far too simple. Repeating this exercise for quadratic and cubic features we were still able to train over the whole dataset, but the problems of high bias remained still.

### 4.2 Scaling issues with large data

This process of feature creation was wasteful as full polynomial expansion gave many features without statistical significance but still necessitated lengthy calculations. Using subset selection methods (such as forward/backward stepwise or the LASSO) and only expanding relevant variables helped weed these out. However, this was only computationally possible on smaller subsets of the data, which might cause us to lose valuable training information.

To maintain most of the information of the dataset while facilitating more complex calculations, we implemented k-means clustering. Running the algorithm until convergence was prohibitively slow, however we found good results with some expedited heuristics. Using different numbers of clusters and performing a naive "nearest centroid" classification we saw improved results as we increased model complexity (Figure 2). We found that clustering was more effective using feature standardization (mean zero/unit variance).

Figure 2: Cross validation of number of clusters using nearest neighbor




### 4.3 Kernel methods add model complexity

In a final bid to add more model complexity, we implemented a kernelized local regression based upon squared exponential distance to centroids. We found that our results were improved by additionally considering the number of data points forming part of each cluster.

Once again, computational expense proved to be our limiting factor and we were only able to complete the kernelized regression with 400 clusters per geographic zone. This produced disappointing results as the means over-simplified our dataset and our kernels were unable to scale to the necessary data sizes.

## 5 Step 2 - Neural Nets

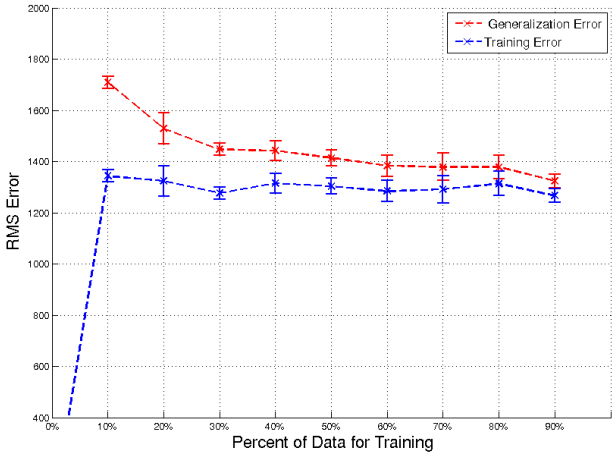
Neural networks are a more principled way to add complexity on large datasets. We implemented feedforward networks to explore model parameters and then augmented these with recurrent structure to improve forecasting accuracy.

We chose our model parameters through k-folds validation on our data, including: sample set size, feature engineering, anomaly filtering, training algorithms, and network topology. Due to the computationally intensive nature of this process, only feedforward networks were considered at this stage. Unless otherwise stated, the networks were trained with 70% of the data, 15% was used for validation stopping, and 15% for testing (chosen at random). This was repeated 10 times per experiment to average out sample error.

### 5.1 Learning Curves

To assess levels of bias and variance in a mid-sized feedforward network (two hidden layers of 30 and 10 neurons, respectively), we trained the network with varying proportions of training data and examined the learning curves. Figure 3 summarizes the results, once again demonstrating a high bias problem even with this complex neural network. This led us to focus our work on developing new features and incorporate increased model complexity.

Figure 3: Learning curves show high bias

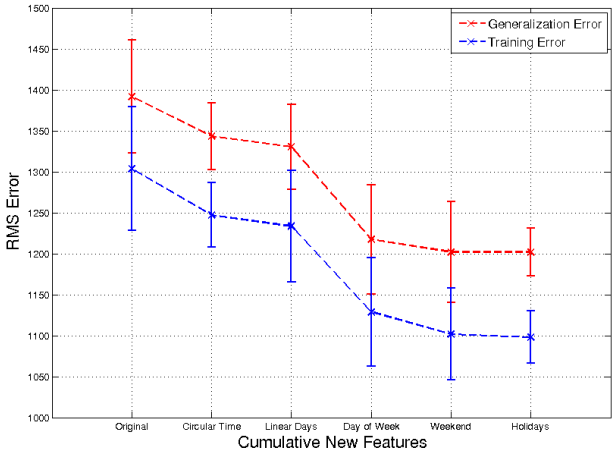



### 5.2 Feature Engineering

While neural networks have the benefit of being able to fit highly non-linear models, modeler insight of how to represent the data can be equally important. To determine the impact of feature engineering, we trained feedforward networks with additional hand engineered features (See Appendix A).

The features were introduced cumulatively, and 10 models were trained for each feature set. Figure 4 shows that human added features can make a significant difference in the overall performance of the model. In particular, circular seasons and hours and the day of the week feature decreased RMS error 3.5% and 8.5%, respectively. That being said, models that can realize more complexity than the mid-sized network (i.e., deeper networks and recurrent networks) have had even better performance, and demonstrate the power of deep learning.

Figure 4: Feature Engineering




## 5.3 Fourier transform Network

To exploit the strong periodicity of the data (see Figure 1) we tried to predict the power loads in frequency space, using Modified Discrete Cosine Transforms (Appendix B). To perform a discrete Fourier transform we first need to fix a time window size, $l$, to split the time series. A natural choice is 24 hours, because the daily pattern of power load variation and temperature are very consistent. We also tried smaller time windows, down to 2 hours.

The transformed dataset is made of samples $(\hat{x}^{(i)}, \hat{y}^{(i)})$ where $\hat{x}^{(i)}$ is the transform of the $i$-th time window of the temperature time series for all 11 zones, plus the date, and $\hat{y}^{(i)}$ is the transform of the $i$-th time window of the power loads time series. This reduces the number of samples by a factor $l$. Each $\hat{x}^{(i)} \in \mathbb{R}^{11\times l+1}$ and each $\hat{y}^{(i)} \in \mathbb{R}^{l}$.

We used a network topology with two hidden layers, the first had the same number of nodes of the input $\hat{x}^{(i)}$ and the second the same of the output $\hat{y}^{(i)}$. To speed up computation and reduce risk of overfitting we applied PCA to $\hat{x}^{(i)}$ and worked on the first 95% of the total variance, significantly reducing dimensionality.

The resulting RMSE of the network for different sizes of the time window of the transform are shown in Figure 5. As we increase the size of the time window we (1) reduce the number of samples in the dataset and (2) increase the dimensionality of each sample by $\sim l$. The complexity of the model thus increases while simultaneously the dataset becomes smaller.

We expected better results from the Fourier transform network model than the non-transformed model, since it should help the network find repeating patterns, however this was not the case. This was in contrast to the more elementary transforms in Figure 4 and Appendix A. Our conclusion was that, except for the most basic transforms, it is better to allow the neural network to find the nonlinear relationships itself.

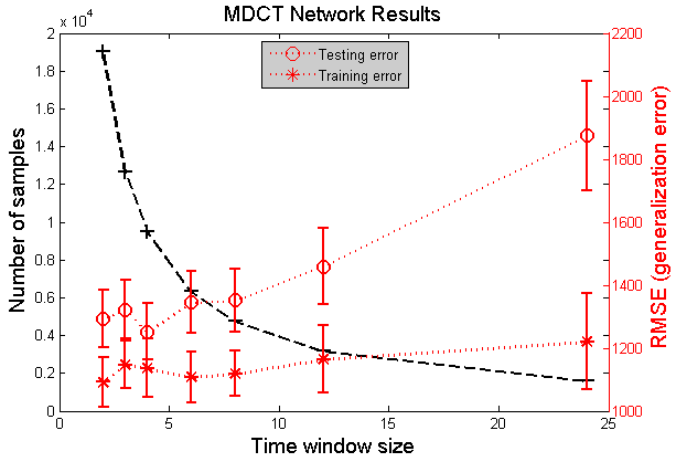



Figure 5: The right axis shows RMSE of the network in transformed space with various sizes of the time window of the transform. The left axis shows the number of samples in the dataset. Training stops when the backpropagation algorithm reaches a minimum or we complete 1000 iterations. Performance is evaluated against 15% test data.

## 5.4 Anomaly Filtering

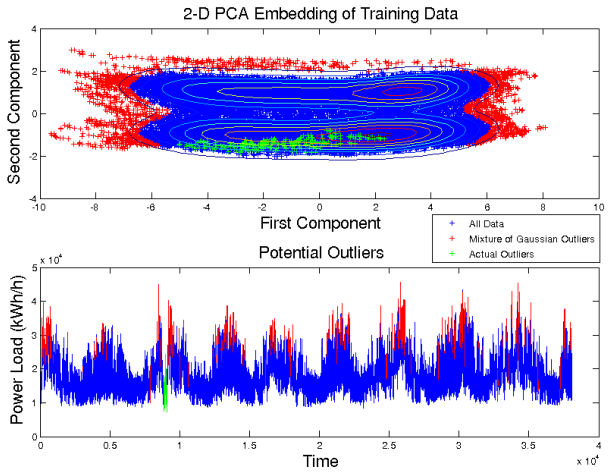
Literature [8] suggests that filtering time series data for anomalies may reduce error for neural networks. We use a mixture of gaussians to estimate the least likely data, and label it as anomalous. We fit ten through 100 multi-variate gaussians to the data in 18-dimensional feature space (from Feature Engineering), and identified the lowest 1%, 5%, and 10% likelihood data with "anomaly features." Neural networks were then trained in this new feature space, however there was no discernible improvement in the generalization error of these models.

Diving deeper into the data, Figure 6 shows the anomalous data points identified by mixture of gaussians in red (embedded in 2-D with PCA) and how it maps to the time series. Mixture of gaussians identified peaks in demand in the summer and winter, which were not particularly anomalous. Gaussian methods also failed to identify known outlier data, such as the huge power outages experienced on January 5, 2005 (marked in green in Figure 6). Although we might hope these green points are flagged as outliers, they were actually probable data points in the gaussian model, as you can see by mapping to the first two principal components. It is possible that alternative methods could provide modeling gains, but we decided this was not worth pursuing.

## 5.5 Training algorithm

Neural networks, and in particular deep neural networks, are notoriously difficult to train due to their non-convex objective functions. We examined five training algorithms to determine the best and fastest solution. Levenberg-Marquardt ("LM", a combined quasi-Newton and gradient ascent algorithm) was

Figure 6: Anomaly Filtering




found to converge fastest and almost always produce the best error. BFGS (quasi-Newton), Scaled Conjugate Gradient, and Conjugate Gradient with Polak-Ribire updates were all slower and had worse error than LM. LM with Bayesian Regularization produced the lowest error with 2-layer networks (though, not with 4), but convergence time was prohibitively long to be useful. Table 2 summarizes RMS error for the different algorithms on different sized networks.

Table 2: Training Algorithms

| | RMSE | |
|---|---|---|
| **Training Algorithm** | **2-Layer** | **4-Layer** |
| Levenberg-Marquardt | 1,392 | 1,263 |
| BFGS Quasi-Newton | 1,506 | 2,602 |
| Bayesian Regulation | 1,290 | 1,887 |
| Scaled Conjugate Gradient | 1,781 | 1,681 |
| Conjugate Gradient | 1,596 | 1,593 |

### 5.6 Recurrent Neural Networks

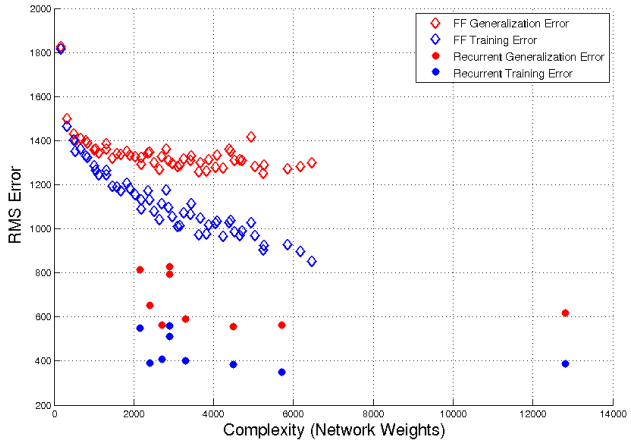
Using a purely feedforward network we ignore some of the temporal structure time series data presents. In particular, this structure can be exploited through the use of recurrent neural networks where we explicitly construct a sequential representation of our data.

Recurrent neural networks may have multiple types of feedback loops. The two that were employed here were input delays and feedback delays. Each type of delay effectively increases the number of input nodes by providing the network with delayed information along with current information. In the case of input delays, multiple consecutive time steps of the input features are presented to the network simultaneously. For feedback delays, the output of the model is provided to input nodes, along with previous data. This can either be done with "open" loops, where the known output is provided as an input, or with "closed" loops, which connects the network output to the input directly. Here, we have trained and forecasted one-step-ahead with with open loops. To predict farther ahead, closed loops need to be used. Briefly experimenting with closed loops, error increased ∼10% due to "clipping" of peak and trough demand.

### 5.7 Network Topology

Performance of the networks varies highly depending on the number of hidden layers and neurons per layer. Figure 7 illustrates network performance as a function of complexity (number of weights). Feedforward network achieve near optimal performance around 2,700 weights (3-layers of 50, 30, and 10 neurons, respectively), with RMS error of 1,269. Additional complexity through more neurons did not improve performance, but deeper networks occasional produced better results (however, performance was erratic due to finding local-non-global optimum).

Figure 7: Topology Selection




Recurrent neural networks performed drastically better for comparable complexity. All recurrent networks shown in Figure 7 are 2-layers – we experimented with deeper networks, however training time increased rapidly (i.e., ≥8 hours per model). Best 2-layer results were achieved with 50 and 25 neurons per layer, respectively, one input tap delay, and one through 48 hours of feedback tap delay (∼4,500 weights).

## 6 Conclusion

Deep neural networks are able to accurately gauge electricity load demand across our grid. It is clear from Table 1 that recurrent neural networks outperformed all of our competing methods by a significant margin. We believe that, in general, deep learning architectures are well suited to the problems of power load forecasting since we have large availability of relevant data with complicated nonlinear underlying processes.

We were disappointed with our output using competing nonlinear kernelized methods, and perhaps using pre-built SVM packages would have produced superior results. Due to their poor scaling properties we did not pursue our initial work in Gaussian process modeling. However, potential for sparse computational approximations might provide interesting future avenues of research [10] [11]. We believe that within deep learning, cutting edge techniques such as restricted boltzmann machines might allow us to train more complex models efficiently.

# Appendix

## A Transformation of Variables

Within our dataset, it was important to frame our feature variables in ways that were conducive to sensible models, and exploiting natural symmetries. For example, each training example was taken from one of 20 distinct geographical zones. Since we had no idea, a priori, of how these zones relate we found better results modeling zone $z \in \mathbb{R}^{20}$ than naively $z \in \{1, 2, .., 20\}$.

On the other hand, much of the observed structure to the data was found to be periodic in nature (daily, weekly, seasonal). With that in mind we took care, particularly for our linear models, to reflect this a priori symmetry in our data. For variables with inherent symmetry present (hours:24, months:12 etc) we transformed to polar coordinates via a (sin,cos) pair of variables to seek solutions of given periodicity.

Figure 8: Transformation of Variables

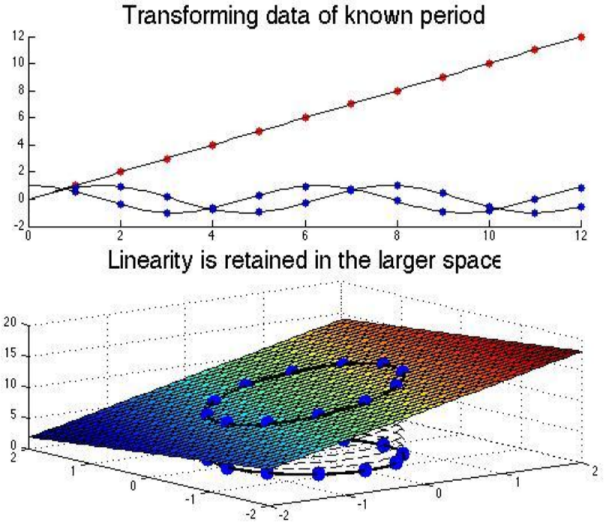



Figure 9: $x \rightarrow [y_1, y_2] = [sin(\frac{2\pi x}{\omega}), cos(\frac{2\pi x}{\omega})]$ we show $x$ as a red dot mapped to $y$ in blue. We can then fit the desired relationship as linear in $\mathbb{R}^2$ for the new circular $y$ features

## B Frequency Space Transform

More generally, we might want to exploit the periodicity in the data (Figure1) when we are not sure about the exact period. For example, we know there are pattens around the working day which might be more easily seen in the transformed Fourier space. We implemented the Modified Discrete Cosine Transform (MDCT) [12] which performs a *discrete* transform over fixed size windows returning real coefficients and well-behaved edges.

The time windows overlap by half of their length. For example in case of 24 hours time windows the first sample starts at 0 and ends at 24 of the first day, the second sample from 12 to 12 of the second day, the third from 0 to 24 of the second day, and so on. The original signal is recovered by summing the overlapping parts of each inverse transform (time-domain aliasing cancellation). It is well known that minimizing error in the frequency space is equivalent to the original problem. We were motivated by mp3 compression, which uses similar techniques, to use this to express our data in a more efficient manner.

## C Stacked Autoencoder

Building on the literature [10] we trained feed forward neural networks with stacked autoencoders. Greedy layer-wise training [5] [6] has been shown to improve network performance in scenarios where large amounts of unlabeled data is available, but minimal labeled data. While all of the data used in this paper are labeled, it was our hope that greedy layer-wise training would help avoid local-non-global minima in the optimization problem. However, results with this technique were discouraging, with no significant performance gain but added computational complexity.

## References


[1] `http://www.eia.gov/totalenergy/data/annual/pdf/sec3_11.pdf`

[2] `http://www.bea.gov/itable/error_NIPA.cfm`

[3] `http://www.kaggle.com/c/global-energy-forecasting-competition-2012-load-forec`

[4] Gaussian Processes for Machine Learning - Rasmussen and Williams

[5] Y. Bengio, "Learning Deep Architectures for AI", *Foundations and Trends in Machine Learning*, vol 2. , no. 1, 2009.

[6] Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In Schölkopf, B., Platt, J., & Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems* 19 (NIPS'06), pp. 153-160. MIT Press.

[7] `http://www.teslaforecast.com/`

[8] J. Connor, R.D. Martin, and L.E. Atlas, "Recurring Neural Networks and Robust Time Series Prediction", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, March 1994.

[9] `http://repository.lib.ncsu.edu/ir/bitstream/1840.16/6457/1/etd.pdf`

[10] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7), 1527-1554.

[11] `http://arxiv.org/pdf/1106.5779v1.pdf`.

[12] Malvar, Henrique S. Signal processing with lapped transforms. Artech House, Inc., 1992.