Crystal Mosley
MSDS 458 Sec 55

Final Project

"How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption"

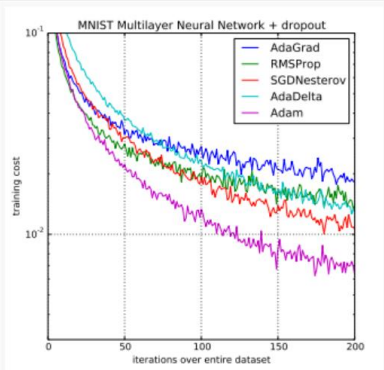
Introduction

Long short-term memory (LSTM) is a type of recurrent neural network (RNN) architecture that was developed to help deal with the exploding and vanishing gradient issues that occur when training and RNN. RNN's are designed to learn and predict on sequence data. For an RNN to make a prediction, it needs information from previous data – this is called long-term dependency because of the distance between the applicable information and the point where it needs to make the prediction. Because of this widening distance, RNN's have a hard time learning these dependencies due to either exploding or vanishing gradients. "These problems arise during training of a deep network when the gradients are being propagated back in time all the way to the initial layer. The gradients coming from the deeper layers [must] go through continuous matrix multiplications because of the chain rule, and as they approach the earlier layers, if they have small values (<1), they shrink exponentially until they vanish and make it impossible for the model to learn, this is the *vanishing gradient problem*. While on the other hand if they have large values (>1) they get larger and eventually blow up and crash the model, this is the *exploding gradient problem*", (Alese). The following experiment will entail developing an LSTM RNN to model and forecast future electricity consumption by answering the question: "Given recent power consumption, what is the expected power consumption for the week ahead?". This tutorial, provided by Jason Brownlee, goes through multivariate time series data and models the following neural network (NN) types:

- Univariate and multivariate Encoder-Decoder LSTM's for multi-step time series forecasting
- CNN-LSTM Encoder-Decoder model for multi-step time series forecasting
- ConvLSTM Encoder-Decoder model for multi-step time series forecasting

All models are evaluated using the *walk-forward validation* method. This method requires the model to make a "one-week prediction [first], then the actual data for that week is made available to the model so it is used as the basis for making a prediction on the subsequent week", (Brownlee). One thing to note is the use of the Adam optimizer. This specific optimizer has shown great effects in deep learning because it can achieve good and fast results. Adam was applied to a logistic regression algorithm, MLP algorithm, and a CNN – the conclusions demonstrated that Adam efficiently solved each problem better than AdaGrad, RMSProp, SGDNesterov and AdaDelta (see **Figure 1**). The goal of the RMSE is to be under 465 kilowatts across a seven-day forecast. The author of the tutorial explored each model using activation = relu, epochs = 70, batch_size = 16, and n_input = 7. I compare his results against the performance tweaks of mine to see how each affect the results of the time series data.

Figure 1: Results of Optimizers against NN's



Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron
Taken from Adam: A Method for Stochastic Optimization, 2015.

Benefits of Adam: (Ba, Kingma)

- Straightforward to implement
- Computationally efficient
- Little memory requirements
- Invariant to diagonal rescale of the gradients
- Well suited for problems that are large in terms of data and/or parameters
- Appropriate for non-stationary objectives
- Appropriate for problems with very noisy/or sparse gradients
- Hyper-parameters have intuitive interpretation and typically require little tuning

Univariate Multi-Step LSTM

Initial tests: epochs, batch_size, n_input = 70, 16, 7
No activation function.

- Results – lstm: [487.153] 477.7, 476.3, 462.7, 506.5, 473.2, 419.9, 579.0
  - Over the 465 kilowatt goal.

Initial tests: epochs, batch_size, n_input = 70, 16, 14
No activation function.

- Results – lstm: [450.600] 420.5, 477.8, 439.6, 453.7, 429.1, 390.4, 529.7

Initial tests show higher than the goal, or just under the goal of 465 kilowatts without an activation function being utilized.

---

epochs, batch_size = 70, 16

Figure 2: n_input = 7 days

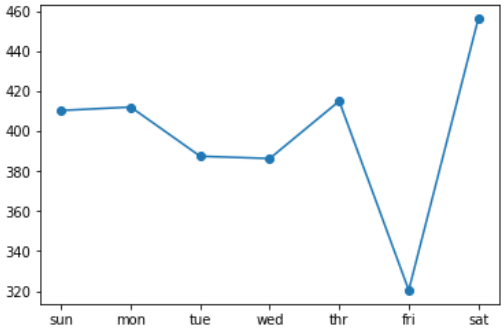
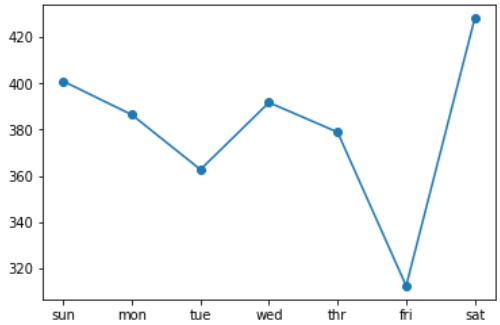


Figure 3: n_input = 14 days



Results Figure 2 – lstm: [400.084] 410.3, 412.0, 387.4, 386.3, 415.0, 320.5, 456.1
Results Figure 3 – lstm: [381.650] 400.9, 386.4, 362.8, 391.7, 378.9, 312.5, 428.0

epochs, batch_size = 140, 16

Figure 4: n_input = 7 days

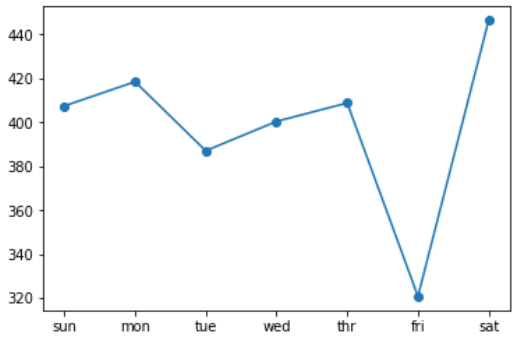
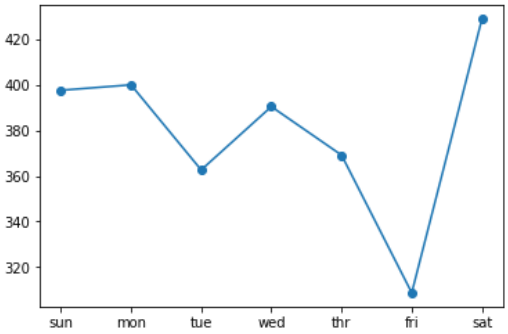


Figure 5: n_input = 14 days



Results Figure 4 - lstm: [397.259] 403.4, 423.6, 376.8, 394.3, 402.5, 312.5, 453.2
Results Figure 5 - lstm: [381.342] 397.7, 400.1, 362.7, 390.5, 369.2, 308.7, 429.0

epochs, batch_size = 140, 32

Figure 6: n_input = 7 days

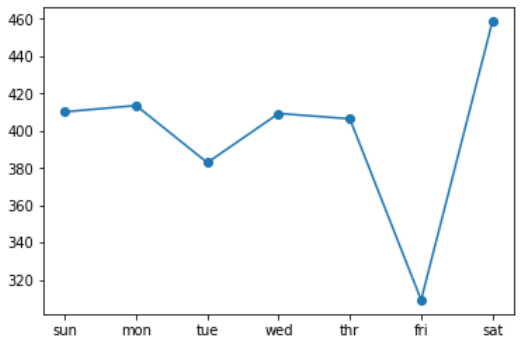
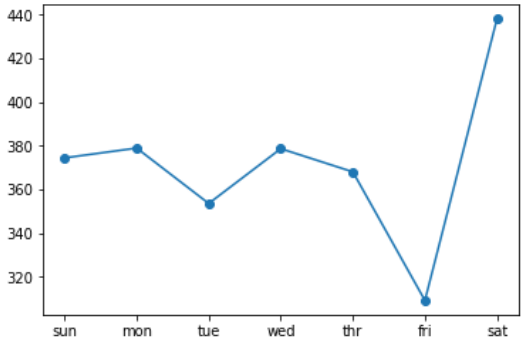


Figure 7: n_input = 14 days



Results Figure 6 - lstm: [400.803] 410.2, 413.6, 382.9, 409.3, 406.4, 309.2, 458.7
Results Figure 7 - lstm: [373.247] 374.5, 378.9, 353.6, 378.7, 368.0, 309.2, 438.1

Epochs, batch_size, n_input = 140, 16, 7 show a sweet spot in **Figure 4**; however, the best model resulted with epochs, batch_size, n_input = 140, 32, 14 (**Figure 7**). This is showing that doubling the parameters and the amount of days to forecast gives the best RMSE result.

Experimenting with *softmax* vs. *relu* showed bad results – almost quadruple the *relu* results; therefore, *relu* activation function is the only function I used for the rest of the experiments. The author did not utilize a neurons parameter, so I added this parameter back to the original parameter set (epochs, batch_size = 70, 16) after these initial 10 experiments and the results showed quite a difference, and almost beat out the best model that had double the parameter values:

- Results with n_input = 7, neurons = 4 – lstm: [395.387] 407.0, 414.9, 379.2, 388.8, 401.5, 316.5, 447.4
- Results with n_input = 14, neurons = 4 – lstm: [379.281] 379.7, 377.2, 353.9, 376.4, 399.7, 349.4, 414.5

Since the results showed better performance, I ran the same experiments of changing parameters to see how the results would differ:

- epochs, batch_size, neurons, n_input = 140, 16, 4, 7
  - lstm: [403.465] 405.8, 431.3, 391.0, 393.2, 405.2, 309.0, 470.8
- epochs, batch_size, neurons, n_input = 140, 16, 4, 14
  - lstm: [379.941] 395.5, 380.7, 370.0, 385.6, 377.3, 317.9, 424.5

- epochs, batch_size, neurons, n_input = 140, 32, 4, 7

    - lstm: [403.612] 419.2, 414.5, 397.5, 400.0, 402.8, 316.4, 460.9
- epochs, batch_size, neurons, n_input = 140, 32, 4, 14
    - lstm: [371.643] 369.4, 379.1, 359.2, 376.9, 373.9, 304.3, 428.0

Univariate Multi-Step Encoder-Decoder LSTM

This next model adds onto the previous simple LSTM and adds an encoder-decoder. This model will be comprised of two sub models, the encoder to read and encode the input sequence, and the decoder that will read the encoded input sequence and make a one-step prediction for each element in the output sequence.

epochs, batch_size, n_input = 20, 16, 7
- Results – lstm: [389.487] 391.9, 414.6, 355.3, 376.2, 406.4, 332.0, 439.6

epochs, batch_size, n_input = 20, 16, 14
- Results – lstm: [392.121] 372.9, 403.9, 342.4, 401.9, 371.9, 321.8, 503.6

epochs, batch_size, n_input = 40, 16, 7
- Results – lstm: [386.953] 384.6, 416.1, 351.5, 375.7, 398.0, 319.2, 449.5

epochs, batch_size, n_input = 40, 16, 14
- Results – lstm: [373.969] 388.3, 380.8, 338.8, 377.8, 363.7, 290.3, 457.3

epochs, batch_size, n_input = 40, 32, 7
- Results – lstm: [395.603] 394.7, 426.7, 366.8, 369.5, 420.3, 346.4, 435.8

epochs, batch_size, n_input = 40, 32, 14
- Results – lstm: [368.007] 366.8, 377.0, 324.9, 370.2, 356.6, 311.0, 452.6

epochs, batch_size, n_input, neurons = 20, 16, 7, 4
- Results - lstm: [404.760] 398.5, 414.6, 351.7, 396.7, 401.9, 324.3, 518.0

epochs, batch_size, n_input, neurons = 20, 16, 14, 4
- Results – lstm: [385.421] 377.9, 413.3, 361.6, 370.3, 393.5, 350.5, 425.1

epochs, batch_size, n_input, neurons = 40, 16, 7, 4
- Results – lstm: [389.445] 381.9, 411.4, 357.7, 374.3, 407.6, 329.4, 451.5

epochs, batch_size, n_input, neurons = 40, 16, 14, 4
- Results – lstm: [367.550] 387.2, 364.9, 331.3, 368.1, 375.0, 289.2, 439.3

epochs, batch_size, n_input, neurons = 40, 32, 7, 4
- Results – lstm: [390.997] 392.6, 416.8, 357.6, 380.3, 398.3, 319.8, 456.9

epochs, batch_size, n_input, neurons = 40, 32, 14, 4
- Results – lstm: [379.900] 383.6, 397.8, 349.3, 372.3, 395.4, 333.6, 420.3

Multivariate Multi-Step Encoder-Decoder LSTM

Next, the previous Encoder-Decoder LSTM model will be updated to use each of the eight time series variables to predict the next standard week of daily total power consumption. Each 1D (one-dimensional) time series will be modeled as a separate sequence of input. This will allow the LSTM to create an internal representation of each input sequence that will be interpreted by the decoder together.

epochs, batch_size, n_input = 50, 16, 7
- Results – lstm: [457.246] 470.8, 527.2, 454.7, 445.1, 433.9, 383.4, 473.1

epochs, batch_size, n_input = 50, 16, 14
- Results – lstm: [386.105] 372.2, 419.7, 349.5, 385.3, 386.4, 342.5, 437.7

epochs, batch_size, n_input = 100, 16, 7
- Results – lstm: [384.610] 389.1, 398.1, 340.8, 397.3, 368.2, 285.2, 484.5

epochs, batch_size, n_input = 100, 16, 14
- Results – lstm: [1594.227] 1644.6, 1536.8, 1546.1, 1630.9, 1532.0, 1512.4, 1743.5

epochs, batch_size, n_input = 100, 32, 7
- Results – lstm: [429.832] 360.7, 427.8, 373.0, 418.4, 485.8, 461.9, 465.4

epochs, batch_size, n_input = 100, 32, 14

- Results – lstm: [385.926] 376.9, 388.5, 340.8, 388.2, 368.8, 297.5, 508.1

epochs, batch_size, n_input, neurons = 50, 16, 7, 4
- Results – lstm: [501.823] 419.2, 421.7, 402.7, 440.6, 489.6, 553.7, 711.8

epochs, batch_size, n_input, neurons = 50, 16, 14, 4
- Results – lstm: [464.841] 428.5, 422.5, 407.5, 454.0, 489.4, 500.9, 536.5

epochs, batch_size, n_input, neurons = 100, 16, 7, 4
- Results – lstm: [498.828] 406.4, 400.5, 376.0, 484.0, 452.8, 461.4, 789.2

epochs, batch_size, n_input, neurons = 100, 16, 14, 4
- Results – lstm: [523.095] 507.0, 613.9, 433.2, 486.5, 447.3, 383.7, 714.0

epochs, batch_size, n_input, neurons = 100, 32, 7, 4
- Results – lstm: [385.870] 385.0, 399.9, 345.3, 391.5, 371.9, 293.2, 487.2

epochs, batch_size, n_input, neurons = 100, 32, 14, 4
- Results – lstm: [581.978] 541.7, 477.9, 489.7, 604.5, 519.3, 517.1, 840.7

Univariate Multi-Step Encoder-Decoder CNN-LSTM

The next model set will demonstrate the following: “A convolutional neural network, or CNN, can be used as the encoder in an encoder-decoder architecture. The CNN does not directly support sequence input; instead, a 1D CNN is capable of reading across sequence input and automatically learning the salient features. These can then be interpreted by an LSTM decoder as per normal. We refer to hybrid models that use a CNN and LSTM as CNN-LSTM models, and in this case we are using them together in an encoder-decoder architecture”, (Brownlee).

epochs, batch_size, n_input = 20, 16, 7
- Results – lstm: [401.731] 414.1, 415.3, 355.9, 381.3, 431.5, 357.0, 447.2

epochs, batch_size, n_input = 20, 16, 14
- Results – lstm: [392.966] 401.9, 398.3, 350.8, 398.8, 373.5, 319.1, 487.3

epochs, batch_size, n_input = 40, 16, 7
- Results – lstm: [407.859] 423.0, 449.6, 370.0, 379.8, 425.3, 355.9, 441.1

epochs, batch_size, n_input = 40, 16, 14
- Results – lstm: [396.324] 415.1, 397.1, 362.4, 391.7, 377.2, 334.4, 480.3

epochs, batch_size, n_input = 40, 32, 7
- Results – lstm: [401.829] 422.0, 408.1, 359.5, 385.5, 395.0, 315.5, 502.2

epochs, batch_size, n_input = 40, 32, 14
- Results – lstm: [390.131] 397.5, 415.6, 367.6, 378.3, 371.4, 311.0, 470.9

epochs, batch_size, n_input, neurons = 20, 16, 7, 4
- Results – lstm: [397.474] 423.9, 404.5, 348.0, 377.9, 399.4, 317.2, 488.5

epochs, batch_size, n_input, neurons = 20, 16, 14, 4
- Results – lstm: [370.404] 375.9, 378.0, 328.3, 370.7, 366.4, 320.3, 440.6

epochs, batch_size, n_input, neurons = 40, 16, 7, 4
- Results – lstm: [404.840] 424.1, 411.0, 343.8, 406.1, 400.4, 320.4, 502.4

epochs, batch_size, n_input, neurons = 40, 16, 14, 4
- Results – lstm: [381.769] 409.3, 405.2, 342.4, 351.6, 386.9, 340.2, 426.9

epochs, batch_size, n_input, neurons = 40, 32, 7, 4
- Results - lstm: [398.078] 416.4, 405.7, 337.4, 376.0, 404.2, 323.8, 497.9

epochs, batch_size, n_input, neurons = 40, 32, 14, 4
- Results – lstm: [401.445] 426.0, 402.1, 370.5, 393.9, 380.1, 320.9, 495.0

Univariate Multi-Step Encoder-Decoder ConvLSTM

The last model set will incorporate a convolutional LSTM – a further extension of the CNN-LSTM model. "This combination is called a Convolutional LSTM, or ConvLSTM for short, and like the CNN-LSTM is also used for spatio-temporal data. Unlike an LSTM that reads the data in directly in order to calculate internal state and state transitions, and unlike the CNN-LSTM that is interpreting the output from CNN models, the ConvLSTM is using convolutions directly as part of reading input into the LSTM unit's themselves", (Brownlee).

epochs, batch_size, n_input = 20, 16, 7

- Results – lstm: [386.107] 405.4, 413.3, 342.9, 376.5, 391.8, 339.1, 424.9

epochs, batch_size, n_input = 20, 16, 14

- Results – lstm: [367.429] 370.0, 404.0, 339.3, 359.4, 365.4, 333.1, 395.2

epochs, batch_size, n_input = 40, 16, 7

- Results - lstm: [387.594] 384.6, 417.4, 375.1, 381.3, 385.6, 334.5, 427.6

epochs, batch_size, n_input = 40, 16, 14

- Results - lstm: [377.914] 371.0, 407.8, 343.2, 380.1, 341.5, 312.9, 468.0

epochs, batch_size, n_input = 40, 32, 7

- Results - lstm: [377.649] 369.3, 417.0, 360.3, 386.4, 384.1, 319.2, 399.4

epochs, batch_size, n_input = 40, 32, 14

- Results - lstm: [381.928] 407.6, 383.4, 353.4, 379.3, 366.5, 329.4, 443.2

epochs, batch_size, n_input, neurons = 20, 16, 7, 4

- Results - lstm: [429.203] 388.1, 464.6, 397.8, 409.6, 455.0, 437.6, 445.5

epochs, batch_size, n_input, neurons = 20, 16, 14, 4

- Results – lstm: [384.795] 389.8, 394.5, 371.0, 402.6, 363.5, 332.3, 432.0

epochs, batch_size, n_input, neurons = 40, 16, 7, 4

- Results – lstm: [413.012] 385.6, 481.5, 396.5, 421.3, 378.0, 333.8, 473.7

epochs, batch_size, n_input, neurons = 40, 16, 14, 4

- Results - lstm: [391.153] 397.4, 435.9, 377.2, 403.3, 353.6, 333.4, 426.5

epochs, batch_size, n_input, neurons = 40, 32, 7, 4

- Results – lstm: [403.707] 407.6, 438.1, 377.5, 398.4, 368.5, 330.5, 486.4

epochs, batch_size, n_input, neurons = 40, 32, 14, 4

- Results – lstm: [387.129] 403.1, 394.1, 359.4, 395.1, 370.7, 324.2, 451.1

Conclusion

Out of all the models, all results were well below the goal of 465 kilowatts except for the Multivariate Multi-Step Encoder-Decoder LSTM models. Four of the 12 different models within this model class showed more than 500 kilowatts – with one being over 1500; these are deemed insignificant since they were over the value from the naïve bayes model. The model with the best performance overall came from the Univariate Multi-Step Encoder-Decoder ConvLSTM model with parameters of *epochs, batch_size, n_input = 20, 16, 14*. The parameters that showed the most change in performance were when the epochs and neurons were added. Depending on what type of model is being ran, the parameters show different effects – there's no trend between all models and parameters; this is because of how the encoders and decoders are evaluating each model and interpreting the results to make their predictions.

References

Alese, Eniola. "The Curious Case of the Vanishing & Exploding Gradient." Medium, Learn.Love.AI, 5 June 2018, medium.com/learn-love-ai/the-curious-case-of-the-vanishing-exploding-gradient-bf58ec6822eb.

Brownlee, Jason. “How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption.” Machine Learning Mastery, 2 Mar. 2019, machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/.

Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” ArXiv.org, 30 Jan. 2017, arxiv.org/abs/1412.6980.