A Window-Based Approach to Training Deep Neural Networks for Predictive Sequence Modeling

Kirori, Zachary &Wasike, Jotham
Kirinyaga University, Kenya
Correspondence: *zkirori@kyu.ac.ke*

Abstract

Deep machine learning potentially holds the key to unlocking the door to modern applied computational intelligence. Presently, it is becoming progressively possible to process great amounts of data whether static or arriving in streams of varying velocities using deep learning models. Applications are innumerably many ranging from time series data modeling, signal processing, image analysis, natural language processing to object recognition among others. The critical area of predictive data modeling requires efficient and carefully selected algorithms and models for effective and accurate predictions. In this paper, we present a novel deep machine learning Neural Network for predictive tasks based on a fixed size window of time steps, tested on a well-known dataset on customer arrivals to an airline. At the core of the architecture is a Multi-Layer Perceptron – a classical deep learning Neural Network optimized on a number of dimensions that include the training algorithm, batch size, number of iterations, and the loss function among others. We present experimental results and conclude that, upon tuning and optimization, classical deep learning neural networks such as the Multi -Layer Perceptron (MLP) have comparable predictive abilities compared to advanced neural networks such as the Recurrent Neural Networks (RNN) and Convolution Neural Network (CNN).

Keywords: Deep Learning, Predictive Sequence Modeling, Time Series Data Analysis, Multi-Layer Perceptron, Deep Learning Optimization, Fixed Window Method

Introduction

The sub-field of Deep Machine Learning (DML) in the larger field of Artificial Intelligence (AI) undoubtedly holds the key to solving some of the classical computational problems in natural language understanding, image analysis, signal processing, computer vision, navigation; tasks that traditionally have been considered difficult. It has been made progressively possible to train larger and larger computational models within short times, with ease and using minimal computational resources of memory and bandwidth (Bengio, 2012).

Rooted in computational statistics and relying heavily on the efficiency of numerical algorithms and Deep Neural Networks (DNNs), DML techniques capitalize on the world's increasingly powerful computing platforms and the availability of datasets of immense size to analyze and give solutions to problems where recommendation approaches fail (Schmidhuber, 2015; LeCun, et al, 2015).

DML is primarily an optimization procedure which, in this context, involves numerical computation of parameters for a system designed to make optimal decisions based on yet unseen data by choosing parameters that exhibit the best values with respect to a given learning problem. (Bergstra and Bengio Y., 2012; Bottou, et el, 2017; Vankadara,2015)

Typically, these tasks are characterized by large amounts of training data, high dimensionality, ill-conditioning that require extensively many cycles of computing power (Chaudhuri & Ghosh, 2016).

Literature Review

Most work using ANN to manipulate Time-Series data focus on modeling and forecasting. This section reports on a selected number especially in regression modeling of time series data to rightfully place this work in context.

(a) Regression with Time Steps

Some sequence problems may have a varied number of time steps per sample. For example, we may have measurements of a physical machine leading up to a point of failure or a point of surge (Venkatraman, 2017; Patel, Chaudhary & Garg, 2016). Each incident would be a sample and the observations that lead up to the event would be the time steps, while the variables observed would be the features (Gamboa John. 2017).

Time steps provide one way to phrase time series problems. Instead of phrasing the past observations as separate input features, we can use them as time steps of the one input feature. This technique is especially required by most stateful Neural Networks such as the Recurrent Neural Network (CNN), the Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU) (Olof M., 2016).

(b) Regression with Memory between Batches

The stateful networks have memory, giving them ability to remember across long sequences. Typically, the state within the network is reset after each training batch when fitting the model, as well as during prediction and evaluation. This means that the network can build state over the entire training sequence and even maintain that state if needed to make predictions providing us with finer control over the internal state (Talagala, Hyndman & Athanasopoulos, 2018).

One of the requirements for this approach is that the training data should not be shuffled when fitting the network. It also requires explicit resetting of the network state after each exposure to the training data (epoch). Finally, when the network layer is constructed, the stateful parameter must be set true and instead of specifying the input dimensions, we must hard code the number of samples in a batch, number of time steps in a sample and number of features in a time step by setting the batch input shape parameter (Bao, et el, 2017).

(c) Window Method

In the window method, a time series problem is framed so that a selected number of recent time steps are used to make the prediction for the next time step. In this case the size of the window is a parameter that is often tuned for each problem (Busseti, Osband & Wong, 2012; Bontempi, 2013).

For instance, given the current time (t) we may desire to predict the value at the next time in the sequence i.e. (t + 1), by relying on the current time (t) as well as a selected number previous time steps, say (t-1, t-2, …,t-N) for an N-size window. Phrased as a regression problem the input variables would be t-N,.., t-2, t-1, t and the output variable would be t+1. (Chaudhuri, Ghosh, 2016).

(d) Optimization for Regression Modeling

Optimization problems in machine learning arise through the definition of prediction and loss of functions that appear in measures of expected and empirical risk that one aims to minimize. There are two varieties of optimization problems that arise in machine learning: the first involves convex optimization problems, derived from use of logistic regression or support vector machines, while the second typically involves highly complex and problems with non-convex error functions, derived from use of deep neural networks. Deep Neural Networks are trained using the Back propagation especially the Back Propagation Through Time (BPTT) which is numerically formulated as a highly non-convex optimization problem in a very high dimensional feature space. Algorithm (Ngiam, Coates, Lahiri, Prochnow, Le Q., & Ng, 2011, Bottou, Curtis, & Nocedal, 2017) .

However, the training process requires extreme skill and care. For instance, it is crucial to initialize the optimization process with a good starting point through parameter tuning and to monitor its progress while correcting conditioning issues as they appear

(Bergstra. & Bengio, 2012). A great deal of these successes lie in the choice, regularization of the training algorithm as well as the domain of application.

Unfortunately, attempts to optimize these models such as increasing model size and training data - which is necessary for good prediction accuracy on complex tasks, requires significant amount of computing cycles proportional to the product of model size and training data volume. Due to the computational requirements of deep learning almost all deep models are trained on Graphic Processing Units (GPUs) (Nikhil, et el, 2016). According to Schmidhuber, (2015), the tremendous success of Deep Neural Networks (DNNs), in a wide range of practically relevant applications has triggered a race to build larger and larger DNNs (Simonyan & Zisserman, 2014), which need to be trained with more and more data, to solve learning problems in fast extending fields of applications.

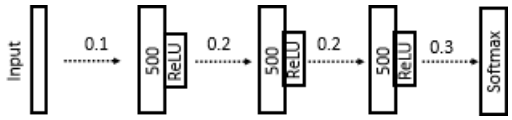
Optimization methods for machine learning fall into two broad categories namely First Order (1^{st} Order) and Second Order (2^{nd} Order). Of the 1^{st} Order methods, the stochastic and batch techniques are key. The prototypical stochastic optimization method is the Stochastic Gradient Method (SGD) where the target value is chosen randomly from a set of target values [1..N] in a positive step-size (Parker, 2012; Lee, et el, 2011; Josh, et al. 2016). Each iteration of this method is thus very cheap, involving only the computation of the gradient corresponding to one sample. Similarly, due to the sum structure of the empirical risk, a batch method can easily benefit from parallelization since the bulk of the computation lies in evaluations of empirical risk and its gradient. Further, calculations of these quantities can even be done in a distributed manner.

Use of the Multi-Layer Perceptron

The experiments were conducted using a fully connected Multi Layer Percentron (MLP) of three (3) input layers optimized using dropout at each layer's input to improve the generalization capability and its potential non- linearity addressed by the rectified linear activation unit (ReLU). The latter has the effect of preventing saturation of the gradient when the network becomes very deep. (Raudys, Mockus, 1999) The last layer of the network uses a softmax function whose basic layer block is formalized as

$$x' = f_{dropout.p}(x)$$
$$y = W.x' + b$$
$$h = ReLU(y)$$

ReLU helps to stack the networks deeper and dropout largely prevent the co-adaption of the neurons to help the model generalize well especially on some small datasets. However, if the network is too deep, most neuron will hibernate as the ReLU totally halve the negative part. The dropout rates at the input layer, hidden layers and the softmax layer were varied as {0.1, 0.2, 0.3}, respectively as seen in figure 1 below



Experiment & Results

In this section we present the experimental setup that includes the problem definition, the dataset, the DML platform of choice, the results as well as their comparative analysis.

(a) Problem Definition

The DML problem selected for this study is a typical regression scenario of time series data representing the number of airline passengers arriving at an international airport.

This is a prediction problem where given a year and a month, the task is to predict the number of international airline passengers in units of 1,000 collected over a period of 144 months.

The time series prediction is phrased as a regression problem where given the number of passengers (in units of thousands) this month, last month and previous months, what is the number of passengers next month.

The initial pre-processing step is to convert the given dataset into the required window of several months in the past. For purposes of the experiment, a window of three (3) months was selected as it was found to optimize the results. In this regard, the first column contains two months' (t-3) passenger count before the current month. Subsequently, t-2, t-1 and t represent the remaining window period up to the present month. The next month's (t+1) passenger count, is the target prediction.

(b) Dataset

The dataset is available for free from the Data Market webpage as a .CSV downloadable file with the filename "*international-airline-passengers.csv*".

(c) Deep Learning Platform

the selected development platform consisted of a set of Python DML libraries and frameworks available for the experiment under the permissive MIT license namely: Keras and Tensor flow. Tensor flow is one of the two numerical backend platforms in Python that provide the basis for Deep Learning research and development.

Keras runs on Python 2.7 or 3.5 and can seamlessly execute on GPUs and CPUs based on available hardware the underlying frameworks.

Experiment

The time series problem was phrased as a regression problem with a window size of three (3) recent time steps that were used to make the prediction for the next time step given the current time step. In this case the input variables are t-3, t-2, t-1, t and the output variable is t+1.

The code below was used import all of the functions and classes used to model this problem in the Science Python (SciPy) environment within the Keras deep learning library.

```
# Multilayer Perceptron to Predict International Airline Passengers (t+1, given t, t-1, t-2)
import numpy
import matplotlib.pyplot as plt import matplotlib.pyplot as plt1
import matplotlib.pyplot as plt2 import matplotlib.pyplot as plt4
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense #Default MLP Neural Network
```

With time series data, the sequence of values is important. The method that was used for purposes of stratified cross validation was to split the ordered dataset into train and test datasets. This was necessary in order get an idea of the skill of the model on new unseen data. The code below was used to calculate the index of the split point and separates the data into the training dataset with two thirds (2/3) or roughly 67% of the available observations used to train the model, leaving the remaining a third (1/3) or roughly 33% for testing the model.

```
# split into train and test sets size_Train = int(len(dataset) * 0.67)
size_Test = len(dataset) - size_Train
train_Set, test_Set = dataset[0:size_Train,:],
dataset[size_Train:len(dataset),:]
```

Next a function to create a new dataset was defined in accordance with the window size as described above. The function takes two arguments, the dataset which is a Python Number array that we want to convert into a dataset and the look back which is the number of previous time steps to use as input variables to predict the next time period.

This has the role to create a dataset where X is the number of passengers at a given time (t) and Y is the number of passengers at the next time (t + 1). The value of the look back argument was set to three (3) to conform to the selected window size.

A sample of the dataset with this formulation looks as follows:

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1): inputX, OutputY = [], []
    for i in range(len(dataset)-look_back-1): a = dataset[i:(i+look_back), 0]
    inputX.append(a) OutputY.append(dataset[i + look_back, 0])
return numpy.array(inputX), numpy.array(OutputY)
```

This function was applied to reshape the datasets by overriding the default look back value with the window size as below.

```
# reshape dataset
    look_back = 3
    trainX, trainY = create_dataset(train_Set, look_back)
    testX, testY = create_dataset(test_Set, look_back)
```

The effect of this function on the first few rows of the dataset are seen in table 1 below.

Table 1: Reshaped Dataset

S/N	X3	X2	X1	X	Y

o					
1	112	118	132	129	121
2	118	132	129	121	135
3	132	129	121	135	148
4	129	121	135	148	148

Source: (Authors)

Comparing these first 4 rows to the original dataset sample listed in the previous section, the X=t and Y=t+1 pattern in the numbers is clearly visible.

The parameters that were found to optimize the network capacity are a hidden layer of 14 neurons, a second hidden layer of 8 neurons, 1 output layer of neurons, 300 epochs, a batch size of size 2.

```
# create and fit Multilayer Perceptron model
    model = Sequential()
    model.add(Dense(8, input_dim=look_back, activation='relu'))
    model.add(Dense(1)) model.compile(loss='mean_squared_error',
    optimizer='adam')
    model.fit(trainX, trainY, epochs=300, batch_size=2, verbose=2)
```

Once the model is fitted, the subsequent activity is to estimate its performance on the train and test datasets. The role of this is to provide a point of reference when comparing new models. The technique applied was the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE) as illustrated in the code below.

```
# Estimate model performance
    trainScore = model.evaluate(trainX, trainY, verbose=0)
```

```
percTr = (1000.0-(math.sqrt(trainScore)))/10 print('Train Score: %.2f MSE
(%.2f RMSE) Accuracy %.2f %%' % (trainScore, math.sqrt(trainScore),
percTr))
```

Finally, predictions were generated using the model for both the train and test datasets to get a visual indication of the skill of the model. Once prepared, the results were plotted, showing the original dataset in figure 2, predictions on training and test sets in figure 3 and figure 4 respectively. Subsequently the combined data is shown in figure 5 below. The code that produces these statistical results is shown below

```
testScore = model.evaluate(testX, testY, verbose=0) percTs = (1000.0-
(math.sqrt(testScore)))/10 print('Test Score: %.2f MSE (%.2f RMSE) Accuracy
%.2f %%'%(testScore, math.sqrt(testScore), percTs))

# generate predictions for training trainPredict =
model.predict(trainX) testPredict = model.predict(testX)

# shift train predictions for plotting trainPredictPlot =
numpy.empty_like(dataset) trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_b ack, :] = trainPredict

# shift test predictions for plotting testPredictPlot =
numpy.empty_like(dataset) testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:l en(dataset)-1, :] =
testPredict

# plot Combined Graphs
plt4.plot(dataset, color="green",
label="dataset") plt4.plot(trainPredictPlot, color="blue", label="train")
plt4.plot(testPredictPlot, label="test", color="red")
```

```
plt4.title("Combined Graphs", color="magenta") plt4.xlabel("months",
color="blue") plt4.ylabel("passengers in '000s",
color="green") plt4.legend()
plt4.show()
```

Results and Analysis

The extract of five rows of summary statistics based on the 300 epochs as well as the MSE and RMSE values are indicated below:

Epoch 296/300
92/92 [========- 0s 544us/step - loss:
500.0288 Epoch 297/300
92/92 [========- 0s 544us/step - loss: 533.4240 Epoch 298/300
92/92 [======= 0s 544us/step - loss: 504.7706 Epoch 299/300
92/92 [======== 0s 489us/step - loss: 505.2075 Epoch 300/300
92/92 [======== 0s 544us/step - loss: 498.6430

Train Score: 487.92 MSE (22.09 RMSE) Acc: 97.79 %
Test Score: 2135.39 MSE (46.21 RMSE) Acc: 95.38 %

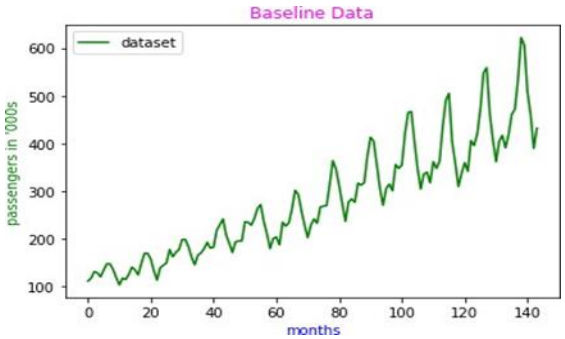


Figure 2: Baseline Data (Source: Author)

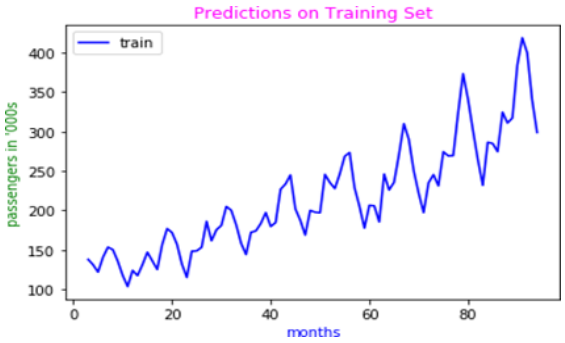


Figure 3: Training Set Predictions (Source: Author)

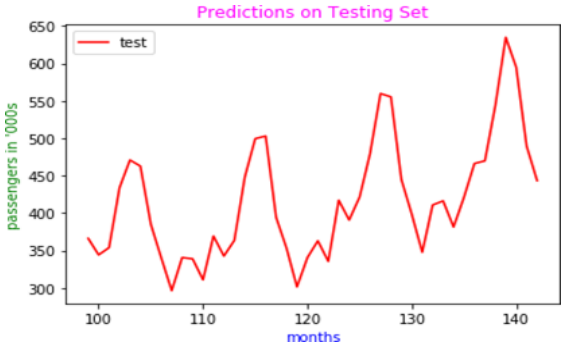


Figure 4: Testing Set Prediction (Source: Author)

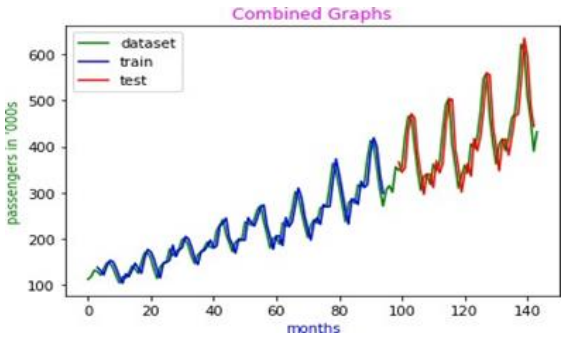


Figure 5: Combined Predictions (Source: Author)

Conclusion

In this research, the study has demonstrated the applicability of a typical neural network with parameter tuning to the problem of sequence data modeling. It is clear from the results above that it performs well on both the training and the testing datasets with minimal error rates. It is therefore imperative to conclude that basic neural networks

such as MLP can perform equally as well as advanced neural networks such as RNNs, CNNs, Boltzman Machines among others with careful fine tuning, optimization, parameter search and pre- processing steps.

Recommendations

This study may be improved by 1) comparative analysis with other neural models 2) experimentation with different training algorithms beyond adaptive moment (ADAM) 3) automatic hyper-parameter search using grid search procedures 4) experimentation with different window sizes and 5) considering additional tests on varied time variant data sets.

References

Bao, W, Yue, J, & Rao, Y. (2017). A Deep Learning Framework for Financial Time Series Using Stacked Autoencoders and Long- Short Term Memory. *PLoS ONE* 12(7): e0180944. https://doi.org/10.1371/journal.pone.0180944

Bengio, Y., (2012). Practical Recommendations for Gradient- Based Training of Deep Architectures, arXiv:1206.5533v2

Bergstra, J. & Bengio, Y. (2012). Random Search for Hyper- Parameter Optimization, *Journal of Machine Learning Research*, pp. 281-305

Bottou, L., Curtis, F. E., & Nocedal, J. (2017). Optimization Methods for Large-Scale Machine Learning, arXiv:1606.04838v2 [stat.ML]

Cristian, J & Gamboa, B.(2017). Deep Learning for Time-Series Analysis. ArXiv 2017

Lee, V. C., Ngiam, J., Coates, A. & Lahiri, A. (2011). On Optimization Methods for Deep Learning, I*n Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA.

Le, Q , Ngiam, J , Coates, A, Lahiri, A. , Prochnow, B, & Ng A. (2011). On optimization methods for deep learning. ICML Conference. CVPR Conference

Minal, P, Sanjay, C, & Sanjay, G. (2016). Machine Learning Based Statistical Prediction Model for Improving Performance of Live Virtual Machine Migration. Hindawi Publishing Corporation. *Journal of Engineering*. Volume 2016. Available at: https://doi.org/10.1155/2016/3061674

Nikhil, J., Abhinav, B., Sam, W., Todd G., & Laxmikant, V. K., (2016). *Evaluating HPC Networks via Simulation of Parallel Workloads*

Olof, M. (2016). C-RNN-GAN: Continuous Recurrent Neural Networks with Adversarial Training, Xiv:1611.09904v1[cs.AI]

Parker, C., (2012). Unexpected Challenges in Large Scale Machine Learning, *In the Proceedings of 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining, Algorithms, Systems, Programming Models and Applications*

Patterson, J. & Gibson, A. (2016). Deep Learning: A Practitioner's Approach :Gravenstein Highway : O'Reilly Media

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview, *Neural Networks*, Vol 61 pp. 85–117

Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. Arxiv Preprint Arxiv:1409.1556

Thiyanga, S, Talagala, R. Hyndman J, & George, A. (2018). Meta-Learning How to Forecast Time Series. Working paper. Available at: http://business.monash.edu/econometrics-and-businessstatistics/research/publication

Zhiguang, W, & Weizhong, Y. (2016). Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline.