Deep LSTM-RNN Regression Model for Optimizing Warehouse Inventory at Amazon.com

Rob Kempf, TS8 Technologies, Austin TX, rob@ts8t.com

June 2016

... coming events cast their shadow before. - Thomas Campbell

Abstract

Nearly every aspect of Amazon.com is affected by inventory. Optimize inventory and benefits abound. Using machine learning techniques proven in other fields, Amazon might improve short-term demand forecasts for each product at each warehouse.

1 Introduction

Predicting demand can be incredibly complex. With so many potential factors, forecasting is often more art than science. But when markets are healthy and of sufficient volume, experience suggests patterns of current demand can encapsulate complexity, reflecting factors otherwise unmeasurable.

1.1 Purpose

The purpose of this paper is to outline a proof of concept deep LSTM-RNN regression model for optimizing warehouse inventory at Amazon.com.

1.2 Description

References included describe Recurrent Neural Networks (RNNs) that use Long Short-Term Memory (LSTM) cells. The proposed implementation is a deep RNN with hidden layers, outputting an *integer regression vector* rather than the more common classification output. It also extends the standard LSTM-RNN model by predicting an *aggregate future period* instead of the standard single period.

1.3 Hypothesis

The hypothesis is a deep LSTM-RNN regression model, given a pattern of current demand, can accurately predict future demand and continually adjust predictions in real-time.

2 Demand Vectors

2.1 Current Demand Input Vector

Let p identify a unique physical *product* stocked by Amazon. Let l identify the *location* of a warehouse. Let t identify a regular time $period_t$ such that

$$p = 1, 2, 3..., P$$

$$l = 1, 2, 3..., L$$

Then for each Amazon product p there is a current demand input vector of length L (one element per warehouse) called

$$\mathbf{x}_t$$

Each element of $\mathbf{x}_t$ is the integer quantity of that product ordered during time $period_t$ where location l is the single best Amazon shipping origin regardless of current inventory.

2.2 Future Demand Output Vector

Future demand is measured and predicted for the aggregate period

$$Period_a = A \times period_t$$
$$\text{where } A > 1$$

starting at the end of current $period_t$ (optionally delayed by a latency constant). So each continuing sequence of current demand input vectors $(\mathbf{x}_1 ... \mathbf{x}_t)$ is followed by one future demand output vector of length L called

$$\mathbf{y}_a$$

Here 'a' indicates an *aggregate* time stamp, while 't' refers to the *regular* time stamps denoting smaller period current demand input vectors. Time stamps always mark the beginning of a $period_t$ or a $Period_a$.

So each *pattern* of current demand over $(period_1 ... period_t)$ points to and predicts an *aggregate* future demand over $Period_a$.

3 Prediction Model

3.1 Deep LSTM-RNN Regression

$\mathbf{x}_1$ through $\mathbf{x}_t$ are mapped to $\mathbf{y}_a$ by *forward pass* as follows.*

$$\begin{aligned}
\mathbf{h}_t^{(0)} &:= \mathbf{x}_t \\
\mathbf{h}_t^{(n)} &:= \mathcal{L}_t^{(n)}\left(\mathbf{h}_t^{(n-1)}, \mathbf{h}_{t-1}^{(n)}\right) \\
\mathbf{y}_a &:= \lceil \mathbf{W}^{(N),(N+1)}\mathbf{h}_t^{(N)} + \mathbf{b}^{(N+1)} \rceil
\end{aligned}$$

where

$$\begin{aligned}
t =&\ \text{current regular time stamp} \\
a =&\ \text{aggregate time stamp just after } \textit{period}_t \\
n =&\ \text{hidden layer number} \\
N =&\ \text{total hidden layers} \\
\mathbf{h}_t^{(n)} =&\ n^{th} \text{ hidden layer vector at time } \textit{period}_t \\
\mathbf{x}_t =&\ \text{current demand input vector at time } \textit{period}_t \\
\mathcal{L}_t^{(n)} =&\ \text{LSTM composite activation function} \\
\mathbf{W} =&\ \text{weight matrix} \\
\mathbf{b} =&\ \text{bias vector} \\
\mathbf{y}_a =&\ \text{future demand output vector at time } \textit{Period}_a
\end{aligned}$$

*Notation extended from Weninger, et all.

3.2 Long Short-Term Memory

LSTM cells allow an RNN to retain memory longer than standard logistic cells. LSTM's also solve the 'disappearing gradient problem' while training with back propagation. The composite activation function looks like this.

$$\begin{aligned}\mathcal{L}_t^{(n)}\left(\mathbf{h}_t^{(n-1)}, \mathbf{h}_{t-1}^{(n)}\right) &= \mathbf{o}_t^{(n)} \otimes \tanh\left(\mathbf{c}_t^{(n)}\right) \\ \mathbf{c}_t^{(n)} &= \mathbf{f}_t^{(n)} \otimes \mathbf{c}_{t-1}^{(n)} + \mathbf{i}_t^{(n)} \otimes \tanh\left(\mathbf{W}^{(n-1),(n)}\mathbf{h}_t^{(n-1)} + \mathbf{W}^{(n),(n)}\mathbf{h}_{t-1}^{(n)} + \mathbf{b}_c^{(n)}\right)\end{aligned}$$

where both $\otimes$ and $\tanh$ are applied element-wise and

$$\begin{aligned}\mathbf{o}_t^{(n)} &= \text{output gate} \\ \mathbf{c}_t^{(n)} &= \text{cell state variable} \\ \mathbf{f}_t^{(n)} &= \text{forget gate} \\ \mathbf{i}_t^{(n)} &= \text{input gate} \\ \mathbf{b}_c^{(n)} &= \text{cell bias}\end{aligned}$$

4 Magnitude Assumptions

Amazon.com stocks hundreds of millions of products at over 100 warehouses. Assume P = 250,000,000 physical products and L = 150 warehouse locations. Let $period_t$ = 30 minutes and $A = 336$. That makes $Period_a$ = 1 week. So a series of 30-minute current demand measurements will be used to predict future aggregate demand for the immediately following week.

More hidden layers allow greater non-linear complexity in the model. Common depths are 4 to 8 hidden layers. Let N = 8.

A quarter billion products in stock means one would need a quarter billion models running to forecast them all at once. Perhaps this is less problematic than it sounds. Each prediction model would be quite small, with vectors of just 150 elements each. Each would only need run once every 30 minutes and could be staggered across each $period_t$.

5 Design Trade-offs

There are basically two distinct approaches to designing this type of prediction system.

5.1 One Model per Product

The approach suggested in this paper is to build one model per product and estimate stock required for that one product across all warehouses. Current demand input vectors $\mathbf{x}_t$ have size = L, the total number of warehouse locations.

This type of system has many small models and provides lots of flexibility. Only a hundred or so locations need to be considered. Since every model is independent of any other, new models are quickly deployed and old ones dropped as products are added or discontinued from the mix.

The price of this flexibility is loss of possible inter-product correlation. Many products have related demand profiles and their combined relationship is lost when models are run independently.

5.2 One Model per Warehouse

The second approach is to build one model for each warehouse and estimate stock required at just that warehouse for every product stocked there. Current demand input vectors $\mathbf{x}_t$ have size = P, the total number of products stocked at a given warehouse.

Techniques for dealing with vectors of this size should begin with dimensionality reduction. Even so, this is certainly big data. But modern frameworks can easily split RAM usage across lots of systems and can flexibly spread computation across many separate machines.

The benefit of using one model per warehouse is the potential for significant demand correlation between products. Using Principal Component Analysis (PCA) might dramatically reduce vector size while retaining sufficient variance. Nonetheless, the major downside of this approach occurs when adding or deleting products, eventually requiring a newly sized input vector and consequently a new PCA transform.

5.3 Prototype Design

Using one model per product is easier to prototype, generally more flexible and more easily scaled.

6 Operational Pipeline

6.1 Example Horizon

Going further back in time means getting more data. But going back too far can include data no longer helpful in predicting future outcomes. The beauty of LSTM-RNNs is the LSTM cells teach themselves when to forget irrelevant old data. Even so, one must choose a horizon for the example dataset.

If we choose a 3 year example horizon with $period_t = 30$ minutes, then there are 52,608 possible instances of prediction $period_t$ (if a leap year). Let T be the maximum possible value of t, where $t = 1$ is oldest and $t = T$ is newest.

$$T = 52,608$$

$$t = 1, 2, 3, ..., T$$

6.2 Normalization and Scaling

Each current demand input vector $\mathbf{x}_t$ contains integer values of product orders in some unknown domain, all ≥ 0. These will be normalized and scaled before training the model.

For a given product, we can assemble the matrix $\boldsymbol{X}_p$ sized (LxT) which accounts for every order best shipped from each location over the entire example horizon.

All elements of $\boldsymbol{X}_p$ will first be normalized to a mean of zero and then scaled so standard deviation equals one. Normalizing and scaling the inputs tends to make training more stable and simplifies some of the initial choices for hyper-parameters.

$$\boldsymbol{\mu}_{\boldsymbol{X}_p} = 0$$

$$\boldsymbol{\sigma}_{\boldsymbol{X}_p} = 1$$

In this model, normalizing/scaling output vectors $\mathbf{y}_a$ is likely to have little effect on training issues so, at least at first, output vectors will not be adjusted.

6.3 Validation and Test Sets

If each example sequence is at least one week long, then there are only 52 completely unique sequences to consider per year. To increase the number of examples, sequences can be allowed to overlap. But a requirement for independence in training and test is that no overlapping sequences may exist in any other set.

One solution is to first split the three year time horizon into 36 months, then shuffle the three Januaries. Randomly select one for the validation set and two for the training set. Then shuffle the three Februaries and randomly select one for the test set and the other two for the training set. Do the same with the other months until the validation set has 6, the test set has 6 and the training set has 24 months.

6.4 Examples

The data are still just a series of order volumes. These data should be consumed as example inputs holding sequences with at least 'A' $\mathbf{x}_t$ demand input vectors. Each input sequence is then tagged with its corresponding example $\mathbf{y}_a$ single future output vector.

To increase the number of examples, allow sequences to overlap when stepping through each month, and in the case of the training set, perhaps across consecutive months. The first current demand input vectors for the first 7 days in a month are tagged with the total demand output vector for days 8 through 14. Then include the next $period_t$ forward. Tag these (A+1) new current demand input vectors with the total demand output vector that follows. Continue until you run out of contiguous periods for the input vector. Then begin the process again at $period_t = 2$, and so on. For the output vector, data can be reported from another set, just of course not past the end of the last data in the example horizon.

6.5 Training

Training the model defines the values in $\mathbf{W}$ and $\mathbf{b}$. Incremental offline training can the be done again and again starting with the current operational weights and bias. Reshuffle the months as above before selecting new examples for training, validation and test.

6.6 Prediction

After each incremental training session, hand off the newly optimized model to Amazon systems that place orders to restock each warehouse.

7 Proof of Concept

The first step is to pick a few products and gather order data over the last 3 years.

An algorithm must be selected for collecting $period_t$ order quantities only on those orders *best delivered* from the chosen warehouse. Just because an actual order was delivered from a particular location in the past does not mean it was the best point of departure. All historical orders should be judged anew by an algorithm which decides from where it would best ship today.

Using separate validation and test sets, measure error rates and compare to real world operation of Amazon's current demand prediction system.

The prototype is probably best demonstrated in Python using the TensorFlow API. This code can be provided upon request.

8 Discussion

One might argue the need to account for wildly different buying seasons when making real-time weekly demand predictions. For example, demand is generally much greater for many products during pre-holiday seasons.

Perhaps, but experience has shown a deep LSTM-RNN, as defined here for market time series, copes quite well with seasonal differences. Each pattern of current demand vectors seems to hold a kind of signature. It could be that future seasonal demand is also somehow embedded in such a signature.

9 References

J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber. Evolving Memory Cell Structures for Sequence Learning. In Artificial Neural Networks– ICANN 2009, pp. 755–764. Springer, 2009.

Y. Bengio, P. Simard, and P. Frasconi. Learning Long-term Dependencies with Gradient Descent is Difficult. Neural Networks, IEEE Transactions on, 5 (2):157–166, 1994.

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling Temporal Dependencies in High-dimensional Sequences: Application to Polyphonic Music Generation and Transcription. arXiv preprint arXiv:1206.6392, 2012.

N. Jaitly, Q. Le, O. Vinyals, I. Sutskever and S. Bengio. An Online Sequence-to-Sequence Model Using Partial Conditioning. Google Brain, Mountain View, CA 2015.

R. Jozefowicz, W. Zaremba and I. Sutskever. An Empirical Exploration of Recurrent Network Architectures. Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: WCP volume 37. 2015.

F. Gers, J. Schmidhuber, and F. Cummins. Learning to Forget: Continual Prediction with LSTM. Neural Computation, 12(10):2451–2471, 2000.

A. Graves. Generating Sequences with Recurrent Neural Networks. arXiv preprint arXiv:1308.0850, 2013.

A. Graves. RNNLIB: A Recurrent Neural Network Library for Sequence Learning Problems. http://sourceforge.net/projects/rnnl/, 2013.

A. Graves. Supervised Sequence Labelling with Recurrent Neural Networks. PhD thesis, Technical University of Munich, 2008.

A. Graves, A. Mohamed, and G. Hinton. Speech Recognition with Deep Recurrent Neural Networks. In Proc. of ICASSP, pages 6645–6649, Vancouver, Canada, May 2013. IEEE.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. IEEE Signal Processing Magazine, 29(6):82–97, 2012.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. Neural Computation 9(8):1735-1780, 1997.

L. Kaiser and I. Sutskever. Neural GPUs Learn Algorithms. Conference paper at ICLR 2016.

M. Luong, Q. Le, I. Sutskever, O. Vinyals, L. Kaiser. Multi-Task Sequence to Sequence Learning. conference paper at ICLR 2016.

J. Martens, James and I. Sutskever. Learning Recurrent Neural Networks with Hessian-free Optimization. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), pp. 1033–1040, 2011.

T. Mikolov, et al. Learning Longer Memory in Recurrent Neural Networks. arXiv preprint arXiv:1412.7753, 2014.

R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. How to Construct Deep Recurrent Neural Networks. In Proc. of ICLR, 2014.

R. Pascanu, T. Mikolov, and Y. Bengio. On the Difficulty of Training Recurrent Neural Networks. arXiv preprint arXiv:1211.5063, 2012.

M. Sundermeyer, R. Schluter, and H. Ney. LSTM Neural Networks for Language Modeling. In Proc. of INTERSPEECH, Portland, OR, USA, 2012.

I. Sutskever, O. Vinyals, and Q. Le. Sequence to Sequence Learning with Neural Networks. In Advances in Neural Information Processing Systems, pp. 3104–3112, 2014.

O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a Foreign Language. conference paper at ICLR 2015.

F. Weninger, J. Bergmann, and B. Schuller. Introducing CURRENNT - the Munich Open-Source CUDA RecurREnt Neural Network Toolkit. Editor: Mikio Braun. In Journal of Machine Learning Research 99, 2014.

W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. arXiv preprint arXiv:1409.2329, 2014.