This is the Title

A Thesis
Submitted to the Faculty
of
Drexel University
by
Jae Hoon Kim
in partial fulfillment of the
requirements for the degree
of
Master in Computer Science
June 2017

Table of Contents

List of Tables . . . iii

List of Figures . . . iv

Abstract . . . v

1. Introduction . . . 1

2. Background . . . 2

2.1 Deep Learning . . . 2

2.1.1 Basic Concepts . . . 2

2.1.2 Recurrent Neural Network . . . 2

2.1.3 Auto-encoder . . . 5

2.2 Deep Learning Library . . . 5

2.2.1 DL4J . . . 5

2.3 Machine Learning with Driving Data . . . 6

3. Data Set . . . 7

4. Technical Approach . . . 8

4.1 Cross Validation . . . 8

4.2 LSTM . . . 8

4.3 Dimensionality Reduction . . . 8

4.4 Sampling . . . 8

5. Experiment Result . . . 10

6. Conclusion and Future work . . . 11

Bibliography . . . 12

List of Tables

5.1 Raw 1/10 Result Summary 10

List of Figures

2.1 RNN 2

2.2 Unfold RNN 3

4.1 LSTM without Auto-Encoder 8

4.2 LSTM with Auto-Encoder 8

5.1 Accuracy 10

Abstract

This is the Title
Jae Hoon Kim
Santiago Ontañón, Ph.D.

[To do: Write an abstract here]

Chapter 1: Introduction

Introduction Section

1. Problem Statement

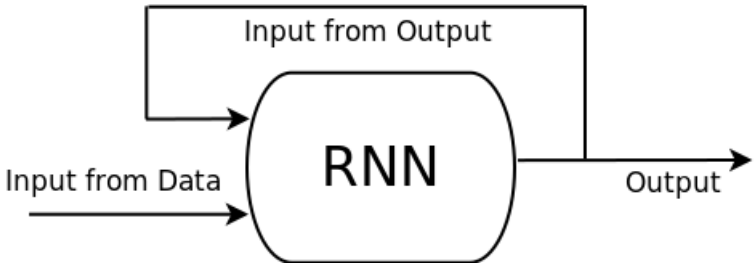

Figure 2.1: RNN

Chapter 2: Background

This chapter presents the necessary background to understand the experiments presented later in the paper. This chapter is dived to three sections. First Section 2.1 covers basic deep learning including recurrent neural network and auto-encoder. Second Section 2.2 compares deep learning libraries: DL4J, Theano, and Tensorflow. The last Section 2.3 introduces current trend of machine learning with driving data.

2.1 Deep Learning

This section covers basic concept of deep learning. First Subsection 2.1.1 introduces to deep learning, and next Section 2.1.2 deals with recurrent neural network and long short term memory neural network. The last Subsection 2.1.3 covers auto-encoders.

2.1.1 Basic Concepts

This part Explain basic deep learning [...]

2.1.2 Recurrent Neural Network

1. Basic concepts of recurrent neural networks:

   A recurrent neural network (RNN) is a neural network that is specialized for processing a sequence of input values [1]. Figure 2.1 illustrates abstract structure of RNN. RNN has two input. One input is from data such as normal neural network input but another input is from previous output and the property gives benefit for sequential input data. This is because past output affects to current output. With sequential data, previous data can affect to current data and RNN considers previous output for current output. It means that even input from data are equal if previous output are different, current output are also different.

   For example, human language sentences have series of words and meaning of words is different in different context. RNN can be used for that. Another example is driving data which is used later on the paper. Driving data are multi-dimension data with time domain. RNN can be used for the data because it is sequential data with time domain.

   To describe RNN in math equation, let $\vec{x}^t = \{x_1^t, ..., x_n^t\} \in \mathbb{R}^n$ be a vector that represents the input data at time t, $h^t = \{h_1^t, ..., h_m^t\}$ be result from hidden layer on time t, and o^t be output on time t. The tensor U is a set of neurons for input data, The tensor V is a set of neurons for data from hidden layer and the tensor W is a set of neurons for previous output.

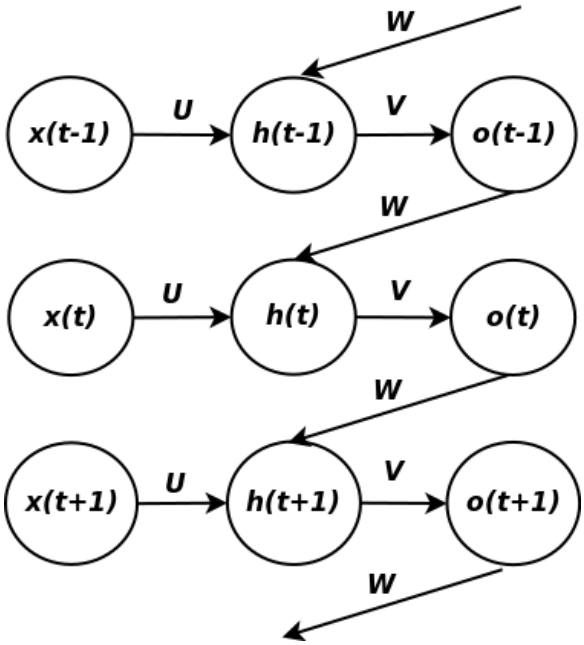

Figure 2.2: Unfold RNN

The figure 2.2 illustrates unfolded RNN with defined symbols. The result from hidden layer can be calculated by

$$h^t = f(Ux^t + Wo^{t-1} + b_h)$$

where $f()$ is activation function for hidden layer and b is bias for hidden layer. Then the output can be calculated by

$$o^t = g(Vh^t + b_o)$$

where $g()$ is activation function for output layer and b_o is bias for output layer. Therefore, RNN can be written in

$$o^t = g(Vf(Ux^t + Wo^{t-1} + b_h) + b_o)$$

RNN is not always like the figure 2.2. Input from previous output can be replaced by result of previous hidden layer. The main idea of RNN is when neural network decides current output it considers previous state.

2. Long Short Term Memory (LSTM) networks:

   Basic RNN have the problem of long term dependency. It means that an RNN is not able to memorize long term knowledge. To solve the problem, LSTMs were introduced by Hochreiter and Schmidhuber [2]. An LSTM has a cell state and it solves the long term memory problem in RNN because cell state can store long term memory [santi: explain what is "cell state"].

   Bengio mentioned on his paper [3] that each layers in deep neural network should have a purpose. LSTM can be divided to four layers: forget gate, input gate, input moduleation gate, and output gate. Each layers have different purpose [santi: this paragraph does not flow very well], even if Bengio mentioned it, why is it important? you need to explain WHY is it that each layer should have a purpose. Otherwise, we just have to blindly believe in Bengio :).

   [santi: this paragraph does not read very well, in blue below, I have an alternative version that flows better (just as an example)] Colah's blog [1] develops an intuition for concept of LSTM and explains purpose of each layers. The paper [4] describes LSTM in mathematical terms. In this paper, I'm going to summarize both the Colah's blog and the paper [4]

[1] http://colah.github.io/posts/2015-08-Understanding-LSTMs

Let us first provide an intuitive description of how LSTMs work (following Christopher Olah's blog [2]), before providing a more in-depth formalization:

(a) An intuitive ~~concept~~ explanation of LSTMs:

Colah's [santi: "Colah" → "Christopher Olah"] blog explains concept of LSTM and summarizes four layers in LSTM with each purpose. [santi: in LaTeX, the way to separate paragraphs is to leave a blank line between them, not using the double backslash (which should not be used often).]

The first layer is forget gate. The forget gate filters what data to forget or throw away. It consists of sigmod [santi: instead of saying "It consists of sigmod", you should say something like "The units in this layer use a sigmoid activation function" (same comment for all the other layers below)] layer with previous output and current input data [santi: a reader not familiar with LSTMs will not know what is "previous output" or "current input", so you need a figure that explains it]. The result of the forget gate is multiplied by previous cell state to build current cell state.

The second layer is input gate and also consists of sigmod layer with previous output and current input data. The result of the input gate is multiplied by the result of input modulation gate then it affects to cell state [santi: all of these terms should appear in a figure, otherwise, a reader not familiar with it will not understand what you are saying. But also, even if you have a figure, you need to explain what each term is, e.g., explain what is "input modulation", what is "cell state", etc.]. The meaning of input gate is to decide~~s~~ what data to remember.

The third layer is input modulation gate. It consists of tanh with previous output and current input. It is multiplied by the result of second layer and affects to current cell state.

The last layer is output gate. The layer decides what data should be passed to current output from previous output and current input. the output gate also consists of sigmoid with two inputs: last output and current input. The result of the layer is multiplied by current cell state passed tanh.

(b) Modeling LSTM in mathematical terms: LSTM is described in mathematical term in the paper [4]. I'm going to summarize it on this paper.
Let
$T_{n,m} : \mathbb{R}^n \to \mathbb{R}^m$ be an affine transform
x_t be size n vector for current input
h_{t-1} be size n vector for last output
h_t be size n vector for current output
c_{t-1} be size n vector for last cell state
c_t be size n vector for current cell state

$$\begin{bmatrix} input\_gate \\ forget\_gate \\ output\_gate \\ input\_modulation\_gate \end{bmatrix} = \begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} sigm & 0 & 0 & 0 \\ 0 & sigm & 0 & 0 \\ 0 & 0 & sigm & 0 \\ 0 & 0 & 0 & tanh \end{bmatrix} T_{2n,4n} \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}$$

$$= \begin{bmatrix} sigm & 0 & 0 & 0 \\ 0 & sigm & 0 & 0 \\ 0 & 0 & sigm & 0 \\ 0 & 0 & 0 & tanh \end{bmatrix} \begin{bmatrix} {T^{hi}}_{n,n} & {T^{xi}}_{n,n} \\ {T^{hf}}_{n,n} & {T^{xf}}_{n,n} \\ {T^{ho}}_{n,n} & {T^{xo}}_{n,n} \\ {T^{hg}}_{n,n} & {T^{xg}}_{n,n} \end{bmatrix} \begin{bmatrix} h_{t-1} \\ x_x \end{bmatrix}$$

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs

$$= \begin{bmatrix} sigm & 0 & 0 & 0 \\ 0 & sigm & 0 & 0 \\ 0 & 0 & sigm & 0 \\ 0 & 0 & 0 & tanh \end{bmatrix} \begin{bmatrix} {T^{hi}}_{n,n} h_{t-1} + {T^{xi}}_{n,n} x_x \\ {T^{hf}}_{n,n} h_{t-1} + {T^{xf}}_{n,n} x_x \\ {T^{ho}}_{n,n} h_{t-1} + {T^{xo}}_{n,n} x_x \\ {T^{hg}}_{n,n} h_{t-1} + {T^{xg}}_{n,n} x_x \end{bmatrix}$$

Therefore,

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} sigm({T^{hi}}_{n,n} h_{t-1} + {T^{xi}}_{n,n} x_x) \\ sigm({T^{hf}}_{n,n} h_{t-1} + {T^{xf}}_{n,n} x_x) \\ sigm({T^{ho}}_{n,n} h_{t-1} + {T^{xo}}_{n,n} x_x) \\ tanh({T^{hg}}_{n,n} h_{t-1} + {T^{xg}}_{n,n} x_x) \end{bmatrix}$$

$\{i, f, o, g\}$ is a set of output from each layers. The current cell state $c_t = f \cdot c_{t-1} + i \cdot g$ the current output $h_t = o \cdot tanh(c_t)$

2.1.3 Auto-encoder

In this section, I summarize 'Reducing the dimensionality of data with neural networks' by Hinton [5]. The paper introduces to the method to reduce high dimensional data by neural networks named auto-encoder. The result of the paper shows that reducing dimension by auto-encoder gives better performance than PCA.

1. Intuition concepts of auto-encoder
   When auto-encoder is traind, it has two layers: encode and decode. The encode layer receives input and passes output in lower dimension data to decode layer. The decode layer recovers dimension to input data dimention from output of encode layer. The error of auto-encoder is measured by difference between input data and output from decode layer. In other words, encoder layer compresses input data and decoder layer decompresses data from encoder layer that should be similar as original input data. That methods guarantees that the data from encoder layer keeps almost all properties of input data in lower dimension. That is the reason why output data from decode layer is similar as original input data. In math, relationship between encoder and decoder layer is similar as inverse matrix of each other.

2. Multilayer Auto-encoder

2.2 Deep Learning Library

2.2.1 DL4J

DL4J [3] is open source, distributed deep-learning library for Java and Scala. It also integrates with Hadoop and Spark.

Theano

Theano [4] is a Python library for machine learning research. It integrates with Numpy and uses GPU power. The library is written in C code and optimizes user functions.

Tensorflow

Tensorflow [5] is an open source software library for numerical computation using data flow graphs. The biggest difference from other libraries is that Tensorflow treats all operator as node. Another

[3] https://deeplearning4j.org/
[4] http://deeplearning.net/software/theano/
[5] https://www.tensorflow.org/

strong point is that Tesorflow allows developer to deploy computation to one or more CPUs or GPUs.

2.3 Machine Learning with Driving Data

Chapter 3: Data Set

The drive simulation data is from *Children's Hospital of Philadelphia* (CHOP). The simulator for the data can record 100 features with 60 samples per second from driver. The simulator has four different tracks. On the paper, the data has 16 set: 8 from expert and 8 from inexpert. Each expert and inexpert data set consist of two set of four different tracks.

Chapter 4: Technical Approach

4.1 Cross Validation

There are only 16 data set. It is not enough to train neural network for general case. Whether the neural network works well on the problem or not, I used cross validation. I split 16 data set to 4 groups. Each groups have 2 expert and 2 inexpert data. The neural network is trained four times with different train (3 groups) and test (1 group) set.

4.2 LSTM

The LSTM neural network is used to classify data because the data has time domain and LSTM neural network can handle serial data. On the paper, LSTM neural network is built with 16, 32, 64, 128, and 256 hidden neurons. The output from LSTM is sent to output layer which has two neurons. By using softmax, output from two neurons is classified. If it has [0, 1], it is classified to expert. Otherwise, it is classified to inexpert.

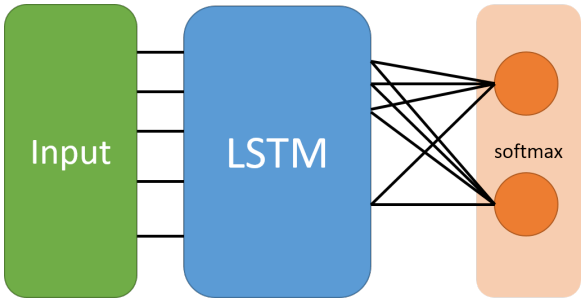

Figure 4.1: LSTM without Auto-Encoder

4.3 Dimensionality Reduction

The data has 100 features so dimensionality reduction is necessary. First way is selecting meaningful features and 23 features are selected. Another way to reduce dimension is auto-encoder. Before giving data to LSTM neural network, data is sent to auto-encoder layer and less dimension data from auto-encoder is sent to LSTM.

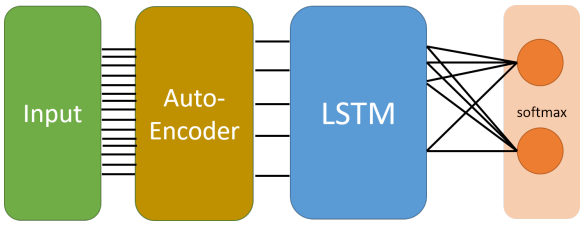

Figure 4.2: LSTM with Auto-Encoder

4.4 Sampling

The simulator records 60 samples per second. It is too often recorded. I reduce data by two way. First way is sampling one per ten and sampling one per twenty. The sampled data might not

represent ten or twenty data, so average data from every ten or twenty data is used. The second way can reduce noise because there is case in first way that sampled data has more noise than other data.

Chapter 5: Experiment Result

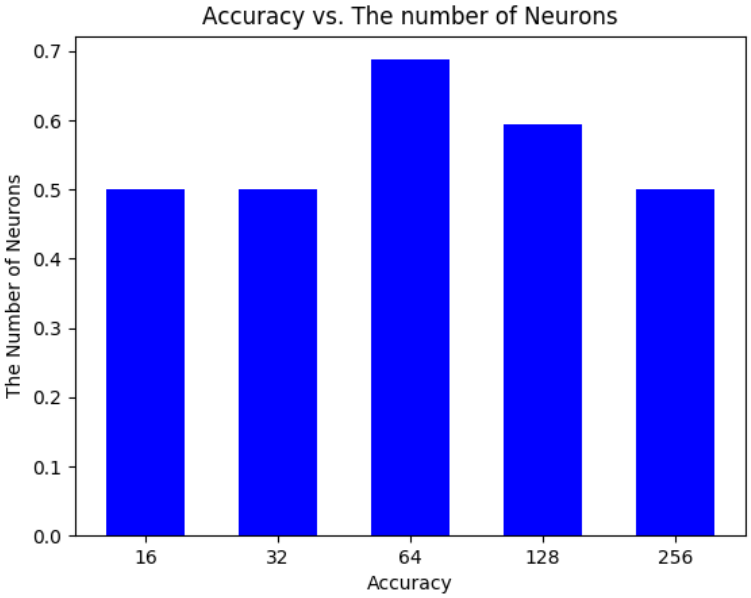

Figure 5.1: Accuracy

Table 5.1: Raw 1/10 Result Summary

		The number of neurans				
Test	n	16	32	64	128	256
1	1	0.5	0.25	1.0	0.5	0.5
	2	0.75	0.75	0.5	0.5	0.5
	3	1.0	0.5	0.5	0.75	0.5
	4	0.5	0.5	0.75	0.5	0.5
2	1	0.5	0.75	0.75	0.75	0.75
	2	0.25	0.25	0.75	0.5	0.25
	3	0.5	0.5	0.25	0.75	0.25
	4	0.25	0.25	0.5	0.25	0.75
3	1	0.75	0.25	1.0	0.75	0.25
	2	0.25	0.5	0.75	0.5	1.0
	3	0.5	0.75	0.75	0.5	0.5
	4	0.5	0.25	0.75	0.75	0.5
4	1	0.75	0.5	0.5	0.5	0.5
	2	0.25	0.5	0.5	1.0	0.25
	3	0.0	0.75	1.0	0.75	0.25
	4	0.75	0.75	0.75	0.25	0.75
Average		0.5	0.5	0.6875	0.59375	0.5

Chapter 6: Conclusion and Future work

Conclusion and Future work

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[3] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[4] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[5] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.