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

List of Figures

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

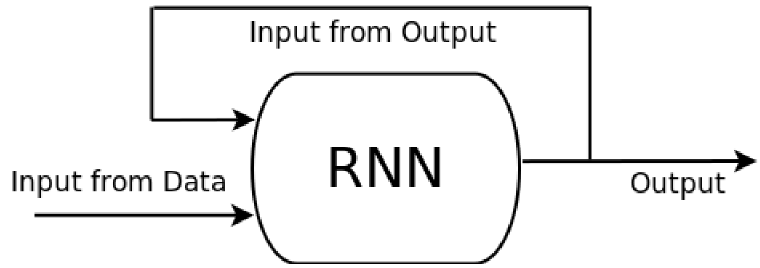

Figure 2.1: RNN change variables

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. 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, $\vec{h^t} = \{h_1^t, ..., h_m^t\} \in \mathbb{R}^m$ be result from hidden layer on time t, and $\vec{o^t} = \{o_1^t, ..., o_l^t\} \in \mathbb{R}^l$ be output on time t. For transform matrix, let $\mathbf{U} : \mathbb{R}^n \to \mathbb{R}^m$ be a transform matrix for input data, $\mathbf{V} : \mathbb{R}^m \to \mathbb{R}^l$ be a transform matrix for data from hidden

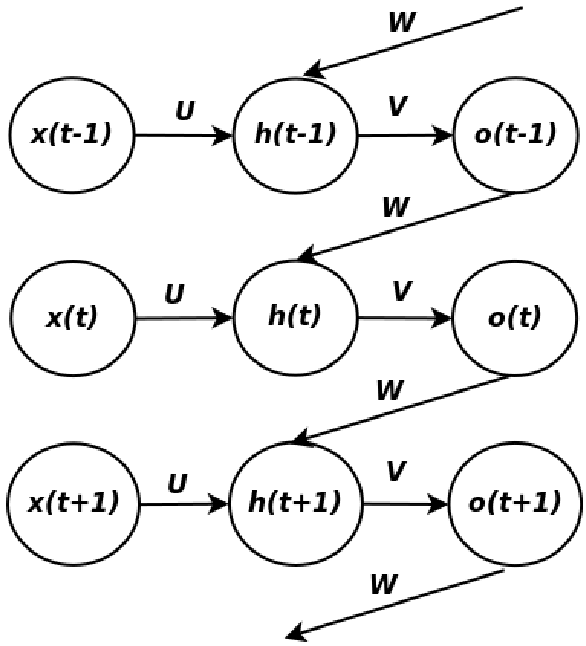

Figure 2.2: Unfold RNN change variables

layer, and $\mathbf{W} : \mathbb{R}^l \to \mathbb{R}^m$ be a transform matrix for previous output. The figure 2.2 illustrates unfolded RNN with defined symbols. The result from hidden layer can be calculated by

$$\vec{h^t} = \mathbf{f}(\vec{x^t}\mathbf{U} + \vec{o^{t-1}}\mathbf{W} + \vec{b_h})$$

where $\mathbf{f}()$ is activation function for hidden layer and it is applied for each elements in given vector, $\vec{b_h} \in \mathbb{R}^m$ is a vector for hidden layer bias. Then the output can be calculated by

$$\vec{o^t} = \mathbf{g}(\vec{h^t}\mathbf{V} + \vec{b_o})$$

where $\mathbf{g}()$ is activation function for output layer and it is applied for each elements in given vector, $\vec{b_o} \in \mathbb{R}^l$ is a vector for output layer bias. Therefore, RNN can be written in

$$\vec{o^t} = \mathbf{g}(\mathbf{f}(\vec{x^t}\mathbf{U} + \vec{o^{t-1}}\mathbf{W} + \vec{b_h})\mathbf{V} + \vec{b_o})$$

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 has the problems of long term dependency. When RNN passes previous output for current output, some information in the input might not need or might need for future. RNN does not have ability to filter unnecessary information or to store necessary information. Long Short Term Memory (LSTM) neural network can handle these problems.

   LSTM is a type of RNN and introduced by Hochreiter and Schmidhuber [2]. LSTM solves long term dependency problems in RNN by memory cells. LSTM manages memory cells as storage of knowledge. LSTM filters unnecessary information from memory cells and records necessary information on memory cells.

   The structure of LSTM consists of four gates: forget gate, input gate, input modulation gate, and output gate. Each gates have different purpose and Christopher Olah's blog [1] develops an intuition for concept of LSTM and explains purpose of each gates. The paper [3] describes LSTM in mathematical term. Let us first provide an intuitive description of how LSTM work by following Christopher Olah's blog, before providing a more in-depth formalization.

(a) An intuitive explanation of LSTMs:

The easiest way to understand LSTM is to understand memory cell and purpose of four different gates. Memory cell makes LSTM different from RNN. Memory cell stores important information and filters unimportant information for future. Three of four gates are involved in the memory cells to store and filter information.

The first gate affecting to memory cells is forget gate. The forget gate decides unnecessary data from input data and previous output then applies it to memory cells. Other two gates are input gate and input modulation gate and these also affect to memory cells. The two gates decide what information remember then apply it to memory cells. The last gate is output gate and it does not directly affect to memory cells. Output gate also has two inputs from input data and previous output, and output from output gate is multiplied by memory cells to make final output. The Figure 2.3 illustrates LSTM neural network with four gates and next part describes more detail of LSTM and the figure in mathematical terms.

(b) Modeling LSTMs in mathematical terms:

To describe LSTM in mathematical terms, 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, $\vec{i^t} = \{i_1^t, ..., i_m^t\} \in \mathbb{R}^m$ be result from input gate on time t, $\vec{m^t} = \{m_1^t, ..., m_m^t\} \in \mathbb{R}^m$ be result from input modulation gate on time t, $\vec{f^t} = \{f_1^t, ..., f_m^t\} \in \mathbb{R}^m$ be result from forget gate on time t, $\vec{o^t} = \{o_1^t, ..., o_m^t\} \in \mathbb{R}^m$ be result from output gate on time t, $\vec{c^t} = \{c_1^t, ..., c_m^t\} \in \mathbb{R}^m$ be memory cells on time t, and $\vec{h^t} = \{h_1^t, ..., h_m^t\} \in \mathbb{R}^m$ be final result on time t. Each gates have transform matrices for inputs. Let $\mathbf{T_{n,m}} : \mathbb{R}^n \to \mathbb{R}^m$ be a transform matrix from n dimension to m dimension. Let $\mathbf{T^{ii}_{n,m}}$ be a transform matrix for input from data in input gate, $\mathbf{T^{oi}_{m,m}}$ be a transform matrix for input from previous output in input gate, $\mathbf{T^{im}_{n,m}}$ be a transform matrix for input from data in input modulation gate, $\mathbf{T^{om}_{m,m}}$ be a transform matrix for input from previous output in input modulation gate, $\mathbf{T^{if}_{n,m}}$ be a transform matrix for input from data in forget gate, $\mathbf{T^{of}_{m,m}}$ be a transform matrix for input from previous output in forget gate, $\mathbf{T^{io}_{n,m}}$ be a transform matrix for input from data in output gate, $\mathbf{T^{oo}_{m,m}}$ be a transform matrix for input from previous output in output gate.

The result of each gates are computed as following:

$$\vec{i^t} = \mathbf{sigm}(\vec{x^t}\mathbf{T^{ii}_{n,m}} + \vec{h^{t-1}}\mathbf{T^{oi}_{m,m}} + \vec{b_i})$$

Input gate uses sigmoid function $\mathbf{sigm}()$ for activation function and $\vec{b_i} \in \mathbb{R}^m$ is a vector for input gate bias.

$$\vec{m^t} = \mathbf{tanh}(\vec{x^t}\mathbf{T^{im}_{n,m}} + \vec{h^{t-1}}\mathbf{T^{om}_{m,m}} + \vec{b_m})$$

Input modulation gate uses tanh function $\mathbf{tanh}()$ for activation function and $\vec{b_m} \in \mathbb{R}^m$ is a vector for input modulation gate bias.

$$\vec{f^t} = \mathbf{sigm}(\vec{x^t}\mathbf{T^{if}_{n,m}} + \vec{h^{t-1}}\mathbf{T^{of}_{m,m}} + \vec{b_f})$$

Forget gate uses sigmoid function $\mathbf{sigm}()$ for activation function and $\vec{b_f} \in \mathbb{R}^m$ is a vector for forget gate bias.

$$\vec{o^t} = \mathbf{sigm}(\vec{x^t}\mathbf{T^{io}_{n,m}} + \vec{h^{t-1}}\mathbf{T^{oo}_{m,m}} + \vec{b_o})$$

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

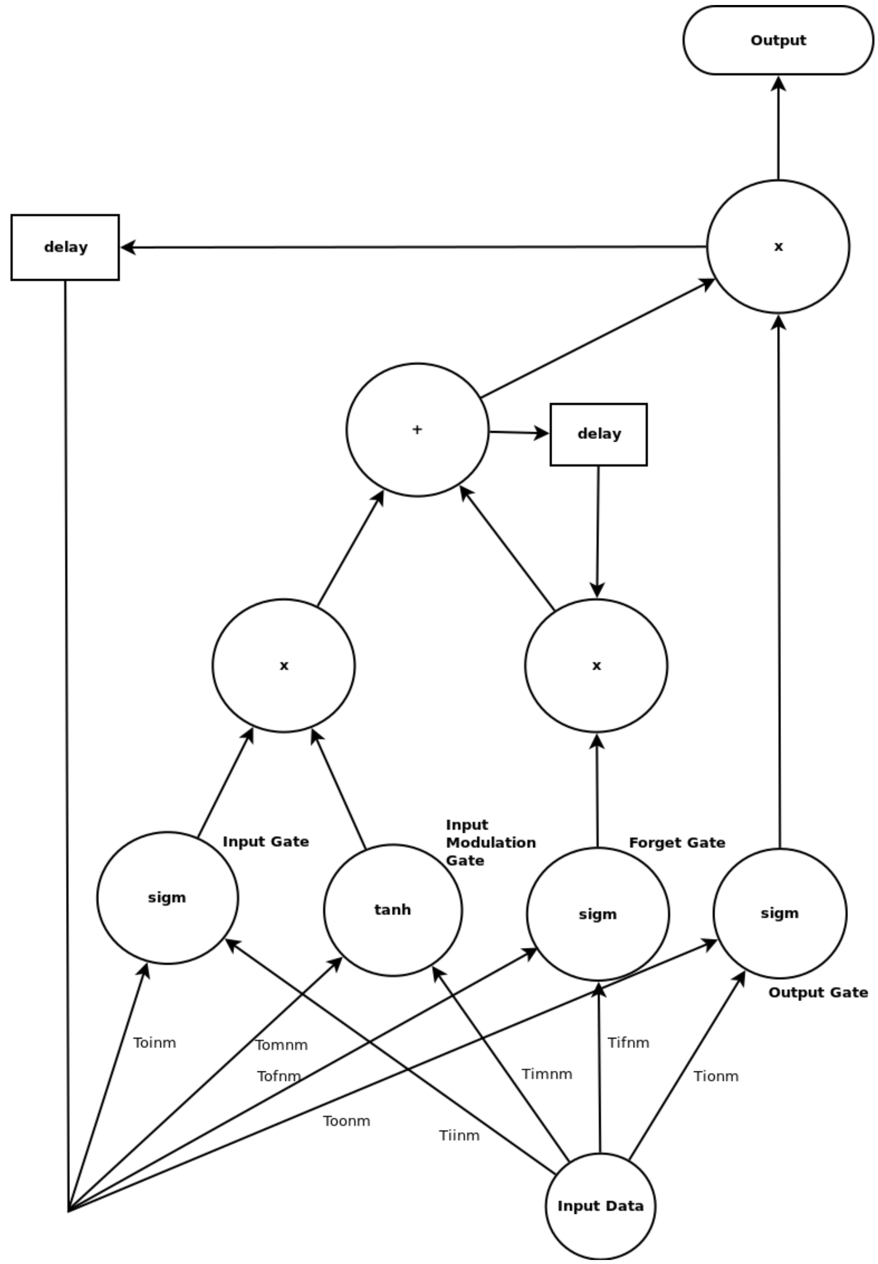

Figure 2.3: LSTM change variables

Output gate uses sigmoid function **sigm**() for activation function and $\vec{b_o} \in \mathbb{R}^m$ is a vector for output gate bias.
Memory cells are computed as

$$\vec{c^t} = \vec{i^t} * \vec{m^t} + \vec{f^t} * \vec{c^{t-1}}$$

And final result is computed as

$$\vec{h^t} = \vec{c^t} * \vec{o^t}$$

Where $*$ is component-wise multiplication of two vectors.

LSTM neural network described above is one example of LSTMs. There are many other

LSTMs but all LSTMs has memory cells to store information for long term memory and has four gates: input, input modulation, forget, and output gate.

2.1.3 Auto-encoder

In this section, I summarize 'Reducing the dimensionality of data with neural networks' by Hinton [4]. 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 [2] is open source, distributed deep-learning library for Java and Scala. It also integrates with Hadoop and Spark.

Theano

Theano [3] 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 [4] 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 strong point is that Tesorflow allows developer to deploy computation to one or more CPUs or GPUs.

2.3 Machine Learning with Driving Data

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

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.

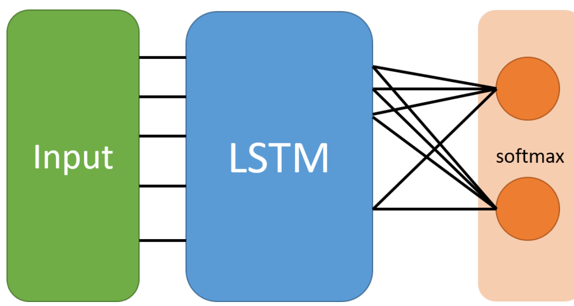

Figure 4.1: LSTM without Auto-Encoder

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.

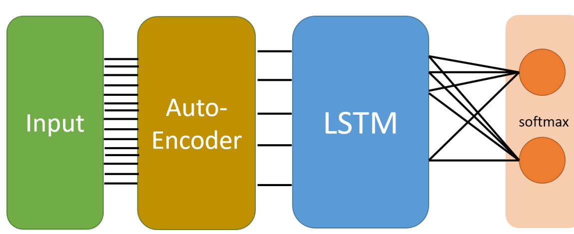

Figure 4.2: LSTM with Auto-Encoder

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

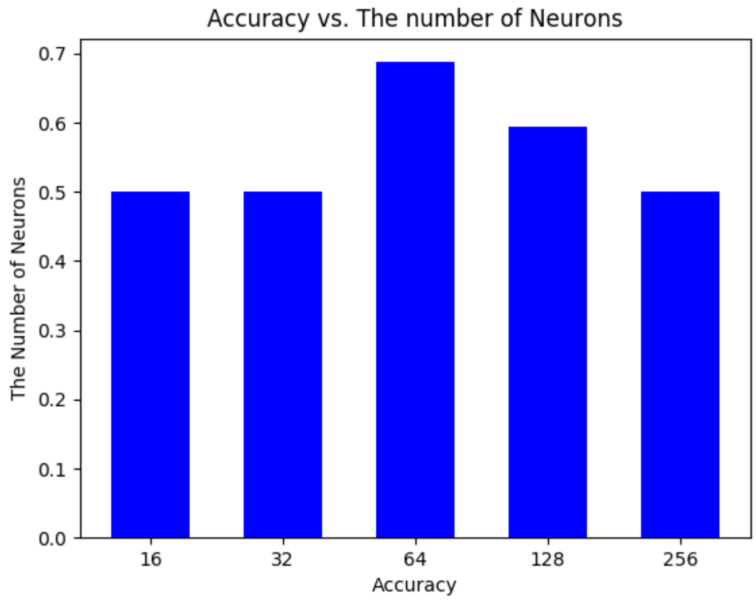

Figure 5.1: Accuracy

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

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