UNIVERSITY OF SCIENCE
ADVANCED PROGRAM IN COMPUTER SCIENCE

THAI THIEN - 1351040
VAN DUY VINH - 1351050

SỬ DỤNG LATEX TRONG KHOÁ LUẬN TỐT NGHIỆP

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

HO CHI MINH CITY, 2017

UNIVERSITY OF SCIENCE
ADVANCED PROGRAM IN COMPUTER SCIENCE

THAI THIEN - 1351040
VAN DUY VINH - 1351050

SỬ DỤNG LATEX TRONG KHOÁ LUẬN TỐT NGHIỆP

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

THESIS ADVISOR
NGHIÊM QUỐC MINH

HO CHI MINH CITY, 2017

ACKNOWLEDGMENTS

Tôi xin chân thành cảm ơn ...
Xin cám ơn!

Tp. Hồ Chí Minh, ngày ... tháng ... năm 2016
Sinh viên thực hiện
Thai Thien Van Duy Vinh

Contents

List of Figures

List of Tables

ABSTRACT

Tóm tắt khóa luận: trình bày tóm tắt vấn đề nghiên cứu, các hướng tiếp cận, cách giải quyết vấn đề và một số kết quả đạt được. Bản tóm tắt dài từ 1 đến 2 trang.

Chapter 1

Introduction

Chapter 2

Technical Issues

2.1 Framework

2.1.1 Torch

Torch [1] is Lua scientific computing framework. Torch support high performing matrix calculation via multi-dimensional array call Tensor. Torch are built with C/C++, CUDA backend. Torch author choose Lua because Lua works well with C/C++ [**collobert2011torch7**]. Thus, Torch is high performing and support GPU. Torch have neural network package (nn) package. Computation graph must be define before forward pass. A simple, single linear layer network can be easily defined with few line of code (see listing 2.1).

```
-- simple y = Ax + b linear layer
l = nn.Linear(2,3)
-- forward pass
x = torch.Tensor(2)
y = l:forward(x) -- vector dimension of 3
```

Listing 2.1: Simple linear layer in Torch

However, when a model need multiple module, such as multilayer perceptron (MLP), these module must be put into container. Figure A.1 illustrates on function of each nn container . In order to construct two-layer perception (eq 2.1), linear, tanh and

[1] http://torch.ch/

softmax module must be packed into sequential module (see listing 2.2).

$$
\begin{aligned}
h &= tanh(W_1 * x + b_1) \\
y &= softmax(W_2 * h + b2)
\end{aligned}
\tag{2.1}
$$

```
model = nn.Sequential()
model:add(nn.Linear(2,3))
model:add(nn.Tanh())
model:add(nn.Linear(3,5))
model:add(nn.SoftMax())
-- forward
x = torch.Tensor(2)
y = model:forward(x)
```

Listing 2.2: MLP in Torch

Torch provide nngraph package support build more complicate model. For example, define MLP in (eq 2.1) use nngraph (see listing 2.3)

```
model = nn.Sequential()
model:add(nn.Linear(2,3))
model:add(nn.Tanh())
model:add(nn.Linear(3,5))
model:add(nn.SoftMax())
-- forward
x = torch.Tensor(2)
y = model:forward(x)
```

Listing 2.3: MLP using nngraph

Sample code on training a model, see Appendix A.1

2.1.2 Theano

Theano [2] is a deep learning library on Python. It basic function is similar to Torch: matrix calculation, support GPU. Theano is define-and-run schema, which a computer graph must be built before it is executed.

```
x = T.dmatrix('x')
y = T.dmatrix('y')
z = x + y
f = function([x, y], z)
```

[2] http://deeplearning.net/software/theano/

```
f([[1, 1], [2, 2]], [[3, 3], [4, 4]])
# result [[4, 4], [6, 6]]
```

Listing 2.4: Define function in Theano

Comparing to Torch7, Theano are slower on most benchmark [**collobert2011torch7**]. Theano does not provide nice template like linear layer. Thus, model must be defined from equation. It give researcher more control over mathematics aspect but cause more trouble for beginner. A sample code for MLP A.2. One more problem is that the 'define-and-run' scheme does not suitable for recursive neural network due to recompile the computation graph each training sample take time.

2.1.3 Pytorch

PyTorch uses same backend as Torch. However, PyTorch specially designed for Python. Pytorch have pre-define module (Linear layer, Convolution layer) like Torch. However, Pytorch does not require to pack model into container. In Pytorch a network are defined in forward-pass thanks to Dynamic Neural Networks feature. Therefore, user can use Python control flow to define a network. For example, one can use for loop to run recurrent neural network (see) .The features allows us to implement Recursive Neural Network for NLP, which the network change for every sample, much more easier.

```
rnn = torch.
```

Listing 2.5: RNN

Chapter 3

Experiment

3.1 Dataset

3.1.1 Stanford Sentiment Treebank

In this thesis, we use Standford Sentiment Treebank (SST) dataset [**socher2013recursive**]. Standford Sentiment Treebank contains 11,855 sentences. Each data sentence consist of fined-grain sentiment labeled phrases in constituency parse tree structure (see **Figure 3.1**). There are total 215,154 phrases in whole dataset. The dataset was splitted into train/dev/test contain 8544/1101/2210 sentences each for training and evaluation models. After remove neutral sentiment sentences, there are 6920/872/1821 sentences remained in train/dev/test set.

SST dataset are publicly available online [1].

Preprocess

We use preprocess source code from [**socher2013recursive**] implementation [2] to preprocess SST.

3.1.2 Amazon Movies Review

We get Amazon Movies and TV reviews (4,607,047 reviews) and Amazon Book reviews (22,507,155 reviews) [**he2016ups**]. Listing **??** is sample of one book review.

[1] https://nlp.stanford.edu/sentiment/index.html

[2] `https://github.com/stanfordnlp/treelstm`

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.  He is having a wonderful time playing these old hymns.  The music  is at times hard to read because we think the book was published for singing from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


Listing 3.1: Amazon reviews sample

We extract reviewText and overall from review dataset. We use overall as sentiment score for reviews, with 5 is very positive and 0 is very negative.

We get Amazon Movies and TV reviews (4,607,047 reviews) and Amazon Book reviews (22,507,155 reviews) [**he2016ups**] as our training dataset for our word embedding layer. First, we join both dataset and shuffle. Then, we group review of same produce together. Finally, for each product, we sort reviews based on score. Hence, neighbor reviews have similar sentiment and talk about same product, except at the boundary of two produces. We also create one version of dataset which we only shuffle the data without group or sort for comparision. We train our word representation from Amazon dataset using Glove [**pennington2014glove**] on 15 iteration with windows size of 20.

Table 3.1: My caption

	Constituency Tree-LSTM	LSTM
Glove 42B		
Glove 840B		
Glove (Amazon)	88.45	
Glove (Amazon Sorted)	88.85	
Paragram-Phrase XXL		
SSWEu		

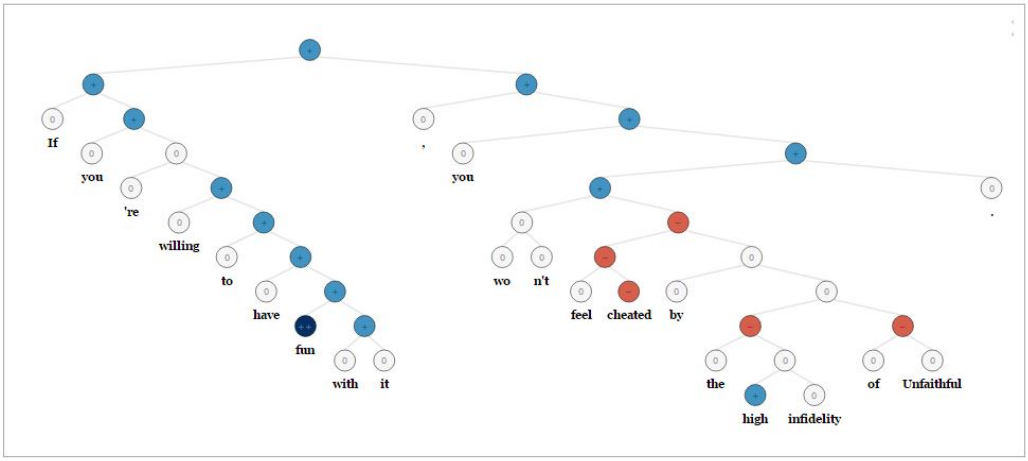

Figure 3.1: A parsed sentence in SST [a]

[a]Render by Pytreebank `https://github.com/JonathanRaiman/pytreebank`

3.2 Embedding

We run experiment on variety of pre-trained word representation.

3.3 title

Danh mục công trình của tác giả

1. Tạp chí ABC
2. Tạp chí XYZ

TABLE OF CONTENTS

Appendix A

APPENDICES 1

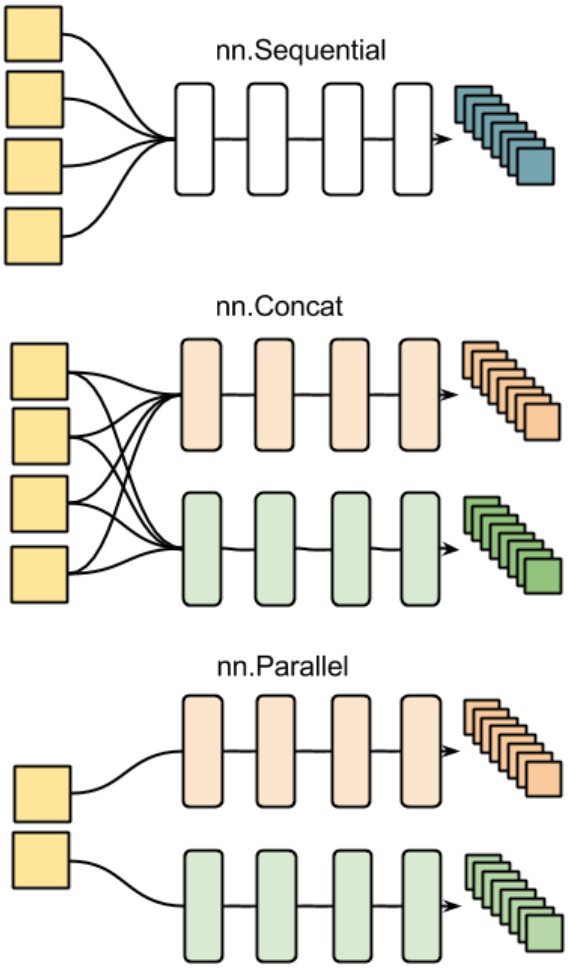

Figure A.1: Torch nn container [a]

[a] https://github.com/soumith/cvpr2015/blob/master/Deep%20Learning%20with%20Torch.ipynb

```
require 'nn'
require 'optim'

model = nn.Sequential()
model:add(nn.Linear(1,1))

criterion = nn.MSECriterion()

x = torch.Tensor{{1,2,3,4,5,6,7,8,9,10}}
x = x:t()
y = torch.Tensor{{3,5,7,9,11,13,15,17,19,21}}
y = y:t()

params, gradParams = model:getParameters()

function feval(params)
gradParams:zero()
local outputs = model:forward(x)
local loss = criterion:forward(outputs,y)
local dloss_doutput = criterion:backward(outputs,y)
model:backward(x, dloss_doutput)
return loss, gradParams
end

local optimState = {
learningRate = 0.01
}

for epoch = 1, 100 do
optim.sgd(feval,params, optimState)
end

test = torch.Tensor{{1,3,5,7,9,11,100}}
test = test:t()

print (model:forward(test))
```

Listing A.1: MLP using nngraph

```
import numpy as np
import theano
import theano.tensor as T
import theano.tensor.nnet as nnet
```

```
5
6  class MLP:
7  def __init__(self):
8  x = T.dvector()
9  y = T.dscalar()
10 t1 = np.array(np.random.rand(3, 3), dtype=theano.config.floatX)
11 theta1 = theano.shared(t1)  # 3x3 weight matrix
12 t2 = np.array(np.random.rand(4, 1), dtype=theano.config.floatX)
13 hid1 = MLP.sigmoid_layer(x, theta1)  # hidden layer
14 theta2 = theano.shared(t2)  # 4x1 weight matrix
15 output_layer = T.sum(MLP.sigmoid_layer(hid1, theta2))
16 fc = (output_layer - y)**2
17 self.cost = theano.function(inputs=[x,y], outputs = fc,updates=[
18 (theta1, Xor.grad_desc(fc, theta1)),
19 (theta2, Xor.grad_desc(fc, theta2))
20 ])
21 self.run_forward = theano.function(inputs=[x],outputs=output_layer)
22
23 @staticmethod
24 def sigmoid_layer(x, w):
25 b = np.array([1], dtype=theano.config.floatX)
26 new_x = T.concatenate([x,b])
27 m = T.dot(w.T, new_x)
28 h = nnet.sigmoid(m)
29 return h
30
31
32 @staticmethod
33 def grad_desc(cost, theta):
34 alpha = 0.1
35 return theta - (alpha* T.grad(cost, wrt=theta))
36
37 def forward(self, x):
38 output = self.run_forward(x)
39 return output
40
41
42 def train(self, train_x, train_y, n_epoch):
43 cur_cost = 0
44 for epoch in range(n_epoch):
45 for i in range(len(train_x)):
46 cur_cost = self.cost(train_x[i],train_y[i])
```

```
47 if epoch % 1000 == 0:
48 print(cur_cost)
49 print ('train complete')
```

Listing A.2: Theano MLP

Appendix B

APPENDICES 2

Đây là phụ lục 2.