

# Selecting Hyperparameter for Multilayer Perceptron

## COMP 4211 - Tutorial 05

Chun-Kit Yeung

Hong Kong University of Science and Technology

2018-03-16

# Objective

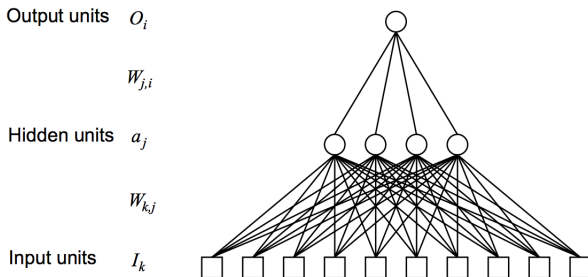
In this tutorial, you will learn the basic terminology and workflow in TensorFlow.

## Agenda

- 1 What is TensorFlow?
- 2 How can you setting up an environment to run TensorFlow?
- 3 How to use TensorFlow?

# Recap

- Multilayer perceptron/deep neural network.



# TensorFlow

## What is TensorFlow?

# What is TensorFlow?

- It is a **deep learning library** supported by Google.
- It provides lots of functions on tensors (n-dimensional array) for automatically computing their derivatives.

't'
'e'
'n'
's'
'o'
'r'

tensor of dimensions [6]  
(vector of dimension 6)

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

tensor of dimensions [6,4]  
(matrix 6 by 4)

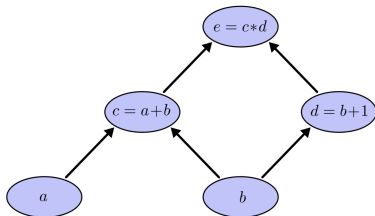
2	1	8	8	8
2	8	5	2	1
2	4	9	0	5
2	3	3	0	8
7	7	1	5	2

tensor of dimensions [4,4,2]

Example of tensor

# Why does it call TensorFlow?

- Tensorflow is basically a package for you to define a **computation graph**.




- This defines **how the tensors should be flowed in the graph** during computation.

# Tensorflow vs Numpy

- Both provides API to deal with tensor.
- Tensorflow support tensor operation on both GPU and CPU, while Numpy support CPU solely.
- TensorFlow does the computation based on a defined computation graph. (Declarative programming). Numpy can do the computation on-the-fly. (Imperative programming)

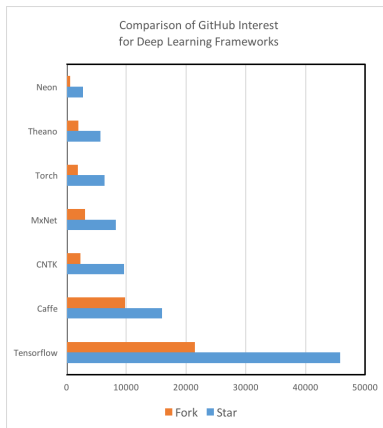
# Other Deep Learning Libraries

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
TensorFlow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	

Extract from <https://svds.com/getting-started-deep-learning/>



# Other Deep Learning Libraries



Extract from <https://svds.com/getting-started-deep-learning/>

# Personal Comments

As of March 2018, I found that most of them support a high level interface similar to scikit-learn, so below is the comments about the pros and cons on the low-level interface.

- TensorFlow:

- + Safe bet for most projects because there is a huge community.
- + TensorBoard for visualization
- – Support declarative programming only. (Imperative programming is supported in Tensorflow1.5, yet it is not stable.)
- – Not easy to learn (if only declarative programming is supported.)
- – Not efficient in terms of runtime and memory allocation.

- MXNet

- + Support both declarative and imperative programming.
- + Efficient in terms of runtime and memory allocation.
- + Support lots of programming languages.
- – Not easy to learn. (Getting better when Gluon is introduced in ver. 1.0.)

# Personal Comments

## ● PyTorch

- + Support both declarative and imperative programming.
- + Efficient in terms of runtime and memory allocation.
- + Easier to learn if you know numpy already.
- – Not many high level interface is supported. We have to write our training code. (Yet, they provides automatic gradient function.)
- – No commercial support. It is in early development stage.
- – Limited tutorials. (The situation will be getting better.)

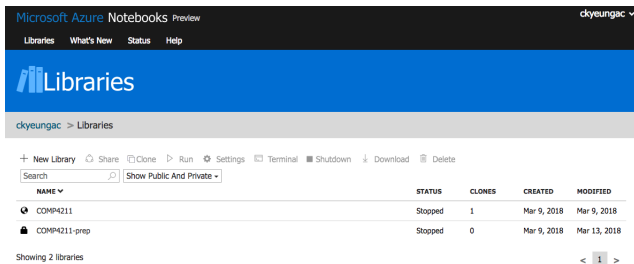
## ● Keras

- + High level interface for TensorFlow/MXNet.
- + Easy to learn. (Similar to scikit-learn.)
- – Runtime performance is bad.
- – Not flexible to make changes in neural network architecture.

# Setting up the working environment

Getting ready for TensorFlow.

- 1 As Azure ML Studio does not support TensorFlow, it is better for us to set-up an environment using Azure Notebook (<https://notebooks.azure.com/>).
- 2 Set up your account in Azure Notebook.
- 3 Go to the libraries page. ([https://notebooks.azure.com/<your\\_username>/libraries](https://notebooks.azure.com/<your_username>/libraries))



- 4 Click “+ New Library”

## After clicking “+ New Library”

- 5 Click “From Github”.
- 6 Do the following configuration in the canvas

Create New Library ? ×

New

From GitHub

GitHub repository

`https://github.com/ckyeungac/COMP4211_Spring2018/`

☐ Clone recursively

Library Name

COMP 4211

Library ID ?

`ckyeungac/libraries/hkust-comp4211`

Import

Cancel

- 7 Click “Import”

# Let's code

Having a taste in TensorFlow.

To better understand today tutorial, the following .ipynb is covered:

- T05\_Single\_Layer\_Neural\_Network\_with\_TensorFlow.ipynb



# Importing TensorFlow

```
import tensorflow as tf  
print(tf.__version__)    # return '1.X.X'
```

# Placeholder

Placeholder is dummy nodes that provide entry points for data to computational graph. Let's say we have a dataset where there are 786 features with 10 labels. We can use placeholder to define the our input.

```
# defining the input to the computation graph  
x = tf.placeholder(tf.float32, shape=[None, 786]) # None means  
                                         not specified  
y_true = tf.placeholder(tf.float32, shape=[None, 10])
```

# Variable

Variable is represent shared, persistent state manipulated by the program. A `tf.Variable` represents a tensor whose value can be changed by running ops on it.

```
# defining the variables to be optimized  
weights = tf.Variable(tf.zeros([784, 10]))  
biases = tf.Variable(tf.zeros([10]))
```

# Tensor Operation

Many tensor operations are available, google it when you need. Here are some useful ones:

```
# These are part of the operations supported by TensorFlow
logits = tf.matmul(x, weights) + biases # z = XW + b
y_pred = tf.nn.softmax(logits) # transform to a probability distribution
y_pred_cls = tf.argmax(y_pred, axis=1) # pick the index with the highest probability
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y_true) # calculate the cross_entropy loss for each sample
cost = tf.reduce_mean(cross_entropy) # take the mean of the cross_entropy
```

# Optimizer

In neural network, there is a cost/loss function you would like to minimize. Optimizer is a handy tool that provide means for you to optimize your network w.r.t your cost function.

```
# defining the optimisation method  
optimizer = tf.train.GradientDescentOptimizer(  
    learning_rate=0.5  
) .minimize(cost)
```

Many different optimization algorithms are supported in Tensorflow<sup>1</sup>, such as MomentumOptimizer, AdamOptimizer, etc..

---

<sup>1</sup>[https://www.tensorflow.org/api\\_guides/python/train](https://www.tensorflow.org/api_guides/python/train).

# Running the Computation Graph

Once your network (computation graph) is defined, we would like to run our network. Let's say we would like to train our network, we would do:

```
# get the batch data
x_batch, y_true_batch = data.train.next_batch(batch_size)

# define the data that is fed to the network by dict
feed_dict_train = {x: x_batch,
                    y_true: y_true_batch}

# pass the feed_dict and run the computation graph
session.run(optimizer, feed_dict=feed_dict_train)

# the above code will run the optimize we defined with the
feed_dict_train.
```

# Appendix

The softmax function is used in various multiclass classification methods. It is defined as

$$\text{softmax}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)}$$

In MNIST dataset, we use the softmax in the output layer, and you can think of it as

$$\hat{\mathbf{y}} = \frac{1}{\sum_{k=0}^9 \exp(z_k)} \begin{bmatrix} \exp(z_0) \\ \exp(z_1) \\ \vdots \\ \exp(z_9) \end{bmatrix} = \begin{bmatrix} P(y=0|\mathbf{x}; \mathbf{W}) \\ P(y=1|\mathbf{x}; \mathbf{W}) \\ \vdots \\ P(y=9|\mathbf{x}; \mathbf{W}) \end{bmatrix}$$

i.e. each output node is a probability.