# Introduction of Theano (1)

Hung-yi Lee

# Introduction

- Theano is a Python library that lets you to define, optimize, and evaluate mathematical expressions.
  - Especially useful for machine learning
- Prerequisite: python and numpy
  - http://cs231n.github.io/python-numpy-tutorial/
- Target of this class:
  - Introduce theano from the beginning, so you can build your own DNN by it

# Installation

# Install Theano

- How to install Theano
  - http://deeplearning.net/software/theano/install.html#install
- Make sure there is no error message when you "import theano" in python
  - E.g.

```
xnor@Speech-GTX-780 ★ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import theano
>>>
```

# Basic

# Review Machine Learning

- ***Define a function set (Model)***: f(x; w)
  - x: input
  - w: model parameters
- ***Define what is the best function***: Define a cost function C(f)
- ***Pick the best function by data***: Training
  - In deep learning, this is usually done by gradient descent.

# Power of Theano

- After defining a cost function, Theano can automatically compute the gradients.
- To use Theano in deep learning you only have to learn
  - How to define a function
  - How to compute the gradient
- …….. Then that's it.

# Define function - Overview

- E.g. Define a function $f(x) = x^2$, then compute f(-2)

```
1 import theano
2
3 x = theano.tensor.scalar()
4 y = x ** 2
5 f = theano.function([x],y)
6
7 print f(-2)
8
```

Type the left code in a file name “xxx.py”

Execute “python xxx.py”

You should get 4

Step 0. Declare that you want to use Theano (line 1)

Step 1. Define input variable x (line 3)

Step 2. Define output variable y (line 4)

Step 3. Declare the function as f (line 5)

Step 4. Use the function f (line 7)

# Define function - Overview

- E.g. Define a function $f(x) = x^2$, then compute f(-2)

```
1 import theano
2
3 x = theano.tensor.scalar()
4 y = x ** 2
5 f = theano.function([x],y)
6
7 print f(-2)
8
```

Type the left code in a file name “xxx.py”

Execute “python xxx.py”

You should get 4

=

```
1 def f(x):
2         return x ** 2
3
4 print f(-2)
5
6
```

So why we define a function by Theano.

It will be clear when we compute the gradients.

# Define function – Step1. Define Input Variables

- A variable can be a scalar, a matrix or a tensor

```
1 import theano
2
3 a = theano.tensor.scalar()
4 b = theano.tensor.matrix()
5 c = theano.tensor.matrix('ha ha ha')
6
7 print a
8 print b
9 print c
```

Line 3: declare a scalar a

Line 4: declare a matrix b

Line 5: declare a matrix c with name “ha ha ha”

The name of a variable only make difference when you try to print the variable.

# Define function – Step1. Define Input Variables

- A variable can be a scalar or a matrix

```
1 import theano
2
3 a = theano.tensor.scalar()
4 b = theano.tensor.matrix()
5 c = theano.tensor.matrix('ha ha ha')
6
7 print a
8 print b
9 print c
```

Line 7,8,9: let's print the three variables a, b, c to see what we get

```
<TensorType(float64, scalar)>
<TensorType(float64, matrix)>
ha ha ha
```

**a, b, c are symbols without any values**

# Define function – Step1. Define Input Variables

- A variable can be a scalar or a matrix

```
1 import theano
2
3 a = theano.tensor.scalar()
4 b = theano.tensor.matrix()
5 c = theano.tensor.matrix('ha ha ha')
6
7 print a
8 print b
9 print c
```

=

```
1 import theano
2 import theano.tensor as T
3
4 a = T.scalar()
5 b = T.matrix()
6 c = T.matrix('ha ha ha')
7
```

simplification

# Define function – Step2. Define Output Variables

- Output variables are defined based on their relations with the input variables
- Below are some examples

```
 3 x1 = T.scalar()
 4 x2 = T.scalar()
 5 x3 = T.matrix()
 6 x4 = T.matrix()
 7
 8 y1 = x1 + x2
 9
10 y2 = x1 * x2
11
12 y3 = x3 * x4
13
14 y4 = T.dot( x3 , x4 )
15
```

y1 equals to x1 plus x2

y2 equals to x1 times x2

y3 is the ***elementwise*** multiplication of x3 and x4

y4 is the matrix multiplication of x3 and x4

# Define function – Step 3. Declare Function

```
f = theano.function([x],y)
```

Declare the function as f

Function input

Function output

Note: the input of a function should be a list.

That is, always put the input in "[ ]"

```
f = theano.function(
        inputs=[x],
        outputs=y
        )
```

Define the function input and output explicitly.

(equivalent to the above usage)

# Define function – Step 3. Declare Function

```
 1 import theano
 2 import theano.tensor as T
 3
 4 x1 = T.scalar()
 5 x2 = T.scalar()
 6
 7 y1 = x1 * x2
 8 y2 = x1 ** 2 + x2 ** 0.5
 9
10 f = theano.function([x1, x2],[y1, y2])
11
12 z =  f(2,4)
13 print z
```

# Define function – Step 4. Use Function

```
 1 import theano
 2 import theano.tensor as T
 3
 4 x1 = T.scalar()
 5 x2 = T.scalar()
 6
 7 y1 = x1 * x2
 8 y2 = x1 ** 2 + x2 ** 0.5
 9
10 f = theano.function([x1, x2],[y1, y2])
11
12 z =  f(2,4)
13 print z
```

Line 12: simply use the function f you declared as a normal python function

Line 13: Print the function output -> `[array(8.0), array(6.0)]`

(The theano function output is a `numpy.ndarray`.)

# Define function
# - Examples for Matrix

```
 1 import theano
 2 import theano.tensor as T
 3
 4 a = T.matrix()
 5 b = T.matrix()
 6
 7 c = a * b
 8 d = T.dot( a , b )
 9
10 F1 = theano.function([a, b],c)
11 F2 = theano.function([a, b],d)
12
13 A = [[1, 2],[3, 4]]
14 B = [[2, 4],[6, 8]]
15 C = [[1, 2],[3, 4],[5, 6]]
16
17 print F1( A , B )
18 print F2( C , B )
```

Output:

```
[[  2.   8.]
 [ 18.  32.]]
[[ 14.  20.]
 [ 30.  44.]
 [ 46.  68.]]
```

Be careful that the dimensions of the input matrices should be correct.

# Compute Gradients

- Computing the gradients with respect to a variable is so simple.
- Given a function with input variable x and output variable y
  - To compute dy/dx, simply g = T.grad( y , x )
  - Note: To compute the gradient, y should be a scalar.
- That's it!

# Compute Gradients - Example 1

```
 1 import theano
 2 import theano.tensor as T
 3
 4 x = T.scalar('x')
 5 y = x ** 2
 6 g = T.grad( y , x )
 7
 8 f = theano.function( [x] , y )
 9 f_prime = theano.function( [x] , g )
10
11 print f(-2)
12 print f_prime(-2)
```

g = dy/dx = 2x

output “-4”

# Compute Gradients - Example 2

```
 1 import theano
 2 import theano.tensor as T
 3
 4 x1 = T.scalar()
 5 x2 = T.scalar()
 6
 7 y = x1 * x2
 8 g = T.grad(y,[x1,x2])
 9
10 f = theano.function([x1, x2] , y)
11 f_prime = theano.function([x1, x2] , g)
12
13 print f(2,4)
14 print f_prime(2,4)
```

compute the gradients with respect to multiple variables

$g = [\ \partial y/\partial x1\ ,\partial y/\partial x2\ ]$
$= [\ x2\ ,\ x1\ ]$

# Compute Gradients - Example 3

$$If\ A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

$$If\ B = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$$

(Note that the dimensions of A and B is not necessary 2 X 2. Here is just an example.)

```
1  import theano
2  import theano.tensor as T
3
4  A = T.matrix()
5  B = T.matrix()
6
7  C = A * B
8  D = T.sum(C)
9
10 g = T.grad( D , A )
11
12 y_prime = theano.function( [A, B], g )
13
14 A = [[1, 2],[3, 4]]
15 B = [[2, 4],[6, 8]]
16
17 print y_prime(A,B)
```

$$C = \begin{bmatrix} a_1 b_1 & a_2 b_2 \\ a_3 b_3 & a_4 b_4 \end{bmatrix}$$ (line 7)

$$D = a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4$$ (line 8)

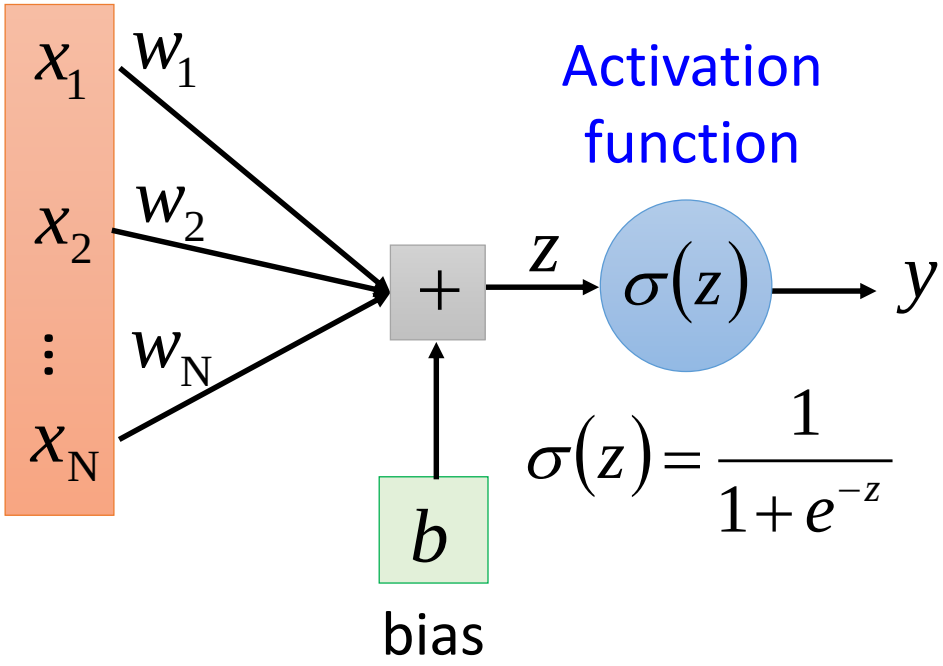
$$g = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$$ (line 10)

You cannot compute the gradients of C because it is not a scalar

# Single Neuron

# Single Neuron

- First, let's implement a neuron




In this stage, let's assume the model parameters w and b are known

```
 1 import theano
 2 import theano.tensor as T
 3 import random
 4
 5 x = T.vector()
 6 w = T.vector()
 7 b = T.scalar()
 8
 9 z = T.dot(w,x) + b
10 y = 1/( 1 + T.exp( -z ) )
11
12 neuron = theano.function(
13         inputs=[x,w,b],
14         outputs=[y]
15         )
16
17 w = [-1, 1]
18 b = 0
19 for i in range(100):
20         x = [random.random(), random.random()]
21         print x
22         print neuron(x,w,b)
```

y = neuron(x; w, b)

# Single Neuron – Shared Variables

- In the last example, a neuron is a function with input x, w and b.
- However, we usually only consider x as ***input***. w and b are ***model parameters***.
  - It would be more intuitive if we only have to write "neuron(x)" when using a neuron
  - The model parameters w and b still influence neuron(.), but in an implicit way.
- In Theano, the model parameters are usually stored as ***shared variables***.

```
 1 import theano
 2 import theano.tensor as T
 3 import random
 4 import numpy
 5
 6 x = T.vector()
 7 w = theano.shared(numpy.array([1.,1.]))
 8 b = theano.shared(0.)
 9
10 z = T.dot(w,x) + b
11 y = 1/( 1 + T.exp( -z ) )
12
13 neuron = theano.function(
14           inputs=[x],
15           outputs=y
16           )
17
18 print w.get_value()
19 w.set_value([0.,0.])
20
21 for i in range(100):
22           x = [random.random(), random.random()]
23           print x
24           print neuron(x)
```

w and b are declared as shared variables.

Initial value

(The shared variables are not symbols.)

x is the only input

The function can access the shared variables.

To get or change the values of the shared variables, you have to use “get_value()” and “set_value()”.

# *Single Neuron – Training*

- Define a cost function C
- Then compute $\left[\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \cdots, \frac{\partial C}{\partial w_N}\right]$ and $\frac{\partial C}{\partial b}$

```
 5 x = T.vector()
 6 w = theano.shared(numpy.array([-1.,1.]))
 7 b = theano.shared(0.)
 8
 9 z = T.dot(w,x) + b
10 y = 1/( 1 + T.exp( -z ) )
11
12 neuron = theano.function(
13         inputs=[x],
14         outputs=y
15         )
16
17 y_hat = T.scalar()
18 cost = T.sum((y-y_hat)**2)
19
20 dw,db = T.grad(cost,[w,b])
21
22 gradient = theano.function(
23         inputs=[x,y_hat],
24         outputs=[dw,db]
25         )
```

Reference output value

Define Cost

Computing Gradients

Declare the function for computing gradients

# *Single Neuron – Gradient Descent*

$$w_1 \leftarrow w_1 - \eta \frac{\partial C}{\partial w_1}, \ldots, w_N \leftarrow w_N - \eta \frac{\partial C}{\partial w_N}, b \leftarrow b - \eta \frac{\partial C}{\partial b}$$

## *Tedious Way:*

```
27 x = [1, -1]
28 y_hat = 1
29 for i in range(100):
30         print neuron(x)
31         dw,db = gradient(x,y_hat)
32         w.set_value( w.get_value() - 0.1 * dw )
33         b.set_value( b.get_value() - 0.1 * db )
34         print w.get_value(),b.get_value()
```

Compute gradient

Update Parameters

Line 31: use the function gradient (defined in the last page) to compute the gradients

Line 32, 33: use the gradients to update the model parameters

# *Single Neuron – Gradient Descent*

$$w_1 \leftarrow w_1 - \eta \frac{\partial C}{\partial w_1}, \ldots, w_N \leftarrow w_N - \eta \frac{\partial C}{\partial w_N}, b \leftarrow b - \eta \frac{\partial C}{\partial b}$$

## *Effective Way:*

```
22 gradient = theano.function(
23         inputs=[x,y_hat],
24         updates=[(w,w-0.1*dw),(b,b-0.1*db)]
25         )
26
27 x = [1, -1]
28 y_hat = 1
29 for i in range(100):
30         print neuron(x)
31         gradient(x,y_hat)
32         print w.get_value(),b.get_value()
```

Line 24: updates="a list of pairs"

Each pair is in the form of (shared-variable, an expression).

Whenever this function runs, it will replace the value of each shared variable with the result of the expression.

# Single Neuron – Gradient Descent

$$w_1 \leftarrow w_1 - \eta \frac{\partial C}{\partial w_1}, \ldots, w_N \leftarrow w_N - \eta \frac{\partial C}{\partial w_N}, b \leftarrow b - \eta \frac{\partial C}{\partial b}$$

## Effective Way:

```
23 def MyUpdate(paramters,gradients):
24         mu = 0.1
25         parameters_updates = \
26         [ (p , p - mu * g) for p,g in izip(paramters,gradients) ]
27         return parameters_updates
28
29 gradient = theano.function(
30         inputs=[x,y_hat],
31         updates=MyUpdate([w,b],[dw,db])
32         )
```

In deep learning, usually sophisticated update strategy is needed.

In this case, you may want to use a function to return the pair list for parameter update.

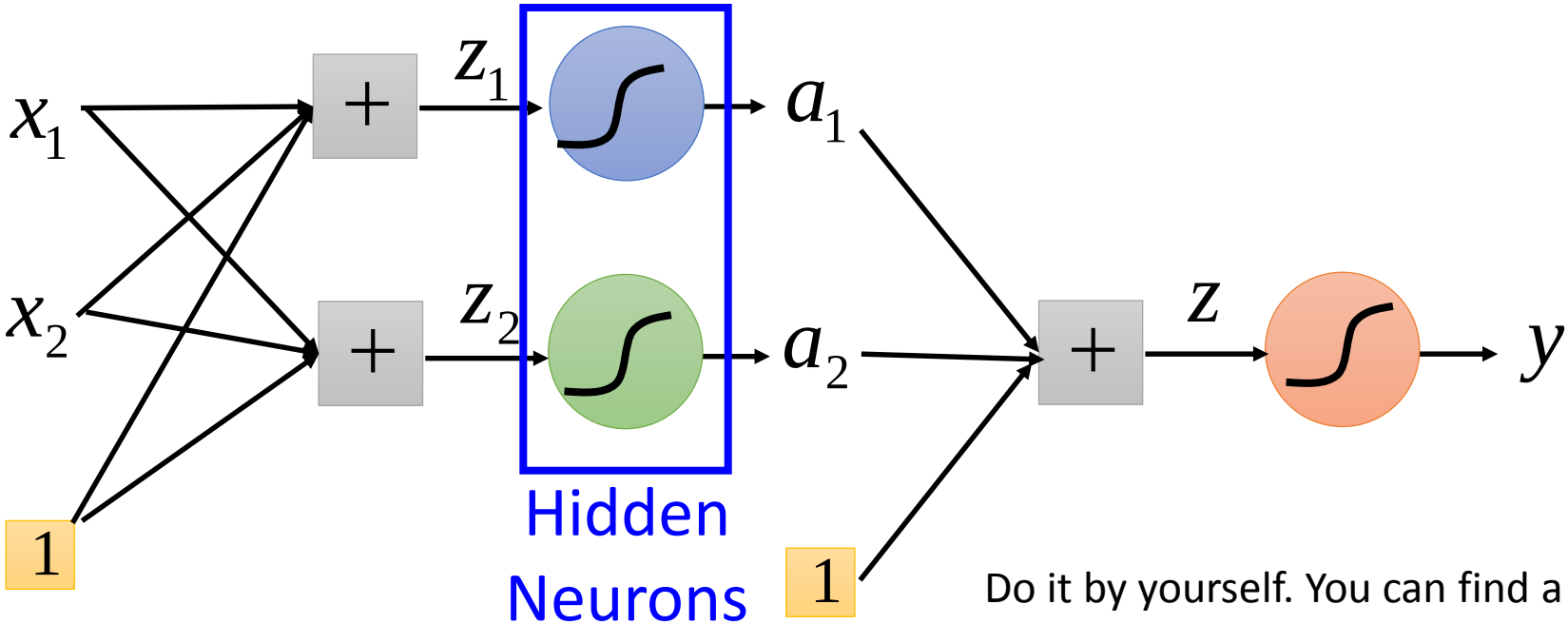
What is izip?
https://docs.python.org/2/library/itertools.html#itertools.izip

# Tiny Neural Network

# XOR gate

Can you use three neurons to simulate an XOR gate?

| Input | | Output |
|---|---|---|
| $x_1$ | $x_2$ | y |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




Do it by yourself. You can find a sample code in the following pages.

# XOR gate

```
 1 import theano
 2 import theano.tensor as T
 3 import numpy
 4 from itertools import izip
 5
 6 x = T.vector()
 7 w1 = theano.shared(numpy.random.randn(2))
 8 b1 = theano.shared(numpy.random.randn(1))
 9 w2 = theano.shared(numpy.random.randn(2))
10 b2 = theano.shared(numpy.random.randn(1))
11 w = theano.shared(numpy.random.randn(2))
12 b = theano.shared(numpy.random.randn(1))
13
14 a1 = 1/( 1 + T.exp( -1 * (T.dot(w1,x) + b1 )) )
15 a2 = 1/( 1 + T.exp( -1 * (T.dot(w2,x) + b2 )) )
16 y = 1/( 1 + T.exp( -1 * (T.dot(w,[a1,a2]) + b )) )
17
18 y_hat = T.scalar()
19 cost = - (y_hat*T.log(y)+(1-y_hat)*T.log(1-y)).sum()
20
21 dw,db,dw1,db1,dw2,db2 = T.grad(cost,[w,b,w1,b1,w2,b2])
```

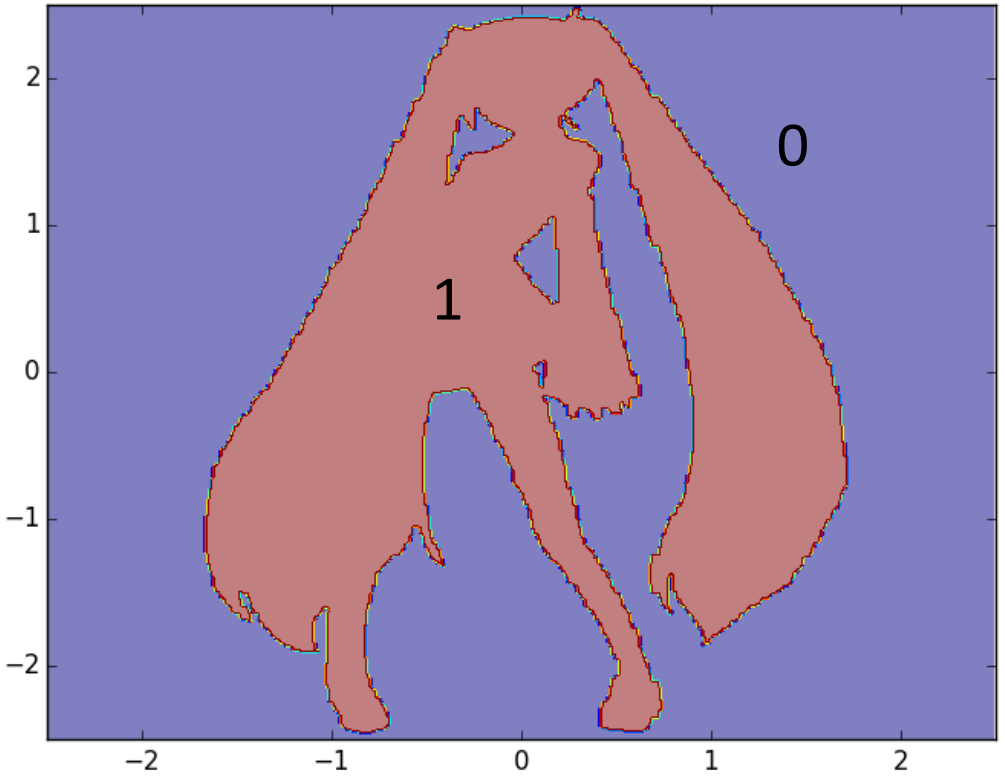
# XOR gate

```
23 def MyUpdate(paramters,gradients):
24         mu = 0.01
25         parameters_updates = \
26         [(p,p-mu*g) for p,g in izip(paramters,gradients)]
27         return parameters_updates
28
29 g = theano.function(
30 inputs=[x,y_hat],
31 outputs=[y,cost],
32 updates=MyUpdate([w,b,w1,b1,w2,b2],[dw,db,dw1,db1,dw2,db2])
33         )
34
35 for i in range(100000):
36         y1,c1 = g([0, 0], 0)
37         y2,c2 = g([0, 1], 1)
38         y3,c3 = g([1, 0], 1)
39         y4,c4 = g([1, 1], 0)
40         print c1+c2+c3+c4
41         print y1,y2,y3,y4
```

(continued from previous page)

# Neural Network

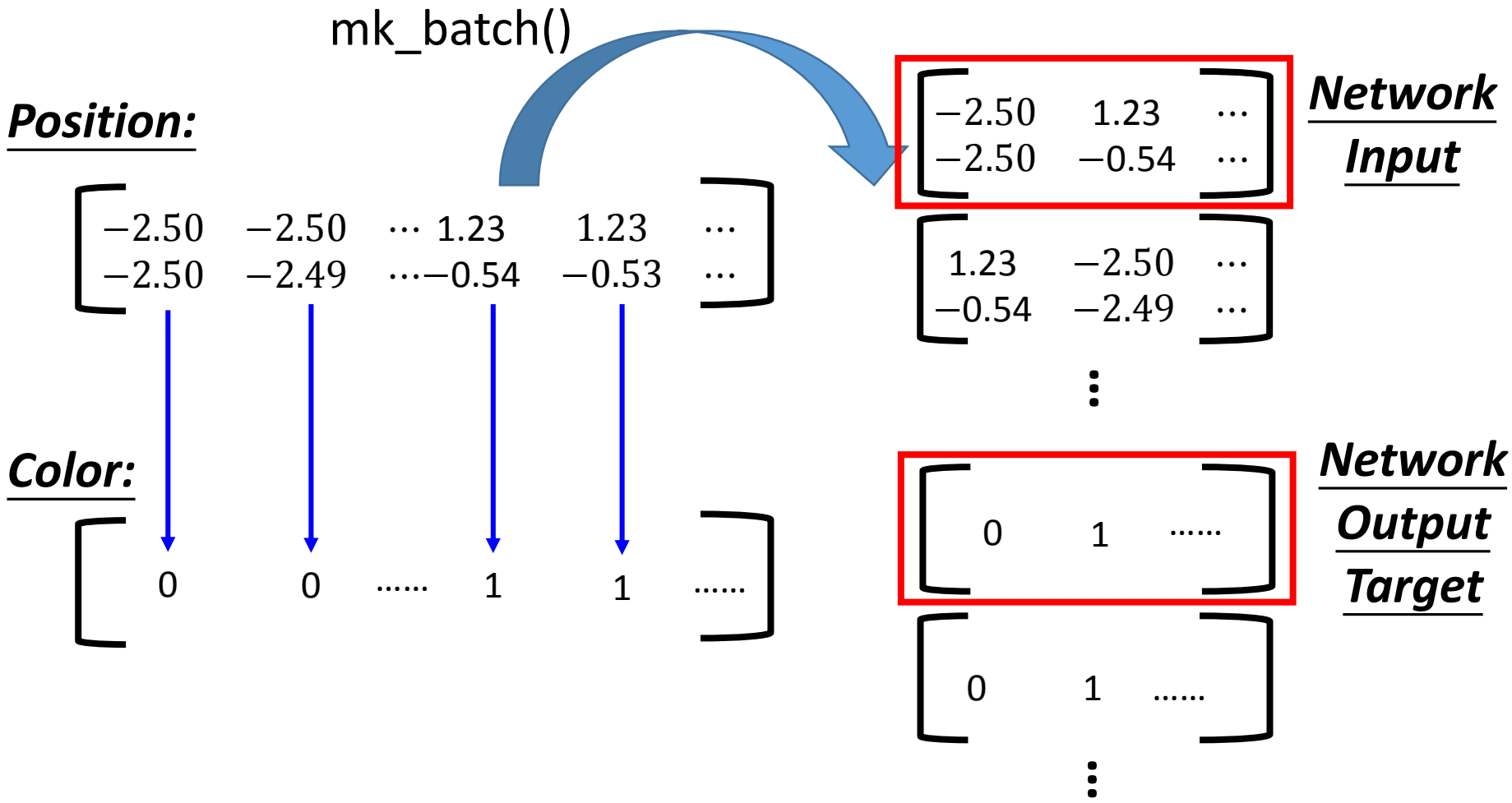
# Function of Miku



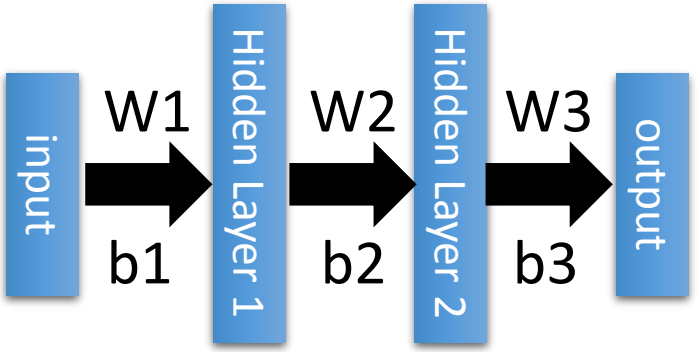

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/theano/miku

(1st column: x, 2nd column: y, 3rd column: output (1 or 0) )

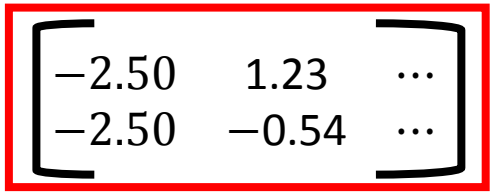
# Make Minibatch



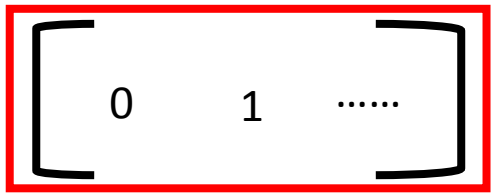
# Defining Network




- Declare network Input
  - x = T.matrix('input',dtype='float32')
- Declare network output
  - y_hat = T.matrix('reference',dtype='float32')
- Declare network parameters
  - W1 = … (matrix), W2 = …, W3 = …
  - b1 = … (vector), b2 = … (vector), b3 = … (vector)
  - parameters = [W1,W2,W3,b1,b2,b3]

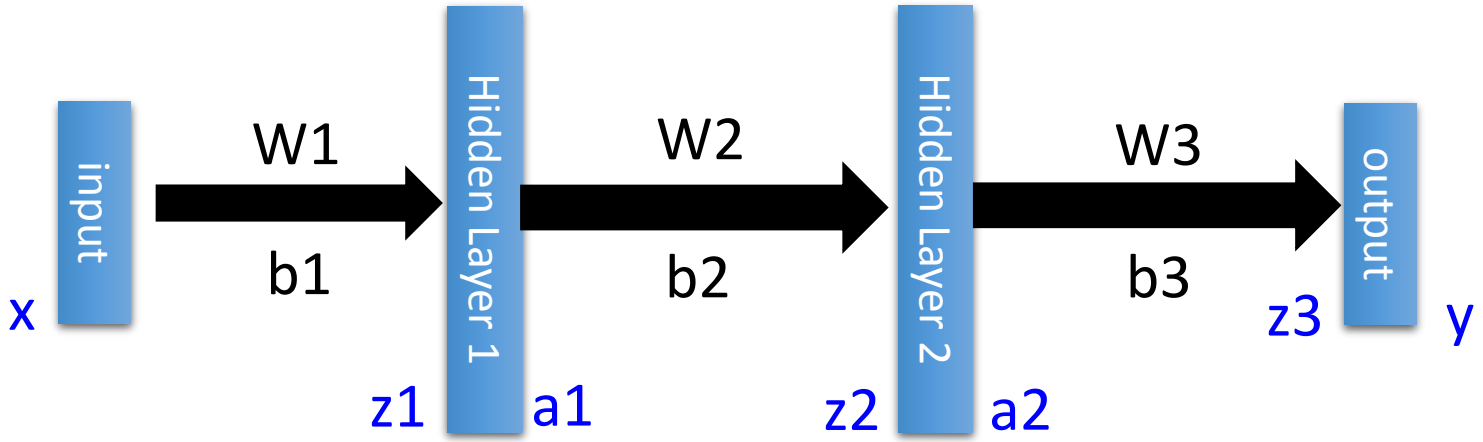



***Network Input***




***Network Output Target***

# Defining Network

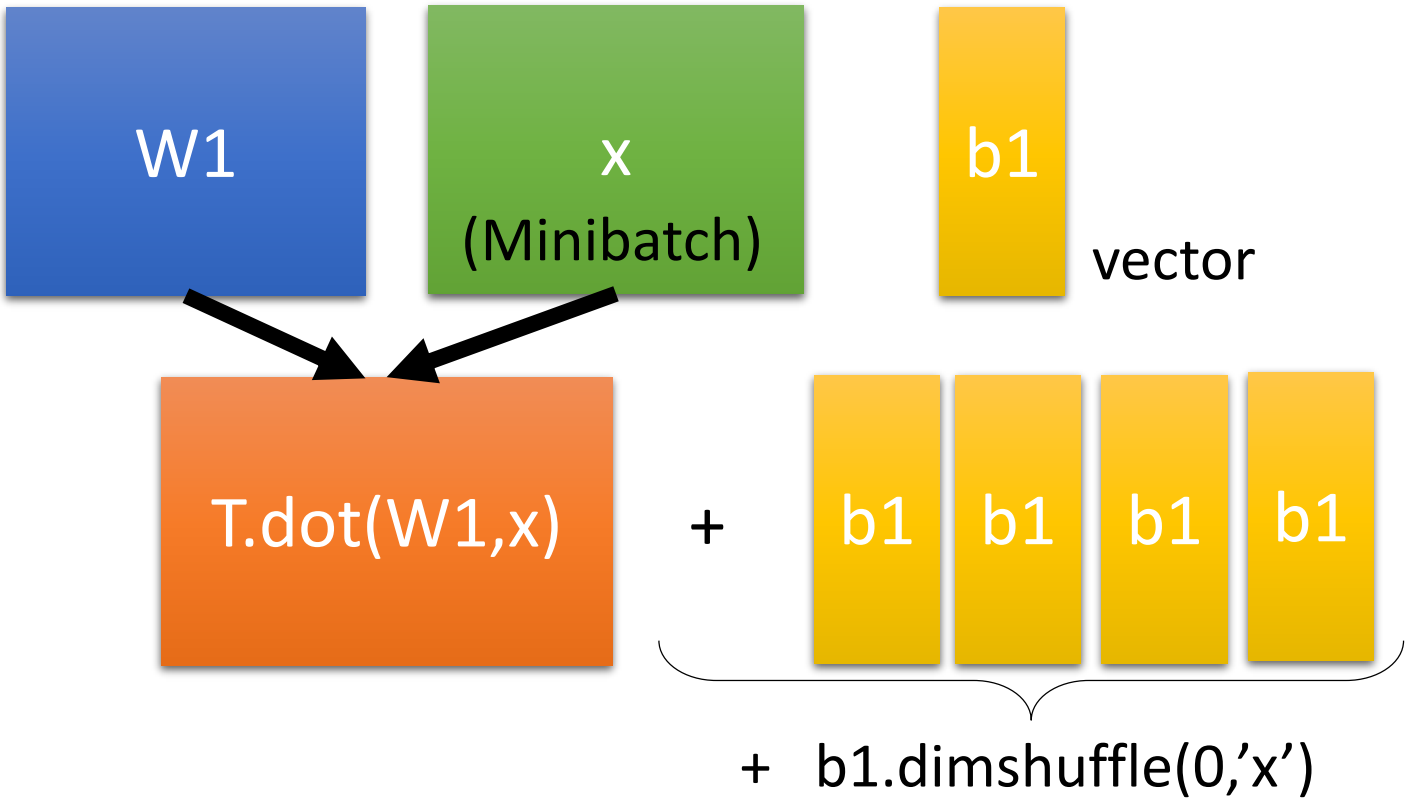



```
z1 = T.dot(W1,x) + b1.dimshuffle(0,'x')
a1 = 1/(1+T.exp(-z1))
z2 = T.dot(W2,a1) + b2.dimshuffle(0,'x')
a2 = 1/(1+T.exp(-z2))
z3 = T.dot(W3,a2) + b3.dimshuffle(0,'x')
y = 1/(1+T.exp(-z3))
```

# Minibatch

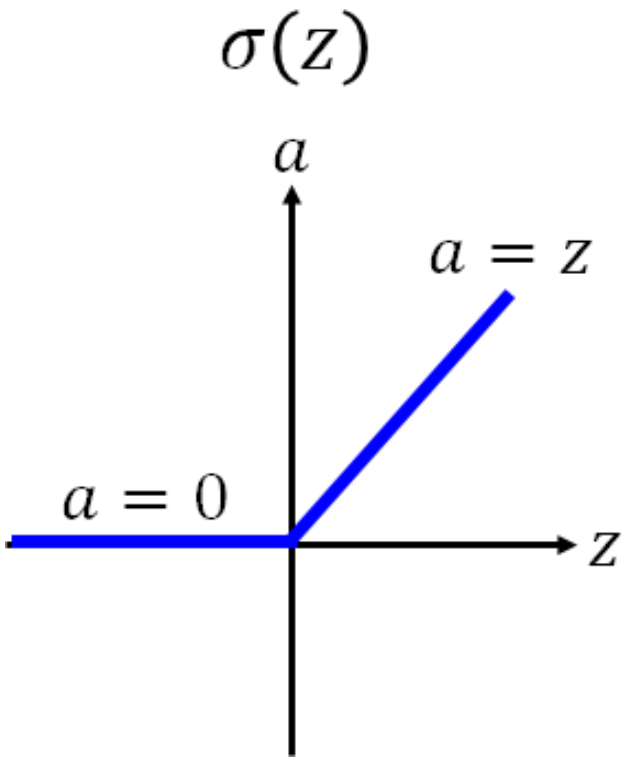
Ref:
https://theano.readthedocs.org/en/rel-0.6rc3/library/tensor/basic.html

```
z1 = T.dot(W1,x) + b1.dimshuffle(0,'x')
```




More about computing: http://videolectures.net/deeplearning2015_coates_deep_learning/

# Activation Function - ReLU




If z<0: a = 0
else: a = z

a = T.switch(z<0,0,z)

***Other Approaches:***

a = T.maximum(z,0)

a = z * (z > 0)

# Declaring Functions

```
cost = T.sum( (y - y_hat)**2 ) / batch_size
```

```
gradients = T.grad(cost,parameters)
```

Where is backpropagation?

Theano will do backpropagation automatically

```
train = theano.function(
        inputs=[x,y_hat],
        updates=MyUpdate(parameters,gradients),
        outputs=cost
        )
```

```
test = theano.function(
        inputs=[x],
        outputs=y
        )
```

# Training & Testing

- Training:

100 epochs

Generating Batches

```
for t in range(100):
        cost = 0
        X_batch, Y_hat_batch \
                = mk_batch(X_all,Y_hat_all,data_size,batch_number)
        for i in range(batch_number):
                cost += train(X_batch[i],Y_hat_batch[i])
        cost /= batch_number
        print cost
```

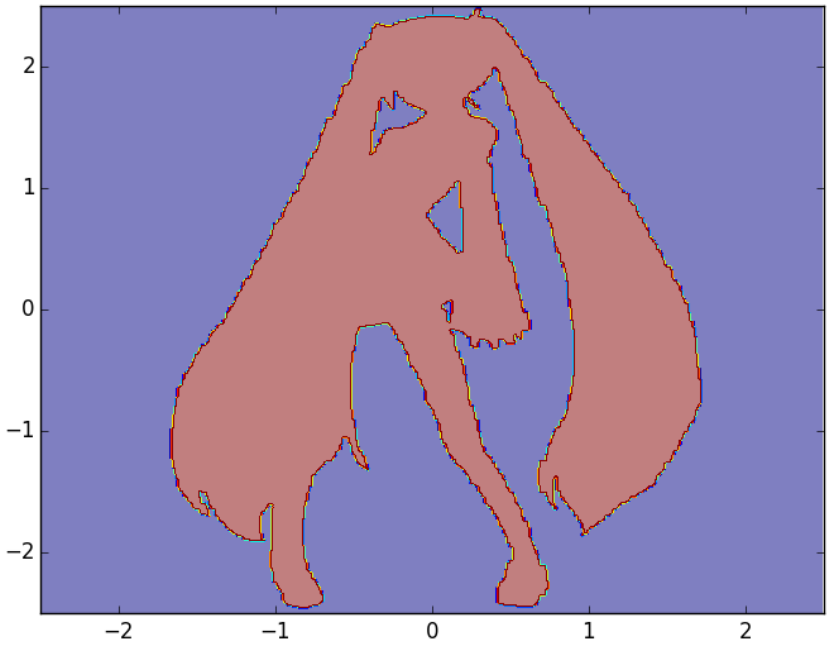
Update parameters after seeing each batch

In real learning problem, you should use a validation set to control when to stop.

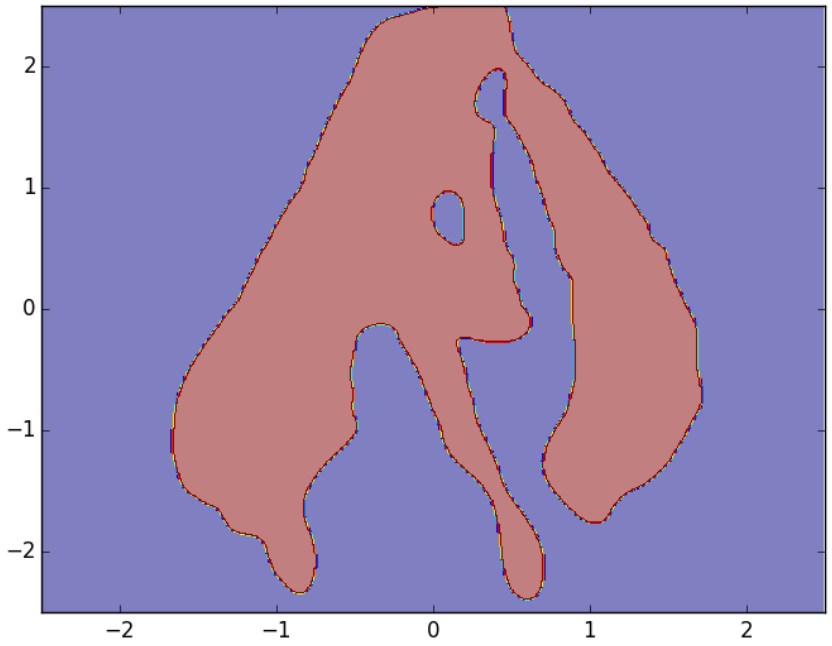
- Testing: `Y = test(X_all)`

# Results

Reference




Network Output




Network Structure: 2-1000-1000-1

Batch size: 1000

Activation function: sigmoid

Learning rate: 0.01 (fixed)

# Configuration

# Configuration

- There are three ways to set the configuration of Theano:
  - 1. modify $HOME/.theanorc
  - 2. set theano.config.<property> in your python code
  - 3. use THEANO_FLAGS when running your python code
    - E.g. THEANO_FLAGS=mode=FAST_RUN,device=gpu python YourCode.py
- Ref: http://deeplearning.net/software/theano/library/config.html

# Configuration - GPU

- test_gpu.py

  Multiplying two 10000 X 10000 matrices

```
 1 import theano
 2 import theano.tensor as T
 3 import numpy
 4 import time
 5
 6 X = T.matrix()
 7 Y = T.matrix()
 8 Z = T.dot(X,Y)
 9 f = theano.function([X,Y],Z)
10
11 x = numpy.random.randn(10000,10000)
12 y = numpy.random.randn(10000,10000)
13 tStart = time.time()
14 z = f(x,y)
15 tEnd = time.time()
16 print "It cost %f sec" % (tEnd - tStart)
```

Command: `python test_gpu.py`

Output: `It cost 21.249996 sec`

If the machine has multiple GPUs, using gpu0, gpu1 …. to select a card.

Command: `THEANO_FLAGS=device=gpu python test_gpu.py`

Output: `It cost 21.045571 sec`

GPU is not helpful? Why?

# Configuration - GPU

```
 1 import theano
 2 import theano.tensor as T
 3 import numpy
 4 import time
 5
 6 X = T.matrix(dtype='float32')
 7 Y = T.matrix(dtype='float32')
 8 Z = T.dot(X,Y)
 9 f = theano.function([X,Y],Z)
10
11 x = numpy.random.randn(10000,10000).astype(dtype='float32')
12 y = numpy.random.randn(10000,10000).astype(dtype='float32')
13 tStart = time.time()
14 z = f(x,y)
15 tEnd = time.time()
16 print "It cost %f sec" % (tEnd - tStart)
17
```

Command: `THEANO_FLAGS=device=gpu python test_gpu.py`

Output: `It cost 0.843893 sec` More than 20 times faster

Why?

Ref: http://deeplearning.net/software/theano/tutorial/using_gpu.html

# More about Configuration

- mode=DEBUG_MODE
  - Theano provides information to help you debug
- profile=true
  - Theano will analyze your program, showing a breakdown of how much is spent on each operation.

# To Learn More …

- Theano Exercises
  - https://github.com/goodfeli/theano_exercises
- Theano Tutorial
  - http://deeplearning.net/software/theano/tutorial/index.html#tutorial

# Acknowledgement


- 感謝 陳俊宏 同學發現投影片 上的錯誤