AI abc
An Introduction to Machine Learning
algorithm, big data, coding

劉晉良
Jinn-Liang Liu
清華大學計算與建模科學研究所
Institute of Computational and Modeling Science
National Tsing Hua University, Taiwan

Oct. 2, 2019

Abstract

Based on Google's MNIST for ML (Machine Learning) beginners, I introduce a basic knowledge (**abc**) of Supervised ML (an important part of Artificial Intelligence) from the perspective of **a**lgorithm, **b**ig data, and **c**oding. The audience can acquire some fundamentals of SML in ***one hour*** that include mathematical properties of Learning, algorithmic approaches to deal with Big Data, and a glimpse of Python Coding in AI. It is hoped that one can briefly understand the Python code given in the talk and run it successfully in ***one day***. This may help one to ponder whether one should spend her or his ***life*** to pursue AI. MNIST is a database of handwritten digits created by Yann LeCun, a pioneer in modern AI, and is short for Mixed National Institute of Standards and Technology.

Visit my Webpage

- AI abc: An Introduction to Machine Learning
- Gradient Descent and Backpropagation in Machine Learning (Automatic Differentiation: Forward & Reverse Modes, Jacobian)
- Convolution in Machine Learning (Convolution)

Part I Computer Programming (Browse and Use) GitHub

1. Colab: Proj1: tf3.ipynb (MNIST Project)
   A. Style Transfer (YouTube2, Code2), B. Time Series (YouTube3), C. YOLO (YouTube4, Code3), D. Stock (YouTube5, Code4), E. Game (OpenAI RL Code5)
2. TensorFlow, TensorBoard (YouTube1)
3. PyTorch (PyTorch Autograd, PyTorch入門, MNIST.ipynb, MNIST.py)
4. Python Programming
5. C++ Programming

A. Cloud Computing by Colab: Google Chrome => Login Google Account => Click tf1.ipynb => 選擇開啟工具 => Google Colaboratory => Click Triangle (Run Cell) => Done! => Proj1 => Click tf3.ipynb (MNIST Project) => Run => Done!

B. Local Computing by jupyter: Install Anaconda3-4.2.0 (or more Anaconda) => Anaconda Navigator => Environments => Install TensorFlow => Home => Launch jupyter => jupyter => Files on 筆電 => Click on tf1.ipynb => Run tf1.ipynb (Done!)

How to run py code on jupyter: tf1.py => Creat a new file tf1.ipynb with only one line “import tf1” => Run the cell of “import

Part II Supervised Learning (Read and Work)

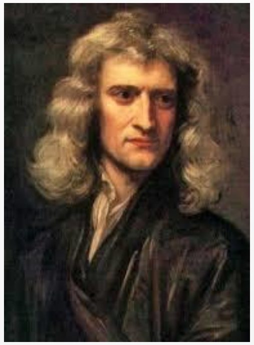
1. A Simple Learning Model: Classification, Target, Hypothesis, Training Data, Learning Algorithm, Weights, Bias, Supervised and Unsupervised Learning
2. Google Tutorial for ML Beginners: Image Recognition, MNIST, Softmax Regression (92%), Cross Entropy, Gradient Descent, Back Propagation, Computation Graph (TF mnist 1.0)
3. Tensorflow and Deep Learning I (by Martin Gorner) : Deep Learning Network, ReLU, Learning Rate (98%), Overfitting, Dropout (98.2%), Convolutional Neural Network (99.3%) (TF mnist 3.1)
4. Tensorflow and Deep Learning II (by Martin Gorner) (RNN1): Batch Normalization (99.5%) (TF mnist 4.2), Data Whitening, Fully Connected Network, TensorFlow API, MNIST Record (Kaggle: 100%), Recurrent Neural Network, Deep RNN, Long Short Term Memory, Gated Recurrent Unit, Language Model

Part III Theories of Deep Learning

1. Lectures at MIT (Book: Deep Learning by Goodfellow, Bengio, Courville)
2. Lectures at Chicago
3. Lectures at Stanford

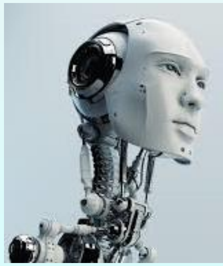
Planning and Learning

Planning : $y = f(x) = \mathrm{a}x^2 + \mathrm{b}x + \mathrm{c}$, f : **known**
x : input, y : output, a, b, c : **known**
x, y : variables

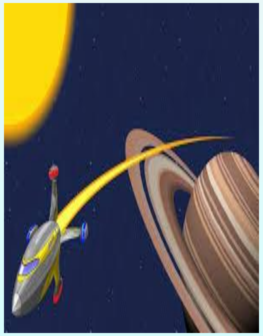


$F = ma$ → $f(x, t)$ →

Learning : $y = f(x) = \mathrm{a}x^2 + \mathrm{b}x + \mathrm{c}$, f : **unknown**
x : input, y : output, a, b, c : **unknown**
Learn a, b, c (regression parameters)
# of parameters : 1,000,000,000,000

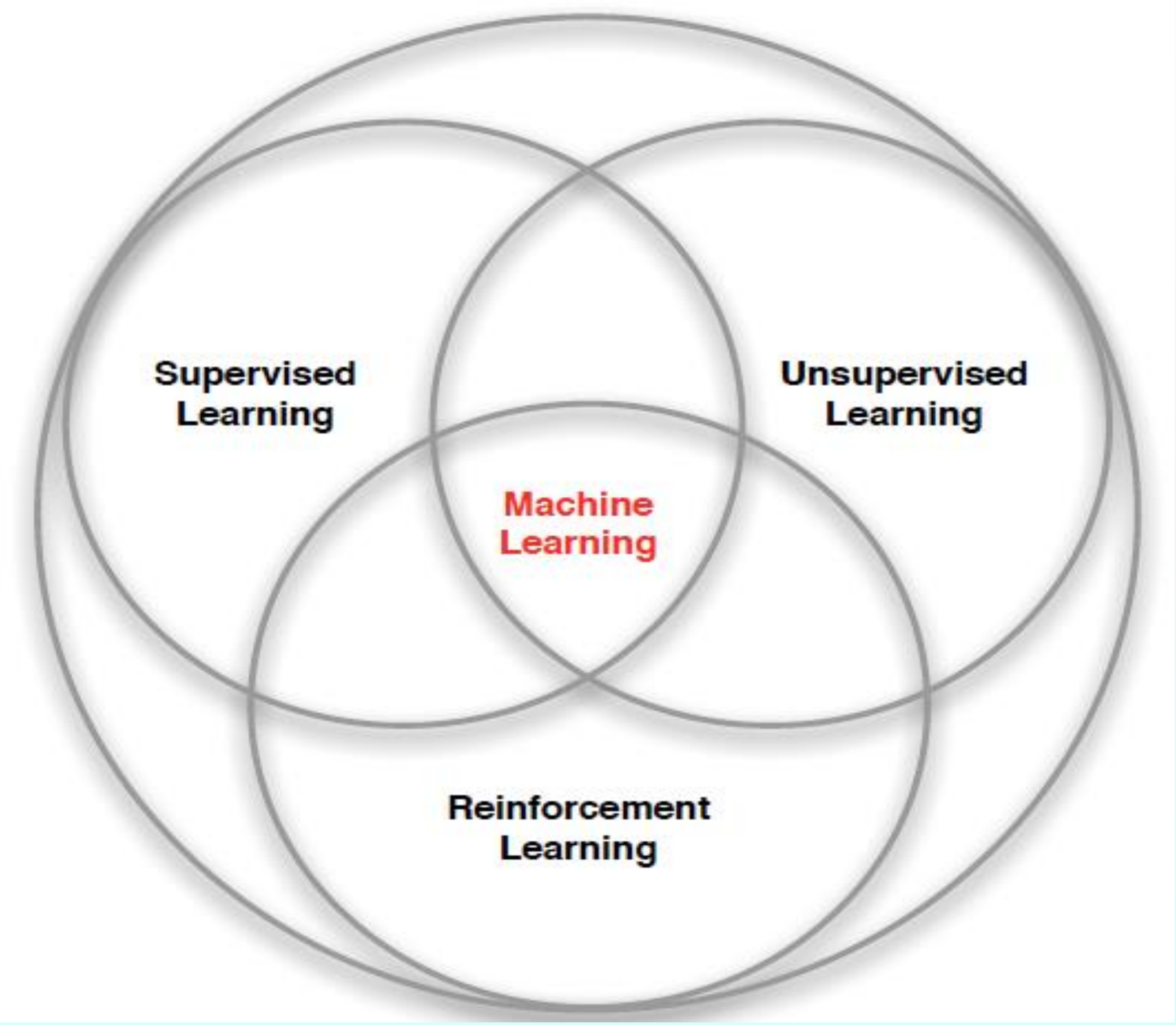


$y = wx + b$ → $f(x, t)$ →



聽說讀寫食衣住行育樂醫金..., a^2bc.

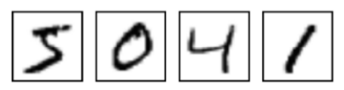
A



I

from *UCL Course on RL*, David Silver

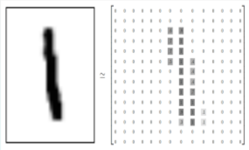
Google TensorFlow MNIST for ML Beginners



MNIST : Mixed National Institute of Standards and Technology database (training: 55,000 images; testing: 10,000; validating: 5,000) by Yann LeCun

input x =  ⟹ **algorithm** $y = wx + b$ ⟹ **output y = 5**

70,000 x data points; 10 y labels: 0, 1, 2, …, 9

x =  **784 pixels (intensities)** ⟹ **x a vector in $[0,1]^{784}$**

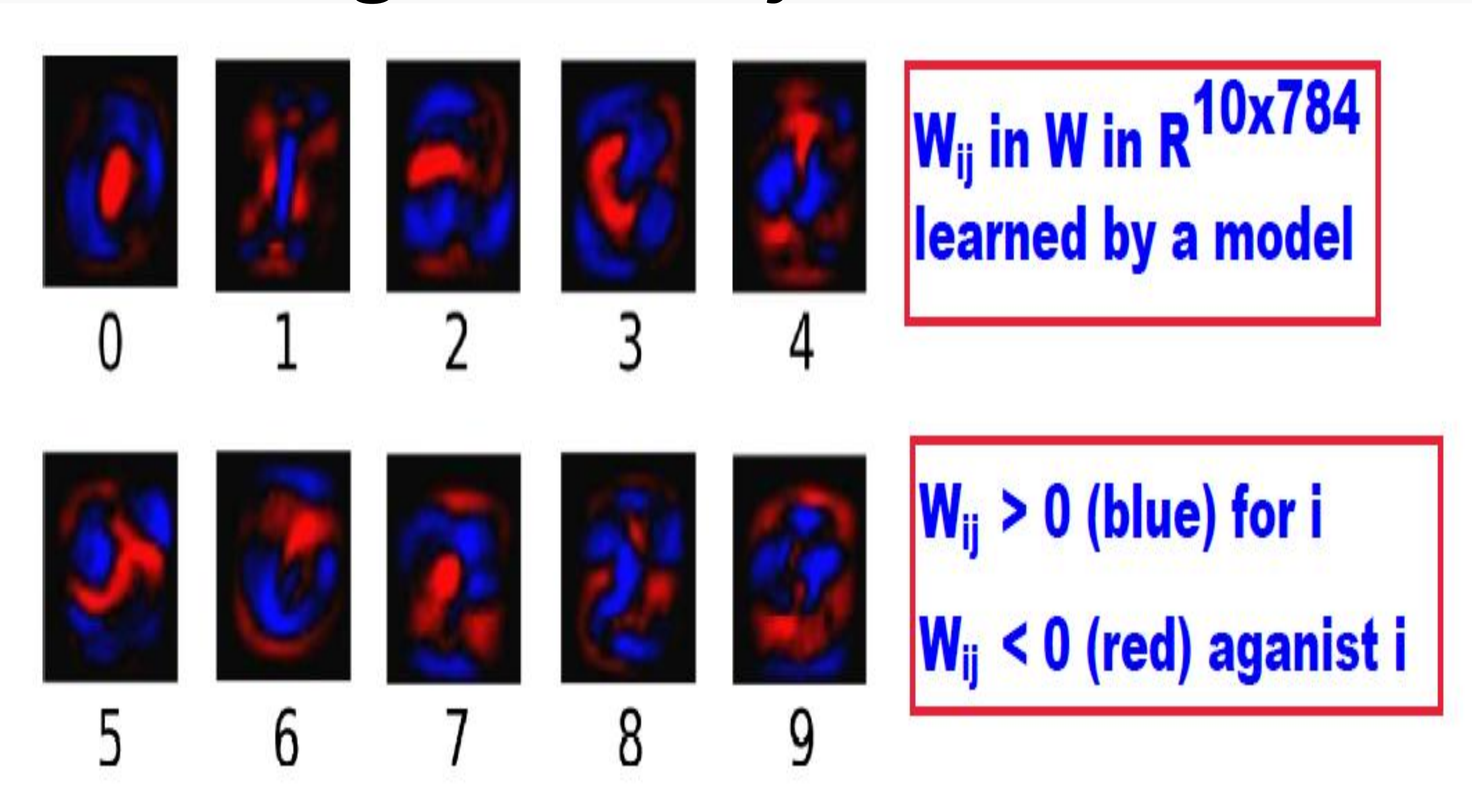
y a one-hot vector in $\{0,1\}^{10}$ ⟹ **w a matrix in $R^{10 \times 784}$**

big data ⟹ **10×784 enough?** ⟹ **$100 \times 10 \times 784$?** ⟹

big W ⟹ **Deep Learning** **scalar, vector, matrix, tensor**

What is *W* (learned parameter)? What do you want to learn?

Algorithm 1: $y = Wx + b$



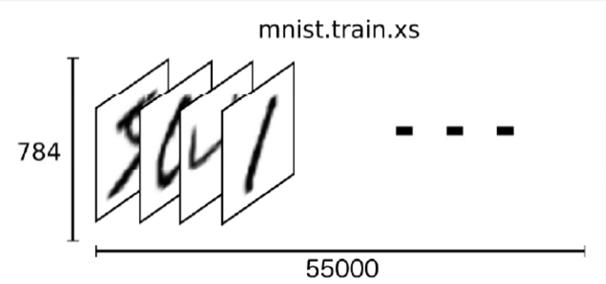
TensorFlow MNIST Code

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])
```

$y = Wx + b$ **y in $\{0,1\}^{10}$**

***x* vector in $[0,1]^{784}$**

***W* matrix in $R^{10 \times 784}$**




```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

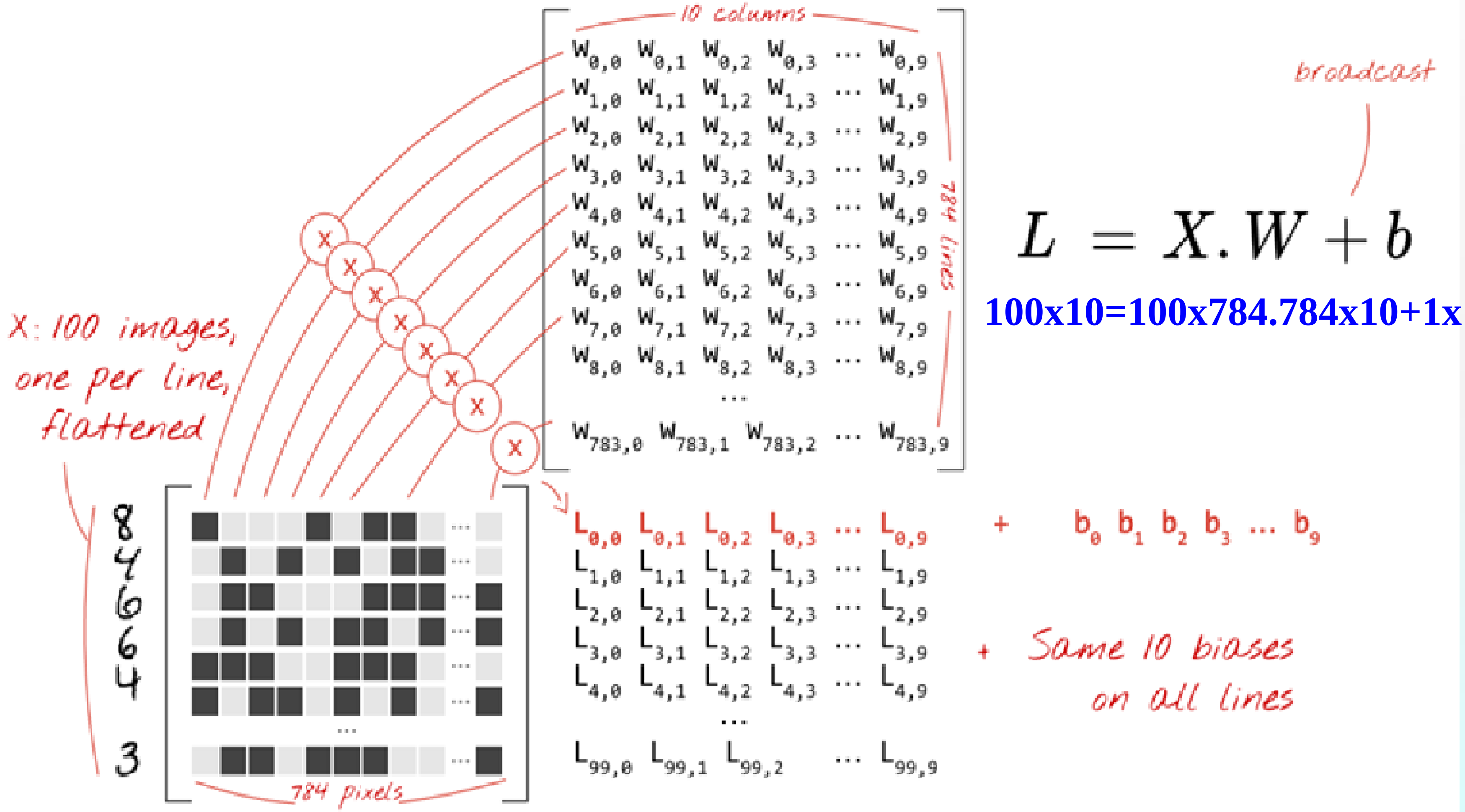
p

xW not Wx

broadcasting +

```
7 y = tf.nn.softmax(tf.matmul(x, W) + b)
```

In matrix notation, 100 images at a time

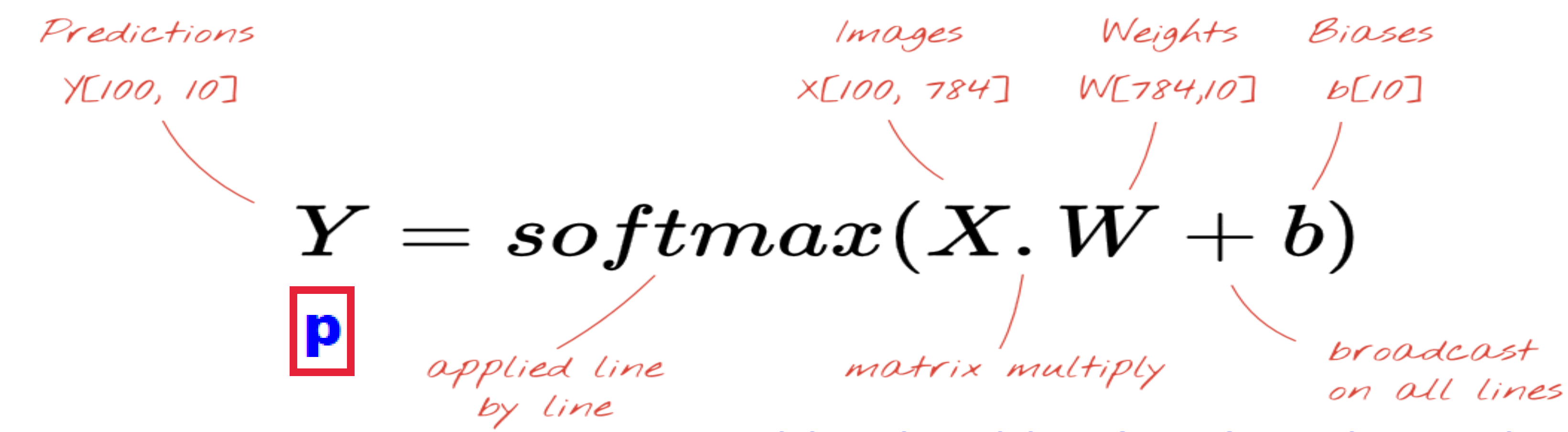


$$L = X.W + b$$

100x10=100x784.784x10+1x10

from ***TensorFlow and Deep Learning***, **Martin Gorner**

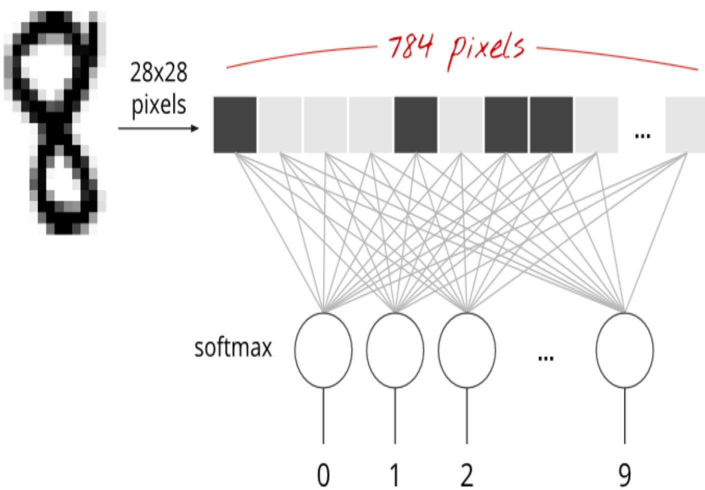
```
7 y = tf.nn.softmax(tf.matmul(x, W) + b)
```



L: 100x10=100x784.784x10+1x10

Very simple model: softmax classification

$$L = <L_0, L_1, \ldots, L_9>$$

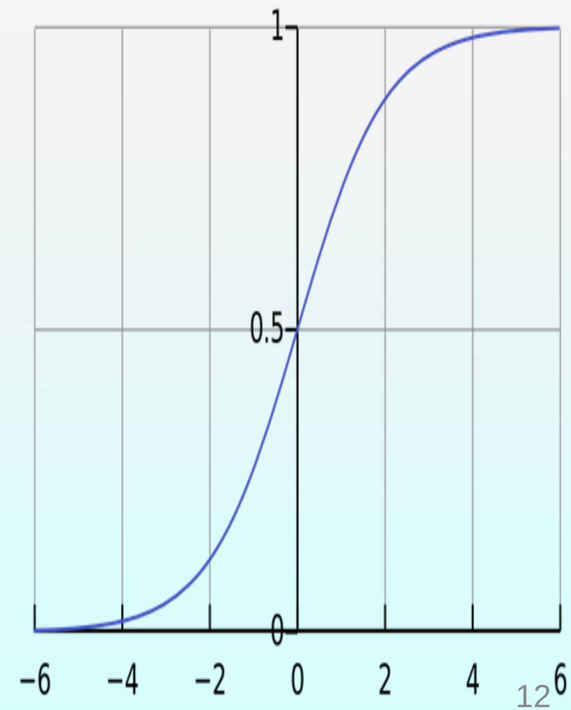


$$\|e^L\| = \sum_{n=0}^{9} e^{L_n}$$

weighted sum of all pixels + bias

$$softmax(L_n) = \frac{e^{L_n}}{\|e^L\|}$$

neuron outputs



```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
```

Success ?

Boltzmann's Entropy: $S = k \ln W$

W: Number of Microsates

2nd Law of Thermodyn.: $\delta Q = T\, dS$

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	1	0	0	0

actual probabilities, "one-hot" encoded

Cross entropy: $-\sum Y_i' . log(Y_i)$ $\quad Y = softmax(X.W + b)$

this is a "6"

computed probabilities

0.1	0.2	0.1	0.3	0.2	0.1	0.9	0.2	0.1	0.1
0	1	2	3	4	5	6	7	8	9

Gibbs's Entropy = Shannon's Entropy

$H = - \sum_i p_i \ln p_i$ (Measure of Uncertain.)

$H = - \sum_i p_i \ln p_i$

$= - \sum_i (1/6) \ln(1/6) = \ln 6 = 1.79$

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Learning Rate (Hyperparameter) = Stepping Length = 0.5

Gradient Descent

Jinn-Liang Liu

$$D_{\mathbf{p}} f(\mathbf{x}) = \lim_{t\to 0} \frac{f(\mathbf{x}+t\mathbf{p}) - f(\mathbf{x})}{t}$$

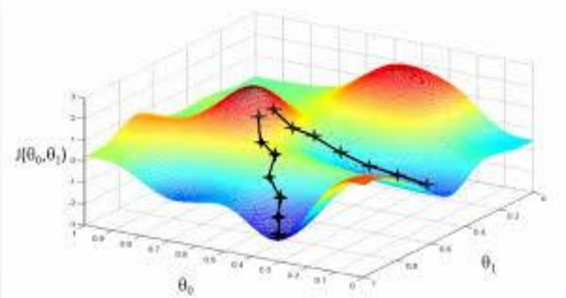
$(\mathbf{x} = (x, y)$ any fixed point, $\mathbf{p} = (p_1, p_2)$ any unit direction$)$

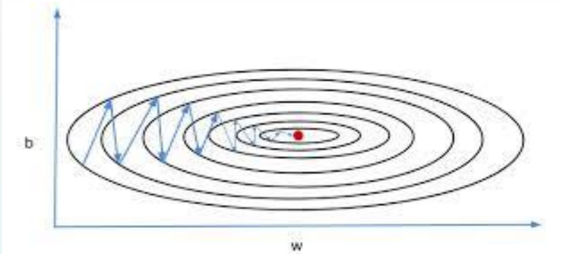
$$= \frac{\partial f(x,y)}{\partial x} p_1 + \frac{\partial f(x,y)}{\partial y} p_2$$

$$= \nabla f(x,y) \cdot \mathbf{p} = |\nabla f|\,|\mathbf{p}| \cos\theta$$

$\Rightarrow$ Min value of $D_{\mathbf{p}} f$ is $-|\nabla f|$ in $\mathbf{p} = -\nabla f / |\nabla f|$ with $\theta = 180°$

$(\nabla = \left\langle \frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right\rangle$ = grad = del, the **gradient operator**$)$





The **method of gradient (steepest) descent** is thus an iterative process

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$$

of changing (updating) our current location $\mathbf{x}_{k-1} = (x_{k-1}, y_{k-1})$ by deciding the next **stepping length** α_k in our **predicted (gradient)** direction $\mathbf{p}^{(k)} = -\nabla f(\mathbf{x}_{k-1})$.

Optimization (Gradient Descent)

```
10 train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

1D Example: Minimize $y = f(x) = x^2, \forall x \in R^1$.

Method: $x_k = x_{k-1} + \alpha_{k-1} p_{k-1}$ with $x_0 = 2$, $\alpha_{k-1} = \frac{1}{2}, \forall k$.

$$\nabla f(x) = \frac{df(x)}{dx} = 2x \quad p_0 = \frac{-\nabla f(x_0)}{|\nabla f(x_0)|} = \frac{-4}{|4|} = -1 \text{ (go west if } x_0 > 0)$$

$$\begin{aligned}
x_1 &= x_0 + \alpha_0 p_0 = 2 - \frac{1}{2} = \frac{3}{2} \Rightarrow \\
x_2 &= x_1 + \alpha_1 p_1 = \frac{3}{2} - \frac{1}{2} = 1 \Rightarrow \\
x_3 &= x_2 + \alpha_2 p_2 = 1 - \frac{1}{2} = \frac{1}{2} \Rightarrow \\
x_4 &= x_3 + \alpha_3 p_3 = \frac{1}{2} - \frac{1}{2} = 0 = x^* \text{ (optimizer)} \Rightarrow \\
y^* &= f(x^*) = 0 \text{ (optimal value)}.
\end{aligned}$$

2D Example:

Minimize $z = f(\mathbf{x}) = \frac{x^2}{4^2} + y^2, \forall \mathbf{x} = \langle x, y \rangle \in R^2.$

Method: $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{p}_{k-1}$ with $\mathbf{x}_0 = \left\langle 2, \frac{1}{4} \right\rangle$, $\alpha_{k-1} = \frac{\sqrt{5}}{4}, \forall k.$

$$\nabla f(\mathbf{x}) = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\rangle = \left\langle \frac{x}{8}, 2y \right\rangle$$

$$\mathbf{p}_0 = \frac{-\nabla f(\mathbf{x}_0)}{|\nabla f(\mathbf{x}_0)|} = \frac{-\langle \frac{1}{4}, \frac{1}{2} \rangle}{|\langle \frac{1}{4}, \frac{1}{2} \rangle|} = \frac{-\langle \frac{1}{4}, \frac{1}{2} \rangle}{\frac{\sqrt{5}}{4}} = -\left\langle \frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}} \right\rangle$$

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{p}_0 = \left\langle 2, \frac{1}{4} \right\rangle - \frac{\sqrt{5}}{4} \left\langle \frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}} \right\rangle = \left\langle \frac{7}{4}, \frac{-1}{4} \right\rangle$$

$$\mathbf{p}_1 = \frac{-\nabla f(\mathbf{x}_1)}{|\nabla f(\mathbf{x}_1)|} = \frac{-\langle \frac{7}{32}, \frac{-1}{2} \rangle}{|\langle \frac{7}{32}, \frac{-1}{2} \rangle|}$$

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \cdots \Rightarrow \cdots$$

How to Differentiate Entropy (Error)?

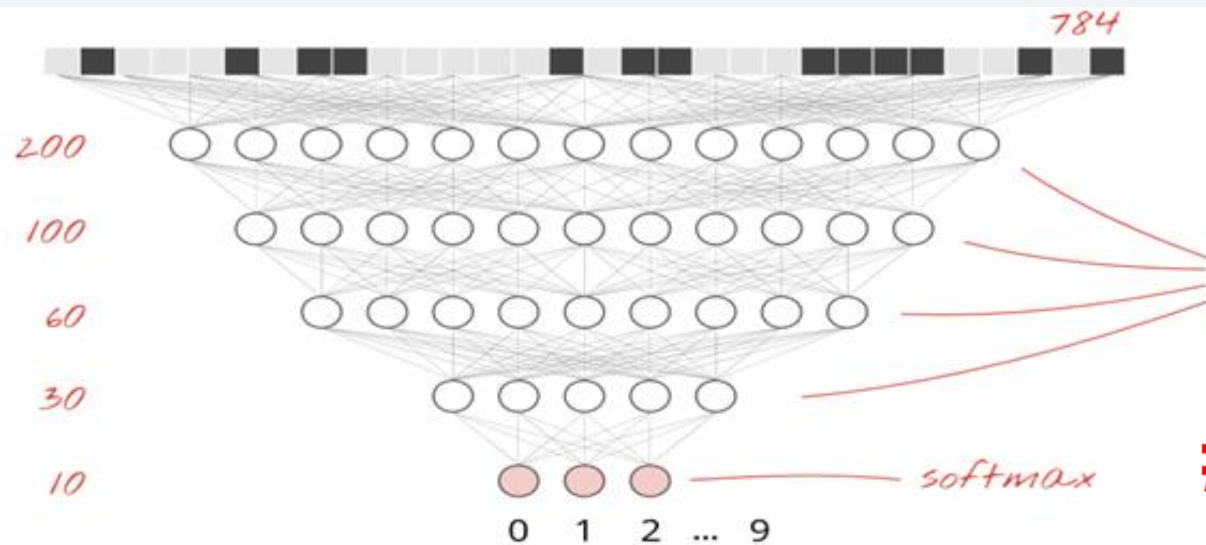
$$\frac{\partial E(W)}{\partial W_{ij}} = \frac{\partial f_3(f_2(f_1(W)))}{\partial W_{ij}},\ \#W_{ij} = 7840,\quad L = f_1(W) = XW + b$$

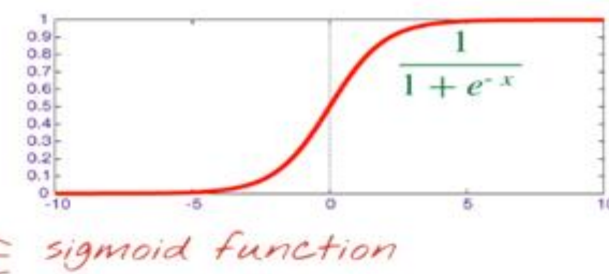
$$y = f_2(L) = softmax(L) \qquad E(W) = f_3(y) = -y\_ln(y)$$

Gradient Decent: $W_k = W_{k-1} - 0.5\nabla E(W_{k-1})$

```
9  cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
10 train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

$\#W_{ij} = 784 \times 200 + 200 \times 100 + 100 \times 60 + 60 \times 30 + 30 \times 10 = 184900$



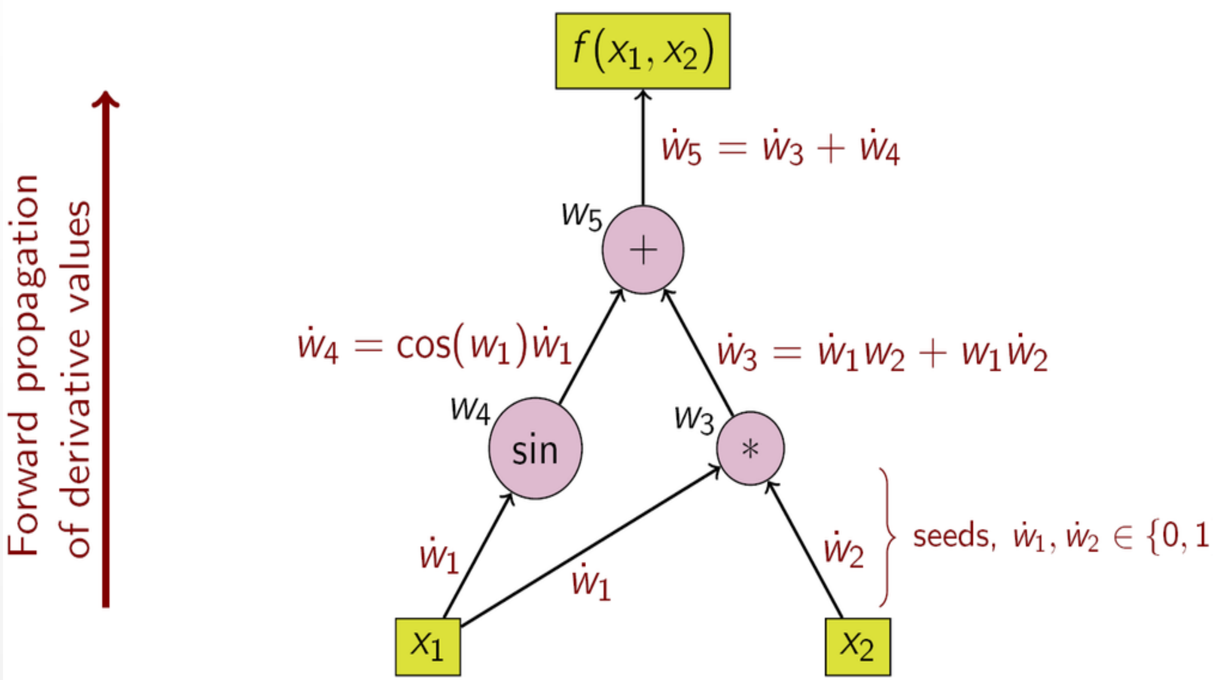


Deep Learning

$\#W_{ij} = 10^{12}$???

Chain Rule

$$y = f(x_1, x_2)$$
$$= x_1 x_2 + \sin x_1$$
$$= w_1 w_2 + \sin w_1$$
$$= w_3 + w_4$$
$$= w_5$$



$$y = f(g(h(x))) = f(g(h(w_0))) = f(g(w_1)) = f(w_2) = w_3$$

$$\frac{dy}{dx} = \frac{dy}{dw_2} \frac{dw_2}{dw_1} \frac{dw_1}{dx}$$

forward mode $$\frac{dw_i}{dx} = \frac{dw_i}{dw_{i-1}} \frac{dw_{i-1}}{dx}$$

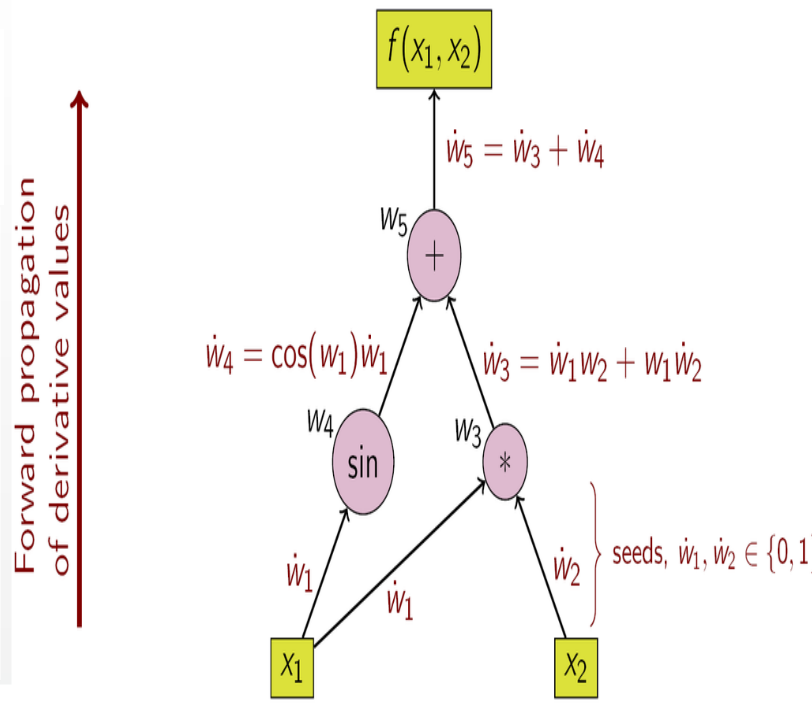
reverse mode $$\frac{dy}{dw_i} = \frac{dy}{dw_{i+1}} \frac{dw_{i+1}}{dw_i}$$

$$\nabla f(x_1, x_2) = \left\langle \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2} \right\rangle = \left\langle \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2} \right\rangle = \langle x_2 + \cos x_1, x_1 \rangle = \langle p_1, p_2 \rangle$$

Forward Propagation

$$\dot{w} = \frac{\partial w}{\partial x} \qquad \dot{w}_1 = \frac{\partial x_1}{\partial x_1} = 1 \qquad \dot{w}_2 = \frac{\partial x_2}{\partial x_1} = 0$$

$$\begin{aligned} w_1 &= x_1 & \dot{w}_1 &= 1 \text{ (seed)} \\ w_2 &= x_2 & \dot{w}_2 &= 0 \text{ (seed)} \\ w_3 &= w_1 \cdot w_2 & \dot{w}_3 &= w_2 \cdot \dot{w}_1 + w_1 \cdot \dot{w}_2 \\ w_4 &= \sin w_1 & \dot{w}_4 &= \cos w_1 \cdot \dot{w}_1 \\ w_5 &= w_3 + w_4 & \dot{w}_5 &= \dot{w}_3 + \dot{w}_4 \end{aligned}$$



$$\dot{w}_3 = \frac{\partial w_3}{\partial x_1} = \frac{\partial (w_1 w_2)}{\partial x_1} = \dot{w}_1 w_2 + w_1 \dot{w}_2$$

$$\begin{aligned} y &= f(x_1, x_2) \\ &= x_1 x_2 + \sin x_1 \\ &= w_1 w_2 + \sin w_1 \\ &= w_3 + w_4 \\ &= w_5 \end{aligned}$$

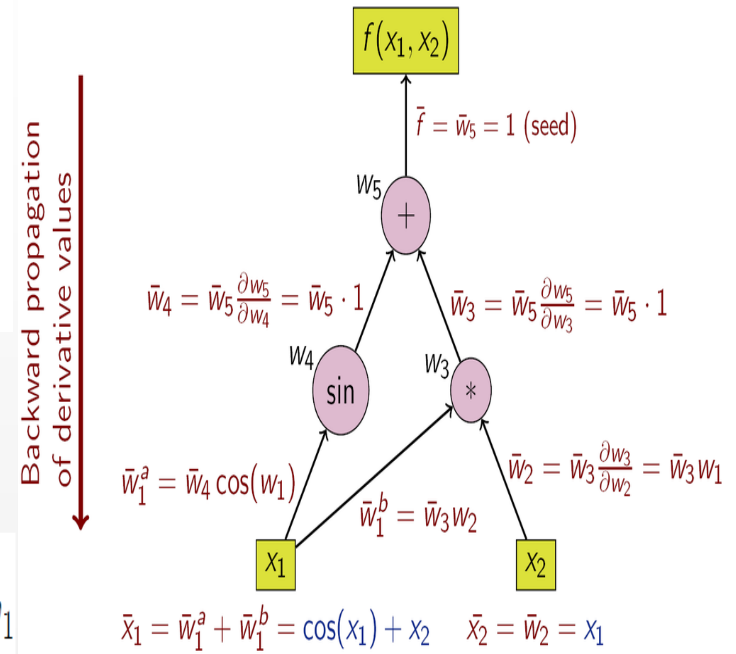
$$\left\langle \frac{\partial f}{\partial x_1}, \right\rangle = \langle x_2 + \cos x_1, \rangle = \left\langle \dot{w}_5, \right\rangle$$

$$= \left\langle \dot{w}_3 + \dot{w}_4, \right\rangle = \langle w_2 + \cos w_1, \rangle$$

Backward Propagation

$$\bar{w} = \frac{\partial y}{\partial w}$$

$$\begin{aligned}
\bar{w}_5 &= 1 \text{ (seed)} \\
\bar{w}_4 &= \bar{w}_5 \\
\bar{w}_3 &= \bar{w}_5 \\
\bar{w}_2 &= \bar{w}_3 \cdot w_1 \\
\bar{w}_1 &= \bar{w}_3 \cdot w_2 + \bar{w}_4 \cdot \cos w_1
\end{aligned}$$



$$\bar{w}_5 = \frac{\partial y}{\partial w_5} = 1, \quad \bar{w}_4 = \frac{\partial y}{\partial w_4} = \frac{\partial y}{\partial w_5}\frac{\partial w_5}{\partial w_4} = \bar{w}_5 \frac{\partial (w_3 + w_4)}{\partial w_4} = \bar{w}_5$$

$$\bar{w}_3 = \frac{\partial y}{\partial w_3} = \frac{\partial y}{\partial w_5}\frac{\partial w_5}{\partial w_3} = \bar{w}_5$$

$$\bar{w}_2 = \frac{\partial y}{\partial w_2} = \bar{w}_5 \frac{\partial (w_1 w_2 + \sin w_1)}{\partial w_2} = \bar{w}_5 w_1$$

$$\bar{w}_1 = \frac{\partial y}{\partial w_1} = \bar{w}_5 \frac{\partial (w_1 w_2 + \sin w_1)}{\partial w_1} = \bar{w}_5 (w_2 + \cos w_1)$$

$$\left\langle \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2} \right\rangle = \langle x_2 + \cos x_1, \ x_1 \rangle = \langle \bar{w}_1, \ \bar{w}_2 \rangle$$

$$= \langle \bar{w}_3 w_2 + \bar{w}_4 \cos w_1, \ \bar{w}_3 w_1 \rangle$$

$$\begin{aligned}
y &= f(x_1, x_2) \\
&= x_1 x_2 + \sin x_1 \\
&= w_1 w_2 + \sin w_1 \\
&= w_3 + w_4 \\
&= w_5
\end{aligned}$$

Backward Propagation (Hooray!)

$f : \mathbb{R}^n \to \mathbb{R}^m$ $m \ll n$ ⇒ **Backward**

$f(x_1, x_2) = x_1 x_2 + \sin x_1$ **n = 2, m=1** ⇒

Backward : Forward = m : n = 1 : 2 = 1s : 2s

Algorithm 1: $y = Wx + b$ n = 7850, m=10

Backward : Forward = 1 : 7850 = 1s : 13m

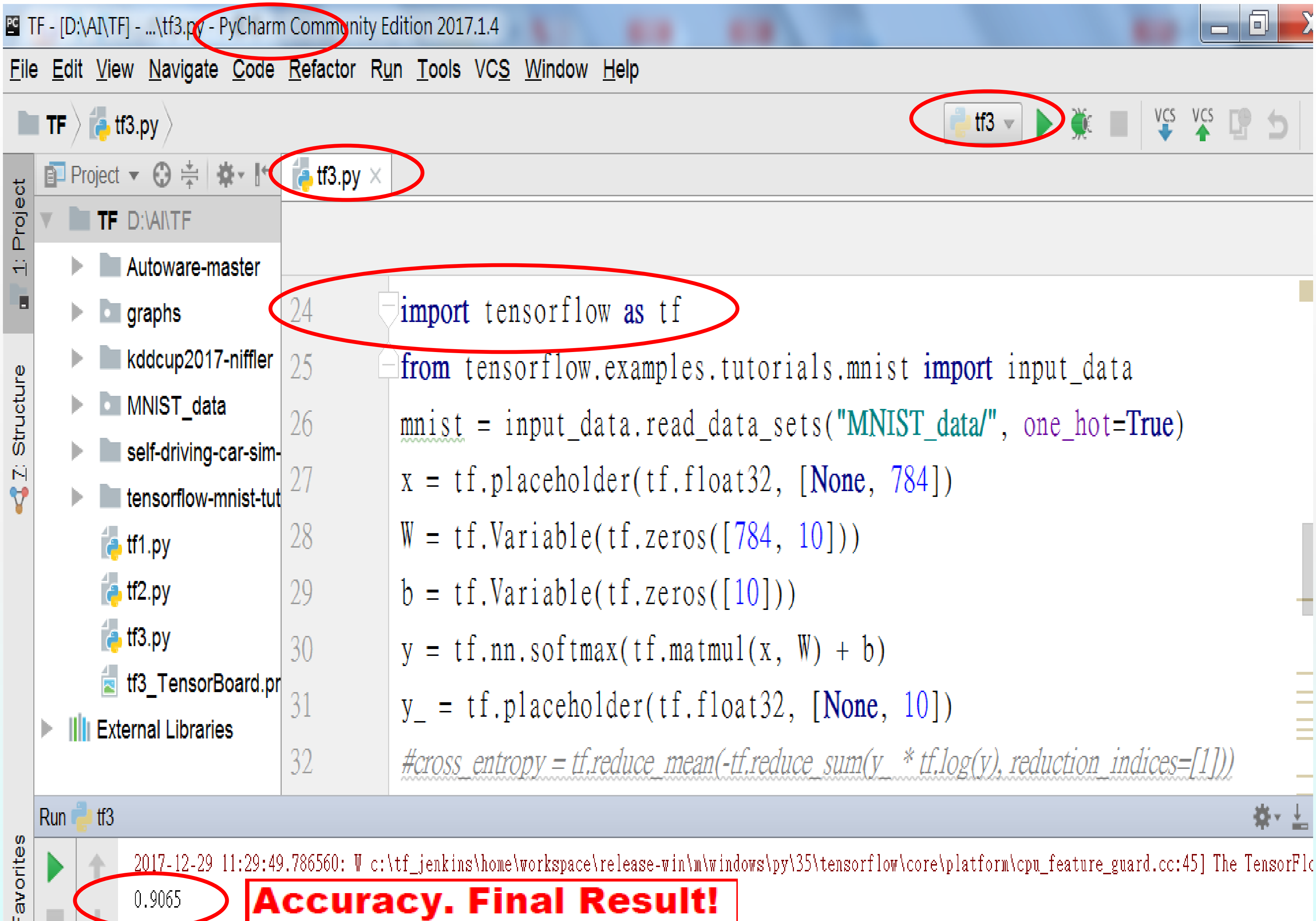
$$\frac{\partial E}{\partial W_{ij}}?\ E = f\left(g \ldots (h(W))\right), \#W_{ij} = 1{,}000{,}000{,}000{,}000$$

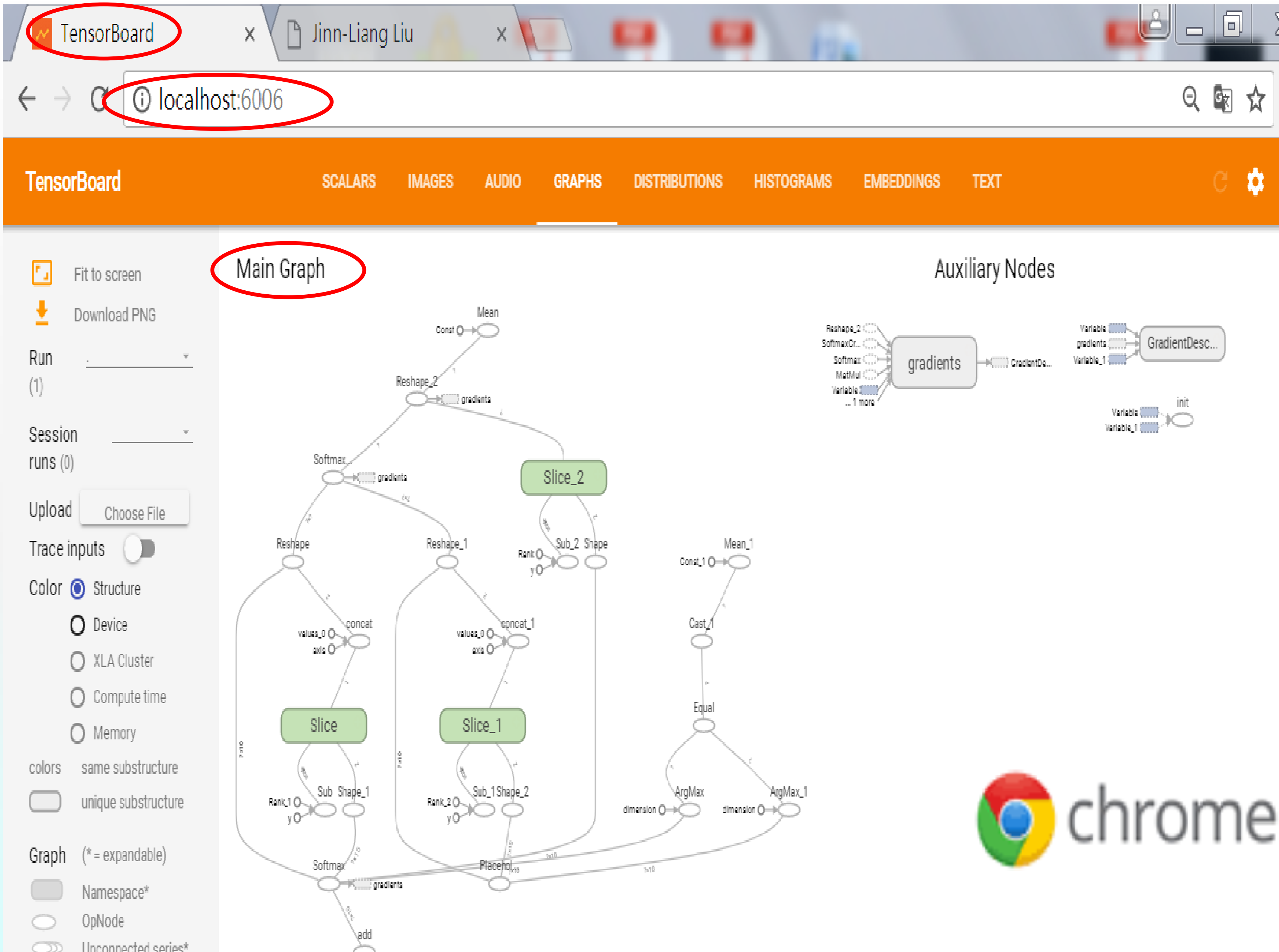
Backward : Forward = 1s : 31710 years

Run the Code (Algorithm)

```
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labe
```

This should be about 92%. **Accuracy. Final Result!**


TF - [D:\AI\TF] - ...\tf3.py - PyCharm Community Edition 2017.1.4
File Edit View Navigate Code Refactor Run Tools VCS Window Help
TF
tf3.py
tf3
Project
TF D:\AI\TF
Autoware-master
graphs
kddcup2017-niffler
MNIST_data
self-driving-car-sim-
tensorflow-mnist-tut
tf1.py
tf2.py
tf3.py
tf3_TensorBoard.pr
External Libraries
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])
#cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
Run tf3
2017-12-29 11:29:49.786560: W c:\tf_jenkins\home\workspace\release-win\m\windows\py\35\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFl
0.9065
Accuracy. Final Result!
1: Project
7: Structure
Favorites


TensorBoard
Jinn-Liang Liu
localhost:6006
TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
TEXT
Fit to screen
Download PNG
Run
(1)
Session runs (0)
Upload
Choose File
Trace inputs
Color
Structure
Device
XLA Cluster
Compute time
Memory
colors
same substructure
unique substructure
Graph
(* = expandable)
Namespace*
OpNode
Unconnected series*
Main Graph
Auxiliary Nodes
Mean
Reshape_2
Softmax
Slice_2
Reshape
Reshape_1
concat
concat_1
Slice
Slice_1
Sub_2
Shape
Mean_1
Cast_1
Equal
ArgMax
ArgMax_1
Sub
Shape_1
Sub_1
Shape_2
Placehol...
add
gradients
GradientDesc...
init
chrome

Coding! Coding! Coding!

Variable, Function y = f(x): Declare, Define, Call

Python is OOP: Class, Object

```
import tensorflow as tf
```
Class: tensorflow, Declare Object tf

```
x = tf.placeholder(tf.float32, [None, 784])
```
Declare x, Function Call: placehoder()

```
W = tf.Variable(tf.zeros([784, 10]))
```
Declare and Define W

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```
Declare y using nn, softmax(), matmul()

```
sess = tf.InteractiveSession()
```
Declare Object sess

```
for _ in range(1000):
```
for loop

```
  batch_xs, batch_ys = mnist.train.next_batch(100)
```
Declare and Define

```
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```
Call run() in sess

feed_dict: python dictionary maps from tf. placeholder vars to data

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Thank You