Lecture 2 Neural Network

Tian Sheuan Chang

Source: CS231n

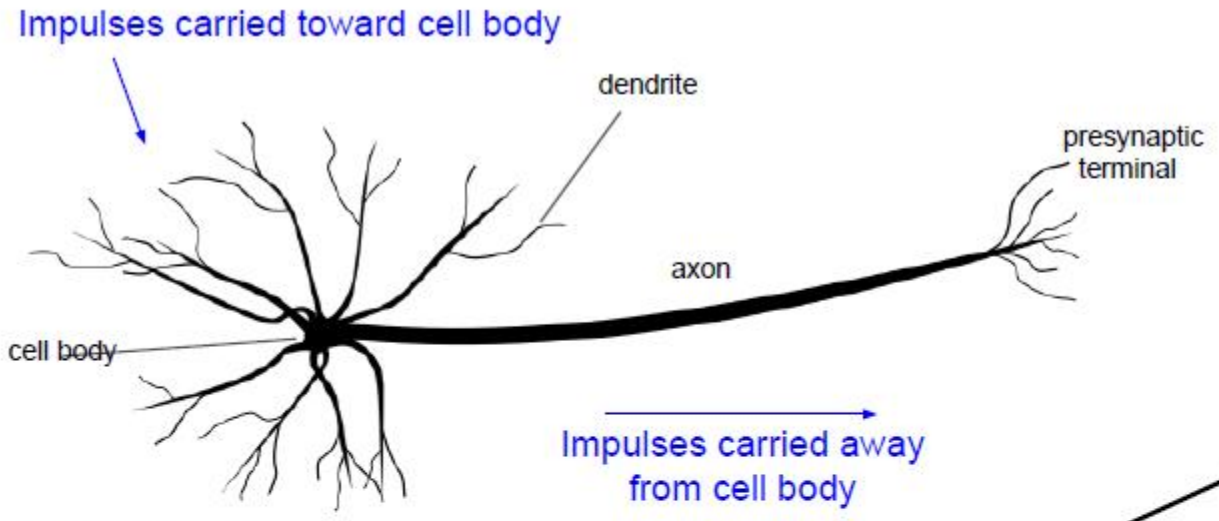
Outline

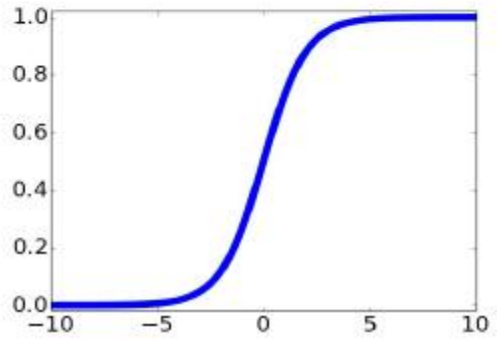
- Deep feedforward network
  - Motivation
  - Single layer neural network (perceptron)
  - Multi-layer neural network
- Backpropagation
- 2-Layer NN example with Keras/Pytorch (optional)

NEURAL NETWORK

Neural Network (Perceptron)

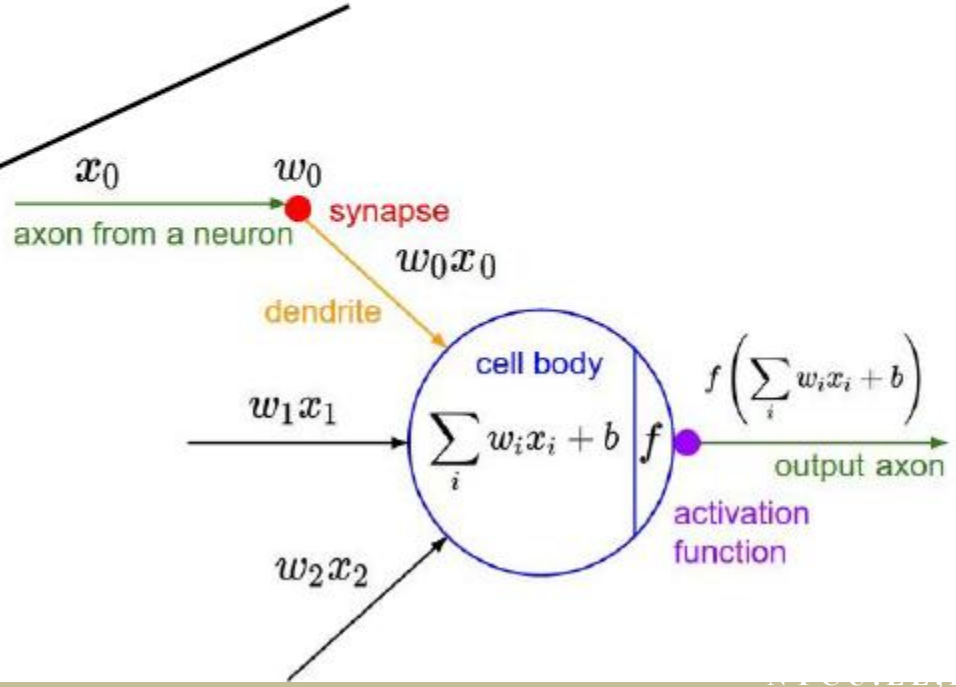
- Invented in 1954 by Rosenblatt
- Inspired by neurobiology



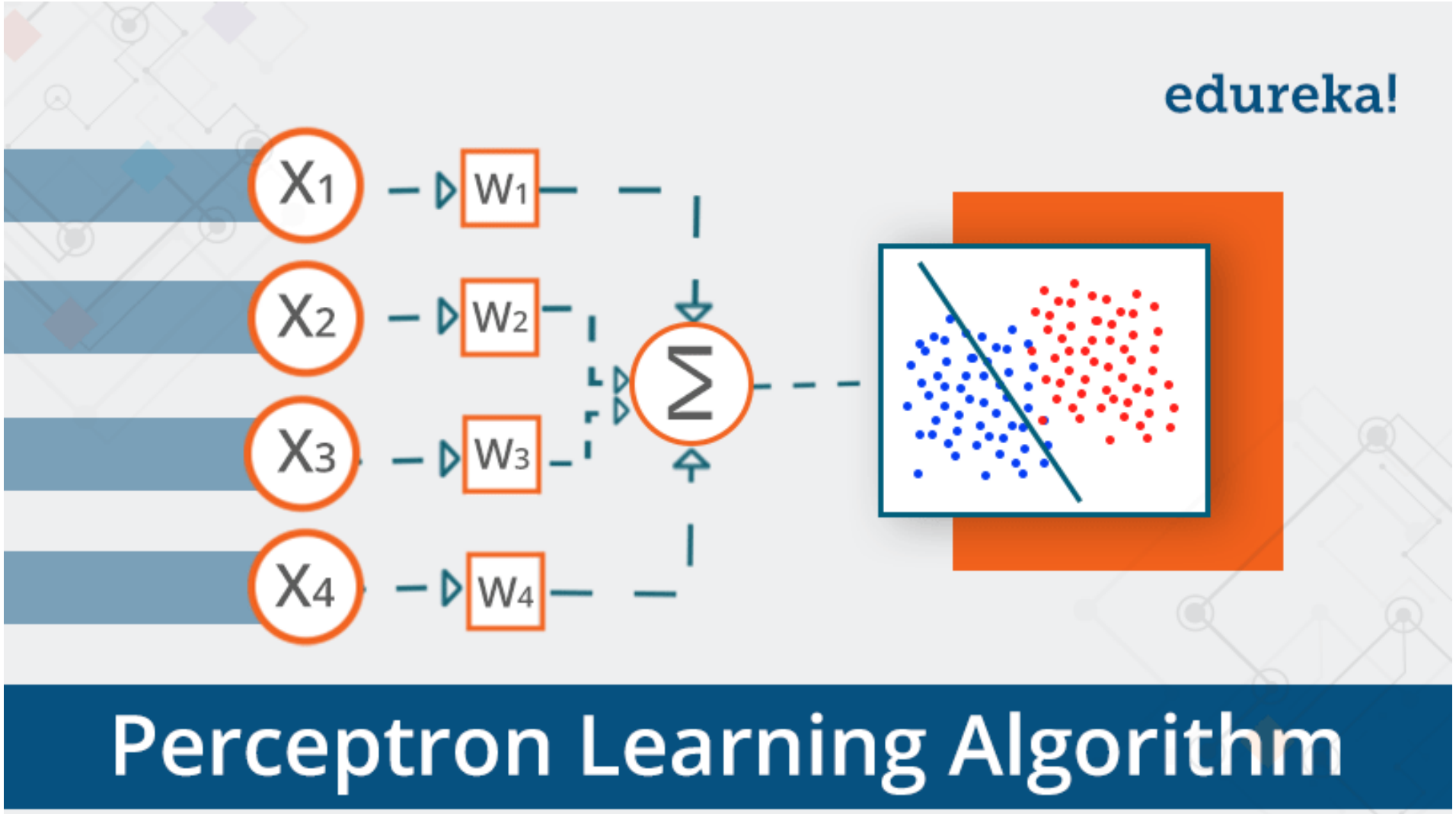


sigmoid activation function

$$\frac{1}{1+e^{-x}}$$

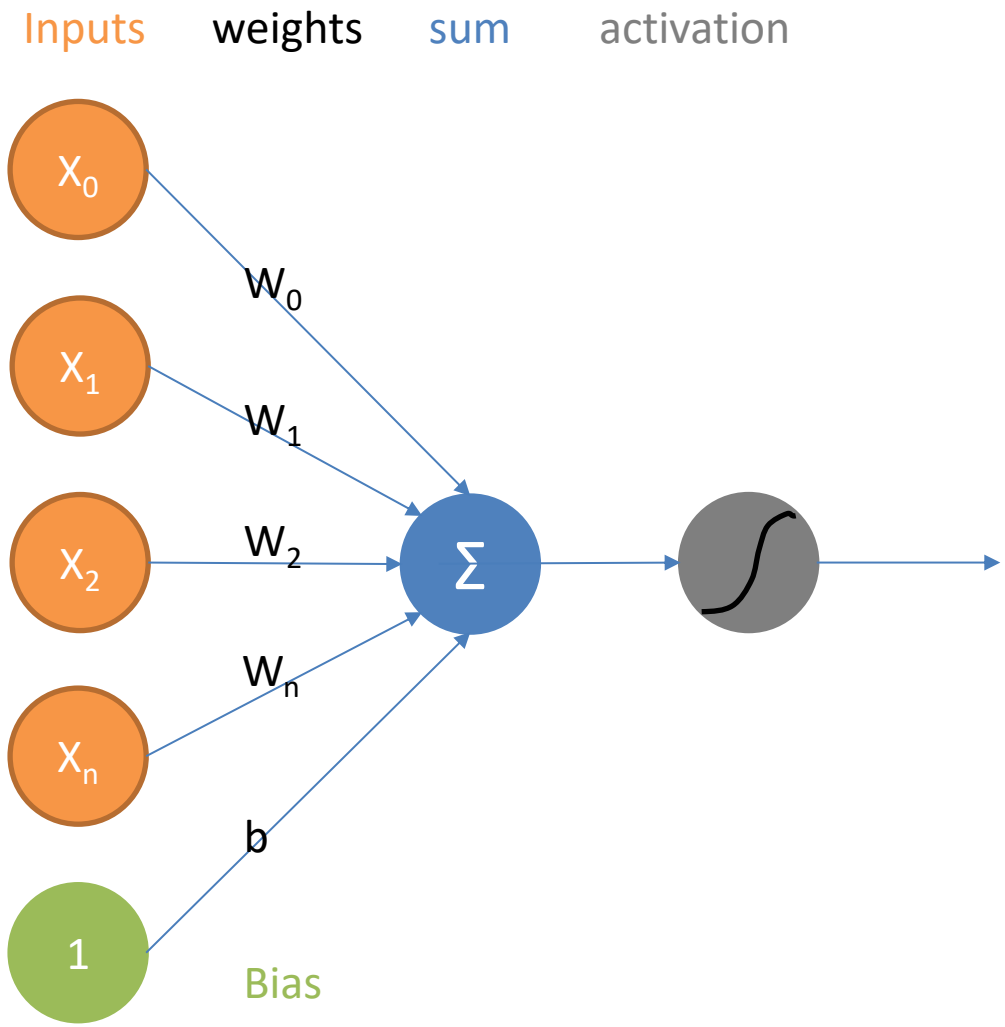


Source: cs231n



https://www.edureka.co/blog/perceptron-learning-algorithm/

Single Layer Neural Network



Source: Nick Locascio, intro to deep learning,

Forward Pass of Single Layer NN

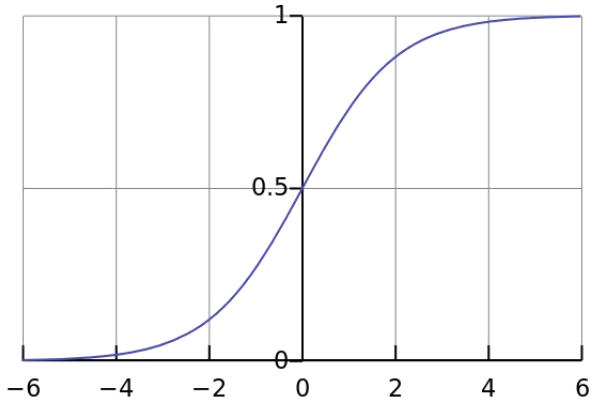
Activation Function

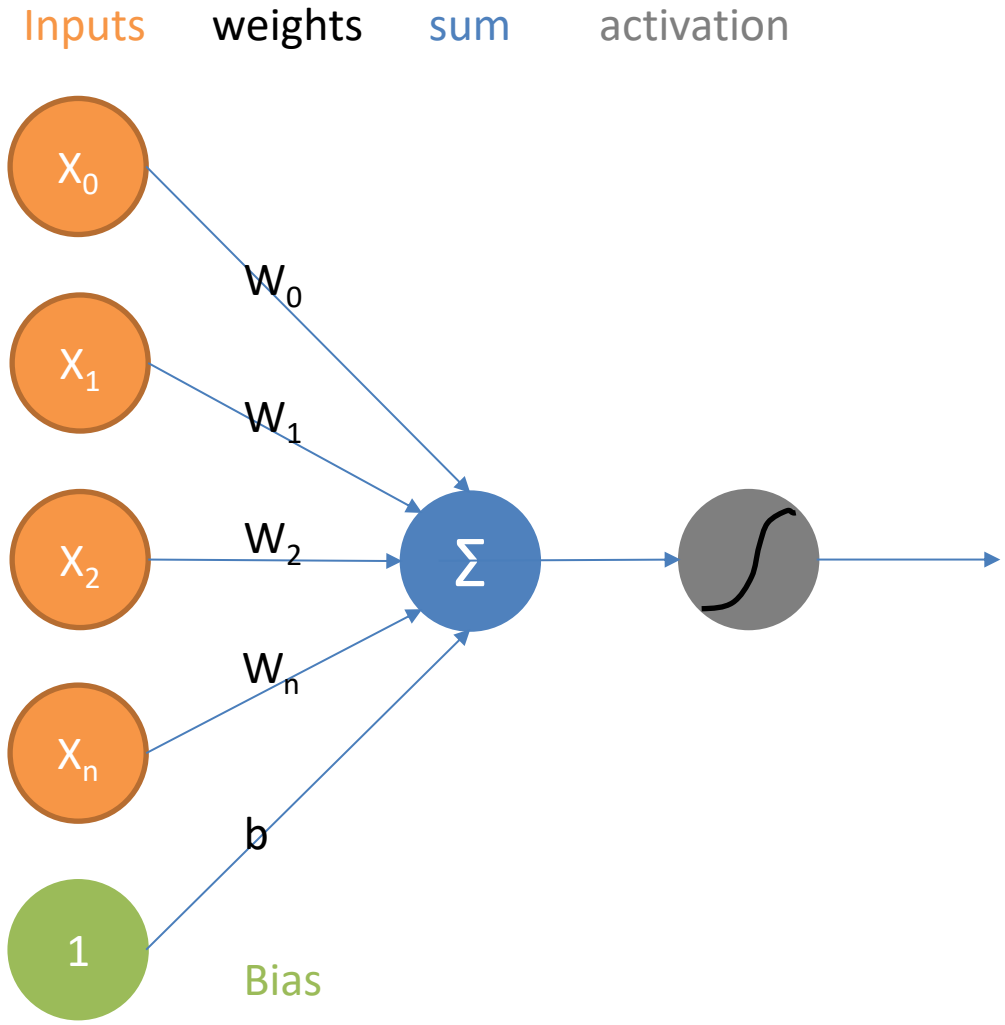
$$output = g((\sum_{i=0}^{N} x_i * w_i) + b)$$

$$X = x_0, x_1, ...x_n$$

$$W = w_0, w_1, ...w_n$$

$$g(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$





Source: Nick Locascio, intro to deep learning,

Perceptron Forward Pass

$$output = g($$

(2*0.1) +

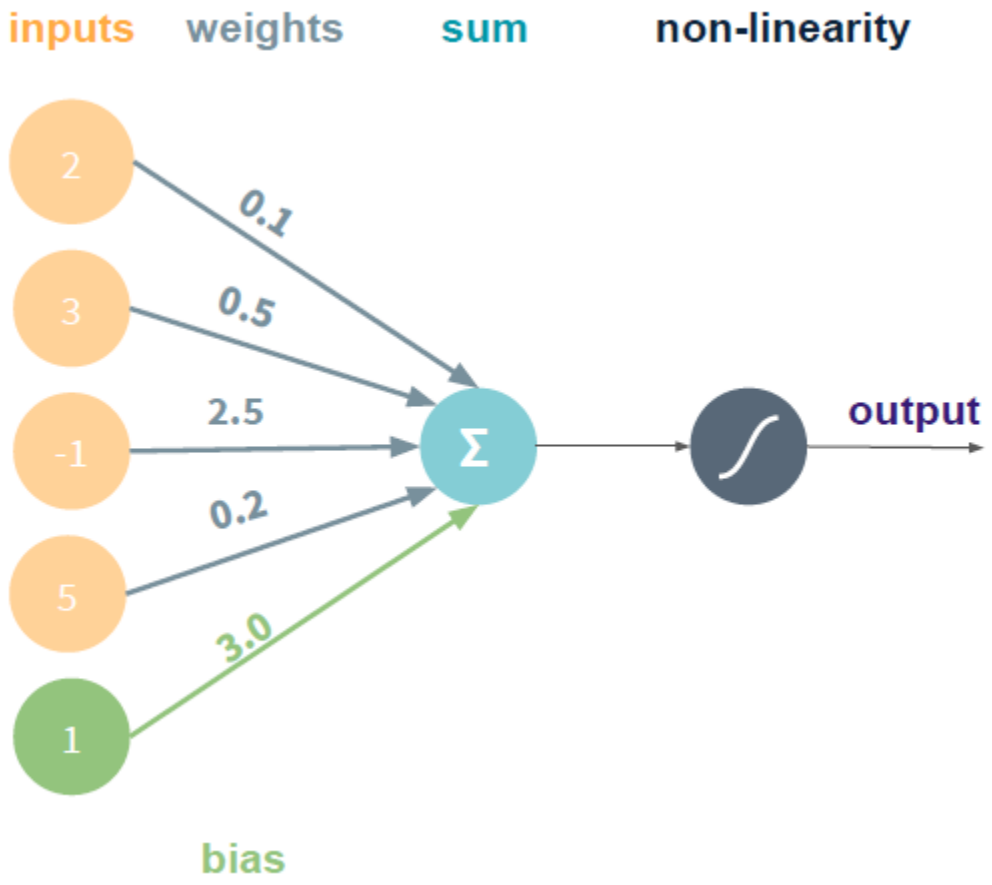
(3*0.5) +

(-1*2.5) +

(5*0.2) +

(1*3.0)

)

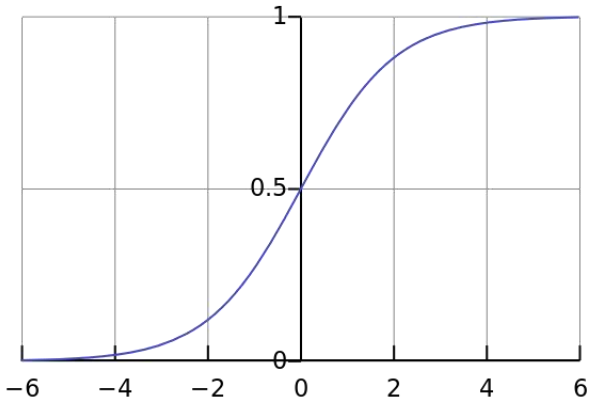


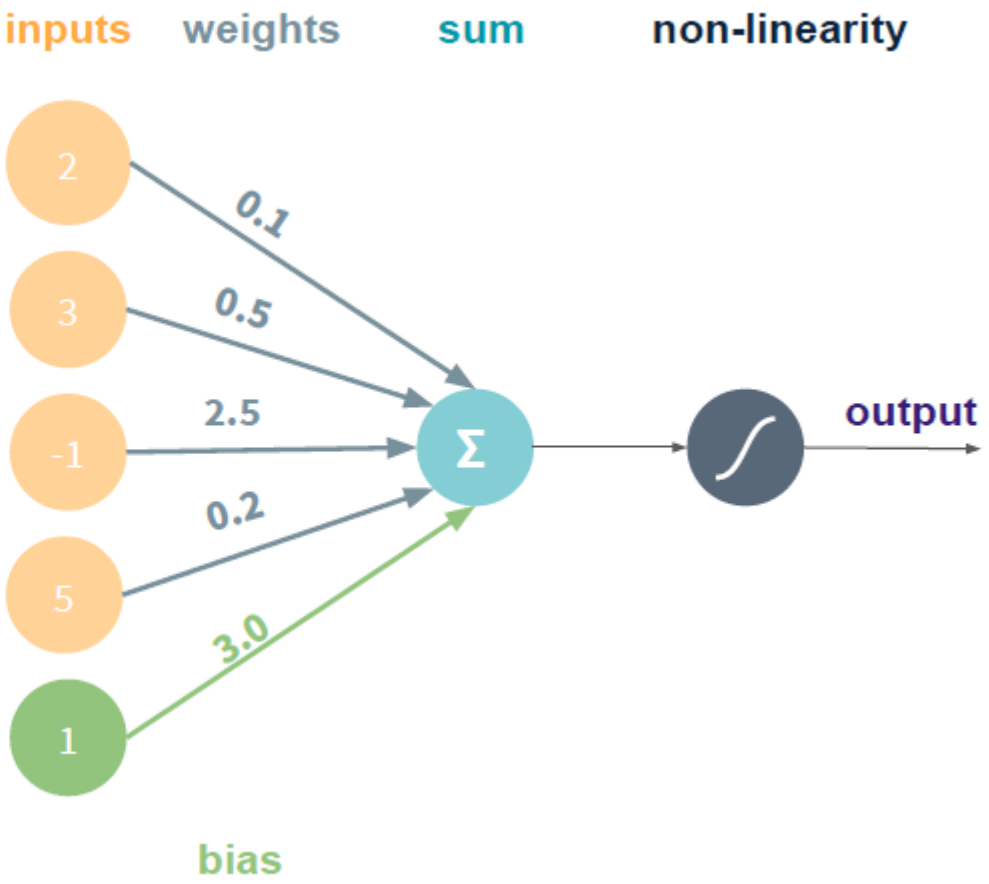
Source: Nick Locascio, intro to deep learning,

Perceptron Forward Pass

$$output = g(3.2) = \sigma(3.2)$$

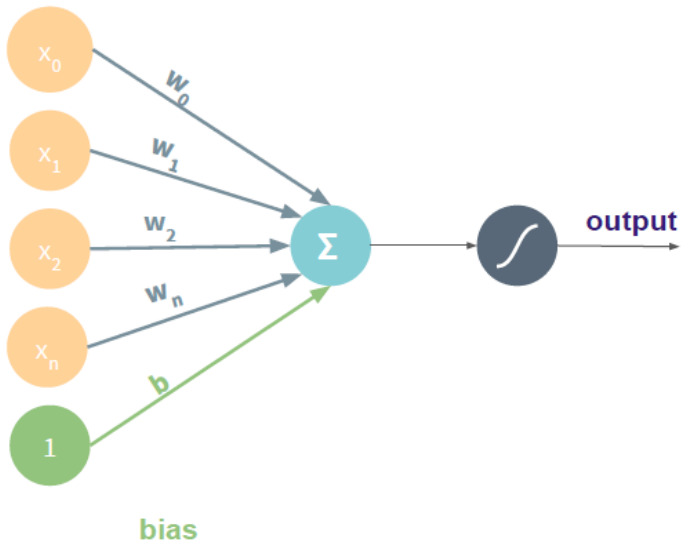
$$= \frac{1}{(1 + e^{-3.2})} = 0.96$$

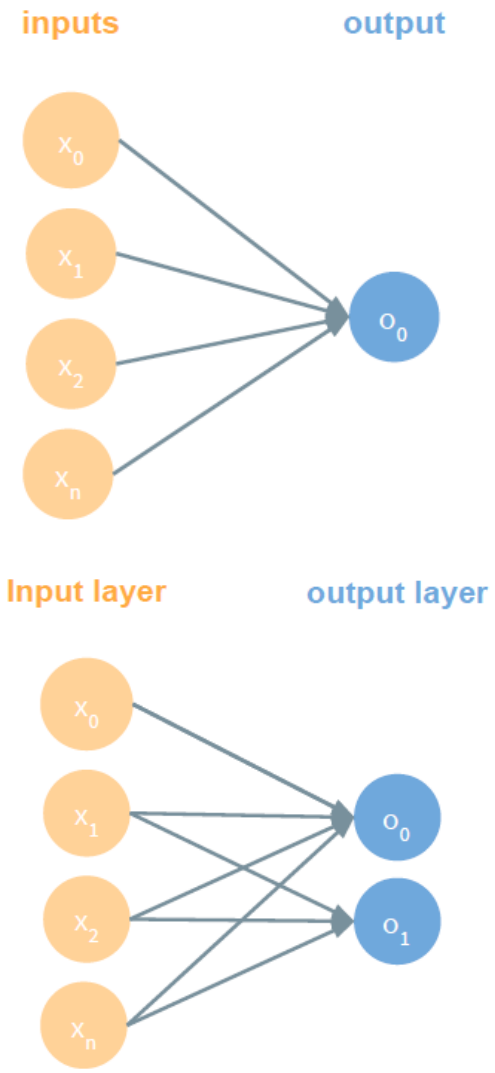


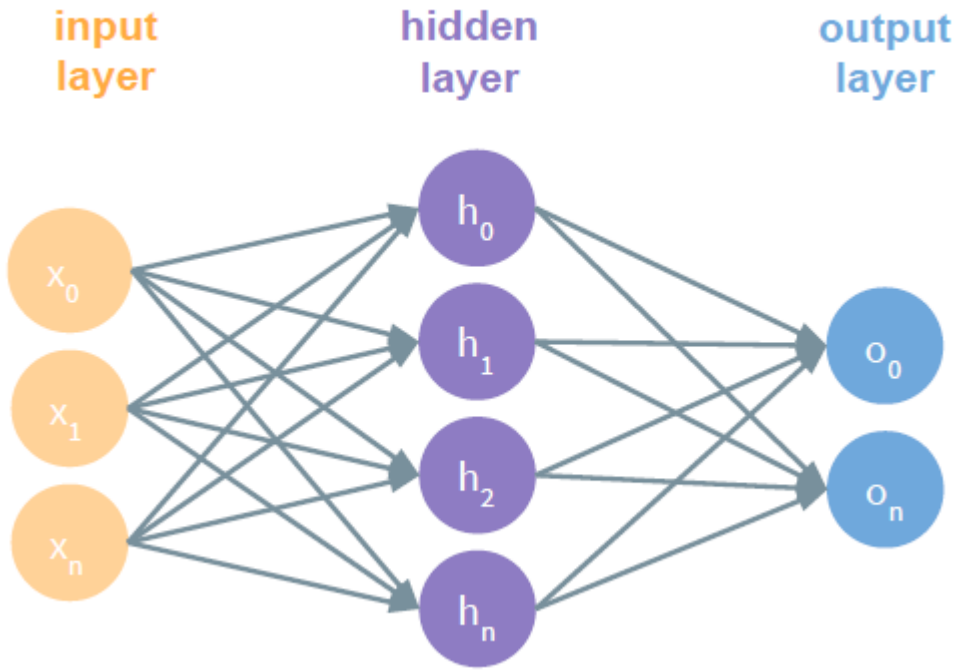


Source: Nick Locascio, intro to deep learning,

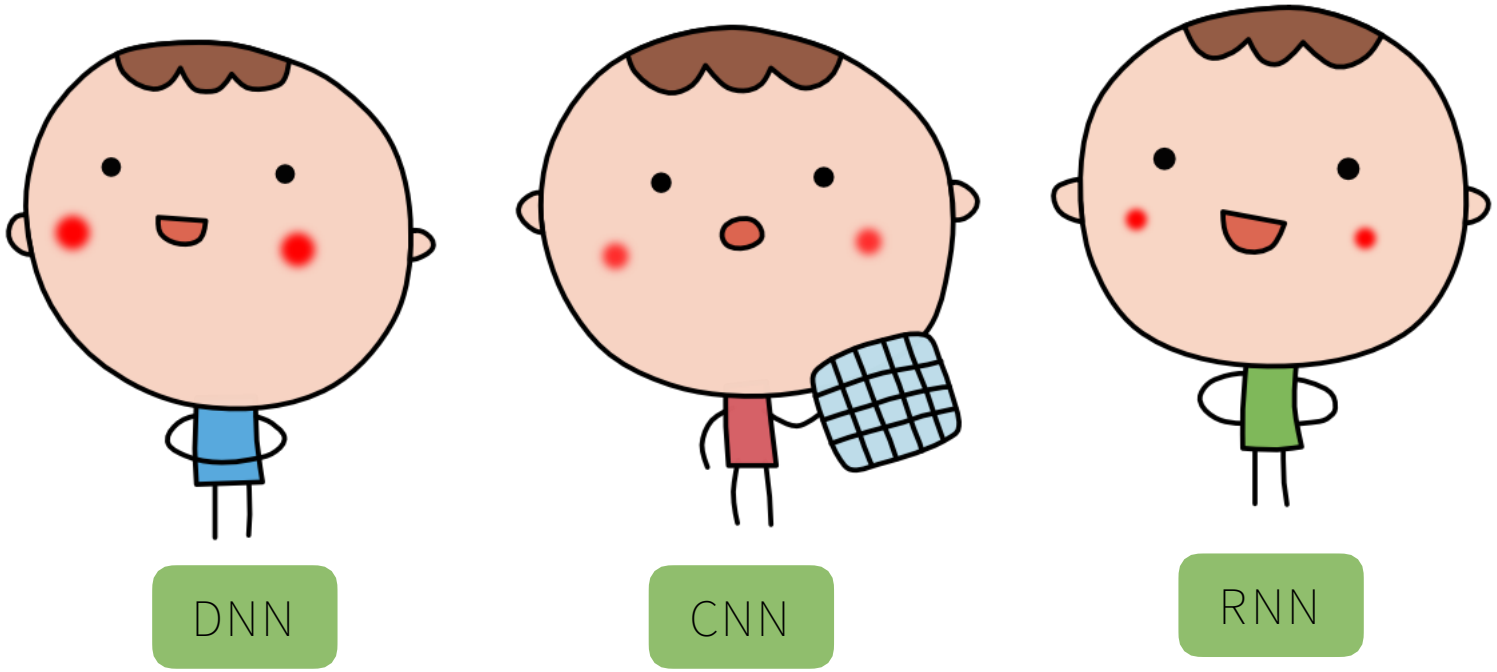
Fully Connected Neural Network (全連接網路)





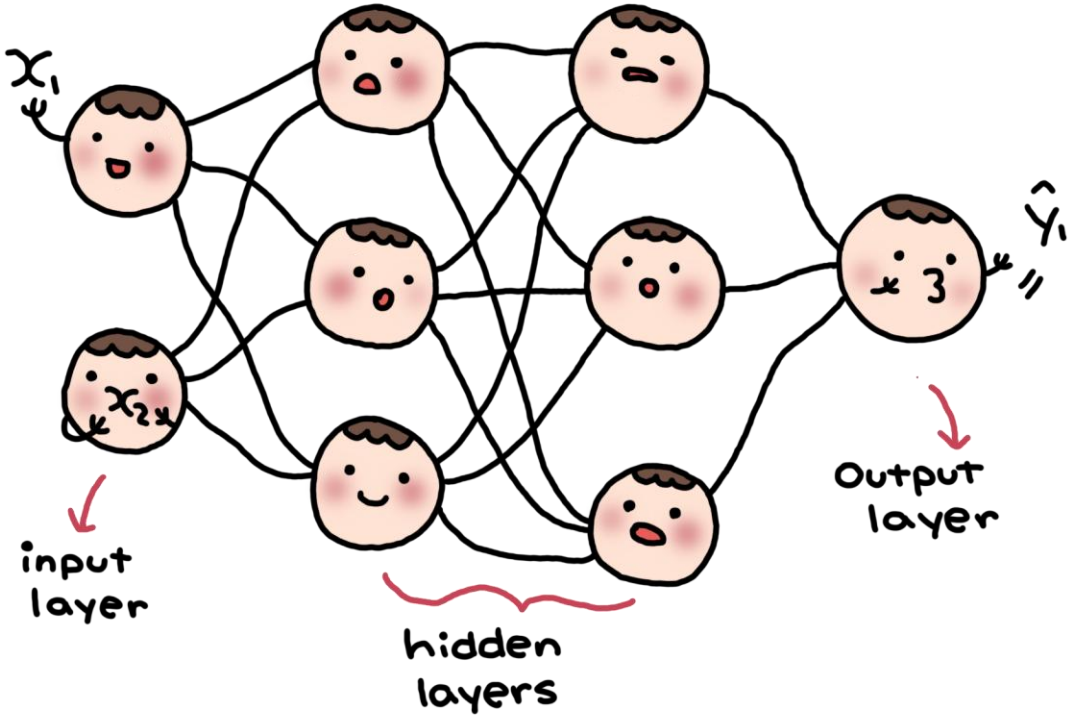


- 很多個 neurons 連接成 network
- Universal approximation theorem: 一層隱藏層的網路，可近似任意函數

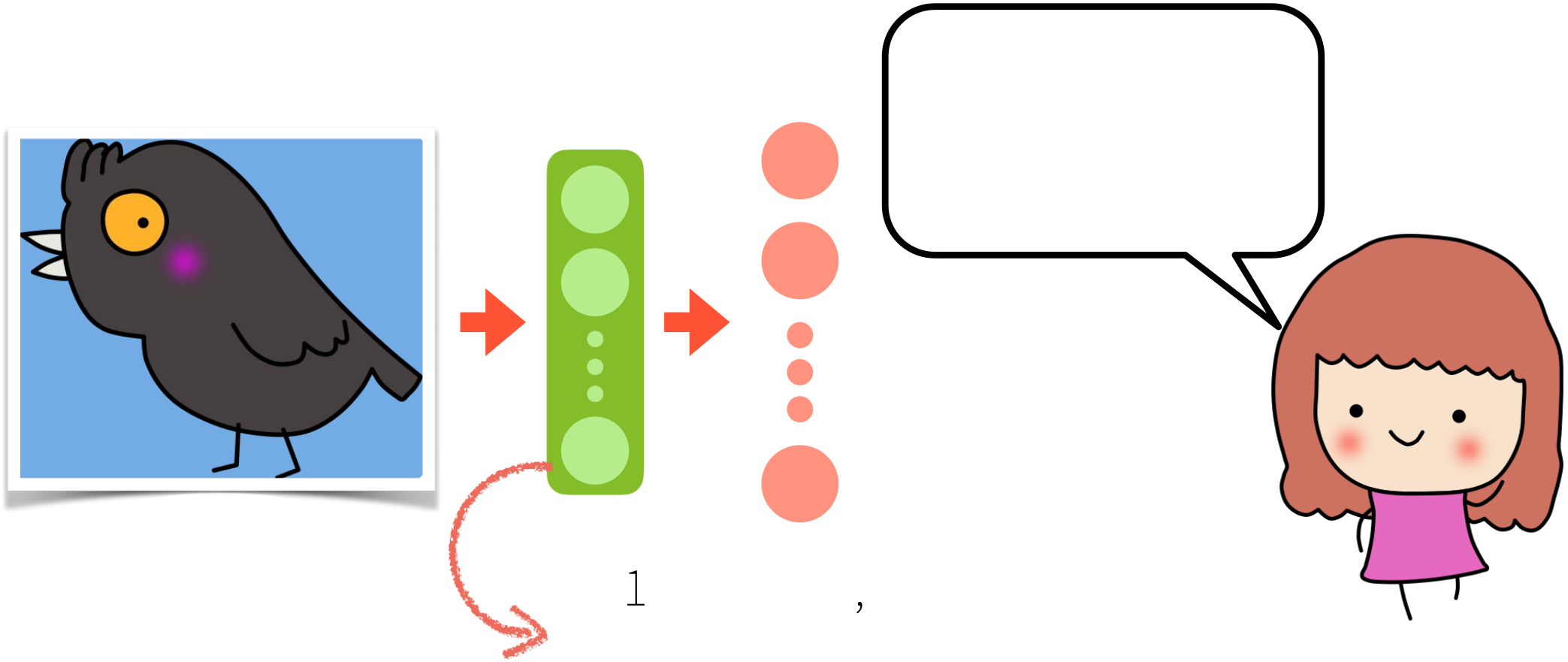


,

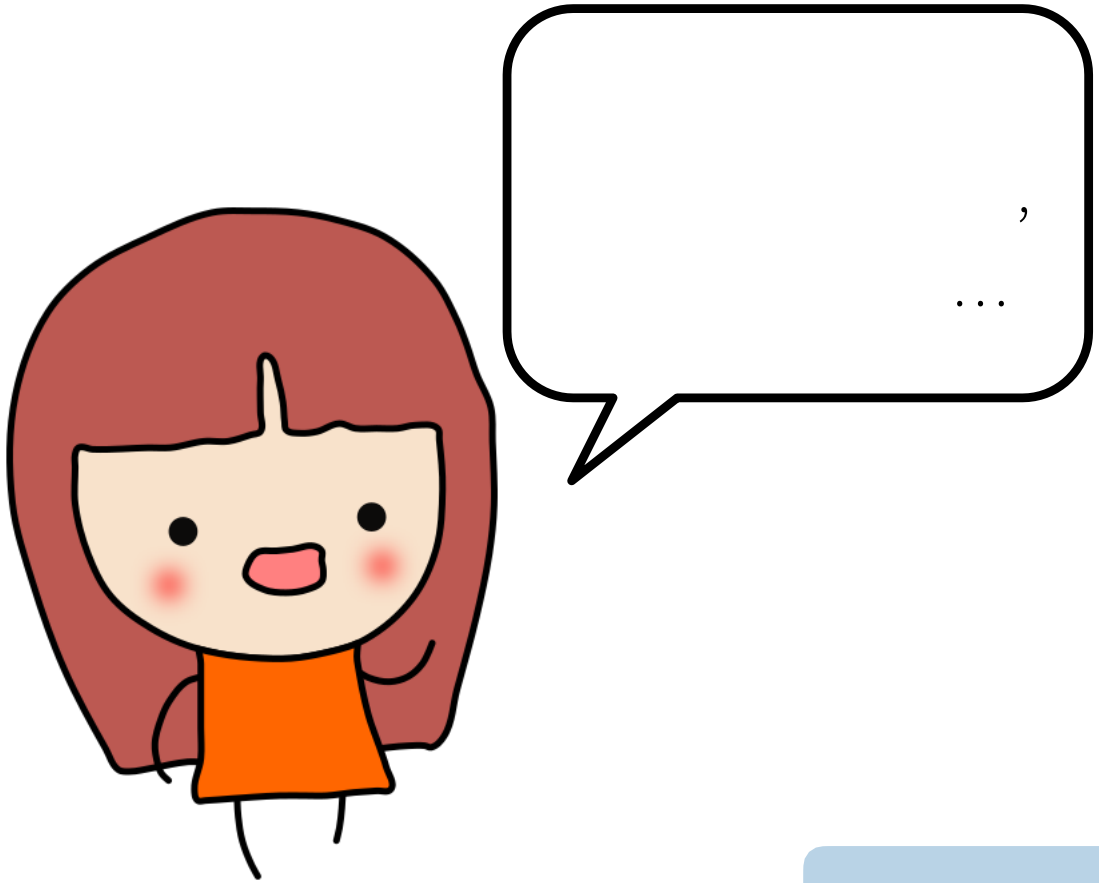
Source: Yen-Long Tsai

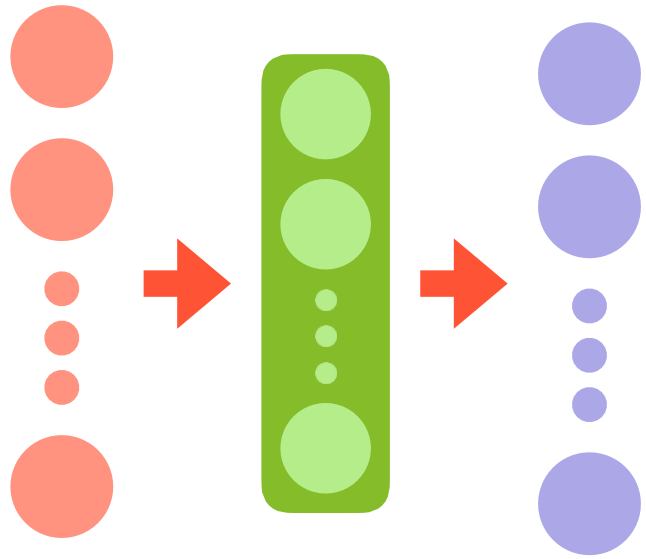


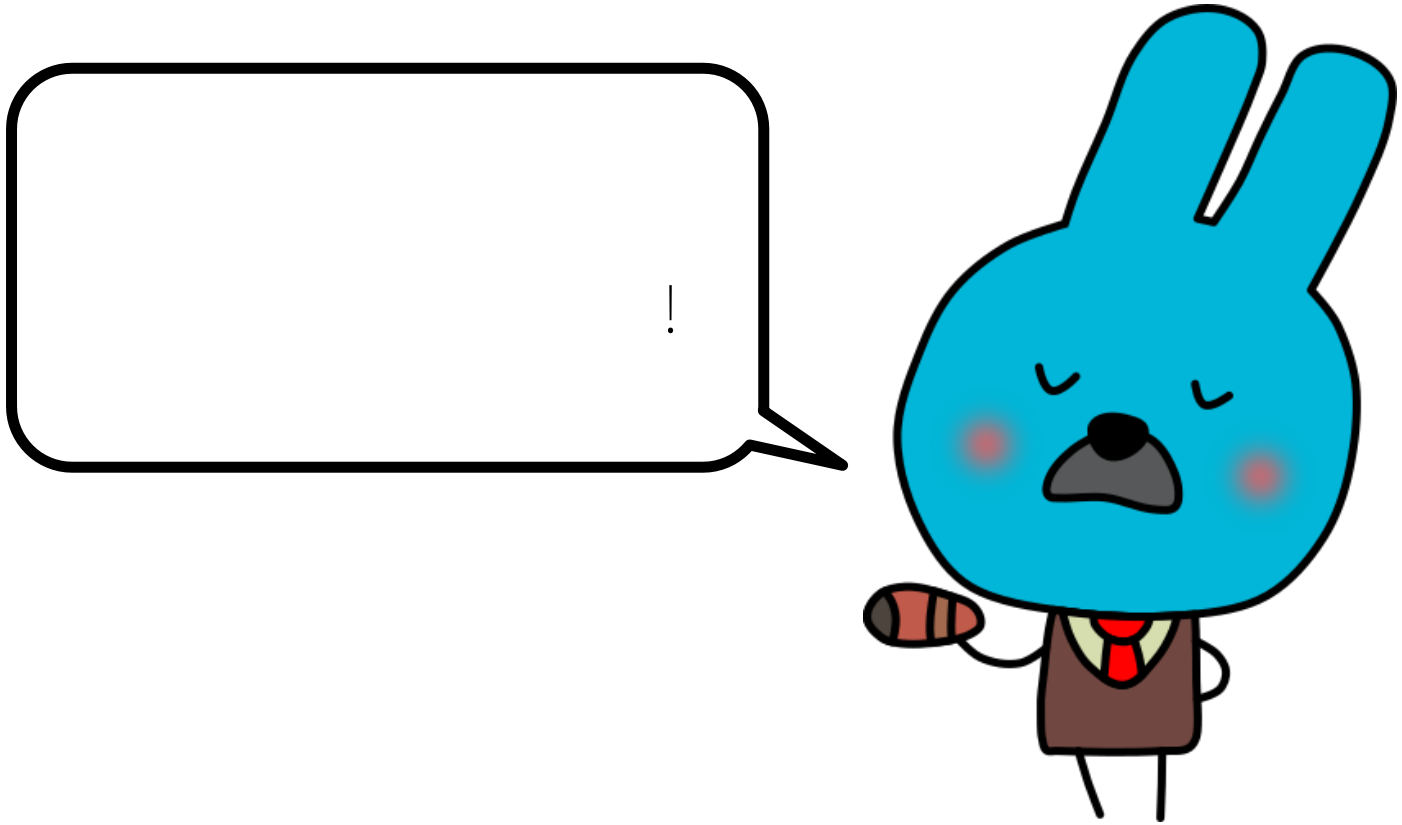
Source: Yen-Long Tsai



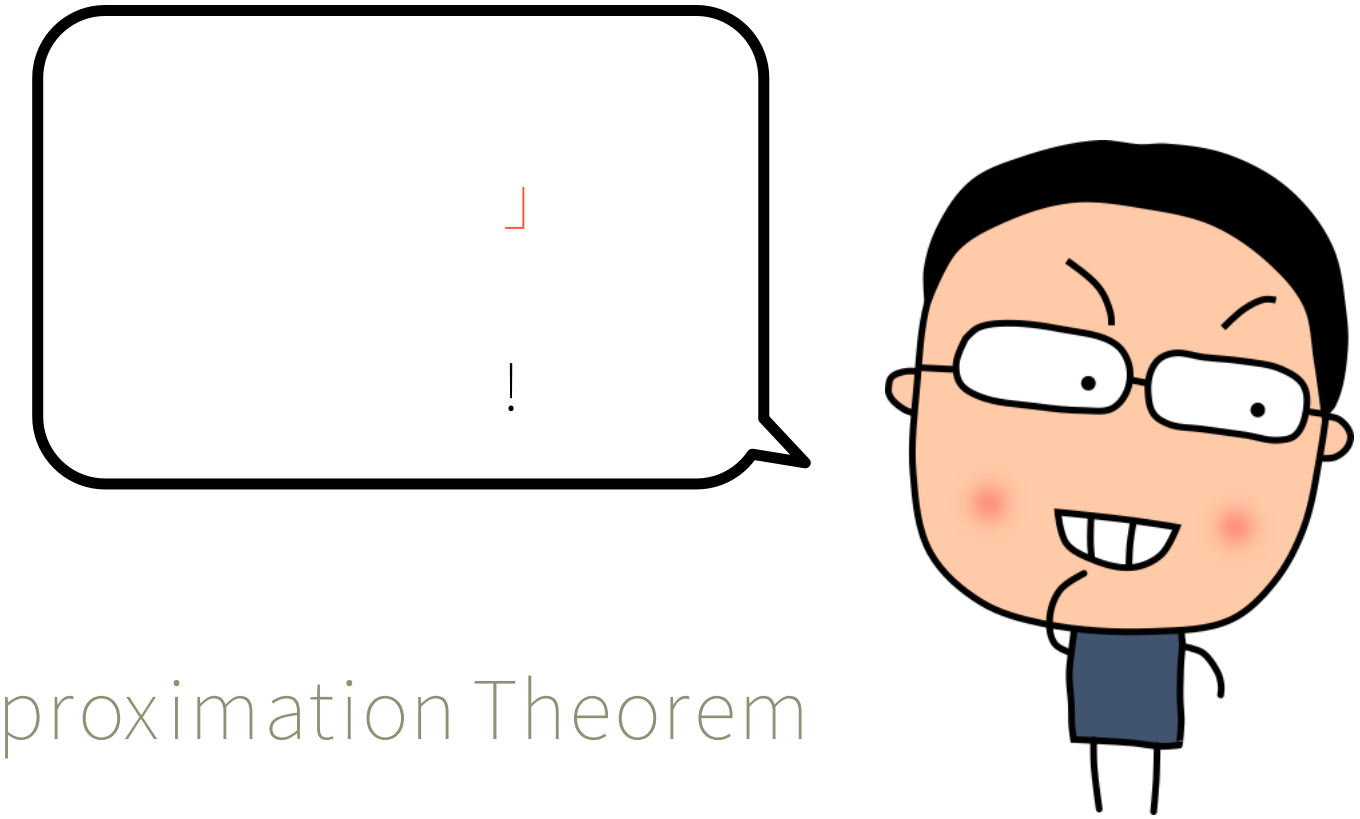
1 ,







Source: Yen-Long Tsai



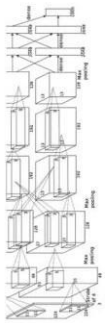
Universal Approximation Theorem

Source: Yen-Long Tsai

為什麼叫做 Deep Learning？

- 當 hidden layers 層數夠多 (一般而言大於三層)
  就稱為 Deep neural network

8 layers



16.4%

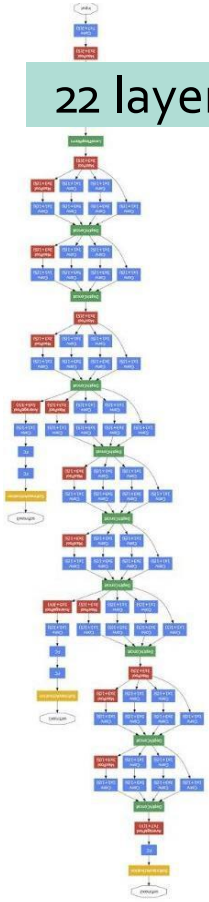
AlexNet (2012)

19 layers

7.3%

VGG (2014)

22 layers



6.7%

GoogleNet (2014)

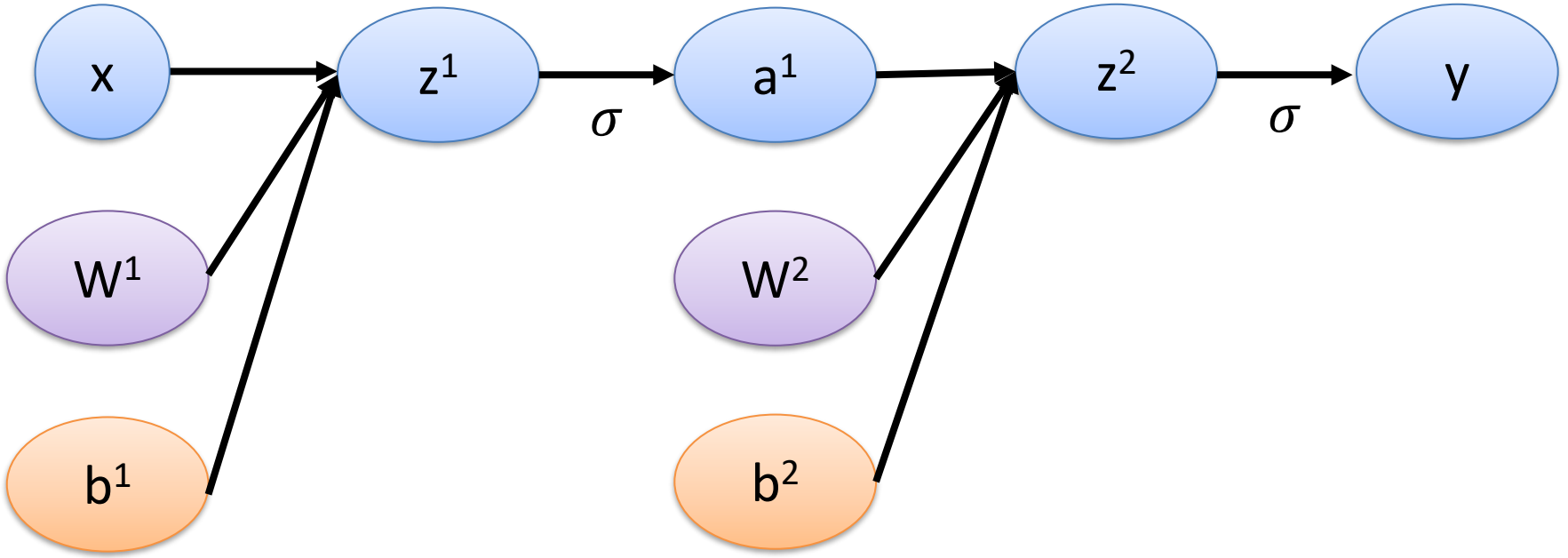
(Slide Credit: Hung-Yi Lee) http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

NEURAL NETWORK TRAINING (BACK PROPAGATION)

Neural Network in Math Form
Feedforward Network

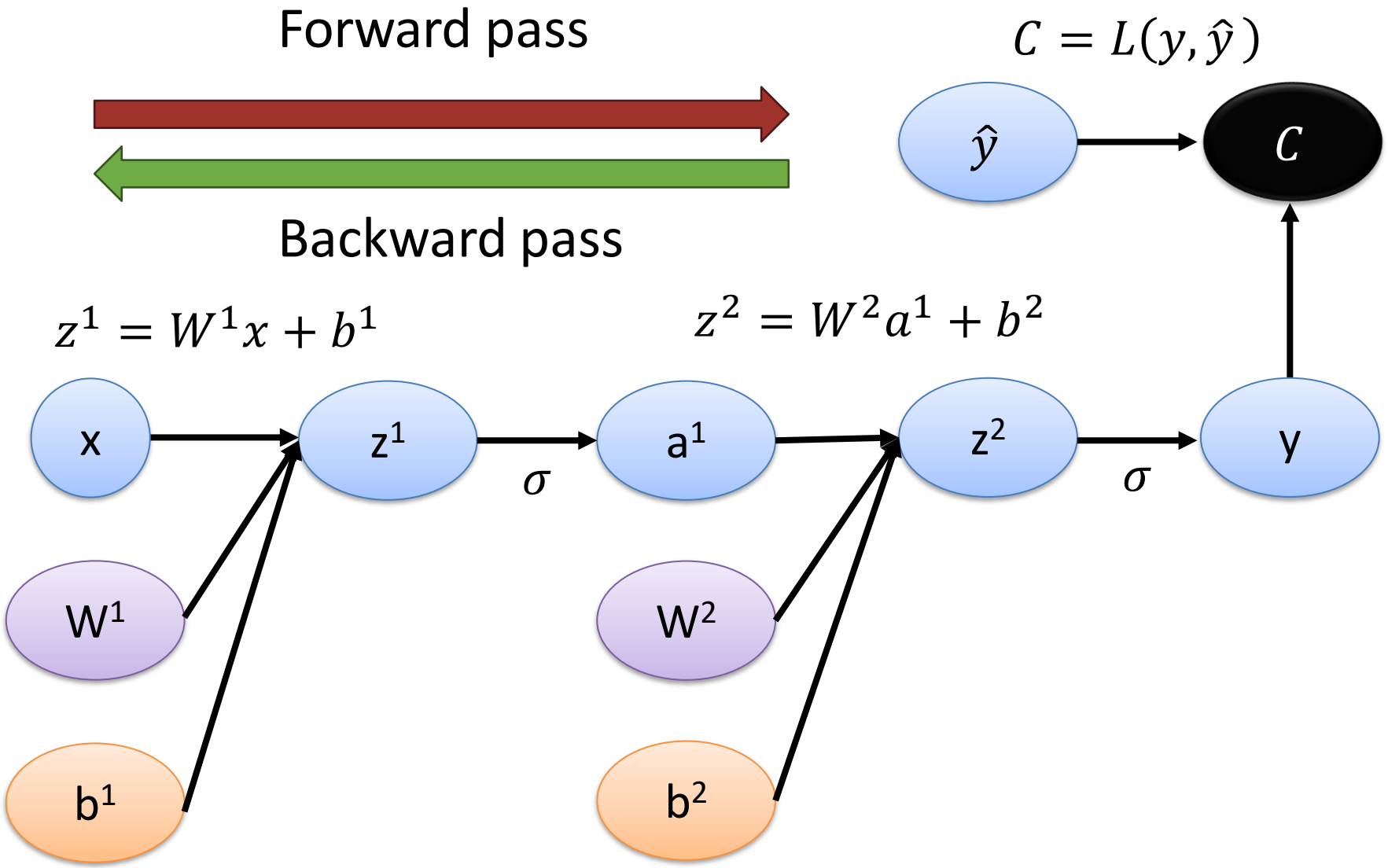
$$y = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

$$z^1 = W^1 x + b^1 \qquad z^2 = W^2 a^1 + b^2$$



Source: Optimization, Hung-Yi Lee

Loss Function of Feedforward Network



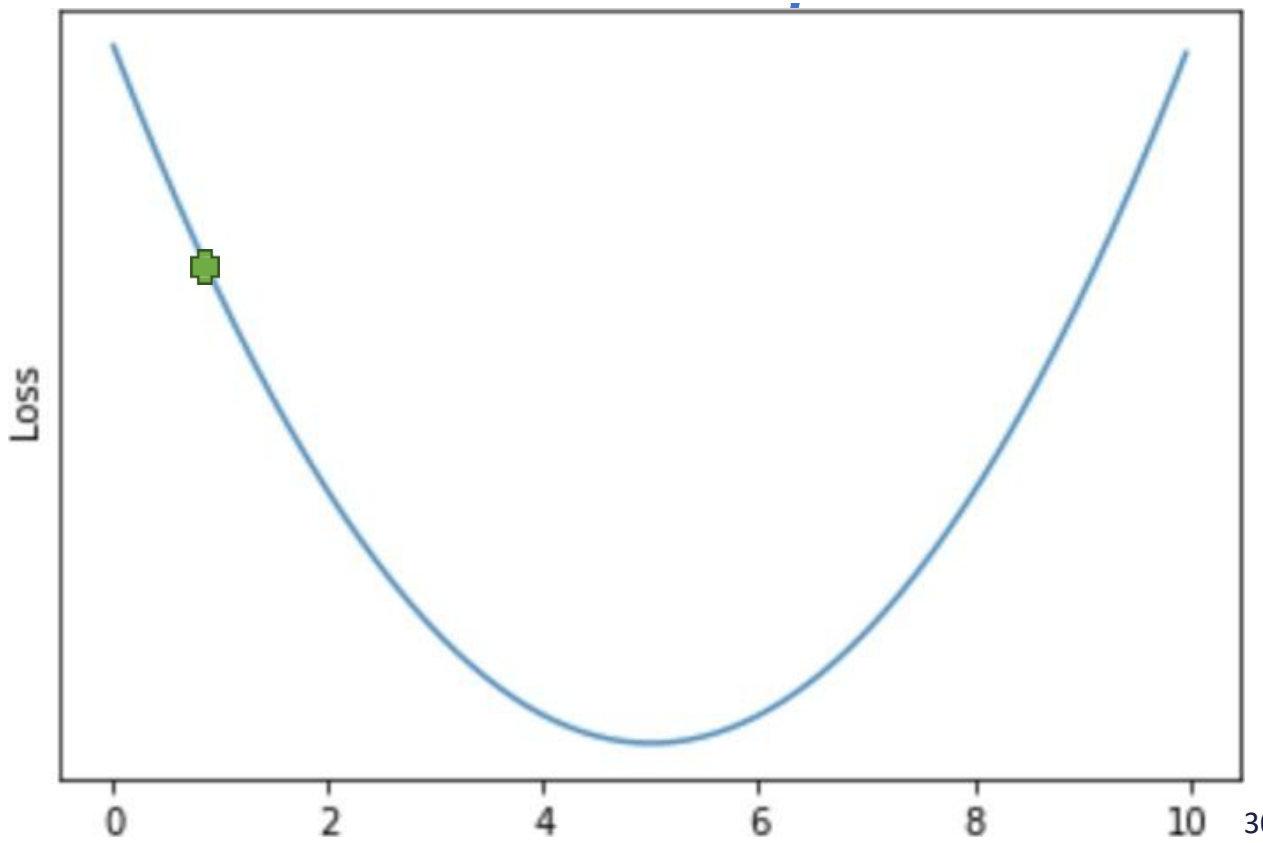
Source: Optimization, Hung-Yi Lee

Minimization of a function

- Optimize the network to minimize the loss with respect to all neurons:

$$\mathcal{L}_{CE}(\hat{y}, y)$$

$$-\nabla_{\theta}\mathcal{L}_{CE}(\hat{y}, y)$$

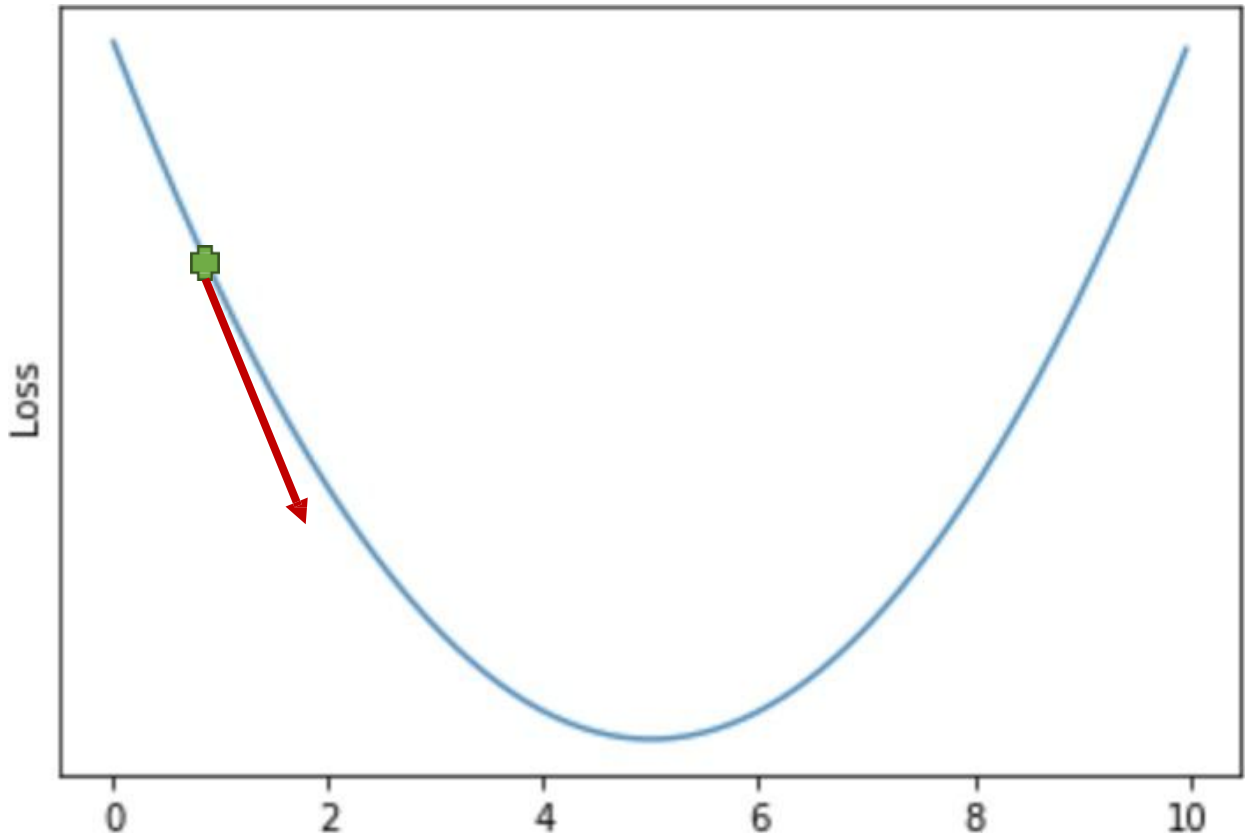


Minimization of a function

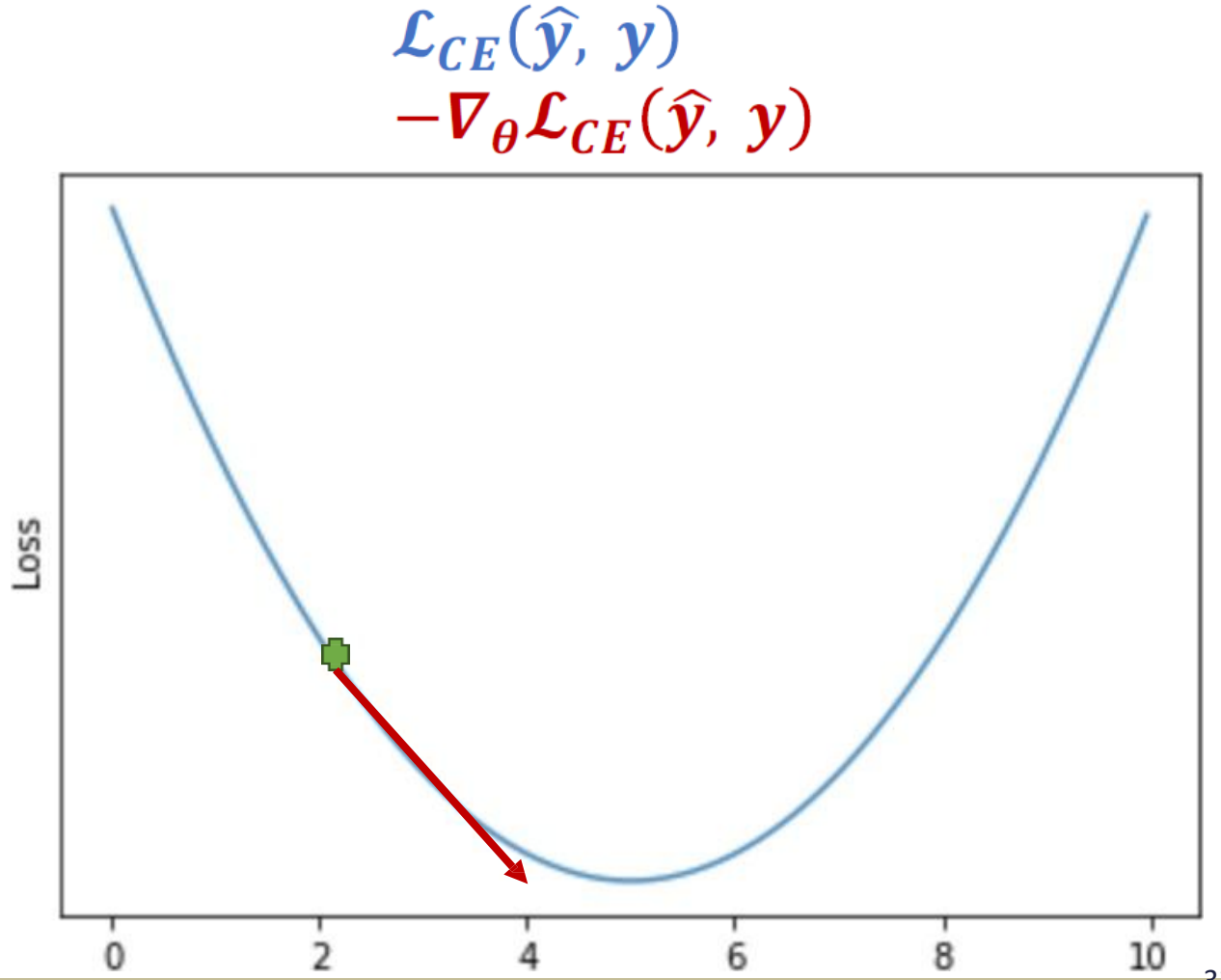
- Optimize the network to minimize the loss with respect to all neurons θ:

$$\mathcal{L}_{CE}(\hat{y}, y)$$

$$-\nabla_{\theta}\mathcal{L}_{CE}(\hat{y}, y)$$



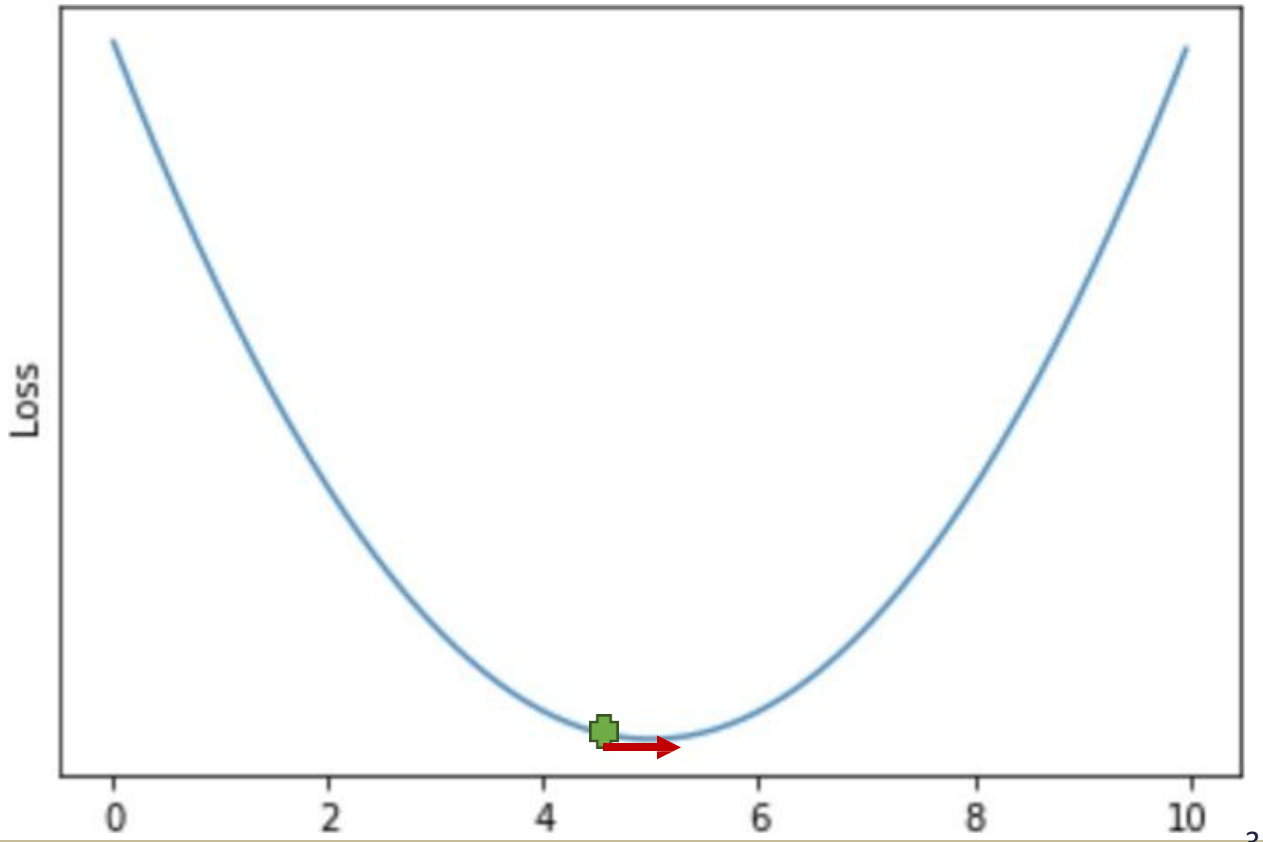
Minimization of a function



Minimization of a function

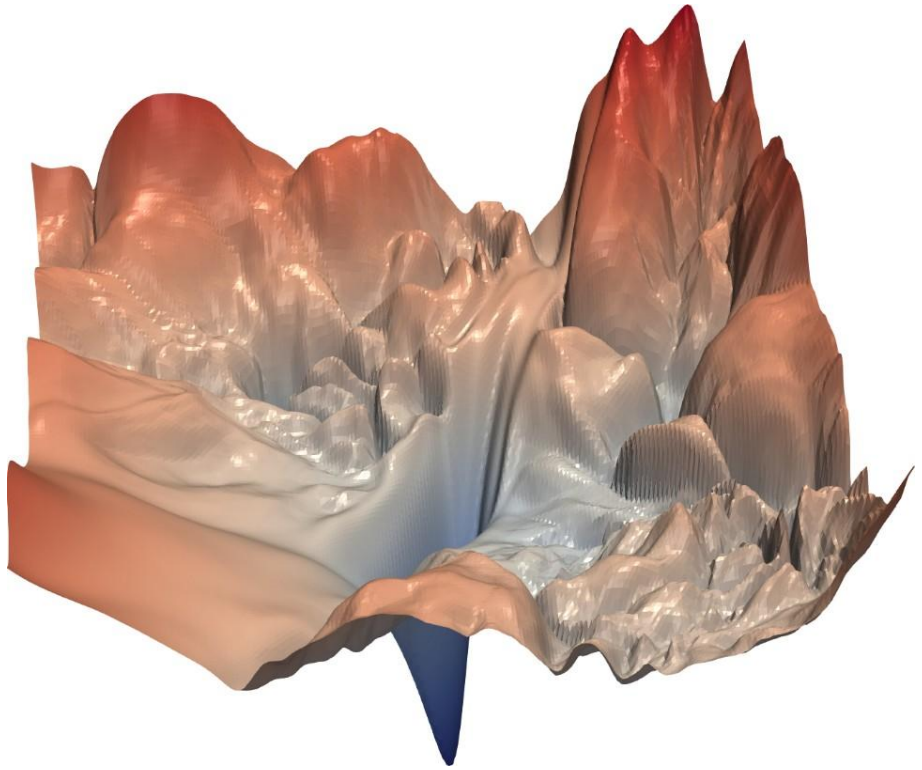
$\mathcal{L}_{CE}(\hat{y}, y)$

$-\nabla_{\theta}\mathcal{L}_{CE}(\hat{y}, y)$



Minimization of a function in DL models

- Million of parameters to optimize together!
  - Highly non-convex!

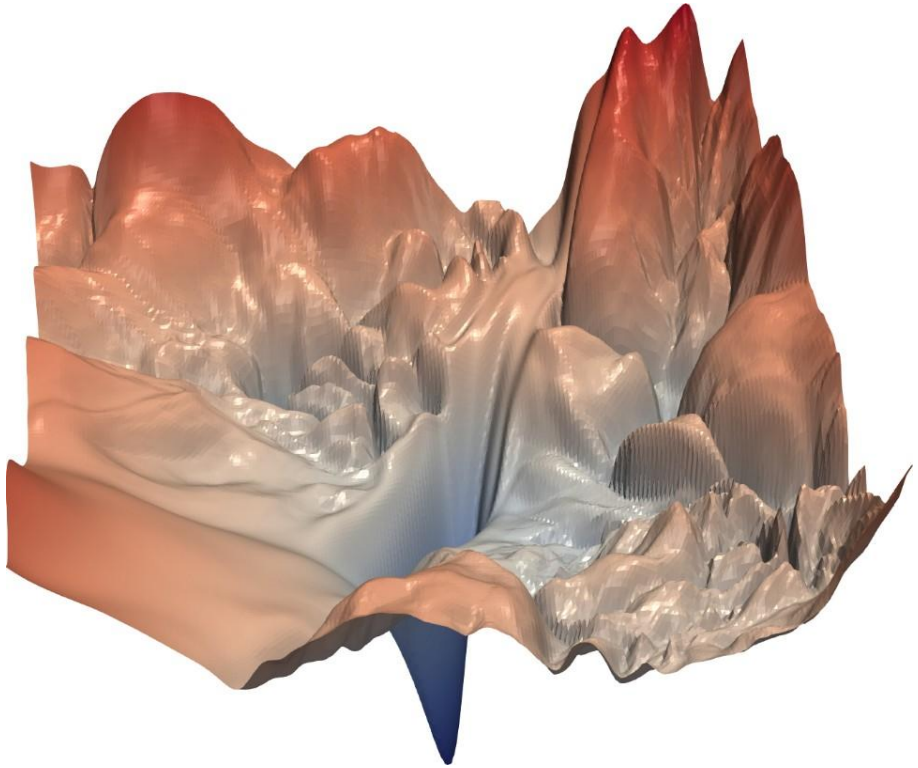


Multiple dimensional derivatives are called **gradients**

Visualizing the Loss Landscape of Neural Nets, [Li et al. NeurIPS2018]

Minimization of a function in DL models

- In theory a lot of bad local minima, but in practice a DNN usually find a good local minima close to optimum [LeCun et al. Nature 2015].



Although, this is debated [Goldblum et al. ICLR 2020]

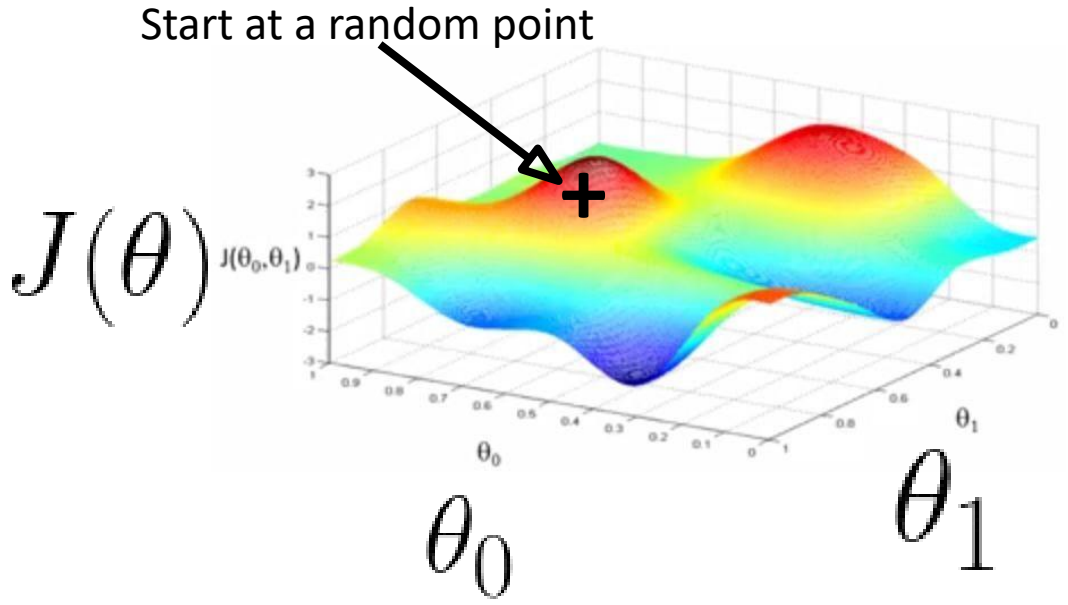
Training Neural Network: Objective and Gradient Descent

$$arg_\theta \min \frac{1}{N} \sum_i^N \text{loss}(f(x^{(i)};\theta), y^{(i)})$$

$J(\theta)$

loss function

$$\theta = W_1, W_2 \ldots W_n$$

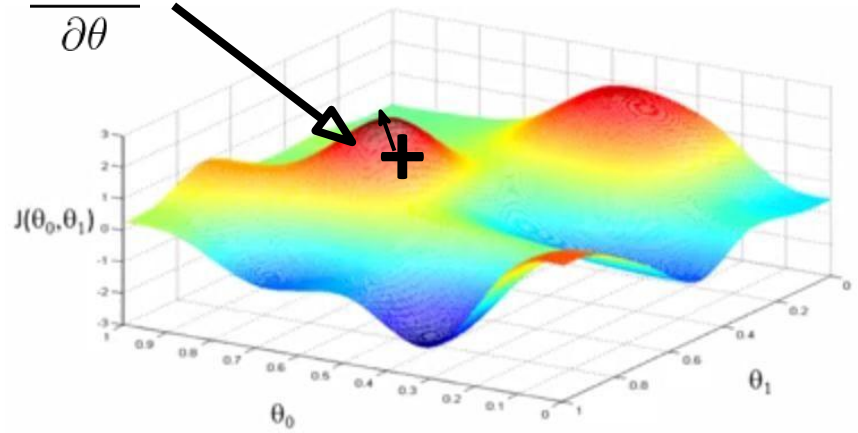


Source: Nick Locascio, intro to deep learning,

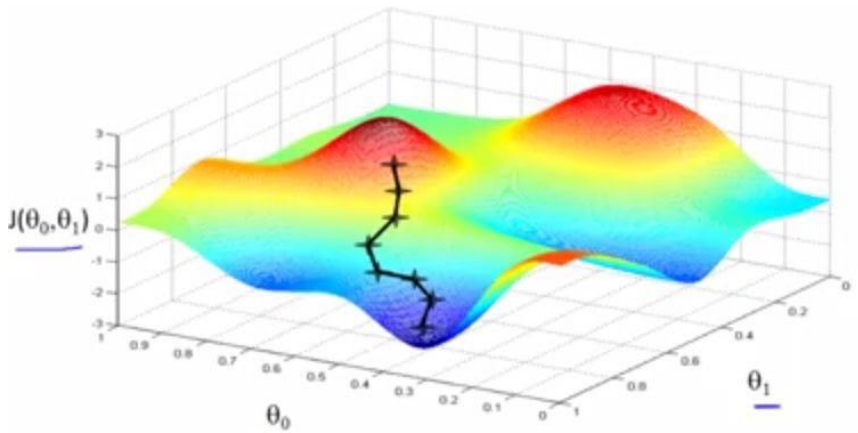
How to Minimize Loss?

Compute gradient

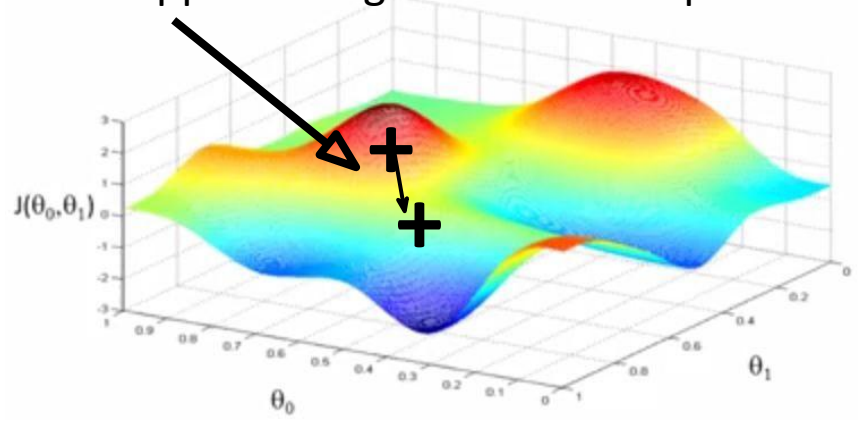
$$\frac{\partial J(\theta)}{\partial \theta}$$



Repeat



Move in direction opposite of gradient to new point

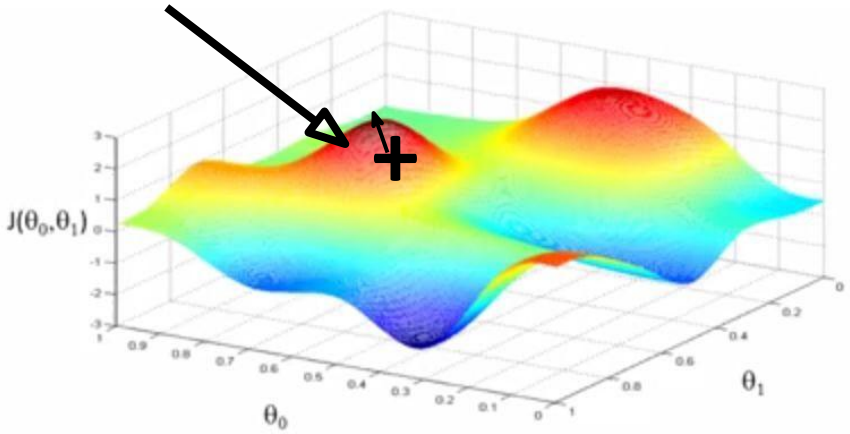


This is called stochastic gradient descent (SGD)

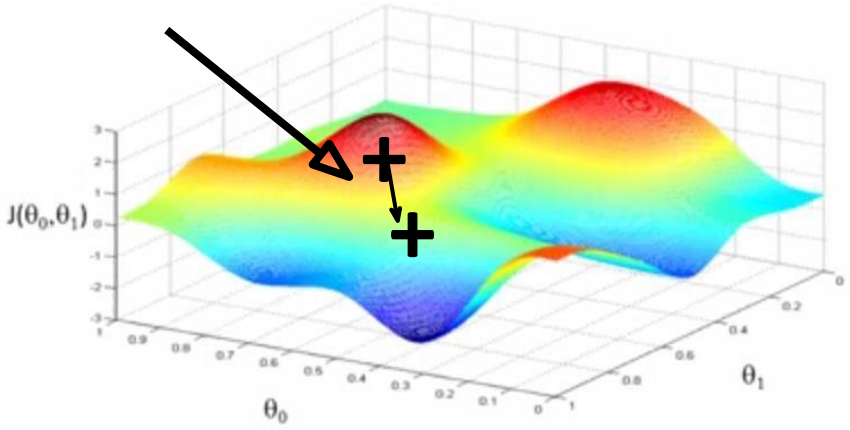
Source: Nick Locascio, intro to deep learning,

Gradient Descent (梯度下降法)

計算梯度 $\frac{\partial L}{\partial \theta}$



沿著 gradient 的反方向走



Initialize θ randomly

For N Epochs

- For each training example (x, y):
  - Compute Loss Gradient: $\frac{\partial L}{\partial \theta}$
  - Update θ with update rule:

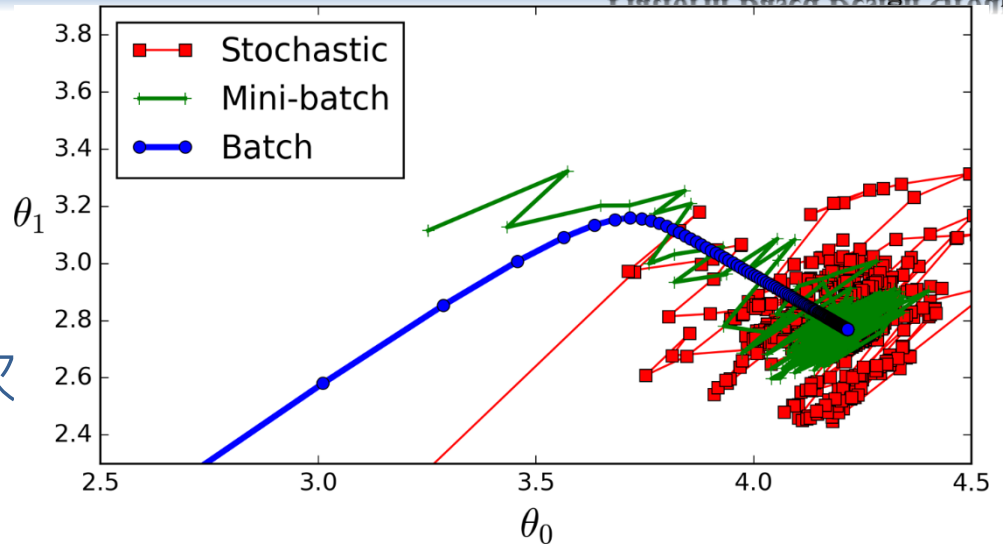
$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$

沿著 gradient 的反方向走　　Learning rate

- How to Compute Gradient? $\frac{\partial L}{\partial \theta}$

Source: Nick Locascio, intro to deep learning,

Gradient Descent 的缺點



- 一個 epoch 更新一次，收斂速度很慢
  - 一個 epoch 等於看過所有 training data 一次

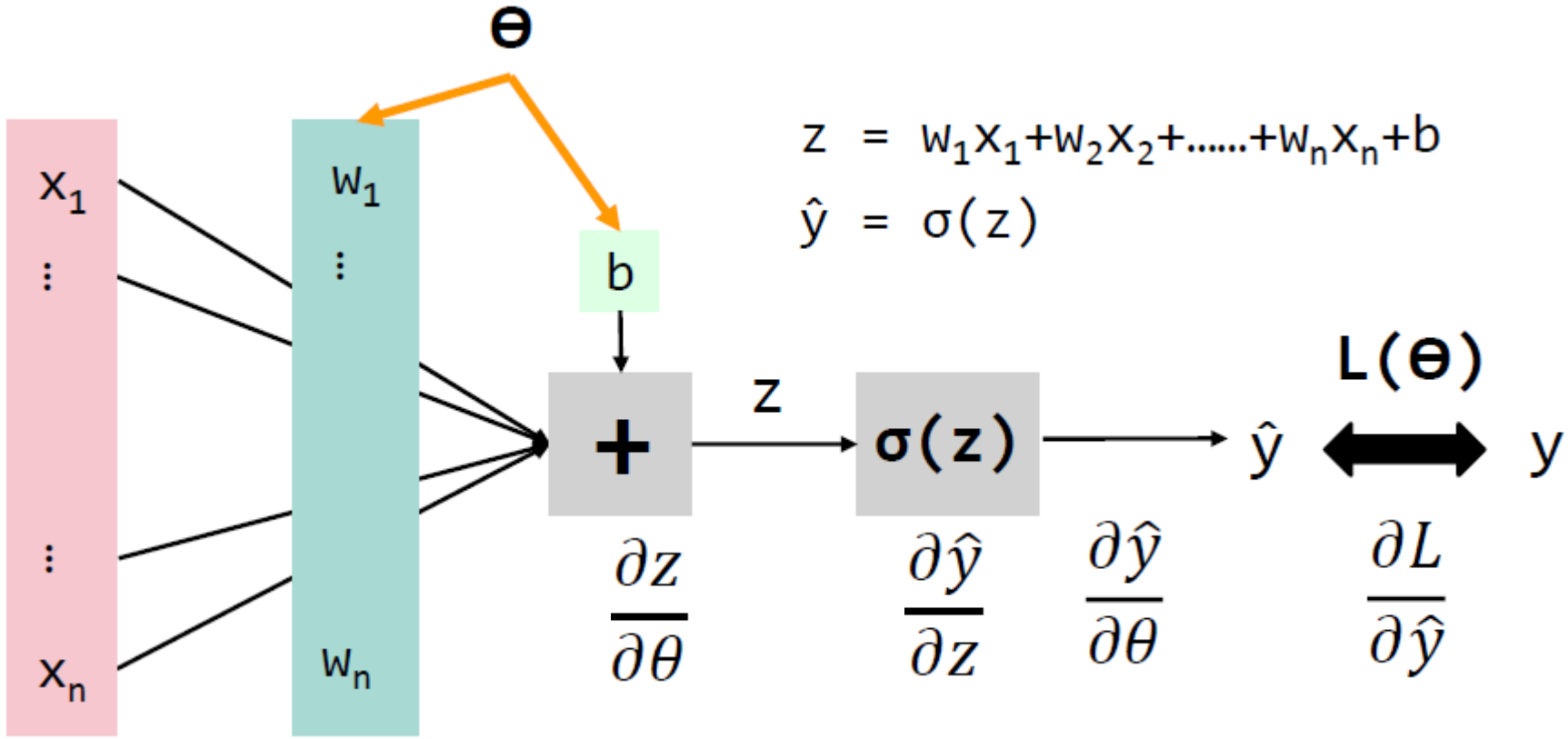
加速方法

- 隨機抽一筆資料 stochastic gradient descent (SGD)
  - 一筆一筆更新也很慢
- 一次看多筆資料 mini-batch SGD (主流方法)
  - 依照 mini-batch 把所有 training data 拆成多份
  - 假設全部有 1000 筆資料
    - Batch size = 100 可拆成 10 份 ， 一個 epoch 內會更新 10 次
    - Batch size = 10 可拆成 100 份 ， 一個 epoch 內會更新 100 次
  - 如何設定 batch size?
    - 常用 32, 128, 256, …

https://www.zhihu.com/question/64134994

Back Propagation

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$



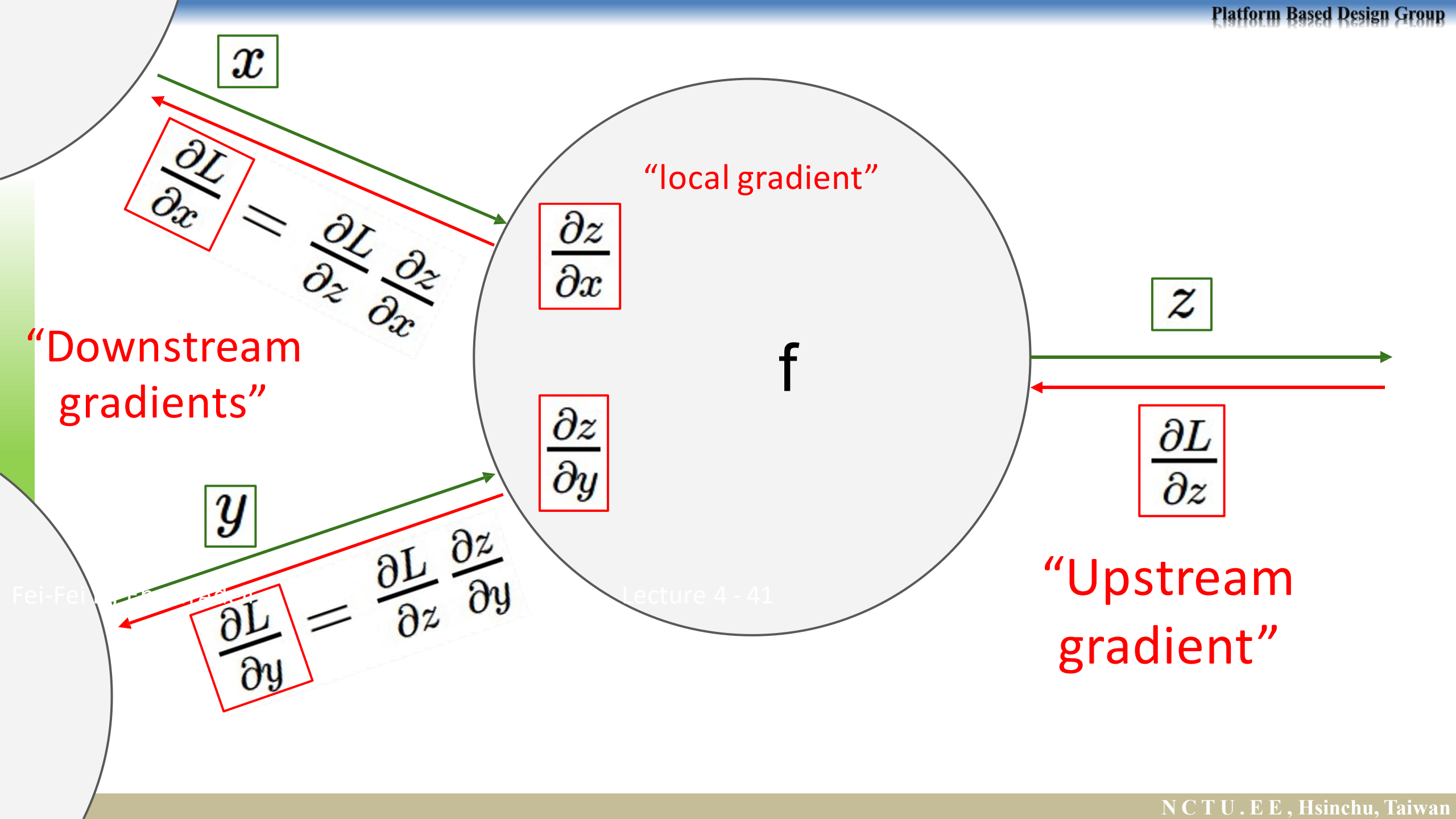
$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta}$$

Upstream gradient

Local gradient

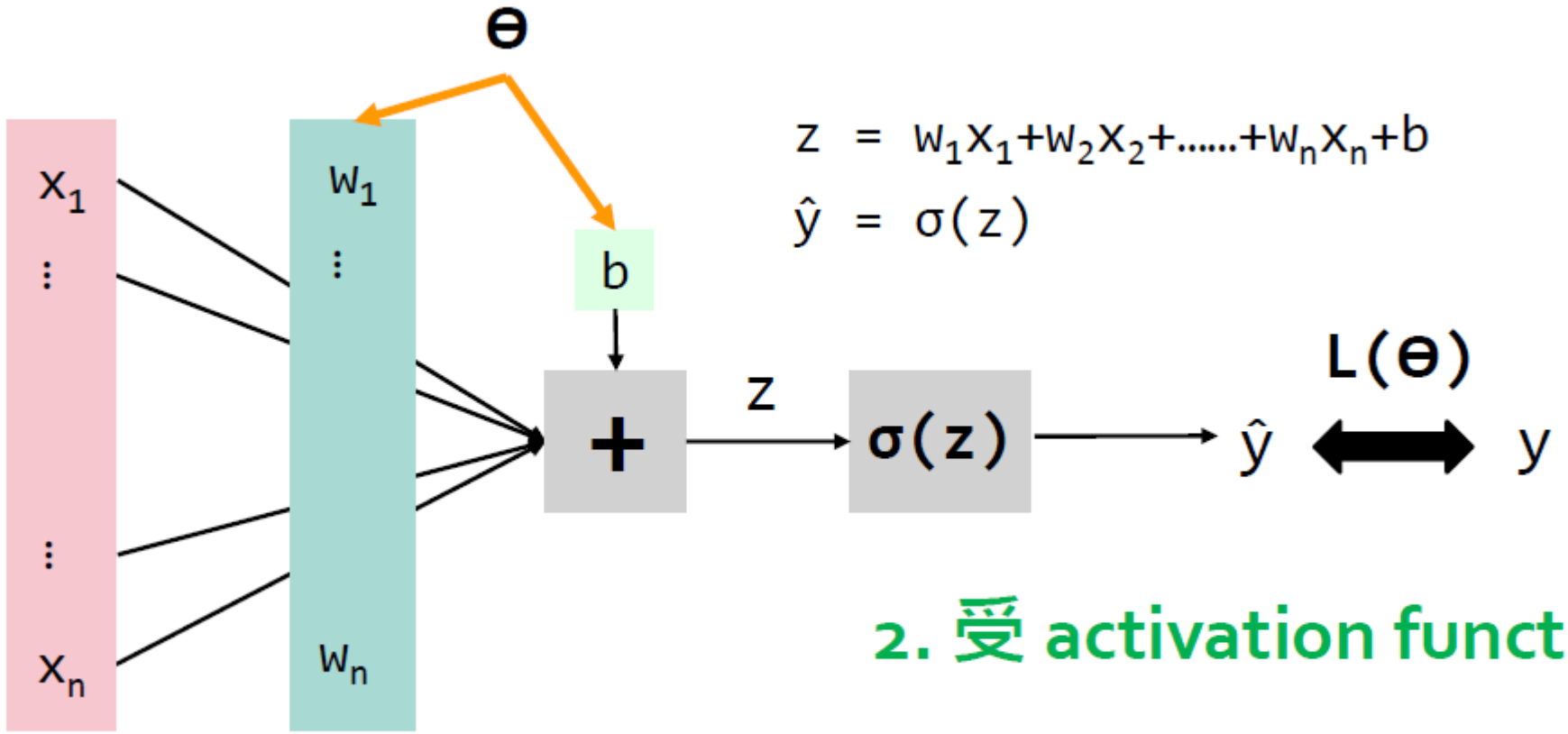
Chain rule (又叫 Back propagation 倒傳遞)

Source:手把手教你深度學習實務


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"Downstream gradients"
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

Back Propagation

$$\theta = \theta - \eta \frac{\partial L}{\partial \theta}$$



$$z = w_1 x_1 + w_2 x_2 + \ldots\ldots + w_n x_n + b$$

$$\hat{y} = \sigma(z)$$

2. 受 activation function 影響

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \theta} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial \theta}$$

Chain rule (又叫 Back propagation 倒傳遞)

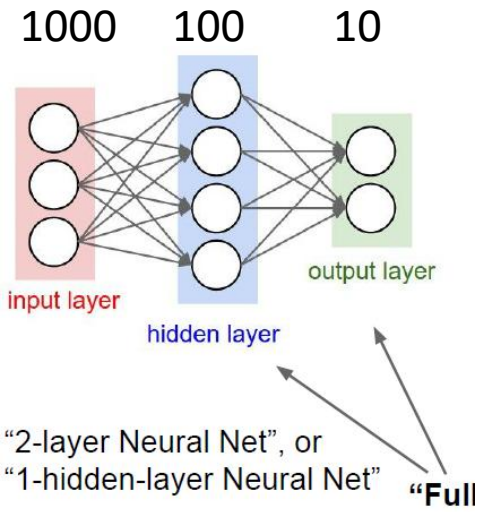
1. 受 loss function 影響

Source:手把手教你深度學習實務

A 2 layer NN with ~ 20 lines of code

```
1  import numpy as np
2  from numpy.random import randn
3
4  N, D_in, H, D_out = 64, 1000, 100, 10
5  x, y = randn(N, D_in), randn(N, D_out)
6  w1, w2 = randn(D_in, H), randn(H, D_out)
7
8  for t in range(2000):
9    h = 1 / (1 + np.exp(-x.dot(w1)))
10   y_pred = h.dot(w2)
11   loss = np.square(y_pred - y).sum()
12   print(t, loss)
13
14   grad_y_pred = 2.0 * (y_pred - y)
15   grad_w2 = h.T.dot(grad_y_pred)
16   grad_h = grad_y_pred.dot(w2.T)
17   grad_w1 = x.T.dot(grad_h * h * (1 - h))
18
19   w1 -= 1e-4 * grad_w1
20   w2 -= 1e-4 * grad_w2
```

- Random input and output
- Random weight initialization
- Forward pass
  - Sigmoid activation
  - MSE loss
- Backward pass
  - Gradient for 2nd layer
  - Gradient for 1st layer
  - Weight update



Source: cs231n

APPENDIX CALCULUS ON COMPUTATIONAL GRAPHS:
BACKPROPAGATION (OPTIONAL)
HOW TO COMPUTE GRADIENT AUTOMATICALLY
AUTOMATIC DIFFERENTIATION

Reference

- Textbook: Deep Learning
  - Chapter 6.5
- Calculus on Computational Graphs: Backpropagation
  - https://colah.github.io/posts/2015-08-Backprop/
- On chain rule, computational graphs, and backpropagation
  - http://outlace.com/Computational-Graph/

Problem: How to compute gradients?

$$s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$$ Nonlinear score function

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$ SVM Loss on predictions

$$R(W) = \sum_k W_k^2$$ Regularization

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W_1) + \lambda R(W_2)$$ Total loss: data loss + regularization

If we can compute $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}$, we can learn W_1 and W_2

(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

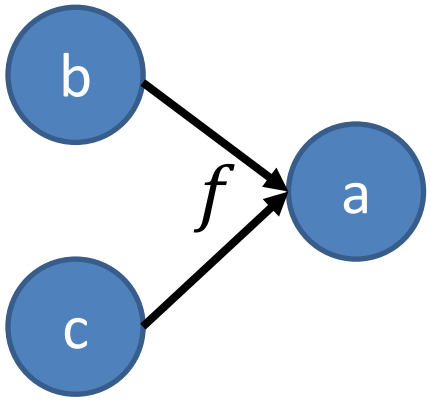
Problem: Very tedious: Lots of matrix calculus, need lots of paper

Problem: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

Problem: Not feasible for very complex models!

Computational Graph

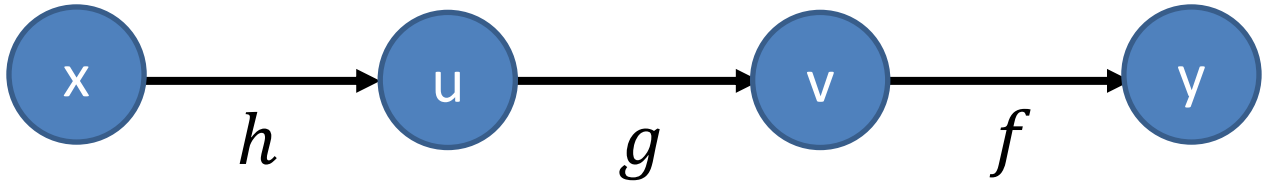
$$a = f(b, c)$$



- A “language” describing a function
  - Node: variable (scalar, vector, tensor ……)
  - Edge: operation (simple function)

Example $y = f\left(g\left(h(x)\right)\right)$

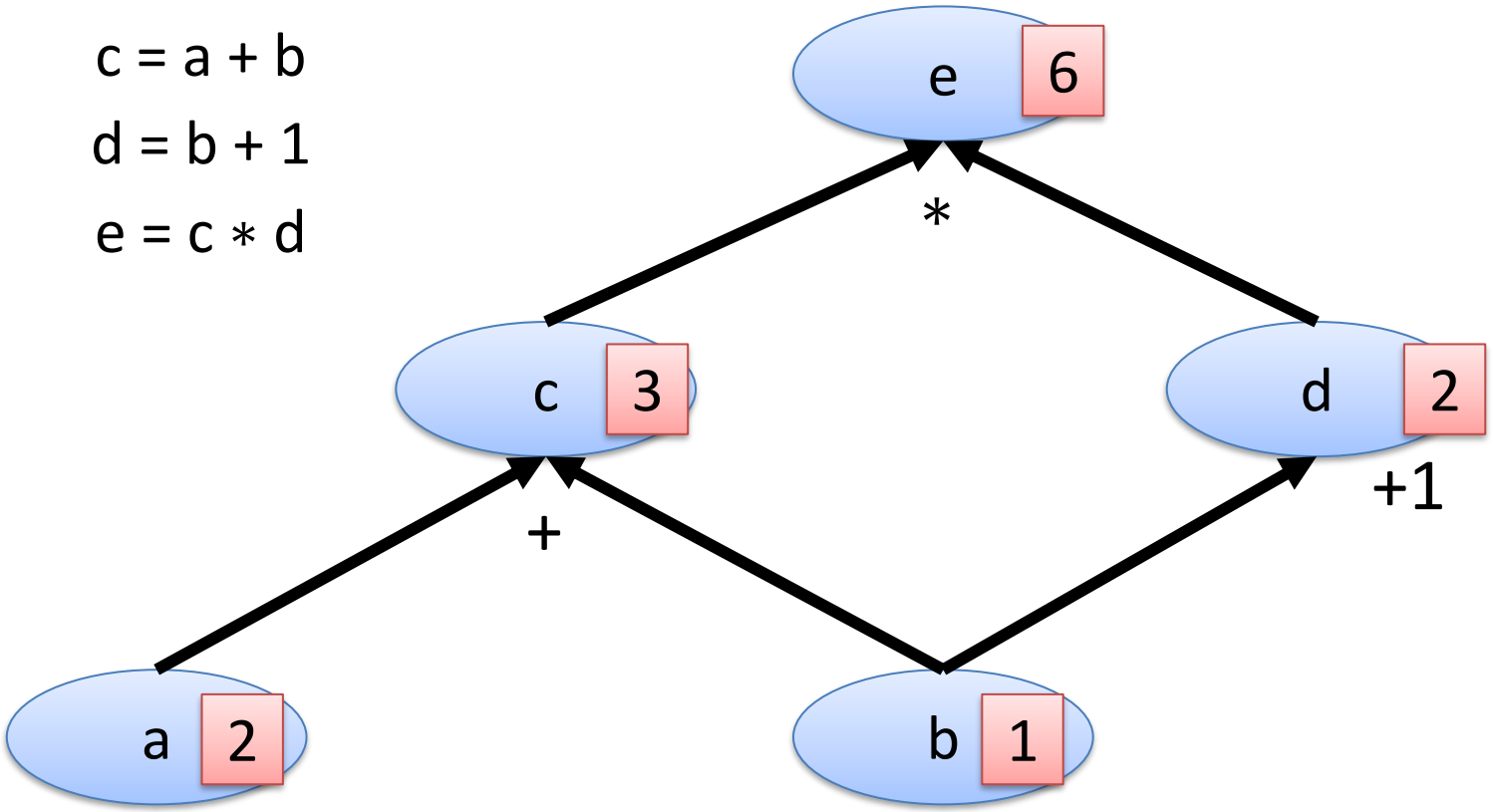
$$u = h(x) \quad v = g(u) \quad y = f(v)$$



Source: Backpropagation,李宏毅

Computational Graph

- Example: e = (a+b) ∗ (b+1)

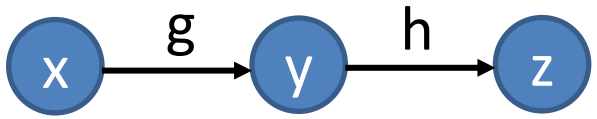


Source: Backpropagation,李宏毅

Review: Chain Rule

Case 1

$$z = f(x) \Rightarrow y = g(x) \quad z = h(y)$$

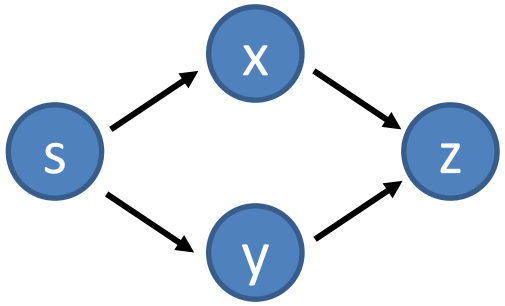


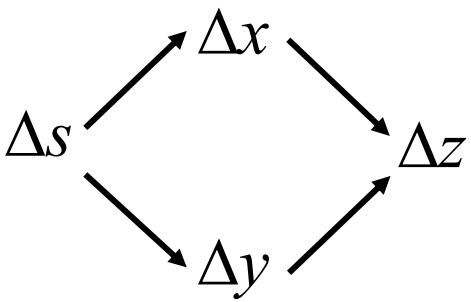
$$\Delta x \to \Delta y \to \Delta z$$

$$\frac{dz}{dx}$$

Case 2

$$z = f(s) \Rightarrow x = g(s) \quad y = h(s) \quad z = k(x, y)$$





$$\frac{dz}{ds}$$

Source: Backpropagation,李宏毅

Computational Graph

- Example: e = (a+b) ∗ (b+1)

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

c = a + b

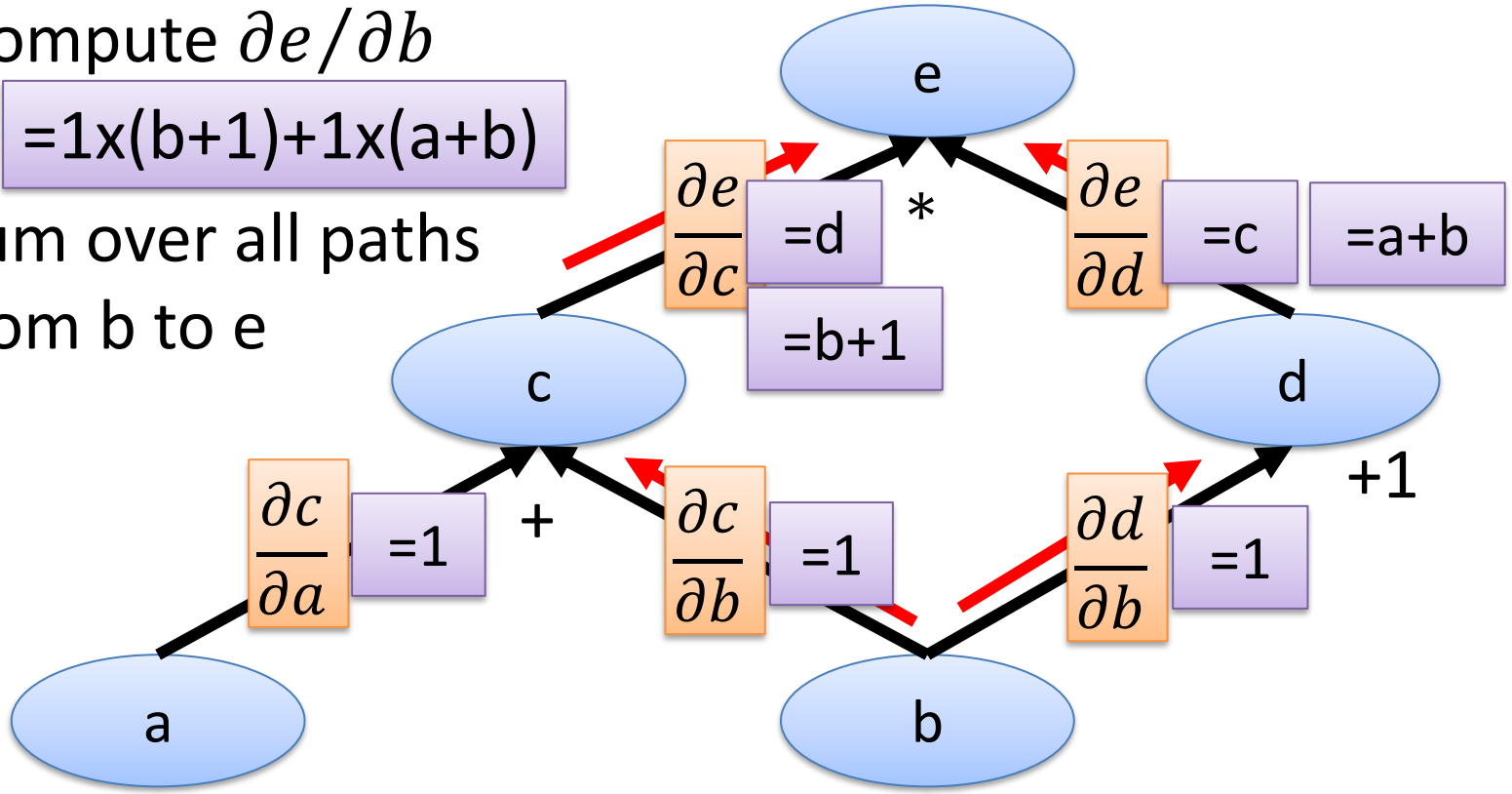
d = b + 1

e = c ∗ d

Compute $\partial e/\partial b$

=1x(b+1)+1x(a+b)

Sum over all paths from b to e



Source: Backpropagation,李宏毅

Computational Graph

- Example: e = (a+b) ∗ (b+1)

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

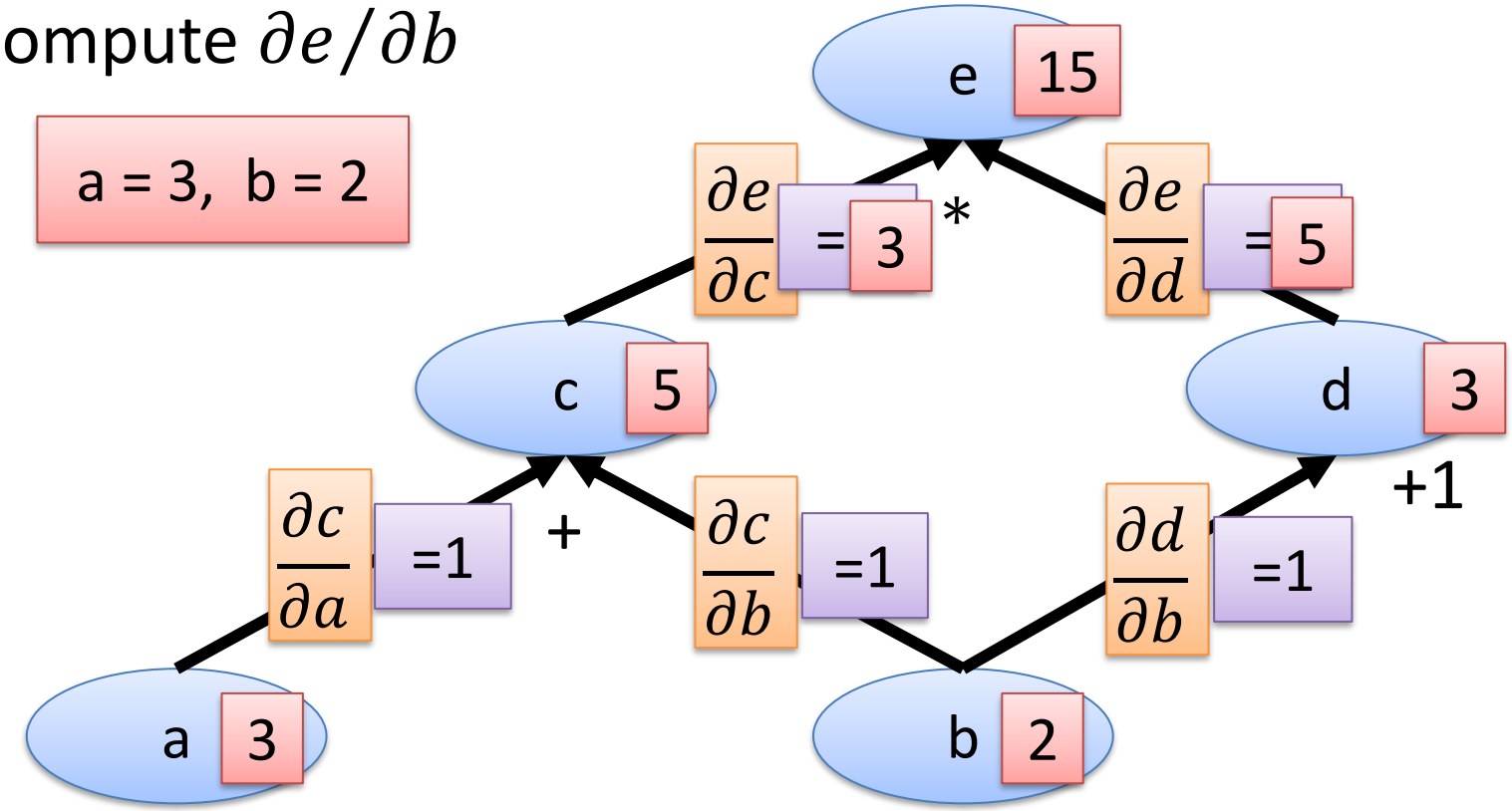
c = a + b

d = b + 1

e = c ∗ d

Compute $\partial e/\partial b$

a = 3, b = 2

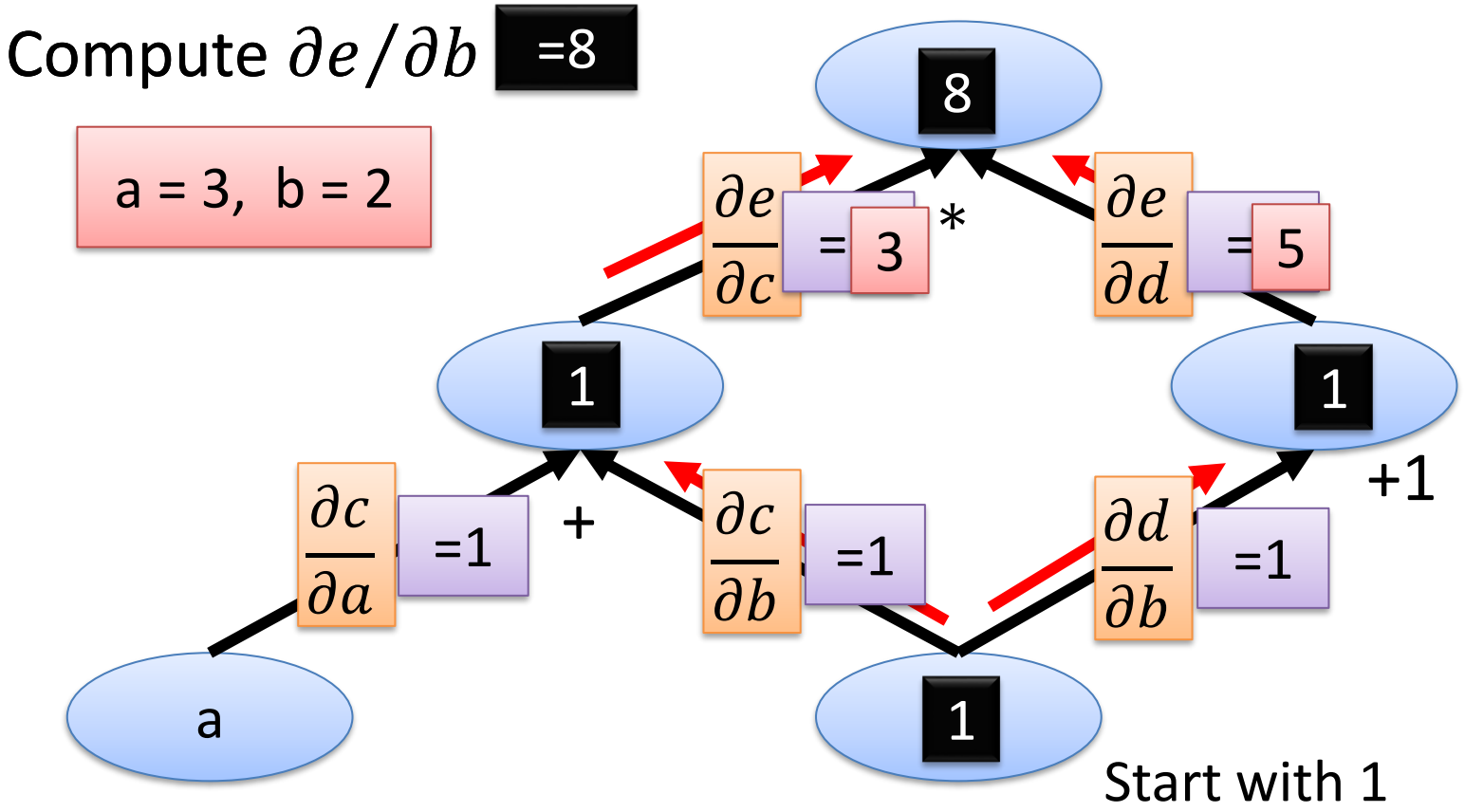


Source: Backpropagation,李宏毅

Computational Graph

Forward mode

- Example: e = (a+b) ∗ (b+1)

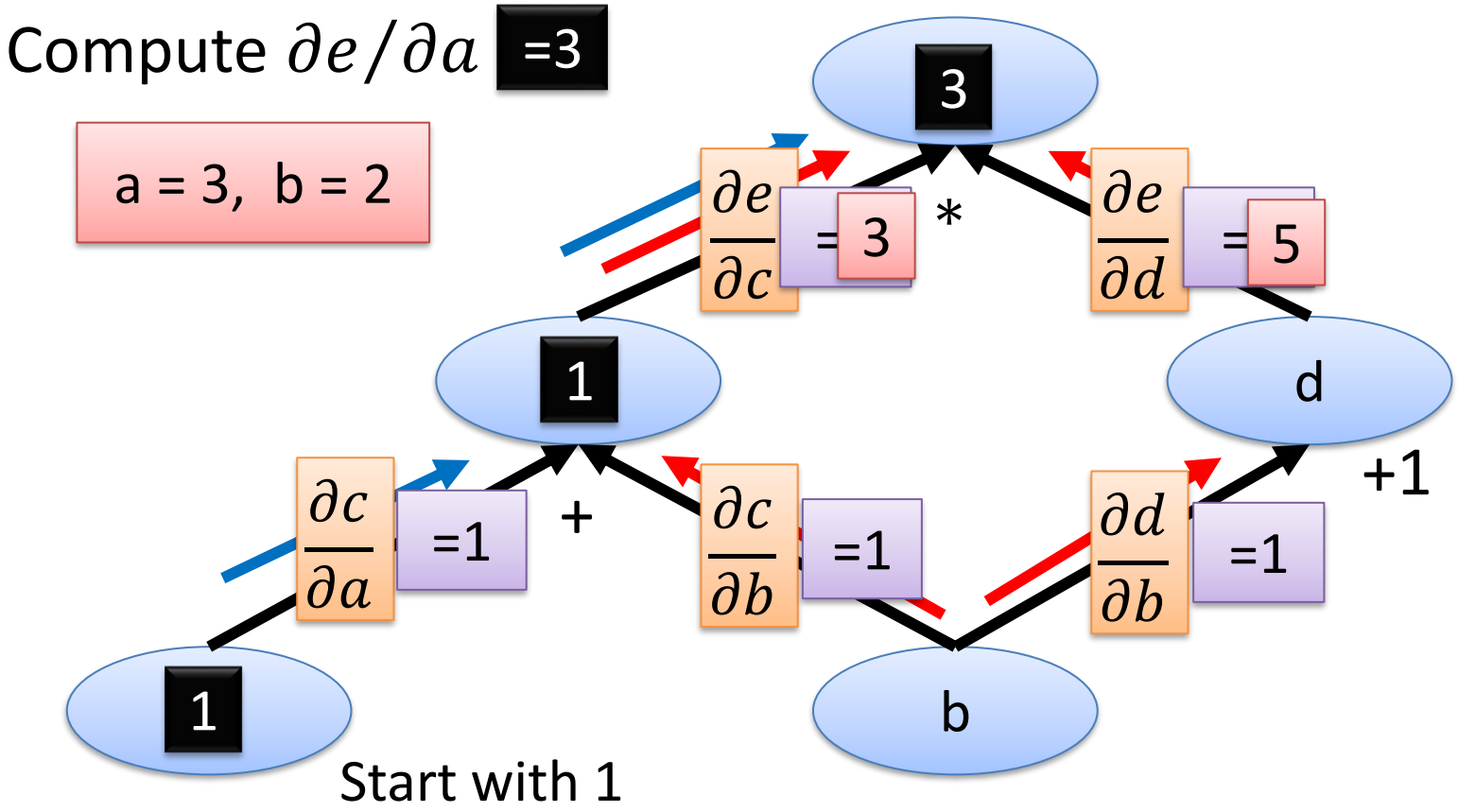


Source: Backpropagation,李宏毅

Computational Graph

Forward mode

- Example: e = (a+b) ∗ (b+1)



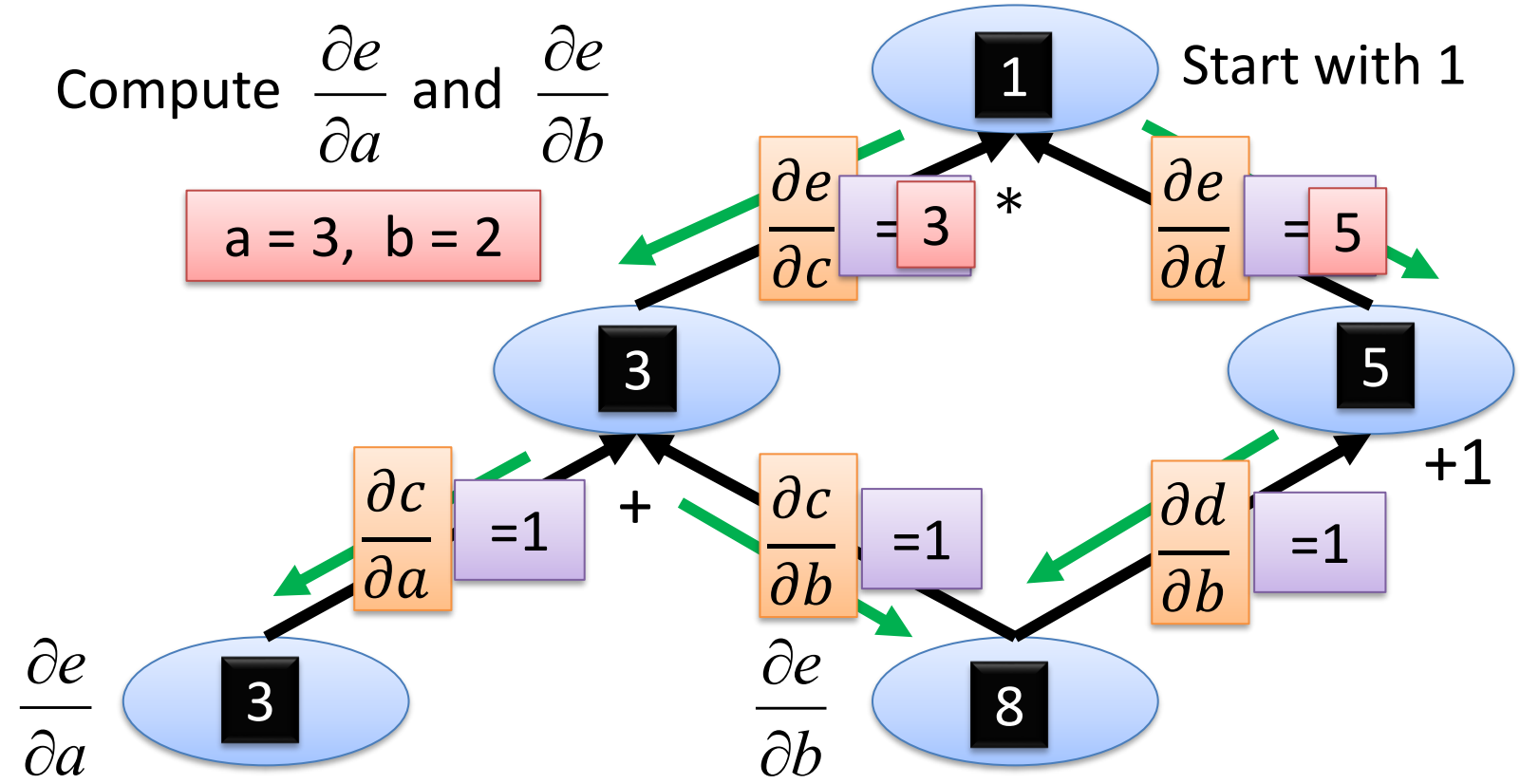
Source: Backpropagation,李宏毅

Computational Graph

- Example: e = (a+b) ∗ (b+1)

Reverse mode

What is the benefit?



Source: Backpropagation,李宏毅

Forward Mode v.s. Reverse Mode

- Forward mode
  - Forward-mode differentiation starts at an input to the graph and moves towards the end. At every node, it sums all the paths feeding in. Each of those paths represents one way in which the input affects that node. By adding them up, we get the total way in which the node is affected by the input, it's derivative.
  - tracks how one input affects every node
  - imagine a function with a million inputs and one output. Forward-mode differentiation would require us to go through the graph a million times to get the derivatives.
  - Good for a function with lots of outputs
- Reverse mode
  - starts at an output of the graph and moves towards the beginning. At each node, it merges all paths which originated at that node
  - tracks how every node affects one output
  - Reverse-mode differentiation can get them all in one fell swoop for a function with a million inputs and one output.
    - Massive speedup

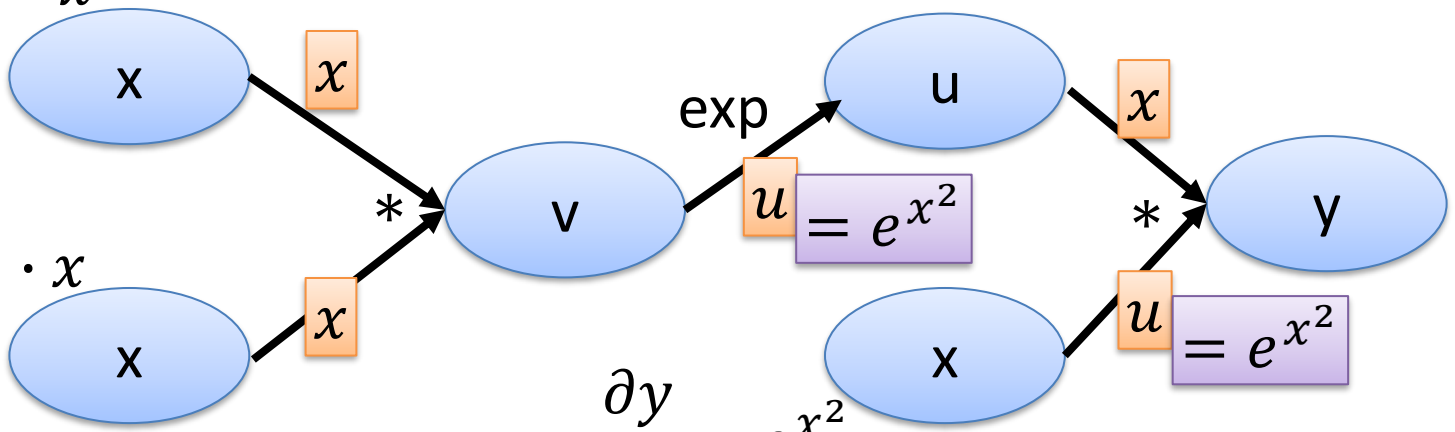
Computational Graph

- Parameter sharing: the same parameters appearing in different nodes

$$y = xe^{x^2} \qquad \frac{\partial y}{\partial x} =? \quad e^{x^2} + x \cdot e^{x^2} \cdot 2x$$

$$\frac{\partial y}{\partial x} = x \cdot e^{x^2} \cdot x$$

$$\frac{\partial y}{\partial x} = x \cdot e^{x^2} \cdot x$$



Source: Backpropagation,李宏毅

Reverse Mode Differentiation and its Automatic Application to Code

- Backpropagation is an special case of reverse mode differentiation
- Old idea, but software support for automatic differentiation is not perfect outside neural network
- Efficient differentiation (Reverse mode differentiation) is surprising recent
- Software tools like Theano and Tensorflow or other recent tools do automatic differentiation
  - Do most of what people want and easy to use
  - That is also why they are widely adopted

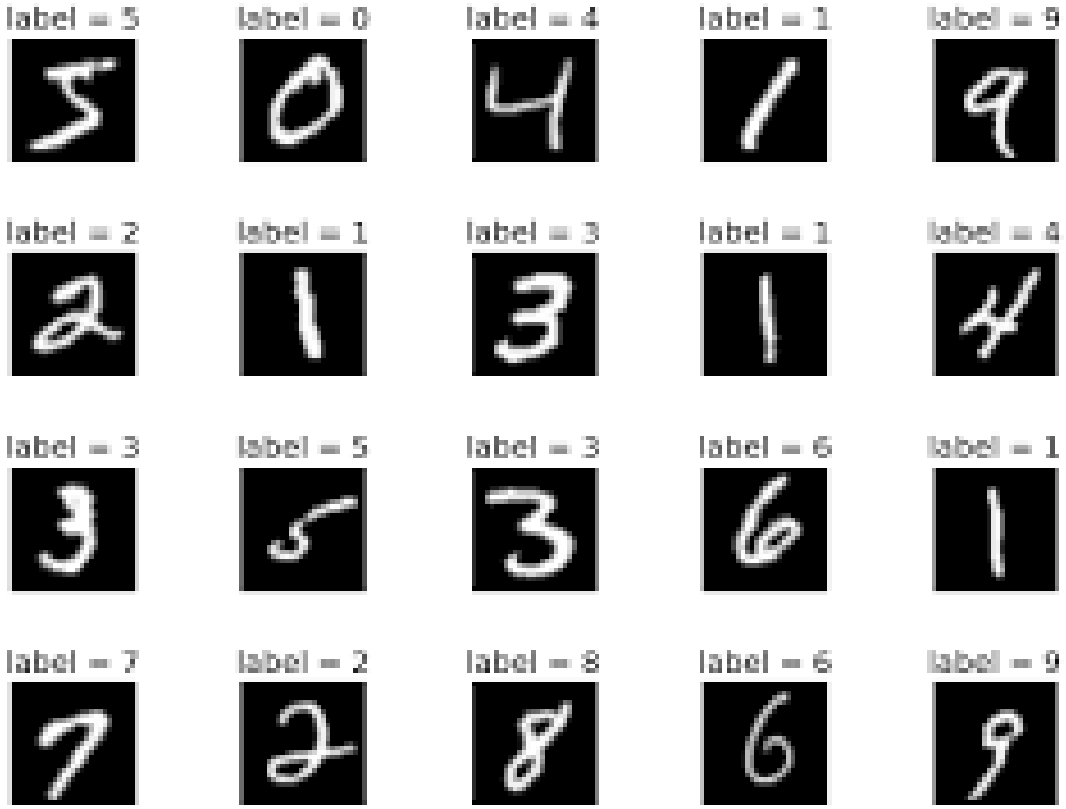
自動微分與反向傳播（AD）

- 計算圖與鏈式法則：節點（張量 / 運算）、邊（Jacobian）。
- Reverse-mode AD（反向模式）對「多參數→單標量 loss」最有效，是 DL 主流。
- PyTorch Autograd 的基本行為：
  - requires_grad、backward()、.grad 累積、zero_grad。
- 常見陷阱：
  - 多次 backward 需 retain_graph；
  - train() vs eval() 只影響 BN/Dropout；推論用 torch.no_grad()。

NN EXAMPLE WITH KERAS/PYTORCH (OPTIONAL)

MNIST: 手寫數字辨識

- https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/2.1-a-first-look-at-a-neural-network.ipynb



First 2-Layer NN Example with Keras

- Live demo
- https://github.com/fchollet/deep-learning-with-python-notebooks

Listing 2.1 Loading the MNIST dataset in Keras

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

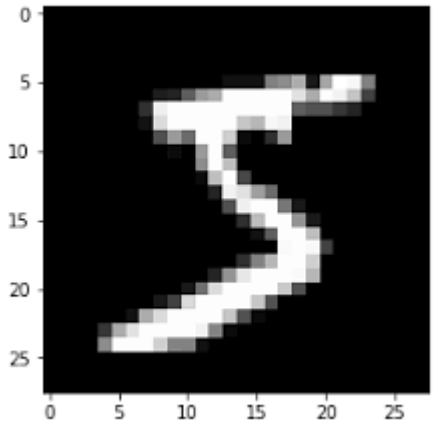
Listing 2.2 The training data

```
>>> train_images.shape
(60000, 28, 28)
>>> len(train_labels)
60000
>> train_labels
[5 0 4 ..., 5 6 8]
```

Listing 2.3 The test data

```
>>> test_images.shape
(10000, 28, 28)
>>> len(test_labels)
10000
>>> test_labels
[7 2 1 ..., 4 5 6]
```

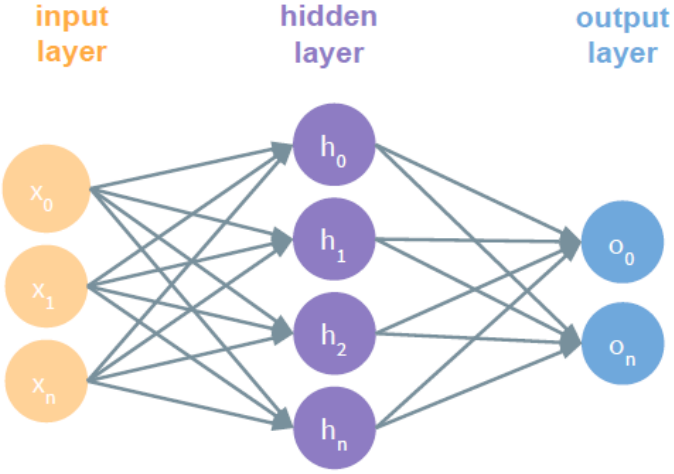
0. 資料前處理
先看看資料長的樣子



Listing 2.4 The network architecture

1. 網路結構

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```



Listing 2.5 The compilation step

2. 決定模型的 loss function
3. 訓練相關設定參數 (選擇optimizer)

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Listing 2.6 Preparing the image data

0. 資料前處理
Data normalization

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Listing 2.7 Preparing the labels

0. 資料前處理
Label 轉one-hot encoding

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Listing 2.8 Training the network

```
>>> network.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [==============================] - 9s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [========================>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

4. 編譯模型 (Compile model)
5. 開始訓練囉！(Fit model)

Listing 2.9 Evaluating the network

```
>>> test_loss, test_acc = network.evaluate(test_images, test_labels)
>>> print('test_acc:', test_acc)
test_acc: 0.9785
```

Keras in Tensorflow 2

Setup

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```


https://keras.io/examples/vision/mnist_convnet/

Prepare the data

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

0. 資料前處理
Data normalization

0. 資料前處理
Label 轉one-hot encoding

Build the model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Train the model

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/15
422/422 [==============================] - 13s 29ms/step - loss: 0.7840 - accuracy: 0.7643 - v
Epoch 2/15
422/422 [==============================] - 13s 31ms/step - loss: 0.1199 - accuracy: 0.9639 - v
Epoch 3/15
```


Evaluate the trained model

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.023950600996613503
Test accuracy: 0.9922000169754028
```

Tensorflow 2.0

- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quickstart/beginner.ipynb
- Crash course
  - https://colab.research.google.com/drive/17u-pRZJnKN0gO5XZmq8n5A2bKGrfKEUg?fbclid=IwAR2acj-FdOPkLCoehFiCQp_A7tD-ZmNEclpCncicOpi6DNcd8MxCWWGZdMY
  - https://colab.research.google.com/drive/1UCJt8EYjlzCs1H1d1X0iDGYJsHKwu-NO

Example

- https://github.com/fchollet/deep-learning-with-python-notebooks
  - 3.5 classifying movie reviews: binary classification
  - 3.6 classifying newwires: multi-class classifiation
  - 3.7 predicting house prices: regression

Pytorch:nn

- Running example: Train a two-layer ReLU network on random data with L2 loss

Higher-level wrapper for working with neural nets

Use this! It will make your life easier

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
    y_pred = model(x)           #forward pass (predict)
    loss = loss_fn(y_pred, y)   #calculate loss

    optimizer.zero_grad()       #clear accumulated gradients
    loss.backward()             #Backward pass (back propagation)
    optimizer.step()            #update weights
```

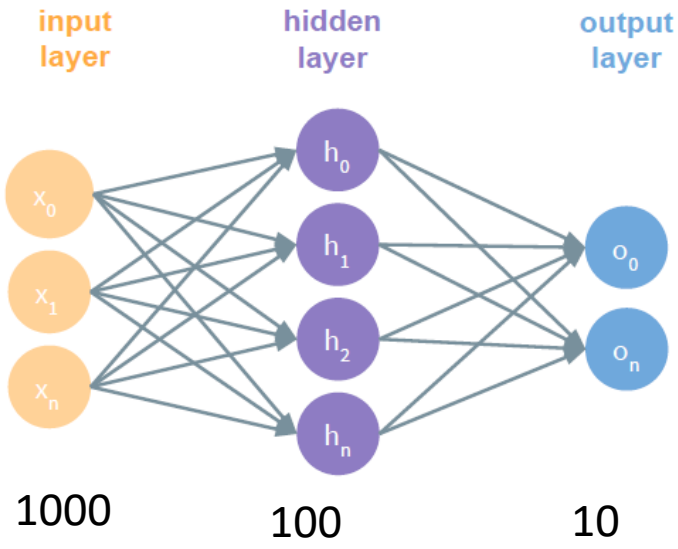
Pytorch:nn

Create random tensors for data

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
    y_pred = model(x)           #forward pass (predict)
    loss = loss_fn(y_pred, y)   #calculate loss

    optimizer.zero_grad()       #clear accumulated gradients
    loss.backward()             #Backward pass (back propagation)
    optimizer.step()            #update weights
```

Pytorch:nn

1. 網路結構




```python
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
    y_pred = model(x)           #forward pass (predict)
    loss = loss_fn(y_pred, y)   #calculate loss

    optimizer.zero_grad()       #clear accumulated gradients
    loss.backward()             #Backward pass (back propagation)
    optimizer.step()            #update weights
```

Pytorch:nn

2. 決定模型的 loss function
3. 訓練相關設定參數 (選擇 optimizer)

5. 開始訓練囉！
Forward pass: feed data to model, and compute loss

```python
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
    y_pred = model(x)           #forward pass (predict)
    loss = loss_fn(y_pred, y)   #calculate loss

    optimizer.zero_grad()       #clear accumulated gradients
    loss.backward()             #Backward pass (back propagation)
    optimizer.step()            #update weights
```

Pytorch:nn

```python
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
    y_pred = model(x)           #forward pass (predict)
    loss = loss_fn(y_pred, y)   #calculate loss

    optimizer.zero_grad()       #clear accumulated gradients
    loss.backward()             #Backward pass (back propagation)
    optimizer.step()            #update weights
```

5. 開始訓練囉！

Backward pass: compute gradient with respect to all model weights (they have requires_grad=True)

Pytorch:nn

```python
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(reduction='sum')

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
    y_pred = model(x)           #forward pass (predict)
    loss = loss_fn(y_pred, y)   #calculate loss

    optimizer.zero_grad()       #clear accumulated gradients
    loss.backward()             #Backward pass (back propagation)
    optimizer.step()            #update weights
```

Use an optimizer for different update rules

After computing gradients, use optimizer to update params and zero gradients

Pytorch 2.0: offer compiled version

```
import torch
import torchvision.models as models

model = models.resnet18().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
compiled_model = torch.compile(model)

x = torch.randn(16, 3, 224, 224).cuda()
optimizer.zero_grad()
out = compiled_model(x)
out.sum().backward()
optimizer.step()
```