深度學習Deep Learning (5)

112-1

朱學亭老師

課程大綱

- W1-課程介紹/Introduction
- W2-Python/Colab and TensorFlow
- W3-Numpy/Pandas and PyTorch
- W4-Sklearn and 機器學習
- W5-Off
- W6-神經網路, TensorFlow, PyTorch
- W7-自動光學檢查(AOI)-1
- W8-自動光學檢查(AOI)-2
- W9-Midterm presentation
- W10-RNN
- W11-GAN
- W12-Yolo
- W13-NLP1-Word2Vec
- W14-NLP2-Seq2Seq,Atention
- W15-NLP3-Transformer,BERT
- W16-AICUP 1
- W17-AICUP 2
- W18-Final presentation

期中考的題庫

https://reurl.cc/6Qj0yV

期中專題

https://aidea-web.tw/topic/285ef3be-44eb-43dd-85cc-f0388bf85ea4

Kissipo

Kissipo = KISS principle + IPO model

KISS principle

https://en.wikipedia.org/wiki/KISS_principle

IPO model

https://en.wikipedia.org/wiki/IPO_model

Deep Learning Programming framework

1. Python: https://www.python.org/
2. NumPy: https://numpy.org/
3. pandas: https://pandas.pydata.org/
4. scikit-learn: https://scikit-learn.org/
5. TensorFlow: https://www.tensorflow.org/
6. Keras: https://keras.io/
7. PyTorch: https://pytorch.org/

啟思博學習深度學習（2023夏季班）

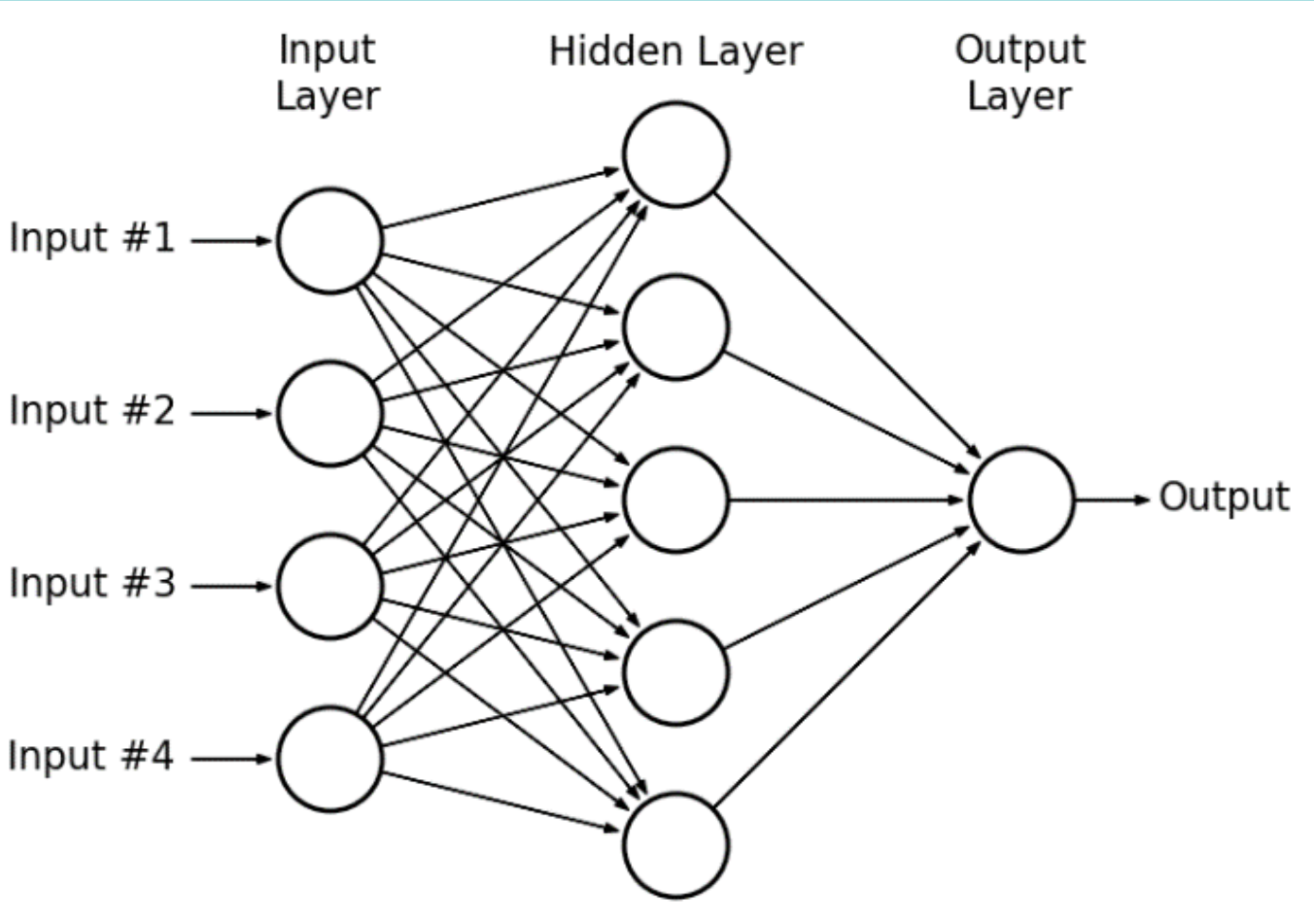
完成線上課程，
可以加學期成績5分

AICUP得獎
也可以加學期成績(直接90起跳)

https://reurl.cc/m0R3oG

(1) 神經網路模型概念-REVIEW

神經網路模型


Input Layer
Hidden Layer
Output Layer
Input #1
Input #2
Input #3
Input #4
Output

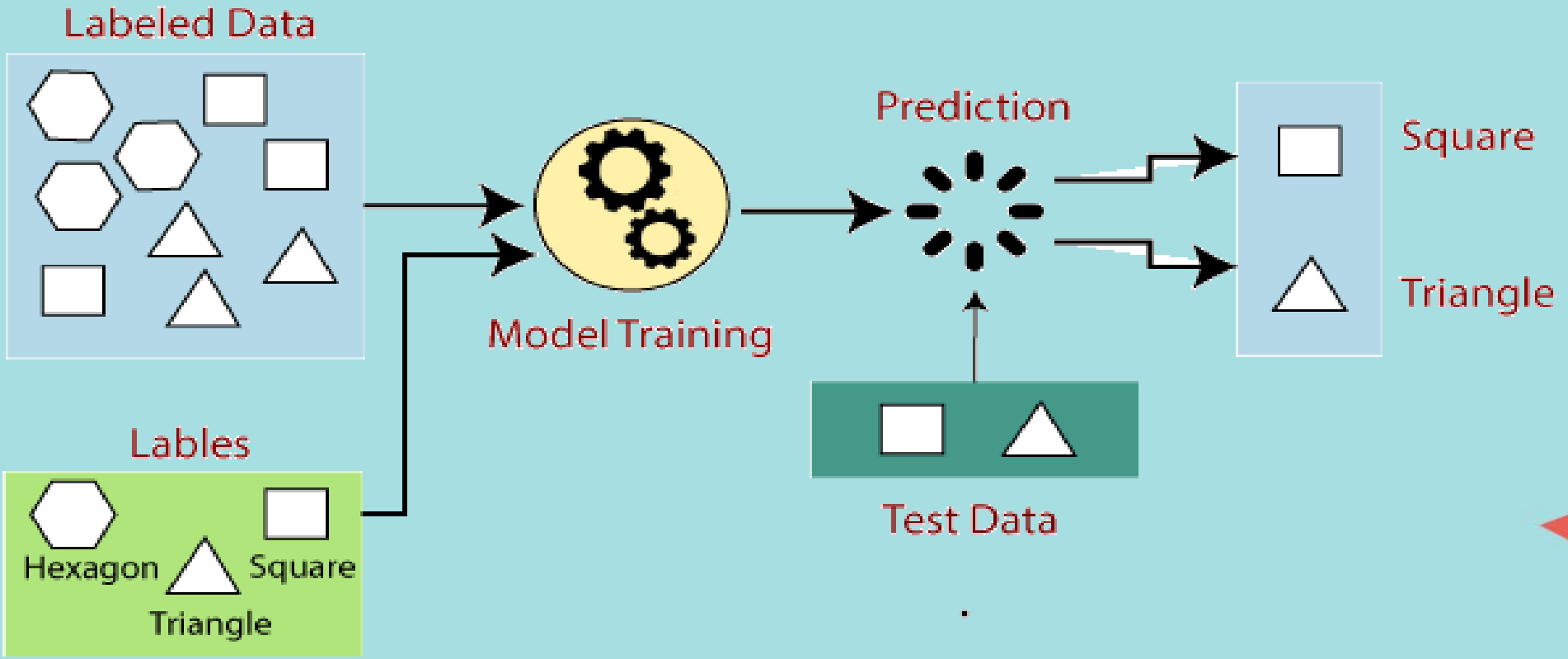
問題構建 (Framing)：機器學習主要術語

- 什麼是（監督式）機器學習？簡單來說，它的定義如下：
  - 機器學習系統通過學習如何組合輸入資訊來對從未見過的資料做出有用的預測。
- 機器學習的基本術語
  - **標籤 (Labels)**
  - **特徵 (Features)**
  - **樣本 (Examples)**
  - **模型 (Models)**
  - 回歸與分類 (Regression vs. classification)

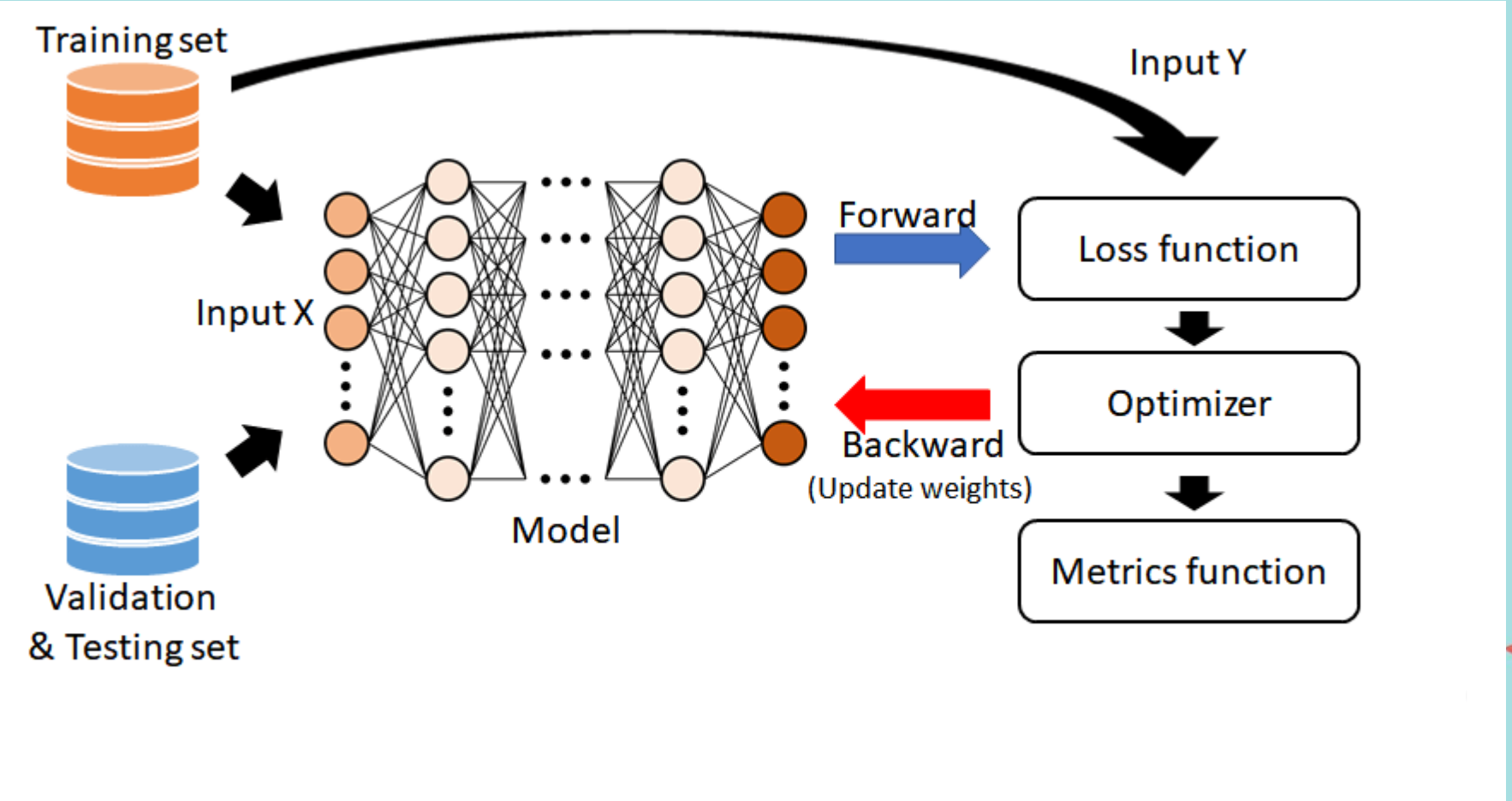
回歸與分類 (Regression vs. classification)

- A **regression** model (回歸) predicts continuous values. For example, regression models make predictions that answer questions like the following:
  - What is the value of a house in California?
  - What is the probability that a user will click on this ad?
- A **classification** model (分類) predicts discrete values. For example, classification models make predictions that answer questions like the following:
  - Is a given email message spam or not spam?
  - Is this an image of a dog, a cat, or a hamster?

Features and Labels
特徵和標籤



Deep learning 流程

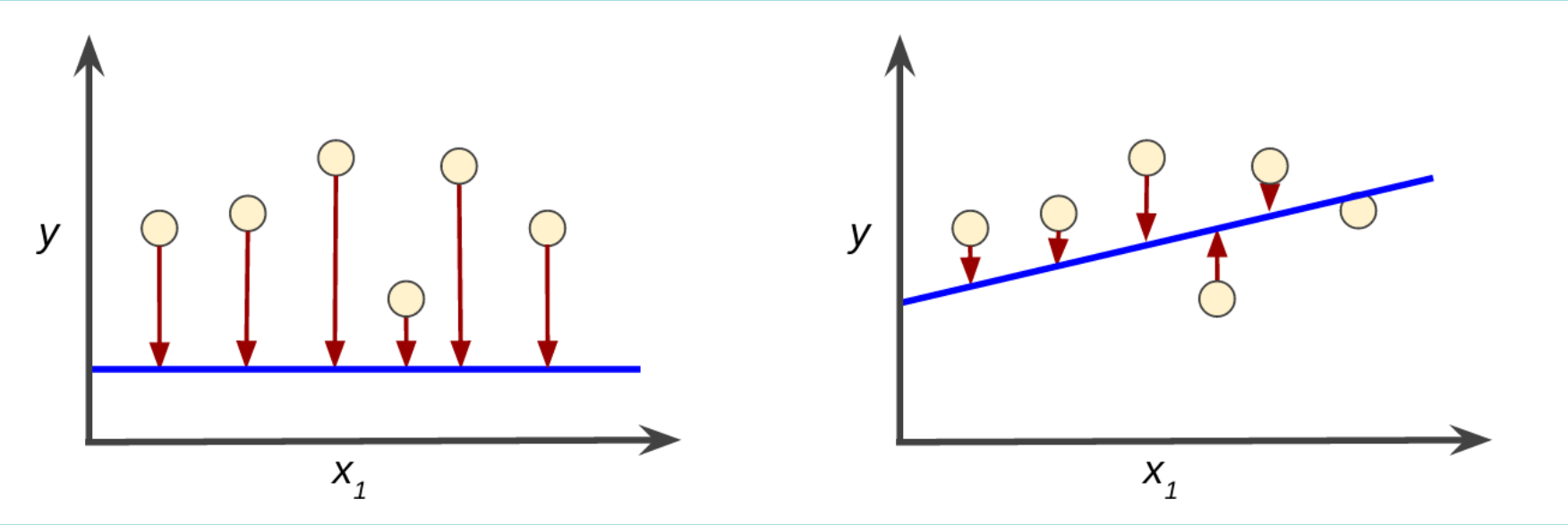


Types of Artificial Neural Networks

- Feed Forward Neural Networks
- Convolutional neural network (CNN)
- Recurrent neural network (RNN)
- Sequence-to-sequence model(encoder-decoder)

Training and Loss

- Training a model means learning (determining) good values for all the weights and the bias from labeled examples.



誤差值Error

- 均方誤差(Mean square error，MSE) $MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$
- 平均絕對值誤差(Mean absolute error，MAE)，$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|$
- 交叉熵(cross-entropy)

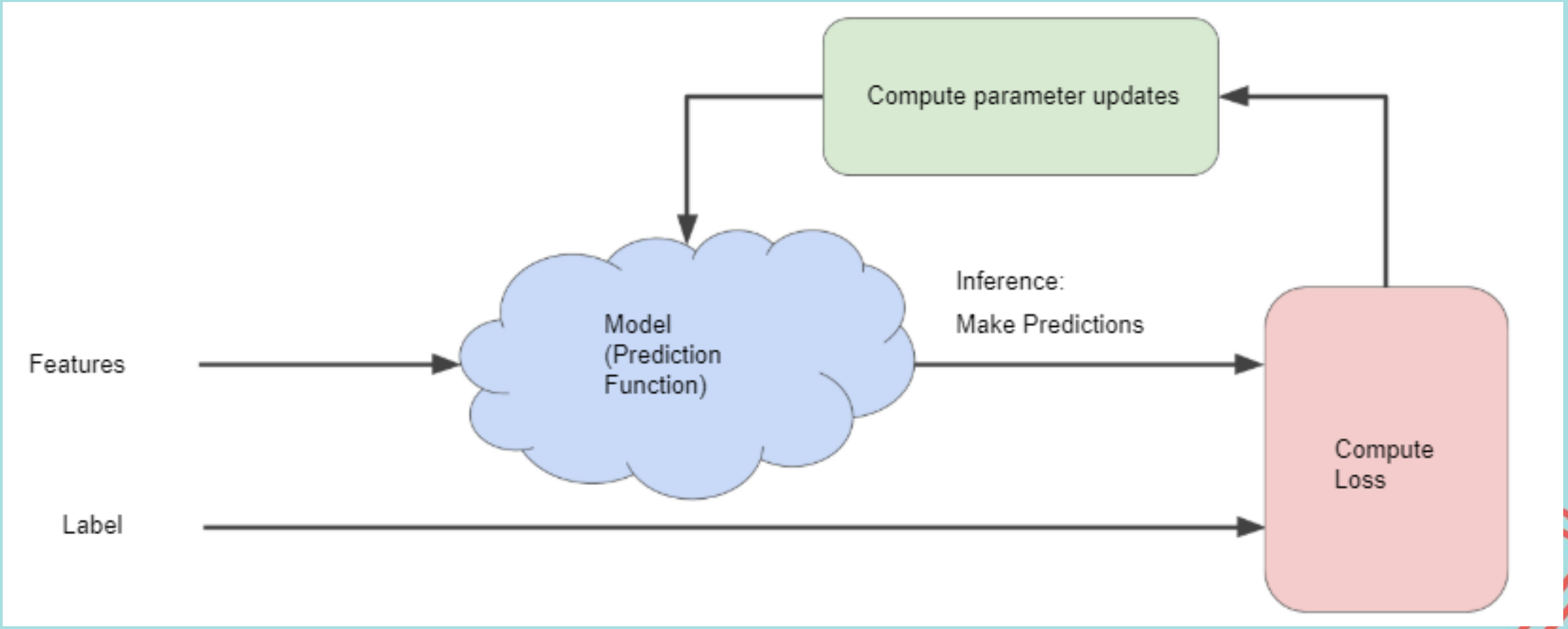
- 損失函數(loss function)=**實際值-預測值**
- 熵函數 $H(X) = \sum_{i} -p_i log_2(p_i)$

如何根據誤差來調整參數

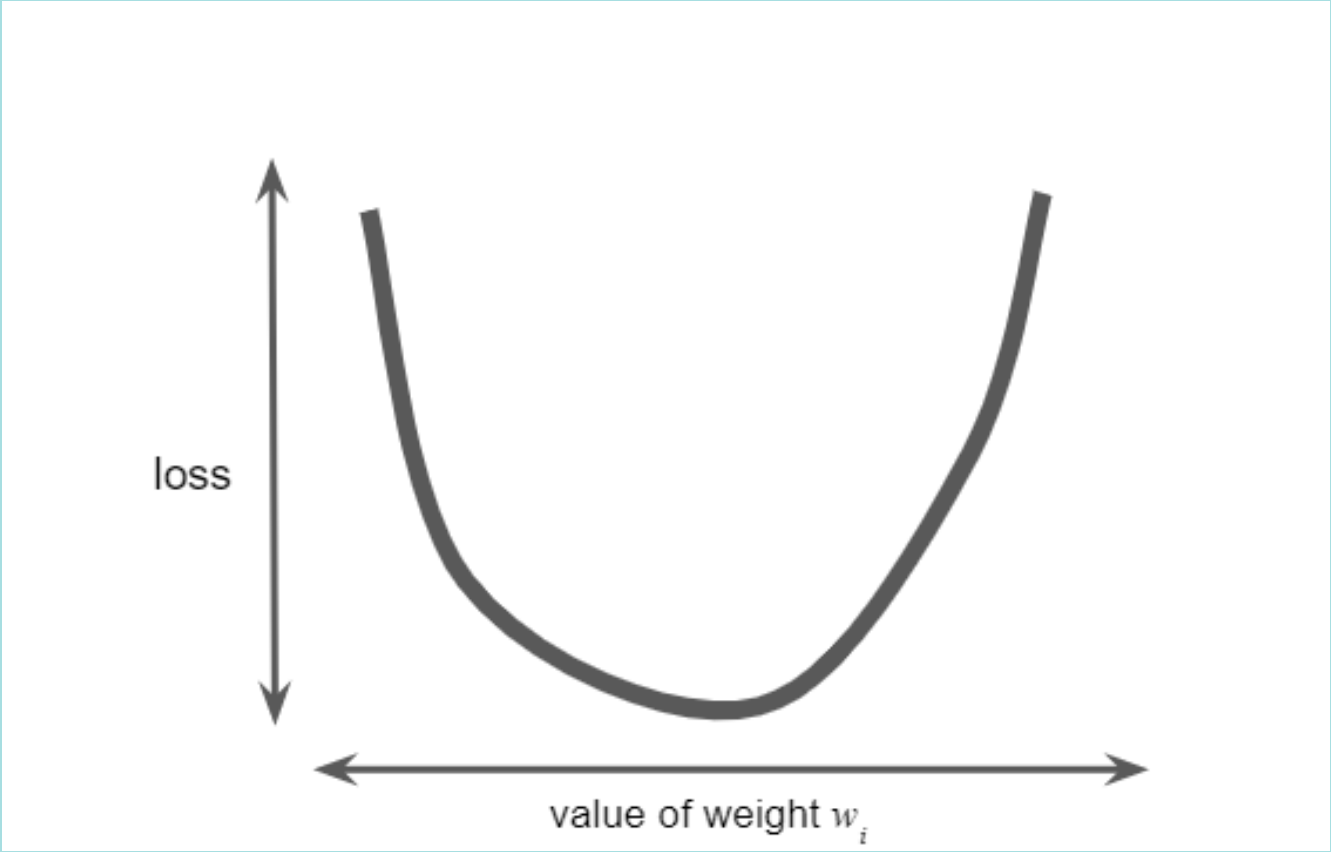
- 反向傳播（Backpropagation)
- 最優化算法

- 梯度下降法(Gradient descent)

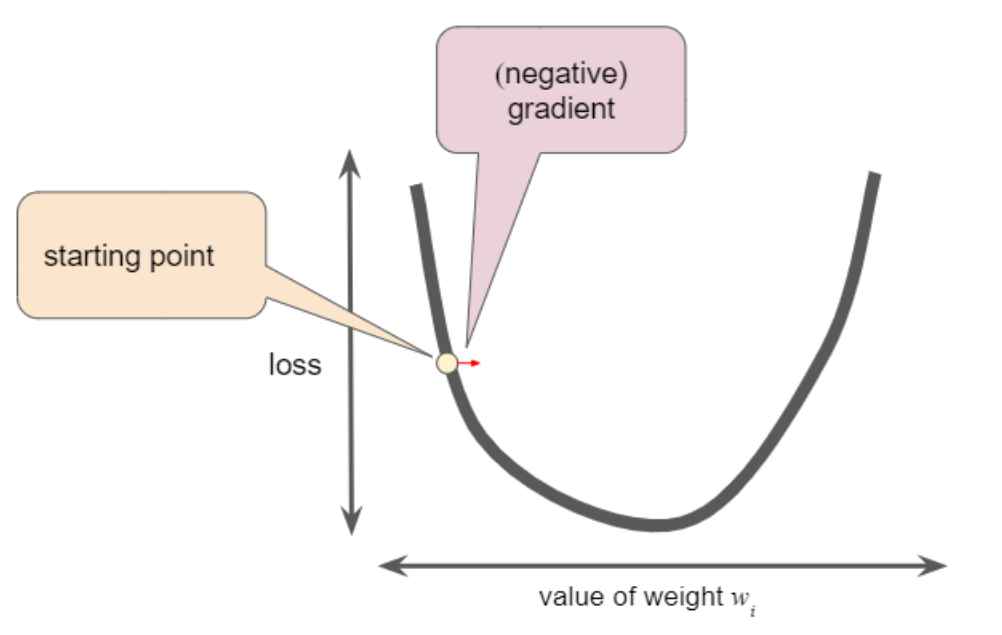
Reducing Loss: An Iterative Approach

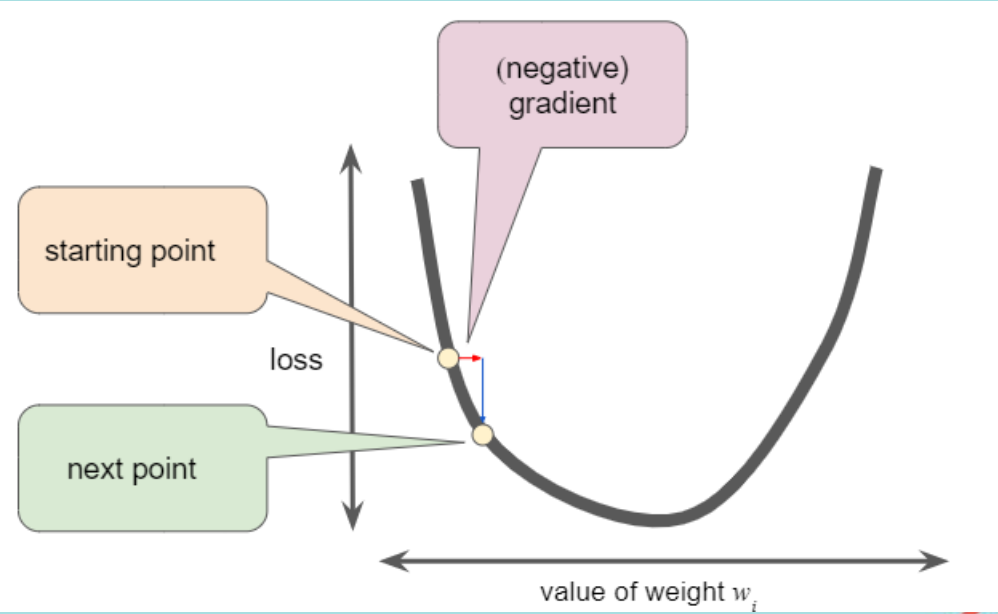


Reducing Loss: Gradient Descent (梯度下降)

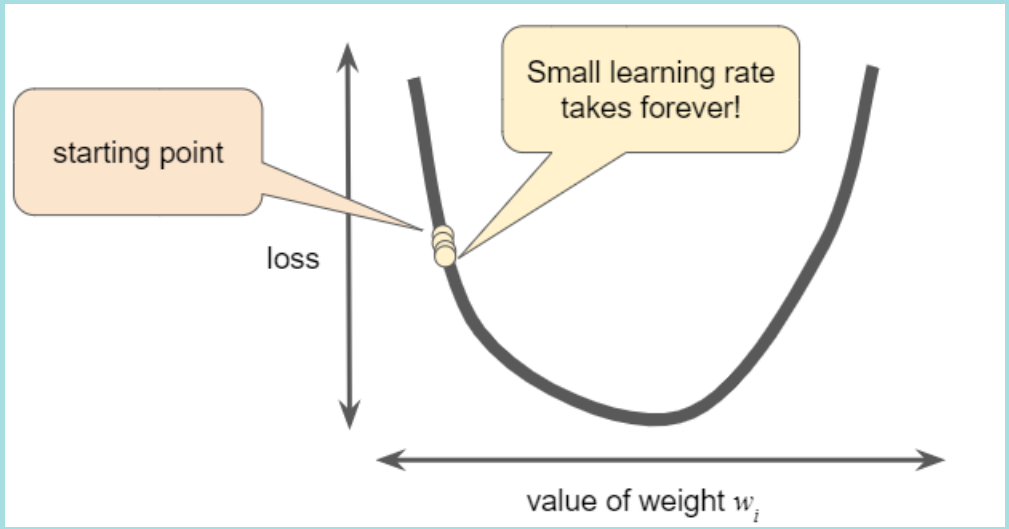


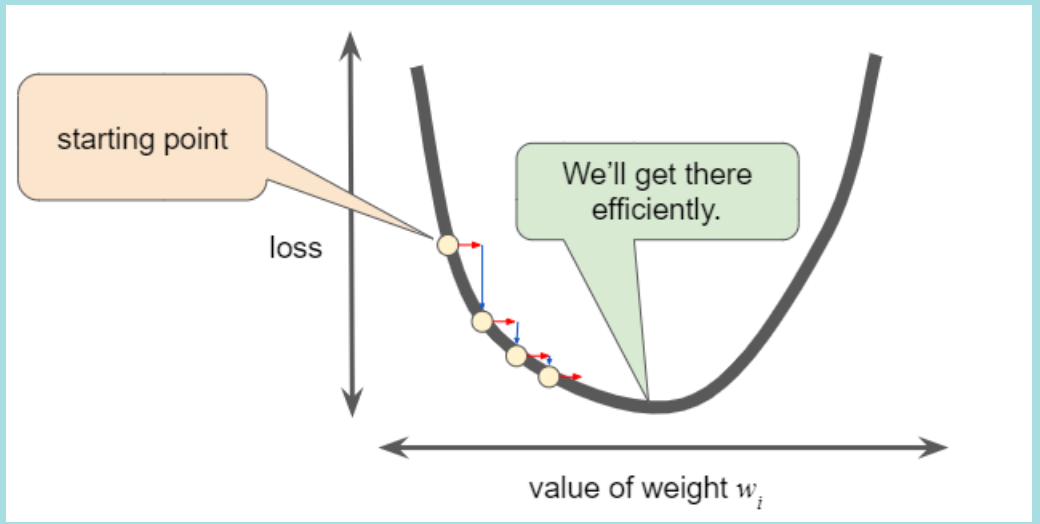
Gradient descent relies on negative gradients

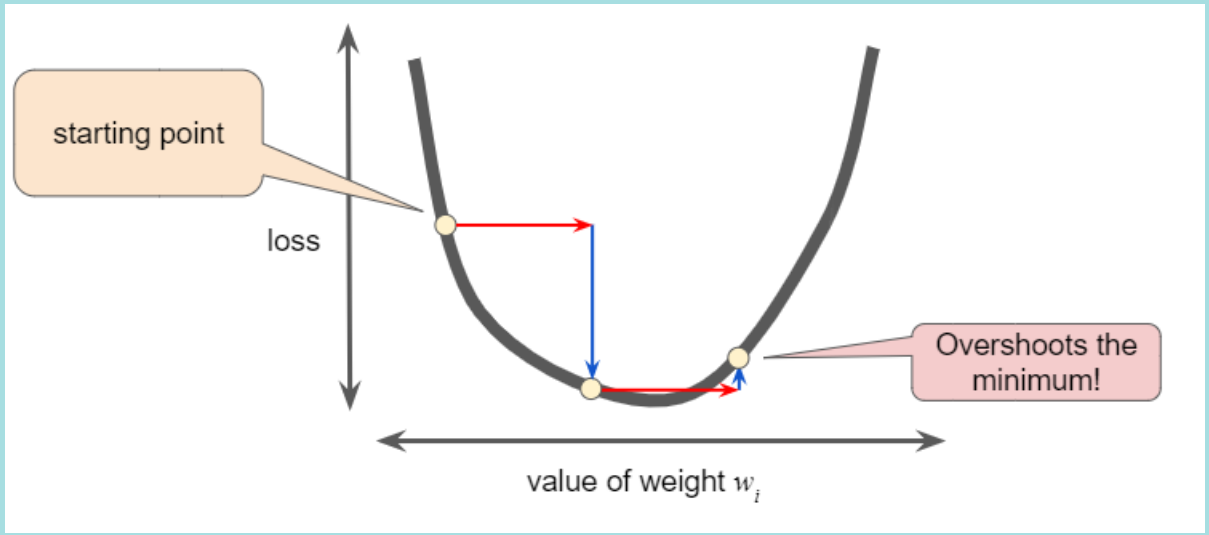




Reducing Loss: Learning Rate(學習率)







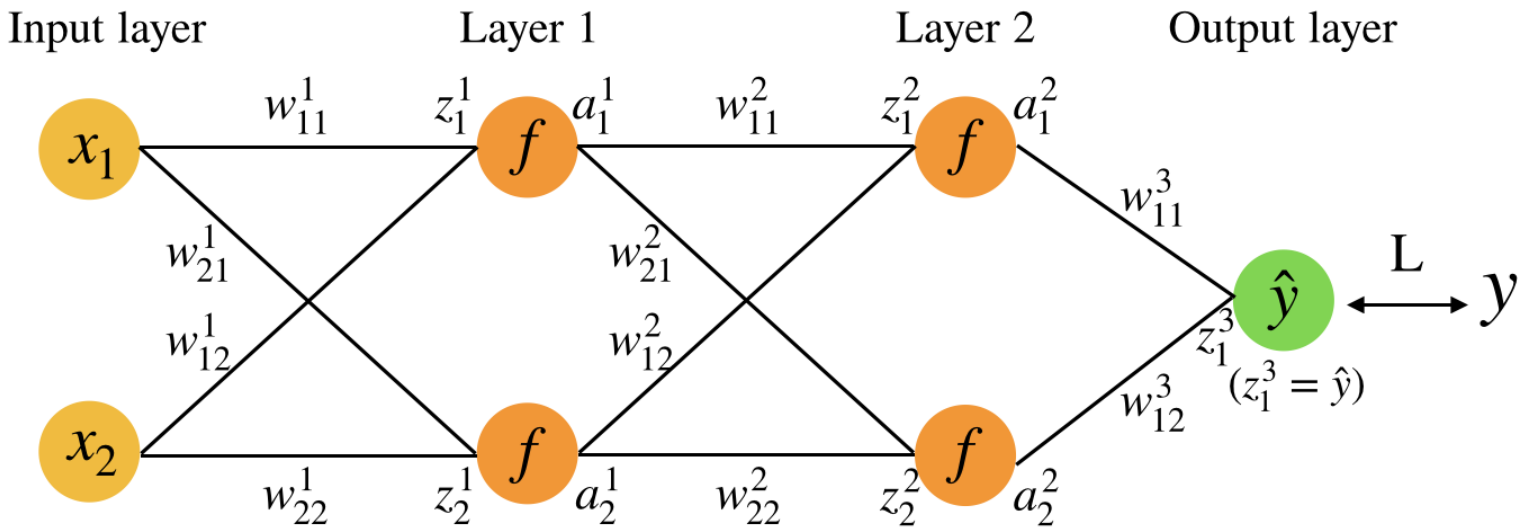
優化器(Optimizer)

- *SGD-準確率梯度下降法 (stochastic gradient decent)*
- *Momentum*
- *AdaGrad-* Adaptive *gradient*
- *Adam= AdaGrad+ Momentum*
- RMSProp (Root Mean Square Prop)

反向傳播計算

- 反向傳播算法的兩個階段：激勵傳播與權重更新。
- 激勵傳播
  - （前向傳播階段）將訓練輸入送入網絡以獲得激勵響應；
  - （反向傳播階段）將激勵響應同訓練輸入對應的目標輸出求差，從而獲得輸出層和隱藏層的響應誤差。
- 權重更新
  - 將輸入激勵和響應誤差相乘，從而獲得權重的梯度；
  - 將這個梯度乘上一個比例並取反後加到權重上。

Backpropagation(**BP**)-Notation



w_{11}^1 ← Layer 1 ← Layer 1 Neuron 1 to Input layer Neuron 1

$z_1^1 = x_1 w_{11}^1 + x_2 w_{12}^1$, z_1 : activation function input

$a_1^1 = f(z_1^1)$, f : activation function, a_1^1 : activation output

$\hat{y}$: predict value

Backpropagation(**BP**)-Operation

- **Chain rule:**
  - $y = f(x), z = g(y),\ \ z = g(f(x)) = (g \circ f)(x)$

$$(g \circ f)'(x) = \frac{dg}{dx} = \frac{dg}{df}\frac{df}{dx}$$

  - $z = f(x, y)$, where $x = g(t), y = h(t)$

$$\frac{df}{dt} = \frac{\partial f}{\partial g}\frac{dg}{dt} + \frac{\partial f}{\partial h}\frac{dh}{dt}$$

- **Loss function:**

$$L = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y} - y)^2$$

$$L' = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)$$

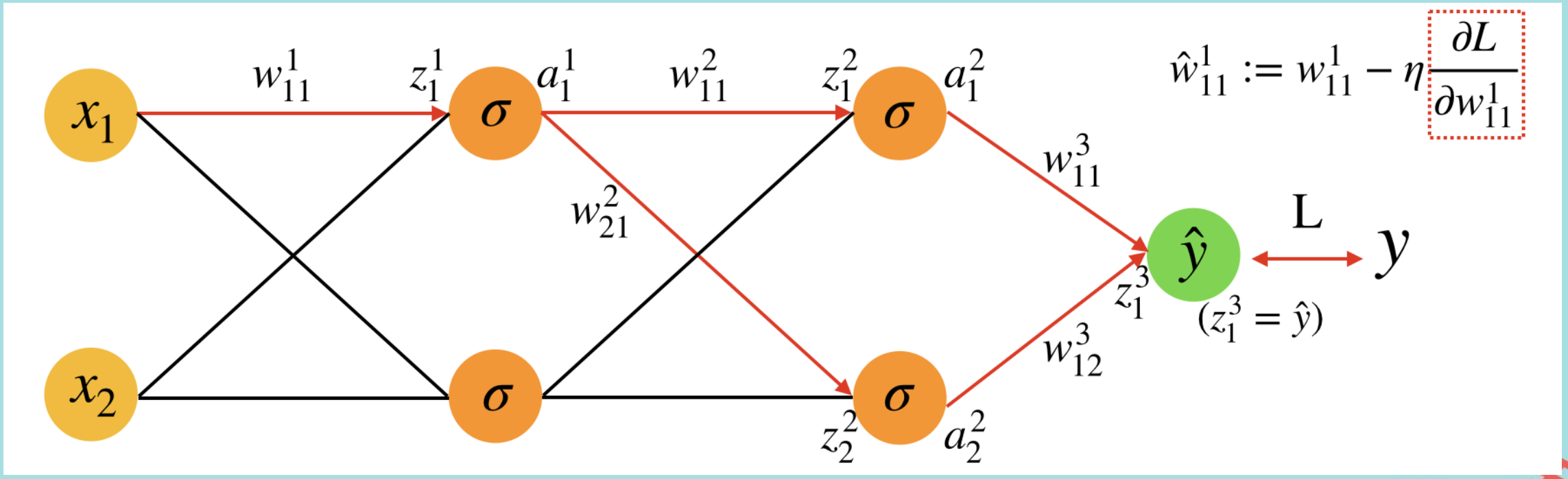
- **Sigmoid function:**

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

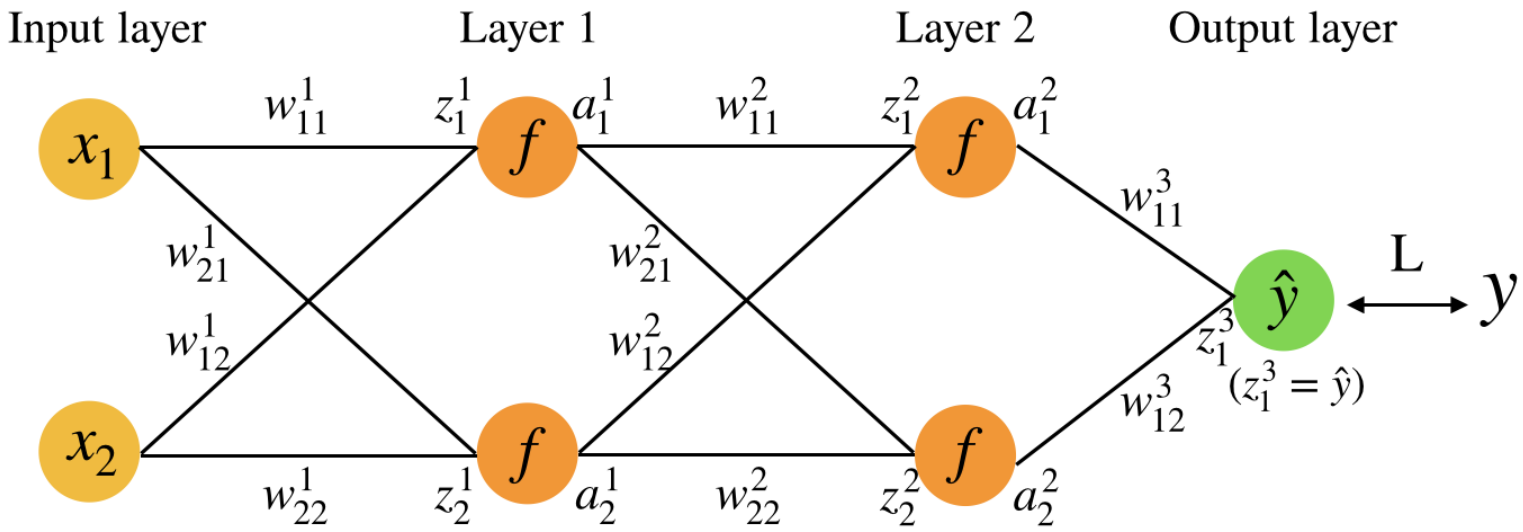
- **Derivative of sigmoid function:**

$$\begin{aligned}\sigma(x)' &= \frac{d}{dx}(1 + e^{-x})^{-1} \\ &= -(1 + e^{-x})^{-2}(-e^{-x}) \\ &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= \frac{1}{1 + e^{-x}}\frac{e^{-x}}{1 + e^{-x}} \\ &= \sigma(x)(1 - \sigma(x))\end{aligned}$$

Backpropagation(BP)-w^1_{11}的梯度下降



Backpropagation(**BP**)



w_{11}^1 ← Layer 1 ← Layer 1 Neuron 1 to Input layer Neuron 1

$z_1^1 = x_1 w_{11}^1 + x_2 w_{12}^1$, z_1 : activation function input

$a_1^1 = f(z_1^1)$, f : activation function, a_1^1 : activation output

$\hat{y}$: predict value

Backpropagation(**BP**)

- The gradient of w_{11}^1 :

$$\frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial z_1^3}\left[\sum_{i=1}^{2} \frac{\partial z_1^3}{\partial a_i^2}\frac{\partial a_i^2}{\partial z_i^2}\frac{\partial z_i^2}{\partial a_1^1}\right]\frac{\partial a_1^1}{\partial z_1^1}\frac{\partial z_1^1}{\partial w_{11}^1} = (\hat{y}-y)\left[\sum_{i}^{2} w_{1i}^3 \sigma'(z_i^2) w_{i1}^2\right]\sigma'(z_1^1)x_1$$

1. **2.** **3.**

1. $$\frac{\partial L}{\partial z_1^3} = \frac{\partial}{\partial \hat{y}}\frac{1}{2}(\hat{y}-y)^2 = (\hat{y}-y),\ (z_1^3 = \hat{y})$$

2. $$\frac{\partial z_1^3}{\partial a_1^2} = \frac{\partial}{\partial a_1^2}(a_1^2 w_{11}^3 + a_2^2 w_{12}^3) = w_{11}^3,\ (z_1^3 = a_1^2 w_{11}^3 + a_2^2 w_{12}^3)$$

3. $$\frac{\partial a_1^2}{\partial z_1^2} = \sigma'(z_1^2) = \sigma(z_1^2)(1-\sigma(z_1^2)),\ (a_1^2 = \sigma(z_1^2))$$

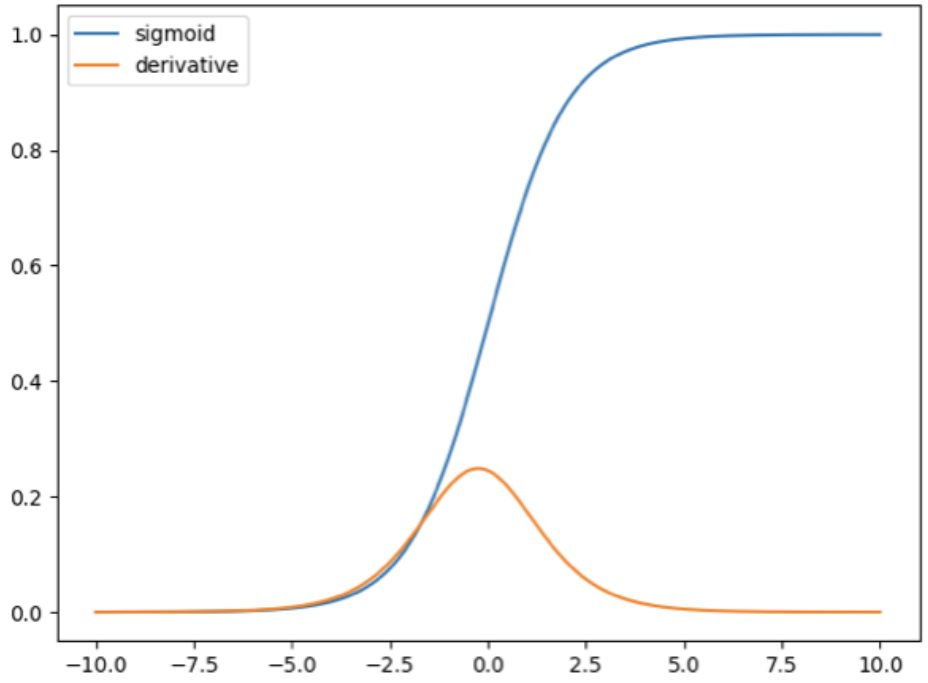
Gradient Problem

- **Gradient vanishing (exploding)**

$$\frac{\partial L}{\partial w_{11}^{1}} = (\hat{y} - y)[\sum_{i}^{2} w_{1i}^{3}\sigma'(z_i^2)w_{i1}^{2}]\sigma'(z_1^1)x_1$$

1. 2. 3.

1. $(\hat{y} - y)$: Defined by your loss function .
2. w : Defined by your initialization weight .
3. $\sigma'(z)$: Defined by your activation function .



Stochastic Gradient Descent(隨機梯度下降)

- Different learning rates are used at different learning time points, and of course different directions are considered.
- 在不同學習的時間點用不同的學習率，當然還有考慮不同的方向。

優化器(Optimizer)

- Update weights by gradient

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

- W為權重(weight)參數，L為損失函數(loss function)，η是學習率(learning rate)，$\partial L/\partial W$是損失函數對參數的梯度(微分)

AdaGrad

- Adaptive Gradient

$$W \leftarrow W - \eta \frac{1}{\sqrt{n+\epsilon}} \frac{\partial L}{\partial W}$$

$$n = \sum_{r=1}^{t} (\frac{\partial L_r}{\partial W_r})^2$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{\sum_{r=1}^{t} (\frac{\partial L_r}{\partial W_r})^2 + \epsilon}} \frac{\partial L}{\partial W}$$

- 在AdaGrad Optimizer 中，η 乘上 $1/\sqrt{(n+\epsilon)}$ 再做參數更新，出現了一個n的參數，n為前面所有梯度值的平方和，利用前面學習的梯度值**平方和**來調整learning rate ，ϵ 為平滑值，加上 ϵ 的原因是為了不讓分母為0，ϵ 一般值為1e-8

Adam

- Adam Optimizer 把Momentum 跟 AdaGrad這二種Optimizer結合

$$W \leftarrow W - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L_t}{\partial W_t}$$

$$v_t = \beta_1 v_{t-1} + (1 - \beta_2) (\frac{\partial L_t}{\partial W_t})^2$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

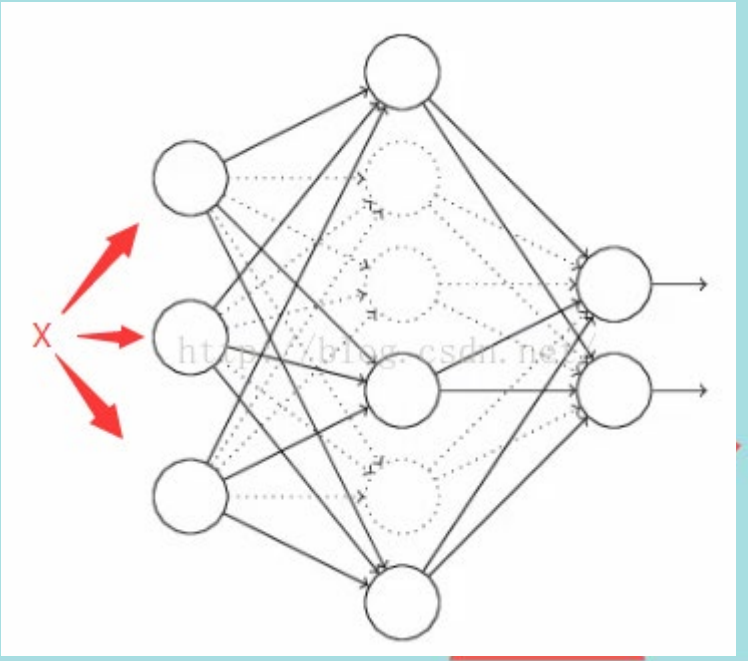
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Optimizer比較

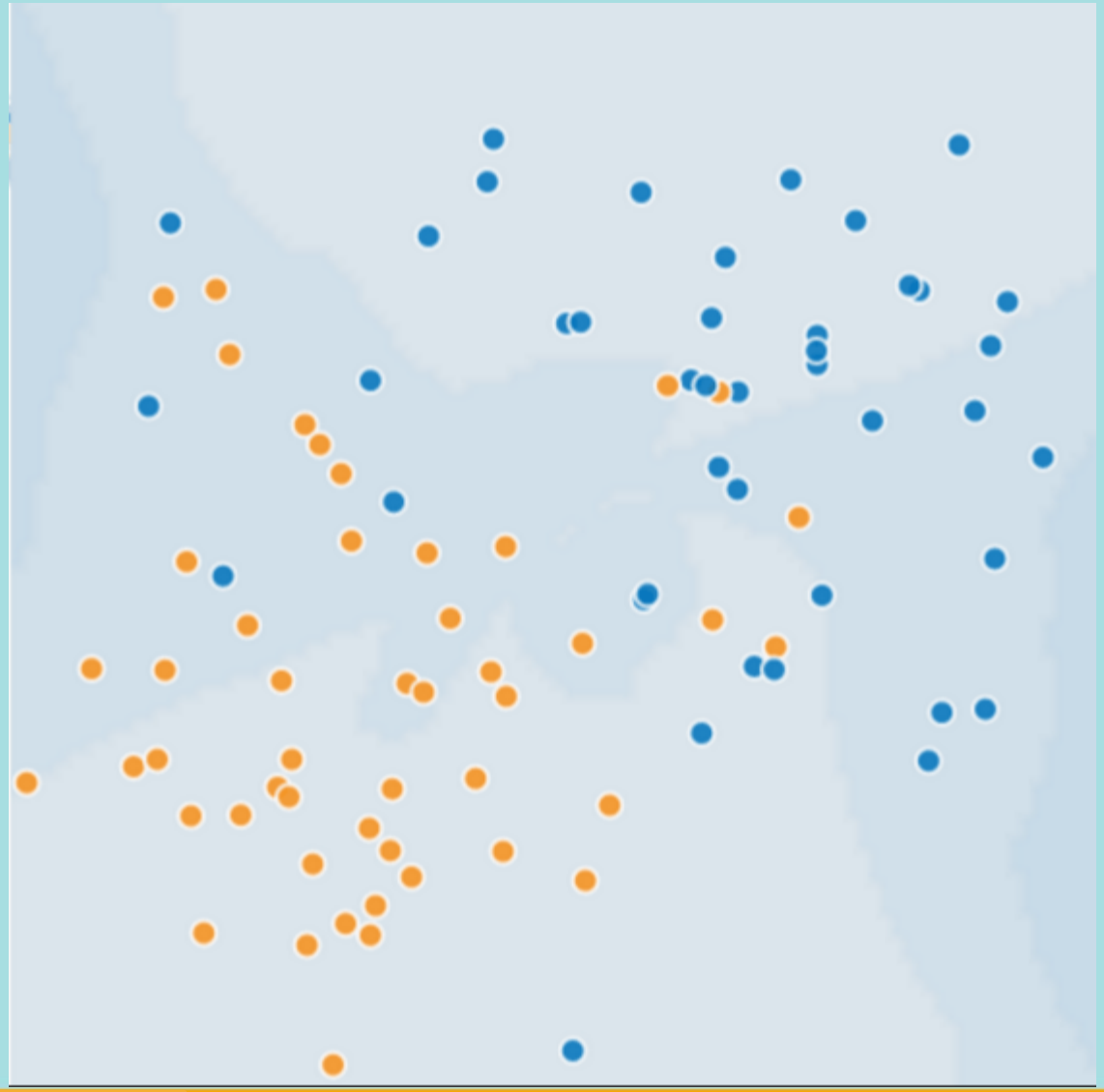
Optimizer	特點
SGD	• 有機會跳出目前局部收斂進而進到另一個局部收斂而得到最小值，而得到全局最小值 • 需自行設定learning rate，較難選擇到合適的learning rate • 會造成loss function有嚴重的震蕩 • 需要較長時間收斂至最小值
Momentum	• 能夠在相關方向加速SGD，抑制SGD的嚴重震蕩，進而加快收斂 • 需自行設定learning rate與β，有可能會使參數的移動方向偏移梯度下分的方向，進而導至沒有那麼快速的收斂
AdaGrad	• 能夠自動調整learning rate，進而調整收斂 • 適合處理稀疏梯度 • 依然需要人工設置一個全局的learning rate • 後期，分母梯度平方的累加會越來越大，會使梯度趨近於0，使得訓練結束
Adam	• 結合了AdaGrad與Momentum的優點 • 適用於大數據集和高維空間的資料 • 目前最常使用的一個Optimizer

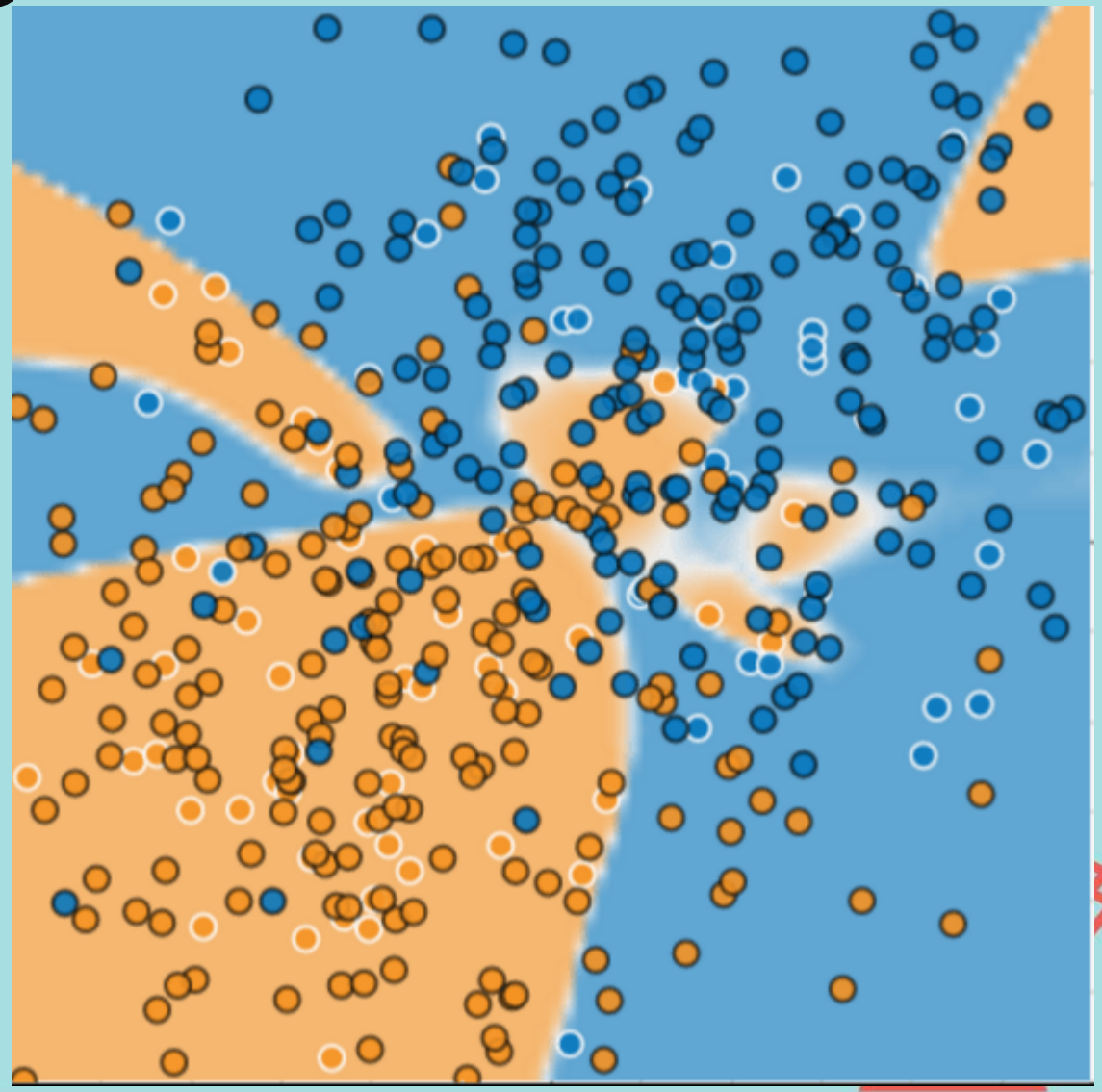
Overfitting和regularization

- Overfitting
- Regularization
- Dropout: 在訓練的時候只算部分的神經元

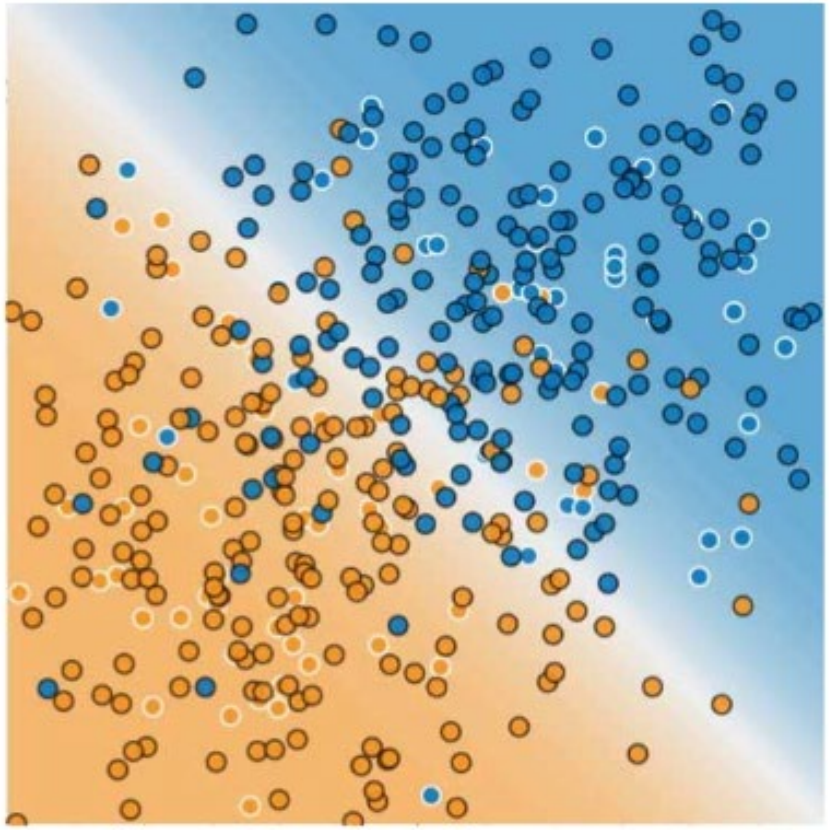


Overfitting

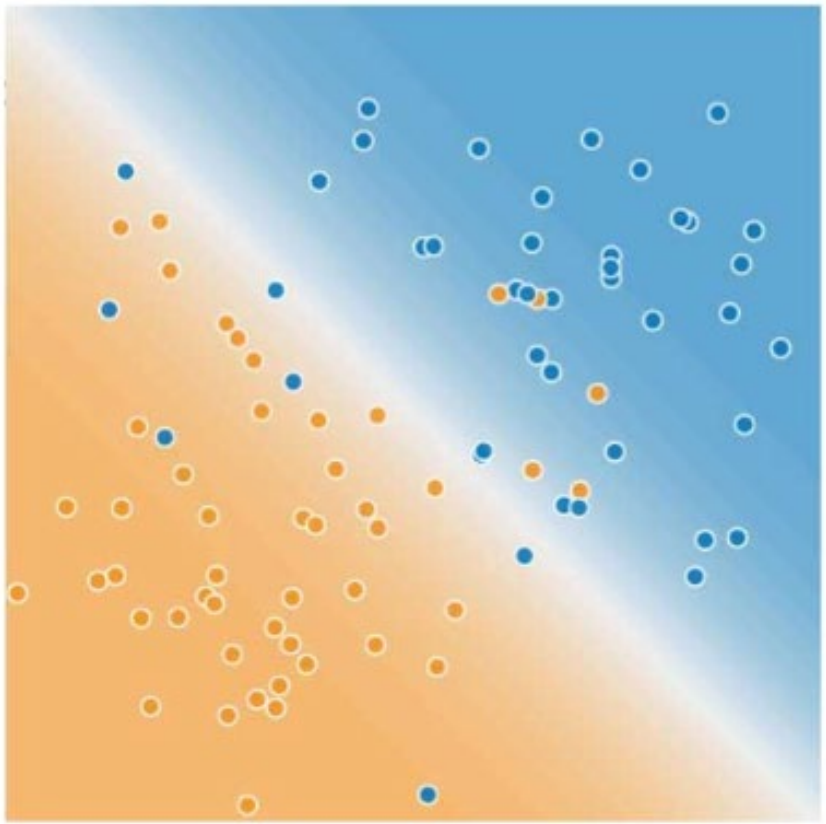




Training and Test Sets

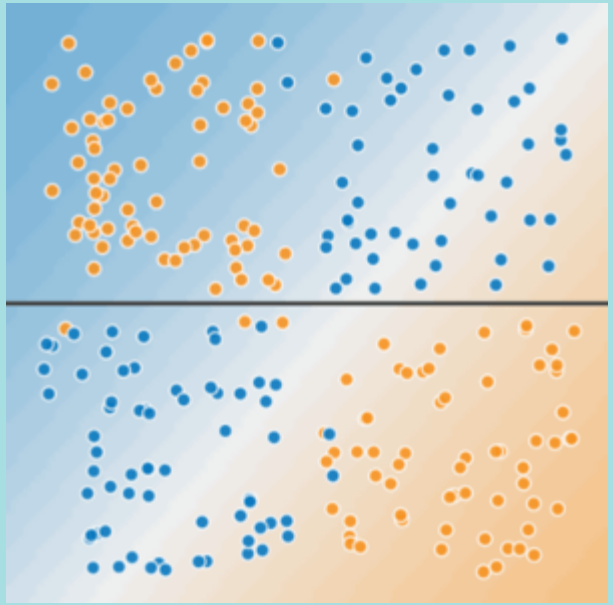


Training Data



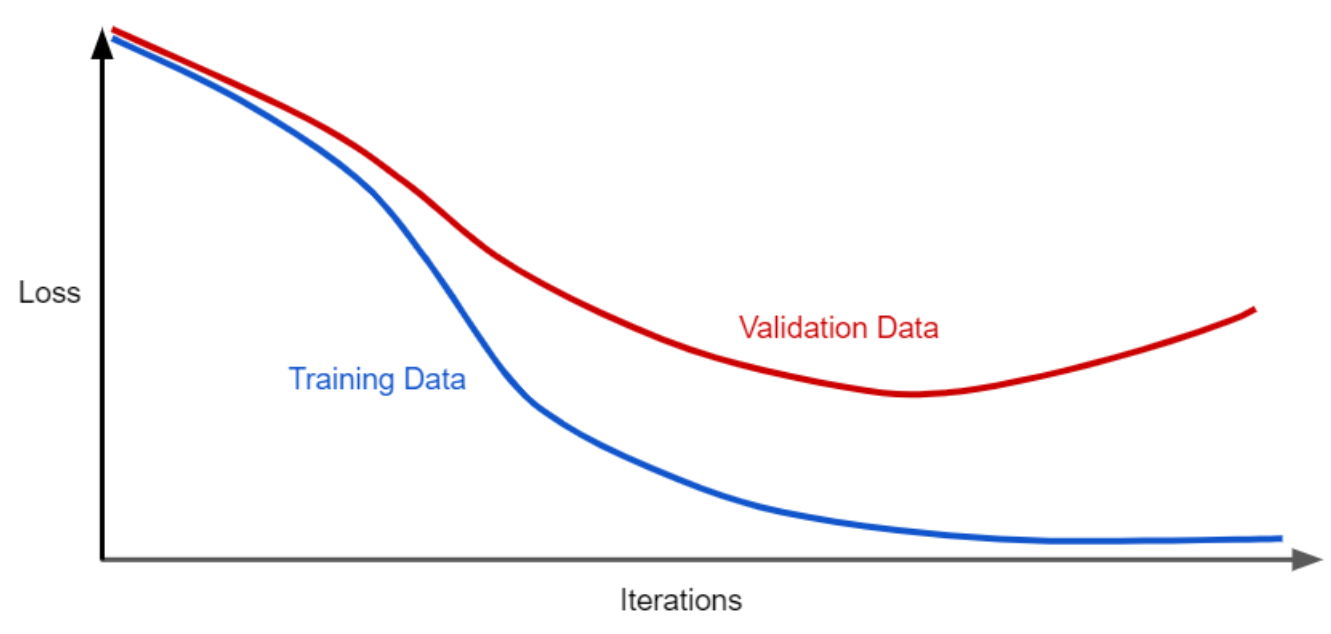
Test Data

Feature crosses (特徵交叉)



- feature cross is a synthetic feature that encodes nonlinearity in the feature space by multiplying two or more input features together. (The term cross comes from cross product.)
- 特徵交叉是一種合成特徵，它通過將兩個或多個輸入特徵相乘在一起來編碼特徵空間中的非線性。（術語“交叉”來自交叉積。）

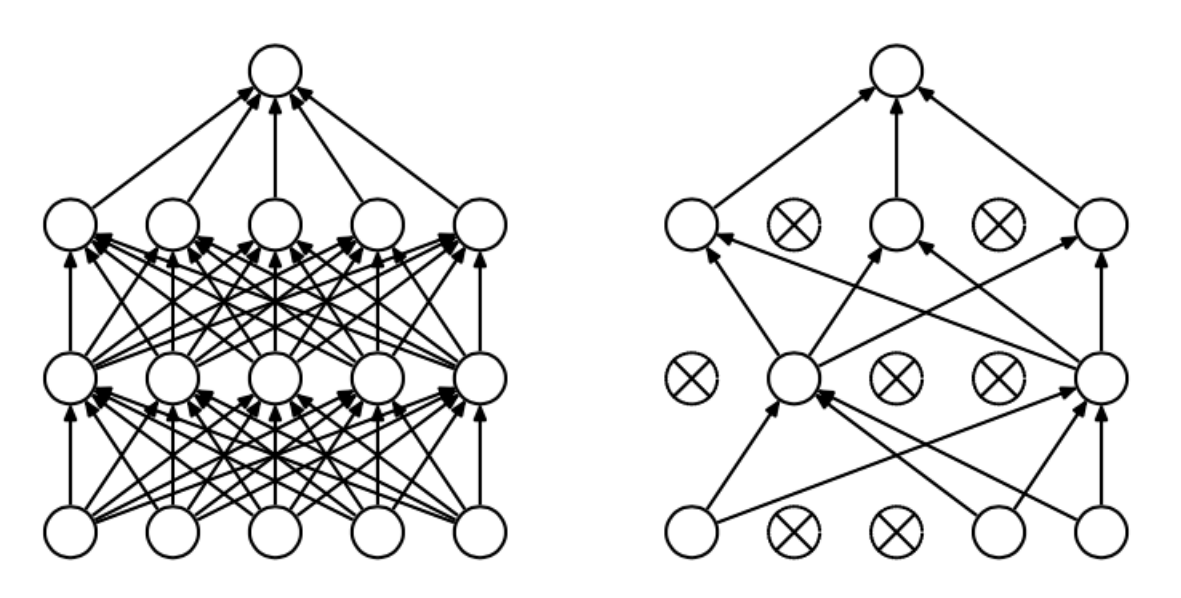
Regularization (正規化)



$$\text{minimize}(\text{Loss}(\text{Data}|\text{Model}) + \text{complexity}(\text{Model}))$$

$$L_2 \text{ regularization term} = ||\boldsymbol{w}||_2^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

Dropout (丢弃)



- A form of regularization useful in training neural networks. Dropout regularization works by removing a random selection of a fixed number of the units in a network layer for a single gradient step.

Classification:
True vs. False and Positive vs. Negative

Confusion matrix

True Positive (TP): • Reality: A wolf threatened. • Shepherd said: "Wolf." • Outcome: Shepherd is a hero.	**False Positive (FP):** • Reality: No wolf threatened. • Shepherd said: "Wolf." • Outcome: Villagers are angry at shepherd for waking them up.
False Negative (FN): • Reality: A wolf threatened. • Shepherd said: "No wolf." • Outcome: The wolf ate all the sheep.	**True Negative (TN):** • Reality: No wolf threatened. • Shepherd said: "No wolf." • Outcome: Everyone is fine.

Accuracy (正確性)

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • **Number of TP results: 1**	**False Positive (FP):** • Reality: Benign • ML model predicted: Malignant • **Number of FP results: 1**
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • **Number of FN results: 8**	**True Negative (TN):** • Reality: Benign • ML model predicted: Benign • **Number of TN results: 90**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

ROC Curve (接收者操作特徵曲線)

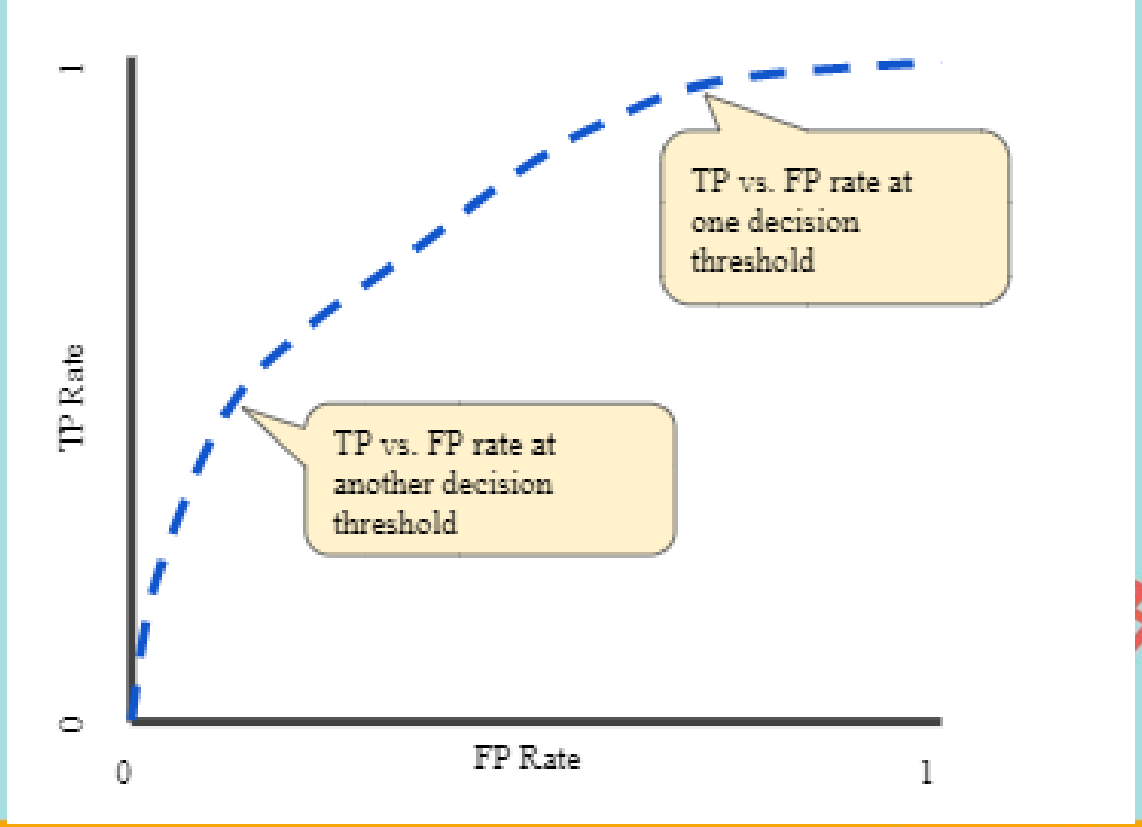
Receiver operating characteristic curve

True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

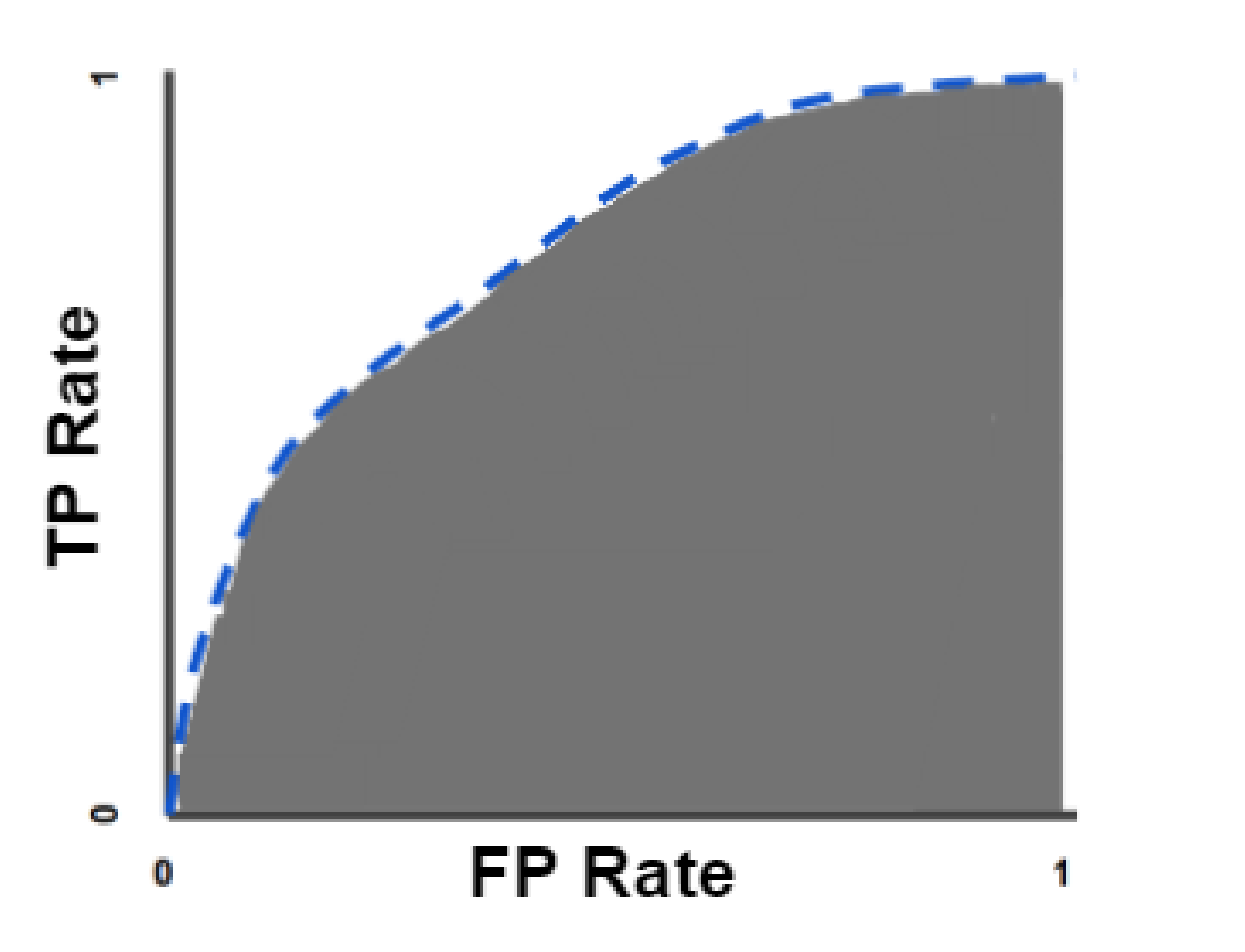
$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (**FPR**) is defined as follows:

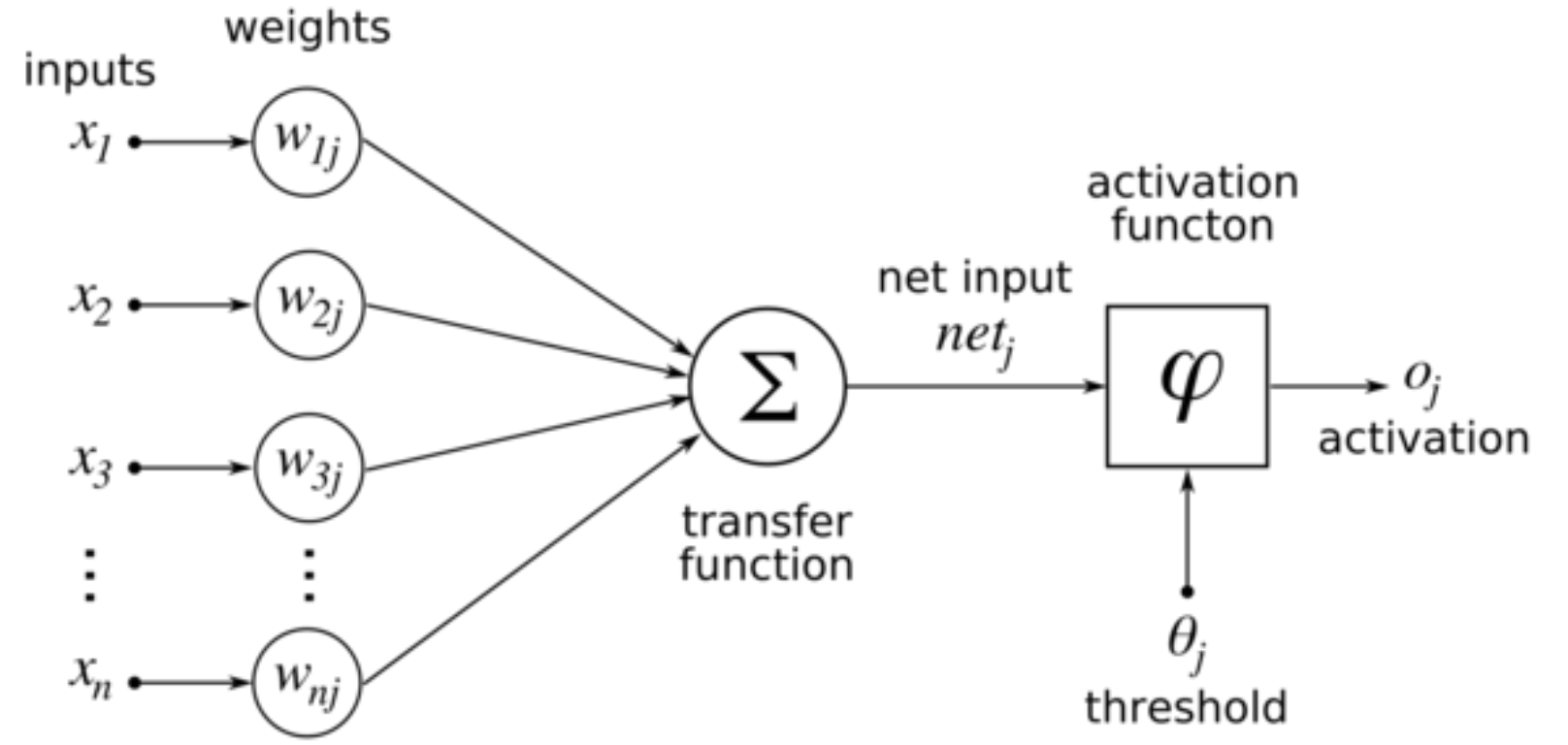
$$FPR = \frac{FP}{FP + TN}$$



AUC: Area Under the ROC Curve



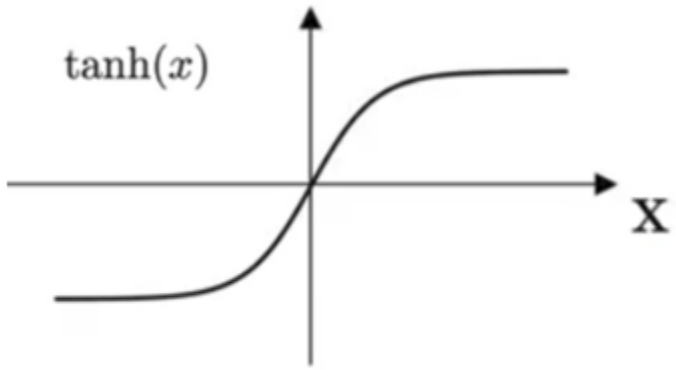
Activation function (激勵函數)



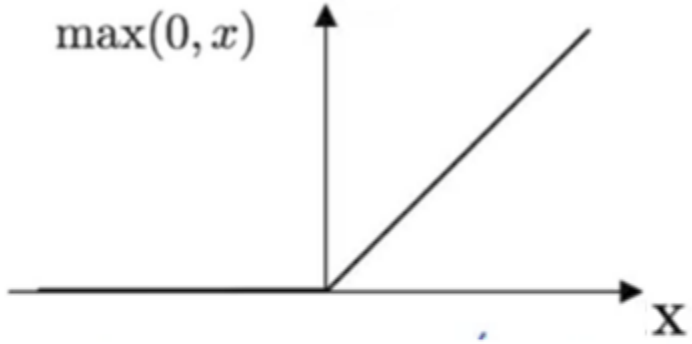
If the activation function is not used in the neural network, then in the neural network, the linear combination of the input of the above layer is the output of this layer. The output and input still cannot be separated from the linear relationship, and the deep neural network is lost. significance.
在類神經網路中如果不使用激勵函數，那麼在類神經網路中皆是以上層輸入的線性組合作為這一層的輸出（也就是矩陣相乘），輸出和輸入依然脫離不了線性關係，做深度類神經網路便失去意義。

Activation Functions

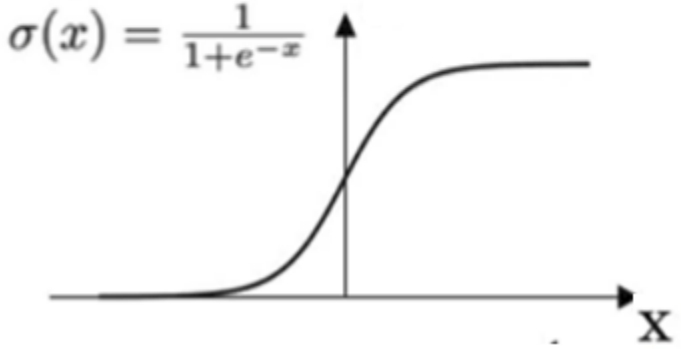
Hyper Tangent Function



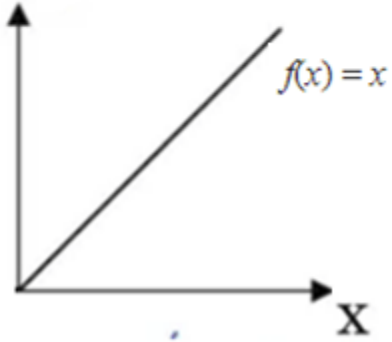
ReLU Function



Sigmoid Function

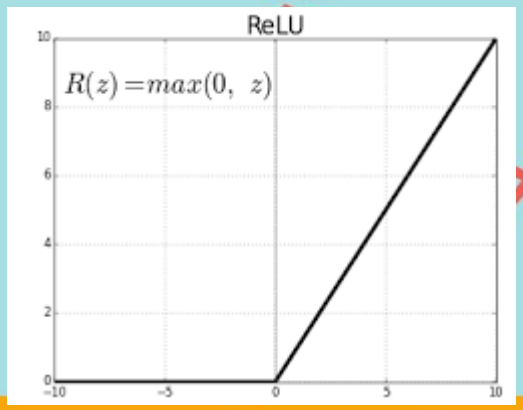


Identity Function



ReLU (Rectified Linear Unit)線性整流函數

- 梯度消失問題 (vanishing gradient problem)
- 類神經網路的稀疏性 (Sparseness of neural networks)
- 全有全無律 (all or none law)

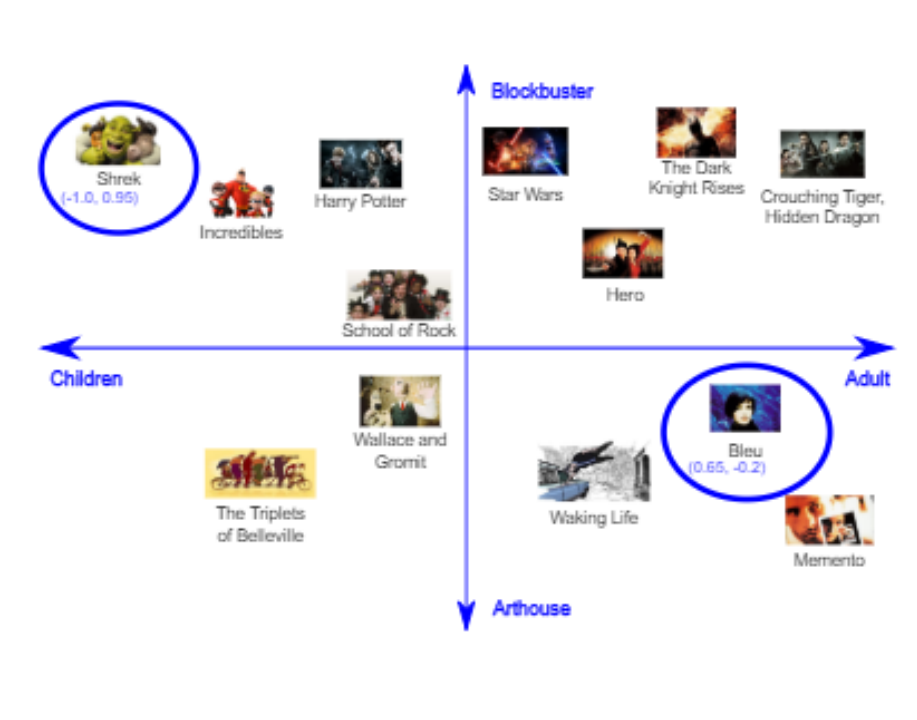


One-Hot-Encoding

ID	Gender
1	Male
2	Female
3	Not Specified
4	Not Specified
5	Female

ID	Male	Female	Not Specified
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	0	1	0

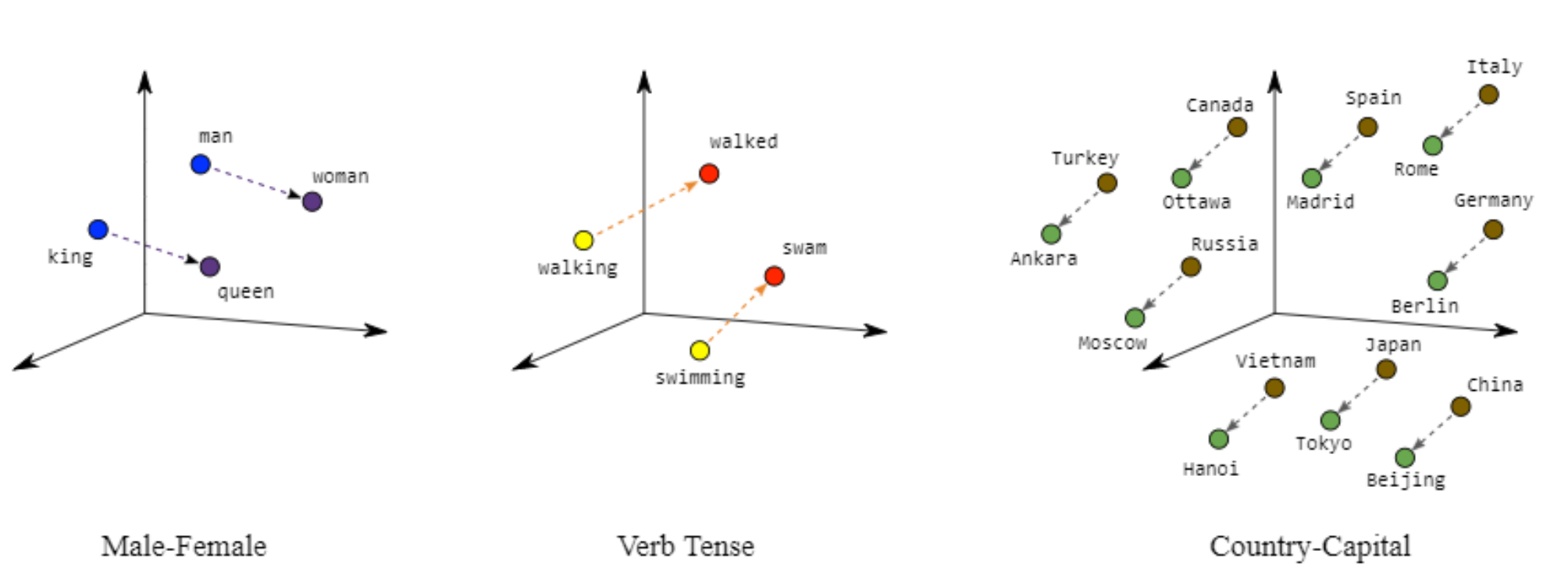
Embedding



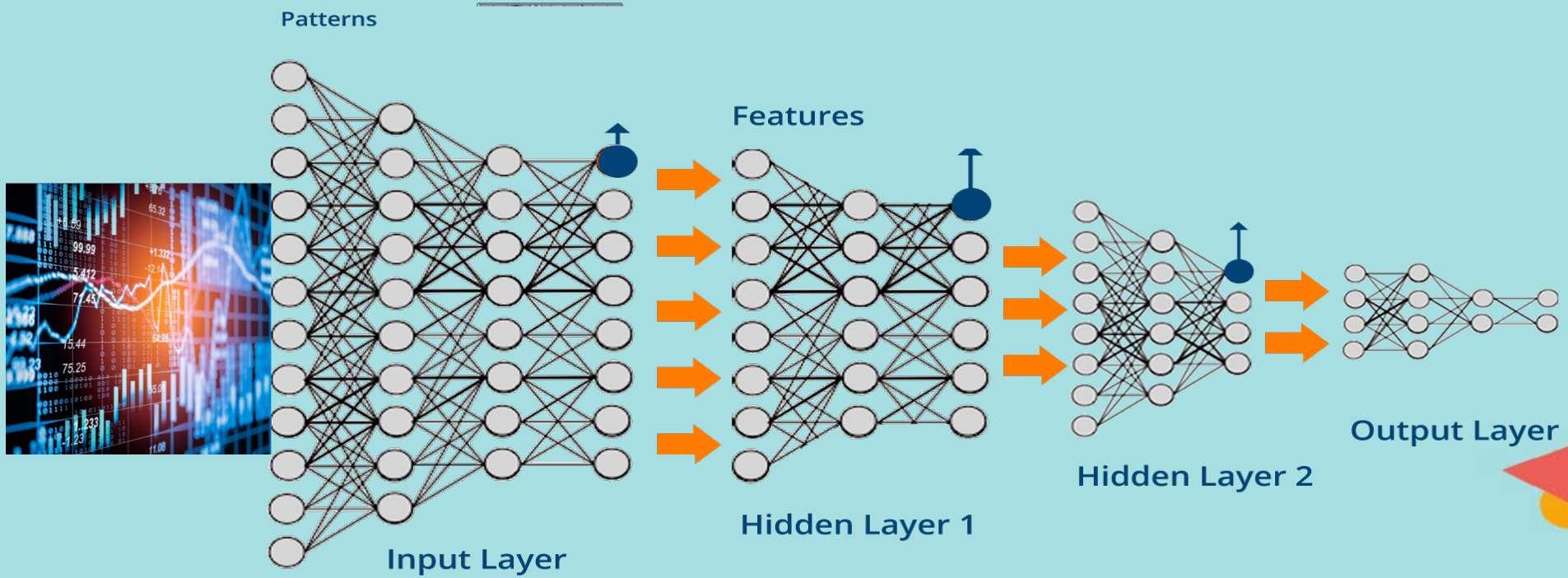
使用One-hot 方法編碼的向量會很高維也很稀疏。假設我們在做自然語言處理（NLP）中遇到了一個包含2000個詞的字典，當時用One-hot編碼時，每一個詞會被一個包含2000個整數的向量來表示，其中1999個數字是0 ，要是我的字典再大一點的話這種方法的計算效率豈不是大打折扣？
訓練神經網絡的過程中，每個嵌入的向量都會得到更新。
通過嵌入層Embedding轉換成D維向量的內容。

Vectors encoded using the One-hot method are very dimensional and sparse. Suppose we encounter a dictionary containing 2,000 words in Natural Language Processing (NLP). When using One-hot encoding, each word is represented by a vector containing 2,000 integers, where the 1999 number is 0. If my dictionary is a little bigger, is the calculation efficiency of this method not greatly reduced?
Each embedded vector is updated during the training of the neural network. The content of the D-dimensional vector is converted by the embedded layer Embedding.

Embeddings: Translating to a Lower-Dimensional Space

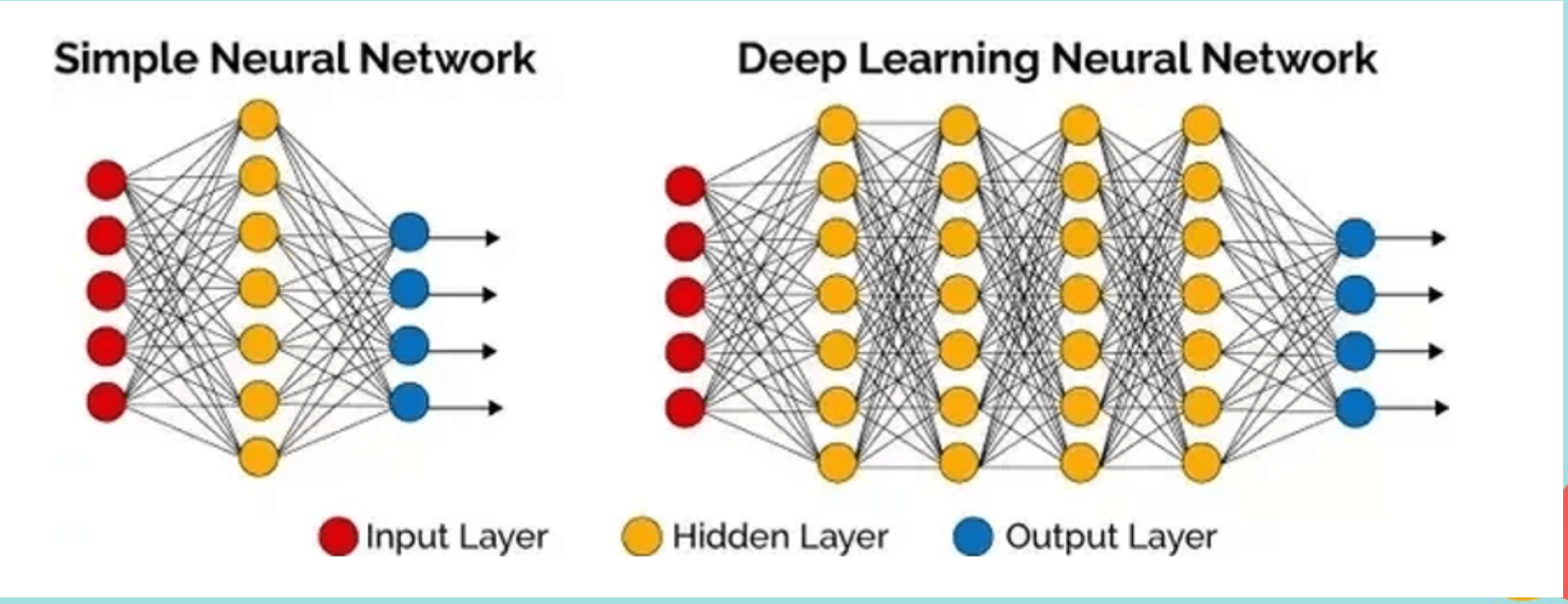


Tensor Flow/張量流

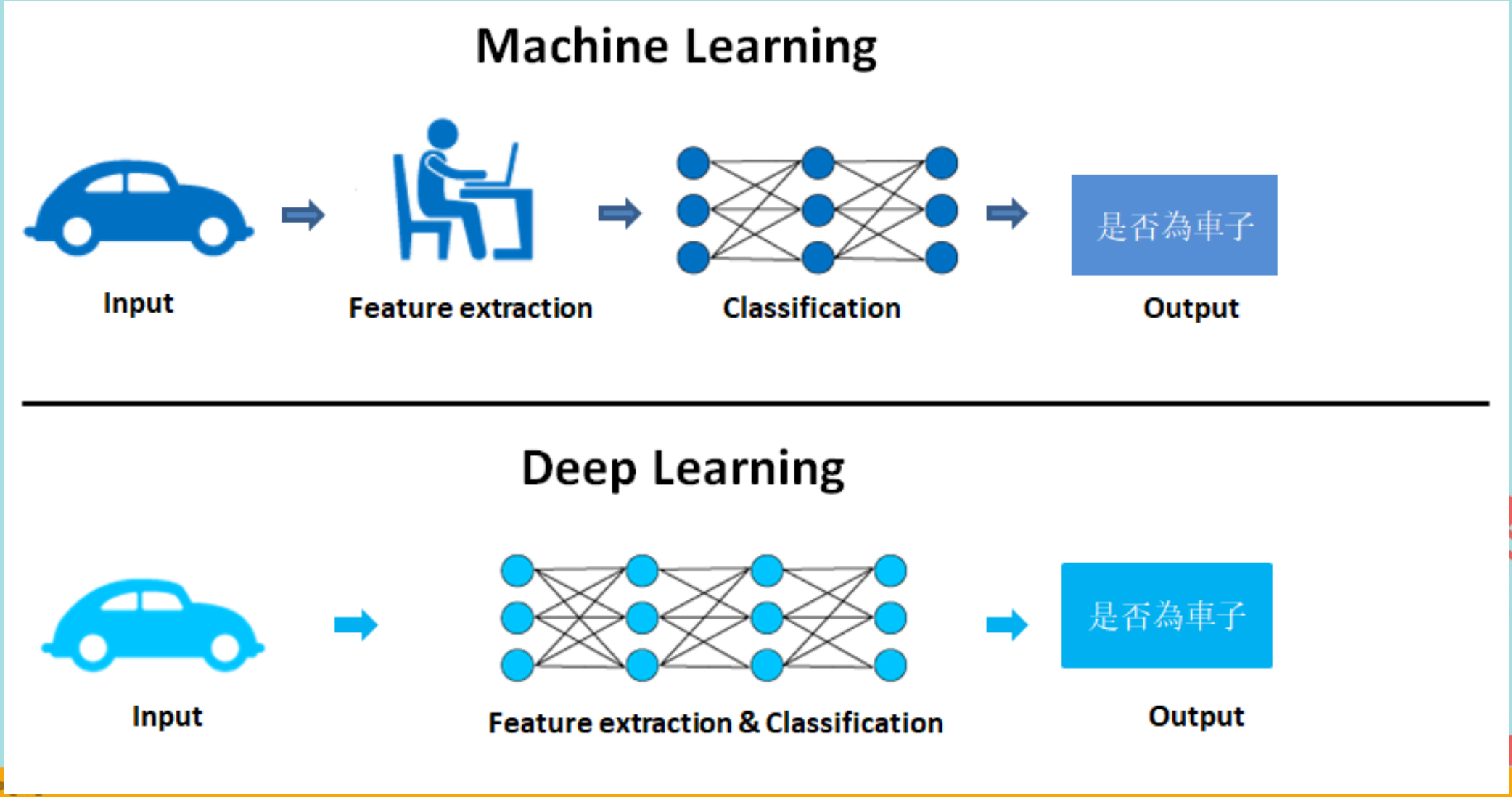


(2)深度學習模型-卷積神經網絡CNN

Deep Learning



Machine learning and Deep learning



IPO模型學深度學習

- (1) IPO-M: 先知道要用什麼樣的深度學習模型
- (2) IPO-I: 研究如何餵資料給模型
- (3) IPO-P:模型的訓練
- (4) IPO-P:模型的評估
- (5) IPO-O:模型的輸出

Deep learning basics

- Deep learning basics
  - Neural Networks神經網路
  - Training and Loss訓練和損失
  - Backpropagation(BP)反向傳播算法
  - Gradient Descent/Optimizer梯度下降法
  - ROC Curve and AUC接收者操作特徵曲線
  - Overfitting & Regularization過適
  - Activation Functions激勵函數
  - Loss Functions損失函數
  - Confusion matrix混淆矩陣
  - Transfer learning遷移學習

Deep learning models

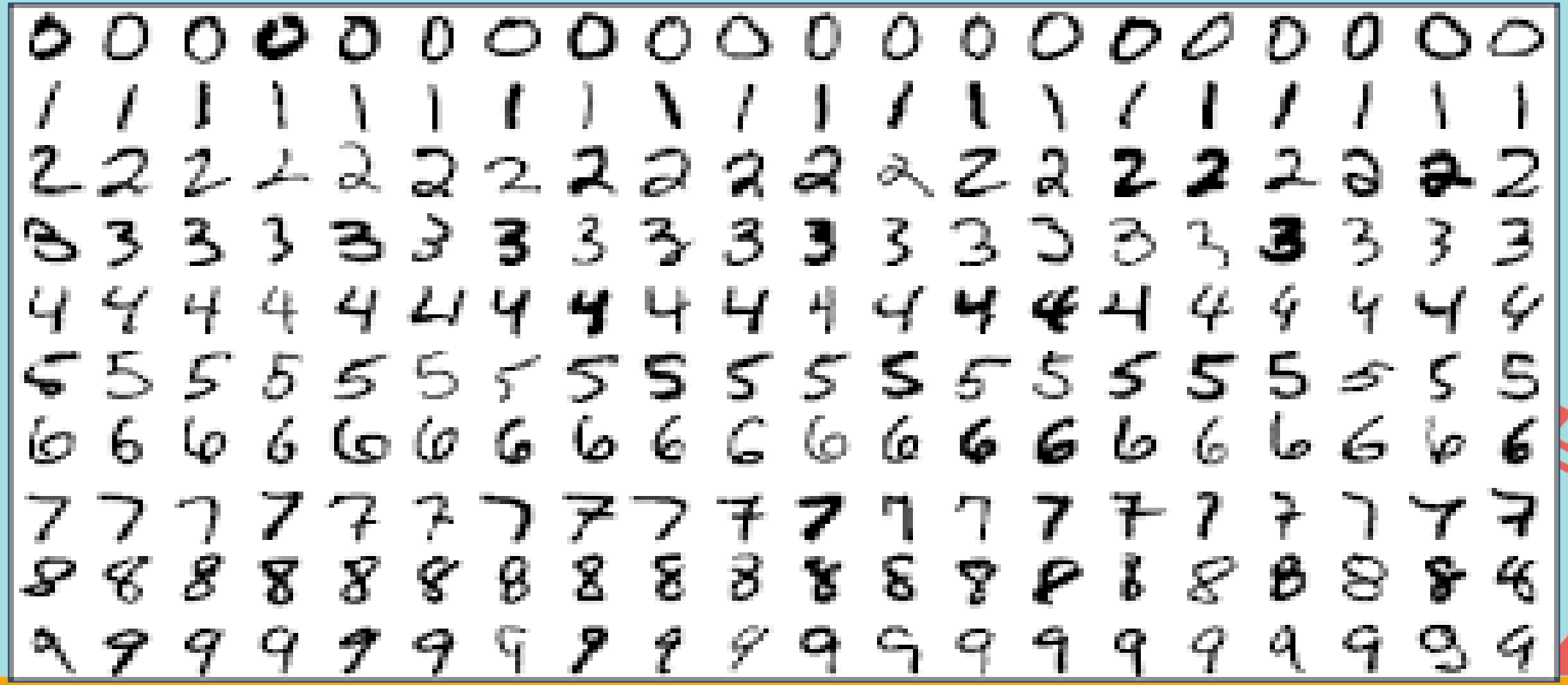
- Feed Forward Neural Networks (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Networks (RNN)
- Autoencoder
- Generative Adversarial Network (GAN)
- Attention mechanism
- Transformers

深度學習應用

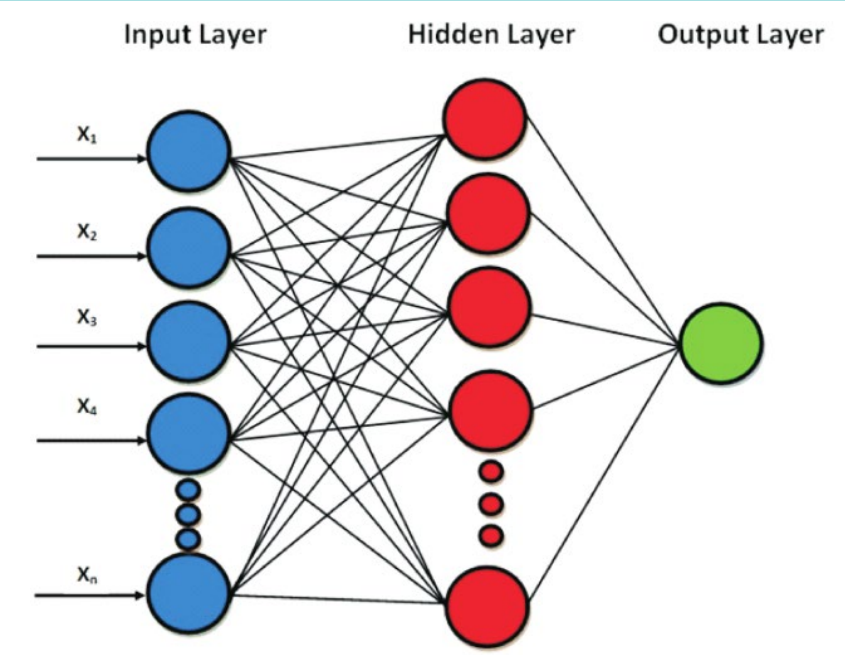
- Deep learning topics for Image processing (IP)
  - Image Classification
  - Image Segmentation
  - Image Captioning
  - Image Generation
- Deep learning topics for Natural Language Processing (NLP)
  - Authorship Attribution
  - Sentiment analysis
  - Text Summarization
  - Question Answering
- Deep Learning for Games
  - Q-Learning
  - Deep Reinforcement Learning
  - Deep Q-Learning

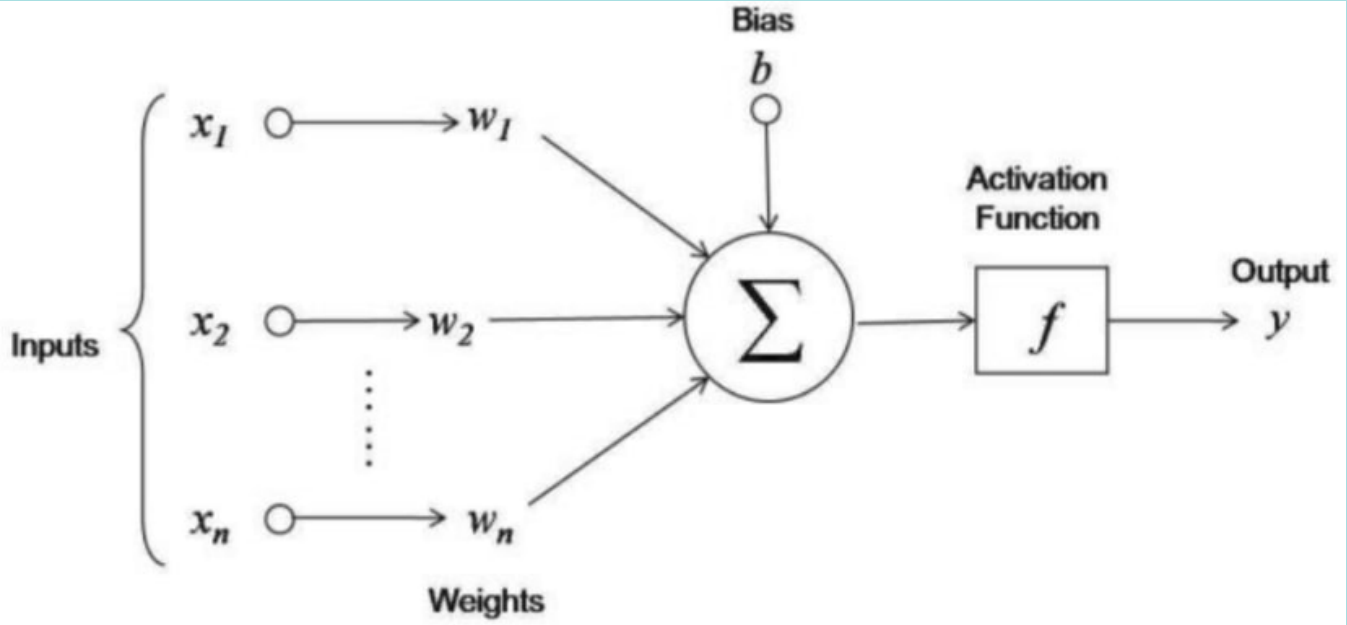
手寫數位識別

機器學習的起始點



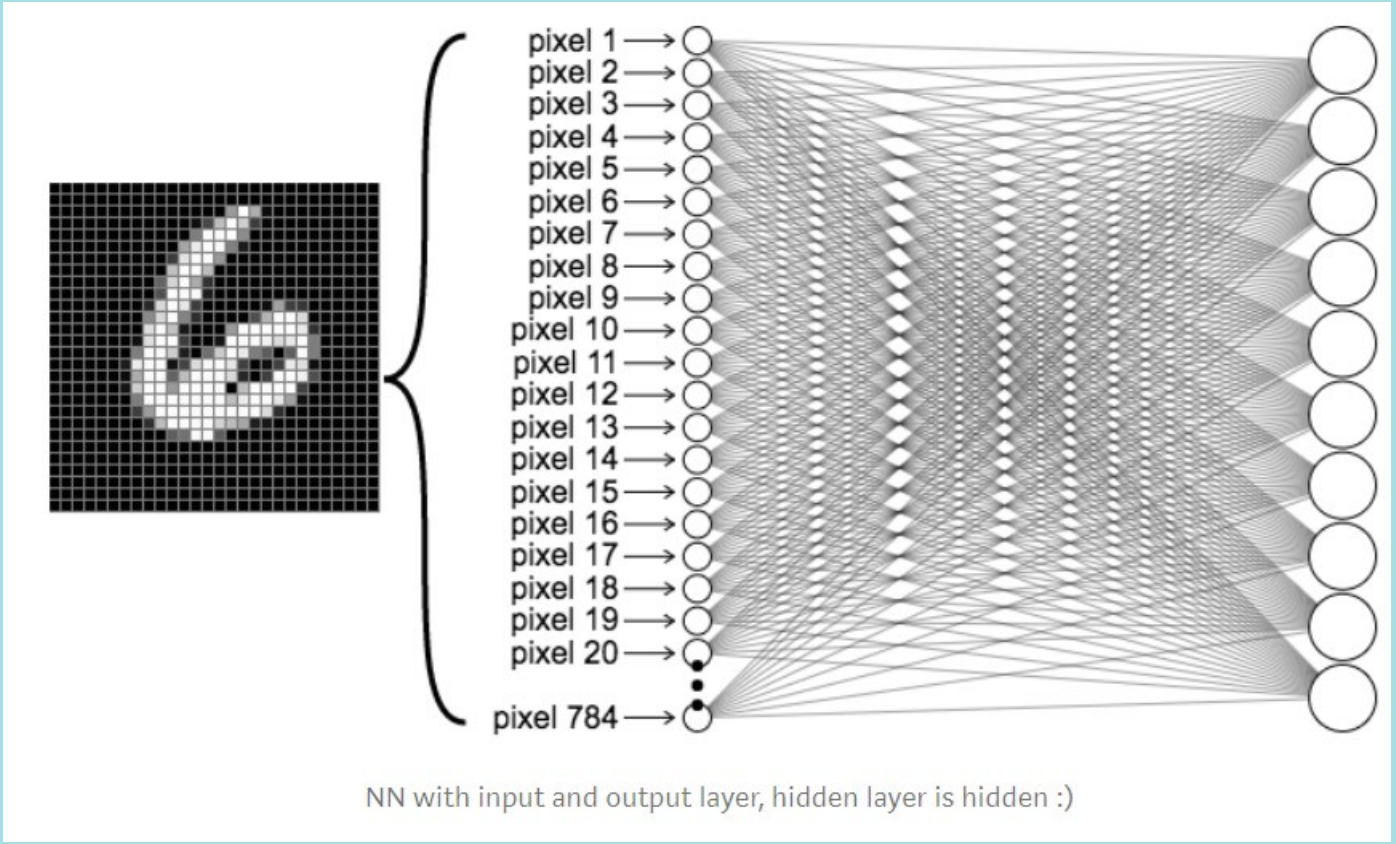
Feed Forward Neural Networks





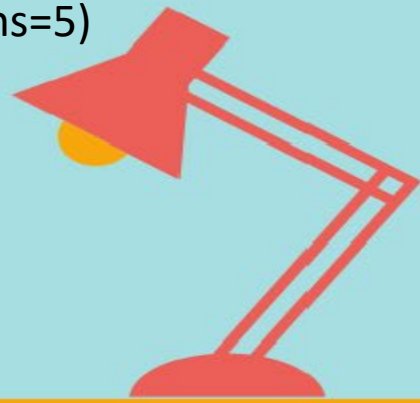
MNIST handwritten digit dataset

Feed Forward Neural Networks



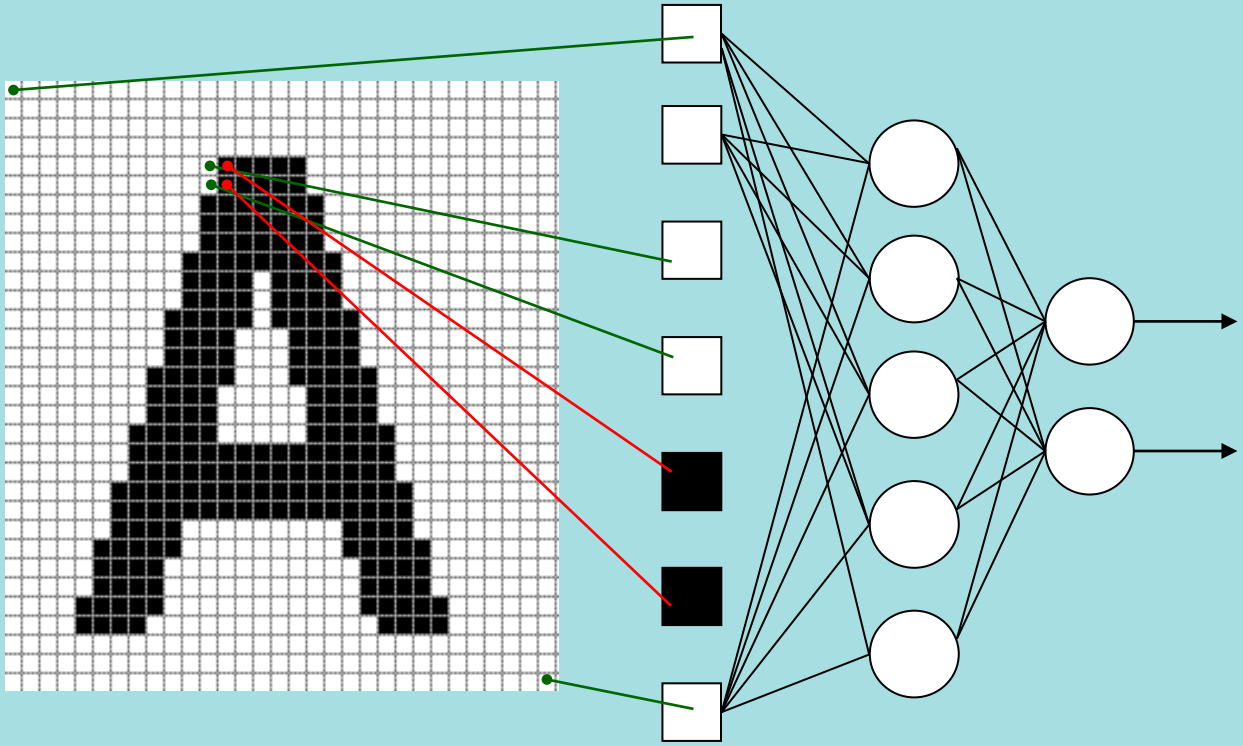
NN with input and output layer, hidden layer is hidden :)

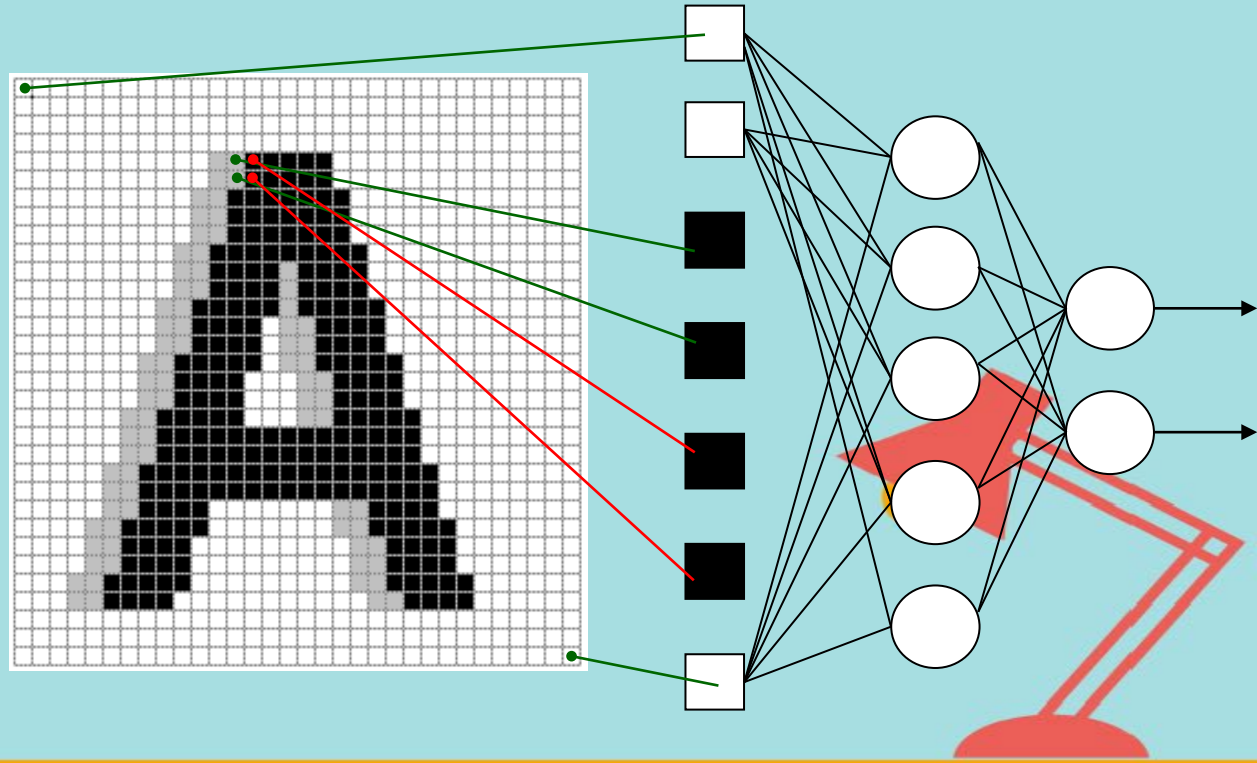
```
model = keras.models.Sequential([
  keras.layers.Flatten(input_shape=(28, 28)),
  keras.layers.Dense(128, activation='relu'),
  keras.layers.Dropout(0.2),
  keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```



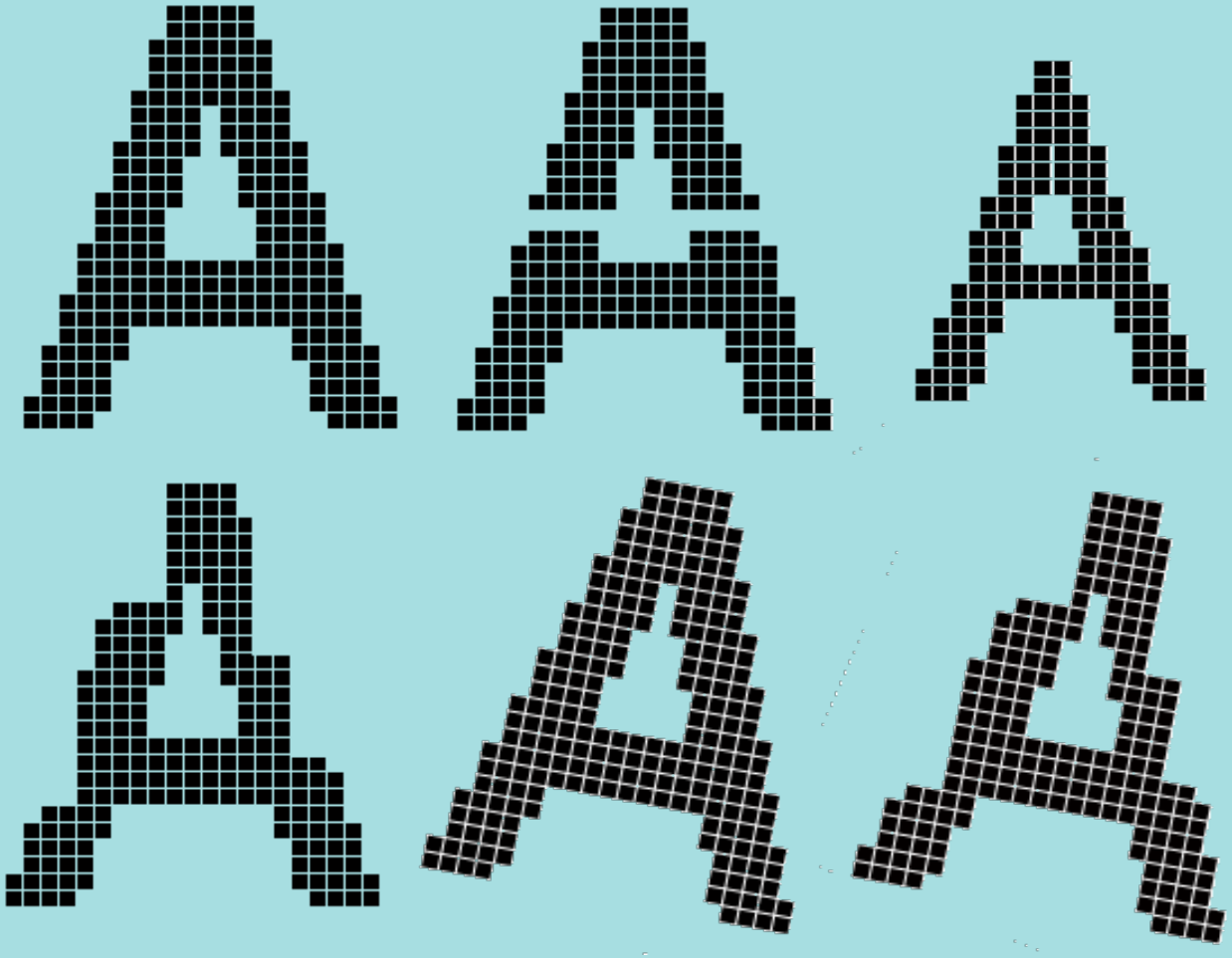
Drawbacks of Feed Forward neural networks

Little or no invariance to shifting, scaling, and other forms of distortion





Scaling, and other forms of distortion



CNN History

Yann LeCun, Professor of Computer Science
The Courant Institute of Mathematical Sciences
New York University
Room 1220, 715 Broadway, New York, NY 10003, USA.
(212)998-3283 yann@cs.nyu.edu

In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

Convolutional neural network(CNN)

Add convolution and pooling layers before feedforward neural network

卷積計算

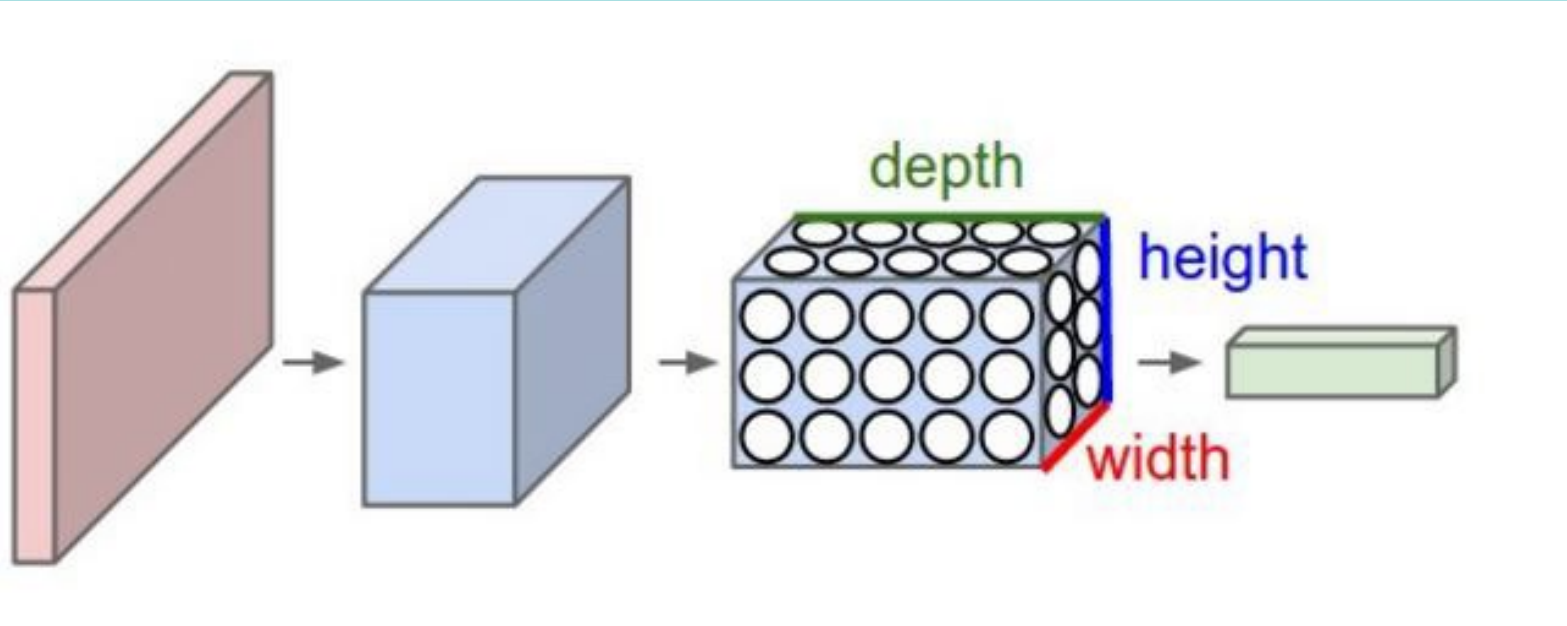
- 步伐(stride)
- 填充(padding)

池化運算

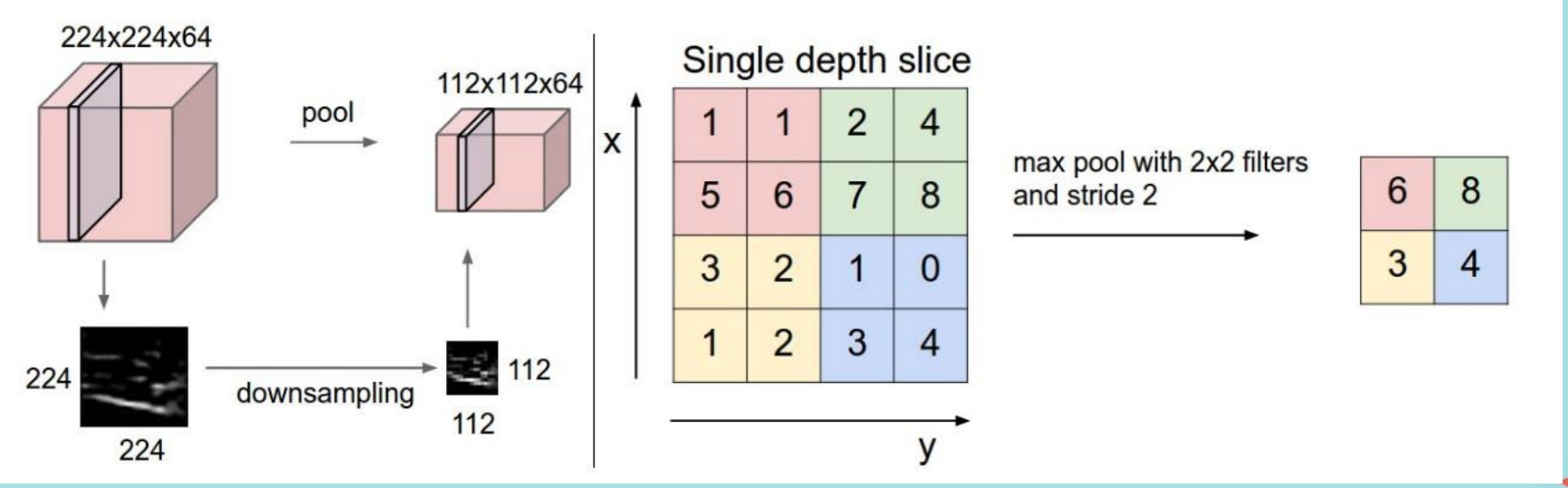
```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid', data_format=None, dilation_rate=(1, 1),
activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None,
activity_regularizer=None, kernel_constraint=None,
bias_constraint=None)
```

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None,
padding='valid', data_format=None)
```

Convolutional layer



Pooling layer



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convoluti

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product → 3 -1

6 x 6 image

Convoluti

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

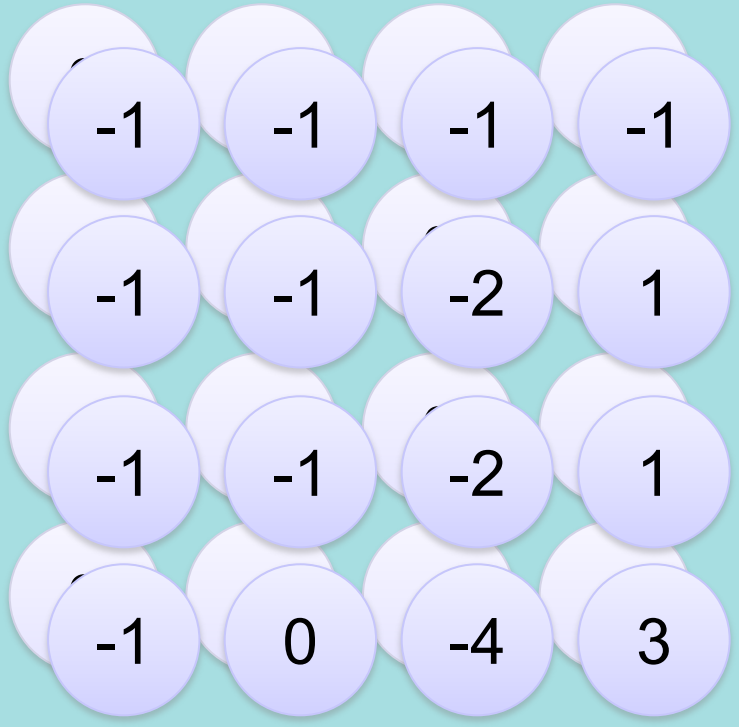
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter

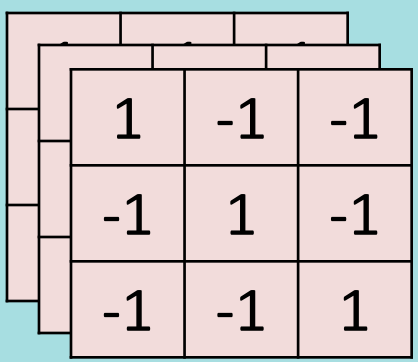


Feature Map

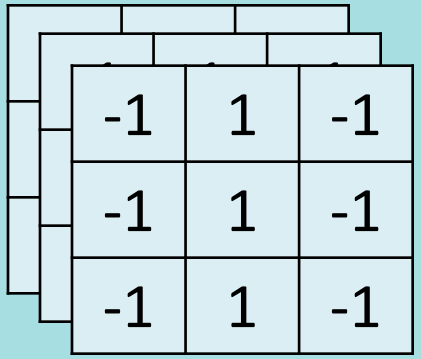
Two 4 x 4 images

Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

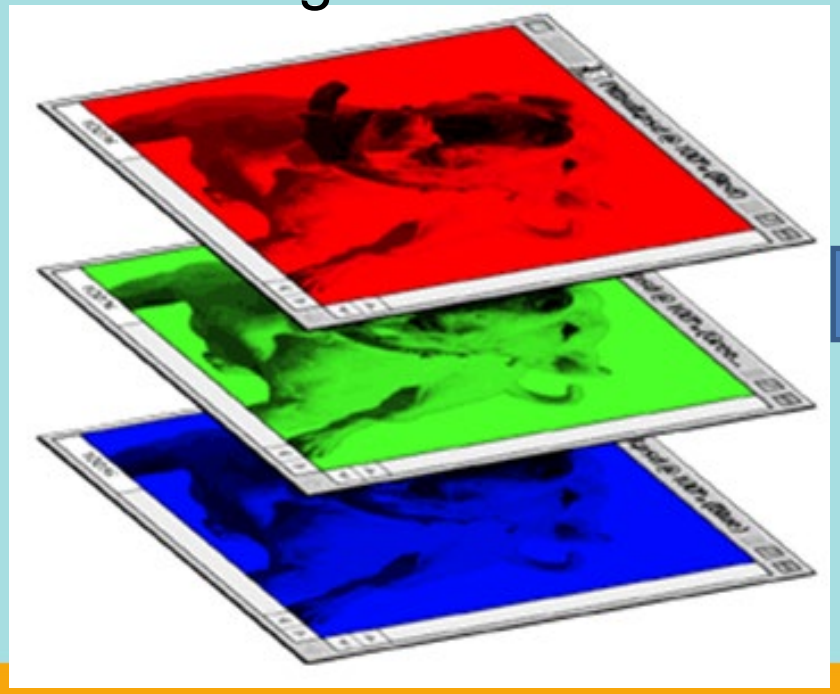


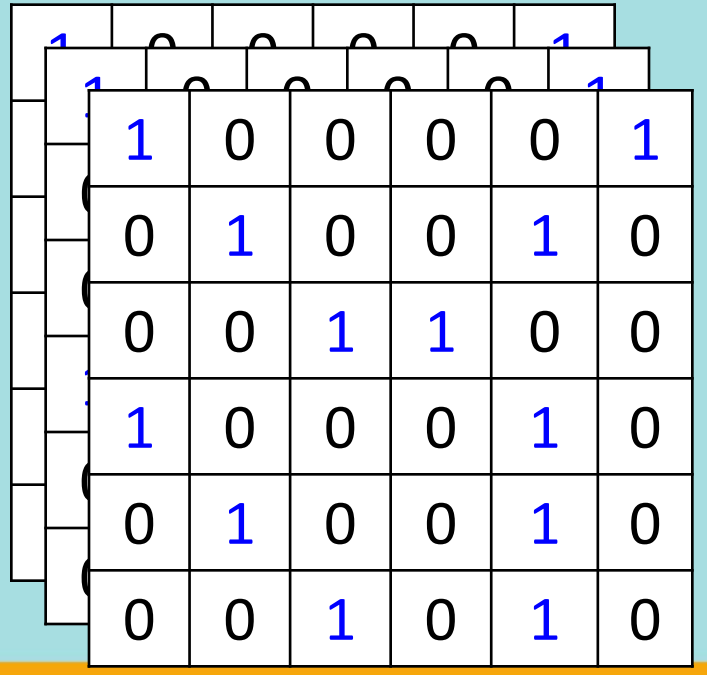
Filter 1



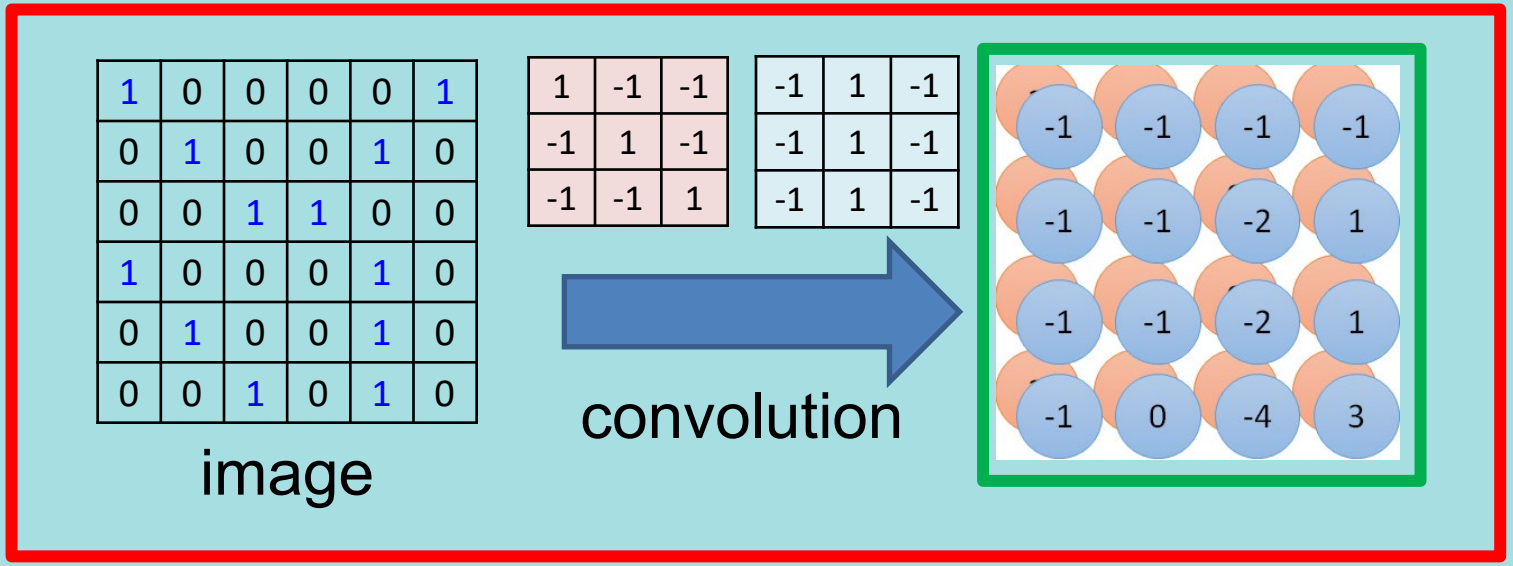
Filter 2

Color image



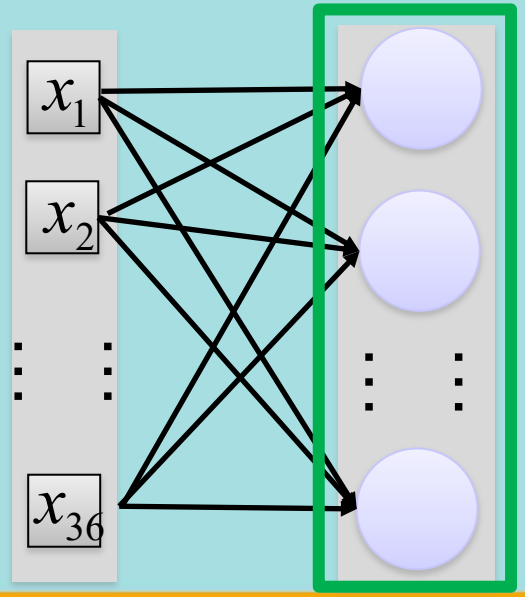


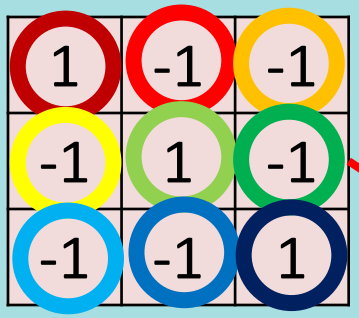
Convolution v.s. Fully Connected



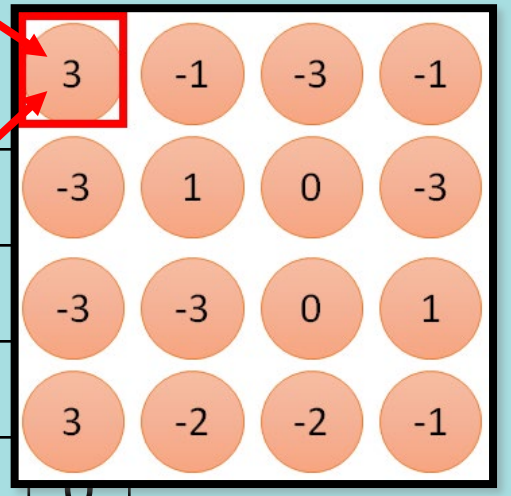
Fully-connected

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0





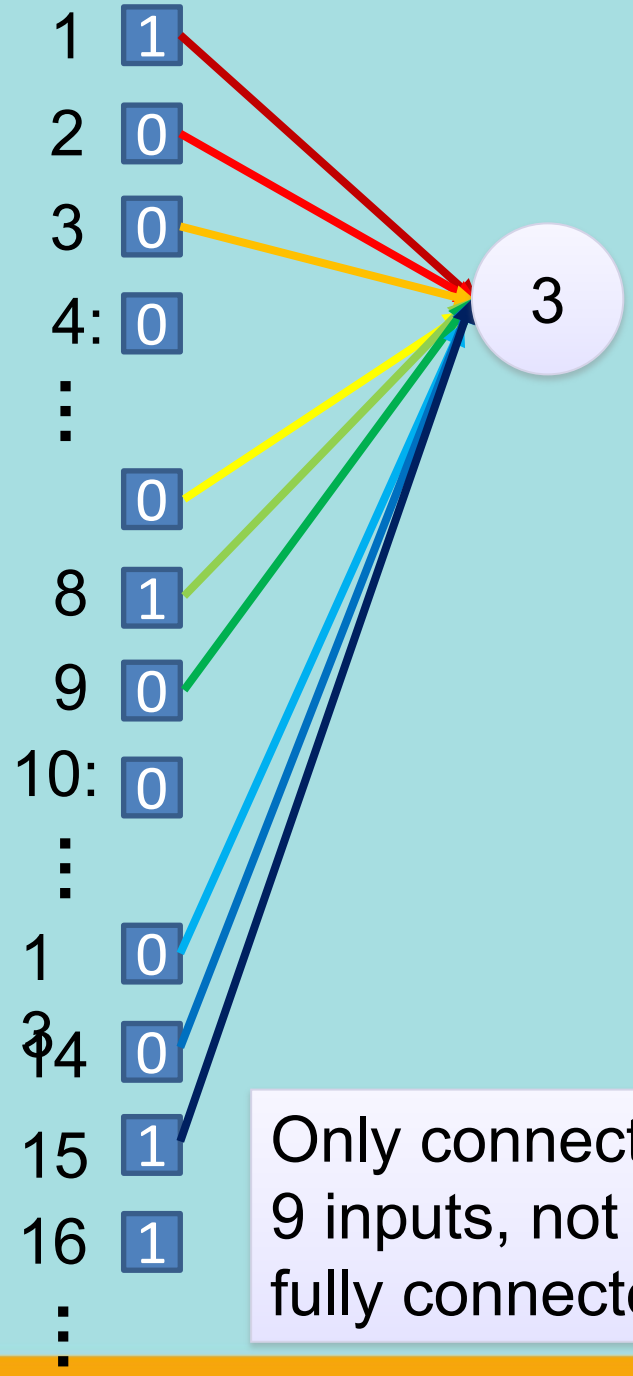
Filter 1



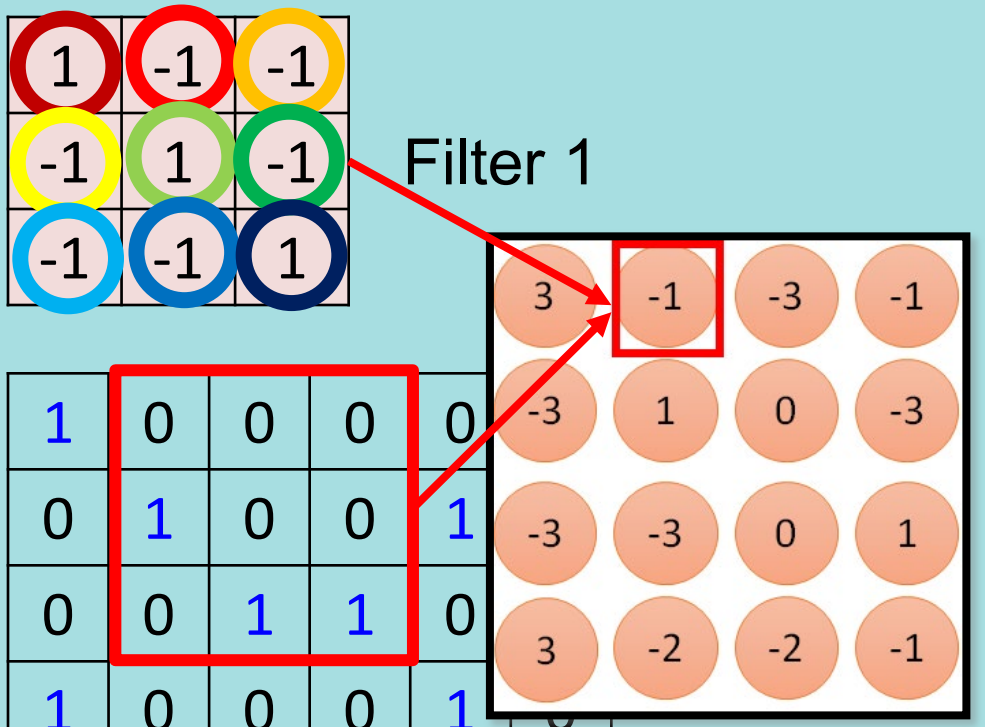
1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

fewer parameters!

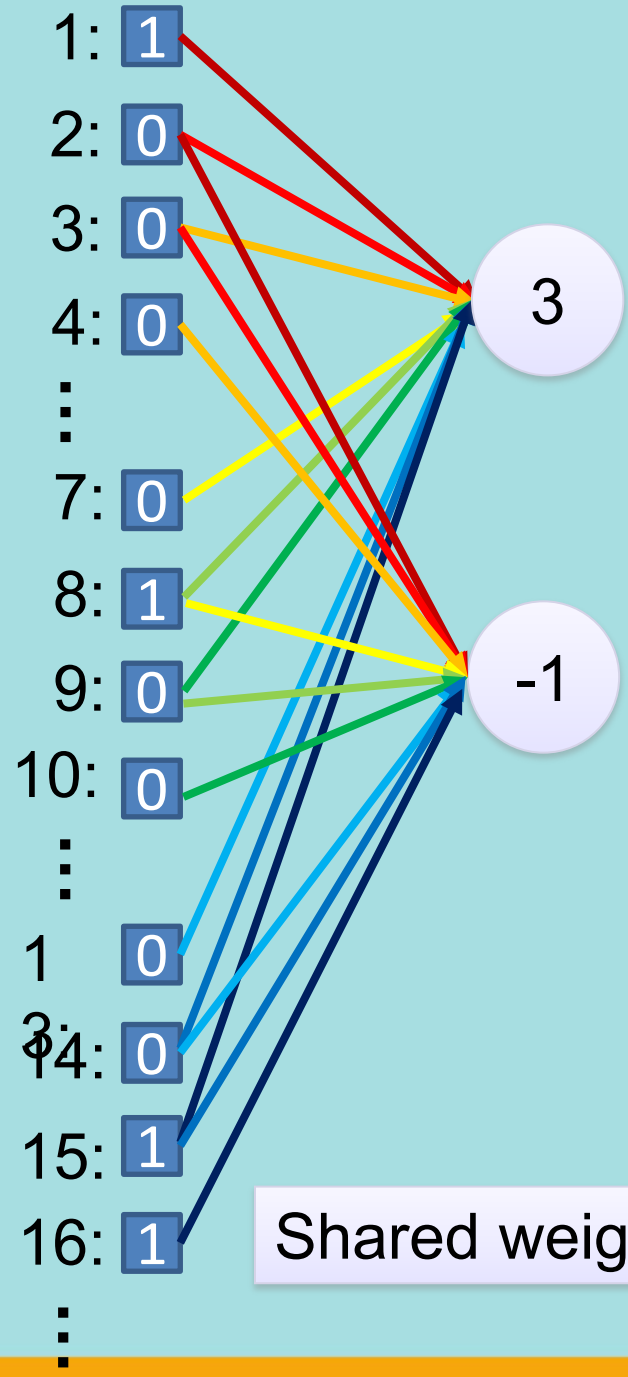


Only connect to 9 inputs, not fully connected



Fewer parameters

Even fewer parameters



Shared weights

Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

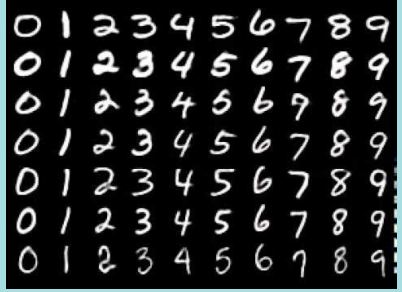
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

-1	-1	-1	-1
-1	-1	-2	1
-1	-1	-2	1
-1	0	-4	3

The whole CNN



Convolution

Max Pooling

Convolution

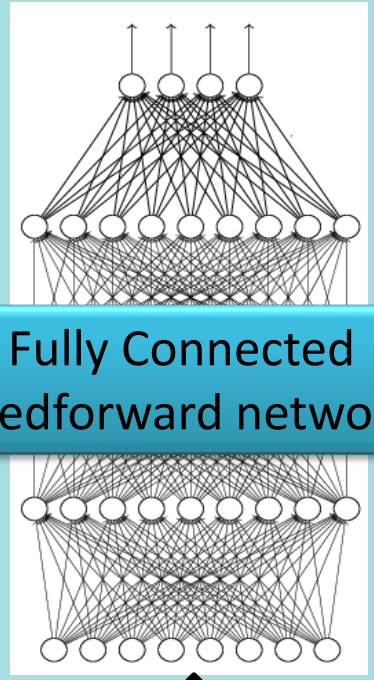
Max Pooling

Can repeat many times

Flattened

Fully Connected Feedforward network

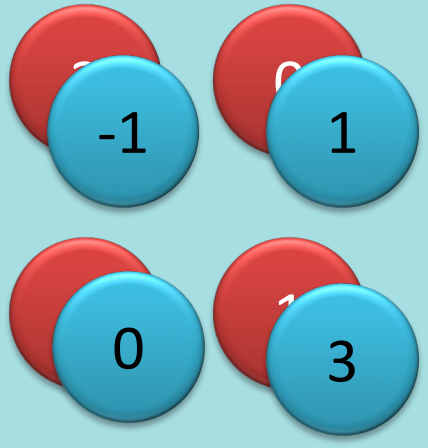
0, 1, 2, 3……



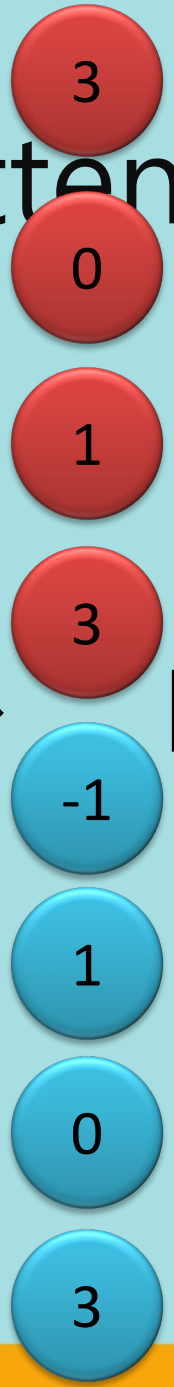
A CNN compresses a fully connected network in two ways:

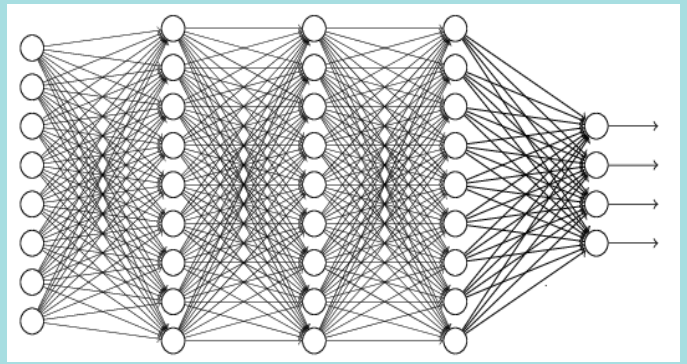
- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

Flattening



Flattened



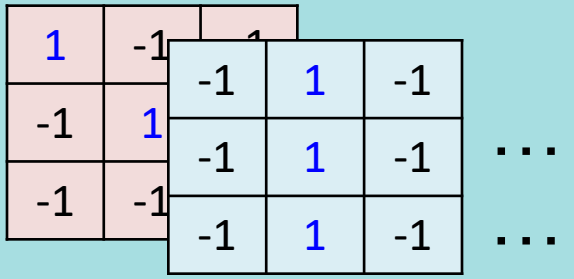


Fully Connected
Feedforward network

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
                input_shape=(28,28,1)) )
```



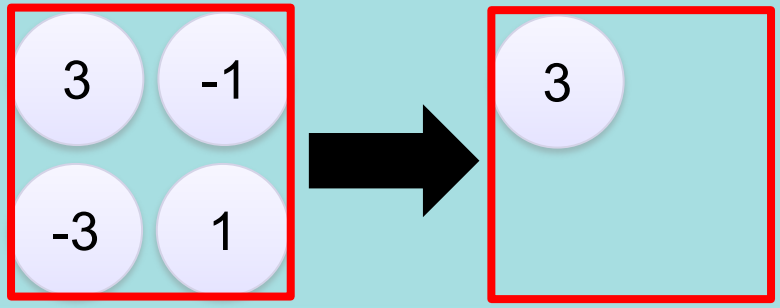
There are 25 3x3 filters.

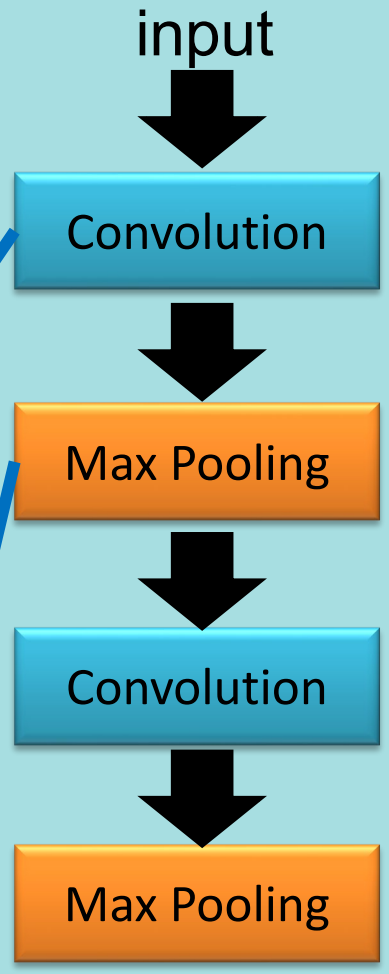
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

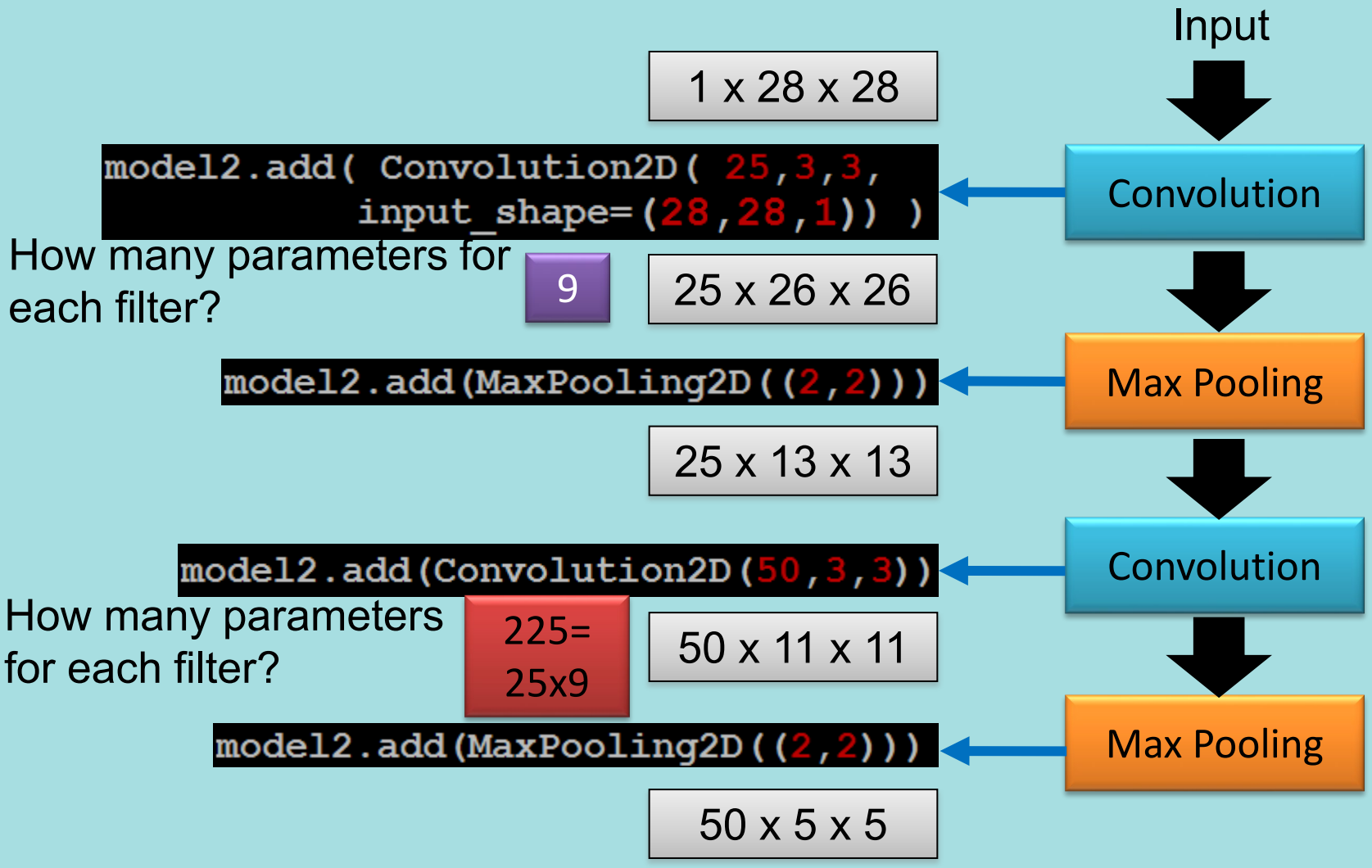
```
model2.add(MaxPooling2D((2,2)))
```





CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***



CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***

Input

1 x 28 x 28

Convolution

25 x 26 x 26

Max Pooling

25 x 13 x 13

Convolution

50 x 11 x 11

Max Pooling

50 x 5 x 5

Flatten

```
model2.add(Flatten())
```

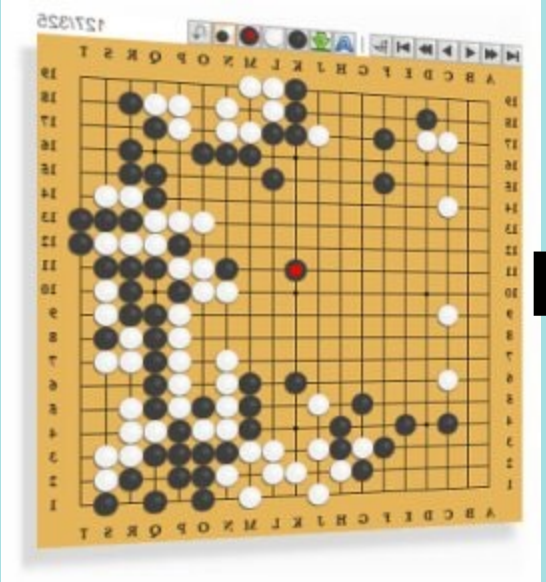
1250

Fully connected feedforward network

```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

Output

AlphaGo

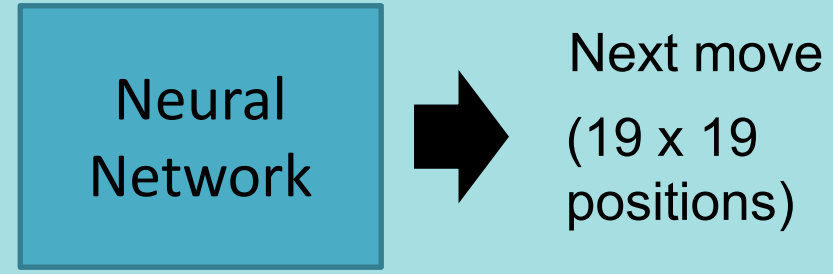


19 x 19 matrix

Black: 1

white: -1

none: 0



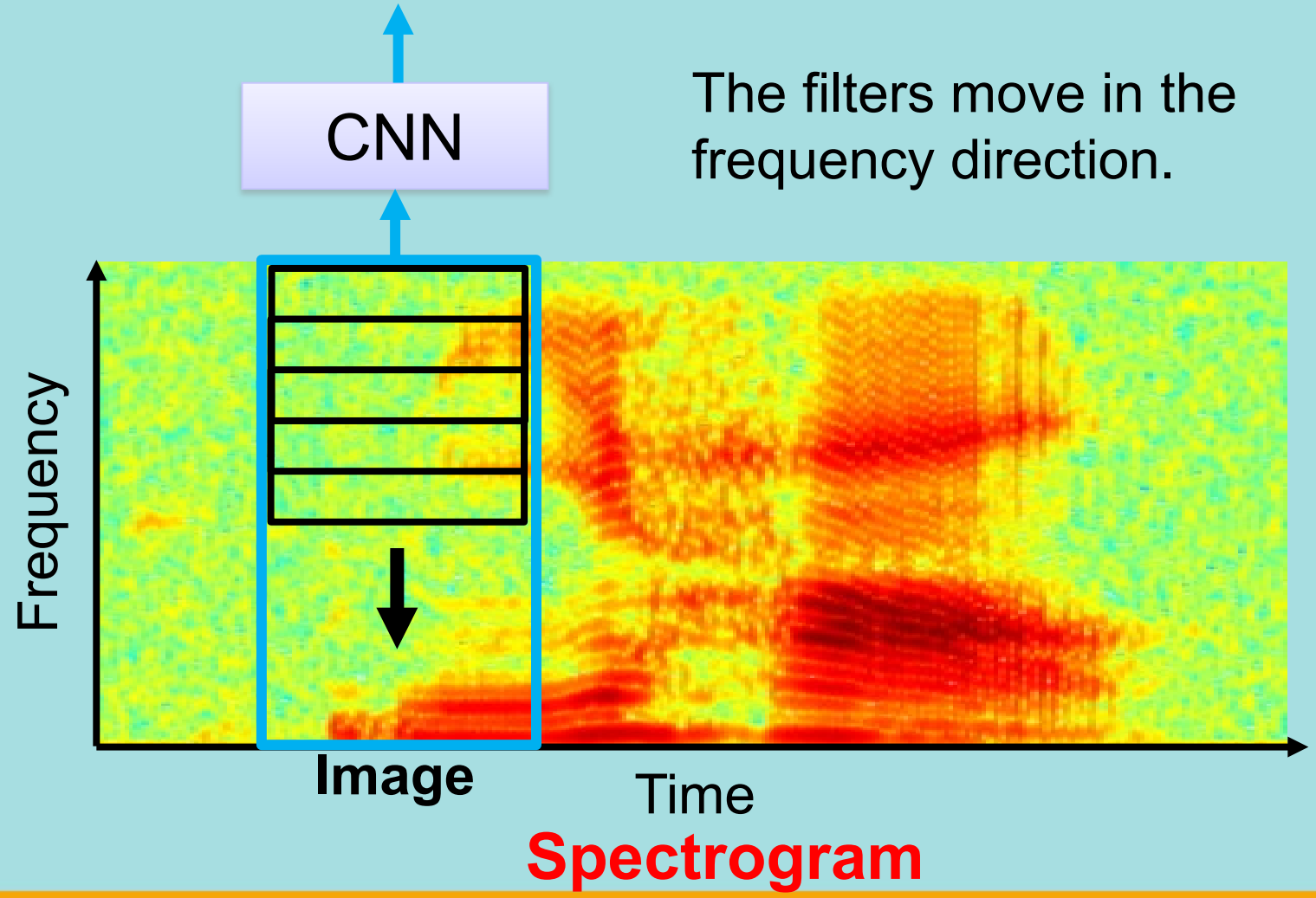
Next move

(19 x 19 positions)

Fully-connected feedforward network can be used

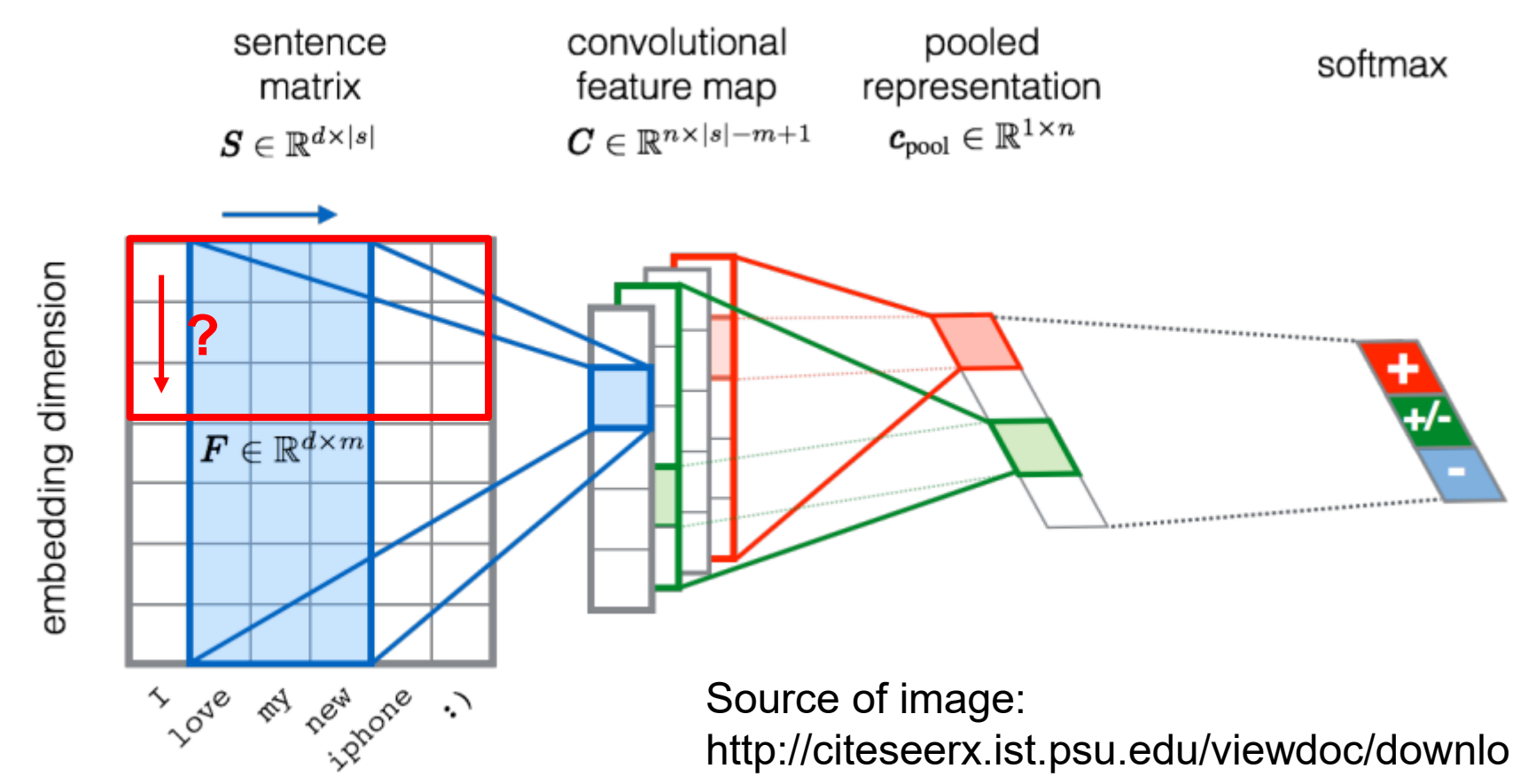
But CNN performs much better

CNN in speech recognition



The filters move in the frequency direction.

CNN in text classification

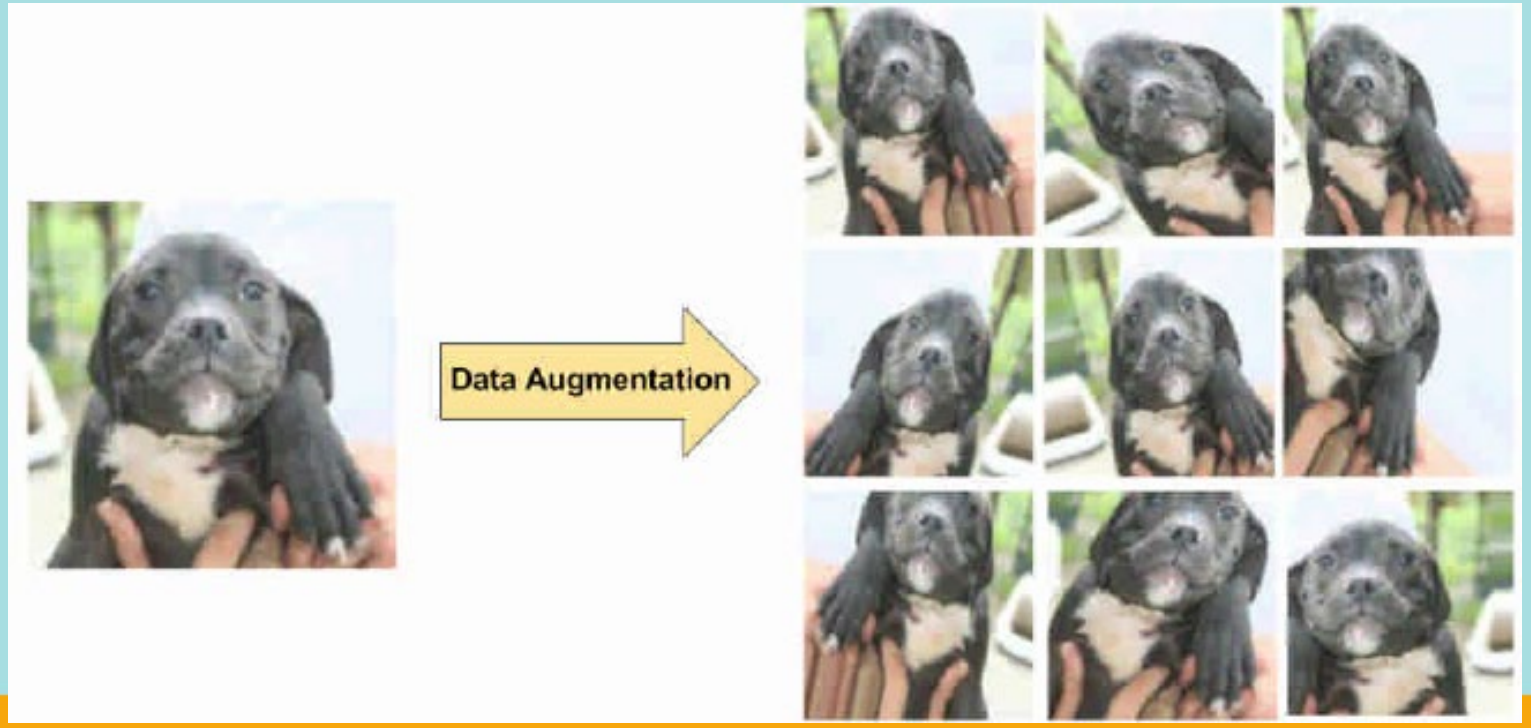


Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

Deep Convolutional Neural Networks

- e.g., Millions of images
- Applying dropout is unnecessary
- The value of data augmentation is also diminished

– Nine new images generated from original image using random transformations



Input
3x3 conv, 64
3x3 conv, 64
Pool 1/2
3x3 conv, 128
3x3 conv, 128
Pool 1/2
3x3 conv, 256
3x3 conv, 256
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool 1/2
FC 4096
FC 4096
FC 1000
Softmax

VGGNet

- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1
  and 2x2 MAX POOL , stride 2

- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers

- ZFNet: 11.7% top 5 error in ILSVRC’ 13
- VGGNet: 7.3% top 5 error in ILSVRC’ 14

VGGNet

- **Why use smaller filters? (3x3 conv)**

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

- **What is the effective receptive field of three 3x3 conv (stride 1) layers?**

7x7

But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

Input
3x3 conv, 64
3x3 conv, 64
Pool
3x3 conv, 128
3x3 conv, 128
Pool
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
Pool
FC 4096
FC 4096
FC 1000
Softmax

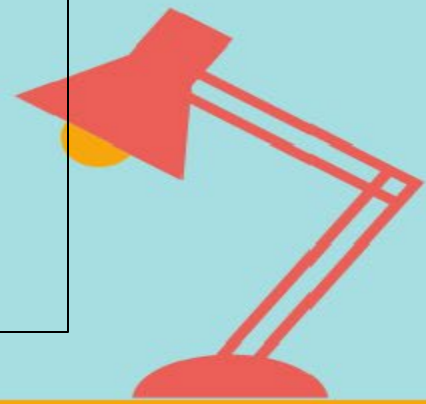
VGGNet

VGG16:

TOTAL memory: 24M * 4 bytes ~= 96MB / image

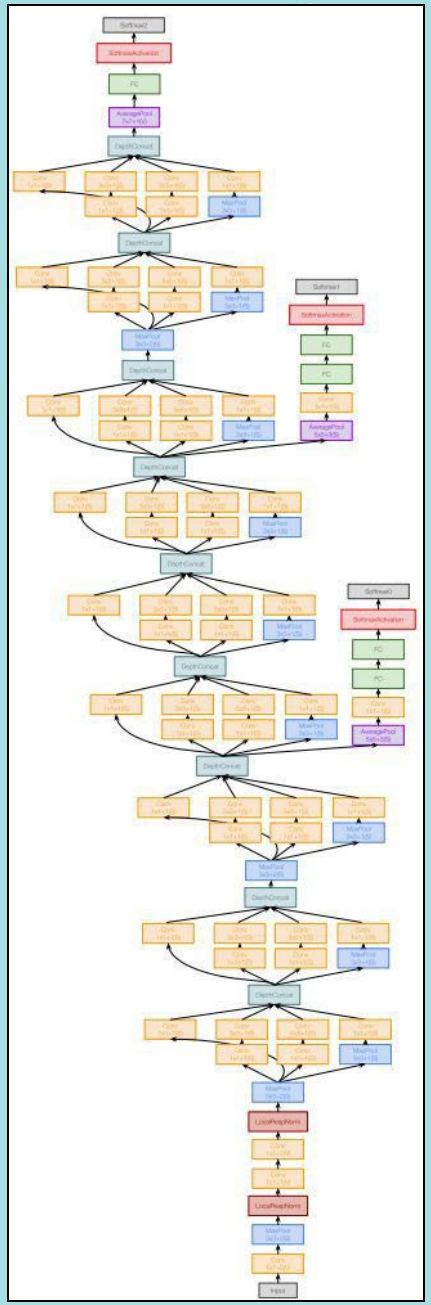
TOTAL params: 138M parameters

Input	memory: 224*224*3=150K	params: 0
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
3x3 conv, 64	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
Pool	memory: 112*112*64=800K	params: 0
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
3x3 conv, 128	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
Pool	memory: 56*56*128=400K	params: 0
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
3x3 conv, 256	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
Pool	memory: 28*28*256=200K	params: 0
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 14*14*512=100K	params: 0
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
3x3 conv, 512	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
Pool	memory: 7*7*512=25K	params: 0
FC 4096	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC 4096	memory: 4096	params: 4096*4096 = 16,777,216
FC 1000	memory: 1000	params: 4096*1000 = 4,096,000



GoogleNet

- *Going Deeper with Convolutions - Christian Szegedy et al.; 2015*
- ILSVRC 2014 competition winner
- Also significantly deeper than AlexNet
- x12 less parameters than AlexNet
- Focused on computational efficiency

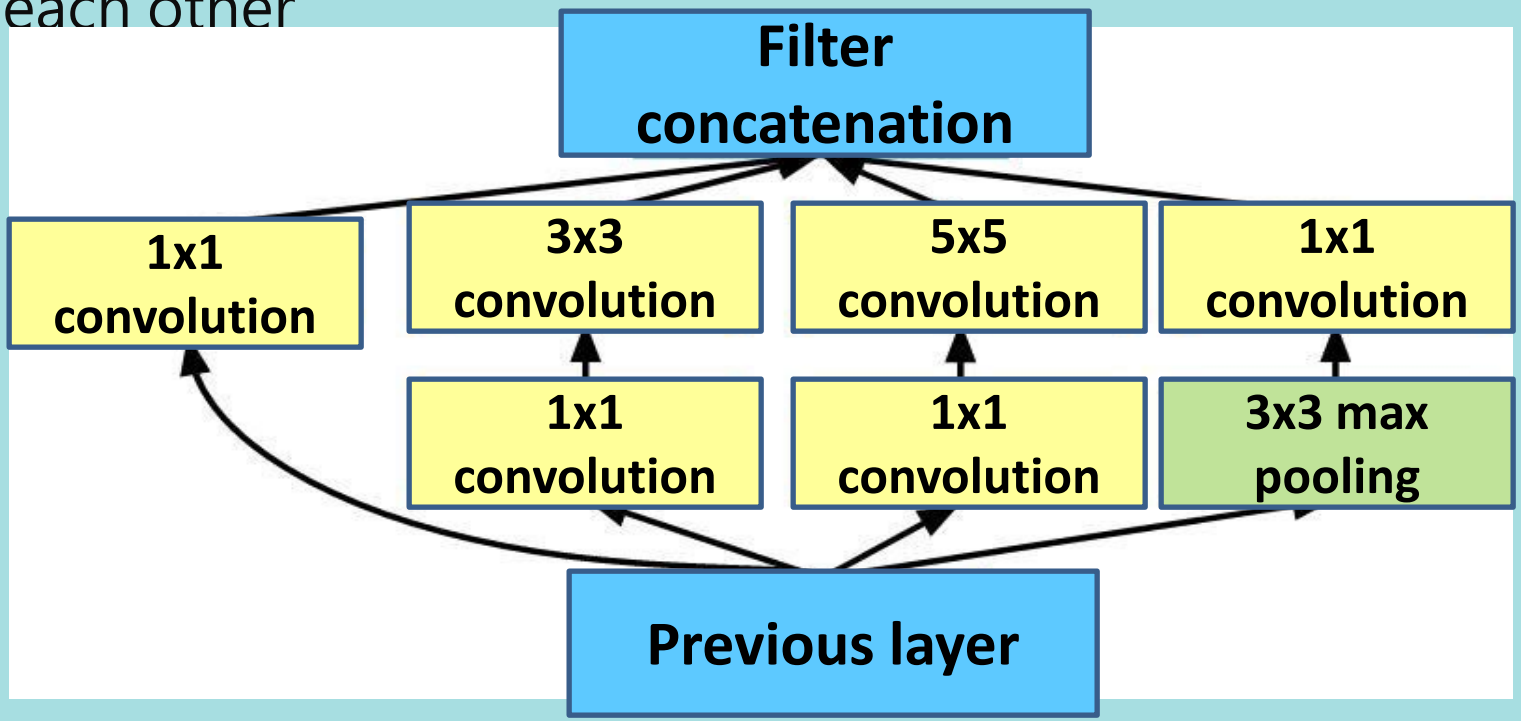


GoogleNet

- 22 layers
- Efficient "**Inception**" **module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC'14 classification winner (6.7% top 5 error)

GoogleNet

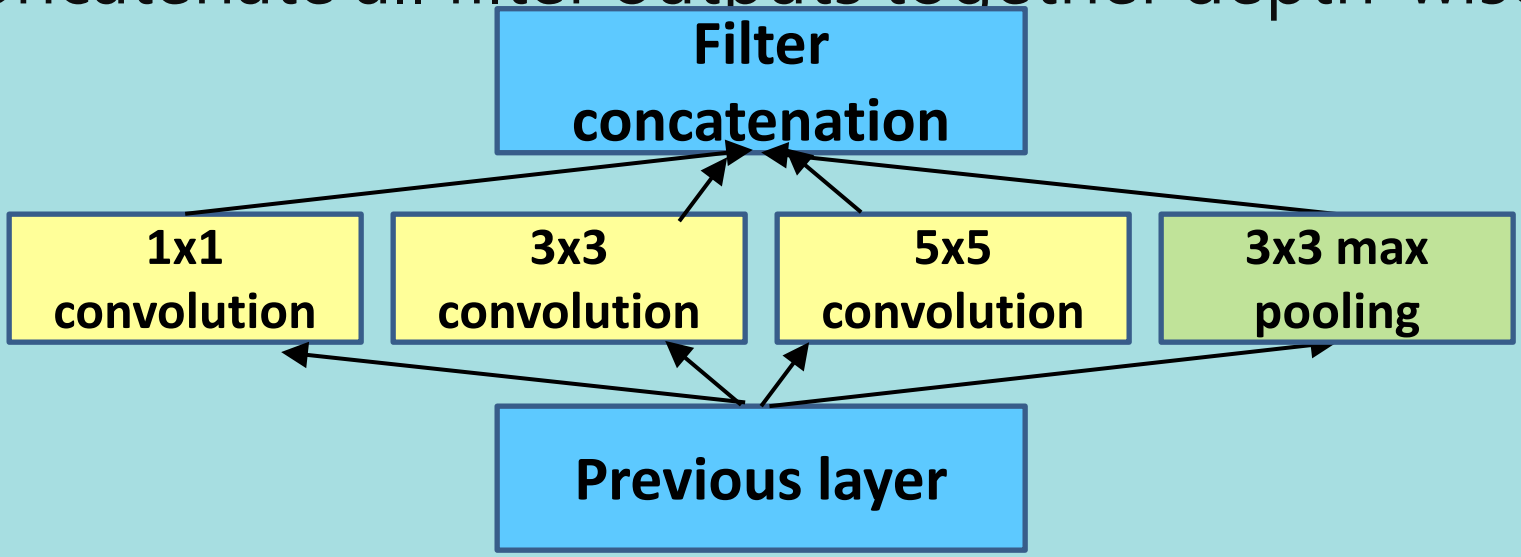
"**Inception module**" : design a good local network topology (network within a network) and then stack these modules on top of each other



GoogleNet

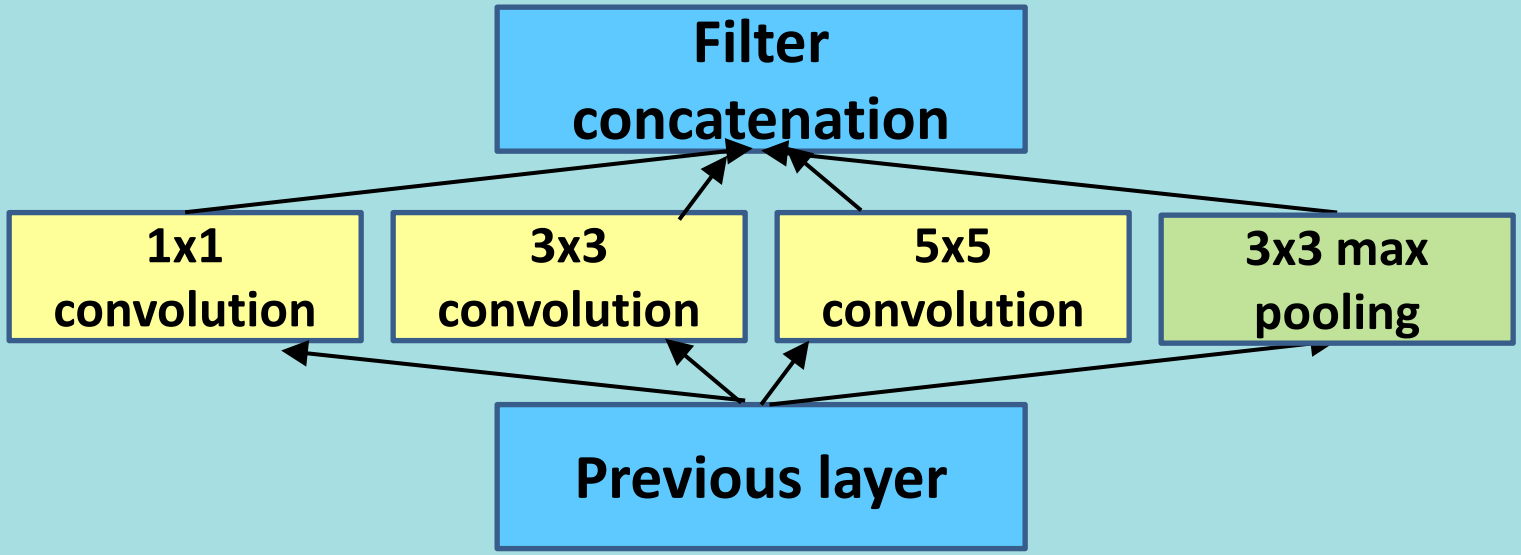
Naïve Inception Model

- Apply parallel filter operations on the input :
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise



GoogleNet

- What’ s the problem with this?
  High computational complexity



GoogleNet

- Output volume sizes:

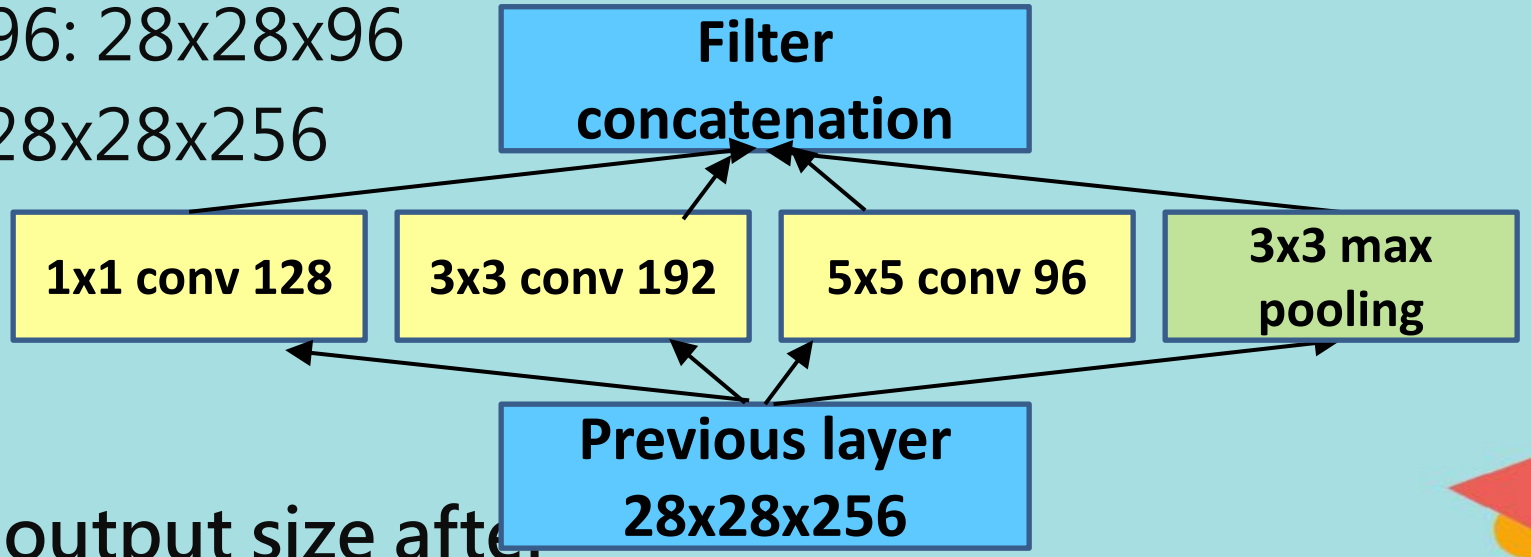
1x1 conv, 128: 28x28x128

3x3 conv, 192: 28x28x192

5x5 conv, 96: 28x28x96

3x3 pool: 28x28x256

Example:



- What is output size after filter concatenation?

28x28x(128+192+96+256) = 28x28x672

GoogleNet

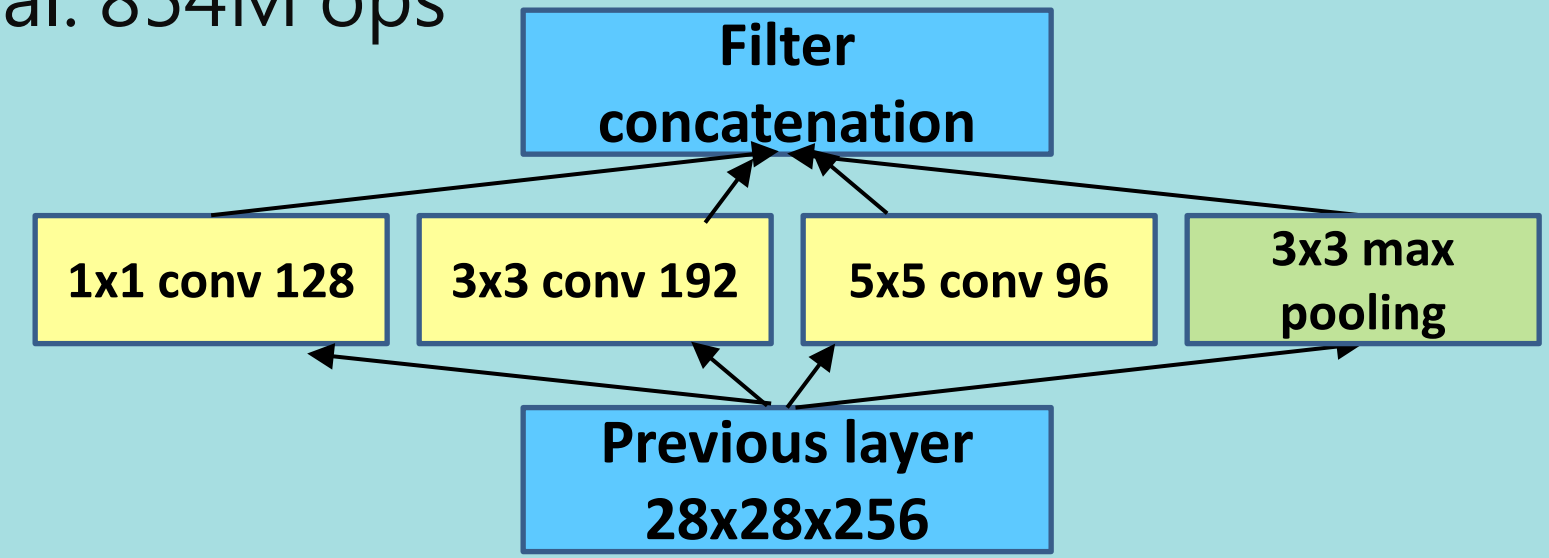
- **Number of convolution operations:**

1x1 conv, 128: 28x28x128x1x1x256

3x3 conv, 192: 28x28x192x3x3x256

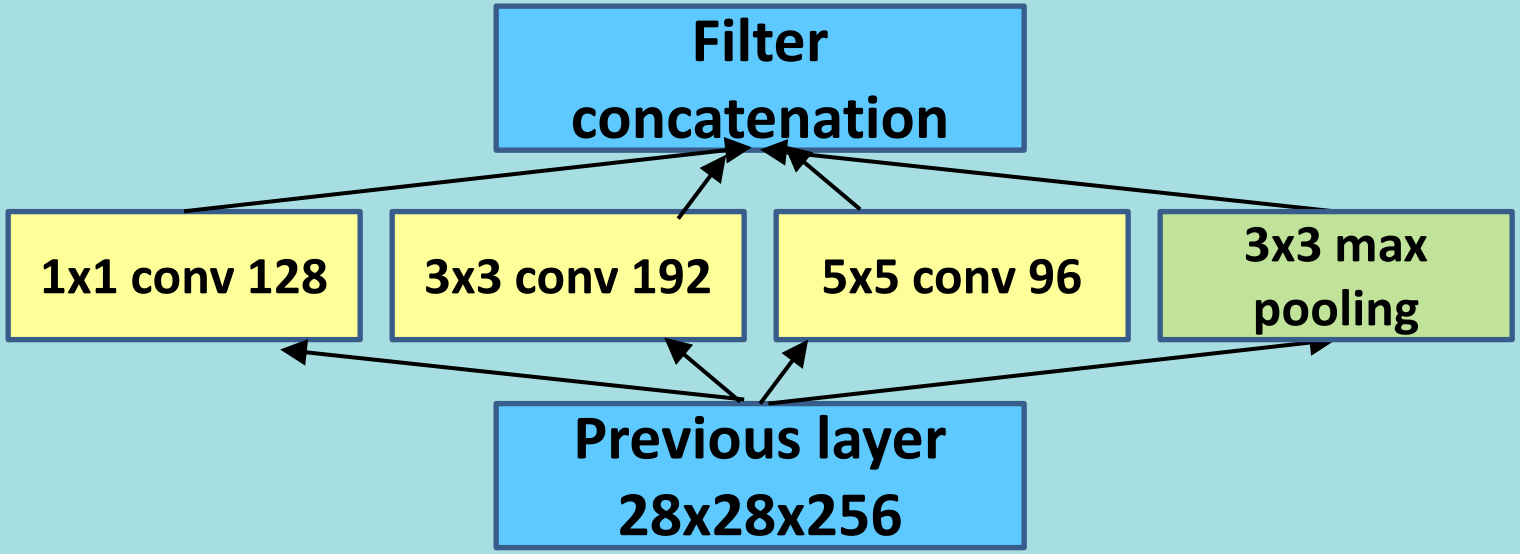
5x5 conv, 96: 28x28x96x5x5x256

Total: 854M ops



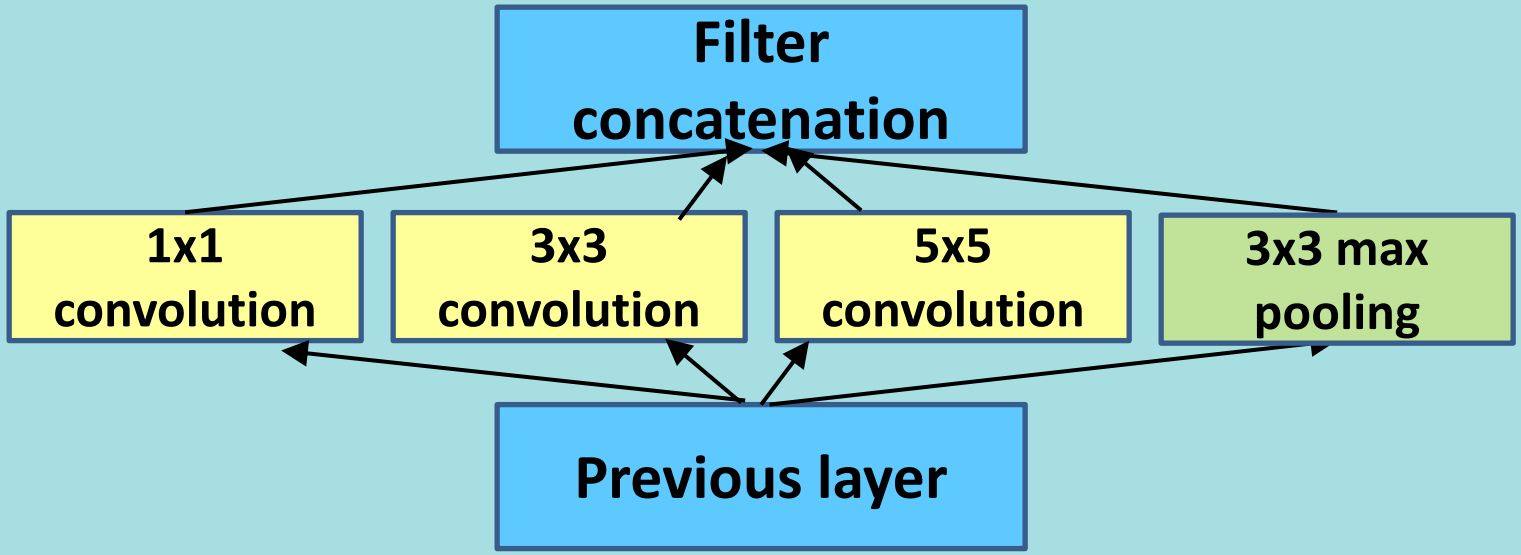
GoogleNet

- **Very expensive compute!**
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer.



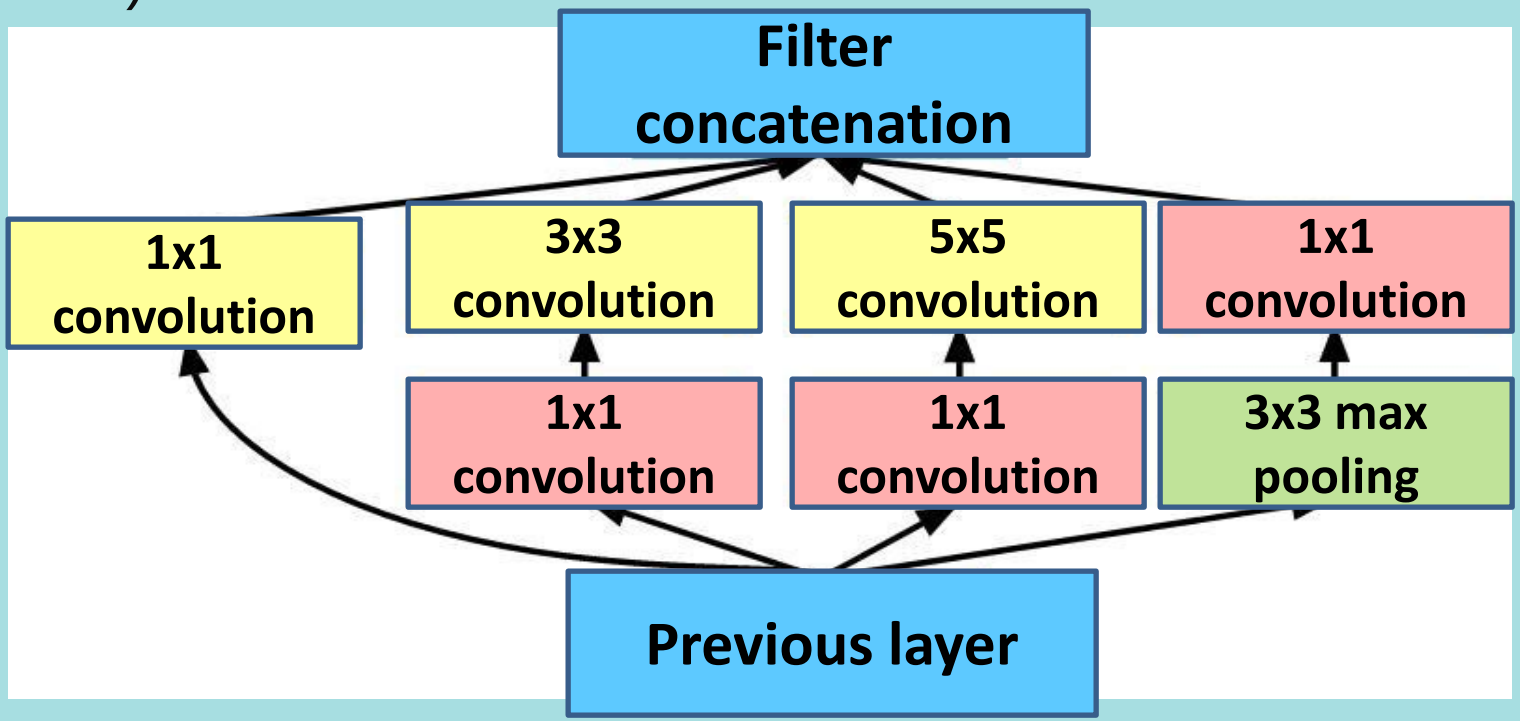
GoogleNet

- **Solution:** "bottleneck" layers that use 1x1 convolutions to reduce feature depth (from previous hour).

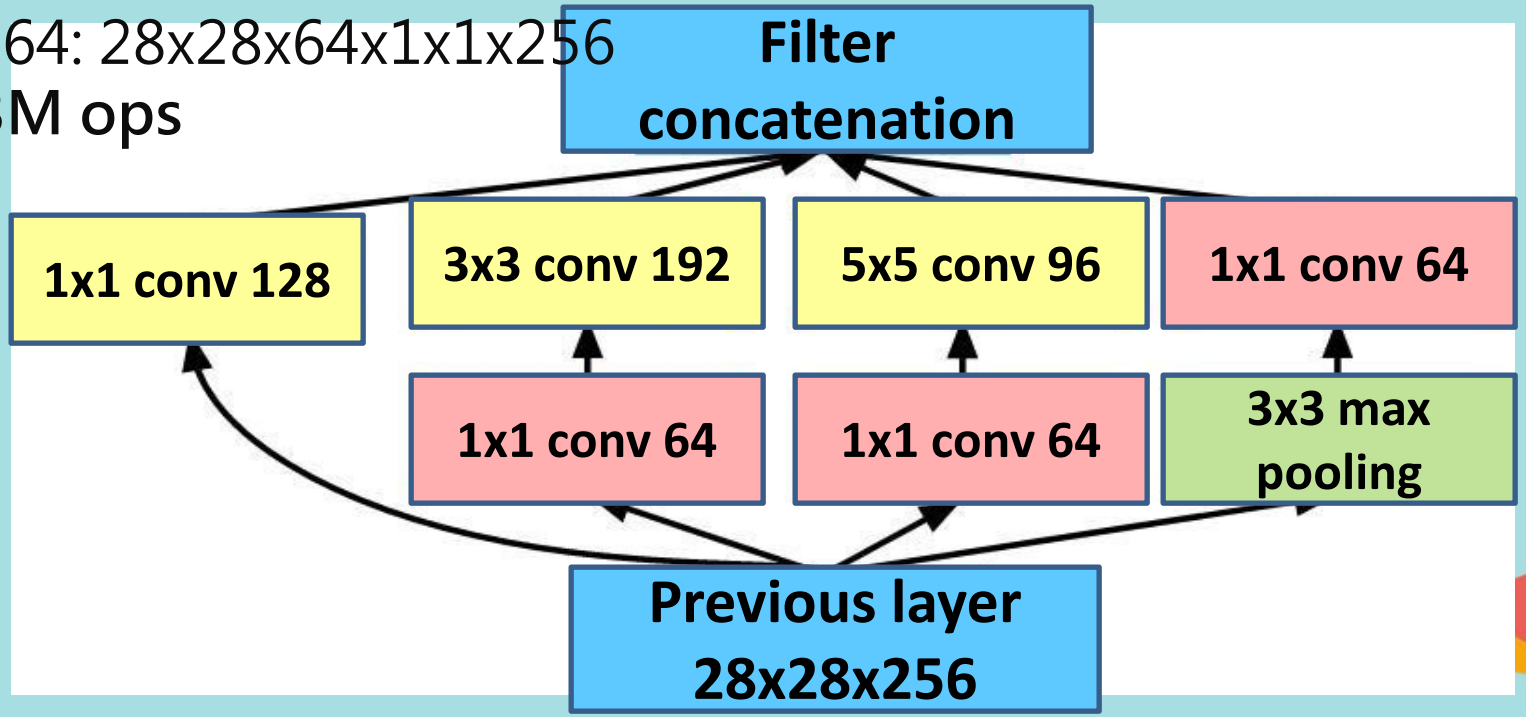


GoogleNet

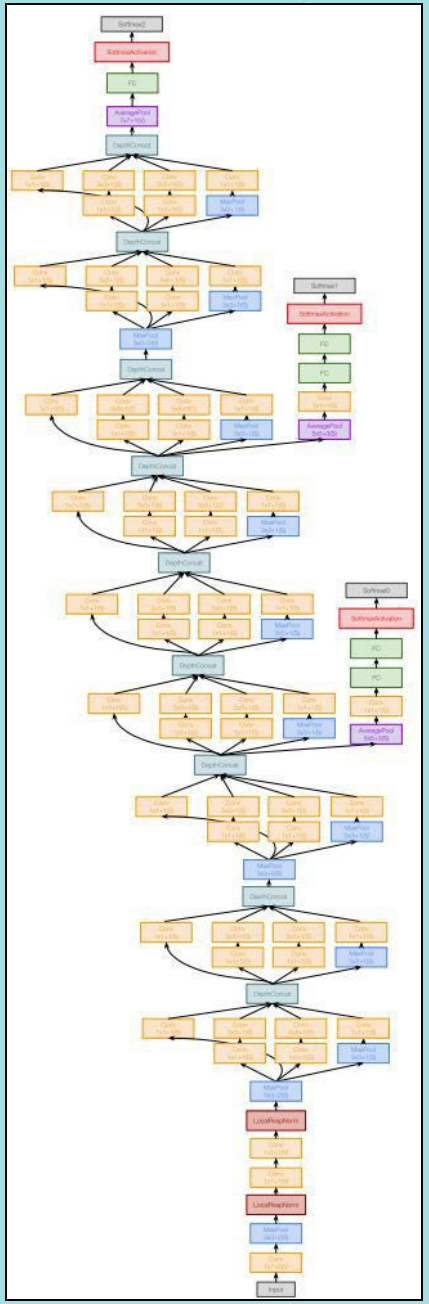
- **Solution:** "bottleneck" layers that use 1x1 convolutions to reduce feature depth (from previous hour).



- **Number of convolution operations:**
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 64: 28x28x64x1x1x256
  1x1 conv, 128: 28x28x128x1x1x256
  3x3 conv, 192: 28x28x192x3x3x64
  5x5 conv, 96: 28x28x96x5x5x264
  1x1 conv, 64: 28x28x64x1x1x256
  Total: 353M ops



- Compared to 854M ops for naive version



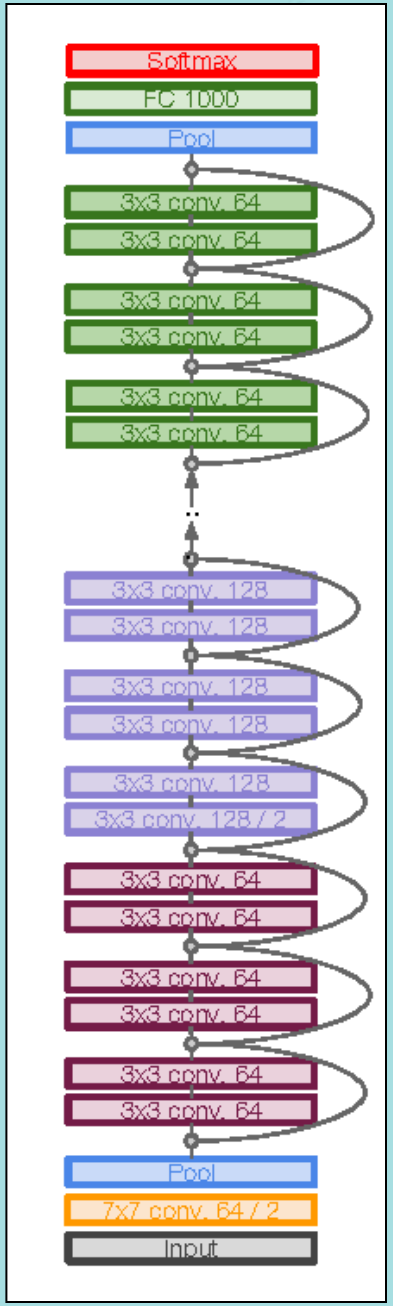
GoogleNet

Details/Retrospectives :

- Deeper networks, with computational efficiency
- 22 layers
- Efficient “Inception” module
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

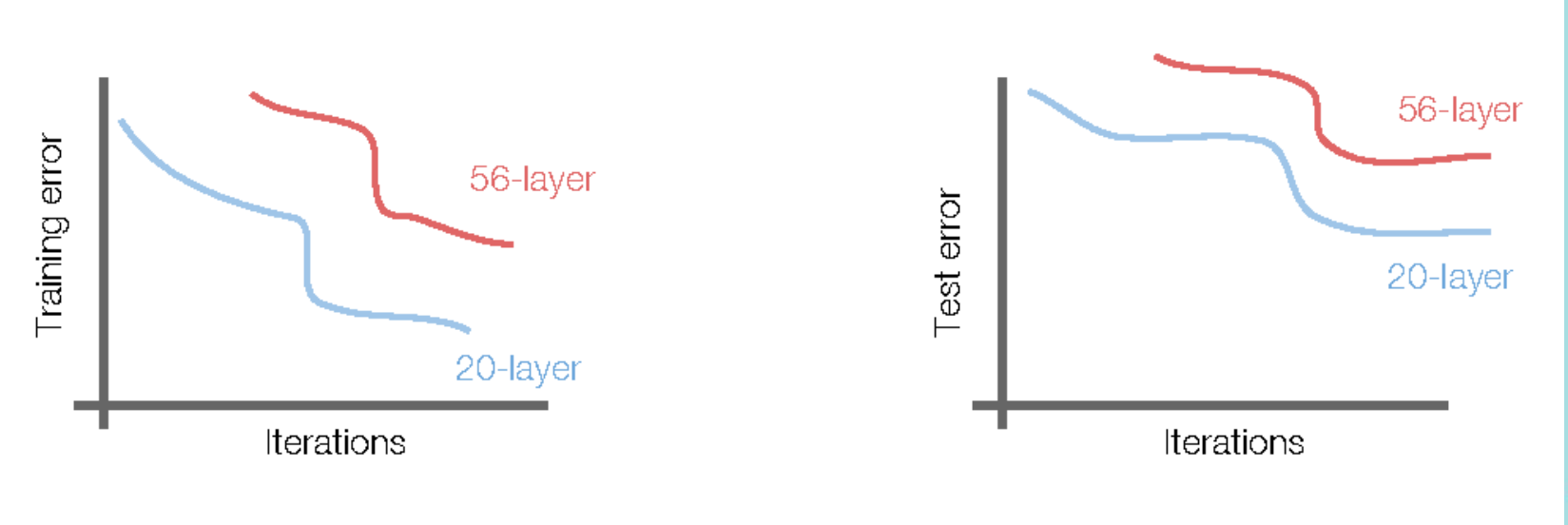


ResNet

- ILSVRC’15 classification winner (3.57% top 5 error, humans generally hover around a 5-10% error rate) Swept all classification and detection competitions in ILSVRC’15 and COCO’15!

ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?



- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!

ResNet

- **Hypothesis**: The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution**: Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.

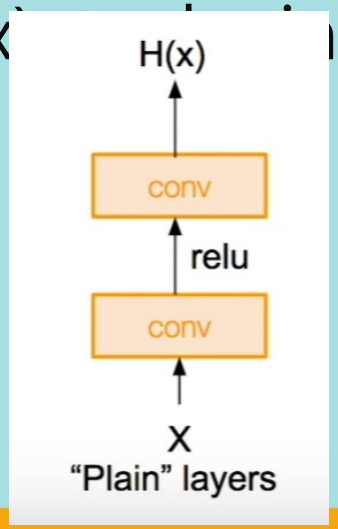
- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

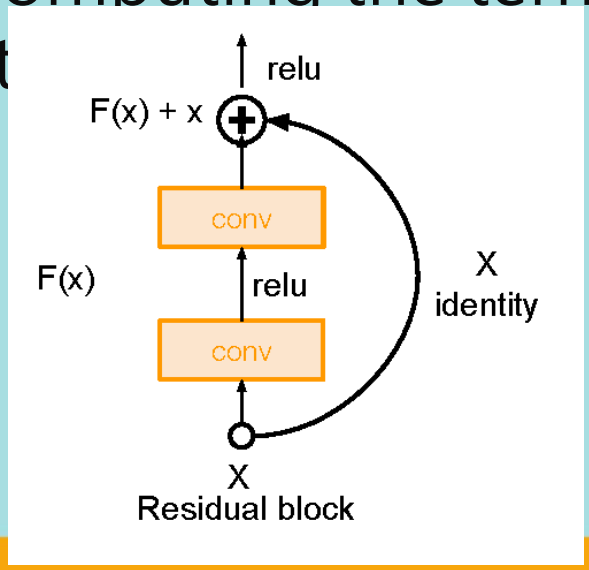
ResNet

Residual Block

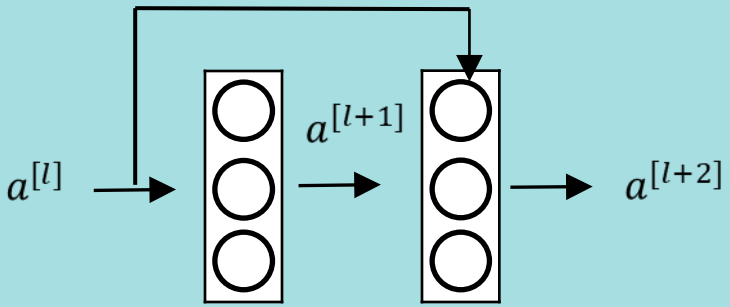
Input x goes through conv-relu-conv series and gives us F(x). That result is then added to the original input x. Let' s call that H(x) = F(x) + x.

In traditional CNNs, H(x) would just be equal to F(x). So, instead of just computing that transformation (straight from x to F(x)), we' re computing the term that we have to *add*, F(x





ResNet



Short cut/ skip connection

$a^{[l]}$ → **Linear** → **ReLU** → **Linear** → **ReLU** → $a^{[l+2]}$

$a^{[l+1]}$

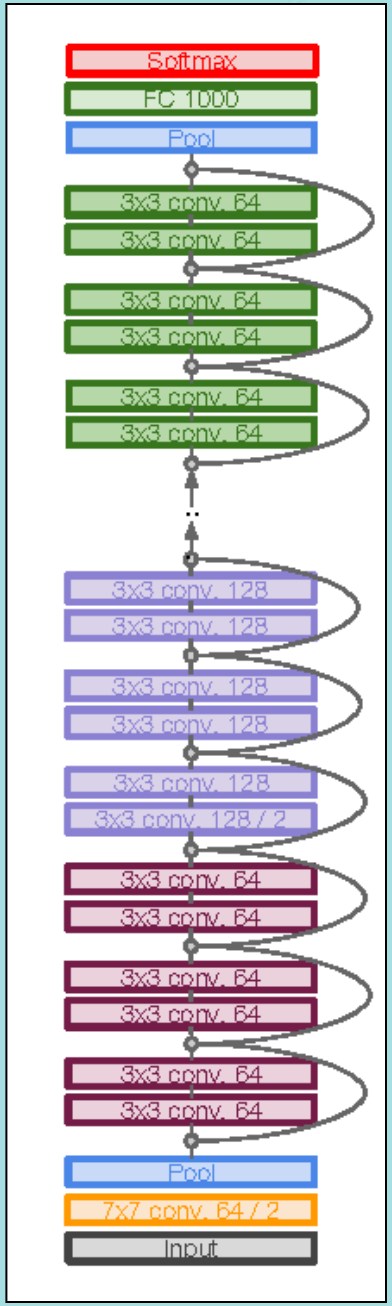
$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$$

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

$$a^{[l+2]} = g(z^{[l+2]})$$

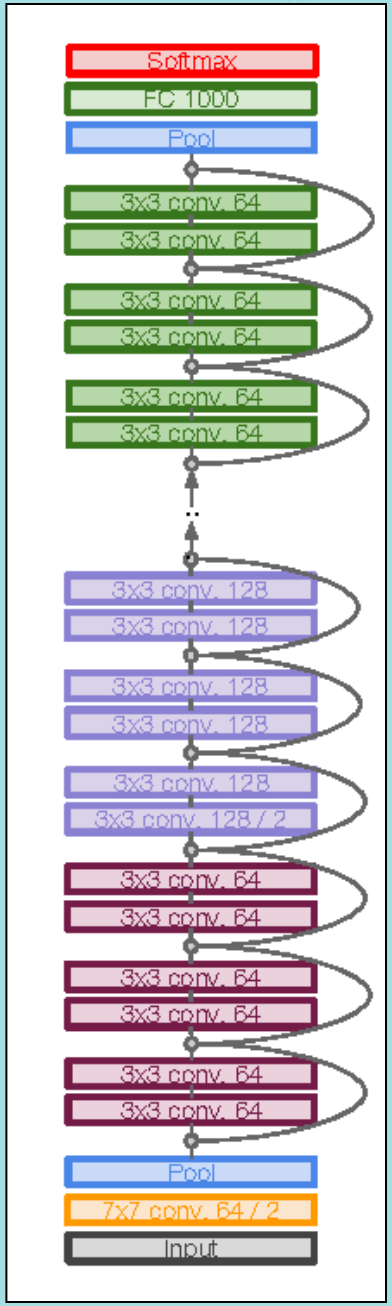
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]}) = g(W^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$



ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



ResNet

- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

Thanks!

Q&A