

深度學習教學小計畫

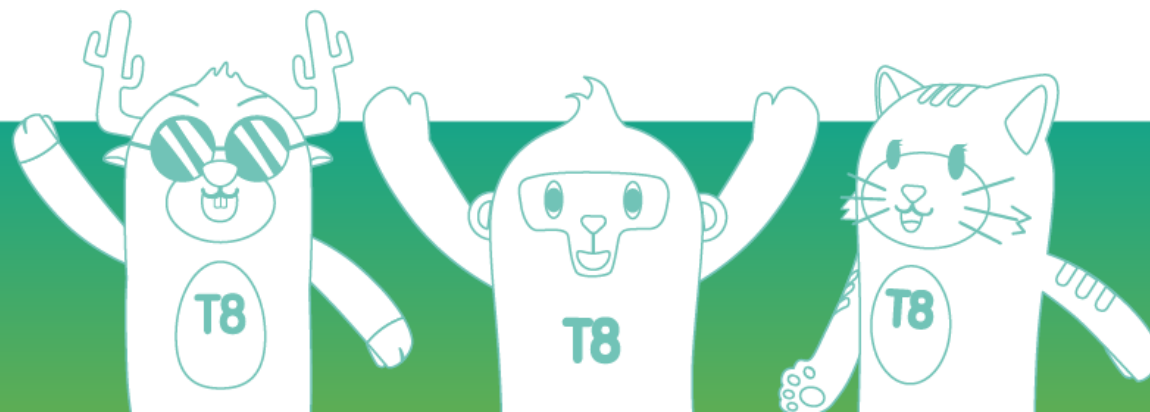
■ 授課講師 陳少君

■ 教材編寫 陳少君

緯育 *TibaMe*

即學・即戰・即就業

<https://www.tibame.com/>



學習目標：

- 了解教學方式與進度

Module 0.

教學目的方 式與進度

目的方式與進度

為何而學

機器學習利用正確有效的資料訓練模型，並利用驗證資料及測試資料調整其參數，持續改進模型，使其分類、分群、迴歸等演算法能被有效應用

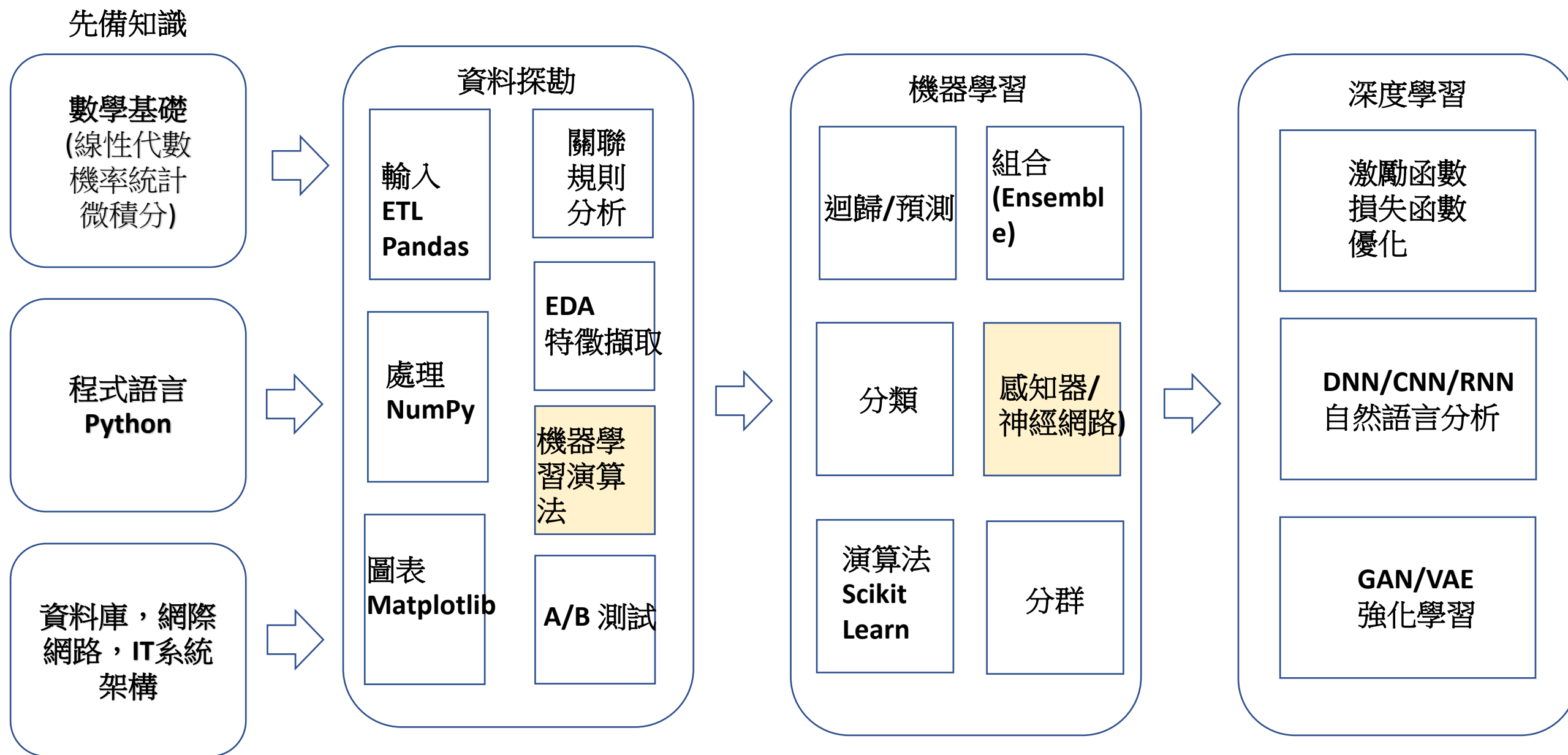
機器學習的一個分支：類神經網路，將成為未來深度學習的基礎。其他各演算法在各行各業也有其廣泛的應用領域。

承先啟後

先備知識：數學概念，Python/R 程式語言，資料探勘，機器學習

進階：深度學習之應用與演進

資料科學知識示意地圖



第一天(9月30日)：深度學習介紹

深度學習介紹

Tensorflow 2.0介紹

損失函數

優化(1)

第二天(10月7日)：神經網路，CNN

優化(2)

Tensorboard

CNN原理及應用

第三天(10月13日)：CNN應用，文字探勘

CNN應用：物件偵測

文字探勘

RNN介紹

第四天(10月21日)：LSTM/GRU，降維，AutoEncoder，風格遷移

LSTM/GRU

降維

AutoEncoder

風格遷移

第五天(10月28日)：增強式學習，Keras，複習
增強式學習 (Reinforcement Learning)
Keras
深度學習複習
(Bonus: Transformer/BERT)

2 小時

深度學習概論
Tensorflow



2 小時

類神經網路
激活函數



2 小時

損失函數
優化演算法

預習：CNN
作業：作業一

試解：

$$5x + 2y = 10$$

$$x + 4y = 11$$

機器學習不是在算參數權重(weight)嗎，把 x, y 當成 $w_1 w_2$ ，把方程式左邊的係數(5,2) (1,4)定為輸入($x_1 x_2$)，右邊(10,11)為向量 y ，轉換成感知器的公式：

$$5 * w_1 + 2 * w_2 = 10$$

$$1 * w_1 + 4 * w_2 = 11$$

$$\begin{bmatrix} 5 & 2 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} -10 \\ -11 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 2 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 10 \\ 11 \end{bmatrix}$$

如果我們故意不直接求解，而是用更多聯立方程式去逼近，比方 $2*w_1 + 2*w_2 = 7$ ，我們是否也可解出 (w_1, w_2)？這樣作其實就是訓練感知器，如果在感知器後頭接上非線性激勵函數(sigmoid / tanh / relu)，豈不形成類神經網路之一層？層層疊加就是深度學習網路之前饋(feed forward)部分
機器學習其實是個複雜的**函數映射**，線性代數適合**表述並計算出所需參數 w**

[看機器學習如何求解多元一次方程？ - 每日頭條 \(kknews.cc\)](http://kknews.cc)

兩大陣營各有千秋，簡單的說，TensorFlow 易懂好上手，Pytorch 適合做研究。TensorFlow 有 Tensorboard 的可視化功能，Pytorch 支持動態圖 (Dynamic Graph)。建議未來兩者都學。

以實例比較其不同：

[PyTorch vs TensorFlow — spotting the difference | by Kirill Dubovikov | Towards Data Science](#)

Pytorch 學習網站：

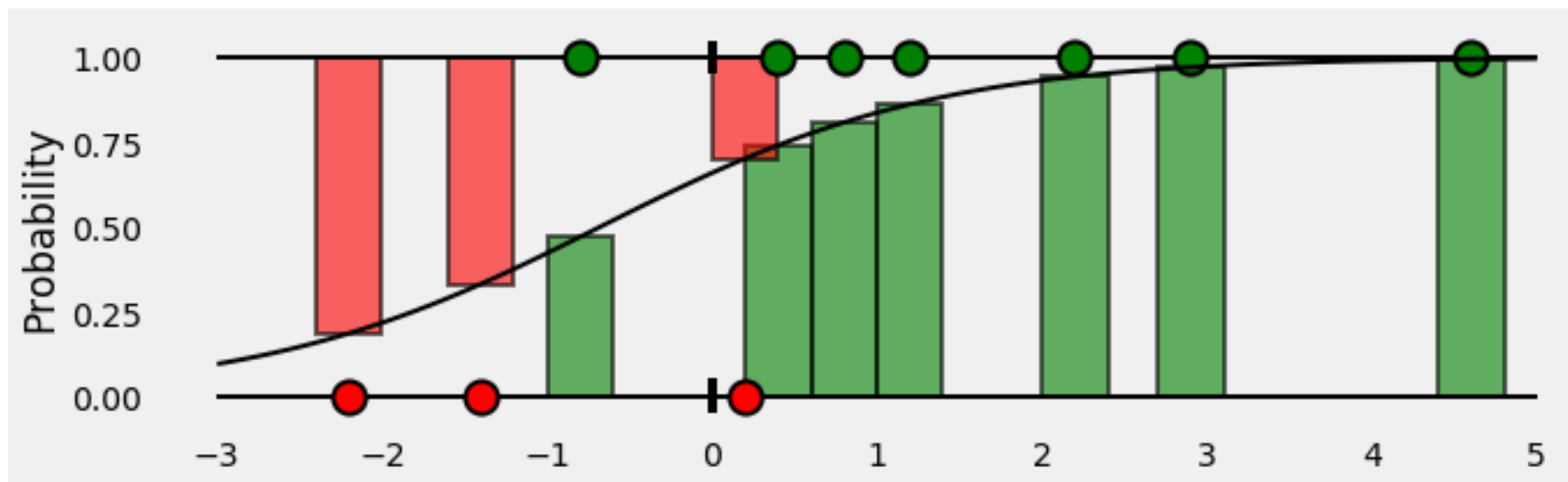
[Learning PyTorch with Examples — PyTorch Tutorials 1.9.1+cu102 documentation](#)

Class	範例/說明
Tensor	運算之主角
Operation	tf.add等運算元
Graph	Tensor的運算圖
Session	執行graph
Variable	像tensor之參數儲存處
Optimizer	以minimize method降低損失
Estimator	逐漸被淘汰之class較難開發卻適於軟體工程
Dataset	處理與載入資料
Iterator	將Dataset依序輸入模型
Saver	儲存參數供繼續訓練

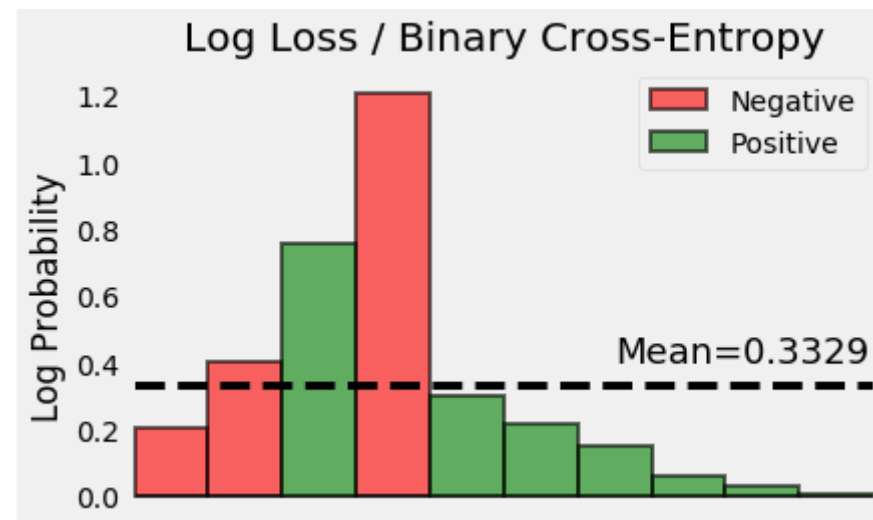
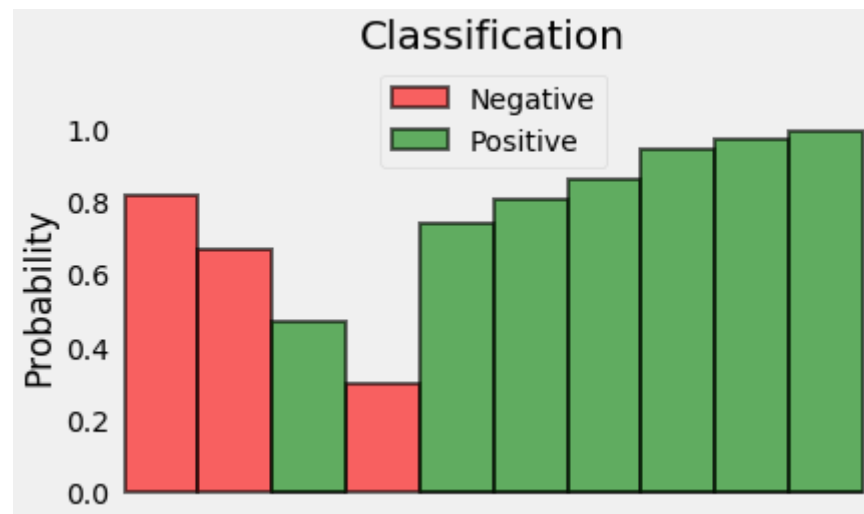
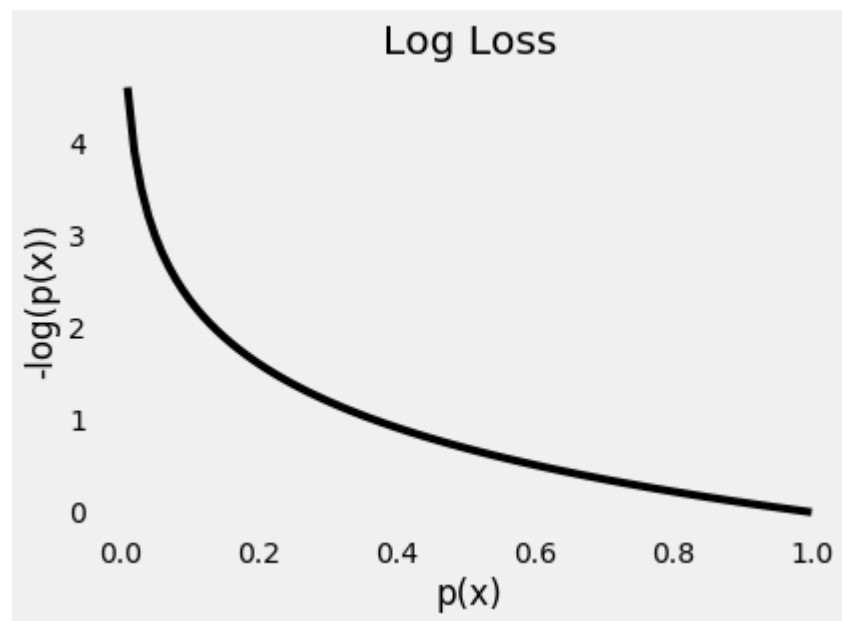
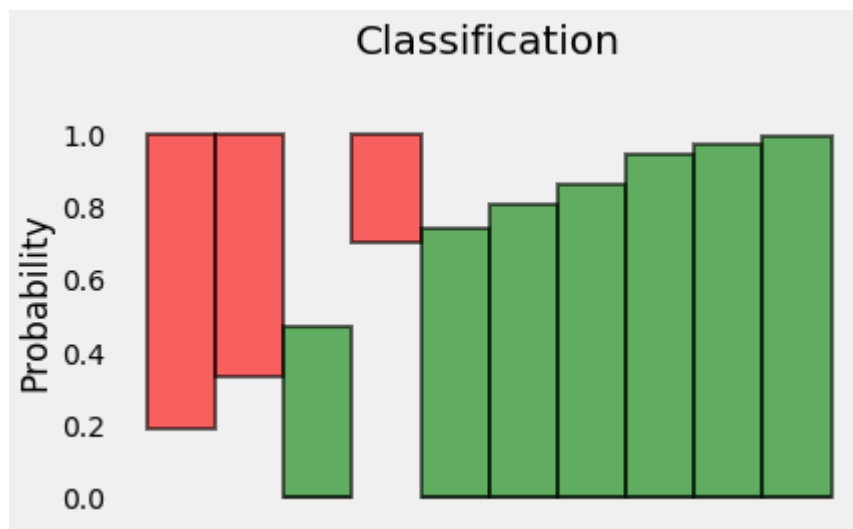
[The 10 Most Important TensorFlow Classes - dummies](#)

Log Loss (Cross Entropy)計算

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$



Log Loss (Cross Entropy) 計算



[Understanding binary cross-entropy / log loss: a visual explanation | by Daniel Godoy | Towards Data Science](#)

大致分為

Probability Loss:

Binary Cross Entropy : 兩類分類

Categorical Cross Entropy : 多類分類，目標值(label)為獨熱(one hot)編碼

Sparse Categorical Cross Entropy:(互斥)多類分類，目標值為整數編碼

Poisson: 取張量元素平均值

KL Divergence: 取機率對數加總之負數

Regression Loss

MSE : Mean Squared Error

MAE : Mean Absolute Error

Cosine Similarity : 向量之角度

Huber : 小的用平方大的用線性

[Tensorflow Loss Functions | Loss Function in Tensorflow \(analyticsvidhya.com\)](https://www.analyticsvidhya.com/blog/2016/08/complete-guide-to-loss-function-in-deep-learning/)

BinaryCrossentropy Loss Function :

`y_true = [0,1,0,0]`

`y_pred = [-18.6, 0.51, 2.94, -12.8]` #這是用 logit, API 可決定

SparseCategoricalCrossentropy Loss Function :

`y_true = [1,2]`

`y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]` #Probability

CategoricalCrossentropy Loss Function :

`y_true = [[0,1,0], [0,0,1]]`

`y_pred = [[0.05, 0.95, 0], [0.1, 0.8, 0.1]]` #Probability

損失函數需要知道 y_{pred} 是否為 Logit，何謂 Logit？(機率是 Probability(P))

Odds (賠率) = Probability / (1 - Probability)

Logit = $\ln(\text{Odds}) = \ln(P / (1 - P))$

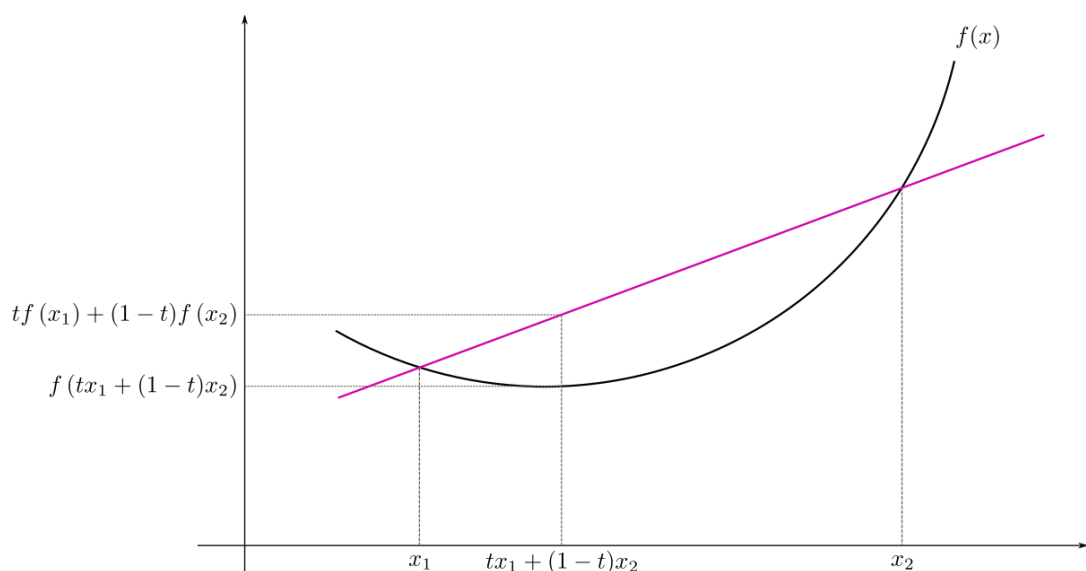
\ln 是底為歐拉數的自然對數

Logit 函數可將 P 轉成 Logit， $\text{Logit}(0.5) = 0$ ，所以如果 Logit 值 > 0 ，機率 > 0.5
反函數就是 Sigmoid

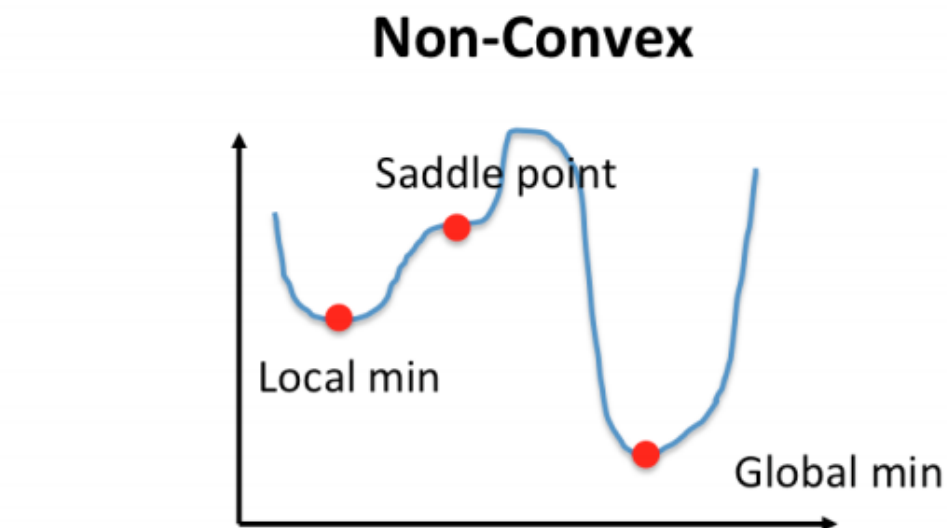
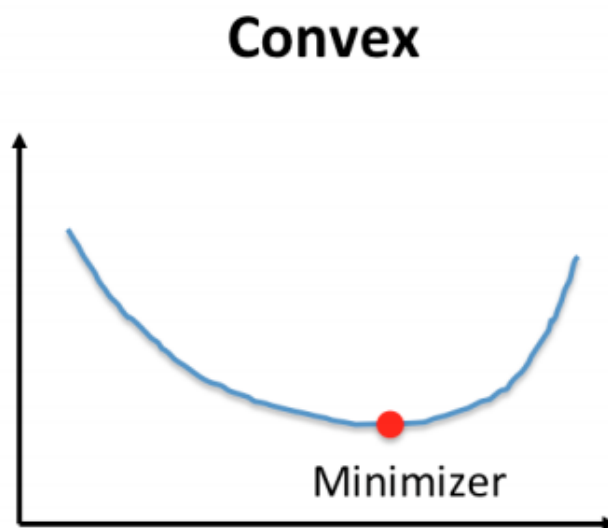
[LOGIT function calculator and graph \(medcalc.org\)](http://medcalc.org)

任何一條穿過凸函數曲線的直線，在相交兩點間，直線 y 之值永遠較凸函數 $f(x)$ 之值大。

非凸函數之優化，可能造成本地優化(local optimum) 或鞍點(saddle point)



wikipedia.com



printerest.com

- **Adam** (Adaptive Moment Estimation) is a replacement optimization algorithm for stochastic gradient descent (**SGD**, fixed learning rate) for training deep learning models.
- Adam combines the best properties of the **AdaGrad** (每個參數都有 learning rate) and **RMSProp** (取每個參數 average gradient 當個別 learning rate) algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
- Adam is relatively easy to configure where the default configuration parameters do well on most problems.
- 在Tensorflow 的參數設置：learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08.

[Gentle Introduction to the Adam Optimization Algorithm for Deep Learning \(machinelearningmastery.com\)](https://machinelearningmastery.com/gentle-introduction-to-the-adam-optimization-algorithm-for-deep-learning/)

2 小時

優化演算法(二)



2 小時

Tensorboard
CNN介紹



2 小時

CNN原理架
構與應用

預習：Yolo, 文字探勘
作業：作業二

[TensorFlow 2 quickstart for beginners](#) | [TensorFlow Core](#)

問題點：

輸入是甚麼：[MNIST database – Wikipedia](#)

Load Data 之後是甚麼？2 tuples of NumPy NDArrays for training and testing.

Sequential 是甚麼？A Fully Connected Neural Network function returning a model.

Data 進入 Dense 時是甚麼？NumPy NDArray. [tf.keras.layers.InputLayer](#) | [TensorFlow Core v2.6.0](#)

輸出的結果為何是logit？SoftMax 之後才變成probability？

Data 在 Dense NN 是 Tensor.

Tensor 是甚麼？是不是TensorFlow 才有？[Introduction to Tensors](#) | [TensorFlow Core](#)

Tensor 和 Numpy.NDArray 的關係：Tensor 有更多attributes，可利用 GPU/TPU

Model.compile 有哪些 metrics？

[Module: tf.keras.metrics](#) | [TensorFlow Core v2.6.0](#)

- 由Loss function 得知當時的 loss value L
- 從 Loss Function 對 權重 w 的一次偏微分 $\partial L / \partial w$ ，代入當時的 x, y 得到 梯度 Gradient $G(w)$
- 將原 w 減去 $G(w)$ 得到新的 w ，這就是Descent
- 因為 w 改變，如果 $y = wx + b$ ，斜率是 w ，則下個 x 出現時， y 的算法會被調整
- 直到 w 固定下來，回歸直線才固定

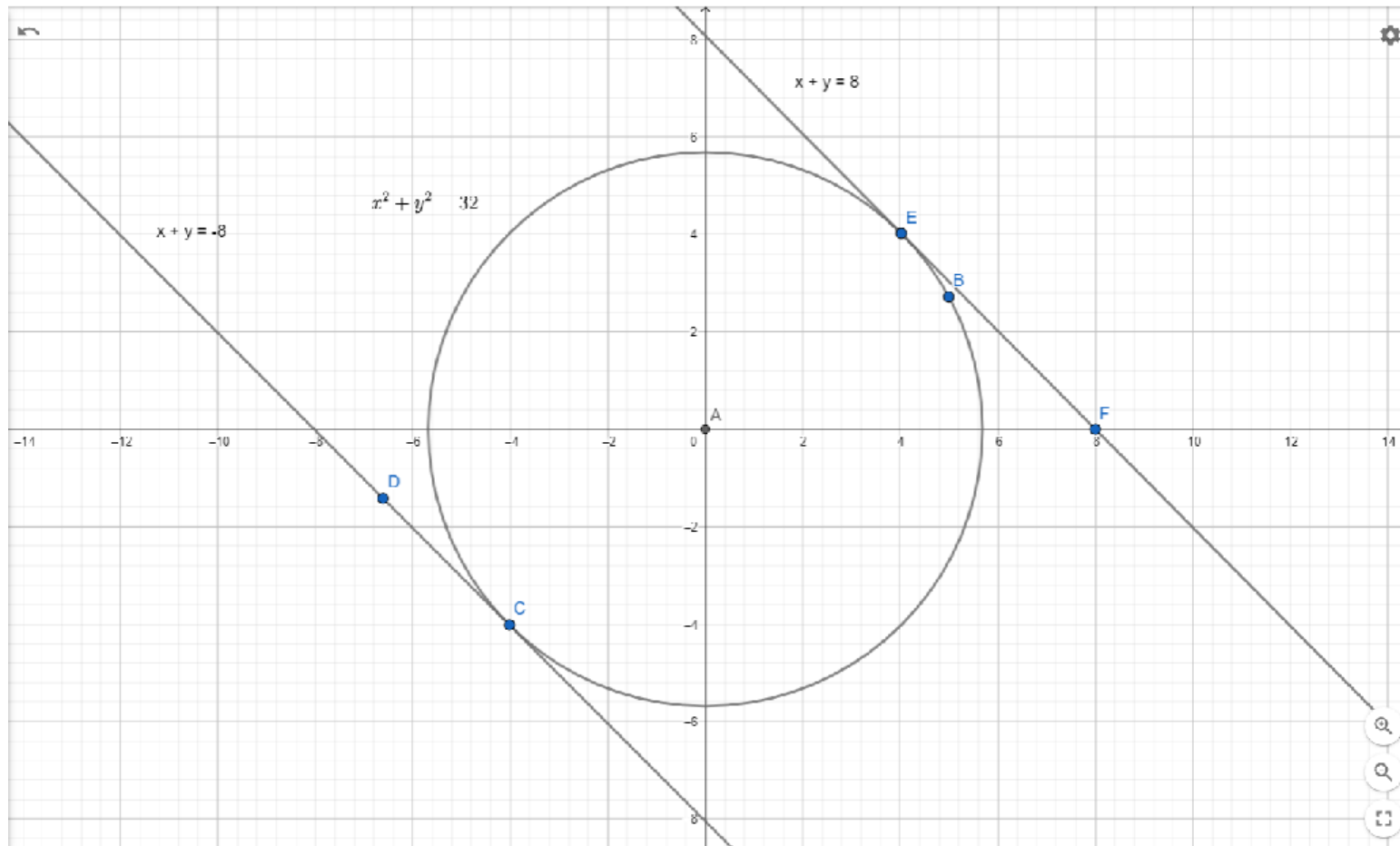
完整觀念

[Gradient Descent with Linear Regression | Kaggle](#)

基本觀念

[iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 \(ithome.com.tw\)](#)

Lagrange Multiplier



要求 $f(x,y) = x + y$
限制條件是 $h(x,y) = x^2 + y^2 - 32 = 0$

$L(x, y, \lambda) = f(x,y) + \lambda h(x,y)$
對 x, y, λ 分別取偏微分

$$L = x + y + \lambda(x^2 + y^2 - 32)$$

$$\begin{aligned}\frac{\partial L}{\partial x} &= 1 + 2\lambda x = 0 \\ \frac{\partial L}{\partial y} &= 1 + 2\lambda y = 0 \\ \frac{\partial L}{\partial \lambda} &= x^2 + y^2 - 32 = 0\end{aligned}$$

$$\begin{aligned}x &= -1/(2\lambda) \\ y &= -1/(2\lambda) \\ 1/(4\lambda^2) + 1/(4\lambda^2) - 32 &= 0 \\ 1/(4\lambda^2) &= 16 \\ \lambda^2 &= 1/(16 * 4) = 1/64 \\ \lambda &= 1/8 \\ X &= 4 \\ Y &= 4\end{aligned}$$

[A Gentle Introduction To Method Of Lagrange Multipliers \(machinelearningmastery.com\)](http://machinelearningmastery.com)

[Wolfram|Alpha Widgets: "Lagrange Multipliers" - Free Mathematics Widget \(wolframalpha.com\)](https://www.wolframalpha.com/widget)

目的：經由調整Loss Function (增加 L1, L2等)達到降低過適(Overfitting)。

L1: (LASSO) $\|\mathbf{w}\|_1 = |w_1| + |w_2| + \dots + |w_N|$

L2: (RIDGE) $\|\mathbf{w}\|_2 = (|w_1|^2 + |w_2|^2 + \dots + |w_N|^2)^{\frac{1}{2}}$

$$\hat{y} = w_1x_1 + w_2x_2 + \dots + w_Nx_N + b$$

Logistic Regression Loss Function: $L(y_{\text{hat}}, y) = y \log y_{\text{hat}} + (1 - y) \log(1 - y_{\text{hat}})$

$$Loss = Error(y, \hat{y})$$

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N |w_i|$$

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^N w_i^2$$

可參考：

[Regularization in Machine Learning - GeeksforGeeks](#)

Scaler	純量的變化：loss, accuracy etc
Graph	靜態張量流動圖
Distribution	權重(非純量)之分布 Kernel/bias
Histogram	直方圖表示之(非純量) Kernel/bias etc
Image	使用的影像

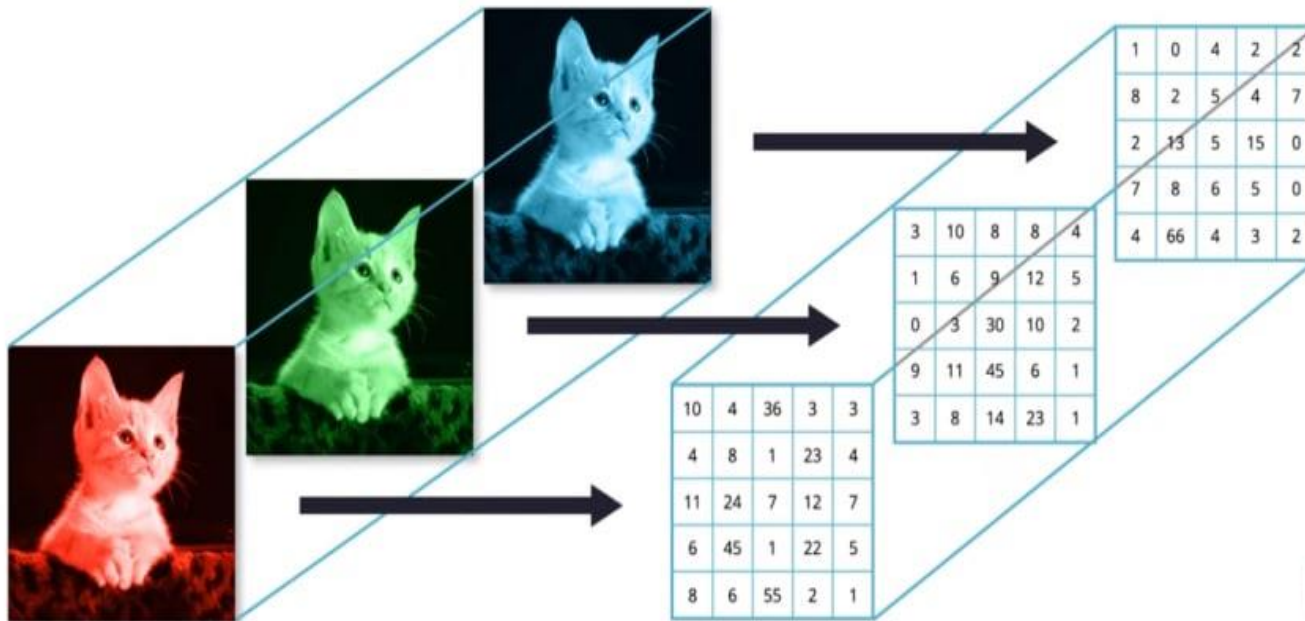
[Jupyter Notebook Viewer](#): Tensorboard explanations

Random Number Functions

import	function	parameters	returned type	returned data type	notes
numpy	random.rand	size=shape #(d1, d2)	Array of shape	Float [0.0,1.0)	From uniform distribution
random/numpy	randint random.randint	start, end #(0,10)	Integer	Integer between [start, end]	numpy ver. has shape. from discrete uniform distribution
numpy	random.random	size=shape #((d1,d2))	Array of shape	Float [0.0,1.0)	From uniform distribution
numpy	random.randn	size=shape #(d1, d2)	Array of shape	Normally distributed float number	From standard normal distribution
numpy	random.normal	mean, stdev #(d1, d2)	Array of shape	Normally distributed float	From normal distribution

Colored and Gray Images

RGB Image

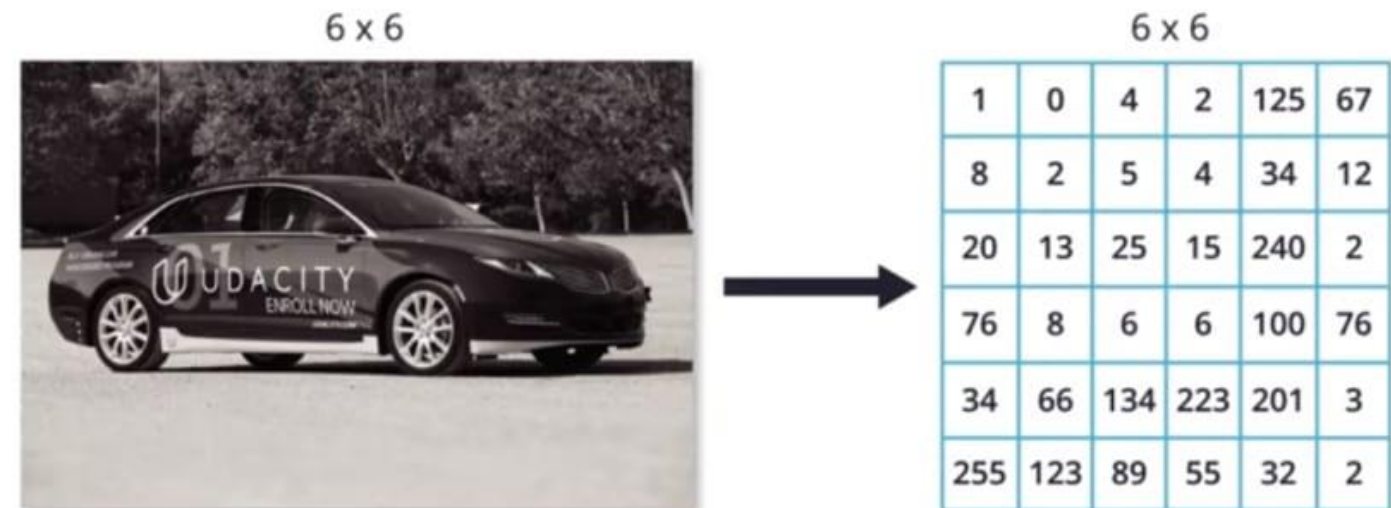


3D Array of shape (2,2,3):
`numpy.array([[[1,2,3],
 [3,4,5],
 [[5,6,7],
 [7,8,9]]])`

3D Array of shape (2,2,1):
`numpy.array([[[1],[2]],
 [[3],[4]]))`

2D Array of shape (2,2):
`numpy.array([[1,2],
 [3,4]]))`

Grayscale Image



直接在瀏覽器上展示CNN的各功能

CNN Explainer Introduction (0:00-0:22)

Overview (0:27-0:37)

Convolutional Elastic Explanation View (0:37-0:46)

Convolutional, ReLU, and Pooling Interactive Formula Views (0:46-1:21)

Flatten Elastic Explanation View (1:22-1:41)

Softmax Interactive Formula View (1:41-2:02)

Engaging Learning Experience: Understanding Classification (2:06-2:28)

Interactive Tutorial Article (2:29-2:54)

[瀏覽器上展示CNN說明視頻](#)

互動網頁：

[CNN Explainer \(poloclub.github.io\)](https://poloclub.github.io/CNN-Explainer/)

Same and Valid Padding

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+



Bias = 1

+ 1 = -25

Output

-25				...
				...
				...
				...
...

Padding = 'SAME'
Input/Output Same size

Padding = 'Valid'
No Padding

[Same and Valid Padding](#)

1 x 1 Convolution

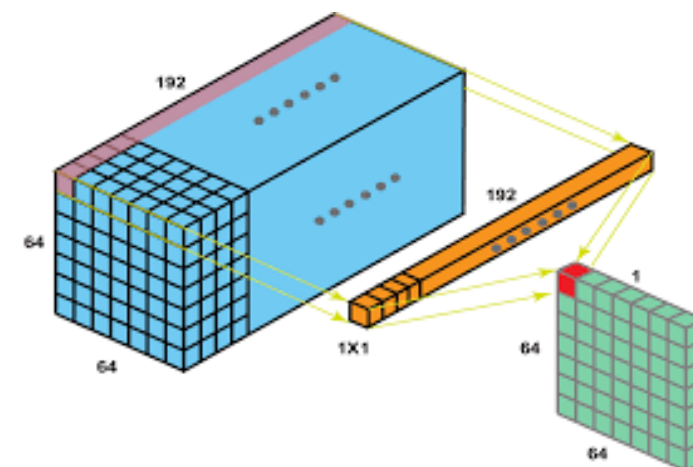
背景: 每個 FILTER 產生一個 CHANNEL，太多 LAYERS 產生太多 CHANNEL 造成 運算資源浪費且減慢速度，1 x 1 FILTER 可解決此問題

做法：1 x 1 FILTER 跨越所有 CHANNELS 做 CONVOLUTION，並將其映射 (PROJECT) 為單一 CHANNEL。所以是一個有效的改變(增加或減少)CHANNEL 數目的工具。

用途：常用在GOOGLE INCEPTION/RESNET 環境中

RESNET: match channel numbers between input and output

INCEPTION: reduce channel numbers



$$\#parameters = \#CH(I) * \#CH(O) + \#CH(O)$$

#CH : number of channels

[A Gentle Introduction to 1x1 Convolutions to Manage Model Complexity \(machinelearningmastery.com\)](https://machinelearningmastery.com/a-gentle-introduction-to-1x1-convolutions-to-manage-model-complexity/)

2 小時

CNN物件偵測
(YOLO)



2 小時

期中複習
文字探勘

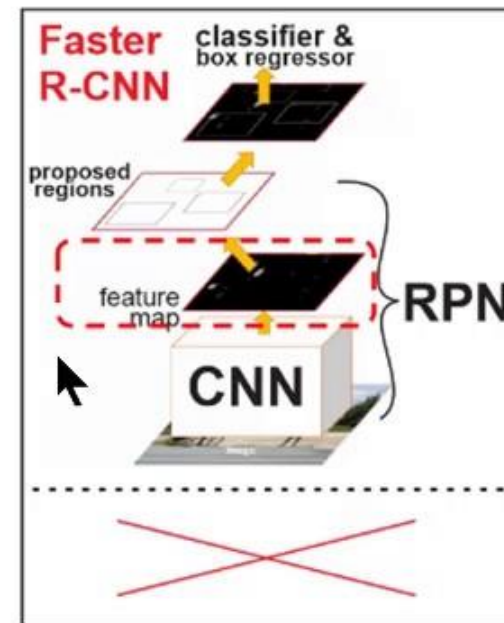
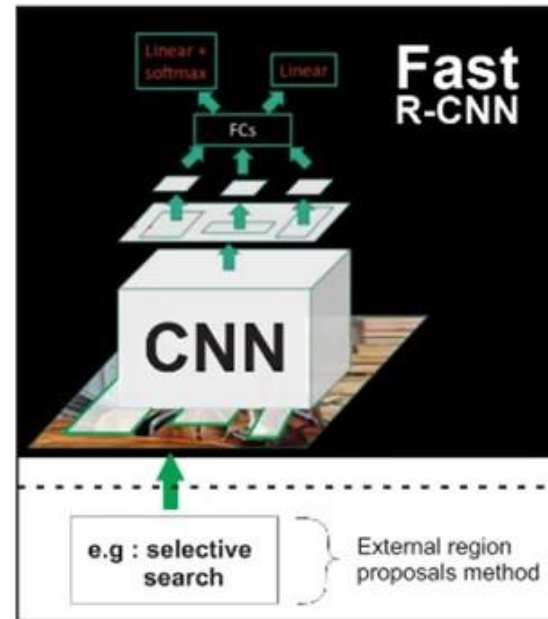
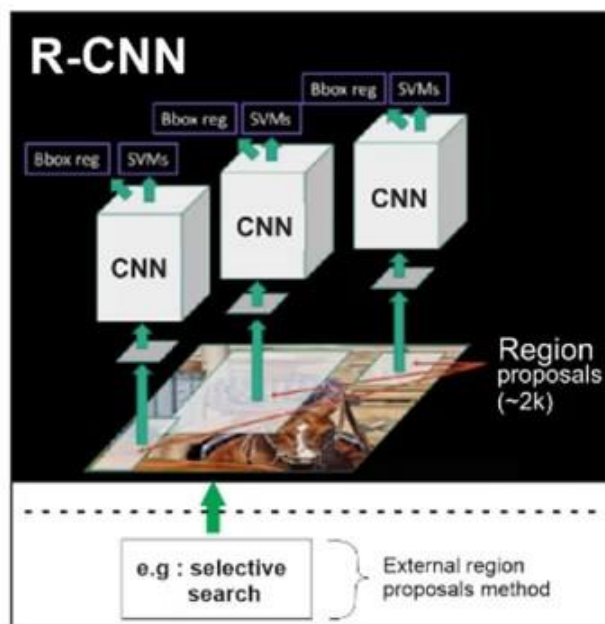


2 小時

文字探勘
CBOW
SKIPGRAM

預習：LSTM/GRU, 降維, AutoEncoder
作業：作業三

1. Introduction#02



	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

* Stanford lecture notes on CNN by Fei Fei Li and Andrej Karpathy

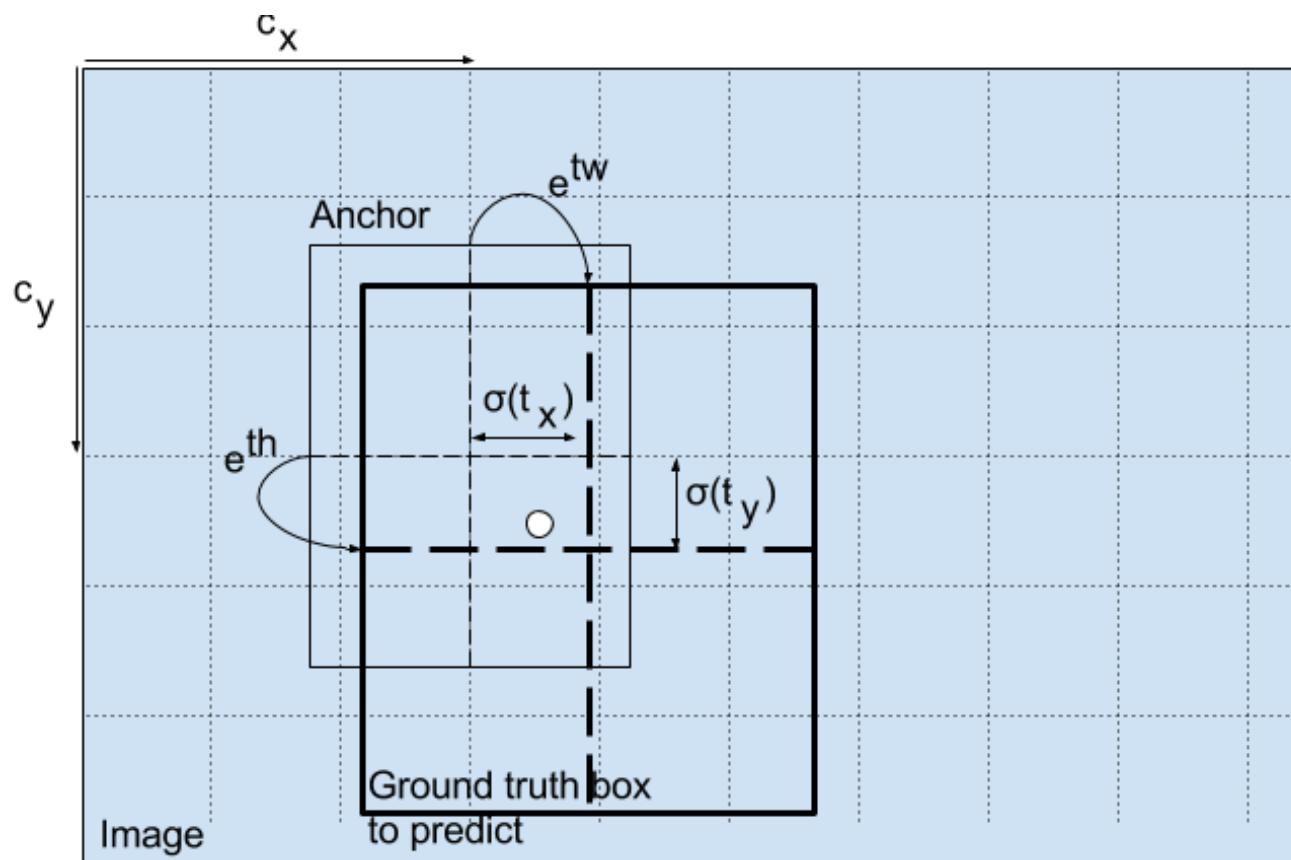
5

By Ardian Umam

<https://jhui.github.io/2017/03/15/Fast-R-CNN-and-Faster-R-CNN>

Why Anchor Box?

YOLO V3 的每個GRID CELL 都有 3x3個 以此CELL質心為中心的錨點框 (ANCHOR BOX)。利用他們預測此GRID CELL所負責的所有QUALIFIED BOUNDING BOXES。大中小加起來理論上有 $((13 \times 13) + (26 \times 26) + (52 \times 52)) \times 3 = 10647$ 個候選框



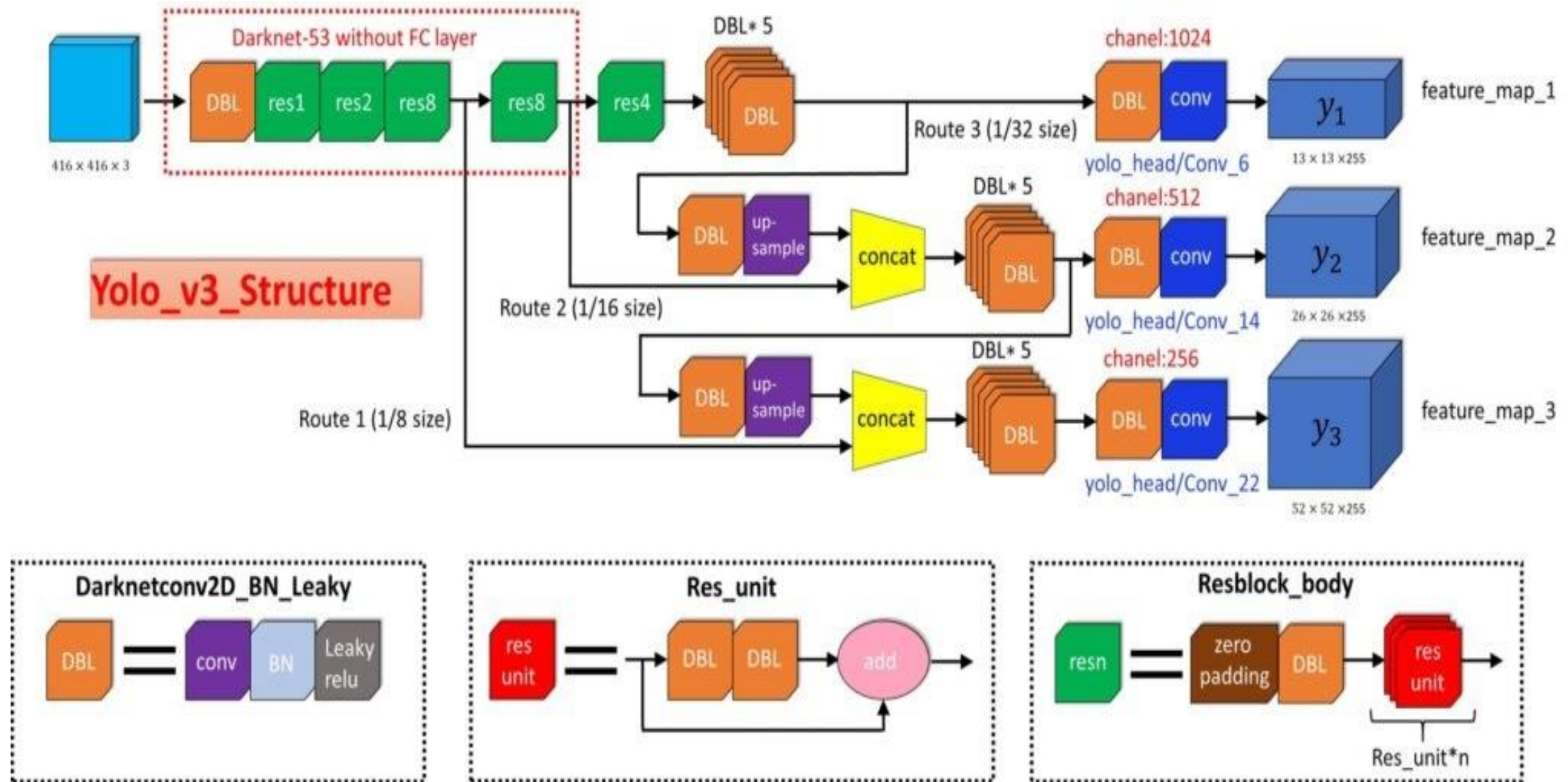
$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

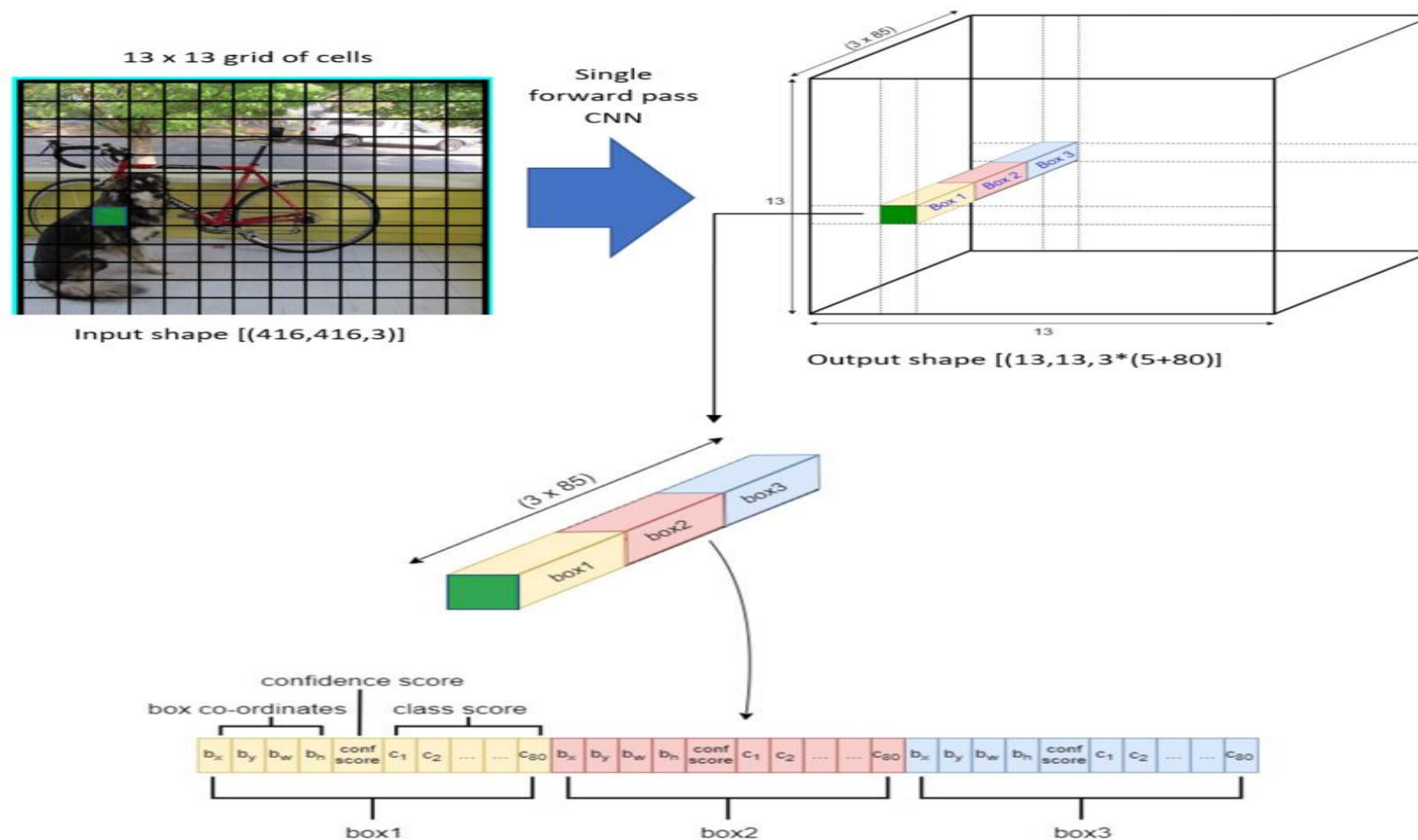
$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

Yolo v3 Prediction



Yolo v3 data structure



<https://machinelearning.space.com/yolov3-tensorflow-2-part-1/>

Yolo loss function

Regression
loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence
loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2$$

Classification
loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Input : $\mathcal{B} = \{b_1, \dots, b_N\}$, $\mathcal{S} = \{s_1, \dots, s_N\}$, N_t
 \mathcal{B} is the list of initial detection boxes
 \mathcal{S} contains corresponding detection scores
 N_t is the NMS threshold

begin

$\mathcal{D} \leftarrow \{\}$

while $\mathcal{B} \neq \text{empty}$ **do**

$m \leftarrow \text{argmax } \mathcal{S}$

$\mathcal{M} \leftarrow b_m$

$\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{M}; \mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$

for b_i in \mathcal{B} **do**

if $\text{iou}(\mathcal{M}, b_i) \geq N_t$ **then**

| $\mathcal{B} \leftarrow \mathcal{B} - b_i; \mathcal{S} \leftarrow \mathcal{S} - s_i$

end

NMS

end

end

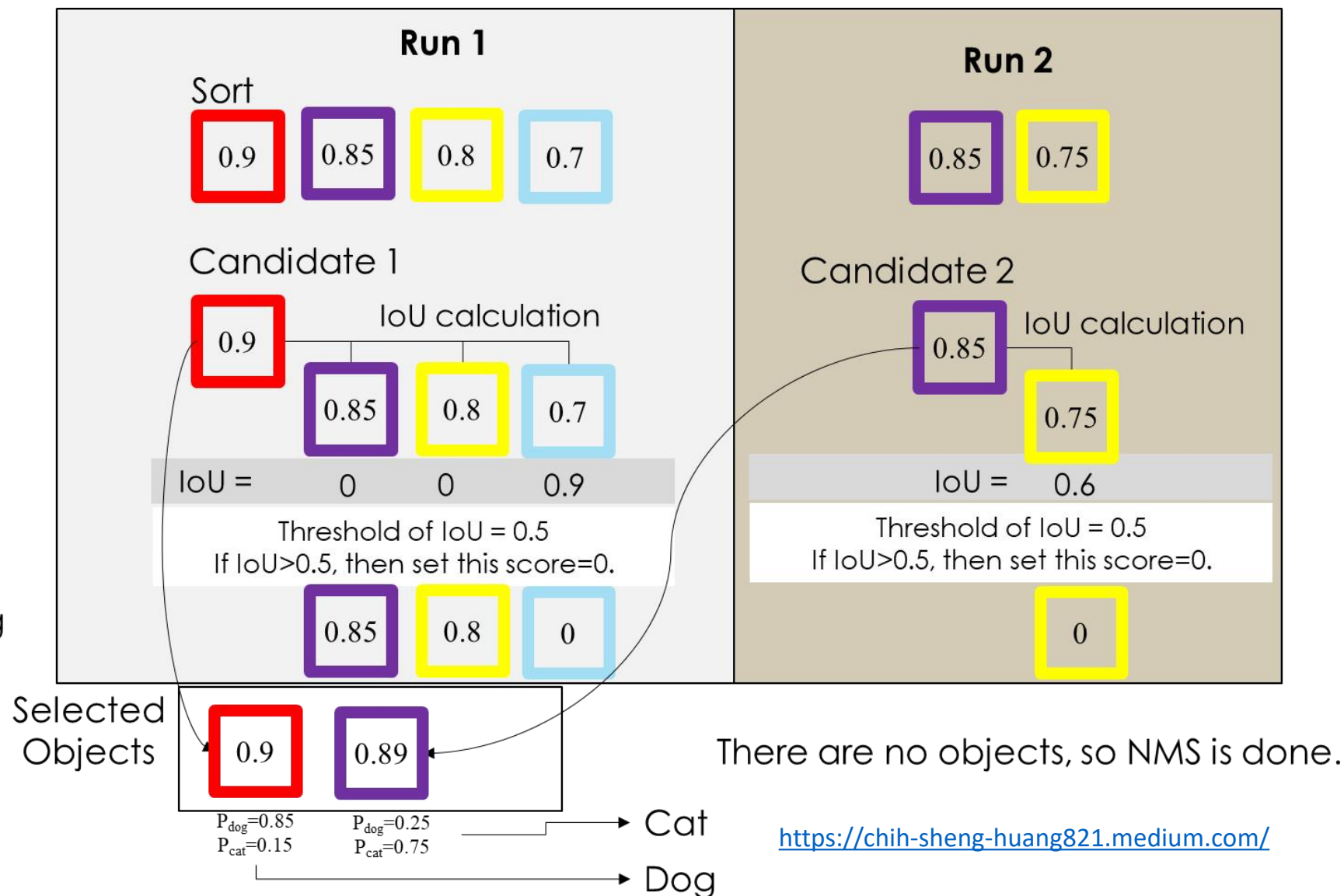
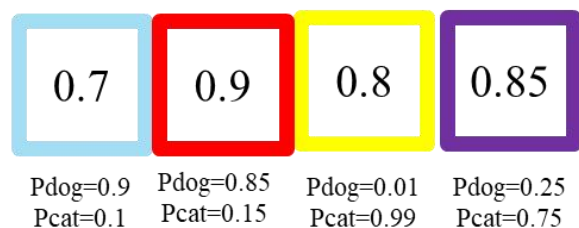
return \mathcal{D}, \mathcal{S}

end

NMS: 2 classes, Threshold 0.5



Score for 4 candidates



<https://chih-sheng-huang821.medium.com/>

Embedding: 文字轉向量，為什麼要轉向量？幾度空間的向量？

為了要計算距離，字與字，句與句，段與段，文與文。

有了這些，就可以用向量距離，以三角幾何學計算相似度。(GenSim = Generate Similarity)

字彙的空間，叫做文句集 (Corpus)，和當時的應用範圍有關，並非全世界所有的字。

轉向量的方法？One Hot Encoding, Label Encoding (Index), Word2Vec

Word2Vec最精簡(dense)，可指定向量維度(dim)，且向量最接近人類賦予的意義(Why?)。

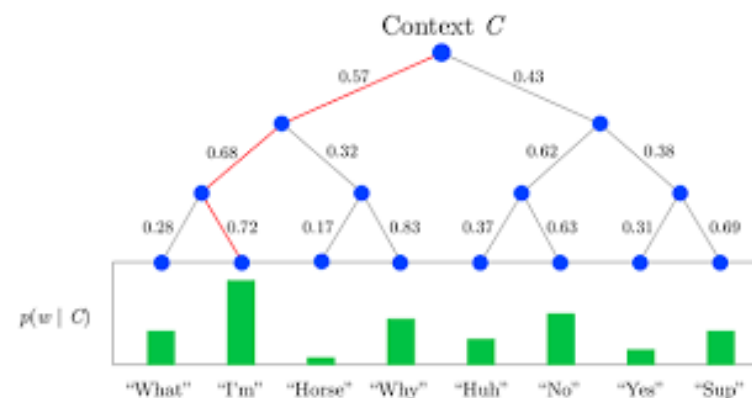
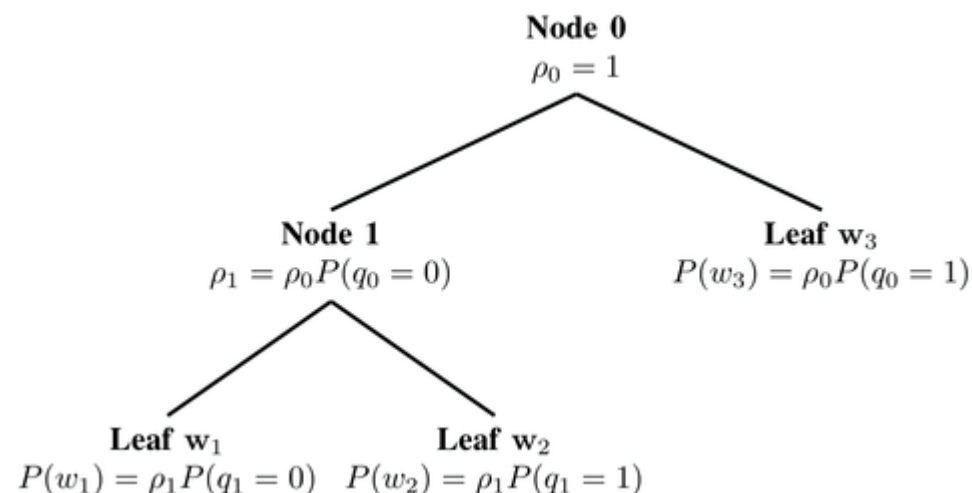
可以Gensim 為工具，以CBOW 和 SkipGram為演算法實現 Word2Vec

Tf.keras.layers.embedding 這一層就像NN，每個參數都得用Corpus訓練，才能得到vectors

https://www.tensorflow.org/text/guide/word_embeddings

Softmax 是用來分類出是哪個字，但Corpus有太多的字，比方100000個，CBOW的單層Softmax根本做不到，要分層以加速。

Hierarchical softmax (H-Softmax) is an approximation inspired by binary trees that was proposed by Morin and Bengio (2005) [\[3\]](#). H-Softmax essentially replaces the flat softmax layer with a hierarchical layer that has the words as leaves, as can be seen in Figure 1.



在Skip-Gram 的演算法中，因要產生字對表達是否為周圍字(context word)，因此需要產生Negative Samples。但是理論上所有Corpus 裡非在 windows內的全是Negative，難道將所有的非鄰接字都做成樣本嗎？當然不可能。如果我們真的那樣做，每次運算要調整的權數就太多了。所以我們往往只取4,5到20個當做negative samples，以適當的損失函數來完成正確的訓練

以下文件對Skip-Gram 的 Negative Sampling 有詳細說明
<https://www.tensorflow.org/tutorials/text/word2vec>

Skipgram

shalt	not	make	a	machine
input		output		
make		shalt		
make		not		
make		a		
make		machine		

Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Gensim = Generate Similarity (產生類似的文件)

主要優點：

- 毋須將整個文件庫(Corpus)載入主記憶體中，可見進批次進行分析整理
- 可利用計算機硬體之多核結構加速運行
- 非監督式學習無須標籤
- 可與多種作業環境及平台整合

應用舉例：

- Word2Vec：文字轉向量技術
- TF-IDF (Term frequency–inverse document frequency)：文字轉向量技術
- 潛在語意分析(LSA：Latent Semantec Analysis)

學習：

[Gensim - Introduction – Tutorialspoint](#)
[gensim: Core Concepts \(radimrehurek.com\)](#)

TF-IDF (Term frequency–inverse document frequency) :

目的就是要以一個數字表達在文件集的特異/重要程度

TF: 在本句(文件)出現次數(頻率 = 次數/總次數)除以本句(文件)總字數

IDF: 在Corpus出現的句(文件)數除以有此字出現的句(文件)數

TF代表在該句的 分量，IDF代表在該文件集的特異程度

TF * IDF 之值 即為該字在該句中的TF-IDF 值

實際公式IDF有時用log，稍有不同

TF = (Frequency of the word in the sentence) / (Total number of words in the sentence)

IDF: $\log((\text{Total number of sentences (documents)}) / (\text{Number of sentences (documents) containing the word}))$

某字的TF-IDF向量 可以說是這個字在該文件集中的特徵向量，可用來做相似度運算

<https://stackabuse.com/python-for-nlp-creating-tf-idf-model-from-scratch/>

2 小時

RNN原理
LSTM與GRU



2 小時

RNN進階應用
PCA 與 t-SNE



2 小時

AutoEncoder
圖像風格

預習：GAN，強化學習
作業：作業四

T = 100 D = 10 M = 15 K = 1	輸入層 (Input Layer)	隱藏層 (Hidden Layer)	輸出層 (Output Layer)	總數
DNN	輸入攤平(flatten)	輸入(T*D) * 隱藏(M) * 步數(T)	步數(T) * 隱藏(M)	1.5M
DNN 權數公式	$T * D = 1000$	$(T * D) * M * T = 1.5 \text{ Million}$	$T * M * K = 1500$	1.5M
RNN	$W_{xh} = D * M = 150$	$W_{hh} = M * M = 225$	$W_{ho} = M * K = 15$	385

DEMO9_SPAM_DETECTION.ipynb 說明

RNN 必須有 $N \times T \times D$

我們設定 $T = \text{data_train.shape}[1]$ 也就是整句話

我們希望 Embedding 的維數(Dimensionality) = 20 , RNN 隱藏神經元 = 15

$D = 20$

$M = 15$

#先將所有字彙轉成20維度向量 , 然後再做LSTM/RNN訓練

```
i = Input(shape=(T,))
```

```
x = Embedding(V + 1, D)(i)
```

```
x = LSTM(M, return_sequences=True)(x)
```

```
x = GlobalMaxPooling1D()(x)
```

```
x = Dense(1, activation='sigmoid')(x)
```

```
model = Model(i, x)
```

Steps Involved in the PCA

Step 1: Standardize the dataset.

Step 2: Calculate the covariance matrix for the features in the dataset.

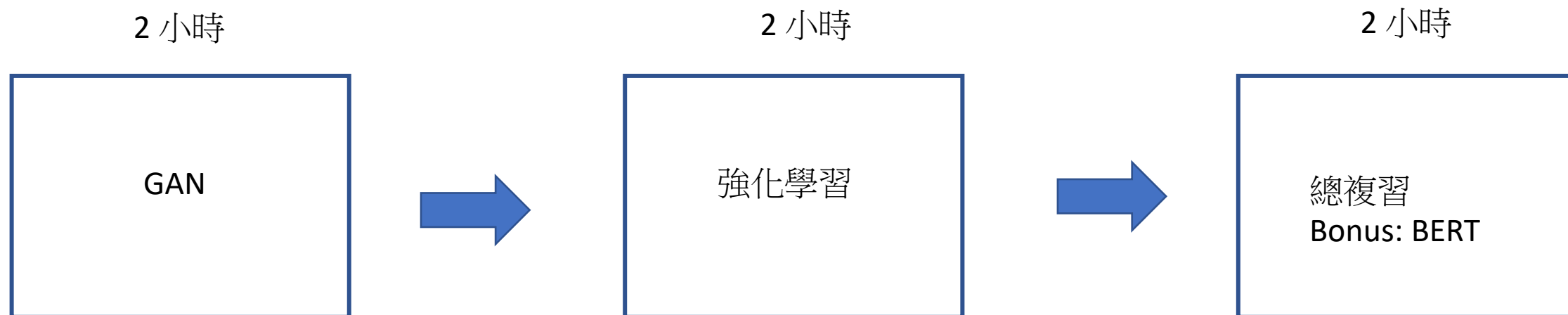
Step 3: Calculate the eigenvalues and eigenvectors for the covariance matrix.

Step 4: Sort eigenvalues and their corresponding eigenvectors.

Step 5: Pick k eigenvalues and form a matrix of eigenvectors.

Step 6: Transform the original matrix.

[Understanding Principle Component Analysis\(PCA\) step by step. | by The Nobles | Analytics Vidhya | Medium](#)



預習：GAN，強化學習
作業：作業五

原理：將兩個互相映射的GAN，以強迫loss consistent (一致) 的方式造成來源和目的影像在轉換過程中擁有類似的風格或結構。

CycleGAN最大的特色是不需要成對的影像，就可以學習到風格是如何轉換的。如果一匹棕色的馬映射到一批斑馬，那麼一群棕色的馬就可以轉換成一群斑馬，不同情境下的馬，都會映射成類似情境的斑馬。維持轉換(映射)的一致性。

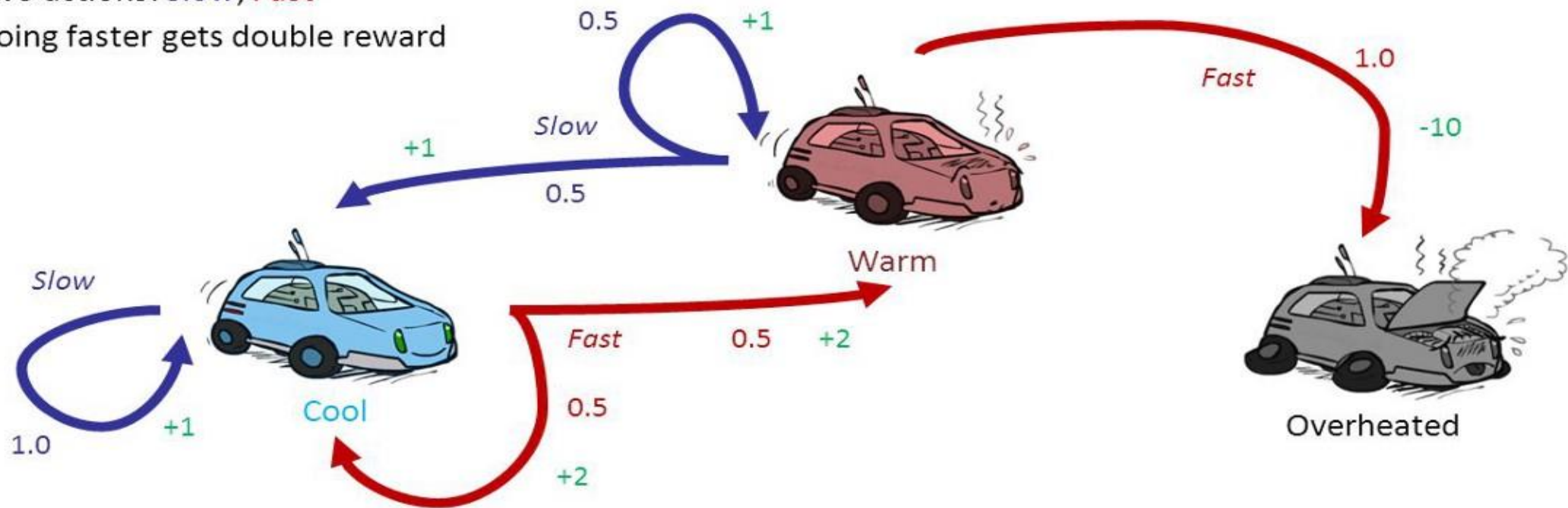
CycleGAN和Pix2Pix十分類似，不同點在於：

- CycleGAN 使用 Instance Normalization 而非 Batch Normalization
- CycleGAN 使用 ResNet Image Generator

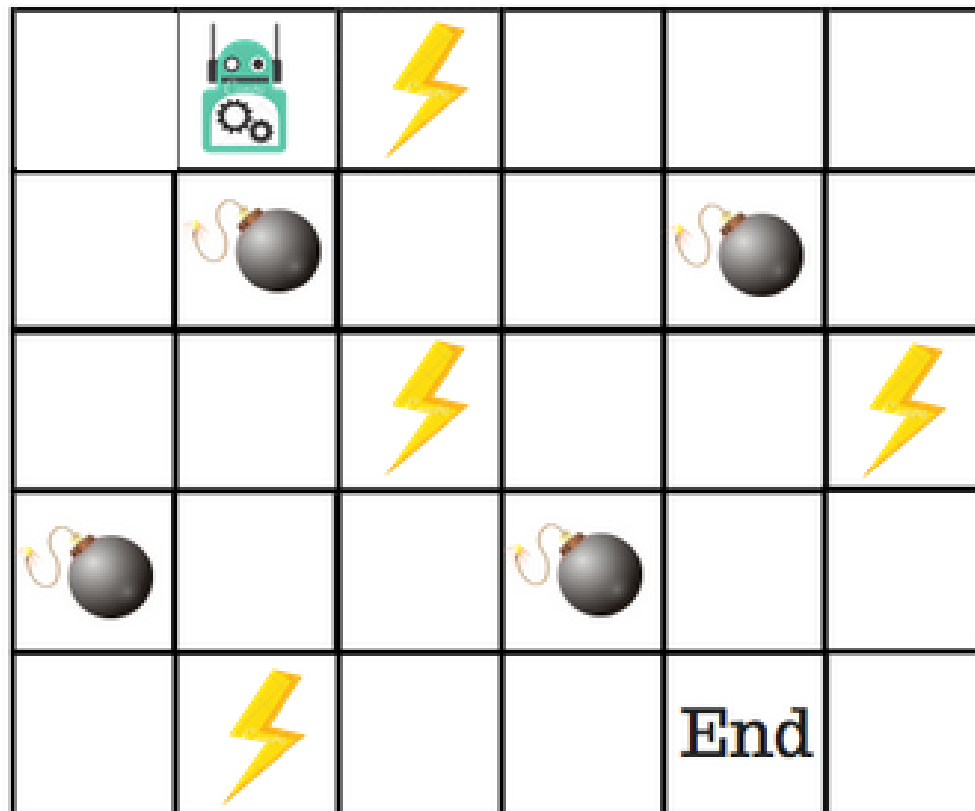
[CycleGAN | TensorFlow Core](#)

Example: Racing

- A robot car wants to travel far, quickly
- Three states: **Cool**, **Warm**, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Q-Learning Table

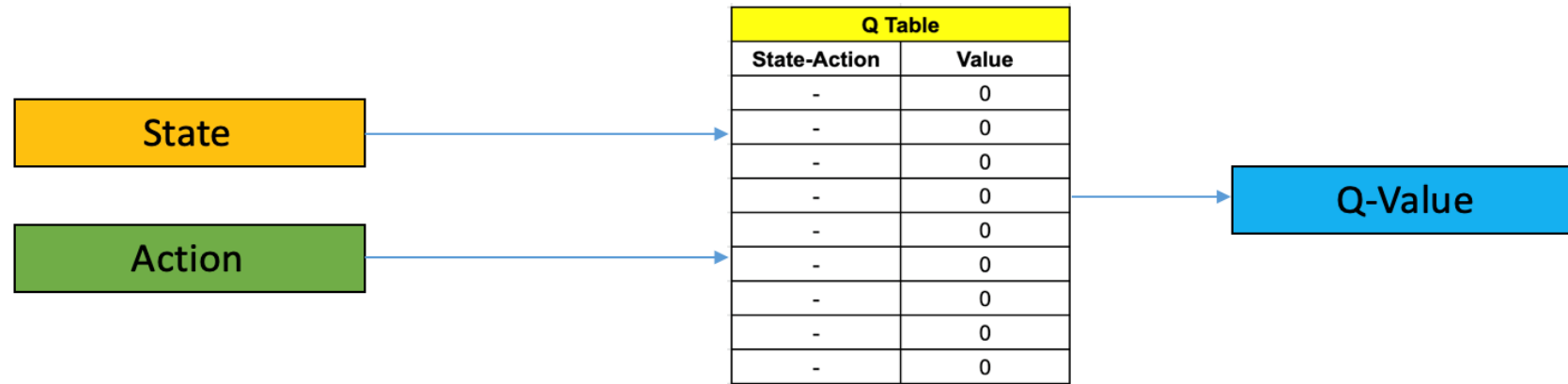


Actions :

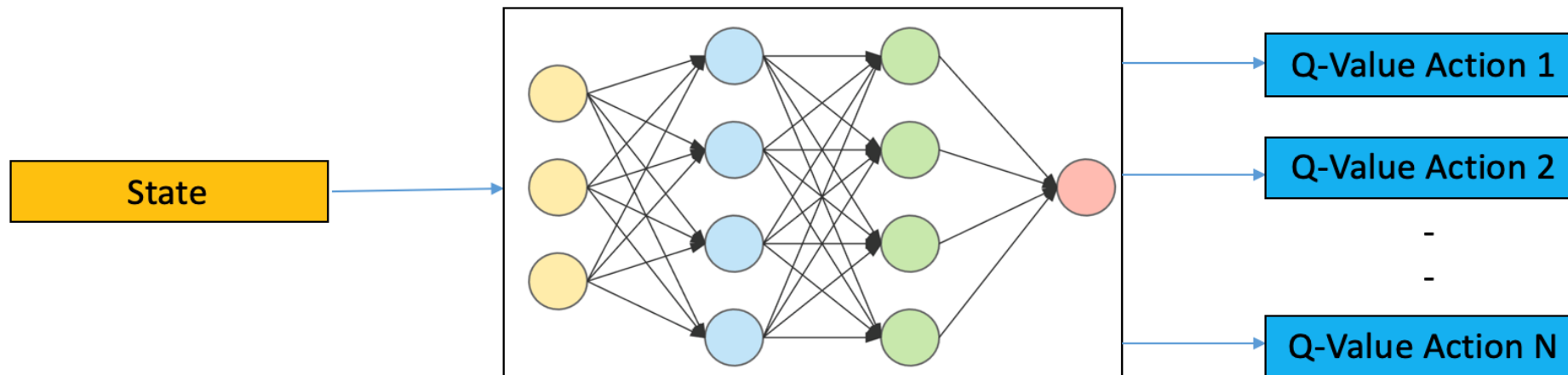
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

[An introduction to Q-Learning: reinforcement learning \(freecodecamp.org\)](https://www.freecodecamp.org/learn/2022/javascript-algorithms-and-data-structures/intermediate-javascript-algorithms-and-data-structures/q-learning/)

從 Q-table 到 DQN



Q Learning



Deep Q Learning

DQN其實就是用神經網路去模擬(逼近)Q-Table，DQN 直接產生各action再取最大值，而Q-Table 則找產生最大Q值的action。

一旦action被(policy)決定，Environment就會提供 reward 和下一個 state。Q-table 會根據新的state 去衡量所有可能的action，DQN則 輸入state 即可產生下一個action。

當State數變成無窮大時，Q-table將不敷使用，DQN經由訓練，其w與b的值將可取代被訓練後的 $Q(s,a)$ 而達到同樣效果

從 Q-table 到 DQN：例子

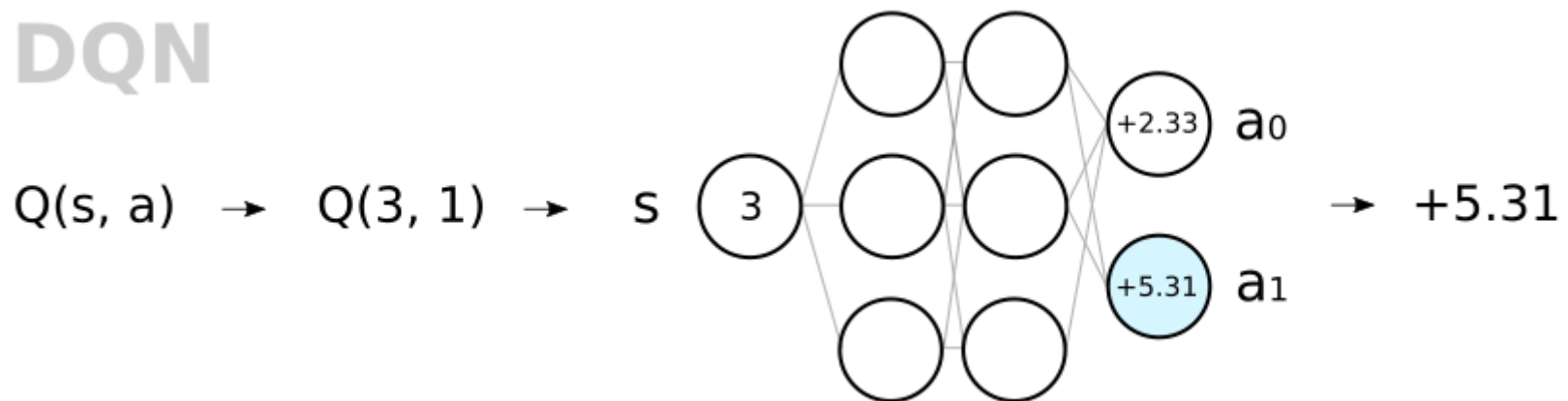
Q-Table

$Q(s, a) \rightarrow Q(3, 1) \rightarrow$

	S0	S1	S2	S3	S4
a0	+4.21	+3.24	+1.84	+2.33	+3.73
a1	+2.53	+7.44	+3.34	+5.31	+6.22

$\rightarrow +5.31$

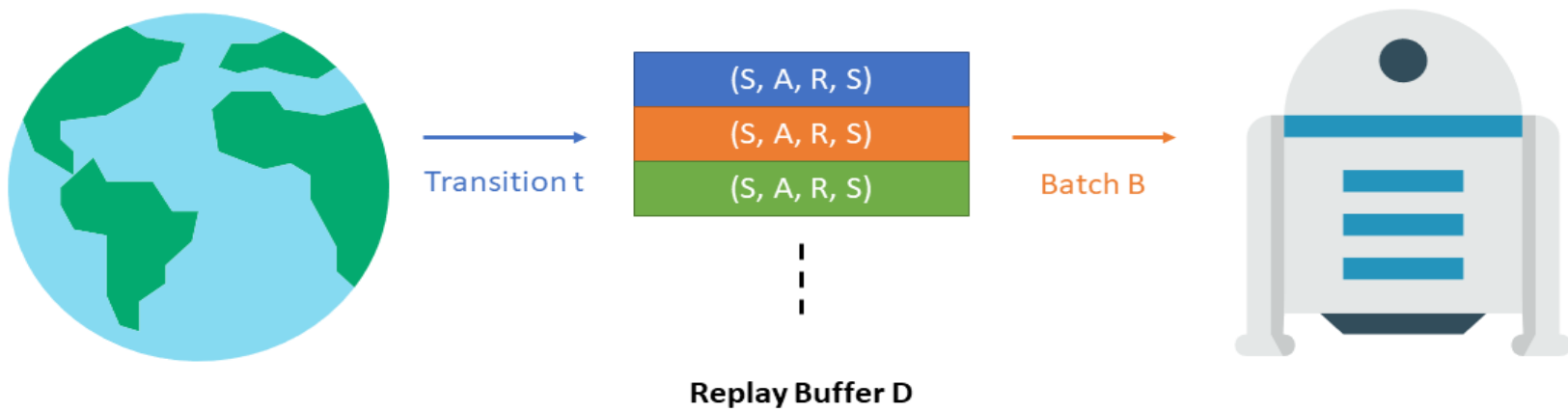
DQN



訓練時有效而多元，無須真實世界的模擬或執行，只靠機器重覆回放 (s, a, r, s') 值收斂較快因而結果穩定

Experience Replay

- Save transitions $(S_t, A_t, R_{t+1}, S_{t+1})$ into buffer and sample batch B
- Use batch B to train the agent



Bonus: BERT的原理 與應用

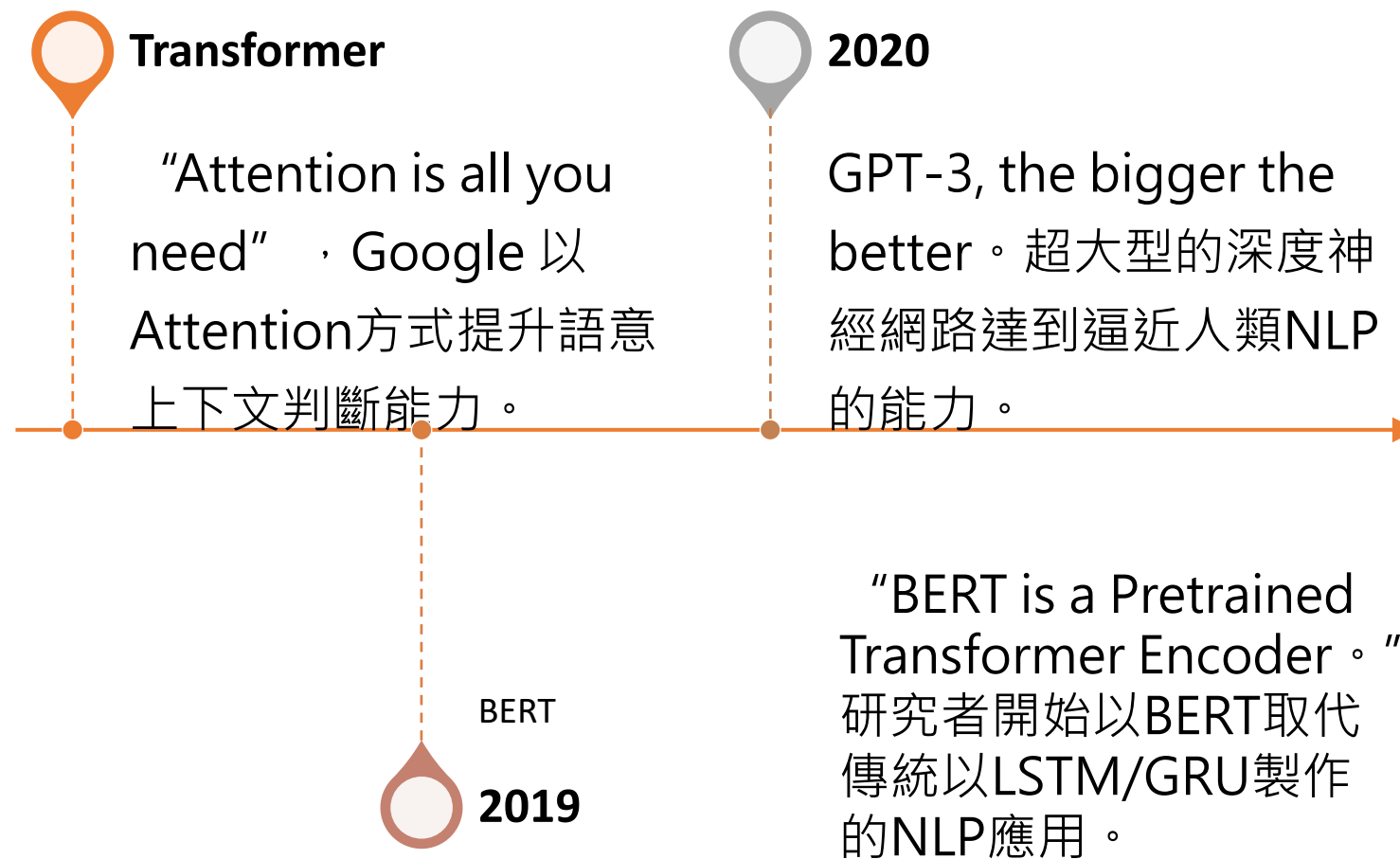
學習目標：

- 了解Transformer
- 了解BERT

BERT的原理與應用

- BERT的歷史淵源
- BERT的簡介
- BERT的應用：假新聞分類器
- BERT的其他應用

BERT的歷史淵源



- Elmo：雙向預訓LSTM based context-based Language Model
- ULM-FiT：在RNN/NLP 領域開始Transfer Learning 就像 CNN在 Computer Vision領域
- Transformer: 包括 Encoder和 Decoder，主要是建立自然語言翻譯的語言模型，他完全揚棄了 LSTM 模式而採用 Attention 模式。他的 Encoder-Deoder模式可用來做許多 downstream applications.
- OpenAI Transformer：利用 Transformer Decoder 做 Predicting Future Tokens。Encoder只留下self-attention layer. 應用包括 分類，蘊涵(entailment)，相似(similarity)，選擇等。But it is forward direction only.

- BERT (Bidirectional Encoder Representations from Transformers), 是一個 Pretrained Transformer 的 Encoder。Transformer 源自 Attention is all you need 的構想，揚棄傳統的 RNN/LSTM/GRU 的作法，不以順序(sequence)而以位置(position)達到平行訓練的目的，同時以一套 Self-Attention 的計算法更準地抓到了文句的上下文(Context)，進而提高幾乎所有 NLP 應用的準確度，包括分類與預測等等。
- 由於 BERT 系統龐大，即使是 BASE model 也有上億個神經元，非普通系統所能承受。我們一般都用 Pretrained BERT Model 再接上 Fine Tuning Model 來完成分類或其他應用，效果比起用 RNN/LSTM 好很多。

- BERT的輸入有三個Embeddings，分別是 Token Embeddings (可能由 Word2Vec, Glove等產生)，Position Embeddings紀錄順序位置，以及 Segment Embeddings 紀錄相關或隨機字句。
- 在預訓時，BERT嘗試同時預測(15%)被掩蓋的字(Masked Language Model，MLM)以及下一句子預測 (Next Sentence Prediction)。
- BERT其實就是 預訓過的 Transformer Encoder，因為同時保有 字義，位置 和 段落 輸入的 embeddings，運用 Transformer Self-Attention 的能力，只要在後頭接上簡單的Linear Classifier or Regressor，就可完成許多NLP的應用。

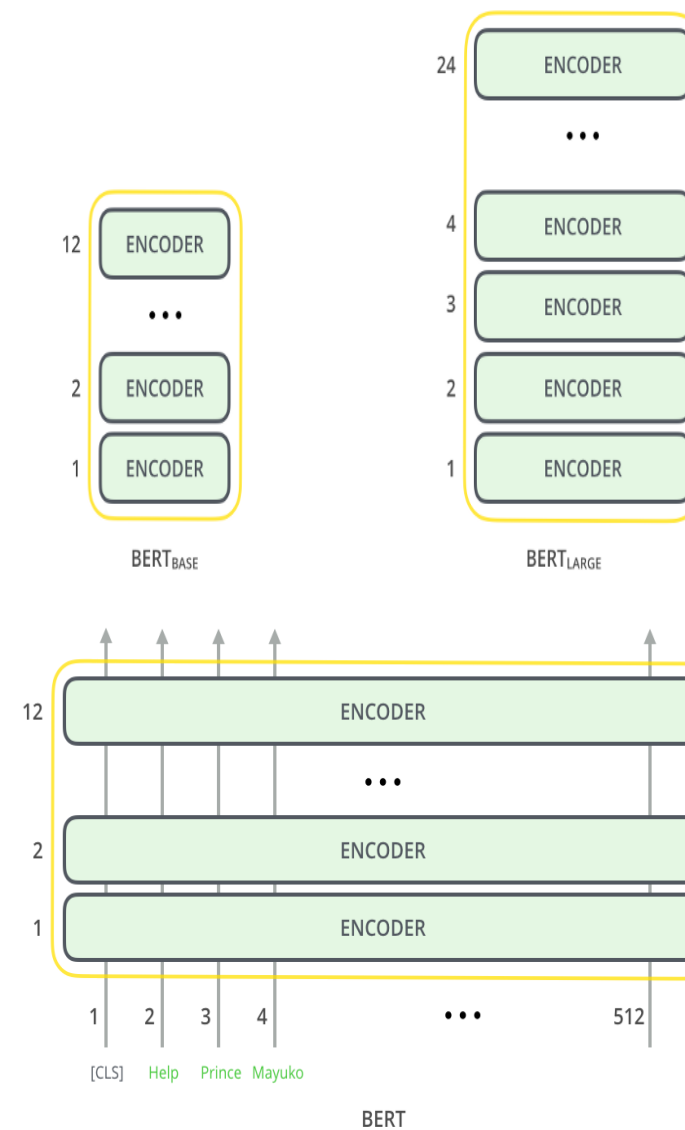
常用的 BERT模型有兩種:

BERT BASE – 和OpenAI Transformer 大小差不多(12層)

BERT LARGE – 研發用非常大的模型(24層)

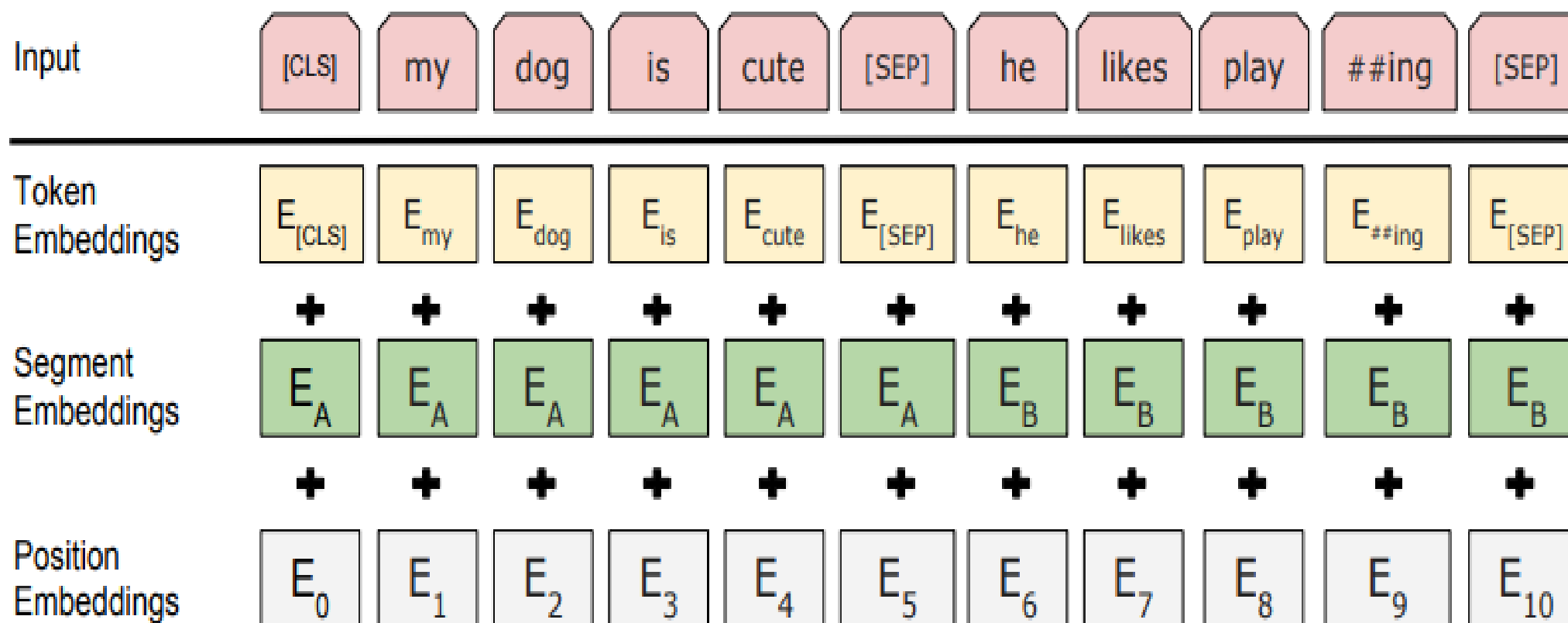
前面說過 BERT 基本上就是一個訓練過的 Transformer Encoder如右圖。BASE/LARGE 有 12/16 Attention Head， 784/1024 Hidden Unit。

第一個input token 有一個特殊的 Classification Token [CLS]，Input token 不斷往上走，每一層都有self-attention層和Feed-forward層，然後在交給下一個Encoder。

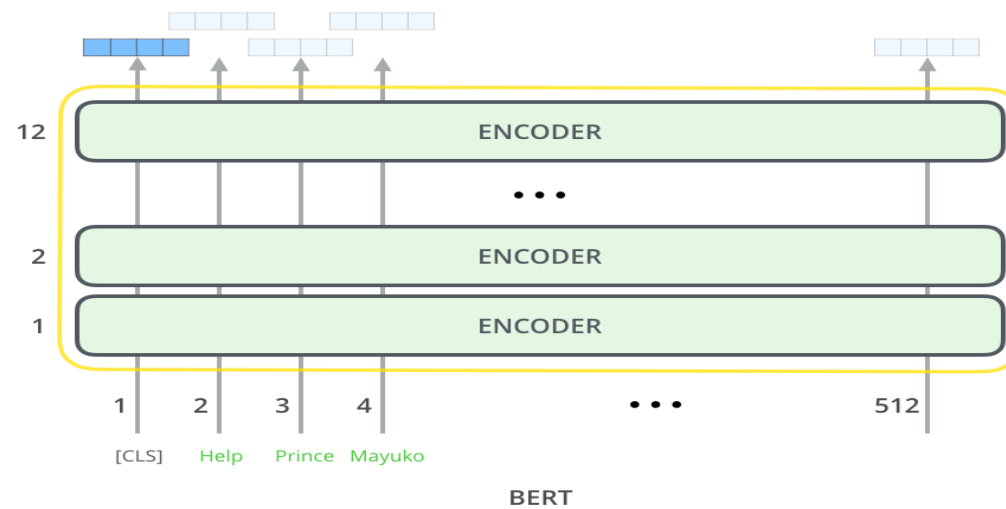


- BERT的輸入Embeddings其實是哪三個Embeddings的組合：
- Token Embedding: 就是你提供文章的句子加上[CLS][SEP][MASK]等特殊字符的向量，[CLS]代表句子的開端；[SEP]是相鄰(關)句子的間隔，像是句號；[MASK]是掩蓋掉的字符。
- Segment Embedding: 和[SEP]配合，表示第幾個句子。
- Position Embedding: 輸入中的第幾個位置。

BERT的簡介：輸入Embeddings

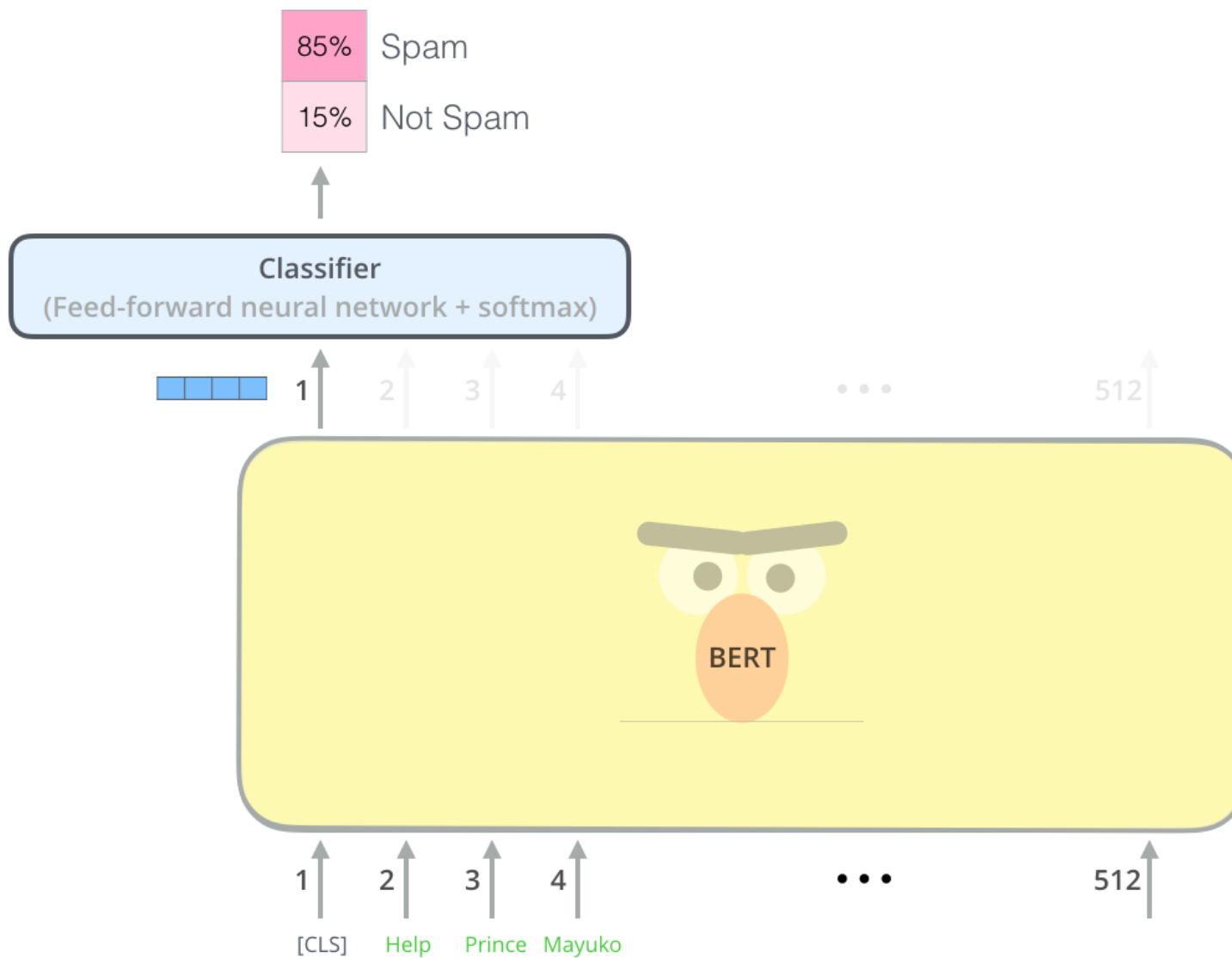


輸出部分，512個輸出每個都有一個hidden_size=768(BERT Base)的向量，如果是做句子分類，其實只須注意第一個位置即可，也就是相對於[CLS]位置的那個。



這個向量可以輸入到我們設計的分類器. BERT作者僅用了單層的NN分類器即達到良好效果.

BERT的應用：假新聞分類器(圖示)



- 以新聞真偽這個應用的分類例題來說，我們主要在做dataset preprocessing以及 Classifier Model的 Fine Tune Training，中間主要的分類引擎是BERT BASE UNCASED。一旦將Label好的Real News 和 Fake News 的 Training Set, Test Set 和 Validation Set 分好，Preprocess就算完成。我們的PyTorch Fine Tune Python 程式主要就是將接口定義好，超參數設好，其實和其他的Classifier沒有甚麼不同。
- 例題：<https://towardsdatascience.com/bert-text-classification-using-pytorch-723dfb8b6b5b>
- BERT fine-tune on fake news detection.ipynb
- Use: PyTorch interface for BERT by Hugging Face

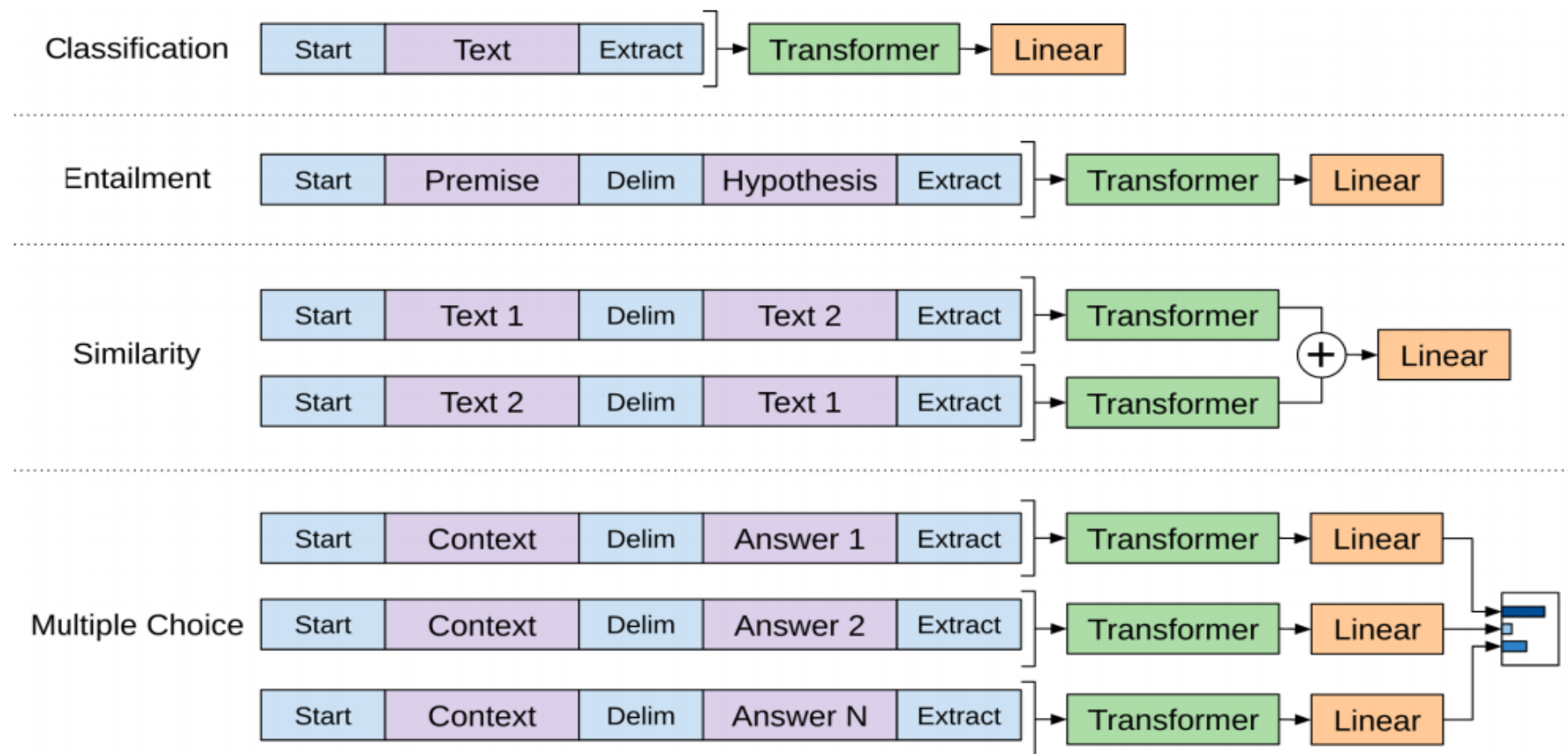
- 1. 下載Kaggle的Real/Fake News Dataset news.csv，放在Google Drive /transformers/data 裡：
<https://www.kaggle.com/nopdev/real-and-fake-news-dataset>
- 2. 在Colab 執行 **Preprocessing of Fake News Dataset.ipynb**
- 3. 要將Goggle Drive Mount 到 Colab的 Container，中間需要 Authorization Code
`from google.colab import drive`
`drive.mount('/content/drive')`
- 4. 按比例寫入 train.csv, test.csv 和 valid.csv

- `tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')`
- `text_field = Field(use_vocab=False, tokenize=tokenizer.encode, lower=False,`
- `include_lengths=False, batch_first=True,.....`
- `train, valid, test = TabularDataset.splits(path=source_folder, train='train.csv'`
- `# Iterators`
- `train_iter = BucketIterator(train, batch_size=16, sort_key=lambda x: len(x.text),`
- `.....`
- `test_iter =`
- `valid_iter =.....`
- `.`

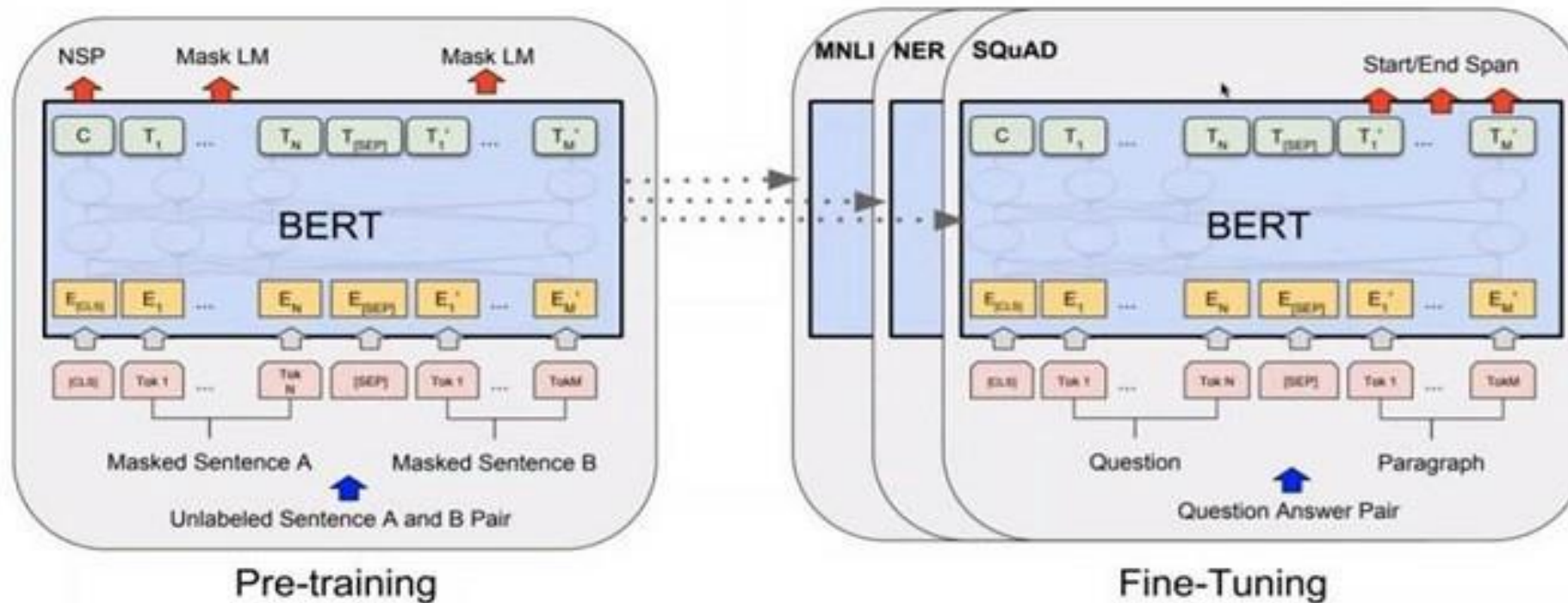
- `class BERT(nn.Module):`
- - `def __init__(self):`
 - `super(BERT, self).__init__()`
 - `options_name = "bert-base-uncased"`
 - `self.encoder = BertForSequenceClassification.from_pretrained(options_name)`
- `def forward(self, text, label):`
- `loss, text_fea = self.encoder(text, labels=label)[:2]`
- `return loss, text_fea`

- `model = BERT().to(device)`
- `optimizer = optim.Adam(model.parameters(), lr=2e-5)`
- - `train(model=model, optimizer=optimizer)`
- `# show loss, accuracy...`
- `# Evaluate`
- 作業檔案: BERT fine-tune on fake news detection.ipynb

BERT的作者號稱還可做以下這些NLP應用：分類，蘊涵，相似度比較，多選等，以下是簡單架構示意圖



BERT的的其他應用：模式



左邊是pre-trained: NSP: Next Sentence Prediction, Mask LM: Masked Language Model

右邊是fine-tuning應用：NER: Named Entity Recognition, SQuAD: 回答系統, MNLI：自然語言介面, 以及最常見的分類

- 延伸學習任務一，用BERT MLM pretrained model 做中間字的預測，比方輸入
 - text = "[CLS] Who was Jim Henson ? [SEP] Jim Henson was a puppeteer [SEP] "
 - masked_index = 8
 - tokenized_text[mask_index] = '[MASK]'
 - 然後猜出是 Henson
 - <https://github.com/aniruddhachoudhury/BERT-Tutorials/tree/master/Blog%201>
 - 程式檔名：BERT_Tutorial_1.ipynb
-
- 延伸學習任務二，用 [The Corpus of Linguistic Acceptability \(CoLA\)](#) dataset 判斷文法是否正確
 - 程式檔名：BERT_Fine_Tuning_Sentence_Classification.ipynb

Bonus: Transfer Learning

學習目標：

- 了解轉換學習
- 了解 TFLite

轉換學習：TFLite 與 AIoT

能夠讓行動(Mobile)和嵌入式(Embedded)應用使用機器學習(ML)的一組軟體工具，這些應用因而可在 iOS, Android, Raspberry Pi上執行。

但是如何將訓練好的模型輸出給行動應用開發者呢？

高階步驟: (訓練端Training Side)

#1 正常訓練模型

從server/desktop訓練模型

把model 和 weights 遷移到行動裝置

一旦在行動裝置，model即可做

prediction/inference

#2 用Tensorflow Lite Converter 轉換模型

到.tflite 檔.

假設是 Keras API

.tflite 只是一個檔案模式, 就像 model.save() 產生 .h5 檔一樣

在 “import tensorflow as tf” 之後加幾行 code

高階步驟: (行動開發者端)

#3 在行動開發者端, 使用 Tensorflow lite 庫.
支持 Java/C++ (Android) 和 Swift/Objective-C (iOS)
用 Tensorflow Lite Interpreter 載入 .tflite 檔並且
用現有模型做預測。
數據需要和Tensorflow Lite interpreter相匹配.

要改變的程式，訓練端

```
#convert the model to TFLite format
```

```
converter =
```

```
tf.lite.TFLiteConverter.from_keras_model(model)
```

```
tflite.model = converter.convert()
```

```
with open( "converted_model.tffile" , wb) as f:  
    f.write(tflite_model)
```

TFLite vs TF Serving

行動裝置也可用Serving直接對server做API calls.
TF Lite 讓你直接在行動/嵌入裝置使用模型，無需使用網路。
如果在地鐵或通訊不良處，依賴雲端不是很好的選擇。
但如果模型複雜需大量運算TF Lite 就不是很好的選擇。

對無ML背景的行動裝置開發者:
預訓練好的模型包括: Object detection, pose
estimation, smart reply.

<https://www.tensorflow.org/lite/models>

學習目標：

- 了解 推薦系統
- 實作推薦系統

Bonus: Recommender System

推薦系統的製作

推薦系統三元素：(User, Item, Rating)通常以電影為例

Ratings 數據一定非完整

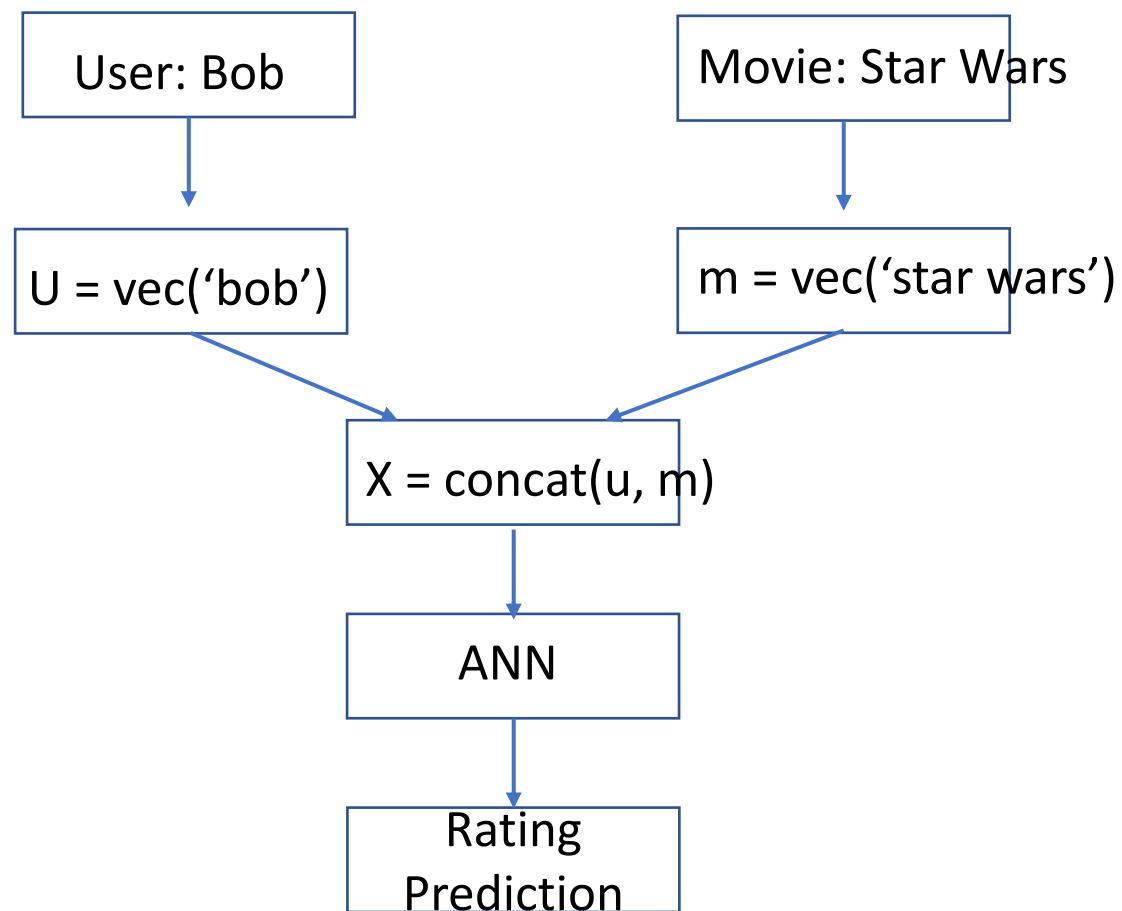
運用機器學習去趨近 $f(\text{user}, \text{item}) \rightarrow \text{rating}$

如果我們已有預測函數 $f()$, 我們只需選擇user尚未看過的項目(比方說電影)從最高依降序推薦.

但 (user, item) 非數值而是範疇型(categorical), 然而NN需要做矩陣運算

在NLP中我們已得到靈感，用 embedding，將(user, item)轉成相關的向量

推薦系統流程：



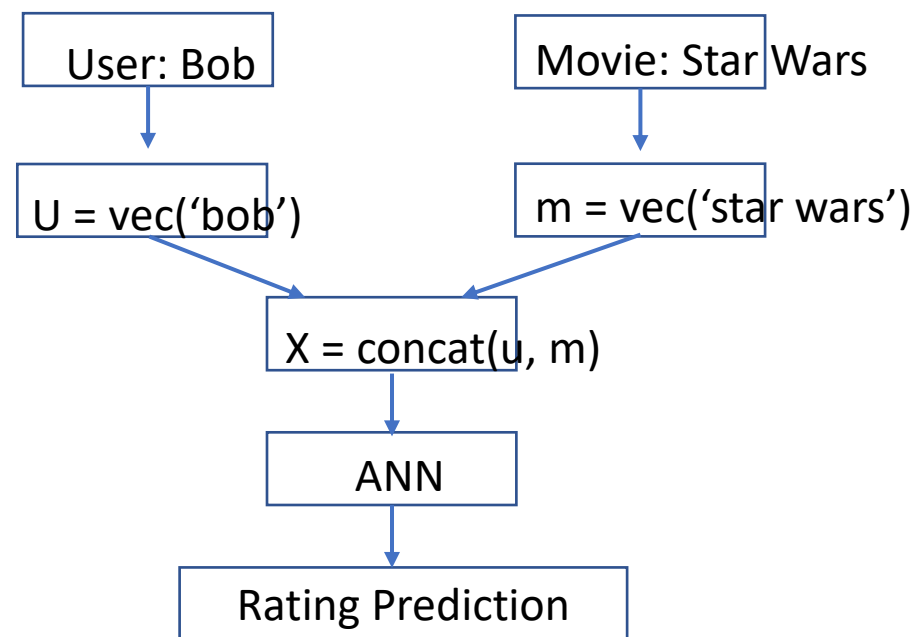
Pseudocode :

```
u = Input(shape((1,))  
m = Input(shape((1,))  
u_emb = Embedding(num_users, embedding_dim)(u)  
m_emb = Embedding(num_movies, embedding_dim)(m)  
#combine into a single feature factor  
x=Concat(u_emb, m_emb)  
#ANN  
x=Dense(512, activation= 'relu' )(x)  
X=Dense(1)(x)
```

用Functional API, 而非 sequential API

Sequential API 無法合併兩個模型，
因為每一層只能連接到另一層。

Functional API 提供NN的彈性，
比方GAN的設計。



Demo21_Recommender System.ipynb

Note: The ID 需要連續，因為它是對向量的索引.

需要先扁平化(flatten)再接續 (concatenate): $N \times 1 \times D \rightarrow N \times D$

Homework: 在老師示範此檔案之後，嘗試尋求改進val_loss的任何方法。

Hint: 改變 hyper parameter.