

# 應用機器學習

Brian Chan 陳醒凡

# 課程目標

1. 了解基本的數據分析
2. 了解基本的機器學習(Machine Learning)方法
3. 掌握Python的基本操作和一些有用的package
4. 處理及從網上下載數據
5. 在Python上應用機器學習

# 今天課堂 概要

## Introduction to deep learning

1. Neural network design
2. Training approach
3. Regularization
4. Example of deep learning (case with code)
5. Self-learning resources

# TYPES OF NEURAL NETWORK

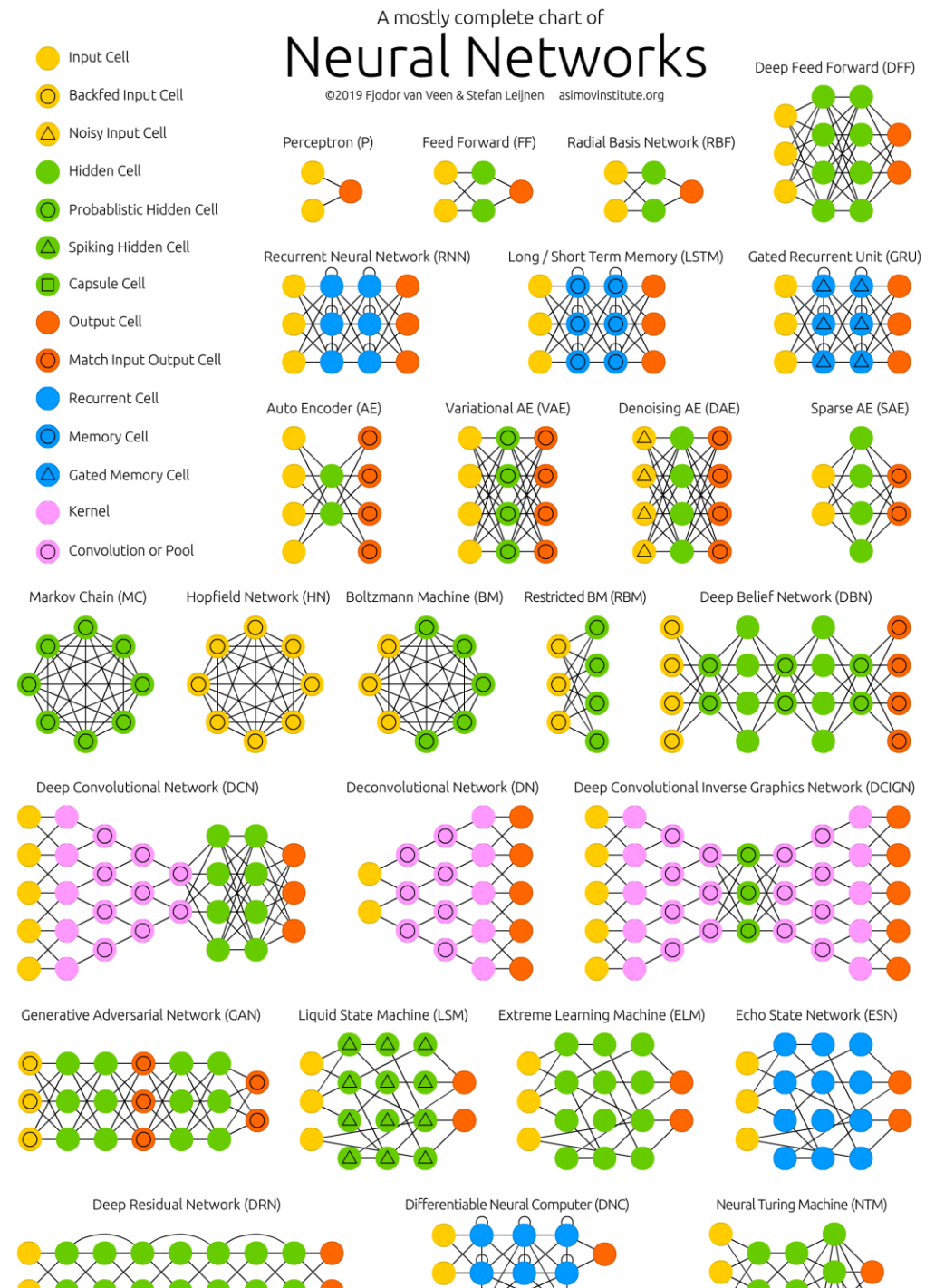
Feedforward

Recurrent network

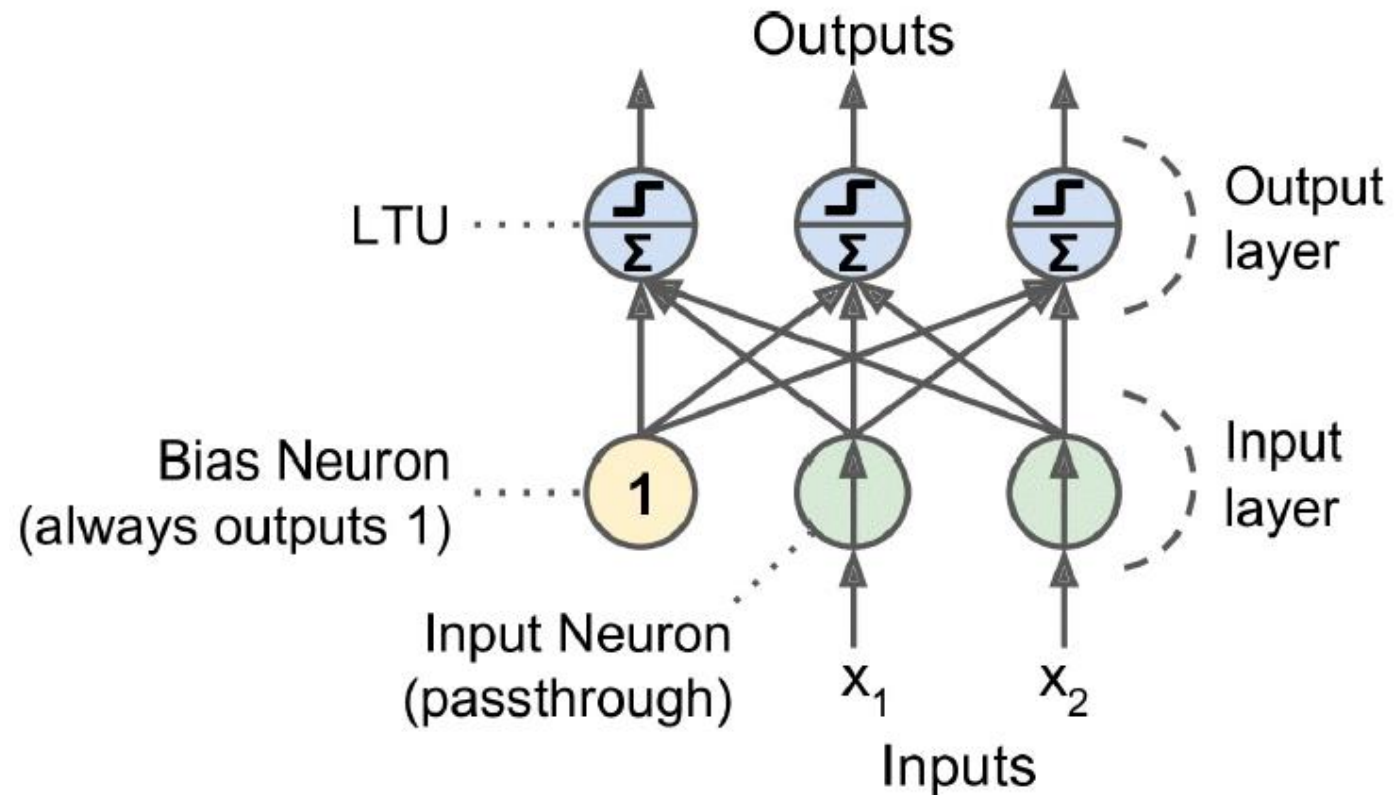
Convolutional network

Auto-encoder network

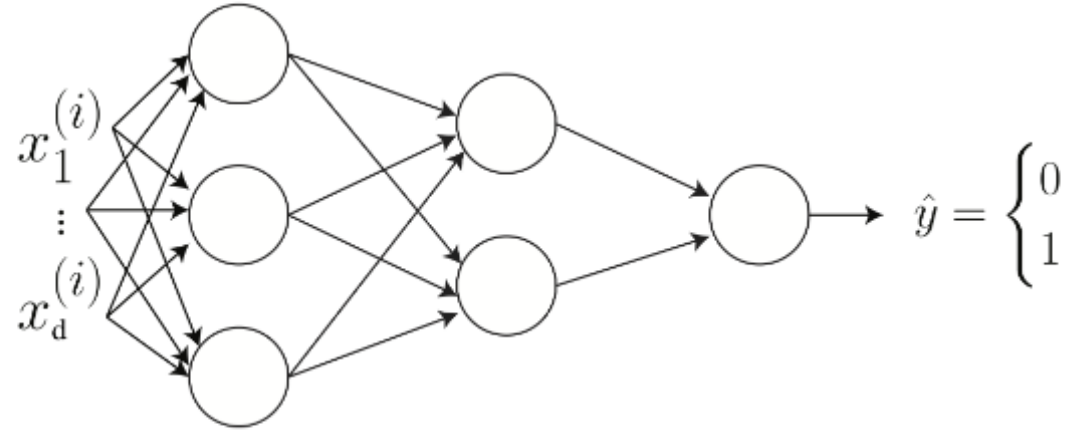
etc.



# NEURAL NETWORK DESIGN



Consider a two layer neural network. On the left, the input is a flattened image vector  $x^{(1)}, \dots, x_d^{(i)}$ . In the first hidden layer, notice how all inputs are connected to all neurons in the next layer. This is called a *fully connected* layer.



The next step is to compute how many parameters are in this network. One way of doing this is to compute the forward propagation by hand.

$$z^{[1]} = W^{[1]}x^{(i)} + b^{[1]} \quad (3.1)$$

$$a^{[1]} = g(z^{[1]}) \quad (3.2)$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \quad (3.3)$$

$$a^{[2]} = g(z^{[2]}) \quad (3.4)$$

$$z^{[3]} = W^{[3]}a^{[2]} + b^{[3]} \quad (3.5)$$

$$\hat{y}^{(i)} = a^{[3]} = g(z^{[3]}) \quad (3.6)$$

# ACTIVATION FUNCTION

$$g(x) = \frac{1}{1 + \exp(-w^T x)}$$

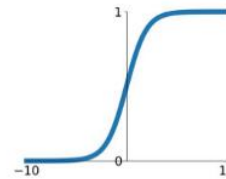
$$g(z) = \frac{1}{1 + e^{-z}} \quad (\text{sigmoid})$$

$$g(z) = \max(z, 0) \quad (\text{ReLU})$$

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (\text{tanh})$$

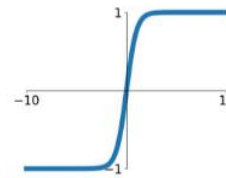
**Sigmoid**

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



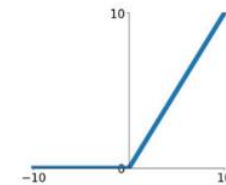
**tanh**

$$\tanh(x)$$

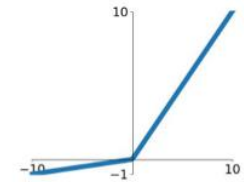


**ReLU**

$$\max(0, x)$$



**Leaky ReLU**  
 $\max(0.1x, x)$

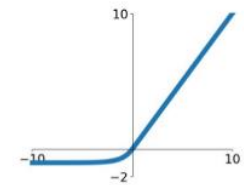


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



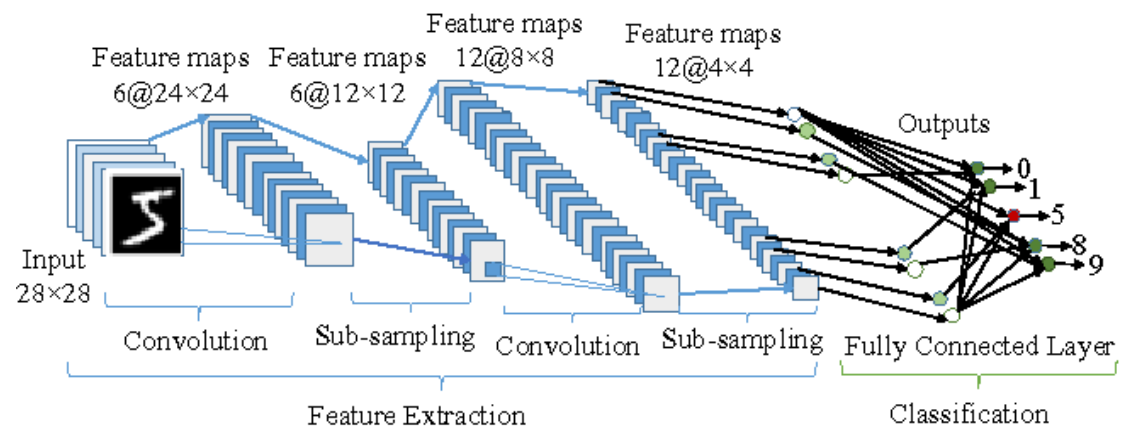
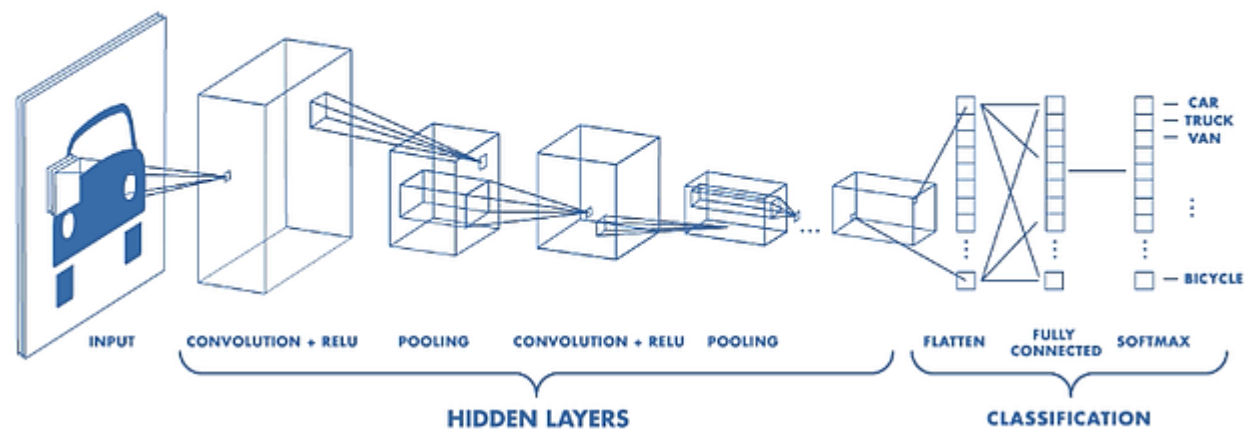
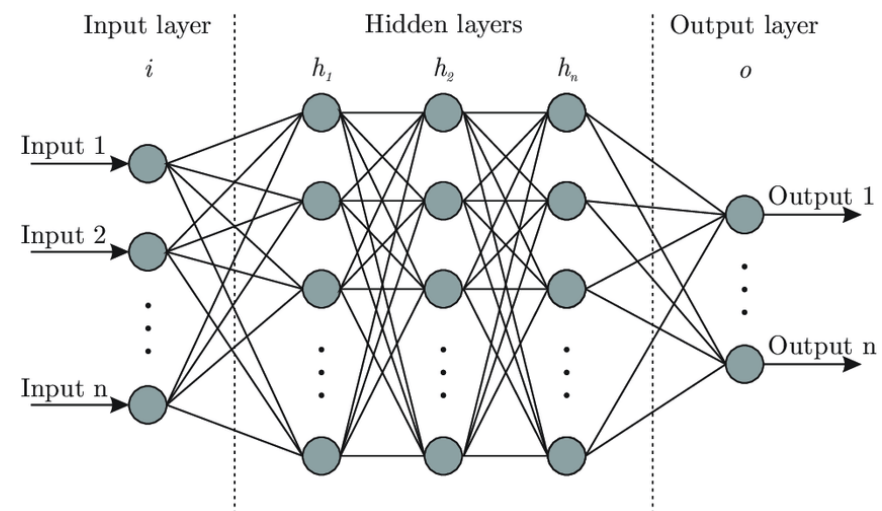
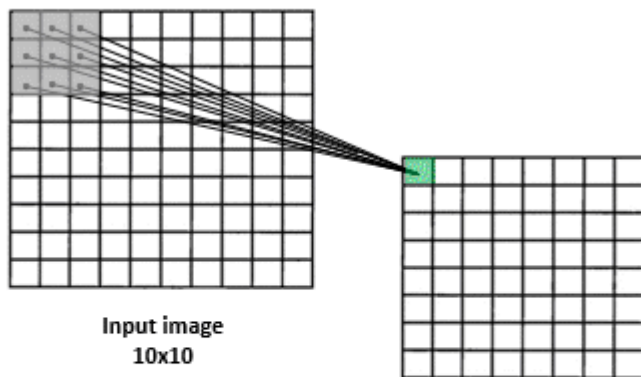


Fig. 1. CNN block diagram.





# CONVOLUTION



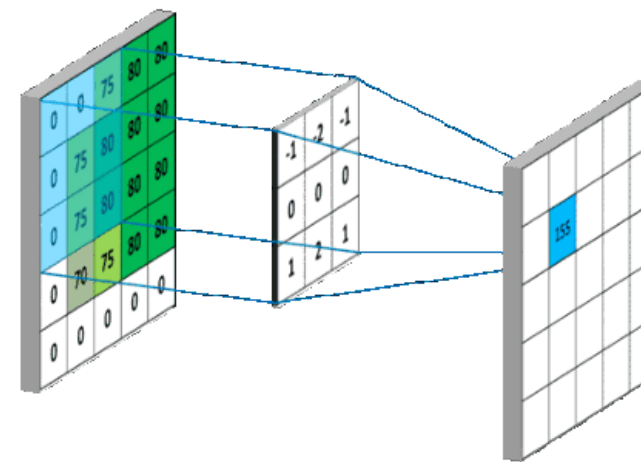
Feature map



Kernel



Hidden neuron



# POOLING

Max Pooling

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2  
pool size

100	184
12	45

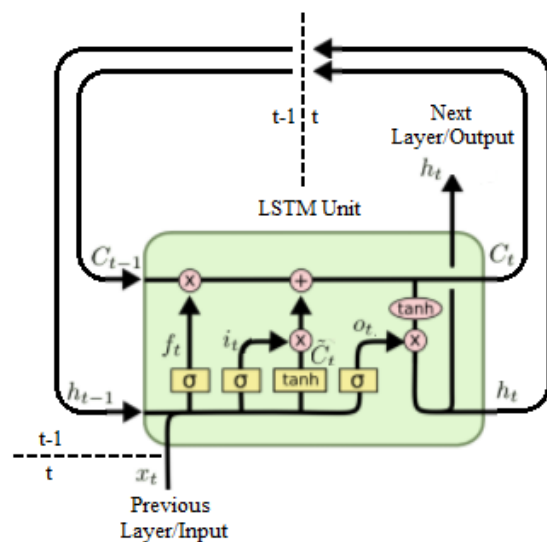
Average Pooling

31	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

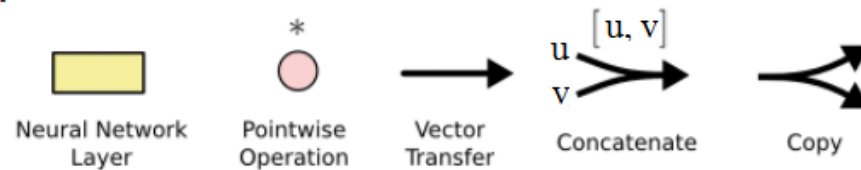
2 x 2  
pool size

36	80
12	15

# LSTM CELL



$$\begin{aligned}
 f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
 i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
 \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
 C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
 o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
 h_t &= o_t * \tanh(C_t)
 \end{aligned}$$



# INITIALIZING NEURAL NETWORKS

<https://www.deeplearning.ai/ai-notes/initialization/>

# TRAINING ALGO: MINI-BATCH GRADIENT DESCENT

Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

## Upsides

The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.

The batched updates provide a computationally more efficient process than stochastic gradient descent.

The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

## Downsides

Mini-batch requires the configuration of an additional “mini-batch size” hyperparameter for the learning algorithm. (e.g. 64, 128, 256, ...)

Error information must be accumulated across mini-batches of training examples like batch gradient descent.

# EPOCH - BATCH SIZE - ITERATIONS

Epoch: One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Batch Size: Total number of training examples present in a single batch.

Iterations: The number of batches needed to complete one epoch.

We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

Where Batch Size is 500 and Iterations is 4, for 1 complete epoch.

# REGULARIZATION

Regularization with penalization of L1 & L2

Dropout (Srivastava 2014)

# DROPOUT

The term “dropout” refers to dropping out units (both hidden and visible) in a neural network.

Simply put, dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass.

## **Some Observations:**

- Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
- Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.



# MNIST DATASET

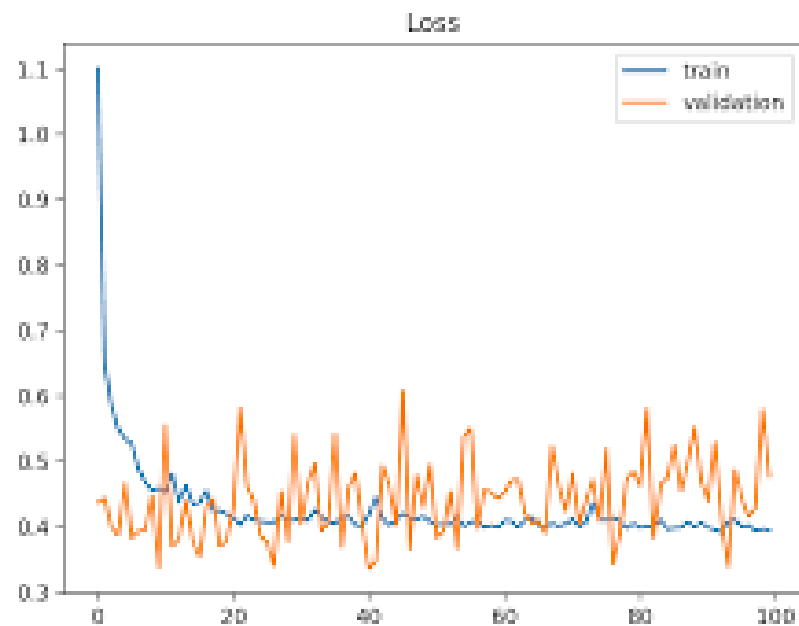
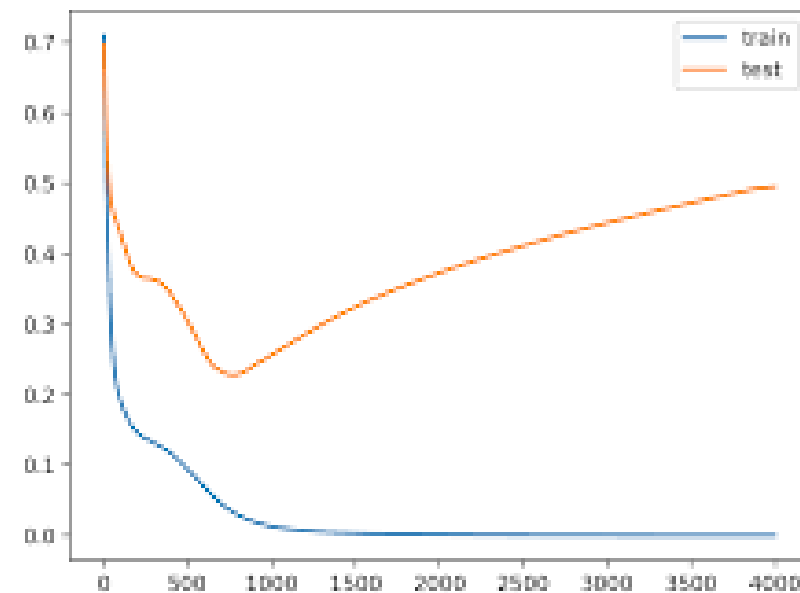
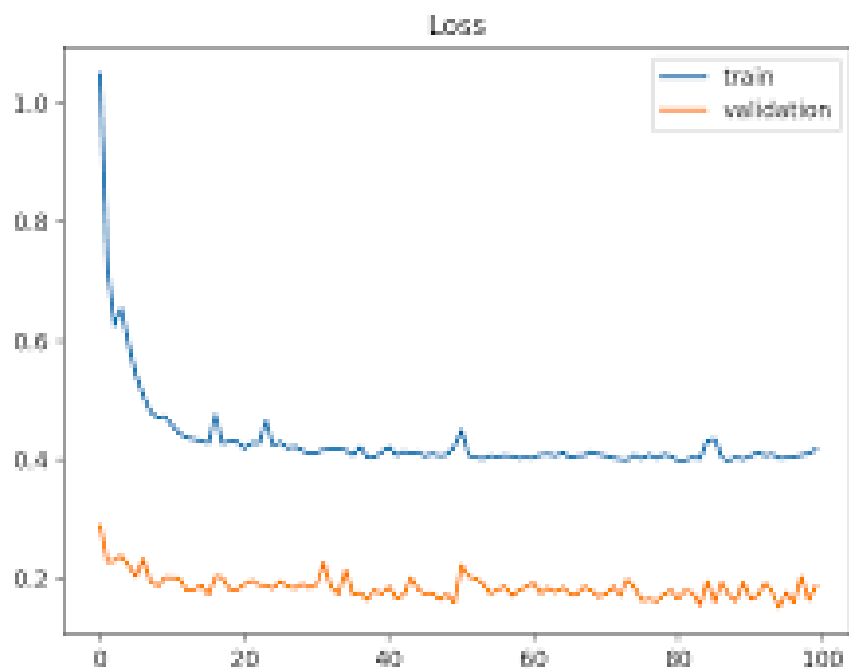
<http://yann.lecun.com/exdb/mnist/>

# INSTALL KERAS

Run code on anaconda prompt: `pip install keras`

<https://www.tensorflow.org/install/gpu>

# TRAIN AND VALIDATION ERROR



# 今天課堂 概要

## Introduction to deep learning

1. Neural network design
2. Training approach
3. Regularization
4. Example of deep learning

# 下一課...

1. Medium
2. Reference books

# SUGGESTED BOOKS

Deep Learning with Python

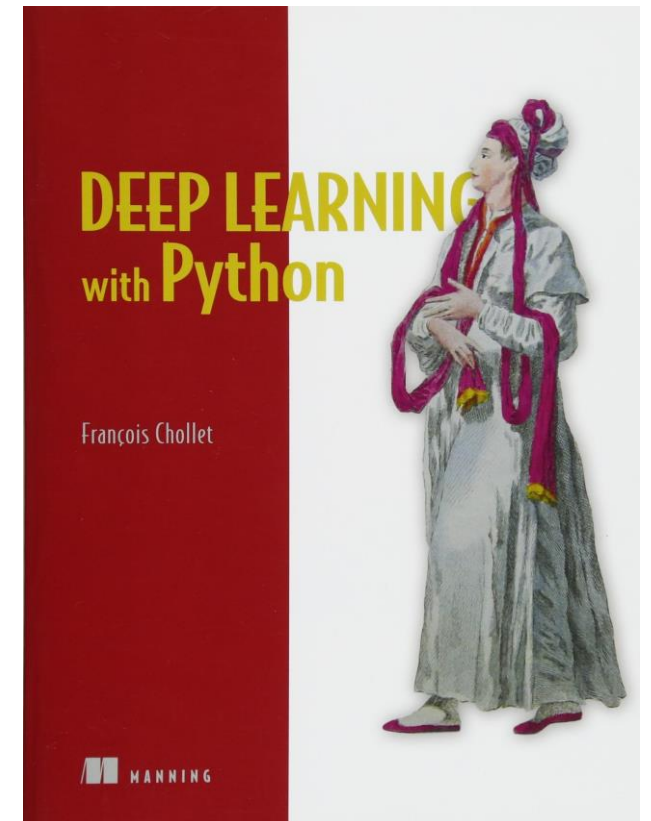
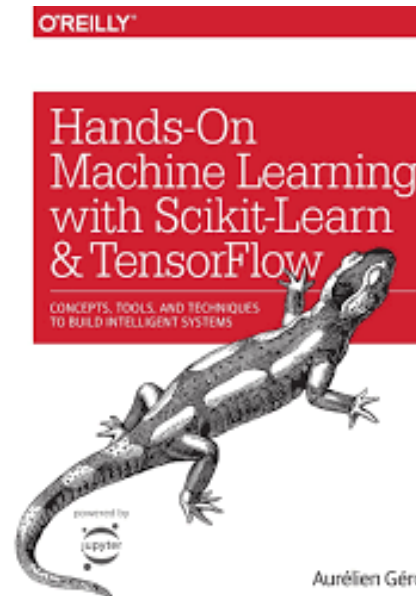
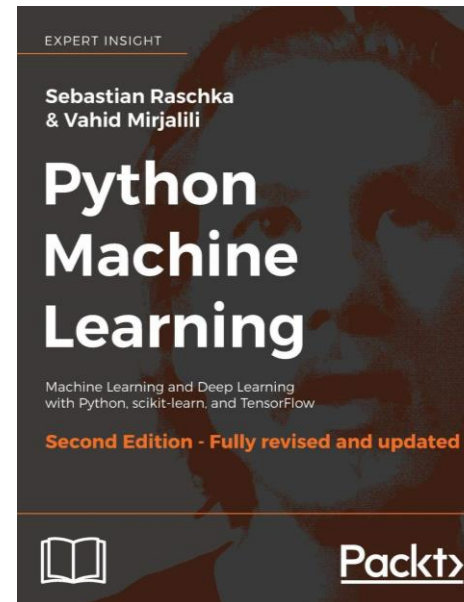
by *Francois Chollet*

Hands-On Machine Learning with Scikit-Learn  
and TensorFlow

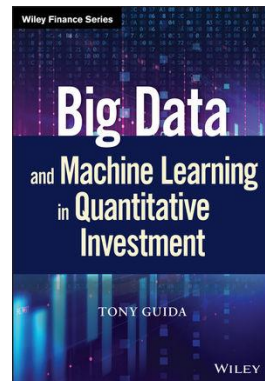
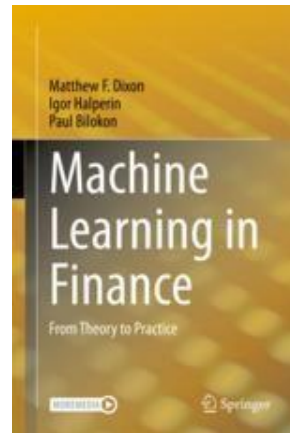
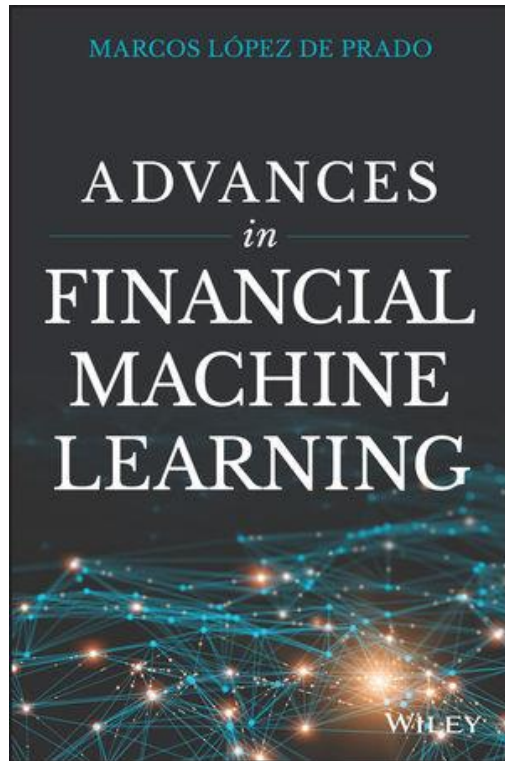
by *Aurelien Geron*

Python Machine Learning

by *Sebastian Raschka*



# MACHINE LEARNING FOR QUANTITATIVE FINANCE



## Machine Learning in Finance From Theory to Practice

by Dixon, Matthew F., Halperin, Igor, Bilokon, Paul

## Big Data and Machine Learning in Quantitative Investment

by Tony Guida

## Advances in Financial Machine Learning

by Marcos Lopez de Prado, Steven Jay Cohen

[https://github.com/mfrdixon/ML\\_Finance\\_Codes](https://github.com/mfrdixon/ML_Finance_Codes)