Deep Learning

Eun Soo Choi, Ph.D.

Monday March 22, 2021

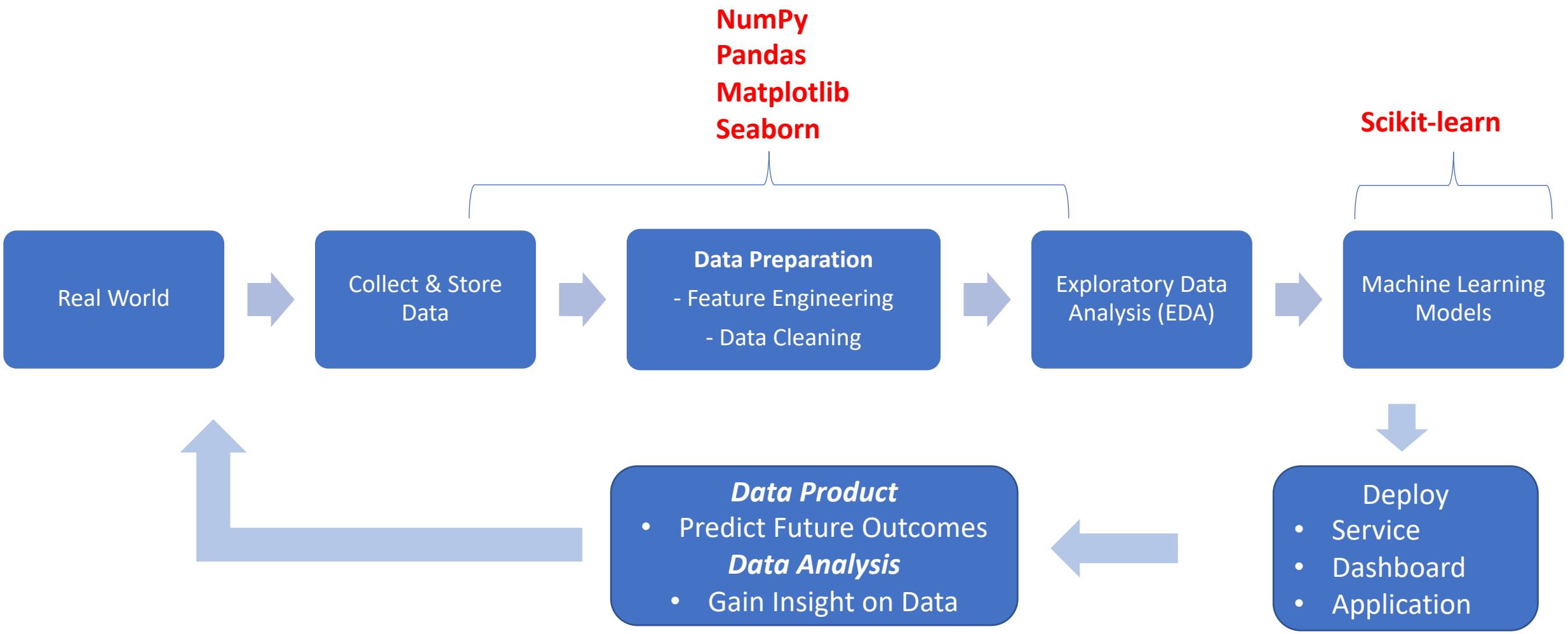
Outline

1. Introduction
2. Standard Neural Network
3. Set up ML/DL process
4. Convolutional Neural Network (If time permits)

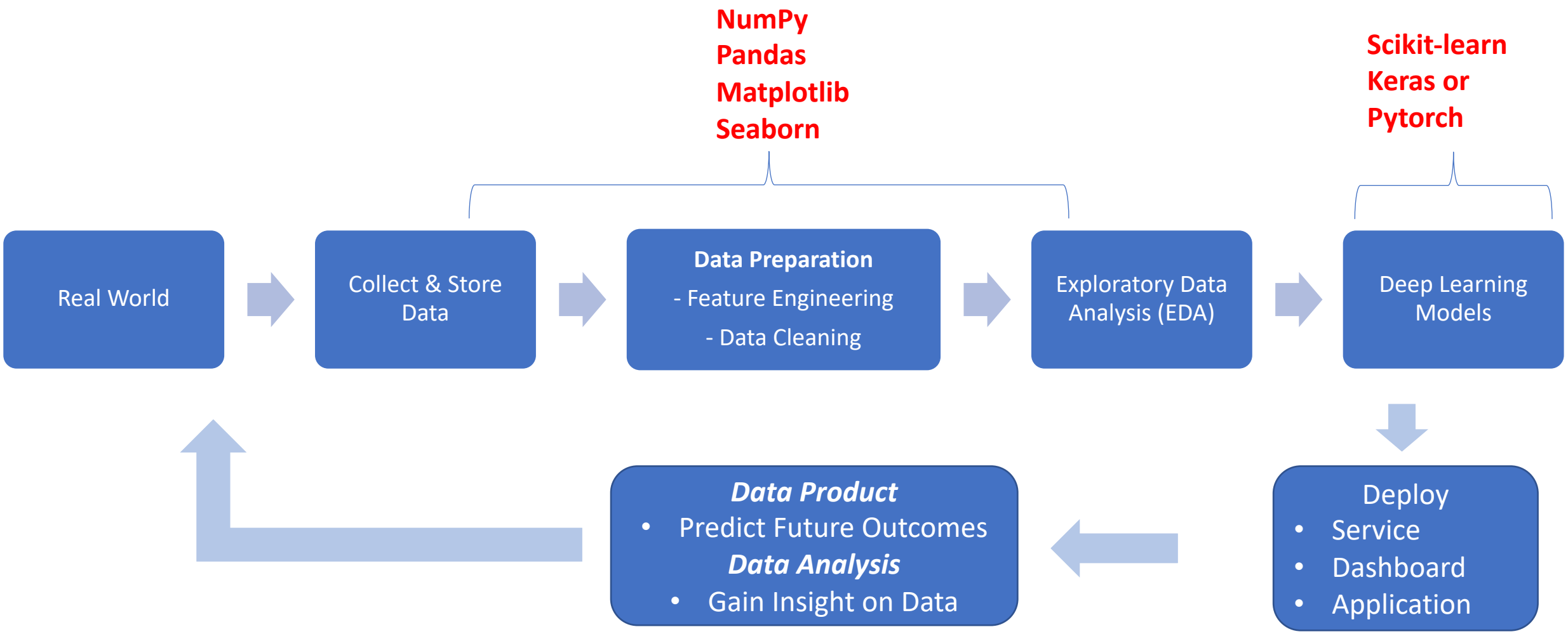
5. Generative Adversarial Networks (GAN)
6. Recurrent Neural Network in Natural Language Process (NLP)

Introduction

ML Pathway


NumPy
Pandas
Matplotlib
Seaborn
Scikit-learn
Real World
Collect & Store Data
Data Preparation
- Feature Engineering
- Data Cleaning
Exploratory Data Analysis (EDA)
Machine Learning Models
Deploy
• Service
• Dashboard
• Application
Data Product
• Predict Future Outcomes
Data Analysis
• Gain Insight on Data

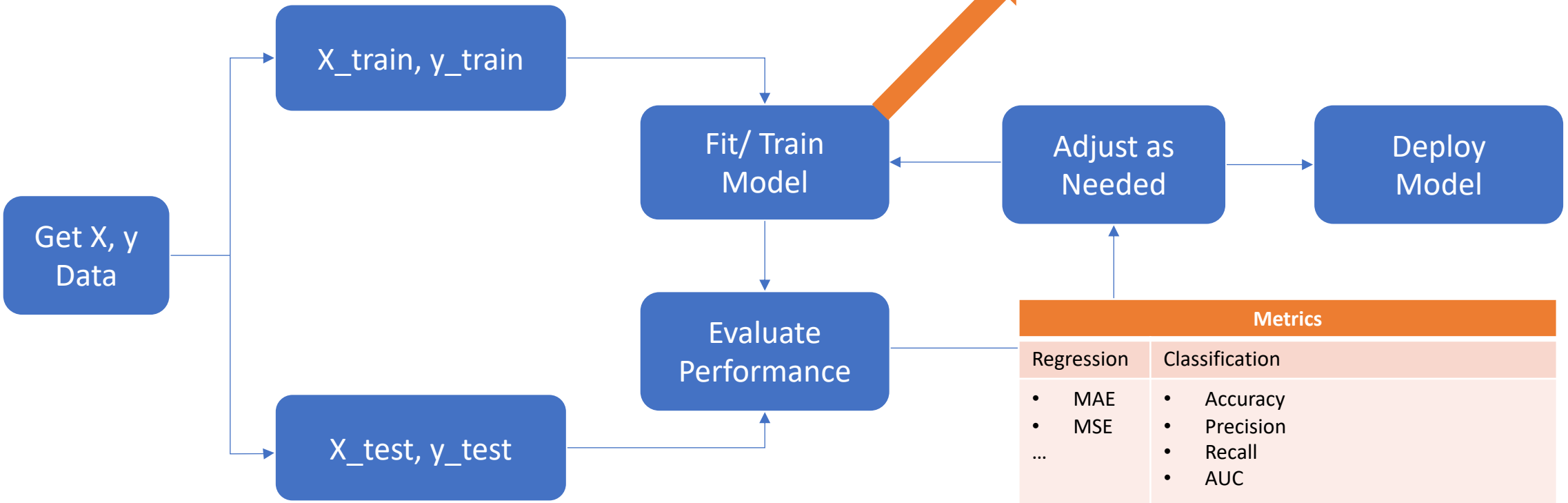
DL Pathway


NumPy
Pandas
Matplotlib
Seaborn
Scikit-learn
Keras or
Pytorch
Real World
Collect & Store Data
Data Preparation
- Feature Engineering
- Data Cleaning
Exploratory Data Analysis (EDA)
Deep Learning Models
Deploy
• Service
• Dashboard
• Application
Data Product
• Predict Future Outcomes
Data Analysis
• Gain Insight on Data

Goal: Find model parameters that minimize Loss/Cost function

ML Process

Scikit-learn **Scikit-learn**



- Loss/Cost function
- *Analytical* Solver

Loss function

Regression	Classification
• MAE • MSE …	• Binary Classification: Binary Cross-Entropy: $-\frac{1}{N}\sum_{i=1}^{N} y^{(i)} log\hat{y}^{(i)} + \left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)$ • Multi-Class Classification: ovr or Categorical Cross-Entropy(multinomial): $-\frac{1}{N}\sum_{i=1}^{N} log\left(\frac{e^{z_p}}{\sum_{j=1}^{C} e^{z_j}}\right)$

Metrics

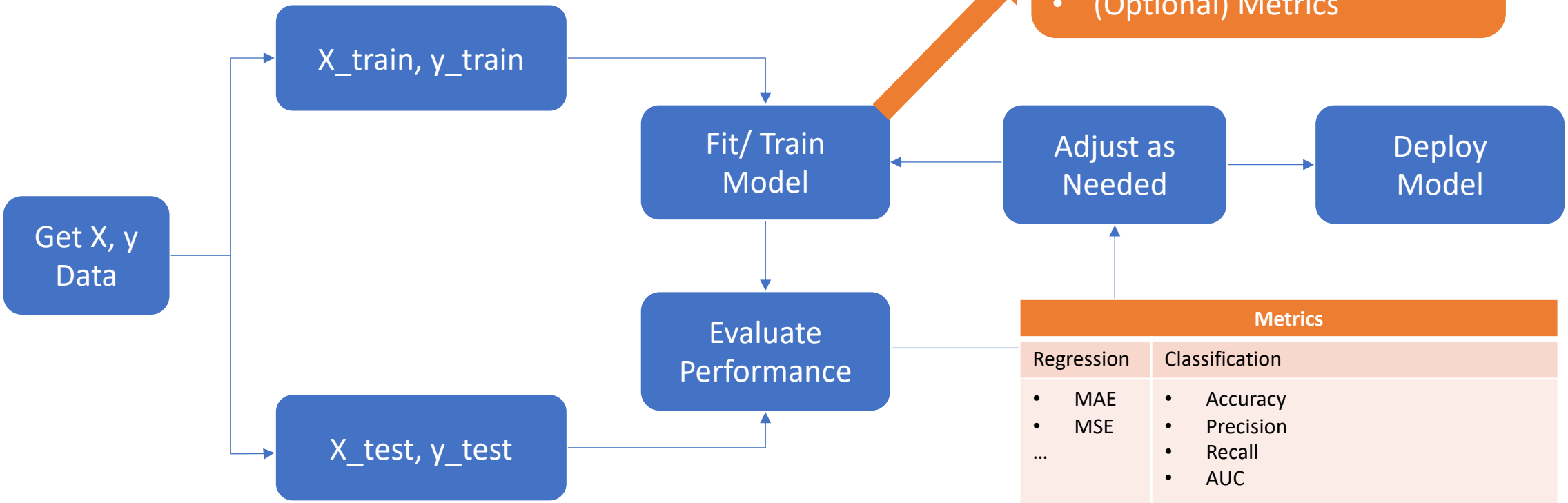
Regression	Classification
• MAE • MSE …	• Accuracy • Precision • Recall • AUC …

Goal: Find model parameters that minimize Loss/Cost function

DL Process

Scikit-learn

Scikit-learn Keras or Pytorch



- Loss/Cost function
- Optimizer: Gradient Descent
- (Optional) Metrics

Loss function

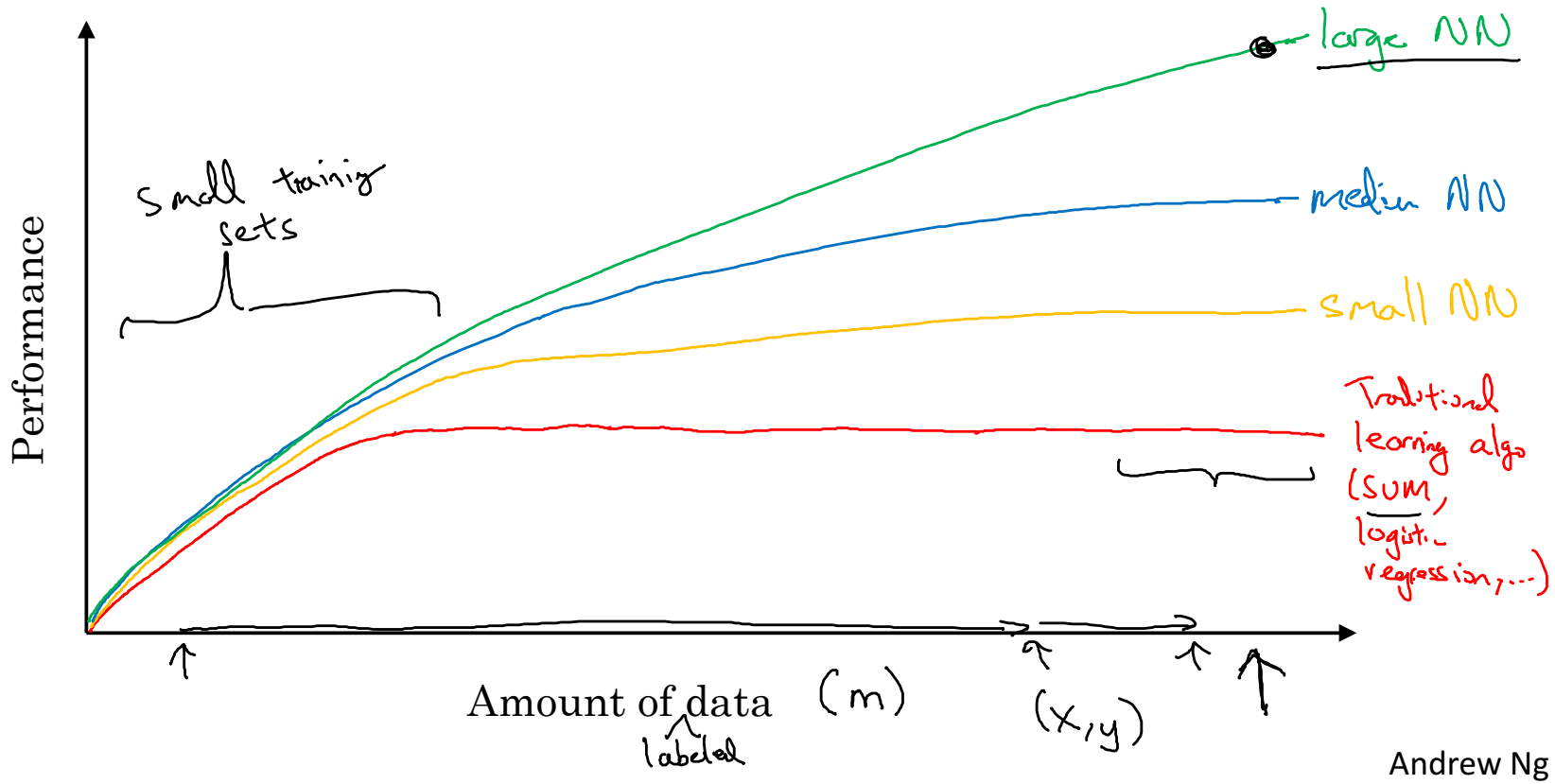
Regression	Classification
• MAE • MSE …	• Binary Classification: Binary Cross-Entropy: $-\frac{1}{N}\sum_{i=1}^{N} y^{(i)} log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})$ • Multi-Class Classification: Categorical Cross-Entropy: $-\frac{1}{N}\sum_{i=1}^{N} log\left(\frac{e^{z_p}}{\sum_{j=1}^{C} e^{z_j}}\right)$

Metrics

Regression	Classification
• MAE • MSE …	• Accuracy • Precision • Recall • AUC …

Why and when to use Deep Learning?

Scale drives deep learning progress

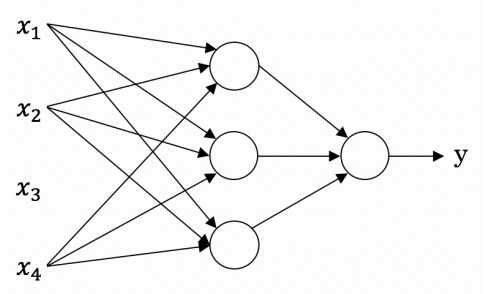


- Data
- Computation (GPU/TPU)
- Algorithms

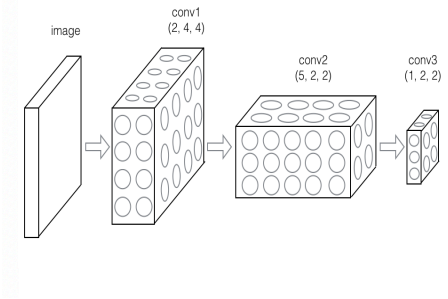
Andrew Ng

Neural Network examples

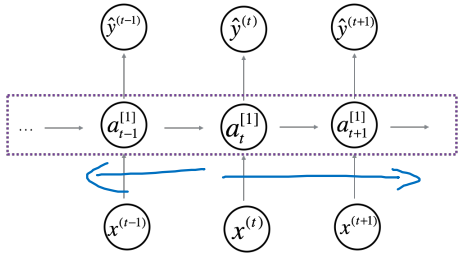
Input(x)	Output (y)	Application	
Home features	Price	Real Estate	Standard NN
Ad, user info	Click on ad? (0/1)	Online Advertising	
Image	Object (1,…,1000)	Photo tagging	CNN
Audio	Text transcript	Speech recognition	RNN
English	Chinese	Machine translation	
Image, Radar info	Position of other cars	Autonomous driving	Custom/ Hybrid



Standard NN



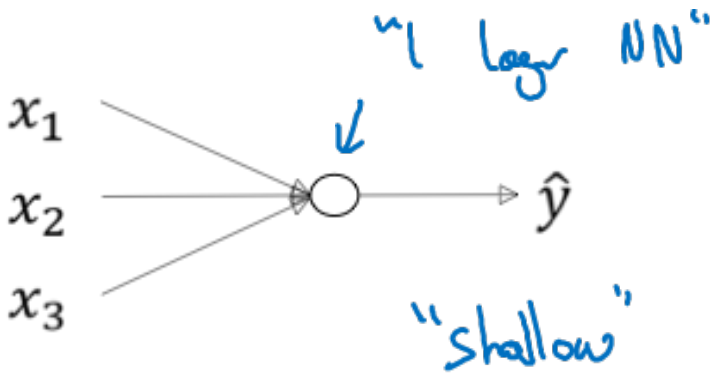
Convolutional NN



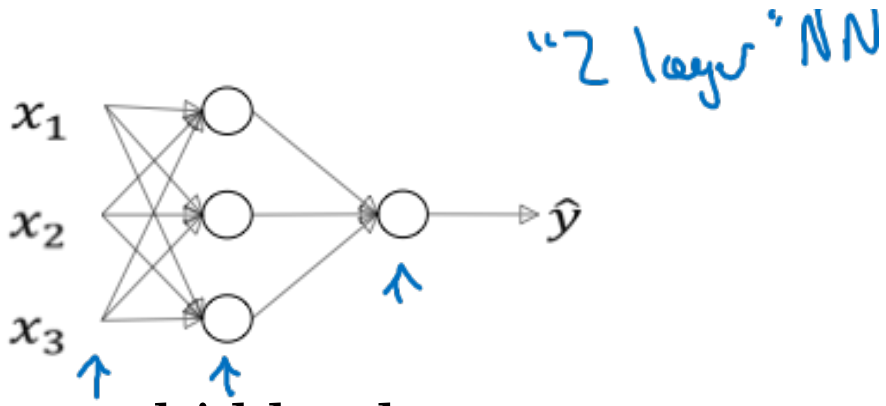
Recurrent NN

Standard Neural Network (ANN)

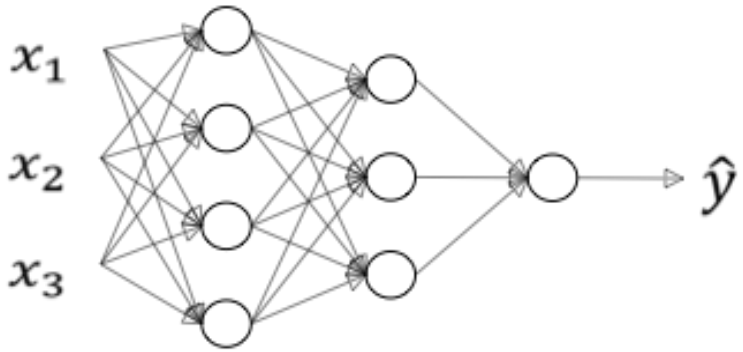
Deep Neural Networks: # of hidden layers >= 2



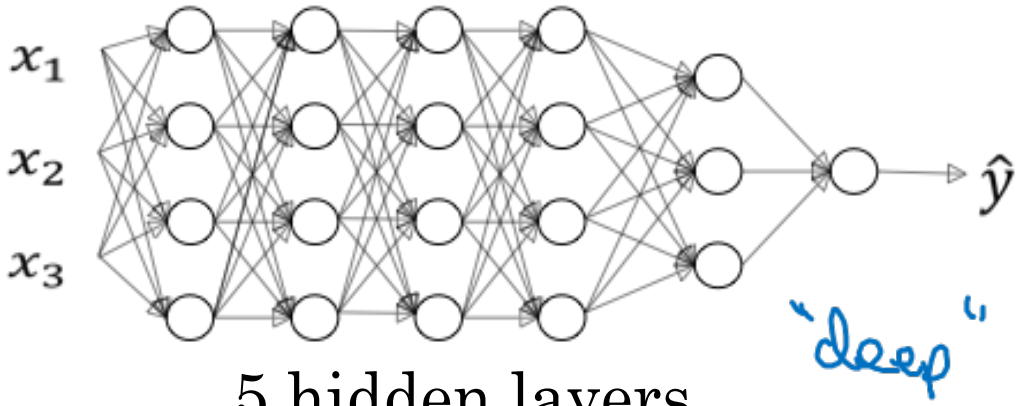
logistic regression



1 hidden layer



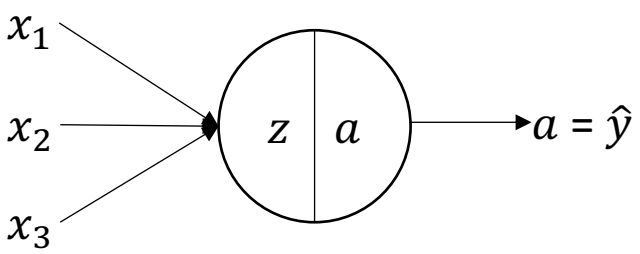
2 hidden layers



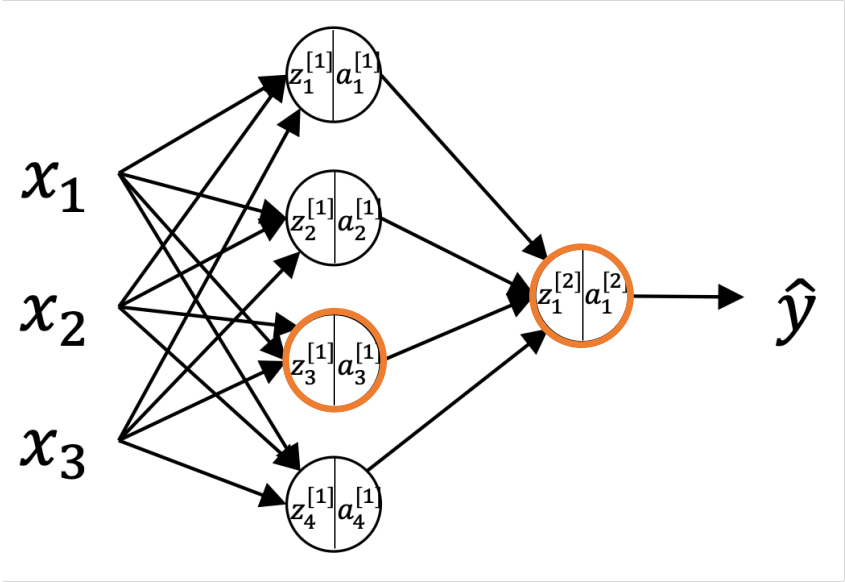
5 hidden layers

Neural Network Representation

: Linear combination(z) + Non-linear activation (a)



$$z = w^T x + b$$
$$a = g(z) = \hat{y}$$

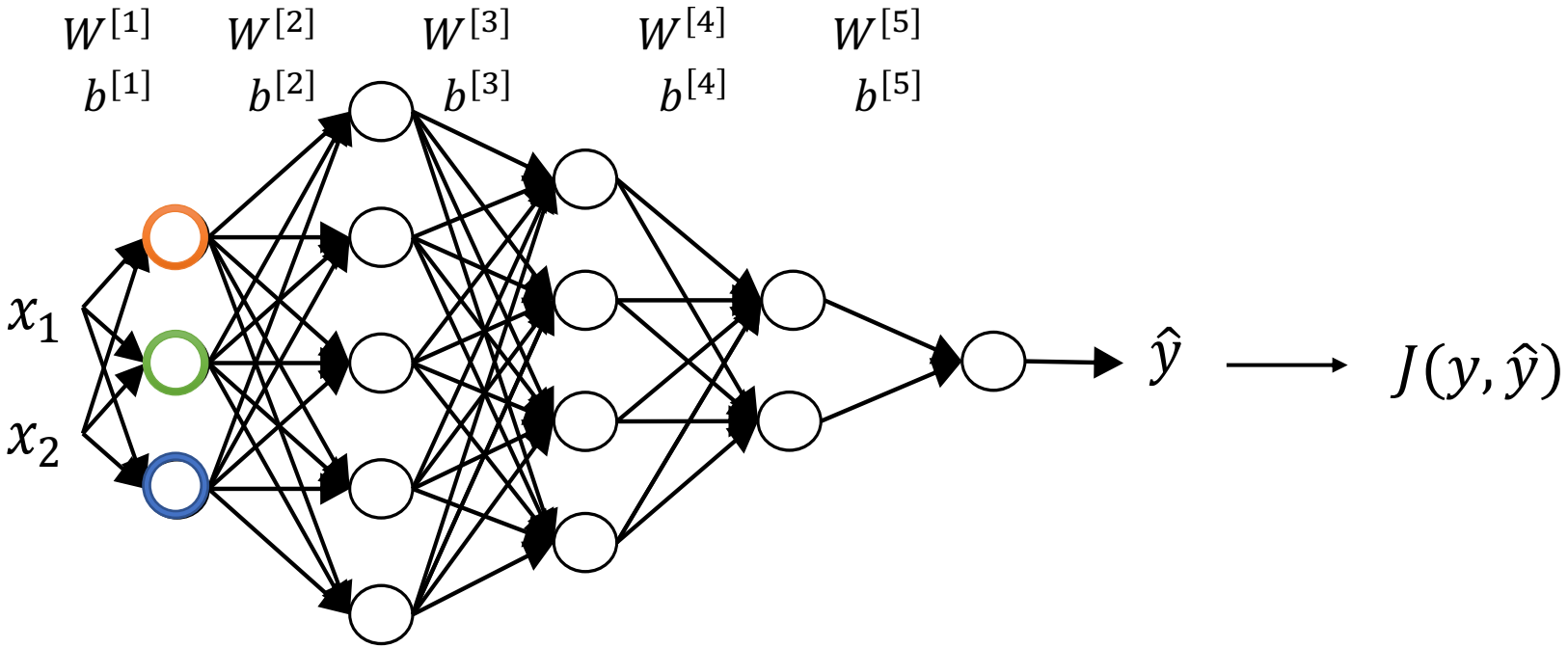


$$z_3^{[1]} = w^T x + b$$
$$a_3^{[1]} = g(z_3^{[1]})$$

$$z_1^{[2]} = w^T a^{[1]} + b$$
$$a_1^{[2]} = g\left(z_1^{[2]}\right) = \hat{y}$$

Neural Network Representation

: Linear combination(z) + Non-linear activation (a)



$$W^{[1]} = \begin{bmatrix} \circ & \circ \\ \circ & \circ \\ \circ & \circ \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} \circ \\ \circ \\ \circ \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$z^{[1]} = W^{[1]}x + b^{[1]} = \begin{bmatrix} \circ \\ \circ \\ \circ \end{bmatrix}$$

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

$$X = [x^{(1)} \quad \dots \quad x^{(N)}]$$

$$Z^{[1]} = W^{[1]}X + b^{[1]} = \begin{bmatrix} \circ & & \circ \\ \circ & \dots & \circ \\ \circ & & \circ \end{bmatrix}$$

$$A^{[1]} = g(Z^{[1]})$$

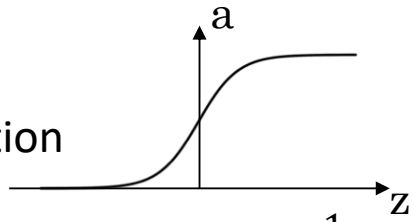
Activation function

- Why need non-linear activation function?

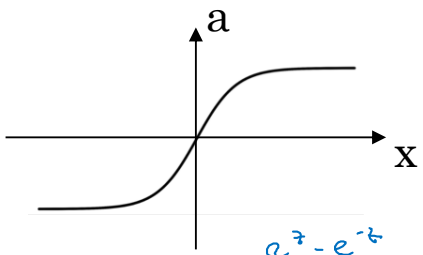
  Limit output ranges, Without it, Linear + Linear = Linear

- Strategy: Use relu + proper learning rate, Try out Leaky relu, elu,

Output layer
for classification



sigmoid: $a = \dfrac{1}{1+e^{-z}}$

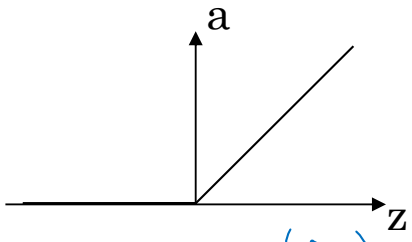


tanh: $a = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K$$

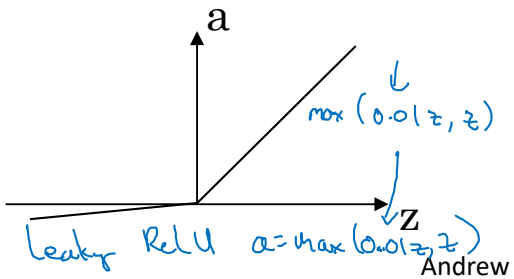
softmax

Hidden layer



ReLU $a = \max(0, z)$

Rectified Linear Unit

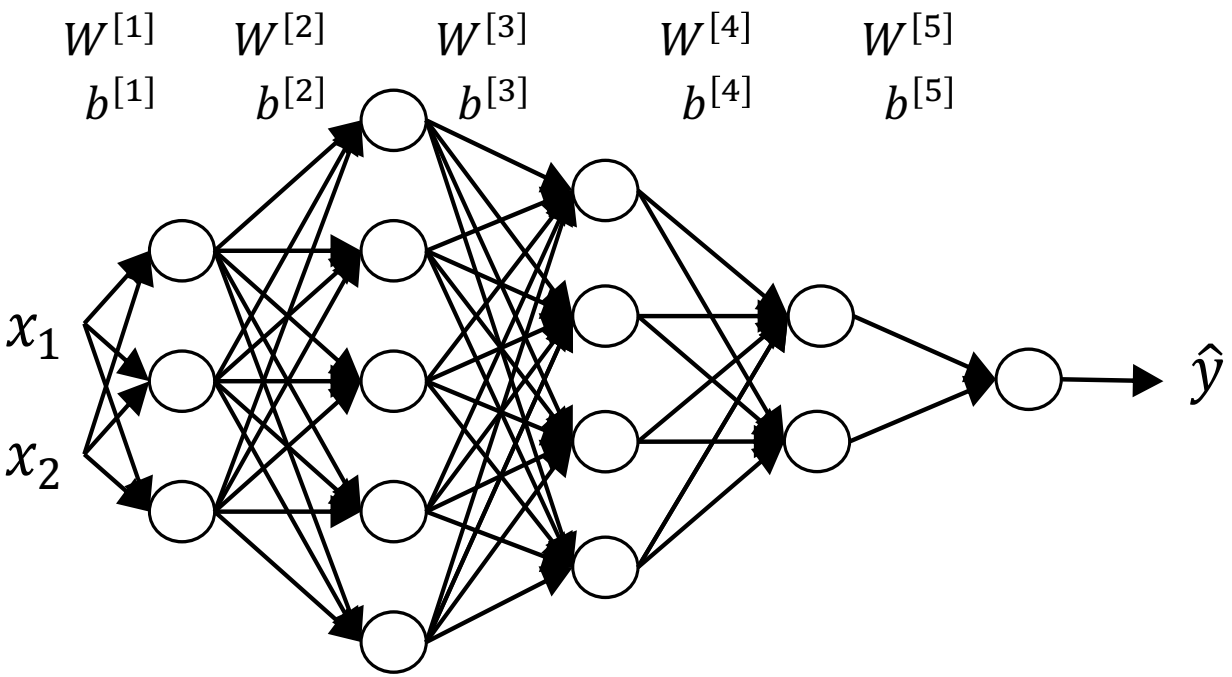


Leaky ReLU $a = \max(0.01z, z)$

Andrew Ng

elu (Exponential Linear Unit)
selu (Scaled elu)
softplus
softsign
…

Forward Propagation



$\longrightarrow J(y, \hat{y})$

So far…

Parameters: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$, $W^{[3]}$, $b^{[3]}$ …

Cost function: $J(W, b) = \frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, \hat{y}^{(i)})$

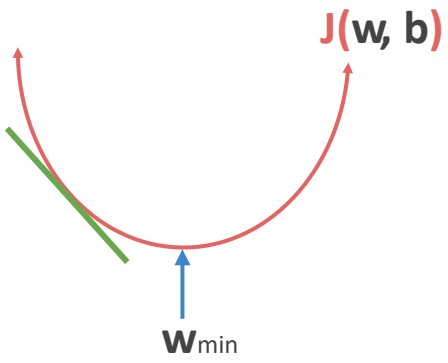
Hyperparameters: # of layers, L

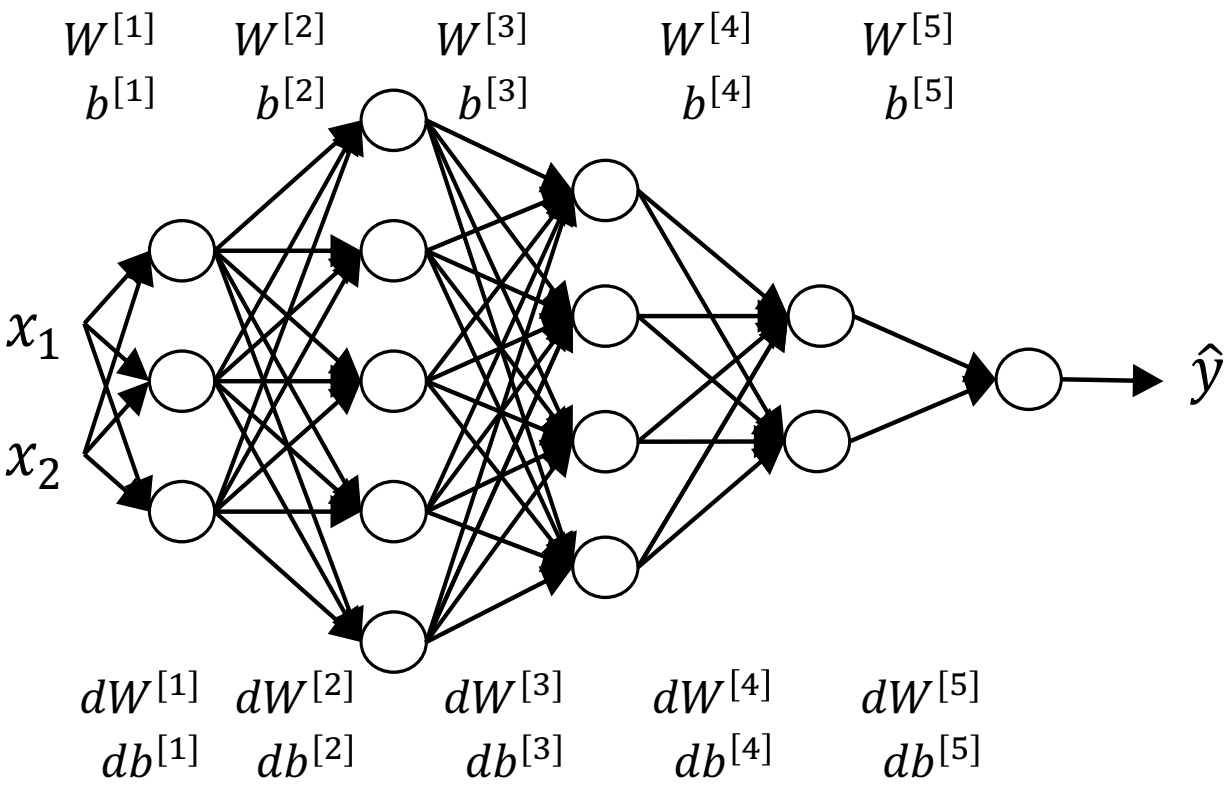
# of units in layer $l, n^{[l]}$

activation function

$W^{[l]}$ initialization & regularization

Backward Propagation





So far…

Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ …

Cost function: $J(W,b) = \frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, \hat{y}^{(i)})$

$\longrightarrow \quad J(y, \hat{y})$

Gradient Descent to optimize parameters to reduce $J(y, \hat{y})$

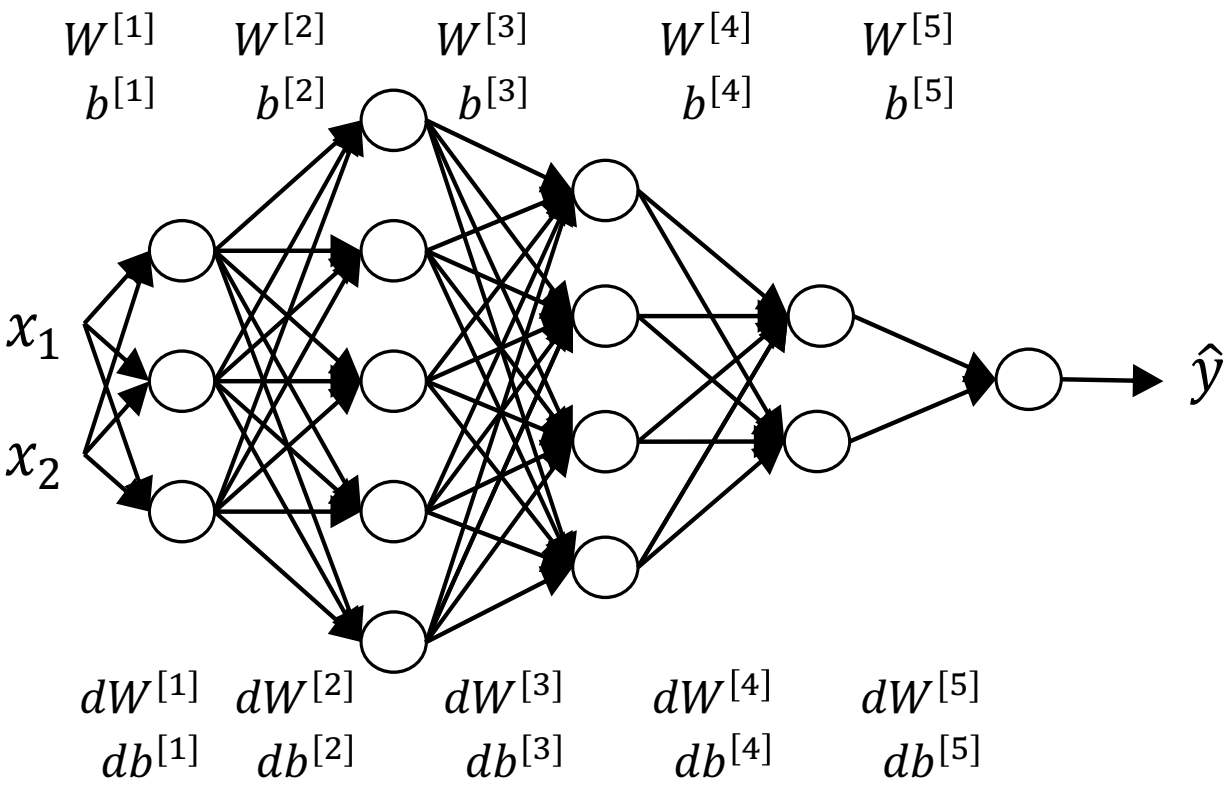
- Compute Gradients: $\frac{\partial J}{\partial W^{[5]}}, \frac{\partial J}{\partial b^{[5]}}, \frac{\partial J}{\partial W^{[4]}}, \frac{\partial J}{\partial b^{[4]}}, \ldots$
- Update:

$$W^{[5]} = W^{[5]} - \alpha \frac{\partial J}{\partial W^{[5]}}$$

$$b^{[5]} = b^{[5]} - \alpha \frac{\partial J}{\partial b^{[5]}}$$

$$\vdots$$

(Forward + Backward Propagation) x (m//BATCH_SIZE+1) x epoch



So far…

Parameters: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$, $W^{[3]}$, $b^{[3]}$ …

Cost function: $J(W,b) = \frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, \hat{y}^{(i)})$

Using Gradient Descent, update parameters:

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial J}{\partial W^{[l]}}$$

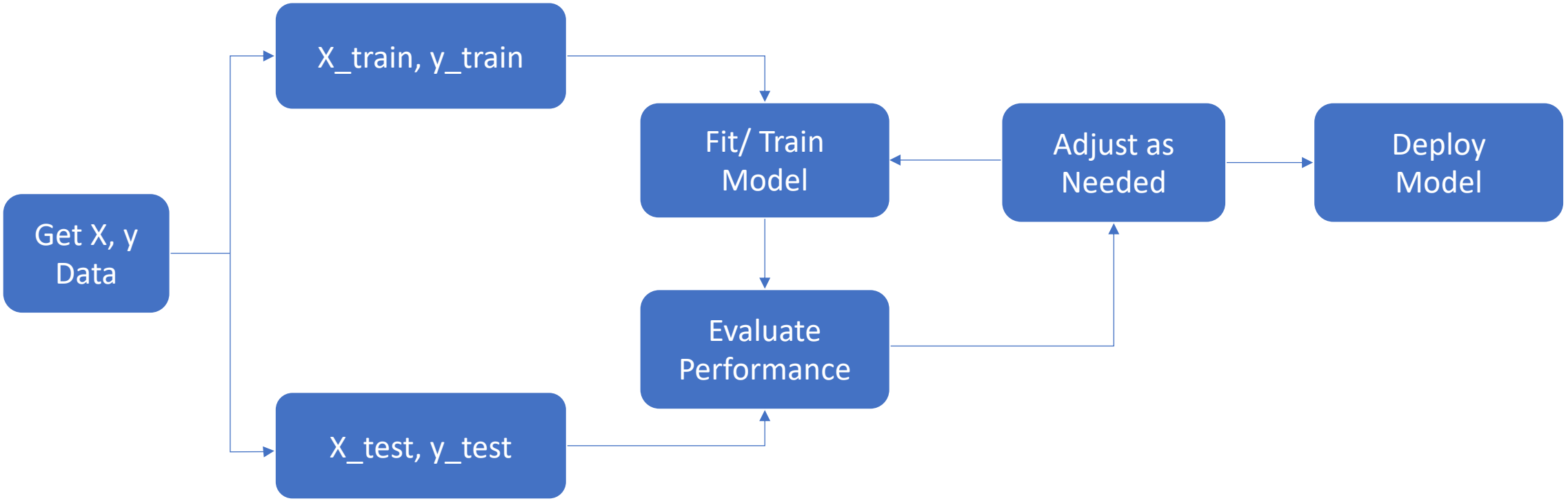
$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

$\longrightarrow J(y, \hat{y})$

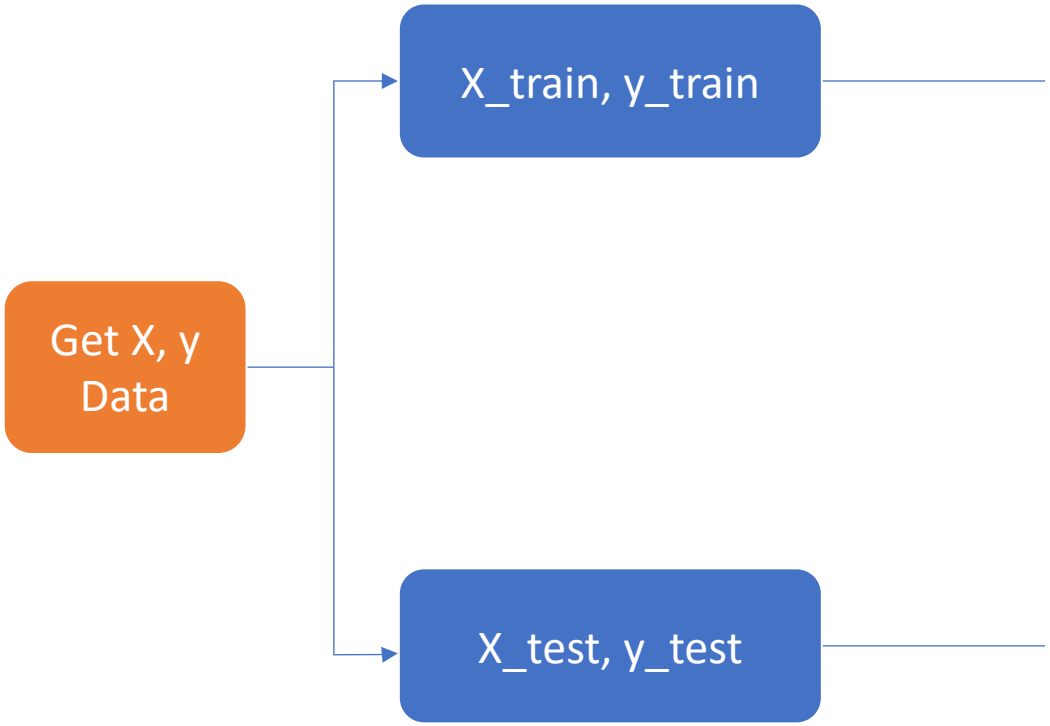
Hyperparameters:
- # of layers, L
- # of units in layer $l, n^{[l]}$
- activation function
- $W^{[l]}$ initialization & regularization
- Optimizer (gradient descent method)
- Learning rate, α
- Learning rate decay
- batch size, N = 64, 128, 256, 512, …
- # of iterations: epochs

Set up ML/DL process

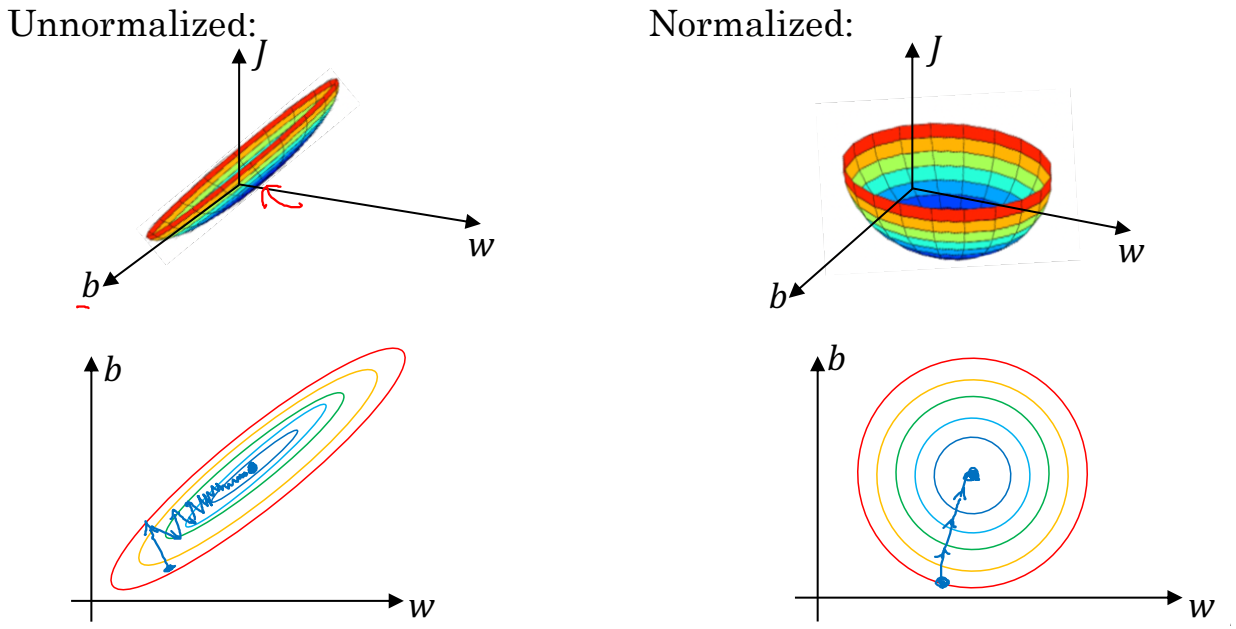
Set up ML/DL process



Set up ML/DL process

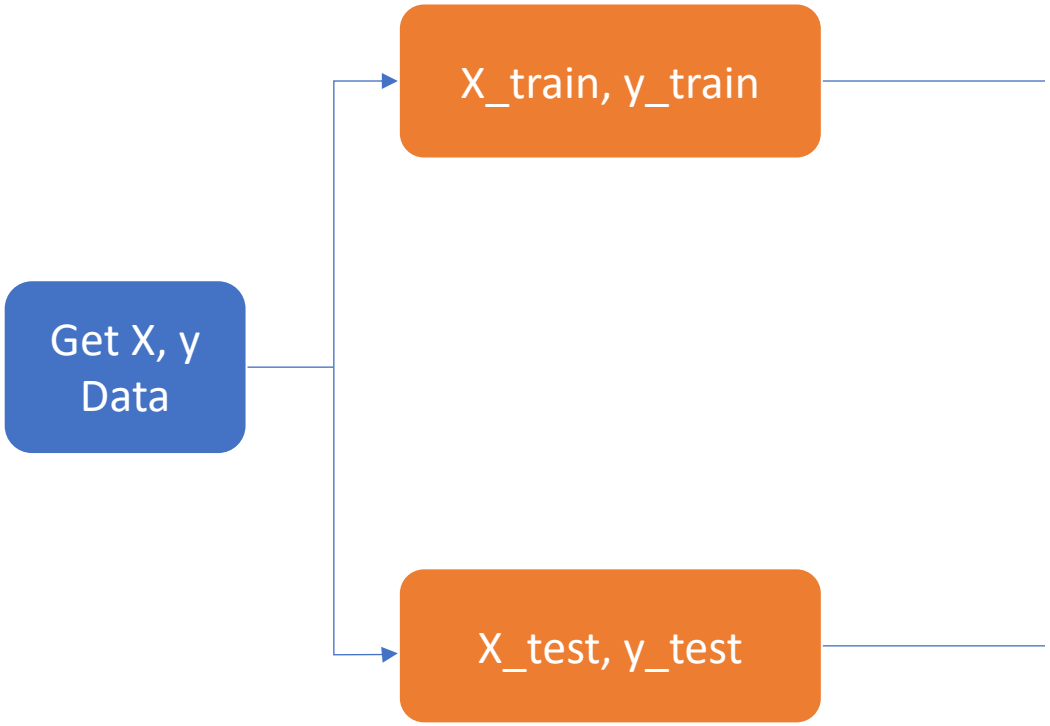


- ❑ **Normalize inputs($a^{[0]}$): To speed up learning**



- ❑ **(Later on) Batch Normalization ($z^{[l]}$)**

Set up ML/DL process



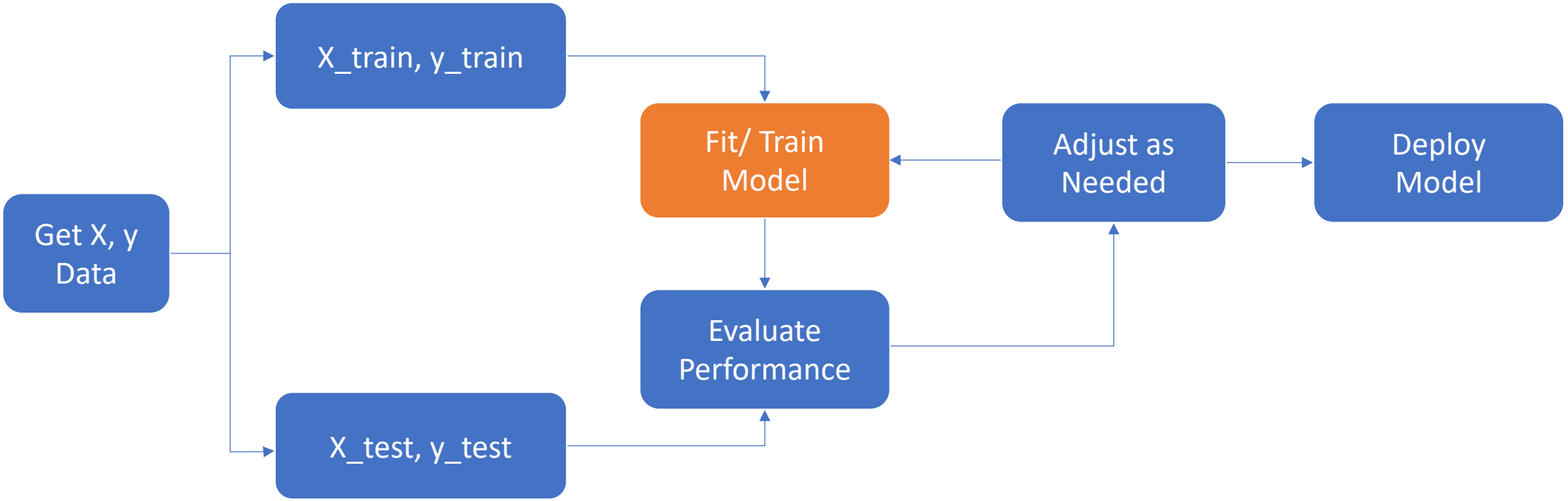
❑ train test split

- Small data = 100, 1K, 10K
  - train/test = 70/30%
  - train/valid/test = 60/20/20%
- Big data = 10K, 100K, 1M
  - train/valid/test = 98/1/1%
    = 99.5/0.4/0.1%

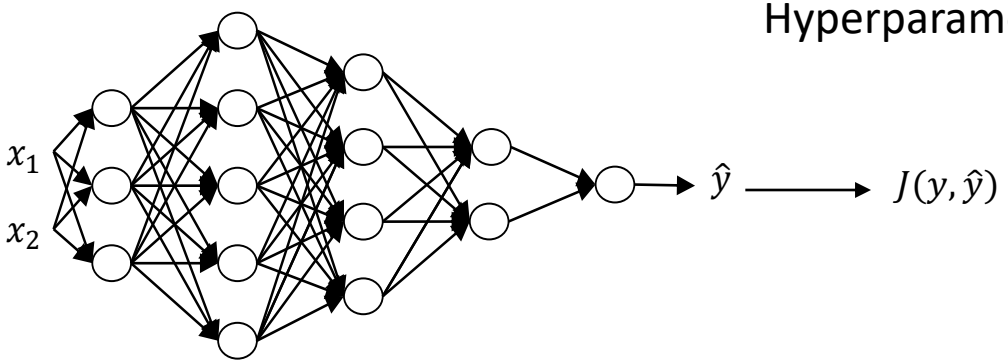
➢ Make sure dev and test sets come from the same distribution

Set up ML/DL process

Hyperparameters: # of layers, L
# of units in layer $l, n^{[l]}$
activation function
$W^{[l]}$ initialization & regularization
Optimizer (gradient descent method)
Learning rate, α
Learning rate decay
batch size, N = 64, 128, 256, 512, …
# of iterations: epochs



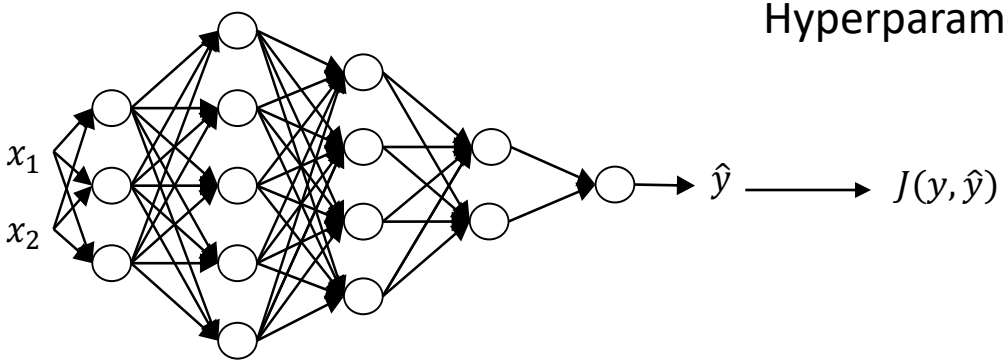
Fit/ Train Model



Hyperparameters: # of layers, L
# of units in layer $l, n^{[l]}$
activation function
$W^{[l]}$ initialization & regularization
Optimizer (gradient descent method)
Learning rate, α
Learning rate decay
batch size, N = 64, 128, 256, 512, …
# of iterations: epochs

- # of layers, L
- tf.keras.layers.Dense(units, activation=None, kernel_initializer="glorot_uniform", bias_initializer="zeros")
  - # of units in layer $l, n^{[l]}$
  - activation function: relu, leacky relu, elu, selu,… for hidden layers, sigmoid, softmax, tanh for classification output layer
  - $W^{[l]}$ initialization: random & small to prevent vanishing/exploding gradients
    - glorot uniform(Xavier uniform initializer): [-limit, limit], where limit = sqrt(6 / (fan_in + fan_out),
    - glorot normal: stddev = sqrt(2 / (fan_in + fan_out))
    - random uniform, random normal, truncated normal

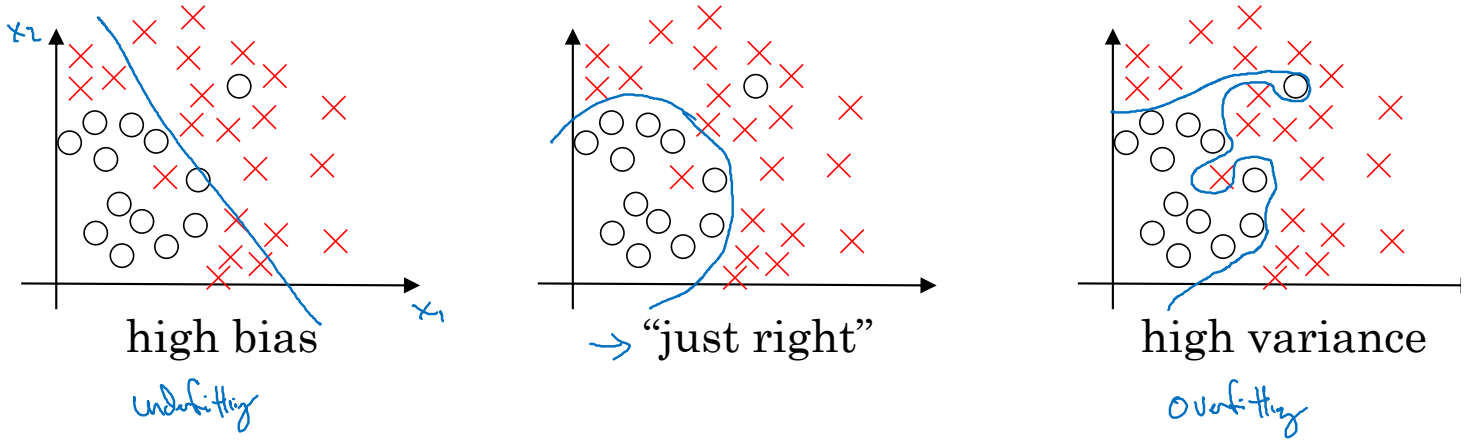
Fit/ Train Model

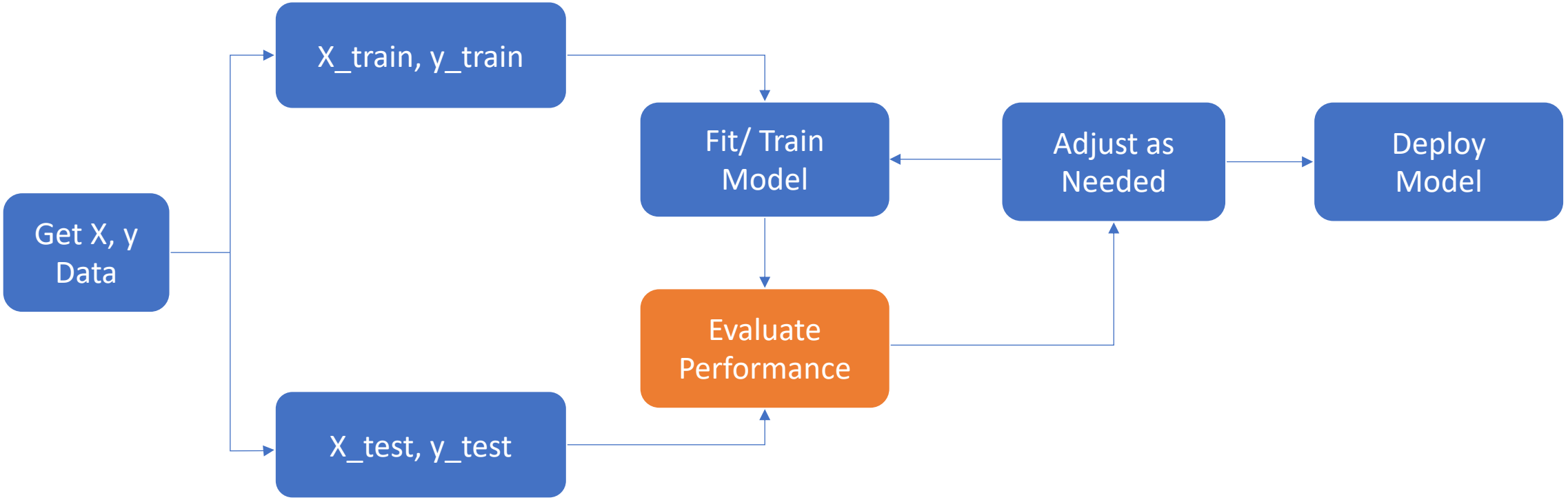


Hyperparameters: # of layers, L
# of units in layer $l, n^{[l]}$
activation function
$W^{[l]}$ initialization & regularization
Optimizer (gradient descent method)
Learning rate, α
Learning rate decay
batch size, N = 64, 128, 256, 512, …
# of iterations: epochs

- Model.compile(optimizer="rmsprop", loss=None, metrics=None)
  - Learning rate, α = 0.0001, 0.001, 0.01, 0.1, 1
  - Optimizer (gradient descent method):
    - Gradient Descent with moment (SGD): lr = 0.01, momentum(beta_1, dW)=0
    - RMSprop: lr=0.001, rho(beta_2, dW^2)=0.9, momentum=0, epsilon=1e-7
    - Adam: lr = 0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7
- Model.fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None)
  - Learning rate decay: Exponential decay, Piecewise Constant Decay, …
    tf.keras.callbacks.LearningRateScheduler(schedule, verbose=0)
    tf.keras.callbacks.ReduceLROnPlateau(…)
  - batch size, N = 64, 128, 256, 512, …
  - # of iterations: epochs

Set up ML/DL process





Evaluate Performance

Reducing (avoidable) bias and variance

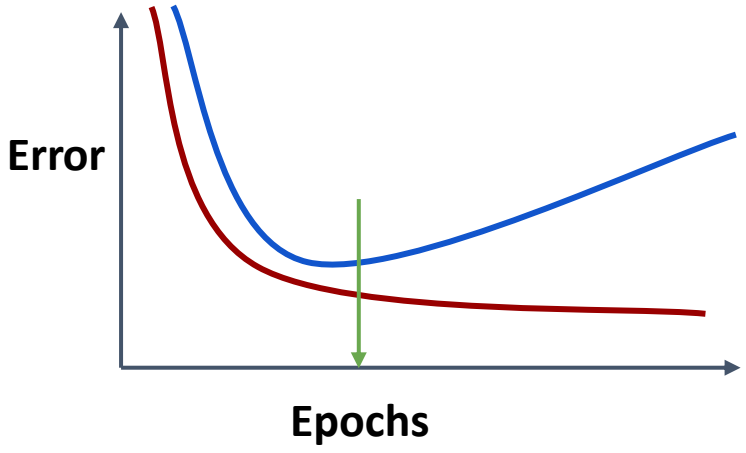
Human-level loss	**1%**	**1%**	
Avoidable bias			• Train bigger model • Train longer • Better optimization algorithms (SGD with momentum, RMSprop, Adam..) • NN architecture/hyperparameters search
Training loss	**5%**	**1%**	

Variance			• More data
			• Early stopping
			• Regularization
			  • L1, L2, L1_L2 Regularization
			  • Drop-out
			  • (Batch Normalization)
			  • Data augmentation
			• NN architecture/hyperparameters search

Valid loss	**6%**	**5%**	

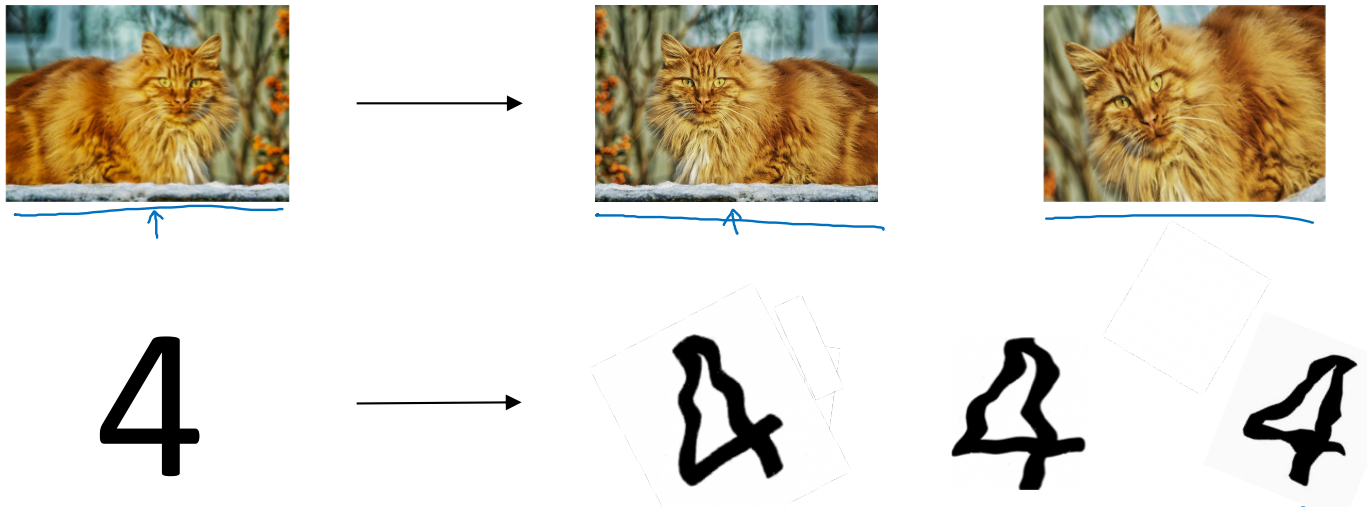
Reduce Variance (Overfitting)

- Early Stopping: tf.keras.callbacks.EarlyStopping(…)



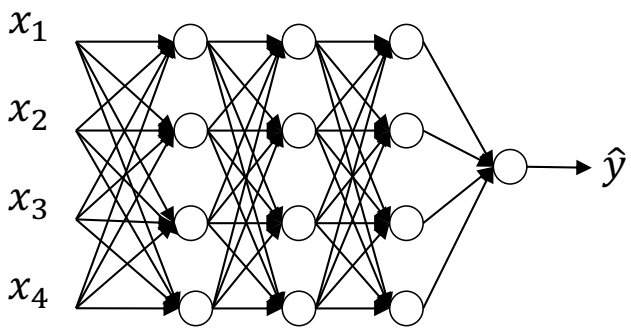
- Data Augmentation:

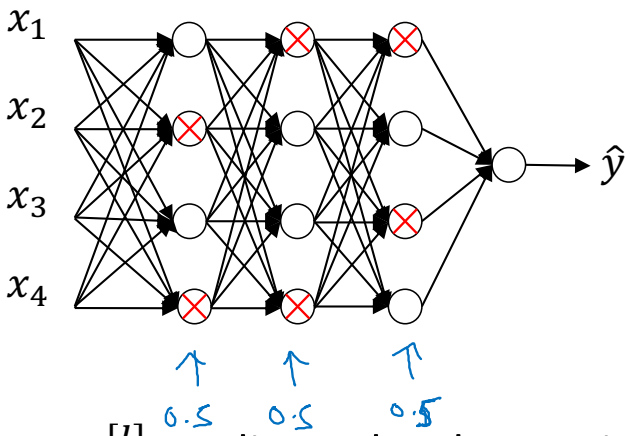
  tf.keras.preprocessing.image.ImageDataGenerator(…



Reduce Variance (Overfitting)

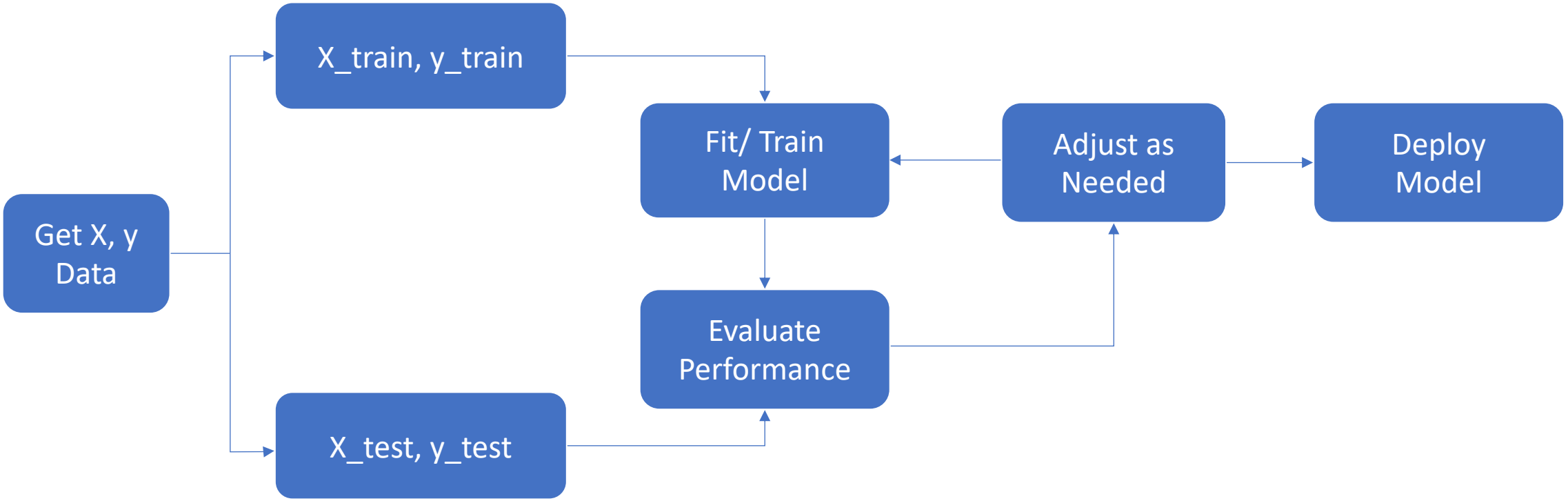
- Dropout regularization: applies only when training
  tf.keras.layers.Dropout(rate, noise_shape=None, seed=None, **kwargs)





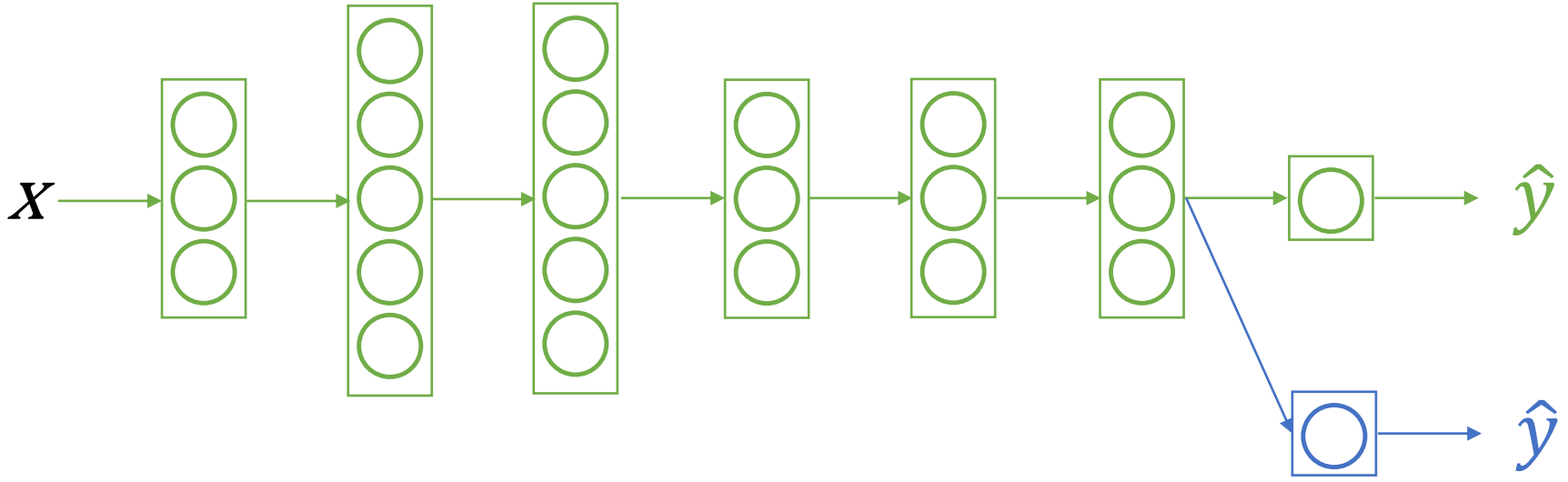
- Batch Normalization: speed up learning, after $z^{[l]}$, before $a^{[l]}$ applies only when training
  tf.keras.layers.BatchNormalization(…)
  - Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
  - This adds some noise to the values $z^{[l]}$ within that minibatch similar to dropout. This has a slight regularization effect.

Set up ML/DL process



Transfer learning + Fine tuning

- Task A and B have the same(or similar) input x
- The amount of data Task A >> Task B
- Low level features from Task A can be helpful for learning Task B

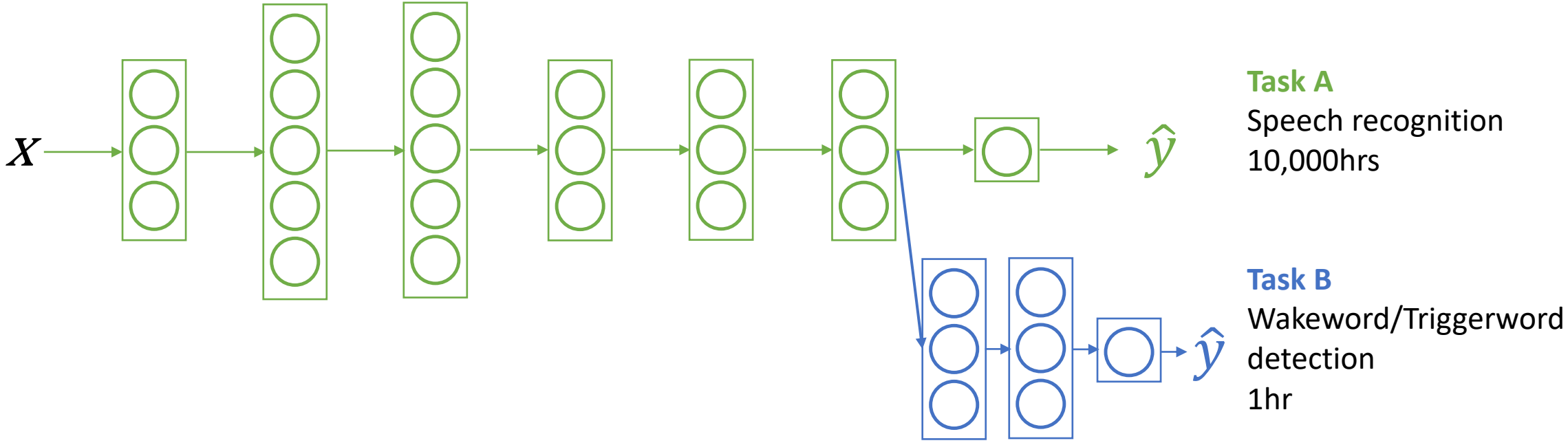


Task A
Image recognition
1,000,000

Task B
Radiology diagnosis
1,000

Transfer learning + Fine tuning

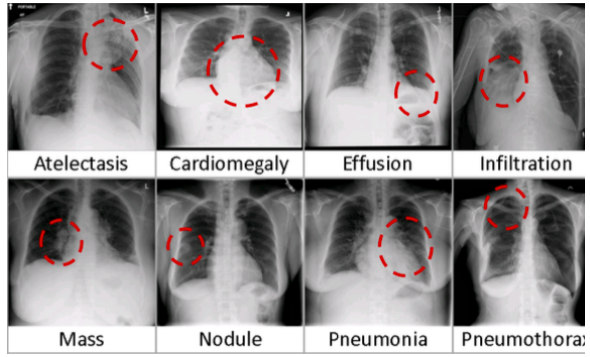
- Task A and B have the same(or similar) input x
- The amount of data Task A >> Task B
- Low level features from Task A can be helpful for learning Task B

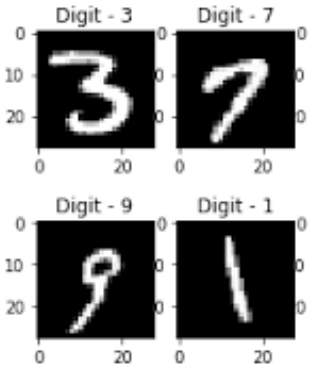


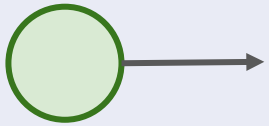
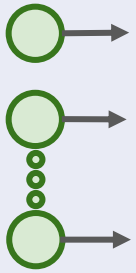
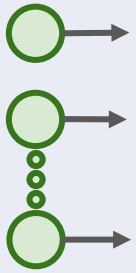
Probably it’s already over timed.
Thank you! Any question?

Do you want to know Convolutional Neural Network (CNN)?

Multi-Task Classification





	Binary Classification	Multi-label Classification (Mutually Non-Exclusive Classes)	Multi-Class Classification (Mutually Exclusive Classes)
	y=[car] y=[pneumonia]	y=[car, stop sign, traffic lights, pedestrians,…] y=[pneumonia, atelectasis, cardiomegaly,…]	y=[0, 1, 2, 3, 4, 5, 6 7, 8, 9]
	$y = [0 \ or \ 1]$ 	$y = \begin{bmatrix} 0 \ or \ 1 \\ \vdots \\ 0 \ or \ 1 \end{bmatrix}$ 	$y = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$ Only one class is 1 
Output layer Activation function	Sigmoid $\hat{y} = \frac{1}{1 + e^{-z}}$	Sigmoid $\hat{y}_j = \frac{1}{1 + e^{-z_j}}$ for $j = 1, \dots, C$	Softmax $\hat{y}_j = \frac{e^{z_j}}{\sum_{j=1}^{C} e^{z_j}}$ for $j = 1, \dots, C$
Cost function	$-\frac{1}{N}\sum_{i=1}^{N} y^{(i)} log\hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$	$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_j^{(i)} log\hat{y}_j^{(i)} + \left(1 - y_j^{(i)}\right) \log\left(1 - \hat{y}_j^{(i)}\right)$	$-\frac{1}{N}\sum_{i=1}^{N} log\left(\frac{e^{z_p}}{\sum_{j=1}^{C} e^{z_j}}\right)$

Convolutional Neural Network (CNN)

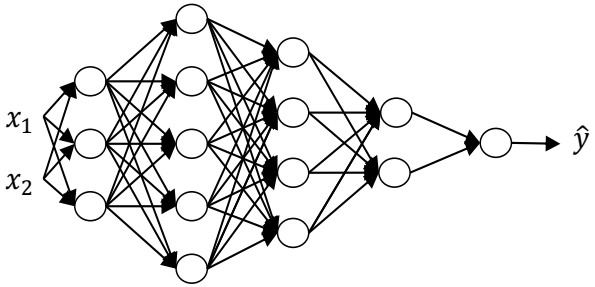
- Conv layer – Pooling layer – Fully-connected layer
- Classic Networks

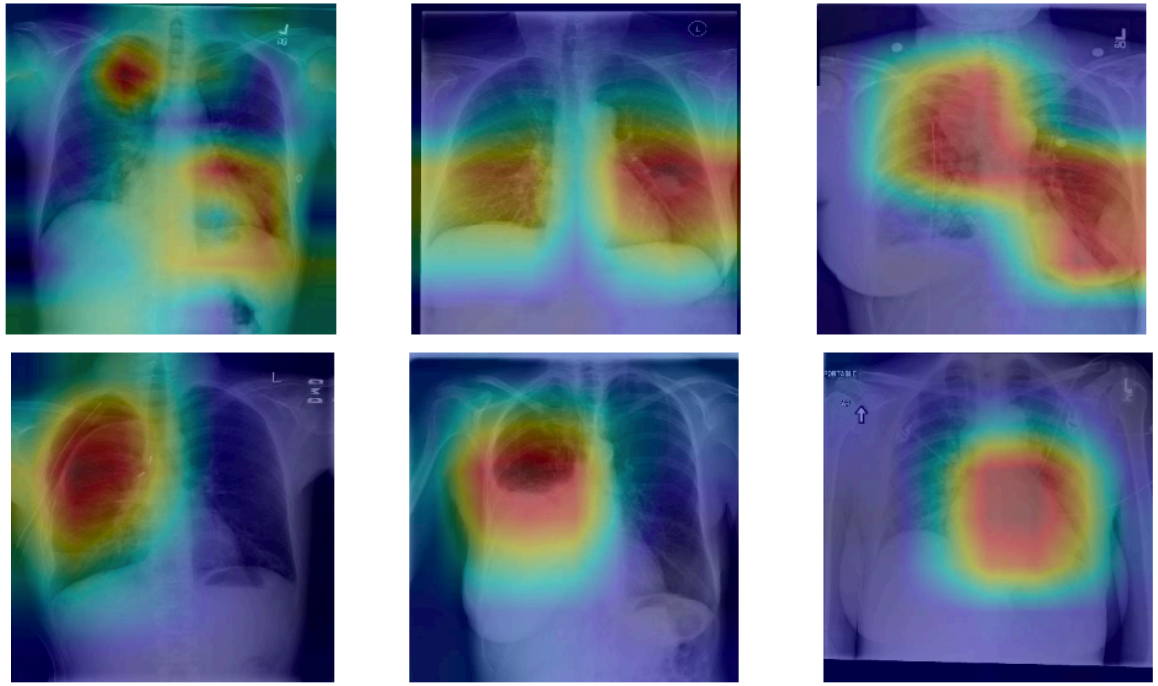
Computer Vision problems

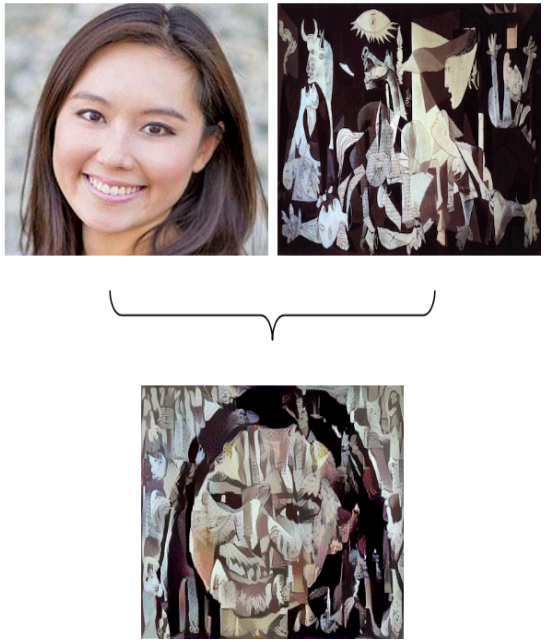
- Image Classification
- Object Detection
- Neural Style transfer

Cat? (0/1)

Image size: 1000 x 1000 x 3 = 3M
$n^{[0]}$ = 3M, $n^{[1]}$ = 1000
$W^{[1]}$ =:(1000, 3M) = 3 billion







➢ Solution: Convolution

Convolution

2D

Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

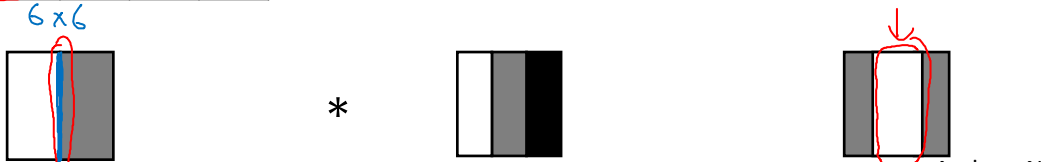
1	0	-1
1	0	-1
1	0	-1

3 x 3

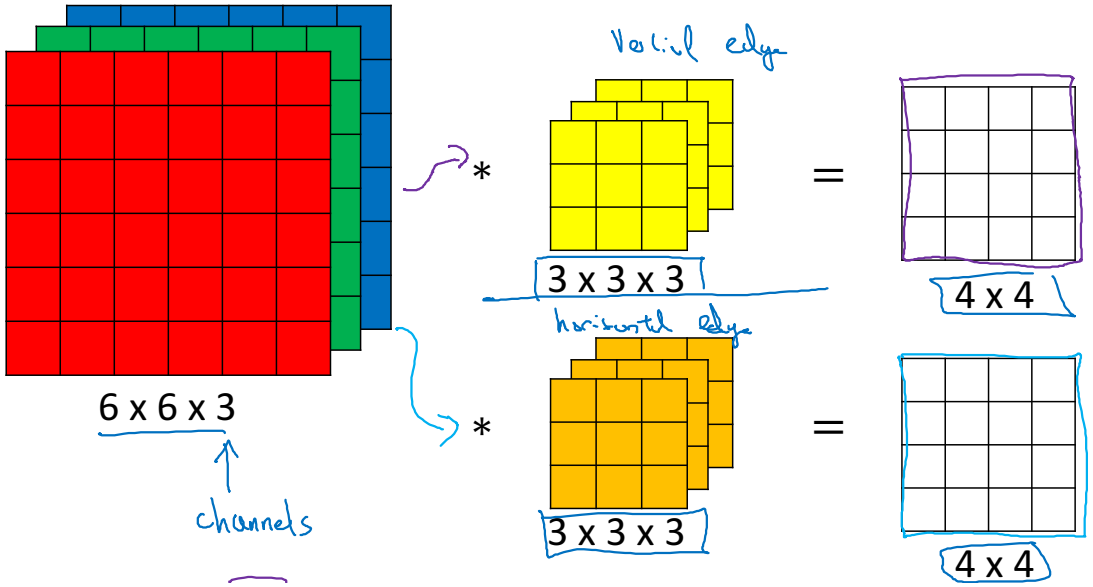
=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

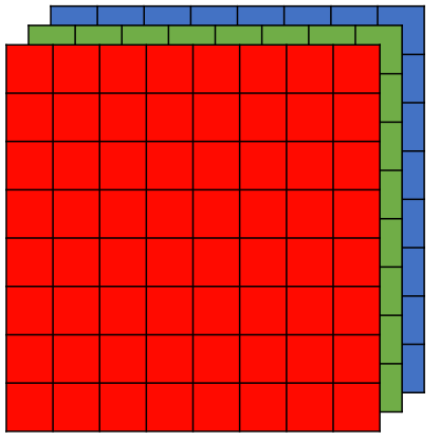
4x4



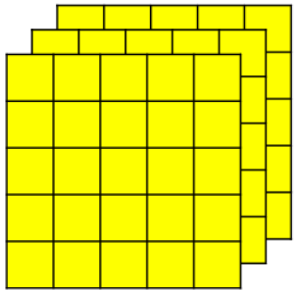
2D over volumes



Convolution with padding and stride

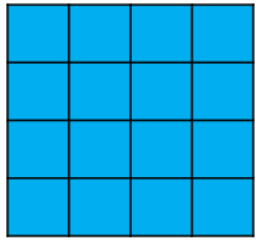


*

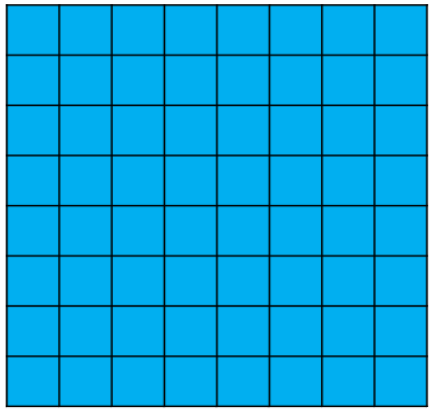


=

"valid"
p = 0



"same"

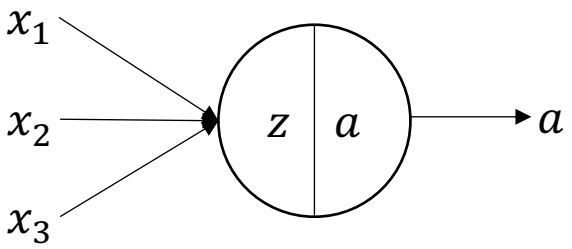


Input image size: n x n x 3
padding p or padding='valid' or 'same'

kernel size: f x f x 3
stride s

Output size: $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$x $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$x 1

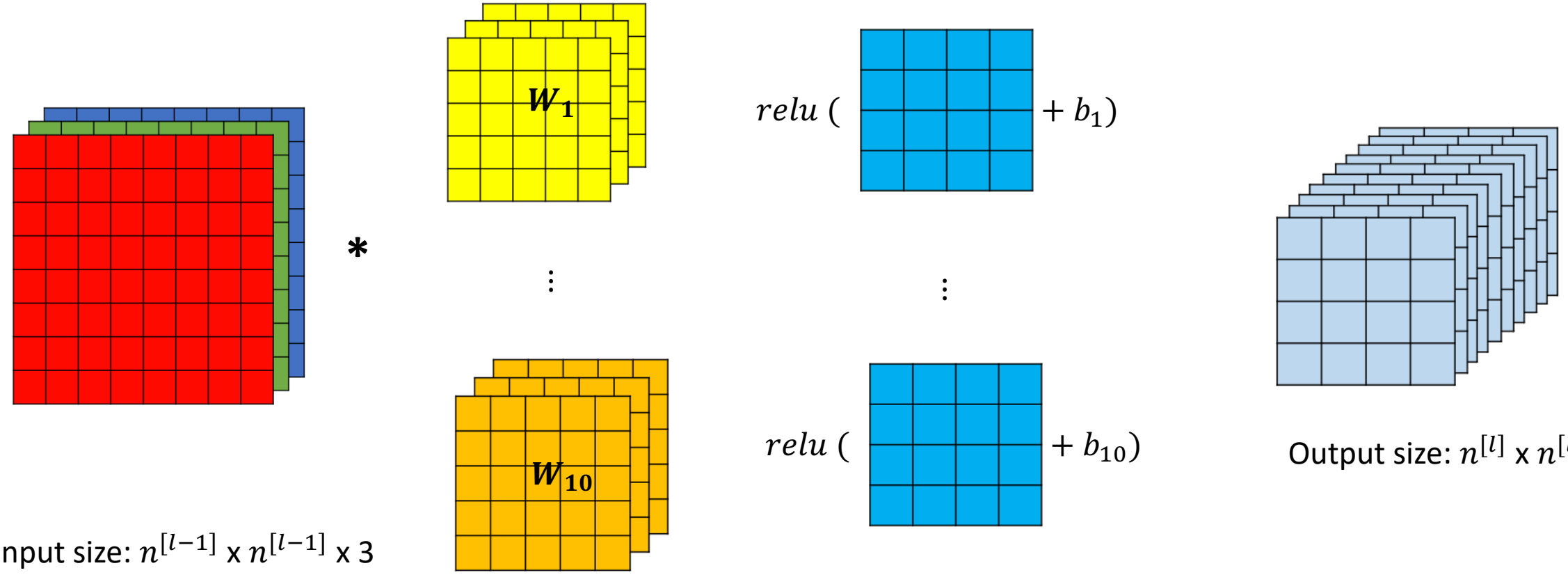
Convolution layer



Why convolution?

- Parameter sharing
- Sparsity of connections

tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding="valid", activation=None, kernel_initializer="glorot_uniform",…)



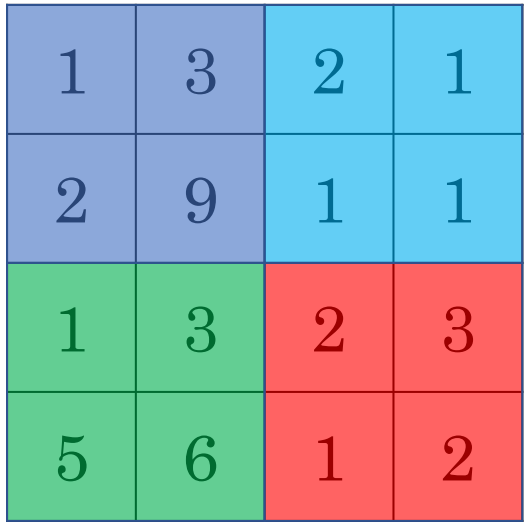
Input size: $n^{[l-1]}$ x $n^{[l-1]}$ x 3

$n_c^{[l]}$: # filters
kernel size: f x f x 3

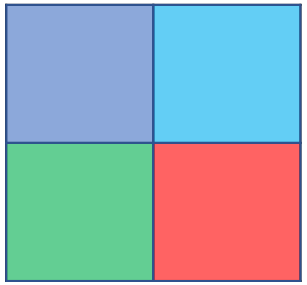
Output size: $n^{[l]}$ x $n^{[l]}$ x $n_c^{[l]}$

Pooling layer

- Max pooling: tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding="valid",…)
- Average pooling: tf.keras.layers.AveragePooling2D(pool_size=(2, 2), strides=None, padding="valid",…)



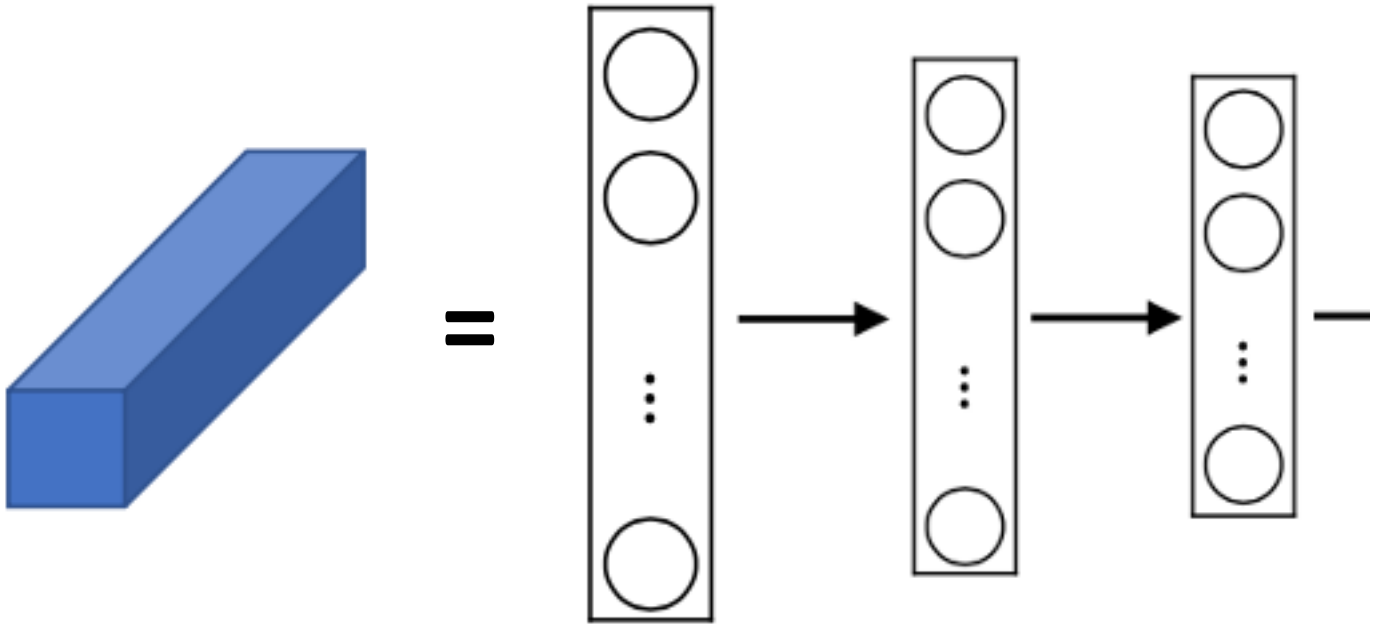
pool_size = (2, 2)
strides = (2, 2)



Output size: $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ x $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

Fully-Connected layer

- Flatten: tf.keras.layers.Flatten()
- Dense: tf.keras.layers.Dense(units, activation=None, kernel_initializer="glorot_uniform”,…)

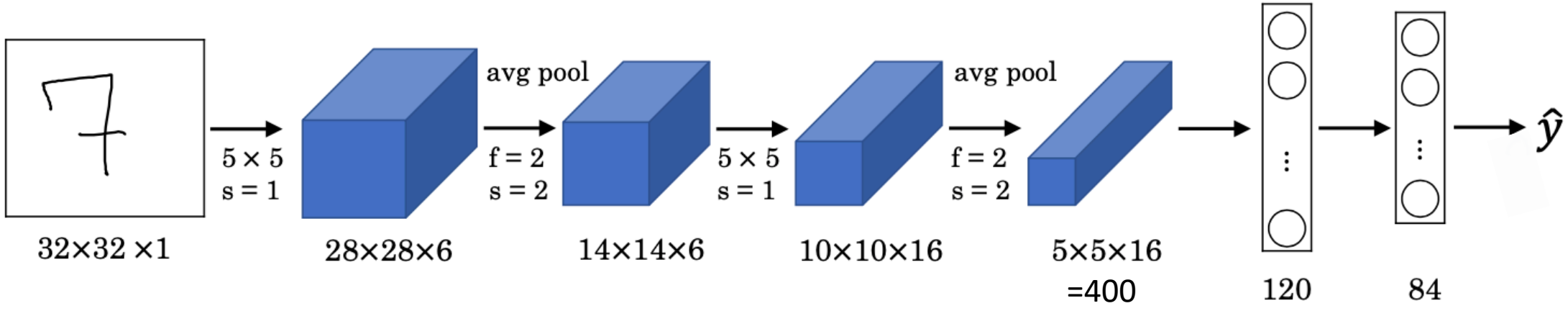


5 x 5 x 16 = 400 400 120 84

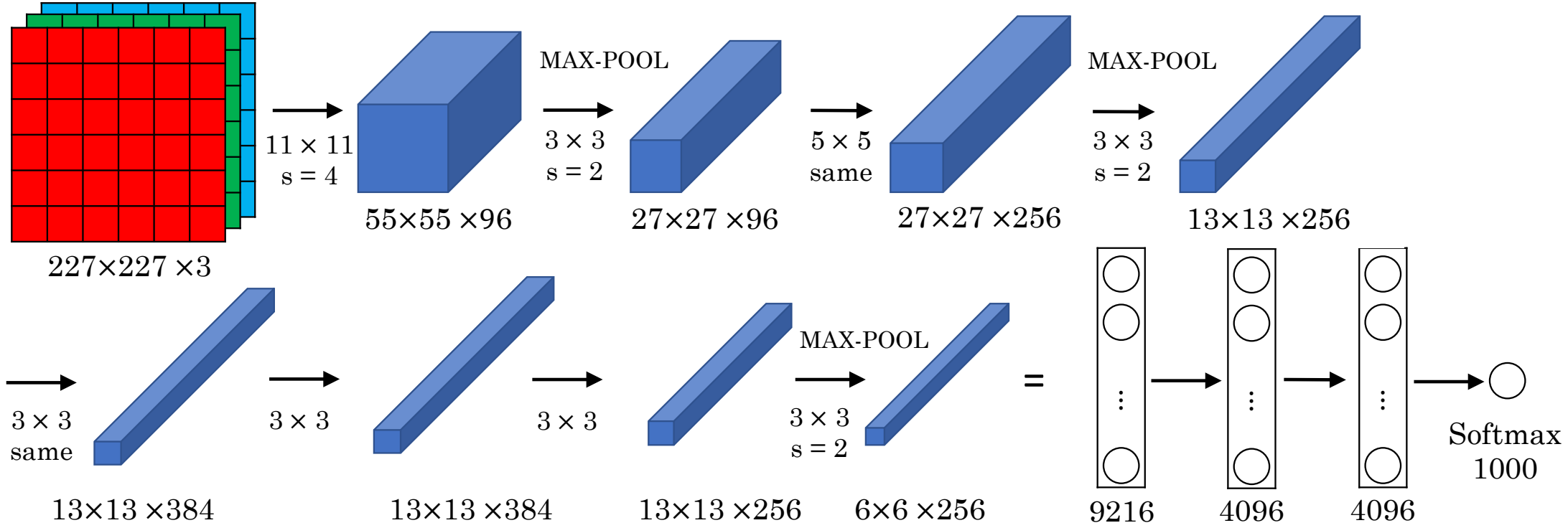
Classic networks

- LeNet-5
- AlexNet
- VGG-16
- ResNet
- Inception

LeNet-5


5 × 5
s = 1
avg pool
f = 2
s = 2
5 × 5
s = 1
avg pool
f = 2
s = 2
$\hat{y}$
32×32 ×1
28×28×6
14×14×6
10×10×16
5×5×16
=400
120
84

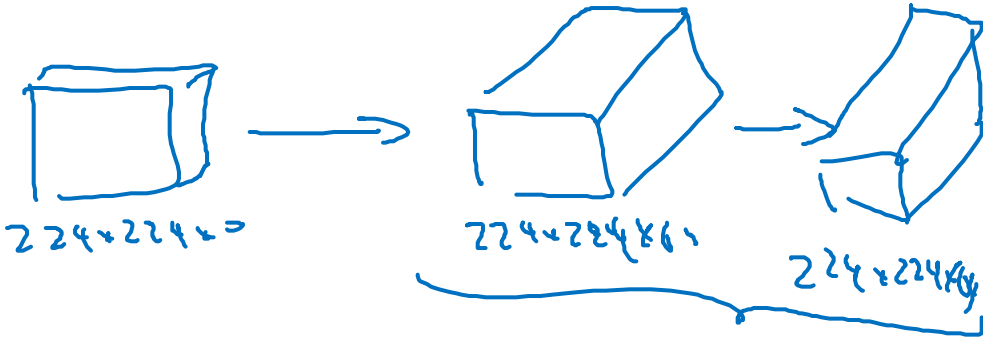
AlexNet

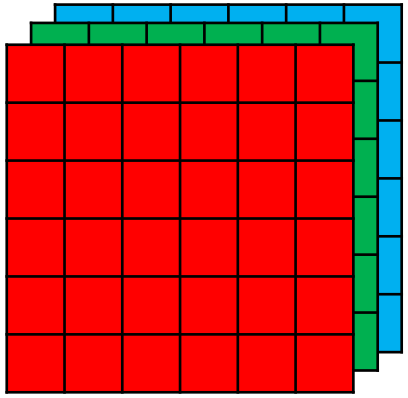

227×227 ×3
11 × 11
s = 4
55×55 ×96
MAX-POOL
3 × 3
s = 2
27×27 ×96
5 × 5
same
27×27 ×256
MAX-POOL
3 × 3
s = 2
13×13 ×256
3 × 3
same
13×13 ×384
3 × 3
13×13 ×384
3 × 3
13×13 ×256
MAX-POOL
3 × 3
s = 2
6×6 ×256
=
9216
4096
4096
Softmax
1000

VGG - 16

VGG-19

CONV = 3×3 filter, s = 1, same MAX-POOL = 2×2 , s = 2





224×224 ×3

→ [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] ×3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

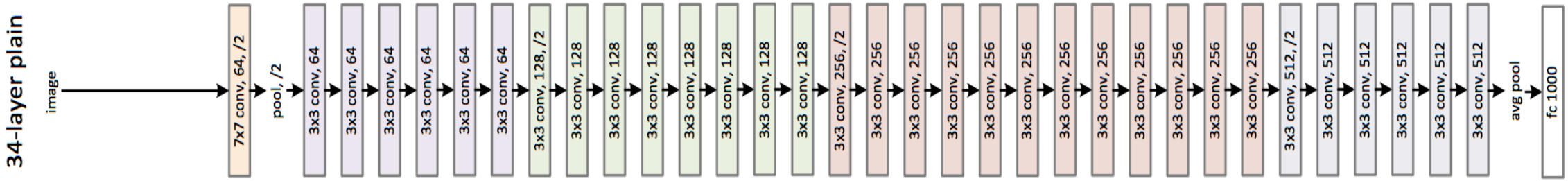
$n_H, n_W \downarrow$ $n_C \uparrow$ ~138M

[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]

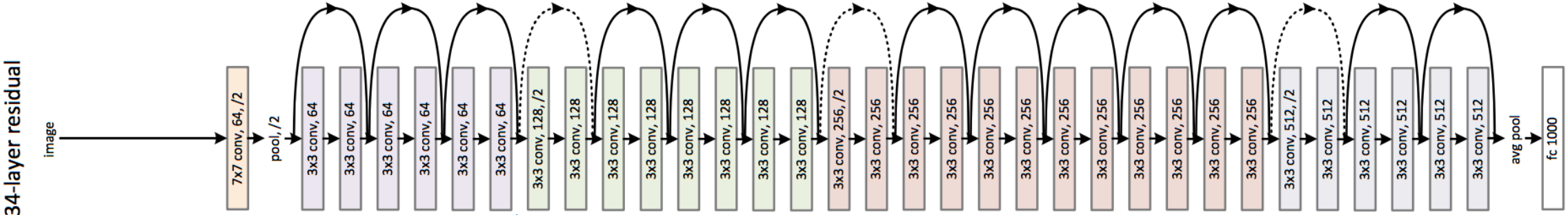
Andrew Ng

ResNet

Plain

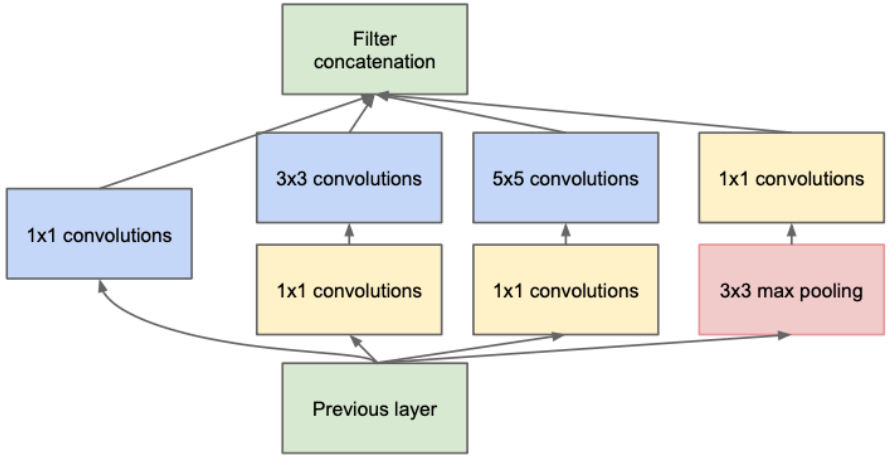


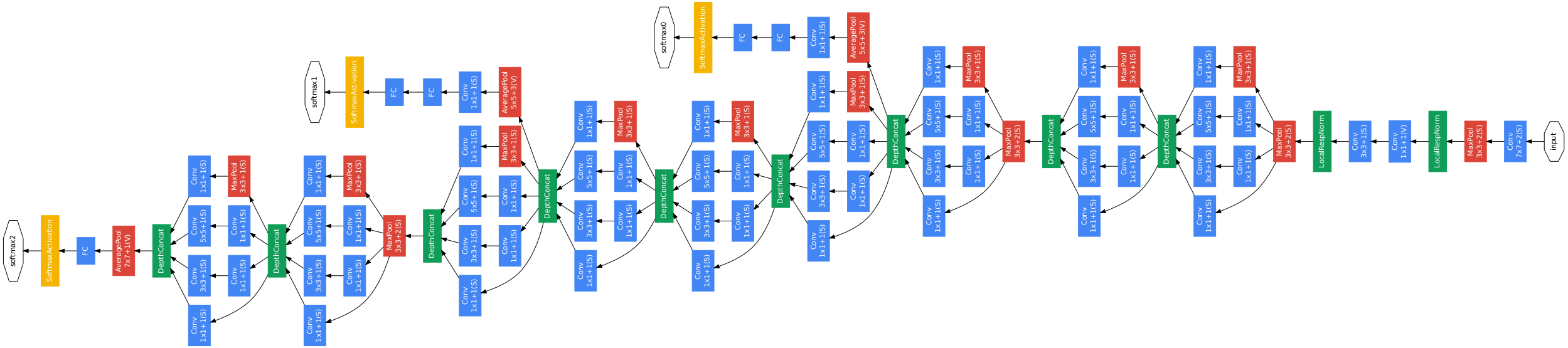
ResNet



Inception (GoogLeNet)

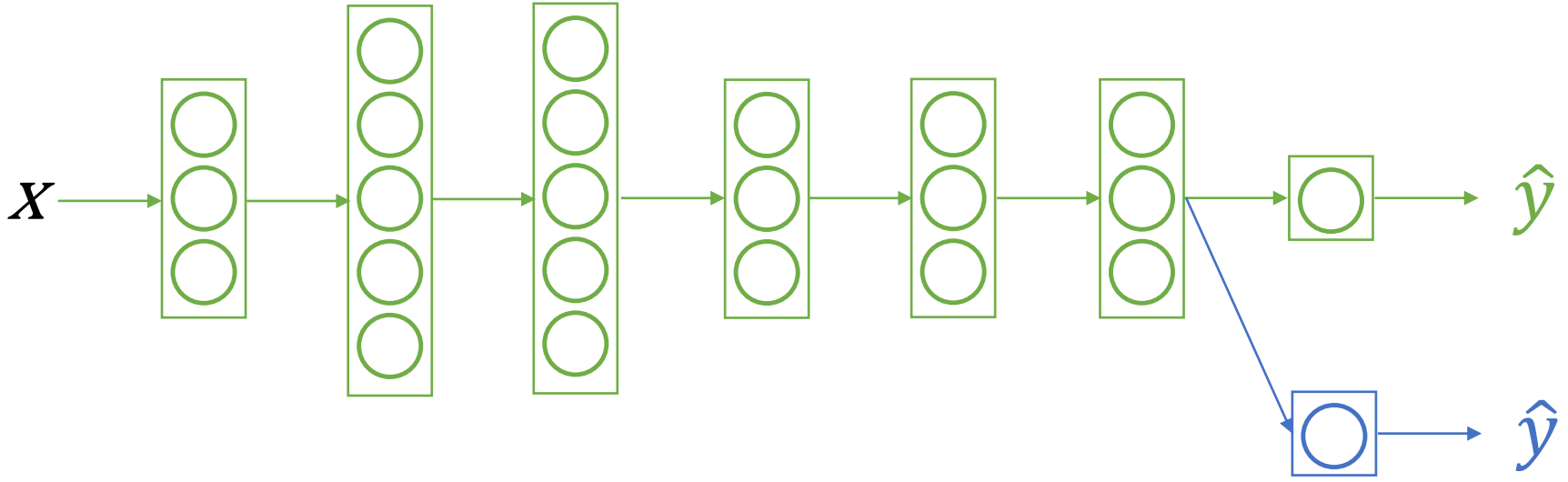
Inception module




input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat

Transfer learning + Fine tuning

- Task A and B have the same(or similar) input x
- The amount of data Task A >> Task B
- Low level features from Task A can be helpful for learning Task B



Task A
Image recognition
1,000,000

Task B
Radiology diagnosis
1,000

Thank you! Any question?

Reference

- Jose Portilla, Udemy Python for Data Science and Machine Learning
  https://www.udemy.com/course/python-for-data-science-and-machine-learning-bootcamp/
- Andrew Ng, Stanford CS230 Deep Learning
  https://cs230.stanford.edu/