Deep Learning: Technical Introduction

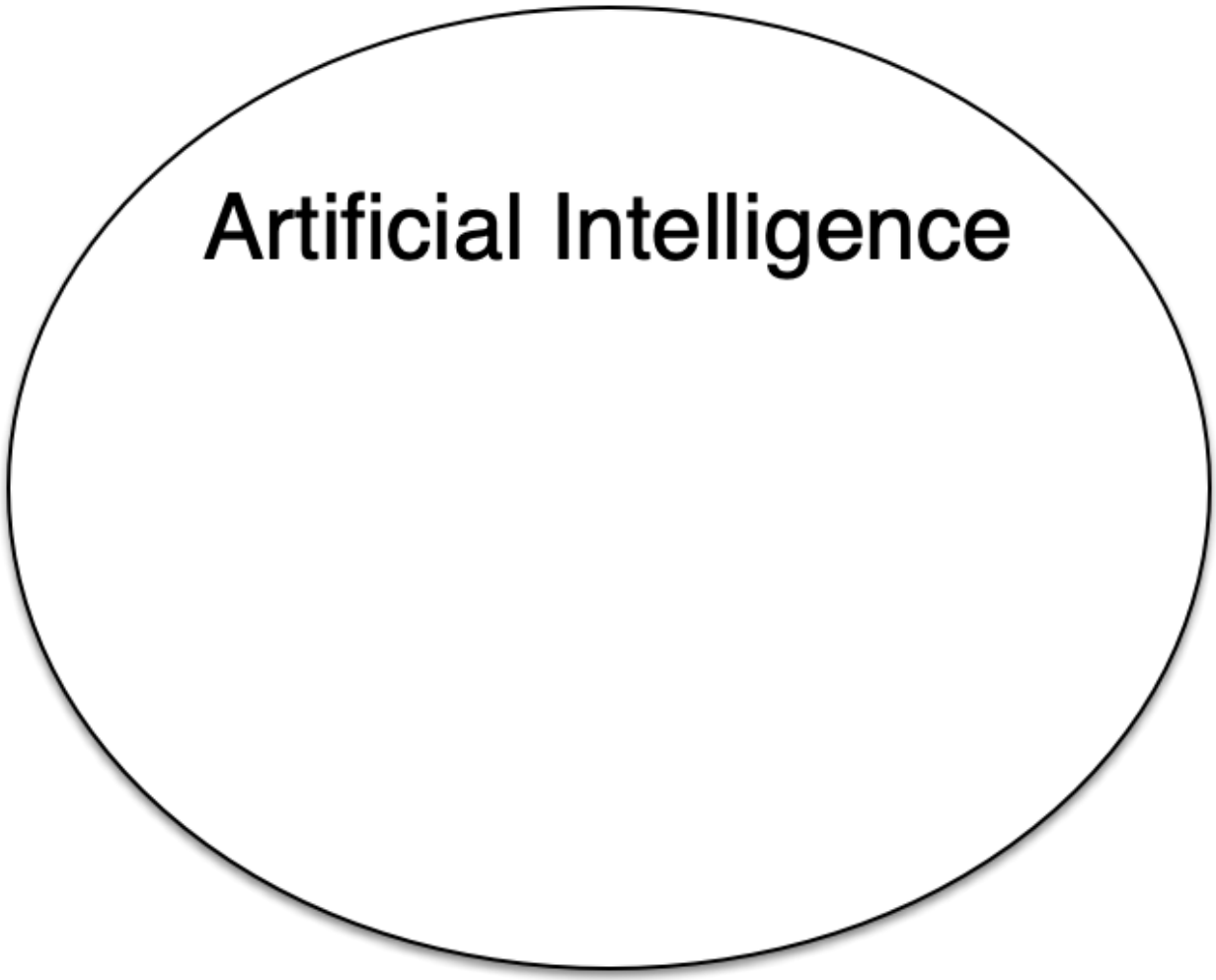
Renaud Florquin

26 novembre 2018

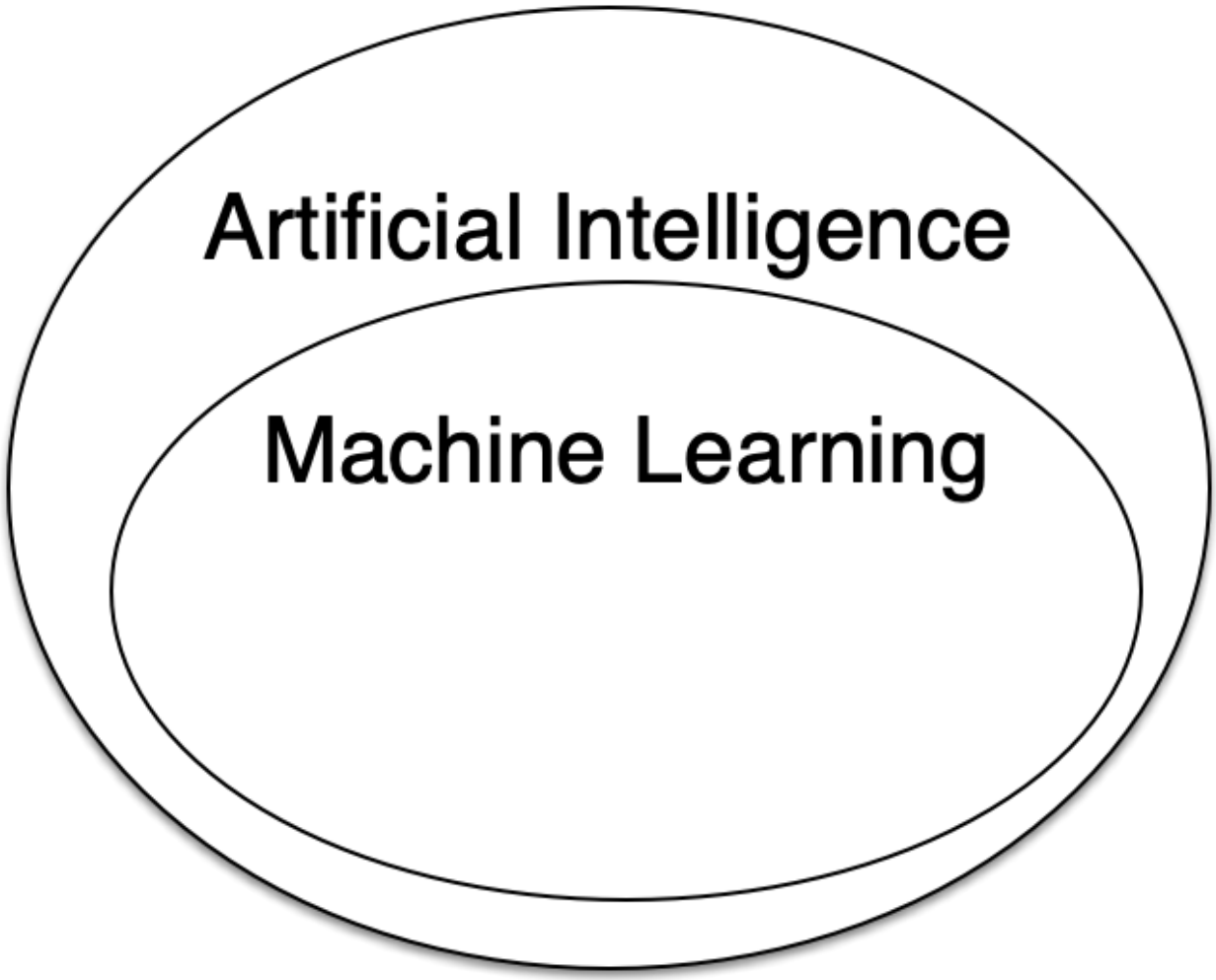
Technical Introduction

Technical Introduction to *Deep Learning*

Articial Intelligence



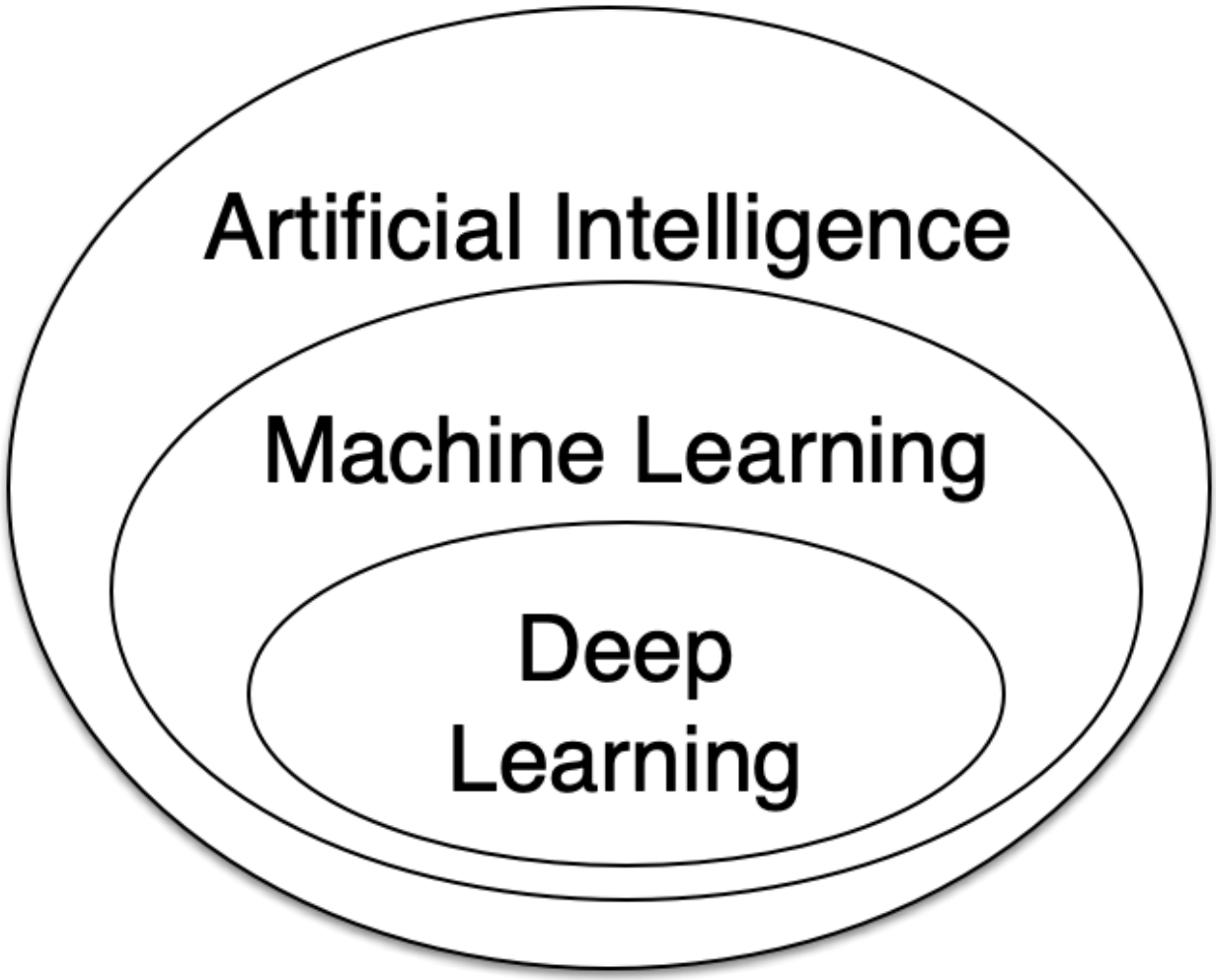
Machine Learning



Machine Learning

- Supervised learning
- Unsupervised learning
- Reinforcement learning

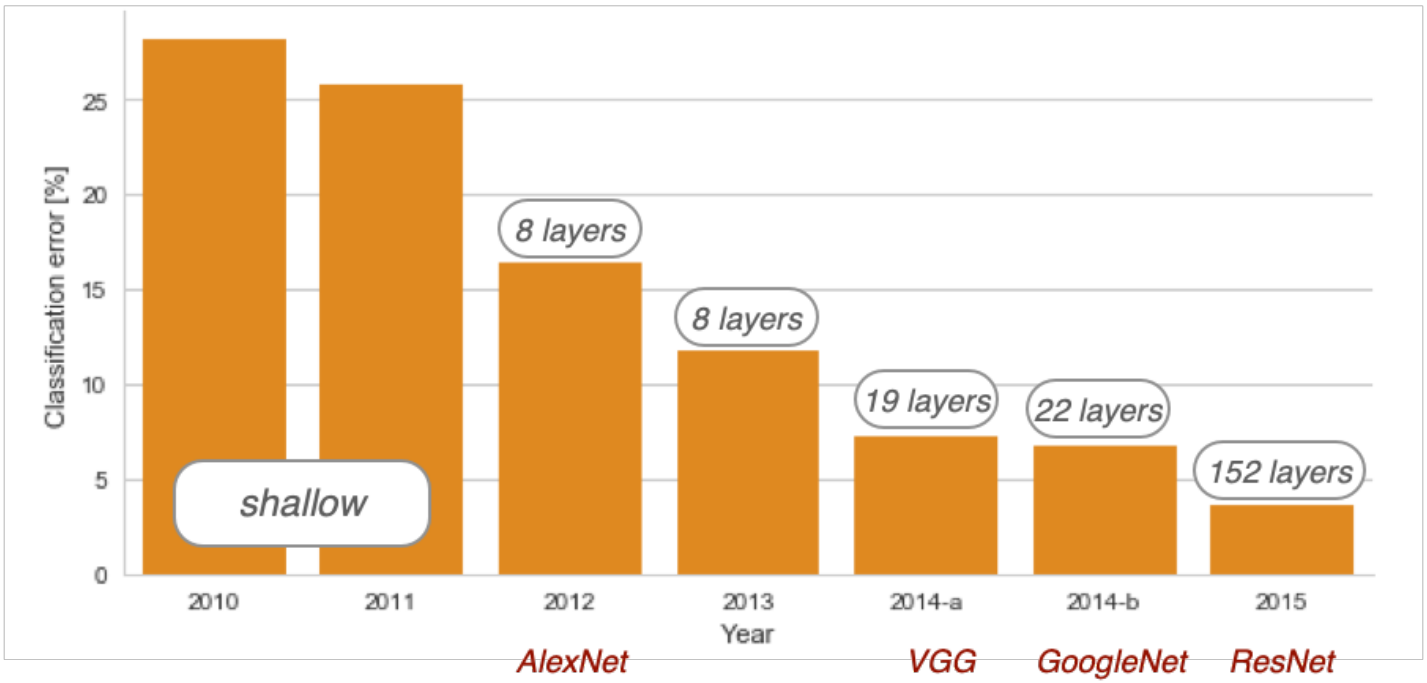
Deep Learning



Deep Learning

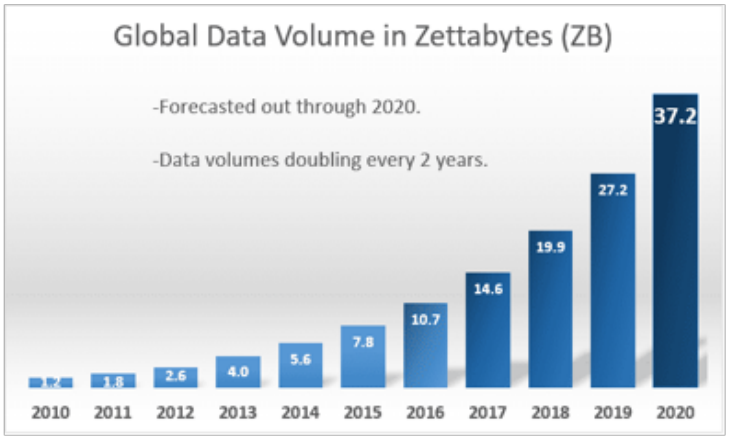
ImageNet Large Scale Visual Recognition Competition

- 1.2M training images / 1000 classes
- Based on WordNet hierarchy (ontology)

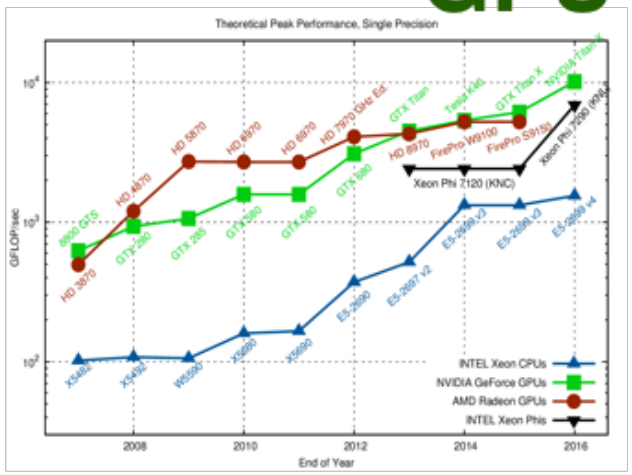


Deep Learning

Big Data



GPU



Algorithms

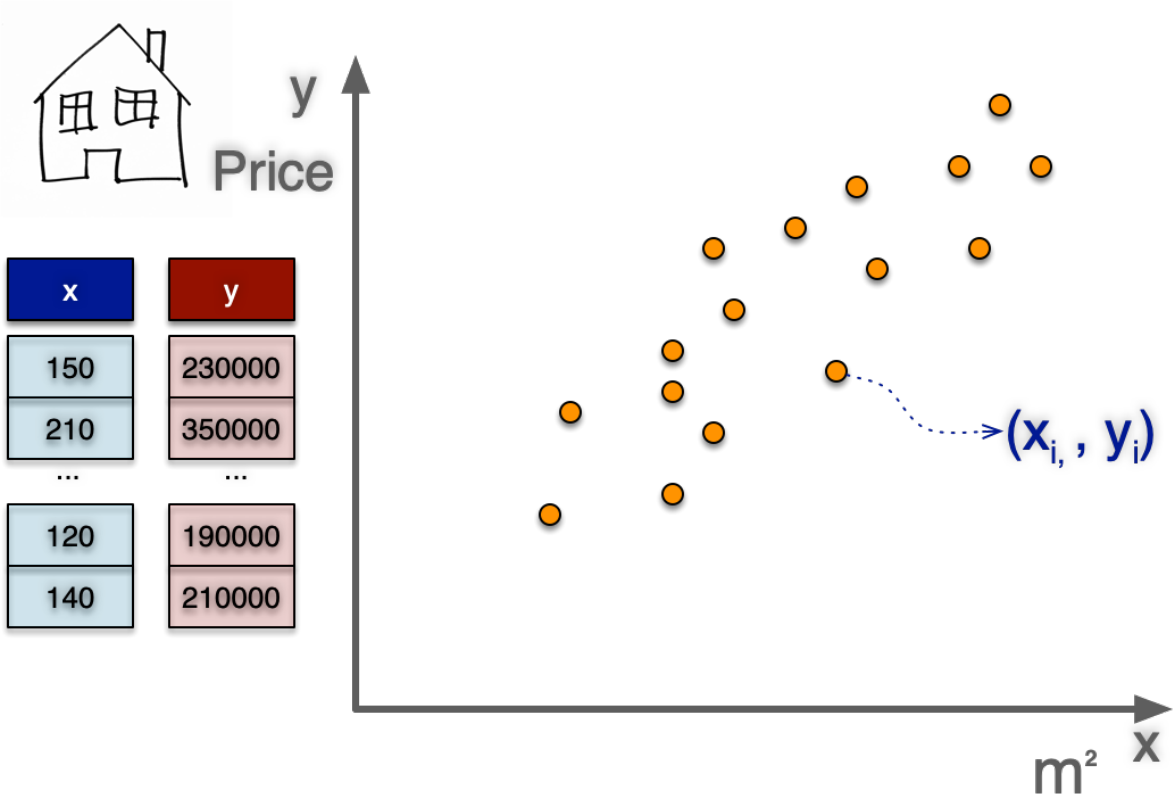
Outline

- Part I
  - Linear Regression
  - Logistic Regression
  - Neural Network
  - Convolutional Neural Network (CNN or ConvNet)
  - Recurrent Neural Network (RNN)
- Part II : Other aspects
  - Evaluation
  - Regularization
  - Transfer learning
  - Gradient-Descent optimizations
  - Software and hardware infrastructure
- Conclusion

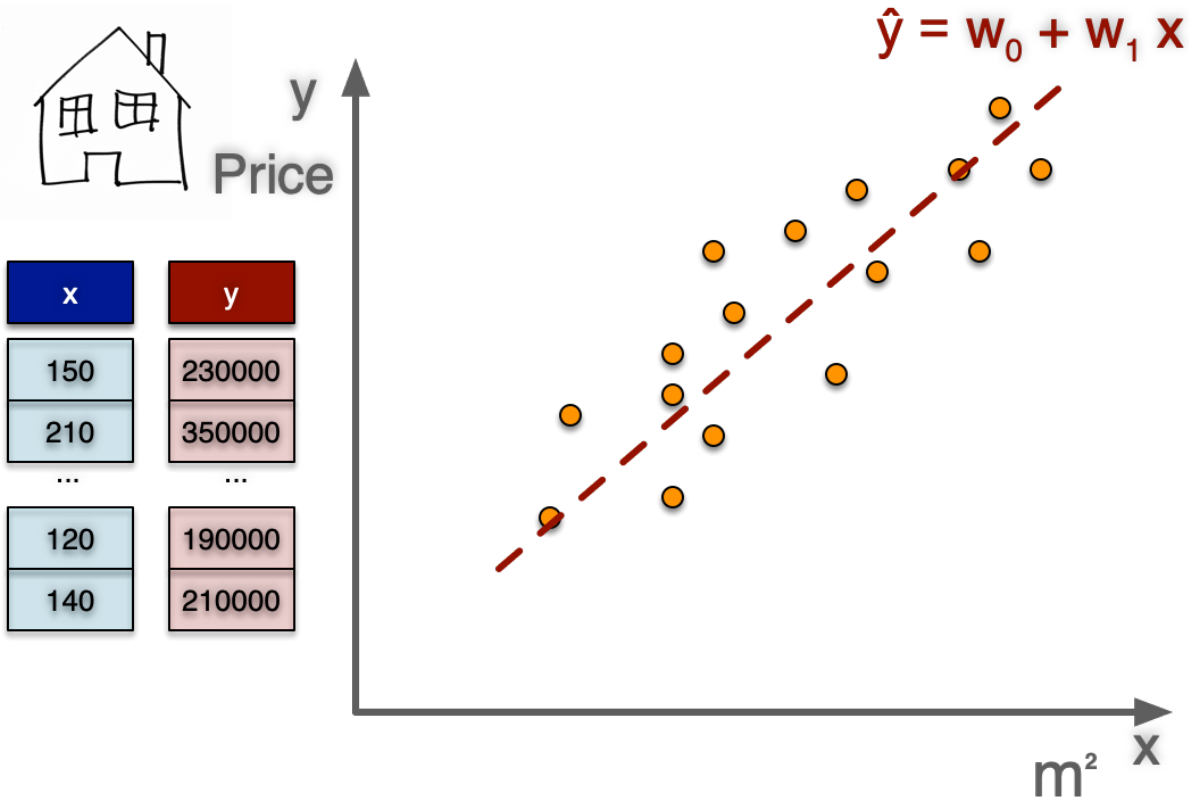
Simple linear regression

Simple linear regression

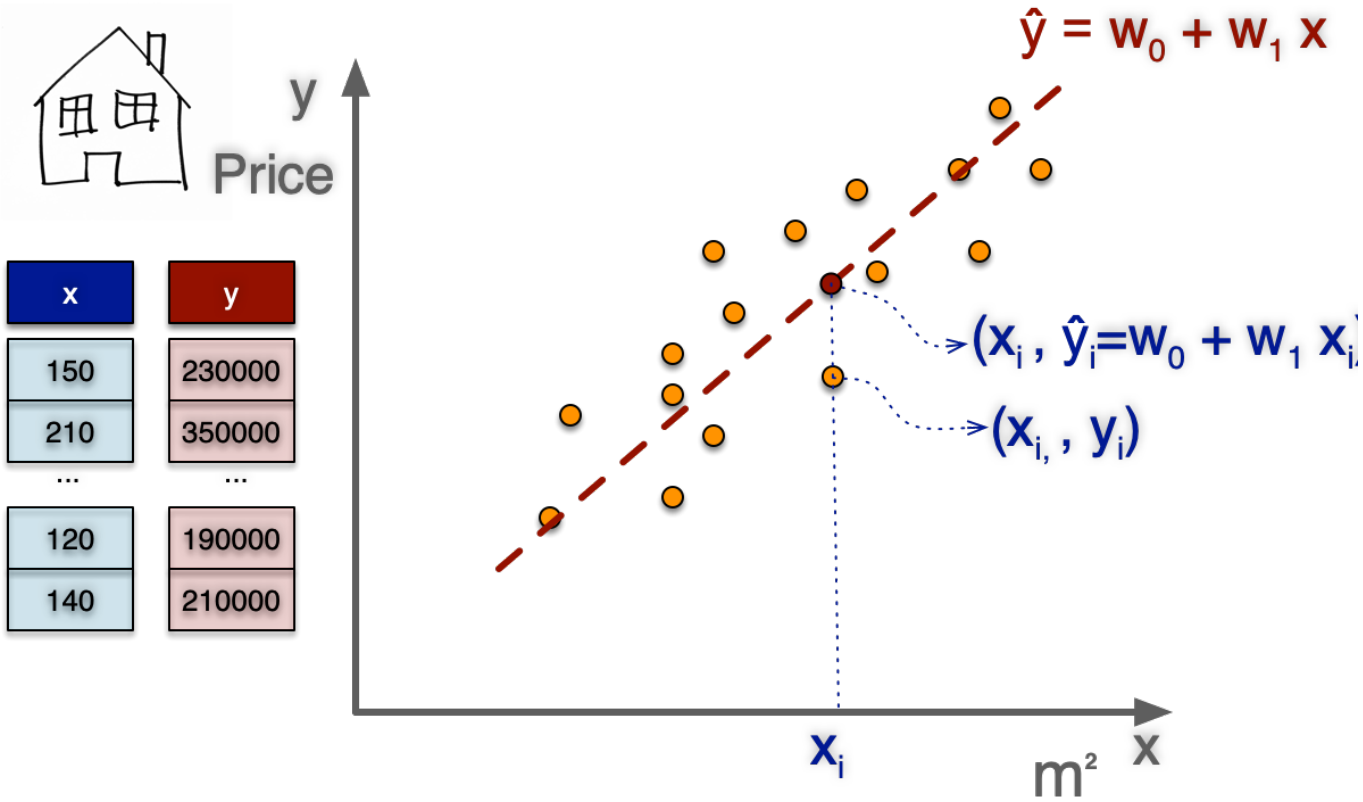
Simple linear regression



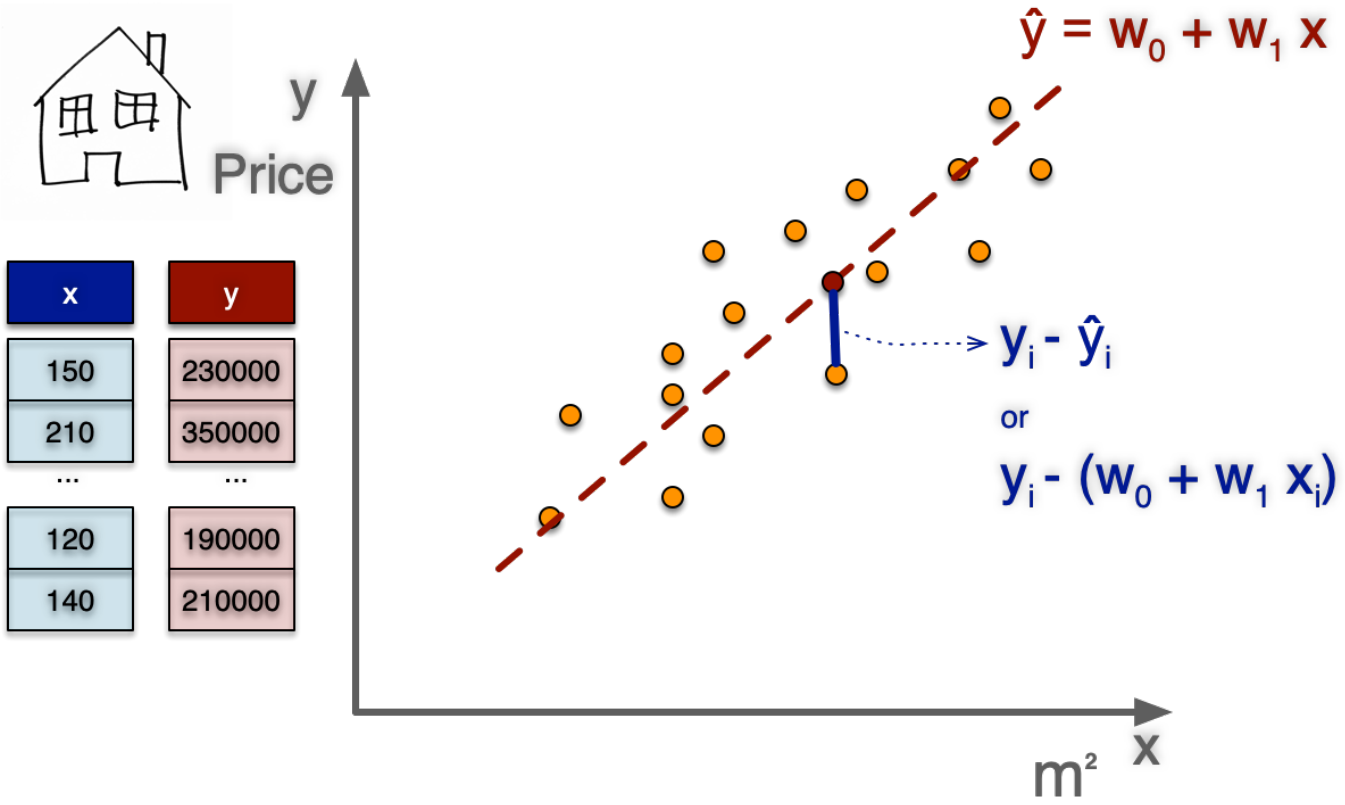
Simple linear regression



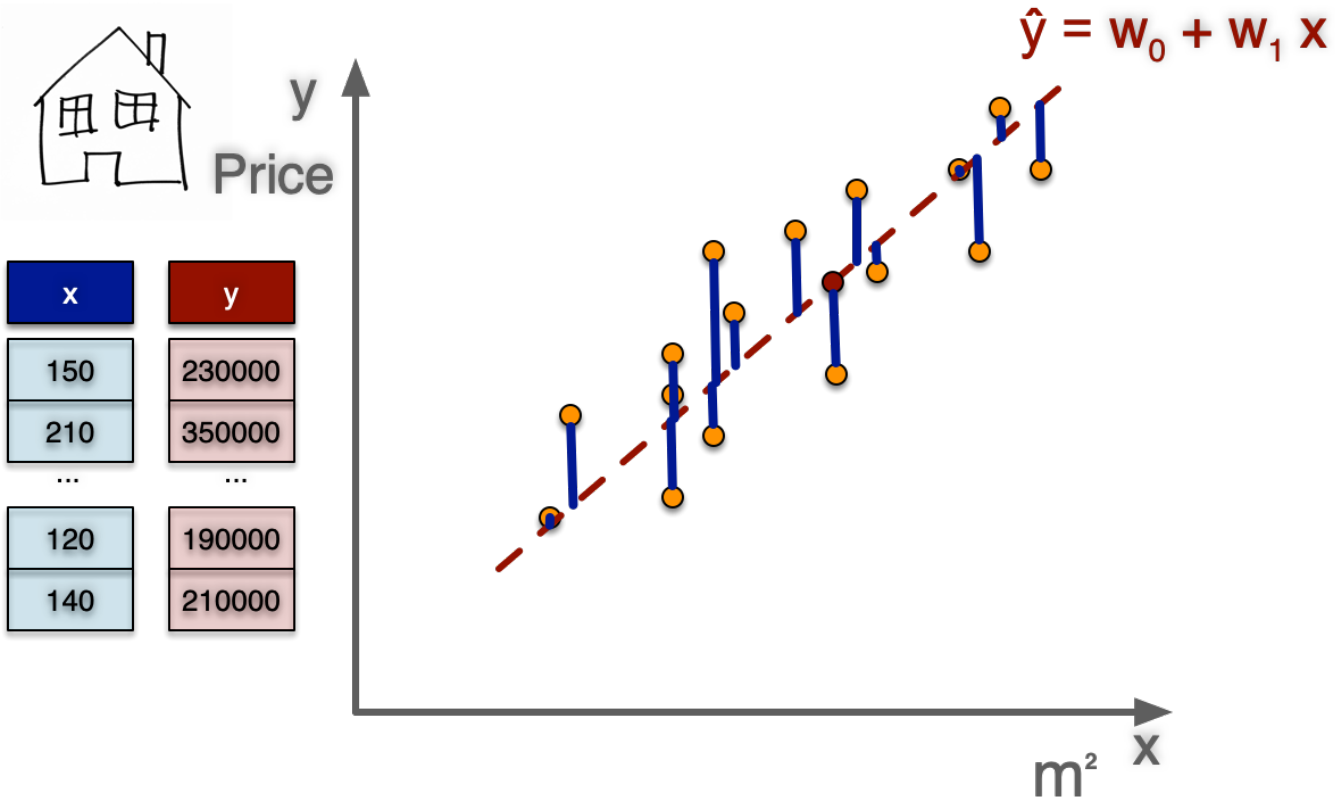
Simple linear regression



Simple linear regression



Simple linear regression



Simple linear regression

Mathematical notation

- m : number of training examples
- x : input variables / features (e.g. ex. surface habitable)
- y : output variable or target or label / (e.g. price)
- $\hat{y}(x)$ *or* $h(x)$: Hypothesis function (or Model)

$$\hat{y}(x) = h(x) = w_0 + w_1 x \tag{1}$$

$$y^{(i)} = \hat{y}(x^{(i)}) + \epsilon = w_0 + w_1 x^{(i)} + \epsilon \tag{2}$$

- w_0, w_1 Parameters of Hypothesis function
- Cost function (Mean Squared Error)

$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right)^2 \tag{3}$$

Cost function minimization

- Cost function (Mean Squared Error)

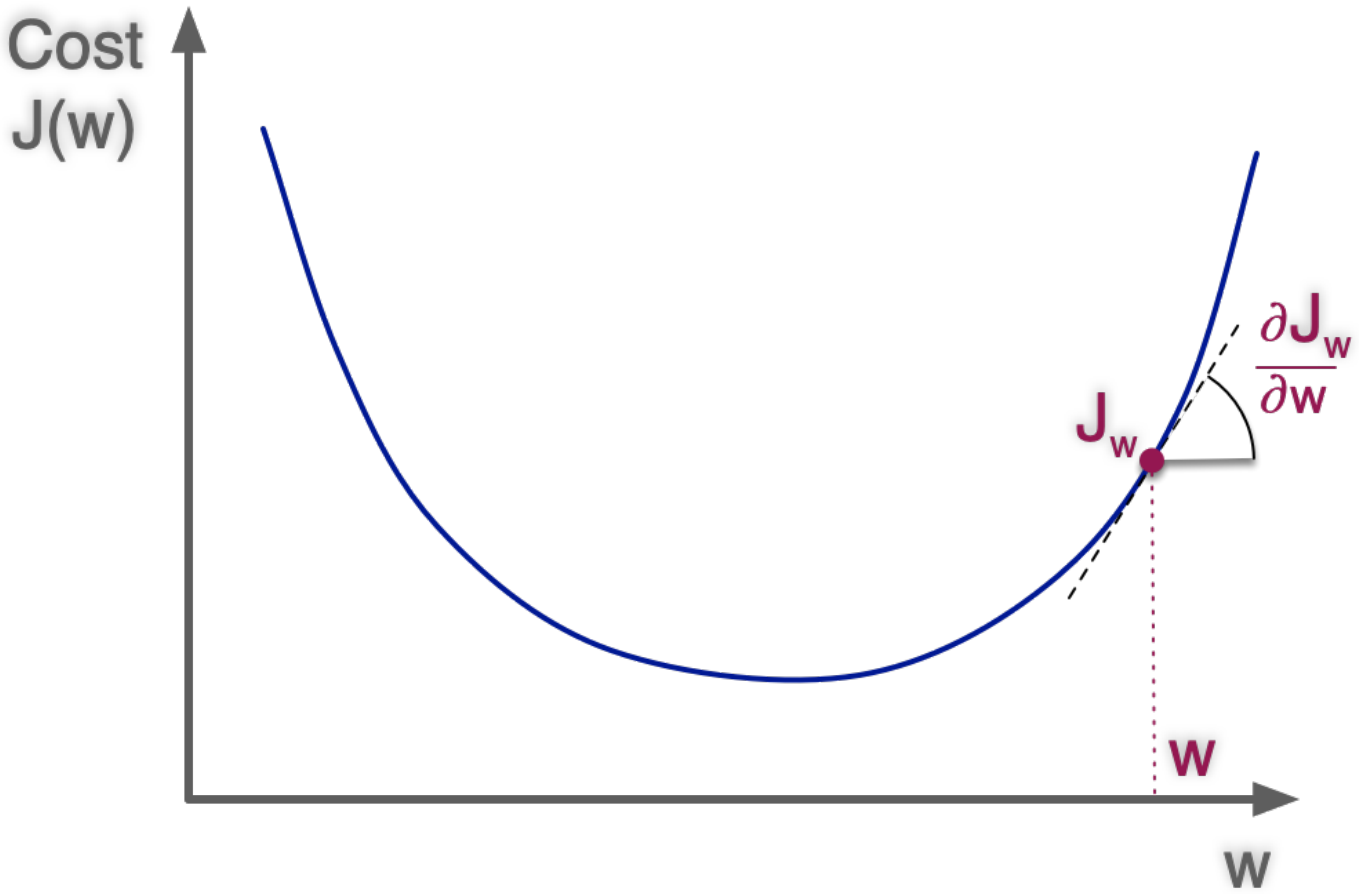
$$J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(w_0 + w_1 x^{(i)} - y^{(i)} \right)^2 \tag{4}$$

- Optimization

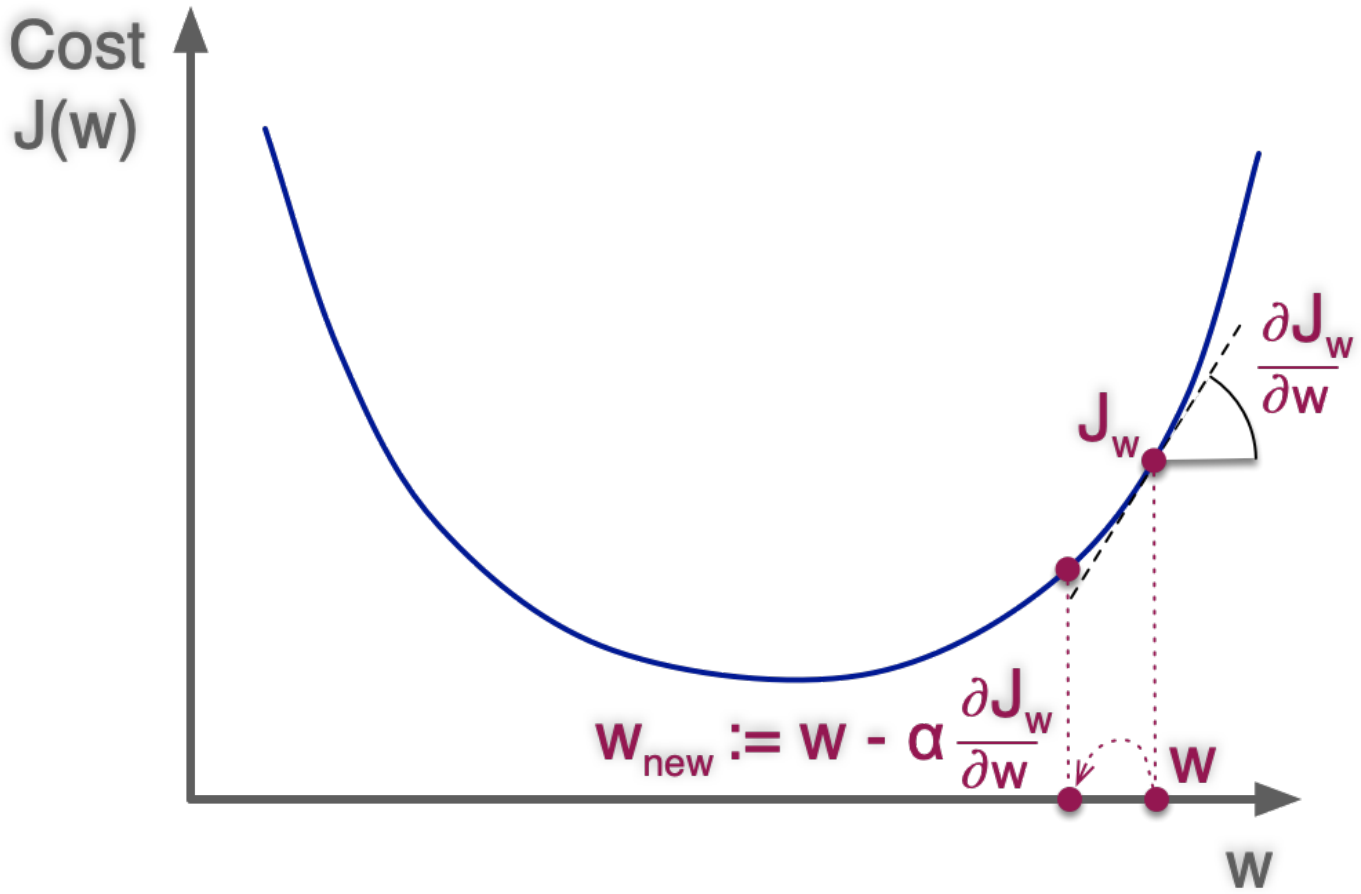
$$\begin{cases} \frac{\partial J(w_0,w_1)}{\partial w_0} = \frac{1}{m} \sum_{i=1}^{m} \left((w_0 + w_1 x^{(i)}) - y^{(i)} \right) \\ \frac{\partial J(w_0,w_1)}{\partial w_1} = \frac{1}{m} \sum_{i=1}^{m} \left((w_0 + w_1 x^{(i)}) - y^{(i)} \right) x^{(i)} \end{cases} \tag{5}$$

- Methods
  - Analytic approach
  - Gradient Descent Algorithm

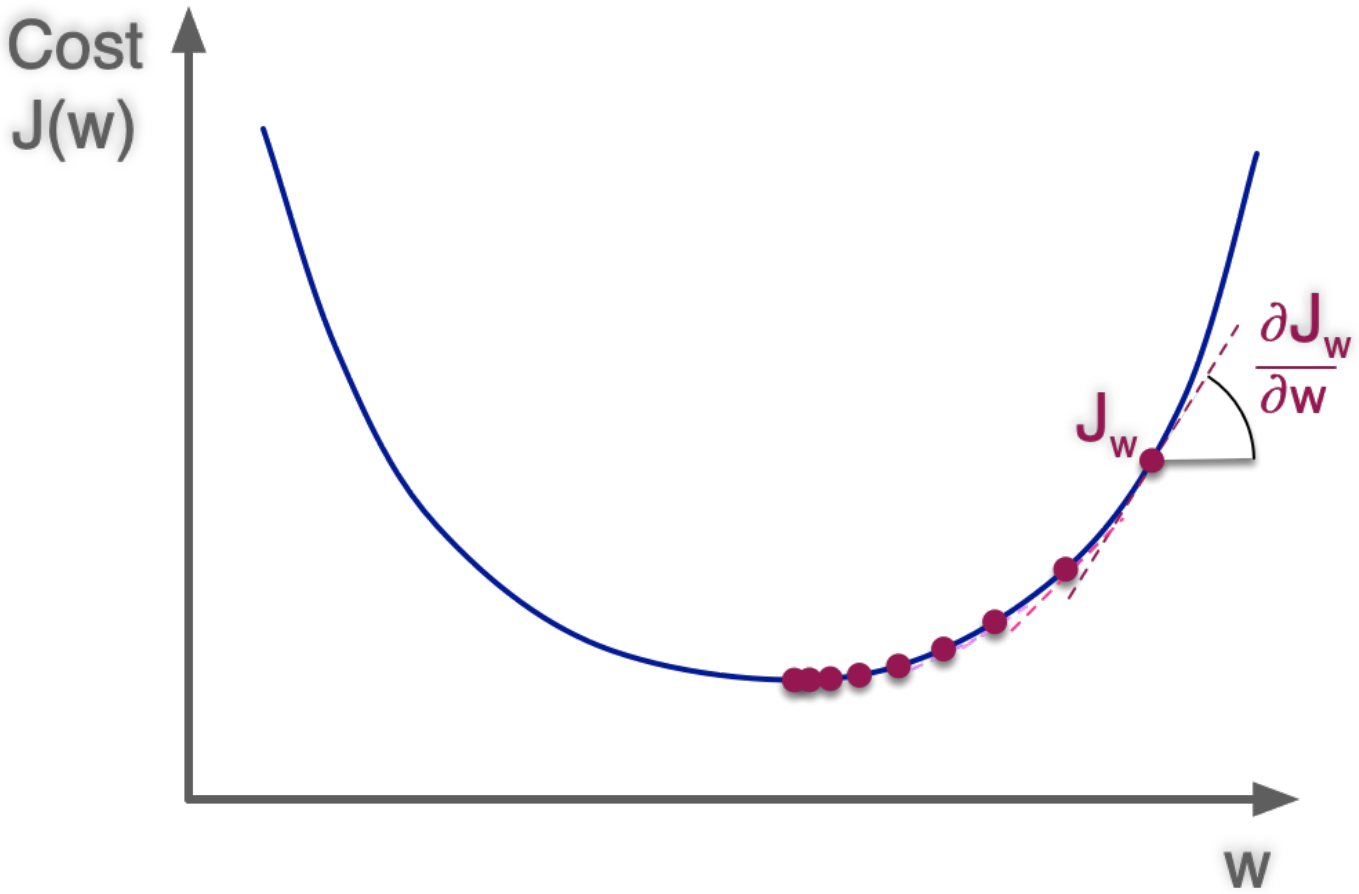
Gradient Descent Algorithm



Gradient Descent Algorithm



Gradient Descent Algorithm

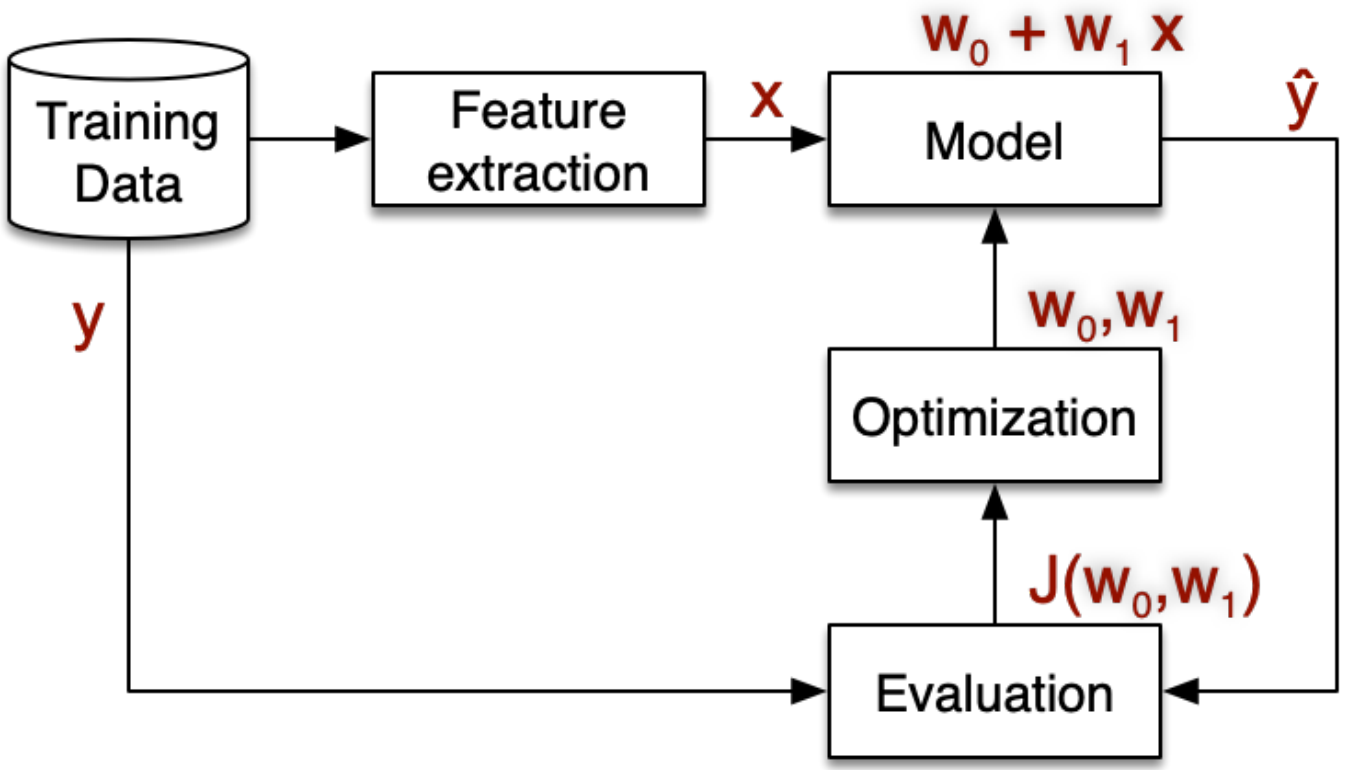


Gradient Descent Algorithm

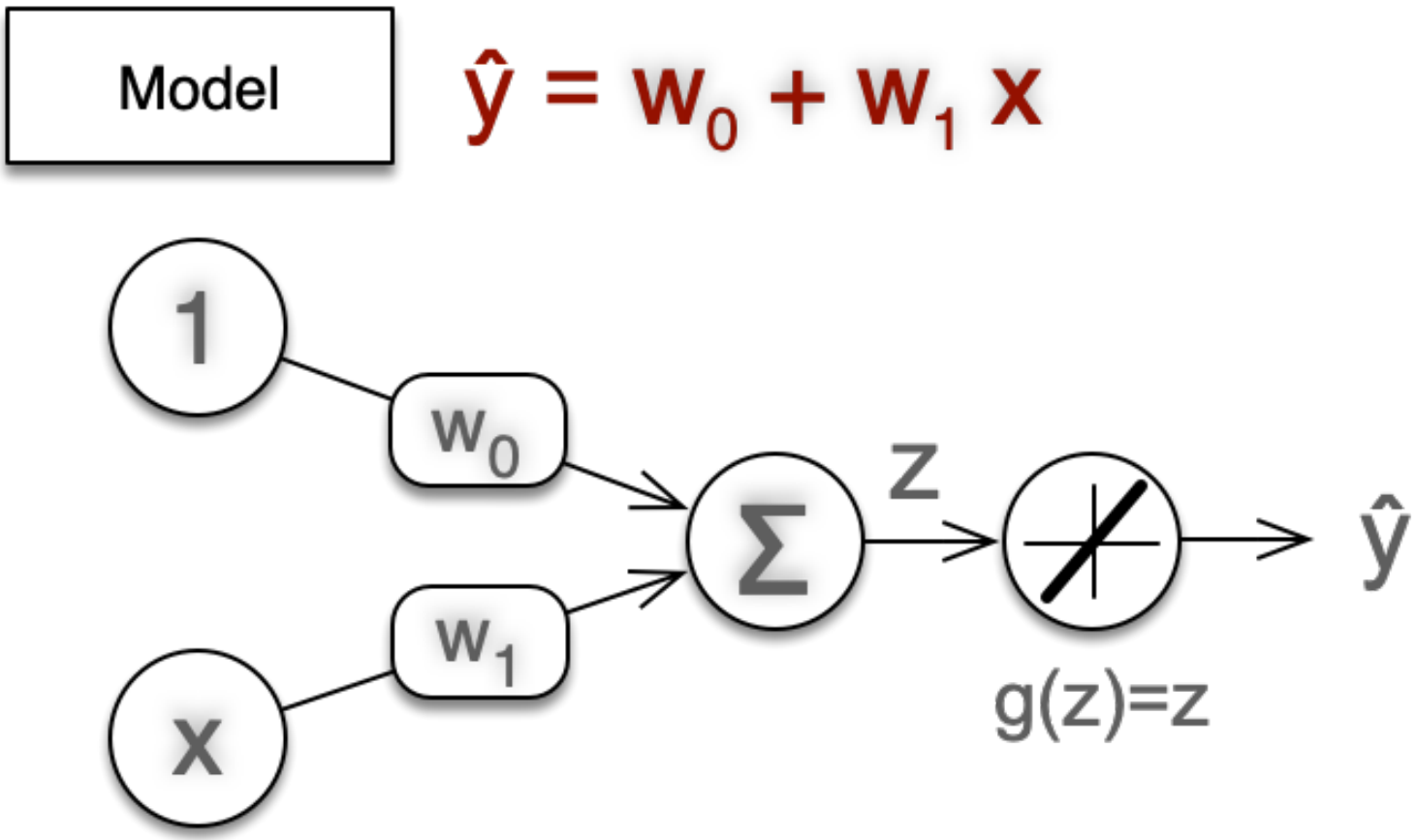
Mathematical notation

$$\begin{cases} w_0 := w_0 - \alpha \frac{\partial J(w_0, w_1)}{\partial w_0} \\ w_1 := w_1 - \alpha \frac{\partial J(w_0, w_1)}{\partial w_1} \end{cases} \tag{6}$$

Linear Regression and Machine Learning



Linear Regression and Neural Network



Questions

- Evaluation ?
- Multiple features ?
- Linear ?
- Classification ?

Linear Regression with Multiple Variables

Linear Regression with Multiple Variables

- n : number of features
- m : number of training examples
- i^{th} training example :

$$x^{(i)^T} = \begin{pmatrix} 1 & x_1^{(i)} & x_2^{(i)} & \dots & x_n^{(i)} \end{pmatrix} \tag{7}$$

- Training examples

$$\mathbf{X} = \begin{pmatrix} - \left(x^{(1)}\right)^T - \\ - \left(x^{(2)}\right)^T - \\ \dots \\ - \left(x^{(m)}\right)^T - \end{pmatrix} \quad \mathbf{Y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix} \tag{8}$$

Linear Regression with Multiple Variables

- Model

$$\hat{y}(x) = w_0 + w_1 x_1 + w_2 x_2 + ... + w_n x_n \tag{9}$$

$$\hat{\mathbf{Y}}(\mathbf{X}) = \mathbf{X}\mathbf{W} \tag{10}$$

- Cost function

$$J(w_0, w_1, ..., w_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right)^2 \tag{11}$$

Linear Regression with Multiple Variables

- Gradient $J(w_0, w_1, ..., w_n)$

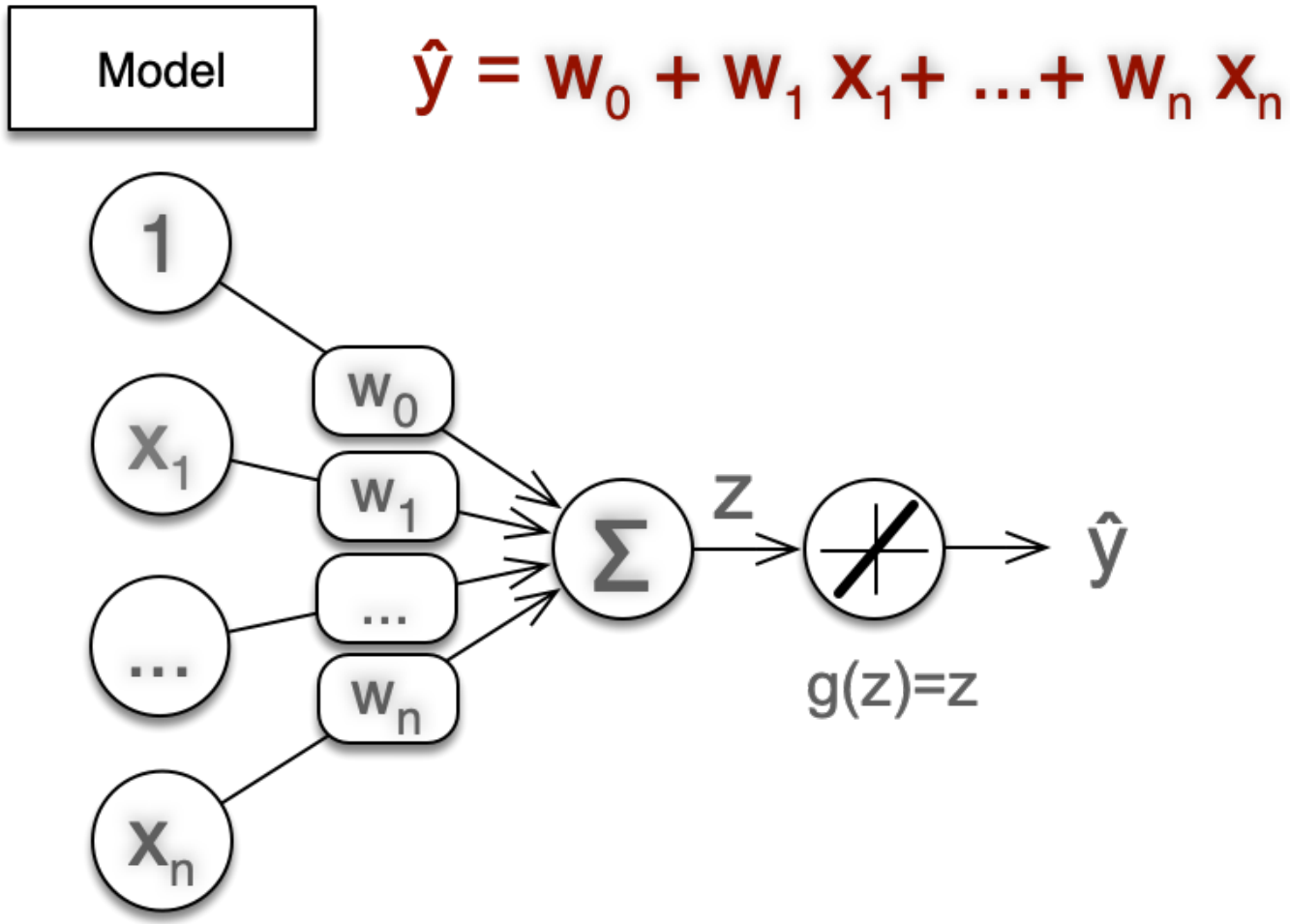
$$\frac{\partial J(w_0, w_1, w_n)}{\partial w_i} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right) x^{(i)} \tag{12}$$

$$\nabla \mathbf{J} = \frac{1}{m} (\hat{\mathbf{Y}}(x) - \mathbf{Y})^t \mathbf{X} \tag{13}$$

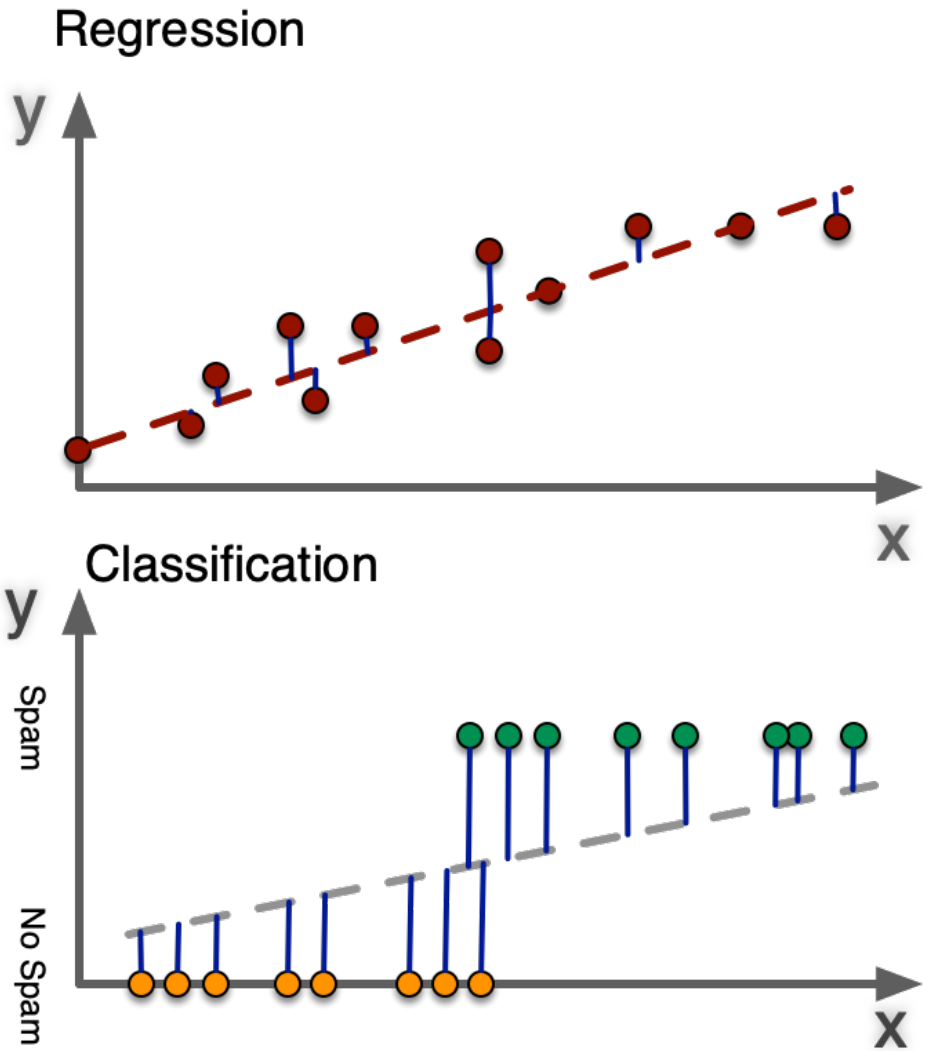
- Gradient Descent Algorithm

$$\mathbf{W} := \mathbf{W} - \alpha \, \nabla \mathbf{J} \tag{14}$$

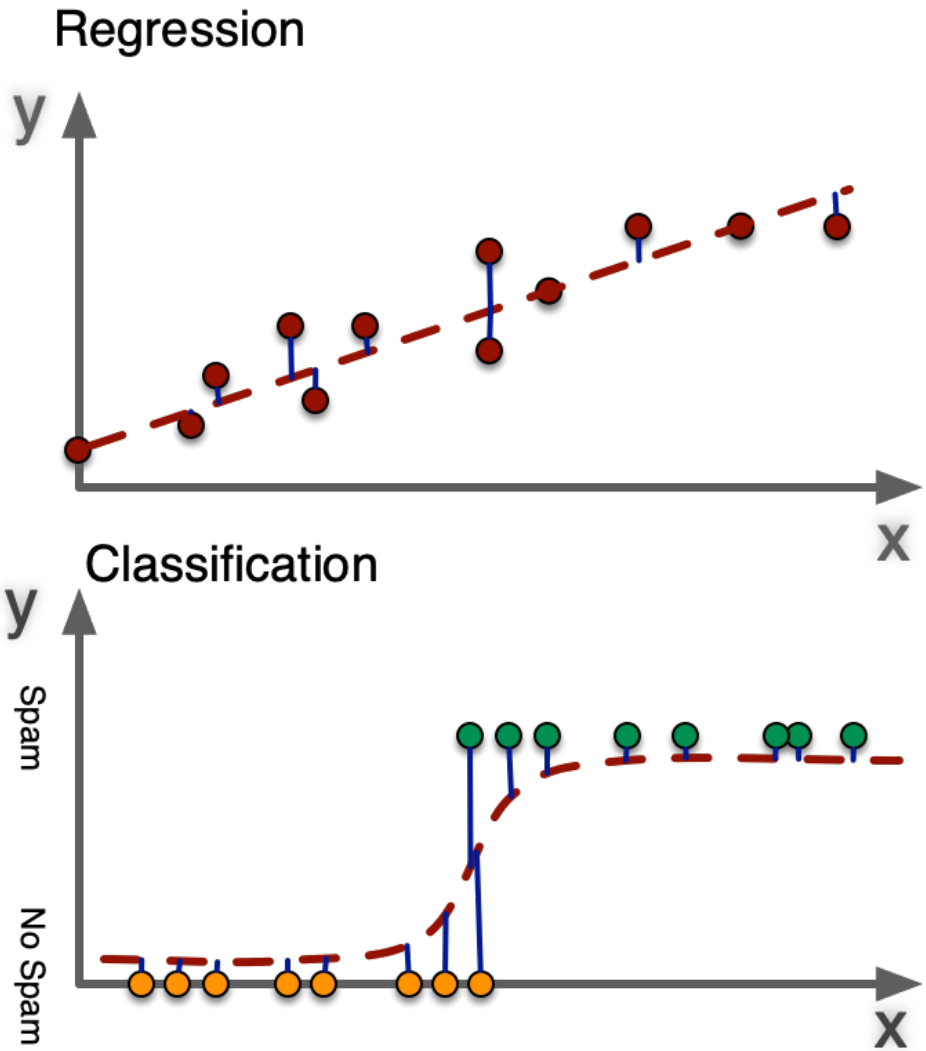
Linear Regression with Multiple Variables and Neural network



Regression versus Classification



Regression versus Classification



Logistic Regression

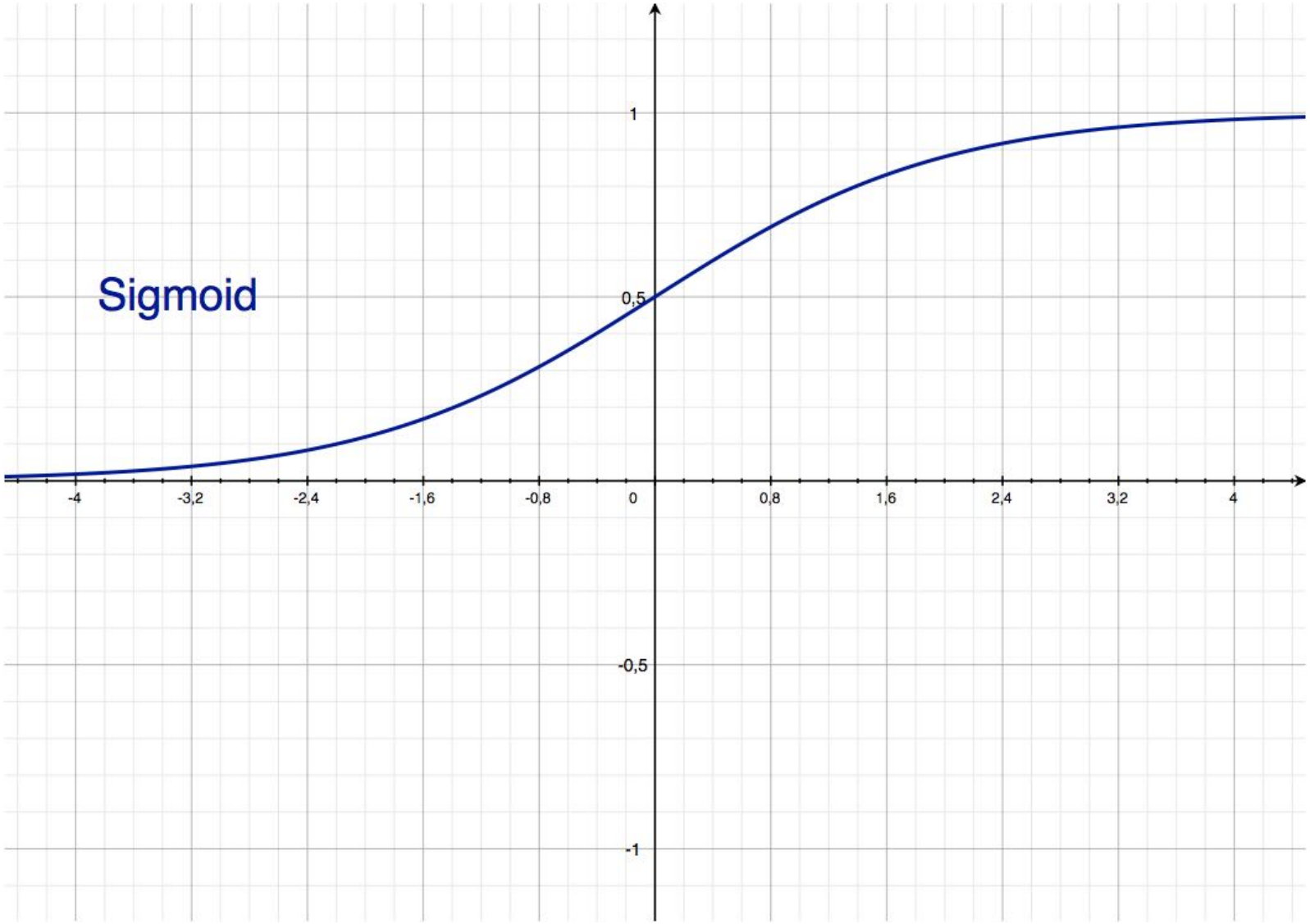
- Linear Regression

$$\hat{\mathbf{Y}}(\mathbf{X}) = \mathbf{X}\mathbf{W} \tag{15}$$

- Logistic Regression

$$\begin{cases} \mathbf{Z} = \mathbf{X}\mathbf{W} \\ \hat{\mathbf{Y}} = g(\mathbf{Z}) \\ g(z) = \frac{1}{1+e^{-z}} \end{cases} \tag{16}$$

Logistic Regression



Logistic Regression

- Cost function

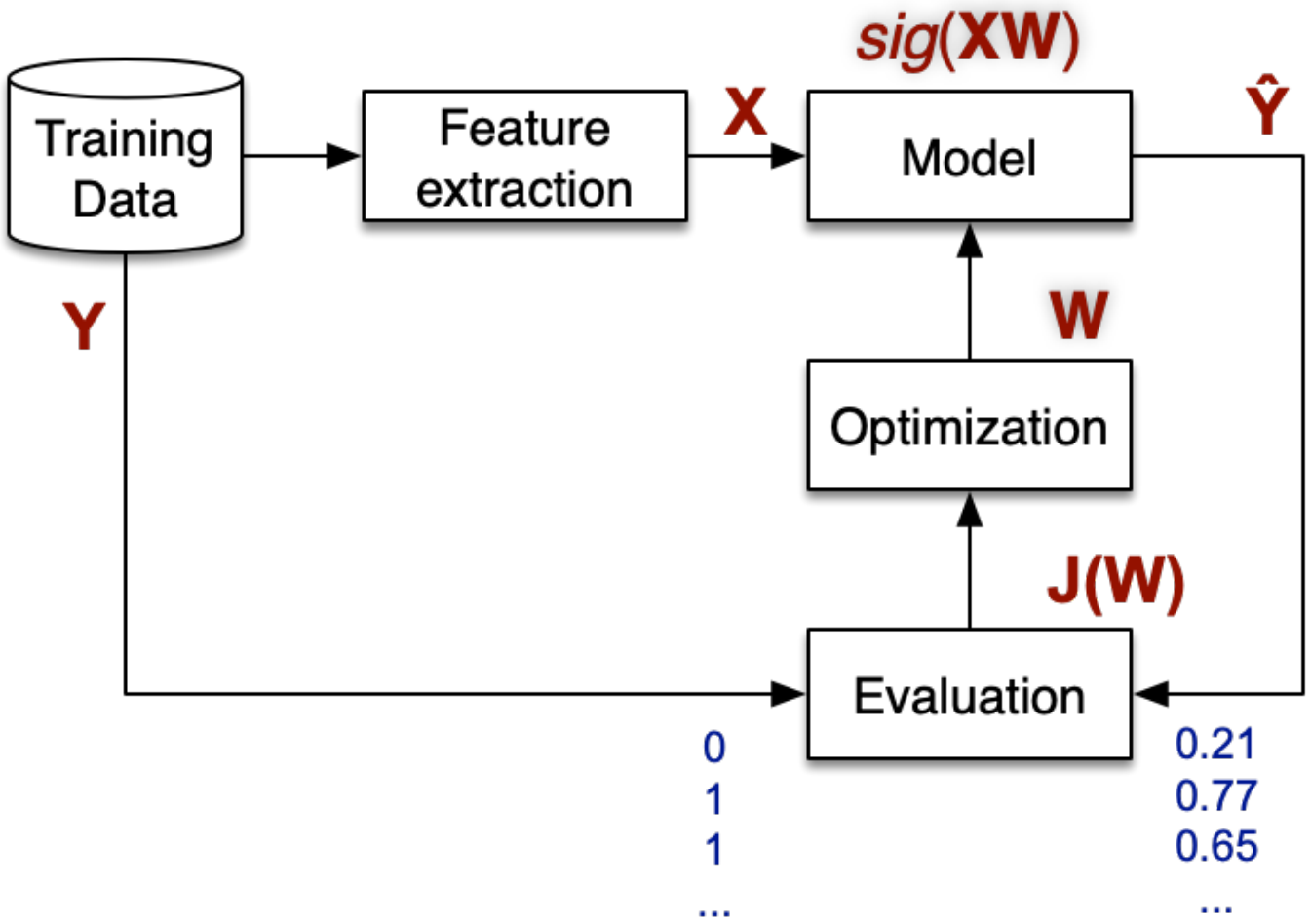
$$J(w_0, w_1, ..., w_n) = \frac{1}{m}\sum_{i=1}^{m}[- y^{(i)} log(\hat{y}(x^{(i)})) - (1 - y^{(i)}) log(1 - \hat{y}(x^{(i)}))]$$

- Gradient $J(w_0, w_1, ..., w_n)$

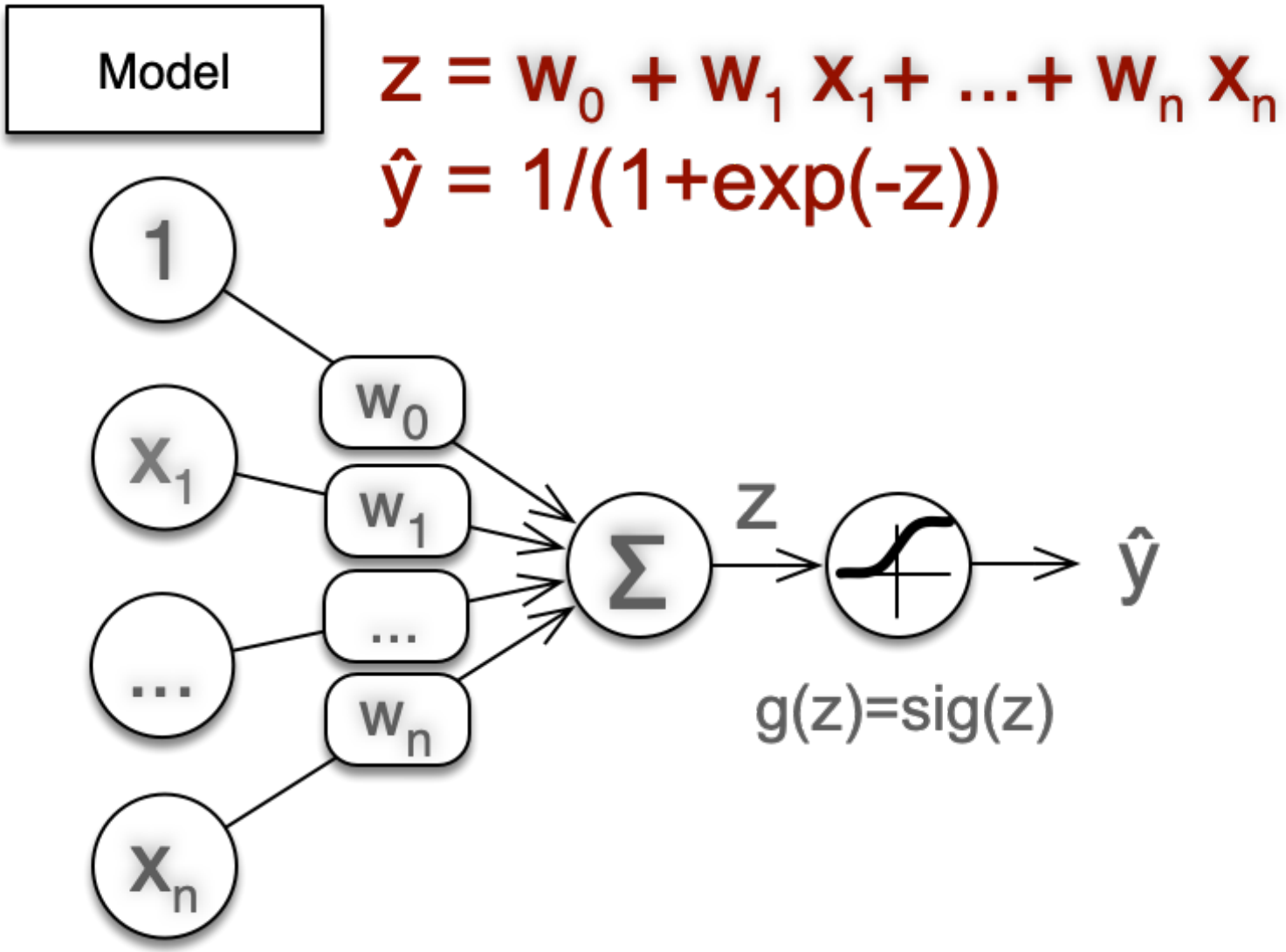
$$\nabla \mathbf{J} = \frac{1}{m}(\hat{\mathbf{Y}}(x) - \mathbf{Y})^t \mathbf{X}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

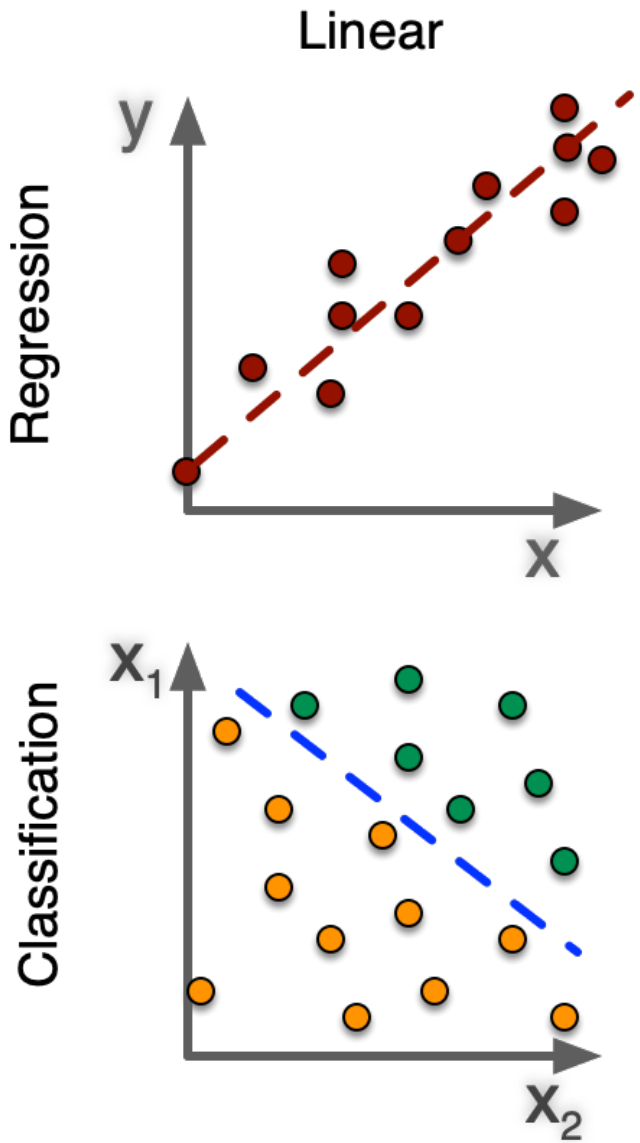
Logistic Regression and Machine Learning



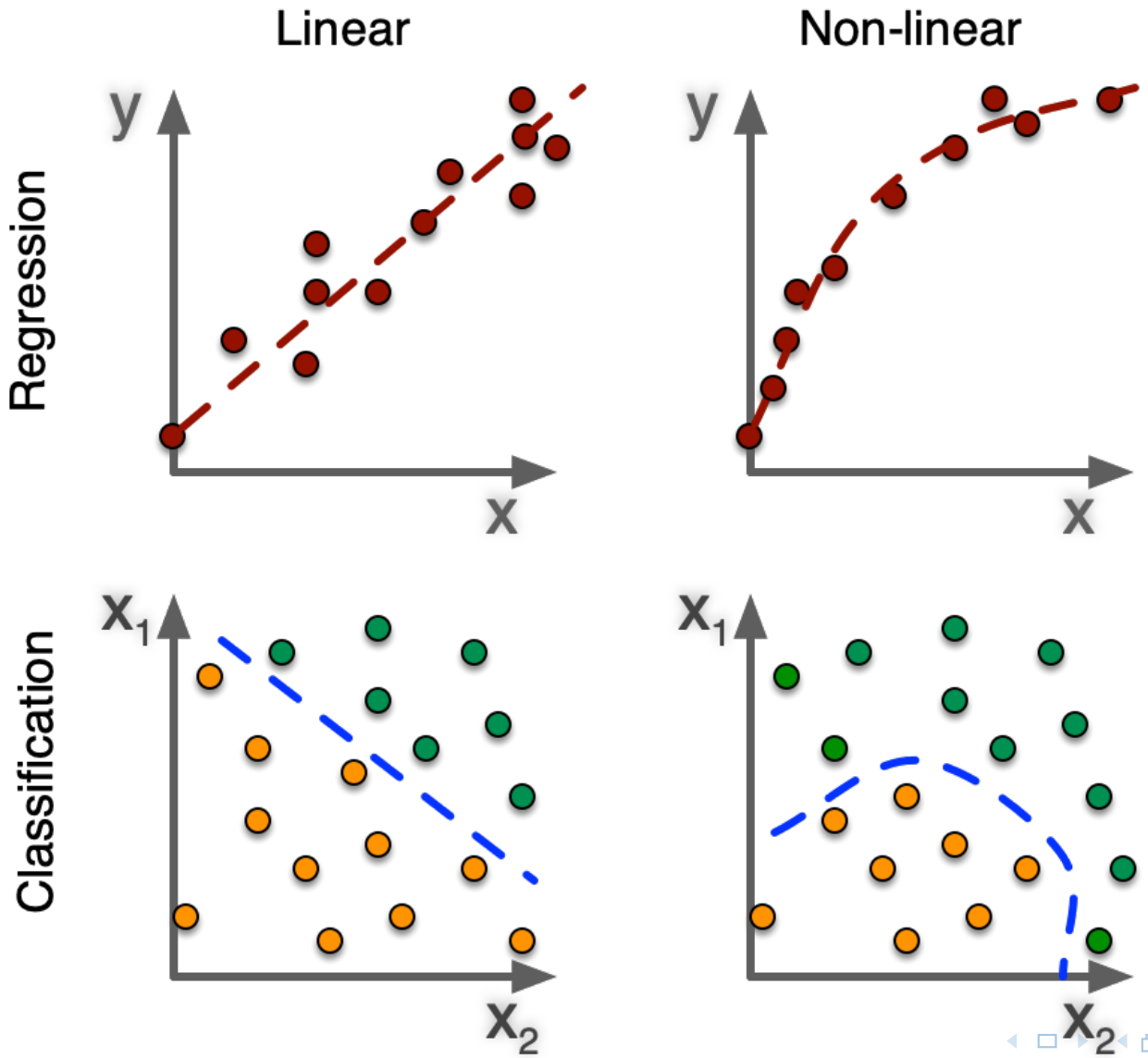
Logistic Regression and Neural network



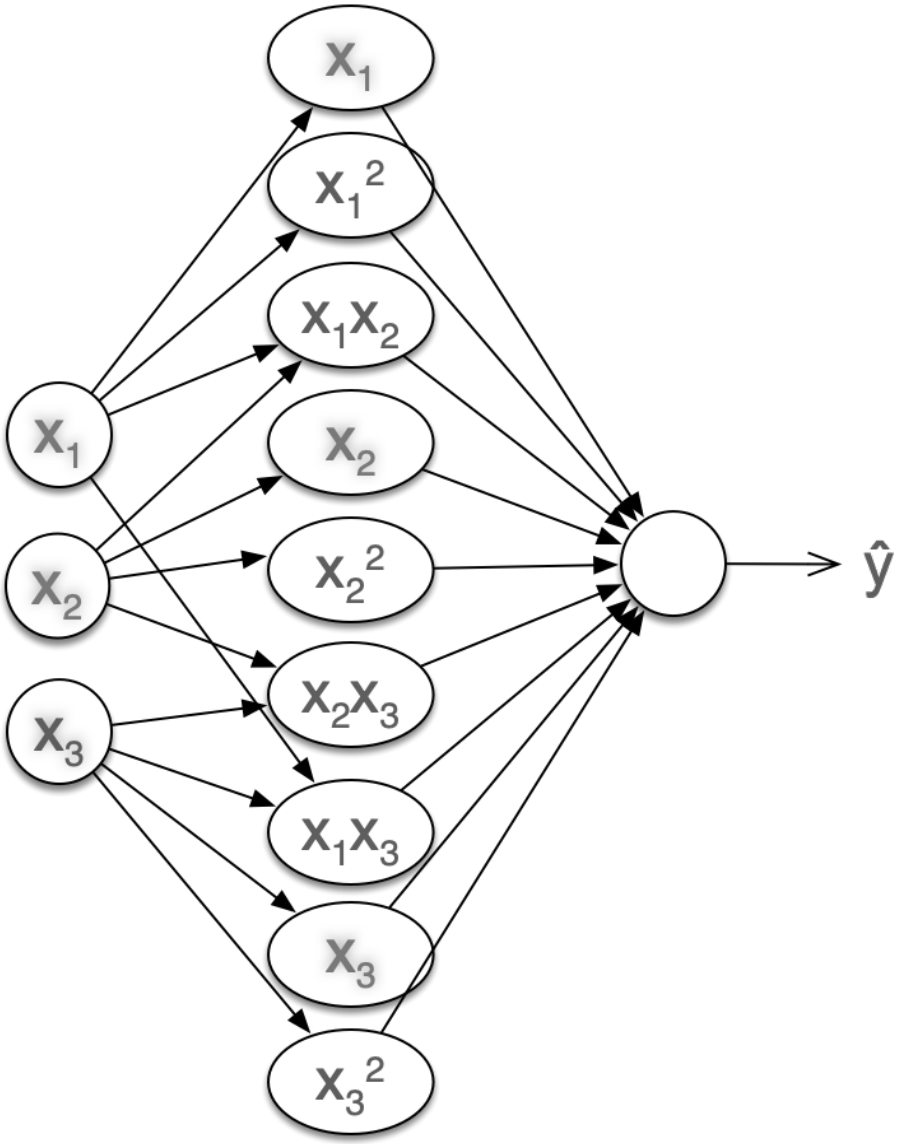
Linear versus non-linear



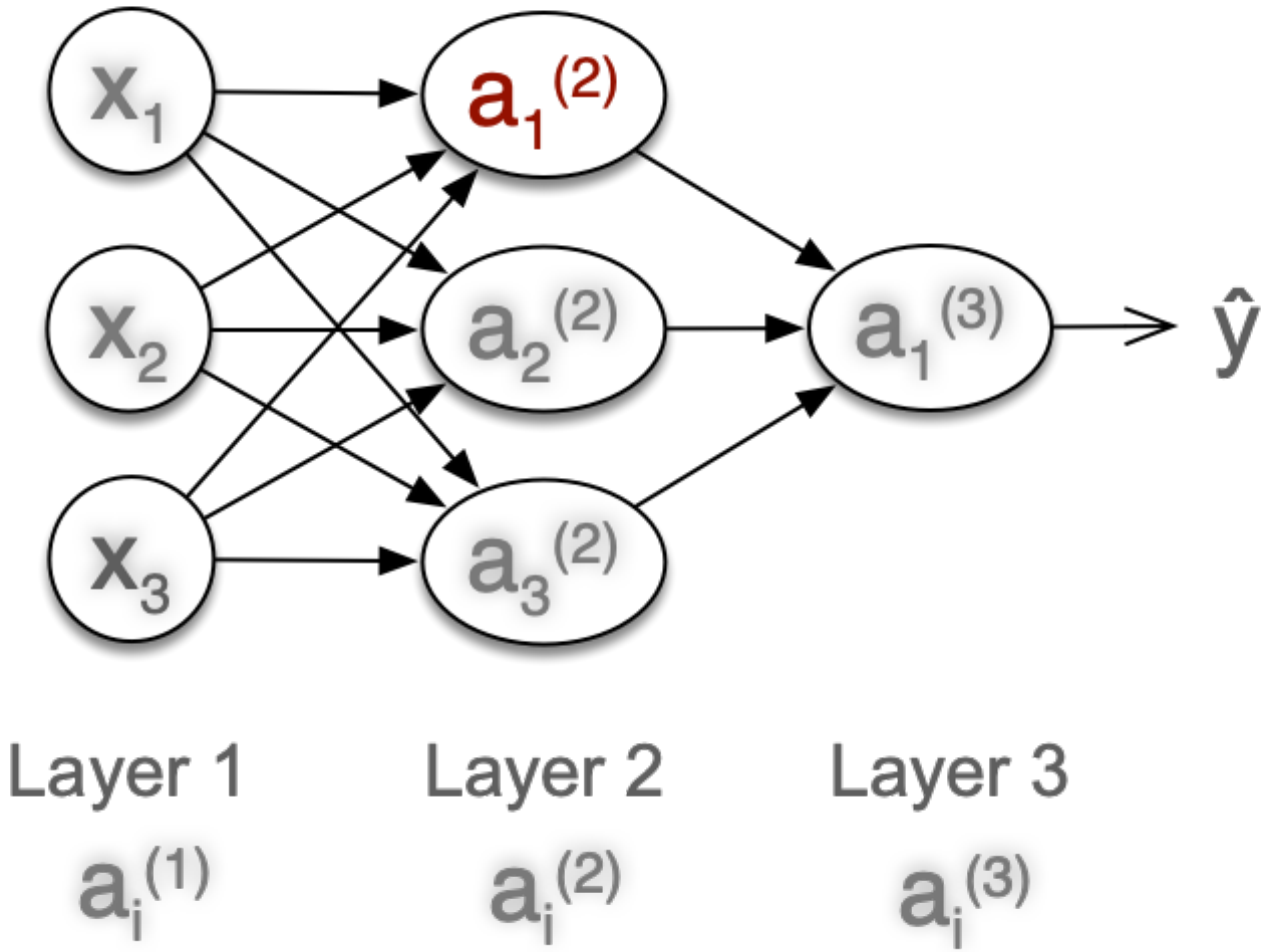
Linear versus non-linear



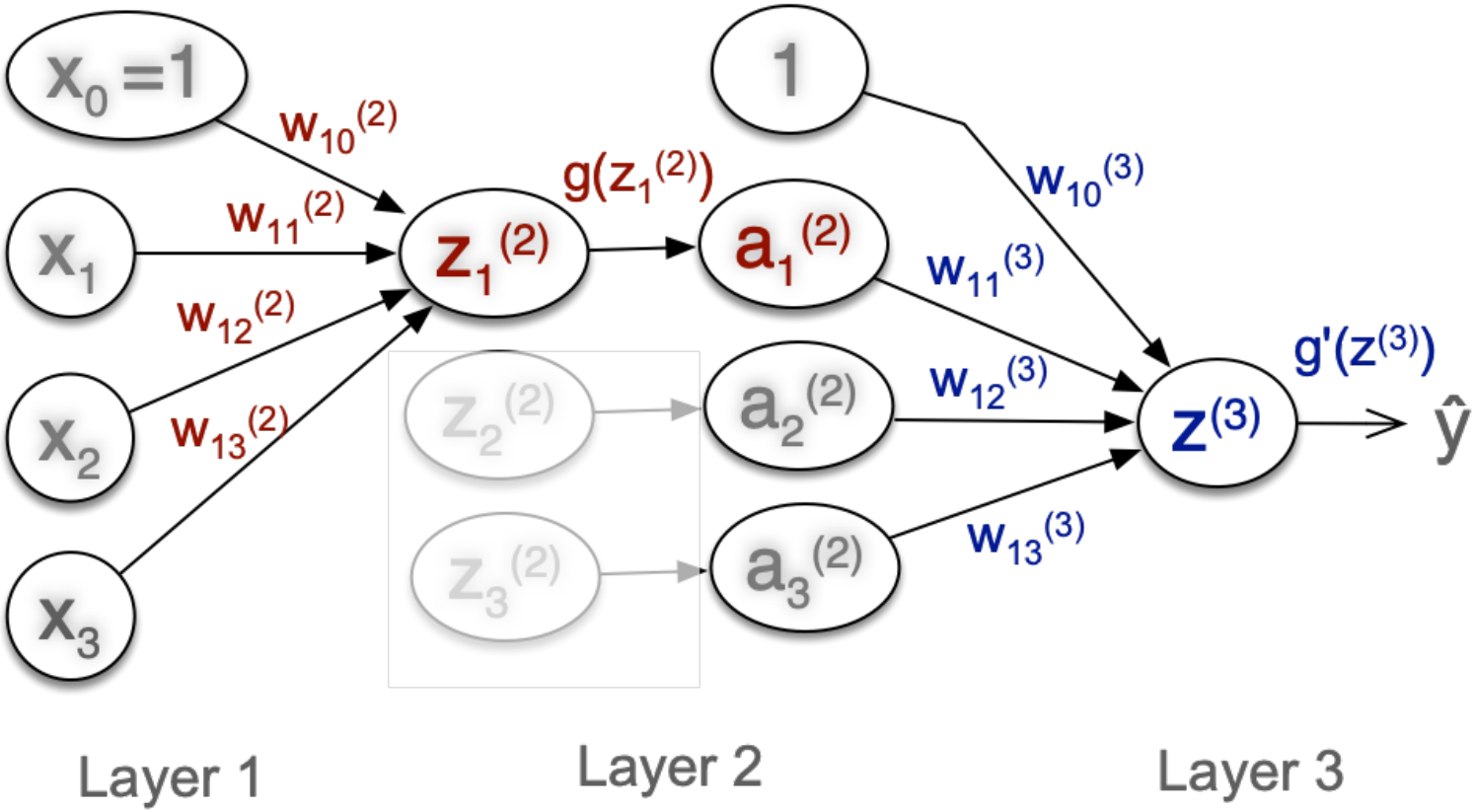
Linear versus non-linear



Neural Network



Neural Network



Neural Network

- Forward Propagation

$$\begin{cases} z_1^{(2)} = w_{10}^{(2)} x_0 + w_{11}^{(2)} x_1 + w_{12}^{(2)} x_2 + w_{13}^{(2)} x_3 \\ z_2^{(2)} = w_{20}^{(2)} x_0 + w_{21}^{(2)} x_1 + w_{22}^{(2)} x_2 + w_{23}^{(2)} x_3 \\ z_3^{(2)} = w_{30}^{(2)} x_0 + w_{31}^{(2)} x_1 + w_{32}^{(2)} x_2 + w_{33}^{(2)} x_3 \end{cases} \tag{17}$$

$$\begin{cases} a_1^{(2)} = g(z_1^{(2)}) \\ a_2^{(2)} = g(z_2^{(2)}) \\ a_3^{(2)} = g(z_3^{(2)}) \end{cases} \tag{18}$$

$$z_1^{(3)} = w_{10}^{(3))} a_0^{(2)} + w_{11}^{(3))} a_1^{(2)} + w_{12}^{(3))} a_2^{(2)} + w_{13}^{(3))} a_3^{(2)} \tag{19}$$

$$\hat{y}(x_0, x_1, x_2, x_3) = a_1^{(3)} = g(z_1^{(3)}) \tag{20}$$

Neural Network

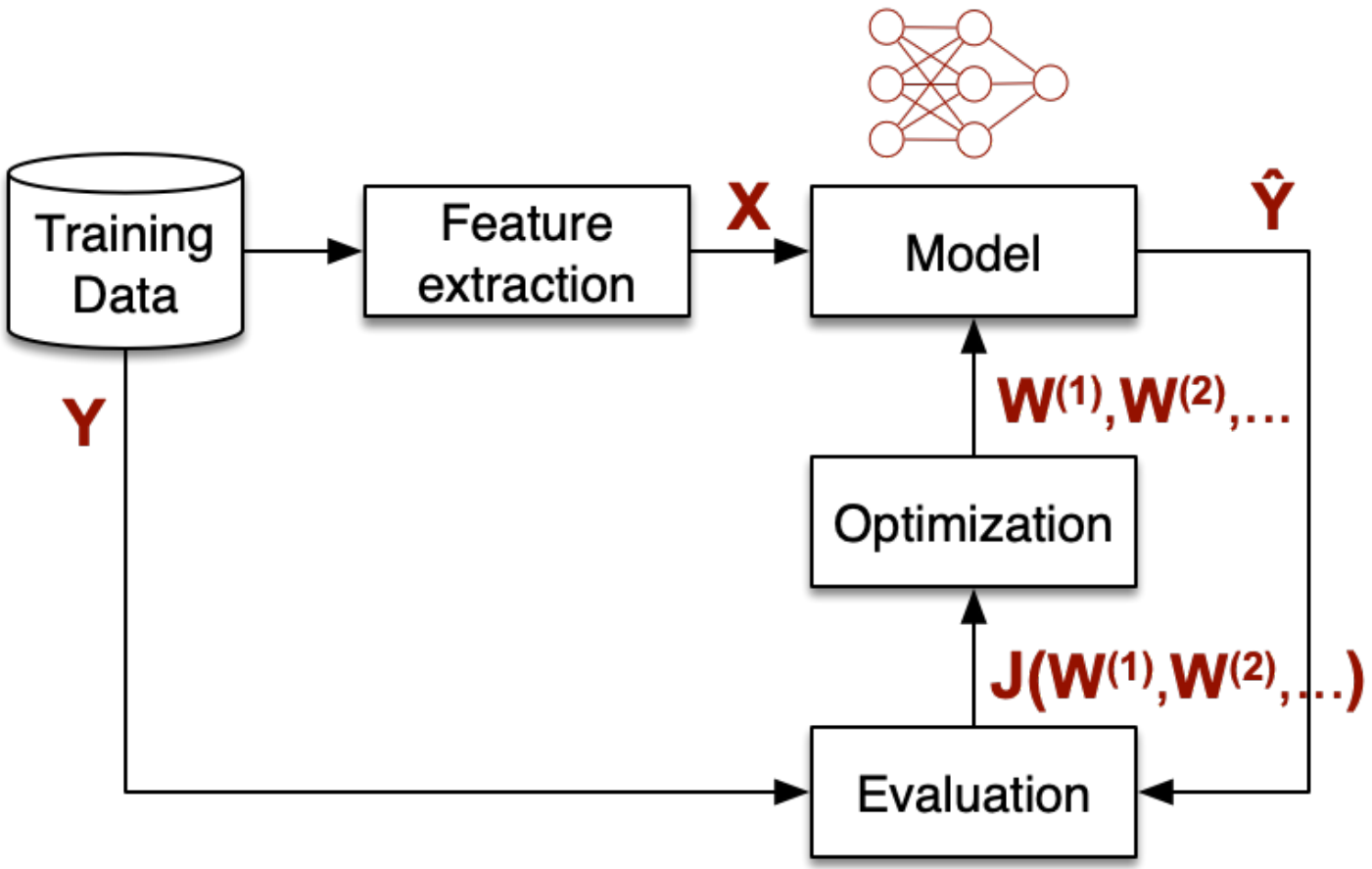
- Forward Propagation (Vectors)

$$\mathbf{X} = \mathbf{A}^{(1)} \tag{21}$$

$$\begin{cases} \mathbf{Z}^{(2)} = \mathbf{W}^{(2)}\mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} = g(\mathbf{Z}^{(2)}) \end{cases} \tag{22}$$

$$\begin{cases} \mathbf{Z}^{(3)} = \mathbf{W}^{(3)}\mathbf{A}^{(2)} \\ \hat{\mathbf{Y}}(\mathbf{X}) = g(\mathbf{Z}^{(3)}) \end{cases} \tag{23}$$

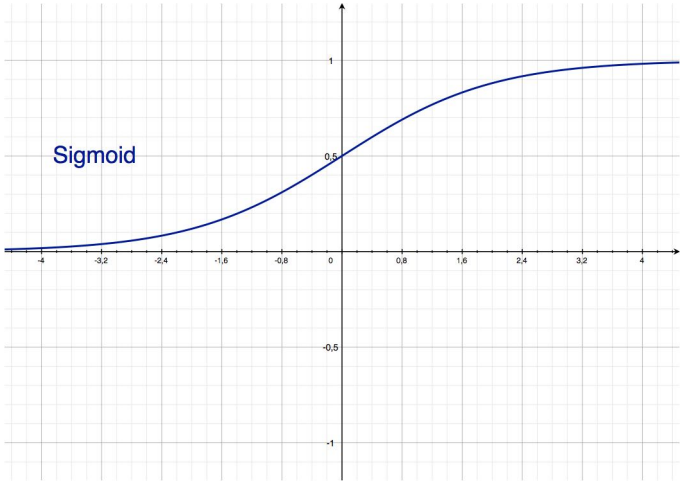
Neural Network and Machine Learning

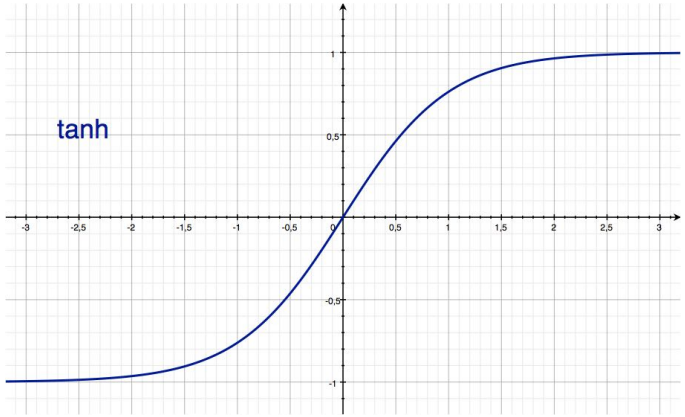


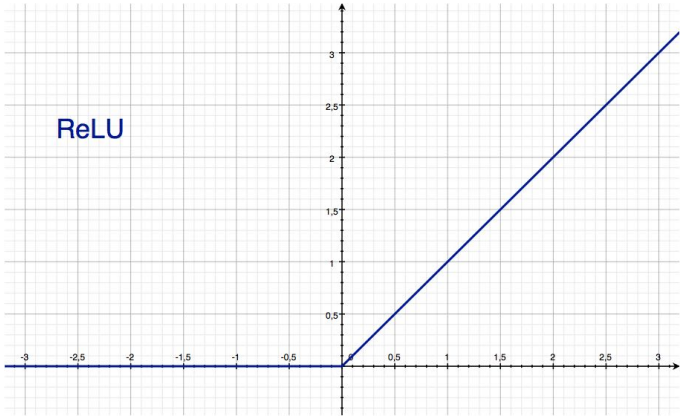
Neural Network

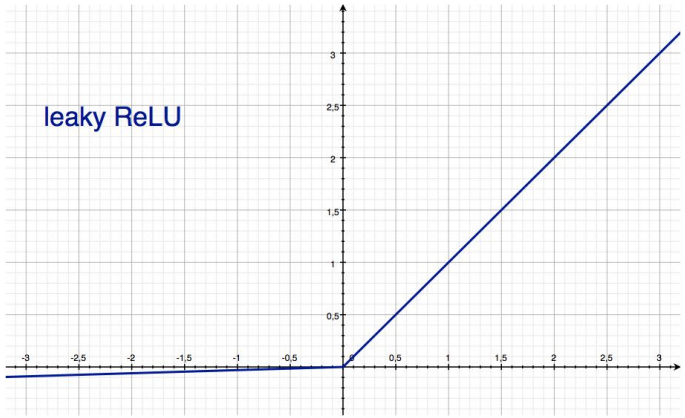
- Activation functions : g()
- Gradient

Neural Network : Activation functions









Neural Network : Gradient evaluation

- Analytical way
- Numerical way
- Gradient backward Propagation

Neural Network : Backward Propagation

$$f(...) = (ax_1 + bx_2)^2 - c \tag{24}$$

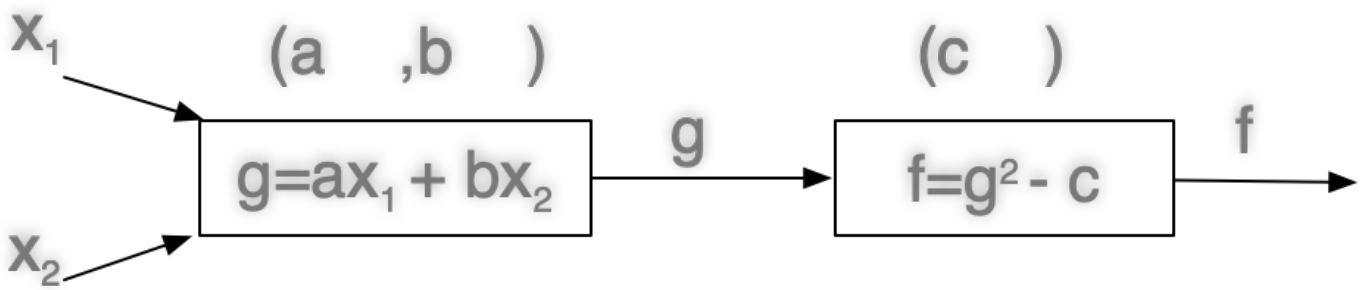
$$\begin{cases} f(...) = g^2 - c \\ g(...) = ax_1 + bx_2 \end{cases} \tag{25}$$

$$\begin{cases} \frac{\partial f}{\partial a} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial a} = 2(ax_1 + bx_2)x_1 \\ \frac{\partial f}{\partial b} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial b} = 2(ax_1 + bx_2)x_2 \end{cases} \tag{26}$$

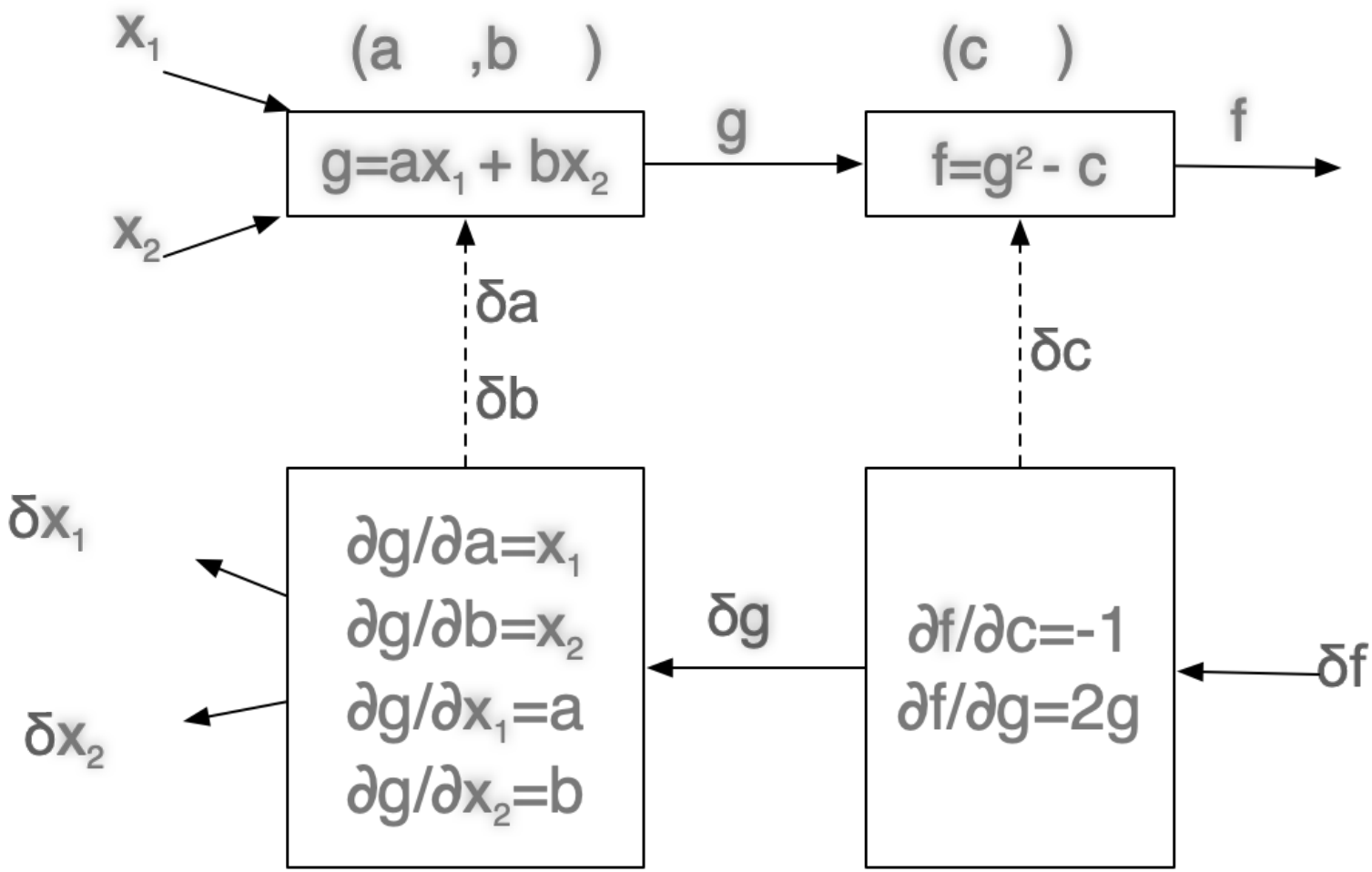
$$x_1 = 2, x_2 = -1, a = 2, b = 1, c = 2$$

$$\begin{cases} \frac{\partial f}{\partial a} = 12 \\ \frac{\partial f}{\partial b} = -6 \end{cases} \tag{27}$$

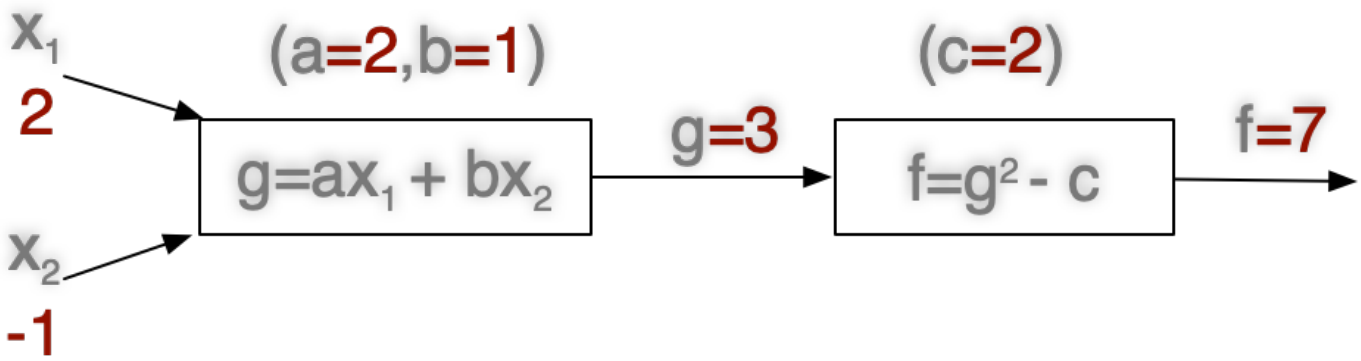
Neural Network : Backward Propagation



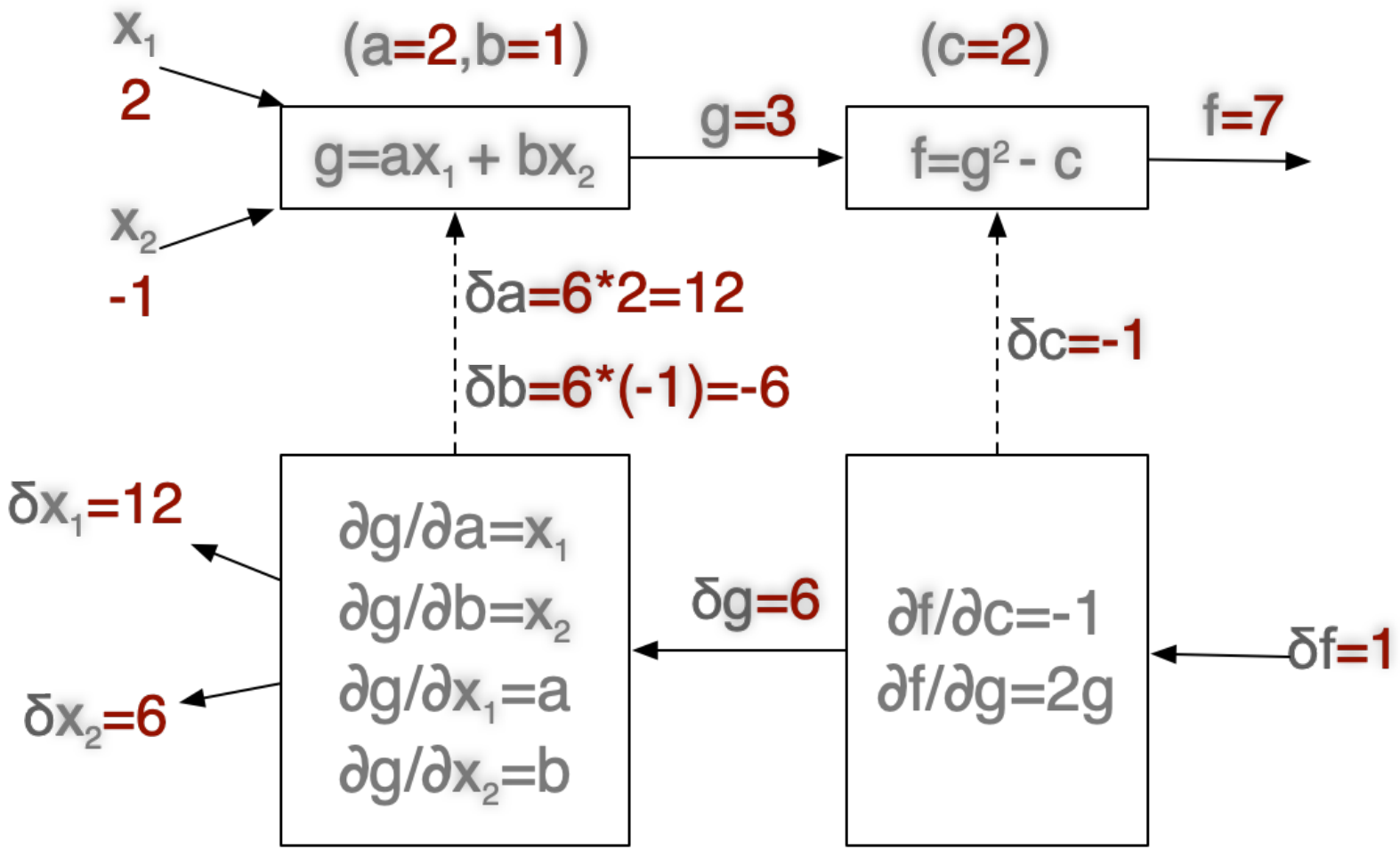
Neural Network : Backward Propagation



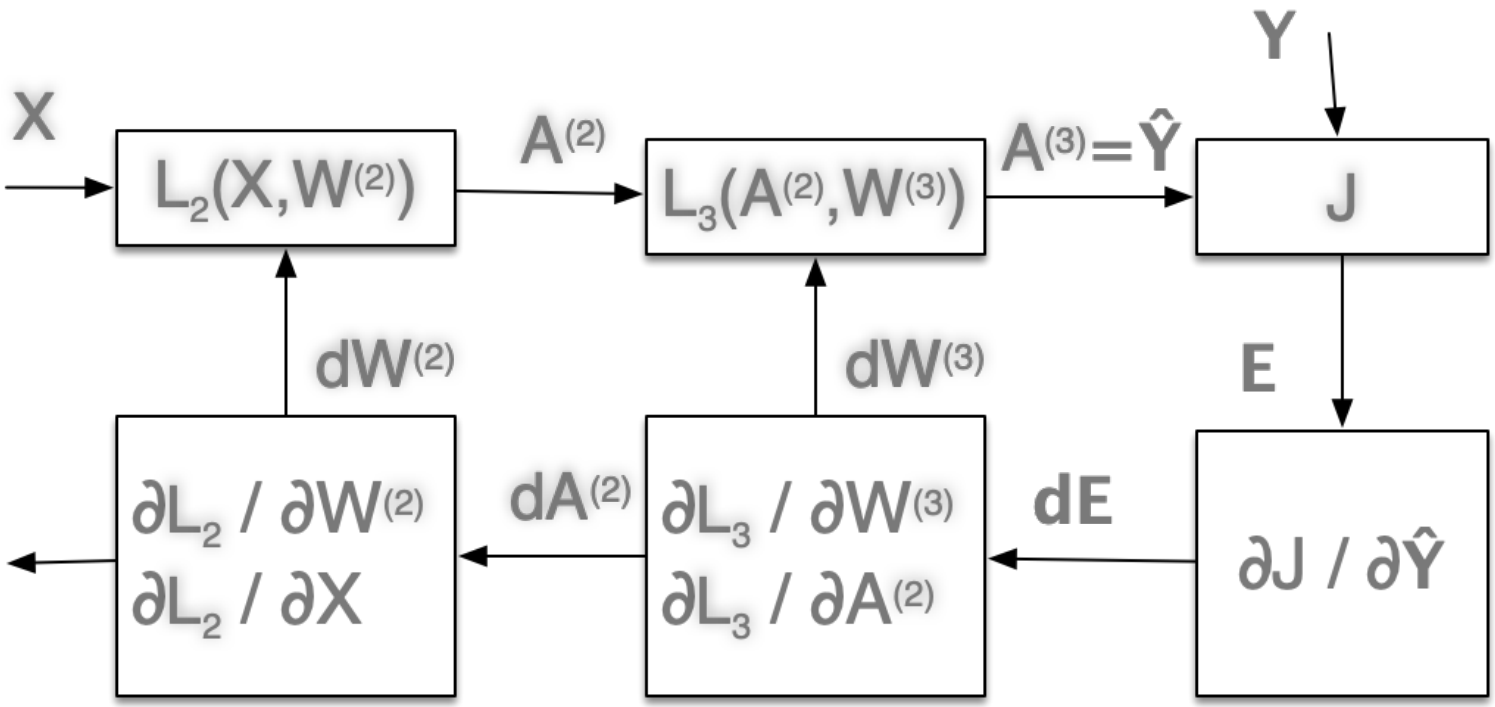
Neural Network : Backward Propagation



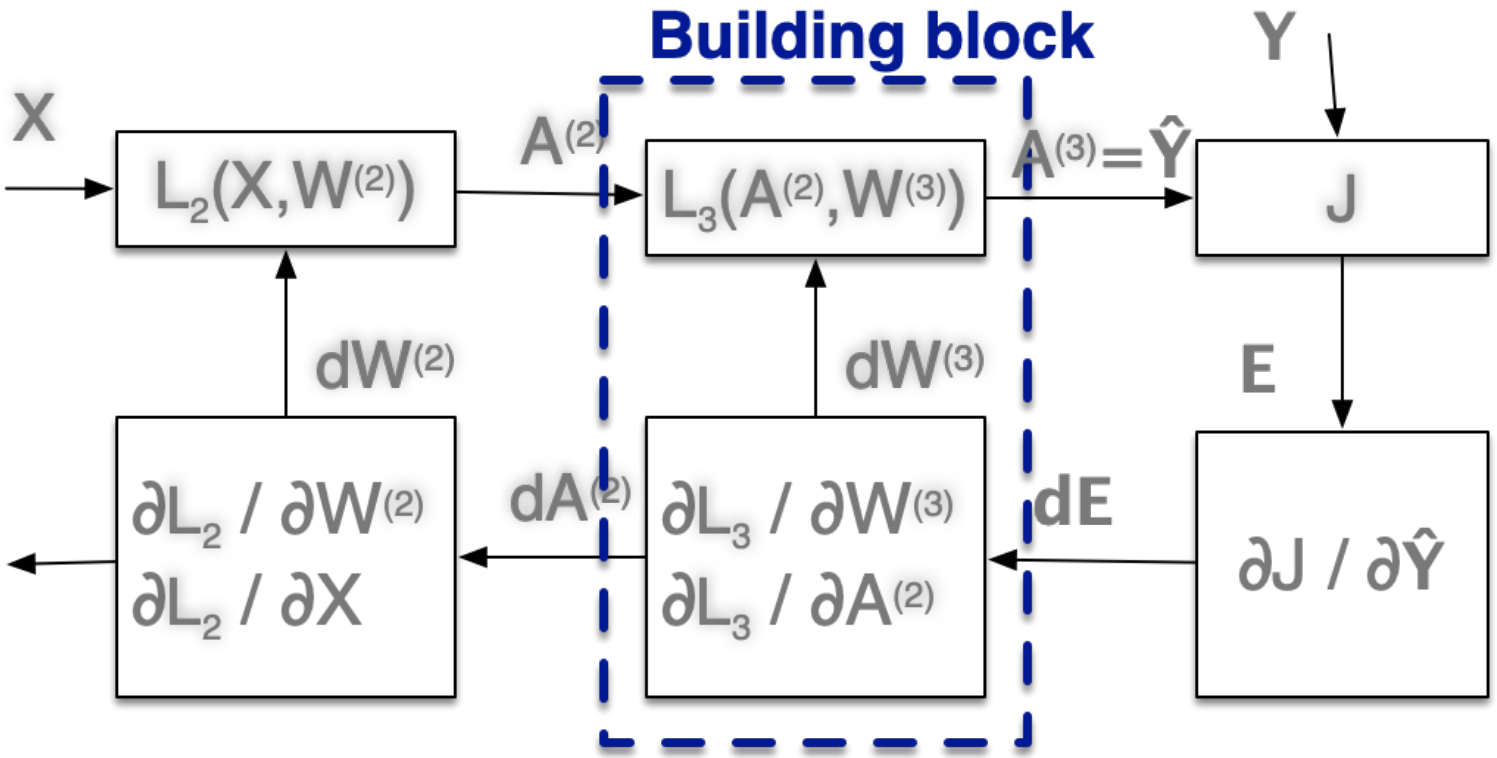
Neural Network : Backward Propagation



Neural Network : Backward Propagation



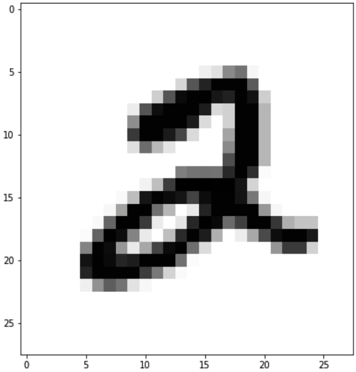
Neural Network : Backward Propagation



Neural Network (Example)

- MNIST : DATABASE of handwritten digits
- 28x28 pixels image

Neural Network (Example)



00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0d	19	64	7a	07	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	21	97	d0	fc	fc	fc	92	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	28	98	f4	fc	fd	e0	d3	fc	e8	28	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	0f	98	ef	fc	fc	fc	d8	1f	25	fc	fc	3c	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	60	fc	fc	fc	fc	d9	1d	00	25	fc	fc	3c	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	b5	fc	fc	dc	a7	1e	00	00	4d	fc	fc	3c	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	1a	80	3a	16	00	00	00	00	64	fc	fc	3c	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	9d	fc	fc	3c	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	6e	79	7a	79	ca	fc	c2	03	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	0a	35	b3	fd	fd	ff	fd	fd	e4	23	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	05	36	e3	fc	f3	e4	aa	f2	fc	fc	e7	75	06	00	00	00	00	00	00	00
00	00	00	00	00	00	00	06	4e	fc	fc	7d	3b	00	12	d0	fc	fc	fc	fc	57	07	00	00	00	00	00	00
00	00	00	00	00	00	05	87	fc	fc	b4	10	00	15	cb	fd	f7	81	ad	fc	fc	b8	42	31	31	00	00	00
00	00	00	00	00	03	88	fc	f1	6a	11	00	35	c8	fc	d8	41	00	0e	48	a3	f1	fc	fc	df	00	00	00
00	00	00	00	00	69	fc	f2	58	12	49	aa	f4	fc	7e	1d	00	00	00	00	00	59	b4	b4	25	00	00	00
00	00	00	00	00	e7	fc	f5	cd	d8	fc	fc	fc	7c	03	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	cf	fc	fc	fc	fc	b2	74	24	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	0d	5d	8f	79	17	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

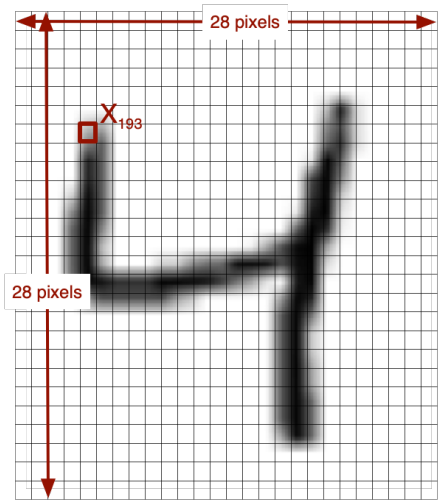
Neural Network (Example)

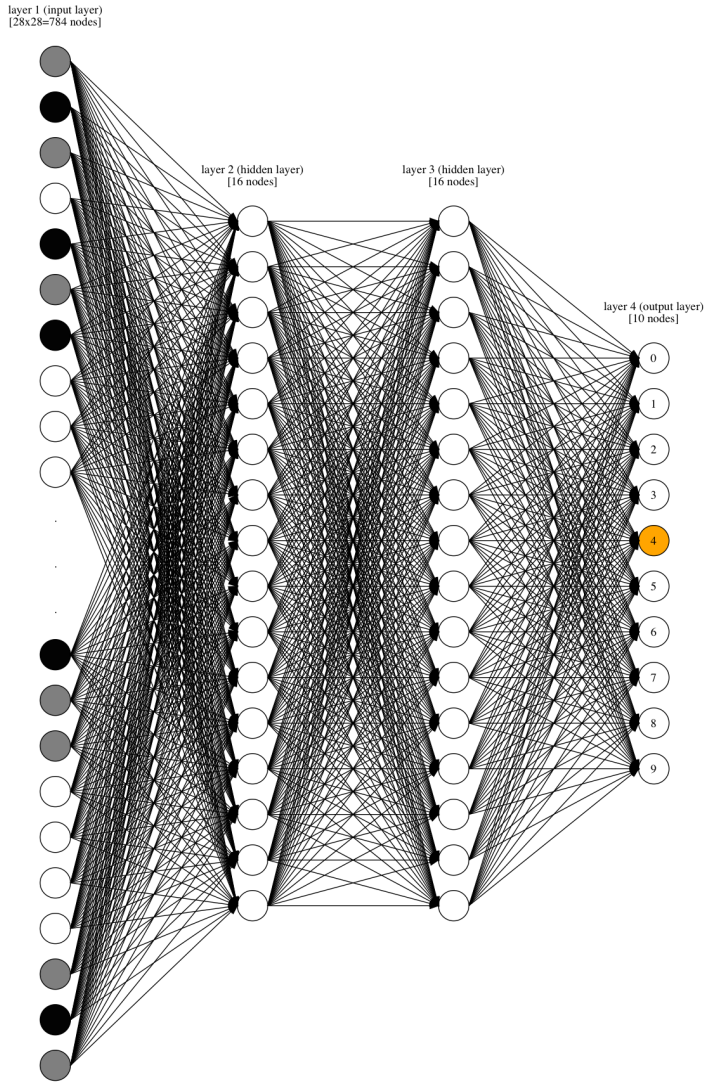
- Data :
  - Input : 28x28 gray pixels = vector<784> : 0..255
  - Training set : 60000 images
  - Test set : 10000 images
- Topology :
  - Input : 28x28=784 nodes
  - Output : 10 nodes (sofmax)

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} \tag{28}$$

  - 2 hidden layers (16 nodes)
  - Number of parameters : 13002
  - Loss function : cross-entropy

Neural Network (Example)





Neural Network (Example) using Keras

In [14]:
```
simple_model = models.Sequential()

simple_model.add(layers.Dense(16, activation='relu', input_shape=(28*28*1,)))
simple_model.add(layers.Dense(16, activation='relu'))
simple_model.add(layers.Dense(10, activation='softmax'))
```

In [15]:
```
simple_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                12560
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
dense_3 (Dense)              (None, 10)                170
=================================================================
Total params: 13,002
Trainable params: 13,002
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
simple_model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Neural Network (Example) using Keras

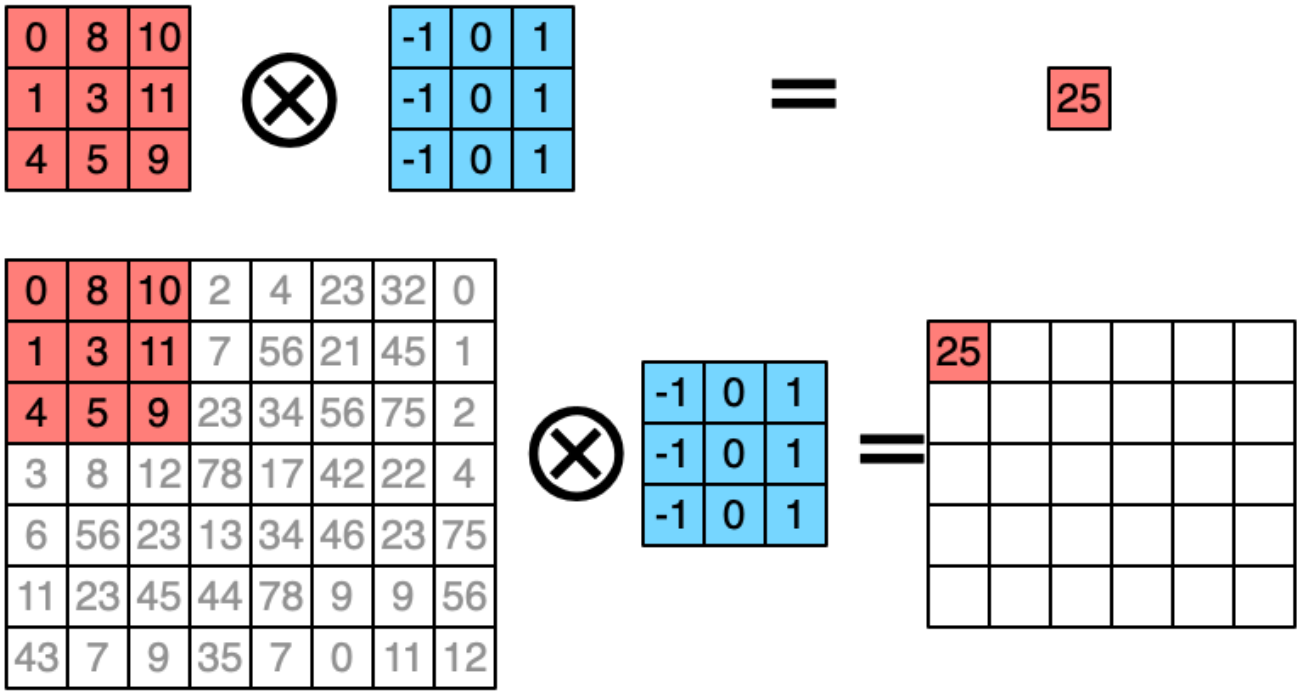
```
In [17]: simple_model.fit(train_images, train_labels, epochs=10, batch_size=64)

Epoch 1/10
60000/60000 [==============================] - 1s - loss: 0.5220 - acc: 0.8526
Epoch 2/10
60000/60000 [==============================] - 1s - loss: 0.2665 - acc: 0.9234     - ETA
Epoch 3/10
60000/60000 [==============================] - 1s - loss: 0.2302 - acc: 0.9339
Epoch 4/10
60000/60000 [==============================] - 1s - loss: 0.2070 - acc: 0.9413
Epoch 5/10
60000/60000 [==============================] - 1s - loss: 0.1895 - acc: 0.9458
Epoch 6/10
60000/60000 [==============================] - 1s - loss: 0.1778 - acc: 0.9485
Epoch 7/10
60000/60000 [==============================] - 1s - loss: 0.1692 - acc: 0.9509
Epoch 8/10
60000/60000 [==============================] - 2s - loss: 0.1610 - acc: 0.9528
Epoch 9/10
60000/60000 [==============================] - 1s - loss: 0.1544 - acc: 0.9553
Epoch 10/10
60000/60000 [==============================] - 1s - loss: 0.1500 - acc: 0.9562

Out[17]: <keras.callbacks.History at 0x1293b3c88>

In [20]: simple_model.evaluate(test_images, test_labels)

10000/10000 [==============================] - 0s

Out[20]: [0.16147010266780853, 0.9524]
```

Convolutional Neural Network

- Motivations :
  - Reduce number of parameters
  - Feature extractor
- ConvNet (CNN) :
  - Convolution filter
  - Pooling

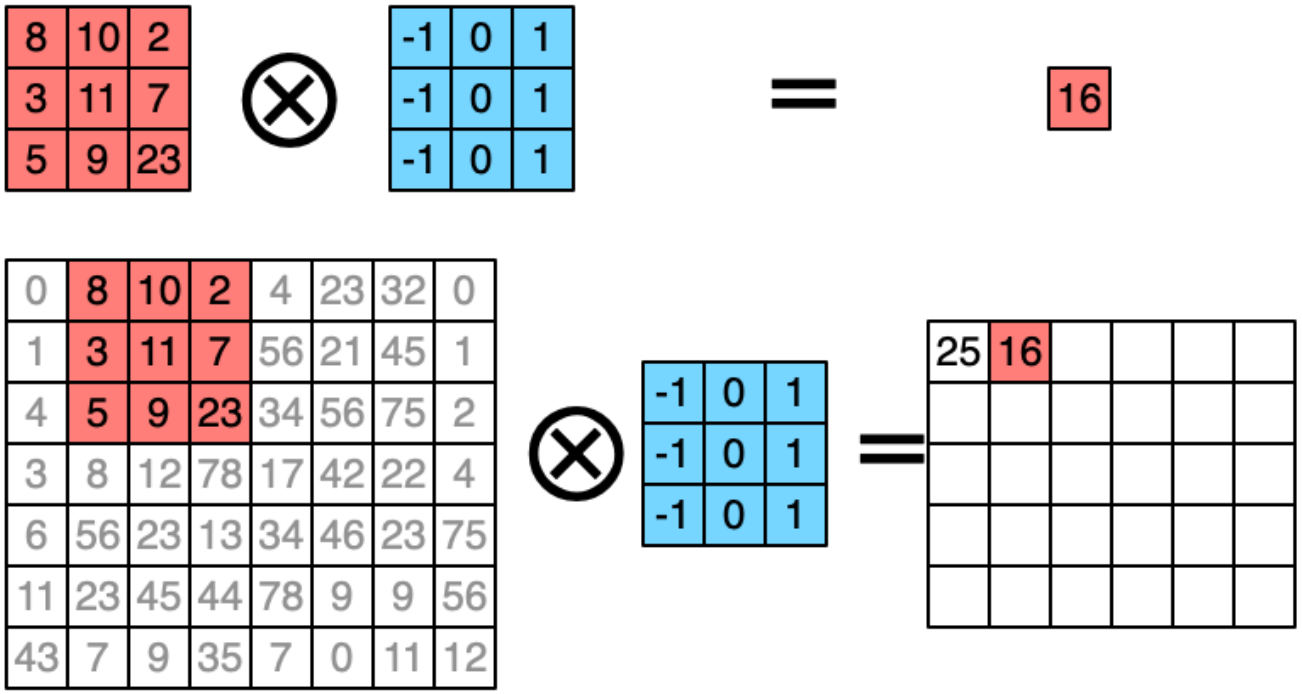
ConvNet : Convolution

$$X \otimes W = \sum\sum X_{i,j} W_{i,j}$$



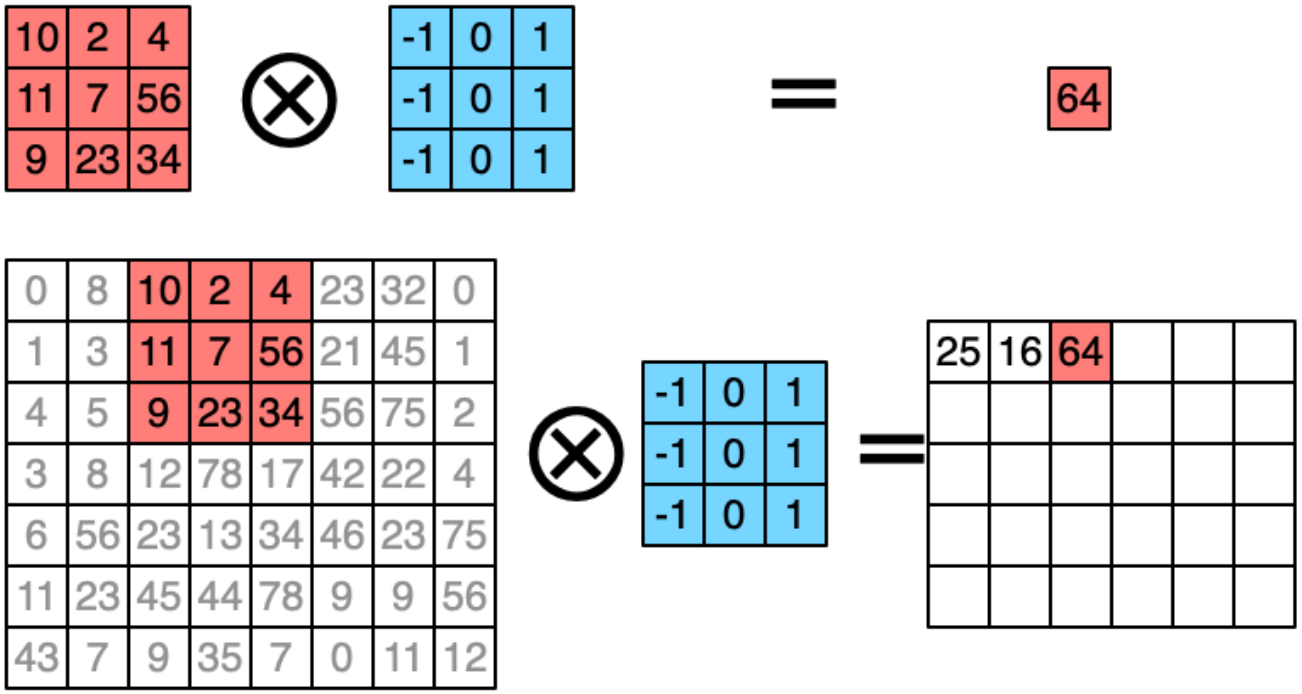
ConvNet : Convolution

$$X \otimes W = \sum\sum X_{i,j} W_{i,j}$$



ConvNet : Convolution

$$X \otimes W = \sum\sum X_{i,j} W_{i,j}$$



ConvNet : Convolution

$$X \otimes W = \sum\sum X_{i,j} W_{i,j}$$

46	23	75
9	9	56
0	11	12

$\otimes$

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

$=$ 88

0	8	10	2	4	23	32	0
1	3	11	7	56	21	45	1
4	5	9	23	34	56	75	2
3	8	12	78	17	42	22	4
6	56	23	13	34	46	23	75
11	23	45	44	78	9	9	56
43	7	9	35	7	0	11	12

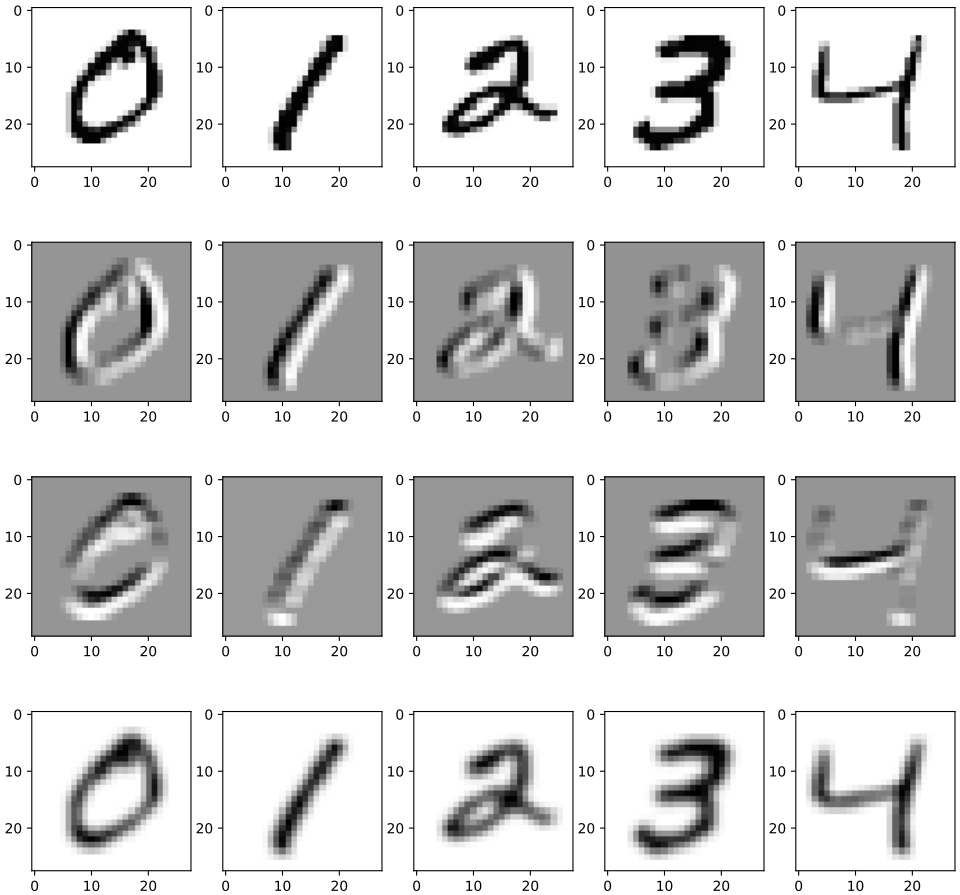
$\otimes$

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

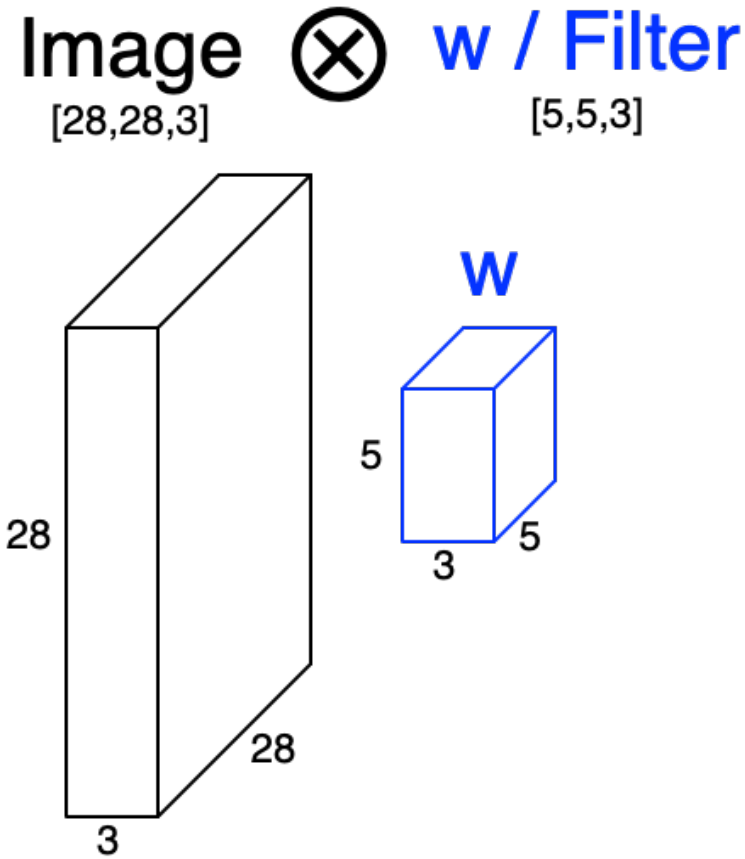
$=$

25	16	64	68	58	-97
24	92	75	11	35	-112
31	45	41	30	35	-63
60	48	49	-38	-75	38
17	6	42	-37	-76	88

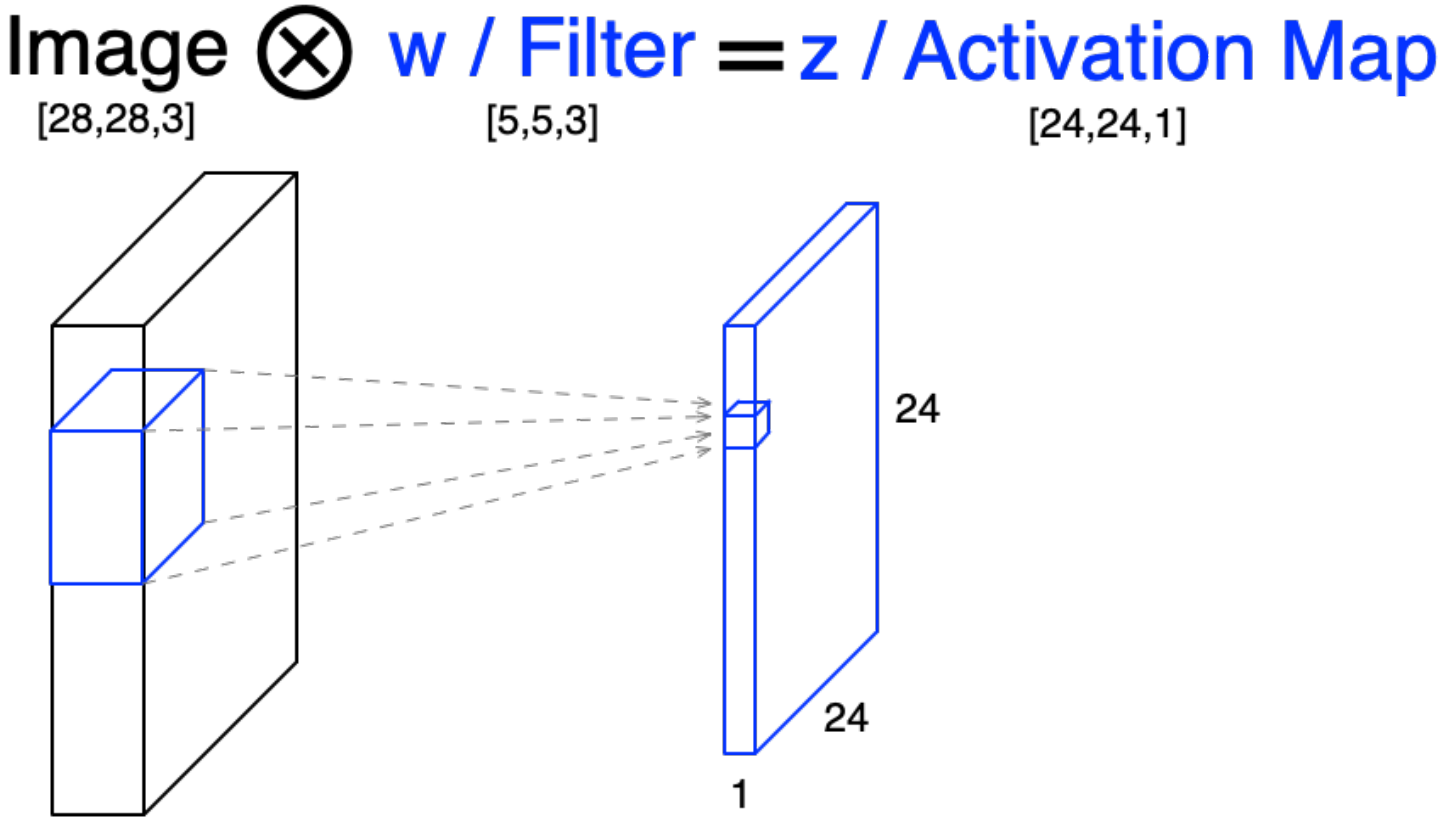
ConvNet : Convolution examples



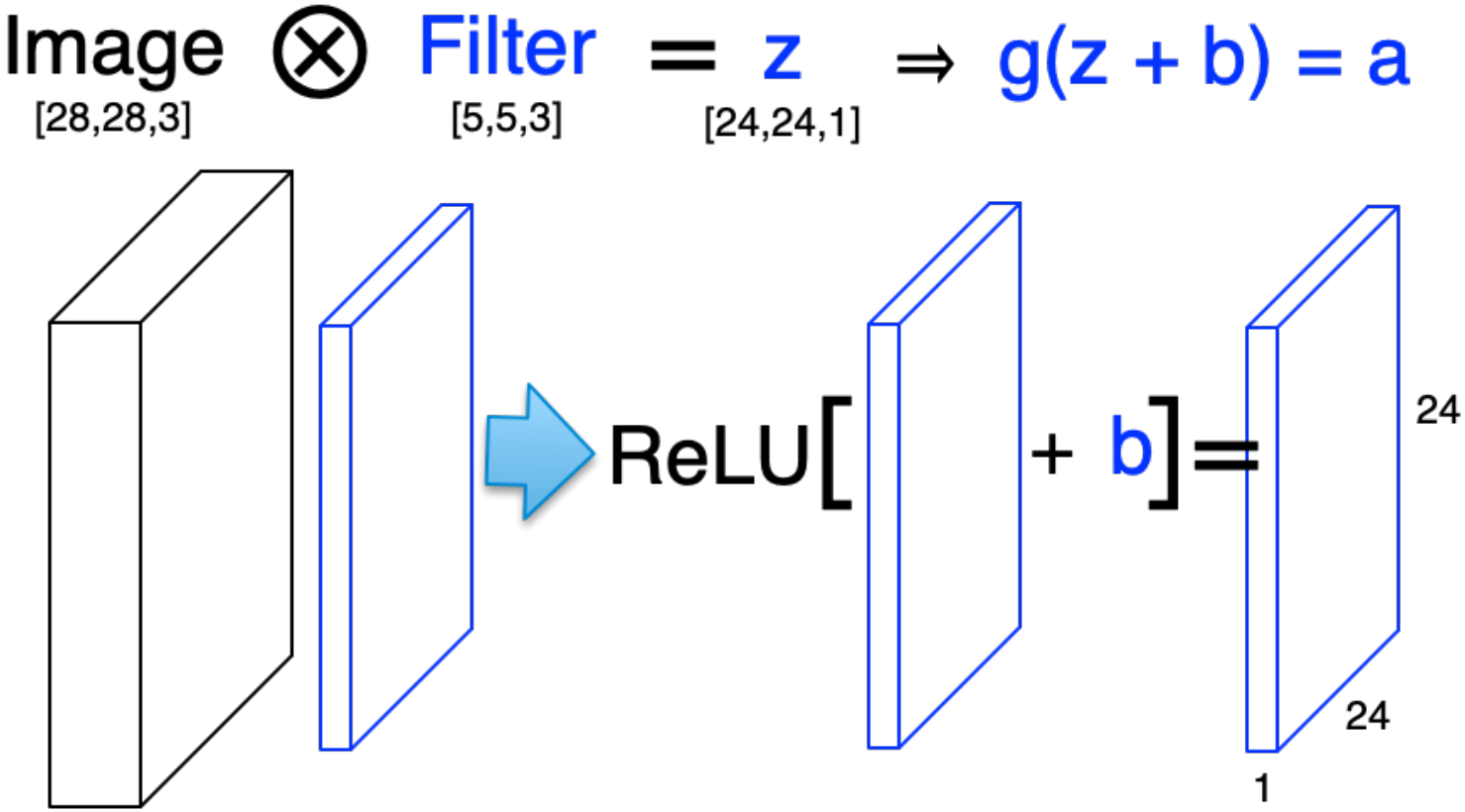
ConvNet : Convolution



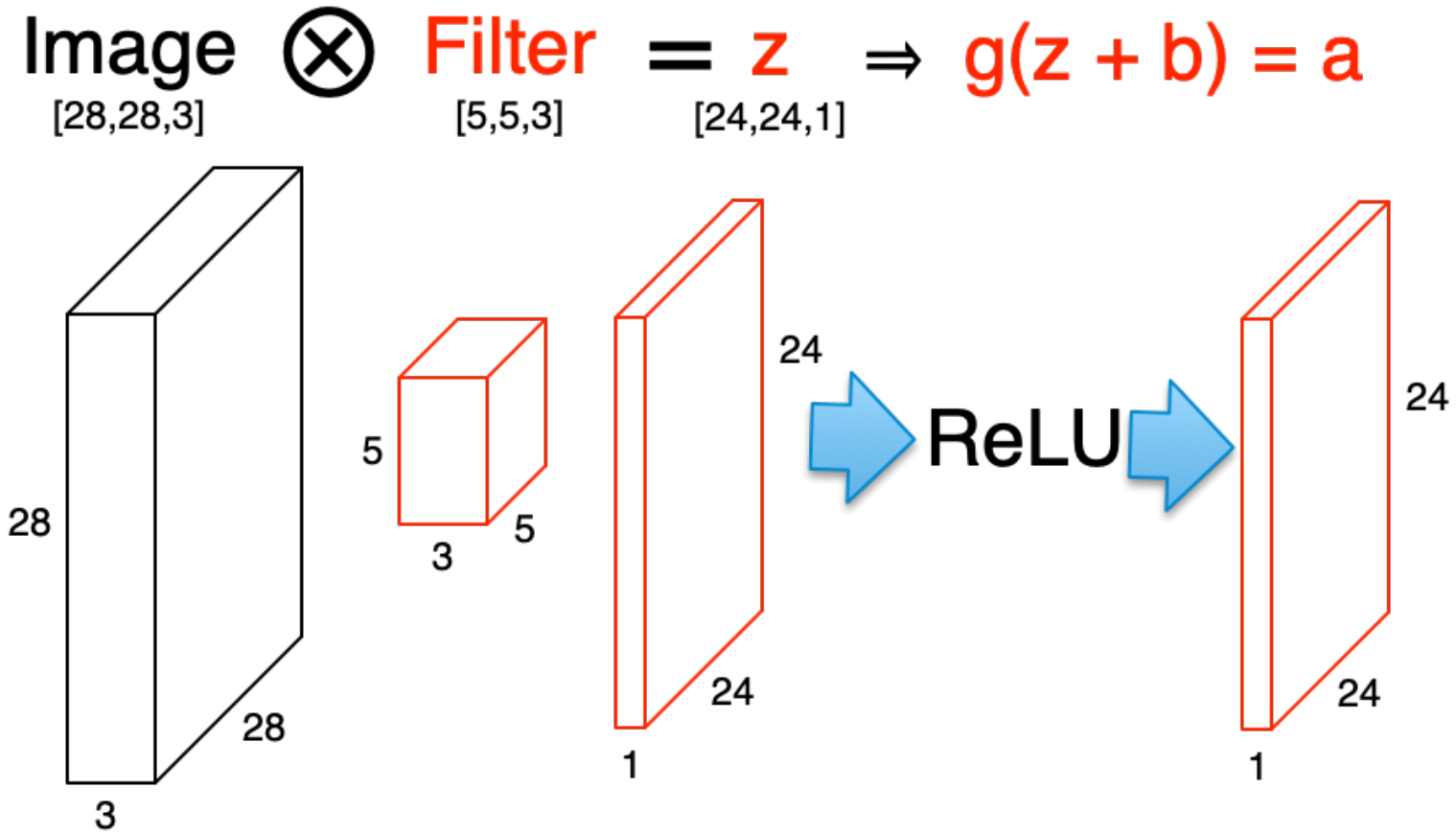
ConvNet : Convolution



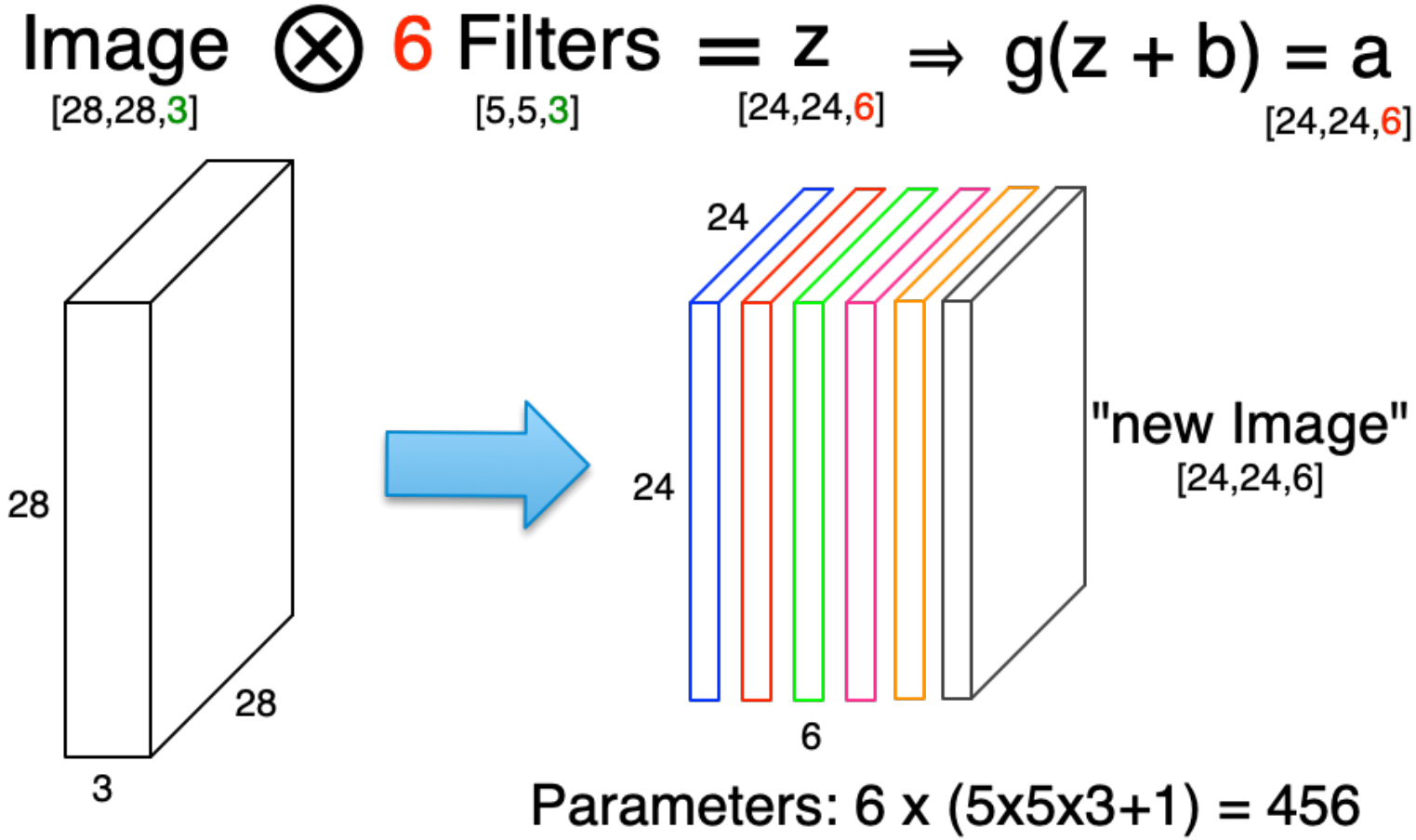
ConvNet : Convolution



ConvNet : Convolution



ConvNet : Convolution



ConvNet : Pooling

Filter: 2x2
Stride: 2

25	16	64	68	58	0
24	92	75	11	35	0
31	45	41	30	35	0
60	48	49	0	0	38
17	6	42	0	0	88
32	0	24	2	19	34

92		

ConvNet : Pooling

Filter: 2x2
Stride: 2

25	16	64	68	58	0
24	92	75	11	35	0
31	45	41	30	35	0
60	48	49	0	0	38
17	6	42	0	0	88
32	0	24	2	19	34

92	75	

ConvNet : Pooling

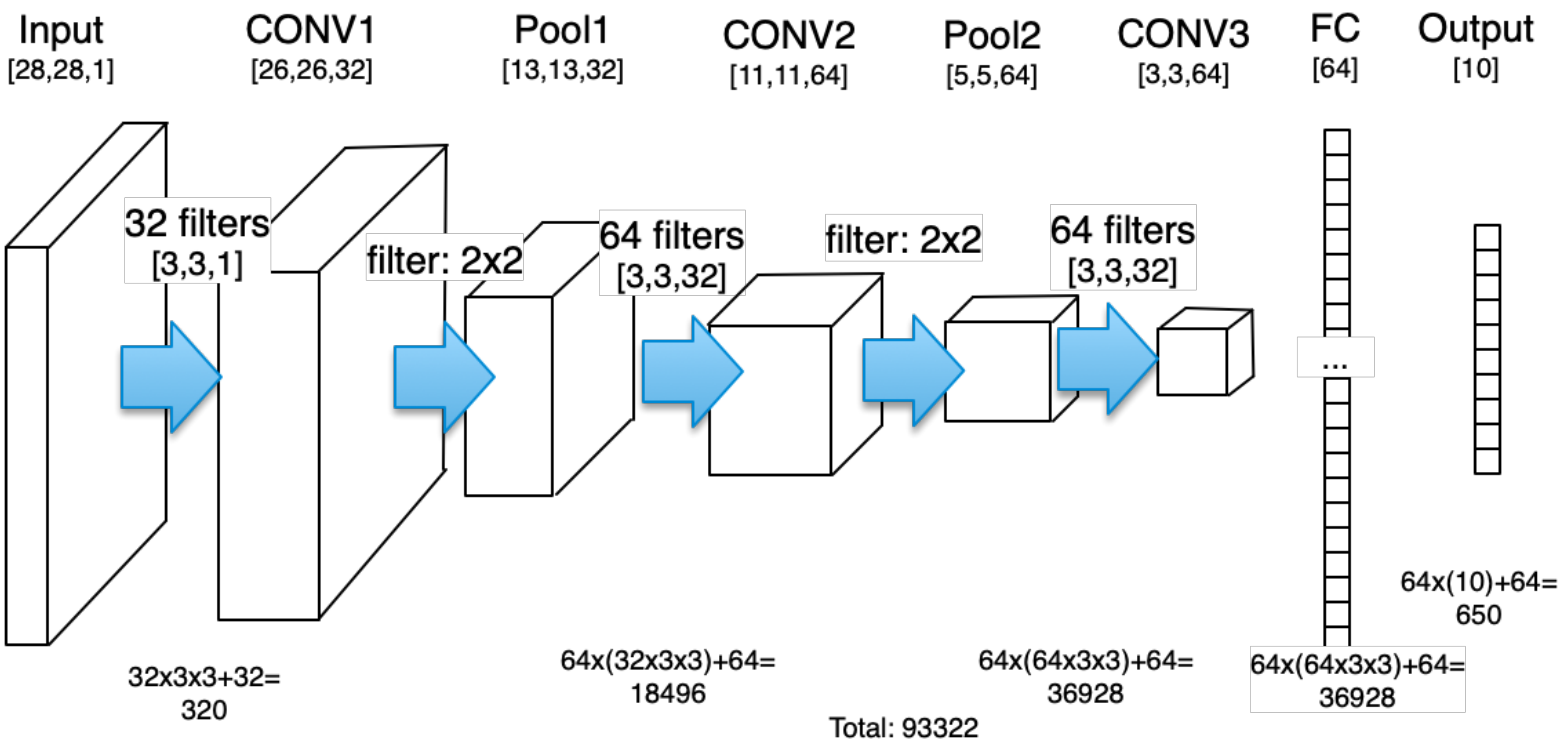
Filter: 2x2
Stride: 2

25	16	64	68	58	0
24	92	75	11	35	0
31	45	41	30	35	0
60	48	49	0	0	38
17	6	42	0	0	88
32	0	24	2	19	34

92	75	58
60	49	38
32	42	88

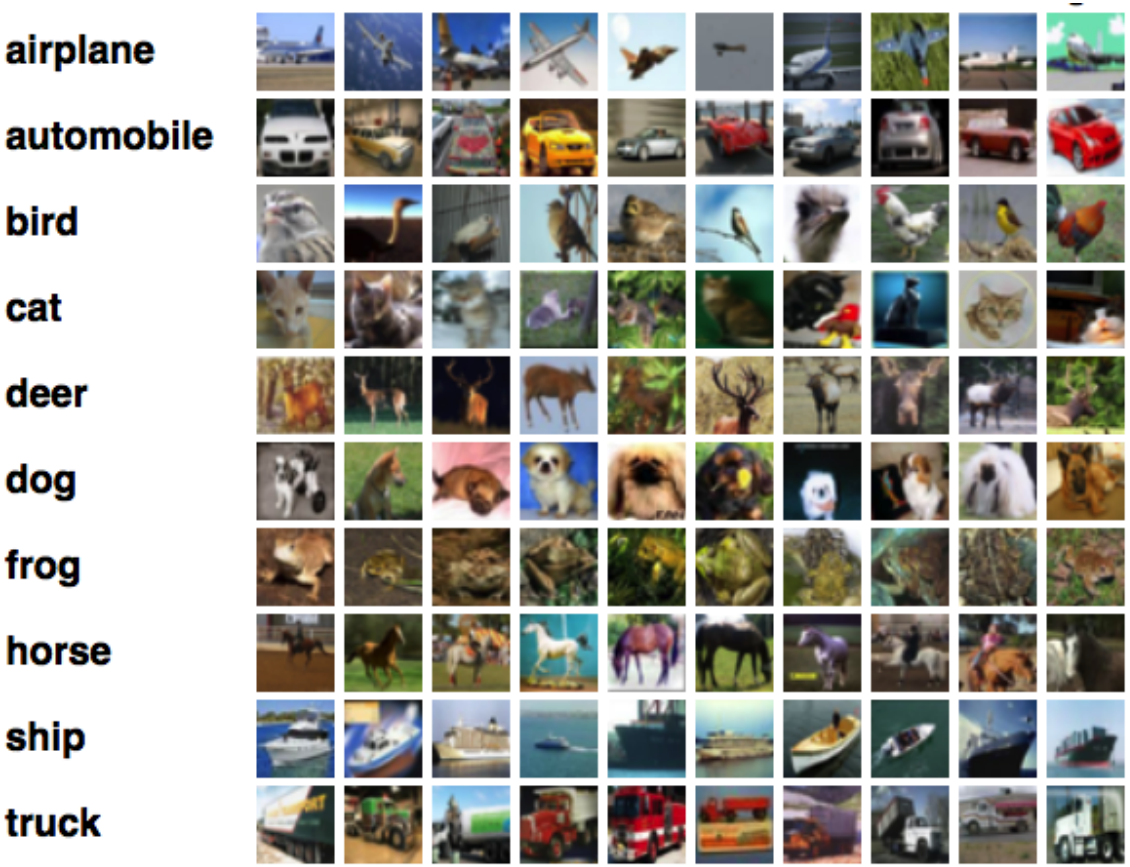
Parameters: 0

ConvNet : overview



ConvNet : Example

- CIFAR-10 : DATABASE of 60000 tiny images
- 32x32x3 colored images based on 10 classes (label)



ConvNet : Example using Keras

In [21]:
```
model = Sequential()

model.add(Conv2D(16, (3,3), activation='relu', padding='same', input_shape=(32,32,3)))
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(Conv2D(64, (3,3), activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(NUM_CLASSES, activation='softmax'))
```

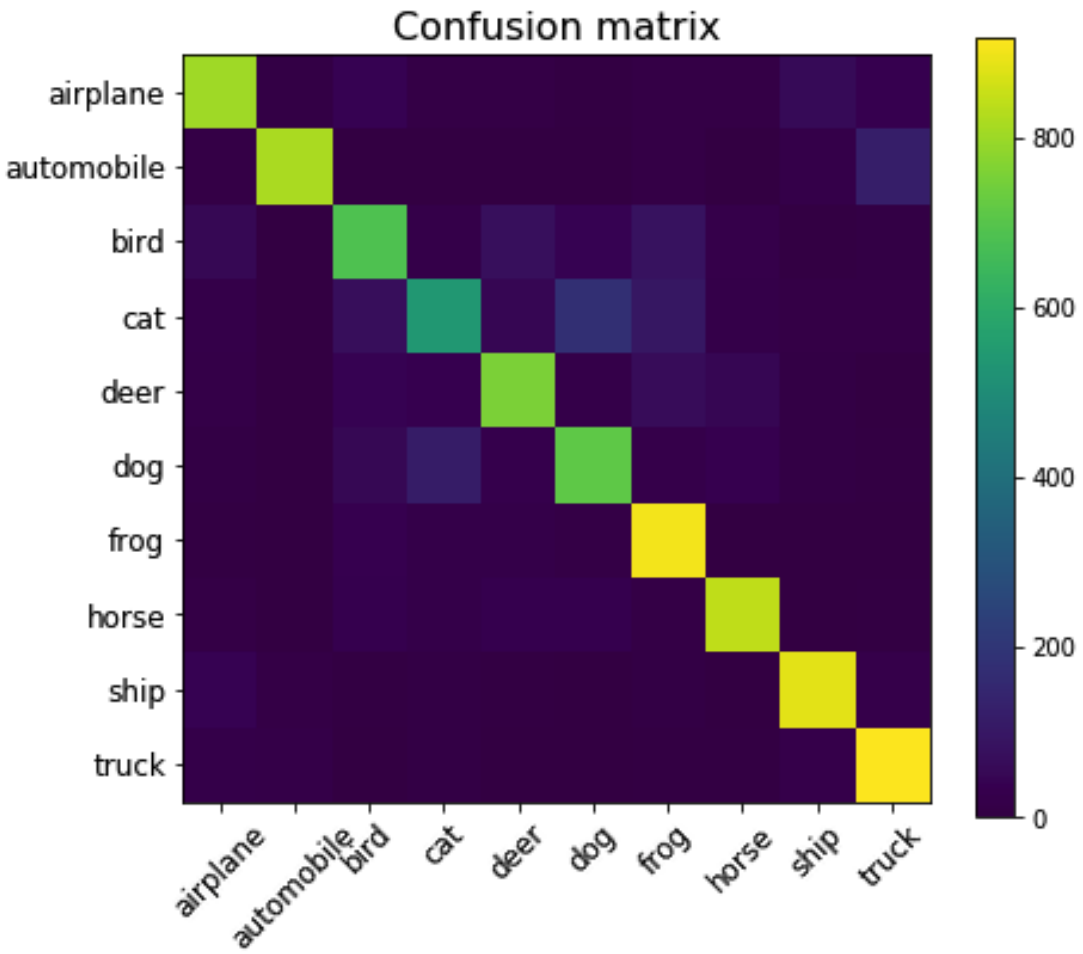
In [22]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 32, 32, 16)        448
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 32, 32, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 32)        9248
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               1048832
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,084,234
Trainable params: 1,084,234
Non-trainable params: 0
```

ConvNet : Example using Keras

```
In [13]: model.fit(x_train2, y_train2, epochs=10, batch_size=32)

Epoch 1/10
50000/50000 [==============================] - 247s - loss: 1.3442 - acc: 0.5169
Epoch 2/10
50000/50000 [==============================] - 267s - loss: 0.9575 - acc: 0.6631
Epoch 3/10
50000/50000 [==============================] - 284s - loss: 0.8379 - acc: 0.7080
Epoch 4/10
50000/50000 [==============================] - 267s - loss: 0.7680 - acc: 0.7327
Epoch 5/10
50000/50000 [==============================] - 264s - loss: 0.7137 - acc: 0.7528
Epoch 6/10
50000/50000 [==============================] - 251s - loss: 0.6702 - acc: 0.7662
Epoch 7/10
50000/50000 [==============================] - 238s - loss: 0.6401 - acc: 0.7770
Epoch 8/10
50000/50000 [==============================] - 244s - loss: 0.6105 - acc: 0.7874
Epoch 9/10
50000/50000 [==============================] - 246s - loss: 0.5906 - acc: 0.7946
Epoch 10/10
50000/50000 [==============================] - 250s - loss: 0.5722 - acc: 0.8004
```

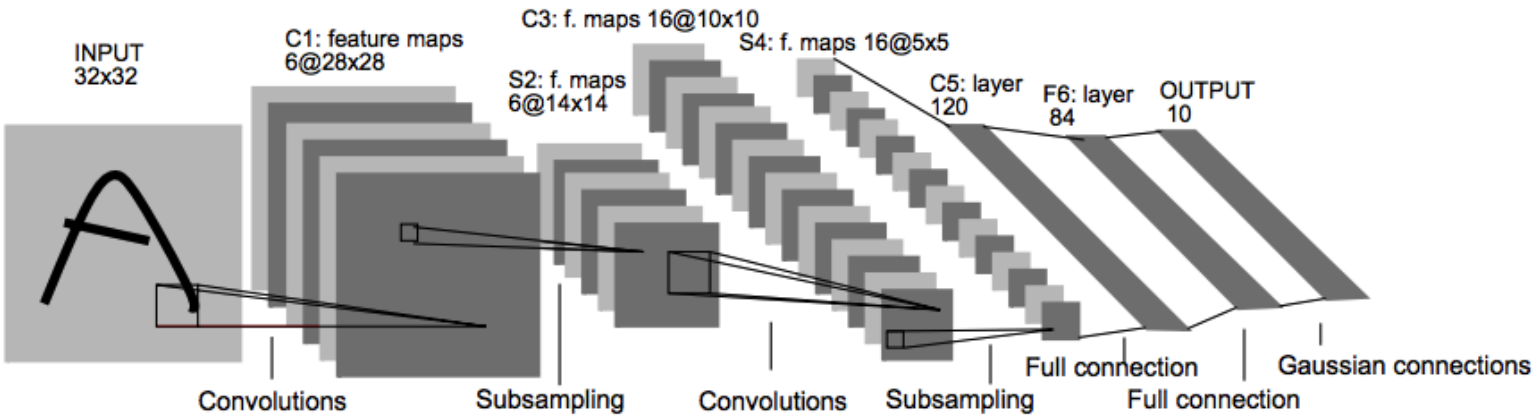
ConvNet : Example




```
Test accuracy: 0.7862
```

ConvNet : LeNet5 (1998)

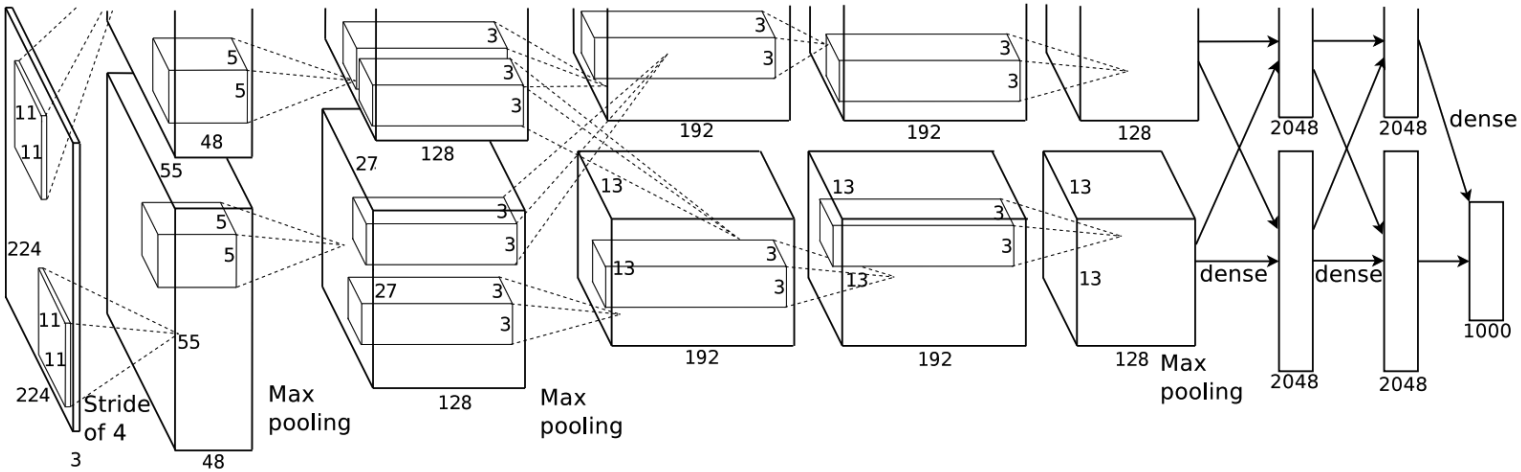
- 32x32 B&W images (MNIST : handwritten digits)
- 7 layers
- +/- 60K parameters
- Key contributions : Convolution, Average-sampling



[From : Gradient-Based Learning Applied to Document Recognition ; Y. LeCun, L. Bottou, Y. Bengio, P. Haffner]

ConvNet : AlexNet (2012)

- 227x227x3 images (ImageNet / 1000 classes)
- +/- 60M parameters (8 layers)
- Train on 2 GPUs for 6 days
- Key contributions : GPU, ReLU activation, Dropout, Image augmentation



[From : ImageNet Classification with Deep Convolutional Neural Networks ; Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton]

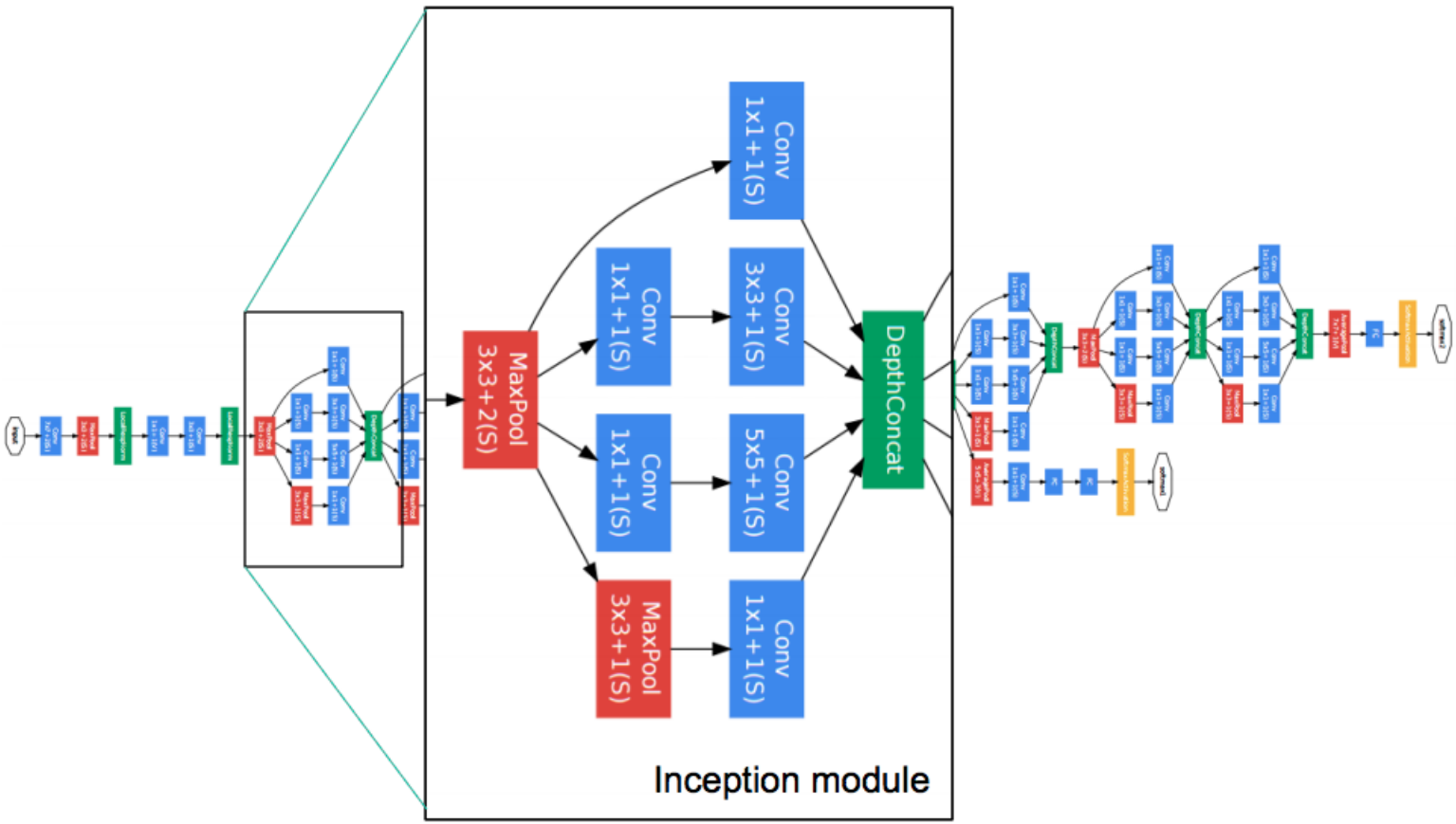
ConvNet : VGG-16 (2014)

- +/- 138M parameters (19 layers)
- Train on 4 GPUs for 2-3 weeks
- Key contributions : Uniformity

[From : Very Deep Convolutional Networks for Large-scale Image Recognition ; K. Simonyan, A. Zisserman]

ConvNet : Inception Network / GooLeNet (2014)

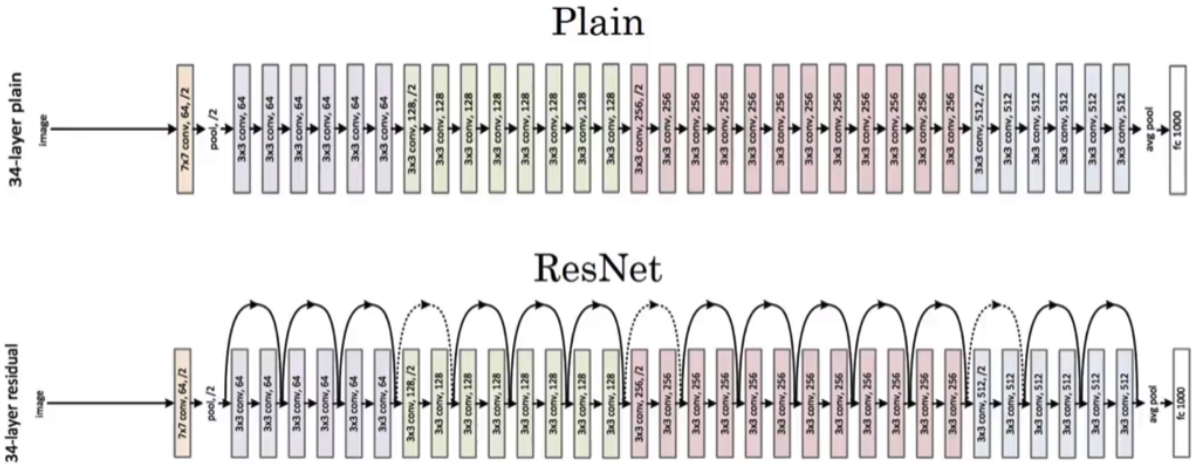
- +/- 25M parameters (22 layers)
- Train on 8 GPUs for 2 weeks
- Key contributions : Network in Network



[From : Going Deeper with Convolutions ; C. Szegedy]

ConvNet : ResNet (2015)

- +/- 60M parameters (152 layers)
- Train on 8 GPUs for 2-3 weeks
- Key contributions : Deep learning, Shortcut connections



[From : Deep Residual Learning for Image Recognition ; K. He, X. Zhang, S. Ren, J. Sun]

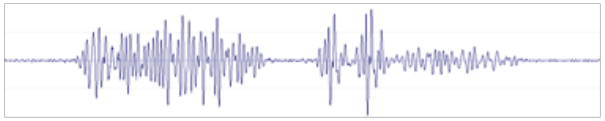
RNN - Recurrent Neural Network

RNN - Recurrent Neural Network

cat

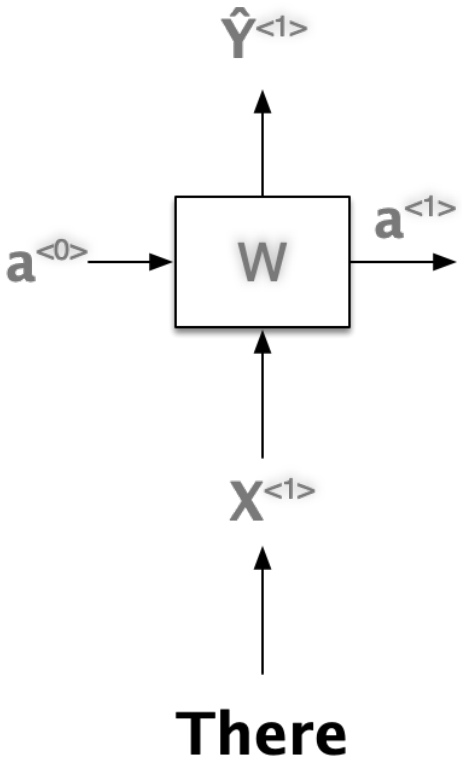
"Voulez-vous chanter avec moi?" → *"Do you want to sing with me?"*

"There is nothing to like in this movie" →



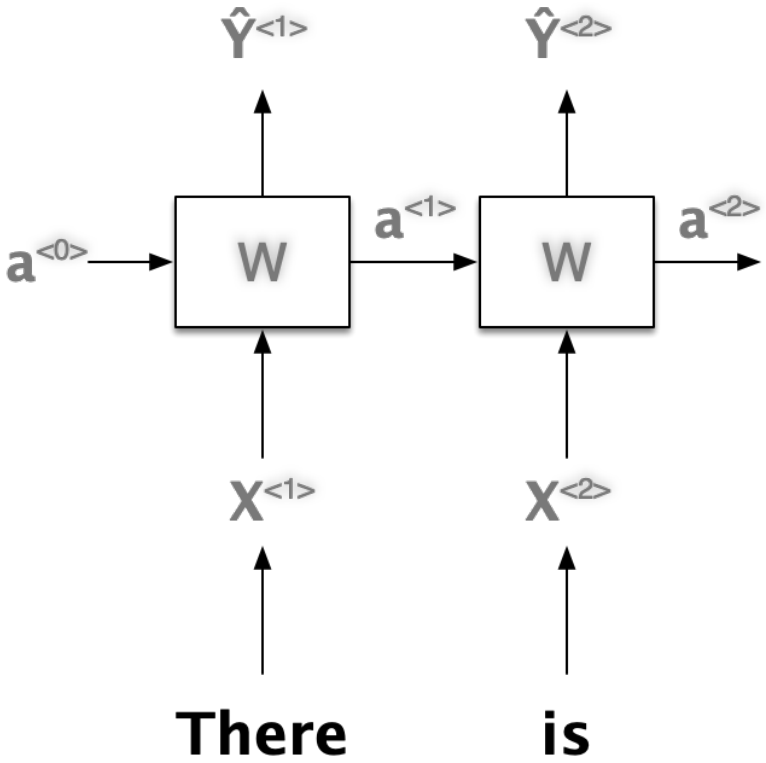
→ *"The quick brown fox jumped... "*

RNN - Recurrent Neural Network



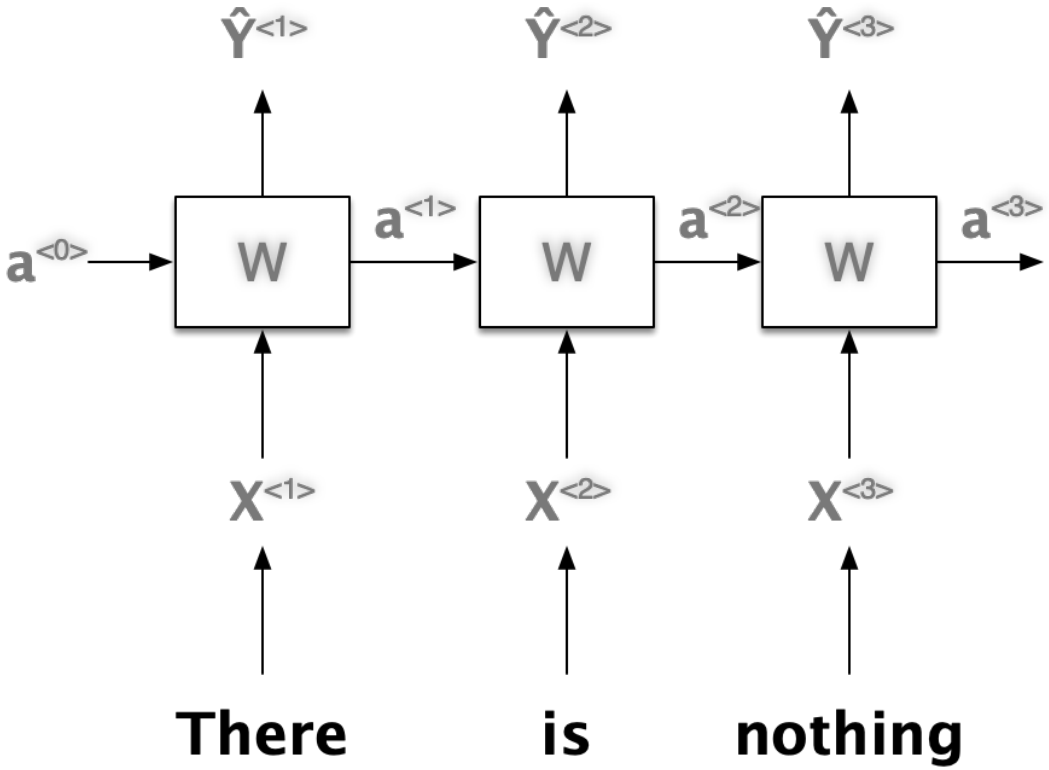
"There is nothing to like in this movie"

RNN - Recurrent Neural Network



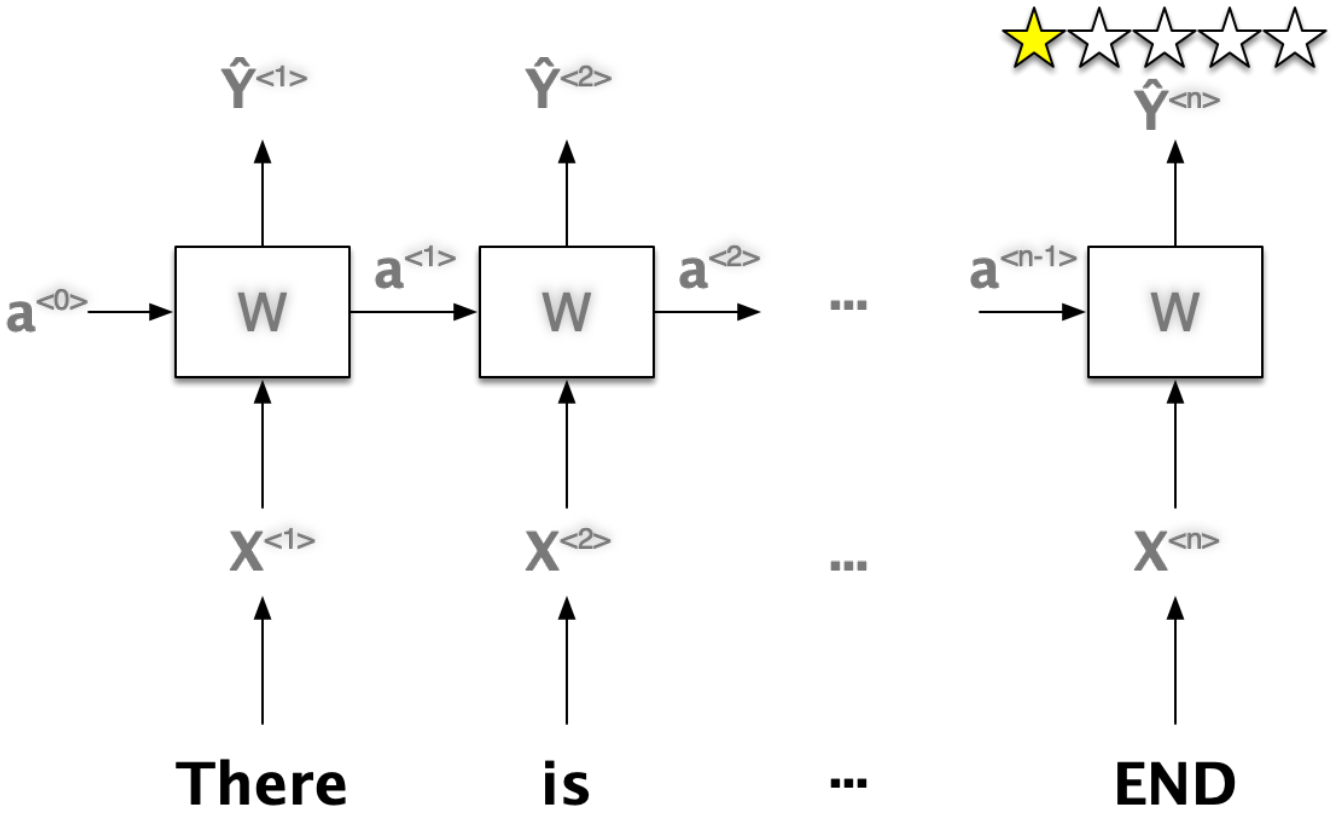
"There is nothing to like in this movie"

RNN - Recurrent Neural Network



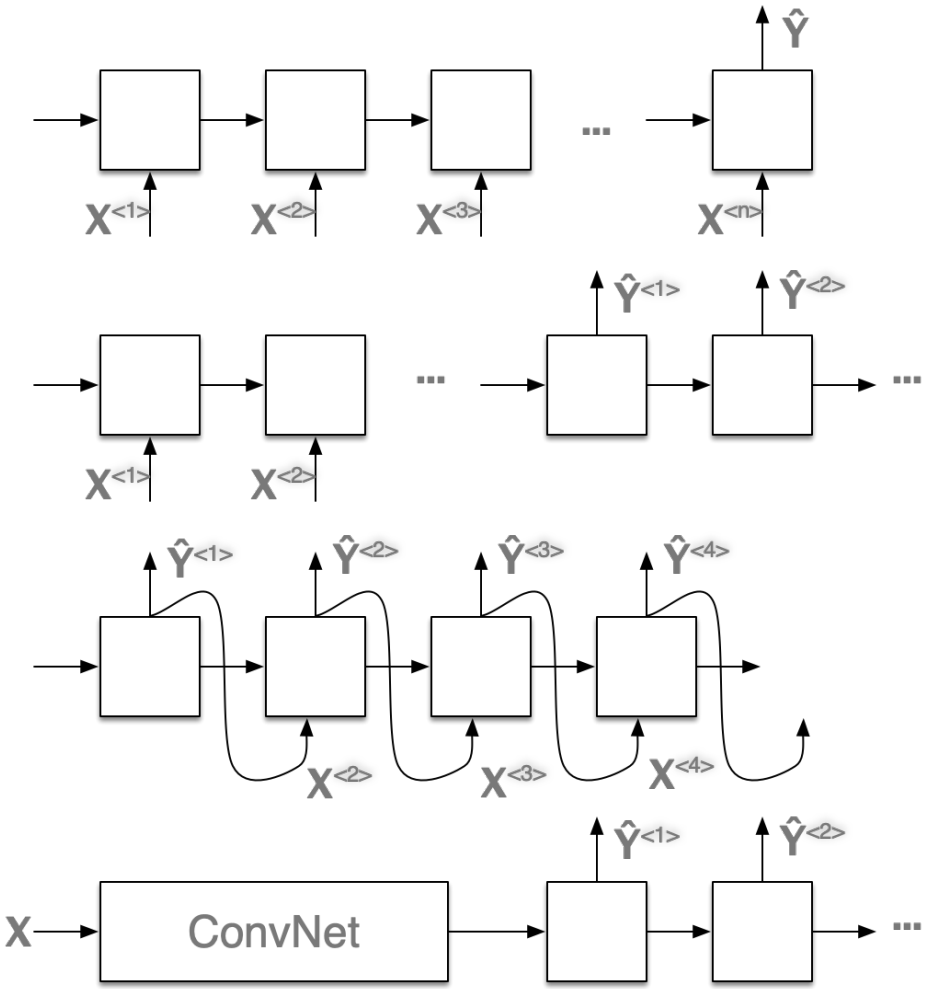
"There is nothing to like in this movie"

RNN - Recurrent Neural Network



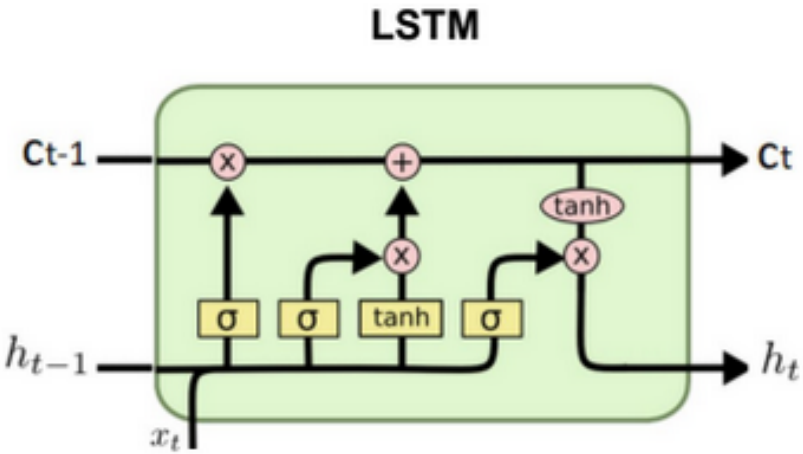
"There is nothing to like in this movie"

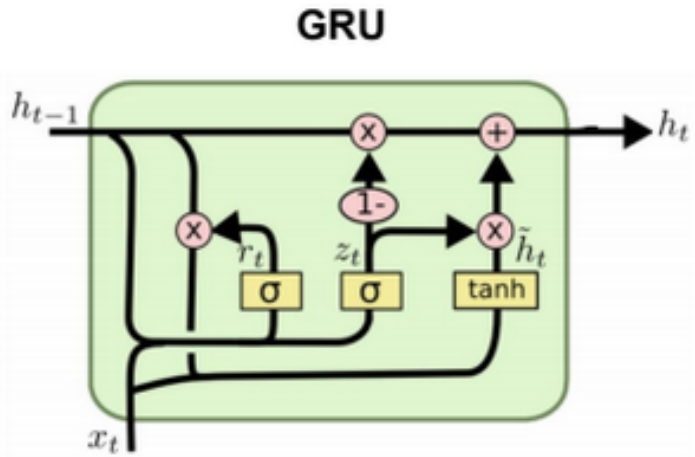
RNN - Recurrent Neural Network



RNN - Recurrent Neural Network

- Vanishing gradient
  - LSTM (Long Short Term Memory)
  - GRU (Gated Recurent Unit)



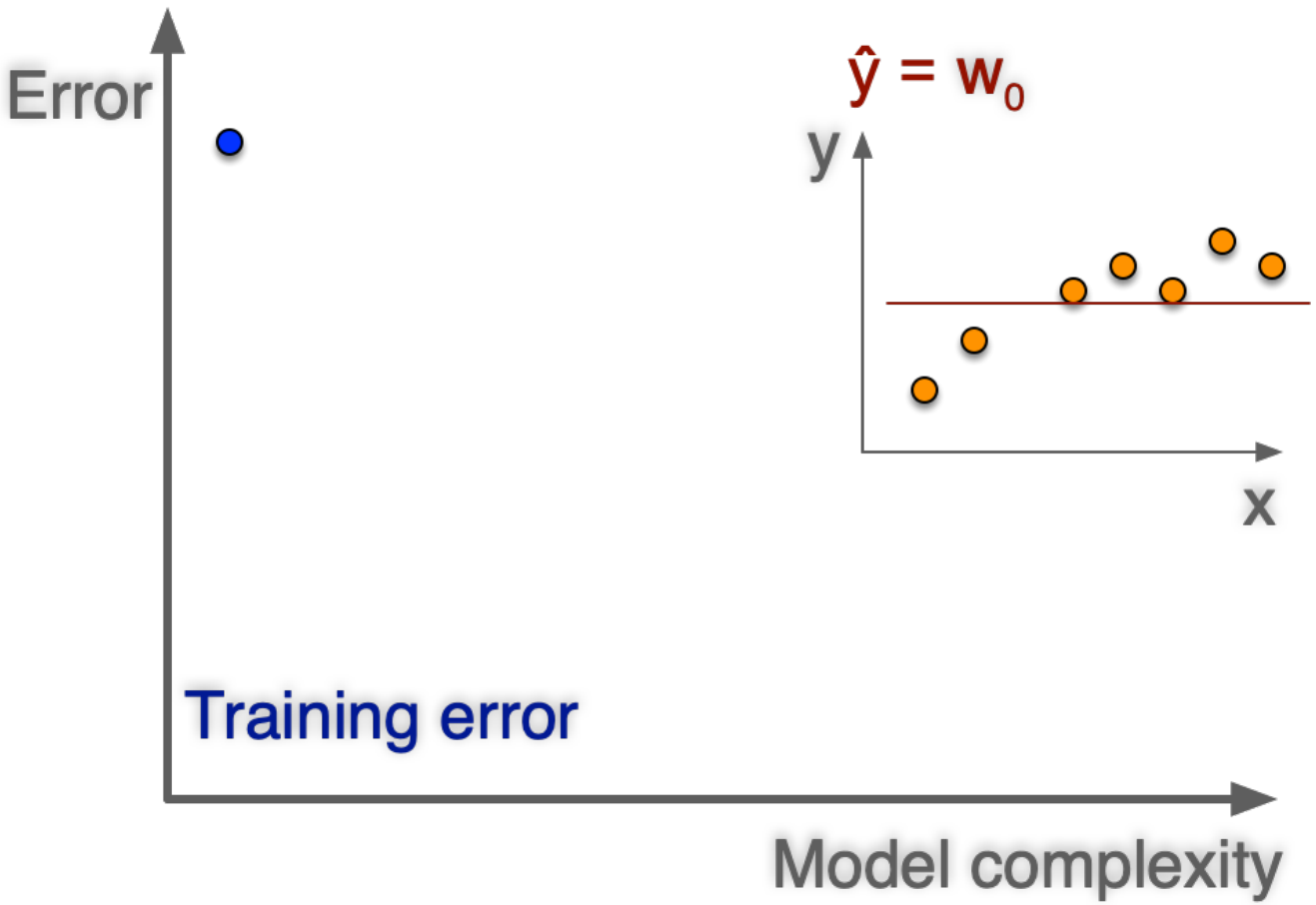


Outline

- Part I
  - Linear Regression
  - Logistic Regression
  - Neural Network
  - Convolutional Neural Network (CNN or ConvNet)
  - Recurrent Neural Network (RNN)
- Part II : Other aspects
  - Evaluation
  - Regularization
  - Transfer learning
  - Gradient-Descent optimizations
  - Software and hardware infrastructure
- Conclusion

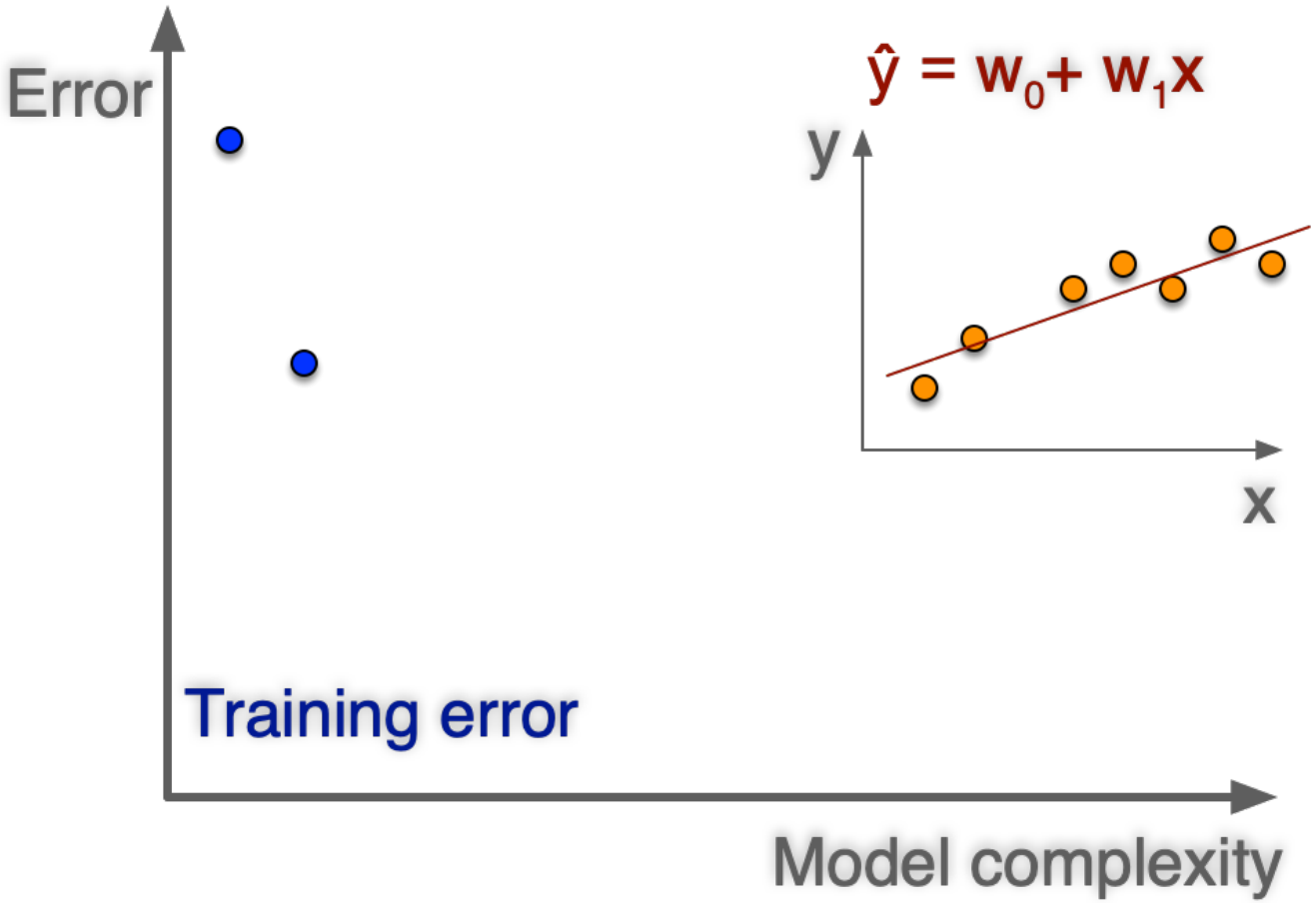
Evaluation

Training Error



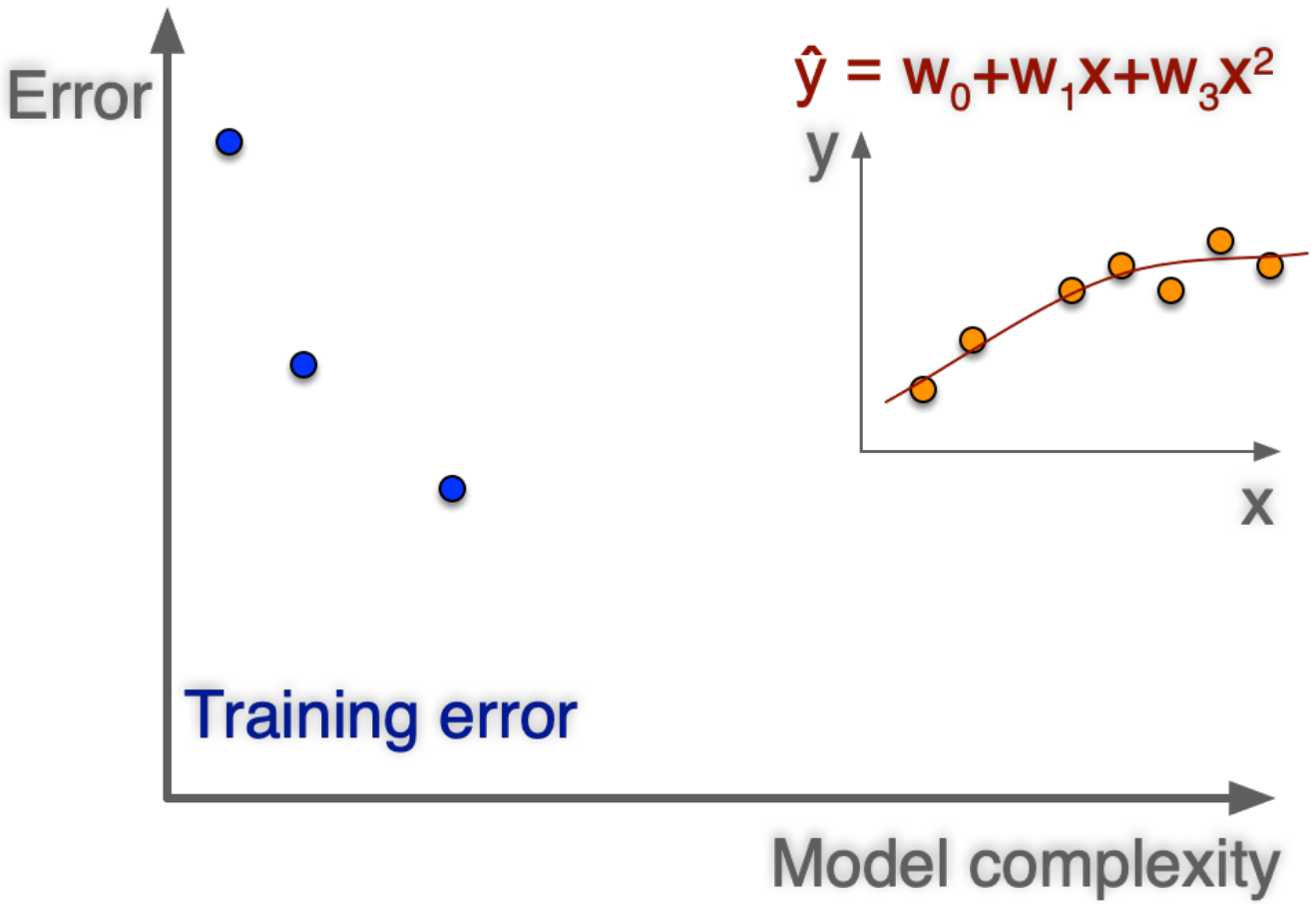
Evaluation

Training Error



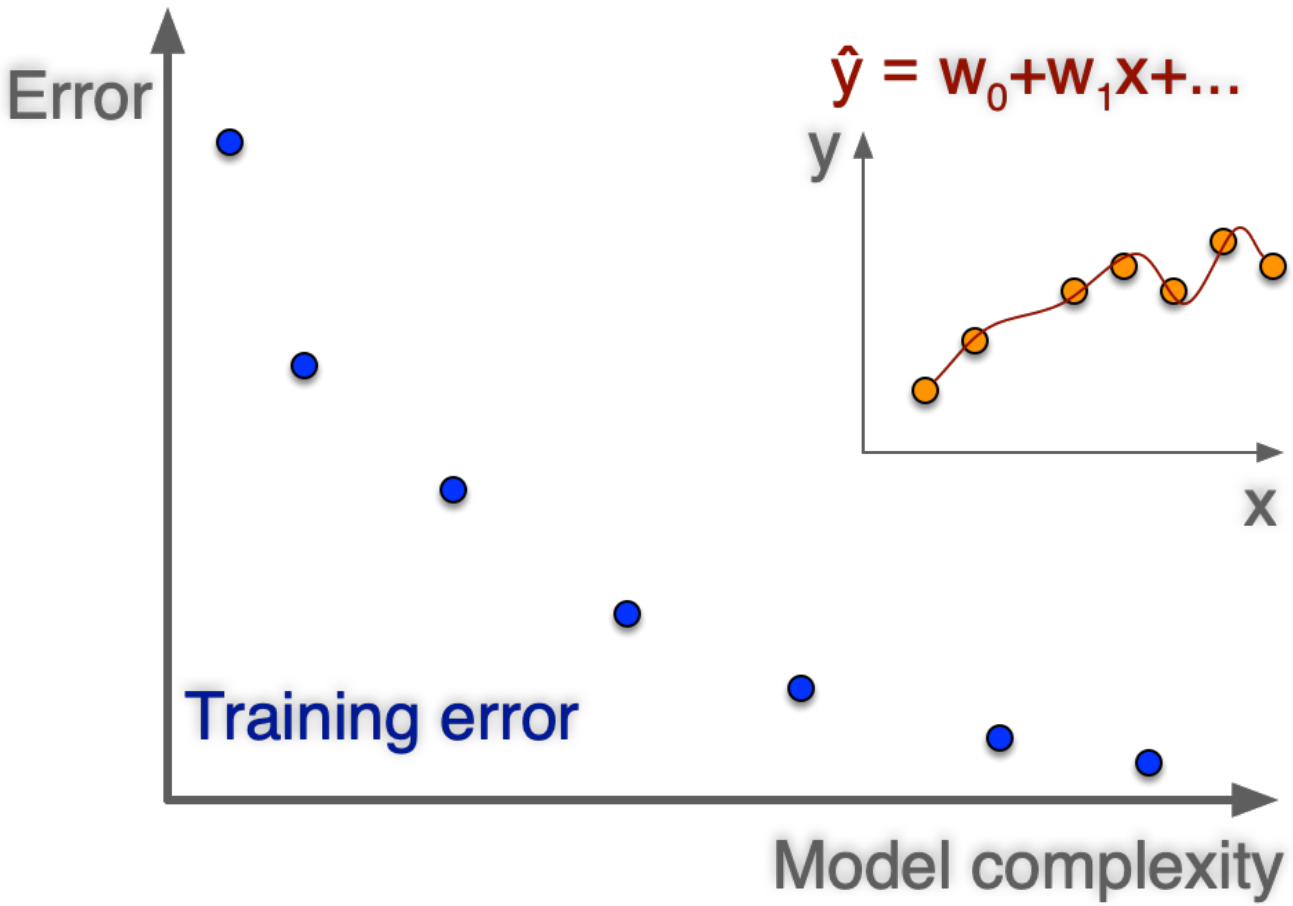
Evaluation

Training Error



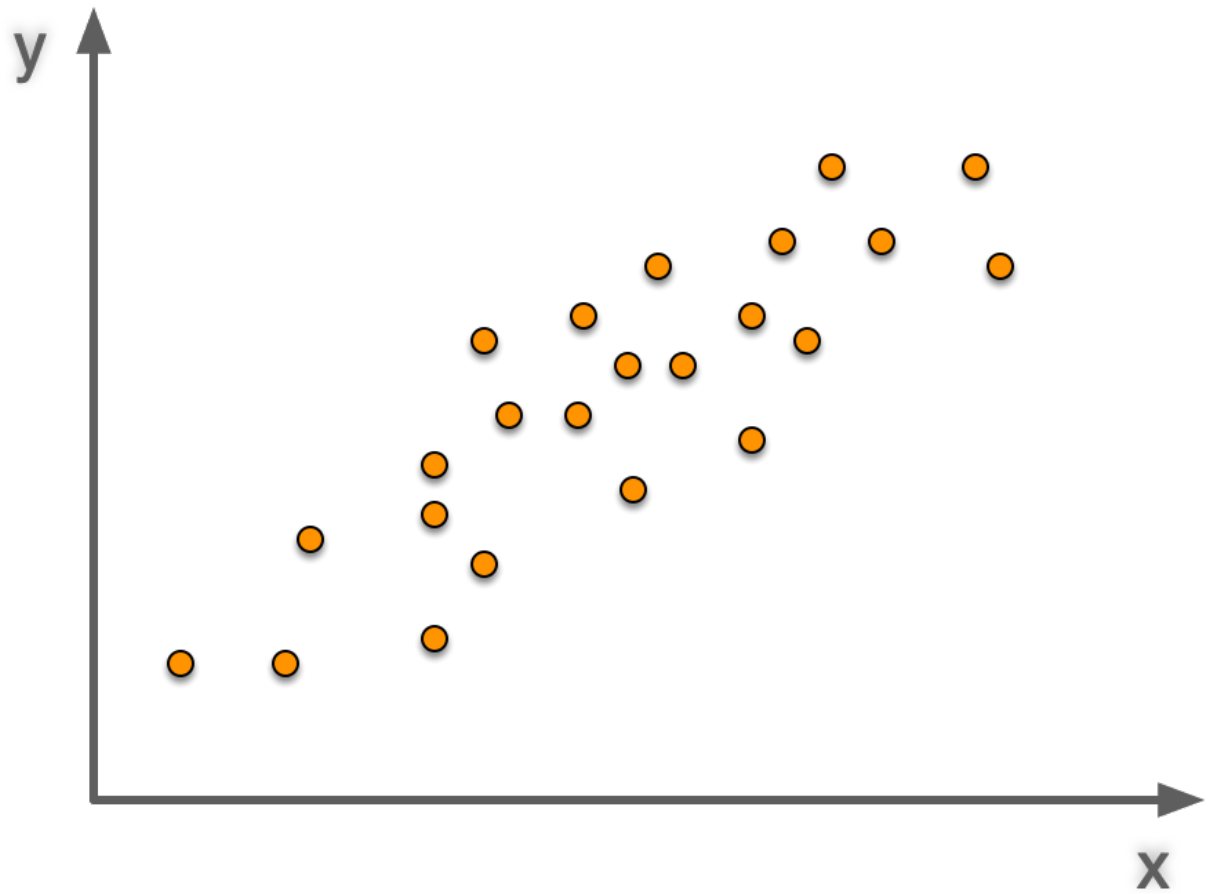
Evaluation

Training Error



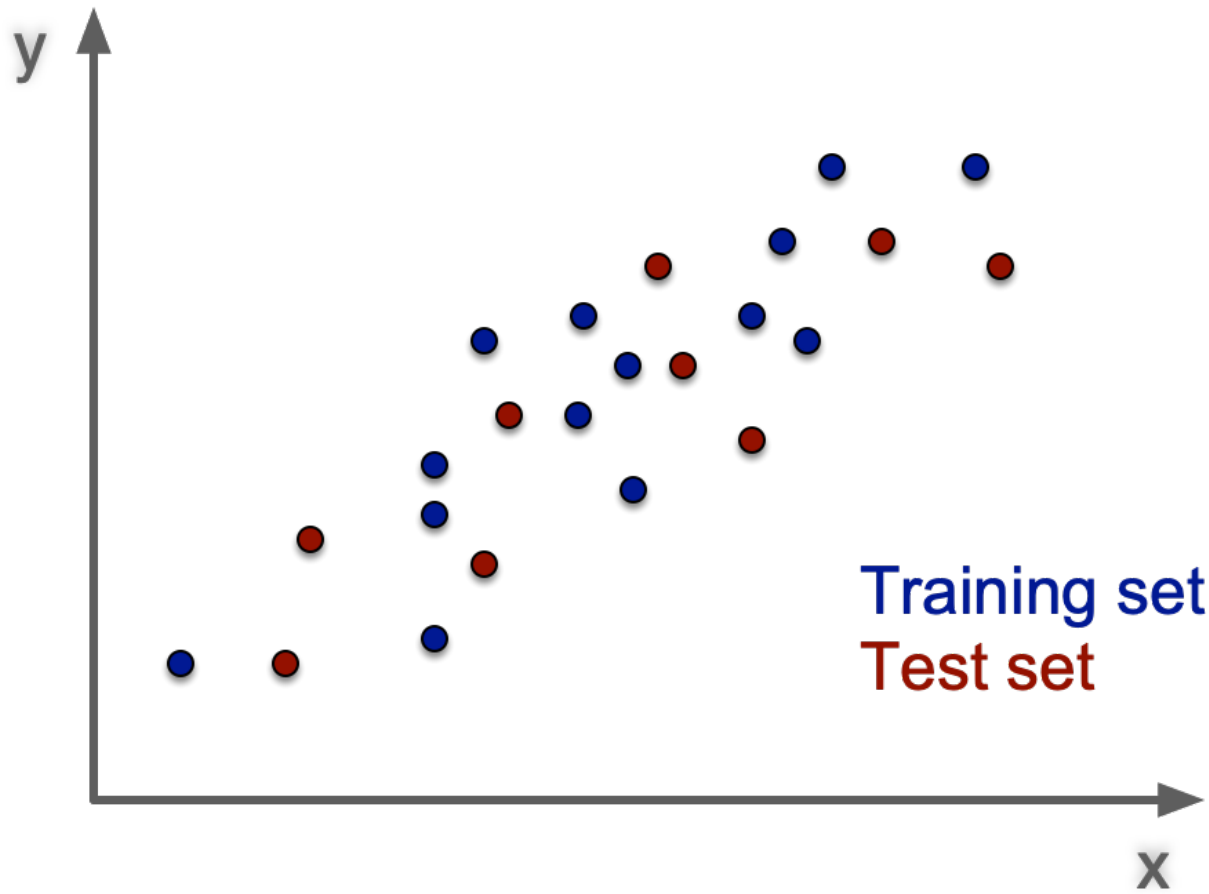
Evaluation

Forming a Test set



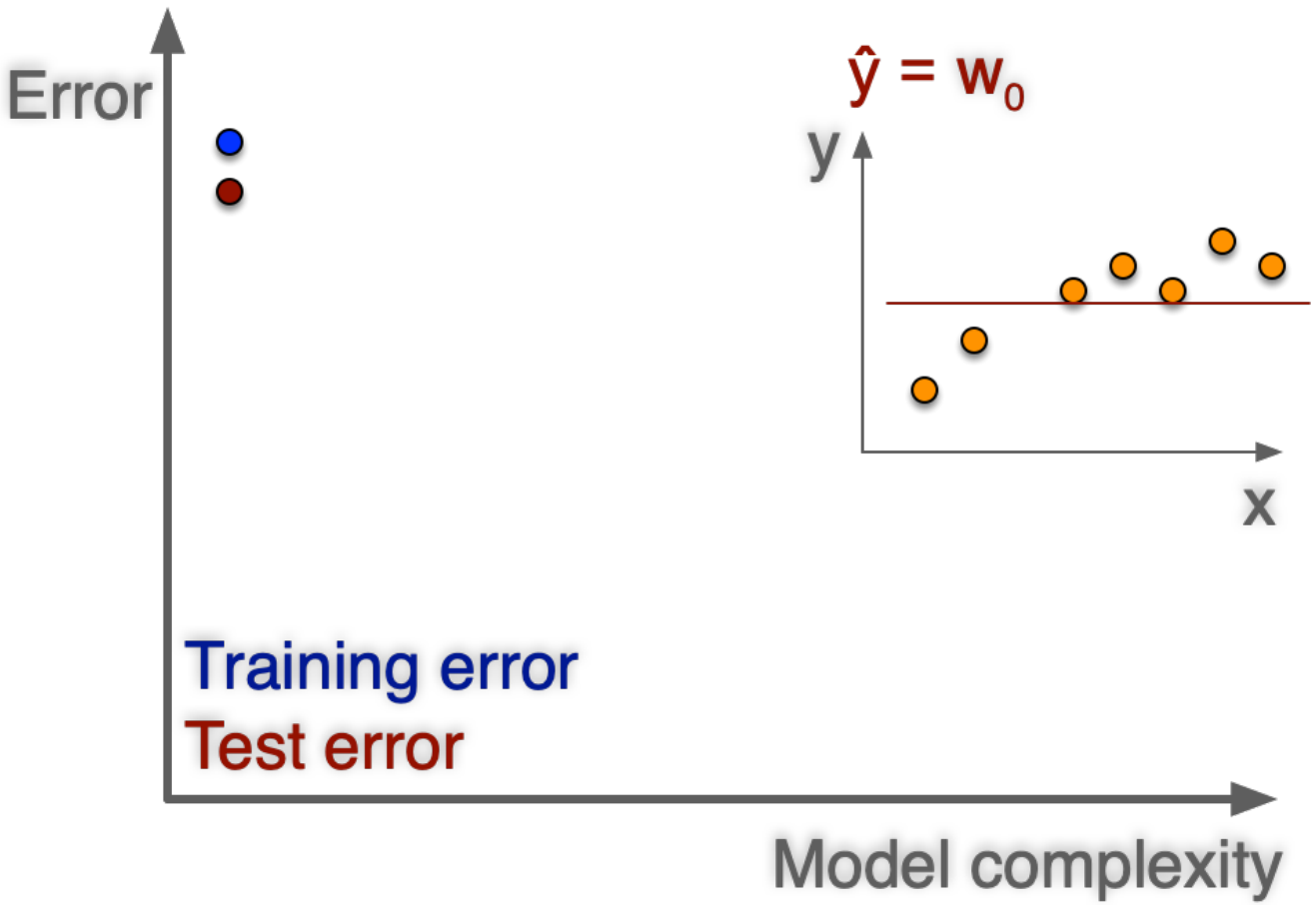
Evaluation

Forming a Test set



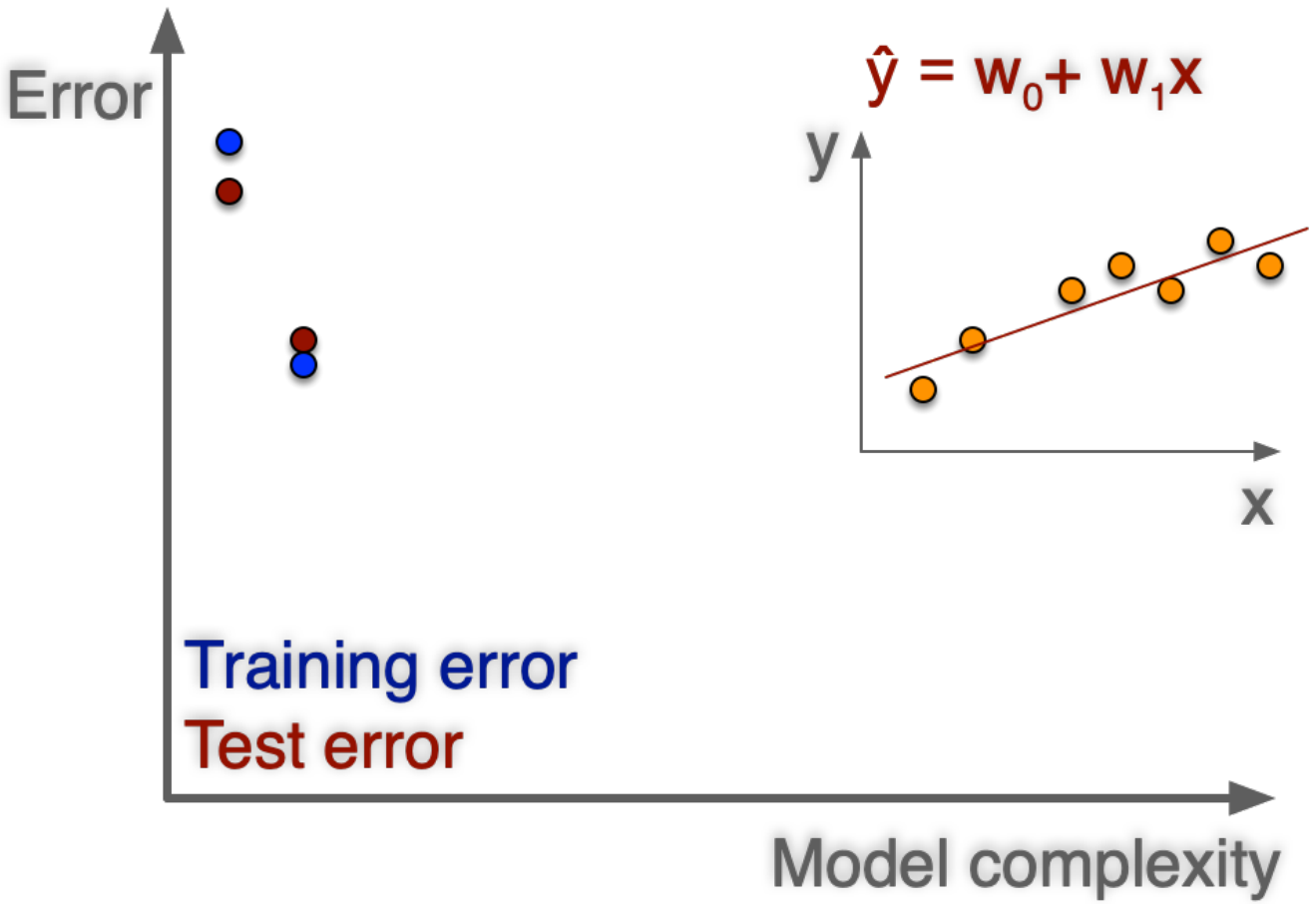
Evaluation

Training error versus Test error



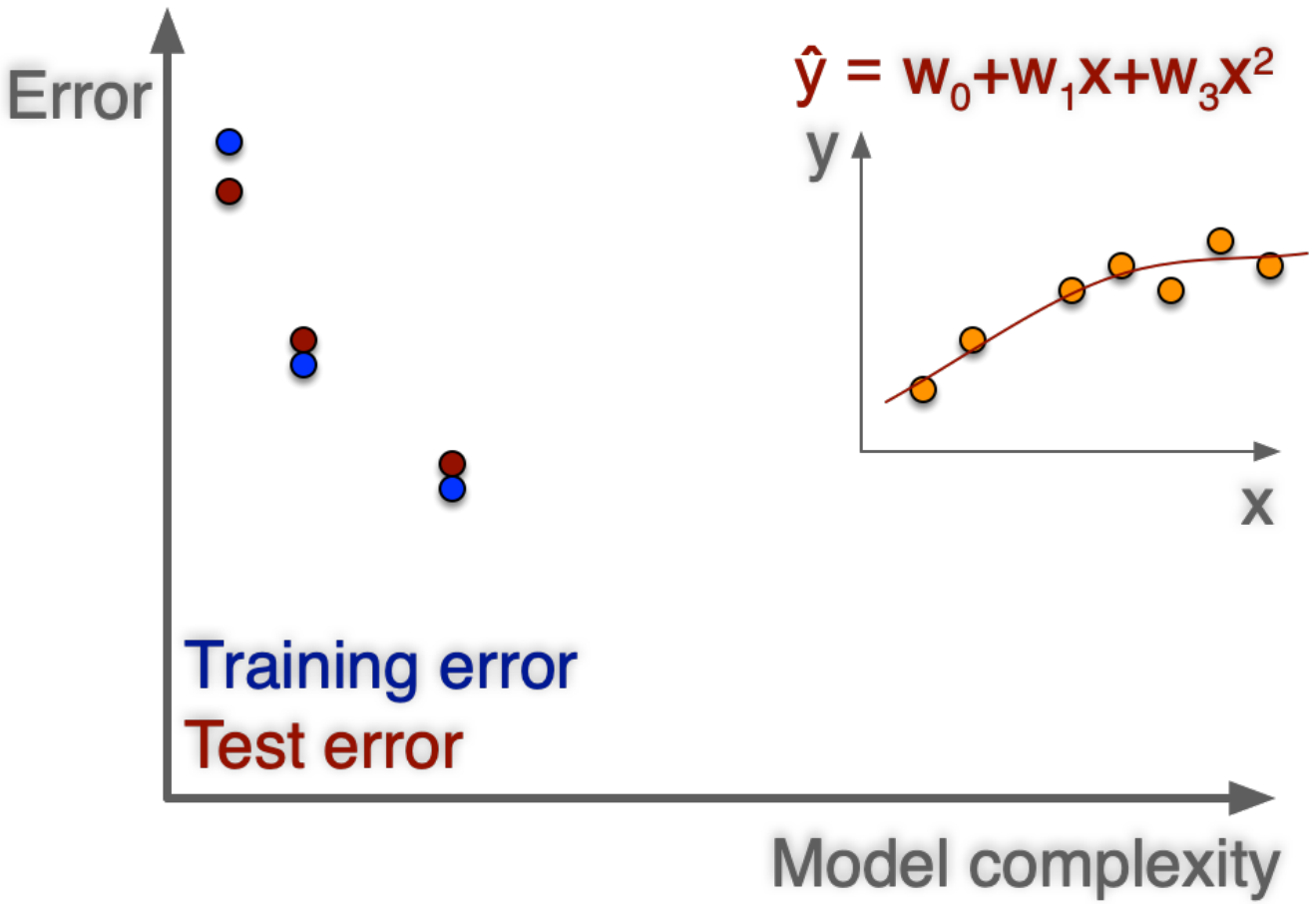
Evaluation

Training error versus Test error



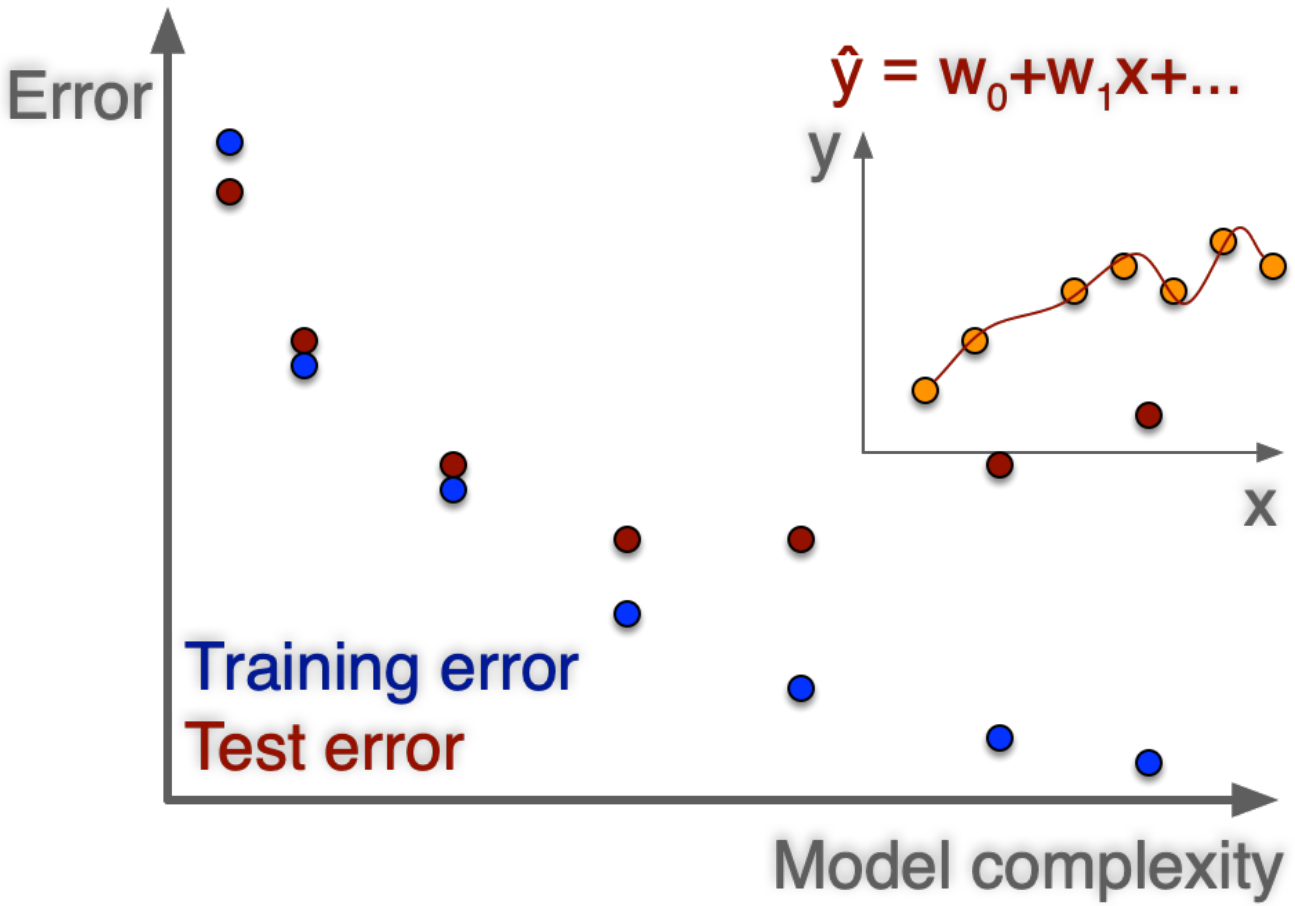
Evaluation

Training error versus Test error



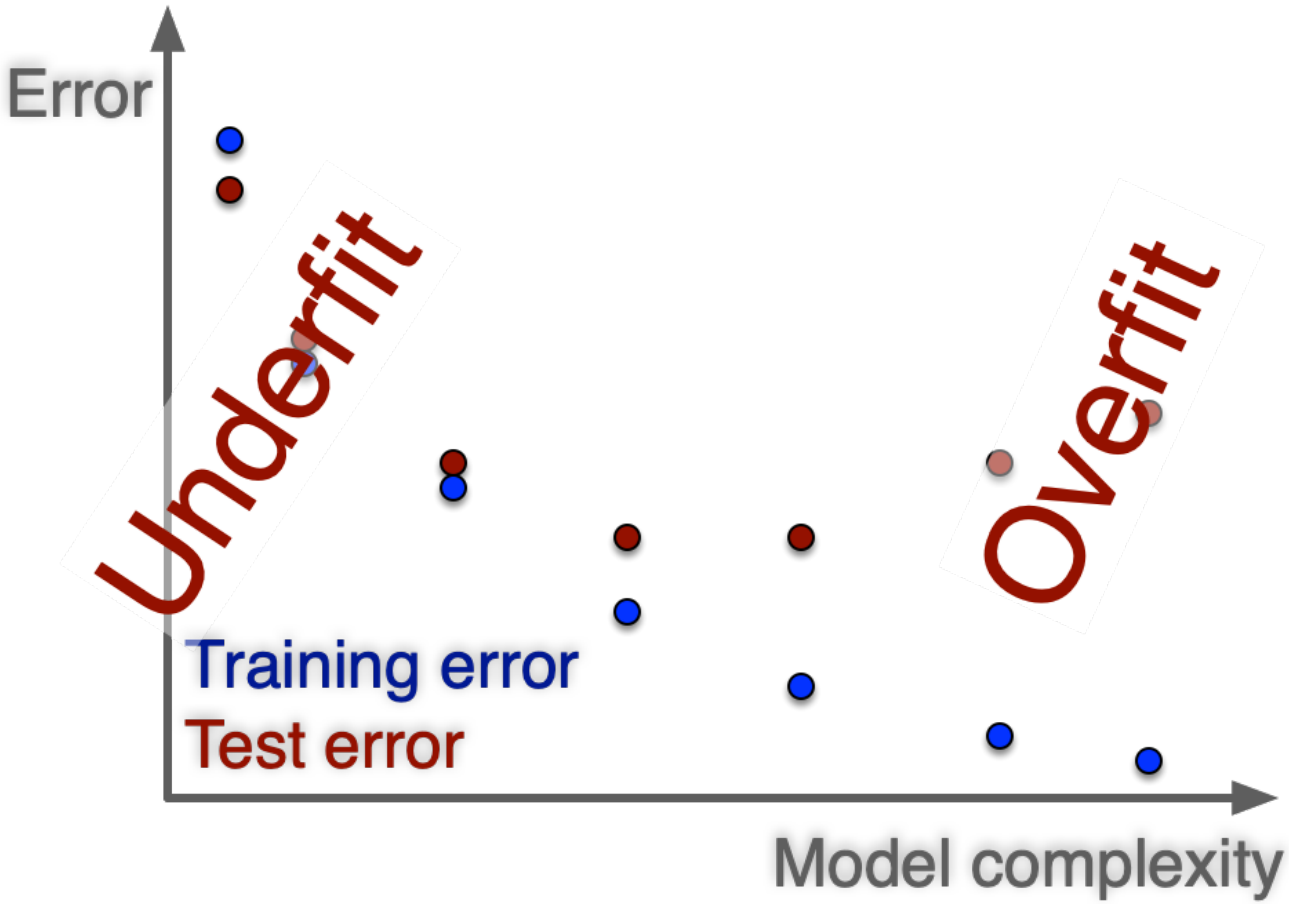
Evaluation

Training error versus Test error



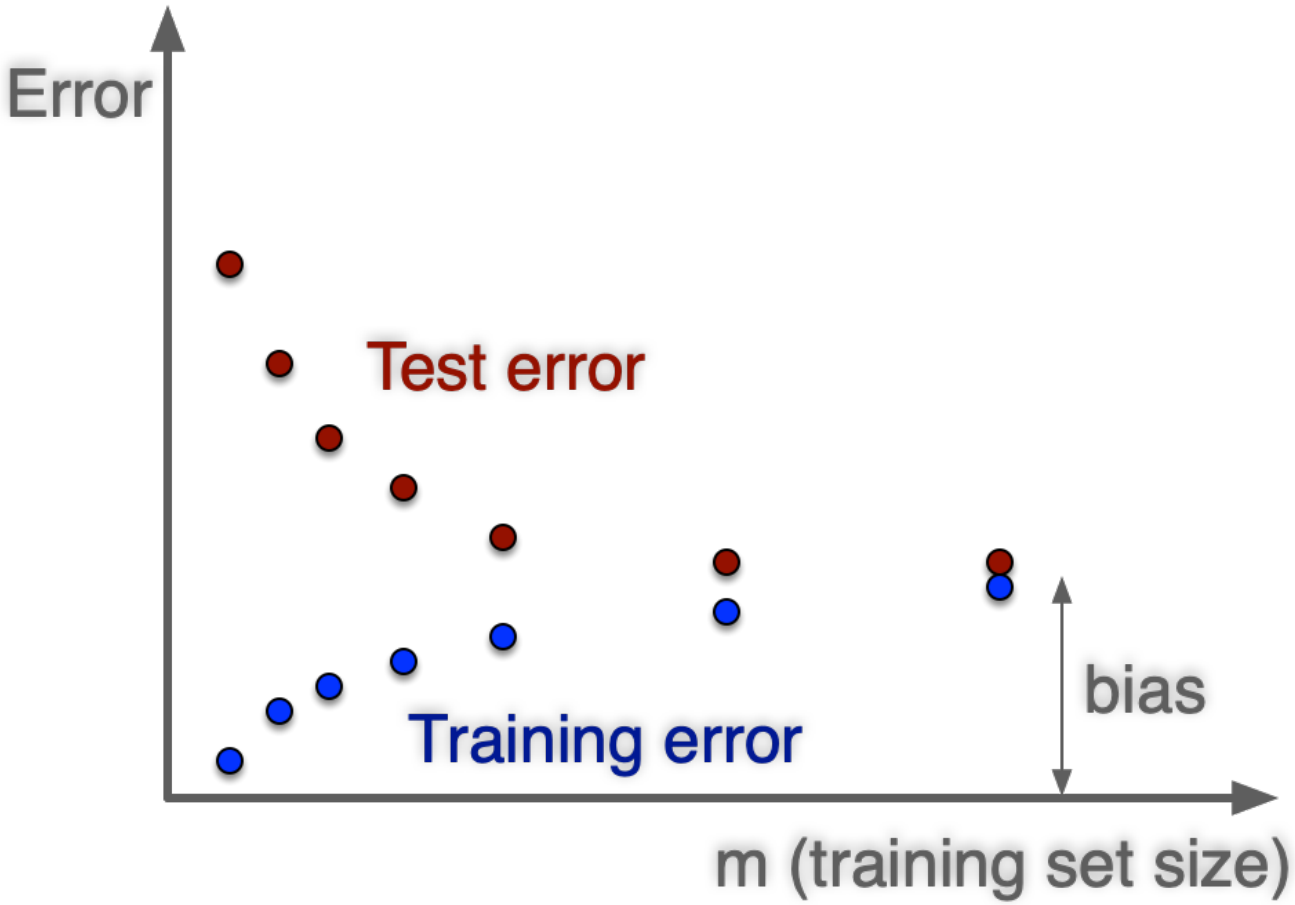
Evaluation

Training error versus Test error



Evaluation

Learning Curves



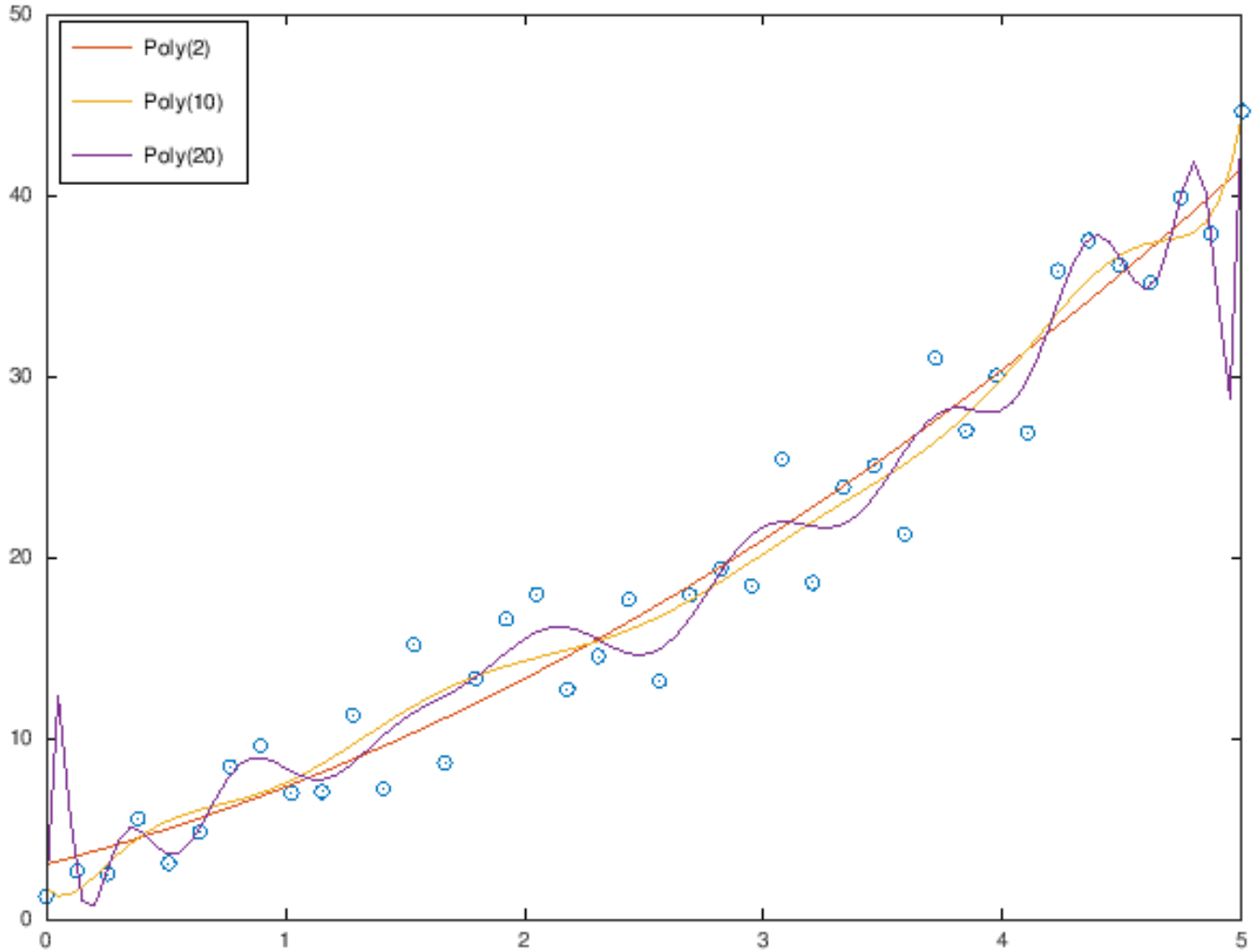
Evaluation

Learning Curves

Regularization

- Parameter Norm Penalties
- Dataset Augmentation
- Dropout

Regularization : Parameter Norm Penalties



Regularization : Parameter Norm Penalties

$$P_2(x) = 0.86x^2 + 3.4x + 3$$

$$P_{10}(x) = 0.024x^{10} - 0.65x^9 + 7.3x^8 - 46x^7 + 174x^6 - 412x^5 + 598x^4 - 502x^3 + 209x^2 - 24x^1 + 3.8$$

$$P_{20}(x) = -0.002x^{20} + 0.14x^{19} - 3.5x^{18} + 53.7x^{17} - 567x^{16} + 4386x^{15} - 25682x^{14} + 116260x^{13} - 411550x^{12} + 1145000x^{11} - 2503000x^{10} + 4277200x^9 - 5655400x^8 + 5693000x^7 - 4260400x^6 + 2290200x^5 - 840530x^4 + 194470x^3 - 24642x^2 + 1244.8x^1 + 3.6973$$

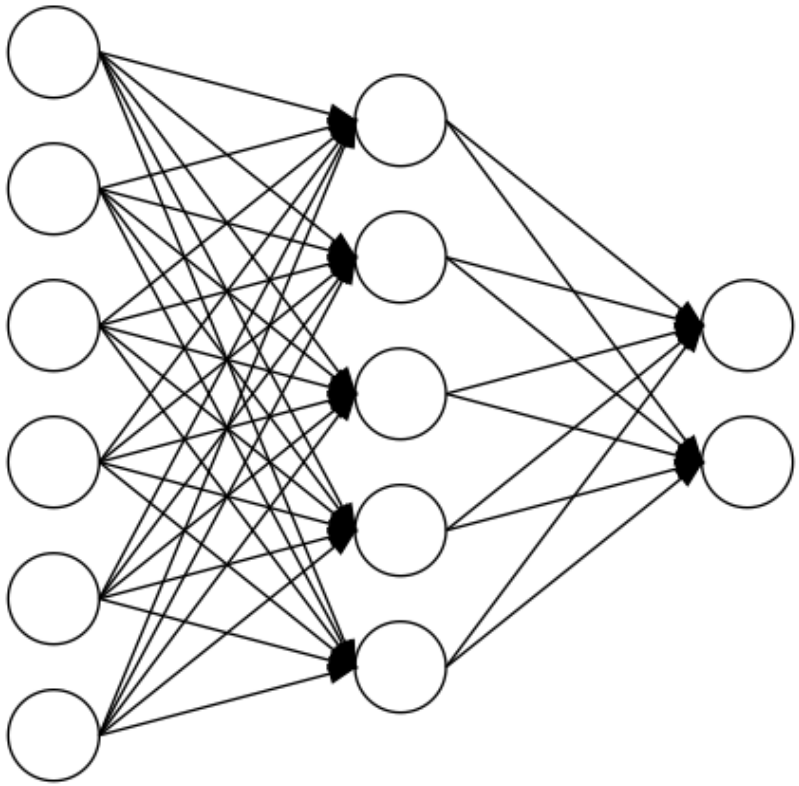
Regularization : Parameter Norm Penalties

$$J(\mathbf{W}) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} * \|\mathbf{W}\| \tag{29}$$

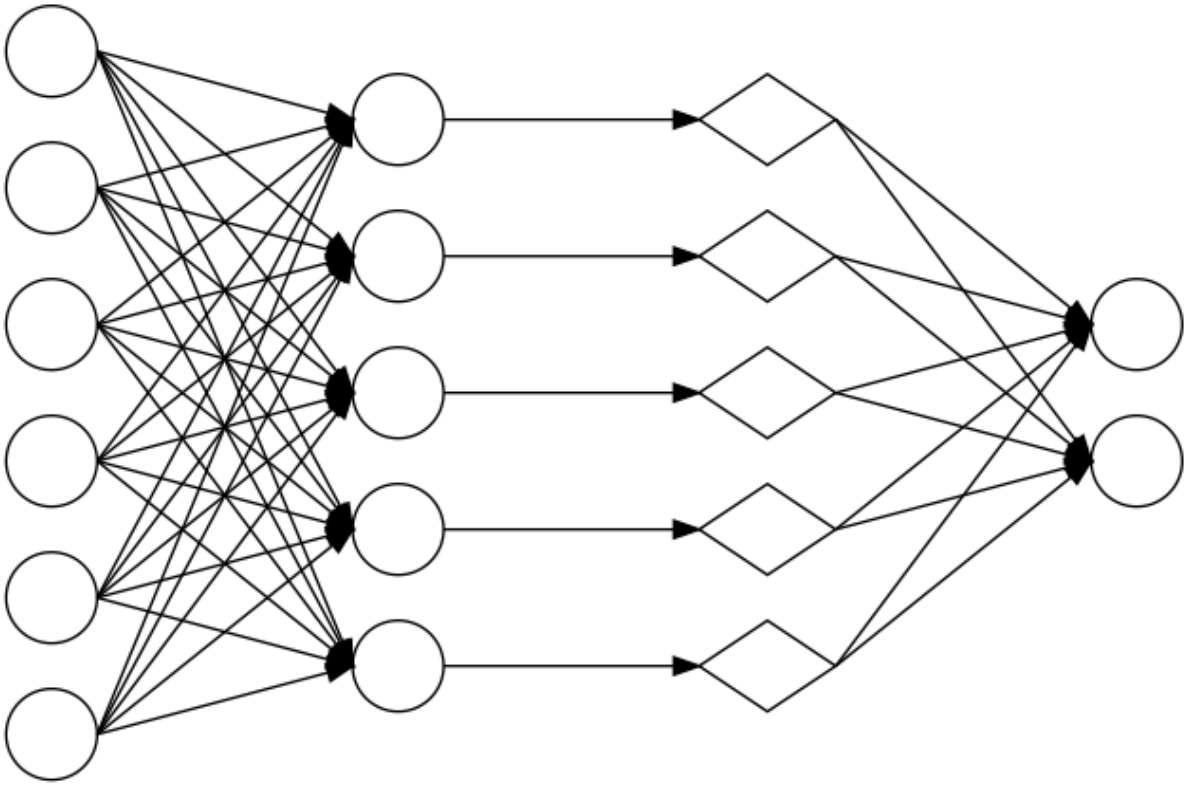
$$Ridge \Rightarrow \|\mathbf{W}\|_2 \tag{30}$$

$$Lasso \Rightarrow \|\mathbf{W}\|_1 \tag{31}$$

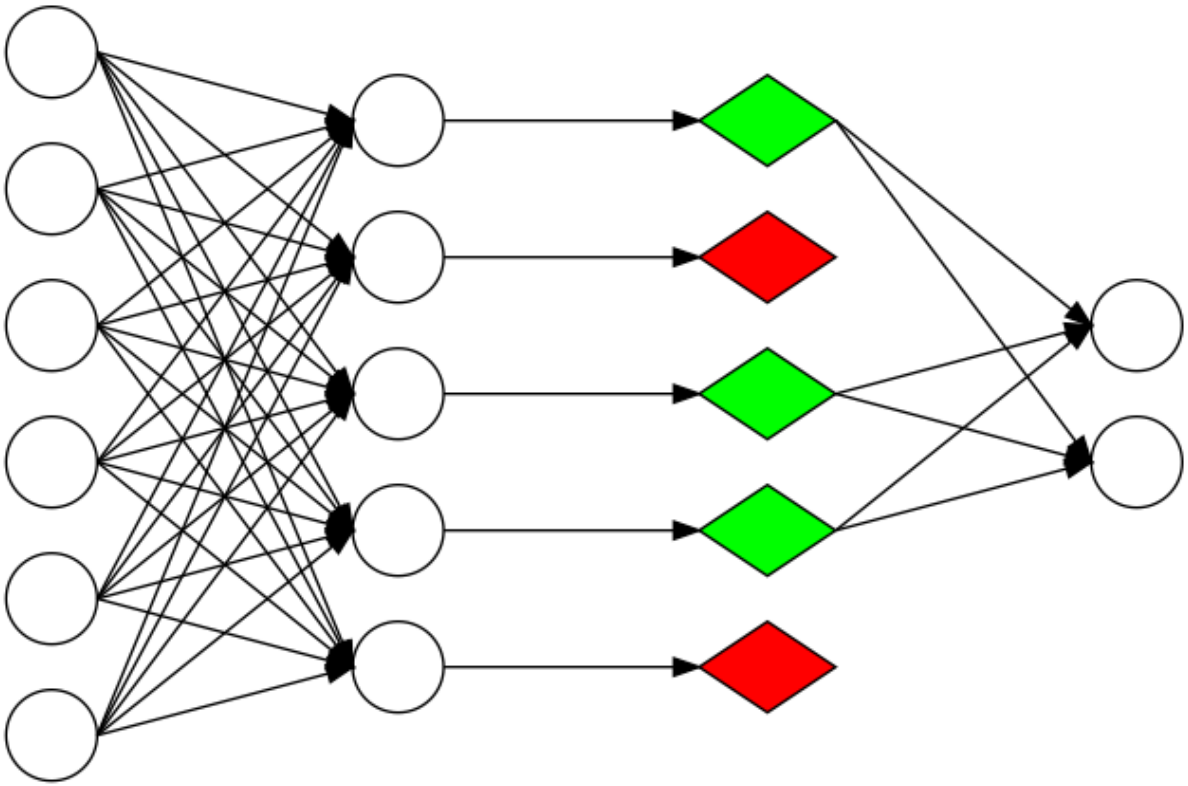
Regularization : Dropout



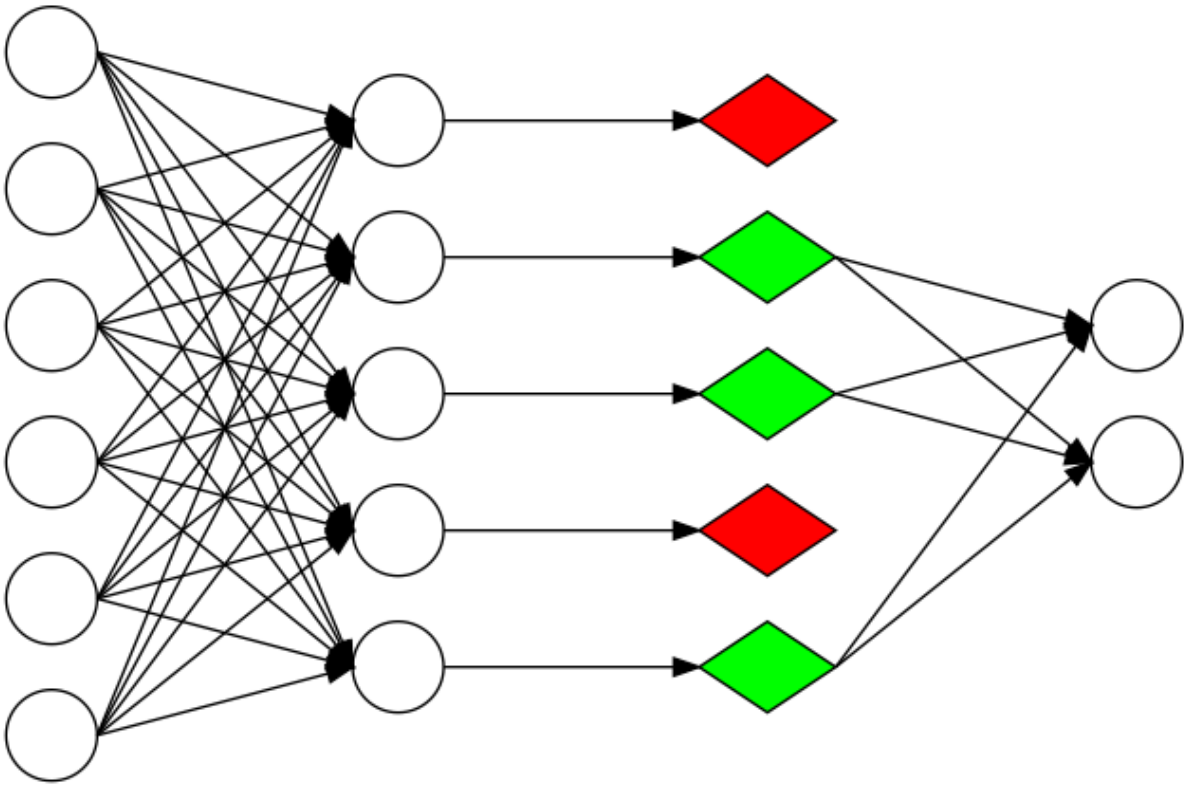
Regularization : Dropout



Regularization : Dropout

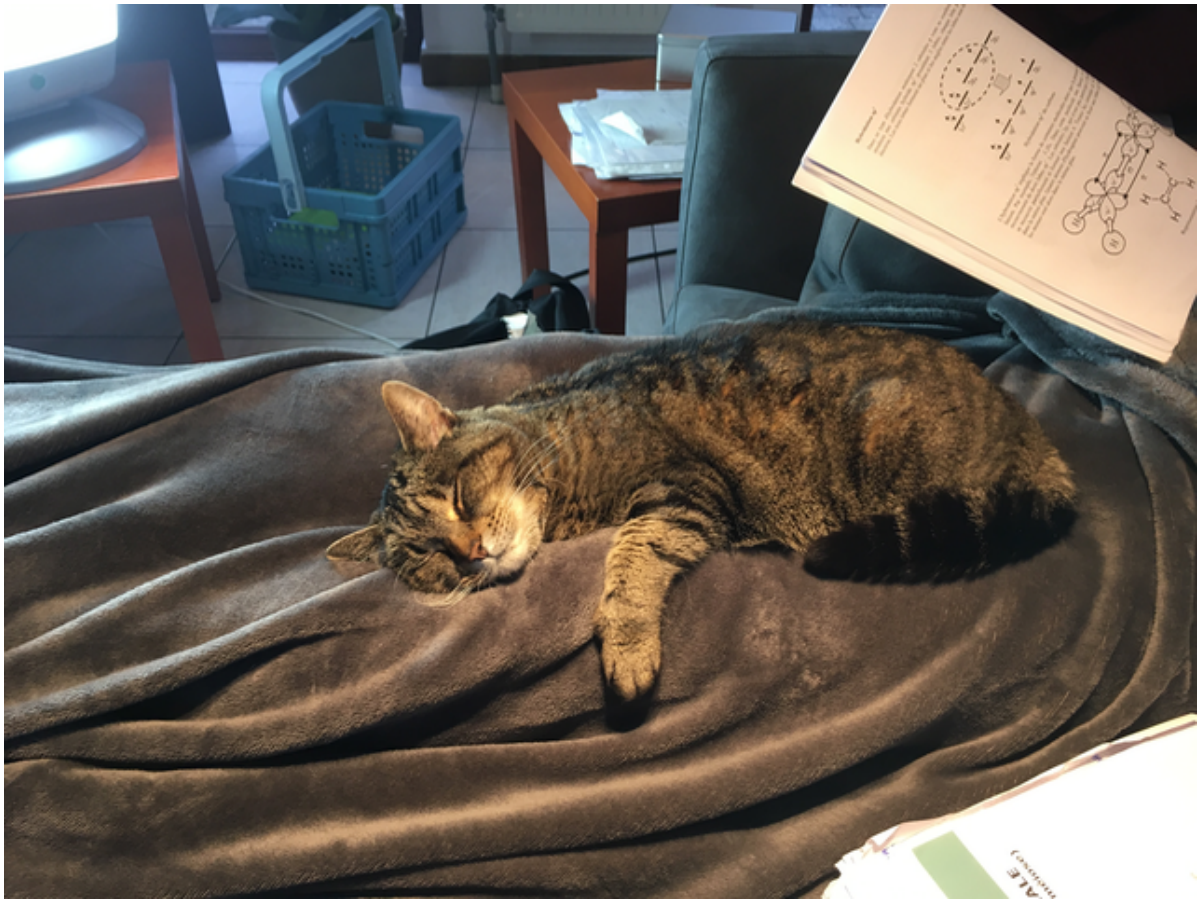


Regularization : Dropout



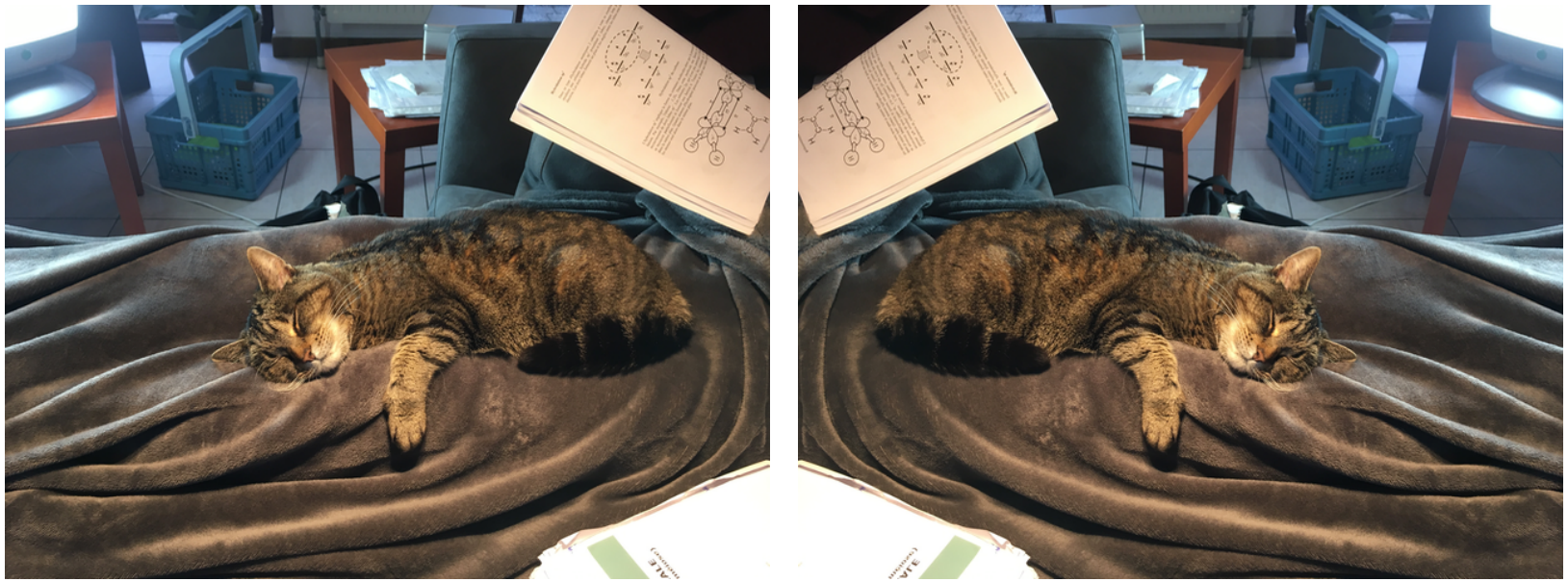
Regularization : Dataset Augmentation

Label : Cat



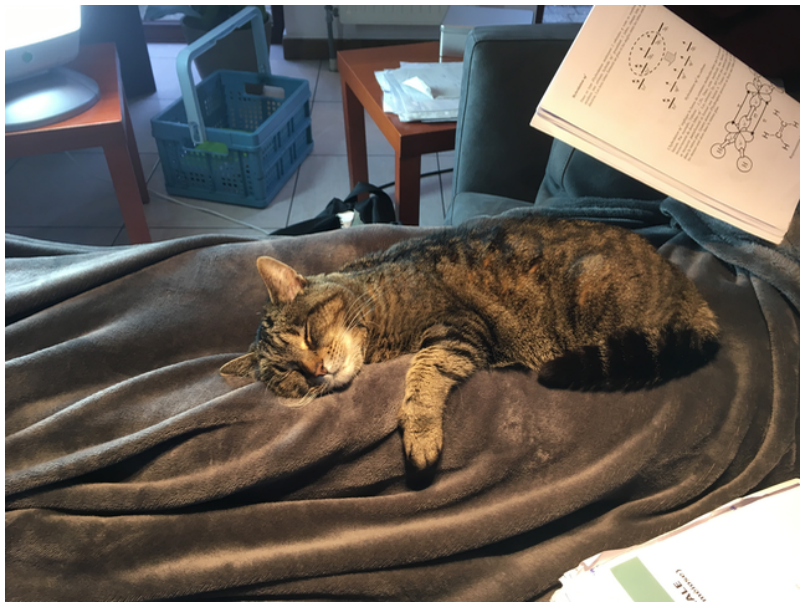
Regularization : Dataset Augmentation

Label : Cat



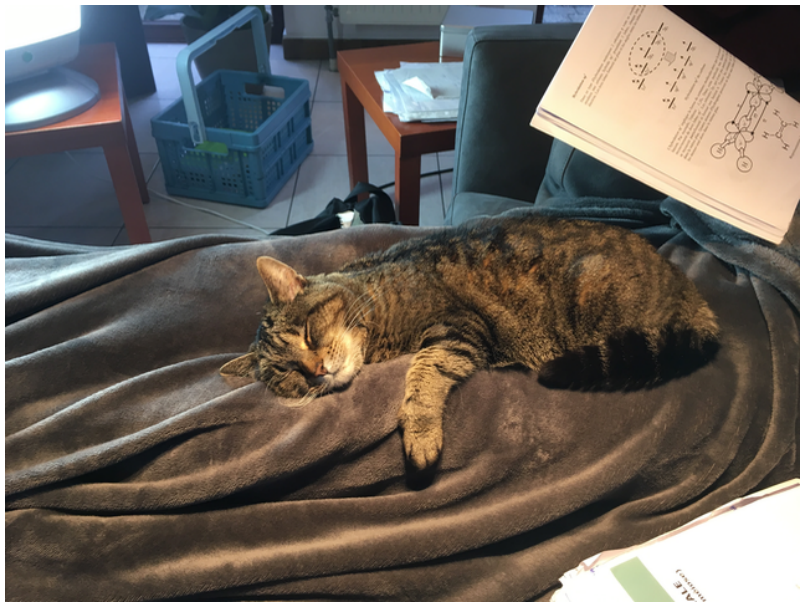
Regularization : Dataset Augmentation

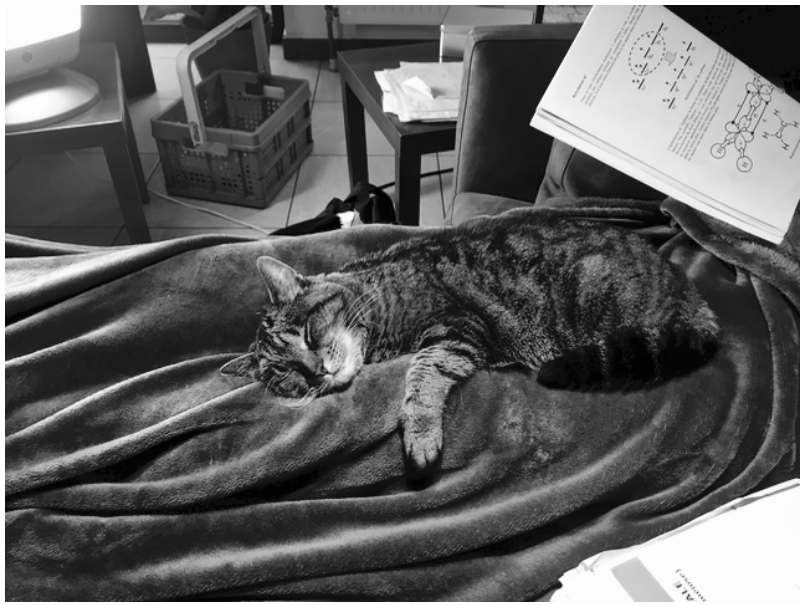
Label : Cat



Regularization : Dataset Augmentation

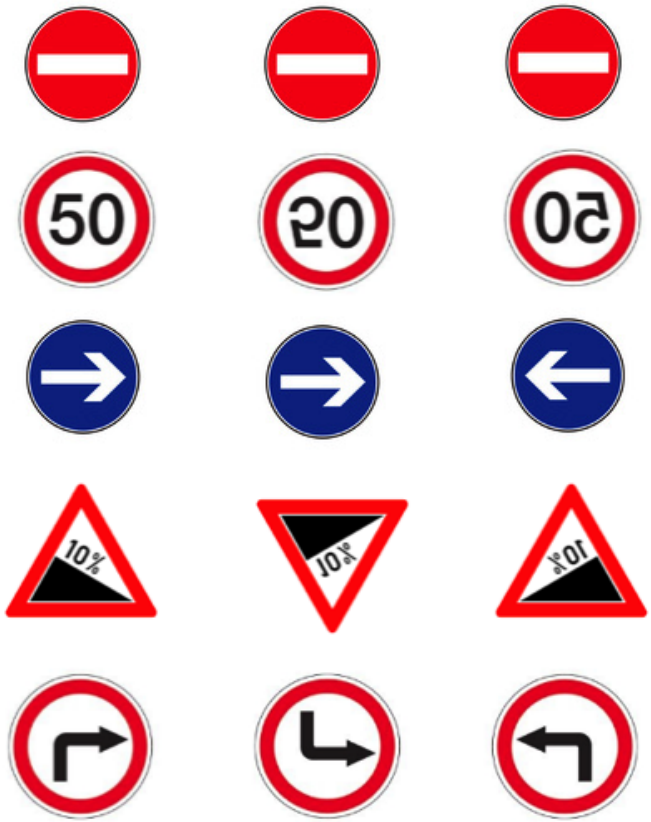
Label : Cat





Regularization : Dataset Augmentation

But ...



Gradient-Descent Optimizations

- No convex (local minima)
- Loss and Gradient

$$J(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} j(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) \tag{32}$$

$$\nabla J(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial j(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})}{\partial \mathbf{W}} \tag{33}$$

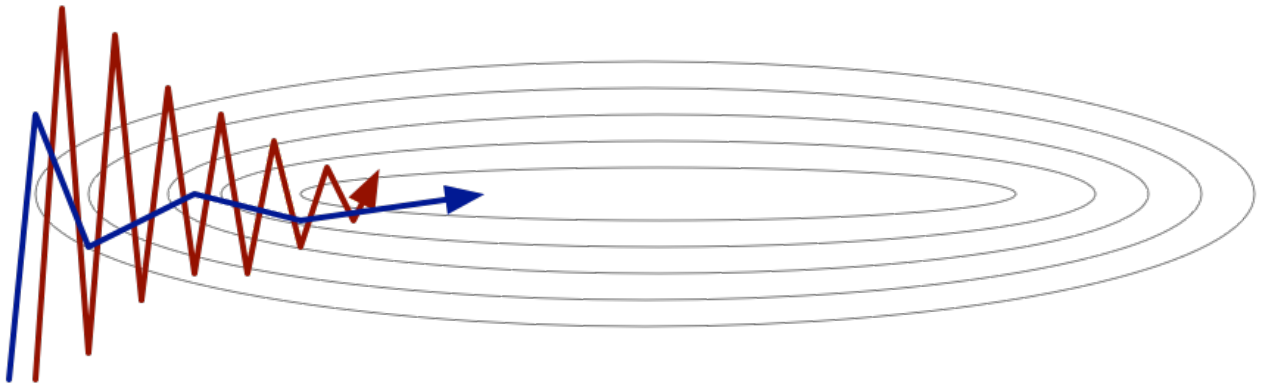
- Gradient approximation
  - N=m $\Rightarrow$ Full gradient
  - N=1 $\Rightarrow$ Stochastic Gradient Descent (SGD)
  - m>N>1 $\Rightarrow$ mini-batch

Gradient-Descent Optimizations

- Decrease Learning Rate (α) during training

$$\mathbf{W} := \mathbf{W} - \alpha \nabla \mathbf{J} \tag{34}$$

- Gradient Descent Optimization Algorithms : Momentum, Adagrad, RMSProp, Adam, ...

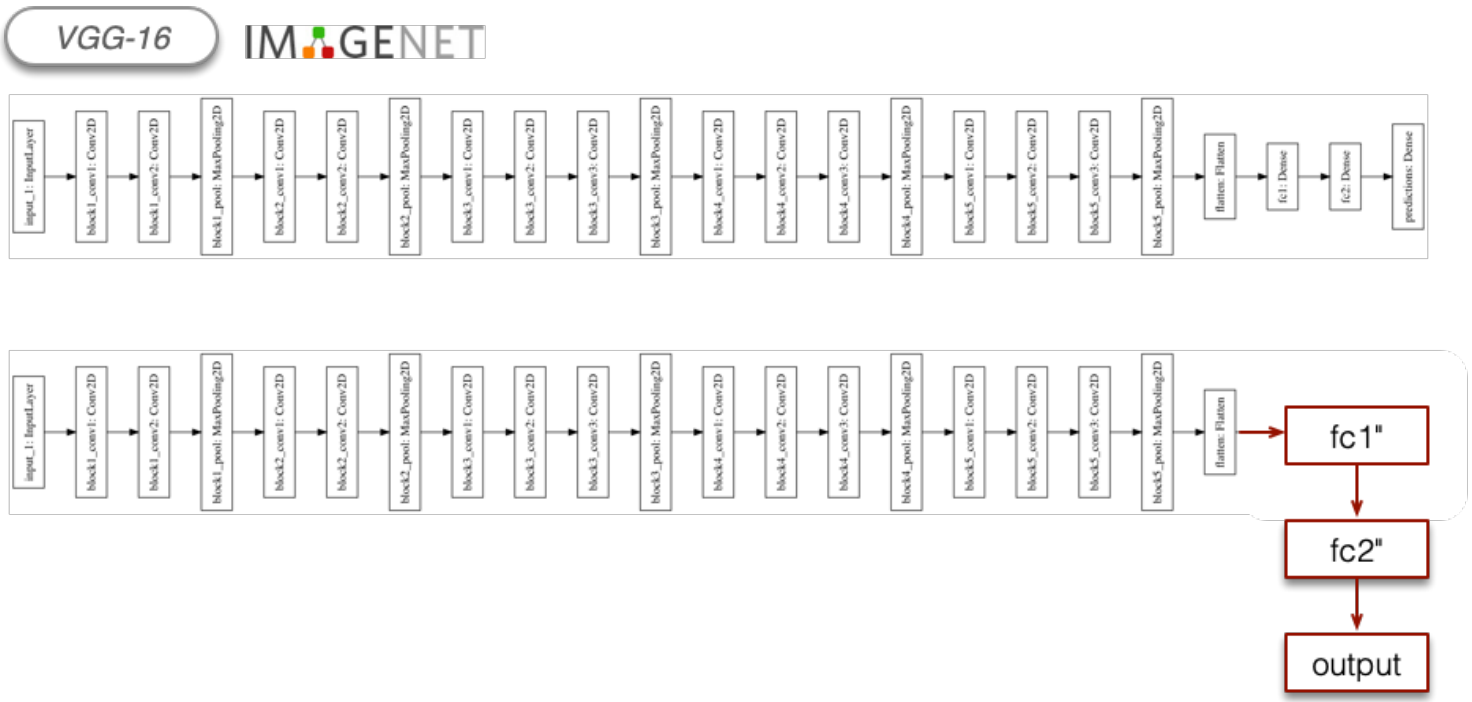


[see http ://ruder.io/optimizing-gradient-descent/]

Transfer Learning

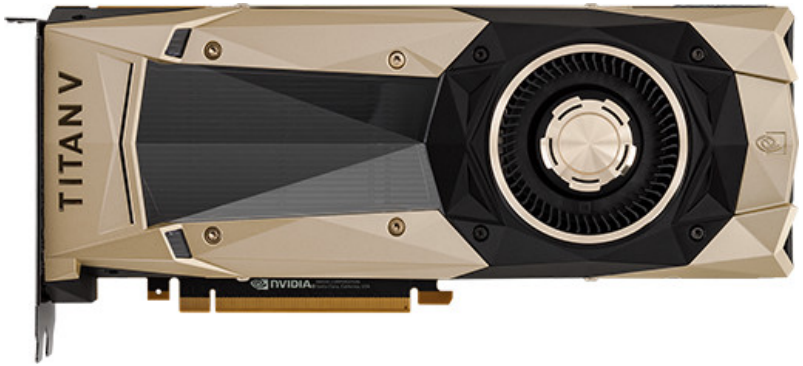
VGG-16 IMAGENET

Transfer Learning



Software and Hardware Infrastructure

- GPU-based : NVIDIA + CUDA



- Solutions :
  - Local solutions with or without GPU
  - Cloud solutions : Google, Amazon, Azure, ... Paperspace, Crestle

Software and Hardware Infrastructure

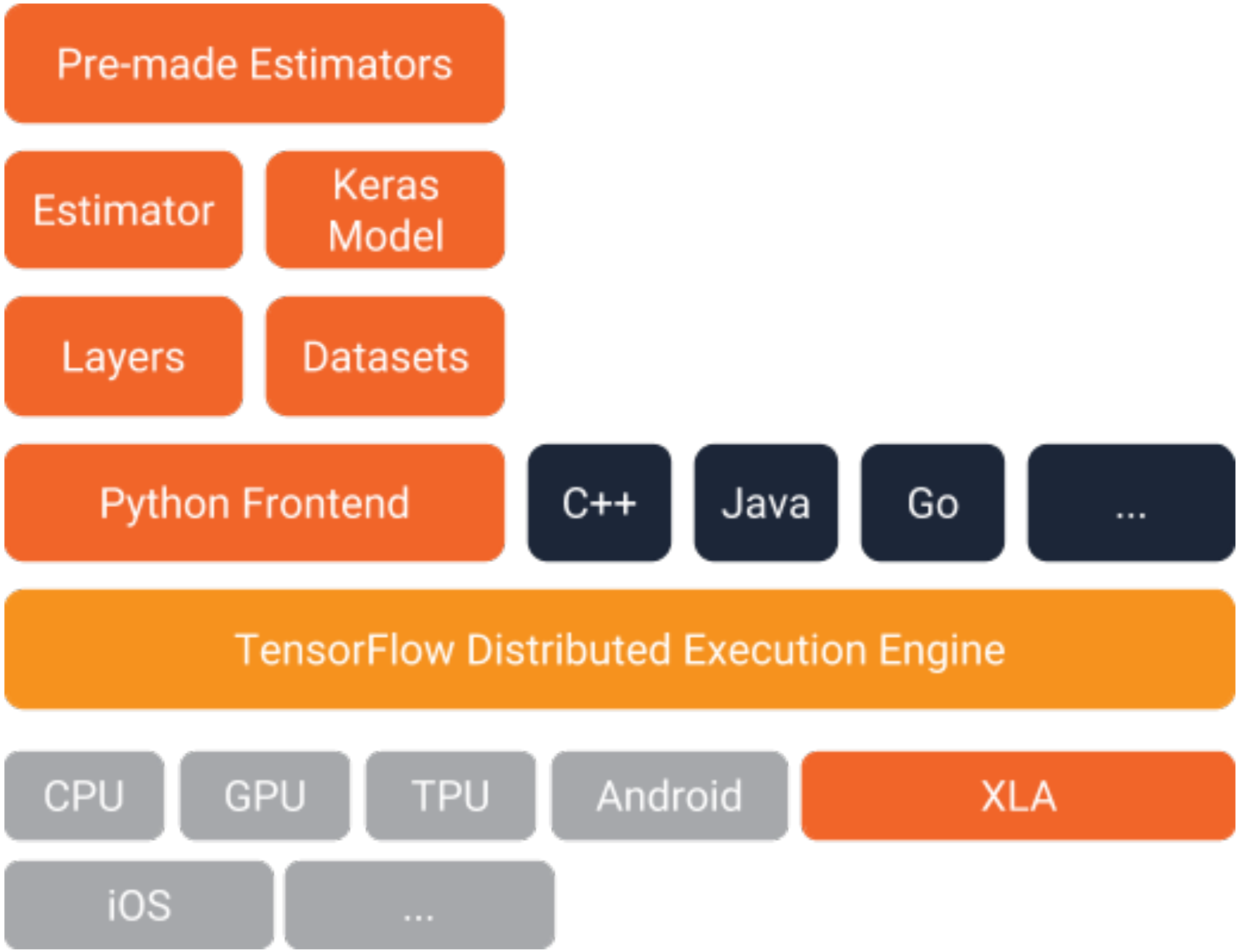
Software libraries

- TensorFlow 

- Keras 

- Theano theano
- Caffe Caffe
- Torch 

TensorFlow



[From : https ://developers.googleblog.com/2017/09/introducing-tensorflow-datasets.html]

TensorFlow

In [2]:
```
# Parameters
learning_rate = 0.01
training_epochs = 1000
display_step = 50
```

In [3]:
```
# Training Data
train_X = numpy.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,
                         7.042,10.791,5.313,7.997,5.654,9.27,3.1])
train_Y = numpy.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,
                         2.827,3.465,1.65,2.904,2.42,2.94,1.3])
n_samples = train_X.shape[0]
```

In [4]:
```
# tf Graph Input
X = tf.placeholder("float")
Y = tf.placeholder("float")

# Set model weights
W = tf.Variable(rng.randn(), name="weight")
b = tf.Variable(rng.randn(), name="bias")
```

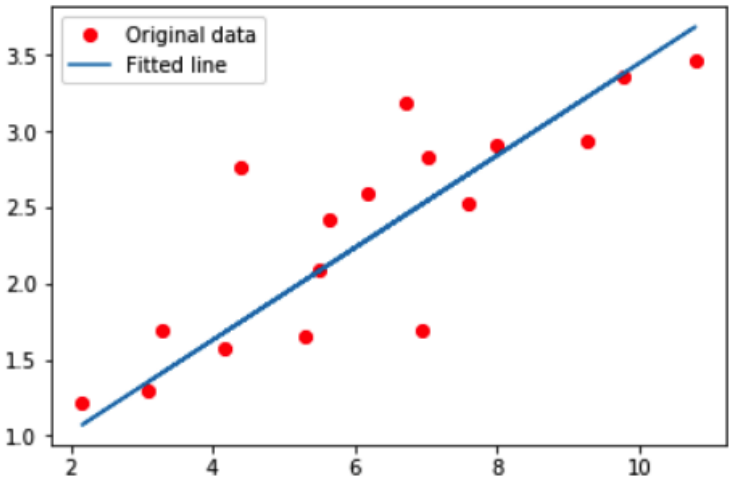
In [5]:
```
# Construct a linear model
pred = tf.add(tf.multiply(X, W), b)
```

In [6]:
```
# Mean squared error
cost = tf.reduce_sum(tf.pow(pred-Y, 2))/(2*n_samples)
# Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

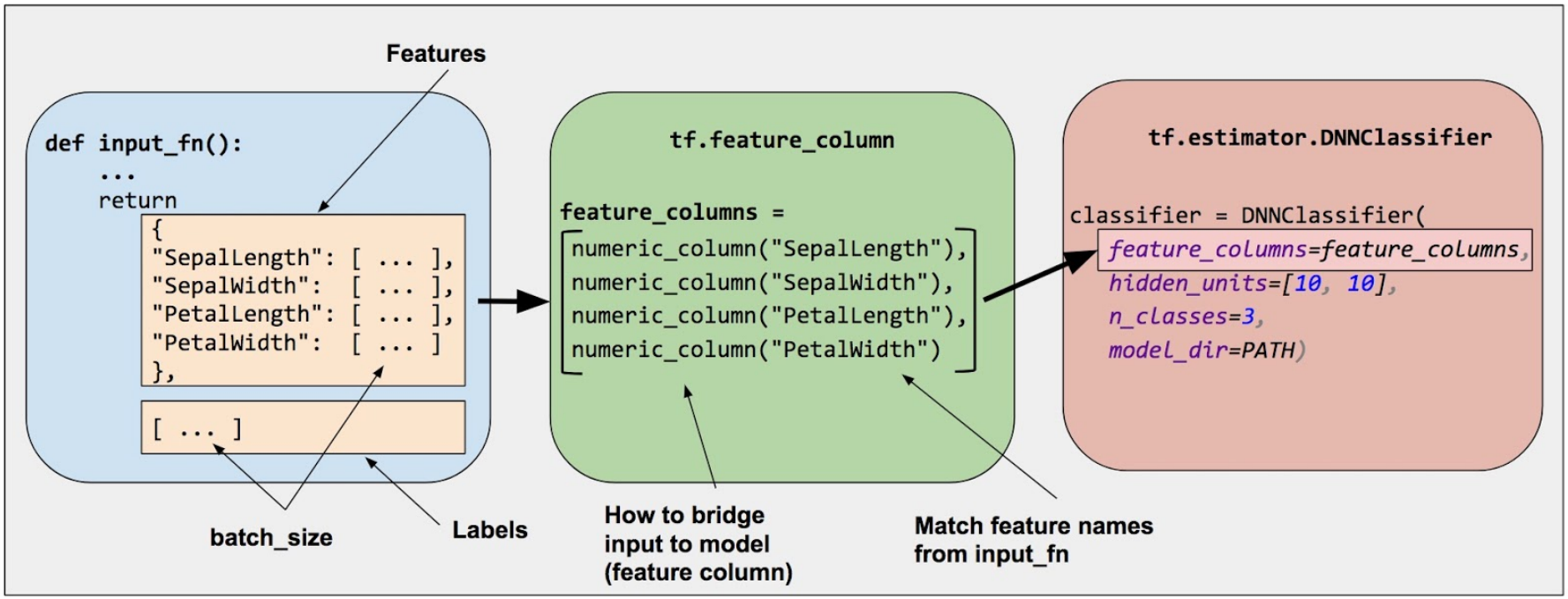
In [7]:
```
# Initialize the variables (i.e. assign their default value)
```

TensorFlow

In [10]:
```
# Start training
with tf.Session() as sess:
    sess.run(init)

    # Fit all training data
    for epoch in range(training_epochs):
        for (x, y) in zip(train_X, train_Y):
            sess.run(optimizer, feed_dict={X: x, Y: y})

    training_cost = sess.run(cost, feed_dict={X: train_X, Y: train_Y})

    #Graphic display
    plt.plot(train_X, train_Y, 'ro', label='Original data')
    plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Fitted line')
    plt.legend()
    plt.show()
```

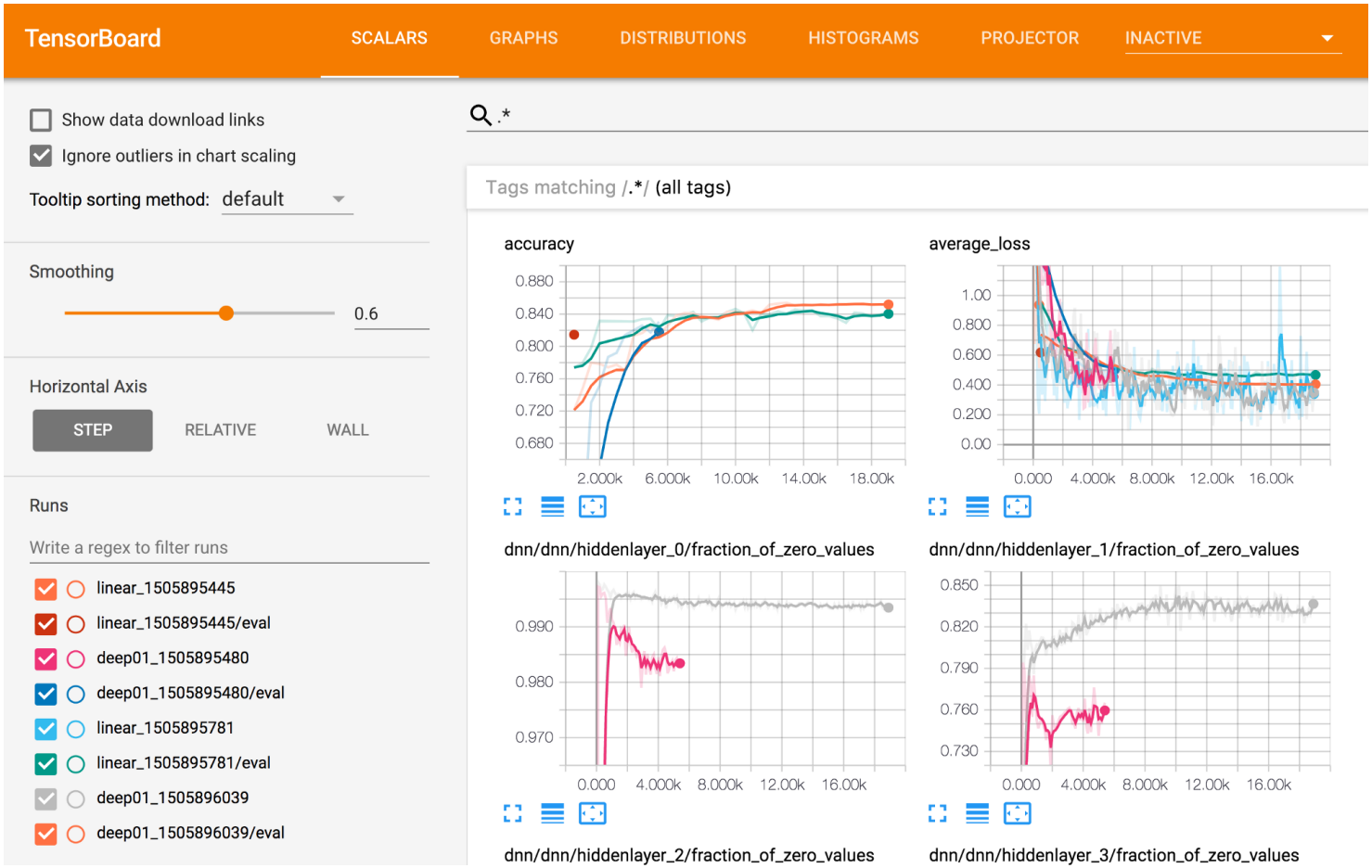


TensorFlow



[From : https ://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html]

TensorFlow

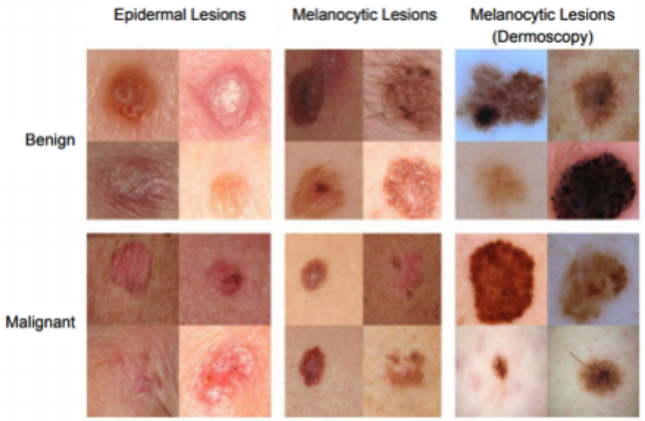


[From : https ://towardsdatascience.com/visualizing-your-model-using-tensorboard-796ebb73e98d]

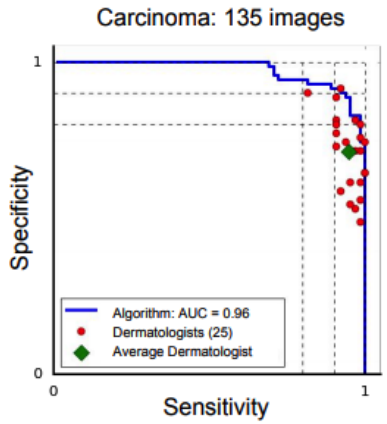
Outline

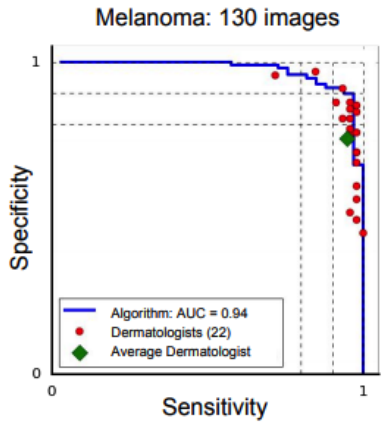
- Part I
  - Linear Regression
  - Logistic Regression
  - Neural Network
  - Convolutional Neural Network (CNN or ConvNet)
  - Recurrent Neural Network (RNN)
- Part II : Other aspects
  - Evaluation
  - Regularization
  - Transfer learning
  - Gradient-Descent optimizations
  - Software and hardware infrastructure
- Conclusion

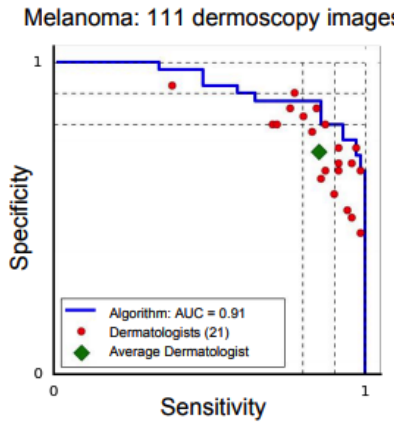
Image classification



Test set: Dermatologist Comparison (376 images)

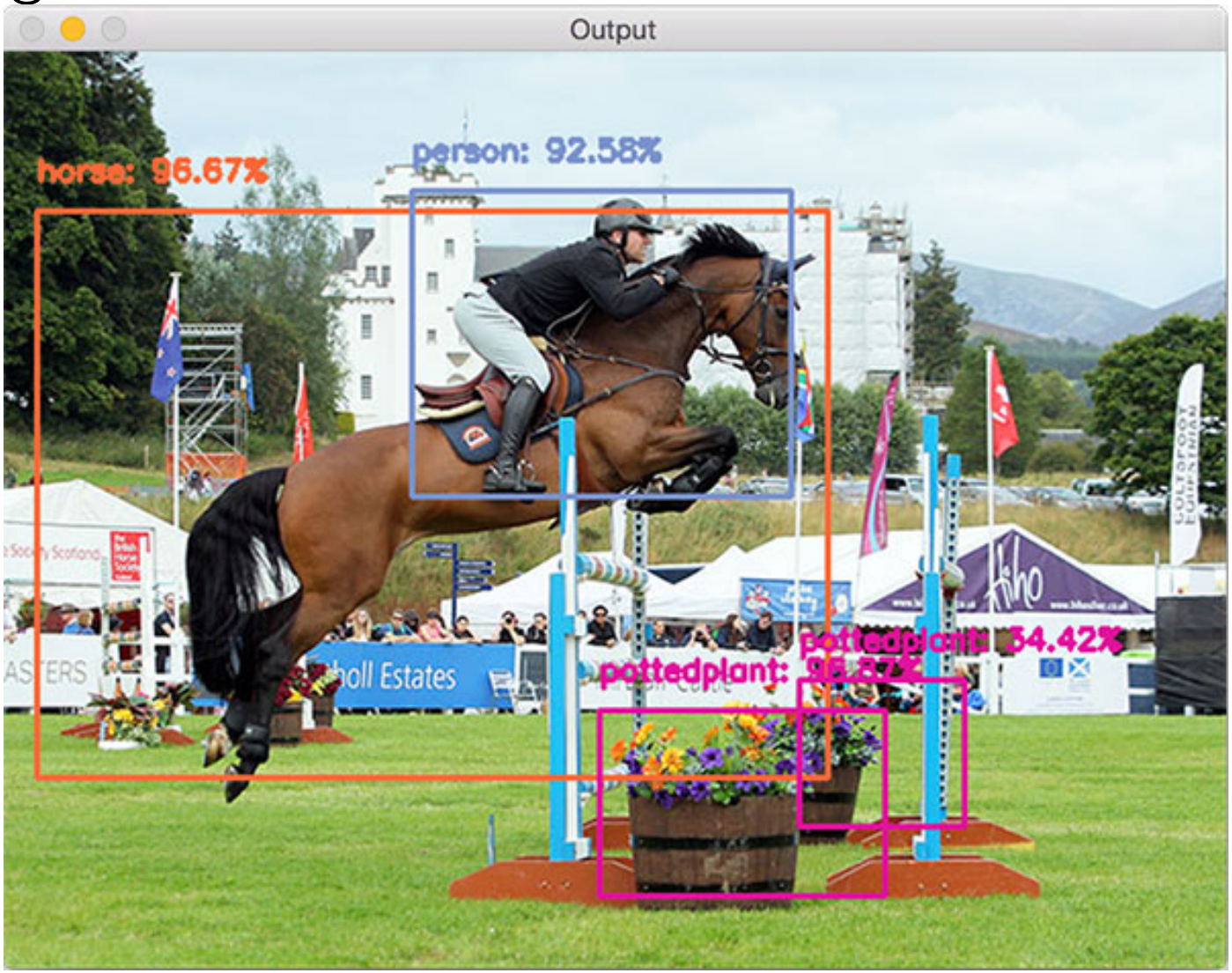






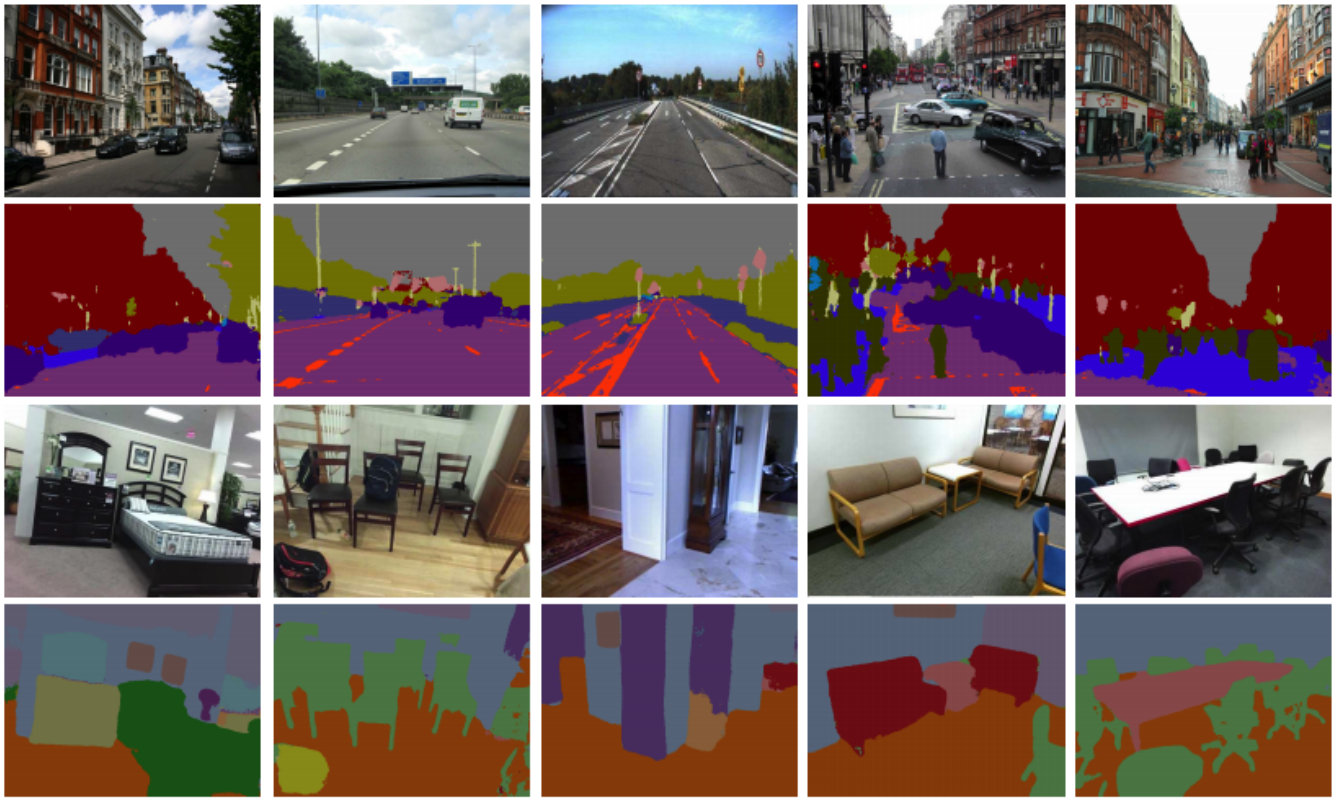
[From : Dermatologist-level classification of skin cancer with deep neural networks ; A. Esteva et Al.]

Image detection and localisation



[From :
https ://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/]

Image Semantic Segmentation



[From : SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation ; Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla]

Generating Image Descriptions

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

[From : Deep Visual-Semantic Alignments for Generating Image Descriptions ; Andrej Karpathy, Li Fei-Fei]

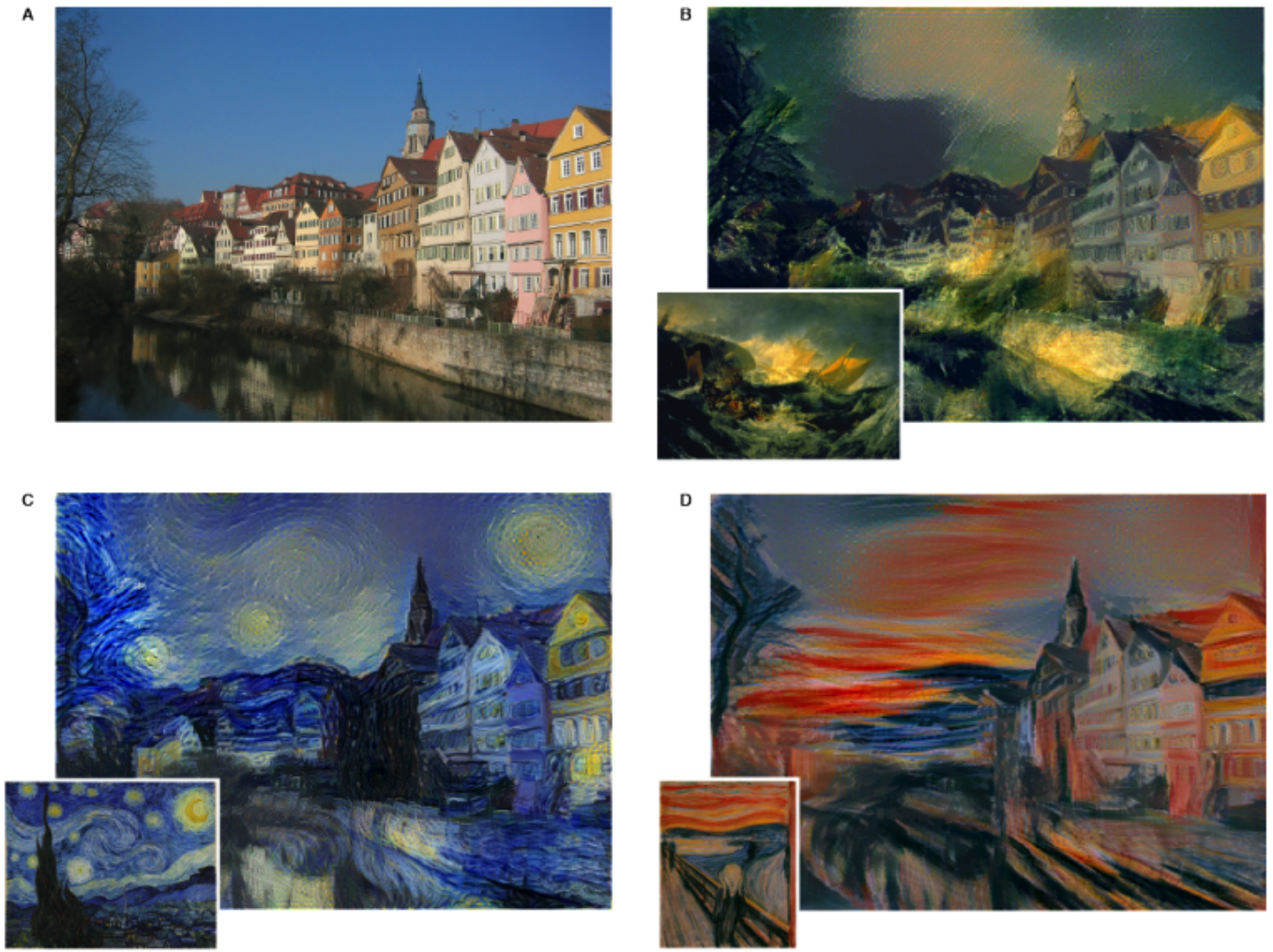
Generating Image Descriptions

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

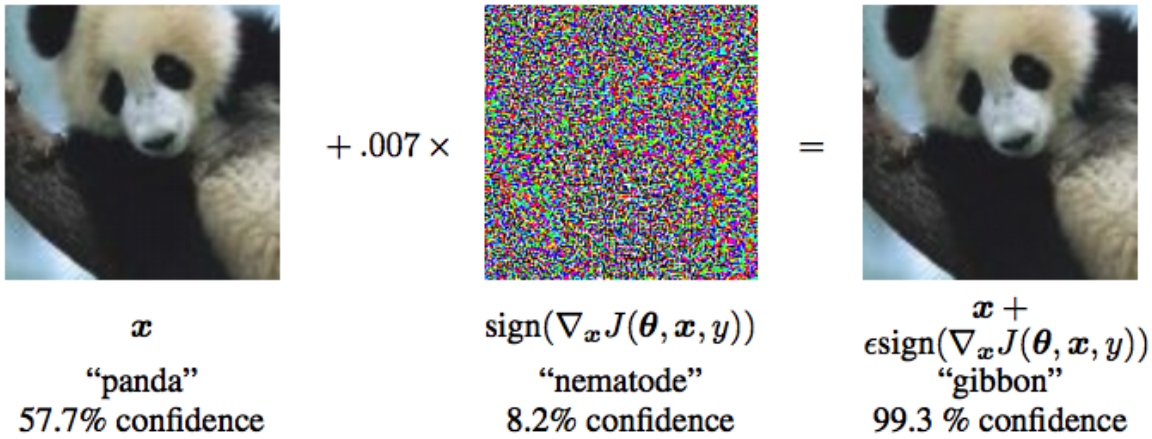
[From : Deep Visual-Semantic Alignments for Generating Image Descriptions ; Andrej Karpathy, Li Fei-Fei]

Style transfer

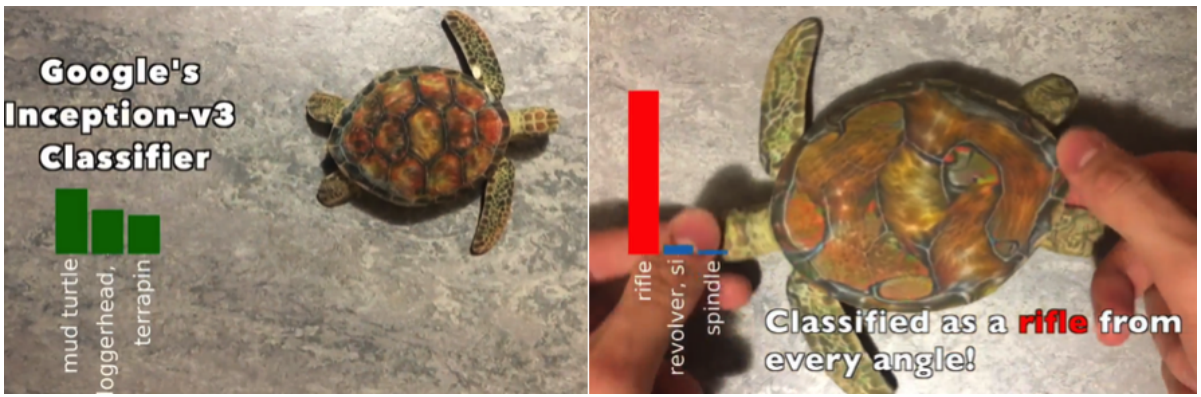


[From : A Neural Algorithm of Artistic Style ; Leon A. Gatys, Alexander S. Ecker, Matthias Bethge]

Adversarial Examples

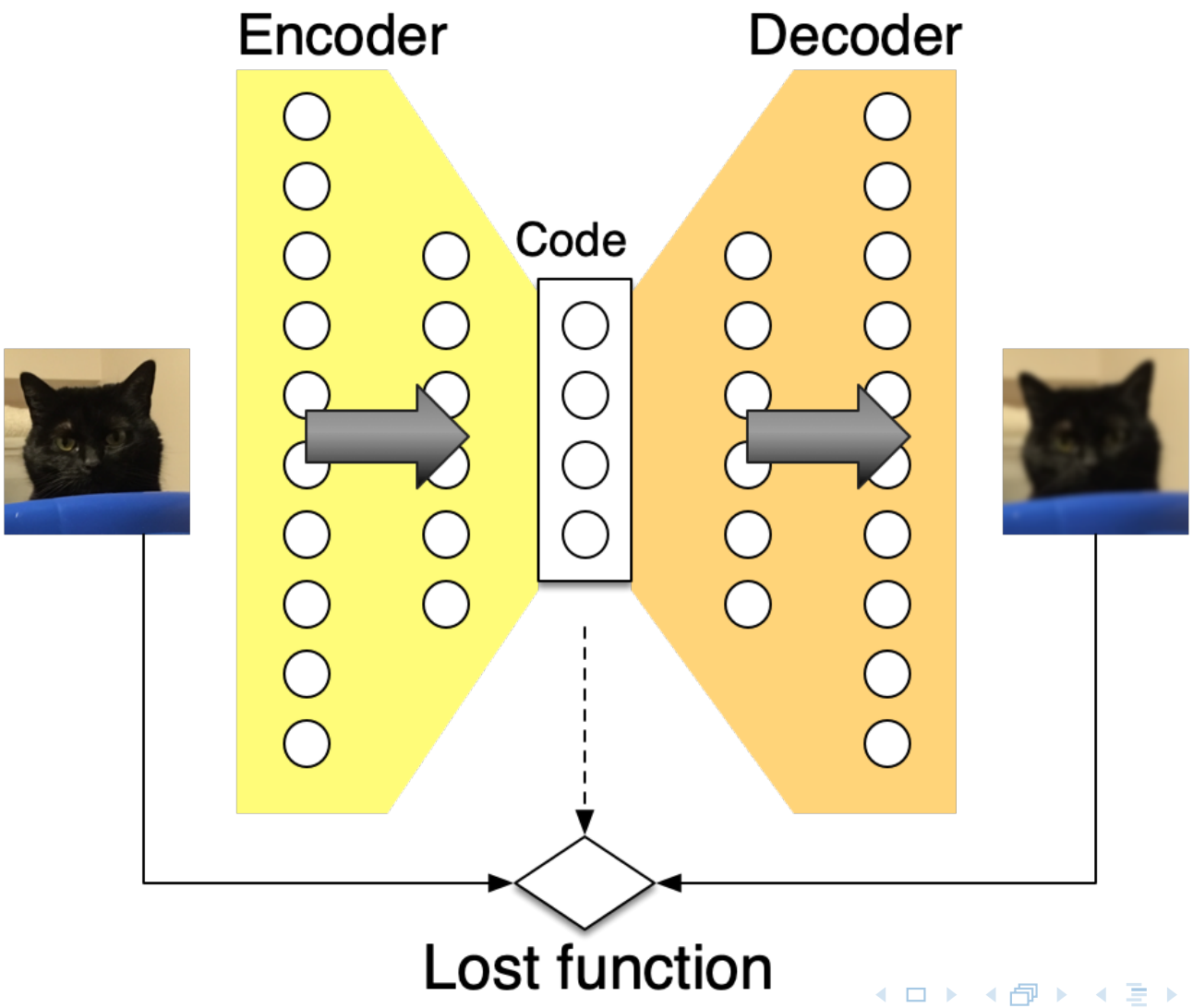


[From : Explaining and Harnessing Adversarial Examples ; I. Goodfellow, J. Shlens, C. Szegedy]

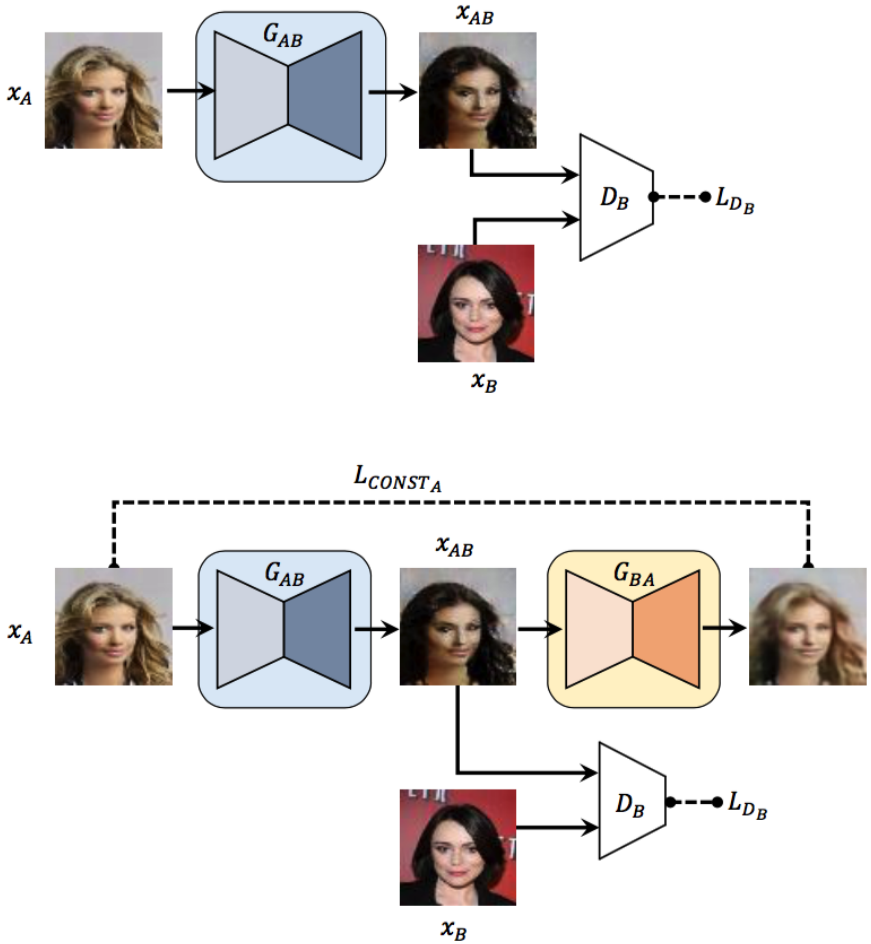


[From : https ://www.labsix.org/physical-objects-that-fool-neural-nets/]

Auto-encoder



Generative Adversarial Network : GAN

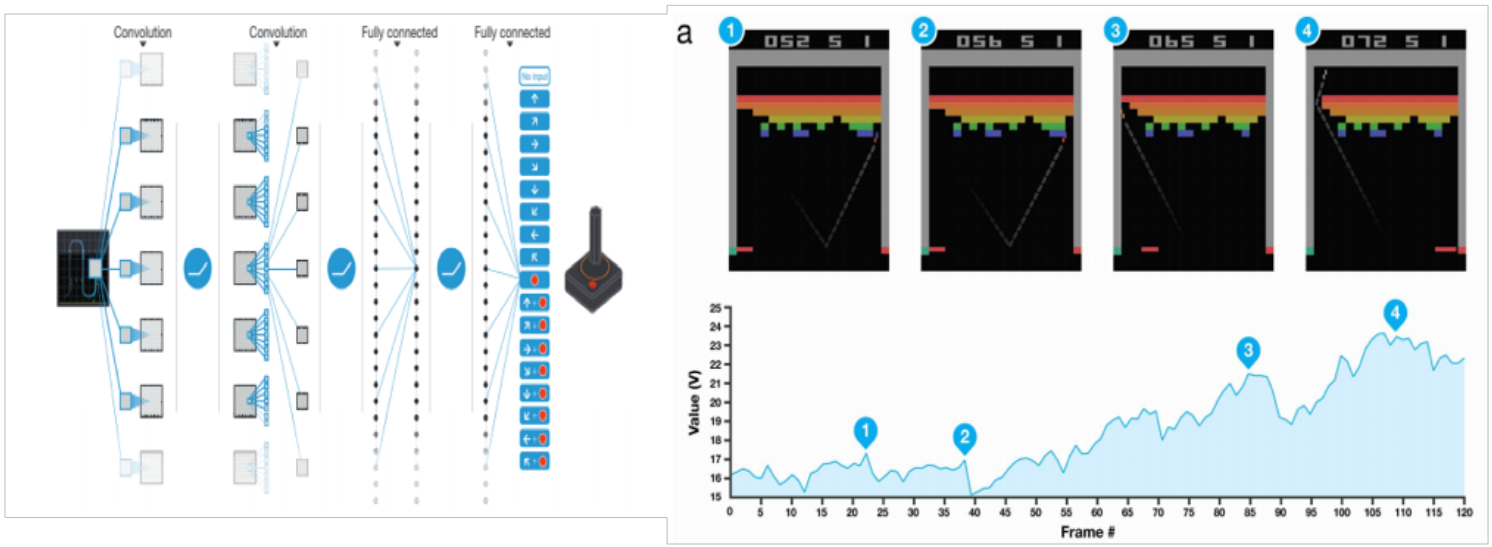


[From : Learning to Discover Cross-Domain Relations with Generative Adversarial Networks ; Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, Jiwon Kim]

Generative Adversarial Network : GAN

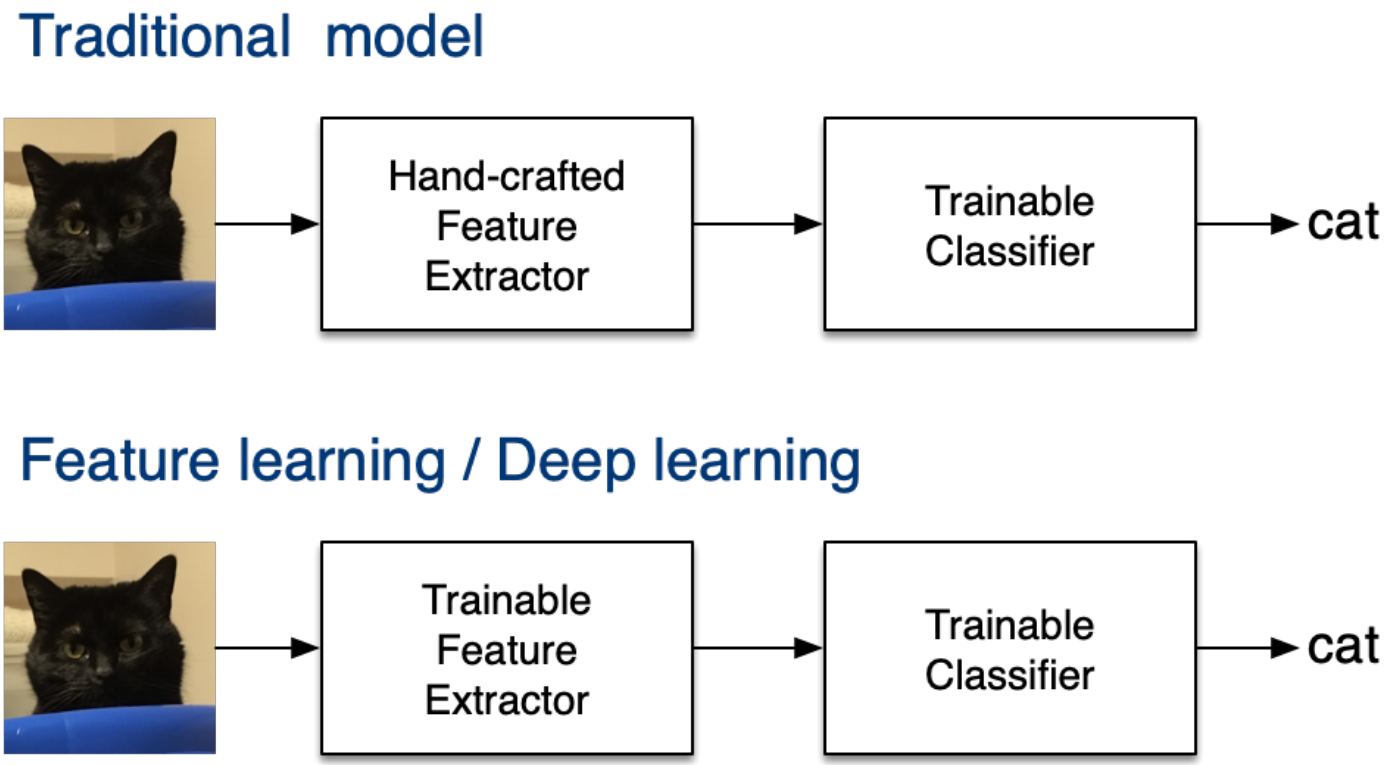
[From : Learning to Discover Cross-Domain Relations with Generative Adversarial Networks ; Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, Jiwon Kim]

Deep Reinforcement learning



[From : Human-level control through deep reinforcement learning ; Volodymyr Mnih, ...]

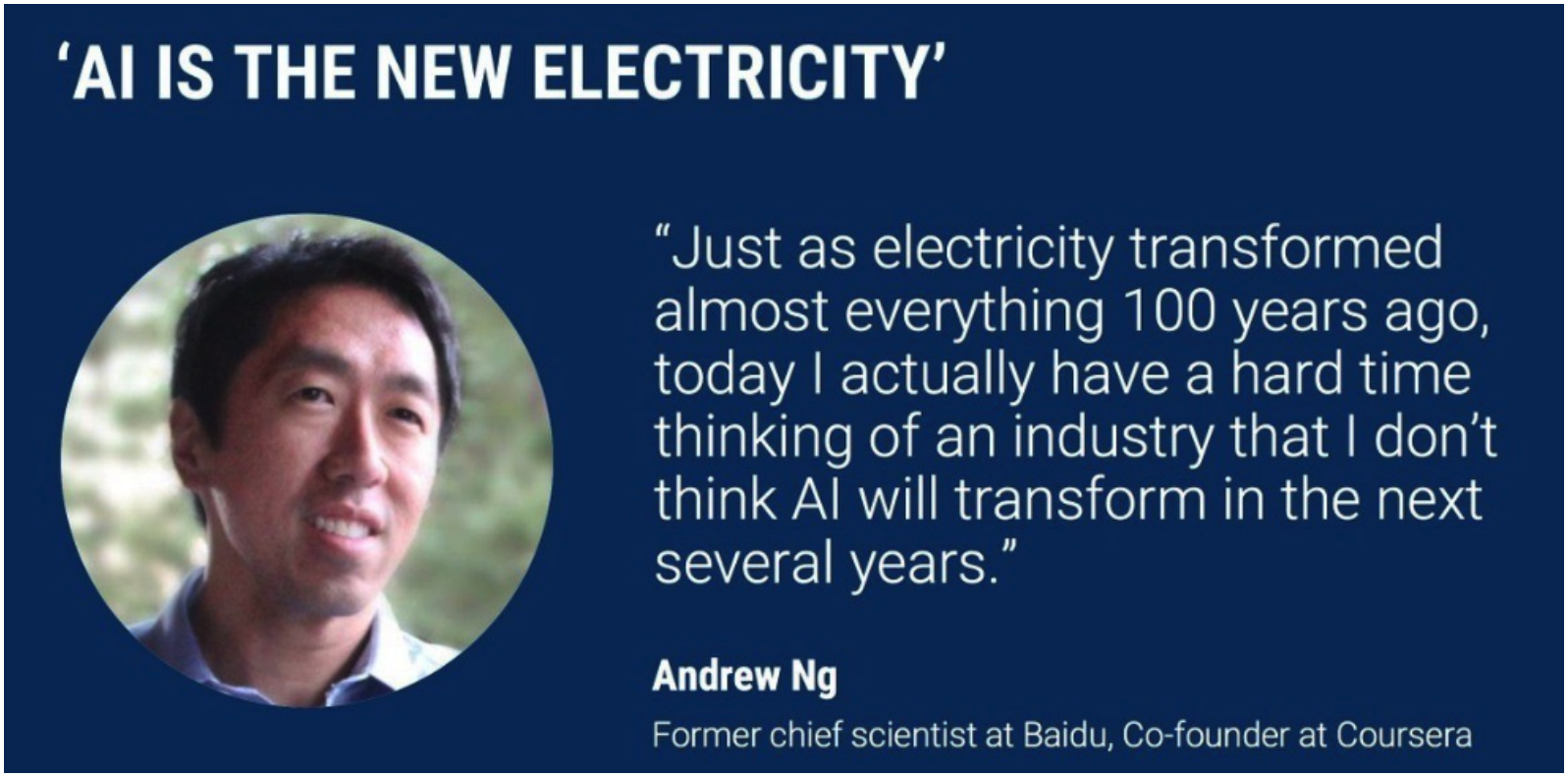
Deep Learning



References

- Books
  - Deep Learning ; Ian Goodfellow and Yoshua Bengio ; MIT Press ; 0262035618
  - Deep Learning with Python ; Francois Chollet ; Manning ; 1617294438
  - Fundamentals of Deep Learning ; Nikhil Buduma ; O'Reilly ; 1491925612
- MOOC : Coursera - Andrew Ng
  - Machine Learning
  - Deep Learning
- Site : www.kaggle.com

Conclusion


'AI IS THE NEW ELECTRICITY'
"Just as electricity transformed almost everything 100 years ago, today I actually have a hard time thinking of an industry that I don't think AI will transform in the next several years."
Andrew Ng
Former chief scientist at Baidu, Co-founder at Coursera