Python for Deep Learning

19 September 2020

@Maths, CMU

1

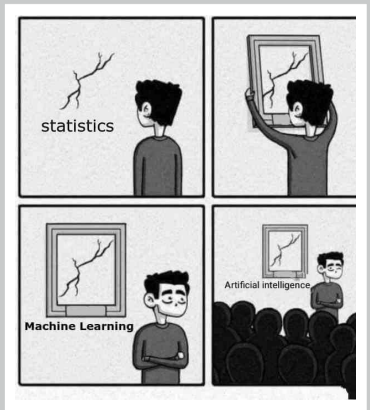
Notepad++ : https://notepad-plus-plus.org/downloads/

Visual Studio Code : https://code.visualstudio.com/

2

3

4

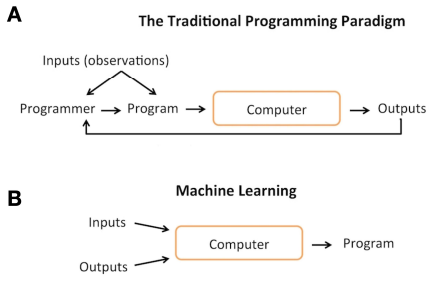


What is Machine Learning?

- Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead.
- https://en.wikipedia.org/wiki/Machine_learning

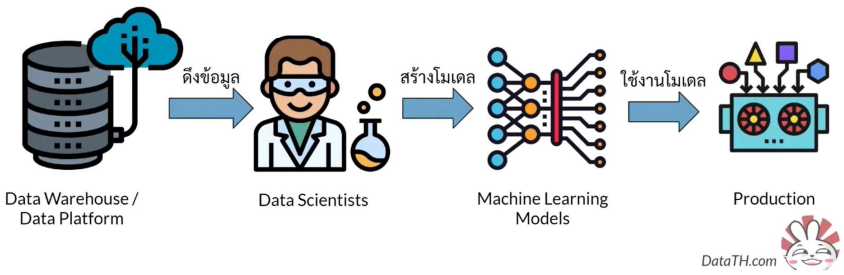
5

Comparison between traditional programming (A) and machine learning (B)



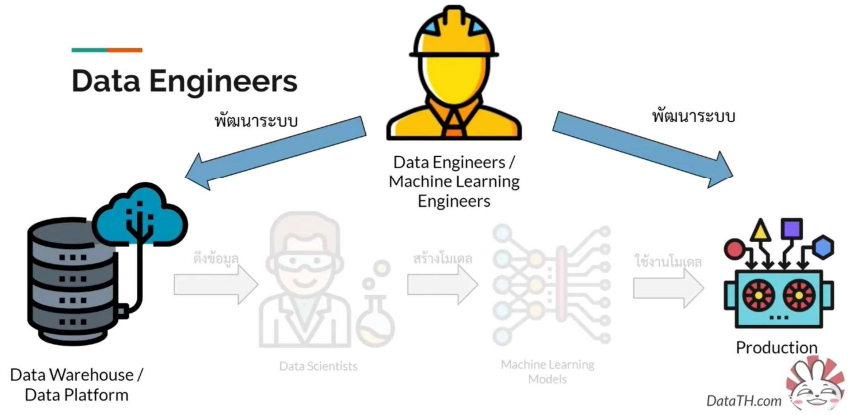
6

Data Science



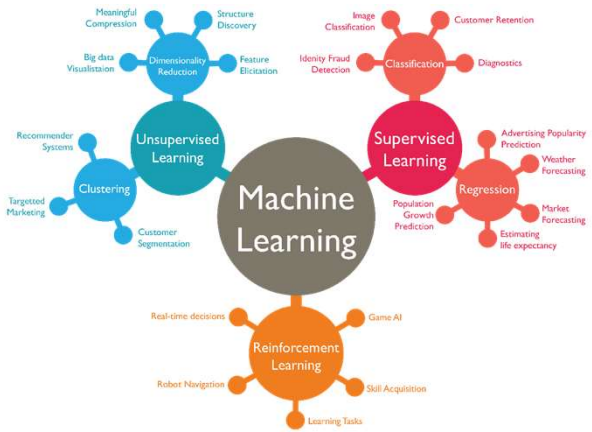
7

Data Engineers



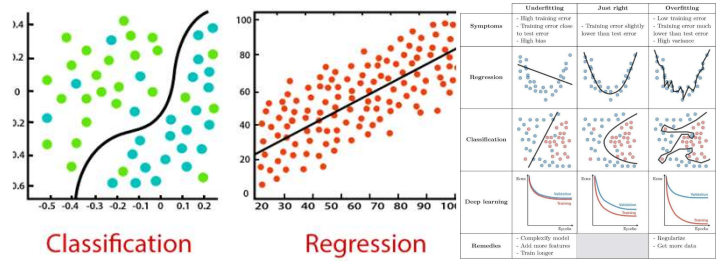
8

9



10

Regression vs Classification



11

Python for Deep Learning

Implement Python with Colab
A brief of Artificial Neural Networks
…
PyTorch package : Tools for Deep Learning
…
Deep Learning : Deep Neural Networks
Convolution Neural Networks : CNNs
Generative Adversarial Networks : GANs

12

A Brief of Artificial Neural Networks

Perceptron : The First Neural Networks

…

Deep Learning : Multi-layer Perceptron / Deep Neural Networks

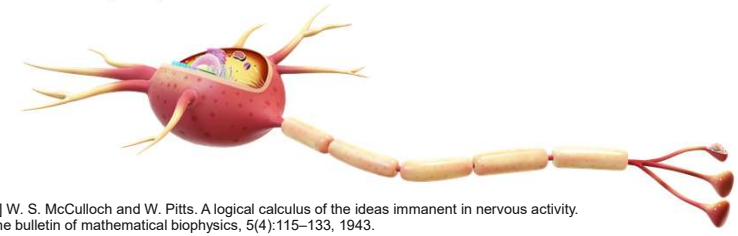
13

Artificial Neural Networks : ANNs

14

A brief of Artificial Neural Networks

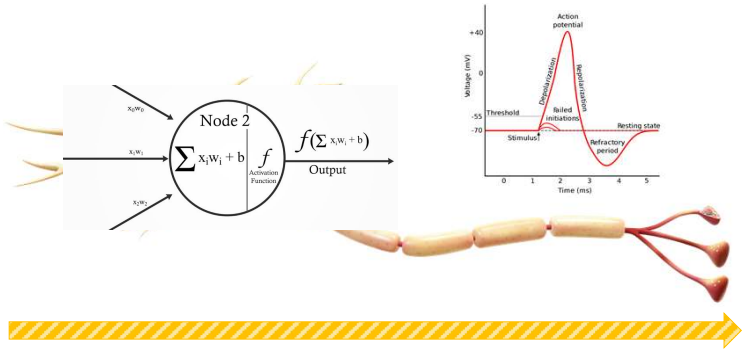
- 1943 ANNs began with Warren McCulloch and Walter Pitts [1] who drew an analogy between biological neurons and simple logic gates *with binary outputs.*



[1] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

15

A brief of Artificial Neural Networks

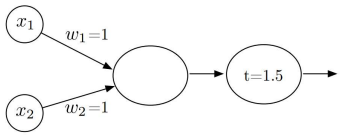


16

A brief of Artificial Neural Networks

- 1943 ANNs began with Warren McCulloch and Walter Pitts [1] who drew an analogy between biological neurons and simple logic gates with binary outputs.

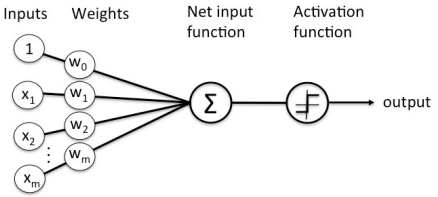
x_1	x_2	Out
0	0	0
0	1	0
1	0	0
1	1	1



[1] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

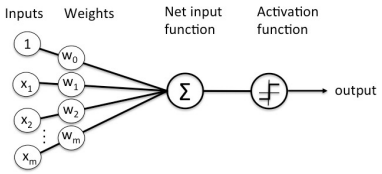
A brief of Artificial Neural Networks

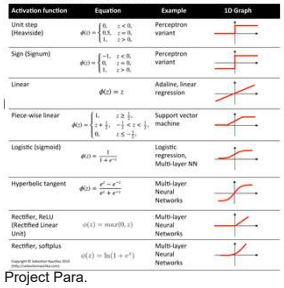
- 1949 Hebb's rule : "neurons that fire together, wire together" was proposed by Donald O. Hebb.



A brief of Artificial Neural Networks

- 1957 Frank Rosenblatt [2] published the first concept of the Perceptron learning rule.





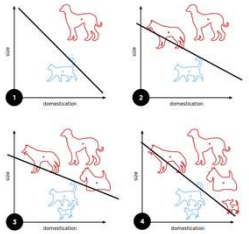
[2] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

A brief of Artificial Neural Networks

- 1957 Frank Rosenblatt [2] published the first concept of the Perceptron learning rule.

Perceptron Learning Algorithm

$k \leftarrow 1; \mathbf{w}_k \leftarrow \mathbf{0}$.
While there exists $i \in \{1, 2, \ldots, n\}$ such that $y^i(\mathbf{w}_k \cdot \mathbf{x}^i) \leqslant 0$:
  Pick an arbitrary $j \in \{1, 2, \ldots, n\}$ such that $y^j(\mathbf{w}_k \cdot \mathbf{x}^j) \leqslant 0$.
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + y^j x^j$.
  $k \leftarrow k + 1$.
Return $\mathbf{w}_k$.



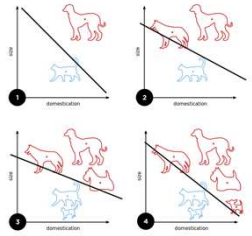
[2] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

A brief of Artificial Neural Networks

- Rosenblatt's perceptron scheme [2] :

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $x^{(i)}$:
   Calculate the output value.
   Update the weights.

$$\Delta w_j = \eta\left(target^{(i)} - output^{(i)}\right)x_j^{(i)}$$



[2] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

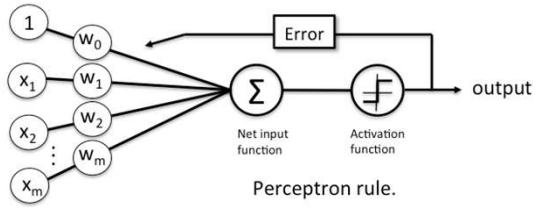
21

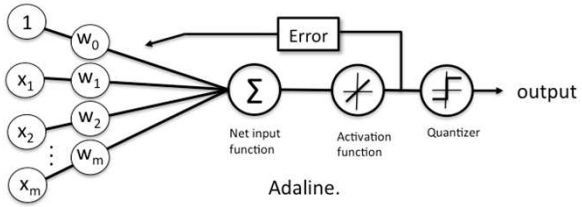
A brief of Artificial Neural Networks

- 1960 Bernard Widrow and Tedd Hoff [3] proposed the idea of the Adaptive Linear Neuron (Adaline).

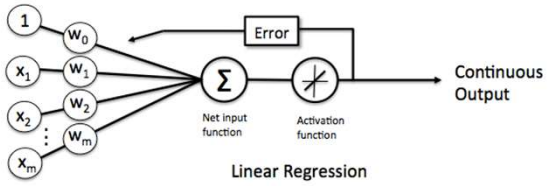
[3] B. Widrow et al. Adaptive "Adaline" neuron using chemical "memistors". Number Technical Report 1553-2. Stanford Electron. Labs., Stanford, CA, October 1960.

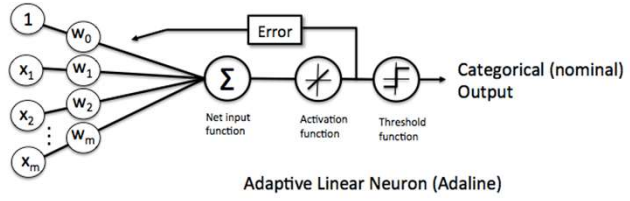
22





23



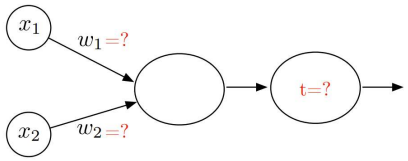


24

A brief of Artificial Neural Networks

- 1967 Marvin L. Minsky and Seymour A. Papert [4] showed that a single-layer perceptron can't implement XOR.

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	0

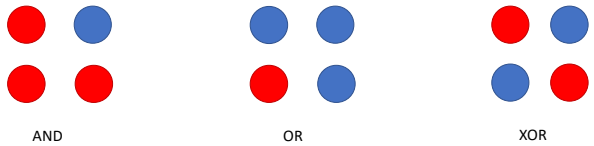


[4] Minski, M. L., & Papert, S. A. (1969). Perceptrons: an introduction to computational geometry. MA: MIT Press, Cambridge.

25

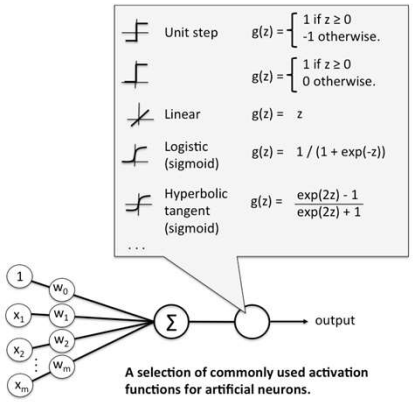
A brief of Artificial Neural Networks

- 1967 Marvin L. Minsky and Seymour A. Papert [4] showed that a single-layer perceptron can't implement XOR.
- Neurons, a dead end? Start of the first "AI Winter"



[4] Minski, M. L., & Papert, S. A. (1969). Perceptrons: an introduction to computational geometry. MA: MIT Press, Cambridge.

26

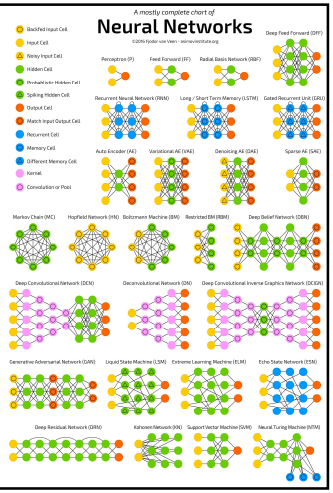


27

A brief of Artificial Neural Networks

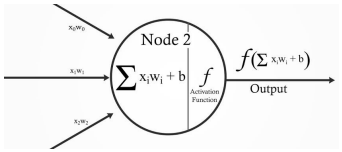
- 1974 Backpropagation was constructed and developed by many researchers, i.e. Paul Werbos, David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams
- Who Invented Backpropagation?
- http://people.idsia.ch/~juergen/who-invented-backpropagation.html
- 2nd "AI Winter" in the late 1990's and 2000's

Source : https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

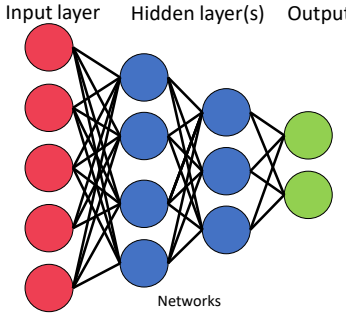


28

Artificial Neural Networks



Neural



Networks

29

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

"On-line" mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\boldsymbol{w}, b$

Gradient Descent

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - Compute output $\hat{y}^{[i]} = \sigma(\boldsymbol{w}^T x^{[i]} + b)$
    - Calculate $error = y^{[i]} - \hat{y}^{[i]}$
    - Compute $\nabla_{\boldsymbol{w}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_m}\right]^T$ and $\frac{\partial \mathcal{L}}{\partial b}$:
      - For $j \in \{1, 2, \dots, m\}$:
        - $\frac{\partial \mathcal{L}}{\partial w_j} = -error \times \sigma'(\boldsymbol{w}^T x^{[i]} + b) x_j^{[i]}$
        - $\frac{\partial \mathcal{L}}{\partial b} = -error \times \sigma'(\boldsymbol{w}^T x^{[i]} + b)$
    - Update $\boldsymbol{w}, b$:
      $\boldsymbol{w} \coloneqq \boldsymbol{w} + \eta \nabla_{\boldsymbol{w}} \mathcal{L},\ b \coloneqq b + \eta \frac{\partial \mathcal{L}}{\partial b}$

30

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

"On-line" mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\boldsymbol{w}, b$

Batch mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - Initialize $\Delta \boldsymbol{w} \coloneqq 0, \Delta b \coloneqq 0$
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\Delta \boldsymbol{w}, \Delta b$:
  - Update $\boldsymbol{w}, b$
    $\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta \boldsymbol{w},\ b \coloneqq b + \Delta b$

31

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

Batch mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - Initialize $\Delta \boldsymbol{w} \coloneqq 0, \Delta b \coloneqq 0$
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\Delta \boldsymbol{w}, \Delta b$:
  - Update $\boldsymbol{w}, b$
    $\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta \boldsymbol{w},\ b \coloneqq b + \Delta b$

Batch Gradient Descent

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - Initialize $\Delta \boldsymbol{w} \coloneqq 0, \Delta b \coloneqq 0$
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - Compute $\nabla_{\boldsymbol{w}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_m}\right]^T$ and $\frac{\partial \mathcal{L}}{\partial b}$
    - Update $\Delta \boldsymbol{w}, \Delta b$:
      - $\Delta \boldsymbol{w} \coloneqq \Delta \boldsymbol{w} + \eta \nabla_{\boldsymbol{w}} \mathcal{L}, \Delta b \coloneqq \Delta b + \eta \frac{\partial \mathcal{L}}{\partial b}$
  - Update $\boldsymbol{w}, b$
    $\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta \boldsymbol{w},\ b \coloneqq b + \Delta b$

32

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

Gradient Descent

("On-line" mode)

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - Compute output $\hat{y}^{[i]} = \sigma(\boldsymbol{w}^T x^{[i]} + b)$
    - Calculate $error = y^{[i]} - \hat{y}^{[i]}$
    - Compute $\nabla_{\boldsymbol{w}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_m}\right]^T$ and $\frac{\partial \mathcal{L}}{\partial b}$
      - For $j \in \{1, 2, \dots, m\}$:
        - $\frac{\partial \mathcal{L}}{\partial w_j} = -error \times \sigma'(\boldsymbol{w}^T x^{[i]} + b) x_j^{[i]}$
        - $\frac{\partial \mathcal{L}}{\partial b} = -error \times \sigma'(\boldsymbol{w}^T x^{[i]} + b)$
    - Update $\boldsymbol{w}, b$:

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \eta \nabla_{\boldsymbol{w}} \mathcal{L},\ b \coloneqq b + \eta \frac{\partial \mathcal{L}}{\partial b}$$

Batch Gradient Descent

(Batch mode)

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - Initialize $\Delta \boldsymbol{w} \coloneqq 0, \Delta b \coloneqq 0$
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - Compute $\nabla_{\boldsymbol{w}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_m}\right]^T$ and $\frac{\partial \mathcal{L}}{\partial b}$
    - Update $\Delta \boldsymbol{w}, \Delta b$:
      - $\Delta \boldsymbol{w} \coloneqq \Delta \boldsymbol{w} + \eta \nabla_{\boldsymbol{w}} \mathcal{L}, \Delta b \coloneqq \Delta b + \eta \frac{\partial \mathcal{L}}{\partial b}$
  - Update $\boldsymbol{w}, b$

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta \boldsymbol{w},\ b \coloneqq b + \Delta b$$

33

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

"On-line" mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\boldsymbol{w}, b$

Batch mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - Initialize $\Delta \boldsymbol{w} \coloneqq 0, \Delta b \coloneqq 0$
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\Delta \boldsymbol{w}, \Delta b$:
  - Update $\boldsymbol{w}, b$

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta \boldsymbol{w},\ b \coloneqq b + \Delta b$$

34

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

"On-line" mode

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - For every $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\boldsymbol{w}, b$

"On-line" mode II (alternative)

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For each t iterations:
  - Pick random $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\boldsymbol{w}, b$

35

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

"On-line" mode II (alternative)

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For each t iterations:
  - Pick random $(x^{[i]}, y^{[i]}) \in D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\boldsymbol{w}, b$

Stochastic Gradient Descent

("On-line" mode)

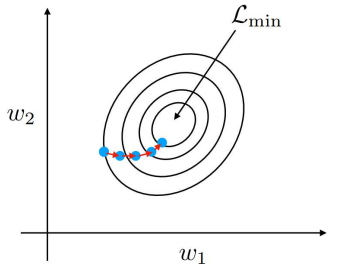
- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For each t iterations:
  - Pick random $(x^{[i]}, y^{[i]}) \in D$:
    - Compute output $\hat{y}^{[i]} = \sigma(\boldsymbol{w}^T x^{[i]} + b)$
    - Calculate $error = y^{[i]} - \hat{y}^{[i]}$
    - Compute $\nabla_{\boldsymbol{w}} \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial w_1}, \dots, \frac{\partial \mathcal{L}}{\partial w_m}\right]^T$ and $\frac{\partial \mathcal{L}}{\partial b}$
    - Update $\boldsymbol{w}, b$:

$$\boldsymbol{w} \coloneqq \boldsymbol{w} + \eta \nabla_{\boldsymbol{w}} \mathcal{L},\ b \coloneqq b + \eta \frac{\partial \mathcal{L}}{\partial b}$$

36

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$



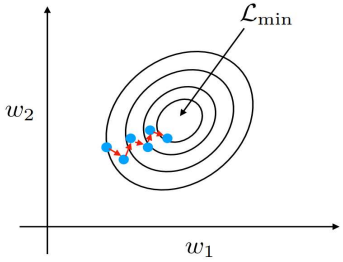
Batch Gradient Descent
as surface plot

37

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

Stochastic Gradient Descent
as surface plot



38

General Learning Principle

- Let $D = \{(x^{[i]}, y^{[i]}) \in \mathbb{R}^m \times \{0,1\} : i \in \{1, 2, \dots, n\}\}$

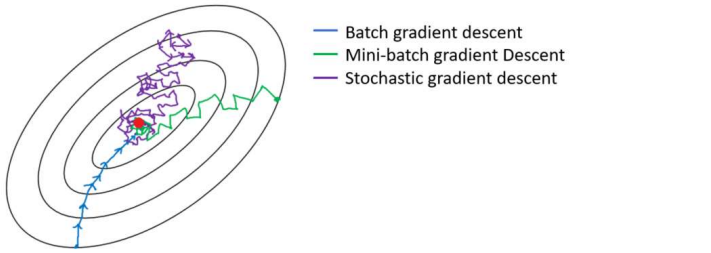
Minibatch mode
(on-line + batch)

Most common used in deep leaning!!

- Initialize $\boldsymbol{w} \coloneqq 0^m, b \coloneqq 0$
- For every training epoch:
  - Initialize $\Delta \boldsymbol{w} \coloneqq 0, \Delta b \coloneqq 0$
  - For every $\{(x^{[i]}, y^{[i]}), \dots, (x^{[i+k]}, y^{[i+k]})\} \subset D$:
    - (a) Compute output (prediction)
    - (b) Calculate error
    - (c) Update $\Delta \boldsymbol{w}, \Delta b$
  - Update $\boldsymbol{w}, b$:
    $\boldsymbol{w} \coloneqq \boldsymbol{w} + \Delta \boldsymbol{w}, \; b \coloneqq b + \Delta b$

39

General Learning Principle

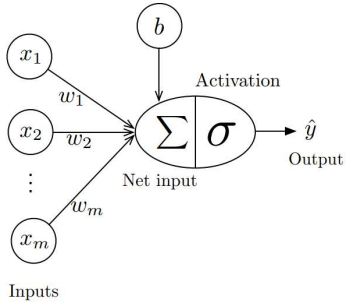


Source : https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

40

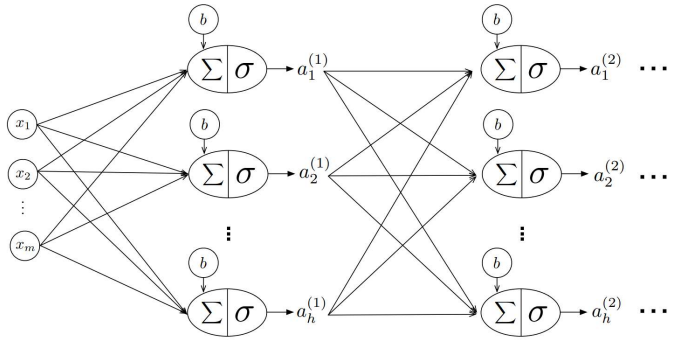
Linear Algebra in DL

- Output $\hat{y} \coloneqq \sigma(z)$ where $z = x^T \boldsymbol{w} + b, \boldsymbol{w} \in \mathbb{R}^{m \times 1}$
- For 1 example : $x \in \mathbb{R}^{m \times 1}$,
  - $z = x^T \boldsymbol{w} + b$
  - $\hat{y} = \sigma(z)$
- For n examples : $X \in \mathbb{R}^{n \times m}$
  - $X\boldsymbol{w} + b = \begin{bmatrix} (x^{[1]})^T \boldsymbol{w} + b \\ \vdots \\ (x^{[n]})^T \boldsymbol{w} + b \end{bmatrix} = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \boldsymbol{z} \in \mathbb{R}^{n \times 1}$
  - $\hat{y} = \begin{bmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_n) \end{bmatrix} = \sigma(\boldsymbol{z})$



41

Linear Algebra in DL



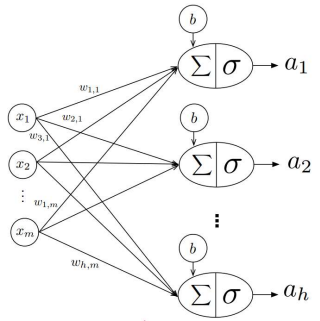
42

Linear Algebra in DL (Based on PyThorch)

- Output $a \coloneqq \sigma(z)$

where $z = xW^T + b,\ W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,m} \\ \vdots & \ddots & \vdots \\ w_{h,1} & \cdots & w_{h,m} \end{bmatrix}$

- For 1 example : $x = [x_1, \dots, x_m] \in \mathbb{R}^{1 \times m}$,
  - $\sigma(xW^T + b) = a \in \mathbb{R}^{1 \times h}$
- For n examples : $X \in \mathbb{R}^{n \times m}$
  - $\sigma(XW^T + b) = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} = A \in \mathbb{R}^{n \times h}$



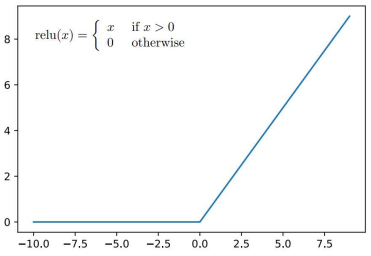
Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

43

AUTOGRAD: AUTOMATIC DIFFERENTIATION

- Suppose we have the following activation function:

$$a(x, w, b) = relu(w \cdot x + b)$$

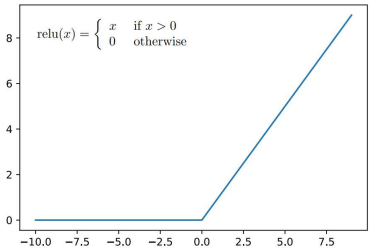


ReLU = Rectified Linear Unit
(prob. the most commonly used activation function in DL)

44

AUTOGRAD: AUTOMATIC DIFFERENTIATION

- You may note that ($relu \coloneqq f$)
  - $f'(x) = \begin{cases} 1, x > 0; \\ 0, x < 0; \\ \infty, x = 0 \end{cases}$
- But in computer science,
  - $f'(x) = \begin{cases} 1, x > 0; \\ 0, x \leq 0 \end{cases}$

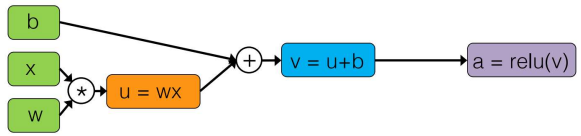


45

AUTOGRAD: AUTOMATIC DIFFERENTIATION

$$a(x,\ w,\ b) = relu(\underbrace{w \cdot x}_{u} + \underbrace{b}_{v})$$

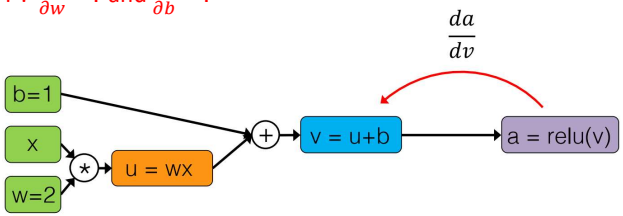
WANT : $\frac{\partial a}{\partial w}$ =? and $\frac{\partial a}{\partial b}$ =?



46

AUTOGRAD: AUTOMATIC DIFFERENTIATION

WANT : $\frac{\partial a}{\partial w}$ =? and $\frac{\partial a}{\partial b}$ =?



47

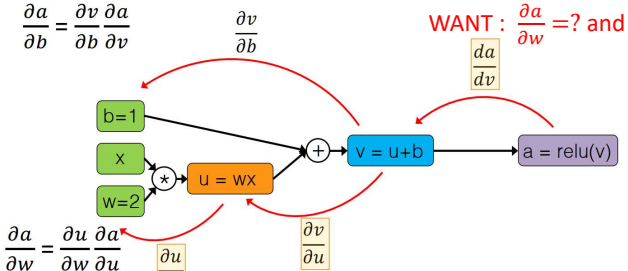
AUTOGRAD: AUTOMATIC DIFFERENTIATION

$\frac{\partial a}{\partial b} = \frac{\partial v}{\partial b}\frac{\partial a}{\partial v}$

$\frac{\partial a}{\partial w} = \frac{\partial u}{\partial w}\frac{\partial a}{\partial u}$

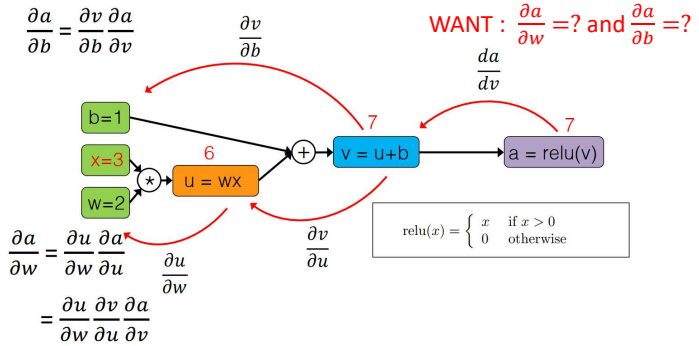
$= \frac{\partial u}{\partial w}\frac{\partial v}{\partial u}\frac{\partial a}{\partial v}$

WANT : $\frac{\partial a}{\partial w}$ =? and $\frac{\partial a}{\partial b}$ =?



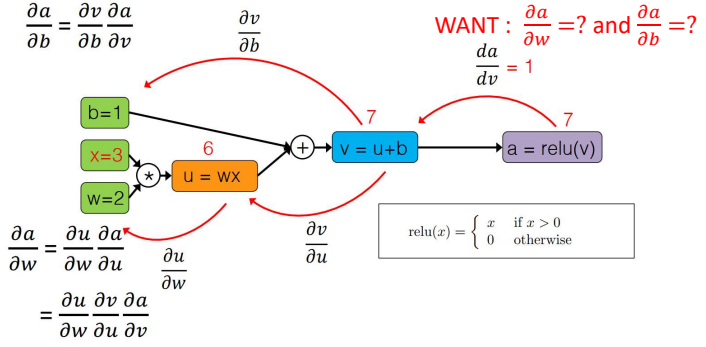
48

AUTOGRAD: AUTOMATIC DIFFERENTIATION



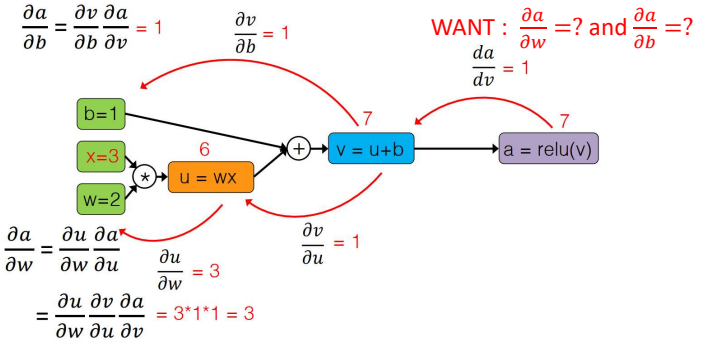
49

AUTOGRAD: AUTOMATIC DIFFERENTIATION



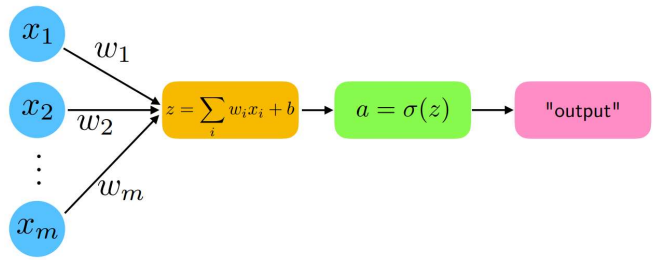
50

AUTOGRAD: AUTOMATIC DIFFERENTIATION



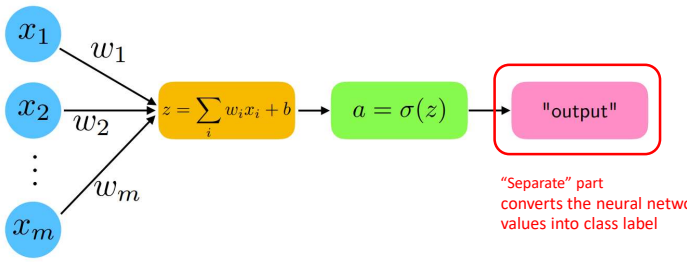
51

Generic Neuron representation (Recap)



52

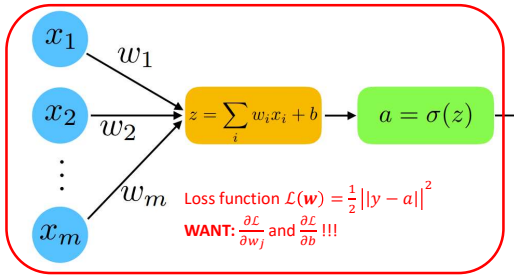
Generic Neuron representation (Recap)



"Separate" part
converts the neural network values into class label

53

Generic Neuron representation (Recap)

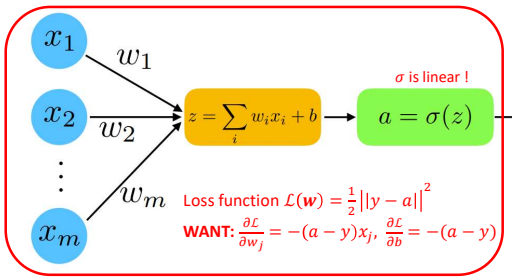


Loss function $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$
WANT: $\frac{\partial \mathcal{L}}{\partial w_j}$ and $\frac{\partial \mathcal{L}}{\partial b}$!!!

"Training" part
computes the neural network values and parameters

54

Generic Neuron representation (Recap)

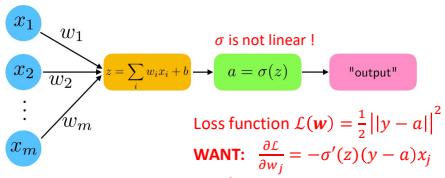


Loss function $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$
WANT: $\frac{\partial \mathcal{L}}{\partial w_j} = -(a - y)x_j$, $\frac{\partial \mathcal{L}}{\partial b} = -(a - y)$

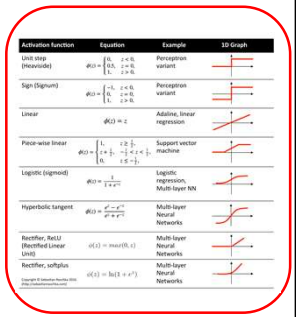
"Training" part
computes the neural network values and parameters

55

Generic Neuron representation (Recap)

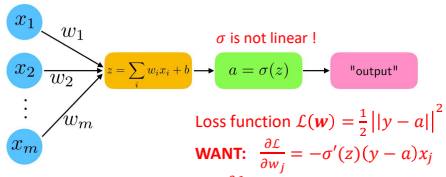


Loss function $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$
WANT: $\frac{\partial \mathcal{L}}{\partial w_j} = -\sigma'(z)(y - a)x_j$
$\frac{\partial \mathcal{L}}{\partial b} = -\sigma'(z)(y - a)$



56

Logistic Regression

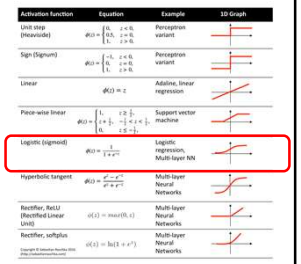


σ is not linear !

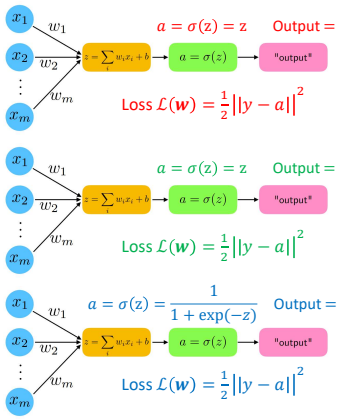
Loss function $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$

WANT: $\frac{\partial \mathcal{L}}{\partial w_j} = -\sigma'(z)(y-a)x_j$

$\frac{\partial \mathcal{L}}{\partial b} = -\sigma'(z)(y-a)$



57



$a = \sigma(z) = z$ Output = z

Loss $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$

Linear regression

$a = \sigma(z) = z$ Output = threshold(z)

Loss $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$

Adaline

$a = \sigma(z) = \frac{1}{1+\exp(-z)}$ Output = $\begin{cases} 1, & if\ a > 0.5 \\ 0, & otherwise \end{cases}$

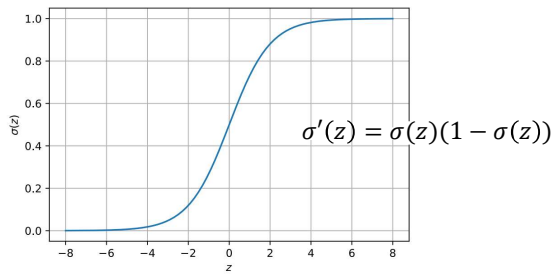
Loss $\mathcal{L}(\boldsymbol{w}) = \frac{1}{2}||y - a||^2$

Logistic regression

58

Logistic Sigmoid Function

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



$\sigma'(z) = \sigma(z)(1 - \sigma(z))$

59

Logistic Regression

WANT :
$P(y = 0|x) \approx 1$ if $y = 0$
$P(y = 1|x) \approx 1$ if $y = 1$

- Assume that outputs are independent Bernoulli random variables with success probability of $a = \sigma(\boldsymbol{w}^T x)$
- The probability mass function P of this distribution over possible outcomes y is

$$P(y|x) = \begin{cases} a, & if\ y{=}1 \\ 1-a, & if\ y = 0 \end{cases}$$

- Equivalent to,

$$P(y|x) = a^y(1-a)^{(1-y)}$$

60

Logistic Regression

- For n training examples, we want to maximize:

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P(y^{[i]} | x^{[i]})$$

61

Logistic Regression

- For n training examples, we want to maximize:

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P(y^{[i]} | x^{[i]})$$

- You may know as "Maximum Likelihood Estimation",

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P\left(y^{[i]} | x^{[i]}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \sigma\left(z^{[i]}\right)^{y^{[i]}} \left(1 - \sigma\left(z^{[i]}\right)^{1-y^{[i]}}\right)$$

62

Logistic Regression

- For n training examples, we want to maximize:

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P(y^{[i]} | x^{[i]})$$

- You may know as "Maximum Likelihood Estimation",

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P\left(y^{[i]} | x^{[i]}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \sigma\left(z^{[i]}\right)^{y^{[i]}} \left(1 - \sigma\left(z^{[i]}\right)^{1-y^{[i]}}\right)$$

- In practice, it is easier to take log in to $\mathcal{L}$, which called log-likelihood,

$$\ell(\boldsymbol{w}) = \log(\mathcal{L}(\boldsymbol{w})) = \sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right) \log(1 - \sigma(z^{[i]}))$$

63

Logistic Regression

In information theory, negative log-likelihood called as binary cross entropy

- For n training examples, we want to maximize:

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P(y^{[i]} | x^{[i]})$$

- You may know as "Maximum Likelihood Estimation",

$$P\left(y^{[1]}, y^{[2]}, \dots, y^{[n]} | x^{[1]}, x^{[2]}, \dots, x^{[n]}\right) = \prod_{i=1}^{n} P\left(y^{[i]} | x^{[i]}; \boldsymbol{w}\right) = \prod_{i=1}^{n} \sigma\left(z^{[i]}\right)^{y^{[i]}} \left(1 - \sigma\left(z^{[i]}\right)^{1-y^{[i]}}\right)$$

- In practice, it is easier to take log in to $\mathcal{L}$, which called log-likelihood,

$$\ell(\boldsymbol{w}) = \log(\mathcal{L}(\boldsymbol{w})) = \sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right) \log(1 - \sigma(z^{[i]}))$$

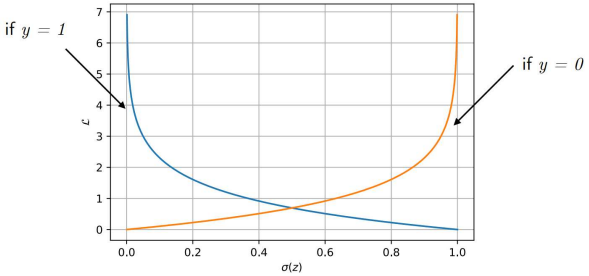
- For convenience, we take subtraction to minimize negative log-likelihood instead of maximize log- likelihood,

$$\mathcal{L}(\boldsymbol{w}) = -\ell(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right) \log(1 - \sigma(z^{[i]}))$$

64

Loss for one training example

In information theory, negative log-likelihood called as binary cross entropy



$$\mathcal{L}(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right)\log(1 - \sigma(z^{[i]}))$$

65

Logistic Regression : Learning rule

- $\mathcal{L}(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right)\log(1 - \sigma(z^{[i]}))$
- $\frac{\partial \mathcal{L}}{\partial w_j} =$

66

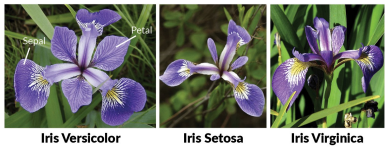
Logistic Regression : Learning rule

- $\mathcal{L}(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right)\log(1 - \sigma(z^{[i]}))$
- $\frac{\partial \mathcal{L}}{\partial w_j} = -\left(y^{[i]} - \sigma\left(z^{[i]}\right)\right) x_j$
- $w_j = w_j + \eta \frac{\partial \mathcal{L}}{\partial w_j}$
- $w_j = w_j - \eta \left(y^{[i]} - \sigma\left(z^{[i]}\right)\right) x_j \quad \Longrightarrow \quad \boldsymbol{w} = \boldsymbol{w} + \eta \nabla_w \mathcal{L}$

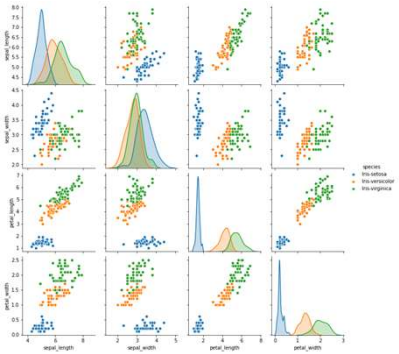
67

Multi-class Classification

Iris dataset

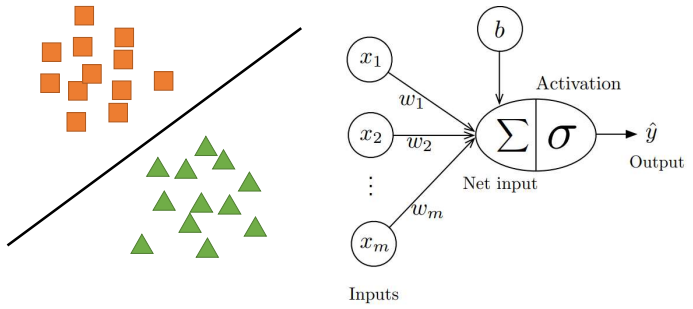


Iris Versicolor | Iris Setosa | Iris Virginica



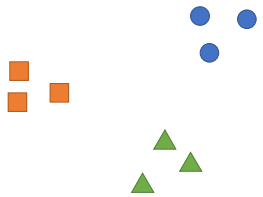
68

Binary-class Classification (Recap)



69

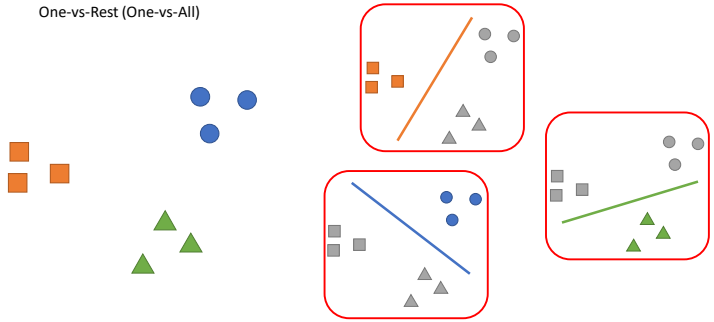
Multi-class Classification with Multiple Binary Classifiers



70

Multi-class Classification with Multiple Binary Classifiers

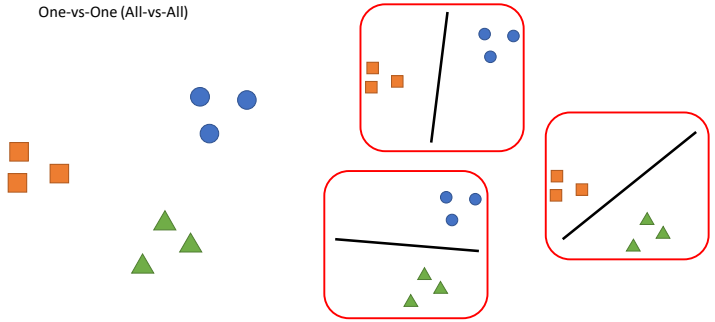
One-vs-Rest (One-vs-All)



71

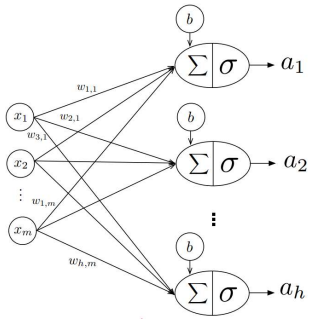
Multi-class Classification with Multiple Binary Classifiers

One-vs-One (All-vs-All)



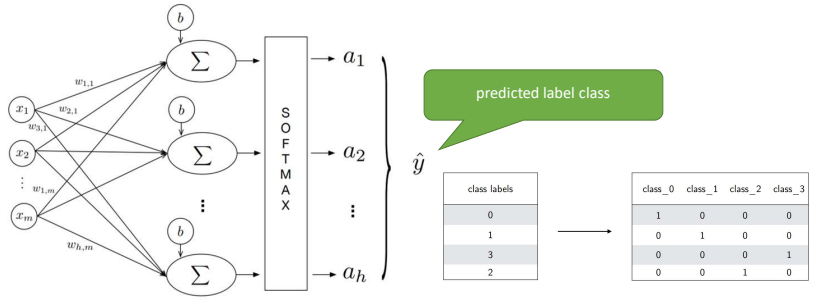
72

Another approach could be…



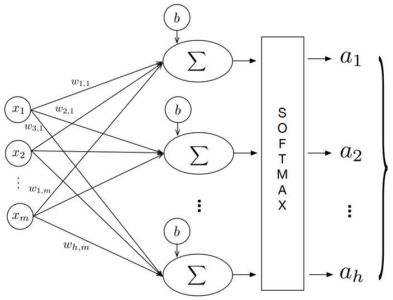
73

Multinomial Logistic Regression / Softmax Regression



74

Multinomial Logistic Regression / Softmax Regression



$$P\left(y^{[i]} = t \middle| z_t^{[i]}\right) = \sigma_{softmax}\left(z_t^{[i]}\right) = \frac{e^{z_t^{[i]}}}{\sum_{j=1}^{h} e^{z_j^{[i]}}},$$

$$t \in \{1, 2, \ldots, h\}$$

Loss function (Multi-category) Cross Entropy for h different class labels

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{j=1}^{h} -y_j^{[i]} \log(a_j^{[i]})$$

75

Logistic Regression vs Softmax Regression

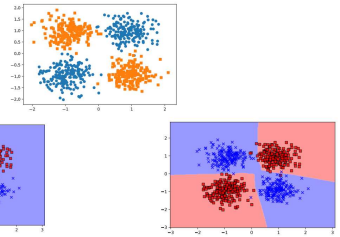
	Logistic Regression	Softmax Regression
Activation function : σ	Sigmoid : $\sigma_{sigmoid} = \frac{e^z}{1+e^z}$	Softmax : $\sigma_{softmax} = \frac{e^{z_t}}{\sum_{j=1}^{h} e^{z_j}}$
Loss function : $\mathcal{L}$	Binary cross entropy : $\mathcal{L} = -\sum_{i=1}^{n} y^{[i]} \log\left(\sigma(z^{[i]})\right) + \left(1 - y^{[i]}\right)\log(1 - \sigma(z^{[i]}))$	(Multi-category) Cross Entropy : $\mathcal{L} = \sum_{i=1}^{n}\sum_{j=1}^{h} -y_j^{[i]} \log(a_j^{[i]})$

76

Multi-layer Perceptron : MLP

- 1967 Marvin L. Minsky and Seymour A. Papert [4] showed that a single-layer perceptron can't implement XOR.
- Neurons, a dead end?

  Start of the first "AI Winter"

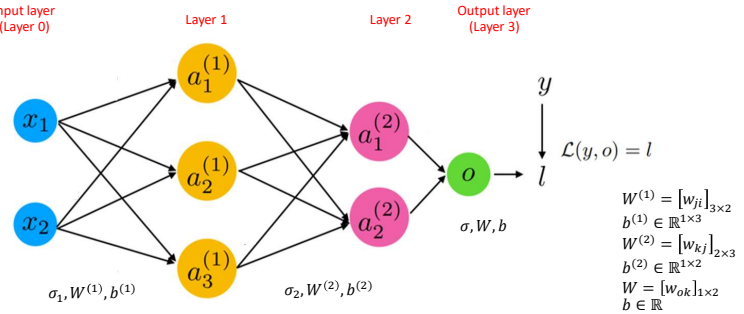


[4] Minski, M. L., & Papert, S. A. (1969). Perceptrons: an introduction to computational geometry. MA: MIT Press, Cambridge.

77

Multi-layer Perceptron : MLP

#(input node) = 2
#(output node) = 1
#(hidden layer) = 2

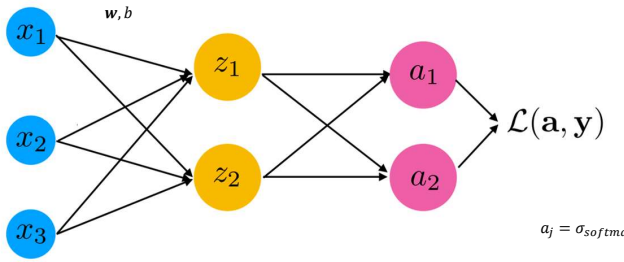


Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

78

Softmax Regression structure (recap)

#(input node) = 3
#(output node) = 2
#(hidden layer) = 0



Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

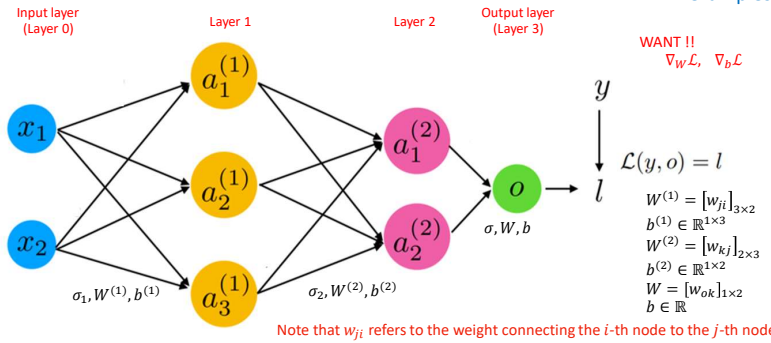
79

Multi-layer Perceptron : MLP

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{n=1}^{N}\left(y^{(n)} - o^{(n)}\right)^2$$

N examples

WANT !!
$\nabla_W \mathcal{L}$, $\nabla_b \mathcal{L}$

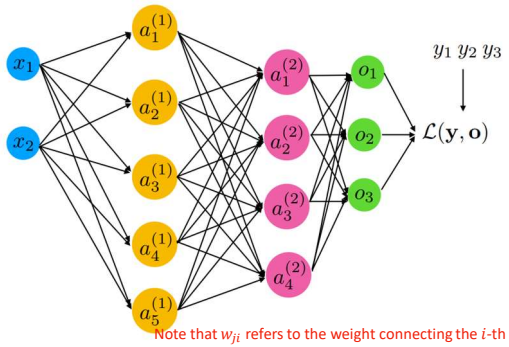


Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

80

Multi-layer Perceptron : MLP

#(input node) = ?
#(output node) = ?
#(hidden layer) = ?

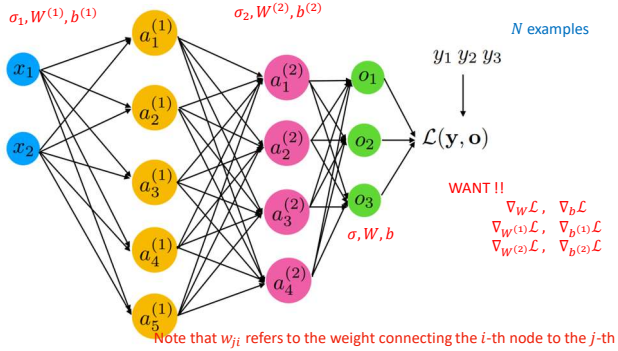


Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

81

MLP x Backpropagation

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{i=1}^{3}\left(y_i^{(n)} - o_i^{(n)}\right)^2$$



WANT !!

$\nabla_W \mathcal{L}$, $\nabla_b \mathcal{L}$
$\nabla_{W^{(1)}} \mathcal{L}$, $\nabla_{b^{(1)}} \mathcal{L}$
$\nabla_{W^{(2)}} \mathcal{L}$, $\nabla_{b^{(2)}} \mathcal{L}$

#(input node) = 2
#(output node) = 3
#(hidden layer) = 2

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

82

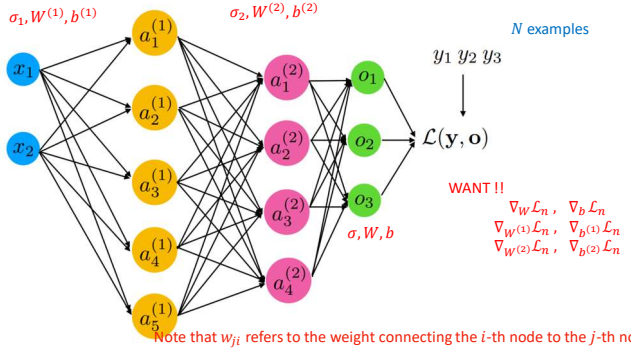
MLP x Backpropagation

- Let $\mathcal{D} = \left\{\left(x^{[i]}, y^{[i]}\right) \in \mathbb{R}^D \times \mathbb{R}^K : i \in \{1, 2, \ldots, N\}\right\}$ be a training set
- For N training examples : $\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right)^2$
- For each n-th training example : $\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right)^2$
- Note that $\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$

83

MLP x Backpropagation

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$$



WANT !!

$\nabla_W \mathcal{L}_n$, $\nabla_b \mathcal{L}_n$
$\nabla_{W^{(1)}} \mathcal{L}_n$, $\nabla_{b^{(1)}} \mathcal{L}_n$
$\nabla_{W^{(2)}} \mathcal{L}_n$, $\nabla_{b^{(2)}} \mathcal{L}_n$

#(input node) = 2
#(output node) = 3
#(hidden layer) = 2

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

84

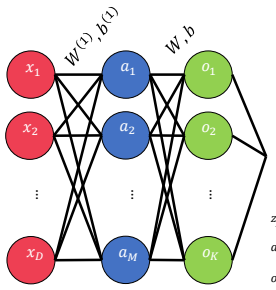
MLP x Backpropagation (Simple example)

#(training example) = 1
#(input node) = D
#(output node) = K
#(hidden layer) = 1
#(hidden node) = M

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

85

MLP x Backpropagation (Simple example)



$y_1, y_2, \dots, y_K$

$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o})$

$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$

$a_j = h(z_j)$

$o_k = \sum_{j=1}^{M} w_{kj} a_j + b_k$

$\sigma_1 = h, \sigma = \text{linear}$

$W^{(1)} = [w_{ji}]_{M \times D}, \quad b^{(1)} \in \mathbb{R}^{1 \times M}$

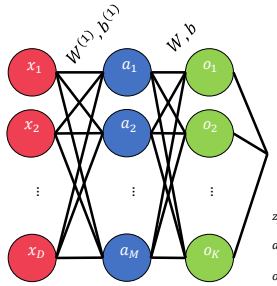
$W = [w_{kj}]_{K \times M}, \quad b \in \mathbb{R}^{1 \times K}$

#(training example) = 1
#(input node) = D
#(output node) = K
#(hidden layer) = 1
#(hidden node) = M

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

86

MLP x Backpropagation (Simple example)



$\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2} \sum_{k=1}^{K} \left(y_k^{[n]} - o_k^{[n]} \right)^2$

$y_1, y_2, \dots, y_K$

$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$

WANT !!

$\nabla_W \mathcal{L}, \quad \nabla_b \mathcal{L}$

$\nabla_{W^{(1)}} \mathcal{L}, \quad \nabla_{b^{(1)}} \mathcal{L}$

$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$

$a_j = h(z_j)$

$o_k = \sum_{j=1}^{M} w_{kj} a_j + b_k$

$\sigma_1 = h, \sigma = \text{linear}$

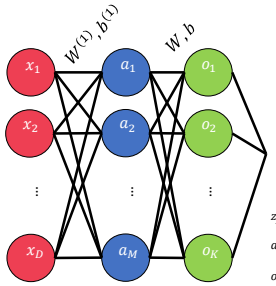
$W^{(1)} = [w_{ji}]_{M \times D}, \quad b^{(1)} \in \mathbb{R}^{1 \times M}$

$W = [w_{kj}]_{K \times M}, \quad b \in \mathbb{R}^{1 \times K}$

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

87

MLP x Backpropagation (Simple example)



$\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2} \sum_{k=1}^{K} \left(y_k^{[n]} - o_k^{[n]} \right)^2$

$y_1, y_2, \dots, y_K$

$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N} \mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$

WANT !!

$\nabla_W \mathcal{L}_n, \quad \nabla_b \mathcal{L}_n$

$\nabla_{W^{(1)}} \mathcal{L}_n, \quad \nabla_{b^{(1)}} \mathcal{L}_n$

$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$

$a_j = h(z_j)$

$o_k = \sum_{j=1}^{M} w_{kj} a_j + b_k$

$\sigma_1 = h, \sigma = \text{linear}$

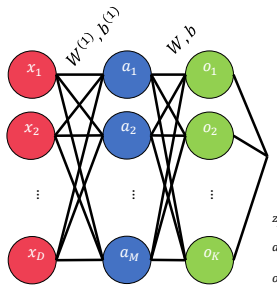
$W^{(1)} = [w_{ji}]_{M \times D}, \quad b^{(1)} \in \mathbb{R}^{1 \times M}$

$W = [w_{kj}]_{K \times M}, \quad b \in \mathbb{R}^{1 \times K}$

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

88

MLP x Backpropagation (Simple example)



$\sigma_1 = h, \sigma = \text{linear}$

$W^{(1)} = [w_{ji}]_{M\times D}, \quad b^{(1)} \in \mathbb{R}^{1\times M}$

$W = [w_{kj}]_{K\times M}, \quad b \in \mathbb{R}^{1\times K}$

$$\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right)^2$$

$y_1, y_2, \dots, y_K$

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$$

$$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$$
$$a_j = h(z_j)$$
$$o_k = \sum_{j=1}^{M} w_{kj} a_j + b_k$$

WANT !!

$$\nabla_W \mathcal{L}_n = \left[\frac{\partial \mathcal{L}_n}{\partial w_{1,1}}, \dots, \frac{\partial \mathcal{L}_n}{\partial w_{kj}}\right]^T,$$

$$\nabla_b \mathcal{L}_n = \left[\frac{\partial \mathcal{L}_n}{\partial b_1}, \dots, \frac{\partial \mathcal{L}_n}{\partial b_K}\right]^T,$$

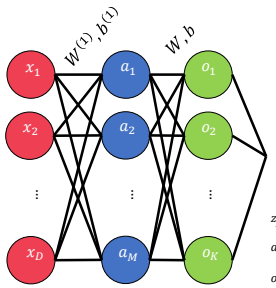
$$\nabla_{W^{(1)}} \mathcal{L}_n = \left[\frac{\partial \mathcal{L}_n}{\partial w_{1,1}^{(1)}}, \dots, \frac{\partial \mathcal{L}_n}{\partial w_{ji}^{(1)}}\right]^T,$$

$$\nabla_{b^{(1)}} \mathcal{L}_n = \left[\frac{\partial \mathcal{L}_n}{\partial b_1^{(1)}}, \dots, \frac{\partial \mathcal{L}_n}{\partial b_M^{(1)}}\right]^T$$

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

89

MLP x Backpropagation (Simple example)



$\sigma_1 = h, \sigma = \text{linear}$

$W^{(1)} = [w_{ji}]_{M\times D}, \quad b^{(1)} \in \mathbb{R}^{1\times M}$

$W = [w_{kj}]_{K\times M}, \quad b \in \mathbb{R}^{1\times K}$

$$\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right)^2$$

$y_1, y_2, \dots, y_K$

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$$

$$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$$
$$a_j = h(z_j)$$
$$o_k = \sum_{j=1}^{M} w_{kj} a_j + b_k$$

WANT !!

$$\frac{\partial \mathcal{L}_n}{\partial w_{kj}} = ???$$

$$\frac{\partial \mathcal{L}_n}{\partial b_k} = ???$$

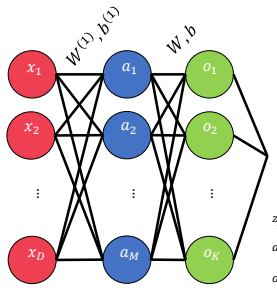
$$\frac{\partial \mathcal{L}_n}{\partial w_{ji}^{(1)}} = ???$$

$$\frac{\partial \mathcal{L}_n}{\partial b_m^{(1)}} = ???$$

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

90

MLP x Backpropagation (Simple example)



$\sigma_1 = h, \sigma = \text{linear}$

$W^{(1)} = [w_{ji}]_{M\times D}, \quad b^{(1)} \in \mathbb{R}^{1\times M}$

$W = [w_{kj}]_{K\times M}, \quad b \in \mathbb{R}^{1\times K}$

$$\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o}) = \frac{1}{2}\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right)^2$$

$y_1, y_2, \dots, y_K$

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{o}) = \sum_{n=1}^{N}\mathcal{L}_n(\boldsymbol{y}, \boldsymbol{o})$$

$$z_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}$$
$$a_j = h(z_j)$$
$$o_k = \sum_{j=1}^{M} w_{kj} a_j + b_k$$

WANT !!

$$\frac{\partial \mathcal{L}_n}{\partial w_{kj}} = -\left(y_k^{[n]} - o_k^{[n]}\right) a_j$$

$$\frac{\partial \mathcal{L}_n}{\partial b_k} = -\left(y_k^{[n]} - o_k^{[n]}\right)$$

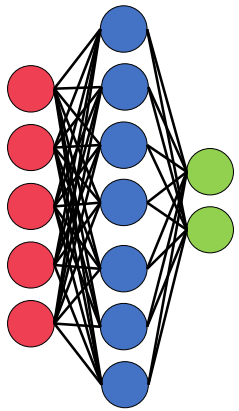
$$\frac{\partial \mathcal{L}_n}{\partial w_{ji}^{(1)}} = -\left(h'(z_j)\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right) w_{kj}\right) x_i$$

$$\frac{\partial \mathcal{L}_n}{\partial b_j^{(1)}} = -\left(h'(z_j)\sum_{k=1}^{K}\left(y_k^{[n]} - o_k^{[n]}\right) w_{kj}\right)$$

Note that w_{ji} refers to the weight connecting the i-th node to the j-th node.

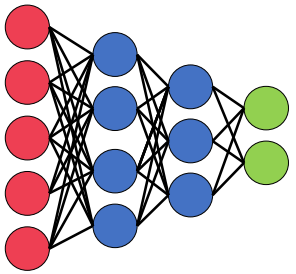
91

White vs Deep Architectures ?



Parameters $= (7 \times 5) + (2 \times 7) + 2 = 51$

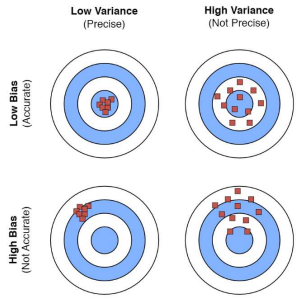
Breadth vs Depth

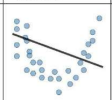
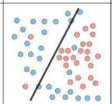
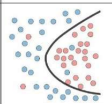
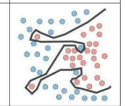


Parameters $= (4 \times 5) + (3 \times 4) + (2 \times 3) + 3 = 41$

92

Underfitting & Overfitting

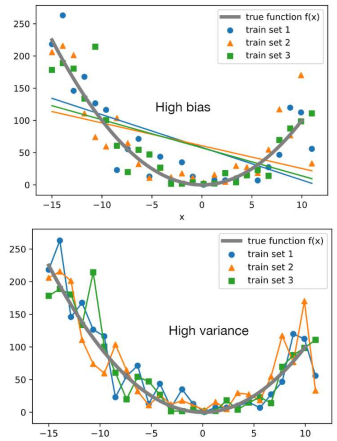


	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

93

Bias-Variance Decomposition

- Definition :
  - $Bias_\theta[\hat{\theta}] = E[\hat{\theta}] - \theta$
  - $Var_\theta[\hat{\theta}] = E[\hat{\theta}^2] - (E[\hat{\theta}])^2$
    $= E\left[(E[\hat{\theta}] - \hat{\theta})^2\right]$

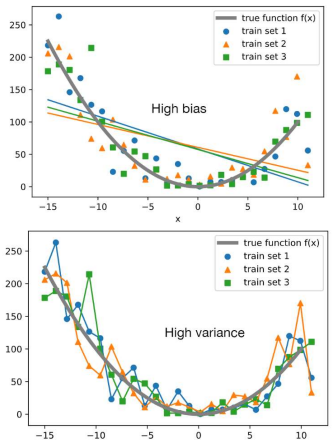


94

Bias-Variance Decomposition

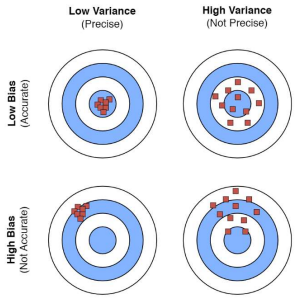
- Definition :
  - $Bias_\theta[\hat{\theta}] = E[\hat{\theta}] - \theta$
  - $Var_\theta[\hat{\theta}] = E[\hat{\theta}^2] - (E[\hat{\theta}])^2$
    $= E\left[(E[\hat{\theta}] - \hat{\theta})^2\right]$

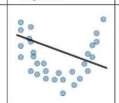
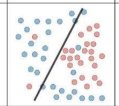
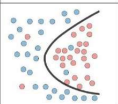
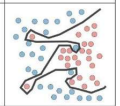
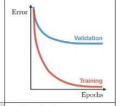
- [Example] Squared loss :
  - $E[(y - \hat{y})^2] = (E[\hat{\theta}] - \theta)^2 + E\left[(E[\hat{\theta}] - \hat{\theta})^2\right]$
    $= (Bias_\theta[\hat{\theta}])^2 + Var_\theta[\hat{\theta}]$



95

Underfitting & Overfitting

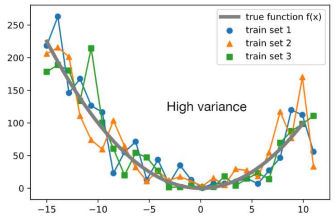


	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

96

How to reduce overfitting

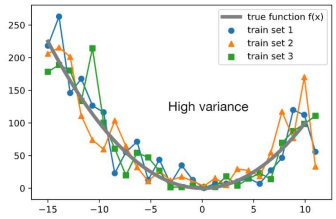
- Early stopping
- Regularization
- Dropout



97

How to reduce overfitting

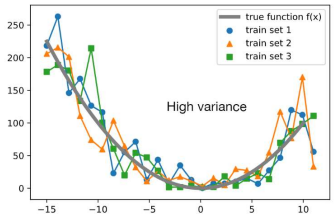
- Early stopping
- Regularization
- Dropout essentially decreases the size of the no. of parameters to be accounted for during the process of training/learning



98

How to reduce overfitting

- Early stopping
- Regularization
- Dropout



99

How to reduce overfitting

- Early stopping
- Regularization
- Dropout

- L_1-regularization $\Rightarrow$ LASSO

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(y^{[i]}, o^{[i]}\right) + \lambda\|\boldsymbol{w}\|_1$$

- L_2-regularization $\Rightarrow$ Ridge regression (Thikonov regularization)

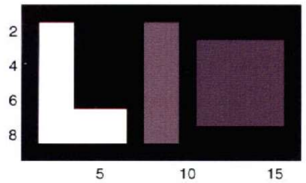
$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(y^{[i]}, o^{[i]}\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- $\alpha L_1 + (1-\alpha)L_2$-regularization $\Rightarrow$ Elastic-net

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}\left(y^{[i]}, o^{[i]}\right) + \lambda\left[\alpha\|\boldsymbol{w}\|_1 + (1-\alpha)\frac{1}{2}\|\boldsymbol{w}\|_2^2\right]$$

100

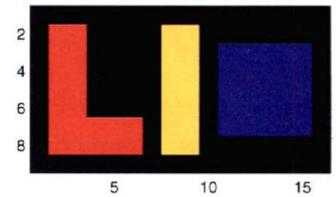
Computer can see images as …



$$\begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&8&8&0&0&0&0&4&4&0&0&0&0&0&0&0\\
0&8&8&0&0&0&0&4&4&0&3&3&3&3&3&0\\
0&8&8&0&0&0&0&4&4&0&3&3&3&3&3&0\\
0&8&8&0&0&0&0&4&4&0&3&3&3&3&3&0\\
0&8&8&0&0&0&0&4&4&0&3&3&3&3&3&0\\
0&8&8&8&8&8&0&4&4&0&3&3&3&3&3&0\\
0&8&8&8&8&8&0&4&4&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}.$$

101

Computer can see images as …



$$X(:,:,1) = \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&1&1&1&1&1&0&1&1&0&0&0&0&0&0&0\\
0&1&1&1&1&1&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix},$$

$$X(:,:,2) = \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix},$$

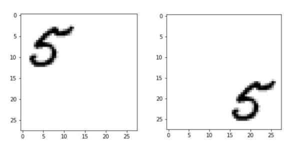
$$X(:,:,3) = \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix},$$

102

Why image classification is Hard

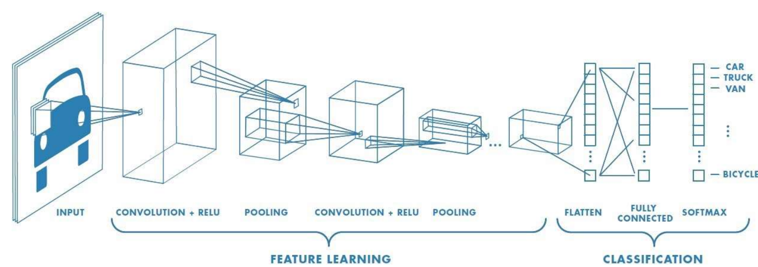
Image Source: twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551

Image Source: https://www.123rf.com/photo_76714328_side-view-of-tabby-cat-face-over-white.html



103

Convolution Neural Networks : CNNs

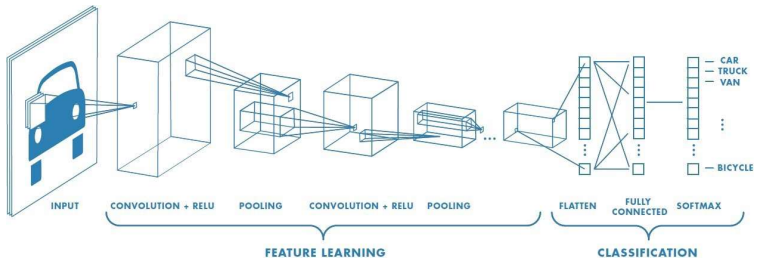


104

ImageNet Large Scale Visual Recognition Challenge : ILSVRC

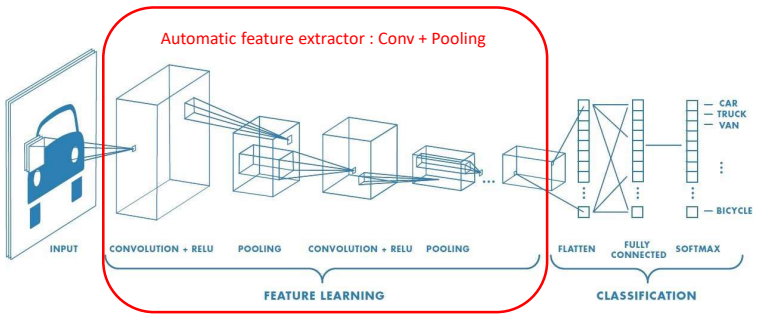
105

Convolution Neural Networks : CNNs



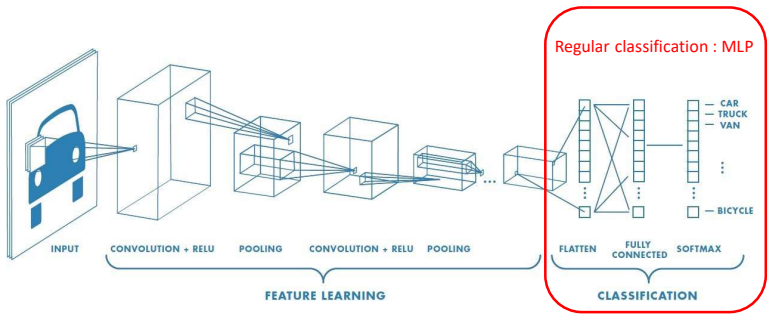
106

Convolution Neural Networks : CNNs



107

Convolution Neural Networks : CNNs



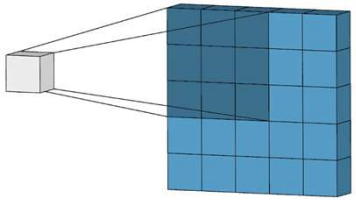
108

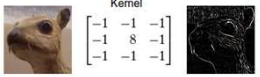
CNNs in 4 steps

- Convolution
- Stride/Padding
- Pooling
- Full connected networks

CNNs in 4 steps

- Convolution
- Stride/Padding
- Pooling
- Full connected networks



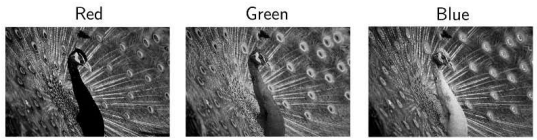


$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$y_j = \sum_{i=0}^{8} w_i x_k$$

CNNs in 4 steps

- Convolution
- Stride/Padding
- Pooling
- Full connected networks

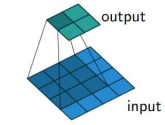


CNNs in 4 steps

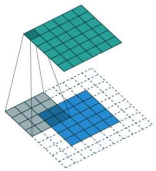
- Convolution
- Stride/Padding
- Pooling
- Full connected networks

CNNs in 4 steps

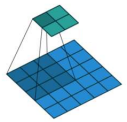
- Convolution
- Stride/Padding
- Pooling
- Full connected networks



No padding, stride=1



padding=2, stride=1

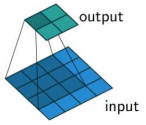


No padding, stride=2

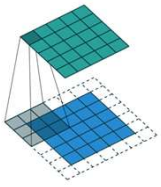
113

CNNs in 4 steps

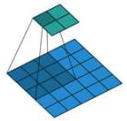
- Convolution
- Stride/Padding
- Pooling
- Full connected networks



No padding, stride=1



padding=2, stride=1



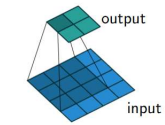
No padding, stride=2

114

CNNs in 4 steps

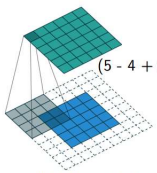
- Convolution
- Stride/Padding
- Pooling
- Full connected networks

$(4 - 3 + 2*0)/1 + 1 = 2$



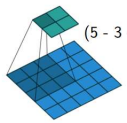
No padding, stride=1

$(5 - 4 + 2*2)/1 + 1 = 6$



padding=2, stride=1

$(5 - 3 + 2*0)/2 + 1 = 2$



No padding, stride=2

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

o := output width
i := input width
p := padding pixels per side
k := kernel width
s := stride

115

CNNs in 4 steps

- Convolution
- Stride/Padding
- Pooling
- Full connected networks

Feature Map

6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

Max Pooling

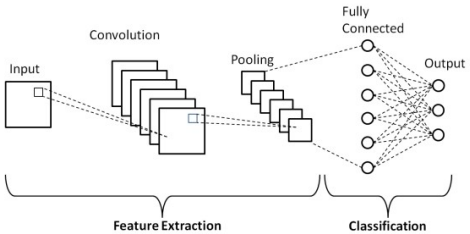
Average Pooling

Sum Pooling

116

CNNs in 4 steps

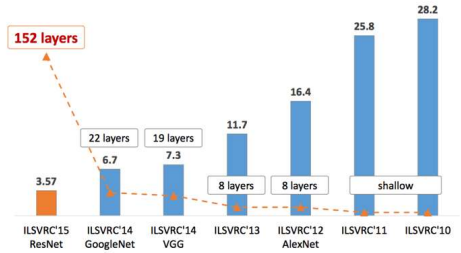
- Convolution
- Stride/Padding
- Pooling
- Full connected networks



117

Various Types of CNNs

- LeNet
- AlexNet
- VGG
- GoogleNet/Inception
- ResNets



Source : https://towardsdatascience.com/various-types-of-convolutional-neural-network-8b00c9a08a1b

118

Various Types of CNNs

- LeNet
- AlexNet
- VGG
- GoogleNet
- ResNets

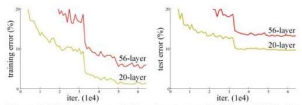


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

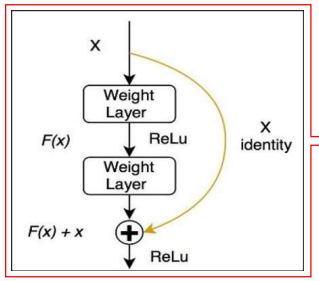
Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [1, 9], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with back-propagation [22].

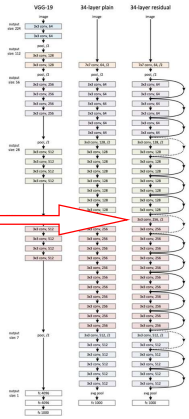
When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more layers to a suitably deep model leads to *higher training error*, as reported in [11, 42] and thoroughly verified by our experiments. Fig. 1 shows a typical example.

Source : https://towardsdatascience.com/various-types-of-convolutional-neural-network-8b00c9a08a1b

119

ResNets : Residual Networks





Paper : https://arxiv.org/abs/1512.03385

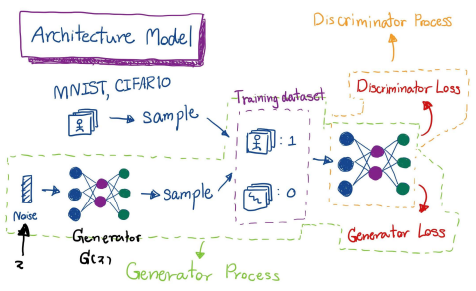
120

To create something from nothing is one of the greatest feelings, ... It's heaven.

By Prince

121

Generative Adversarial Networks : GANs



Right side image created by CycleGAN
Source : https://junyanz.github.io/CycleGAN/

Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

122

Generative Adversarial Networks : GANs

Picture: *These people are not real – they were produced by our generator that allows control over different aspects of the image.*

Source: https://github.com/NVlabs/stylegan

A Style-Based Generator Architecture for Generative Adversarial Networks
Tero Karras (NVIDIA), Samuli Laine (NVIDIA), Timo Aila (NVIDIA)
https://arxiv.org/abs/1812.04948

123

10 Jun 2014

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie[†], Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair[‡], Aaron Courville, Yoshua Bengio[‡]
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates

Paper : https://arxiv.org/pdf/1406.2661.pdf

124

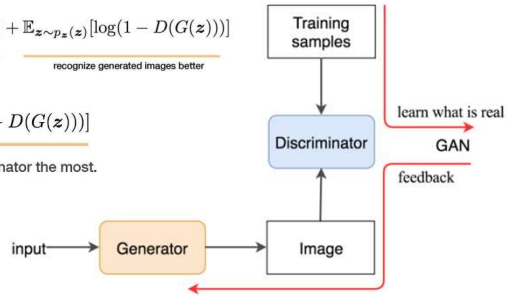
How GANs works?

$$\max_D V(D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

recognize real images better | recognize generated images better

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Optimize G that can fool the discriminator the most.



125

Generative Adversarial Networks : GANs

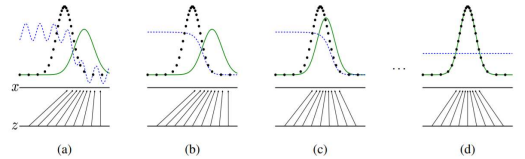


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

126

Generative Adversarial Networks : GANs

GANs is based on the zero-sum non-cooperative game. In short, if one wins the other loses. A zero-sum game is also called minimax. Your opponent wants to maximize its actions and your actions are to minimize them. In game theory, the GAN model converges when the discriminator and the generator reach a Nash equilibrium. This is the optimal point for the minimax equation below.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Zero-Sum Games

Zero-sum game: Rock-Paper-Scissor

P1 \ P2	Rock	Paper	Scissor
Rock	(0 , 0)	(-1 , 1)	(1 , -1)
Paper	(1 , -1)	(0 , 0)	(-1 , 1)
Scissor	(-1 , 1)	(1 , -1)	(0 , 0)

127

What's Nash Equilibrium ?

In the game theory, the Nash Equilibrium is reached when no player will change its strategy after considering all possible strategy of opponents. i.e. in the Nash equilibrium, no one will change its decision even after we will all the player strategy to everyone. A game can have 0, 1 or multiple Nash Equilibria.

Non-zero-sum game: Prisoner's dilemma

Sam \ John	Stay Silent	Betray
Stay Silent	(-1 , -1)	(-3 , 0)
Betray	(0 , -3)	(-2 , -2)

(-2,-2) is Nash equilibrium!

128

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

129

Generative Adversarial Networks : GANs

NIPS 2016 Tutorial:
Generative Adversarial Networks

Ian Goodfellow

OpenAI, ian@openai.com

Abstract

This report summarizes the tutorial presented by the author at NIPS 2016 on *generative adversarial networks* (GANs). The tutorial describes: (1) Why generative modeling is a topic worth studying, (2) how generative models work, and how GANs compare to other generative models, (3) the details of how GANs work, (4) research frontiers in GANs, and (5) state-of-the-art image models that combine GANs with other methods. Finally, the tutorial contains three exercises for readers to complete, and the solutions to these exercises.

Paper : https://arxiv.org/pdf/1701.00160.pdf

130

Training GANs is so HARD!

Many GAN models suffer the following major problems:

- **Non-convergence**: the model parameters oscillate, destabilize and never converge,
- **Mode collapse**: the generator collapses which produces limited varieties of samples,
- **Diminished gradient**: the discriminator gets too successful that the generator gradient vanishes and learns nothing,
- Unbalance between the generator and discriminator causing overfitting, &
- Highly sensitive to the hyperparameter selections.

131

Training GANs is so HARD!

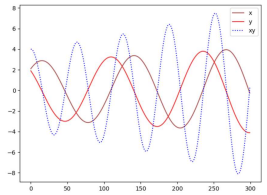
Many GAN models suffer the following major problems:

- **Non-convergence**: the model parameters oscillate, destabilize and never converge,

$$\min_{\mathbf{B}} \max_{\mathbf{A}} V(D, G) = xy$$

$$\Delta x = \alpha \frac{\partial(xy)}{\partial(x)}$$

$$\Delta y = -\alpha \frac{\partial(xy)}{\partial(y)}$$

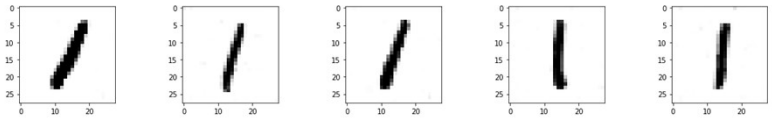


Cost functions may not converge using gradient descent in a minimax game.

132

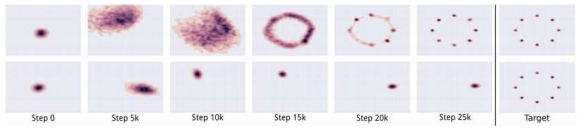
Training GANs is so HARD!

- **Mode collapse**: the generator collapses which produces limited varieties of samples,



133

Training GANs is so HARD!

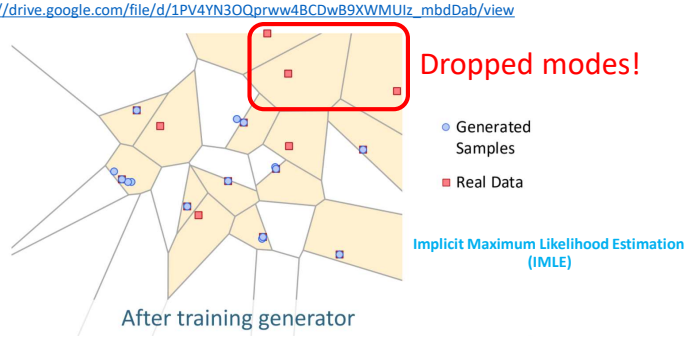


Unrolled GANs are able to fit all of the modes of a mixture of Gaussians in a two-dimensional space.
Paper Unrolled GANs : https://arxiv.org/pdf/1611.02163.pdf

134

Why Mode Collapse Happens

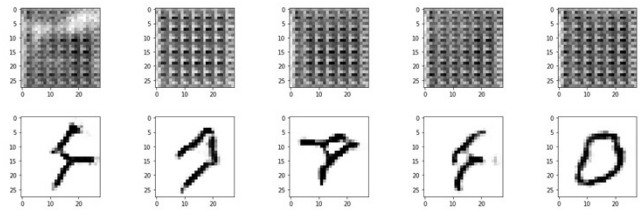
Source : https://drive.google.com/file/d/1PV4YN3OQprww4BCDwB9XWMUIz_mbdDab/view



135

Training GANs is so HARD!

- **Diminished gradient**: the discriminator gets too successful that the generator gradient vanishes and learns nothing,



136

In other words, D and G play the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]. \quad (1)$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right). \qquad \nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

137

Training GANs is so HARD!

Many GAN models suffer the following major problems:

- **Non-convergence**: the model parameters oscillate, destabilize and never converge,
- **Mode collapse**: the generator collapses which produces limited varieties of samples,
- **Diminished gradient**: the discriminator gets too successful that the generator gradient vanishes and learns nothing,
- Unbalance between the generator and discriminator causing overfitting, &
- Highly sensitive to the hyperparameter selections.

138

Some improvements of GANs

- Improving the Network design
  - DCGAN (Deep convolutional generative adversarial networks), pop!
  - CGAN, InfoGAN (using labels to improve GAN),
  - SGAN (Stacked GAN),
  - StyleGAN, etc.

139

Some improvements of GANs

- Improving the cost function

Finally, we did not find evidence that any of the tested algorithms consistently outperforms the original one.

A 2017 Google Brain paper "Are GANs Created Equal?"
Paper : https://arxiv.org/pdf/1711.10337.pdf

Name	Value Function
GAN	$L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$ $L_G^{GAN} = E[\log(D(G(z)))]$
LSGAN	$L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$ $L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$
WGAN	$L_D^{WGAN} = E[D(x)] - E[D(G(z))]$ $L_G^{WGAN} = E[D(G(z))]$ $W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$
WGAN_GP	$L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[(\\|\nabla D(\alpha x - (1 - \alpha G(z)))\\| - 1)^2]$ $L_G^{WGAN\_GP} = L_G^{WGAN}$
DRAGAN	$L_D^{DRAGAN} = L_D^{GAN} + \lambda E[(\\|\nabla D(\alpha x - (1 - \alpha x_p))\\| - 1)^2]$ $L_G^{DRAGAN} = L_G^{GAN}$
CGAN	$L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$ $L_G^{CGAN} = E[\log(D(G(z), c))]$
infoGAN	$L_{D,Q}^{infoGAN} = L_D^{GAN} - \lambda L_I(c, c')$ $L_G^{infoGAN} = L_G^{GAN} - \lambda L_I(c, c')$
ACGAN	$L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c\|x)] + E[P(class = c\|G(z))]$ $L_G^{ACGAN} = L_G^{GAN} + E[P(class = c\|G(z))]$
EBGAN	$L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$ $L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$
BEGAN	$L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$ $L_G^{BEGAN} = D_{AE}(G(z))$ $k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$

140

Improved Techniques for Training GANs

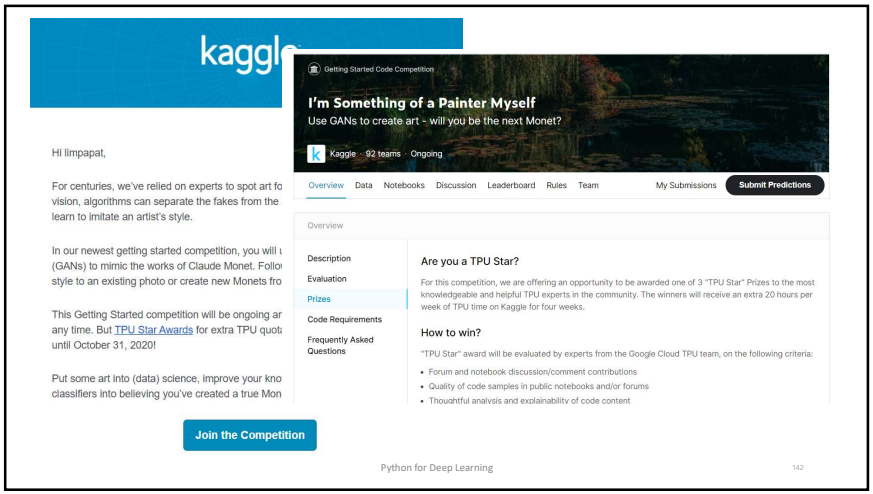
Tim Salimans tim@openai.com
Ian Goodfellow ian@openai.com
Wojciech Zaremba woj@openai.com
Vicki Cheung vicki@openai.com
Alec Radford alec@openai.com
Xi Chen peter@openai.com

Abstract

We present a variety of new architectural features and training procedures that we apply to the generative adversarial networks (GANs) framework. Using our new techniques, we achieve state-of-the-art results in semi-supervised classification on MNIST, CIFAR-10 and SVHN. The generated images are of high quality as con-

Paper : https://arxiv.org/pdf/1606.03498.pdf

141



142

Generative Adversarial Networks : GANs

Source : https://www.youtube.com/watch?v=OGGjXG562WU

The GANs Zoo : https://github.com/hindupuravinash/the-gan-zoo

GANs Applications : https://github.com/nashory/gans-awesome-applications

143