Lecture 3. Perceptron Learning Algorithm (PLA)

Efficiently find a perfect classifier for linearly seperable dataset.

Notation:

Decision Rule: $w_1x_1 + w_2x_2 + \cdots + w_dx_d \underset{y=-1}{\overset{y=+1}{\gtrless}} b$

$$-b + \sum_{i=1}^{d} w_i x_i \underset{y=-1}{\overset{y=+1}{\gtrless}} 0$$

Let $w_0 = -b$, $x_0 = 1$,

Augumented vectors:

$$\underline{w} = (w_0 = -b, w_1, \cdots, w_d) \in \mathbb{R}^{d+1}$$

$$\underline{x} = (x_0 = 1, x_1, \cdots x_d) \in \mathbb{R}^{d+1}$$

New decision rule: $\underline{w}^T \cdot \underline{x} \underset{y=-1}{\overset{y=+1}{\gtrless}} 0$

$$y = h_w(\underline{x}) \triangleq \text{sign}(\underline{w}^T\underline{x}). \quad \text{sign}(t) = \begin{cases} +1, & t>0 \\ -1, & t<0 \end{cases}$$

Suppose training set $\mathcal{D}$ is linearly seperable, find $\underline{w} \in \mathbb{R}^{d+1}$ s.t.

$$E_{in}(\underline{w}) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(y_n \neq h_w(\underline{x}_n)) = 0$$

Input: Training set $\mathcal{D}$ that is linearly seperable.

Output: $\underline{w} \in \mathbb{R}^{d+1}$ that achieves $E_{in}(\underline{w}) = 0$.

Initialization: arbitrary, eg. $\underline{w} = \underline{0}$.

Step 1: Check if $E_{in}(\underline{w}) = 0$.

If yes, stop & output current $\underline{w}$.

Step 2: Let $(\underline{x}_n, y_n)$ be a miss-classified example.

(including points on boundary).

If $y_n = +1$, $\underline{w} \in \underline{w} + \underline{x}_n$

If $y_n = -1$, $\underline{w} \in \underline{w} - \underline{x}_n$

Go to step 1

Lec 4

Binary linear classification

Decision Rule: $y = \text{sign}(\underline{w}^T\underline{x})$.

augmented: $\underline{w} = (w_0, w_1, \cdots, w_d)$

$\underline{x} = (x_0 = 1, x_1, \cdots, x_d)$

Perceptron learning Algorithm

Update step: Pick mis-classified example $(\underline{x}_n, y_n)$, Note: $y_n = +1$ or -1

$$\underline{w} \leftarrow \underline{w} + y_n\underline{x}_n$$

Intuition

Possible real & predicted result.

y_n	$w^T x_n$ (pred.)	is correct prediction?	$y_n w^T x_n$
+1	+	✓	+
+1	−	X	−
−1	+	X	−
−1	−	✓	+
±1	0	X	0

If (x_n, y_n) is correctly classified, $y_n w^T x_n > 0$.

If (x_n, y_n) is mis-classified, $y_n w^T x_n < 0$.

Updating: Suppose (x_n, y_n) is mis-classified.

$$w_{new} = w + y_n x_n,$$

$$y_n w_{new}^T x_n = y_n (w + y_n x_n)^T x_n$$

$$= y_n \cdot w^T \cdot x_n + y_n^2 \|x_n\|^2$$

y_n^2: always +1; $\|x_n\|^2 \geq 0$ (since we have a bias)

$$> y_n \cdot w^T \cdot x_n$$

Strict improvement for x_n!

But it could cause new mis-classification for other examples.

Theorem (Rosenblatt, 1957)

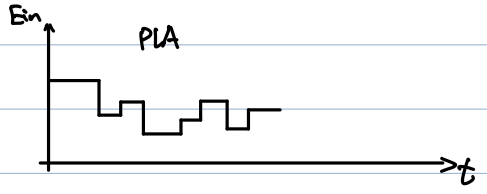
Given a linearly seperable dataset, the PLA terminates in a finite number of steps yielding $E_{in}(w) = 0$.

Proof. Problem 1.3

Remark: The output from the algo. is not unique.

Pocket Algorithm

Extends PLA for dataset that are not linearly unseperable.



Keep the "best" weight vector upto iteration t in the pocket.

Algorithm:

0: Pick time horizon T.

1: Set pocked weight vector $\hat{w}$ to $w(0)$ of PLA.

2: For $t = 0, 1, 2 \ldots, T-1$ do:

3: Run PLA for one update to obtain $w(t+1)$.

4: Evaluate $E_{in}(w(t+1))$

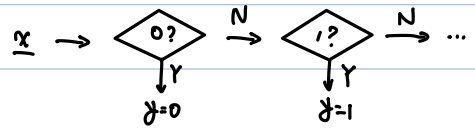
5: If $E_{in}(w(t+1)) < E_{in}(\hat{w})$, set $\hat{w} = w(t+1)$.

6: Return $\hat{w}$.

Heuristic algorithm, performs well in practice.

Note : Multiary classification can be implemented by a sequence of binary classification

e.g. 10 digits



Lecture 5.

Linear Regression

Recall: supervised learning < discrete y_n : classification / continuous y_n : regression

Training set:

$D = \{(\underline{x}_1, y_1), \cdots, (\underline{x}_N, y_N)\}$

$\underline{x}_n \in \mathbb{R}^d$, $y_n \in \mathbb{R}$

Decision Rule: (Hypothesis Set)

$\hat{y} = h(\underline{x})$

$= w_0 + w_1 x_1 + \cdots + w_d x_d$

redefine augmented form : $\underline{w} = (w_0, w_1 \cdots w_d)$

$\underline{x} = (x_0 = 1, x_1 \cdots x_d)$

$\hat{y} = h(\underline{x}) = \underline{w}^T \underline{x}$

Criterion:

$$E_{in}(\underline{w}) = \frac{1}{N} \sum_{n=1}^{N} \overbrace{(y_n - \underline{w}^T \underline{x}_n)^2}^{e_n(\underline{w})}$$

"average squared error"

$e_n(\underline{w})$ is the square error on the n^{th} example.

Goal:

Given D, find $\underline{w} \in \mathbb{R}^{d+1}$ to minimize $E_{in}(\underline{w})$

Example: x = advertising cost in one week ($d=1$)

y = sales in one week

Historical data

$D = \{(x_1, y_1), \cdots, (x_N, y_N)\}$

Fit a linear model

$y = w_0 + w_1 x$

Refine model: $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \text{adv\$ on TV} \\ \cdots \text{ on radio} \\ \cdots \text{ on newspaper} \end{bmatrix}$

($d=3$) $y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$

larger w_i $\Rightarrow$ more profitable x_i.

More algebra:

1) Data matrix

$$X = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \vdots \\ \underline{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times (d+1)}$$

2) Target vector (label)

$$\underline{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^N$$

3) Weight vector

$$\underline{w} \in \mathbb{R}^{d+1}$$

4) Model

$$\hat{\underline{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} \underline{w}^T \underline{x}_1 \\ \underline{w}^T \underline{x}_2 \\ \vdots \\ \underline{w}^T \underline{x}_N \end{bmatrix} = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \vdots \\ \underline{x}_N^T \end{bmatrix} \cdot \underline{w}$$

$$\hat{\underline{y}} = X \cdot \underline{w}$$

5) Error:

$$E_{in}(\underline{w}) = \frac{1}{N} \cdot \| \underline{y} - \hat{\underline{y}} \|^2$$

When is $E_{in}(\underline{w}) = 0$, i.e., $\underline{y} = \hat{\underline{y}}$?

$$y_n = \underline{w}^T \cdot \underline{x}_n, \quad n = 1, 2 \cdots, N.$$

# of linear equation: N

# of variable: $d+1$ (w_i)

In practice, $N \gg d$

$\Rightarrow$ No exact solution for the equ. Instead, we minimize $E_{in}(\underline{w})$.

Least Mean Squared Solution.

(to minimize $E_{in}(\underline{w}) = \frac{1}{N} \| \underline{y} - \hat{\underline{y}} \|^2$)

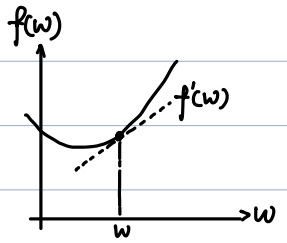
Define $f(\underline{w}) = \| \underline{y} - \hat{\underline{y}} \|^2$

$$= \| X\underline{w} - \underline{y} \|^2$$

$$= \sum_{n=1}^{N} \underbrace{(\underline{w}^T \underline{x}_n - y_n)^2}_{e_n(\underline{w})}$$

Def Gradient of $f(\underline{w})$

$$\nabla f(\underline{w}) = \begin{bmatrix} \partial f / \partial w_0 \\ \partial f / \partial w_1 \\ \vdots \\ \partial f / \partial w_d \end{bmatrix}$$



It points in the direction of steepest increase.

Claim: ① $\nabla f(\underline{w}) = 2X^T(X\underline{w} - \underline{y})$

② Least squared solution $\underline{w}_{LS}$ is s.t.

$$\nabla f(\underline{w}_{LS}) = 0$$

$$2X^T(X\underline{w} - \underline{y}) = 0$$

$$X^T X \underline{w} = X^T \underline{y}$$

Assumption:

The $d+1$ columns of X are linearly independent.

$\equiv$ there exist at least $d+1$ rows of X that are linearly independent.

(rows of X ↑ data samples)

Then $\text{rank}(X) = d+1$.

$\iff X^T X$ is invertible

$$\boxed{\underline{w}_{LS} = (X^T X)^{-1} X^T \underline{y}} = X^\dagger \cdot \underline{y}$$

$X^\dagger \triangleq (X^T X)^{-1} X^T$ is called the pseudo-inverse of X.

$X^\dagger \cdot X = I$ but $X \cdot X^\dagger \neq I$

Note: ① $\underline{y} = X \underline{w}$

$\underline{w} = ? \ y$

② $\hat{\underline{y}}_{LS} = X \cdot \underline{w}_{LS} = \underbrace{X X^\dagger}_{\text{projection matrix}} \underline{y}$

Lecture 6

Geometric Interpretation of least squares

$$\min_{\underline{w}} \|\underline{y} - X\underline{w}\|^2$$

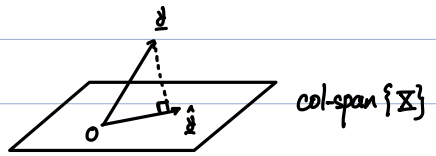
LS solution: $\underline{w}_{LS} = (X^T X)^{-1} X^T \underline{y}$

$\hat{\underline{y}} = X\underline{w}$: linear combination of the columns of X,

$\Rightarrow \hat{\underline{y}}$ is a vector in col-span of $\{X\}$

To minimize $\|\underline{y} - \hat{\underline{y}}\|$ (distance between $\underline{y}$ & $\hat{\underline{y}}$)

The best $\hat{\underline{y}}$ (i.e., $\hat{y}_{LS}$) is the projection of $\underline{y}$ onto col-span $\{X\}$.



$\Rightarrow$ Every column of X is orthogonal to $\underline{y} - \hat{\underline{y}}_{LS}$

$\Rightarrow X^T(\underline{y} - \hat{y}_{LS}) = 0$

$\Rightarrow X^T(\underline{y} - X\underline{w}_{LS}) = 0$

$\Rightarrow X^T X \underline{w}_{LS} = X^T \underline{y} \Rightarrow \underline{w}_{LS} = (X^T X)^{-1} X^T y$ as expected

Regularized Linear Regression / Least Squares

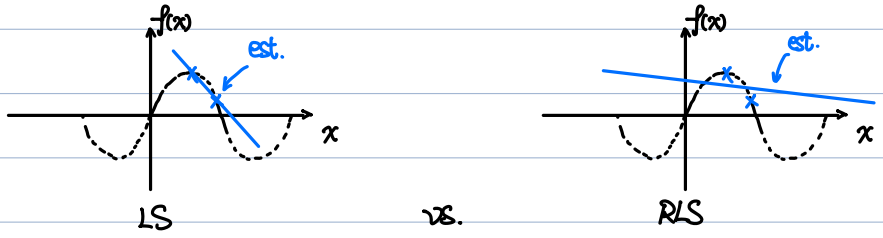
$$\min_{\underline{w}} \|X\underline{w} - \underline{y}\|^2 + \lambda \|\underline{w}\|^2$$

penalize large $\|w\|$

Motivation: avoid "overfitting" $E_{out} - E_{in}$

Textbook: Target function: $f(x) = \sin(\pi x)$

Training Set: 2 points (randomly sampled)



$$f(\underline{w}) = \|X\underline{w} - \underline{y}\|^2 + \lambda\|\underline{w}\|^2$$

$$\frac{\partial f}{\partial w_k} = \frac{\partial}{\partial w_k}\left(\|X\underline{w} - \underline{y}\|^2 + \lambda\sum_{j=0}^{d} w_j^2\right)$$

$$= 2\left[X^T(X\underline{w} - \underline{y})\right]_k + 2\lambda w_k$$

$$= 2\left[X^T(X\underline{w} - \underline{y}) + \lambda\underline{w}\right]_k$$ ← only pick the k^{th} element of the vector

$$\nabla f(\underline{w}) = 2\left[X^T(X\underline{w} - \underline{y}) + \lambda\underline{w}\right]$$

Want $\nabla f(\underline{w}) = \underline{0}$

$$(X^TX + \lambda I)\cdot \underline{w}_{RLS} = X^T\cdot\underline{y}$$

$$\boxed{\underline{w}_{RLS} = (X^TX + \lambda I)^{-1}\cdot X^T\cdot\underline{y}}$$

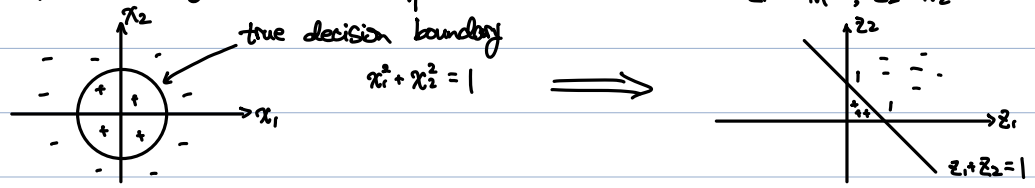
① $\underline{w}_{RLS} = \underline{w}_{LS}$ iff. $\lambda = 0$

② RLS side benefit more numerically stable.

Non-Linear Transformation

Linear function may not be sufficient.

E.g.



Suppose, $h(\underline{z}) = \text{sign}(z_1 + z_2 - 1)$

then $g(\underline{x}) = \text{sign}(x_1^2 + x_2^2 - 1)$

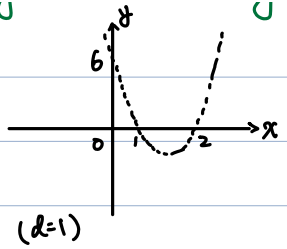
Let $\underline{z} = \Phi(\underline{x})$ be a non-linear transform.

Let $h(\underline{z}) = \underline{w}^T\underline{z}$ be a linear classifier in $\underline{z}$ space.

Then $g(\underline{x}) = h(\Phi(\underline{x}))$ is a non-linear classifier in $\underline{x}$ space.

$\Phi(\cdot)$ is called a feature transform.

Eg. Quadratic Regression



(d=1)

$\underline{z} = (z_0=1, z_1=x, z_2=x^2)$.

$y = \underline{w}^T \cdot \underline{z}$

$= w_0 + w_1 z_1 + w_2 z_2$

$= w_0 + w_1 x + w_2 x^2$

$\underline{z}_1 = (1,0,0)$, $y_1 = 6$

$\underline{z}_2 = (1,1,1)$, $y_2 = 0$

$\underline{z}_3 = (1,2,4)$, $y_3 = 0$

$\Rightarrow \quad Z = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}$, $\underline{y} = \begin{bmatrix} 6 \\ 0 \\ 0 \end{bmatrix}$

$\underline{w}_{LS} = (Z^T Z)^{-1} \cdot Z^T \cdot \underline{y} = Z^{-1} \cdot \underline{y} = \begin{bmatrix} 6 \\ -9 \\ 3 \end{bmatrix}$

$y = 6 - 9 z_1 + 3 z_2$

$= 6 - 9x + 3x^2$.

Lecture 7

Logistic Regression

Recap: Given $\mathcal{D} = \{(\underline{x}_1, y_1), \cdots, (\underline{x}_N, y_N)\}$

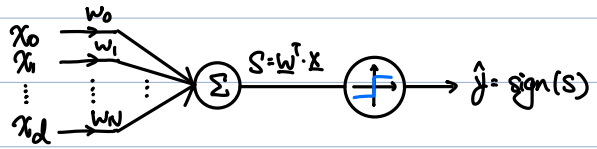
unknown target function $y = f(\underline{x})$,

Hypothesis: $\hat{y} = h(\underline{x})$ where $h \in H$.

Linear classification:

$\underline{x} \in \mathbb{R}^{d+1}$, $y \in \{+1, -1\}$

$\hat{y} = \text{sign}(\underline{w}^T \underline{x})$

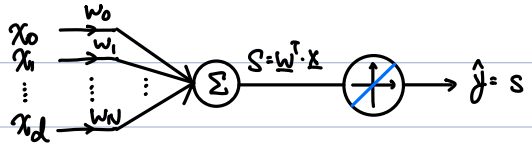


$e(\underline{w}) = 1\{y \neq \text{sign}(S)\}$

Linear Regression

$\underline{x} \in \mathbb{R}^{d+1}$, $y \in \mathbb{R}$,

$\hat{y} = \underline{w}^T \underline{x}$



$e(\underline{w}) = (y - S)^2$

Example:

What if we want randomness in h?

E.g. $\underline{x}$ = average (fat, sugar) in diet,

y = heart attack

Want to say "large $\underline{w}^T\underline{x} \Rightarrow$ more likely that $y=+1$"

Want new target function

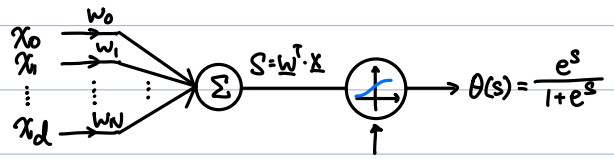
$f_p(\underline{x}) = P_r\{y=+1|\underline{x}\}$

"Soft decision"

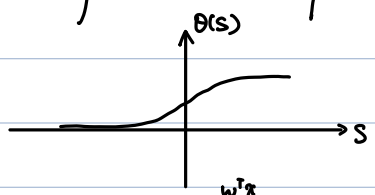
$\Rightarrow$ Solution:

Logistic Regression

$\underline{x} \in \mathbb{R}^{d+1}$, $y \in \{+1, -1\}$



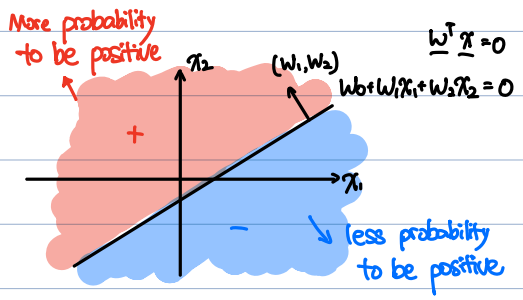
$\theta(s)$: Sigmoid function "S-shaped", tanh $= \frac{e^s - e^{-s}}{e^s + e^{-s}}$



Hypothesis: (guess) $P_r\{y=+1|\underline{x}\} = \theta(s) = \theta(\underline{w}^T\underline{x}) = \frac{e^{\underline{w}^T\underline{x}}}{1+e^{\underline{w}^T\underline{x}}}$

Note: ① Geometry:

($d=2$)



② $P_r\{y=-1|\underline{x}\} = 1 - P_r\{y=+1|\underline{x}\}$

$= 1 - \frac{e^{\underline{w}^T\underline{x}}}{1+e^{\underline{w}^T\underline{x}}}$

$= \frac{1}{1+e^{\underline{w}^T\underline{x}}}$

$= \frac{e^{-\underline{w}^T\underline{x}}}{1+e^{-\underline{w}^T\underline{x}}}$

$= \theta(-\underline{w}^T\underline{x}) = \theta(-s)$

③ Notation:

$$\hat{P}_{\underline{w}}(1|\underline{x}) = \theta(\underline{w}^T\underline{x})$$

estimate for $P_r\{y=+1|\underline{x}\}$

$$\hat{P}_{\underline{w}}(-1|\underline{x}) = \theta(-\underline{w}^T\underline{x})$$

$$\Rightarrow \hat{P}_{\underline{w}}(y|\underline{x}) = \theta(y\underline{w}^T\underline{x}) \quad . \quad y \in \{+1,-1\}$$

$$= \frac{e^{y\underline{w}^T\underline{x}}}{1+e^{y\underline{w}^T\underline{x}}}$$

$$= \frac{1}{1+e^{-y\underline{w}^T\underline{x}}}$$

Error Criterion:

For the n^{th} example in training data,

$$e_n(\underline{w}) = -\log \hat{P}_{\underline{w}}(y_n|\underline{x}_n) \leftarrow \text{"log-loss function"}$$

$$= \log(1+e^{-y\underline{w}^T\underline{x}})$$

Note:

If $\underline{w}^T\underline{x} >> 0$, and $y=+1 \Rightarrow$ loss $= 0$.

$\underline{w}^T\underline{x} << 0$, and $y=-1 \Rightarrow$ loss $= 0$.

Example: $x \in \mathbb{R}^{d+1}$.

Suppose output of logistic regression is

$$\hat{P}_{\underline{w}}(-1|\underline{x}_n) = 0.8$$

$$\hat{P}_{\underline{w}}(+1|\underline{x}_n) = 0.2$$

If $y_n=+1$, then $e_n(\underline{w}) = -\log(0.2) = 1.61$

if $y_n=-1$, then $e_n(\underline{w}) = -\log(0.8) = 0.22$

Note: loss is smaller if $y_n=-1$, because our $\hat{P}_{\underline{w}}$ assigns higher probability for $y_n=-1$.

Suppose $\hat{P}_{\underline{w}}(-1|\underline{x}_n) = 0.999$

$\hat{P}_{\underline{w}}(+1|\underline{x}_n) = 0.001$,

If $y_n=+1$, $e_n(\underline{w}) = -\log(0.001) = 10$

If $y_n=-1$. $e_n(\underline{w}) = -\log(0.999) = 10^{-4}$

Logistic Regression

Given $\mathcal{D} = \{\underline{x}_1, y_1\}, \cdots, \{\underline{x}_N, y_N\}$, find $\underline{w} \in \mathbb{R}^{d+1}$,

to minimize

$$E_{in}(\underline{w}) = \frac{1}{N}\sum_{n=1}^{N} e_n(\underline{w})$$

$$= \frac{1}{N}\sum_{n=1}^{N} \log(1+e^{-y_n\underline{w}^T\underline{x}_n})$$

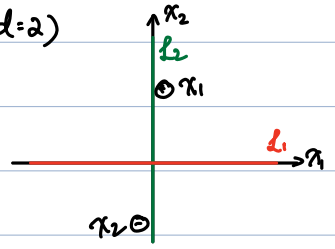
Lecture 8

Why use $e_n(\underline{w}) = -\log \hat{P}_{\underline{w}}(y_n|x_n)$?

$= \log(1+e^{-y_n \underline{w}^T x_n})$

Benefit over linear classification

E.g. ($d=2$)



$N=2$

$\underline{x}_1 = (0.001, 10)$, $y_1 = +1$ $\quad w_1 = (0,1,0)$

$\underline{x}_2 = (-0.001, -10)$, $y_2 = -1$ $\quad w_2 = (0,0,1)$

For linear:

$e_{in}(\underline{w}_1) = 0$

$e_{in}(\underline{w}_2) = 0$

} can't tell which is better

For logistic regression

$E_{in}(\underline{w}_1) = \frac{1}{2}(\log(1+e^{-\underline{w}_1^T \underline{x}_1}) + \log(1+e^{+\underline{w}_1^T \underline{x}_2}))$

$= \frac{1}{2}(\log(1+e^{-0.001}) + \log(1+e^{-0.001}))$

$= 0.693$

$E_{in}(\underline{w}_2) = \frac{1}{2}(\log(1+e^{-\underline{w}_2^T \underline{x}_1}) + \log(1+e^{\underline{w}_2^T \underline{x}_2}))$

$= \frac{1}{2}(\log(1+e^{-10}) + \log(1+e^{-10}))$

$= 5 \times 10^{-5}$

} $\underline{w}_2$ is better

Note: What if $\underline{w}_2 = (0,0,100)$?

Same like $\underline{w}_2$, but loss $E_{in}(\underline{w})$ is lower.

$\Rightarrow$ logistic regression typically requires regulation:

$E_{in}(\underline{w}) + \lambda \|\underline{w}\|^2$

Maximum likelihood Viewpoint

Let $\{(\underline{x}_1, y_1), \cdots (\underline{x}_N, y_N)\}$ be the training dataset

Consider $P(y_1, \cdots y_N | \underline{x}_1, \cdots, \underline{x}_N) \triangleq \Pr\{1^{st} \text{ label} = y_1 \cdots \mid 1^{st} \text{ example} = \underline{x}_1, \cdots\}$

$= \prod_{n=1}^{N} P(y_n|\underline{x}_n)$

by hypothesis

$= \prod_{n=1}^{N} \hat{P}_{\underline{w}}(y_n|\underline{x}_n)$.

Maximum-likelihood approach:

Find $\underline{w} \in \mathbb{R}^{d+1}$ that maximizes $P(y_1 \cdots y_N | \underline{x}_1, \cdots \underline{x}_N)$

$\Leftrightarrow$ maximize $\frac{1}{N} \log \prod_{n=1}^{N} \hat{P}_{\underline{w}}(y_n|\underline{x}_n)$

$$= \frac{1}{N}\sum_{n=1}^{N} \log \hat{P}_{\underline{w}}(y_n|\underline{x}_n)$$

$$\Leftrightarrow \text{minimize } \frac{1}{N}\sum_{n=1}^{N} -\log \hat{P}_{\underline{w}}(y_n|\underline{x}_n)$$

$$= \frac{1}{N}\sum_{n=1}^{N} e_{in}(\underline{w})$$

$$= E_{in}(\underline{w})$$

Cross-Entropy Viewpoint

$$E_{in}(\underline{w}) = -\frac{1}{N}\sum_{n=1}^{N}\left[1(y_n=+1)\log \hat{P}_{\underline{w}}(1|\underline{x}_n) + 1(y_n=-1)\log \hat{P}_{\underline{w}}(-1|\underline{x}_n)\right]$$

Define: Suppose P & Q are two probability distributions over $\mathcal{X} = \{x_1, \cdots x_m\}$.

Eg. Poisson with mean α, $\mathcal{X} = \{0,1,2,\cdots\}$, $P(x) = \frac{\alpha^x}{x!}e^{-\alpha}$

The cross-entropy between P & Q (measurement of distance) is

$$CE(P,Q) = -\sum_{i=1}^{m} P(x_i)\log Q(x_i).$$

For n-th example, consider the distribution

$$P_n = (Pr\{y_n=+1\}, Pr\{y_n=-1\})$$

$$= \begin{cases} (1,0), & \text{if } y_n=1 \\ (0,1), & \text{if } y_n=-1 \end{cases}$$

$$= (1(y_n=+1), 1(y_n=-1))$$

Let $Q_n = (\hat{P}_{\underline{w}}(1,\underline{x}_n), \hat{P}_{\underline{w}}(-1,\underline{x}_n))$

(our estimate for the distribution of y_n given $\underline{x}_n$).

$$\Rightarrow E_{in}(\underline{w}) = \frac{1}{N}\sum_{n=1}^{N} CE(P_n,Q_n).$$

Lec 09

Gradient descent

Example: How to minimize $E_{in}(\underline{w}) = \frac{1}{N}\sum_{n=1}^{N}\log(1+e^{-y_n\underline{w}^T\underline{x}_n})$

Given a differentiable function $f: \mathbb{R}^n \to \mathbb{R}$

want to $\min_{x\in\mathbb{R}^n} f(x)$

Gradient descent is a numerical approach.

$n=1$: (for illustration)

$x \in \mathbb{R}$

$f'(x) \triangleq \frac{df}{dx}$

if x at optimal, i.e. $x = x^*$, $f'(x) = 0$, [x^*: lowest point]

if $x > x^*$: $f'(x) > 0$ $\Rightarrow f(x)$ increases w.r.t. x

if $x < x^*$: $f'(x) < 0$ $\Rightarrow f(x)$ decreases w.r.t. x

1. Initialize $x = x_0$

2. If $f'(x) = 0$, then stop

3. If $f'(x) > 0$, then $x = x - \varepsilon$

4. If $f'(x) < 0$, then $x = x + \varepsilon$

5. Go to step 2.

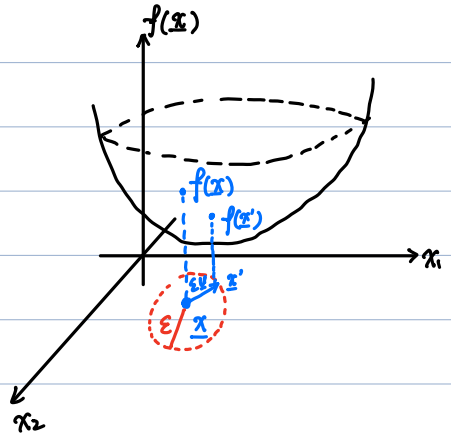
ε: step size — large: less accurate; small: slow progress

Use ε_k, where k is the iteration number:

e.g. $\varepsilon_k \sim \frac{1}{k}$

General n:

$n = 2$



Given current location $\underline{x}$, what should be the next step?

$\underline{x}' \leftarrow \underline{x} + \varepsilon \underline{u}$

ε = step size

$\underline{u}$ = direction of update, $\|u\| = 1$.

Idea: choose direction that maximizes $f(\underline{x}) - f(\underline{x}')$

$f(\underline{x}') = f(\underline{x} + \varepsilon \underline{u})$

$\overset{\text{Taylor}}{=} f(\underline{x}) + \varepsilon \underline{u}^T \nabla f(\underline{x}) + O(\varepsilon^2)$ ← negligible

$\nabla f(\underline{x}) = \begin{bmatrix} \partial f/\partial x_1 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}$

Pick $\underline{u}$ to minimize $\underline{u}^T \nabla f(\underline{x})$

$\underline{u}^T \cdot \underline{v} = \|u\| \cdot \|v\| \cdot \cos\theta \geq -\|u\| \|v\|$ equality when $\cos\theta = -1$.

$$\underline{u}^* = -\frac{\nabla f(\underline{x})}{\|\nabla f(\underline{x})\|}$$

Note: ① $\nabla f(\underline{x})$ points in the direction when $f(\underline{x})$ has max ascent.

② Magnitude $\|f(\underline{x})\|$ in the denominator can be absorbed into ε.

Gradient Decent Algorithm

Initialize x_0 (typically pick at random)

For $t = 0, 1, 2, \cdots$

1. Compute $g_t = \nabla f(x_t)$, $\nabla f(x_t) = \nabla_{\underline{x}} f(\underline{x})\big|_{\underline{x} = x_t}$

2. Select direction $v_t = -g_t$

3. Update $x_{t+1} = x_t + \varepsilon_t v_t$

4. Go to step 1. until stopping criteria are reached

Note: ① ε_t: "learning rate"

② Condition for stopping: $\nabla f(x_t) = 0$ is reasonable.

Lec 10

Gradient Descent

$$x_{t+1} = x_t - \varepsilon_t \nabla f(x_t)$$

E.g. Logistic Regression:

How to minimize $E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N} \log\left(1 + e^{-y_n w^T x_n}\right)$

$$= \frac{1}{N}\sum_{n=1}^{N} e_n(w)$$

GD: $w_{k+1} = w_k - \varepsilon_k \nabla E_{in}(w_k)$

$$= w_k - \varepsilon_k \frac{1}{N}\sum_{n=1}^{N} \nabla e_n(w_k)$$

$$\nabla e_n(w_k) = \nabla_w \log\left(1 + e^{-y_n w^T x_n}\right)$$

$$= \frac{1}{1 + e^{-y_n w^T x_n}} \nabla_w \left(1 + e^{-y_n w^T x_n}\right)$$

$$= \frac{e^{-y_n w^T x_n}}{1 + e^{-y_n w^T x_n}} \nabla_w \left(-y_n w^T x_n\right)$$

$$= \frac{-y_n x_n}{1 + e^{y_n w^T x_n}}$$

Stochastic Gradient Descent (SGD)

GD: $w_{k+1} = w_k - \varepsilon_k \nabla E_{in}(w_k)$

SGD: At each iteration k, select uniformly random the n^{th} example from $\{1, 2, 3, \dots, N\}$ and set $w_{k+1} = w_k - \varepsilon_k \nabla e_n(w_k)$

Note: ① $E[\nabla e_n(w_k)]$

$$= P_r(n=1) \cdot \nabla e_1(w_k) + \cdots + P_r(n=N) \cdot \nabla e_N(w_k)$$

$$= \frac{1}{N}\sum_{n=1}^{N} \nabla e_n(w_k)$$

$$= \nabla E_{in}(w_k)$$

i.e., $\nabla e_n(w_k) = \text{mean} + \text{noise}$

$$= \nabla E_{in}(w_k) + \text{noise}$$

a noisy estimate of the true gradient.

② Full GD ("batch" GD) has a complexity $O(Nd)$ per iteration

Stochastic GD is $O(d)$ Can do many more iterations than GD

(Often there is high redundancy in big datasets

⇒ No need to consider all data points in each step.

Minibatch SGD

At each iteration, select M examples uniformly randomly:

$$\underline{x}_{(1)}, \underline{x}_{(2)}, \cdots, \underline{x}_{(M)}$$

$$\underline{w}_{k+1} = \underline{w}_k - \frac{\varepsilon_k}{M} \sum_{n=1}^{M} e_n(\underline{w}_k)$$

- Better estimate of $\nabla E_{in}(\underline{w})$.
- Parallization across multiple processors.

E.g. Logistic regression:

$$\nabla e_n(\underline{w}) = \frac{-y_n \underline{x}_n}{1 + e^{y_n \underline{w}^T \underline{x}_n}}$$

SGD: $\underline{w}_{k+1} = \underline{w}_k + \varepsilon_k \frac{y_n \underline{x}_n}{1 + e^{y_n \underline{w}^T \underline{x}_n}}$

Note: close connection to the perceptron learning algo.

→ Suppose $(\underline{x}_n, y_n)$ is mis-classified,

$$y_n \underline{w}_k^T \underline{x}_n < 0,$$

$$\Rightarrow e^{y_n \underline{w}^T \underline{x}_n} \text{ is small.}$$

SGD update rule is very close to perceptron, $\underline{w}_{k+1} \approx \underline{w}_k + \varepsilon_k y_n \underline{x}_n$.

→ Suppose $(\underline{x}_n, y_n)$ is correctly classified:

$$y_n \underline{w}^T \underline{x}_n > 0,$$

$$\Rightarrow e^{y_n \underline{w}^T \underline{x}_n} \text{ is large}$$

SGD update rule $\Rightarrow \underline{w}_{k+1} \approx \underline{w}_k$

Lecture 11.

Multiclass Logistic Regression

label $y \in \{1, 2, \cdots, C\}$

Hypothesis:

Let $\Omega = \{\underline{w}_{(1)}, \cdots \underline{w}_{(C)}\}$ be the weight vectors for the C classes.

Hypothesis that

$$\Pr\{y_n = i \mid \underline{x}_n\} = \frac{e^{\underline{w}_{(i)}^T \underline{x}_n}}{\sum_{j=1}^{C} e^{\underline{w}_{(j)}^T \underline{x}_n}}, \quad i = 1, 2, \cdots, C. \quad \text{Softmax function}$$

$$\triangleq \hat{p}(i \mid \underline{x}_n)$$

Error Criterion

$$e_n(\Omega) = -\log \hat{p}(y_n \mid \underline{x}_n) \quad \leftarrow \text{this is a vector}$$

Note: $\nabla_{\Omega} e_n(\Omega) = \begin{bmatrix} \nabla_{\underline{w}_{(1)}} e_n(\Omega) \\ \nabla_{\underline{w}_{(2)}} e_n(\Omega) \\ \vdots \\ \nabla_{\underline{w}_{(C)}} e_n(\Omega) \end{bmatrix}$

SGD update:

For iteration k, $\Omega_k = \{\underline{w}_k(1), \cdots \underline{w}_k(C)\}$

Sample $n \sim$ uniform $\{1, 2, \cdots N\}$.

For $i = 1, 2, \cdots, C$,

Compute $\nabla_{\underline{w}(i)} e_n(\Omega_k)$

$\underline{w}_{k+1}(i) = \underline{w}_k(i) - \varepsilon_k \nabla_{\underline{w}(i)} e_n(\Omega_k)$

Compute $\nabla_{\underline{w}(i)} e_n(\Omega)$: ← n^{th} example $(\underline{x}_n, y_n)$ is chosen

For $i = y_n$,

$$\nabla_{\underline{w}(i)} e_n(\Omega) = \nabla_{\underline{w}(i)} \left(-\log \frac{e^{\underline{w}(y_n)^T \underline{x}_n}}{\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n}} \right)$$ ← $\underline{w}(y_n)$ is the weight of class y_n

$$= \nabla_{\underline{w}(i)} \left(-\underline{w}(y_n)^T \underline{x}_n + \log \left(\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n} \right) \right)$$

$$= -\underline{x}_n + \frac{1}{\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n}} \cdot \nabla_{\underline{w}(i)} \left(\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n} \right)$$

$$= -\underline{x}_n + \frac{e^{\underline{w}(i)^T \underline{x}_n}}{\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n}} \underline{x}_n .$$

For $i \neq y_n$,

$$\nabla_{\underline{w}(i)} e_n(\Omega) = \nabla_{\underline{w}(i)} \left(-\underline{w}(y_n)^T \underline{x}_n + \log \left(\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n} \right) \right)$$

$$= \frac{e^{\underline{w}(i)^T \underline{x}_n}}{\sum_{j=1}^{C} e^{\underline{w}(j)^T \underline{x}_n}} \underline{x}_n$$

Softmax Regression for $C=2$ (Binary classification)

$$\hat{P}(1|\underline{x}) = \frac{e^{\underline{w}(1)^T \underline{x}}}{e^{\underline{w}(1)^T \underline{x}} + e^{\underline{w}(2)^T \underline{x}}} = \frac{e^{(\underline{w}(1) - \underline{w}(2))^T \underline{x}}}{e^{(\underline{w}(1) - \underline{w}(2))^T \cdot \underline{x}} + 1}$$

$$\hat{P}(2|\underline{x}) = \frac{e^{\underline{w}(2)^T \underline{x}}}{e^{\underline{w}(1)^T \underline{x}} + e^{\underline{w}(2)^T \underline{x}}} = 1 - \hat{P}(1|\underline{x})$$

$\equiv$ logistic regression with $\underline{w} = \underline{w}(1) - \underline{w}(2)$.

GD/SGD in linear regression

$$E_{in}(\underline{w}) = \frac{1}{N} \sum_{n=1}^{N} e_n(\underline{w})$$

$$e_n(\underline{w}) = (\underline{w}^T \underline{x}_n - y_n)^2$$

$$\nabla e_n(\underline{w}) = \nabla_{\underline{w}} (\underline{w}^T \underline{x}_n - y_n)^2$$

$$= 2(\underline{w}^T \underline{x}_n - y_n) \cdot \nabla_{\underline{w}} (\underline{w}^T \underline{x}_n - y_n)$$

$$= 2(\underline{w}^T \underline{x}_n - y_n) \cdot \underline{x}_n$$

GD/SGD: as $k \to \infty$, $\underline{w}_k$ converges to the least square solution

$$w_{LS} = (X^T X)^{-1} X^T y.$$

We have perfect solution, & why we need GD/SGD?

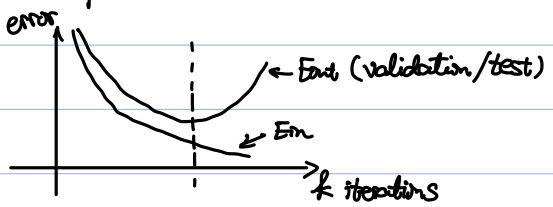
① Computational complexity:

GD: $O(Nd)$, SGD: $O(d)$, mini-batch SGD: $O(md)$ per iteration

② Exact solution may not be desirable (over-fitting) ← more important

We only care about E_{out} (test error), not E_{in}.

In practice, we run a few iteration, do validation & stop when error over the validation dataset is small.



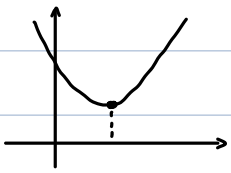
Lecture 12

GD / SGD for non-convex Functions

Convex functions:

e.g. logistic regression
linear regression

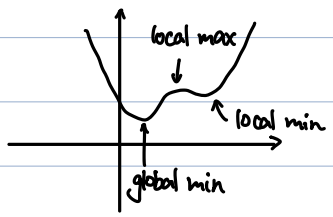
$E_{in}(w)$



⇒ global minimum. when $\nabla E_{in}(w) = 0$

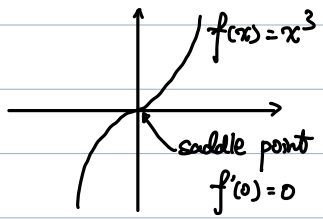
Non-convex functions:

Eg.



Saddle points: $\nabla f(x) = 0$, but x is neither a local minimum nor a local maximum.

1-D:



Very slow progress near saddle points.

⇒ pre-set # iteration runs out.

n-D space:

① 1-D saddle points for any direction.

② local minimum in some direction & local maximum in some other direction.

③ both 1 & 2 combined.

⇒ Highly likely to have saddle points in high-dimensional space.

Stopping Conditions:

to reduce the chances of stopping at a saddle points

① $\nabla E_n(\underline{w}) \approx 0$

② $E_n(\underline{w})$ is small.

③ # of iterations is large.

SGD with Momentum

Basic SGD:

$$\underline{g}_k = \nabla e_n(\underline{w}_k)$$

$$\underline{w}_{k+1} = \underline{w}_k - \varepsilon_k \underline{g}_k$$

SGD + momentum

$$\underline{g}_k = \nabla e_n(\underline{w}_k)$$

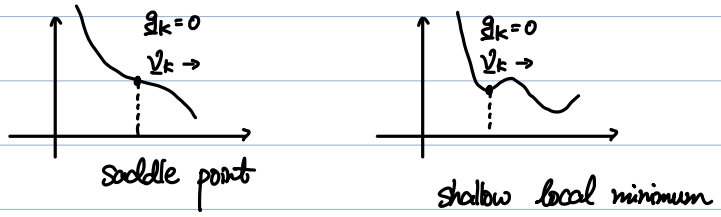
$$\underline{v}_k = -\varepsilon_k \underline{g}_k + \mu \underline{v}_{k-1}$$

e.g. $\mu = 0.9$

↖ accumulation of previous movements

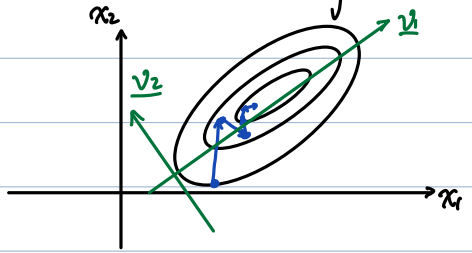
$$\underline{w}_{k+1} = \underline{w}_k + \underline{v}_k$$

① Momentum helps SGD escape flat regions



"heavy ball momentum"

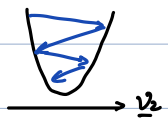
② Momentum can lead to faster convergence even for convex function.



GD strongly favors the $\underline{v}_2$ direction

ping-ponging

Slice $f(x)$ along $\underline{v}_2$ direction:



But overall trend ("momentum")

is moving along $\underline{v}_1$ direction:

SGD + momentum ! (less ping-ponging)

Nestrov Momentum (1983)

$$v_k = -\varepsilon_k \nabla e_n(\underbrace{w_k + \mu v_{k-1}}_{\text{intermediate point}}) + \mu v_{k-1}$$

$$w_{k+1} = w_k + v_k$$

Can prove better convergence.

For convex functions:

- Full GD: distance between w_k & w^* $O(\frac{1}{k})$
- with nesterov momentum: $O(\frac{1}{k^2})$

Lecture 13

Multi-layer perception (MLP)

Recall perception: $\hat{y} = \text{sign}(w^T \cdot x_i)$

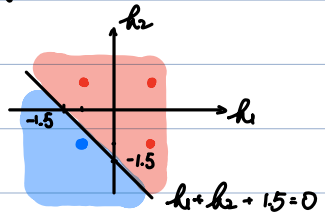
For non-seperable dataset:

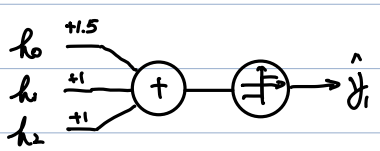
1. Pocket algorithm
2. Non-linear ~~transformation~~ $\Rightarrow$ linear seperable.

Note: too many candidates, not necessary a good transformation

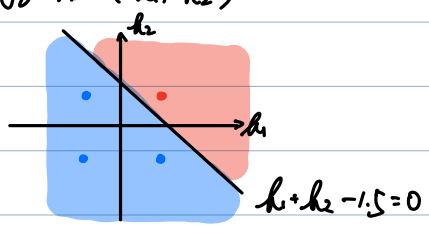
Example:

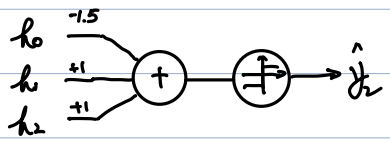
1. $y_1 = OR(h_1, h_2)$



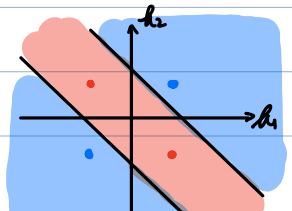


2. $y_2 = AND(h_1, h_2)$

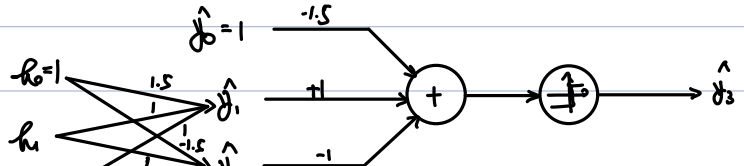




3. $y_3 = XOR(h_1, h_2)$



Multi-layer Perception (2-layer neural network)



$\hat{y}_3 = \text{AND}(\hat{y}_1, \bar{\hat{y}}_2)$

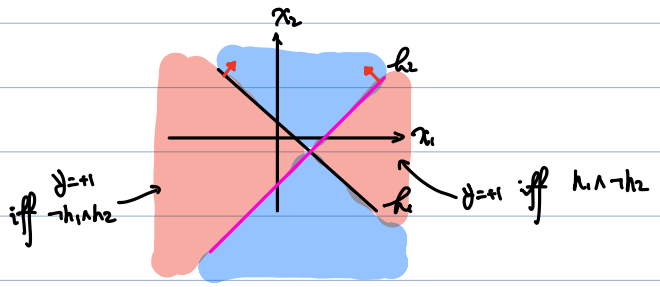
However, we would like to compute $y = f(\underline{x})$.

One more layer: function $h_1(\underline{x})$ & $h_2(\underline{x})$

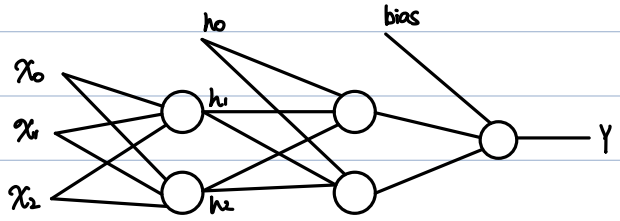
Example:

$h_1(\underline{x}) = \text{sign}(x_1 + x_2 - 1)$

$h_2(\underline{x}) = \text{sign}(-x_1 + x_2 + 2)$



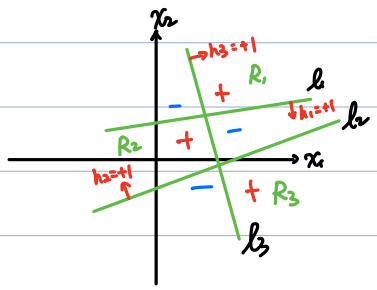
$Y = \text{XOR}(h_1(\underline{x}), h_2(\underline{x}))$



3-layer MLP

Lecture 14

Example:



Fact: Any decision rule can be expressed by intersection of hyperplanes can be interpreted by using a 3-layer MLP.

R_1 : $h_1=-1, h_2=+1, h_3=+1 \iff \text{AND}(\bar{h}_1, h_2, h_3)$

R_2 : $h_1=+1, h_2=+1, h_3=-1 \iff \text{AND}(h_1, h_2, \bar{h}_3)$

R_3 : $h_1=+1, h_2=-1, h_3=+1 \iff \text{AND}(h_1, \bar{h}_2, h_3)$

$y=+1$, iff $x_1 \in R_1 \cup R_2 \cup R_3$

$y = \text{OR}(R_1, R_2, R_3)$

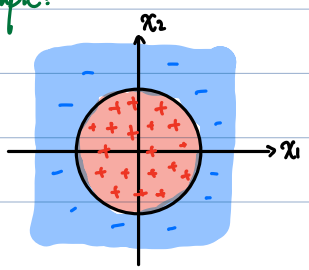
3-layer MLP:

layer 1: compute h_1, h_2, h_3

layer 2: AND function

layer 3: OR function

Example:

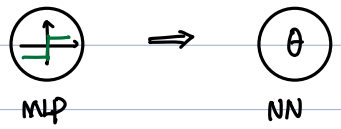


Target: $y = \text{sign}(1 - x_1^2 - x_2^2)$

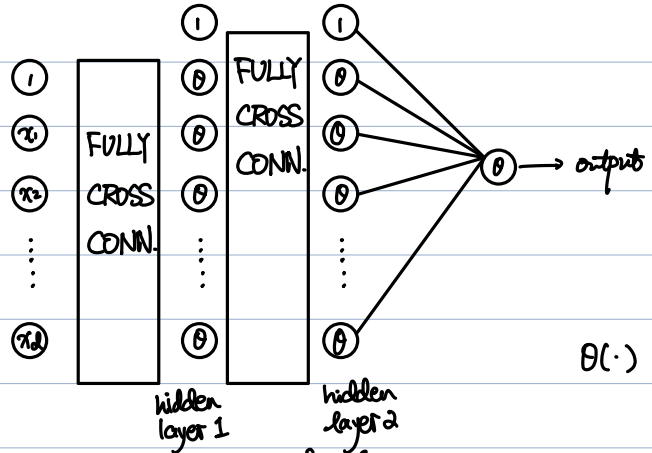
Can use more lines for approx.

Fact: A 3-layer MLP with sufficiently many hidden units in each layer can approximate any smooth decision boundary.

Neural Network (NN)

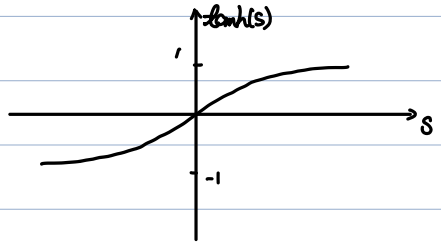


Usually use a softer function than sgn(x) for easier optimization & learning.



$\theta(\cdot)$ is the activation function.

Example: $\theta(\cdot) = \tanh(s) = \frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$



*Universal Approx. Thm.: If the activation function is smooth, single layer can be sufficiently enough to approx. any function.

- $\theta(s) \approx s$ if $|s| \approx 0$
- Saturation if $|s| >> 1$
- $\theta'(s) = 1 - \theta^2(s)$

Notation:

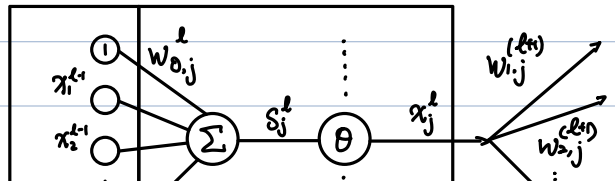
Input layer: $l = 0$

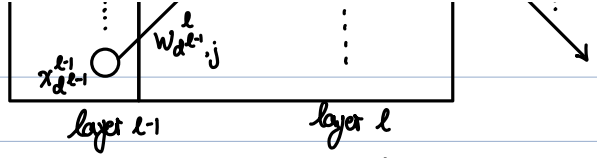
Hidden layer: $1 \le l \le L-1$

Output layer: $l = L$

$d^{(l)}$: number of nodes in layer l: $0 \le l \le L$ (excluding bias)

$w_{i,j}^{(l)}$: weight connection node i in layer $l-1$ to node j in layer l.





Input to node j in layer l: $s_j^{(l)}$

Output of node j in layer l: $x_j^{(l)}$

$$x_j^{(l)} = \theta(s_j^{(l)})$$

$$s_j^{(l)} = w_{0,j}^{(l)} + \sum_{i=1}^{d^{(l-1)}} w_{i,j}^{(l)} \cdot x_j^{(l-1)}$$

Lecture 15. dp. 7.1 & 7.2

Vector notation

$$\underline{s}^{(l)} = \begin{bmatrix} s_1^{(l)} \\ s_2^{(l)} \\ \vdots \\ s_{d^{(l)}}^{(l)} \end{bmatrix}, \quad \theta(\underline{s}^{(l)}) = \begin{bmatrix} \theta(s_1^{(l)}) \\ \theta_2(s_2^{(l)}) \\ \vdots \\ \theta(s_{d^{(l)}}^{(l)}) \end{bmatrix}, \quad \underline{x}^{(l)} = \begin{bmatrix} 1 \\ x_1^{(l)} \\ x_2^{(l)} \\ \vdots \\ x_{d^{(l)}}^{(l)} \end{bmatrix} = \begin{bmatrix} 1 \\ \theta(\underline{s}^{(l)}) \end{bmatrix}$$

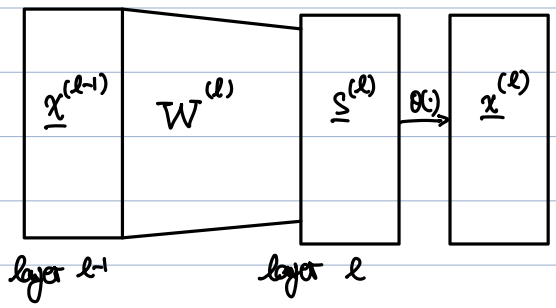
$$W^{(l)} = \{ w_{i,j}^{(l)} \}_{\substack{0 \le i \le d^{(l-1)} \\ 1 \le j \le d^{(l)}}} \text{ (matrix)}$$

$$\underline{s}^{(l)} = (W^{(l)})^T \cdot \underline{x}^{(l-1)}$$

Forward Propagation

$$\underline{x}^{(0)} \xrightarrow{W^{(1)}} \underline{s}^{(1)} \xrightarrow{\theta(\cdot)} \underline{x}^{(1)} \xrightarrow{W^{(2)}} \underline{s}^{(2)}$$

$$\cdots \longrightarrow \underline{x}^{(l-1)} \xrightarrow{W^{(l)}} \underline{s}^{(l)} \xrightarrow{\theta(\cdot)} \underline{x}^{(l)}$$



Pseudo Code

Input: $\underline{x}^{(0)} = (1, x_1, x_2 \cdots x_d)$

For $l = 1, 2, \cdots L$ do

$$\underline{s}^{(l)} = (W^{(l)})^T \underline{x}^{(l-1)}$$

$$\underline{x}^{(l)} = \begin{bmatrix} 1 \\ \theta(\underline{s}^{(l)}) \end{bmatrix}$$

Computational Complexity

of nodes: $V = \sum_{i=1}^{L} d^{(i)}$

# of edges: $E = \sum_{l=0}^{L-1} (d^{(l)}+1)\, d^{(l+1)}$

# of computation: $O(V + E)$

V: activation function; E: Matrix Multiplication

Training of Neural Network

Loss function:

Let $\Omega = \{W^{(1)}, W^{(2)}, \cdots W^{(L)}\} \in \mathbb{R}^3$ be our model parameters

$e_n(\Omega)$ loss function w.r.t. example (x_n, y_n)

Example: linear regression for output layer:

square error: $(x^{(L)} - y_n)^2$

logistic: \
softmax: / regression for output layer.

log-loss: $-\log \hat{P}(y_n | x_n)$

Stochastic Gradient Descent

In iteration k, suppose Ω_k is the current set of parameters.

- Select (x_n, y_n) at random, and compute:

$$\nabla_\Omega e_n(\Omega_k) = \frac{\partial e_n(\Omega_k)}{\partial W_{(i,j)}^{(l)}}, \forall i,j,l.$$

- $W_{i,j}^{(l)} \leftarrow W_{i,j}^{(l)} - \varepsilon \cdot \frac{\partial e_n(\Omega_k)}{\partial W_{(i,j)}^{(l)}}, \forall i,j,l.$

Numerical Approach (finite difference)

Replace $W_{i,j}^{(l)}$ by $W_{i,j}^{(l)} + \Delta w$ while keeping all other weights fixed.

Let Ω_k^+ be the resultant set of parameters

$$\frac{\partial e_n(\Omega_k)}{\partial W_{i,j}^{(l)}} = \frac{e_n(\Omega_k^+) - e_n(\Omega_k)}{\Delta w}$$

$O(V+E)$ complexity to compute $e_n(\Omega_k^+)$ by forward propagation.

Overall complexity is $O(VE + E^2)$ for each iteration.

Typically, # of weights (E) $\sim$ 10 M.

Lecture 16

Back Propagation

Note:

1. $e(\Omega)$ is memoryless. No knowledge about any previous layers.

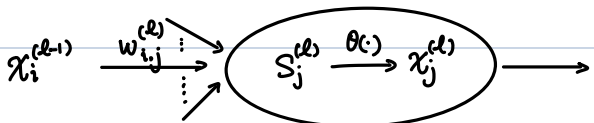
$$e(\Omega) = e(s^{(l)}, w^{(l+1)}, \cdots, w^{(L)})$$

($s^{(l)}$: all info up to layer l.)

$$= e((s_1^{(l)}, \cdots s_j^{(l)} \cdots, s_{d^{(l)}}^{(l)}), w^{(l+1)}, \cdots, w^{(L)})$$

2. Only $s_j^{(l)}$ depends on $w_{i,j}^{(l)}$.

$$\Rightarrow \frac{\partial e(\Omega)}{\partial w_{i,j}^{(l)}} = \frac{\partial e(\Omega)}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{i,j}^{(l)}}$$



$$s_j^{(l)} = w_{0,j}^{(l)} + \sum_{k=1}^{d^{(l-1)}} w_{k,j}^{(l)} \cdot x_k^{(l-1)}$$

$$\Rightarrow \frac{\partial s_j^{(l)}}{\partial w_{i,j}^{(l)}} = x_i^{(l-1)}$$

Let $\delta_j^{(l)} \triangleq \frac{\partial e(\omega_k)}{\partial s_j^{(l)}}$ (Sensitivity of $e(\omega_k)$ to layer l input)

$$\Rightarrow \boxed{\frac{\partial e(\omega_k)}{\partial w_{i,j}^{(l)}} = x_i^{(l-1)} \cdot \delta_j^{(l)}}$$

$$\frac{\partial e(\omega_k)}{\partial w^{(l)}} \triangleq \left[\frac{\partial e(\omega_k)}{\partial w_{i,j}^{(l)}}\right]_{\substack{0 \le i \le d^{(l-1)} \\ 1 \le j \le d^{(l)}}}, \quad x^{(l-1)} = \begin{bmatrix} x_0^{(l-1)} \\ \vdots \\ x_{d^{(l-1)}}^{(l-1)} \end{bmatrix}, \quad \delta^{(l)} = \begin{bmatrix} \delta_1^{(l)} \\ \vdots \\ \delta_{d^{(l)}}^{(l)} \end{bmatrix}$$

Then $\frac{\partial e(\omega_k)}{\partial w^{(l)}} = \underset{n \times 1}{x^{(l-1)}} \left(\underset{1 \times d}{\delta^{(l)}}\right)^T$

To compute $\delta^{(l)}$ "backwards"

① Consider $l = L$,

let $d^{(L)} = 1$ for simplicity.

$$s^{(L)} \longrightarrow \boxed{\theta} \longrightarrow x^{(L)}, \quad g(x^{(L)}, y) = e(\omega_k)$$

$$\delta^{(L)} = \frac{\partial e(\omega_k)}{\partial s^{(L)}} = \frac{\partial g(x^{(L)}, y)}{\partial s^{(L)}}$$

$$= \frac{\partial g(x^{(L)}, y)}{\partial x^{(L)}} \cdot \frac{\partial x^{(L)}}{\partial s^{(L)}}$$

$$= \frac{\partial g(x^{(L)}, y)}{\partial x^{(L)}} \cdot \theta'(s^{(L)})$$

Example: Regression:

$$g(x^{(L)}, y) = (x^{(L)} - y)^2$$

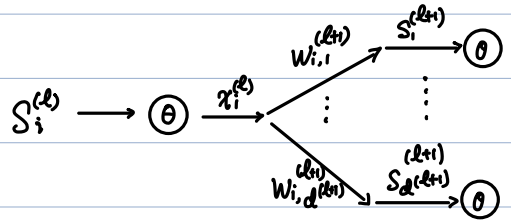
$$\delta^{(L)} = 2(x^{(L)} - y) \cdot \theta'(s^{(L)})$$

② Intermediate node: $\delta_i^{(l)}$

$$\delta_i^{(l)} = \frac{\partial e(\omega_k)}{\partial s_i^{(l)}}$$

$$= \frac{\partial e(\omega_k)}{\partial x_i^{(l)}} \cdot \frac{\partial x_i^{(l)}}{\partial s_i^{(l)}}$$

$$= \frac{\partial e(\omega_k)}{\partial x_i^{(l)}} \cdot \theta'(s_i^{(l)})$$



Recall: chain rule

If $z = f(x, y)$, and $\begin{cases} x = g(s,t) \\ y = h(s,t) \end{cases}$

then $\frac{\partial z}{\partial s} = \frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial s} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial s}$

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial t} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial t}$$

$$\Rightarrow \frac{\partial e(\omega_k)}{\partial x_i^{(l)}} = \sum_{j=1}^{d^{(l+1)}} \frac{\partial e(\omega_k)}{\partial s_j^{(l+1)}} \cdot \frac{\partial s_j^{(l+1)}}{\partial x_i^{(l)}}$$

$$= \sum_{j=1}^{d^{(l+1)}} \delta_j^{(l+1)} \cdot w_{i,j}^{(l+1)}$$

$$\Rightarrow \boxed{\delta_i^{(l)} = \left[\sum_{j=1}^{d^{(l+1)}} \delta_j^{(l+1)} \cdot w_{i,j}^{(l+1)}\right] \cdot \theta'(s_i^{(l)})}$$

$$\Rightarrow \quad \underline{\delta}^{(l)} = \left[\hat{W}^{(l+1)} \cdot \underline{\delta}^{(l+1)} \right] \otimes \theta'(s^{(l)})$$

pointwise multiplication

where $\underline{\delta}^{(l)} = \begin{bmatrix} \delta_1^{(l)} \\ \vdots \\ \delta_{d^{(l)}}^{(l)} \end{bmatrix}$, $\theta'(\underline{s}^{(l)}) = \begin{bmatrix} \theta'(s_1^{(l)}) \\ \vdots \\ \theta'(s_{d^{(l)}}^{(l)}) \end{bmatrix}$

$$\hat{W}^{(l+1)} \triangleq \begin{bmatrix} w_{1,1}^{(l+1)} & \cdots & w_{1,d^{(l+1)}}^{(l+1)} \\ \vdots & \ddots & \vdots \\ w_{d^{(l)},1}^{(l+1)} & \cdots & w_{d^{(l)},d^{(l+1)}}^{(l+1)} \end{bmatrix}$$

Lecture 17

Backpropagation Algorithm

Input: (x, y), $\Omega = \{W^{(1)}, W^{(2)}, \cdots, W^{(L)}\}$

Output: $\frac{\partial e(\Omega)}{\partial w_{ij}^{(l)}}$, $l = 1, 2, \cdots, L$

1. Run forward propagation to compute:

$\{s^{(l)}, x^{(l)}\}$, $l = 1, 2, \cdots, L$

$e(\Omega) = g(x^{(L)}, y)$

2. $\underline{\delta}^{(L)} = \left[\nabla_{x^{(L)}} g(\underline{x}^{(L)}, y) \right] \otimes \theta'(s^{(L)})$

3. For $l = L-1$ to 1 do:

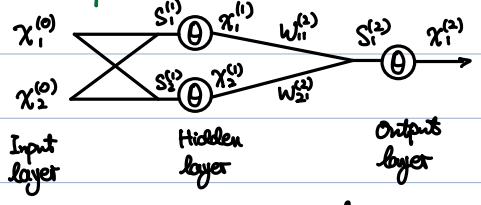
$\underline{\delta}^{(l)} = \left[\hat{W}^{(l+1)} \underline{\delta}^{(l+1)} \right] \otimes \theta'(s^{(l)})$

$\frac{\partial e(\Omega)}{\partial W^{(l)}} = \underline{x}^{(l-1)} (\delta^{(l)})^T$

End.

Complexity: $O(E)$ ← One forward propagation to compute all $\frac{\partial e}{\partial w^{(l)}}$

For example:



$L = 2$

$e(\Omega) = (x_1^{(2)} - y)^2$

layer 2: $\delta_1^{(2)} = \frac{\partial e}{\partial x_1^{(2)}} \cdot \frac{\partial x_1^{(2)}}{\partial s_1^{(2)}}$

$= 2(x_1^{(2)} - y) \cdot \theta'(s_1^{(2)})$

$\frac{\partial e}{\partial w_{11}^{(2)}} = x_1^{(1)} \cdot \delta_1^{(2)}$, $\frac{\partial e}{\partial w_{21}^{(2)}} = x_2^{(1)} \cdot \delta_1^{(2)}$

layer 1: $\delta_1^{(1)} = \theta'(s_1^{(1)}) \cdot \delta_1^{(2)} \cdot w_{11}^{(2)}$

$\delta_2^{(1)} = \theta'(s_2^{(1)}) \cdot \delta_1^{(2)} \cdot w_{21}^{(2)}$

$\frac{\partial e}{\partial w_{ij}^{(1)}} = x_i^{(1)} \cdot \delta_j^{(1)}, \quad i,j \in \{1,2\}$

Note:

① Can do more:

$$\frac{\partial e}{\partial x_1^{(0)}} \triangleq \delta_1^{(0)} = \frac{\partial e}{\partial s_1^{(0)}}$$
$$= \frac{\partial e}{\partial s_1^{(1)}} \cdot \frac{\partial s_1^{(1)}}{\partial x_1^{(0)}} + \frac{\partial e}{\partial s_2^{(1)}} \cdot \frac{\partial s_2^{(1)}}{\partial x_1^{(0)}}$$
$$= \delta_1^{(1)} \cdot w_{11}^{(1)} + \delta_2^{(1)} \cdot w_{12}^{(1)}$$

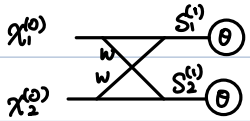
Similarly, we have:

$\frac{\partial e}{\partial x_2^{(0)}} = \delta_2^{(1)} \cdot w_{21}^{(1)} + \delta_2^{(1)} \cdot w_{22}^{(1)}$

Note: 1. if ~~too~~ sensitive to the input, { system may change drastic with small changes in inputs.
Too sensitive to noise, overfit.

2. What if we force

$w_{12}^{(1)} = w_{21}^{(1)} \triangleq w$?



$$\frac{\partial e}{\partial w} = \frac{\partial e}{\partial s_1^{(1)}} \cdot \frac{\partial s_1^{(1)}}{\partial w} + \frac{\partial e}{\partial s_2^{(1)}} \cdot \frac{\partial s_2^{(1)}}{\partial w}$$
$$= \delta_1^{(1)} \cdot x_2^{(0)} + \delta_2^{(1)} \cdot x_1^{(0)}$$

More general case CNN.

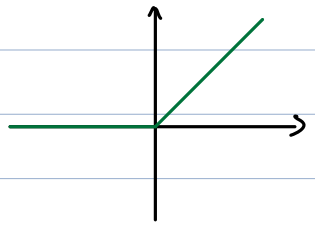
Implementation of Neural Network

1) Choice of activation function

→ Sigmoid $\theta(s) = \frac{e^s}{1+e^s}$

→ tanh $\theta(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$

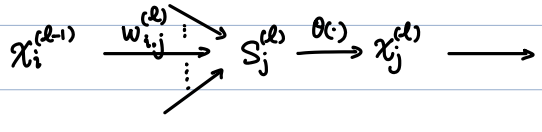
→ Rectified Linear Unit $\theta(s) = \max(0, s)$



- Simple calculation ($\theta'(s) = 0, \forall s < 0 \Rightarrow$ Dead neuron $\Rightarrow$ Less calculation)
- Avoids non-linearity effect

(default in modern NNs)

Issue: Dead neuron



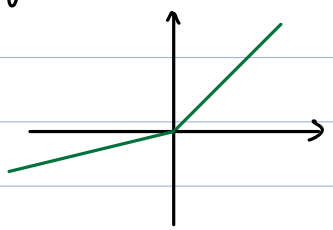
$s_j^{(l)} < 0 \Rightarrow \{ x_j^{(l)} = 0$

$\theta'(s_j^{(l)}) = 0$

$\Rightarrow \frac{\partial e}{\partial w_{ij}^{(l)}} = 0, \ \forall i$

- Weights no longer updated.
- Cause ~40% neurons to die during training.

→ Leaky ReLU



$$\theta(s) = \begin{cases} s, & s > 0 \\ \alpha s, & s \le 0 \end{cases}$$

eg. $\alpha = 0.1$

→ Parametric ReLU

α is selected during training.

Lecture 18

Implementation of Neural Network (Cont.)

2) Input pre-processing

① Normalization

$D = \{(x_1, y_1), \cdots (x_N, y_N)\}$

e.g. x { salary, # of children

Sample mean: $\mu = \frac{1}{N} \sum_n x_n$

Sample standard deviation ← pointwise

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{n=1}^{N} (x_n - \mu)^2}$$

$$x_n \longleftarrow \frac{x_n - \mu}{\sigma}$$ pointwise

⇒ Each element has zero mean & unit variance

② Data augmentation

$(x, y) \leftarrow (F(x), y)$

$F(\cdot)$ can be rotation, flip, crop, scaling, etc.

Eg. Principle Component Analysis

3) Weight Initialization (Sec 8.4)

- Small weight

works ok with small NN.

Vanishing gradient in deep NN.
(weights do not transfer info between layers)

- Large weight

signal grows over layers.

e.g. saturation of $\theta(\cdot)$.

E.g. Xavier Initialization

0 for bias terms

$w_{ij}^{(l)} \sim \mathcal{N}(0, \frac{1}{n})$, mean of 0, variance of $\frac{1}{n}$

where n = # of edges from $l-1$ to Node j of layer l.

$s_j^{(l)} = w_{0,j}^{(l)} + \sum_{i=1}^{n} x_i^{(l-1)} \cdot w_{i,j}^{(l)}$

$\text{var}(s_j^{(l)}) = 0 + n(\frac{1}{n}) = 1$, (assume independence among xs)

Other choices:

$\sim \mathcal{N}(0, \frac{2}{\#\text{ of in} + \#\text{ of out}})$

$\sim$ uniform distr. with var $\frac{1}{n}$

$U(-\frac{\sqrt{6}}{n}, \frac{\sqrt{6}}{n})$

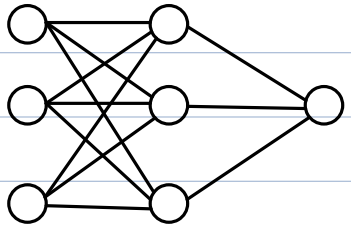
Note: With ReLU, need $\text{var}(w_{ij}^{(l)}) \approx \frac{2}{n}$

4) Dropout

During training, in each SGD update, for each node with probability p set its output to zero. in the forward path.

⇒ During back propagation, only the weights connected to the active nodes are updated.

$$\frac{\partial e}{\partial w_{ij}^{(l)}} = x_i^{(l-1)} \cdot \delta_j^{(l)}$$



Handwaving: by multiple NNs at the same time.

Testing: Use the entire network, but scale the weights by the factor of $(1-p)$

$$w_{ij}^{(l)} \longleftarrow w_{ij}^{(l)} \cdot (1-p)$$

5) Variation of SGD.

① Basic SGD

$$\underline{w}_{t+1} = \underline{w}_t - \varepsilon_t \nabla f(\underline{w}_t)$$

(ε_t: ε_n, e_n)

② SGD with momentum

$$\underline{v}_t = \alpha \underline{v}_{t-1} - \varepsilon_t \nabla f(\underline{w}_t)$$

$$\underline{w}_{t+1} = \underline{w}_t + \underline{v}_t$$

③ Ada Gradient

Vary learning rate across different dimension.

$$\underline{r}_t = \underline{r}_{t-1} + \nabla f(\underline{w}_t) \otimes \nabla f(\underline{w}_t)$$

$$\underline{w}_{t+1} = \underline{w}_t - \varepsilon_0 \nabla f(\underline{w}_t) \otimes \frac{1}{\sqrt{\underline{r}_t + 10^{-5}}}$$

(10^{-5}: small number for stability.)

Advantage:

smaller step size along steeper dimension.

Disadvantage:

Cumulate meaningless past

④ RMS prop

- Graduately forget distant history

$$\underline{r}_t = \alpha \underline{r}_{t-1} + (1-\alpha) \cdot \nabla f(\underline{w}_t) \otimes \nabla f(\underline{w}_t), \quad 0 < \alpha < 1$$

⑤ RMS-Prop + momentum

⑥ Adam

⑤ + correction factor

Lecture 19 Unsupervised Learning

$D = \{\underline{x}_1, \underline{x}_2 \cdots, \underline{x}_N\}$, $\underline{x}_i \in R^d$

- No labels

Eg. D: set of documents

Goal: group by topic

$\underline{x}_i$: histogram of word length in document i.

① Clustering

② Density estimation

③ Dimensionality reduction

Clustering (chapter 6.3.3)

partition D into K disjoint clusters s.t. the elements in each cluster are close to each other.

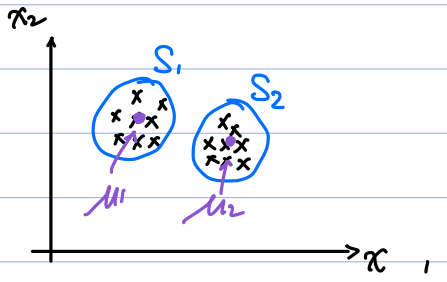
Given D, output

① clusters $S_1, S_2, \cdots S_K$, $S_i \subseteq D$

s.t. $S_i \cap S_j = \phi \quad \forall\ i \neq j$

$\bigcup_{i=1}^{K} S_i = D$

② Cluster center: $\underline{\mu}_1, \underline{\mu}_2, \underline{\mu}_3 \cdots \underline{\mu}_K$, $\underline{\mu}_i \in \mathbb{R}^d$



Error Measure

$E_j = \sum_{\underline{x}_n \in S_j} \| \underline{x}_n - \underline{\mu}_j \|^2$. approximate error for cluster S_j.

Given D and K, define

$E_{in}(S_1, S_2, \cdots S_K, \underline{\mu}_1, \underline{\mu}_2 \cdots \underline{\mu}_K) = \sum_{j=1}^{K} E_j$

$= \sum_{n=1}^{N} \| \underline{x}_n - \underline{\mu}(\underline{x}_n) \|^2$

where $\underline{\mu}(\underline{x}_n) \equiv$ center of cluster for which $\underline{x}_n$ belongs

- Learning problem

$$\min_{\substack{S_1, S_2 \cdots S_K \\ \underline{\mu}_1, \underline{\mu}_2 \cdots \underline{\mu}_K}} E_{in}$$

- Applications
  - Classification (e.g. documents, coins)
  - Recommendation system

Optimal cluster is NP-hard!

Heuristic

K-means cluster

An alternating optimization approach with two subproblems

Subproblem #1.

Given $S_1, \cdots S_K$, find $\underline{\mu}_1 \cdots \underline{\mu}_K$ to minimize E_{in}.

$E_{in} = \sum_{j=1}^{N} E_j$, $E_j = \sum_{\underline{x}_n \in S_j} \| \underline{x}_n - \underline{\mu}_j \|^2$ depends only on S_j.

$\Rightarrow$ Only need to consider

$$\min_{\underline{\mu}_j} E_j(\underline{\mu}_j) \iff \min_{\underline{\mu}_j} \sum \| \underline{x}_n - \underline{\mu}_j \|^2$$

$$\nabla E_j(\underline{\mu}_j) = 0$$

$$\Rightarrow \nabla_{\underline{\mu}_j} \sum_{\underline{x}_n \in S_j} \|\underline{x}_n - \underline{\mu}_j\|^2$$

$$= \sum_{\underline{x}_n \in S_j} -2(\underline{x}_n - \underline{\mu}_j) = 0$$

$$\Rightarrow \sum_{\underline{x}_n \in S_j} \underline{x}_n = \underline{\mu}_j |S_j|$$

$$\underline{\mu}_j = \frac{1}{|S_j|} \sum_{\underline{x}_n \in S_j} \underline{x}_n \text{ , sample average of points in } S_j.$$

Subproblem #2

Given $\underline{\mu}_1, \cdots, \underline{\mu}_K$, find $S_1 \cdots S_K$ to minimize E_{in}.

≡ Given $\underline{x}_i \in D$, which cluster should it be associated?

$\underline{\mu}(\underline{x}_i)$: cluster center for $\underline{x}_i$.

$$E_{in} = \sum_{n=1}^{N} \|\underline{x}_n - \underline{\mu}(\underline{x}_n)\|^2$$

To minimize of $\|\underline{x}_n - \underline{\mu}(\underline{x}_n)\|$ depends only on $\underline{x}_n$ & $\underline{\mu}(\underline{x}_n)$

$$\underline{\mu}(\underline{x}_n) = \underset{\underline{\mu} \in \{\underline{\mu}_1, \cdots \underline{\mu}_K\}}{\operatorname{argmin}} \|\underline{x}_n - \underline{\mu}\|^2$$

Assign $\underline{x}_n$ to the nearest cluster center.

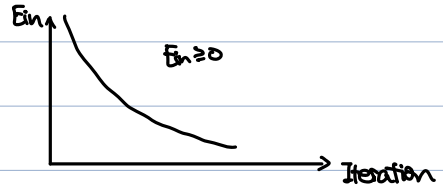
Lecture 25

K-means Algorithm

1. Initialize $\underline{\mu}_1, \cdots \underline{\mu}_N$ randomly
2. Construct $S_1, \cdots S_N$ by solving subproblem #2 ("nearest cluster center")
3. Update $\underline{\mu}_1, \cdots \underline{\mu}_N$ by solving subproblem #1 ("centroid")
4. Repeat step 2 & 3 until converge.

Notes:

① Convergence is guaranteed

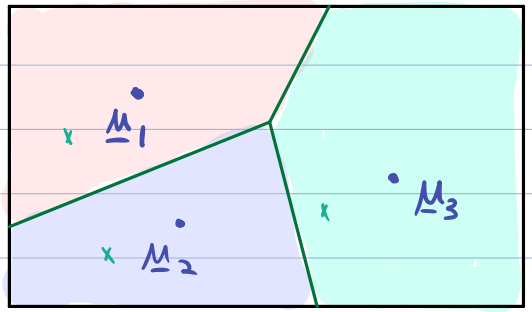


Monotone Convergence Theorem

② Locally optimal

Not global optimal in general

③ End Result:

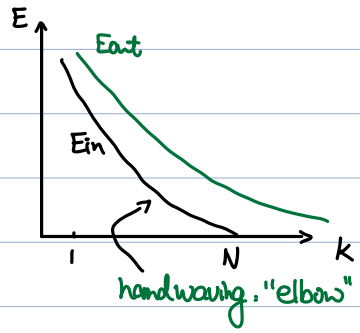


if $k=3$

K-means creates a voronoi diagram ("tessellation")

Test / inference: given new data point $\underline{x}$.

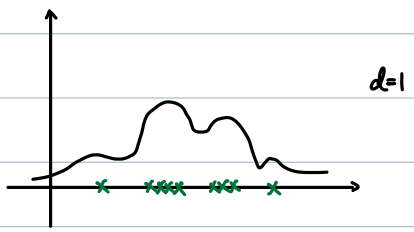
④ How to choose K?



Density Estimation

$D = \{\underline{x}_1, \cdots, \underline{x}_N\}$, $x_i \overset{iid}{\sim} p(\underline{x})$

output $\hat{p}(\underline{x}) \approx p(\underline{x})$



Application: anomaly detection

Review:

Uniform Distribution

Discrete RV X ~ uniform in $\{1,2,\cdots,10\}$

PMF: $Pr\{X=k\} = \frac{1}{10}$, $1 \le k \le 10$

Continuous RV X ~ uniform in (a,b)

$$Pr\{X = \frac{a+b}{2}\} = 0$$

PDF: $f_X(x) = \begin{cases} \frac{1}{b-a}, & a \le x \le b \\ 0, & \text{otherwise} \end{cases}$

Gaussian Distribution

$X \sim N(\mu, \sigma^2)$

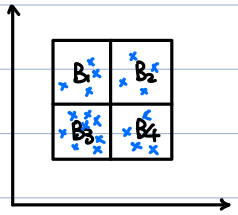
$f_X(\underline{x}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Question: Given sample points, how do we find the best line to fit density function?

1) Histogram Method

Assume $P(\cdot)$ is non-zero only over a bounded region cover region with uniform bins.

$B_1, B_2, \cdots B_k$, each with volume V. (Area / Range within the bin)



Let N_i be the # of samples in bin B_i

Hypothesis: $Pr\{X \in B_i\} = \frac{N_i}{N}$

Assume that the distribution within each bin is uniform. $\Rightarrow$ PDF: $\frac{1}{V}$

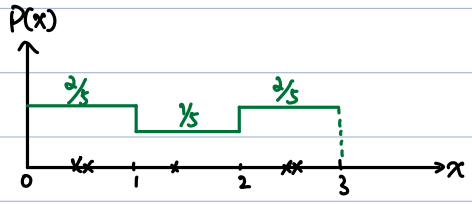
Total probability theorem

$$\hat{P}(x) = \sum_{i=1}^{k} \hat{P}(x \mid x \in B_i) \frac{N_i}{N}$$

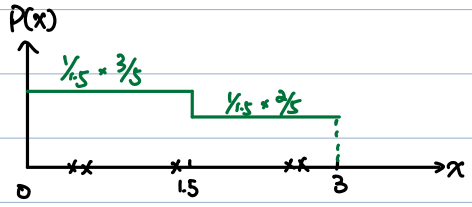
Since we assume uniform distribution in each bin,

$$\hat{P}(x) = \sum_{i=1}^{k} \frac{1}{V} \cdot \mathbb{I}(x \in B_i) \frac{N_i}{N}$$

$$= \begin{cases} \frac{1}{V} \cdot \frac{N_i}{N}, & x \in B_i \\ \vdots \\ \frac{1}{V} \cdot \frac{N_k}{N}, & x \in B_k \\ 0, & \text{otherwise} \end{cases}$$



K=3, V=1 for each bin



K=2, V=1.5 for each bin

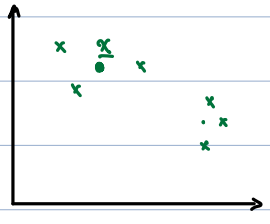
Note: larger k, smaller bin (V↓) $\Rightarrow$ more granularity $\Rightarrow$ overfit

Lecture 2|

2) Nearest Neighbor Estimation

For $\forall x \in \mathbb{R}^d$ (not necessary a sample point),

Let's define $x_{[1]}, x_{[2]} \cdots, x_{[k]}$ be the first k nearest neighbor of x, sorted in ascending order.



Let $r_k(x) = \|x - x_{[k]}\|$ (radius of smallest sphere)

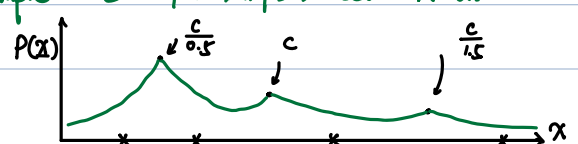
$V_k(x)$ = volume of sphere in $\mathbb{R}^d$ with radius $r_k(x)$

then $\hat{P}(x) \propto \frac{1}{V_k(x)}$

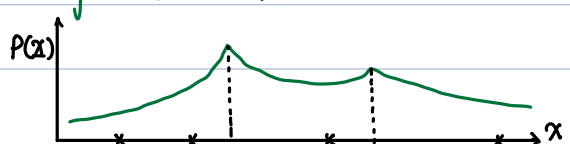
$$\hat{P}(x) = \frac{c}{V_k(x)}$$

To find c, we have $\int \hat{P}(x) = 1$

Example: $D = \{1, 2, 4, 7\}$. set $K = 2$.



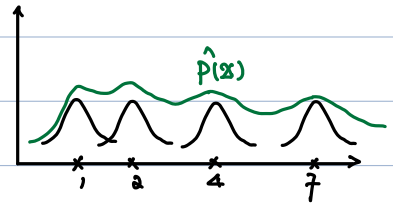
if set $K = 3$.



0 1 2 4 7 0 1 2 2.5 4 4.5 7

Observation: larger K, smoother outputs.

3) Parzen Window Estimation with Gaussian Kernels



→ Each sample has a Gaussian Distribution.

→ Add all the probability

→ Scale to have CDF = 1

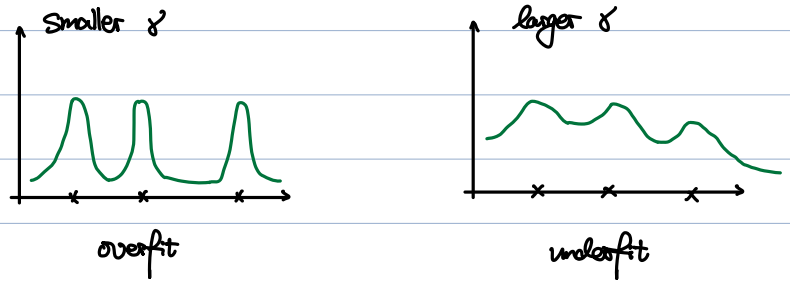
Details: Standard normal distribution (PDF)

$$\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

$$\hat{p}(x) = \frac{1}{k} \sum_{i=1}^{N} \phi\left(\frac{\|x - x_i\|}{\gamma}\right)$$

γ: kernel width, width of each sample affected neighbor.

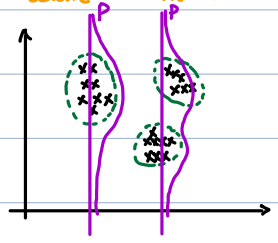
k: Normalizing constant



Note: If more points → complex computation.

Lec 22.

4) Gaussian Mixture Model



Each cluster with its own Gaussian distribution

It suffices to use only k Gaussian distributions, one for each cluster.

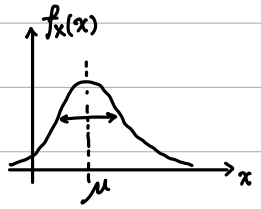
- mean: $\underline{\mu}_1, \underline{\mu}_2, \cdots, \underline{\mu}_k \in \mathbb{R}^d$

  covariance: $\Sigma_1, \Sigma_2, \cdots, \Sigma_k \in \mathbb{R}^d \times \mathbb{R}^d$

Review:

$d=1$: PDF of Gaussian R.V. X with mean μ, and variance σ^2.

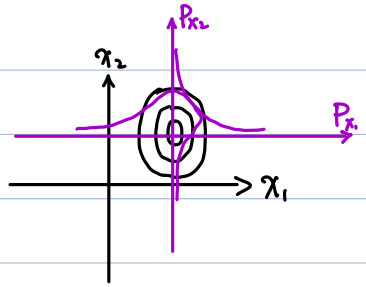
$$f_X(x) \equiv \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\left(\frac{x-\mu}{2\sigma^2}\right)^2}$$



$d=2$: PDF of jointed Gaussian R.V. X_1 and X_2, with mean μ_1 & μ_2, variances σ_1^2 & σ_2^2, and correlation coefficient ρ.

$$\rho = \frac{E[(X_1-\mu_1)(X_2-\mu_2)]}{\sigma_1 \cdot \sigma_2}$$

$$f_{X_1,X_2}(x_1,x_2) \equiv \mathcal{N}(x;\mu_1,\mu_2,\sigma_1^2,\sigma_2^2,\rho)$$
$$= \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \cdot e^{-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2} + \left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right]}$$



General $d \gg 1$:

Consider d jointly Gaussian RVs, $X_1, X_2 \cdots X_d$, with $\mu_1, \mu_2 \cdots \mu_d$ and covariance matrix.

$$\Sigma \triangleq \begin{bmatrix} COV(X_1,X_1) & COV(X_1,X_2) & \cdots & COV(X_1,X_d) \\ \vdots & & & \vdots \\ COV(X_d,X_1) & \cdots & & COV(X_d,X_d) \end{bmatrix}$$

where $COV(X_i,X_j) = E[(X_i-\mu_i)(X_j-\mu_j)]$

$$\Leftrightarrow \Sigma = E[(X-\underline{\mu})(X-\underline{\mu})^T]$$

$$f_{\underline{X}}(\underline{x}) = \mathcal{N}(\underline{x},\underline{\mu},\Sigma)$$
$$= \frac{1}{(2\pi)^{d/2} \cdot \det(\Sigma)^{1/2}} \cdot e^{-\frac{1}{2}(\underline{x}-\underline{\mu})^T \Sigma^{-1} (\underline{x}-\underline{\mu})}$$

Data Generalization Model (Hypothesis)

(for each data point in $\mathcal{D}$)

First, pick a bump (cluster)

$j \in \{1, 2, \cdots, k\}$ according to probability distribution $\{w_1, w_2 \cdots, w_k\}$

$$\begin{cases} w_i > 0 \\ \sum_{i=1}^{k} w_i = 1 \end{cases}$$

Then, generate $\underline{x}$ according to probability density $P(\underline{x}|j) = \mathcal{N}(\underline{x},\underline{\mu}_j,\Sigma_j)$

$$= \frac{1}{(2\pi)^{d/2} \cdot \det(\Sigma_j)^{1/2}} \cdot e^{-\frac{1}{2}(\underline{x}-\underline{\mu}_j)^T \Sigma_j^{-1} (\underline{x}-\underline{\mu}_j)}$$

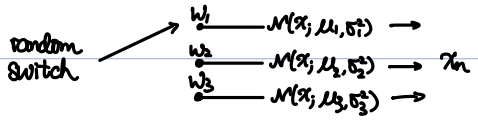
$\Rightarrow$ the overall PDF for $\underline{x}$ is

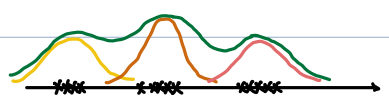
$$P(\underline{x}) = \sum_{j=1}^{k} w_j P(\underline{x}|j) = \sum_{j=1}^{k} w_j \cdot \mathcal{N}(\underline{x},\underline{\mu}_j,\Sigma_j)$$

Example: $d=1$, $k=3$

$\mathcal{D} = \{x_1, x_2 \cdots, x_N\}$

For each index n,





Given $\mathcal{D}$, need to estimate μ_1, μ_2, μ_3 & $\sigma_1, \sigma_2, \sigma_3$.

Given $\mathcal{D} = \{\underline{x}_1, \cdots \underline{x}_N\}$, and k. find a GMM $\Omega = \{w_j, \underline{\mu}_j, \Sigma_j\}_{j=1}^{k}$ that is the best fit for $\mathcal{D}$.

Best-fit: MLE

$\hat{P}_\Omega(\mathcal{D}) = \prod_{n=1}^{N} \hat{P}(\underline{x}_n)$ is maximized.

Minimize $E_n(\Omega) \triangleq -\log \hat{P}_\Omega(\mathcal{D})$

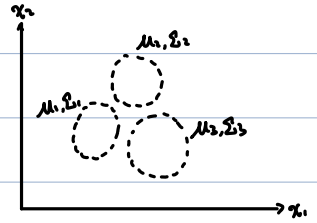
$$= -\sum_{n=1}^{N} \log \hat{P}(\underline{x}_n)$$

$$e_n(\Omega) = -\log \hat{P}(x_n)$$

How to minimize $E_{in}(\Omega)$?

SGD?

Lecture 23



$$D = \{x_1, x_2, \cdots, x_N\}$$

$$P(x) = \sum_{j=1}^{K} w_j \cdot \mathcal{N}(x_j, \mu_j, \Sigma_j)$$

Given set D and k, find the GMM $\Omega = \{w_j, \mu_j, \Sigma_j\}_{j=1}^{k}$ that is the "best fit" for D.

Maximize the likelihood

$$\hat{P}_{\Omega}(D) = \prod_{n=1}^{N} \hat{P}(x_n)$$

$\equiv$ Minimize $E_{in}(\Omega) = -\log \hat{P}_{\Omega}(D)$

$$= \sum_{n=1}^{N} -\log \hat{P}(x_n)$$

$$= \sum_{n=1}^{N} \underbrace{-\log \sum_{j=1}^{k} w_j \cdot \mathcal{N}(x_n; \mu_j, \Sigma_j)}_{e_n(\Omega)}$$

Can use SGD ? No

- Complicated e_n
- $\sum_j w_j = 1$. constrained optimization

The Expectation Maximization (EM) Algorithm

An alternating optimization approach with 2 subproblems.

Subproblem 1. (M step)

Given $D = \{x_1, \cdots, x_N\}$, suppose for each $x_i \in D$, we know the bump (Gaussian distribution) it belongs to.

Let $B_j \in D$ denotes the j^{th} bump.

i.e. all points in B_j are sampled from

$$P(x|j) = \mathcal{N}(x; \mu_j, \Sigma_j)$$

Estimate w_j, μ_j, Σ_j for $j = 1, 2 \cdots k$

$$w_j = \frac{N_j}{N} \quad . \ N_j : \text{\# of points in } B_j$$

$$\mu_j = \frac{1}{N_j} \sum_{x_n \in B_j} x_n \quad \text{(sample mean)}$$

$$\Sigma_j = \frac{1}{N_j} \sum_{x_n \in B_j} (x_n - \mu_j)(x_n - \mu_j)^T$$

(sample covariance)

Subproblem 2. (E step)

Given $\Omega = \{w_j, \underline{\mu}_j, \Sigma_j\}_{j=1}^{k}$, estimate bump membership. ie. for each $\underline{x}_n$, find bump j that is most likely to produce $\underline{x}_n$.

$$\underline{x}_n \in B_{j^*} \quad \text{if} \quad j^* = \underset{j \in \{1,2,\cdots k\}}{\operatorname{argmax}} \; P_r(j|\underline{x}_n)$$

Recall: Bayes' Thm

$$P_r\{j|\underline{x}_n\} = \frac{P_r\{\underline{x}_n|j\} \cdot P_r\{j\}}{P_r(\underline{x}_n)} = \frac{P_r\{\underline{x}_n|j\} \cdot P_r\{j\}}{\sum_{i=1}^{k} P(\underline{x}_n|i) \cdot P_r(i)} = \frac{\mathcal{N}(\underline{x}_n; \underline{\mu}_j, \Sigma_j) \cdot w_j}{\sum_{i=1}^{k} \mathcal{N}(\underline{x}_n; \underline{\mu}_i, \Sigma_i) \cdot w_i}$$

$$\Rightarrow j^* = \underset{j}{\operatorname{argmax}} \; \mathcal{N}(\underline{x}_n; \underline{\mu}_j, \Sigma_j) \cdot w_j$$

EM algo. Summary: (Hard decisions)

1. Initialization: start with arbitary bump membership for each $\underline{x}_n$.
2. Estimate $\Omega = \{w_j, \underline{\mu}_j, \Sigma_j\}_{j=1}^{k}$ given bump membership $\{B_1, \cdots B_k\}$ (subproblem 1)
3. Estimate bump membership $\{B_1, \cdots B_k\}$ given Ω (subproblem 2)

Repeat step 2 & 3 until convergence.

Note: ① Convergence is guaranteed.

② $B_1, \cdots, B_k$ are auxiliary ("hidden" variables).