Machine Learning

Supervised Learning:

Training set $\longrightarrow S_n = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

$x_i \in X$

$y_i \in Y$

$S_r \subseteq \underbrace{X \times Y}_{\text{data space}}$

y_i : regression / classification $\{-1, 1\}$

$\hat{f}: X \longrightarrow Y$

$\hat{f}(x_i) \simeq y_i$ ability to represent the available data, we want our function to generalize to new unseen data.

Goal: given a training set, Learn input-output relation

assumptions $p(x,y) = \underbrace{p_x(x)}_{\text{marginal distribution uncertainty}} p(y|x)$ → foreach x, there is possible output $p(y|x)$

Joint-probability distribution Unknown

in Regression

$y = \underbrace{f(x)}_{\text{regression function}} + \varepsilon$ — random noise eg gaussian

we assume $\hat{f}$ will generalize well to future data

in Classification $p(1|x) = 1 - p(-1|x)$

LOSS FUNCTION quantifies how good is the prediction wrt real data

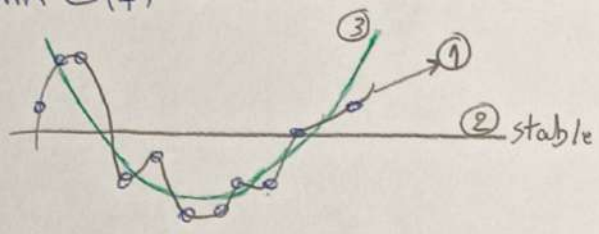
$\ell(y, f(x))$ point-wise error measure * how much we lose if we use the prediction with estimation $\hat{f}(x_i)$ not y_i

Expected Loss (expected risk)

$$\mathcal{E}(f) = \mathbb{E}[\ell(y, f(x))] = \int \underbrace{p(x,y)}_{\text{joint probability}} \cdot \ell(y, f(x))\, dx dy$$

best function = target function = f^* that minimizes expected risk.

$f^* = \text{argmin } \mathcal{E}(f)$ one that minimize $\mathcal{E}$



function: ① error = 0 → too complex → not stable → overfitting

② too stable → underfitting

③ compromise

$\hat{f}_s$: depend on Training set

excess expected Loss ⇓ we want it small

$$\mathcal{E}(f^*) - \mathcal{E}(f_s)$$

$$\mathbb{E}_s\left[\left|\mathcal{E}(f^*) - \mathcal{E}(f_s)\right|\right]$$

consistency $\lim \mathbb{E}_s\left[\left|\mathcal{E}(f^*) - \mathcal{E}(\hat{f}_s)\right|\right] = 0$

Good algorithm: ① Fitting: Fit data well ② Stability: should not change much if data change slightly

Regularization Parameter: controls tradeoff between data-fitting & stability

Empirical Risk

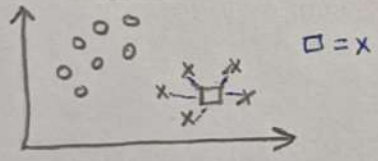
$$\hat{\mathcal{E}}(f) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, f(x_i))$$

Empirical Risk minimization ⇒ $\hat{f} = \text{arg min } \hat{\mathcal{E}}(f)$

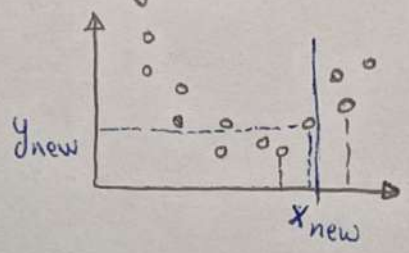
Local Methods

* close or near points in input $\xrightarrow{\text{expected}}$ close or near in output space

Binary classifi.



Regression



we assign to output of x_{new} same y value of closest point

NN: Nearest neighbor

$$y_{new} = \text{NearestNeighbor}(x_{new}, X_n, Y_n)$$

X_n $n \times d$

Y_n $n \times 1$

```
d = zeros(1, n)
for i = 0 .... n-1:
    d(i) = ||x_new - X_n(i,:)||
```

```
i_min = argmin(d)
Y_new = Y_n(i_min)
```

K-NN

Binary Classification

$Y_{new} = KNN(X_{new}, X_n, Y_n, K)$

$d = zeros(1, n)$

$O(nd)$ [for $i = 0 \dots n-1$

$d(i) = \| X_{new} - X_n(i,:) \|$

$O(n \log n)$ $[d_{sort}, i_{sort}] = argsort(d, ascend)$

$O(1)$ $Y_{new} = sign(sum(Y_n(i_{sort}^{0 \to K-1})))$

Regression

$Y_{new} = KNN(X_{new}, X_n, Y_n, K)$

$d = zeros(1, n)$

for $i = 0 \dots n-1$

$d(i) = \| X_{new} - X_n(i,:) \|$

$[d_{sort}, i_{sort}] = argsort(d, asc)$

$Y_{new} = average(Y_n(i_{sort}^{0 \to K-1}))$

KNN

$$f(x) = \frac{1}{K} \sum_{i=1}^{K} y_{j_i}$$

$K = odd$

bigger $K \to$ more stable

KNN depends on K

$f_{S,K}$ → hyper parameter of the method

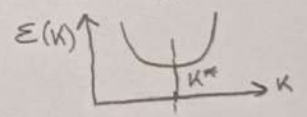
small $K \to$ overfitting

very Large $K \to$ very stable, disregard data, underfitting

behaviour of alg depend on K

best/optimal choice for hyperparameter K: The expected Loss $\mathcal{E}_K$

$$\mathcal{E}_K = \mathbb{E}_S \mathbb{E}_{x,y} (y - \hat{f}_{S,K}(x))^2$$



K^* minimize $\mathcal{E}_K$

$$K^* = \text{argmin } \mathcal{E}_K$$

NB: There is always some noise

$$\mathcal{E}(K) = \mathbb{E}_S \underbrace{\left(f^*(x) - \hat{f}_{SK}(x) \right)^2}_{\text{error term}} + \underbrace{\sigma^2}_{\text{noise}}$$

Idealized K-NN function

$$\tilde{f}_{SK}(x) = \frac{1}{K} \sum f^*(x_l)$$

Standard KNN func $\hat{f}_{SK}(x) = \frac{1}{K} \sum y_l$ → $f^*(x_l) + \delta_{noise}$

adding & subtracting to $\mathcal{E}(K)$ & simplifying ---

$$\mathcal{E}(K) = \underbrace{\left[f^*(x) - \mathbb{E}\,\tilde{f}_{sK}(x)\right]^2}_{\text{bias}} + \underbrace{\mathbb{E}_s\left[\tilde{f}_{sK}(x) - \hat{f}_{sK}(x)\right]^2}_{\text{variance}} + \sigma^2$$

$\mathcal{E}(K)$ = bias + variance

variance: $\mathbb{E}_s\left(\tilde{f}_{sK}(x) - \hat{f}_{sK}(x)\right)^2 + \sigma^2$

$$= \mathbb{E}_s\left[\frac{1}{K}\sum_{\ell \in K_x} f^*(x_\ell) - \frac{1}{K}\sum_{\ell \in K_x} y_\ell\right]^2 + \sigma^2 = \frac{1}{K^2}\mathbb{E}_s\left[\sum_{\ell \in K_x}\left(f^*(x_\ell) - y_\ell\right)^2\right] + \sigma^2$$

$$= \text{-----} = \frac{\sigma^2}{K} + \sigma^2$$

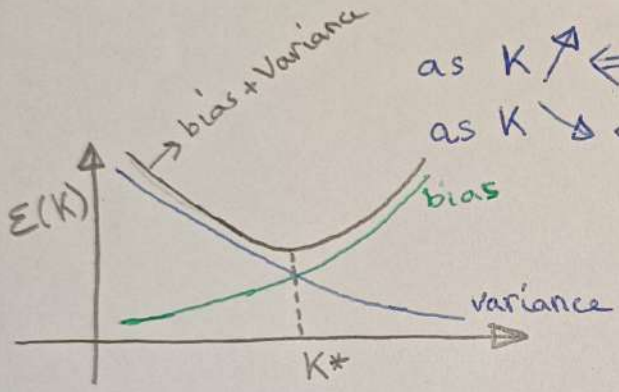
variance depend on K and noise

* Variance has an inverse relation w K

as K ↗ ⟺ variance ↘

as K ↘ ⟺ variance ↗

$\mathcal{E}(K)$ compromise between bias & variance



K^* optimal K

Underfitting → high bias
Overfitting → high variance

Notes : ChatGPT

Bias : * approximating a real-world problem which may be complex by a simple model
* high bias → underfitting → unable to capture patterns of data

Variance : * represents model sensitivity to small fluctuations/noise
* high variance → overfitting → model too complex and fitting training data well, but capturing noise
* overfitting → perform well on training data but fail to generalize to new unseen data

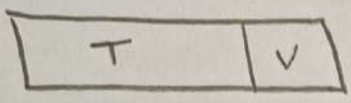
Goal : right balance

Bias → fitting
Variance → stability

Cross Validation

train on some data and validate on new unseen data

To choose K in practices



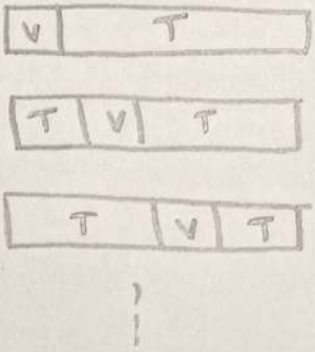
V usually randomly sampled from dataset

K = 1, 3 --- Kmax → for each K, execute procedure and estimate error

⊛ $K^* =$ cross-validation (T, V, Kmax)

↓ optimal K

```
i = 0
for K = 1;2; Kmax
    -get f̂_TK
    -error[i] ← error f̂_TK on V
    -i++
return K  // such that error is minimum
```



⊛ for each K

① Shuffle and split the data into T (training) & V (validation)

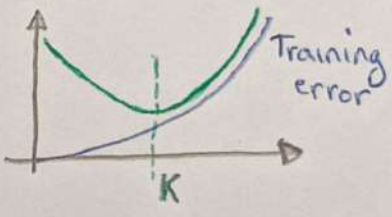
② Train algorithm on T

③ compute empirical loss on V

$$\hat{\varepsilon}_K = \frac{1}{|V|} \sum_{x,y \in V} \left(y - \hat{f}_{T,K}(x)\right)^2$$

we work on V ← $\frac{1}{|V|}$ $\sum_{x,y \in V}$ → because we compute on Validation set

④ Select $\hat{K}$ that minimize $\hat{\varepsilon}_K$



⊛ K-Fold = 5

$$\text{error} = \frac{1}{5} \sum_{i=1}^{5} \text{err}[i]$$

$V_1 \rightarrow$ err[1]

$V_2 \rightarrow$ err[2]

⋮

$V_5 \rightarrow$ err[5]

Linear Models

ERM: empirical error → $\min_{w \in \mathbb{R}^d} \hat{L}(w) = \min \sum_{i=1}^{n} (y_i - w^T x_i)^2$

General Idea: $\hat{\mathcal{E}}(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i))$

proxy for expected error

$$\mathcal{E}(f) = \mathbb{E}[\ell(y, f(x))] = \int p(x,y)\, \ell(y, f(x))\, dx\, dy$$

$p(x,y)$ is unknown

Empirical Risk: $\min \hat{L}(w)$

(*) $w^T x + b = \sum_{i=1}^{n} w_i x_i + b$

function f defined by vector w

$$\hat{L}(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

(NB)
* NN is local
* Linear is global → new value even if far can change model

$$\nabla \hat{L}(w) = \sum_{i=1}^{n} 2(w^T x_i - y_i) x_i = 0$$

x_i is row of $\hat{X}$ matrix

∇ → gradient → allows to find w direction & rate of steepest increase in Loss function

* $L(w) = \|\hat{X}w - \hat{Y}\|^2$

$$\Rightarrow \nabla \hat{L}(w) = 2\hat{X}^T(\hat{X}w - \hat{Y}) = 0$$

$$\hat{X}^T\hat{X}\, w = \hat{X}^T Y$$

$$w = \frac{\hat{X}^T\hat{Y}}{\hat{X}^T\hat{X}}$$

$\hat{X}^T\hat{X} = VSV^T$

must be invertible

$V \begin{pmatrix} \cdot & 0 \\ 0 & \cdot \end{pmatrix} V^T \underset{w}{(\quad)} = \begin{pmatrix} X^T Y \\ \quad \end{pmatrix}$

$n > d$: Under parameterized (over determined)

$\underset{d \times d}{\hat{X}^T\hat{X}}\ \underset{d \times 1}{w} = \underset{d \times 1}{\hat{X}^T Y}$

$O(d^2 n + d^3)$

building of $\hat{X}^T\hat{X}$ complexity $O(d^2 n + d^3)$

multiplic. $X^T w / \hat{X}$ — inverting

$d > n$: Over parameterized

building $XX^T \to O(n^2 d + n^3)$

$\hat{X}\ \overbrace{\hat{X}^T c}^{w} = \hat{Y}$

$$c = \frac{\hat{Y}}{\hat{X}\hat{X}^T} \Rightarrow w = \hat{X}^T c$$

$n \times 1$

building $XX^T \to O(n^2 d + d^3)$

bw

$$w = \hat{X}^T(\hat{X}\hat{X}^T)^{-1}\hat{Y}$$

Pseudo Code

```
w = LS_Train (X̂, Ŷ)
n = X.shape[0]
d = X.shape[1]
if n ≥ d:
    w = X^T Y / X^T X
else:
    c = Ŷ / X̂ X̂^T
    w = X̂^T c
```

TIKHONOV REGULARIZATION

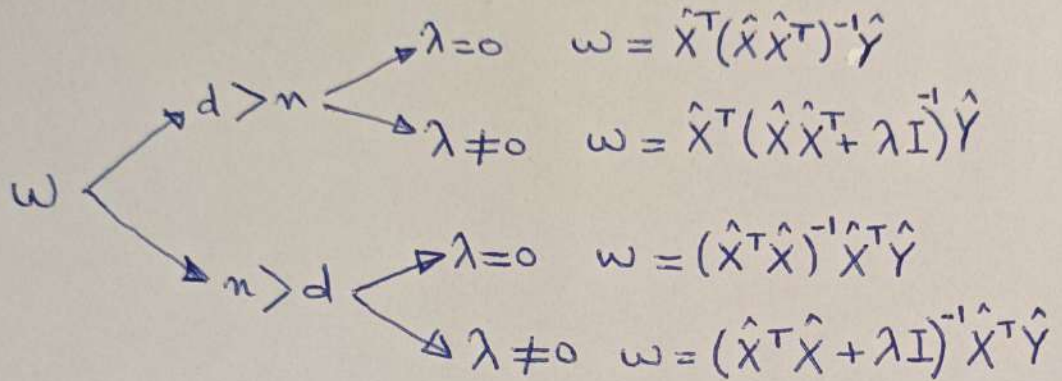
$\lambda > 0$ very small

$$\hat{X}\hat{X}^T = U \begin{pmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n} \end{pmatrix} U^T$$

add small amount to denominator

$$\begin{pmatrix} \frac{1}{\sigma_1+\lambda} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n+\lambda} \end{pmatrix}$$

* if λ is too big: ① matrix becom (0) $y_i = w^T x = 0$ and ② σ_i insignificant, all eigenvalues $\frac{1}{\lambda}$, matrix scales input



Regularized ERM

* $$L_\lambda(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|^2 \iff \|\hat{X}w - \hat{Y}\|^2 + \lambda\|w\|^2$$

penalty regularizer

$$\nabla L_\lambda(w) = 2\hat{X}^T(\hat{X}w - \hat{Y}) + 2\lambda w = 0$$

$$(\hat{X}^T\hat{X} + \lambda I)w = \hat{X}^T\hat{Y}$$

adding $\lambda\|w\|^2$ is called RIDGE REGRESSION

Emperical Risk Minimization: $\boxed{\min \hat{L}(w)}$

$\|w\|^2$: regulizer

Regulizer control → stability
→ prevent overfitting

λ balances error term & regulizer

RLS: Regularized Least Square $\min \hat{L}(w)$

w^* $\min_{w \in \mathbb{R}^D} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda\|w\|^2$

LOGISTIC REGRESSION

* Logistic loss function $l(y, f_w(x)) = \log(1 + e^{-y f_w(x)})$

y not binary

$$\frac{d}{da} \log(1+e^{-a}) p(1|x) + \log(1+e^{a}) p(-1|x)$$

$$= \frac{-e^{-a}}{1+e^{-a}} p(1|x) + \frac{e^{a}}{1+e^{a}} p(-1|x)$$

$$= \frac{-1}{1+e^{a}} p(1|x) + \frac{e^{a}}{1+e^{a}} p(-1|x) = 0$$

$$-p(1|x) + e^{a} p(-1|x) = 0$$

$$e^{a} = \frac{p(1|x)}{p(-1|x)} \star$$

$f(x)$

$$\Leftrightarrow \boxed{a_* = \text{Log} \frac{p(1|x)}{p(-1|x)}}$$

$\log(>1) \Leftrightarrow a > 0$
$\log(<1) \Leftrightarrow a < 0$

Log is monotonic increasing
so $a>0 \rightarrow p(1|x) > p(-1|x)$
$a<0 \rightarrow p(1|x) < p(-1|x)$

$\star \quad e^{a_*} = \frac{p(1|x)}{p(-1|x)}$

$$e^{a_*} = \frac{p(1|x)}{1-p(1|x)}$$

$$p(1|x) = e^{a_*}(1-p(1|x))$$

$$= \frac{1}{1+e^{-a_*}}$$

↘ Log

Binary $y \in \{-1, 1\}$

$$L(f) = \min L(f) = \int l(f(x), y) p(x, y) \, dx\,dy$$

$$= \int \underbrace{(l(f(x), y) p(y|x) dy}_{} p(x) dx$$

Let $L(f(x)) = L(a)$

$$L(a) = \int l(a, y) p(y|x) \, dy$$

$$= l(a,1) p(1|x) + l(a,-1) p(-1,x)$$

derive $L(a)$ where $l = (y-a)^2$

$$\frac{d}{da}\left[(1-a)^2 p(1|x) + (-1-a)^2 p(-1|x)\right] = 0$$

$$\Rightarrow -2(\,) p(1|x) - 2(\,) p(-1|x) = 0$$

$$\Rightarrow -a(p(1|x) + p(-1|x) + p(1|x) - p(\,)$$

$$\star \Rightarrow a_* = p(1|x) - p(-1|x)$$

optimal $a = a_*$

$$f(x) = a_* = \begin{cases} >0 & p(1|x) > p(-1|x) \\ <0 & p(-1|x) > p\text{---} \end{cases}$$

if $p(1|x) > p(-1|x)$ makes sense to predict 1

Same result

$p(1|x) + p(-1|x) = 1$

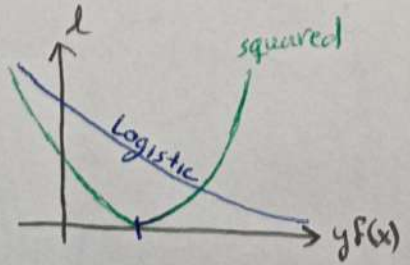
$\star \quad a_* = p(1|x) - p(-1|x)$

add $p(1|x)$ on both sides

$$a_* + p(1|x) = 2p(1|x) - p(-1|x)$$

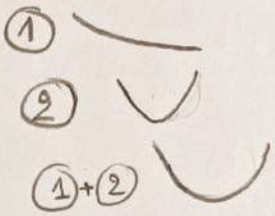
$$p(1|x) = \frac{a_* + p(1|x) + p(-1|x)}{2}$$

$$= \frac{a_* + 1}{2}$$



Logistic Regression Minimizer

Gradient Descent



$$L_\lambda(w) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \log\left(1+e^{-y w^T x_i}\right)}_{①} + \underbrace{\lambda \|w\|^2}_{②}$$

$$\nabla L_\lambda(w) = \underbrace{\frac{1}{n}\sum_{i=1}^{n} \frac{e^{-y_i w^T x_i}\cdot(-y x_i)}{1+e^{-y w^T x_i}}}_{a} + 2\lambda w = 0$$

$f = \log e^u = \frac{u'}{u}$

$f' = \frac{u'}{u}$

$a = F'(w)$

$$\textcircled{*}\ \boxed{w_{t+1} = w_t - \gamma F'(w_t)}$$

ex $w_1 = w_0 - \gamma F'(w_0)$

t = iteration

to memorize

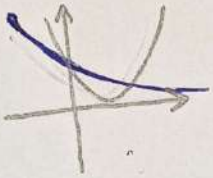
$w^T w \to 2w$

$-y_i x_i w^T = a w^T$

$$* \boxed{w_{t+1} = w_t - \gamma \nabla \hat{L}(w_t)}$$

* GD goes down by e^{-t}

squared is simpler than Logistic



* use newton's law (method) to know if we are close to minimum ??

$$w_{t+1} = w_t - \frac{F'(w_t)}{F''(w_t)}$$

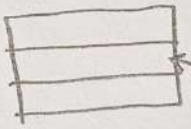
$w = w_{t+1} - w_t$

and $\gamma_t = \frac{1}{F''(w_t)} = \left(F''(w_t)\right)^{-1}$

newton's method is more computational costly

⊛ Gradient descent : * is an optimization alg. * goal to minimize Loss function * goal to find parameters (weight-)

optimization



compute ∇L in pieces

⊛ gradient provides direction of steepest Ascent

⊛ γ : Learning space → positive scalar → controls the size of the steps taken

---

Stochastic Gradient descent SGD used in Larg Datasets

* instead of computing gradient for entire dataset
→ compute gradient of loss for single point or small random subset
* frequent update → faster convergence
* model parameters are updated after procesing each data point or mini-batch

$$\boxed{w_{t+1} = w_t + \gamma_t \nabla L(w_t; x_i, y_i)}$$

$L(w_t; x_i, y_i)$: Loss function evaluated at i-th data point for w

Variations of gradient descent

- mini-batch
- Stochastic GD SGD
- Adam

⋮

SGD slower goes down by $\frac{1}{\sqrt{t}}$

Non-Linear Models

$\hat{Z} = \text{feature\_map}(\hat{X}, \text{"type"}, \text{parameter})$

$i = 1 \dots n$

Linear Model: $w^T x_i = y_i$

Non-Linear: such as $w^T x^2$

change of variable $x^2 = z \rightarrow w^T z$

now linear model of z

* $f(x) = ax^2 + bx = w^T z$

$z = (x^2, x)$ and $w = (a, b)$

* $\hat{Z} = \text{feature\_map}(\hat{X}, \text{"poly"}, \text{degree}=2)$

$i = 1 \dots n$

$z_i = x_i^2$

* $f(x) = ax^3 + bx^2 + cx + d$

$= w^T z$

$w = (a, b, c, d)$

$z = (x^3, x^2, x, 1)$

$\hat{Z} = \text{feature\_map}(\hat{X}, \text{"poly"}, \text{deg}=3)$

$i = 1 \dots n$

$z_i = (x_i^3, x_i^2, x_i, 1)$

* superposition of sin & cos $\sum_{j=1}^{p} c_j \sin(2\pi j x) = f(x) = w^T z$ (p → nb of frequencies)

$w^j = c^j \quad j = 1 \dots p$

$z^j = \sin(2\pi j x)$

* gaussian $\sum_{j=1}^{p} e^{-\frac{|x - m_j|^2}{2\sigma^2}} = f(x)$

m_j = mean

σ = std. dev

all models can be written as:

$$f(x) = \sum_{j=1}^{p} c_j \varphi_j(x)$$

φ : small phi

$\varphi_j : X \longrightarrow \mathbb{R} \quad j = 1 \dots p$

$x_i \rightsquigarrow z_i = (\varphi_1(x_i), \dots, \varphi_p(x_i))$

feature

$z = (\varphi_1(x) \dots \varphi_p(x)) = \Phi(x)$

and $\boxed{f(x) = w^T \Phi(x) = \sum_{j=1}^{p} w^j \varphi_j(x)}$

$w = (\hat{X}^T \hat{X} + n\lambda I) \hat{X}^T \hat{Y}$

$w = \hat{X}^T (\hat{X} \hat{X}^T + n\lambda I) \hat{Y}$

* $w = (\hat{\Phi}^T \hat{\Phi} + n\lambda I) \hat{\Phi}^T \hat{Y} \quad O(p^2 n + p^3)$

$w = \Phi^T \underbrace{(\hat{\Phi} \hat{\Phi}^T + n\lambda I) \hat{Y}}_{c} \quad O(n^2 p + n^3)$

$w = \Phi^T c$

Scales exponentially → too many coefficients

too many params → over parameterization

we choose this bcz p very big

Non-Linear Models

$\hat{Z} = \text{feature\_map}(\hat{X}, \text{"type"}, \text{parameter})$

$i = 1 \dots n$

Linear Model: $w^T x_i = y_i$

Non-Linear: such as $w^T x^2$

change of variable $x^2 = z \rightarrow w^T z$

now linear model of z

* $f(x) = ax^2 + bx = w^T z$

$z = (x^2, x)$ and $w = (a, b)$

* $\hat{Z} = \text{feature\_map}(\hat{X}, \text{"poly"}, \text{degree}=2)$

$i = 1 \dots n$

$z_i = x_i^2$

* $f(x) = ax^3 + bx^2 + cx + d$

$= w^T z$

$w = (a, b, c, d)$

$z = (x^3, x^2, x, 1)$

$\hat{Z} = \text{feature\_map}(\hat{X}, \text{"poly"}, \text{deg}=3)$

$i = 1 \dots n$

$z_i = (x_i^3, x_i^2, x_i, 1)$

* superposition of sin & cos $\sum_{j=1}^{p} c_j \sin(2\pi j x) = f(x) = w^T z$ (p → nb of frequencies)

$w^j = c^j \quad j = 1 \dots p$

$z^j = \sin(2\pi j x)$

* gaussian $\sum_{j=1}^{p} e^{-\frac{|x - m_j|^2}{2\sigma^2}} = f(x)$

m_j = mean

σ = std. dev

all models can be written as:

$$f(x) = \sum_{j=1}^{p} c_j \varphi_j(x)$$

φ : small phi

$\varphi_j : X \longrightarrow \mathbb{R} \quad j = 1 \dots p$

$x_i \rightsquigarrow z_i = (\varphi_1(x_i), \dots, \varphi_p(x_i))$

feature

$z = (\varphi_1(x) \dots \varphi_p(x)) = \Phi(x)$

and $\boxed{f(x) = w^T \Phi(x) = \sum_{j=1}^{p} w^j \varphi_j(x)}$

$w = (\hat{X}^T \hat{X} + n\lambda I) \hat{X}^T \hat{Y}$

$w = \hat{X}^T (\hat{X} \hat{X}^T + n\lambda I) \hat{Y}$

* $w = (\hat{\Phi}^T \hat{\Phi} + n\lambda I) \hat{\Phi}^T \hat{Y}$ $\quad O(p^2 n + p^3)$

$w = \Phi^T \underbrace{(\hat{\Phi} \hat{\Phi}^T + n\lambda I) \hat{Y}}_{c}$ $\quad O(n^2 p + n^3)$

$w = \Phi^T c$

Scales exponentially → too many coefficients

too many params → overparameterization

we choose this bcz p very big

Kernels

Representer Theorem:

$$f_\lambda(x) = \sum_{i=1}^{n} \Phi^T(x_i)\,\Phi(x)\,c_i \qquad K(x_i, x) = \Phi^T(x_i)\,\Phi(x)$$

KRLS_PREDICT $$\boxed{f_\lambda(x) = \sum_{i=1}^{n} K(x_i, x)\,c_i}$$

also $$w = \Phi^T\underbrace{(\Phi\Phi^T + n\lambda I)\hat{Y}}_{c = (\Phi\Phi^T + n\lambda I)\hat{Y}} \Rightarrow \boxed{w = \Phi^T c}$$

KRLS_TRAIN $$\boxed{c = (\hat{K} + n\lambda I)\hat{Y}}$$

Kernels

① **Linear:** $K(x, x') = x^T x'$

ex or

$K(x_i, x_j) = x_i^T \cdot x_j$

$K(x, z) = x^T \cdot z$

② **Polynomial:** $K(x, x') = (x^T x' + 1)^d$

d = degree

ex or

$K(x, z) = (xz + 1)^p$

$K(x_i, x_j) = (x_i^T x_j + 1)^d$

③ **Gaussian:** $K(x, x') = e^{\frac{-\|x - x'\|^2}{2\sigma^2}}$

or $K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$

Kernel : ① Symmetric → $K(x, z) = K(z, x)$ for all $x, z \in \mathbb{R}^d$

② positive definite (p.d)

Neural Networks

Recap Linear $f_w(x) = w^T x \quad x \in \mathbb{R}^d$

best vector $w^* = \underset{w \in \mathbb{R}^d}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, w^T x_i) + \lambda \|w\|^2$

for Logistic regression, we use 'iterative method based on gradient descent

$w_0, \; w_t = w_{t-1} + \gamma \nabla G(w)$ (γ ↓ step)

non-Linear $\Phi: \mathbb{R}^d \to \mathbb{R}^p$ feature map to transform data to linear, adding new dimensions that allow us to create hyperplane that separate data

$w^* = \underset{w \in \mathbb{R}^p}{\text{argmin}} \frac{1}{n} \sum_{i=1} \ell(y_i, w^T \Phi(x_i)) + \lambda \|w\|^2$ solution now lives in P dimension

* P very large → Kernel trick offer efficient approach

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} c_i K(x, x_i)$$

solution is now collection of coefficients
c_i → we have n coeff.

Neural Networks

* Linearly combines input

* Activation Function: σ → Linear / non-Linear

$g(x) = \sum_{i=1}^{n} x_i$ $\quad \sigma$ = threshold

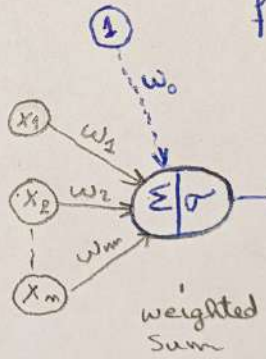
$f(x) = \sigma(g(x)) = \begin{cases} 1 & g(x) > \sigma \\ 0 & \text{otherw} \end{cases}$

y is a binary signal

* Single Layer: $Y = f(x)$

$f(x) = \sigma\left(\sum_{i=1}^{m} w_i x_i\right)$

if w_0 → or $f(x) = \sigma\left(\sum_i w_i x_i + w_0\right)$



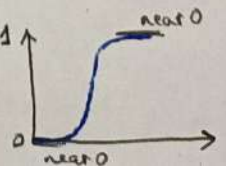
σ: activation func

* Activation Function: allow to account for non-Linearity in model
- take single nb and map it to different numerical value

(1) Sigmoid:

- corresponds to Logistic Regression
- non-zero centered output
- suffer from vanishing gradient
  * issue of vanishing gradient when the gradient becomes very small during back-propagation & weights of network updated very little and learning becomes slow

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



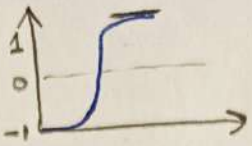
recap
→ gradient of loss function is a vector that points in the direction of steepest increase of loss
→ gradient provide info of how loss would change if we make small adjustments to param

② Tanh:

* it is a scaled version of sigmoid
* suffers from vanishing gradient problem
* output is zero centered (better gradient prop-)

the tanh function maps input x to output in range [-1,1]

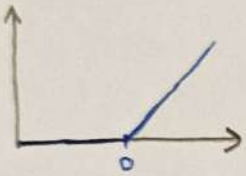
$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



scaled sigmoid = $2\sigma(2x) - 1$

③ ReLU (Rectified Linear Unit) good for images

$$f(x) = \max(0, x)$$



* RELU activation funct. returns x for any positive input and zero for any negative input Looks Linear for positive flat for zero
* it is simple to compute, making it computationally efficient during both forward & backward propagation
* ReLU does not saturate for positive values → which helps in vanishing gradient problem → accelerate training
* weight may irreversibly die

⊛ Dying Relu:

relu becomes inactive (zero output) for all inputs during training and fail to update weights → issue occur due to large gradient flow Relu network → cause to output zero in subsequent updates

Solution : Leaky Relu

④ Leaky ReLU:

* fix dying Relu problem
* introduce small, non-zero slope for negative inputs, allowing the pass of info when input is negative
* prevent neurons from being inactive by allowing small, non-zero gradient for negative input during backward pass

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$$

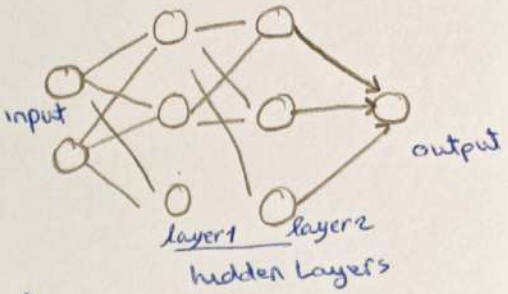
α is a small positive constant that determine slope of function for negative input

Activation functions so important because the introduce non-Linearity into the network.

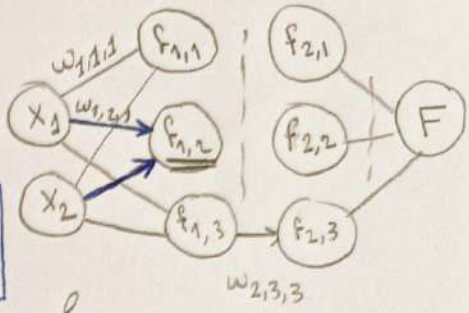
MultiLayer networks

chain of Layers

- Input Layer
- Multiple hidden Layers
- output Layer



* each element of Layer → neuron ; width = dimensionality of hidden Layers
  ↓ each neuron have activation function



$f_{layer,node}$

$w_{layer,node,input}$

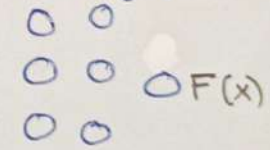
take $f_{1,2}$ $\quad f_{1,2} = \sigma\left[\left(\sum_{i=1}^{2} w_{1,2,i} \cdot x_i\right) + b_{1,2}\right]$

take $f_{2,1}$ $\quad f_{2,1} = \sigma\left[\left(\sum_{i=1}^{3} w_{2,1,i} \cdot f_{1,i}\right) + b_{2,1}\right]$

$$F(x) = \sigma\left(w_{3,1,1} \cdot f_{2,1} + w_{3,1,2} \cdot f_{2,2} + w_{3,1,3} \cdot f_{2,3} + b_{3,1}\right)$$

- Optimizing a neural network : determining the weights that are most appropriate for our problem solution
- different layers can have different Activation function
- same Layer should have same Activation function

How many parameters



18 weights
+ 7 biases
= 25 unknowns

input no bias

Training DNN

* means learning the values for the model parameters (w, bias) from Training Set → use loss func to estimate how much we loose predicting F(x)

Problem: the function we want to minimize is very complicated → difficult for gradient descent.

computing gradient for Complex! function ↓ use gradient descent:

$$w^* = \text{argmin} \frac{1}{n} \sum_{i=1}^{N} \ell(F(x_i, w), y_i)$$

$$w^* = \text{argmin}\, J(w)$$

$$w_{t+1} = w_t + \gamma \nabla J(w_t)$$

gradient = 0 ⇒ J(w) minimal
w^* when gradient = 0

Forward propagation: input flow into network & produce cost

⊛ Backward propagation: info flows back the net to compute gradient

⊛ Backpropagation
- minimize cost funct by adjusting weights & biases
- level of adjustment determined by gradients of cost funct.
- goal is to compute partial derivatives wrt any w or b.

entire dataset too big → we use mini-batch to train

$$w_{t+1} = w_t + \gamma \nabla J(w)$$

Gradient Descent Recap

- mini-batch
- stochastic
  - more computationally efficient
  - more iterations → smaller steps
  - more unstable because in some iterations might go opposite direction

Reverse Model

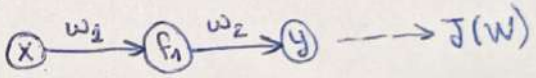
$$\frac{dF}{dx} = \frac{d}{dx}F(x) = \frac{d}{dx} f(g(h(u(v(x)))))$$

start from outer function (right to Left)

chain rule is essential part of DNN:
↳ goal: estimate the gradient for high dimensional spaces wrt each weight

DNN: dense NN

⊛ How does a small change in weight affect loss J(W)
↳ we compute gradient with respect to the nb of w's



ex $\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial y} \cdot \frac{\partial y}{\partial f_1} \cdot \frac{\partial f_1}{\partial w_1}$

Neural Networks → have lot of Local minima → very difficult to find global minimum

ex $\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial y} \cdot \frac{\partial y}{\partial w_2}$

* test, validation error against nb epochs
* we can add regularization term (λ) → cross-validation too costly - not desirable

Summary: we want to find parameters that minimize cost Function

$$w^* = \underset{w}{\operatorname{argmin}} \frac{1}{n} l(\hat{y}_i, y_i), \quad y_i = F(x_i, w)$$

Notes

DNNs → high nb of parameters → (high Variance) high chance of overfitting
⊛ high Variance → overfitting → poor response to noise, does not generalize well

other notes ⊛ high bias → underfitting

Terms: epoch: when entire dataset sent forward & backward through neural network only once

batch: training examples in a mini-batch

Recap: in DNN, we have lot of parameters → can result in overfitting → not well generalize, poor response to noise

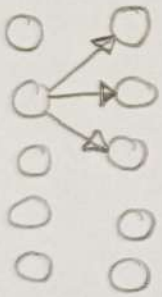
possible solution: Regularization to stabelize learning weights
- balanced ✓
- strong regularization → add significant bias → underfitting → not able to capture patterns
- week regularization → Lead to high variance → overfitting

Convolutional Neural Networks

CNN: special kind of NN with a known grid-like topology

CNN

2 properties: (1) Sparse interaction



NN: output of node is input to all next layer nodes

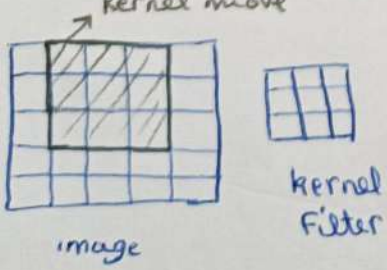
CNN: output of node is input to some neurons in next layer or input comes from a subset of nodes

positives of few parameters → Lower memory requirement, statistical efficiency reduced variance when estimating parameters, computations

(2) Parameter sharing:

in CNN: weight used at every position of the Input

Convolution Operation



Convolution between image & kernel

$$s(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

$s(i,j)$ → value of output (feature map) at position (i,j)

$(K*I)(i,j)$ → convolution operation b/w Kernel/Filter K and the input at position (i,j)

$K(m,n)$ → kernel value at position (m,n)

$I(i+m, j+n)$ → Image value at position $(i+m, j+n)$

(m,n): indices used in summation → dimensions of kernel

CNN: * nowadays, learn features using CNN
* Kernel: is a feature detector slides over inputs to generate feature Map.

ex image $200 \times 200 \times 3$ = size = 120000 weight

CNN: Input / convolution + Relu (activation funct) → pooling → convolution + Relu → pooling (Feature Learning)

Replace output at a Location with a summary statistic of nearby outputs

DNN: Classification [Flatten → Fully connected → Softmax $\frac{e^{y_i}}{\sum_j e^{y_i}}$] (classification)

Parameter sharing: as Kernel slides → able to capture same property in different image regions multiple feature detectors can be used to capture diff. properties

Convolution Layer

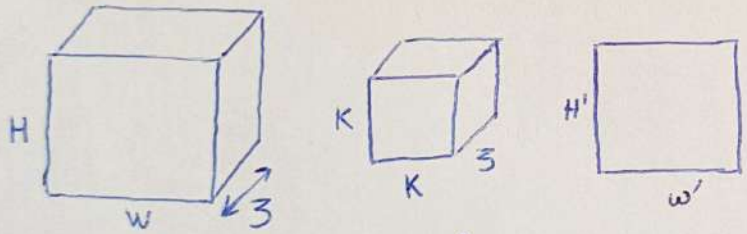
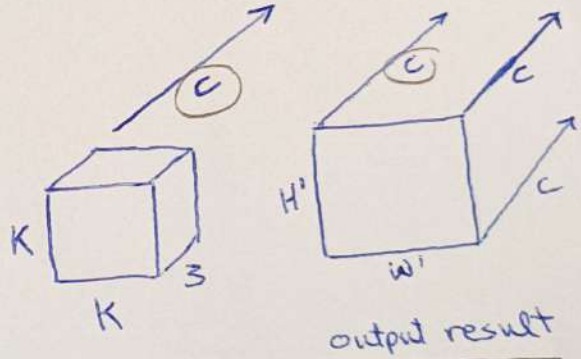


Image I with 3 channels

Kernel



output result

$\Rightarrow K \times K \times 3 \times c$

result $H' \times w' \times c$

we donot have 1 kernel, we have c kernels

* c = nb. of channels (kernels) in convolutional layer

notes terms

padding: → added outside the image → allows us to use the kernel in all pixels of the image → prevent reduction in spatial dimensions

adding extra border to input

size of output controled by:
1. **padding:** extra border to input added
2. **depth:** nb of kernels (Filters) of Layer
3. **Stride:** step used to slide kernel on input
   - >1 → down sampling input data → reduce spatial dimension output

Output Size $O = \frac{W - K + 2P}{S} + 1$

W: size before, K: kernel size, P: padding, S: stride, O: size after convolution

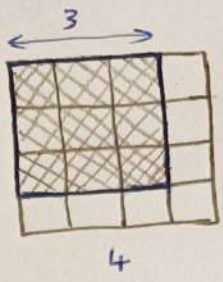
CNN

Output size after Convolution

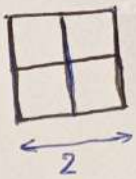
$$O = \frac{W - K + 2P}{S}$$

stride = 1

$$= \frac{4 - 3 + 0}{1} = 2$$



result



Pooling: Further reduce dimensionality

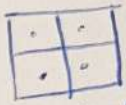
Pooling Functions → ① Average Pooling, ② Max Pooling

2	5	1	5
1	8	3	2
0	6	1	2
5	3	3	0

max pooling

8	5
6	3

Avg pooling



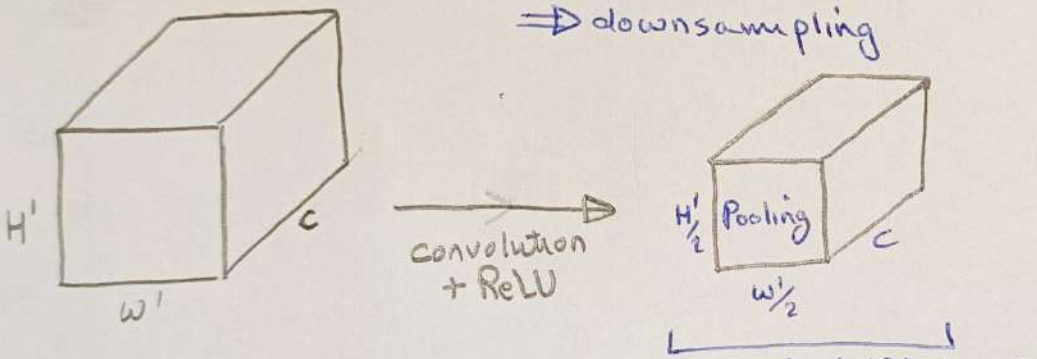
decimal averages

no parameters to be estimated

* suppose in a part of image we have special structure → kernel act with tolerance to it's position

pooling: after convolution, use pooling functions to aggregate elements of result grid (Max, Avg)

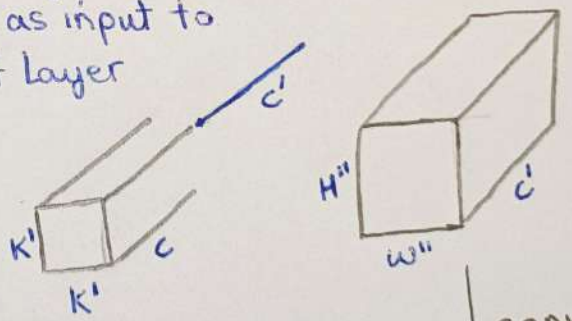
⇒ downsampling



output of this Layer used as input to next Layer

NB:

After CNN (feature recognition), it is common to attach a DNN (dense) for Classification.



convolution + Relu

pooling ...

* Classification → need Flattening operation to convert output of Conventional Layer to proper input for Fully connected Layer

* $\text{Softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_i}}$ so output can be interpreted as probability

* downsample image at different Levels → Low levels, Kernel Learn Low features (small details); high levels, Kernel Learn high features (ex: faces)

MLE MAXIMUM LIKELIHOOD ESTIMATION

* with Bayesian approaches, probability distribution are central objects of interest

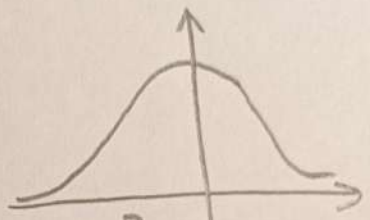
minimization of Loss function: $\min \sum_{i=1}^{n} \ell(f_w(x_i), y_i) \simeq \min \mathbb{E}[\ell(f(x), y)]$

MLE: $\min e^{-\sum_{i=1}^{n} \ell(f_w(x_i), y_i)}$

$(x_i, y_i)_{i=1}^{n} \sim P^{\otimes n}$

$z_i = (x_i, y_i) \in \mathbb{R}^{d+1}$

append



we assume P to be gaussian to reduce estimation to mean

Gaussian

* $P = P_\theta \quad \theta \in \mathbb{R}^p$ → space of parameters

$\theta(\mu, \sigma)$

search for θ that maximize the Likelihood

* $P_\theta(z) = \frac{1}{\sqrt{2\pi}} e^{-(z-\mu)^2/2\sigma^2}$

* if $\theta_1, \theta_2, \ldots, \theta_n$

$(z_1 \ldots z_n) \sim P_\theta(z_1 \ldots z_n) = P_{\theta_1}(z_1) \cdot P_{\theta_2}(z_2) \cdots P_{\theta_n}(z_n)$

Likelihood ← $\hat{\mathcal{L}}(\theta) = \prod_{i=1}^{n} P_\theta(z_i)$

Π : like Σ but for products

* $\mathcal{L}(\theta) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} e^{-(z-\mu)^2/2} \quad \text{where } \sigma = 1 = \frac{1}{(\sqrt{2\pi})^n} e^{-\sum_{i=1}^{n} (z_i-\mu)^2/2}$

note $\log(a e^z) = \log a + \log e^z = \log a + z$

* Instead of selecting θ & keep ones of highest $\mathcal{L}$

optimize: $\max \frac{1}{\sqrt{2\pi}} e^{-\sum_{i=1}^{n} (z_i-\theta)^2/2} \quad z \sim \mathcal{N}(\theta, 1)$

take the Log:

$\Rightarrow \max \log \hat{\mathcal{L}}(\theta) = \text{constant} - \sum_{i=1}^{n} \frac{(z_i-\theta)^2}{2}$

$\Rightarrow \max \sum_{i=1}^{n} (z_i - \theta)^2 / 2$ $\quad \therefore \quad -\frac{d \log \hat{L}(\theta)}{d\theta} = 0$

$= \sum_{i=1}^{n} (z_i - \theta) = 0 \Rightarrow n\theta = \sum_{i=1}^{n} z_i$

$$\hat{\theta} = \frac{1}{n} \sum_{i=1}^{n} z_i$$

* normal distribution is characterized by mean μ & standard dev σ

note mean: $\mu = \frac{1}{n} \sum_{i=1}^{n} x_i$

Pdf: probability dist. function: $f_{(\mu,\sigma)}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2}$

* Multivariant Guassian $\mathcal{N}$ = normal distribution

$z \sim \mathcal{N}(\theta, \sigma^2 I)$ *probability pdf of multivariate distr.

n set of gaussian dist.

* $z \in \mathcal{N}(\mu, \Sigma) \longrightarrow P_\theta(z) = \frac{e^{-(x-\mu)^T \Sigma^{-1} (x-\mu)}}{c}$

each with own mean & covariance Σ

$\mu \in \mathbb{R}^d$

$\Sigma \in \mathbb{R}^{d \times d} = \mathbb{R}^{d^2}$

$\theta(\mu, \Sigma) = \mathbb{R}^{d+d^2}$

how variables are correlated? Σ = covariance Matrix is a symm matrix describing the covariances between diff. dimensions of multivariate distr

This pdf of multivariate dist. describe the likelihood of observing vector x given mean μ & covariance Σ

Multivariate Gaussian dist. → describes joint distribution of multiple variables

μ is a mean vector

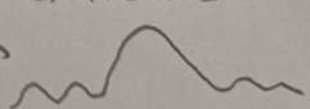
* case of n Guassians, each with it's own μ & σ

$$P_\theta(z) = \frac{1}{c} \sum_{j=1}^{n} e^{-(x-\mu_j)^T \Sigma^{-1} (x-\mu_j)}$$

GMM: Gaussian Mixture Model.

Tails: of distribution, less probable values found at extremes

far from mean → heavy tails, light tails



not heavy tails → tails decay rapidly → extreme events less likely → distribution concentration around mean

decay means value decrease as u move away from mean, values around mean more likely occur

MLE for Linear Regression

MLE benefits → we know the probability distribution → compute the confidence of our prediction

$y_i = w^T x_i + \varepsilon_i$ $\quad i = 1 \dots n$

ε_i: error term

* $p(y_i | x_i ; w)$

$y_i \sim \mathcal{N}(w_*^T x_i, \sigma^2)$

$= \mathcal{N}(w_*^T x_i, \sigma^2)$

$= c\, e^{-(y_i - w_*^T x_i)^2 / 2\sigma^2}$ $\quad c$ = constant

* $P_w(y_1 \dots y_n | x_1 \dots x_n ; w) = \prod_{i=1}^{n} c\, e^{-|y_i - w^T x_i|^2 / 2\sigma^2}$ $\quad P_w(\,)$

Likelihood func in regression $\mathcal{L}(w) = c' e^{-\sum_{i=1}^{n} |y_i - w^T x_i|^2 / 2\sigma^2}$ Likelihood function for observing (x_i, y_i) given w

$\text{Log } L(w) = \text{constant} - \text{expression}$ $\quad$ expression is minimum, Likelihood increase

This is the term inside the exponent → $\min \sum_{i=1}^{n} \frac{(y_i - w_*^T x_i)^2}{2\sigma^2}$ just Least square ERM

Maximize Likelihood equivalent to finding the w that minimize the sum of square errors $\Rightarrow \max \hat{\mathcal{L}}(w) \longrightarrow \min \sum_{i=1}^{n} \frac{|y_i - w_*^T x_i|^2}{2\sigma^2}$

finding w that maximize $\hat{\mathcal{L}}(w)$

Linear Logistic Regression

MLE → does not make sense for classify $y_i \in \{1, -1\}$

$\min \sum_{i=1}^{n} \text{Log}(1 + e^{-w^T x_i y_i})$ $\quad P_w(y_i | x_i)$

$p(1|x) = \frac{1}{1 + e^{-w^T x}}$ $\quad p(-1|x) = 1 - p(1|x) = \frac{e^{-w^T x}}{1 + e^{-w^T x}} = \frac{1}{1 + e^{w^T x}}$

$\hat{L}(w) = \prod_{i=1}^{n} P_w(y_i) = \prod_{i=1}^{n} \frac{1}{1 + e^{-w^T x_i y_i}}$

$\text{Log } \hat{L}(w) = \sum_{i=1}^{n} \log \frac{1}{1 + e^{-w^T x_i y_i}} = -\sum_{i=1}^{n} \log(1 + e^{-w^T x_i y_i})$

Prior and Posterior

in bayesian, λ is related to precision of prior $P(w)$

$$P(u|v) = \frac{P(v|u)\,P(u)}{P(v)}$$

$P(u|v)$: posterior; $P(v|u)$: likelihood; $P(u)$: prior; $P(v)$

prior = initial beliefs

posterior = updated beliefs

Likelihood = probability of observing data given parameters

Bayes theorem: $P(w|X_n, Y_n)$ represent the posterior dist. of coeff. w given X_n & Y_n

$$P(w|Y_n, X_n) = \frac{P(Y_n|X_n, w)\,P(w)}{P(Y_n)} \quad (P(w): \text{prior})$$

$$\Leftrightarrow \quad * P(Y_n|X_n, w) = e^{-\sum_{i=1}^{n}(y_i - w^T x)^2 / 2\sigma^2}$$

$$* P(w) = \frac{1}{c} e^{-\|w\|^2}$$

$$P(w|Y_n, X_n) = \frac{1}{c'} e^{-\sum_{i=1}^{n}\frac{(y_i - w^T x_i)^2}{2\sigma^2}} \cdot e^{-\|w\|^2}$$

$$= \frac{1}{c'} e^{-\left(\sum_{i=1}^{n}\frac{(y_i - w^T x_i)^2}{2\sigma^2} + \|w\|^2\right)}$$

full distribution of w

Maximum A Posterior

MAP

$$\max_{w \in \mathbb{R}^d} e^{-\left(\sum_{i=1}^{n}\frac{|y_i - w^T x_i|^2}{2\sigma^2} + \|w\|^2\right)}$$

map → parameter free

$\Updownarrow$

RLS

$$\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n}(y_i - w^T x_i)^2 + \lambda \|w\|^2$$

λ is param

- Ridge Regression can be derived as the Max. a Posteriori MAP for Linear regression
- Regularization Term in ridge regression determined by scaling Gaussian Prior
- Bayesian → allow computation of variance of parameter estimate → gives a more comprehensive view of parameter uncertainty

Sparsity

Linear model: $f_w(x) = w^T x = \sum_{j=1}^{d} w^j x^j$ $\quad x \to (x^1, \dots x^d)$

detect which variables are important for prediction ??

x^j: features of input x measures ---

which factors have determined our prediction

sparse: few of the coefficients are non-zero

$\|w\|_0 = \{j \mid w^j \neq 0\}$ count non-zero components in w

l_0 norm

X_n: $n \times D$ matrix

X^j: $\in \mathbb{R}^n$ $j = 1 \dots D$ columns

square Loss: $\min_{w \in \mathbb{R}^D} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_w(x_i))^2 + \lambda \|w\|_0$

l_0 norm

Y_n: $\in \mathbb{R}^n$ output vector

$\Rightarrow$ approach is hard considering all training sets with all subsets of variables

We consider 2 approaches:

Greedy Method OMP orthogonal matching Pursuit

MP: Forward Stage-Wise Regression

① initialize
- $r_0 = Y_n$ initial residual
- $w_0 = 0$ initial coefficient vector
- $I_0 = \emptyset$ initial index set

② iterate $i = 1 \dots T-1$

and compute correlation coefficients: a_j

$$a_j = \frac{(r_{i-1}^T x^j)^2}{\|x^j\|^2} \quad \text{for}$$

③ Select K: $K = \underset{j=1 \dots D}{\operatorname{argmax}} a_j$

note $v^j = \operatorname{argmin} \|r_{i-1} - X^j v\|^2$

variable selection with: ① Larger projection on output
② best explains output vector in Least square

④ Update Index: $I_i = I_{i-1} \cup \{K\}$ Union of previous iteration with K. K is index of best variable correlation

⑤ $\omega_i = \omega_{i-1} + \omega_K$ $\omega_K K = v_K \overrightarrow{e_K}$ (K^{th} element ≠ than zero)

coefficient Vector Update

ω_K is a vector with only K^{th} component different than 0. zero

⑥ Residual Update:

$$r_i = r_{i-1} - X_K \omega_K$$

Orthogonal MP Same but

⋮

⑤ coefficient update

$$\omega_i = \underset{\omega \in \mathbb{R}^D}{\operatorname{argmin}} \| Y_n - X_n M_{I_i} \omega \|^2$$

M_{I_i} acts as selector

M_i : D × D matrix

if j in set I
$\Rightarrow (M_i \omega)^j = \omega^j$

if j not in set I
$(M_i \omega)^j = 0$

if solution is sparse & data matrix has columns (features) not too correlated

$\Rightarrow$ OMP recovers with high probability the right vector of coefficients

LASSO and elastic net

LASSO : Least absolute shrinkage & selection operator

based on convex relaxation : use l_1 norm

$$\|w\|_1 = \sum_{j=1}^{D} |w_j|$$

another approach to find sparse solution

Least Squar:

$$w_\lambda = \min \frac{1}{n} \sum_{i=1}^{n} (y_i - f_w(x_i))^2 + \lambda \|w\|_1$$

LASSO : trade-off between complexity of model & fitting data well

objective function
* convex but not differentiable

convex function : if any 2 points in it's domain, the line connecting them lies above the graph of function

$\|w\|_1$ at zero | |

λ : - penality based on absolute value of coefficients
- promoting sparsity in solution
- λ control the tradeoff

Tools needed to find solution:

ISTA Iterative Soft Thresholding ALgorithm

$$w_0 = 0$$

$$w_i = S_{\lambda\gamma}\left(w_{i-1} - \frac{2\gamma}{n} X^T(Y - X w_{i-1})\right)$$

iteration $i = 1 \dots T_{max}$

$S_{\lambda\gamma}$: sub differential Soft thresholding

same

$$w_{t+1} = S_\lambda\left(w_t - \frac{2\gamma}{n} \hat{X}^T(\hat{X} w_t - \hat{Y})\right)$$

Iteration run until convergence is met:
* $\|w_i - w_{i-1}\| \leq \epsilon$ → precise
* max nb of iterations T_{max}

w_i : coeff parameter vector at iteration i

γ : step size (learning rate)

n : nb of data points

λ : reg param

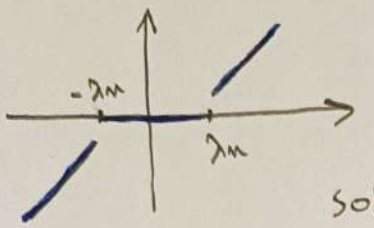
step size $\gamma = \dfrac{n}{2\|X^T X\|}$

* The choice of λ has significant impact on resulting model

λ big → increases penality & shrink more coefficients towards zero.
Less imp coefficients to zero (feature selection)

λ small → Less sparsity
→ Less regularization
→ potentially overfitting more prone to overfitting

$$S_\lambda = \begin{cases} w_j - \lambda n & w_j \geq \lambda n \\ 0 & -\lambda n \leq w_j \leq \lambda n \\ w_j + \lambda n & w_j \leq -\lambda n \end{cases}$$



soft-thresholding is a component-wise operation on w

$$S_\alpha(u) = ||u| - \alpha| + \frac{u}{|u|}$$

$|u| - \alpha|$: perform thresholding by
- if $|u|$ greater than α then $|u| - \alpha$
- if $|u|$ less than α then → zero

subs. α from each component of u

$\frac{u}{|u|}$: ensure the sign of each component in result is same (preserve sign)

* $w_j > \lambda n$ means magnitude of coeff. w_j larger than λn → soft-threshold operation result in w_j being shrunk to zero
* $-\lambda n < w_j < \lambda n$ magnitude between → soft thresholding will shrink w_j toward zero but not become zero
* $w_j < \lambda n$ - soft-th result w_j being pushed towards zero

Combination of these 2 terms result in soft thresholding effect.

Tikhonove → not sparse

LASSO → not stable (might not be stable)

⇒ hybrid algorithm between Tikhonov & Lasso

Elastic Net Algorithm

$$\min \frac{1}{n} \sum_{i=1}^{n} (y_i - f_w(x_i))^2 + \lambda (\alpha \|w\|_1 + (1-\alpha)\|w\|_2^2)$$

$\alpha \in [0,1]$

Decision Trees

partition on the input Space

$$\left\{ \underbrace{A_j \subseteq X,\ j=1 \dots J}_{\text{each } A_j \text{ is subset of input space } X} \;\middle|\; \underbrace{X = \bigcup_{j=1}^{J} A_j}_{\text{Union of all sets } A_j \text{ equal entire input space}},\ \underbrace{A_j \cap A_i = 0,\ i \neq j}_{\text{no elements in common}} \right\}$$

Dyadic Partition

* each cell A_j, side length of each cube 2^{-K}
* A_j is a cube * divide entire space into cubes.

* partition $\mathcal{A}$

partition based estimator learns a constant function

constant function: $\hat{c}_1, \dots, \hat{c}_j$ on each (respective) cell $A_1 \dots A_j$

$$\min_{c_1 \dots c_j} \sum_{j=1}^{j} \sum_{x_i \in A_j} (y_i - c_j)^2$$

we can compute each value of $\hat{c}_j$ separately: $\min_{c_j \in R} \sum_{x \in A_j} (y_i - c_j)^2$

→ to find the constant value for each cell that minimize the sum of squared differences

$j = 1, \dots, J$

$$* \hat{c}_j = \underset{c_j}{\operatorname{argmin}} \sum_{x_i \in A_j} (y_i - c_j)^2$$

and $\hat{c}_j = \frac{1}{n_j} \sum_{x_i \in A_j} y_i$ ($\hat{c}_j$: optimal constant)

n_j = nb of points in A_j cell

equivalently derived by piecewise constant function

$$\mathcal{H} = \left\{ f: X \to \mathbb{R} \;\middle|\; f(x) = \sum_{j=1}^{J} c_j \mathbb{1}_{A_j x},\ \forall x \in X, c_1 \dots c_j \in \mathbb{R} \right\}$$

($\mathcal{H}$: constant)

$$\mathbb{1} = \begin{cases} 1 & x \text{ is in cell } A_j \\ 0 & \text{otherwise} \end{cases}$$

ERM:

$$\min_{f \in \mathcal{H}_{const}} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

$$\min_{c_1 \dots c_j} \sum_{j=1}^{J} \sum_{x_i \in A_j} (y_i - c_j)^2$$

$$\Updownarrow$$

$$\min_{f \in \mathcal{H}_{const}} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

extend the idea beyond piecewise constant functions

∴ Piecewise Linear estimators

$$\mathcal{H}_{\substack{lin \\ linear}} = \left\{ f : X \to \mathbb{R} \;\middle|\; f(x) = \sum_{j=1}^{J} (w_j^T x + b_j) \mathbb{1}_{A_j x}, \; \forall x, w_j, \in \mathbb{R}^d \; b_j \in \mathbb{R}, j = 1 \dots J \right\}$$

ERM of $\mathcal{H}_{Lin}$: $\min \sum_{i=1}^{n} (y_i - f(x_i))^2$

Computation:

solution require solving Least square in each cell

* idea Can now be extended:

$$\mathcal{H} = \left\{ f : X \to \mathbb{R} \mid f(x) = \sum_{j=1}^{J} g(x) \mathbb{1}_{A_j x}, \; \forall x \in X, \; g \in G \right\}$$

G : features, kernels, Neural network ---

solution require solving a seperate ERM in each cell (local erm in each cell

ERM on $\mathcal{H}_G$: $\min_{f \in \mathcal{H}_G} \sum_{i=1}^{n} \ell(y_i, f(x_i))$

* How to select a partition?

Partition Tree

→ Family of partitions

$\mathcal{A}_0, \mathcal{A}_1, \dots \mathcal{A}_Q$ family of partitions

$\longrightarrow$ increasing cardinality

all space $\mathcal{A}_0 = \{X\}$

eg: $\mathcal{A} = \{ A_{1,2} ; A_{3,1} ; A_{3,2} ; A_{3,3} \dots \}$

each partition is a refinement of another partition

now for $A \in \mathcal{A}_q \longrightarrow c_1 \dots c_M$ s.t. $A = \bigcup_{j=1}^{M} c_j$ $\quad J_q \leq M^q$

* $\mathcal{A}_q$ is like a group of peaces (don't overlap) & together cover space
* each partition is a refinement of another partition

Select / Construct partition from partition tree approach

① Uniform Partitioning : the depth parameter q control level of detail → at each level of detail q, use cross validation to evaluate estimator performance → finally select level of best performance

scale param. q can be seen as regularization parameter

② Adaptive Partitioning recursively building a partition → refining a cell if the error in it is larger than threshold τ

let: $A \in \mathcal{A}_q \longrightarrow \hat{L}_A = \min_{f \in \mathcal{H}} \frac{1}{n_A} \sum_{x_i \in A} (y_i - f_A(x_i))^2$

A: cells in $\mathcal{A}_q$

n_A: number of input points in A

So if: $\hat{L}_A \leq \tau$ less or equal threshold τ

τ: threshold like Regularization parameter

⇒ we keep it in partition A (cell) kept in partition

Else $\hat{L}_A > \tau$ split A considering $c_1 \dots c_M$ smaller cells spliting

and $c_1, c_2 \dots c_M \subset \mathcal{A}_{q+1}$ → The next partition s.t. $A = \bigcup_{j=1}^{M} c_j$

Result partition contain cells of different size (refinement)

Greedy approach in building decision Tree

$\mathcal{A}(s,j)$: family of partitions made of 2 sets

$A_1(j,s)$ $\quad$ $A_2(j,s)$

$\{x \in X \mid x^j \leq s\}$ $\quad$ $\{x \in X \mid x^j > s\}$

j: j-th component of x

choose best parameters j and s that minimize

$$\min_{j,s} \left[L_{A_1(s,j)} + L_{A_2(s,j)} \right]$$

$\Rightarrow$ get pair j_*, s_* solve above problem

we get/set $\mathcal{A}_1 = \mathcal{A}(s_*, j_*) \rightarrow$ family of partitions made of 2 cells

$A_{1,1}$ $\quad$ $A_{1,2}$

Recursively apply the above scheme to $A_{1,1}$ & $A_{1,2}$

$B_{2,1}(j,s)$
$\{x \in A_{1,1} \mid x^j \leq s\}$

$B_{2,2}(j,s)$
$\{x \in A_{1,1} \mid x^j > s\}$

$B_{2,3}(j,s)$
$\{x \in A_{1,2} \mid x^j \leq s\}$

$B_{2,4}(j,s)$
$\{x \in A_{1,2} \mid x^j > s\}$

Clustering

Supervised Learning: $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$ $x_i \in \mathbb{R}^d$ $y_i \in \mathbb{R}$

Unsupervised Learning: $S = \{x_1, \dots, x_n\}$

* Clustering: grouping collection of objects into subsets/clusters
* assume the knowledge on the number of clusters $K \leq n$

$k = C(x_i)$ encoder C^* — achieve goal of clustering

cluster Label $k = \{1, \dots, K\}$

- Encoder C^* is a function that assigns each x_i (data point) to a cluster of label k

assign similar or close data points to same cluster

Combinational Clustering $\longrightarrow$ $W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=K} \sum_{C(i')=K} d(x_i, x_{i'})$ — distance or dissimilarity

optimal Clustering assignment $C^* = \text{argmin } W(C)$ unfeasible in practice

K-means

Cluster Centroids: m_k $k = 1, \dots, K$ ① initialize

$\bigcup C_i = X$

$C_i \cap C_j = \Phi \quad \forall i \neq j$

$C(x_i) = \underset{1 \leq k \leq K}{\text{argmin}} \|x_i - m_k\|^2$ — cluster assignment for x_i ② assign data to centroids $c_1 \dots c_K$

$m_1 \dots m_K$ update ③ go to ② and repeat until stop condition

$C_i = \{ x \in X / \|x - m_i\| \leq \|x - m_j\| \quad \forall j \neq i \}$

distance from point x to mean of C_i $\leq$ than distance from x to other means

Lloyd's algorithm

Note

$m_K = \frac{1}{S_K} \sum x_i$

calculate mean centroid of data points assigned to cluster, then update m_K

K-means aim to minimize the function:

$$J(C,m) = \sum_{K=1}^{K} \sum_{C(i)=K} \|x_i - m_K\|^2$$

K = nb of clusters
C = set of data points cluster assignment
m_K = centroid of cluster K
x_i = data point assigned to cluster K

* this is Kmeans → iterative procedure → ① assign data points to nearest centroid
② update centroids

- No guarantee to find global minimum
- algorithm can converge to local minimum

Note: Quality of Solution depend on initial config. of centroids

This is why K-means is run multiple times with different random initialization → best solution of lowest cost selected

$$\min_{m_1 \dots m_K} \sum_{i=1}^{n} \min_{j=1 \dots K} \|x_i - m_j\|^2$$

Update (min) Assign (min)

K-means ++

Kmeans ++ improves initialization step to enhance chances of converging to better solution

way ① try maximize distance between means (1st mean random)

way ② assign probability to each point points far from rest have higher prob. then do lloyd's alg.

* first centroid is chosen as first point in permuted order (random selection from data points)

points chosen farther → Remaining centroids:
- calculate distance from existing centroids to all data points. let D(x) be shortest to nearest centroid
- calculate squared distances & compute probability for choosing new points as centers

→ repeat till all centroids chosen.

The probability dist. ensure that points with large squared distances (farther from existing centroids) are chosen

$$\text{argmax } P(x) = \frac{D(x)^2}{\sum_i D(x_i)^2}$$

pick the point with highest probability