CS 480
Personal Notes

Marcus Chan

Taught by Hongyang Zhang

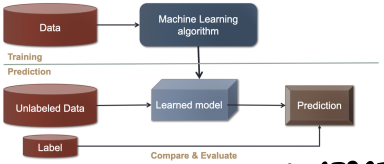
UW CS '25

Chapter 1: Perceptrons

ML

1 "Machine learning" is a branch of AI that focuses on methods that learn from data & make predictions on unseen data.

2 3 phases:
① training;
② prediction; &
③ evaluation.



PARADIGMS OF ML ALGOS (TRAINING)

1 "Supervised model": learning with labelled data (x, y)
eg email classification, image classification

2 "Unsupervised model": discover patterns in unlabeled data x
eg cluster similar data points, reduce data dimension
etc

3 "Semi-supervised model": using both labelled & unlabelled data

WHAT A DATASET LOOKS LIKE

	Training samples						Test samples	
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}'_1$	$\mathbf{x}'_2$
$\mathbb{R}^d \ni$ Feature	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
Label y	+	+	−	+	$\cdots$	−	?	?

- each column is a data point, n in total & each with d features
- y is the "label vector"
- x_1' & x_2' are the test samples whose labels need to be predicted.

(we use "x'" to denote test samples)

INNER PRODUCT: $\langle x, w\rangle$

Define the "inner product" of a & b to be

$$\boxed{\langle a, b\rangle = \sum_j a_j b_j ,}$$

where a_j, b_j are the j^{th} entries of a & b.

LINEAR FUNCTION

1 We say a function f is "linear" if

$$\boxed{f(\alpha x+\beta z) = \alpha f(x) + \beta f(z) \quad \forall \alpha,\beta \in \mathbb{R},\ x, z \in \mathbb{R}^d .}$$

2 Equivalently, f is linear iff there exists $w \in \mathbb{R}^d$ such that

$$\boxed{f(x) = \langle x, w\rangle = \sum_j x_j w_j .}$$

Proof. (⇒) let $w = [f(e_1), \dots, f(e_d)]$, where e_i is the i^{th} coordinate vector. Then

$$\begin{aligned} f(x) &= f(x_1 e_1 + \dots + x_d e_d) \\ &= x_1 f(e_1) + \dots + x_d f(e_d) \\ &= \langle x, w\rangle . \end{aligned}$$

(⇐) Note

$$\begin{aligned} f(\alpha x+\beta z) &= \langle \alpha x+\beta z, w\rangle \\ &= \alpha\langle x, w\rangle + \beta\langle z, w\rangle \\ &= \alpha f(x) + \beta f(z) . \end{aligned}$$ ◪

AFFINE FUNCTION

1 We say f is an "affine function" if there exists a $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ such that

$$\boxed{f(x) = \langle x, w\rangle + b \quad \forall x \in \mathbb{R}^d .}$$

SCORE: $\hat{y}$

1 Given $w \in \mathbb{R}^d$, $b \in \mathbb{R}$, define the "score" at some $x \in \mathbb{R}^d$ to be

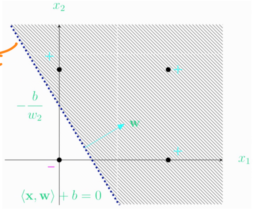
$$\boxed{\text{Score}(x) = \langle x, w\rangle + b .}$$

2 Our "prediction" for y is then

$$\boxed{\hat{y} = \text{sign}(\text{score}(x)) = \begin{cases} +1, & \text{score}(x) > 0 \\ -1, & \text{score}(x) \le 0 . \end{cases}}$$

We want to tune w, b so that "$\hat{y} = y$" for each x.

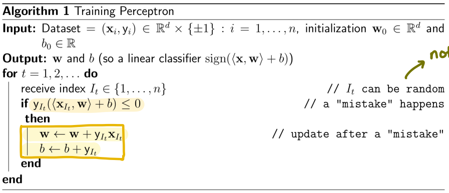
linear separator



- x is free, w & b fixed
- w & b uniquely determine the linear separator.

PERCEPTRONS

Algorithm for training:



Algorithm 1 Training Perceptron

```
Input: Dataset = $(\mathbf{x}_i, \mathrm{y}_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \dots, n$, initialization $\mathbf{w}_0 \in \mathbb{R}^d$ and
       $b_0 \in \mathbb{R}$
Output: $\mathbf{w}$ and $b$ (so a linear classifier $\mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b)$)
for $t = 1, 2, \dots$ do
    receive index $I_t \in \{1, \dots, n\}$                  // $I_t$ can be random
    if $\mathrm{y}_{I_t}(\langle \mathbf{x}_{I_t}, \mathbf{w} \rangle + b) \leq 0$          // a "mistake" happens
     then
        $\mathbf{w} \leftarrow \mathbf{w} + \mathrm{y}_{I_t}\mathbf{x}_{I_t}$                   // update after a "mistake"
        $b \leftarrow b + \mathrm{y}_{I_t}$
    end
end
```

note: we can just set $I_t = t$

- we typically set $w_0 = 0$ & $b_0 = 0$
- we only update after a mistake (aka "lazy update")
- note we are going through the data one by one.

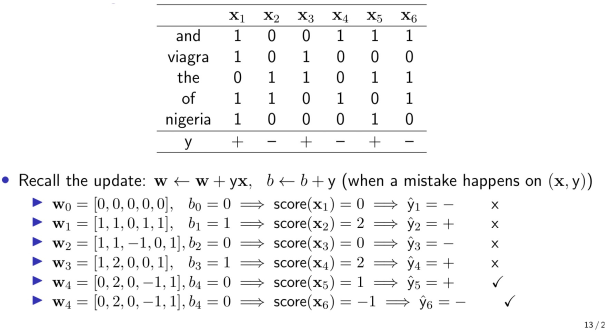
In particular, we want to find $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ such that for all $i = 1, \dots, n$,

$$\boxed{y_i(\langle x_i, w \rangle + b) > 0.}$$

Note that if a mistake happens on (x, y):

$$\begin{aligned} y[\langle x, w_{k+1} \rangle + b_{k+1}] &= y[\langle x, w_k + yx \rangle + b_k + y] \\ &= y[\langle x, w_k \rangle + y\langle x, x \rangle + b_k + y] \\ &= y[\langle x, w_k \rangle + y\|x\|_2^2 + b_k + y] \\ &= y[\langle x, w_k \rangle + b_k] + y^2\|x\|_2^2 + y^2 \\ &= y[\langle x, w_k \rangle + b_k] + \underbrace{\|x\|_2^2 + 1}_{\text{always positive \& } \geq 1.} \quad \because y = \pm 1 \end{aligned}$$

Example: spam filtering.



	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$
and	1	0	0	1	1	1
viagra	1	0	1	0	0	0
the	0	1	1	0	1	1
of	1	1	0	1	0	1
nigeria	1	0	0	0	1	0
y	+	−	+	−	+	−

- Recall the update: $\mathbf{w} \leftarrow \mathbf{w} + \mathrm{y}\mathbf{x}$, $\quad b \leftarrow b + \mathrm{y}$ (when a mistake happens on $(\mathbf{x}, \mathrm{y})$)
  - $\mathbf{w}_0 = [0, 0, 0, 0, 0]$, $\quad b_0 = 0 \implies \mathsf{score}(\mathbf{x}_1) = 0 \implies \hat{\mathrm{y}}_1 = -$ $\quad$ ×
  - $\mathbf{w}_1 = [1, 1, 0, 1, 1]$, $\quad b_1 = 1 \implies \mathsf{score}(\mathbf{x}_2) = 2 \implies \hat{\mathrm{y}}_2 = +$ $\quad$ ×
  - $\mathbf{w}_2 = [1, 1, -1, 0, 1], b_2 = 0 \implies \mathsf{score}(\mathbf{x}_3) = 0 \implies \hat{\mathrm{y}}_3 = -$ $\quad$ ×
  - $\mathbf{w}_3 = [1, 2, 0, 0, 1]$, $\quad b_3 = 1 \implies \mathsf{score}(\mathbf{x}_4) = 2 \implies \hat{\mathrm{y}}_4 = +$ $\quad$ ×
  - $\mathbf{w}_4 = [0, 2, 0, -1, 1], b_4 = 0 \implies \mathsf{score}(\mathbf{x}_5) = 1 \implies \hat{\mathrm{y}}_5 = +$ $\quad$ ✓
  - $\mathbf{w}_4 = [0, 2, 0, -1, 1], b_4 = 0 \implies \mathsf{score}(\mathbf{x}_6) = -1 \implies \hat{\mathrm{y}}_6 = -$ $\quad$ ✓

A TRICK TO HIDE THE BIAS TERM

Note that

$$\boxed{\langle x, w \rangle + b = \langle \underbrace{\begin{pmatrix} x \\ 1 \end{pmatrix}}_{x_{pad}}, \underbrace{\begin{pmatrix} w \\ b \end{pmatrix}}_{w_{pad}} \rangle}$$

This is a "trick" to ignore b in future calculations.

Thus, our new update rule is

$$\boxed{w_{pad} \leftarrow w_{pad} + y x_{pad}.}$$

CONVERGENCE THEOREM (LINEARLY SEPARABLE CASE)

Suppose there exists a w^* such that

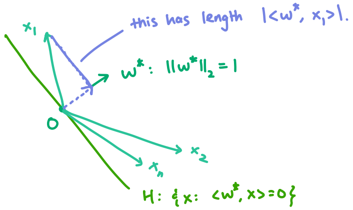
$y_i\langle x_i, w^*\rangle > 0 \quad \forall i=1,\dots,n.$

Assume $\|x_i\|_2 \le C \ \forall i$ and that w^* is normalized so that $\|w^*\|_2 = 1$.

Define the margin $\gamma := \min_i |\langle x_i, w^*\rangle|$.

Then the Perceptron algorithm converges after C^2/γ^2 mistakes.

Idea.



x_1
this has length $|\langle w^*, x_1\rangle|$.
w^*: $\|w^*\|_2 = 1$
O
x_2
x_n
H: $\{x: \langle w^*, x\rangle = 0\}$

- w^* is our "perfect" solution for w (ie the "goal" criteria is satisfied).
- thus, we want to show w "converges" to w^*.

Proof. Recall the update is $w \leftarrow w + yx$.

Define

$$\cos(w, w^*) = \frac{\langle w, w^*\rangle}{\|w\|\ \|w^*\|} = \frac{\langle w, w^*\rangle}{\|w\|}$$

(since we defined $\|w^*\| = 1$).

Consider an update and its effect on $\langle w, w^*\rangle$:

$$\begin{aligned}\langle w, w^*\rangle \longrightarrow\ & \langle w+yx, w^*\rangle \\ &= \langle w, w^*\rangle + \underbrace{y\langle x, w^*\rangle}_{\substack{\text{positive } \because\ w^* \text{ is} \\ \text{perfect}}} \\ &= \langle w, w^*\rangle + |\langle x, w^*\rangle| \\ &\geq \langle w, w^*\rangle + \gamma.\end{aligned}$$

This means for each update, $\langle w, w^*\rangle$ grows by at least $\gamma > 0$.

Similarly, consider an update's effect on $\|w\|_2^2$:

$$\begin{aligned}\|w\|_2^2 = \langle w, w\rangle \longrightarrow\ & \langle w+yx, w+yx\rangle \\ &= \langle w, w\rangle + \underbrace{2y\langle x, x\rangle}_{<0} + \\ &\quad y^2\langle x, x\rangle \\ &= \langle w, w\rangle + 2y\langle w, x\rangle + \underbrace{\|x\|_2^2}_{\le C^2} \\ &\le \langle w, w\rangle + C^2.\end{aligned}$$

This means for each update, $\langle w, w\rangle$ grows by at most C^2.

Now, let $w_0 = 0$. We now know after M updates:

$$\begin{aligned}\langle w_M, w^*\rangle &\geq \langle w_{M-1}, w^*\rangle + \gamma \\ &\geq \langle w_{M-2}, w^*\rangle + 2\gamma \\ &\geq \dots \geq \underbrace{\langle w_0, w^*\rangle}_{=0} + M\gamma \\ &= M\gamma.\end{aligned}$$

Similarly, note

$$\begin{aligned}\langle w_M, w_M\rangle &\le \langle w_{M-1}, w_{M-1}\rangle + C^2 \\ &\le \dots \le \underbrace{\langle w_0, w_0\rangle}_{=0} + MC^2 \\ &\le MC^2.\end{aligned}$$

Since

$$\cos(w, w^*) = \frac{\langle w, w^*\rangle}{\|w\|} \le 1 \implies \langle w, w^*\rangle \le \|w\|$$

Therefore

$$M\gamma \le \langle w, w^*\rangle \le \|w\| \le \sqrt{MC^2} = \sqrt{M}\,C.$$

Rearranging, this tells us that $M \le \frac{C^2}{\gamma^2}$, which finishes the proof. ☑

In particular, the larger γ is, the more separable the data is, and hence the faster the algorithm converges!

ANOTHER PERSPECTIVE ON PERCEPTRONS

Our hypothesis is $\hat{y} = \text{sign}\{\langle w, x\rangle\}$.

We can define our "loss function" as

$$\boxed{\begin{aligned} \ell(w; x_t; y_t) &= -y_t\langle w, x_t\rangle\, \mathbb{I}(\text{mistake on } (x_t, y_t)) \\ &= \begin{cases} -y_t\langle w, x_t\rangle, & \text{if mistake happens} \iff y_t\langle w, x_t\rangle < 0 \\ 0, & \text{otherwise} \end{cases} \\ &= -\min\{0, y_t\langle w, x_t\rangle\}. \end{aligned}}$$

The average of all the loss functions of the data points is then

$$\boxed{L(w) = -\frac{1}{n}\sum_{t=1}^{n} y_t\langle w, x_t\rangle\, \mathbb{I}[\text{mistake on } x_t].}$$

Our gradient descent update:

$$\boxed{w_{t+1} = w_t - \eta_t \nabla_w \ell(w_t, x_t, y_t) = w_t + \eta_t y_t x_t\, \mathbb{I}[\text{mistake on } x_t].}$$

If we set the step size $\eta_t = 1$, then

$$\boxed{w_{t+1} = w_t + y_t x_t},$$

which is our update rule.

PERCEPTRONS ARE NOT UNIQUE

Note perceptrons are not unique as the algorithm terminates as long as there is no mistake.

- it depends on initialization & our sampling rule of I_t.

MAXIMIZE MARGIN

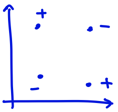
We want to choose w such that

$$\boxed{w = \max_{w:\, \forall i,\, \hat{y}_i y_i > 0}\ \min_{i=1,\dots,n} \frac{\hat{y}_i y_i}{\|w\|}, \quad \hat{y}_i := \langle x_i, w\rangle + b.}$$

XOR DATASET

There is no line that can separate + from −.

x_1	x_2	x_3	x_4
0	1	0	1
0	0	1	1
−	+	+	−



What if we run Perceptron?

Suppose $\exists w, b$ s.t. $y(\langle x, w\rangle + b) > 0$.

Then:

$x_1 = (0,0)$, $y_1 = -$ $\Rightarrow$ $b < 0$

$x_2 = (1,0)$, $y_2 = +$ $\Rightarrow$ $w_1 + b > 0$

$x_3 = (0,1)$, $y_3 = +$ $\Rightarrow$ $w_2 + b > 0$

$\left.\right\}$ $w_1 + w_2 + 2b > 0$

$x_4 = (1,1)$, $y_4 = -$ $\Rightarrow$ $w_1 + w_2 + 2b < 0$.

Hence

$$\underbrace{(w_1 + w_2 + 2b)}_{>0} - \underbrace{(w_1 + w_2 + b)}_{<0} = b > 0,$$

which contradicts our earlier statement that $b < 0$.

HARDNESS RESULT (NON-LINEARLY SEPARABLE CASE)

If there is no perfect separating hyperplane for our data, then the Perceptron algorithm cycles.

Chapter 2: Linear Regression

Idea: Given training data (x_i, y_i), find a $f: \mathcal{X} \to \mathcal{Y}$ such that $f(x_i) \approx y_i$, where

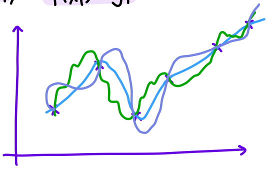
① $x_i \in \mathcal{X} \subseteq \mathbb{R}^d$: the feature vector for the i^{th} training example

② $y_i \in \mathcal{Y} \subseteq \mathbb{R}^t$: t responses

- note we could have $t=1$ or even $t=\infty$

Note for any finite training data (x_i, y_i), $i=1,\dots,n$, there exist infinitely many functions f such that for all i, $f(x_i)=y_i$.

eg



Moreover, our prediction $\hat{y}=f(x)$ can vary significantly on new data x!

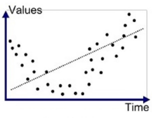
To choose f, we can

① leverage prior knowledge of f; &

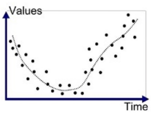
eg if x & y come from a population which follows "rules"

② choose the "simplest" function.

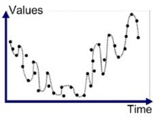
UNDERFITTING, GOOD FITTING, OVERFITTING



Underfitted



Good Fit/Robust



Overfitted

STATISTICAL LEARNING

We assume the training & test data are both iid samples from the same unknown distribution $\mathcal{P}$; ie

$$\boxed{\begin{array}{l}(X_i, Y_i) \sim \mathcal{P} \\ (X, Y) \sim \mathcal{P}.\end{array}}$$

LEAST SQUARES REGRESSION

We want to choose f so that

$$\boxed{f = \min_{f:\mathcal{X}\to\mathcal{Y}} \mathbb{E}\underbrace{\|f(X)-Y\|_2^2}_{\text{this is our least squared error.}} .}$$

REGRESSION FUNCTION: $m(x)$

💡1 Our "regression function" is

$$\boxed{f^*(x) = m(x) = \mathbb{E}[Y \mid X = x].}$$

💡2 However, calculating m requires us to know the distribution of $\mathcal{P}$, ie all pairs (X, Y).

💡3 We show that m is optimal; ie

$$\boxed{m(x) = \min_{f: \mathcal{X} \to \mathcal{Y}} E\|f(X) - Y\|_2^2 .}$$

Proof. First, see that

$$E\|f(X) - Y\|_2^2 = E\|f(X) - m(X) + m(X) - Y\|_2^2$$
$$= E\|f(X) - m(X)\|_2^2 + E\|m(X) - Y\|_2^2$$
$$+ 2E<f(X) - m(X), m(X) - Y> .$$

(using $\|a+b\|_2^2 \equiv \|a\|_2^2 + \|b\|_2^2 + 2<a,b>$)

Then

$$E_{X,Y}[<f(X) - m(X), m(X) - Y>]$$
$$= E_X[E_{Y|X}[<f(X) - m(X), m(X) - Y>]]$$

(by double expectation theorem, see STAT 330)

$$= E_X[<f(X) - m(X), m(X) - \underbrace{E[Y|X]}_{m(X)}>]$$
$$= E_X[<f(X) - m(X), 0>]$$
$$= 0.$$

Hence

$$E\|f(X) - Y\|_2^2 = E\|f(X) - m(X)\|_2^2 + \underbrace{E\|m(X) - Y\|_2^2}_{\text{noise (variance) term}} .$$

– independent wrt f.

Therefore, to reduce $E\|f(X) - Y\|_2^2$, we need to only minimize $E\|f(X) - m(X)\|_2^2$, which is minimal (ie $=0$) when $f = m$!

💡4 However, m is unaccessible since the conditional distribution is unknown, so we need to try to get close to m using the training data.

BIAS-VARIANCE TRADEOFF

💡1 Let f_D be the regressor learned on the training dataset D. Then

$$\boxed{\begin{aligned}\underbrace{E_{D,X,Y}\|f_D(X) - Y\|_2^2}_{\text{test error}} &= \underbrace{E_X\|E_D[f_D(X)] - m(X)\|_2^2}_{\text{bias}^2} \\ &+ \underbrace{E_{D,X}\|f_D(X) - E_D[f_D(X)]\|_2^2}_{\text{variance}} \\ &+ \underbrace{E_{X,Y}\|m(X) - Y\|_2^2}_{\text{noise (variance)}}\end{aligned}}$$

Proof. We have shown

$$E_{X,Y}\|f_D(X) - Y\|_2^2 = E_X\|f_D(X) - m(X)\|_2^2$$
$$+ \underbrace{E_{X,Y}\|m(X) - Y\|_2^2}_{\text{noise - independent wrt } f_D} .$$

Taking E_D of both sides:

$$E_D E_{X,Y}\|f_D(X) - Y\|_2^2 = E_D E_X\|f_D(X) - m(X)\|_2^2$$
$$+ E_{X,Y}\|m(X) - Y\|_2^2 . \quad -①$$

Define $\bar{f}(X) = E_D[f_D(X)]$.

Idea: we can sample multiple f's from various samples D:

$$D_1 \sim \mathcal{P} \to f_{D_1}$$
$$\vdots$$
$$D_n \sim \mathcal{P} \to f_{D_n}$$

then we define $\bar{f}(X) = \text{avg } f_{D_i}(n)$.

Then

$$E_D E_X\|f_D(X) - m(X)\|_2^2$$
$$= E_{D,X}\|f_D(X) - \bar{f}(X) + \bar{f}(X) - m(X)\|_2^2$$
$$= E_{D,X}\|f_D(X) - \bar{f}(X)\|_2^2 + E_{D,X}\|\bar{f}(X) - m(X)\|_2^2$$
$$+ 2E_{D,X}<f_D(X) - \bar{f}(X), \bar{f}(X) - m(X)>.$$

Similarly, see that

$$E_{D,X}<\bar{f}(X) - f_D(X), m(X) - \bar{f}(X)>$$
$$= E_X E_D<\underbrace{m(X) - \bar{f}(X)}_{\text{constant wrt } D}, \bar{f}(X) - f_D(X)>$$
$$= E_X<m(X) - \bar{f}(X), \bar{f}(X) - \underbrace{E_D[f_D(X)]}_{\bar{f}(X)}>$$
$$= 0.$$

Expanding ① yields the result desired. ◼

💡2 In particular, as the model capacity increases,

① the bias term decreases (ie model is more expressively powerful); but

② the variance increases (ie model is less stable).

SAMPLING → TRAINING

1. In practice, we can only calculate the sample average, ie we find f so that

$$\boxed{f = \min_{f:\mathcal{X}\to\mathcal{Y}} \hat{E}\|f(X)-Y\|_2^2 := \frac{1}{n}\sum_{i=1}^{n}\|f(X_i)-Y_i\|_2^2 .}$$

2. However, as our training data size $n\to\infty$, $\hat{E}\to E$ & hopefully argmin $\hat{E}$ → argmin E.

LINEAR REGRESSION

1. In linear regression, our regression functions are "affine"; ie in the form

$$\boxed{f(x) = Wx + b, \quad W \in \mathbb{R}^{t\times d}, \quad b \in \mathbb{R}^{t}.}$$

- t = # of response parameters we want to predict
- d = # of input parameters

2. Again, we can use padding:

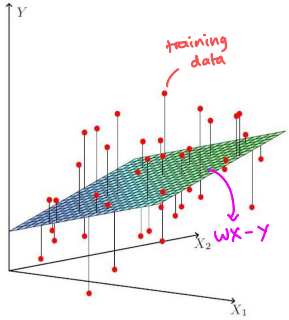
$$\boxed{x \leftarrow \begin{pmatrix} x \\ 1 \end{pmatrix}, \quad W \leftarrow [W, b] \Rightarrow f(x) = Wx}$$

3. In matrix form:

$$\boxed{\begin{gathered}\frac{1}{n}\sum_i \|f(x_i)-y_i\|_2^2 = \frac{1}{n}\|WX-Y\|_F^2 , \\ X \in [x_1,\dots,x_n] \in \mathbb{R}^{(d+1)\times n}, \quad Y = [y_1,\dots,y_n] \in \mathbb{R}^{t\times n}, \\ \|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2}\end{gathered}}$$

4. We want to find W such that

$$\boxed{W = \min_{W\in\mathbb{R}^{t\times(d+1)}} \frac{1}{n}\|WX-Y\|_F^2 .}$$



- geometrically, we want to minimise the sum of distances between the input training data & the resultant hyperplane.

SOLVING LINEAR REGRESSION

1. We define our loss function as

$$\boxed{\mathrm{Loss}(W) = \frac{1}{n}\|WX-Y\|_F^2}$$

2. Taking the derivative wrt W & setting to zero:

$$\boxed{\begin{aligned}\nabla_W \mathrm{Loss}(W) &= \frac{2}{n}(WX-Y)X^T \; (=0) \\ \Rightarrow WXX^T &= YX^T \\ \Rightarrow W &= YX^T(XX^T)^{-1}\end{aligned}}$$

PREDICTION

1. Once we have solved W on the training set (X, Y), we can predict on unseen data X_{test}:

$$\boxed{\hat{Y}_{test} = W X_{test}}$$

2. The "test error" (if true labels were available) is

$$\boxed{\text{test error} = \frac{1}{n_{test}}\|Y_{test} - \hat{Y}_{test}\|_F^2}$$

3. The "training error" is

$$\boxed{\text{training error} = \frac{1}{n}\|Y - WX\|_F^2 .}$$

4. We can minimize the training error to reduce the test error.

ILL-CONDITIONING

1. Consider $X = \begin{bmatrix} 0 & \varepsilon \\ 1 & 1 \end{bmatrix}$, $y = (1 \; -1)$. Solving linear least squares regression:

$$w = yX^T(XX^T)^{-1} = (1 \; -1)\begin{pmatrix} -1/\varepsilon & 1 \\ 1/\varepsilon & 0 \end{pmatrix}$$
$$= (-2/\varepsilon \;\; 1)$$

2. So slight perturbation leads to chaotic behavior!

3. This occurs when X is ill-conditioned: ie close to rank deficient.

ie
- two cols in X are close to linearly dependent
- but corresponding y's are different
- this is a contradiction ⇒ w becomes unstable.

RIDGE REGRESSION

💡1 **Idea**: We instead try to find

$$\boxed{W = \min_{W}\left[\frac{1}{n}\|WX-Y\|_F^2 + \lambda\|W\|_F^2\right]}$$

Why is this better?

Consider $\text{Loss}(W) = \frac{1}{n}\|WX-Y\|_F^2 + \lambda\|W\|_F^2$.

$$\Rightarrow \nabla_W \text{Loss}(W) = \frac{2}{n}(WX-Y)X^T + 2\lambda W \quad (=0)$$

$$\Rightarrow WXX^T - YX^T + \lambda n W = 0$$

$$WXX^T - YX^T + W(\lambda n I) = 0$$

$$WXX^T + W(\lambda n I) = YX^T$$

$$\therefore \quad W = (XX^T + \lambda n I)^{-1}(YX^T)$$

Then $XX^T + n\lambda I$ is far from rank-deficient matrices for large λ. (Proof uses SVD – see MATH 235).

💡2 λ controls our trade-off:

① $\lambda = 0$ reduces to ordinary linear regression;

② $\lambda = \infty$ reduces to $W \equiv 0$; &

③ intermediate λ restricts output to be $\frac{1}{\lambda}$ proportional to input.

💡3 Alternatively, note

$$\boxed{\frac{1}{n}\|WX-Y\|_F^2 + \lambda\|W\|_F^2 = \frac{1}{n}\|W[X \ \ \sqrt{n\lambda I}] - [Y \ \ 0]\|_F^2}$$

So we can also

① augment X with $\sqrt{n\lambda I}$; ie $\tilde{X} = (X \ \ \sqrt{n\lambda I})$

② augment Y with zeroes; ie $\tilde{Y} = (Y \ \ 0)$

(ie data augmentation) to achieve regularization.

Chapter 3: Logistic Regression

MOTIVATION

1. This is for linear classification.
2. We can use $|\langle x,w\rangle|$ (our margin) as a measure of our confidence in the prediction $\hat{y}$.
3. However, as this is un-normalized, it is hard to interpret.

MAXIMUM LIKELIHOOD ESTIMATE

1. We want to directly learn our "confidence"

$$\boxed{p(x;w) := P(Y=1 \mid X=x)}$$

2. Then, if $Y_1,\dots,Y_n, X_1,\dots,X_n$ are independent, then

$$\boxed{\begin{aligned} &P(Y_1=y_1,\dots,Y_n=y_n \mid X_1=x_1,\dots,X_n=x_n) \\ &= \prod_{i=1}^{n} P(Y_i=y_i \mid X_i=x_i) \\ &= \prod_{i=1}^{n} [p(x_i;w)]^{y_i}[1-p(x_i;w)]^{1-y_i} \quad \text{if } y=\{0,1\} \end{aligned}}$$

3. Maximizing the likelihood:

$$\boxed{\begin{aligned} &\max_w \prod_{i=1}^{n} [p(x_i;w)]^{y_i}[1-p(x_i;w)]^{1-y_i} \\ \Leftrightarrow\ &\min_w \sum_{i=1}^{n} [-y_i \log p(x_i;w) - (1-y_i)\log(1-p(x_i;w))] \end{aligned}}$$

4. We thus want to find w which satisfies the above optimization problem.

THE LOGIT TRANSFORM

1. If we assume the log of odds ratio is linear: ie

$$\boxed{\log \frac{p(x;w)}{1-p(x;w)} = \langle x,w\rangle}$$

* we can only perform logistic regression if we assume this!

then

$$\boxed{p(x;w) = \frac{1}{1+\exp(-\langle x,w\rangle)}}$$

↳ this is also called the "sigmoid transformation".

2. Plugging this into the earlier optimization problem, we want to find

$$\boxed{\min_w \sum_{i=1}^{n} \log[1+\exp(-\langle x_i,w\rangle)] + (1-y_i)\langle x_i,w\rangle}$$

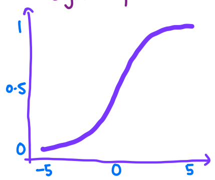
if $y_i \in \{0,1\}$.

3. If instead $y_i \in \{\pm 1\}$, then

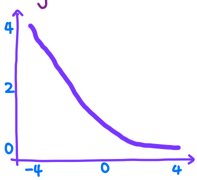
$$\boxed{\min_w \sum_{i=1}^{n} \log[1+\exp(-y_i\langle x_i,w\rangle)]}$$

↳ this is "logistic loss".

sigmoid function



logistic loss



TRAINING LOGISTIC REGRESSION

Our gradient descent algorithm is

$$\boxed{w \leftarrow w - \eta \nabla_w \text{Loss}(w)}$$

PREDICTION

1. We take

$$\boxed{\hat{y}=1 \iff P(Y=1 \mid X=x) > \tfrac{1}{2} \iff \langle x,w\rangle > 0}$$

2. Our decision boundary is still

$$\boxed{H := \{ x : \langle x,w\rangle = 0 \}}$$

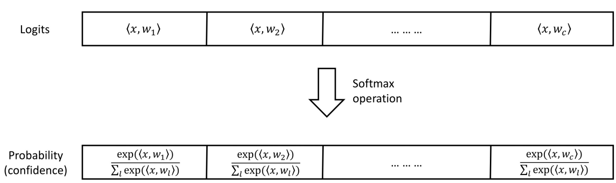
3. So we can predict $\hat{y} = \text{sign}(\langle x,w\rangle)$ as before, but now with confidence $p(x;w)$.

MULTI-CLASS EXTENSION

1. Idea: For a class $y \in \{1,\dots,c\}$, we want to learn $\{w_1,\dots,w_c\}$ for each class.
2. We consider the "softmax" function:

$$\boxed{P(Y=k \mid X=x,\ W=[w_1,\dots,w_c]) = \frac{\exp(\langle x,w_k\rangle)}{\sum_{\ell=1}^{c} \exp(\langle x,w_\ell\rangle)}}$$

- non-negative and sum to 1



Logits	$\langle x,w_1\rangle$	$\langle x,w_2\rangle$	………	$\langle x,w_c\rangle$
Probability (confidence)	$\frac{\exp(\langle x,w_1\rangle)}{\sum_l \exp(\langle x,w_l\rangle)}$	$\frac{\exp(\langle x,w_2\rangle)}{\sum_l \exp(\langle x,w_l\rangle)}$	………	$\frac{\exp(\langle x,w_c\rangle)}{\sum_l \exp(\langle x,w_l\rangle)}$

- we map a real-valued vector to a probability vector
- these are non-negative & sum to 1.

3. Training: again, we use MLE:

$$\boxed{\min_{w} E\left[-\log \frac{\exp(\langle X, w_y\rangle)}{\sum_{\ell=1}^{c} \exp(\langle X, w_\ell\rangle)} \right]}$$

4. Prediction:

$$\boxed{\hat{y} = \underset{k}{\text{argmax}}\ P(Y=k \mid X=x;\ W=[w_1,\dots,w_c]}$$

Chapter 4: Hard-Margin Support Vector Machines

INTRODUCTION

1. We assume $y = \{-1, +1\}$, and don't use padding.

2. Perceptron: we find <u>any</u> $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ such that

$$\boxed{\begin{aligned} &\min_{w,b} 0 \quad \text{s.t.} \quad y_i \hat{y}_i > 0 \quad \forall i, \\ &\qquad \hat{y}_i = \langle x_i, w \rangle + b \\ \Leftrightarrow\ &\min_{w,b} 0 \quad \text{s.t.} \quad y_i \hat{y}_i \geq 1 \quad \forall i \end{aligned}}$$

3. However, the larger the margin, the faster Perceptron converges.

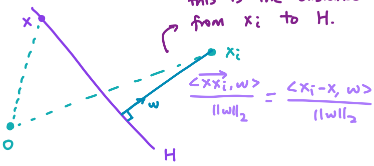
recall # mistakes, $M \leq \frac{C^2}{\gamma^2}$, $\|x_i\|_2 \leq C$, $\gamma = \min_i |\langle x_i, w^* \rangle|$, $\|w^*\|_2 = 1$.

4. So, the goal of hard-margin SVM is to maximize the margin <u>(assuming data is linearly separable)</u>.

DISTANCE FROM A POINT TO A HYPERPLANE

Let $H := \{x : \langle x, w \rangle + b = 0\}$. Then

$$\boxed{\begin{aligned} \text{distance}(x_i, H) &= \frac{|\langle x_i - x, w \rangle|}{\|w\|_2}, \qquad x \in H \\ &= \frac{|\langle x_i, w \rangle - \langle x, w \rangle|}{\|w\|_2} \\ &= \frac{|\langle x_i, w \rangle + b|}{\|w\|_2} \qquad \because x \in H \\ &= \frac{y_i \hat{y}_i}{\|w\|_2} \qquad \because y_i \hat{y}_i = 0 \end{aligned}}$$



x
x_i
w
O
H
$\frac{\langle \overrightarrow{xx_i}, w \rangle}{\|w\|_2} = \frac{\langle x_i - x, w \rangle}{\|w\|_2}$

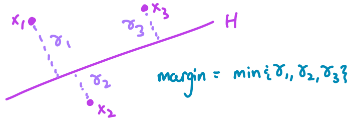
this is the distance from x_i to H.

MARGIN

1. We define the "margin" as the smallest distance to a separating hyperplane H among all separable training data; ie

$$\boxed{\begin{aligned} \text{margin} = \min_i \frac{y_i \hat{y}_i}{\|w\|_2} &= \min_i \frac{|\langle x_i, w \rangle + b|}{\|w\|_2}, \\ H = \{x : \langle x, w \rangle + b &= 0\} \end{aligned}}$$

eg



2. Our goal is to maximize the margin among all hyperplanes: ie find

$$\boxed{\max_{w,b} \min_i \frac{y_i \hat{y}_i}{\|w\|_2} \quad \text{s.t.} \quad y_i \hat{y}_i > 0 \quad \forall i}$$

TRANSFORMING TO STANDARD FORM

💡$_1$ Note for the margin, (w,b) & (cw, cb) has the same loss for $c>0$.

💡$_2$ So, we can fix the numerator arbitrarily to 1:

$$\max_{w,b}\left[\frac{1}{\|w\|_2} \quad \text{s.t.} \quad \min_i y_i \hat{y}_i = 1\right]$$

$$\Rightarrow \min_{w,b}\left[\frac{1}{2}\|w\|_2^2 \quad \text{s.t.} \quad y_i(\langle x_i, w\rangle + b) \geq 1 \quad \forall i\right]$$

COMPARISON TO PERCEPTRON

💡 Hard-margin SVM

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 \quad \text{s.t.} \quad y_i\hat{y}_i \geq 1 \quad \forall i$$

- quadratic programming
- unique solution
- maximal margin

Perceptron

$$\min_{w,b} 0 \quad \text{s.t.} \quad y_i\hat{y}_i \geq 1 \quad \forall i$$

- linear programming
- infinitely many solutions
- convergence rate depends on max margin

SUPPORT VECTORS

💡$_1$ Note that

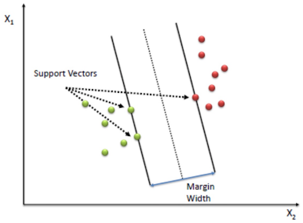
$$y_i\hat{y}_i \geq 1 \quad \forall i \iff \begin{array}{ll} \hat{y}_i \geq +1 & \forall i: \; y_i = +1 \\ \hat{y}_i \leq -1 & \forall i: \; y_i = -1 \end{array}$$

💡$_2$ This yields 3 parallel hyperplanes:

$$H = \{x : \langle x, w\rangle + b = 0\}$$

$$\left.\begin{array}{l} H^+ = \{x : \langle x, w\rangle + b = +1\} \\ H^- = \{x : \langle x, w\rangle + b = -1\} \end{array}\right\} \text{the "supporting" hyperplanes}$$

💡$_3$ "Support vectors" are those where points lie on the supporting hyperplanes.



LAGRANGIAN DUAL

💡$_1$ First, we show

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 \quad \text{s.t.} \quad y_i(\langle x_i, w\rangle + b) \geq 1 \quad \forall i$$

$$= \min_{w,b} \underline{\max_{\alpha>0}} \frac{1}{2}\|w\|_2^2 - \sum_i \alpha_i [y_i(\langle x_i, w\rangle + b) - 1]$$

$\alpha = [\alpha_1, \dots, \alpha_n] \in \mathbb{R}^n$;

$\alpha > 0 \Leftrightarrow \alpha_i \geq 0 \ \forall i$

Proof. Let Δ be the second expression.

See that

$$\Delta = \min_{w,b} \max_{\alpha\geq 0} \frac{1}{2}\|w\|_2^2 - \sum_i \alpha_i [y_i(\langle x_i, w\rangle + b) - 1]$$

If $\exists i$ s.t. $y_i(\langle x_i, w\rangle + b) < 1$, then if we set $\alpha_i = \infty$, it follows that $\Delta = +\infty$, which is the maximal value Δ can take.

Otherwise, ie if $\forall i$, $y_i(\langle x_i, w\rangle + b) \geq 1$, then

$$\Delta = \frac{1}{2}\|w\|_2^2 - \sum_i \underbrace{\alpha_i}_{+ve} \underbrace{[y_i(\langle x_i, w\rangle + b) - 1]}_{+ve}$$

$$\leq \frac{1}{2}\|w\|_2^2.$$

If we set $\alpha_i = 0 \ \forall i$, we get $\Delta = \frac{1}{2}\|w\|_2^2$, which is the max value Δ can take.

Therefore,

$$\Delta = \min_{w,b} \begin{cases} +\infty, & \text{if } \exists i \text{ s.t. } y_i(\langle x_i, w\rangle + b) < 1 \\ \frac{1}{2}\|w\|_2^2, & \text{otherwise} \end{cases}$$

$$= \min_{w,b} \frac{1}{2}\|w\|_2^2 \quad \text{if} \quad y_i(\langle x_i, w\rangle + b) \geq 1$$

as needed. □

💡$_2$ We can swap the min & max:

$$\max_{\alpha\geq 0} \min_{w,b} \frac{1}{2}\|w\|_2^2 - \sum_i \alpha_i [y_i(\langle x_i, w\rangle + b) - 1]$$

(because of "strong duality")

💡$_3$ Now, suppose we fix α, and consider the inner minimization problem.

Then w, b minimizes the function if $\frac{\partial}{\partial w} = \frac{\partial}{\partial b} = 0$.

Let $\text{Loss}(w,b) = \frac{1}{2}\|w\|_2^2 - \sum_i \alpha_i [y_i(\langle x_i, w\rangle + b) - 1]$.

$$\Rightarrow \frac{\partial}{\partial w} = w - \sum_i \alpha_i y_i x_i \ (=0), \quad \frac{\partial}{\partial b} = -\sum_i \alpha_i y_i \ (=0)$$

$$\rightarrow w = \sum_i \alpha_i y_i x_i, \quad \sum_i \alpha_i y_i = 0.$$

💡$_4$ Finally, we consider the "outer" maximization problem.

Plugging in our value of w above:

$$\Rightarrow \text{Loss}(\alpha) = \frac{1}{2}\|\sum_i \alpha_i y_i x_i\|_2^2 - \langle \sum_i \alpha_i y_i x_i, \sum_i \alpha_i y_i x_i \rangle$$

$$- b \underbrace{\sum_i \alpha_i y_i}_{0} + \sum_i \alpha_i$$

$$= -\frac{1}{2}\|\sum_i \alpha_i y_i x_i\|_2^2 + \sum_i \alpha_i \quad \text{s.t.} \quad \sum_i \alpha_i y_i = 0$$

💡$_5$ Thus, our problem becomes

★

$$\max_{\alpha\geq 0} \sum_i \alpha_i - \frac{1}{2}\|\sum_i \alpha_i y_i x_i\|_2^2 \quad \text{s.t.} \quad \sum_i \alpha_i y_i = 0$$

$$= \min_{\alpha\geq 0} -\sum_i \alpha_i + \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle x_i, x_j\rangle \quad \text{s.t.} \quad \sum_i \alpha_i y_i = 0$$

WHY USE THE DUAL FORM?

💡 Idea: If data is not linearly separable, we use a non-linear mapping ϕ to map the data.

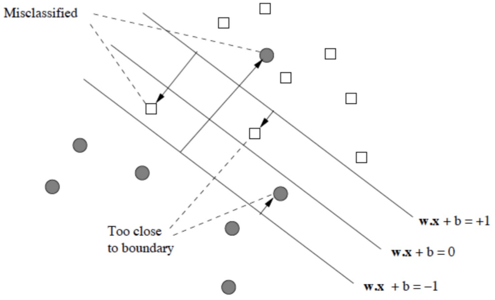
$$\min_{\alpha\geq 0} -\sum_i \alpha_i + \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j)\rangle$$

$$\text{s.t.} \quad \sum_i \alpha_i y_i = 0.$$

Chapter 5: Soft-Margin Support Vector Machines

MOTIVATION

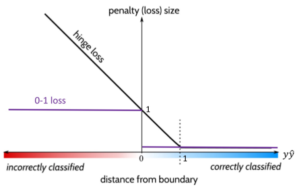
1. Hard-margin SVMs assume the data is linearly separable, but this is not always the case.
2. We want to adapt this to work for non-linearly separable data.
3. To do this, we will penalize our loss if the data falls too close to the boundary, or if the data is misclassified.



THE HINGE LOSS

1. We want to penalize the case where $y(<x,w>+b) < 1$, where $y = \{\pm 1\}$ is our true label, & $\hat{y} = <x,w>+b$ is our predicted confidence.
2. Define the "hinge loss function" to be

$$\boxed{\ell_{hinge}(y\hat{y}) = (1-y\hat{y})^{+} = \begin{cases} 1-y\hat{y}, & y\hat{y}<1 \\ 0, & \text{otherwise} \end{cases}}$$



* note: we define

$$\ell'_{hinge}(t) = \begin{cases} -1, & t<1 \\ 0, & t>1 \\ \alpha, & t=1, \end{cases}$$

where $\alpha \in [-1, 0]$

SOFT-MARGIN SVM

1. The "soft-margin SVM" balances between margin maximization & the hinge loss:

$$\boxed{\min_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_i \underbrace{(1-y_i\hat{y}_i)^{+}}_{\text{we penalize error \& small margin}}, \quad \hat{y}_i = <x_i, w> + b}$$

SOFT VS HARD-MARGIN SVM

1. For hard-margin SVM, we have a hard constraint that $y_i(<x_i,w>+b) \geq 1 \quad \forall i$.
2. For soft-margin SVM, we have a soft constraint; the more you deviate from the margin, the heavier the penalty.

WHY THE HINGE LOSS?

1. Our goal is to find

$$\boxed{\min_{x,w} P_{x,y}(\underset{\text{true label}}{Y} \neq \text{sign}(\underset{\text{predicted label}}{\hat{Y}})) = P(Y\hat{Y} \leq 0)}$$

where $Y \in \{0,1\}$, $\hat{Y} = <X,w>+b$.

2. This is equivalent to

$$\boxed{\min_{x,w} E[I(Y\hat{Y} \leq 0)] = \min_{x,w} E[\ell_{0-1}(Y\hat{Y})],}$$

where I is the indicator function, & ℓ_{0-1} is the 0-1 loss function.

- see diagram to the left for 0-1 loss.

BAYES RULE: $\eta(x)$

Given an instance x, the "Bayes rule" is defined to be

$$\boxed{\eta(x) = \underset{\hat{y}\in\mathbb{R}}{\operatorname{argmin}}\ E[\ell_{0-1}(Y\hat{y}) \mid X=x]}$$

Note that

$$\boxed{\begin{aligned}\eta(x) &= \underset{\hat{y}\in\mathbb{R}}{\operatorname{argmin}}\ E[I(Y\hat{y}\le 0) \mid X=x]\\ &= \underset{\hat{y}\in\mathbb{R}}{\operatorname{argmin}}\ Pr(Y\hat{y}\le 0 \mid X=x)\\ &= \underset{\hat{y}\in\mathbb{R}}{\operatorname{argmin}}\ Pr(Y\neq \operatorname{sign}(\hat{y}) \mid X=x)\end{aligned}}$$

Thus, Bayes rule attempts to minimize the inconsistency between the actual responses & the predicted responses.

CLASSIFICATION-CALIBRATED [LOSS]

We say a loss $\ell(y\hat{y})$ is "classification-calibrated" if for all x,

$$\boxed{\hat{y}(x) = \underset{\hat{y}\in\mathbb{R}}{\operatorname{argmin}}\ E[\ell(Y\hat{y}) \mid X=x]}$$

has the same sign as $\eta(x)$.

In particular, the convex loss ℓ is classification-calibrated iff

① ℓ is differentiable at 0; &

② $\ell'(0) < 0$.

Thus, the classifier that minimizes the expected hinge loss also minimizes the expected 0-1 loss.

LAGRANGIAN DUAL

Our soft-margin svm is

$$\boxed{\min_{w,b} \tfrac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}(1-y_i(<x_i,w>+b))^+}$$

Deriving the dual:

Apply $C\cdot(t_i)^+ = \max\{Ct_i, 0\} = \max_{0\le\alpha_i\le C} \alpha_i t_i$, and set $t_i = 1-y_i(<x_i,w>+b)$ to get

$$\min_{w,b}\ \max_{0\le\alpha\le C} \tfrac{1}{2}\|w\|_2^2 + \sum_{i=1}^{n}\alpha_i(1-y_i(<x_i,w>+b)),$$

$$0\le\alpha\le C \iff 0\le\alpha_i\le C \quad \forall i$$

We can swap min with max, since strong duality holds due to convexity:

$$\max_{0\le\alpha\le C}\ \min_{w,b} \tfrac{1}{2}\|w\|_2^2 + \sum_i \alpha_i[1-y_i(<x_i,w>+b)].$$

We can solve the inner unconstrained problem by setting derivative to 0:

$$\frac{\partial}{\partial w} = w - \sum_i \alpha_i y_i x_i \ (=0), \quad \frac{\partial}{\partial b} = -\sum_i \alpha_i y_i \ (=0)$$

$$\Rightarrow w = \sum_i \alpha_i y_i x_i, \quad b = \sum_i \alpha_i y_i = 0.$$

Substituting these values back into the outer maximization problem:

$$\max_{0\le\alpha\le C} \tfrac{1}{2}\Big\|\sum_{i=1}^{n}\alpha_i y_i x_i\Big\|_2^2 + \sum_{i=1}^{n}\alpha_i$$
$$- \underbrace{\sum_{i=1}^{n}\alpha_i y_i <x_i, \sum_{i=1}^{n}\alpha_i y_i x_i>}_{\|\sum_{i=1}^{n}\alpha_i y_i x_i\|_2^2} - \underbrace{\sum_{i=1}^{n} b\alpha_i y_i}_{0}.$$

$$= \max_{0\le\alpha\le C} \tfrac{1}{2}\Big\|\sum_{i=1}^{n}\alpha_i y_i x_i\Big\|_2^2 + \sum_{i=1}^{n}\alpha_i - \Big\|\sum_{i=1}^{n}\alpha_i y_i x_i\Big\|_2^2$$

$$= \max_{0\le\alpha\le C} \sum_{i=1}^{n}\alpha_i - \tfrac{1}{2}\Big\|\sum_{i=1}^{n}\alpha_i y_i x_i\Big\|_2^2$$

Thus, the dual form is

$$\boxed{\begin{aligned}&\max_{0\le\alpha\le C} \sum_{i=1}^{n}\alpha_i - \tfrac{1}{2}\Big\|\sum_{i=1}^{n}\alpha_i y_i x_i\Big\|_2^2 \quad \text{s.t. } \sum_i \alpha_i y_i = 0\\ =\ &\min_{0\le\alpha\le C} \tfrac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j <x_i,x_j> - \sum_i \alpha_i \quad \text{s.t.}\\ &\sum_i \alpha_i y_i = 0\end{aligned}}$$

Note that if

① $C\to\infty$, we get a hard-margin svm; &

② $C\to 0$, we get a constant classifier.

COMPLEMENTARITY SICKNESS

1 Let $\alpha^* t = \max_{0 \le \alpha \le C} \alpha t$, which we used in the dual proof.

2 Then note that

① $t > 0 \Rightarrow \alpha^* = C$, $\alpha^* = C \Rightarrow t \geq 0$

② $t < 0 \Rightarrow \alpha^* = 0$, $\alpha^* = 0 \Rightarrow t \leq 0$

3 If we let $t = 1 - y_i \hat{y}_i$, then

① $1 > y_i \hat{y}_i \Rightarrow \alpha_i^* = C$, $\alpha_i^* = C \Rightarrow 1 \geq y_i \hat{y}_i$

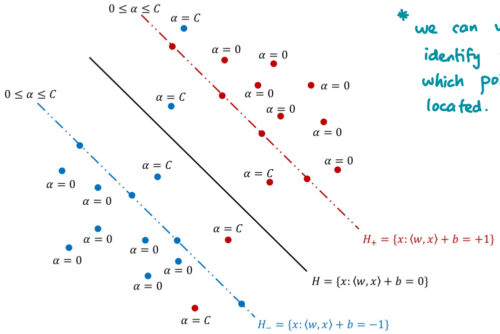
(ie margin/wrong idea)

② $1 < y_i \hat{y}_i \Rightarrow \alpha_i^* = 0$, $\alpha_i^* = 0 \Rightarrow 1 \leq y_i \hat{y}_i$

(ie correctly classified with good confidence)

③ $1 = y_i \hat{y}_i \Rightarrow 0 \leq \alpha_i^* \leq C$, $0 < \alpha_i^* < C \Rightarrow 1 = y_i \hat{y}_i$

(ie correctly classified on $H_{\pm 1}$)



* we can use α to identify the regions which points are located.

RECOVERING w & b FROM DUAL

1 We can obtain w & b via

$$\boxed{w = \sum_i \alpha_i y_i x_i .}$$

2 We also want to set C large enough so ≥ 1 point sits on one of $H_{\pm 1}$; ie $y_i \hat{y}_i = 1$.

- if C is too small, then $\alpha \approx 0$, so $w \approx 0$; then classifier is trivial

3 Then we can recover b via

$$\boxed{1 = y(\langle x, w\rangle + b) \Rightarrow b = y - \langle x, w\rangle}$$

Since $y = \pm 1$.

4 We can then predict new data via

$$\boxed{\hat{y} = \text{sign}(\langle x, w\rangle + b).}$$

Chapter 6: Reproducing Kernels

MOTIVATION

A lot of data are not linearly separable, and requires more complex classifiers.

QUADRATIC CLASSIFIER

1. The "quadratic classifier" has score function

$$\boxed{f(x) = \langle x, Qx \rangle + \sqrt{2} \langle x, p \rangle + b}$$

where $Q \in \mathbb{R}^{d \times d}$, $p \in \mathbb{R}^d$, $b \in \mathbb{R}$ are weights to be learned.

2. We can then predict via

$$\boxed{\hat{y} = \operatorname{sign}(f(x)).}$$

THE POWER OF LIFTING

1. We can express

$$\begin{aligned} f(x) &= \langle x, Qx \rangle + \sqrt{2} \langle x, p \rangle + b \\ &= \underbrace{\langle xx^T, Q \rangle}_{①} + \sqrt{2} \langle x, p \rangle + b \\ &= \underbrace{\langle \overrightarrow{xx^T}, \vec{Q} \rangle}_{②} + \sqrt{2} \langle x, p \rangle + b \\ &= \left\langle \begin{pmatrix} \overrightarrow{xx^T} \\ \sqrt{2} x \\ b \end{pmatrix}, \begin{pmatrix} \vec{Q} \\ p \\ 1 \end{pmatrix} \right\rangle \\ &= \langle \phi(x), w \rangle \end{aligned}$$

where $\phi(x) = \begin{pmatrix} \overrightarrow{xx^T} \\ \sqrt{2} X \\ b \end{pmatrix} \in \mathbb{R}^{d^2+d+1}$, $w = \begin{pmatrix} \vec{Q} \\ p \\ 1 \end{pmatrix} \in \mathbb{R}^{d^2+d+1}$

Aside:

① we define the inner product of 2 matrices to be: for $A = (a_{ij})_{d \times d}$, $B = (b_{ij})_{d \times d}$,

$$\langle A, B \rangle = \sum_{i,j} a_{ij} b_{ij}$$

② we define the vectorization of a matrix $A = (a_{ij})_{d \times d}$

$$\vec{A} = \begin{pmatrix} a_{11} \\ \vdots \\ a_{1d} \\ \vdots \\ a_{d1} \\ \vdots \\ a_{dd} \end{pmatrix} \in \mathbb{R}^{d \times d}$$

2. Thus, the quadratic classifier is linear wrt $\phi(x)$.

THE KERNEL TRICK

1. The feature map ϕ blows up the dimension.

2. But in the dual form of SVM, we only need to consider

$$\begin{aligned} \langle \phi(x), \phi(z) \rangle &= \left\langle \begin{pmatrix} \overrightarrow{xx^T} \\ \sqrt{2} x \\ 1 \end{pmatrix}, \begin{pmatrix} \overrightarrow{zz^T} \\ \sqrt{2} z \\ 1 \end{pmatrix} \right\rangle \\ &= \langle \overrightarrow{xx^T}, \overrightarrow{zz^T} \rangle + \langle \sqrt{2} x, \sqrt{2} z \rangle + 1 \\ &= \langle xx^T, zz^T \rangle + \langle \sqrt{2} x, \sqrt{2} z \rangle + 1 \\ &= (x^T z)^2 + 2(x^T z) + 1 \\ \therefore \langle \phi(x), \phi(z) \rangle &= (\langle x, z \rangle + 1)^2 \end{aligned}$$

3. Thus, the inner product in the higher dimensional space can be computed by the original vectors x & z.

- & we can calculate $\langle x, z \rangle$ in $O(d)$ time.

REPRODUCING KERNELS

1. We call $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a "reproducing kernel" if there exists some feature transform $\phi: \mathcal{X} \to \mathcal{H}$ such that

$$\boxed{\langle \phi(x), \phi(z) \rangle = k(x, z).}$$

2. Note that choosing ϕ uniquely determines k.

MERCER'S THEOREM

$k: \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ is a kernel iff for any $n\in\mathbb{N}$ and $x_1,\dots,x_n\in\mathcal{X}$, the kernel matrix K, where $K_{ij}=k(x_i,x_j)$, is symmetric & PSD.

Terms:

① "symmetric": $K_{ij}=K_{ji}$

② "positive semi-definite" / PSD:

$$\langle\alpha, K\alpha\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j K_{ij} \geq 0.$$

eg $k(x,z) = (\langle x,z\rangle+1)^p$ (polynomial kernel)

$k(x,z)=\exp(-\|x-z\|_2^2/\sigma)$ (Gaussian kernel)

$k(x,z)=\exp(-\|x-z\|_2/\sigma)$ (Laplace kernel)

REPRODUCING PROPERTIES

If k_1, k_2 are kernels, then

① λk_1 is a kernel $\forall\lambda\geq 0$;

② k_1+k_2 is a kernel; &

③ k_1k_2 is a kernel;

If (k_i) is a sequence of kernels, then their limit k, if it exists, is also a kernel.

KERNEL SVM

The kernel svm's primal form is

$$\boxed{\min_{w,b} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}(1-y_i\hat{y}_i)^+, \quad \hat{y}_i = \langle\phi(x_i), w\rangle}$$

and the dual form is

$$\boxed{\begin{aligned}&\min_{0\leq\alpha\leq C} -\sum\alpha_i+\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j k(x_i,x_j)\\ &\text{s.t. } \sum_i \alpha_i y_i = 0\end{aligned}}$$

where ϕ & k are related via Mercer's theorem.

ie $k(x_i,x_j) = \langle\phi(x_i), \phi(x_j)\rangle$

PREDICTION

Suppose that $0\leq\alpha^*\leq C$ optimizes the kernel SVM.

Then, we can recover

$$\boxed{w^* = \sum_{i=1}^{n}\alpha_i^* y_i \phi(x_i).}$$

Finally, our score function is

$$\boxed{\begin{aligned}f(x) &= \langle\phi(x), w^*\rangle\\ &= \langle\phi(x), \sum_{i=1}^{n}\alpha_i^* y_i\phi(x_i)\rangle\\ &= \sum_{i=1}^{n}\alpha_i^* y_i\, k(x,x_i),\end{aligned}}$$

which we can get the prediction from by taking the sign.

Chapter 7: Gradient Descent

MOTIVATION

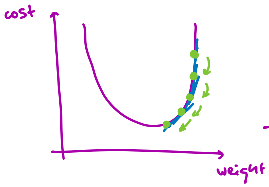
Many ML methods can be classed as optimization problems; ie

$$\boxed{f^* = \min_x f(x), \quad x^* = \text{value of } x \text{ that produces } f^*}$$

Assume f is differentiable with gradient $\nabla f(x)$.

Idea: Choose an initial point $x^{(0)} \in \mathbb{R}^n$ and iteratively calculate

$$\boxed{x^{(k)} = x^{(k-1)} - t \cdot \nabla f(x^{(k-1)})}$$



- negative gradient is steepest decreasing direction at that point
- so if function is convex, alg reaches minimizer

EXAMPLE: PERCEPTRON

For perceptron, our gradient descent is

$$\boxed{w \leftarrow w + t\left[\frac{1}{n}\sum_{i=1}^{n} y_i x_i \mathbb{I}[\text{mistake on } x_i]\right]}$$

Stochastic gradient descent update:

$$\boxed{\begin{gathered} w \leftarrow w + t y_I x_I \mathbb{I}[\text{mistake on } x_I], \\ I \text{ is random} \end{gathered}}$$

EXAMPLE: SOFT-MARGIN SVM

Gradient descent update for soft-margin SVM:

$$\boxed{\begin{aligned} w &\leftarrow w - t\left[\frac{w}{\lambda} + \frac{1}{n}\sum_{i=1}^{\wedge} \ell'_{\text{hinge}}(y_i\hat{y}_i)\, y_i x_i\right] \\ b &\leftarrow b - t\left[\frac{1}{n}\sum_{i=1}^{n} \ell'_{\text{hinge}}(y_i\hat{y}_i)\, y_i\right] \end{aligned}}$$

INTERPRETATION FROM TAYLOR EXPANSION

Note that if we take the Taylor expansion of f at y, we get

$$\boxed{f(y) \approx f(x) + \nabla f(x)^T (y-x) + \frac{1}{2t}\|y-x\|_2^2}$$

Hence

$$\boxed{\min_y f(y) \approx \min_y \underbrace{f(x) + \nabla f(x)^T (y-x) + \frac{1}{2t}\|y-x\|_2^2}_{L(y)}}$$

Then see that

$$\boxed{\begin{gathered} \frac{\partial L(y)}{\partial y} = 0 + \nabla f(x) + \frac{1}{t}(y-x) \quad (=0) \\ \Rightarrow y = x - t \cdot \nabla f(x) \end{gathered}}$$

and this is exactly the gradient descent template.

STEP SIZE

Note the step size cannot be too large or too small.

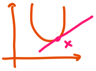
- too large: alg diverges
- too small: alg is too slow

So, we need to find t such that the algorithm converges nicely.

CONVEX FUNCTION

We say f is convex if for any $x, y \in \mathbb{R}^n$,

$$\boxed{f(y) \geq f(x) + \nabla f(x)^T (y-x)}$$



L-LIPSCHITZ CONTINUOUS

1 We say ∇f is "L-Lipschitz continuous" if $LI - \nabla^2 f(x)$ is positive semi-definite, denoted as $LI \succeq \nabla^2 f(x)$, at all $x \in \mathrm{dom}(f)$, where $L \in \mathbb{R}$.

2 Here,

$$\boxed{\nabla^2 f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1^2} & \frac{\partial f}{\partial x_1 x_2} & \cdots & \frac{\partial f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_n x_1} & \frac{\partial f}{\partial x_n x_2} & \cdots & \frac{\partial f}{\partial x_n^2} \end{pmatrix}}$$

3 In other words, we say f is "L-smooth".

CONVERGENCE ANALYSIS FOR CONVEX CASE

Let f be convex, differentiable & L-Lipschitz continuous for some $L \in \mathbb{R}$, with $\text{dom}(f) = \mathbb{R}^n$.

Then if we do gradient descent with fixed step size $t \leq \frac{1}{L}$, we get

$$\boxed{f(x^{(k)}) - f^* \leq \frac{\|x^{(0)} - x^*\|_2^2}{2tk}}$$

We say gradient descent has convergence rate $O(\frac{1}{k})$.

Proof. For any y, we can perform the Taylor expansion:

$$f(y) \leq f(x) + \nabla f(x)^T(y-x) + \tfrac{1}{2}(y-x)^T\nabla^2 f(x)(y-x)$$

$$\leq f(x) + \nabla f(x)^T(y-x) + \tfrac{1}{2}(y-x)^T(LI)(y-x)$$

$$(\because LI \succeq \nabla^2 f(x) \Rightarrow (y-x)^T(LI - \nabla^2 f(x))(y-x) \geq 0)$$

$$= f(x) + \nabla f(x)^T(y-x) + \tfrac{L}{2}\|y-x\|_2^2.$$

Substitute $y = x^+ = x - t\nabla f(x)$:

$$\Rightarrow f(x^+) \leq f(x) + \nabla f(x)^T(x - t\nabla f(x) - x) + \tfrac{L}{2}\|x - t\nabla f(x) - x\|_2^2$$

$$= f(x) - t\|\nabla f(x)\|_2^2 + \frac{Lt^2}{t}\|\nabla f(x)\|_2^2$$

$$= f(x) - (1 - \tfrac{Lt}{2})\|\nabla f(x)\|_2^2$$

$$\leq f(x) - \tfrac{t}{2}\|\nabla f(x)\|_2^2. \quad \text{— ①}$$

This tells us each update decreases the function value by $\geq \frac{1}{2}t\|\nabla f(x)\|_2^2$.

Then, since f is convex, ie

$$f(y) \geq f(x) + \nabla f(x)^T(y-x)$$

$$y = x^* \quad \Rightarrow f(x^*) \geq f(x) + \nabla f(x)^T(x^* - x)$$

$$\Rightarrow f(x) \leq f(x^*) + \nabla f(x)^T(x - x^*)$$

Substitute this into ①:

$$\Rightarrow f(x^+) \leq f(x) - \tfrac{t}{2}\|\nabla f(x)\|_2^2$$

$$\leq f(x^*) + \nabla f(x)^T(x - x^*) - \tfrac{t}{2}\|\nabla f(x)\|_2^2$$

$$\Rightarrow f(x^+) - f(x^*) \leq \frac{1}{2t}\left[2t\,\nabla f(x)^T(x - x^*) - t^2\|\nabla f(x)\|_2^2\right]$$

$$= \frac{1}{2t}\left[2t\,\nabla f(x)^T(x - x^*) - t^2\|\nabla f(x)\|_2^2 - \|x - x^*\|_2^2 + \|x - x^*\|_2^2\right]$$

$$= \frac{1}{2t}\left[\|x - x^*\|_2^2 - \|x - t\nabla f(x) - x^*\|_2^2\right]$$

$$= \frac{1}{2t}\left[\|x - x^*\|_2^2 - \|x^+ - x\|_2^2\right].$$

If we set $x^+ = x^{(i)}$, $x = x^{(i-1)}$, then we get

$$f(x^{(i)}) - f(x^{(i-1)}) \leq \frac{1}{2t}\left[\|x^{(i-1)} - x^*\|_2^2 - \|x^{(i)} - x^*\|_2^2\right].$$

If we sum over iterations,

$$\sum_{i=1}^{k}(f(x^{(i)}) - f(x^*)) \leq \sum_{i=1}^{k}\frac{1}{2t}\left[\|x^{(i-1)} - x^*\| - \|x^{(i)} - x^*\|_2^2\right]$$

$$= \frac{1}{2t}\left[\|x^{(0)} - x^*\|_2^2 - \|x^{(k)} - x^*\|_2^2\right]$$

$$\leq \frac{1}{2t}\|x^{(0)} - x^*\|_2^2,$$

which implies

$$\frac{1}{k}\sum_{i=1}^{k} f(x^{(i)}) \leq f(x^*) + \frac{\|x^{(0)} - x^*\|_2^2}{2tk}.$$

Then, since $f(x^{(i)})$ is decreasing, it follows that

$$f(x^{(k)}) \leq \frac{1}{k}\sum_{i=1}^{k} f(x^{(i)}).$$

Therefore

$$f(x^{(k)}) \leq f(x^*) + \frac{\|x^{(0)} - x^*\|_2^2}{2tk}$$

M-STRONG CONVEXITY

We say f is "m-strong convex" for some $m \in \mathbb{R}$ if $f(x) - m\|x\|_2^2$ is convex.

CONVERGENCE ANALYSIS FOR STRONG CONVEXITY

Let f be m-strongly convex & L-smooth for $L, m \in \mathbb{R}$.

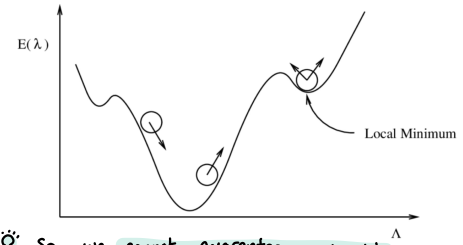
Then gradient descent with fixed step site $t \le \frac{2}{m+L}$ satisfies

$$\boxed{f(x^{(k)}) - f^* \le \gamma^k \frac{L}{2}\|x^{(0)} - x^*\|_2^2, \quad 0 < \gamma < 1}$$

In particular, the convergence rate is $O(\gamma^k)$, which is exponentially fast.

GRADIENT DESCENT FOR NON-CONVEX CASE

For non-convex functions, there may exist local minimums that are not global minimums.



So, we cannot guarantee optimality, and so we will focus on $\|\nabla f(x)\|_2 \le \varepsilon$.

CONVERGENCE ANALYSIS FOR NON-CONVEX CASE

Let f be differentiable & L-lipschitz continuous.

Then gradient descent with fixed step Size $t \le \frac{1}{L}$ satisfies

$$\boxed{\min_{i=0,\dots,k} \|\nabla f(x^{(i)})\|_2 \le \sqrt{\frac{2(f(x^{(0)}) - f^*)}{t(k+1)}}}$$

In other words, the convergence rate is $O(\frac{1}{\sqrt{k}})$, which is optimal for deterministic algorithms.

STOCHASTIC GRADIENT DESCENT

For decomposable optimization, gradient descent involves

$$\boxed{w^+ = w - t \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w)}$$

where n is large, & t is fixed.

Idea: In SGD, our step becomes

$$\boxed{w^+ = w - t\nabla f_I(x), \ I \text{ is a random index, } t = \frac{1}{k}}$$

The convergence rate is $O(\frac{1}{\sqrt{k}})$.

Since randomness leads to a large variance of the estimation of the gradient, SGD requires more iterations, although each iteration requires less computations.

Chapter 8: Multilayer Percepton

MOTIVATION

💡$_1$ We showed no linear classifier can separate the XOR dataset.

💡$_2$ Fixes:

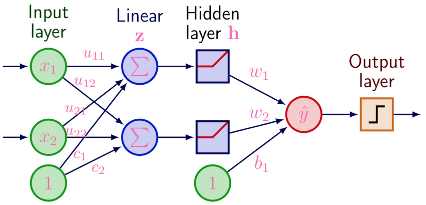
① Use a quadratic classifier;

② Fix the classifier but use a richer input representation.

MULTI-LAYER PERCEPTRON / MLP

💡 Idea: Use a neural network & learn the feature map simultaneously with the linear classifier.

2-LAYER NN



💡 Steps:

① 1st linear transformation: $z = Ux + c$, $U \in \mathbb{R}^{2\times 2}$, $c \in \mathbb{R}^2$

↳ ie $z_1 = u_{11}x_1 + u_{12}x_2 + c_1$

$z_2 = u_{21}x_1 + u_{22}x_2 + c_2$

② Then, we do an element-wise nonlinear activation: $h = \sigma(z)$.

↳ it is important σ is non-linear.

③ 2nd linear transformation: $\hat{y} = \langle h, w \rangle + b$

④ Output layer: $\text{sign}(\hat{y})$ or $\text{sigmoid}(\hat{y})$

EXAMPLE: XOR DATASET

Let $U = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $c = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$

Then let $\sigma(t) = t^+ = \begin{pmatrix} \max(t_1, 0) \\ \max(t_2, 0) \end{pmatrix}$ (RELU)

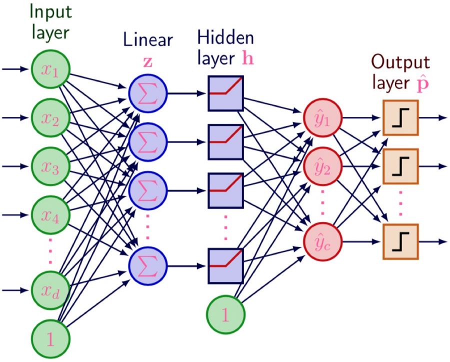
Let $w = \begin{pmatrix} 2 \\ -4 \end{pmatrix}$, $b = -1$.

Then see that

$$x_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},\ y = - \Rightarrow z_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$
$$= \begin{pmatrix} 0 \\ -1 \end{pmatrix}.$$
$$\Rightarrow h_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$
$$\Rightarrow \hat{y} = \langle h, w \rangle - 1$$
$$= -1. \quad (\therefore \text{sign}(\hat{y}) = \text{sign}(y))$$

We can do similar calculations for x_2, x_3, x_4.

MULTI-CLASS CLASSIFICATION



💡 Idea:

$z = Ux + c$
$h = \sigma(z)$ } learning feature h

$\hat{y} = Wh + b$
$\hat{p} = \text{softmax}(\hat{y})$ } learning linear classifier by logistic regression

ACTIVATION FUNCTIONS

💡 Choices for activation function:

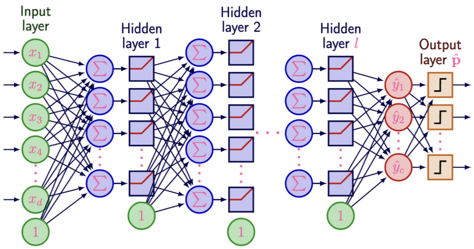
① $\mathrm{sgm}(t) = \frac{1}{1+\exp(-t)}$

② $\tanh(t) = 1 - 2\,\mathrm{sgm}(t)$

③ $\mathrm{relu}(t) = t^{+}$

④ $\mathrm{elu}(t) = (t)^{+} + (t)^{-}(\exp(t)-1)$

MULTI-LAYER NN



1. We need a loss ℓ to measure difference between our prediction $\hat{p}$ & truth y.

2. We also need a training set $D = \{(x_i, y_i)\}$ to train the weights w.

SGD FOR MLP

1. To train w, we can use gradient descent:

$$w \leftarrow w - \eta \cdot \frac{1}{n}\sum_{i=1}^{n} \nabla[\ell\circ f](x_i, y_i; w),$$
$$[\ell\circ f](x_i, y_i; w) = \ell[f(x_i; w), y_i]$$

2. We can also just use a random minibatch $B \subseteq \{1, \dots, n\}$:

$$w \leftarrow w - \eta \cdot \frac{1}{|B|}\sum_{i\in B} \nabla[\ell\circ f](x_i, y_i; w)$$

↳ tradeoff between variance & computation.

3. We can also use a decaying learning rate:

eg $\eta_t = \begin{cases} \eta_0, & t \le t_0 \\ \eta_0/10, & t_0 < t \le t_1 \\ \eta_0/100, & t_1 < t \end{cases}$

COMPUTING THE GRADIENT OF A 2-LAYER NN

1. Model:

$$\boxed{\begin{aligned} x &= \text{input} \\ z &= Wx + b_1 \\ h &= \mathrm{relu}(z) \\ \theta &= Uh + b_2 \\ J &= \tfrac{1}{2}\|\theta - y\|_2^2 \end{aligned}}$$

2. We want to learn the parameters W, b_1, U & b_2.

3. The gradient of the network is defined by

$$\boxed{\frac{\partial J}{\partial W},\ \frac{\partial J}{\partial b_1},\ \frac{\partial J}{\partial U},\ \frac{\partial J}{\partial b_2}}$$

4. Next, since $\mathrm{relu}(x) = \max(x, 0)$, it follows that

$$\boxed{\mathrm{relu}'(x) = \begin{cases} 1, & x > 0 \\ 0, & \text{otherwise} \end{cases}}$$

5. We will show that

$$\boxed{\begin{aligned} \frac{\partial J}{\partial U} &= (\theta - y)h^T \\ \frac{\partial J}{\partial b_2} &= \theta - y \\ \frac{\partial J}{\partial W} &= (U^T(\theta - y) \odot \mathrm{relu}'(z))\,x^T \\ \frac{\partial J}{\partial b_1} &= U^T(\theta - y) \odot \mathrm{relu}'(z) \end{aligned}}$$

where $A \odot B = (A)_{ij}(B)_{ij}$ is the "element-wise" product / "Hadamard product" of the matrices A & B.

Proof. We use the chain rule repetitively.

Note $\frac{\partial J}{\partial \theta} = \theta - y$.

Thus

$$\frac{\partial J}{\partial U} = \frac{\partial J}{\partial \theta} \cdot \frac{\partial \theta}{\partial U} = (\theta - y)h^T$$

Then

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial \theta} \cdot \frac{\partial \theta}{\partial b_2} = (\theta - y)\cdot 1 = \theta - y.$$

Next

$$\frac{\partial J}{\partial h} = \frac{\partial J}{\partial \theta} \cdot \frac{\partial \theta}{\partial h} = U^T(\theta - y).$$

Thus

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial h} \cdot \frac{\partial h}{\partial z} = U^T(\theta - y) \odot \mathrm{relu}'(z)$$

and so

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial z} \cdot \frac{\partial z}{\partial W} = (U^T(\theta - y) \odot \mathrm{relu}'(z))\,x^T$$

lastly,

$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial z} \cdot \frac{\partial z}{\partial b_1} = U^T(\theta - y) \odot \mathrm{relu}'(z) \cdot 1$$
$$= U^T(\theta - y) \odot \mathrm{relu}'(z)$$

and we're done!

UNIVERSAL APPROXIMATION THEOREM

- For any continuous function $f: \mathbb{R}^d \to \mathbb{R}^c$ and any $\varepsilon > 0$, there exists a $k \in \mathbb{N}$, $W \in \mathbb{R}^{k \times d}$, $b \in \mathbb{R}^k$ & $U \in \mathbb{R}^{c \times k}$ such that

$$\boxed{\sup_x \|f(x) - g(x)\|_2 < \varepsilon,}$$

where $g(x) = U(\sigma(Wx+b))$ & σ is the (element-wise) RELU operation.

ie $\|f(x) - g(x)\|_2 < \varepsilon \ \forall x$, s.t. $g(x)$ is at least "ε-close" to $f(x)$.

- This implies that as long as a 2-layer MLP is "wide enough" (ie a large k), it can approximate any continuous function arbitrarily closely.

WHY DEEP LEARNING?

- There exist functions such that a 2-layer MLP needs to be exponentially wide to approximate the function, whereas a 3-layer MLP only needs to be polynomially wide.
- In particular, deep NNs are more parameter efficient.

DROPOUT

- <u>Idea:</u> For each training minibatch, keep each hidden unit with probability q.
- Essentially, there is a different & random network for each training minibatch.
- In particular, hidden units are less likely to collude to overfit training data.
- For testing, we use the full network.

BATCH NORMALIZATION

- <u>Idea:</u> Normalize the input over the minibatch dimensions.

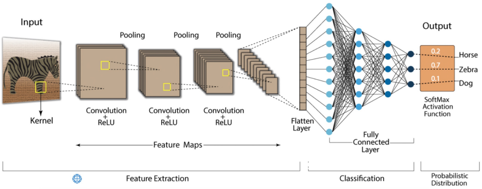
Chapter 9: Convolutional Neural Networks

MOTIVATION

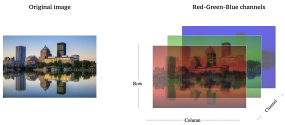
1. In MLPs, it is easy to overfit training data.

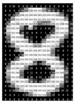
2. Idea: To mitigate this, we can use weight sharing & use a sparse matrix.

CONVOLUTIONAL NEURAL NETWORK / CNN



THE FORM OF IMAGE DATA



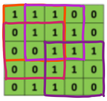


- we can represent an greyscale image as a matrix of values ranging from 0-255
- for RGB images, we can represent them as a tensor (3D matrix) with 3 channels, each corresponding to R, G & B values.

CONVOLUTION [ONE-CHANNEL INPUT]

Idea: Each entry in the output matrix is the inner product of the corresponding "subgrid" in the input matrix and the convolutional filter.

eg



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input matrix

1	0	1
0	1	0
1	0	1

Convolutional 3x3 filter

→

output

4	3	4
2	4	3
2	3	4

- recall: $\langle A, B \rangle = \sum_{i,j} A_{ij} B_{ij} \in \mathbb{R}$
- this is like taking the inner product of the sliding "window" of the input matrix & the filter/kernel successively.

WHY CONVOLUTION?

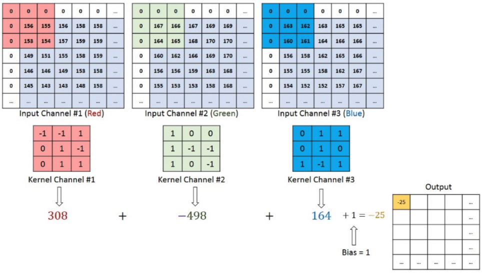
Note traditional image processing algorithms use convolution.

CONVOLUTION [MULTI-CHANNEL INPUT]

Here, we have k input channel matrices corresponding to k kernel channel matrices.

Idea: For each entry of the output matrix, we take the "sliding window inner product" for each kernel channel - input channel pair, and then sum the products together.

eg



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

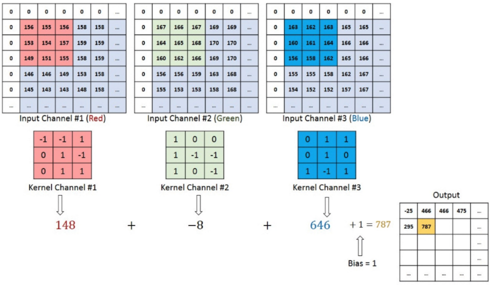
Kernel Channel #3

308 + −498 + 164 + 1 = −25

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

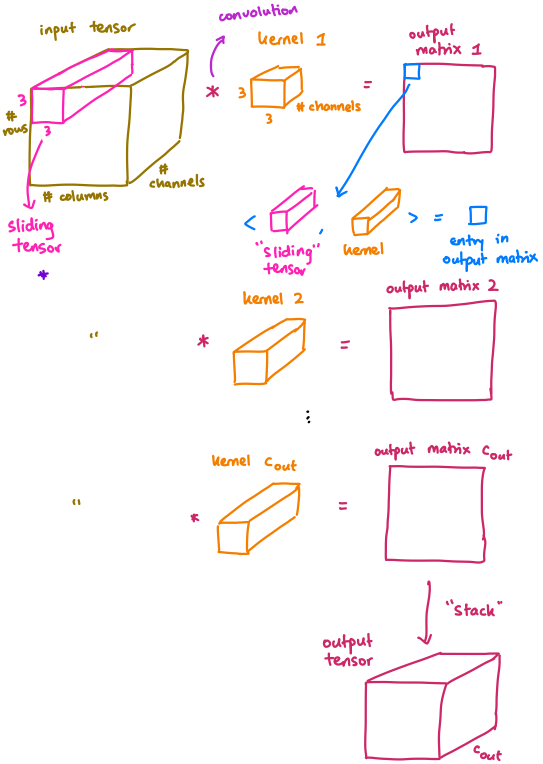
148 + −8 + 646 + 1 = 787

Bias = 1

Output

-25	466	466	475	...
295	787			...
				...
				...
...	...	...	...	...

Another explanation:



- C_{out} = # of output channels

- we can view convolution as successive "sliding inner products" on the input tensor & the C_{out} kernel tensors.

CONTROLLING THE CONVOLUTION

Hyperparameters:

① Filter/kernel size:

- eg 3x3, 5x5
- by default, # of channels on each filter is the same as input

② Number of kernels;

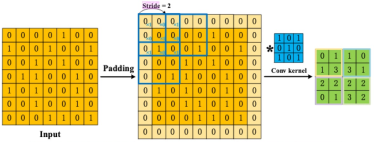
③ "Stride" — how many pixels to move the filter each time; &

- larger stride ⇒ neighboring outputs less similar

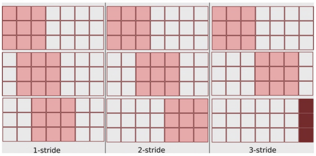
④ "Padding" — add zeroes around input boundary.

- keeps boundary information lossless

PADDING & STRIDE



$$\begin{pmatrix}0&0&0&0&1&0&0\\1&0&0&1&0&0&1\\0&0&1&0&1&0&0\\0&1&1&0&0&1&0\\1&0&1&0&0&1&0\\0&0&1&0&0&1&0\\0&0&0&1&1&0&1\end{pmatrix} \xrightarrow{\text{Padding}} \begin{pmatrix}0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0\\0&1&0&0&1&0&0&1&0\\0&0&0&1&0&1&0&0&0\\0&0&1&1&0&0&1&0&0\\0&1&0&1&0&0&1&0&0\\0&0&0&1&1&0&1&0&0\\0&0&0&0&1&1&0&1&0\\0&0&0&0&0&0&0&0&0\end{pmatrix} * \begin{pmatrix}1&0&1\\0&1&0\\1&0&1\end{pmatrix} \xrightarrow{\text{Conv kernel}} \begin{pmatrix}0&1&1&0\\1&3&3&1\\2&2&2&2\\0&1&3&2\end{pmatrix}$$



SIZE CALCULATION

Sizes:

① Input: $m \times n \times c$

② Filter: $a \times b \times c$

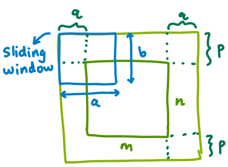
③ Stride: $s \times t$

④ Padding: $p \times q$

We pad p pixels on the top/bottom & q pixels on the left/right.

We move s pixels horizontally & t pixels vertically.

input tensor (front slice)



We can show that

$$\text{output size} = \left\lfloor \frac{m+2p-a}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2q-b}{t} + 1 \right\rfloor$$

WEIGHT SHARING: CNN=MLP

Let our kernel be $W = \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix} \in \mathbb{R}^{2\times2}$ &

our input matrix be $X = \begin{pmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{pmatrix} \in \mathbb{R}^{3\times3}$.

We can define

$$\text{Vector}(X) = (x_{00}, x_{01}, x_{02}, x_{10}, \dots, x_{22})^T \in \mathbb{R}^9.$$

Then note

$$W * X = \begin{pmatrix} w_{00}x_{00} + w_{01}x_{01} + w_{10}x_{10} + w_{11}x_{11} & w_{00}x_{01} + w_{01}x_{02} + w_{10}x_{11} + w_{11}x_{12} \\ w_{00}x_{10} + w_{01}x_{11} + w_{10}x_{20} + w_{11}x_{21} & w_{00}x_{11} + w_{01}x_{12} + w_{10}x_{21} + w_{11}x_{22} \end{pmatrix}.$$
$$:= \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix}.$$

Hence

$$\text{Vector}(W * X) = (c_{00}, c_{01}, c_{10}, c_{11})^T \in \mathbb{R}^4.$$

Next, if we define the "circulant matrix" as

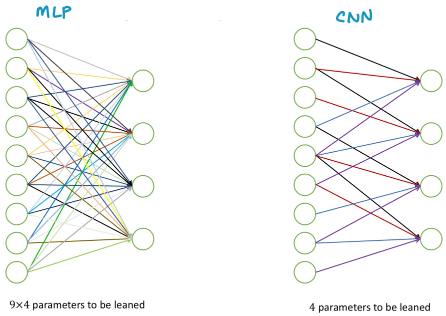
$$W_{circ} = \begin{pmatrix} w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 & 0 \\ 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 \\ 0 & 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} \end{pmatrix} \in \mathbb{R}^{4\times9}$$

See that

$$W_{circ} \text{Vector}(X) = \text{Vector}(W * x).$$

Thus, we can view convolution as multiplying a weight matrix with the input.

Hence, we can view CNN as a MLP, but with weight sharing.



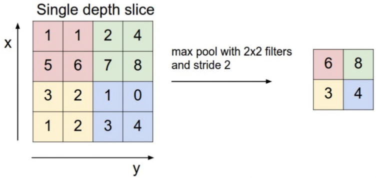
9×4 parameters to be leaned

4 parameters to be leaned

Hence, we can train a CNN faster than a MLP, since there are less parameters to be learnt.

POOLING

1. Idea: "Pooling" down-samples the input size to reduce memory & computation.
2. To do this, we use the same "sliding window" trick as in convolution, and then take the max or average of each window to get the output.
3. We also have a notion of size/stride.



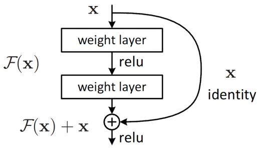
4. Note that pooling by default is performed on each slice separately, so the number of channels is the same between the input & output.
5. If we set the kernel size = input size, this is known as "global pooling".

DEEPER MODELS

- Note deeper models (ie more layers) are better but are more difficult to train.

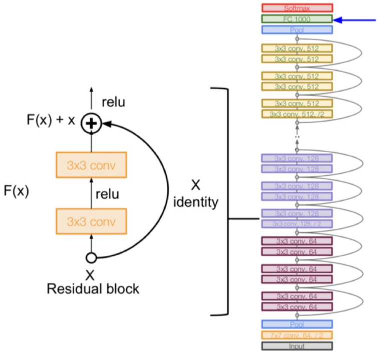
RESIDUAL BLOCK

1. Idea: Add a shortcut connection that allows "skipping" one or more layers.



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathbf{x}$
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

2. This allows more direct backpropagation of the gradient via the shortcut.
3. By "stacking" residual blocks, we can get a "residual network" (or ResNet).



No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

Chapter 10: Transformers

"Transformers" were designed for machine translation tasks; ie given a sentence X with words/tokens $x_1, \dots, x_n$, produce a translation Y with tokens $y_1, \dots, y_m$.

INPUT & OUTPUT

1. Our input is $X = (x_1, \dots, x_n)$ (ie the "prompt"), and our output is $Y = (y_1, \dots, y_m)$.

2. We want to find

$$\boxed{\underset{Y}{\operatorname{argmax}} \; P(y_1, \dots, y_m \mid x_1, \dots, x_n)}$$

AUTO-REGRESSIVE / GREEDY METHOD

Idea: we repeatedly compute

$$\boxed{\underset{y_k}{\operatorname{argmax}} \; P(y_k \mid x_1, \dots, x_n, y_1, \dots, y_{k-1}).}$$

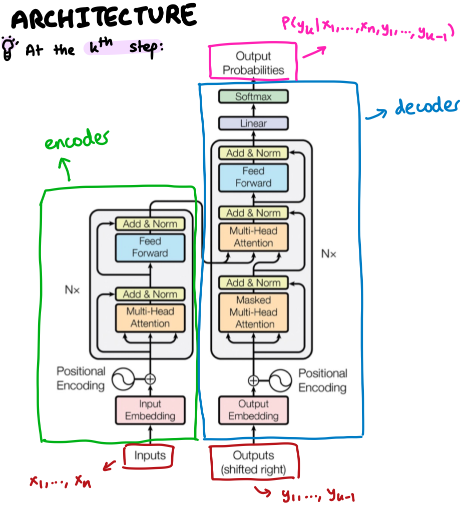
eg

Step 0 X: Where is University of Waterloo?
Step 1 Y: [START]; $\Pr$(It | X [START]) highest
Step 2 Y: [START] It; $\Pr$(is | X [START] It) highest
Step 3 Y: [START] It is; $\Pr$(at | X [START] It is) highest
Step 4 Y: [START] It is at; $\Pr$(Waterloo | X [START] It is at) highest
Step 5 Y: [START] It is at Waterloo; $\Pr$([END] | X [START] It is at Waterloo) highest
Step 6 Y: [START] It is at Waterloo [END]

↳ [START] is a special start token we use at initialization.

ARCHITECTURE

At the k^{th} step:



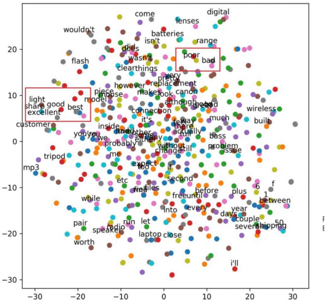
TOKENIZER

The "tokenizer" divides the input sentence into the individual tokens/words.

TOKEN EMBEDDING

1. A "token embedding" is a bijection from tokens to vectors:
   ① we convert the input tokens to vectors of dimension d; and
   ② convert the decoder outputted vectors to output tokens.

2. We want words of similar meaning to be close in the embedding space.



POSITIONAL ENCODING

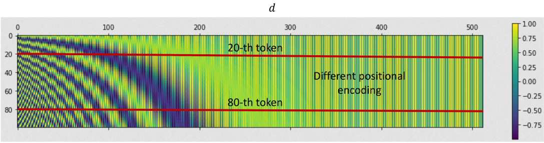
1. **Idea:** the order of tokens in the sentence changes its meaning.

2. We use a positional encoding matrix $W^P \in \mathbb{R}^{n \times d}$:

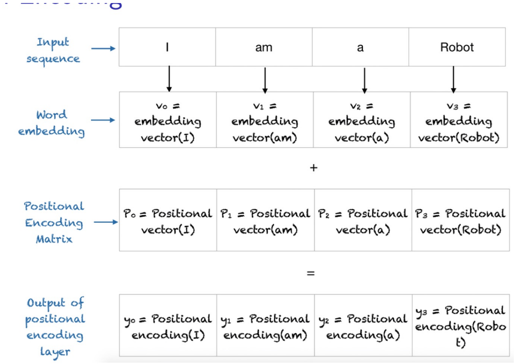
$$W^P_{t,2i} = \sin\left(\frac{t}{10000^{2i/d}}\right), \quad W^P_{t,2i+1} = \cos\left(\frac{t}{10000^{2i/d}}\right),$$
$$i = 0, \dots, \frac{d}{2} - 1$$

↳ no parameter to be learnt!

3. We then just add W^P to the $n \times d$ token embedding.

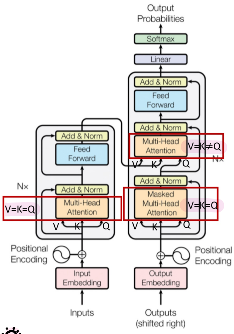


4. Putting it together:



Input sequence	I	am	a	Robot
Word embedding	v_0 = embedding vector(I)	v_1 = embedding vector(am)	v_2 = embedding vector(a)	v_3 = embedding vector(Robot)
	+			
Positional Encoding Matrix	P_0 = Positional vector(I)	P_1 = Positional vector(am)	P_2 = Positional vector(a)	P_3 = Positional vector(Robot)
	=			
Output of positional encoding layer	y_0 = Positional encoding(I)	y_1 = Positional encoding(am)	y_2 = Positional encoding(a)	y_3 = Positional encoding(Robot)

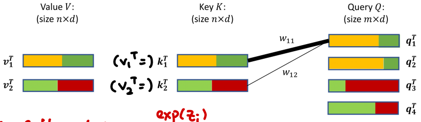
ATTENTION LAYER



1. **Inputs:**
   1. value $V \in \mathbb{R}^{n \times d}$;
   2. key $K \in \mathbb{R}^{n \times d}$; &
   3. query $Q \in \mathbb{R}^{m \times d}$.

2. **Output:** $\mathbb{R}^{m \times d}$ (m row vectors of dimension d).

3. **Idea:**



Let $\text{softmax}(z_i) = \dfrac{\exp(z_i)}{\sum_j \exp(z_j)}$, for $z = (z_1, \dots, z_n)$.

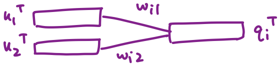
Then $w_{11} = \langle q_1, k_1 \rangle$, $w_{12} = \langle q_1, k_2 \rangle$.

$\Rightarrow$ 1st output row = $\text{softmax}\left(\frac{w_{11}}{\sqrt{d}}\right) v_1^T + \text{softmax}\left(\frac{w_{12}}{\sqrt{d}}\right) v_2^T$.

- note v_1^T contributes more to the output row.
- this is just a weighted average.

Similarly, the

ith output row = $\text{softmax}\left(\frac{w_{i1}}{\sqrt{d}}\right) v_1^T + \text{softmax}\left(\frac{w_{i2}}{\sqrt{d}}\right) v_2^T$.



MATRIX FORM OF ATTENTION

1 Matrix form: let v_i^T, k_i^T & q_i^T be the row vectors of the value, key & query. Let

$$V = \begin{pmatrix} v_1^T \\ \vdots \\ v_n^T \end{pmatrix} \in \mathbb{R}^{n \times d}, \quad K = \begin{pmatrix} k_1^T \\ \vdots \\ k_n^T \end{pmatrix} \in \mathbb{R}^{n \times d},$$
$$Q = \begin{pmatrix} q_1^T \\ \vdots \\ q_m^T \end{pmatrix} \in \mathbb{R}^{m \times d}.$$

Then

$$\begin{aligned} &\text{Attention}(V, K, Q) \\ &= \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V \\ &= \begin{pmatrix} \text{softmax}\left(\frac{\langle q_1, k_1\rangle}{\sqrt{d}}\right) v_1^T + \dots + \text{softmax}\left(\frac{\langle q_1, k_n\rangle}{\sqrt{d}}\right) v_n^T \\ \vdots \\ \text{softmax}\left(\frac{\langle q_m, k_1\rangle}{\sqrt{d}}\right) v_1^T + \dots + \text{softmax}\left(\frac{\langle q_m, k_n\rangle}{\sqrt{d}}\right) v_n^T \end{pmatrix} \\ &\in \mathbb{R}^{m \times d}. \end{aligned}$$

- softmax is a "row-wise" operation

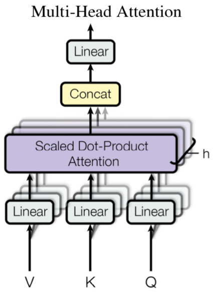
2 There is no learnable parameters so for!

LEARNABLE ATTENTION LAYER & MULTI-HEAD ATTENTION

1 Idea: Replace $Q \to QW^q$, $K \to KW^k$, $V \to VW^v$, where $\{W^q, W^k, W^v\} \in \mathbb{R}^{d \times 64}$ are learnable linear layers.

2 Then our attention layer becomes

$$\begin{aligned} &\text{Attention}(VW^v, KW^k, QW^q) \\ &= \text{softmax}\left(\frac{QW^q (KW^k)^T}{\sqrt{d}}\right) VW^v \end{aligned}$$



3 We can add h=8 linear layers in parallel & concatenate their output later.

- output dimension = 64 x 8 = 512

MASKED MULTI-HEAD ATTENTION

Idea: We mask future words, and input the masked sequence into the attention layer.

FEED-FORWARD LAYER

1 This is just a 2-layer MLP with ReLU activation:

$$\text{MLP}(x) = \max(0, x^T W_1 + b_1^T) \cdot W_2 + b_2^T$$

2 We use layer normalization instead of batch normalization.

- Since batch size is often small

OVERVIEW

A transformer has the following tunable hyperparameters:

① # of layers, $N = 6$;

② output dimension of all modules, $d = 512$;

③ # of heads, $h = 8$.

TRANSFORMER LOSS

We train the transformer by finding

$$\min_{w} \hat{E}[-\langle Y, \log \hat{Y} \rangle]$$

where

① $Y = (y_1, \dots, y_\ell)$ is our output sequence; &

- this is one-hot (ie 0 or 1)

② $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_\ell)$ is the predicted probabilities.

Chapter 11: Large Language Models

COMPUTATIONAL COMPLEXITY

💡$_1$ Self-attention: $O(n^2d + nd^2)$ per layer

$Q \in \mathbb{R}^{n \times d}$, $K^T \in \mathbb{R}^{d \times n}$.

$\Rightarrow$ computing QK^T takes $O(n^2d)$ time.

$QK^T \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{n \times d}$

$\Rightarrow$ computing $\text{softmax}(\frac{QK^T}{\sqrt{d}}) \cdot V$ takes $O(nd^2)$ time.

💡$_2$ Feed-forward: $O(d^3)$ per layer

LABEL SMOOTHING

💡$_1$ Idea: Replace the label Y distribution $p(k|x) = \delta_{k,y}$ with

$$\boxed{p'(k|x) = (1-\varepsilon_{ls})\,\delta_{k,y} + \varepsilon_{ls}\frac{1}{C},}$$

where C is the # of classes.

y	0	1	… … …	0

Label Smoothing

y'	$\frac{\epsilon_{ls}}{C}$	$1 - \frac{C-1}{C}\epsilon_{ls}$	… … …	$\frac{\epsilon_{ls}}{C}$

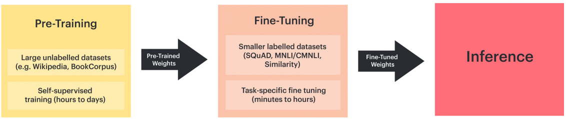
- ε_{ls} is a hyperparameter.

BERT VS GPT

💡$_1$ BERT is solely an encoder, whereas GPT is solely a decoder.

- BERT predicts randomly-sampled middle word
- GPT predicts the next word

PRETRAINING, FINETUNING, INFERENCE



Pre-Training
Large unlabelled datasets (e.g. Wikipedia, BookCorpus)
Self-supervised training (hours to days)
Pre-Trained Weights
Fine-Tuning
Smaller labelled datasets (SQuAD, MNLI/CMNLI, Similarity)
Task-specific fine tuning (minutes to hours)
Fine-Tuned Weights
Inference

- pre-training takes weeks / months
- fine-tuning takes days to weeks/months

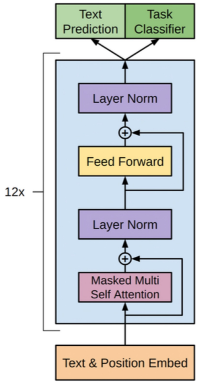
PRE-TRAINING TASKS

💡$_1$ GPT: predict masked words

💡$_2$ BERT: predict middle words given context.

- it is harder to predict the future than the past.

GPT STRUCTURE



PRETRAINING

💡 Goal: we want to find

$$\boxed{\min_{\theta} \hat{E}\left[-\log \prod_{j=1}^{m} p(x_j | x_1, \dots, x_{j-1}; \theta)\right]}$$

log likelihood in predicting next word x_j given previous tokens $x_1, \dots, x_{j-1}$

FINE-TUNING

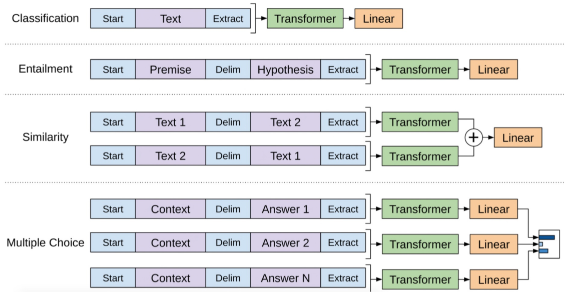
💡$_1$ Goal: We want to find

$$\boxed{\min_{\theta} \underbrace{- \hat{E}\left[\log \prod_{j=1}^{m} p(y | X_{1:m}; \theta)\right]}_{\text{task-aware supervised loss}} - \lambda \underbrace{\hat{E}\left[\log \prod_{j=1}^{m} p(x_j | X_{1;j-1}; \theta\right]}_{\text{pretraining loss}}}$$

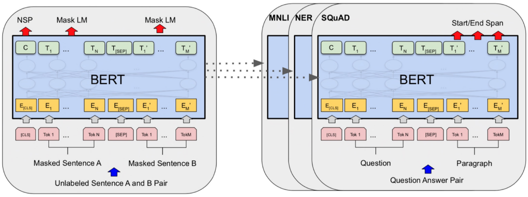
💡$_2$ Tasks:

1. "Classification" - classify text into a class
2. "Entailment" - detemine if a hypothesis contradicts or follows from a premise
3. "Similarity" - predict if two sentences are semantically equivalent
4. "Multiple Choice" - given a context & N possible answers, choose the correct answer

TASK-DEPENDENT ARCHITECTURE



BERT STRUCTURE



Pre-training

Fine-Tuning

PRETRAINING

💡1 Task A: using a masked language model;

① randomly select 15% input tokens, change to [Mask]; and

② add softmax to predict the [Mask] tokens.

💡2 Task B: next sentence prediction (NSP); given 2 sentences A & B, 50% of the time B is the actual next sentence that follows A ("IsNext"), and 50% of the time it is just random ("NotNext").

💡3 The losses for Masked LM & NSP tasks are weighted, summed & minimized.

RoBERTa

💡 Idea: Improve BERT by

① training the model longer;

② use bigger batches;

③ use more data;

④ remove NSP; &

⑤ train on longer sentences.

SENTENCE-BERT

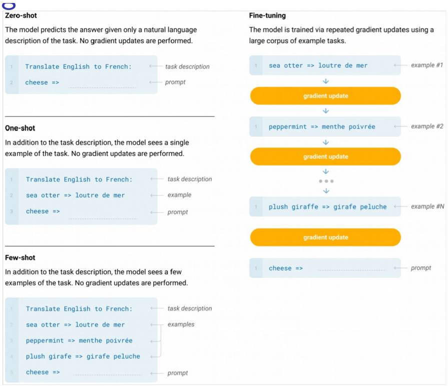
💡1 Idea: Use a twin network to save the representations for future use.

💡2 This drastically reduces the # of times we do inference & the computation time.

GPT-2

💡1 "GPT-2" uses the same training method as GPT, but introduces a new larger dataset.

💡2 It is good for "zero-shot learning"



Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
cheese => ........                ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
sea otter => loutre de mer        ← example
cheese => ........                ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:      ← task description
sea otter => loutre de mer        ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese => ........                ← prompt
```

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
sea otter => loutre de mer            ← example #1
gradient update
peppermint => menthe poivrée          ← example #2
gradient update
...
plush giraffe => girafe peluche       ← example #N
gradient update
cheese => ........                    ← prompt
```

GPT-3

💡1 GPT-3 uses the same training method as GPT/GPT-2, but uses a much larger transformer (100x GPT-2)

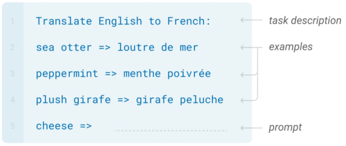
💡2 The larger network introduces

① In-context learning; &

② Chain-of-thought.

IN-CONTEXT LEARNING

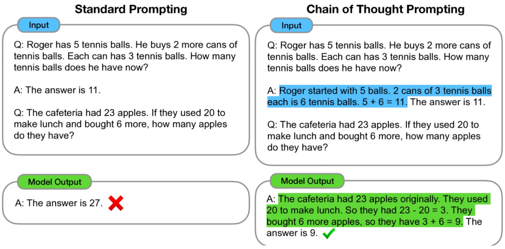
💡 Idea: Giving a few examples in the prompt helps learning.



```
Translate English to French:      ← task description
sea otter => loutre de mer        ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese => ........                ← prompt
```

CHAIN-OF-THOUGHT

💡 Idea: Giving the reasoning process in the prompt helps the learning.



Standard Prompting

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain of Thought Prompting

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

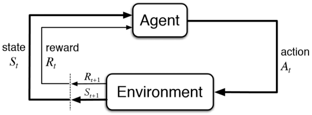
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

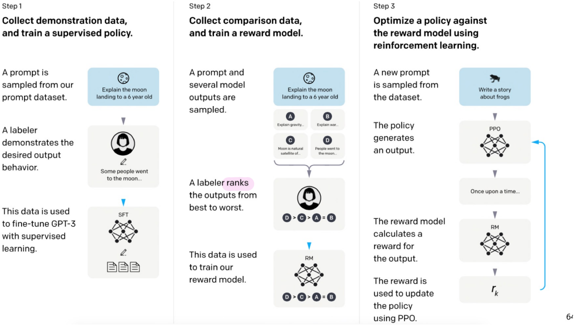
Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

GPT-3.5: REINFORCEMENT LEARNING FROM HUMAN FEEDBACK (RLHF)



see CS486 notes for reinforcement learning details



Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain the moon landing to a 6 year old
Some people went to the moon...
SFT
Explain the moon landing to a 6 year old
A
Explain gravity...
B
Explain war...
C
Moon is natural satellite of...
D
People went to the moon...
D > C > A = B
RM
D > C > A = B
Write a story about frogs
PPO
Once upon a time...
RM
r_k
64

Idea₁: We

① use supervised learning for LLM by BP/SGD;

② freeze the LLM & train the reward model by a loss about ranking; &

③ freeze the reward model, update the LLM using our reward model, & maximize the reward given by the reward model.

Idea₂: We use a ranking model as annotators usually do not give uniformly consistent scores (for the given sentences), but give uniformly consistent rankings.

Chapter 12: Generative Adversarial Networks

MOTIVATION

1. In "generative modelling", we would like to train a network that models a distribution.
2. Idea: We want to design a generative model to generate images.

MODEL

1. Given training data $x_1,\dots,x_n \sim P_{data}(x)$ & the true data density;
2. Parameterize $P_\theta(x)$, the data density estimated by the model.
3. Goal: Estimate θ by minimizing some "distance" between P_{data} (unknown data density) & P_θ;

$$\boxed{\min_\theta \ \text{dist}(P_{data} \,\|\, P_\theta)}$$

4. After training, we can generate new data $x \sim P_\theta(x)$.

PUSH-FORWARD MAPS

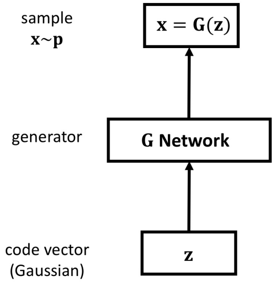
1. Let r be any continuous distribution on $\mathbb{R}^h$. For any distribution p on $\mathbb{R}^d$, there exist "push-forward maps" $G: \mathbb{R}^h \to \mathbb{R}^d$ such that

$$\boxed{z \sim r \ \Rightarrow \ G(z) \sim p.}$$

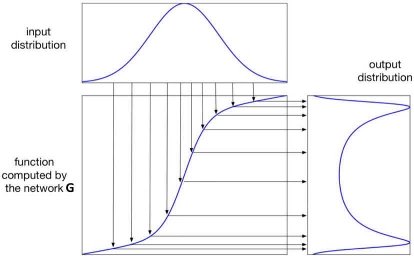
2. WLOG, we can take r to be Gaussian

GENERATING SAMPLES

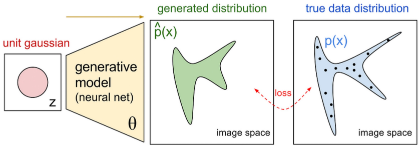
1. Idea: Start by sampling the code vector z from a simple distribution (eg Gaussian).
2. Then, the GAN computes a differentiable function G mapping z to an x in data space.



eg

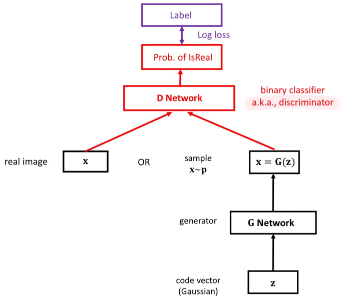


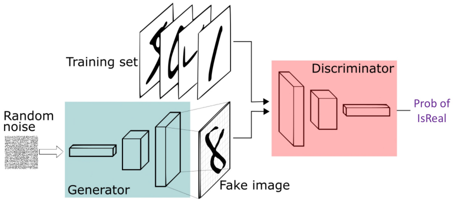
LEARNING THE G NETWORK



Idea: To define the loss to distinguish the 2 distributions, we can use a discriminator.

GENERATIVE ADVERSARIAL NETWORKS





Idea: This is a "zero-sum" game between

① the discriminator — distinguish real images from fake images; &

② the generator — generate images that look like the real one to confuse the discriminator.

DISCRIMINATOR'S GOAL

Idea: For a fixed generator G, minimize a log loss over D (output probability of isReal).

If x is real, minimize $-\log D(x)$;
if x is fake, minimize $-\log(1-D(x))$.

In particular, we want

$$\boxed{\min_{D}\ \underbrace{-\frac{1}{2}\mathop{E}_{x\sim p_{data}}\left[\log D(x)\right]}_{x \text{ is real}}\ \underbrace{-\frac{1}{2}\mathop{E}_{z\sim N(0,I)}\left[\log(1-D(G(z)))\right]}_{x \text{ is fake}}}$$

GENERATOR'S GOAL

Idea: For a fixed discriminator D, maximize a log loss over G (the same loss for the discriminator).

Hence we want to find

$$\boxed{\max_{G}\ \underbrace{-\frac{1}{2}\mathop{E}_{x\sim p_{data}}\left[\log D(x)\right]}_{x \text{ is real}}\ \underbrace{-\frac{1}{2}\mathop{E}_{z\sim N(0,I)}\left[\log(1-D(G(z)))\right]}_{x \text{ is fake}}}$$

PUTTING IT TOGETHER

Hence, we want to find

$$\boxed{\max_{G}\ \min_{D}\ \underbrace{-\frac{1}{2}\mathop{E}_{x\sim p_{data}}\left[\log D(x)\right]}_{x \text{ is real}}\ \underbrace{-\frac{1}{2}\mathop{E}_{z\sim N(0,I)}\left[\log(1-D(G(z)))\right]}_{x \text{ is fake}}}$$

Replacing expectation with the empirical expectation (ie average):

$$\boxed{\min_{G}\ \max_{D}\ \underbrace{\mathop{\hat{E}}_{x\sim p_{data}}\left[\log(D(x))\right] + \mathop{\hat{E}}_{z\sim N(0,I)}\left[\log(1-D(G(z)))\right]}_{V(G,D)}}$$

SOLVER

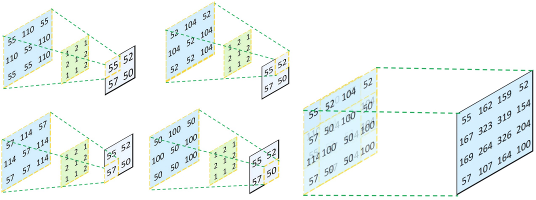
Idea: We can solve this via alternative minimization-maximization:

① **G step:** fix D, update G by one-step gradient descent;

② **D step:** fix G, update D by one-step gradient ascent.

DECONVOLUTION / TRANSPOSED CONVOLUTION

Idea: Use "reverse" convolution to produce a larger matrix from a smaller one.



Multiply Each Element in the Input Layer by Each Value in the Kernel

Combine All Four Resulting Layers Together And Sum the Overlapped Values (Image by Author)

We use a similar "sliding window" trick.

① For each entry in the input, multiply it with the kernel;

② Sum all the results together using "sliding windows".

SOLUTION OF D*

💡 let $p_g(x)$ be the density of x estimated by the generator G.

For a fixed G, the optimal discriminator is

$$\boxed{D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}}$$

Proof. See that

$$V(G,D) = \underset{x\sim p_{data}}{E}[\log D(x)] + \underset{z\sim N(0,I)}{E}[\log(1-D(G(z)))]$$

$$= \int_x p_{data}(x)\log D(x)\,dx + \int_z p_z(z)\log(1-D(G(z)))\,dz$$

↪ let $x = G(z)$

$$= \int_x p_{data}(x)\log D(x)\,dx + \int_x p_g(x)\log(1-D(x))\,dx$$

$$= \int_x \underbrace{p_{data}(x)\log D(x) + p_g(x)\log(1-D(x))}_{f(D(x))}$$

Then the optimal solution is

$$D^*(x) = \underset{D(x)}{\operatorname{argmax}}\, f(D(x)).$$

In particular, we can write $f(D(x))$ as

$f(S) = a\log S + b\log(1-S)$, $S = D(x)$

This is maximized at $S = \frac{a}{a+b}$.

Thus

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

as needed. ▣

SOLUTION OF G*

💡1 $\min_G \max_D V(G,D)$ is achieved iff $p_g = p_{data}$. The optimal objective value is $-\log 4$.

💡2 Thus, the GAN can learn p_{data} exactly if we can solve $\min_G \max_D V(G,D)$ exactly.

Proof. See that

$$V(G,D^*_G) = \underset{x\sim p_{data}}{E}[\log D^*_G(x)] + \underset{z\sim N(0,I)}{E}[\log(1-D^*_G(G(z)))]$$

(let $x = G(z)$)

$$= \underset{x\sim p_{data}}{E}[\log D^*_G(x)] + \underset{x\sim p_g}{E}[\log(1-D^*_G(x))]$$

$$= \underset{x\sim p_{data}}{E}\left[\log \frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right] + \underset{x\sim p_g}{E}\left[\log \frac{p_g(x)}{p_{data}(x)+p_g(x)}\right]$$

For distributions P, Q, we define

$$KL(P\|Q) = \underset{x\sim P}{E}\left[\log \frac{p(x)}{q(x)}\right].$$

Then

$$V(G,D^*_G) = -\log 4 + KL\left(p_{data} \,\middle\|\, \frac{p_{data}+p_g}{2}\right) + KL\left(p_g \,\middle\|\, \frac{p_{data}+p_g}{2}\right)$$

$$= -\log 4 + 2\,JSD(p_{data}\|p_g)$$

$$\geq -\log 4$$

where JSD is the "Jensen-Shannon divergence" (distance between 2 distributions).

Equality holds iff $p_{data} = p_g$, as needed. ▣

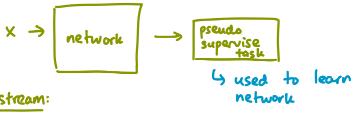
💡3 Thus, GAN works by minimizing the Jensen-Shannon divergence between generated & real data distributions.

Chapter 13: Self-Supervised Learning

1. "Self-supervised learning" is a subclass of unsupervised learning, where we want to learn useful representations through pretraining tasks for downstream tasks.
   - unsupervised: learning with unlabeled data

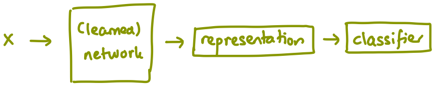
2. Steps:
   ① Pretraining: build a task where the label is pseudo & is constructed from the unlabelled data.
   ② Downstream:
   - Fine-tuning: all trainable parameters
   - Linear evaluation: fix the representation & fine-tuning topping layers

pretraining:


x → network → pseudo supervise task

↳ used to learn network

downstream:


x → (learned) network → representation → classifier

Fine-tuning: update network & classifier

Linear evaluation: fix network, update linear classifier

WHY?

1. Idea: Creating labelled datasets for each task is expensive, but there is a lot of unlabelled data.
2. Self-supervised learning will also not overfit.
3. Challenges:
   ① Select a suitable pretraining task;
   ② No golden rule for comparison for learned feature representations

IMAGE ROTATION

1. Pretraining data: images rotated by multiple of 90° at random
2. Pretraining task: train model to predict rotation degree that was applied

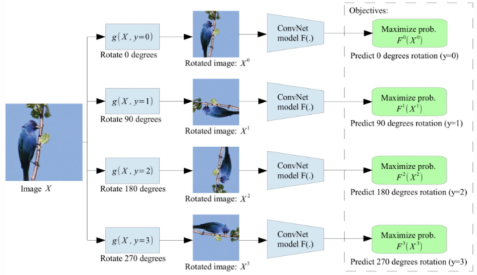
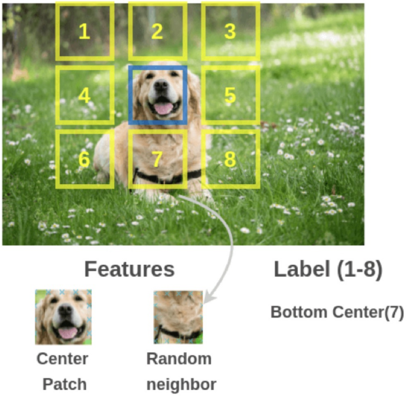


Image X
$g(X, y=0)$
Rotate 0 degrees
Rotated image: X^0
$g(X, y=1)$
Rotate 90 degrees
Rotated image: X^1
$g(X, y=2)$
Rotate 180 degrees
Rotated image: X^2
$g(X, y=3)$
Rotate 270 degrees
Rotated image: X^3
ConvNet model F(.)
Objectives:
Maximize prob. $F^0(X^0)$
Predict 0 degrees rotation (y=0)
Maximize prob. $F^1(X^1)$
Predict 90 degrees rotation (y=1)
Maximize prob. $F^2(X^2)$
Predict 180 degrees rotation (y=2)
Maximize prob. $F^3(X^3)$
Predict 270 degrees rotation (y=3)

RELATIVE PATCH POSITION

Pretraining data: multiple patches extracted from images

Pretraining task: train model to predict relationship between the patches



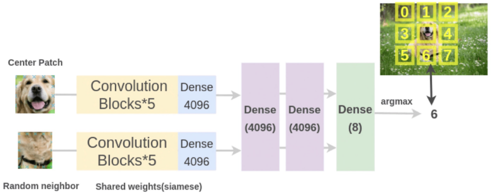
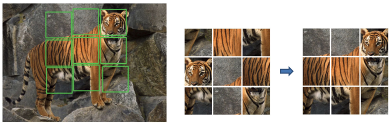
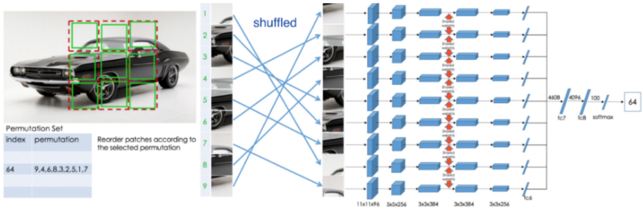


IMAGE JIGSAW PUZZLE

Pretraining data: 9 patches extracted in images

Pretraining task: predict positions of all 9 patches

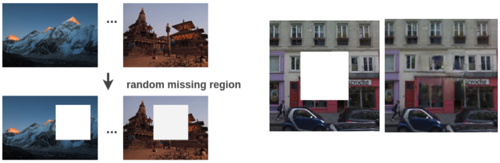


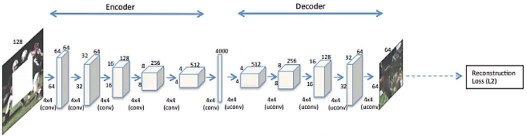


CONTEXT ENCODERS

Pretraining data: remove random region in images

Pretraining task: fill in missing piece in the image



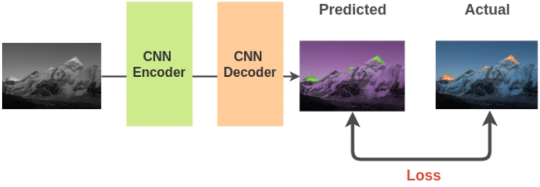


We can improve performance by adding a "GAN" branch.

IMAGE COLORIZATION

Pretraining data: pairs of color & greyscale images

Pretraining task: predict colors of the objects in grayscale images



CROSS-CHANNEL PREDICTION

Pretraining data: remove some of the image color channels

Pretraining task: predict missing channel from the other image channels

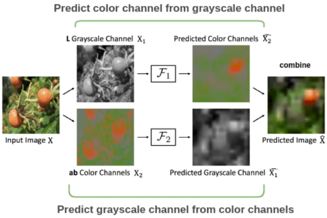
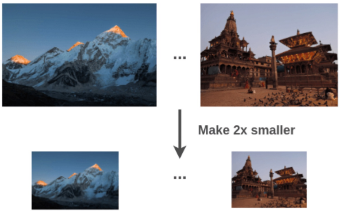
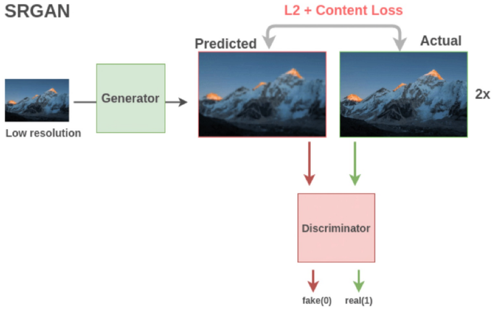

Predict color channel from grayscale channel
L Grayscale Channel X_1
Predicted Color Channels $\widehat{X_2}$
combine
Input Image X
$\mathcal{F}_1$
$\mathcal{F}_2$
Predicted Image $\hat{X}$
ab Color Channels X_2
Predicted Grayscale Channel $\widehat{X_1}$
Predict grayscale channel from color channels

IMAGE SUPER-RESOLUTION

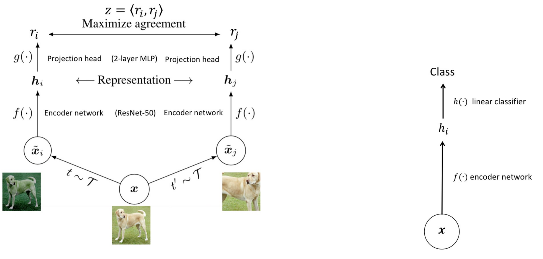
Pretraining data: pairs of regular & downsampled low-resolution images

Pretraining task: predict high resolution image that corresponds to down-sampled low-resolution image





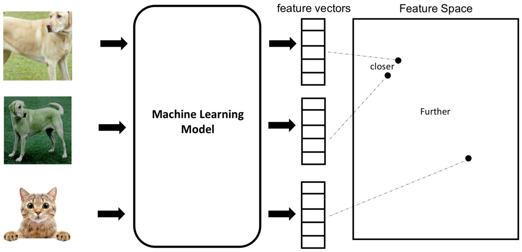
CONTRASTIVE LEARNING: SimCLR



a) Training

b) Testing

Measuring agreement:



Loss function:

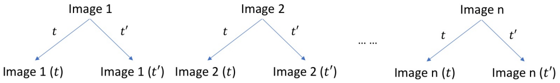


	Image 1 (t')	Image 2 (t)	Image 2 (t')		Image n (t')
Image 1 (t)	z_1	z_2	z_3	… …	z_{2n-1}

We hope z_1 is the largest element

Softmax: $z_i \rightarrow \frac{\exp(z_i)}{\sum_j \exp(z_j)}$

prob_1	prob_2	prob_3	… …	prob_{2n-1}

Loss: $\max_\theta \text{prob}_1 = \frac{\exp(z_1)}{\sum_j \exp(z_j)}$ (a.k.a. InfoNCE loss)

Chapter 14: Evasion Attacks

Idea: we want to modify test images to fool a fixed ML model.

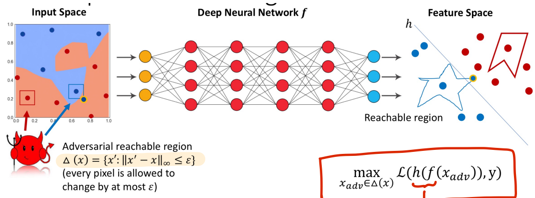
WHITE VS BLACK-BOX ATTACKS

"White-box attacks" are when the attacker needs to know full info about the network, whereas this is not the case for black-box attacks.

UNTARGETED VS TARGETED ATTACKS

1. "Untargeted attacks" are when the goal is to predict a wrong label.
2. "Targeted attacks" are when the goal is to predict a targeted label.

PRINCIPLE OF GENERATING EVASION ATTACKS



Adversarial reachable region $\Delta(x) = \{x' : \|x' - x\|_\infty \le \varepsilon\}$ (every pixel is allowed to change by at most ε)

$$\boxed{\max_{x_{adv}\in\Delta(x)} \mathcal{L}(h(\underbrace{f}_{}(x_{adv})), y)}$$

C = hof is the classifier.

1. Idea: We want to solve

$$\boxed{\max_{\|x_{adv}-x\|_\infty < \varepsilon} \mathcal{L}(C(x_{adv}), y)}$$

2. Different types of solvers:

   ① Zero-order — only access to NN output

   ② First-order — access to gradient info

   ③ Second-order — access to Hessian matrix

3. We focus on first-order solvers.

FAST GRADIENT SIGN METHOD / FGSM

1. Goal: we want to find

$$\boxed{\max_{(x_{adv} \text{ s.t. } \|x_{adv}-x\|_\infty < \varepsilon)} \mathcal{L}(C(x_{adv}, w), y).}$$

- this is hard to solve
- since C is non-convex

2. We can approximate

$$\boxed{\begin{aligned}&\mathcal{L}(C(x_{adv}, w), y)\\ &\approx \mathcal{L}(C(x,w), y) + \langle x_{adv} - x, \nabla_x \mathcal{L}(C(x,w), y)\rangle\end{aligned}}$$

(taylor expansion)

3. Hence, our problem reduces to

$$\boxed{\max_{x_{adv}:\ \|x - x_{adv}\|_\infty < \varepsilon} \langle x_{adv} - x, \nabla_x \mathcal{L}(C(x,w), y)\rangle}$$

4. Closed form solution:

$$\boxed{x^*_{adv} = x + \varepsilon \cdot \text{sign}(\nabla_x \mathcal{L}(C(x,w), y))}$$

Why? – Holder's inequality: $|\langle a, b\rangle| \le \|a\|_p \|b\|_q$, where $\frac{1}{p} + \frac{1}{q} = 1$, $p, q \ge 1$

Then, for any x_{adv}:

$$\begin{aligned}\text{obj}(x_{adv}) &= \langle x_{adv} - x, \nabla_x \mathcal{L}(C(x,w), y)\rangle\\ &\le \|x_{adv} - x\|_\infty \|\nabla_x \mathcal{L}(C(x,w), y)\|_1\\ &\quad \text{(by Holder's ineq)}\\ &\le \varepsilon \cdot \|\nabla_x \mathcal{L}(C(x,w), y)\|_1 .\end{aligned}$$

Next, note

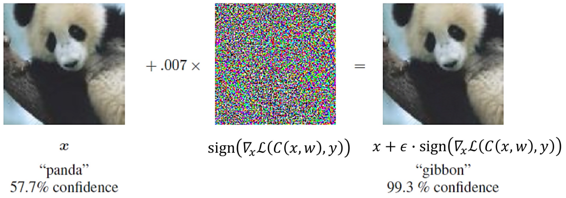
$$\begin{aligned}\text{obj}(x^*_{adv}) &= \text{obj}(x + \varepsilon \cdot \text{sign}(\nabla_x \mathcal{L}(C(x,w), y))\\ &= \langle \varepsilon \cdot \text{sign}(\nabla_x \mathcal{L}(C(x,w), y)),\\ &\qquad \nabla_x \mathcal{L}(C(x,w), y)\rangle\\ &= \varepsilon \cdot \|\nabla_x \mathcal{L}(C(x,w), y)\|_1 .\end{aligned}$$

(since $\text{sign}(a) \cdot a = |a|$, & $\mathcal{L}_1$ norm is just $\sum_{i=1} |x_i|$).

Hence, $\text{obj}(x^*_{adj})$ is the upper bound of the objective function, and so is the solution of the maximization problem. ☒

FACTS ABOUT FGSM

💡1 FGSM is a white-box, non-targeted evasion attack.


+ .007 ×
=
x
"panda"
57.7% confidence
$\text{sign}(\nabla_x \mathcal{L}(C(x,w),y))$
$x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(C(x,w),y))$
"gibbon"
99.3 % confidence

💡2 Issue: ϵ needs to be large for FGSM to be successful

BASIC ITERATIVE METHOD / BIM

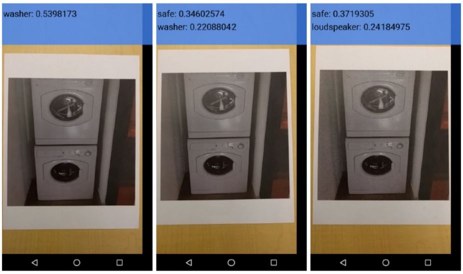
💡1 Idea: Improve FGSM by repeatedly adding noise to the image x in multiple iterations to cause misclassification:

$$\boxed{x^t = x^{t-1} + \underbrace{\gamma}_{\text{step size}} \cdot \text{sign}(\nabla_x \mathcal{L}(C(x^{t-1}, w), y))}$$

💡2 Differences with FGSM:

① step size is different ; &

② BIM uses an iterative procedure, whilst FGSM uses a one-shot procedure.



(b) Clean image (c) Adv. image Distance 4 (d) Adv. image Distance 8

💡3 Issue: For a pre-defined ϵ, x^t may violate the constraint $\|x' - x\|_\infty < \epsilon$ if t is large.

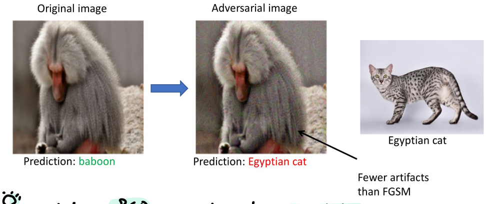
PROJECTED GRADIENT DESCENT / PGD

💡1 Idea: Improve BIM by using a truncation operation:

$$\boxed{x^t = \text{clip}_{(-\epsilon,\epsilon)}(x^{t-1} + \gamma \cdot \text{sign}(\nabla_x \mathcal{L}(C(x^{t-1}, w), y)))}$$

- for pixels with perturbation size $> \epsilon$. "clip" truncates them to ϵ.

💡2 PGD uses "random initialization" for x^0 by adding random noise to the original image from $\text{Unif}(-\epsilon, \epsilon)$.



💡3 Note PGD needs to calculate the gradient multiple times.

TARGETED PGD

💡1 Idea: We can manipulate PGD to be a targeted white-box attack.

💡2 Difference in objective:

① Untargeted:

$$\boxed{\max_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}),\ y_{true})}$$

② Targeted:

$$\boxed{\min_{x_{adv} \in \Delta(x)} \mathcal{L}(C(x_{adv}),\ y_{target})}$$

💡3 Iterations:

① Untargeted:

$$\boxed{x^t_{adv} = \text{clip}_{(-\epsilon,\epsilon)}(x^{t-1} + \gamma \cdot \text{sign}(\nabla_x \mathcal{L}(x^{t-1}, w),\ y_{true}))}$$

② Targeted:

$$\boxed{x^t_{adv} = \text{clip}_{(-\epsilon,\epsilon)}(x^{t-1} - \gamma \cdot \text{sign}(\nabla_x \mathcal{L}(C(x^{t-1}, w),\ y_{target})))}$$

MULTI-TARGETED PGD

1 Idea: Do targeted attacks with PGD for all target classes and choose the one that can fool the classifier.

2 This is an untargeted attack.

Chapter 15: Robustness

DEFENSES AGAINST EVASION ATTACKS: ADVERSARIAL TRAINING

Idea:

$$\boxed{\min_{C} \hat{E}_{x,y\sim D^n} \max_{x'\in\Delta(x)} \mathrm{Loss}(C(x'), y)}$$

outer min: mimic behaviors of attacks

inner max: update weight of neural nets

The adversarial examples attack the latest iterate of the classifier.

FGSM

Idea: Use FGSM to solve the inner maximization.

ENSEMBLE ADVERSARIAL TRAINING

Idea: Use a set of adversarial examples created by several fixed classifiers to train the model.

PGD

Idea: Use PGD to solve the inner max.

But this is computationally expensive to do.

ROBUSTNESS-ACCURACY TRADE-OFF

Idea: Adversarial training suffers from a reduced accuracy on clean samples; ie the "robustness-accuracy trade-off".

To quantify robustness, we can use the robustness error

$$\boxed{\begin{array}{l} R_{rob}(f) := E_{x,y\sim D}(\mathbb{I}[\exists x'\in\Delta(x) \text{ s.t. } f(x')y \le 0]), \\ y=\pm 1,\ f:\mathcal{X}\to\mathbb{R} \text{ is our classifie} \end{array}}$$

& the natural error

$$\boxed{R_{nat}(f) := E_{x,y\sim D}[\mathbb{I}[f(x)y \le 0]]}$$

We want to find

$$\boxed{\min_{f} R_{nat}(f) + \frac{R_{rob}(f)}{\lambda}.}$$

CLASSIFICATION-CALIBRATED SURROGATE LOSS

Idea: We want to design a differentiable surrogate loss for the trade-off.

TRADES

Idea: We want to find

$$\boxed{\min_{f}\left[E_{x,y\sim D} \phi(f(x)y) + E_{x,y\sim D} \max_{x'\in\Delta(x)} \phi\left(\frac{f(x)f(x')}{\lambda}\right)\right]}$$

- ϕ is the classification-calibrated loss (eg 0-1, exp, hinge, etc)

For any distribution D, f, $\Delta(x)$ & $\lambda>0$, we have

$$\boxed{R_{rob}(f) - R^*_{nat} \le \text{TRADES Loss}(f) - R^*_{\phi}}$$

& for any $\Delta(x)$, there exists a D, f & $\lambda>0$ such that

$$\boxed{R_{rob}(f) - R^*_{nat} \ge \text{TRADES Loss}(f) - R^*_{\phi}}$$

where R^*_{ϕ} & R^*_{nat} are the minimal values of $R_{\phi}(f) := E_{x,y\sim D}\phi(f(x)y)$ over f & $R_{nat}(f)$ respectively.

LIMITATIONS OF ADVERSARIAL TRAINING

💡 Idea: AT may not converge.

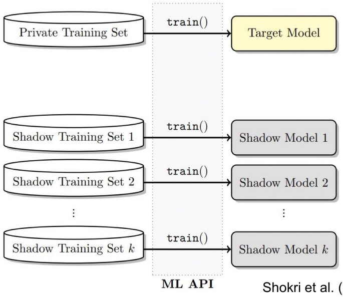
If $f(x) = w^T(x)$, the training dynamics of AT may lead to a cycle.

Chapter 16: Differential Privacy

We need to acknowledge privacy concerns if we train ML models on private data.

MEMBERSHIP INFERENCE

1. Goal: Determine whether a data instance x^* is part of the training dataset of a target model.
   - we assume we have black-box access to the model.
2. Attack technique: shadow training



Shokri et al. (20

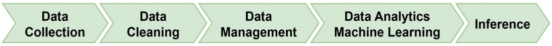
- we can then use these shadow models to replicate the target model
- & then use these to form the attack model

3. Note: these are
   ① not restricted to specific models; &
   ② is prone to overfitting.
   - the more prediction classes we have, the worse the test accuracy.

LOG PERPLEXITY

"(Log) perplexity" is a measurement of how well a model predicts a sample.

DATA SCIENCE LIFE CYCLE



Data Collection
Data Cleaning
Data Management
Data Analytics Machine Learning
Inference

PRIVACY CONCERNS IN DATA SCI LIFE CYCLE

1. Idea: Cloud services requires statistics (eg browser configurations) to monitor its performance.
2. However, users do not want to give up their data as it is very identifiable.
3. Moreover, often analysts will want to analyze sensitive datasets.

DIFFERENTIAL PRIVACY / DP

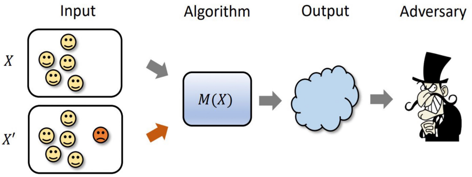
1. We say that a mechanism satisfies DP / ε-DP iff for all inputs X, X' that differ in one entry, we have that

$$\boxed{P(M(X)\in S) \le e^{\varepsilon} P(M(X')\in S)}$$

- probability is over all models M

for all outputs S.

- lower ε <=> more privacy



Note:
- if X, X' differ by adding/removing an entry, this is called "unbounded DP"
- if X, X' differ by replacement of an entry (ie |X|=|X'|), then this is called "bounded DP".

2. Intuitively, the adversary should not be able to use output S to distinguish between any X, X'.
3. Thus, privacy is not violated if one's information is not included in the input dataset.

BASIC COMPOSITION

If $M = (M_1, \dots, M_k)$ is a sequence of ε-DP mechanisms, then M is kε-DP.

POST-PROCESSING

If M(X) is ε-DP, then F(M(X)) is also ε-DP, where F is some function transformation.

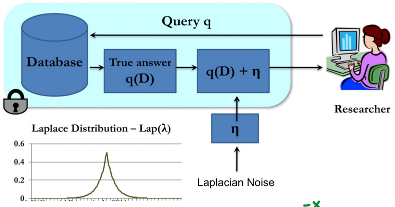
GROUP PRIVACY

If M(X) is ε-DP, & X, X' differ in k entries, then

$$\boxed{P(M(X) \in S) \le e^{k\varepsilon} P(M(X') \in S) \quad \forall S.}$$

LAPLACE MECHANISM

Idea: To achieve DP, we can add Laplacian noise to our model.



$\eta \sim \text{Laplace}(\lambda)$, pdf $\propto \exp\left(\frac{-x}{\lambda}\right)$

mean $= 0$, variance $= 2\lambda^2$

SENSITIVITY (S(q))

Let $q: I \to \mathbb{R}$ be a query. Then we define the "sensitivity" of q, S(q), to be the smallest number such that for any neighboring tables D, D' (ie that differ by one row), we have

$$\boxed{|q(D) - q(D')| \le S(q).}$$

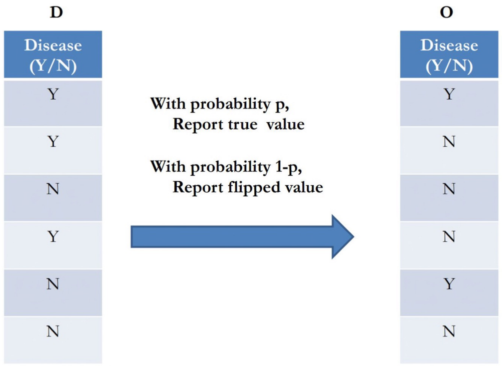
If the sensitivity of the query is S, then if we use

$$\boxed{\lambda = S/\varepsilon}$$

in our Laplacian noise, we are guaranteed to get ε-differential privacy.

DP APPLICATION: DATA COLLECTION

Idea: We can use DP to quantify the privacy of a data collection method.



Disease (Y/N)
Y
Y
N
Y
N
N

Disease (Y/N)
Y
N
N
N
Y
N

ie

$$O_i = \begin{cases} D_i, & \text{prob} = p \\ 1 - D_i, & \text{prob} = 1-p \end{cases}$$

- no privacy: $\gamma = 0$
- complete privacy: $\gamma = \frac{1}{2}$

Specifically, if we have 2 neighboring databases D, D', then for some output O:

$$\boxed{\frac{P(M(D) = O)}{P(M(D') = O)} \le e^{\varepsilon} \iff \frac{1}{1+e^{\varepsilon}} < p < \frac{e^{\varepsilon}}{1+e^{\varepsilon}}}$$

where M is our model.

BOUNDING SENSITIVITY

Idea: In some cases, the sensitivity of a query may be large or infinite.

To mitigate this, we can use

① "clipping" — enforce $x \in [a,b]$ and discard data out of the range

- but this adds bias to the output

② "subsample & aggregate" — partition X into $X_1, \dots, X_n$, apply f over each subset. and aggregate the results.

APPROXIMATE DP / (ε-δ)-DP

We say a mechanism is "approximately DP" if for some ε, δ,

$$\boxed{P(M(X) \in S) \le e^{\varepsilon} P(M(X') \in S) + \delta}$$

for all neighboring data X & X'.

- note δ should be very small.

To achieve this, we can add Gaussian noise.

DP-APPLICATION: DP-SGD

Method:

① Sample a "lot" of points of expected size L by selecting each point to be in the lot with probability $\frac{L}{n}$

② For each point in the lot, compute the gradient $\nabla \ell(\theta_t, x, y)$ & clip so it has ℓ_2 norm $\leq C$

③ Average the clipped gradients & add Gaussian noise

④ Take a step in the negative direction of the resulting vector

⑤ Repeat k times

Limitations:

① Slower than SGD

② Hyperparameter tuning

ε-LOCAL DP

We say M provides "ε-local DP" if for all pairs of (private) data x & x', we have

$$\boxed{P(M(x) \in S) \leq e^{\varepsilon} P(M(x') \in S)}$$

for all outputs S

In particular, M takes in a single user's data, whereas for normal ε-DP, M takes in all users' data.

Chapter 17: Private Data Synthesis

SYNTHETIC DATASET

1. A "synthetic dataset" is a stand-in for the original dataset that has the same format & accurately reflects the statistical properties of the original dataset, but only contains "fake" records.
2. Note that a synthetic dataset does not guarantee privacy.
3. The generation process is ε-DP, & all other queries on the synthetic dataset is just post-processing.
4. However, there are no accuracy guarantees.

NAIVE METHOD

1. Method:
   ① Learn the data distribution and preserve some properties;
   ② Add noise to the learning process; &
   ③ Sample from the learnt distribution.
2. Challenge: what properties to preserve & how to preserve them?

LARGE DATASETS

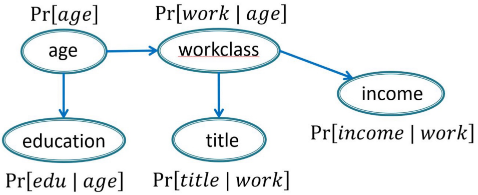
1. Idea: When the dataset is large, the number of combinations in the "joint distribution" is intractable.
2. So, privatizing each count is expensive wrt the privacy cost, and hence is inefficient.

IMPROVED METHOD

1. Idea: Selectively learn some "low-way" marginal distributions with noise, & combine them in a way to approximate the joint distribution.
2. Method:
   ① Learn the correlation among the attributes to select marginals;
   ② Learn the selected marginals;
   ③ Combine the marginals to get the joint distribution; &
   ④ Sample from this joint distribution.

PRIV BAYES

1. Idea: PrivBayes is a Bayesian network we can use to
   ① learn the correlation;
   ② privatize the correlation learning; &
   ③ combine the selected noisy marginals.



2. Method:
   ① Construct a suitable Bayesian network N with ε-DP;
   ② Compute the conditional distributions implied by N;
   ③ Add Laplace noise; &
   ④ Generate synthetic data by sampling from the joint distribution N, by approximating the joint distribution N, using factorization of N.,