

CS 480



Personal Notes

Marcus Chan

Taught by Hongyang Zhang

uw cs '25



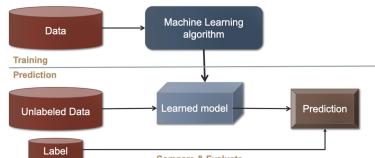
Chapter 1: Perceptrons

ML

Q1 "Machine learning" is a branch of AI that focuses on methods that learn from data & make predictions on unseen data.

Q2 3 phases:

- ① training;
- ② prediction; &
- ③ evaluation.



PARADIGMS OF ML ALGOS (TRAINING)

Q1 "Supervised model": learning with labelled data (x, y)
eg email classification, image classification

Q2 "Unsupervised model": discover patterns in unlabeled data x
eg cluster similar data points, reduce data dimension
etc

Q3 "Semi-supervised model": using both labelled & unlabelled data

WHAT A DATASET LOOKS LIKE

	Training samples					Test samples		
	x_1	x_2	x_3	x_4	\dots	x_n	x'_1	x'_2
$\mathbb{R}^d \ni \text{Feature}$	0	1	0	1	\dots	1	1	0.9
	0	0	1	1	\dots	0	1	1.1
	\vdots	\vdots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots
	1	0	1	0	\dots	1	1	-0.1
Label y	+	+	-	+	\dots	-	?	?

- each column is a data point, n in total & each with d features
- y is the "label vector"
- x' & x'_2 are the test samples whose labels need to be predicted.
- (we use " x " to denote test samples)

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

Q1 Define the "inner product" of a & b to be

$$\langle a, b \rangle = \sum_j a_j b_j,$$

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

LINEAR FUNCTION

Q1 We say a function f is "linear" if

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

Q2 Equivalently, f is linear iff there exists $w \in \mathbb{R}^d$ such that

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

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

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

(\Leftarrow) Note

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

AFFINE FUNCTION

Q1 We say f is an "affine function" if there exists a $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ such that

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

SCORE: \hat{y}

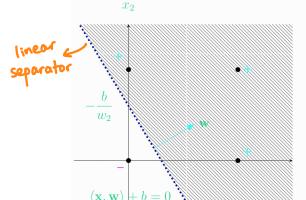
Q1 Given $w \in \mathbb{R}^d$, $b \in \mathbb{R}$, define the "score" at some $x \in \mathbb{R}^d$ to be

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

Q2 Our "prediction" for y is then

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

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



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

PERCEPTRONS

Algorithm for training:

Algorithm 1 Training Perceptron

Input: Dataset = $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\}$: $i = 1, \dots, n$, initialization $\mathbf{w}_0 \in \mathbb{R}^d$ and $b_0 \in \mathbb{R}$

Output: \mathbf{w} and b (so a linear classifier $\text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b)$)

for $t = 1, 2, \dots$ do

- receive index $I_t \in \{1, \dots, n\}$ // I_t can be random
- if $y_{I_t}(\langle \mathbf{x}_{I_t}, \mathbf{w} \rangle + b) \leq 0$ // a "mistake" happens
- then

 - $\mathbf{w} \leftarrow \mathbf{w} + y_{I_t} \mathbf{x}_{I_t}$ // update after a "mistake"
 - $b \leftarrow b + y_{I_t}$

- end

end

- 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 $\mathbf{w} \in \mathbb{R}^d$, $b \in \mathbb{R}$ such that for all $i=1, \dots, n$,

$$y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

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

$$\begin{aligned} y[\langle \mathbf{x}, \mathbf{w}_{k+1} \rangle + b_{k+1}] &= y[\langle \mathbf{x}, \mathbf{w}_k + y \mathbf{x} \rangle + b_k + y] \\ &= y[\langle \mathbf{x}, \mathbf{w}_k \rangle + y \langle \mathbf{x}, \mathbf{x} \rangle + b_k + y] \\ &= y[\langle \mathbf{x}, \mathbf{w}_k \rangle + y \|\mathbf{x}\|_2^2 + b_k + y] \\ &= y[\langle \mathbf{x}, \mathbf{w}_k \rangle + b_k] + y^2 \|\mathbf{x}\|_2^2 + y^2 \\ &= y[\langle \mathbf{x}, \mathbf{w}_k \rangle + b_k] + \underbrace{\|\mathbf{x}\|_2^2 + 1}_{\text{always positive } \& \geq 1}. \end{aligned}$$

Example: spam filtering.

	x_1	x_2	x_3	x_4	x_5	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: $\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}$, $b \leftarrow b + y$ (when a mistake happens on (x, y))

- $\mathbf{w}_0 = [0, 0, 0, 0, 0]$, $b_0 = 0 \implies \text{score}(\mathbf{x}_1) = 0 \implies \hat{y}_1 = - \quad x$
- $\mathbf{w}_1 = [1, 1, 0, 1, 1]$, $b_1 = 1 \implies \text{score}(\mathbf{x}_2) = 2 \implies \hat{y}_2 = + \quad x$
- $\mathbf{w}_2 = [1, 1, -1, 0, 1]$, $b_2 = 0 \implies \text{score}(\mathbf{x}_3) = 0 \implies \hat{y}_3 = - \quad x$
- $\mathbf{w}_3 = [1, 2, 0, 0, 1]$, $b_3 = 1 \implies \text{score}(\mathbf{x}_4) = 2 \implies \hat{y}_4 = + \quad x$
- $\mathbf{w}_4 = [0, 2, 0, -1, 1]$, $b_4 = 0 \implies \text{score}(\mathbf{x}_5) = 1 \implies \hat{y}_5 = + \quad \checkmark$
- $\mathbf{w}_4 = [0, 2, 0, -1, 1]$, $b_4 = 0 \implies \text{score}(\mathbf{x}_6) = -1 \implies \hat{y}_6 = - \quad \checkmark$

13 / 2

A TRICK TO HIDE THE BIAS TERM

Note that

$$\langle \mathbf{x}, \mathbf{w} \rangle + b = \langle \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}, \begin{pmatrix} \mathbf{w} \\ b \end{pmatrix} \rangle$$

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

Thus, our new update rule is

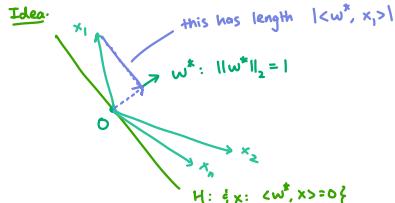
$$\mathbf{w}_{\text{pad}} \leftarrow \mathbf{w}_{\text{pad}} + y \mathbf{x}_{\text{pad}}.$$

CONVERGENCE THEOREM (LINEARLY SEPARABLE CASE)

\bullet Suppose there exists a w^* such that
 $y_i \langle x_i, w^* \rangle > 0 \quad \forall i=1, \dots, n.$

Assume $\|x_i\|_2 \leq C$ 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.



- 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

$$\cos(w, w^*) = \frac{\langle w, w^* \rangle}{\|w\| \|w^*\|} = \frac{\langle w, w^* \rangle}{\|w\|} \quad (\text{since we defined } \|w^*\|=1).$$

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

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

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

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

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

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

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

Since

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

Therefore

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

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

\bullet 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

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

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

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

Our gradient descent update:

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

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

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

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

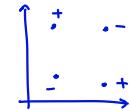
We want to choose w such that

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

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:

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

$$x_2 = (1,0), y_2 = + \Rightarrow w_1 + b > 0 \quad \left\{ \begin{array}{l} w_1 + w_2 + 2b > 0 \\ w_2 + b > 0 \end{array} \right. \quad > 0$$

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

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

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)

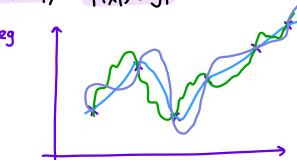
If there is no perfect separating hyperplane for our data, then the Perceptron algorithm cycles.

Chapter 2: Linear Regression

Q₁ Idea: Given training data (x_i, y_i) , find a $f: X \rightarrow Y$ such that $f(x_i) \approx y_i$, where

- ① $x_i \in X \subseteq \mathbb{R}^d$: the feature vector for the i^{th} training example
- ② $y_i \in Y \subseteq \mathbb{R}^t$: t responses
 - note we could have $t=1$ or even $t=\infty$

Q₂ 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 , $f(x_i) = y_i$.



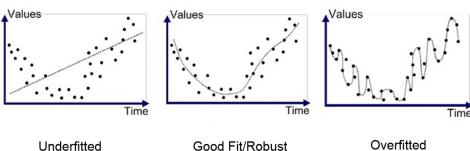
Q₃ Moreover, our prediction $\hat{y} = f(x)$ can vary significantly on new data x !

Q₄ 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



STATISTICAL LEARNING

Q We assume the training & test data are both iid samples from the same unknown distribution P ; ie

$$(x_i, y_i) \sim P$$
$$(x, y) \sim P.$$

LEAST SQUARES REGRESSION

Q We want to choose f so that

$$f = \min_{f: X \rightarrow Y} E \|f(x) - y\|_2^2.$$

this is our least squared error.

REGRESSION FUNCTION: $m(x)$

Our "regression function" is

$$f^*(x) = m(x) = \mathbb{E}[y | x=x].$$

However, calculating m requires us to know the distribution of P , ie all pairs (X, Y) .

We show that m is optimal; ie

$$m(x) = \min_{f: \mathbb{R} \rightarrow \mathbb{R}} \mathbb{E}_{x \sim P} \|f(x) - y\|_2^2.$$

Proof. First, see that

$$\begin{aligned} \mathbb{E} \|f(x) - y\|_2^2 &= \mathbb{E} \|\mathbb{E}[f(x) - m(x) + m(x) - y]\|_2^2 \\ &= \mathbb{E} \|f(x) - m(x)\|_2^2 + \mathbb{E} \|m(x) - y\|_2^2 \\ &\quad + 2 \mathbb{E} \langle f(x) - m(x), m(x) - y \rangle. \end{aligned}$$

Using $\|ab\|_2^2 = \|a\|_2^2 + \|b\|_2^2 + 2\langle a, b \rangle$

Then

$$\begin{aligned} \mathbb{E}_{x,y} [\langle f(x) - m(x), m(x) - y \rangle] &= \mathbb{E}_x [\mathbb{E}_{y|x} [\langle f(x) - m(x), m(x) - y \rangle]] \\ &\quad (\text{by double expectation theorem, see STAT 330}) \\ &= \mathbb{E}_x [\langle f(x) - m(x), m(x) - \mathbb{E}[y|x] \rangle] \\ &= \mathbb{E}_x [\langle f(x) - m(x), 0 \rangle] \\ &= 0. \end{aligned}$$

Hence

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

- independent wrt f .

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

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

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

$$\begin{aligned} \mathbb{E}_{D, X, Y} \|f_D(x) - y\|_2^2 &= \mathbb{E}_x \|\mathbb{E}_D [f_D(x)] - m(x)\|_2^2 \\ &\quad \underbrace{\text{test error}}_{\text{bias}^2} \\ &\quad + \mathbb{E}_{D, X} \|f_D(x) - \mathbb{E}_D [f_D(x)]\|_2^2 \\ &\quad \underbrace{\text{variance}}_{\text{noise}} \\ &\quad + \mathbb{E}_{X, Y} \|m(x) - y\|_2^2 \end{aligned}$$

Proof. We have shown

$$\begin{aligned} \mathbb{E}_{X, Y} \|f_D(x) - y\|_2^2 &= \mathbb{E}_x \|\mathbb{E}_D [f_D(x)] - m(x)\|_2^2 \\ &\quad + \underbrace{\mathbb{E}_{X, Y} \|m(x) - y\|_2^2}_{\text{noise - independent wrt } f_D}. \end{aligned}$$

Taking E_D of both sides:

$$\begin{aligned} \mathbb{E}_D \mathbb{E}_{X, Y} \|f_D(x) - y\|_2^2 &= \mathbb{E}_D \mathbb{E}_X \|\mathbb{E}_D [f_D(x)] - m(x)\|_2^2 \\ &\quad + \mathbb{E}_{X, Y} \|m(x) - y\|_2^2. \quad \text{①} \end{aligned}$$

Define $\bar{f}(x) = \mathbb{E}_D [f_D(x)]$.

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

$$\begin{aligned} D_i \sim P &\rightarrow f_{D_i} \\ &\vdots \\ D_n \sim P &\rightarrow f_{D_n} \end{aligned} \quad \left\{ \begin{array}{l} \text{then we define} \\ \bar{f}(x) = \text{avg } f_{D_i}(x). \end{array} \right.$$

Then

$$\begin{aligned} \mathbb{E}_D \mathbb{E}_X \|f_D(x) - m(x)\|_2^2 &= \mathbb{E}_{D, X} \|f_D(x) - \bar{f}(x) + \bar{f}(x) - m(x)\|_2^2 \\ &= \mathbb{E}_{D, X} \|f_D(x) - \bar{f}(x)\|_2^2 + \mathbb{E}_{D, X} \|\bar{f}(x) - m(x)\|_2^2 \\ &\quad + 2 \mathbb{E}_{D, X} \langle f_D(x) - \bar{f}(x), \bar{f}(x) - m(x) \rangle. \end{aligned}$$

Similarly, see that

$$\begin{aligned} \mathbb{E}_{D, X} \langle \bar{f}(x) - f_D(x), m(x) - \bar{f}(x) \rangle &= \mathbb{E}_X \mathbb{E}_D \langle m(x) - \bar{f}(x), \bar{f}(x) - f_D(x) \rangle \\ &\quad \text{constant wrt } D \\ &= \mathbb{E}_X \langle m(x) - \bar{f}(x), \bar{f}(x) - \underbrace{\mathbb{E}_D [f_D(x)]}_{\bar{f}(x)} \rangle \\ &= 0. \end{aligned}$$

Expanding ① yields the result desired. \blacksquare

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

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

$$f = \min_{f: x \rightarrow y} \hat{E} \|f(x) - y\|_2^2 := \frac{1}{n} \sum_{i=1}^n \|f(x_i) - y_i\|_2^2.$$

However, as our training data size $n \rightarrow \infty$, $\hat{E} \rightarrow E$ & hopefully $\operatorname{argmin} \hat{E} \rightarrow \operatorname{argmin} E$.

LINEAR REGRESSION

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

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

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

Again, we can use padding:

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

In matrix form:

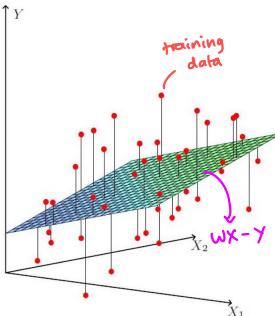
$$\frac{1}{n} \sum_i \|f(x_i) - y_i\|_2^2 = \frac{1}{n} \|Wx - y\|_F^2,$$

$$X \in [\dots, x_n] \in \mathbb{R}^{(d+1) \times n}, \quad Y = [y_1, \dots, y_n] \in \mathbb{R}^{t \times n},$$

$$\|A\|_F = \sqrt{\sum_{i,j} a_{ij}^2}$$

We want to find W such that

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



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

SOLVING LINEAR REGRESSION

We define our loss function as

$$\text{Loss}(W) = \frac{1}{n} \|Wx - y\|_F^2$$

Taking the derivative wrt W & setting to zero:

$$\begin{aligned} \nabla_W \text{Loss}(W) &= \frac{2}{n} (Wx - y) X^T (= 0) \\ \Rightarrow Wx X^T &= y X^T \\ \Rightarrow W &= y X^T (X X^T)^{-1} \end{aligned}$$

PREDICTION

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

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

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

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

The "training error" is

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

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

ILL-CONDITIONING

Consider $X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, $y = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. Solving linear least squares regression:

$$w = y X^T (X X^T)^{-1} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} -2/3 \\ 1/3 \\ 1 \end{pmatrix}$$

So slight perturbation leads to chaotic behavior!

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

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

RIDGE REGRESSION

Idea: We instead try to find

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

Why is this better?

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

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

$$\Rightarrow WX X^T - Y X^T + 2\lambda W = 0$$

$$WX X^T - Y X^T + W(2\lambda I) = 0$$

$$WX X^T + W(2\lambda I) = Y X^T$$

$$\therefore W = (X X^T + 2\lambda I)^{-1} (Y X^T)$$

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

② controls our trade-off:

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

② $\lambda=\infty$ reduces to $W=0$;

③ intermediate λ restricts output to be

$\frac{1}{\lambda}$ proportional to input.

Alternatively, note

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

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 $\tilde{Y} = (Y \mathbf{0})$

(ie data augmentation) to achieve regularization.

Chapter 3: Logistic Regression

MOTIVATION

Q₁: This is for linear classification.

Q₂: We can use $\|x; w\|$ (our margin) as a measure of our confidence in the prediction \hat{y} .

Q₃: However, as this is un-normalized, it is hard to interpret.

MAXIMUM LIKELIHOOD ESTIMATE

Q₁: We want to directly learn our "confidence".

$$p(x; w) := P(Y=1 | X=x)$$

Q₂: Then, if $y_1, \dots, y_n, x_1, \dots, x_n$ are independent, then

$$\begin{aligned} & P(y_1=y_1, \dots, y_n=y_n | x_1=x_1, \dots, x_n=x_n) \\ &= \prod_{i=1}^n P(y_i=y_i | x_i=x_i) \\ &= \prod_{i=1}^n [p(x_i; w)]^{y_i} [1-p(x_i; w)]^{1-y_i} \quad \text{if } y_i \in \{0, 1\} \end{aligned}$$

Q₃: Maximizing the likelihood:

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

Q₄: We thus want to find w which satisfies the above optimization problem.

THE LOGIT TRANSFORM

Q₁: If we assume the log of odds ratio is linear: ie

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

then

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

↳ this is also called the "sigmoid transformation".

Q₂: Plugging this into the earlier optimization problem, we want to find

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

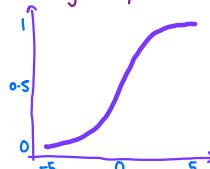
if $y_i \in \{0, 1\}$.

Q₃: If instead $y_i \in \{-1, 1\}$, then

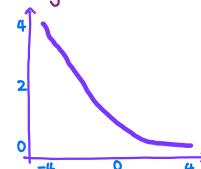
$$\min_w \sum_{i=1}^n \log [1 + \exp(-y_i \langle x_i, w \rangle)]$$

↳ this is "logistic loss".

Sigmoid function



logistic loss



TRAINING LOGISTIC REGRESSION

Q Our gradient descent algorithm is

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

PREDICTION

Q₁ We take

$$\hat{y} = 1 \Leftrightarrow P(Y=1 | X=x) > \frac{1}{2} \Leftrightarrow \langle x, w \rangle > 0$$

Q₂ Our decision boundary is still

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

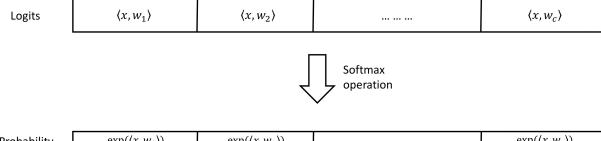
Q₃ So we can predict $\hat{y} = \text{sign}(\langle x, w \rangle)$ as before,
but now with confidence $p(x; w)$.

MULTI-CLASS EXTENSION

Q₁ Idea: For a class $y \in \{1, \dots, c\}$, we want
to learn $\{w_1, \dots, w_c\}$ for each class.

Q₂ We consider the "softmax" function:

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



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

Q₃ Training: again, we use MLE:

$$\min_w E \left[-\log \frac{\exp(\langle x, w_y \rangle)}{\sum_{l=1}^c \exp(\langle x, w_l \rangle)} \right]$$

Q₄ Prediction:

$$\hat{y} = \underset{k}{\operatorname{argmax}} P(Y=k | X=x; W=[w_1, \dots, w_c])$$

Chapter 4: Hard-Margin Support Vector Machines

INTRODUCTION

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

$\textcircled{2}$: Perceptron: we find any $w^*, b \in \mathbb{R}$ such that

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

$\textcircled{3}$: However, the larger the margin, the faster Perceptron converges.

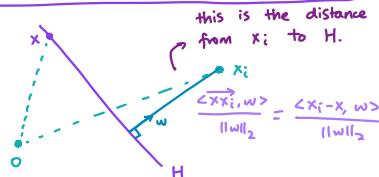
recall # mistakes, $M \leq \frac{C^2}{\gamma^2}$, $\|x_i\|_2 \leq C$, $\gamma = \min_i |\langle x_i, w^* \rangle|$, $\|w^*\|_2 = 1$.

$\textcircled{4}$: So, the goal of hard-margin SVM is to maximize the margin assuming data is linearly separable.

DISTANCE FROM A POINT TO A HYPERPLANE

$\textcircled{1}$: Let $H := \{x : \langle x, w \rangle + b = 0\}$. Then

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

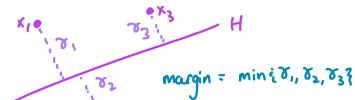


MARGIN

$\textcircled{1}$: We define the "margin" as the smallest distance to a separating hyperplane H among all separable training data; ie

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

eg



$\textcircled{2}$: Our goal is to maximize the margin among all hyperplanes: ie find

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

TRANSFORMING TO STANDARD FORM

- Q₁: Note for the margin, (w, b) & (cw, cb) has the same loss for $c > 0$.
- Q₂: So, we can fix the numerator arbitrarily to 1:

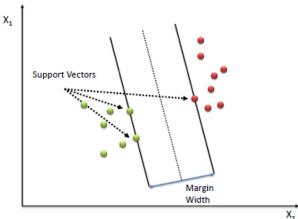
$$\max_{w, b} \left[\frac{1}{\|w\|_2} \text{ s.t. } \min_i y_i \hat{y}_i = 1 \right]$$
$$\Rightarrow \min_{w, b} \left[\frac{1}{2} \|w\|_2^2 \text{ s.t. } y_i (\langle x_i, w \rangle + b) \geq 1 \forall i \right]$$

COMPARISON TO PERCEPTRON

Hard-margin SVM	Perceptron
$\min_{w, b} \frac{1}{2} \ w\ _2^2 \text{ s.t. } y_i \hat{y}_i \geq 1 \forall i$	$\min_{w, b} 0 \text{ s.t. } y_i \hat{y}_i \geq 1 \forall i$
- quadratic programming	- linear programming
- unique solution	- infinitely many solutions
- maximal margin	- convergence rate depends on max margin

SUPPORT VECTORS

- Q₁: Note that
- $$y_i \hat{y}_i \geq 1 \forall i \Leftrightarrow \hat{y}_i \geq +1 \forall i: y_i = +1$$
- $$\hat{y}_i \leq -1 \forall i: y_i = -1$$
- Q₂: This yields 3 parallel hyperplanes:
- $$H = \{x: \langle x, w \rangle + b = 0\}$$
- $$H^+ = \{x: \langle x, w \rangle + b = +1\}$$
- $$H^- = \{x: \langle x, w \rangle + b = -1\}$$
- Q₃: "Support vectors" are those where points lie on the supporting hyperplanes.



LAGRANGIAN DUAL

First, we show

$$\begin{aligned} & \min_{w,b} \frac{1}{2} \|w\|_2^2 \quad \text{s.t. } y_i(\langle x_i, w \rangle + b) \geq 1 \quad \forall i \\ &= \min_{w,b} \max_{\alpha \geq 0} \frac{1}{2} \|w\|_2^2 - \sum_i \alpha_i [y_i(\langle x_i, w \rangle + b) - 1] \\ & \quad \downarrow \\ & \alpha = [\alpha_1, \dots, \alpha_n] \in \mathbb{R}^n; \\ & \alpha \geq 0 \Leftrightarrow \alpha_i \geq 0 \quad \forall i \end{aligned}$$

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

$$\begin{aligned} \Delta &= \frac{1}{2} \|w\|_2^2 - \sum_i \underbrace{\alpha_i}_{\text{tve}} \underbrace{[y_i(\langle x_i, w \rangle + b) - 1]}_{\text{tve}} \\ &\leq \frac{1}{2} \|w\|_2^2. \end{aligned}$$

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

Therefore,

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

as needed. \square

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")

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

Then w, b minimizes the function if

$$\frac{\partial}{\partial w} = \frac{\partial}{\partial b} = 0.$$

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

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

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

Finally, we consider the "outer" maximization problem.

Plugging in our value of w above:

$$\begin{aligned} \Rightarrow \text{Loss}(\alpha) &= \frac{1}{2} \left\| \sum_i \alpha_i y_i x_i \right\|_2^2 - \left\langle \sum_i \alpha_i y_i x_i, \sum_i \alpha_i y_i \right\rangle \\ &\quad - b \sum_i \alpha_i y_i + \sum_i \alpha_i \\ &= -\frac{1}{2} \left\| \sum_i \alpha_i y_i x_i \right\|_2^2 + \sum_i \alpha_i \quad \text{s.t. } \sum_i \alpha_i y_i = 0 \end{aligned}$$

Thus, our problem becomes

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

WHY USE THE DUAL FORM?

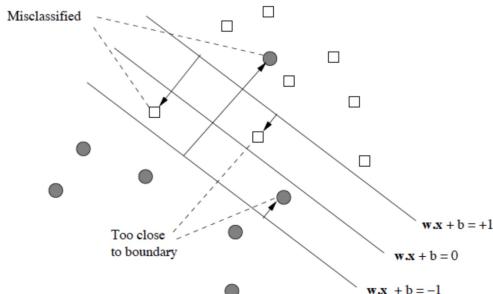
Idea: If data is not linearly separable, we use a non-linear mapping ϕ to map the data.

$$\begin{aligned} \min_{\alpha \geq 0} & - \sum_i \alpha_i + \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \\ \text{s.t. } & \sum_i \alpha_i y_i = 0. \end{aligned}$$

Chapter 5: Soft-Margin Support Vector Machines

MOTIVATION

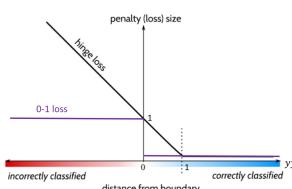
- Q₁: Hard-margin SVMs assume the data is linearly separable, but this is not always the case.
- Q₂: We want to adapt this to work for non-linearly separable data.
- Q₃: To do this, we will penalize our loss if the data falls too close to the boundary, or if the data is misclassified.



THE HINGE LOSS

- Q₁: We want to penalize the case where $y(x, w+b) < 1$, where $y = \pm 1$ is our true label, & $\hat{y} = \langle x, w \rangle + b$ is our predicted confidence.
- Q₂: Define the "hinge loss function" to be

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



* note: we define

$$l_{\text{hinge}}(t) = \begin{cases} -1, & t \leq 1 \\ 0, & t > 1 \\ \alpha, & t = 1 \end{cases}$$

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

SOFT-MARGIN SVM

- Q₁: The "soft-margin SVM" balances between margin maximization & the hinge loss:

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

we penalize error
& small margin

SOFT VS HARD-MARGIN SVM

- Q₁: For hard-margin SVM, we have a **hard constraint** that $y_i(\langle x_i, w \rangle + b) \geq 1 \quad \forall i$.
- Q₂: For soft-margin SVM, we have a **soft constraint**; the more you deviate from the margin, the heavier the penalty.

WHY THE HINGE LOSS?

- Q₁: Our goal is to find

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

true label predicted label

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

- Q₂: This is equivalent to

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

where I is the indicator function, &

δ_{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

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

Note that

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

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 ,

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

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 sum is

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

Deriving the dual:

$$\text{Apply } C(t_i)^+ = \max\{ct_i, 0\} = \max_{0 \leq \alpha_i \leq C} \alpha_i t_i, \text{ and set } t_i = 1 - y_i(\langle x_i, w \rangle + b) \text{ to get}$$

$$\min_{w, b} \max_{0 \leq \alpha_i \leq C} \frac{1}{2} \|w\|_2^2 + \sum_{i=1}^n \alpha_i (1 - y_i(\langle x_i, w \rangle + b)),$$

$$0 \leq \alpha_i \leq C \Leftrightarrow 0 \leq \alpha_i \leq C \quad \forall i$$

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

$$\max_{0 \leq \alpha_i \leq C} \min_{w, b} \frac{1}{2} \|w\|_2^2 + \sum_i \alpha_i (1 - y_i(\langle x_i, w \rangle + b)).$$

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:

$$\begin{aligned}\max_{0 \leq \alpha_i \leq C} \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|_2^2 + \sum_{i=1}^n \alpha_i \\ - \underbrace{\sum_{i=1}^n \alpha_i y_i \langle x_i, \sum_{j=1}^n \alpha_j y_j x_j \rangle}_{\left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|_2^2} - \underbrace{\sum_{i=1}^n \alpha_i y_i}_{0}\end{aligned}$$

$$= \max_{0 \leq \alpha_i \leq C} \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|_2^2 + \sum_{i=1}^n \alpha_i - \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|_2^2$$

$$= \max_{0 \leq \alpha_i \leq C} \sum_{i=1}^n \alpha_i - \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|_2^2$$

Thus, the dual form is

$$\max_{0 \leq \alpha_i \leq C} \sum_{i=1}^n \alpha_i - \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|_2^2 \quad \text{s.t. } \sum_i \alpha_i y_i = 0$$

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

Note that if

① $C \rightarrow \infty$, we get a hard-margin sum; &

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

COMPLEMENTARITY SICKNESS

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

$\textcircled{2}$ Then note that

$$\textcircled{1} t > 0 \Rightarrow \alpha^* = C, \quad \alpha^* = C \Rightarrow t \geq 0$$

$$\textcircled{2} t < 0 \Rightarrow \alpha^* = 0, \quad \alpha^* = 0 \Rightarrow t \leq 0$$

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

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

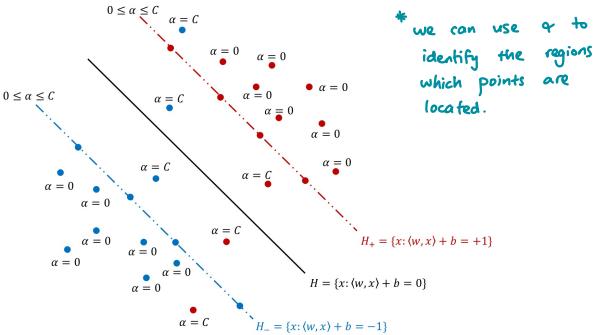
(ie margin/wrong idea)

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

(ie correctly classified with good confidence)

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

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



RECOVERING w & b FROM DUAL

$\textcircled{1}$ We can obtain w & b via

$$w = \sum_i \alpha_i y_i x_i.$$

$\textcircled{2}$ 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.

$\textcircled{3}$ Then we can recover b via

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

Since $y = \pm 1$.

$\textcircled{4}$ we can then predict new data via

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

Chapter 6: Reproducing Kernels

MOTIVATION

- Q₁: A lot of data are not linearly separable, and requires more complex classifiers.

QUADRATIC CLASSIFIER

- Q₁: The "quadratic classifier" has score function

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

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

- Q₂: We can then predict via

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

THE POWER OF LIFTING

- Q₁: We can express

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

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

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

$$A = (a_{ij})_{d \times d}$$

$$\vec{A} = \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \\ \vdots \\ a_{1d} \\ a_{21} \\ \vdots \\ a_{2d} \\ \vdots \\ a_{d1} \\ \vdots \\ a_{dd} \end{pmatrix} \in \mathbb{R}^{d^2}$$

- Q₂: Thus, the quadratic classifier is linear wrt $\phi(x)$.

THE KERNEL TRICK

- Q₁: The feature map ϕ blows up the dimension.

- Q₂: But in the dual form of SVM, we only need to consider

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

- Q₃: 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

- Q₁: We call $k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ a "reproducing kernel" if there exists some feature transform $\phi: \mathcal{X} \rightarrow \mathcal{H}$ such that

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

- Q₂: Note that choosing ϕ uniquely determines k .

MERCER'S THEOREM

$k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ 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)
 $k(x, z) = \exp(-\|x - z\|_1 / \sigma)$ (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 $k_1 k_2$ is a kernel;

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

KERNEL SUM

The kernel SVM's primal form is

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

and the dual form is

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

where ϕ & k are related via

Mercer's theorem.

$$\text{i.e. } k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

PREDICTION

Suppose that $0 \leq \alpha^* \leq C$ optimizes the kernel SVM.
Then, we can recover

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

Finally, our score function is

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

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

Chapter 7: Gradient Descent

MOTIVATION

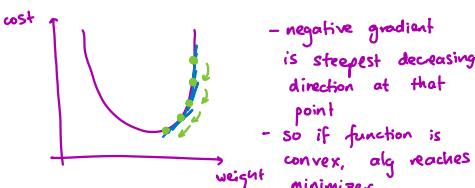
Q₁: Many ML methods can be classed as optimization problems; ie

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

Q₂: Assume f is differentiable with gradient $\nabla f(x)$.

Q₃: Idea: Choose an initial point $x^{(0)} \in \mathbb{R}^n$ and iteratively calculate

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



EXAMPLE: PERCEPTRON

Q₁: For perceptron, our gradient descent is

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

Q₂: Stochastic gradient descent update:

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

EXAMPLE: SOFT-MARGIN SVM

Q₁: Gradient descent update for soft-margin SVM:

$$\begin{aligned} w &\leftarrow w - t \left[\frac{w}{n} + \frac{1}{n} \sum_{i=1}^n x_i^T \text{hinge}(y_i \hat{y}_i) y_i x_i \right] \\ b &\leftarrow b - t \left[\frac{1}{n} \sum_{i=1}^n x_i^T \text{hinge}(y_i \hat{y}_i) y_i \right] \end{aligned}$$

INTERPRETATION FROM TAYLOR EXPANSION

Q₁: Note that if we take the Taylor expansion of f at y , we get

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

Q₂: Hence

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

Q₃: Then see that

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

and this is exactly the gradient descent template.

STEP SIZE

Q₁: Note the step size cannot be too large or too small.

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

Q₂: So, we need to find t such that the algorithm converges nicely.

CONVEX FUNCTION

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

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



L-LIPSCHITZ CONTINUOUS

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

Q: Here,

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

Q: 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

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

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

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

$$\begin{aligned} f(y) &\leq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}(y-x)^T \nabla^2 f(x)(y-x) \\ &\leq f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}(y-x)^T(LI)(y-x) \\ (\because LI \leq \nabla^2 f(x) \Rightarrow (y-x)^T(LI - \nabla^2 f(x))(y-x) \geq 0) \\ &= f(x) + \nabla f(x)^T(y-x) + \frac{1}{2}\|y-x\|_2^2. \end{aligned}$$

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

$$\begin{aligned} \Rightarrow f(x^+) &\leq f(x) + \nabla f(x)^T(x - t \nabla f(x) - x) \\ &\quad + \frac{t}{2} \|x - t \nabla f(x) - x\|_2^2 \\ &= f(x) - t \|\nabla f(x)\|_2^2 + \frac{t^2}{2} \|\nabla f(x)\|_2^2 \\ &= f(x) - (1 - \frac{t^2}{2}) \|\nabla f(x)\|_2^2 \\ &\leq f(x) - \frac{t^2}{2} \|\nabla f(x)\|_2^2. \quad \text{--- (1)} \end{aligned}$$

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

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

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

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

Substitute this into (1):

$$\begin{aligned} \Rightarrow f(x^+) &\leq f(x) - \frac{t^2}{2} \|\nabla f(x)\|_2^2 \\ &\leq f(x^*) + \nabla f(x)^T(x - x^*) - \frac{t^2}{2} \|\nabla f(x)\|_2^2 \\ \Rightarrow f(x^+) - f(x^*) &\leq \frac{1}{2t} \left[2t \nabla f(x)^T(x - x^*) - t^2 \|\nabla f(x)\|_2^2 \right] \\ &= \frac{1}{2t} \left[2t \nabla f(x)^T(x - x^*) - t^2 \|\nabla f(x)\|_2^2 \right. \\ &\quad \left. - \|x - x^*\|_2^2 + \|x - x^*\|_2^2 \right] \\ &= \frac{1}{2t} \left[\|x - x^*\|_2^2 - \|x - t \nabla f(x) - x^*\|_2^2 \right] \\ &= \frac{1}{2t} \left[\|x - x^*\|_2^2 - \|x^+ - x\|_2^2 \right]. \end{aligned}$$

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

$$\begin{aligned} f(x^{(i)}) - f(x^{(i-1)}) &\leq \frac{1}{2t} [\|x^{(i-1)} - x^*\|_2^2 - \|x^{(i)} - x^*\|_2^2] \\ \sum_{i=1}^k (f(x^{(i)}) - f(x^*)) &\leq \sum_{i=1}^k \frac{1}{2t} [\|x^{(i-1)} - x^*\|_2^2 - \|x^{(i)} - x^*\|_2^2] \\ &= \frac{1}{2t} [\|x^{(0)} - x^*\|_2^2 - \|x^{(k)} - x^*\|_2^2] \\ &\leq \frac{1}{2t} \|x^{(0)} - x^*\|_2^2, \end{aligned}$$

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 $\|f(x) - f(x')\|_2^2 \leq m\|x - x'\|_2^2$ is convex.

CONVERGENCE ANALYSIS FOR STRONG CONVEXITY

Θ_1 Let f be m-strongly convex & L-smooth for $L, m \in \mathbb{R}$.

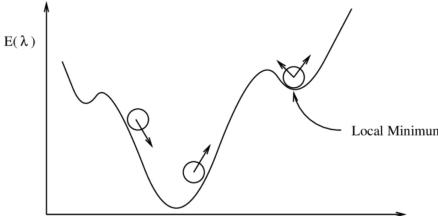
Then gradient descent with fixed step size $t \leq \frac{2}{m+L}$ satisfies

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

Θ_2 In particular, the convergence rate is $O(\gamma^k)$, which is exponentially fast.

GRADIENT DESCENT FOR NON-CONVEX CASE

Θ_1 For non-convex functions, there may exist local minimums that are not global minimums.



Θ_2 So, we cannot guarantee optimality, and so we will focus on $\|\nabla f(x)\|_2 \leq \epsilon$.

CONVERGENCE ANALYSIS FOR NON-CONVEX CASE

Θ_1 Let f be differentiable & L-lipschitz continuous.

Then gradient descent with fixed step size $t \leq \frac{1}{L}$ satisfies

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

Θ_2 In other words, the convergence rate is $O(\frac{1}{\sqrt{k}})$, which is optimal for deterministic algorithms.

STOCHASTIC GRADIENT DESCENT

Θ_1 For decomposable optimization, gradient descent involves

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

where n is large, & t is fixed.

Θ_2 Idea: In SGD, our step becomes

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

Θ_3 The convergence rate is $O(\frac{1}{\sqrt{n}})$.

Θ_4 Since randomness leads to a large variance of the estimation of gradient, SGD requires more iterations, although each iteration requires less computations.

Chapter 8: Multilayer Perceptron

MOTIVATION

Q We showed no linear classifier can separate the XOR dataset.

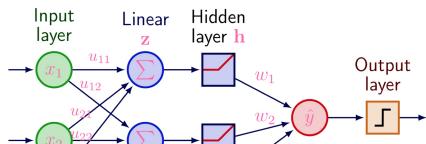
Fixes:

- ① Use a quadratic classifier;
- ② Fix the classifier but use a richer input representation.

MULTI-LAYER PERCEPTRON / MLP

Q 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

$$x_i = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, y = - \Rightarrow z_i = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$$

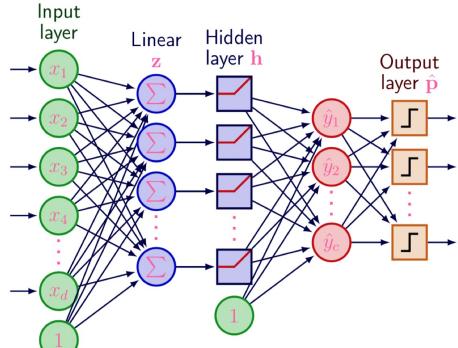
$$\Rightarrow h_i = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

$$\Rightarrow \hat{y} = \langle h, w \rangle - 1$$

$$= -1. (\because \text{sign}(\hat{y}) = \text{sign}(y))$$

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

MULTI-CLASS CLASSIFICATION



Idea:

$$z = Ux + c \quad \} \text{ learning feature } h$$

$$h = \sigma(z)$$

$$\hat{y} = Wh + b \quad } \text{ learning linear classifier}$$

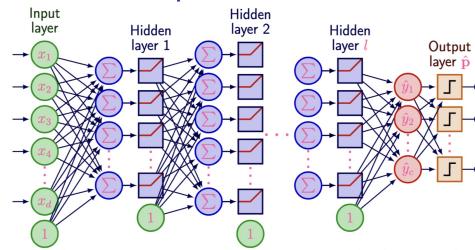
$$\hat{p} = \text{softmax}(\hat{y}) \quad } \text{ by logistic regression}$$

ACTIVATION FUNCTIONS

Choices for activation function:

- ① $\text{sgm}(t) = \frac{1}{1 + \exp(-t)}$
- ② $\tanh(t) = 1 - 2\text{sgm}(t)$
- ③ $\text{relu}(t) = t^+$
- ④ $\text{elu}(t) = (t)^+ + (t)^-(\exp(t) - 1)$

MULTI-LAYER NN



B1 We need a loss l to measure difference between our prediction \hat{y} & truth y .

B2 We also need a training set $D = \{(x_i, y_i)\}$ to train the weights w .

SGD FOR MLP

B3 To train w , we can use gradient descent:

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

$$[l_{\text{of}}](x_i, y_i; w) = l[f(x_i; w), y_i]$$

B4 We can also just use a random minibatch $B \subseteq \{1, \dots, n\}$:

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

↳ tradeoff between variance & computation.

B5 We can also use a decaying learning rate:

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

COMPUTING THE GRADIENT OF A 2-LAYER NN

B6 Model:

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

B7 We want to learn the parameters W, b_1, U & b_2 .

B8 The gradient of the network is defined by

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

B9 Next, since $\text{relu}(x) = \max(x, 0)$, it follows that

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

B10 We will show that

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

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.

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

Thus

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

Next

$$\frac{\partial J}{\partial b_1} = \frac{\partial J}{\partial \theta} \cdot \frac{\partial \theta}{\partial b_1} = U^T(\theta - y).$$

Thus

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

and so

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

lastly,

$$\begin{aligned} \frac{\partial J}{\partial b_1} &= \frac{\partial J}{\partial z} \cdot \frac{\partial z}{\partial b_1} = U^T(\theta - y) \odot \text{relu}'(z) \cdot 1 \\ &= U^T(\theta - y) \odot \text{relu}'(z) \end{aligned}$$

and we're done!

UNIVERSAL APPROXIMATION THEOREM

Q₁ For any continuous function $f: \mathbb{R}^d \rightarrow \mathbb{R}^c$ and any $\epsilon > 0$, 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

$$\sup_x \|f(x) - g(x)\|_2 < \epsilon,$$

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

i.e. $\|f(x) - g(x)\|_2 < \epsilon \quad \forall x$, s.t. $g(x)$ is at least " ϵ -close" to $f(x)$.

Q₂ This implies that as long as a 2-layer MLP is "wide enough" (i.e. a large k), it can approximate any continuous function arbitrarily closely.

WHY DEEP LEARNING?

Q₁ 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.

Q₂ In particular, deep NNs are more parameter efficient.

DROPOUT

Q₁ Idea: For each training minibatch, keep each hidden unit with probability q .

Q₂ Essentially, there is a different & random network for each training minibatch.

Q₃ In particular, hidden units are less likely to collude to overfit training data.

Q₄ For testing, we use the full network.

BATCH NORMALIZATION

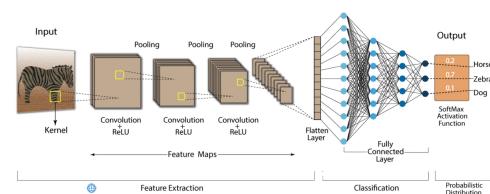
Q₁ Idea: Normalize the input over the minibatch dimensions.

Chapter 9: Convolutional Neural Networks

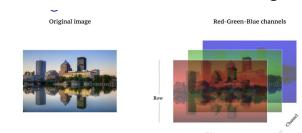
MOTIVATION

- In MLPs, it is easy to overfit training data.
- 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 a grayscale 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

$$\begin{array}{|c|c|c|c|} \hline 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 \\ \hline \end{array} \quad \begin{array}{|c|c|c|} \hline 1 & 0 & 1 \\ \hline 0 & 1 & 0 \\ \hline 1 & 0 & 1 \\ \hline \end{array} \quad \begin{array}{c} 4 \ 3 \ 4 \\ \rightarrow \ 2 \ 4 \ 3 \\ 2 \ 3 \ 4 \end{array}$$

Convolutional 3x3 filter

- recall: $\langle A, B \rangle = \sum_{ij} A_{ij} B_{ij} \in \mathbb{R}$

- 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

Kernel Channel #1 Kernel Channel #2 Kernel Channel #3

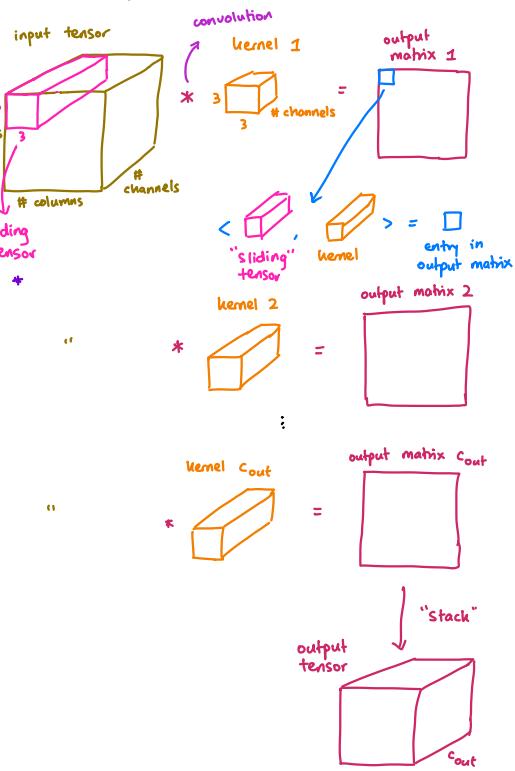
$\begin{bmatrix} -1 & -1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & -1 & 1 \end{bmatrix}$
308	-498	164
$+ \quad \quad \quad + \quad \quad \quad + \quad \quad \quad +$		
		-25

Bias = 1

$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}$
$\begin{bmatrix} 0 & 156 & 155 & 156 & 158 & 149 & \dots \end{bmatrix}$	$\begin{bmatrix} -147 & 166 & 167 & 169 & 169 & \dots \end{bmatrix}$	$\begin{bmatrix} 162 & 163 & 164 & 165 & 165 & \dots \end{bmatrix}$
$\begin{bmatrix} 0 & 155 & 154 & 157 & 159 & 159 & \dots \end{bmatrix}$	$\begin{bmatrix} 164 & 165 & 166 & 170 & 170 & \dots \end{bmatrix}$	$\begin{bmatrix} 160 & 161 & 162 & 169 & 170 & \dots \end{bmatrix}$
$\begin{bmatrix} 0 & 149 & 151 & 153 & 155 & 155 & \dots \end{bmatrix}$	$\begin{bmatrix} 160 & 162 & 166 & 169 & 170 & \dots \end{bmatrix}$	$\begin{bmatrix} 158 & 159 & 162 & 165 & 166 & \dots \end{bmatrix}$
$\begin{bmatrix} 0 & 146 & 146 & 147 & 153 & 155 & \dots \end{bmatrix}$	$\begin{bmatrix} 156 & 157 & 159 & 163 & 163 & \dots \end{bmatrix}$	$\begin{bmatrix} 155 & 155 & 158 & 161 & 167 & \dots \end{bmatrix}$
$\begin{bmatrix} 0 & 145 & 143 & 143 & 150 & 150 & \dots \end{bmatrix}$	$\begin{bmatrix} 155 & 153 & 153 & 158 & 165 & \dots \end{bmatrix}$	$\begin{bmatrix} 154 & 152 & 152 & 157 & 157 & \dots \end{bmatrix}$
Input Channel #1 (Red)	Input Channel #2 (Green)	Input Channel #3 (Blue)
$\begin{bmatrix} -1 & 1 & \\ 0 & 1 & \\ 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & \\ 1 & -1 & \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & \\ 0 & 1 & 0 \\ 1 & -1 & \end{bmatrix}$
Kernel Channel #1	Kernel Channel #2	Kernel Channel #3
\downarrow	\downarrow	\downarrow
148	+	-8
		+
		646
		+ 1 = 787

$$\begin{array}{r} \downarrow \\ 148 \end{array}
 + \quad
 \begin{array}{r} \downarrow \\ -8 \end{array}
 + \quad
 \begin{array}{r} \downarrow \\ 646 \end{array}
 + \begin{array}{l} 1 = 787 \\ \hline \end{array}
 \begin{array}{r} \downarrow \\ \text{Bias} = 1 \end{array}$$

 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 $3 \times 3, 5 \times 5$

- by default, # of channels on each filter is the same as input

② Number of kernels;

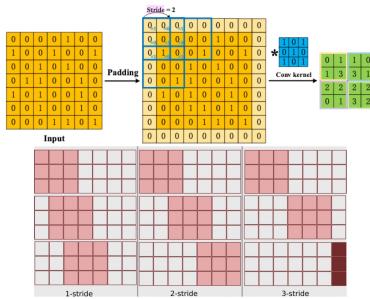
③ "Stride" - how many pixels to move the filter each time; &

- larger stride \Rightarrow neighboring outputs less similar

④ "Padding" - add zeroes around input boundary.

- keeps boundary information lossless

PADDING & STRIDE



SIZE CALCULATION

💡 Sizes:

① Input: $m \times n \times c$

② Filter: $a \times b \times c$

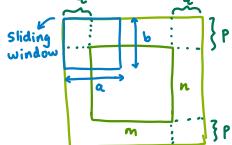
③ Stride: $s \times t$

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

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

WEIGHT SHARING: CNN=MLP

💡 Let our kernel be $W = \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix} \in \mathbb{R}^{2 \times 2}$ & our input matrix be $X = \begin{pmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \end{pmatrix} \in \mathbb{R}^{3 \times 3}$. We can define

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

💡 Then note

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

💡 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

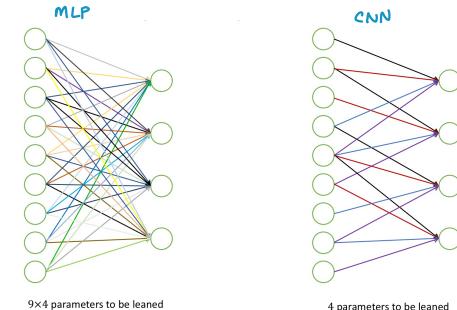
$$W_{\text{circ}} = \begin{pmatrix} w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 & 0 \\ 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 \\ 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 \\ 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 \\ 0 & 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} \end{pmatrix} \in \mathbb{R}^{4 \times 9}$$

💡 See that

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

💡 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.



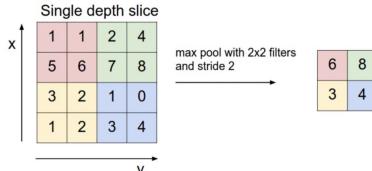
💡 Hence, we can train a CNN faster than a MLP, since there are less parameters to be learnt.

POOLING

Q₁: Idea: "Pooling" down-samples the input size to reduce memory & computation.

Q₂: 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.

Q₃: We also have a notion of size/stride.



Q₄: Note that pooling by default is performed on each slice separately, so the number of channels is the same between the input & output.

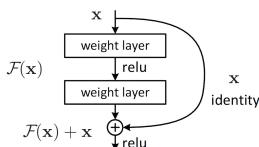
Q₅: If we set the kernel size = input size, this is known as "global pooling".

DEEPER MODELS

Q₁: Note deeper models (ie more layers) are better but are more difficult to train.

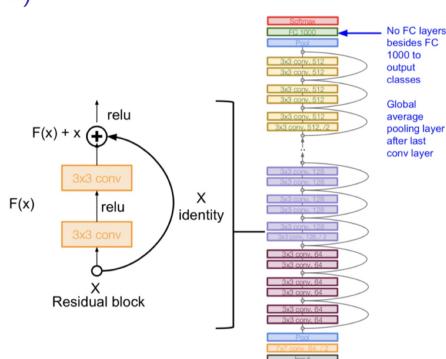
RESIDUAL BLOCK

Q₁: Idea: Add a shortcut connection that allows "skipping" one or more layers.



Q₂: This allows more direct backpropagation of the gradient via the shortcut.

Q₃: 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

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

💡 We want to find

$$\underset{y}{\operatorname{argmax}} P(y_1, \dots, y_m | x_1, \dots, x_n)$$

AUTO-REGRESSIVE / GREEDY METHOD

💡 Idea: we repeatedly compute

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

eg

Step 0 X : Where is University of Waterloo?

Step 1 Y : [START]; $\Pr(\text{It} | X \text{ [START]})$ highest

Step 2 Y : [START] It; $\Pr(\text{is} | X \text{ [START] It})$ highest

Step 3 Y : [START] It is; $\Pr(\text{at} | X \text{ [START] It is})$ highest

Step 4 Y : [START] It is at; $\Pr(\text{Waterloo} | X \text{ [START] It is at})$ highest

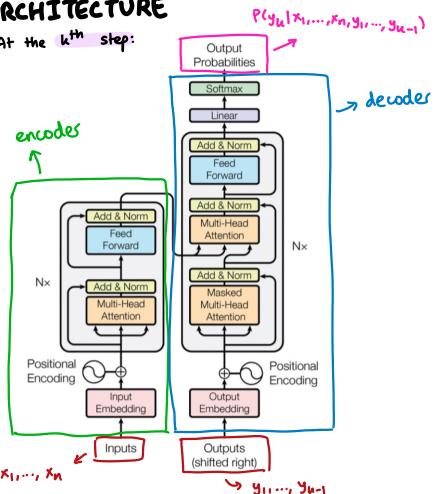
Step 5 Y : [START] It is at Waterloo; $\Pr(\text{[END]} | X \text{ [START] It is at Waterloo})$ highest

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

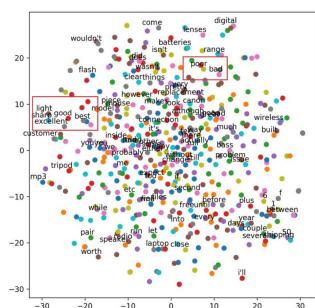
💡 The "tokenizer" divides the input sentence into the individual tokens/words.

TOKEN EMBEDDING

💡 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.

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



POSITIONAL ENCODING

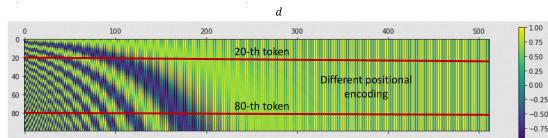
- Θ_1 Idea: the order of tokens in the sentence changes its meaning.
- Θ_2 We use a positional encoding matrix $W \in \mathbb{R}^{n \times d}$:

$$W_{t,2i}^P = \sin\left(\frac{t}{10000^{\frac{2i}{d}}}\right), \quad W_{t,2i+1}^P = \cos\left(\frac{t}{10000^{\frac{2i+1}{d}}}\right),$$

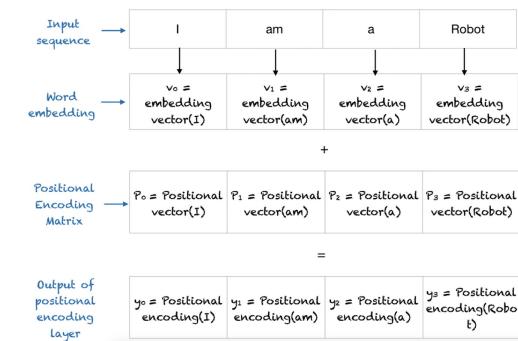
$$i = 0, \dots, \frac{d}{2}-1$$

\hookrightarrow no parameter to be learnt!

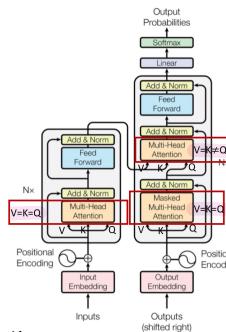
- Θ_3 We then just add W^P to the $n \times d$ token embedding.



- Θ_4 Putting it together:



ATTENTION LAYER

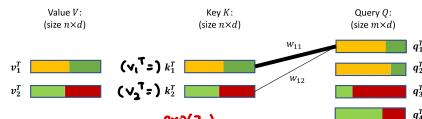


- Θ_1 Inputs:

- ① value $V \in \mathbb{R}^{n \times d}$
- ② key $K \in \mathbb{R}^{n \times d}$
- ③ query $Q \in \mathbb{R}^{n \times d}$

- Θ_2 Output: $\mathbb{R}^{m \times d}$ (m row vectors of dimension d).

- Θ_3 Idea:



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

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

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

- note v_i^T contributes more to the output row.

- this is just a weighted average.

Similarly, the

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



MATRIX FORM OF ATTENTION

Θ_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

Attention(V, K, Q)

$$= \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

$$= \left(\begin{array}{c} \text{softmax}\left(\frac{Q_1K_1^T}{\sqrt{d}}\right)v_1^T + \dots + \text{softmax}\left(\frac{Q_1K_n^T}{\sqrt{d}}\right)v_n^T \\ \vdots \\ \text{softmax}\left(\frac{Q_mK_1^T}{\sqrt{d}}\right)v_1^T + \dots + \text{softmax}\left(\frac{Q_mK_n^T}{\sqrt{d}}\right)v_n^T \end{array} \right) \in \mathbb{R}^{m \times d}$$

- softmax is a "row-wise" operation

Θ_2 There is no learnable parameters so far!

LEARNABLE ATTENTION LAYER & MULTI-HEAD ATTENTION

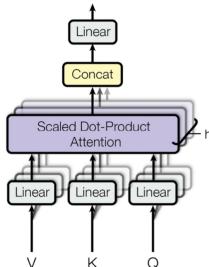
Θ_1 Idea: Replace $Q \rightarrow QW^Q, K \rightarrow KW^K, V \rightarrow VW^V$, where $\{W^Q, W^K, W^V\} \in \mathbb{R}^{d \times 64}$ are learnable linear layers.

Θ_2 Then our attention layer becomes

$$\text{Attention}(VW^V, KW^K, QW^Q)$$

$$= \text{softmax}\left(\frac{QW^Q(KW^K)^T}{\sqrt{d}}\right)VW^V$$

Multi-Head Attention



Θ_3 We can add $n=8$ linear layers in parallel & concatenate their output later.

- output dimension = $64 \times 8 = 512$

MASKED MULTI-HEAD ATTENTION

Θ_1 Idea: We mask future words, and input the masked sequence into the attention layer.

FEED-FORWARD LAYER

Θ_1 This is just a 2-layer MLP with ReLU activation:

$$\text{MLP}(x) = \max(0, x^T w_1 + b_1^T) \cdot w_2 + b_2^T$$

Θ_2 We use layer normalization instead of batch normalization.

- Since batch size is often small

OVERVIEW

Θ_1 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

Θ_1 We train the transformer by finding

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

where

- ① $Y = (y_1, \dots, y_L)$ is our output sequence; &
- this is one-hot (ie 0 or 1)
- ② $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_L)$ is the predicted probabilities.

Chapter III:

Large Language Models

COMPUTATIONAL COMPLEXITY

Θ_1 : Self-attention: $O(n^3d + nd^2)$ per layer

$$Q \in \mathbb{R}^{nxd}, K \in \mathbb{R}^{ndn}$$

\Rightarrow computing QK^T takes $O(n^3d)$ time.

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

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

Θ_2 : Feed-forward: $O(d^3)$ per layer

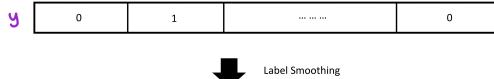
LABEL SMOOTHING

Θ_1 : Idea: Replace the label y distribution

$$p(k|x) = \delta_{k,y} \text{ with}$$

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

where C is the # of classes.



- ε_{ls} is a hyperparameter.

BERT VS GPT

Θ_1 : BERT is solely an encoder, whereas

GPT is solely a decoder.

- BERT predicts randomly-sampled middle word

- GPT predicts the next word

PRETRAINING, FINE-TUNING, INFERENCE



- pre-training takes weeks/months

- fine-tuning takes days to weeks/months

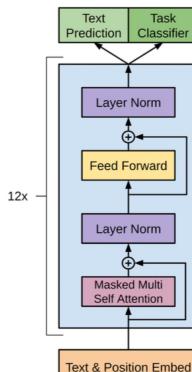
PRE-TRAINING TASKS

Θ_1 : GPT: predict masked words

Θ_2 : BERT: predict middle words given context.

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

GPT STRUCTURE



PRETRAINING

Θ_1 : Goal: we want to find

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

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

FINE-TUNING

Θ_1 : Goal: We want to find

$$\min_{\Theta} -\hat{E}[\log \prod_{j=1}^m p(y_j | x_{1:m}; \Theta)] - \lambda \hat{E}[\log \prod_{j=1}^m p(x_j | x_{1:j-1}; \Theta)]$$

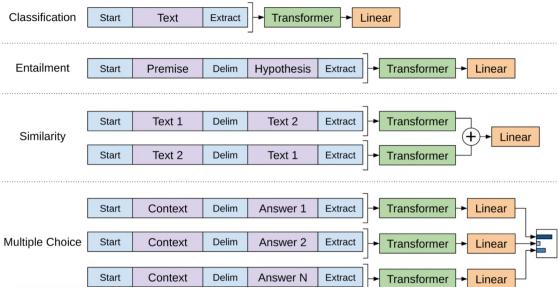
task-aware supervised loss

pretraining loss

Θ_2 : Tasks:

- ① "classification" - classify text into a class
- ② "Entailment" - determine 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



SENTENCE-BERT

- 💡 **Idea:** Use a twin network to save the representations for future use.
 This drastically reduces the # of times we do inference & the computation time.

GPT-2

- 💡 **GPT-2** uses the same training method as GPT, but introduces a new larger dataset.
 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: cheese => ...

task description

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

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: sea otter => loutre de mer

task description

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: sea otter => loutre de mer

examples

peppermint => menthe poivrée

examples

plush giraffe => girafe peluche

examples

cheese => ...

prompt

GPT-3

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



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.

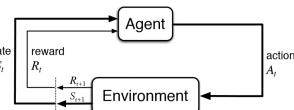
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

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?
 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. ✓

APT-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.



Step 2
Collect comparison data, and train a reward model.



A prompt and several model outputs are sampled.

A labeler demonstrates the desired output behavior.



Some people went to the moon...

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

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.

Once upon a time...

The reward is used to update the policy using PPO.

B₁ Idea: 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.

B₂ 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

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

MODEL

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

$$\min_{\theta} \text{dist}(p_{\text{data}} || p_\theta)$$

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

PUSH-FORWARD MAPS

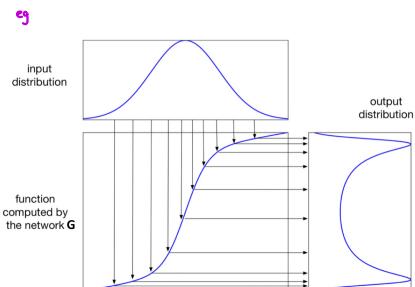
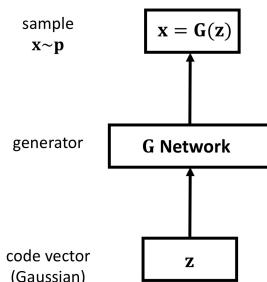
- Let r be any continuous distribution on \mathbb{R}^n . For any distribution p on \mathbb{R}^d , there exist "push-forward maps" $G: \mathbb{R}^n \rightarrow \mathbb{R}^d$ such that

$$z \sim r \Rightarrow G(z) \sim p.$$

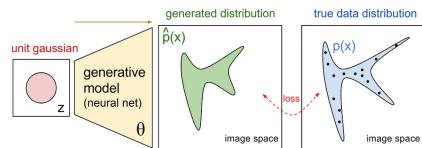
- WLOG, we can take r to be Gaussian.

GENERATING SAMPLES

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

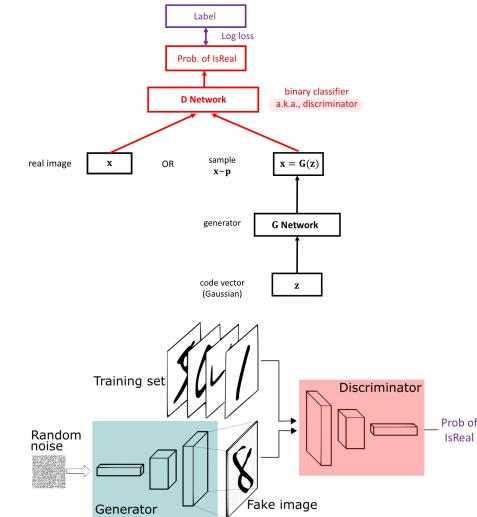


LEARNING THE G NETWORK



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 x is real).

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

In particular, we want

$$\min_D -\frac{1}{2} E_{x \sim p_{\text{data}}} [\log D(x)] - \frac{1}{2} E_{z \sim N(0, I)} [\log(1 - D(G(z)))]$$

x is real x 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

$$\max_G -\frac{1}{2} E_{x \sim p_{\text{data}}} [\log D(x)] - \frac{1}{2} E_{z \sim N(0, I)} [\log(1 - D(G(z)))]$$

x is real x is fake

PUTTING IT TOGETHER

Hence, we want to find

$$\max_G \min_D -\frac{1}{2} E_{x \sim p_{\text{data}}} [\log D(x)] - \frac{1}{2} E_{z \sim N(0, I)} [\log(1 - D(G(z)))]$$

x is real x is fake

Replacing expectation with the empirical expectation (ie average):

$$\min_G \max_D \hat{E}_{x \sim p_{\text{data}}} [\log(D(x))] + \hat{E}_{z \sim N(0, I)} [\log(1 - D(G(z)))]$$

$\nabla G, D$

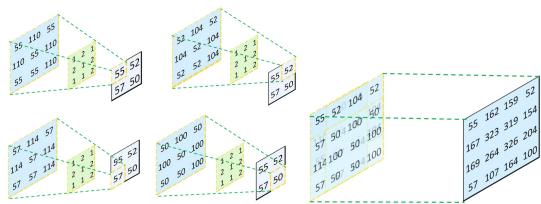
SOLVER

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.



SOLUTION OF D*

Q: Let $p_g(x)$ be the density of x estimated by the generator G . For a fixed G , the optimal discriminator is

$$D_G^*(x) = \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + p_g(x)}$$

Proof. See that

$$\begin{aligned} VCG, D) &= E_{x \sim P_{\text{data}}} [\log D(x)] + E_{z \sim N(0, I)} [\log (1 - D(G(z)))] \\ &= \int_X P_{\text{data}}(x) \log D(x) dx + \int_Z p_g(z) \log (1 - D(G(z))) dz \\ &\quad \hookrightarrow \text{let } x = G(z) \\ &= \int_X P_{\text{data}}(x) \log D(x) dx + \int_X p_g(x) \log (1 - D(x)) dx \\ &= \int_X P_{\text{data}}(x) \log D(x) + p_g(x) \log (1 - D(x)) \underbrace{\}_{f(D(x))} \end{aligned}$$

Then the optimal solution is

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

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

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

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

Thus

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

as needed. \square

SOLUTION OF G*

Q: $\min_{G} \max_{D} V(G, D)$ is achieved iff $P_g = P_{\text{data}}$.

The optimal objective value is $-\log 4$.

Thus, the GAN can learn P_{data}

exactly if we can solve $\min_{G} \max_{D} V(G, D)$ exactly.

Proof. See that

$$\begin{aligned} VCG, D_G^*) &= E_{x \sim P_{\text{data}}} [\log D_G^*(x)] + E_{z \sim N(0, I)} [\log (1 - D_G^*(G(z)))] \\ &\quad (\text{let } x = G(z)) \\ &= E_{x \sim P_{\text{data}}} [\log D_G^*(x)] + E_{x \sim p_g} [\log (1 - D_G^*(x))] \\ &= E_{x \sim P_{\text{data}}} [\log \frac{P_{\text{data}}(x)}{P_{\text{data}}(x) + p_g(x)}] + E_{x \sim p_g} [\log \frac{p_g(x)}{P_{\text{data}}(x) + p_g(x)}] \end{aligned}$$

For distributions P, Q , we define

$$KL(P || Q) = E_{x \sim P} [\log \frac{P(x)}{Q(x)}].$$

Then

$$\begin{aligned} VCG, D_G^*) &= -\log 4 + KL(P_{\text{data}} || \frac{P_{\text{data}} + p_g}{2}) \\ &\quad + KL(p_g || \frac{P_{\text{data}} + p_g}{2}) \\ &= -\log 4 + 2 JSD(P_{\text{data}} || p_g) \\ &\geq -\log 4 \end{aligned}$$

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

Equality holds iff $P_{\text{data}} = p_g$, as needed. \square

Q3

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

Chapter 13: Self-Supervised Learning

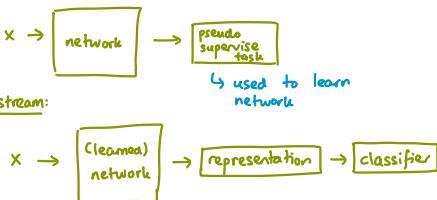
Q₁ "Self-supervised learning" is a subclass of unsupervised learning, where we want to learn useful representations through pretraining tasks for downstream tasks.

- unsupervised: learning with unlabeled data

Q₂ 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 toping layers

pretraining:



Fine-tuning: update network & classifier

Linear evaluation: fix network, update linear classifier

WHY?

Q₁ Idea: Creating labelled datasets for each task is expensive, but there is a lot of unlabelled data.

Q₂ Self-supervised learning will also not overfit.

Q₃ Challenges:

- ① Select a suitable pretraining task;
- ② No golden rule for comparison for learned feature repre