CS 480 Personal Notes

Marcus Chan

Taught by Hongyang Zhang

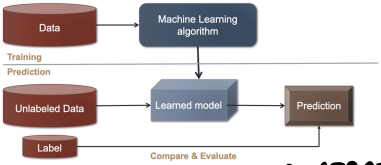
UW CS '25

Chapter 1: Perceptrons

ML

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

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	
	x_1	x_2	x_3	x_4	$\cdots$	x_n	x_1'	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

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

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

LINEAR FUNCTION

We say a function f is "linear" if

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

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

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

($\Leftarrow$) Note

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

∎

AFFINE FUNCTION

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: $\hat{y}$

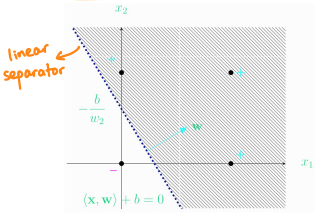
Given $w \in \mathbb{R}^d$, $b \in \mathbb{R}$, define the "score" at some $x \in \mathbb{R}^d$ to be

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

Our "prediction" for y is then

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

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



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

PERCEPTRONS

Algorithm for training:

Algorithm 1 Training Perceptron

Input: Dataset $= (\mathbf{x}_i, \mathrm{y}_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n$, 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 $t = 1, 2, \ldots$ **do**
  receive index $I_t \in \{1, \ldots, n\}$ // 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
    $\mathbf{w} \leftarrow \mathbf{w} + \mathrm{y}_{I_t}\mathbf{x}_{I_t}$
    $b \leftarrow b + \mathrm{y}_{I_t}$ // update after a "mistake"
  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 $i = 1, \ldots, n$,

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

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.

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

A TRICK TO HIDE THE BIAS TERM

Note that

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

(x_{pad}, w_{pad})

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

Thus, our new update rule is

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

CONVERGENCE THEOREM (LINEARLY SEPARABLE CASE)

Suppose there exists a w^* such that

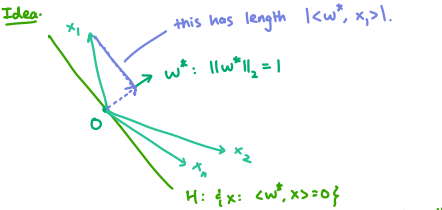
$$y_i \langle x_i, w^* \rangle > 0 \quad \forall i = 1, \dots, n.$$

Assume $\|x_i\|_2 \leq C \ \forall i$ 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:



- 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 $\langle w, w^* \rangle$:

$$\langle w, w^* \rangle \longrightarrow \langle w + yx, w^* \rangle$$

$$= \langle w, w^* \rangle + \underbrace{y \langle x, w^* \rangle}_{\text{positive } \because w^* \text{ is perfect}}$$

$$= \langle w, w^* \rangle + |\langle x, w^* \rangle|$$

$$\geq \langle w, w^* \rangle + \gamma.$$

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

Similarly, consider an updates effect on $\|w\|_2^2$:

$$\|w\|_2^2 = \langle w, w \rangle \longrightarrow \langle w + yx, w + yx \rangle$$

$$= \langle w, w \rangle + \underbrace{2y \langle x, x \rangle}_{< 0} + y^2 \langle x, x \rangle$$

$$= \langle w, w \rangle + 2y \langle w, x \rangle + \underbrace{\|x\|_2^2}_{\leq C^2}$$

$$\leq \langle w, w \rangle + C^2.$$

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:

$$\langle w_M, w^* \rangle \geq \langle w_{M-1}, w^* \rangle + \gamma$$

$$\geq \langle w_{M-2}, w^* \rangle + 2\gamma$$

$$\geq \dots \geq \underbrace{\langle w_0, w^* \rangle}_{=0} + M\gamma$$

$$= M\gamma.$$

Similarly, note

$$\langle w_M, w_M \rangle \leq \langle w_{M-1}, w_{M-1} \rangle + C^2$$

$$\leq \dots \leq \underbrace{\langle w_0, w_0 \rangle}_{=0} + MC^2$$

$$\leq MC^2.$$

Since

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

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. ☑

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

ANOTHER PERSPECTIVE ON PERCEPTRONS

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

2. We can define our "loss function" as

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

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

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

4. Our gradient descent update:

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

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

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

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

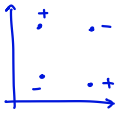
We want to choose w such that

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

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:

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

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

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

$\Big\}$ $w_1 + w_2 + 2b > 0$

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

Hence

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

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

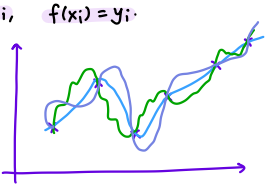
HARDNESS RESULT (NON-LINEARLY SEPARABLE CASE)

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

Chapter 2: Linear Regression

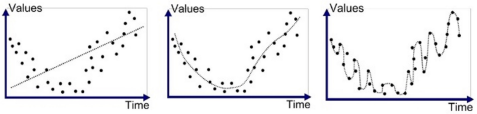
1. Idea: Given training data (x_i, y_i), find a $f: \mathcal{X} \to \mathcal{Y}$ such that $f(x_i) \approx y_i$, where
   - ① $x_i \in \mathcal{X} \subseteq \mathbb{R}^d$: the feature vector for the i^{th} training example
   - ② $y_i \in \mathcal{Y} \subseteq \mathbb{R}^t$: t responses
     - note we could have $t=1$ or even $t=\infty$
2. 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



3. Moreover, our prediction $\hat{y} = f(x)$ can vary significantly on new data x!
4. 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

$$(x_i, y_i) \sim \mathcal{P}$$
$$(X, Y) \sim \mathcal{P}.$$

LEAST SQUARES REGRESSION

We want to choose f so that

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

this is our least squared error.

REGRESSION FUNCTION: $m(x)$

1. Our "regression function" is

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

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

3. We show that m is optimal; ie

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

Proof. First, see that

$$\begin{aligned} E\|f(X) - Y\|_2^2 &= E\|f(X) - m(X) + m(X) - Y\|_2^2 \\ &= E\|f(X) - m(X)\|_2^2 + E\|m(X) - Y\|_2^2 \\ &\quad + 2E\langle f(X) - m(X),\ m(X) - Y\rangle. \end{aligned}$$

(using $\|a+b\|_2^2 = \|a\|_2^2 + \|b\|_2^2 + 2\langle a, b\rangle$)

Then

$$\begin{aligned} &E_{X,Y}\left[\langle f(X) - m(X),\ m(X) - Y\rangle\right] \\ &= E_X\left[E_{Y|X}\left[\langle f(X) - m(X),\ m(X) - Y\rangle\right]\right] \\ &\quad \text{(by double expectation theorem, see STAT 330)} \\ &= E_X\left[\langle f(X) - m(X),\ m(X) - \underbrace{E[Y \mid X]}_{m(X)}\rangle\right] \\ &= E_X[\langle f(X) - m(X),\ 0\rangle] \\ &= 0. \end{aligned}$$

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

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

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:

$$\left.\begin{aligned} D_1 \sim P &\to f_{D_1} \\ &\vdots \\ D_n \sim P &\to f_{D_n} \end{aligned}\right\} \text{ then we define } \bar{f}(x) = \text{avg } f_{D_i}(x).$$

Then

$$\begin{aligned} &E_D E_X\|f_D(X) - m(X)\|_2^2 \\ &= E_{D,X}\|f_D(X) - \bar{f}(X) + \bar{f}(X) - m(X)\|_2^2 \\ &= E_{D,X}\|f_D(X) - \bar{f}(X)\|_2^2 + E_{D,X}\|\bar{f}(X) - m(X)\|_2^2 \\ &\quad + 2E_{D,X}\langle f_D(X) - \bar{f}(X),\ \bar{f}(X) - m(X)\rangle. \end{aligned}$$

Similarly, see that

$$\begin{aligned} &E_{D,X}\langle \bar{f}(X) - f_D(X),\ m(X) - \bar{f}(X)\rangle \\ &= E_X E_D\langle \underbrace{m(X) - \bar{f}(X)}_{\text{constant wrt } D},\ \bar{f}(X) - f_D(X)\rangle \\ &= E_X\langle m(X) - \bar{f}(X),\ \bar{f}(X) - \underbrace{E_D[f_D(X)]}_{\bar{f}(X)}\rangle \\ &= 0. \end{aligned}$$

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

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

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

LINEAR REGRESSION

1. 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

2. Again, we can use padding:

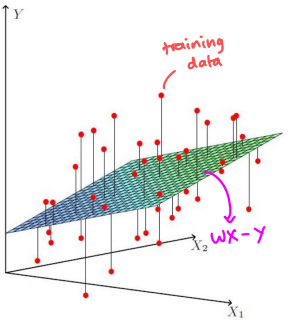
$$x\in\begin{pmatrix}x\\1\end{pmatrix}, \quad W\in[W,b] \Rightarrow f(x)=Wx$$

3. In matrix form:

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

4. 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

1. We define our loss function as

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

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

$$\nabla_W \text{Loss}(W) = \frac{2}{n}(WX-Y)X^T \quad (=0)$$
$$\Rightarrow WXX^T = YX^T$$
$$\Rightarrow W = YX^T(XX^T)^{-1}$$

PREDICTION

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

$$\hat{Y}_{test} = WX_{test}$$

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

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

3. The "training error" is

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

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

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

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

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

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

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

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

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

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.

Alternatively, note

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

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"

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

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

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

3. **Maximizing the likelihood:**

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

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

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

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

then

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

↳ this is also called the "sigmoid transformation".

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

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

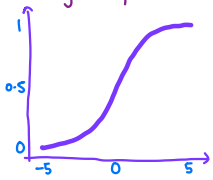
if $y_i \in \{0,1\}$.

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

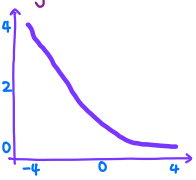
$$\min_w \sum_{i=1}^{n} \log[1+\exp(-y_i \langle x_i, w\rangle)]$$

↳ this is "logistic loss".

Sigmoid function



logistic loss



TRAINING LOGISTIC REGRESSION

Our gradient descent algorithm is

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

PREDICTION

We take

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

Our decision boundary is still

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

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

MULTI-CLASS EXTENSION

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

We consider the "softmax" function:

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

- non-negative and sum to 1

Logits	$\langle x, w_1 \rangle$	$\langle x, w_2 \rangle$	... ...	$\langle x, w_c \rangle$

Softmax operation

Probability (confidence)	$\frac{\exp(\langle x, w_1 \rangle)}{\sum_i \exp(\langle x, w_i \rangle)}$	$\frac{\exp(\langle x, w_2 \rangle)}{\sum_i \exp(\langle x, w_i \rangle)}$	... ...	$\frac{\exp(\langle x, w_c \rangle)}{\sum_i \exp(\langle x, w_i \rangle)}$

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

Training: again, we use MLE:

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

Prediction:

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

Chapter 4: Hard-Margin Support Vector Machines

INTRODUCTION

1. We assume $y=\{-1,+1\}$, and don't use padding.
2. Perceptron: we find any $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ such that

$$\min_{w,b} 0 \quad \text{s.t.} \quad y_i \hat{y}_i > 0 \ \forall i, \qquad \hat{y}_i = \langle x_i, w \rangle + b$$

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

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 (assuming data is linearly separable).

DISTANCE FROM A POINT TO A HYPERPLANE

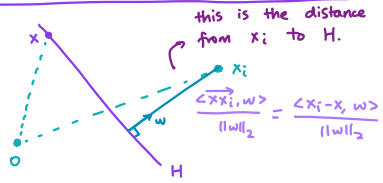
Let $H := \{x : \langle x, w \rangle + b = 0\}$. Then

$$\text{distance}(x_i, H) = \frac{|\langle x_i - x, w \rangle|}{\|w\|_2}, \quad x \in H$$

$$= \frac{|\langle x_i, w \rangle - \langle x, w \rangle|}{\|w\|_2}$$

$$= \frac{|\langle x_i, w \rangle + b|}{\|w\|_2} \quad \because x \in H$$

$$= \frac{y_i \hat{y}_i}{\|w\|_2} \quad \because y_i \hat{y}_i > 0$$

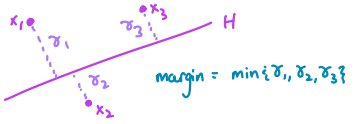


MARGIN

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

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

eg



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

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

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

$_1$ **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

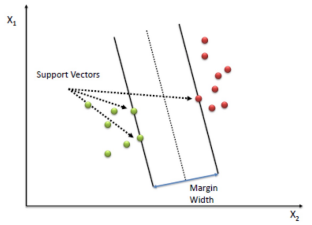
$$y_i\hat{y}_i \geq 1 \quad \forall i \iff \begin{cases} \hat{y}_i \geq +1 & \forall i: y_i = +1 \\ \hat{y}_i \leq -1 & \forall i: y_i = -1 \end{cases}$$

$_2$ This yields 3 parallel hyperplanes:

$$H = \{x: \langle x, w\rangle + b = 0\}$$
$$H^+ = \{x: \langle x, w\rangle + b = +1\}$$
$$H^- = \{x: \langle x, w\rangle + b = -1\}$$

} the "supporting" hyperplanes

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



LAGRANGIAN DUAL

First, we show

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

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

$\alpha = [\alpha_1, \ldots, \alpha_n] \in \mathbb{R}^n$;
$\alpha \geq 0 \iff \alpha_i \geq 0 \ \forall i$

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

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

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

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. ∎

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

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

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

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

Finally, we consider the "outer" maximization problem.

Plugging in our value of w above:

$$\Rightarrow \text{Loss}(\alpha) = \frac{1}{2}\left\|\sum_i \alpha_i y_i x_i\right\|_2^2 - \left\langle \sum_i \alpha_i y_i x_i, \sum_i \alpha_i y_i x_i \right\rangle - b \underbrace{\sum_i \alpha_i y_i}_{0} + \sum_i \alpha_i$$

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

Thus, our problem becomes

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

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

WHY USE THE DUAL FORM?

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

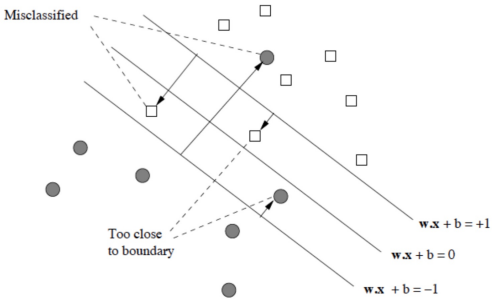
$$\min_{\alpha \geq 0} -\sum_i \alpha_i + \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle$$

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

Chapter 5: Soft-Margin Support Vector Machines

MOTIVATION

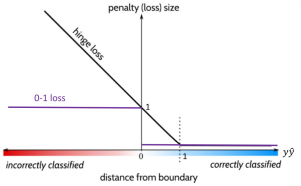
1. Hard-margin SVMs assume the data is linearly separable, but this is not always the case.
2. We want to adapt this to work for non-linearly separable data.
3. To do this, we will penalize our loss if the data falls too close to the boundary, or if the data is misclassified.



THE HINGE LOSS

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

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



* 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

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

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

we penalize error & small margin

SOFT VS HARD-MARGIN SVM

1. For hard-margin SVM, we have a hard constraint that $y_i(\langle x_i,w\rangle+b) \geq 1 \quad \forall i$.
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

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

true label; predicted label

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

2. This is equivalent to

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

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

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

Note that

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

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,

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

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

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

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 $t_i = 1 - y_i(\langle x_i, w\rangle + b)$ to get

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

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

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

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

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:

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

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

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

Thus, the dual form is

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

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

Note that if

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

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

COMPLEMENTARITY SICKNESS

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

Then note that

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

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

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

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

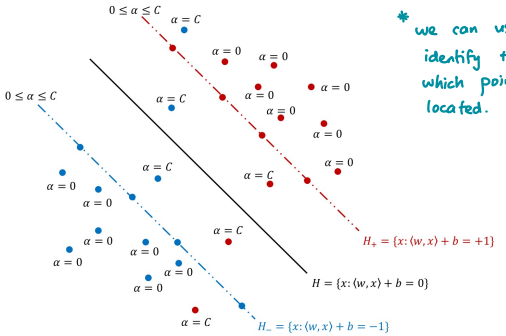
(ie margin / wrong idea)

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

(ie correctly classified with good confidence)

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

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



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

RECOVERING w & b FROM DUAL

We can obtain w & b via

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

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.

Then we can recover b via

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

Since $y = \pm 1$.

We can then predict new data via

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

Chapter 6: Reproducing Kernels

MOTIVATION

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

QUADRATIC CLASSIFIER

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.

We can then predict via

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

THE POWER OF LIFTING

We can express

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

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

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

$$\overrightarrow{A} = \begin{pmatrix} a_{11} \\ \vdots \\ a_{1d} \\ a_{21} \\ \vdots \\ a_{dd} \end{pmatrix} \in \mathbb{R}^{d \times d}$$

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

THE KERNEL TRICK

The feature map ϕ blows up the dimension.

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

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

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

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

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

Note that choosing ϕ uniquely determines k.

MERCER'S THEOREM

$_1$ $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ 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.

$_2$ 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

$_1$ 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;

$_2$ If (k_i) is a sequence of kernels, then their limit k, if it exists, is also a kernel.

KERNEL SVM

$_1$ 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

$$\min_{0 \leq \alpha \leq C} -\sum \alpha_i + \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

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

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

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

PREDICTION

$_1$ Suppose that $0 \leq \alpha^* \leq C$ optimizes the kernel SVM.

$_2$ Then, we can recover

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

$_3$ 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

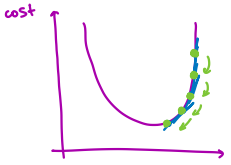
Many ML methods can be classed as optimization problems; ie

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

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

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

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



- 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

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

Stochastic gradient descent update:

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

I is random

EXAMPLE: SOFT-MARGIN SVM

Gradient descent update for soft-margin SVM:

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

$$b \leftarrow b - t\left[\frac{1}{n}\sum_{i=1}^{n} \ell'_{\text{hinge}}(y_i \hat{y}_i)\, y_i\right]$$

INTERPRETATION FROM TAYLOR EXPANSION

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

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

Hence

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

Then see that

$$\frac{\partial L(y)}{\partial y} = 0 + \nabla f(x) + \frac{1}{t}(y-x) \quad (=0)$$

$$\Rightarrow y = x - t \cdot \nabla f(x)$$

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 $x, y \in \mathbb{R}^n$,

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

L-LIPSCHITZ CONTINUOUS

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 $x \in \text{dom}(f)$, where $L \in \mathbb{R}$.

Here,

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

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

CONVERGENCE ANALYSIS FOR CONVEX CASE

💡₁ 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

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

💡₂ We say gradient descent has convergence rate $O(\frac{1}{k})$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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,

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

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

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

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 $L, m \in \mathbb{R}$.

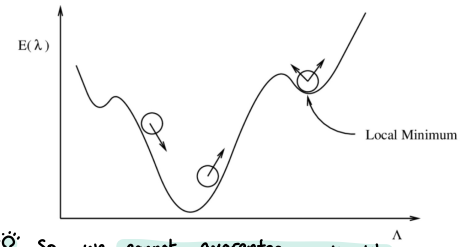
Then gradient descent with fixed step size $t \leq \frac{2}{m+L}$ satisfies

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

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 $\|\nabla f(x)\|_2 \leq \varepsilon$.

CONVERGENCE ANALYSIS FOR NON-CONVEX CASE

Let f be differentiable & L-lipschitz continuous.

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

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

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

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

where n is large, & t is fixed.

Idea: In SGD, our step becomes

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

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

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 Percepton

MOTIVATION

h_1 We showed no linear classifier can separate the XOR dataset.

h_2 Fixes:

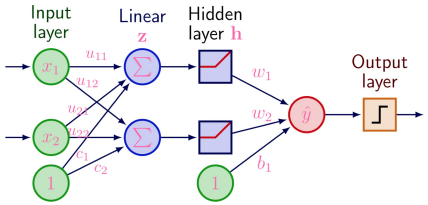
① Use a quadratic classifier;

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

$x_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $y = -$ $\Rightarrow z_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \end{pmatrix}$

$= \begin{pmatrix} 0 \\ -1 \end{pmatrix}$.

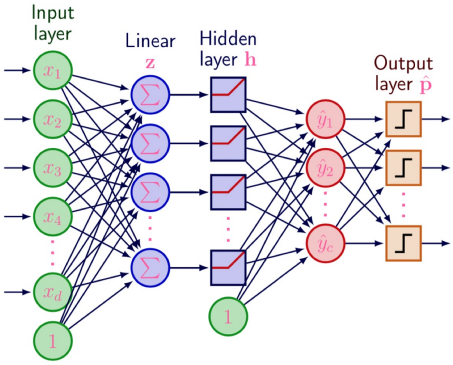
$\Rightarrow h_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

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

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

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

MULTI-CLASS CLASSIFICATION



Idea:

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

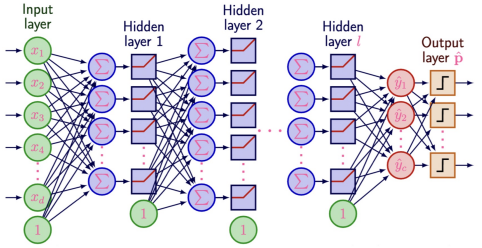
$\hat{y} = Wh + b$
$\hat{p} = \text{softmax}(\hat{y})$ } learning linear classifier by logistic regression

ACTIVATION FUNCTIONS

Choices for activation function:

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



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

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

SGD FOR MLP

To train w, we can use gradient descent:

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

$$[\ell \circ f](x_i, y_i; w) = \ell[f(x_i; w), y_i]$$

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.

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

Model:

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

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

The gradient of the network is defined by

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

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

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

We will show that

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

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

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

and so

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

lastly,

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

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 $k \in \mathbb{N}$, $W \in \mathbb{R}^{kxd}$, $b \in \mathbb{R}^k$ & $U \in \mathbb{R}^{cxk}$ such that

$$\sup_x \|f(x) - g(x)\|_2 < \varepsilon.$$

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

ie $\|f(x) - g(x)\|_2 < \varepsilon \quad \forall x$, 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

Idea: 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

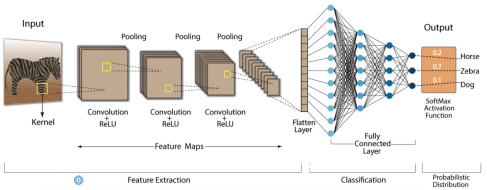
Idea: Normalize the input over the minibatch dimensions.

Chapter 9: Convolutional Neural Networks

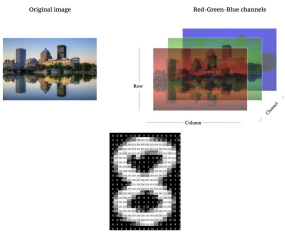
MOTIVATION

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

- 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?

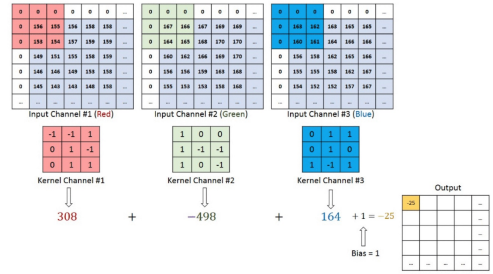
Note traditional image processing algorithms use convolution.

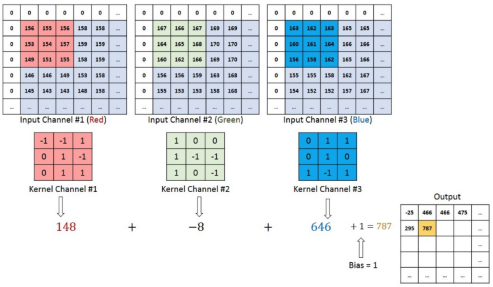
CONVOLUTION [MULTI-CHANNEL INPUT]

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

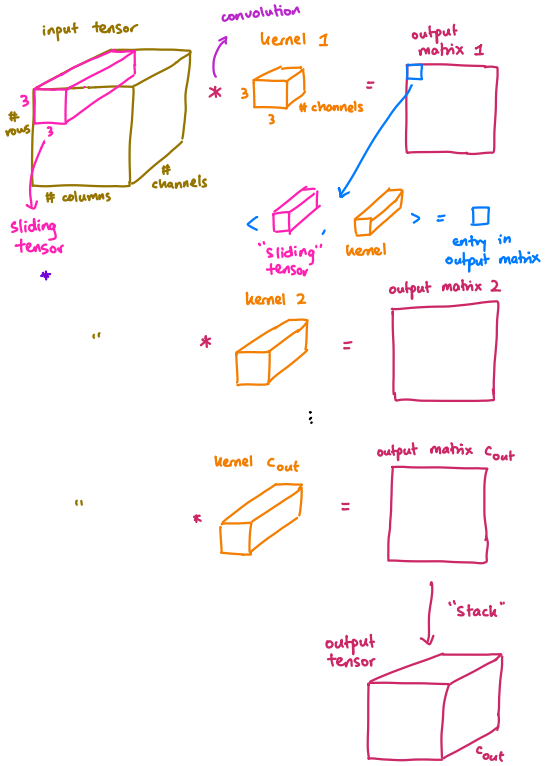
2. Idea: For each entry of the output matrix, we take the "sliding window inner product" for each kernel channel - input channel pair, and then sum the products together.

eg





3. 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:

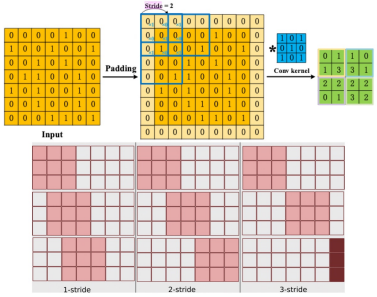
① Filter / kernel size:
- eg 3x3, 5x5
- by default, # of channels on each filter is the same as input

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



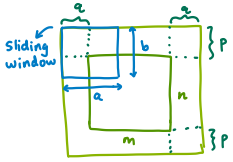
SIZE CALCULATION

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 $W = \begin{pmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \end{pmatrix} \in \mathbb{R}^{2\times 2}$ & 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

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

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

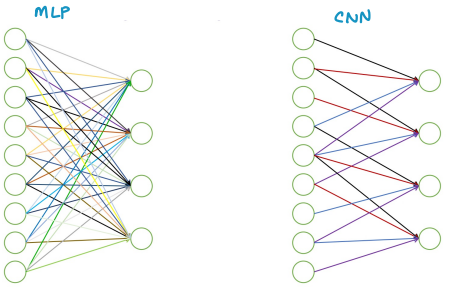
$$W_{circ} = \begin{pmatrix} w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 & 0 \\ 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} & 0 \\ 0 & 0 & 0 & 0 & w_{00} & w_{01} & 0 & w_{10} & w_{11} \end{pmatrix} \in \mathbb{R}^{4\times 9}$$

See that

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

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

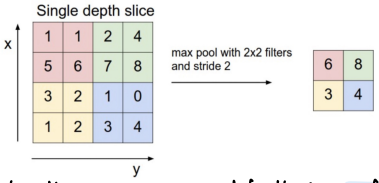
Hence, we can view CNN as a MLP, but with weight sharing.



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.



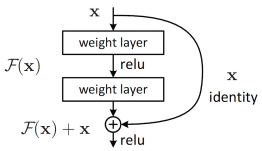
4. Note that pooling by default is performed on each slice separately, so the number of channels is the same between the input & output.
5. If we set the kernel size = input size, this is known as "global pooling".

DEEPER MODELS

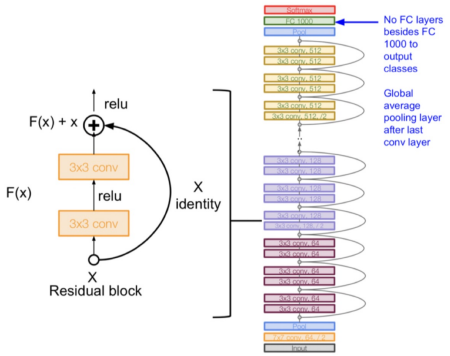
1. Note deeper models (ie more layers) are better but are more difficult to train.

RESIDUAL BLOCK

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



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



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

$$\operatorname*{argmax}_Y P(y_1, \dots, y_m \mid x_1, \dots, x_n)$$

AUTO-REGRESSIVE / GREEDY METHOD

Idea: we repeatedly compute

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

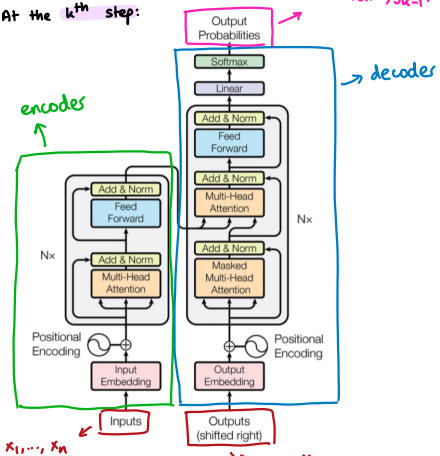
eg

Step 0 X: Where is University of Waterloo?
Step 1 Y: [START]; $\Pr(\text{It} \mid X \text{ [START]})$ highest
Step 2 Y: [START] It; $\Pr(\text{is} \mid X \text{ [START] It})$ highest
Step 3 Y: [START] It is; $\Pr(\text{at} \mid X \text{ [START] It is})$ highest
Step 4 Y: [START] It is at; $\Pr(\text{Waterloo} \mid X \text{ [START] It is at})$ highest
Step 5 Y: [START] It is at Waterloo; $\Pr(\text{[END]} \mid X \text{ [START] It is at Waterloo})$ 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: $P(y_k \mid x_1, \dots, x_n, y_1, \dots, y_{k-1})$



TOKENIZER

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

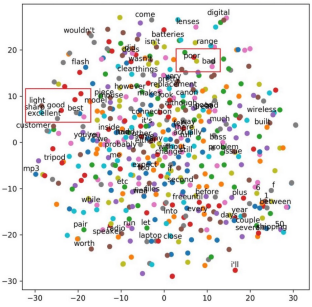
TOKEN EMBEDDING

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

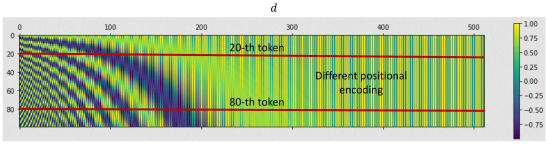
1. Idea: the order of tokens in the sentence changes its meaning.
2. We use a positional encoding matrix $W^P \in \mathbb{R}^{n \times d}$:

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

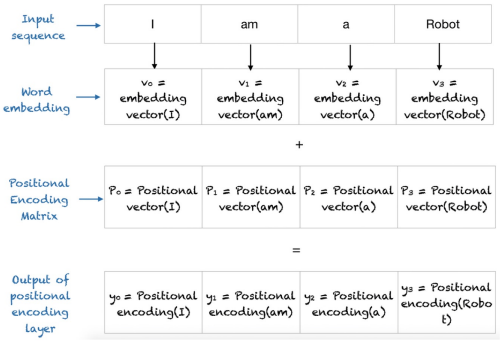
$$i = 0, \ldots, \frac{d}{2} - 1$$

↳ no parameter to be learnt!

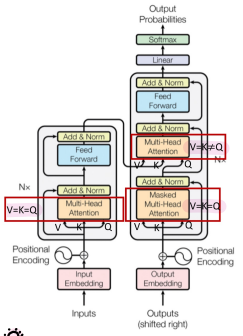
3. We then just add W^P to the $n \times d$ token embedding.



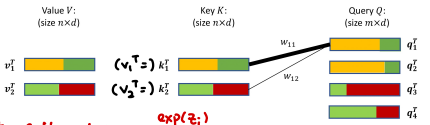
4. Putting it together:



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:



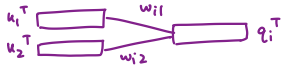
Let $\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$, for $z = (z_1, \ldots, z_n)$.

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 i^{th} 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

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

- Softmax is a "row-wise" operation

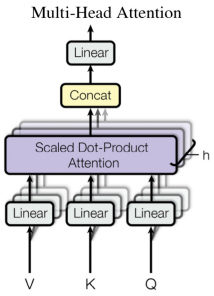
2. There is no learnable parameters so far!

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 = $64 \times 8 = 512$

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

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

where

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

Chapter 11: Large Language Models

COMPUTATIONAL COMPLEXITY

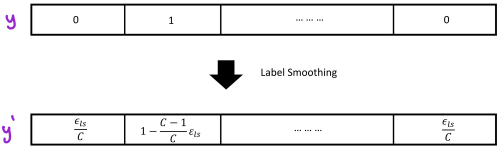
1. Self-attention: $O(n^2d + nd^2)$ per layer
   - $Q \in \mathbb{R}^{n \times d}$, $K^T \in \mathbb{R}^{d \times n}$.
   - ⇒ computing QK^T takes $O(n^2d)$ time.
   - $QK^T \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{n \times d}$
   - ⇒ computing $\text{softmax}(\frac{QK^T}{\sqrt{d}}) \cdot V$ 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

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

where C is the # of classes.

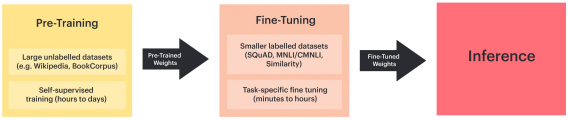


- ε_{ls} is a hyperparameter.

BERT VS GPT

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

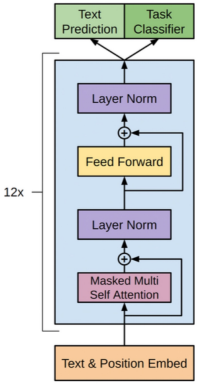


- pre-training takes weeks/months
- fine-tuning takes days to weeks/months

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

1. Goal: we want to find

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

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

FINE-TUNING

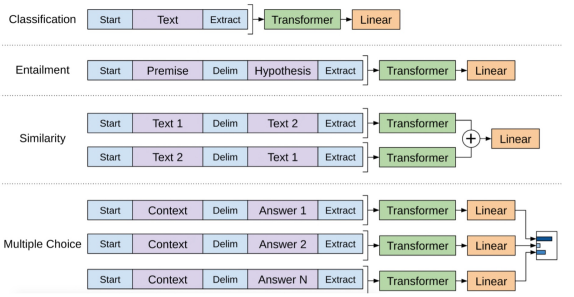
1. Goal: We want to find

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

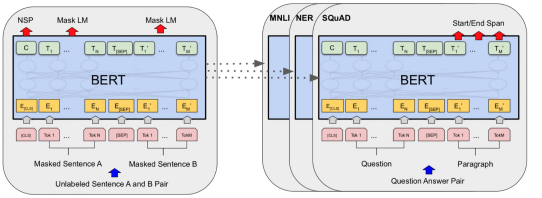
task-aware supervised loss — pretraining loss

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



BERT STRUCTURE



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

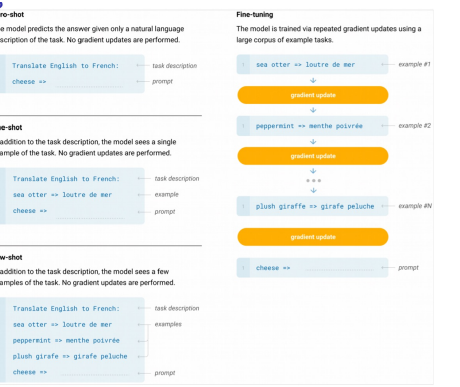
Idea: Improve BERT by
1. training the model longer;
2. use bigger batches;
3. use more data;
4. remove NSP; &
5. train on longer sentences.

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"



GPT-3

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