DS 4440

(More on) Linear Models

Last Time: The Perceptron!

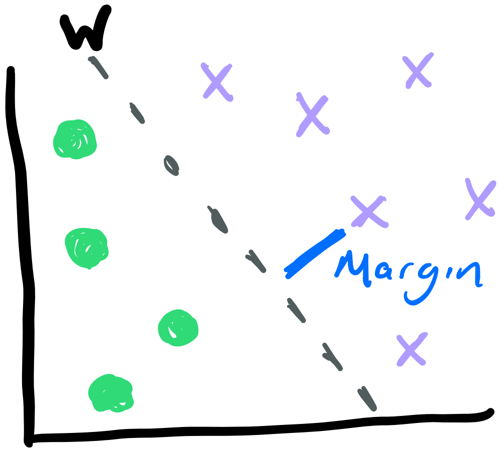
$$\hat{y}_i = \begin{cases} 1 & \text{if } w \cdot x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

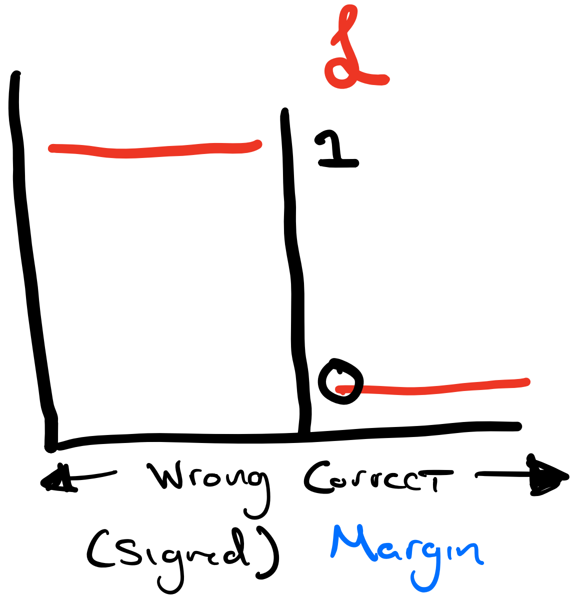
* Assumes bias terms folded into x_i, w.

Given $\langle x, y \rangle$, we fit this, i.e., find $\hat{w}$ to ↓ minimize a loss

$$\mathcal{L}(\hat{y}_i, y_i) = \begin{cases} \emptyset & \text{if } \hat{y}_i = y_i \\ 1 & \text{otherwise} \end{cases}$$

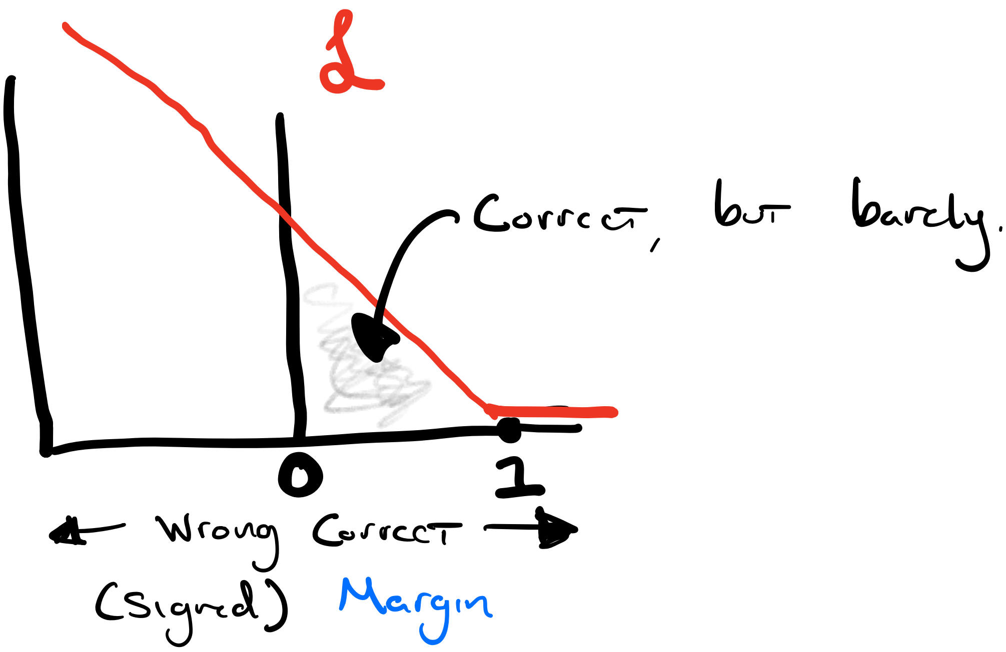
This is a 0/1 loss. It's simple, but not a great choice. (Why?)





One alternative: Hinge loss

$$\mathcal{L}_H(\hat{W}x_i, y_i) \stackrel{def}{=} \max\{0, 1 - y_i(\hat{W}x_i)\}$$



But there are many loss functions we can use.

In general

$$\hat{w} \leftarrow \arg\min_{w} \mathcal{L}(x, y \mid w)$$

$$= \arg\min_{w} \sum_{i=1}^{N} \mathcal{L}(x_i, y_i \mid w)$$

Only minimizing loss may result in overfitting. So we often add Regularization on parameters, e.g.,

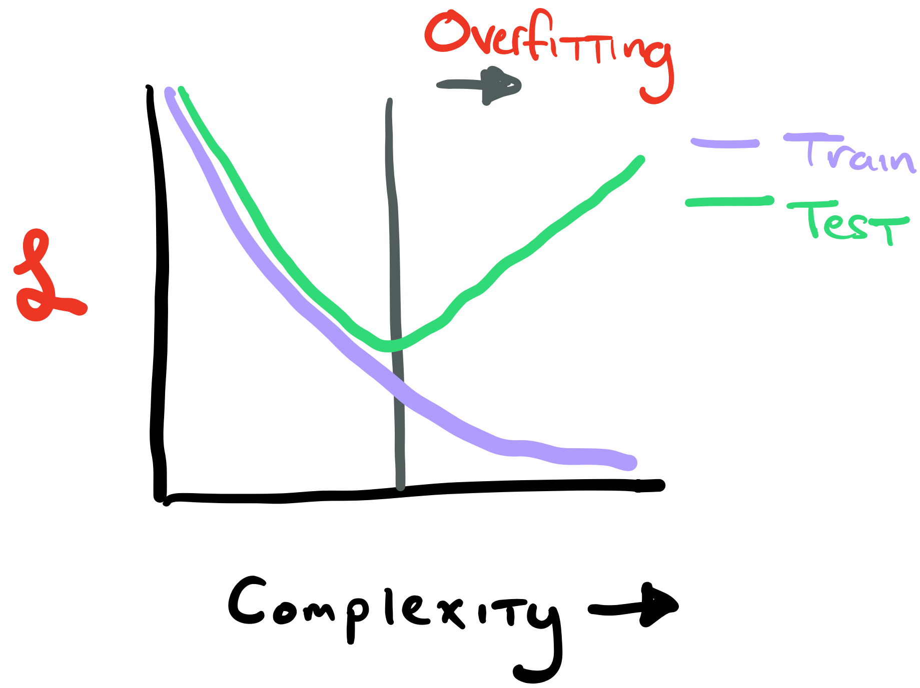
$$R(w) \stackrel{\text{def}}{=} \|w\|_2^2 = \sum_{j=1}^{D} w_j^2$$

Composite objective then:

$$\arg\min_{w} \underbrace{\mathcal{L}(x, y \mid w)}_{\text{Empirical loss}} + \underbrace{\lambda R(w)}_{\text{Regularizer}}$$

λ is a hyperparameter that dictates regularization "strength".

$\uparrow \lambda$ (bias) $\rightarrow$ Lower Variance



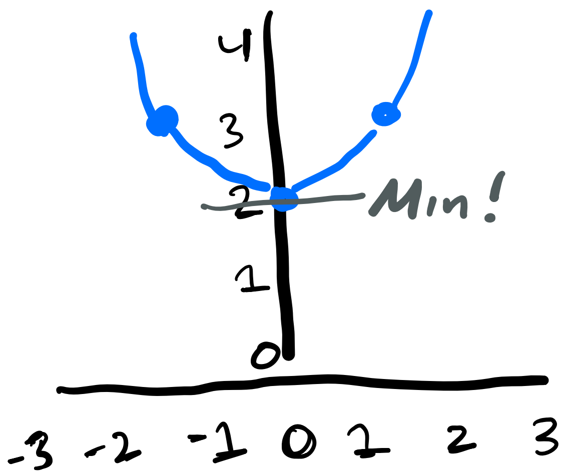
$\boxed{\text{Central Q: How to find } \hat{w}?}$

We did something kind of ad-hoc for Perceptron. A more general strategy — and the workhorse of Deep Learning — is gradient based

Optimization.

A quick refresher:

$$f(x) = x^2 + 2$$



Minimum value is where

$$\frac{d}{dx} f(x) = 0$$

$$\frac{d}{dx} f(x) = 2x$$

Minimum @ $x = 0$.

Refresher! Derivative Rules

$$\frac{d}{dx} c = 0 \qquad \frac{d}{dx} x \cdot c = c$$

$$\frac{d}{dx} \lg x = \frac{1}{x}$$

$$\frac{d}{dx} x^K = K \cdot x^{(K-1)}$$ Power rule

$$\frac{d}{dx}[f(x) + g(x)] = \frac{d}{dx} f(x) + \frac{d}{dx} g(x)$$

Sum rule

$$\frac{d}{dx} f(x) \cdot g(x) = \left(\frac{d}{dx} f(x)\right) \cdot g(x) + f(x) \cdot \left(\frac{d}{dx} g(x)\right)$$

Product rule

$$\frac{d}{dx} f(\underbrace{g(x)}_{u}) = \frac{d}{du} f(u) \cdot \frac{d}{dx} u$$

Chain rule

Gradients ∇

$$\vec{w} = [w_1 \; \dots \; w_d] \qquad f(\vec{w})$$

$$\nabla_{\vec{w}} f(\vec{w}) = \left[\frac{\partial f}{\partial w_1} \; \dots \; \frac{\partial f}{\partial w_d} \right]$$

e.g. $f(\vec{w}) = w_1^3 \cdot w_2 + 5w_2 + w_1 \cdot w_3$

$$\nabla_{\vec{w}} f(\vec{w}) = [3w_1^2 w_2 + w_3, \; w_1^3 + 5, \; w_1]$$

If f maps to $\underline{m}$ outputs we have a Jacobian

$$\begin{array}{cc} m & \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(x) & \cdots & \frac{\partial}{\partial x_d} f_1(x) \\ \cdots & \ddots & \cdots \\ \frac{\partial}{\partial x_1} f_m(x) & & \frac{\partial}{\partial x_d} f_m(x) \end{bmatrix} \\ & d \end{array} \qquad \underbrace{\nabla_x f(x)_1}_{d\text{-dims}}$$

Useful Identities

$A \in \mathbb{R}^{n \times m}, \quad x \in \mathbb{R}^{m \times 1} \qquad \boxed{\nabla_x A x = A}$

$$(\nabla_x A x)_i = \nabla_x (A_{i1} x_1 + \dots + A_{im} x_m)$$
$$= [A_{i1} \; \dots \; A_{im}]$$

$A \in \mathbb{R}^{n \times m}, \quad x \in \mathbb{R}^{1 \times n} \qquad \boxed{\nabla_x x A = A^T}$

$$(\nabla_x x A)_j = \nabla_x (x_1 A_{1j} + \dots x_n A_{nj})$$
$$= [A_{1j} \; A_{2j} \; \dots \; A_{nj}]$$ jth col

$$\rightarrow \nabla_x x A = A^T$$

Elementwise Operations λ

$$y = \lambda \underset{x}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}} = \begin{bmatrix} \lambda(x_1) \\ \lambda(x_2) \\ \vdots \\ \lambda(x_m) \end{bmatrix} \qquad \nabla_x y = ?$$

$$= m \begin{bmatrix} \frac{\partial}{\partial x_1} \lambda(x_1) & \dots & \frac{\partial}{\partial x_m} \lambda(x_1) \\ \dots \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} \lambda(x_1) & & \emptyset \\ \dots \end{bmatrix}$$

$$\left[\frac{\partial}{\partial x_1}\lambda(x_m) \cdots \frac{\partial}{\partial x_m}\lambda(x_m)\right] \quad \left[\phi \quad \cdots \quad \frac{\partial}{\partial x_m}\lambda(x_m)\right]$$

M

Revisiting the Start of Today, let's consider the ∇ of Hinge loss + l_2 Regularizer. (Exercise!)

$$\mathcal{L}_H(\hat{w}x_i, y_i) \overset{\text{def}}{=} \text{Max}\{0, \underline{1 - y_i(\hat{w}x_i)}\}$$

$$\nabla_w\left[\underline{(1 - y_i(\hat{w}x_i))} + \lambda \frac{1}{2}\sum_j w_j^2\right]$$

$$= \nabla_w - y_i(\hat{w}x_i) + \frac{\lambda}{2}\nabla_w \underbrace{\sum_j w_j^2}_{\substack{0 \ \forall k \quad k \neq j \\ 2w_j \text{ when } j=k}}$$

$$= \underline{-y_i x_i} + \lambda w$$

Drops if ϕ is Max

Least Squares regression (from an ML view)

$$\underbrace{\begin{array}{cc} & d \\ n & \begin{bmatrix} \quad & \quad & \quad \end{bmatrix} \\ & X \end{array} \quad \begin{array}{cc} & 1 \\ n & \begin{bmatrix} \quad \end{bmatrix} \\ & y \end{array}}_{\text{Training data}} \quad \underbrace{\begin{array}{cc} & 1 \\ d & \begin{bmatrix} \quad \end{bmatrix} \\ & w \end{array}}_{\text{Parameters}}$$

$$\mathcal{L} = \sum_{i=1}^{n} \left(y_i - \underbrace{x_i w}_{\hat{y}_i} \right)^2$$

$$= \sum \left(\| \quad \| \right)$$ Sum of Squared errors.

$$\nabla_w (y - Xw)^2$$

$$= \nabla_u u^2 \cdot \nabla_w u$$

$$= 2(\underset{n\times 1}{y} - \underset{n\times 1}{Xw}) \cdot (\underset{d\times n}{-X^T})$$

$$= -2X^T(y - Xw)$$

$$= \underbrace{-2\underset{d\times n}{X^T}\underset{n\times 1}{y}}_{d\times 1} + \underbrace{2\underbrace{X^TX}_{d\times d}\underset{d\times 1}{w}}_{d\times 1}$$

Set to $\vec{0}$.

$$X^TXw = X^Ty$$

$$w^* = (X^TX)^{-1}X^Ty$$

For LR we can solve analytically.

But not always the case!

A more flexible approach is

Gradient Descent, an iterative optimization algorithm.

$$\hat{W}^{(t+1)} \leftarrow \hat{W}^{(t)} - \underbrace{\alpha}_{\text{learning rate}} \nabla_W \mathcal{L}(\underbrace{x, y}_{\text{usually batched}} \mid W^{(t)})$$

Let's see in Colab!