CSCI 567: Machine Learning

Vatsal Sharan
Fall 2022

Lecture 2, Sep 1

USC University of Southern California

Administrivia

- HW1 is out
- Due in about 2 weeks (9/14 at 2pm). **Start early!!!**
- Post on Ed Discussion if you're looking for teammates.

Recap

Supervised learning in one slide

Loss function: What is the right loss function for the task?

Representation: What class of functions should we use?

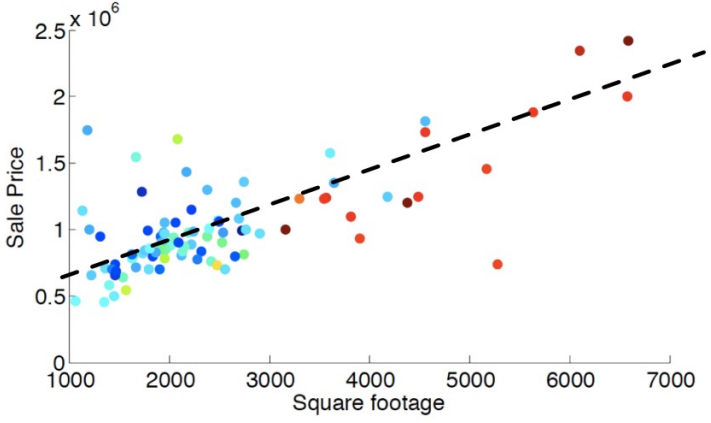
Optimization: How can we efficiently solve the empirical risk minimization problem?

Generalization: Will the predictions of our model transfer gracefully to unseen examples?

All related! And the fuel which powers everything is data.

Linear regression

Predicted sale price = **price_per_sqft** × square footage + **fixed_expense**



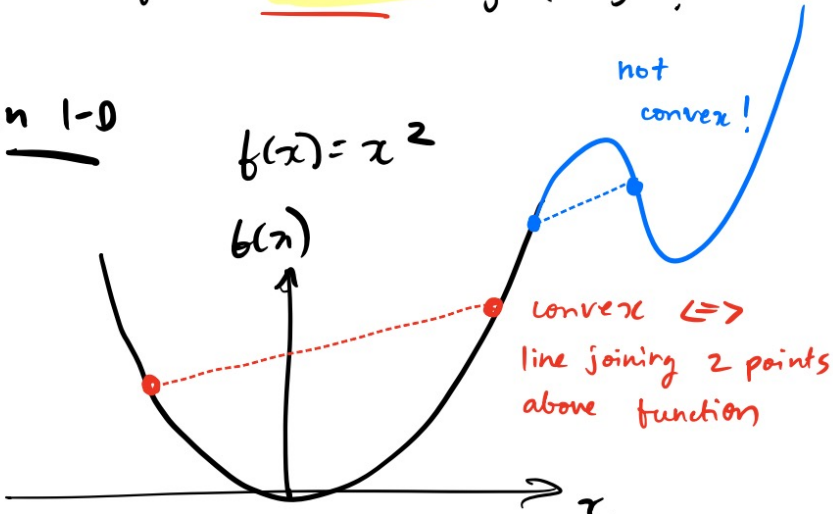
How to solve this? Find **stationary points**

Are stationary points minimizers?

Yes, for convex objectives !

In 1-D

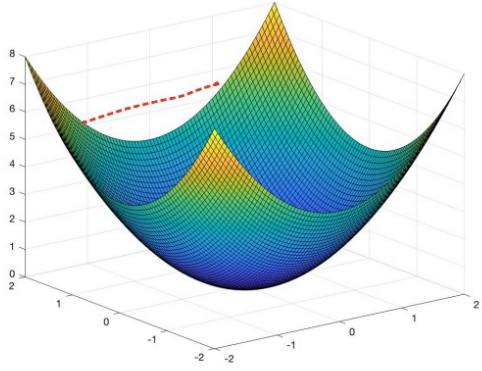
$f(x) = x^2$



convex <=> line joining 2 points above function

$f''(x) \geq 0 \quad \forall x$ ⟶

In high dimensions :



$\nabla^2(F(x))$ is positive semi-definite (psd)

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_i (\tilde{\boldsymbol{x}}_i^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_i)^2$$

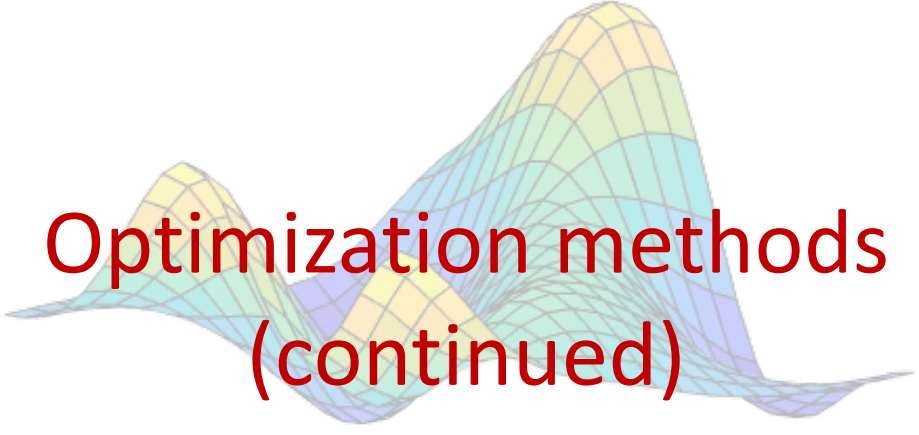
Find stationary points:

$$\begin{aligned} \nabla \mathrm{RSS}(\tilde{\boldsymbol{w}}) &= 2 \sum_i \tilde{\boldsymbol{x}}_i (\tilde{\boldsymbol{x}}_i^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_i) \propto \left(\sum_i \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^{\mathrm{T}} \right) \tilde{\boldsymbol{w}} - \sum_i \tilde{\boldsymbol{x}}_i y_i \\ &= (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} \end{aligned}$$

where

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} \tilde{\boldsymbol{x}}_1^{\mathrm{T}} \\ \tilde{\boldsymbol{x}}_2^{\mathrm{T}} \\ \vdots \\ \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{n \times (d+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^n$$

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}})^{-1} \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}$$



Optimization methods (continued)

Problem setup

Given: a function $\mathrm{F}(\boldsymbol{w})$

Goal: minimize $\mathrm{F}(\boldsymbol{w})$ (approximately)

Two simple yet extremely popular methods

Gradient Descent (GD): simple and fundamental

Stochastic Gradient Descent (SGD): faster, effective for large-scale problems

Gradient is the *first-order information* of a function.

Therefore, these methods are called *first-order methods*.

Gradient descent

GD: keep moving in the *negative gradient direction*

Start from some $\boldsymbol{w}^{(0)}$. For $t = 0, 1, 2, \ldots$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla F(\boldsymbol{w}^{(t)})$$

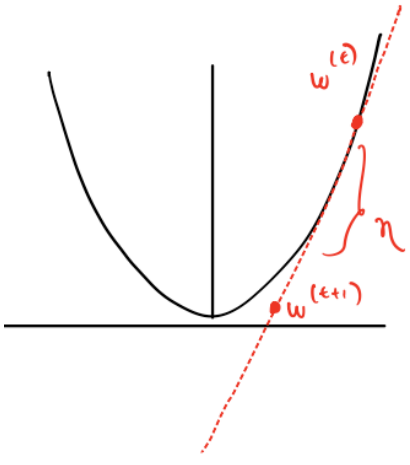
where $\eta > 0$ is called step size or learning rate

- in theory η should be set in terms of some parameters of F
- in practice we just try several small values
- might need to be changing over iterations (think $F(w) = |w|$)
- adaptive and automatic step size tuning is an active research area

Why GD?

Intuition: First-order Taylor approximation

$$F(w) \approx F(w^{(t)}) + \nabla F(w^{(t)})^T (w - w^{(t)})$$



$$F(w^{(t+1)}) \approx F(w^{(t)}) - \eta \| \underbrace{\nabla F(w^{(t)})}_{\geq 0} \|_2^2$$

$$\Rightarrow F(w^{(t+1)}) \overset{\approx}{\leq} F(w^{(t)})$$

(this is only an approximation, and can be invalid if step size is too large)

Convergence guarantees for GD

Many results for GD (and many variants) on *convex objectives*.
They tell you how many iterations t (in terms of ε) are needed to achieve

$$F(\boldsymbol{w}^{(t)}) - F(\boldsymbol{w}^{*}) \leq \varepsilon$$

Convergence guarantees for GD

Many results for GD (and many variants) on *convex objectives*.
They tell you how many iterations t (in terms of ε) are needed to achieve

$$F(\boldsymbol{w}^{(t)}) - F(\boldsymbol{w}^*) \leq \varepsilon$$

Even for *nonconvex objectives*, some guarantees exist:
e.g. how many iterations t (in terms of ε) are needed to achieve

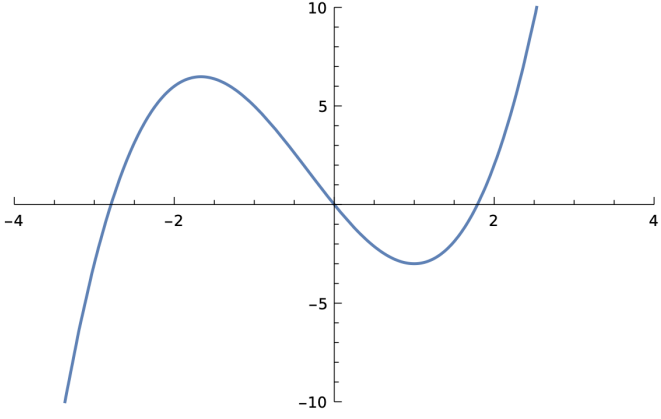
$$\left\|\nabla F(\boldsymbol{w}^{(t)})\right\| \leq \varepsilon$$

that is, how close is $\boldsymbol{w}^{(t)}$ as an approximate stationary point

for convex objectives, stationary point ⇒ global minimizer
for nonconvex objectives, what does it mean?

Stationary points: non-convex objectives

A stationary point can be a local minimizer or even a local/global maximizer (but the latter is not an issue for GD).

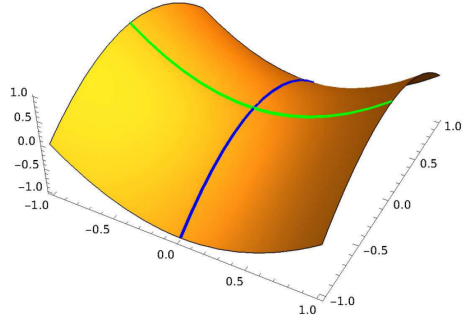


$$f(w) = w^3 + w^2 - 5w$$

Stationary points: non convex objectives

A stationary point can also be *neither a local minimizer nor a local maximizer!*

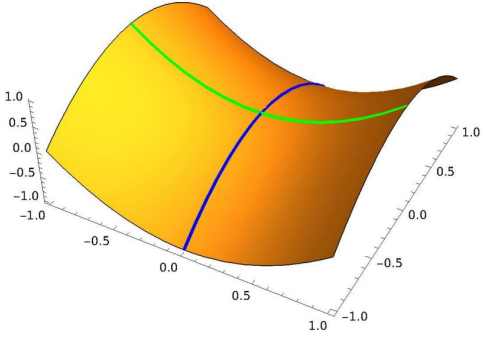
- $f(\boldsymbol{w}) = w_1^2 - w_2^2$
- $\nabla f(\boldsymbol{w}) = (2w_1, -2w_2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- local max for blue direction $(w_1 = 0)$
- local min for green direction $(w_2 = 0)$



Switch to Colab

Stationary points: non convex objectives

This is known as a saddle point

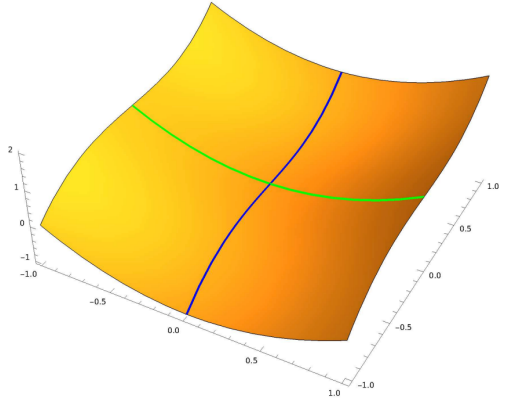


- but GD gets stuck at $(0, 0)$ only if initialized along the green direction
- so not a real issue especially *when initialized randomly*

Stationary points: non convex objectives

But not all saddle points look like a "saddle"...

- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- not local min/max for blue direction $(w_1 = 0)$

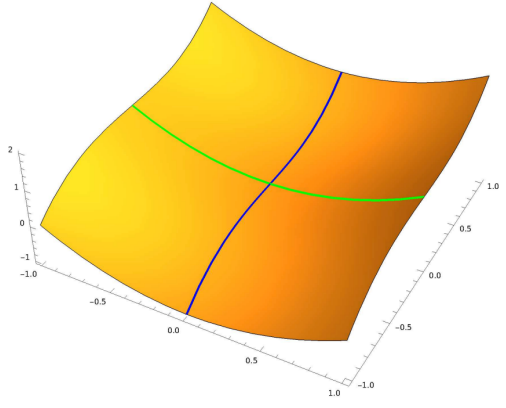


Switch to Colab

Stationary points: non convex objectives

But not all saddle points look like a "saddle"...

- $f(\boldsymbol{w}) = w_1^2 + w_2^3$
- $\nabla f(\boldsymbol{w}) = (2w_1, 3w_2^2)$
- so $\boldsymbol{w} = (0, 0)$ is stationary
- not local min/max for blue direction $(w_1 = 0)$
- GD gets stuck at $(0, 0)$ for *any initial point with $w_2 \geq 0$ and small η*



Even worse, distinguishing local min and saddle point is generally *NP-hard*.

Stochastic Gradient descent

GD: keep moving in the *negative gradient direction*
SGD: keep moving in the *noisy negative gradient direction*

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \tilde{\nabla} F(\boldsymbol{w}^{(t)})$$

where $\tilde{\nabla} F(\boldsymbol{w}^{(t)})$ is a random variable (called **stochastic gradient**) s.t.

$$\mathbb{E}\left[\tilde{\nabla} F(\boldsymbol{w}^{(t)})\right] = \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(unbiasedness)}$$

Stochastic Gradient descent

GD: keep moving in the *negative gradient direction*
SGD: keep moving in the *noisy negative gradient direction*

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \tilde{\nabla} F(\boldsymbol{w}^{(t)})$$

where $\tilde{\nabla} F(\boldsymbol{w}^{(t)})$ is a random variable (called **stochastic gradient**) s.t.

$$\mathbb{E}\left[\tilde{\nabla} F(\boldsymbol{w}^{(t)})\right] = \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(unbiasedness)}$$

- Key point: it could be much faster to obtain a stochastic gradient!
- Similar convergence guarantees, usually needs more iterations but each iteration takes less time.

Switch to Colab

Summary: Gradient descent & Stochastic Gradient descent

- GD/SGD coverages to a stationary point
- for convex objectives, this is all we need

Summary: Gradient descent & Stochastic Gradient descent

- GD/SGD coverages to a stationary point
- for convex objectives, this is all we need
- for nonconvex objectives, can get stuck at local minimizers or “bad” saddle points (random initialization escapes “good” saddle points)
- recent research shows that *many problems have no “bad” saddle points or even “bad” local minimizers*
- justify the practical effectiveness of GD/SGD (default method to try)

Second-order methods

GD: first-order Taylor approximation

$$F(w) \approx F(w^{(t)}) + \nabla F(w^{(t)})^T (w - w^{(t)})$$

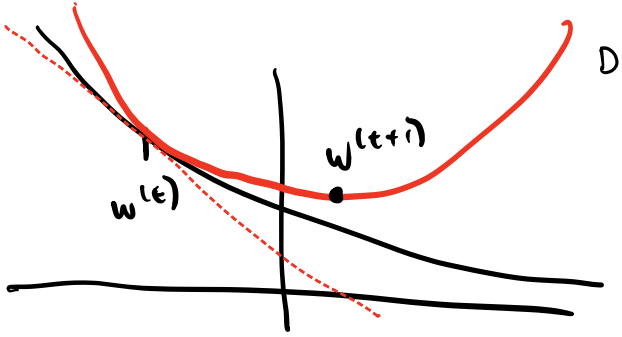
$$f(y) \approx f(x) + f'(x)(y-x) + \frac{f''(x)}{2}(y-x)^2$$

What about a second-order Taylor approximation?

$$F(w) \approx F(w^{(t)}) + \nabla F(w^{(t)})^T (w - w^{(t)}) + \frac{1}{2}(w - w^{(t)})^T H_t (w - w^{(t)})$$

where $H_t = \nabla^2 F(w^{(t)}) \in \mathbb{R}^{d \times d}$ is Hessian of F at $w^{(t)}$

$$H_{t,i,j} = \frac{\partial^2 F(w)}{\partial w_i \partial w_j}\Big|_{w = w^{(t)}}$$



Define $\tilde{F}(w) :=$ 2nd order approximation

Set $\nabla \tilde{F}(w) = 0$ to find minima.

$$\nabla F(w^{(t)}) + H_t w - H_t w^{(t)} - \frac{1}{2} H_t w^{(t)} = 0$$

$$H_t w = H_t w^{(t)} - \nabla F(w^{(t)})$$

$$\Rightarrow \quad w = w^{(t)} - H_t^{-1} \nabla F(w^{(t)})$$

Newton's method: $w^{(t+1)} \leftarrow w^{(t)} - H_t^{-1} \nabla F(w^{(t)})$

GD: $w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla F(w^{(t)})$

Newton's method

* no learning rate
* super fast convergence
* Know & invert Hessian
  (inversion takes $O(d^3)$ time naively)

GD

* need to tune η
* slower convergence
* fast!
  $O(d)$ time

Linear classifiers

The Setup

Recall the setup:

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{d}}$
- output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- goal: learn a mapping $f : \mathbb{R}^{\mathsf{d}} \to [\mathsf{C}]$

This lecture: **binary classification**

- Number of classes: $\mathsf{C} = 2$
- Labels: $\{-1, +1\}$ (cat or dog)

Representation: Choosing the function class

Let's follow the recipe, and pick a function class $\mathcal{F}$.

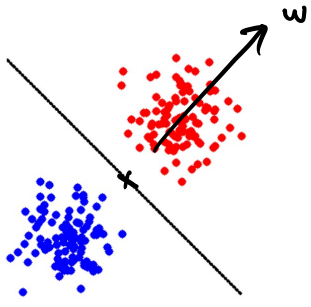
We continue with linear models, but how to predict a label using $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$?

Sign of $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$ predicts the label:

$$\text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \begin{cases} +1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} > 0 \\ -1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \leq 0 \end{cases}$$

(Sometimes use sgn for sign too.)

Representation: Choosing the **function class**

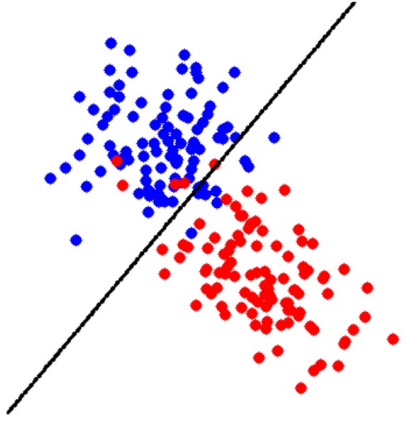


Good choice if data is linearly separated.

Definition: The function class of seperating hyperplanes is defined as:

$$\mathcal{F} = \{ f(x) = \mathrm{sgn}(w^T x) : w \in \mathbb{R}^d \}.$$

Still makes sense for “almost” linearly separable data

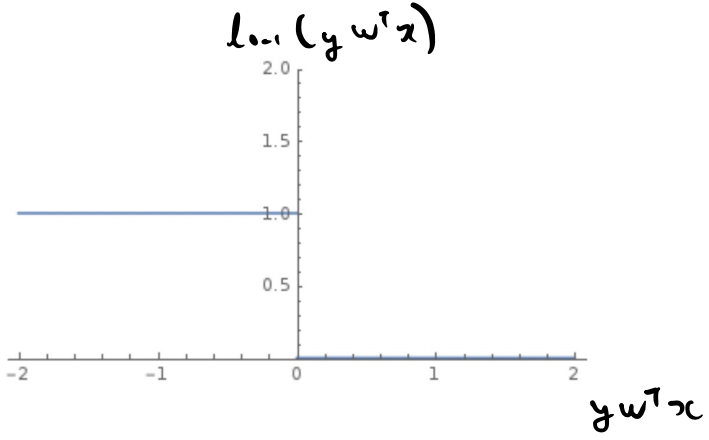


Choosing the **loss function**

Most common loss $\ell(f(x), y) = \mathbb{1}(f(x) \neq y)$

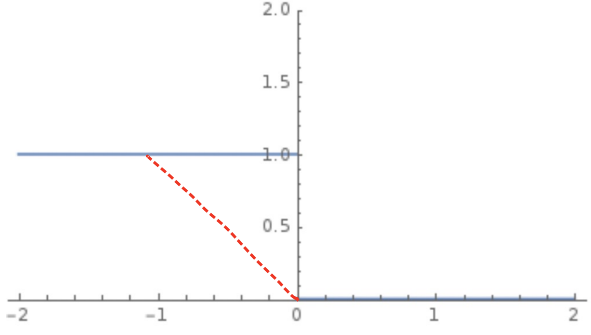
Loss as a function of $yw^T x$

$$\ell_{0-1}(yw^T x) = \mathbb{1}(yw^T x \leq 0)$$



Choosing the loss function: **minimizing 0/1 loss is hard**

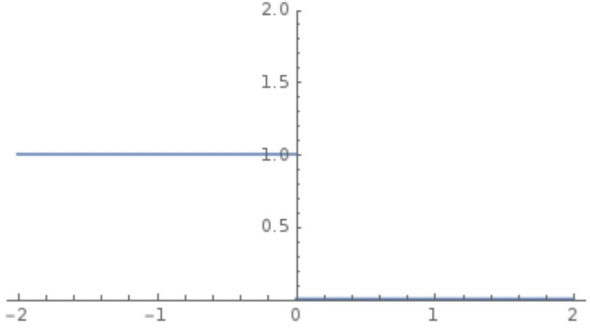
However, 0-1 loss is not convex.



Even worse, minimizing 0-1 loss is NP-hard in general.

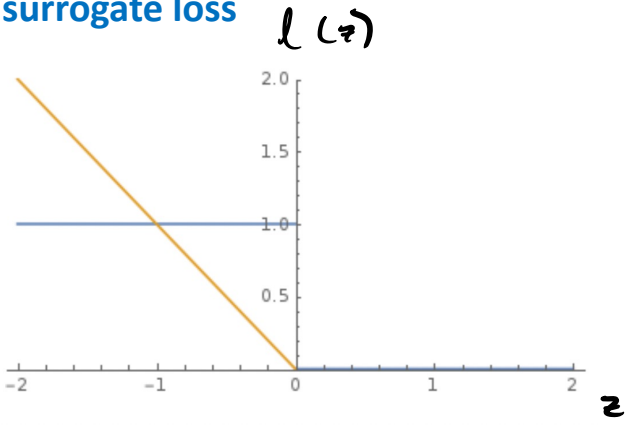
Choosing the loss function: **surrogate losses**

Solution: use a **convex surrogate loss**



Choosing the loss function: surrogate losses

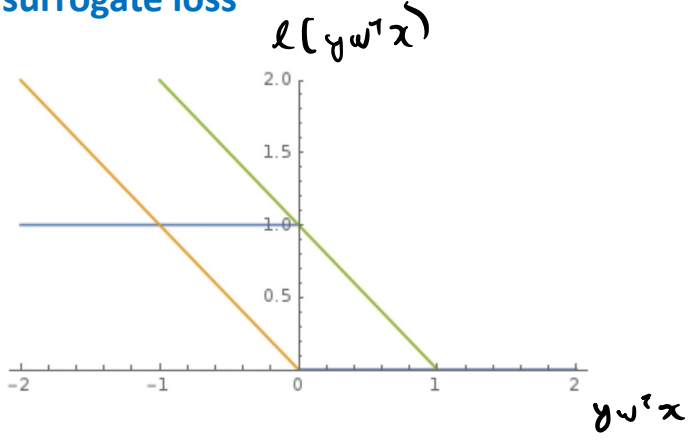
Solution: use a **convex surrogate loss**



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)

Choosing the loss function: surrogate losses

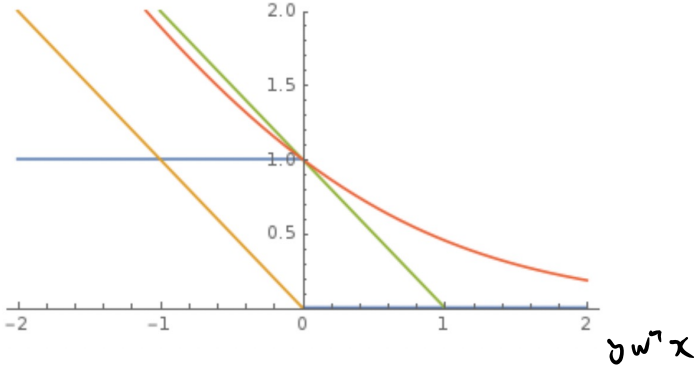
Solution: use a **convex surrogate loss**



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$(used in SVM and many others)

Choosing the loss function: surrogate losses

Solution: use a **convex surrogate loss**



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$(used in SVM and many others)
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z))$ (used in logistic regression; the base of $\log$ doesn't matter)

Onto **Optimization**!

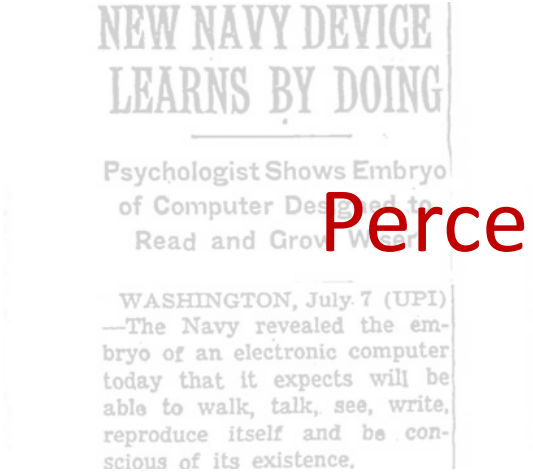
Find ERM:

$$w^* = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \left(\sum_{i=1}^{n} \ell(y_i w^T x_i) \right)$$

where $\ell(\cdot)$ is a convex surrogate loss.

- No closed-form solution in general (in contrast to linear regression)
- We can use our **optimization** toolbox!

New York Times, 1958



NEW NAVY DEVICE LEARNS BY DOING

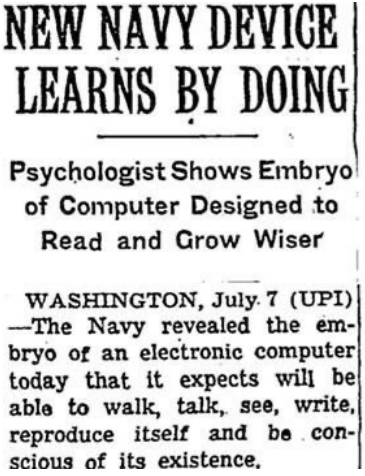
Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

Perceptron

*The Navy last week demonstrated the embryo of an electronic computer named the **Perceptron** which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control."*

New York Times, 1958



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

*The Navy last week demonstrated the embryo of an electronic computer named the **Perceptron** which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control."*

Recall perceptron loss

$$F(w) = \frac{1}{n} \sum_{i=1}^{n} \ell_{percep}(y_i w^T x_i)$$

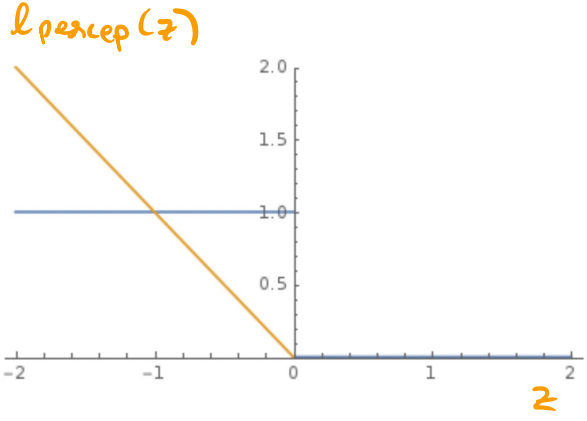
$$= \frac{1}{n} \sum_{i=1}^{n} \max\{0, -y_i w^T x_i\}$$

Let's try GD/SGD.

What's the gradient:

$$0, \quad z > 0$$

$$-1, \quad z \leq 0$$



Applying GD to perceptron loss

Gradient is

$$\nabla F(w) = (1/n) \sum_{i=1}^{n} -\mathbb{1}[y_i w^T x_i \leq 0] y_i x_i$$

(only misclassified examples count)

bD: $w \leftarrow w + \frac{\eta}{n} \sum_{i=1}^{n} \mathbb{1}[y_i w^T x_i \leq 0] y_i x_i$

need the entire training set for every bD update.

Applying SGD to perceptron loss

How to get a stochastic gradient?

Objective function of most ML tasks is a finite sum, trick applies generally

→ pick one example $i \in [n]$ uniformly at random, let

$$\tilde{\nabla} F(w^{(t)}) = -\mathbb{1}[y_i w^T x_i \leq 0]\, y_i x_i$$

Unbiased. why?

$$\mathbb{E}[\tilde{\nabla} F(w^{(t)})] = \frac{1}{n}\sum_{i=1}^{n} -\mathbb{1}[y_i w^T x_i \leq 0]\, y_i x_i$$

$$= \nabla F(w^{(t)})$$

SGD update: $w \leftarrow w + \eta\, \mathbb{1}(y_i w^T x_i \leq 0)\, y_i x_i$

fast! one datapoint per update

Perceptron algorithm

SGD with $\eta = 1$ on perceptron loss.

Initialize $w = 0$

Repeat

- Pick $x_i \sim \text{Unif}(x_1, \ldots, x_n)$
- If $\text{sgn}(w^T x_i) \neq y_i$

$$w \leftarrow w + y_i x_i$$

Perceptron algorithm: Intuition

Say that w makes mistake on (x_i, y_i)

$$y_i w^T x_i < 0$$

Consider $w' = w + y_i x_i$

$$y_i (w')^T x_i = y_i w^T x_i + y_i^2 x_i^T x_i$$

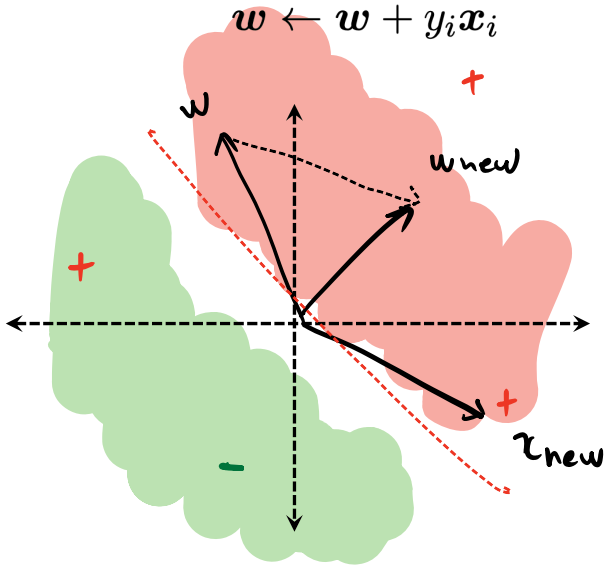
if $x_i \neq 0$

$$y_i (w')^T x_i > y_i w^T x_i$$

Perceptron algorithm: visually

Repeat:

- Pick a data point $\boldsymbol{x}_i$ uniformly at random
- If $\text{sgn}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i) \neq y_i$



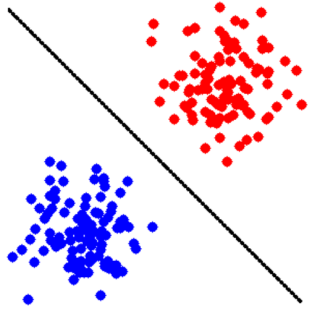
Related to question in class: if there are multiple ways to classify data:

Perceptron itself could find any of these hyperpl

HW1: Theory for perceptron!

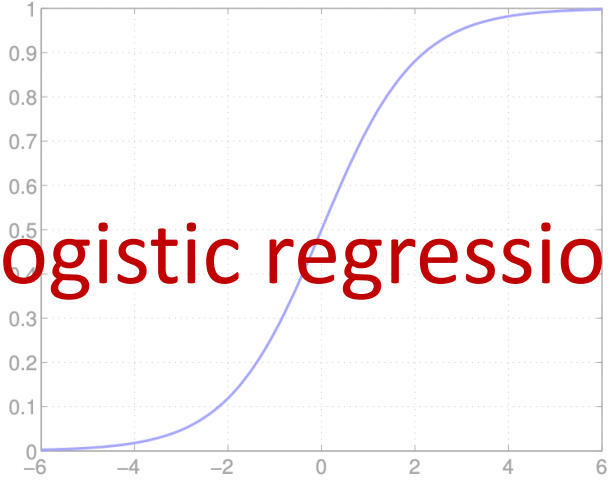
(HW 1) If training set is linearly separable

- Perceptron *converges in a finite number of steps*
- training error is 0



There are also guarantees when the data are not linearly separable.

Logistic regression



Logistic loss

$$F(w) = \frac{1}{n} \sum_{i=1}^{n} \ell_{\log}(y_i w^T x_i)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \ln\left(1 + e^{-y_i w^T x_i}\right)$$

Predicting probabilities

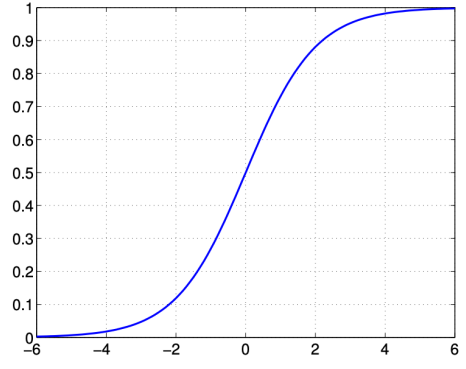
Instead of $\{\pm 1\}$, predict the probability.
(i.e. regression on probability)

Sigmoid + linear model:

$$\mathbb{P}(y=+1 \mid x, w) = \sigma(w^T x)$$

where $\sigma(z) = \dfrac{1}{1+e^{-z}}$

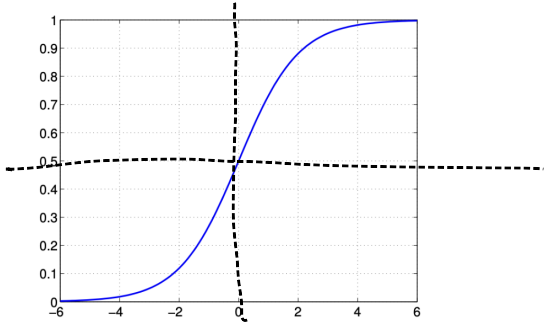
(sigmoid)



The sigmoid function

Properties of sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

- between 0 and 1 (good as probability)
- $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) \geq 0.5 \Leftrightarrow \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \geq 0$, consistent with predicting the label with $\mathrm{sgn}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$
- larger $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \Rightarrow$ larger $\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) \Rightarrow$ higher *confidence* in label 1
- $\sigma(z) + \sigma(-z) = 1$ for all z
- Therefore, the probability of label -1 is



$$\begin{aligned}\mathbb{P}(y = -1 \mid \boldsymbol{x}; \boldsymbol{w}) &= 1 - \mathbb{P}(y = +1 \mid \boldsymbol{x}; \boldsymbol{w}) \\ &= 1 - \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \sigma(-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})\end{aligned}$$

Therefore, we can model $\mathbb{P}(y \mid \boldsymbol{x}; \boldsymbol{w}) = \sigma(y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \dfrac{1}{1 + e^{-y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}}$

Maximum likelihood estimation

What we observe are labels, not probabilities.

Take a **probabilistic view**

- assume data is independently generated in this way by some $\boldsymbol{w}$
- perform Maximum Likelihood Estimation (MLE)

Specifically, what is the probability of seeing labels $y_1, \cdots, y_n$ given $\boldsymbol{x_1}, \cdots, \boldsymbol{x_n}$, as a function of some $\boldsymbol{w}$?

$$P(\boldsymbol{w}) = \prod_{i=1}^{N} \mathbb{P}(y_i \mid \boldsymbol{x_i}; \boldsymbol{w})$$

MLE: find $\boldsymbol{w}^*$ that **maximizes the probability** $P(w)$

Maximum likelihood solution

$$
\begin{aligned}
\boldsymbol{w}^* &= \operatorname*{argmax}_{\boldsymbol{w}} P(\boldsymbol{w}) = \operatorname*{argmax}_{\boldsymbol{w}} \prod_{i=1}^{n} \mathbb{P}(y_i \mid \boldsymbol{x_i}; \boldsymbol{w}) \\
&= \operatorname*{argmax}_{\boldsymbol{w}} \sum_{i=1}^{n} \ln \mathbb{P}(y_i \mid \boldsymbol{x_i}; \boldsymbol{w}) \\
&= \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} -\ln \mathbb{P}(y_i \mid \boldsymbol{x_i}; \boldsymbol{w}) \\
&= \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} \ln(1 + e^{-y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x_i}}) \\
&= \operatorname*{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} \ell_{\mathsf{logistic}}(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i) \\
&= \operatorname*{argmin}_{\boldsymbol{w}} F(\boldsymbol{w})
\end{aligned}
$$

Minimizing logistic loss is exactly doing MLE for the sigmoid model!

SGD to logistic loss

$$
\begin{aligned}
\boldsymbol{w} &\leftarrow \boldsymbol{w} - \eta \tilde{\nabla} F(\boldsymbol{w}) \\
&= \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \ell_{\text{logistic}}(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i) \\
&= \boldsymbol{w} - \eta \left(\frac{\partial \ell_{\text{logistic}}(z)}{\partial z} \Big|_{z = y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i} \right) y_i \boldsymbol{x}_i \\
&= \boldsymbol{w} - \eta \left(\frac{-e^{-z}}{1 + e^{-z}} \Big|_{z = y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i} \right) y_i \boldsymbol{x}_i \\
&= \boldsymbol{w} + \eta \sigma(-y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i) y_i \boldsymbol{x}_i \\
&= \boldsymbol{w} + \eta \mathbb{P}(-y_i \mid \boldsymbol{x}_i; \boldsymbol{w}) y_i \boldsymbol{x}_i
\end{aligned}
$$

$\mathbb{E}[\tilde{\nabla} F(w)] = \nabla F(w)$

(i is drawn uniformly from $[n]$)

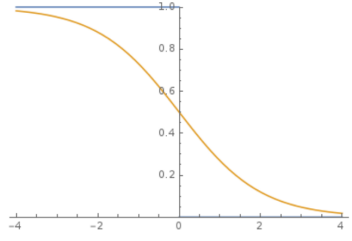
(chain rule)

$\frac{\partial}{\partial z}(\log(1+e^{-z})) = \frac{-e^{-z}}{1+e^{-z}}$

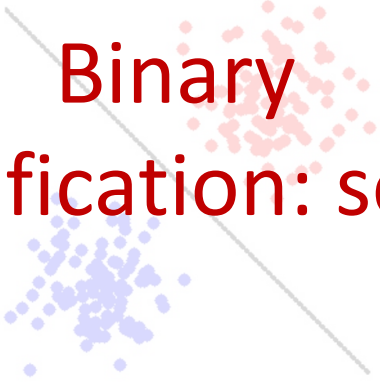
$\sigma(-z) = 1 - \sigma(z) = 1 - \frac{1}{1+e^{-z}} = \frac{e^{-z}}{1+e^{-z}}$

This is a *soft version of Perceptron!*

$\mathbb{P}(-y_i | \boldsymbol{x}_i; \boldsymbol{w})$ versus $\mathbb{I}[y_i \neq \text{sgn}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)]$



Binary classification: so far



Supervised learning in one slide

Loss function: What is the right loss function for the task?

Representation: What class of functions should we use?

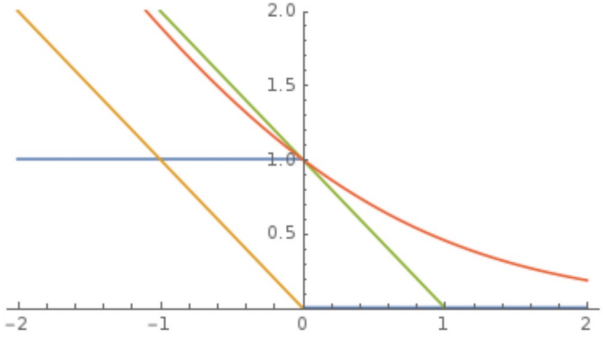
Optimization: How can we efficiently solve the empirical risk minimization problem?

Generalization: Will the predictions of our model transfer gracefully to unseen examples?

All related! And the fuel which powers everything is data.

Loss function

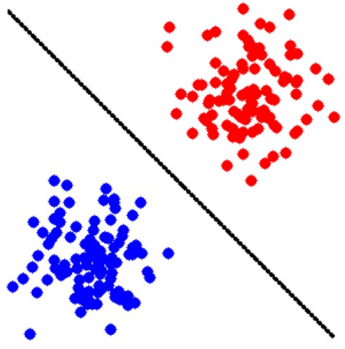
Use a **convex surrogate loss**



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$(used in SVM and many others)
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z))$ (used in logistic regression; the base of $\log$ doesn't matter)

Representation

Definition: The **function class of separating hyperplanes** is defined as $\mathcal{F} = \{f(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x}) : \boldsymbol{w} \in \mathbb{R}^d\}$.



Optimization

Empirical risk minimization (ERM) problem:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}\in\mathbb{R}^{\mathsf{d}}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)$$

Solve using a suitable optimization algorithm:

- **GD:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla F(\boldsymbol{w})$
- **SGD:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \tilde{\nabla} F(\boldsymbol{w})$ $\qquad (\mathbb{E}[\tilde{\nabla} F(\boldsymbol{w})] = \nabla F(\boldsymbol{w}))$
- **Newton:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \left(\nabla^2 F(\boldsymbol{w})\right)^{-1} \nabla F(\boldsymbol{w})$

Generalization

Rich theory! Let’s see a glimpse ☺

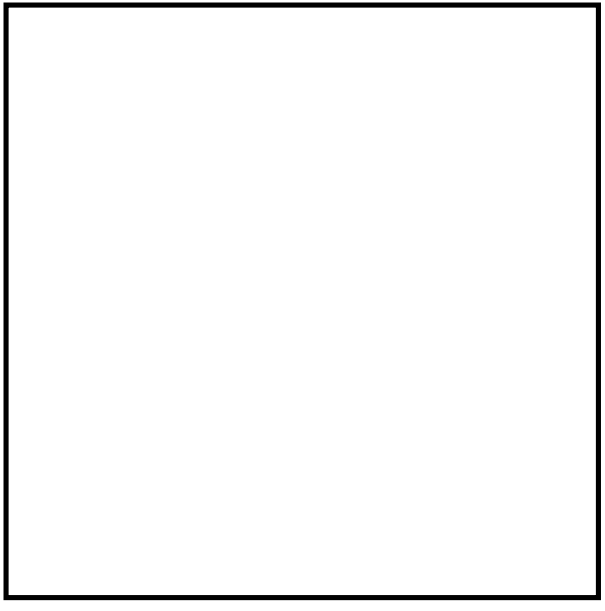
Generalization

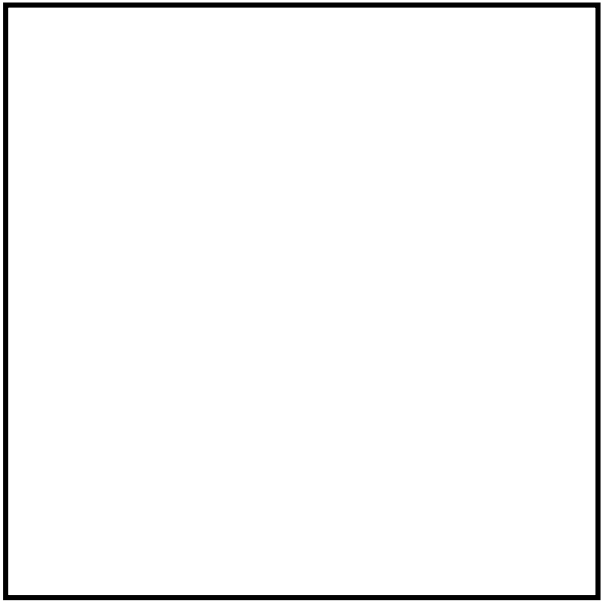
Reviewing definitions

- Input space: $\mathcal{X}$
- Output space: $\mathcal{Y}$
- Predictor: $f(\boldsymbol{x}) : \mathcal{X} \to \mathcal{Y}$
- Distribution D over $(\boldsymbol{x}, y)$.
- Let D^n denote the distribution of n samples $\{(\boldsymbol{x}_i, y_i), i \in [n]\}$ drawn i.i.d. from D.
- Risk of a predictor $f(\boldsymbol{x})$ is $R(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim D}\Big[\ell(f(\boldsymbol{x}), y)\Big]$
- Consider the 0-1 loss, $\ell(f(\boldsymbol{x}, y)) = \mathbb{1}(f(\boldsymbol{x}) \neq y)$.

Next time, we'll see some generalization theory!

Intuition: When does ERM generalize?





Relaxing our assumptions

- We assumed that the function class is finite-sized. Results can be extended to **infinite function classes** (such as separating hyperplanes).
- We considered 0-1 loss. Can extend to **real-valued loss** (such as for regression).
- We assumed realizability. Can prove similar theorem which guarantees small generalization gap **without realizability** (but with an ϵ^2 instead of ϵ in the denominator). This is called agnostic learning.

Rule of thumb for generalization

Suppose the functions f in our function class $\mathcal{F}$ have d parameters which can be set. Assume we discretize these parameters so they can take W possible values each. How much data do we need to have small generalization gap?

A useful rule of thumb: to guarantee generalization, make sure that your training data set size n is at least linear in the number d of free parameters in the function that you're trying to learn.