

CSCI 567: Machine Learning

Vatsal Sharan
Spring 2026

Lecture 2, Jan 23



USC University of
Southern California

Administrivia

- HW1 is out
- Due in about 3 weeks (2/11 midnight). **Start early!!!**
- Remember:
 - Point of the homeworks is for you to assimilate the concepts
 - Using AI could prevent you from doing that
 - Our goal with grading HWs is to give feedback, make use of it

Recap

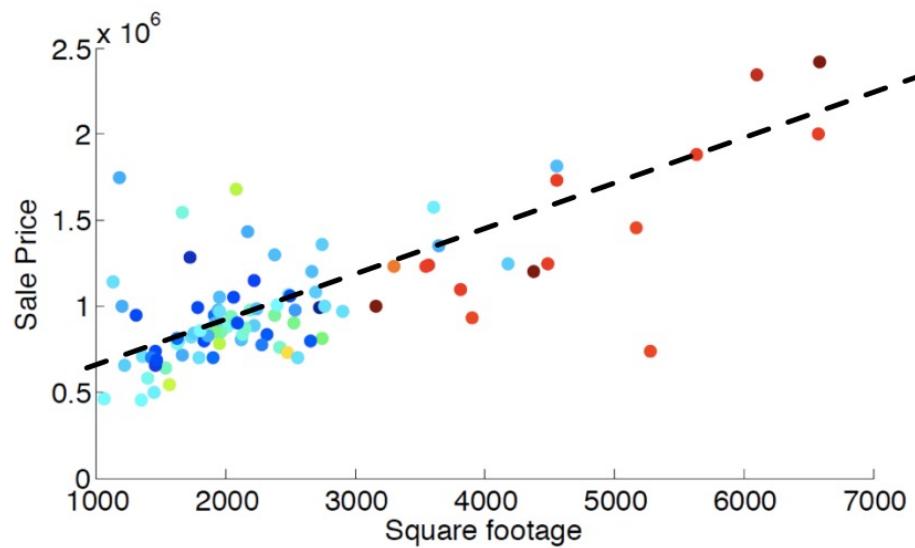
Supervised learning in one slide

- 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**

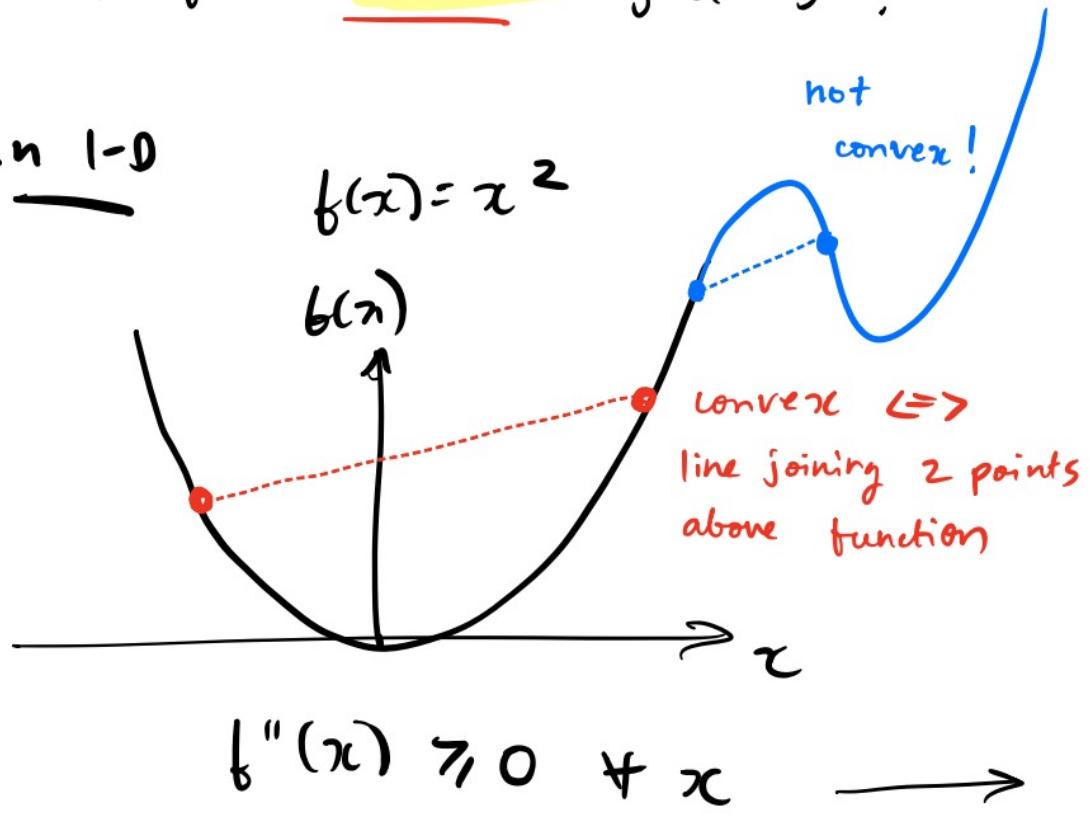


How to solve this? Find **stationary points**

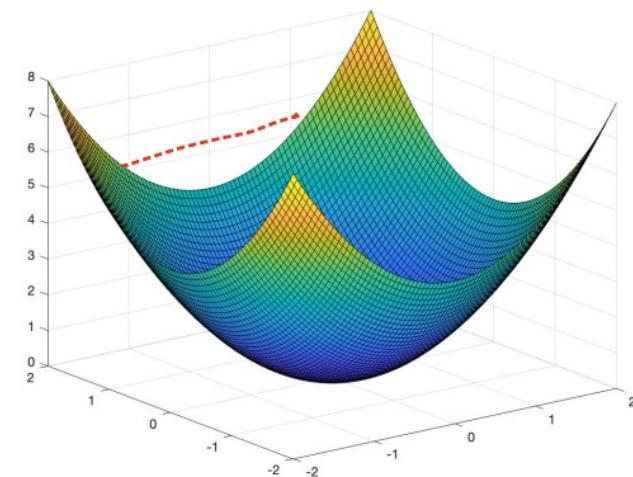
Are stationary points minimizers?

Yes, for convex objectives !

In 1-D



In high dimensions:



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

General least square solution

Objective

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

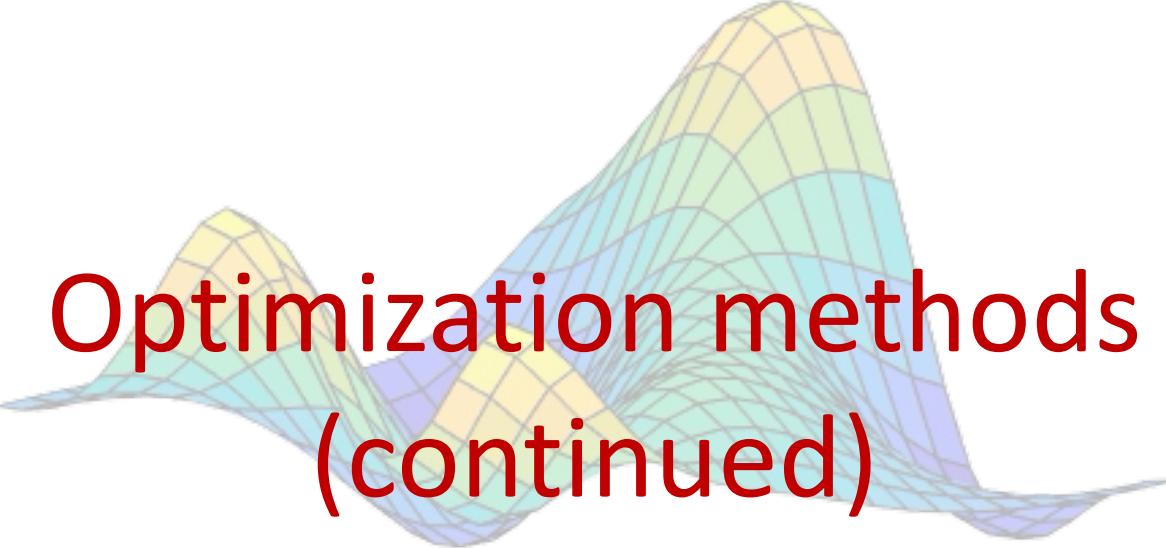
Find stationary points:

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

where

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

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



Optimization methods (continued)

Problem setup

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

Goal: minimize $F(\mathbf{w})$ (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 $\mathbf{w}^{(0)}$. For $t = 0, 1, 2, \dots$

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

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

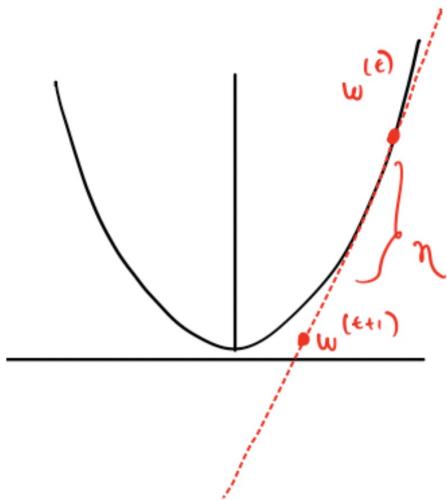
- in theory η should be set in terms of some parameters of F
- in practice we just try several small values
- might need to be changing over iterations (think $F(w) = |w|$)
- adaptive and automatic step size tuning is an active research area

Why GD?

Intuition: First-order Taylor approximation

$$f(w^{(t+1)}) \approx f(w^{(t)}) + \nabla f(w^{(t)})^T \underbrace{(w^{(t+1)} - w^{(t)})}_{-\eta \nabla F(w^{(t)})}$$

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



For $\mathbf{w} = \mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla F(\mathbf{w}^{(t)})$, we can write,

$$\begin{aligned} F(\mathbf{w}^{(t+1)}) &\approx F(\mathbf{w}^{(t)}) - \eta \|\nabla F(\mathbf{w}^{(t)})\|_2^2 \\ \implies F(\mathbf{w}^{(t+1)}) &\lesssim F(\mathbf{w}^{(t)}) \end{aligned}$$

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

Switch to Colab

optimization.ipynb ☆

File Edit View Insert Runtime Tools Help

+ Code + Text

```
# optimization.ipynb
# This notebook illustrates gradient descent for a two-dimensional objective function.
```

This code defines a gradient descent loop and a plot of the objective function. The plot shows contour lines and a red arrow indicating the direction of the gradient step.

```
    this_theta[1] = last_theta[1] - eta * grad1
    theta.append(this_theta)
    J.append(cost_func(*this_theta))

    # Annotate the objective function plot with coloured points indicating the
    # parameters chosen and red arrows indicating the steps down the gradient.
    for j in range(1,N):
        ax.annotate('', xy=theta[j], xytext=theta[j-1],
                    arrowprops={'arrowstyle': '->', 'color': 'orange', 'lw': 1},
                    va='center', ha='center')
    ax.scatter(*zip(*theta), facecolors='none', edgecolors='r', lw=1.5)

    # Labels, titles and a legend.
    ax.set_xlabel(r' $w_1$ ')
    ax.set_ylabel(r' $w_2$ ')
    ax.set_title('objective function')

plt.show()
```

objective function

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

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

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

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

Even for *nonconvex objectives*, some guarantees exist:

e.g. how many iterations t (in terms of ε) are needed to achieve

$$\|\nabla F(\mathbf{w}^{(t)})\| \leq \varepsilon$$

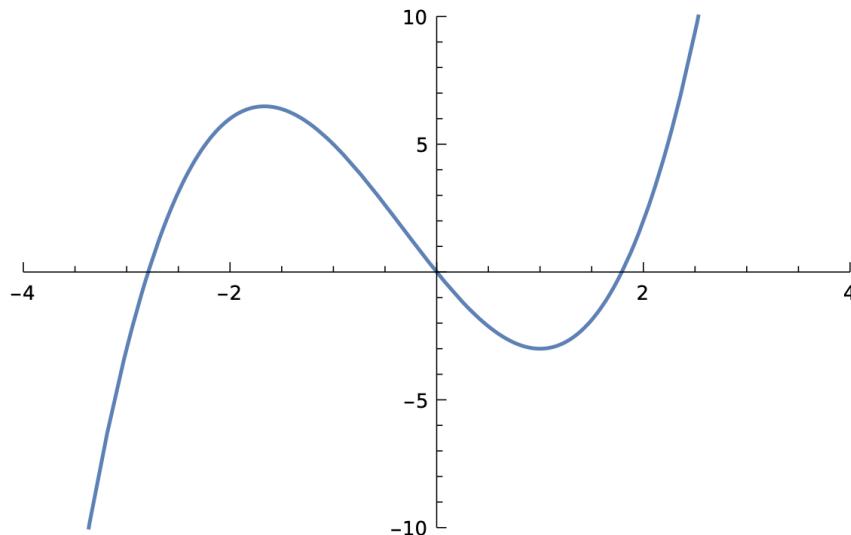
that is, how close is $\mathbf{w}^{(t)}$ as an approximate stationary point

for convex objectives, stationary point \Rightarrow 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).

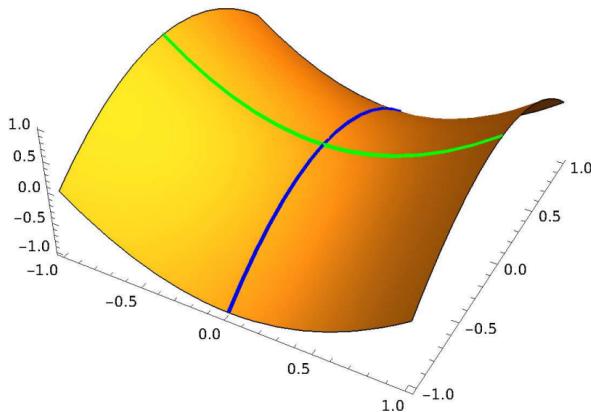


$$f(w) = w^3 + w^2 - 5w$$

Stationary points: non-convex objectives

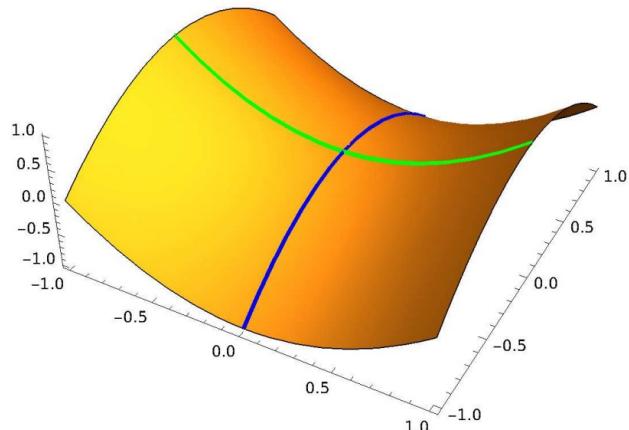
A stationary point can also be *neither a local minimizer nor a local maximizer!*

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



Stationary points: non-convex objectives

This is known as a saddle point

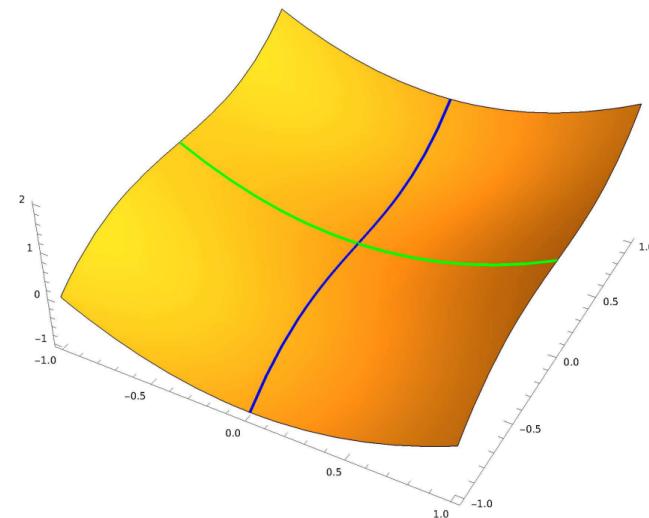


- but GD gets stuck at $(0,0)$ only if initialized along the **green direction**
- so not a real issue especially *when initialized randomly*

Stationary points: non-convex objectives

But not all saddle points look like a “saddle” ...

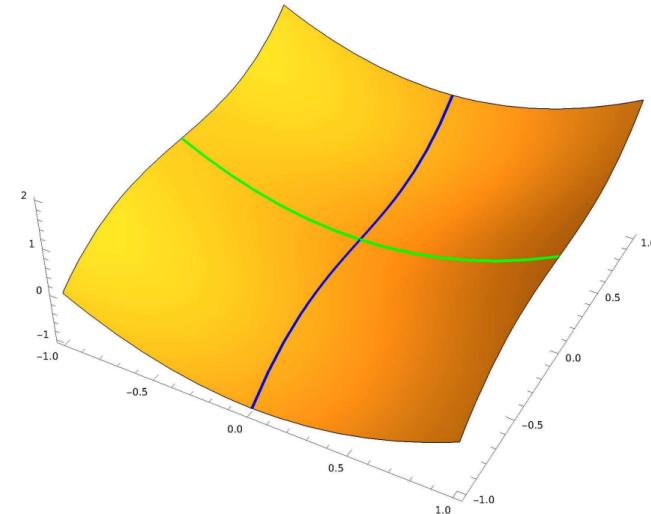
- $f(\mathbf{w}) = w_1^2 + w_2^3$
- $\nabla f(\mathbf{w}) = (2w_1, 3w_2^2)$
- so $\mathbf{w} = (0, 0)$ is stationary
- not local min/max for blue direction
($w_1 = 0$)



Stationary points: non-convex objectives

But not all saddle points look like a “saddle” ...

- $f(\mathbf{w}) = w_1^2 + w_2^3$
- $\nabla f(\mathbf{w}) = (2w_1, 3w_2^2)$
- so $\mathbf{w} = (0, 0)$ 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.

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

Stochastic Gradient descent

GD: keep moving in the *negative gradient direction*

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

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

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

$$\mathbb{E} [\tilde{\nabla} F(\mathbf{w}^{(t)})] = \nabla F(\mathbf{w}^{(t)}) \quad (\text{unbiasedness})$$

- 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.

Summary: Gradient descent & Stochastic Gradient descent

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

Summary: Gradient descent & Stochastic Gradient descent

- GD/SGD converges 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

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

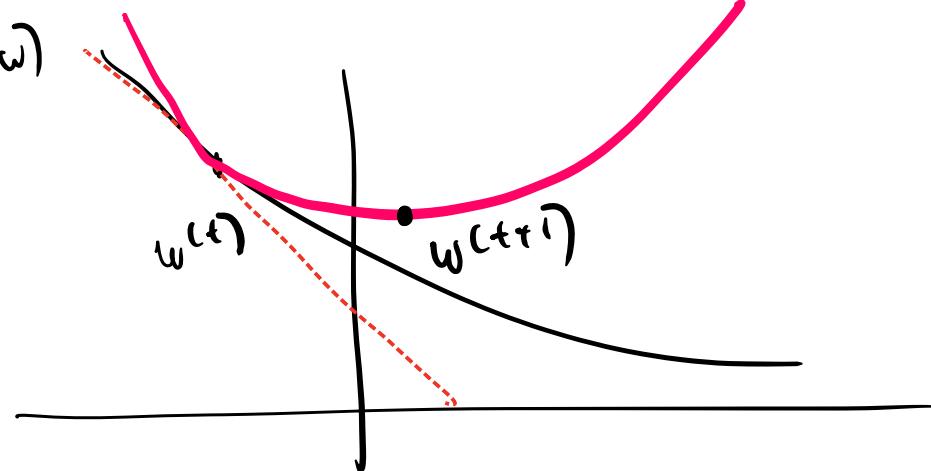
$$f(y) \approx f(x) + f'(x)(y-x) + \frac{f''(x)}{2}(y-x)^2$$

What about a second-order Taylor approximation?

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

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

$$(H_t)_{i,j} = \left. \frac{\partial^2 F}{\partial w_i \partial w_j} \right|_{w=w^{(t)}}$$



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

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

Exercise, show that:

$$\frac{dF(\mathbf{w}^{(t)})}{d\mathbf{w}} = 0$$

$$\frac{d}{d\mathbf{w}} \left(-\frac{1}{2} \mathbf{w}^\top \mathbf{H}_t \mathbf{w}^{(t)} \right) = -\frac{1}{2} \mathbf{H}_t \mathbf{w}^{(t)}$$

$$\frac{d}{d\mathbf{w}} \left(\nabla F(\mathbf{w}^{(t)})^\top (\mathbf{w} - \mathbf{w}^{(t)}) \right) = \nabla F(\mathbf{w}^{(t)})$$

$$\frac{d}{d\mathbf{w}} \left(-\frac{1}{2} (\mathbf{w}^{(t)})^\top \mathbf{H}_t \mathbf{w} \right) = -\frac{1}{2} \mathbf{H}_t \mathbf{w}^{(t)}$$

$$\frac{d}{d\mathbf{w}} \left(\frac{1}{2} \mathbf{w}^\top \mathbf{H}_t \mathbf{w} \right) = \mathbf{H}_t \mathbf{w}$$

$$\frac{d}{d\mathbf{w}} \left(\frac{1}{2} (\mathbf{w}^{(t)})^\top \mathbf{H}_t \mathbf{w}^{(t)} \right) = 0$$

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

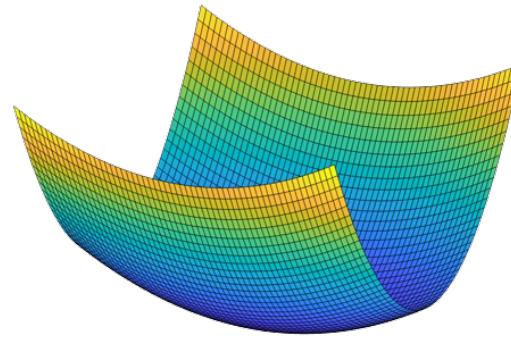
$$H_t w = H_t w^{(t)} \Rightarrow F(w^{(t)})$$

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

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

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

Newton's Method	Gradient Descent
No learning rate	Need to tune learning rate
Super fast convergence	Slower convergence
Know and invert Hessian (inversion takes $O(d^3)$ time naively)	Fast! (only takes $O(d)$ time)



If optimization objective is very flat along a certain direction, 2nd order methods maybe better

Linear classifiers



The Setup

Recall the setup:

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

This lecture: **binary classification**

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

Representation: Choosing the **function class**

Lets follow the recipe, and pick a function class \mathcal{F} .

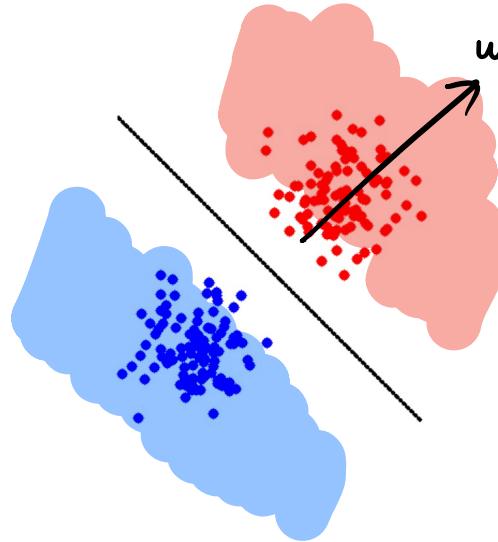
We continue with linear models, but how to predict a label using $\mathbf{w}^\top \mathbf{x}$?

Sign of $\mathbf{w}^\top \mathbf{x}$ predicts the label:

$$\text{sign}(\mathbf{w}^\top \mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}^\top \mathbf{x} > 0, \\ -1 & \text{if } \mathbf{w}^\top \mathbf{x} \leq 0. \end{cases}$$

(Sometimes use sgn for sign too.)

Representation: Choosing the **function class**

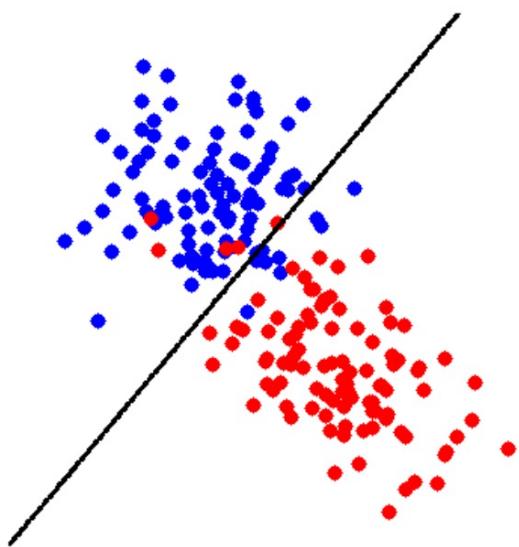


Good choice if
data is linearly
separated.

Def: The function class of separating hyperplanes is defined as :

$$\mathcal{F} = \{ f(x) = \text{sign}(w^T x); w \in \mathbb{R}^d \}$$

Still makes sense for “almost” linearly separable data



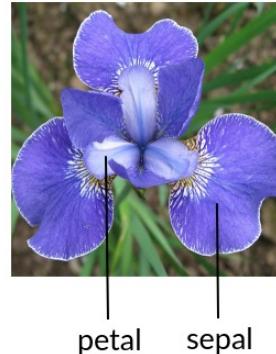
Iris dataset

One of the most classical and well-known datasets in machine learning.

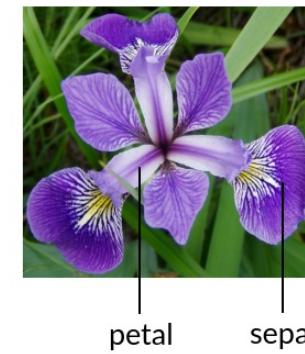
Collected by Edgar Anderson, first used by Ronald Fisher.

Anderson: “*There for mile after mile one could gather irises at will and assemble for comparison one hundred full-blown flowers of Iris versicolor and of Iris sentosa canadensis, each from a different plant, but all from the same pasture, and picked on the same day and measured by the same person with the same apparatus. The result is, to ordinary eyes, a few pages of singularly dry statistics, but to the bio-mathematician a juicy morsel quite worth looking ten years to find.*”

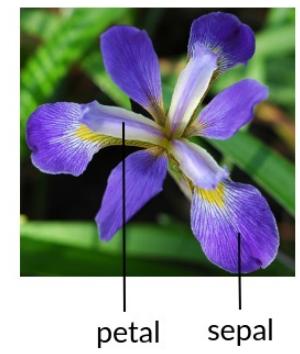
iris setosa



iris versicolor

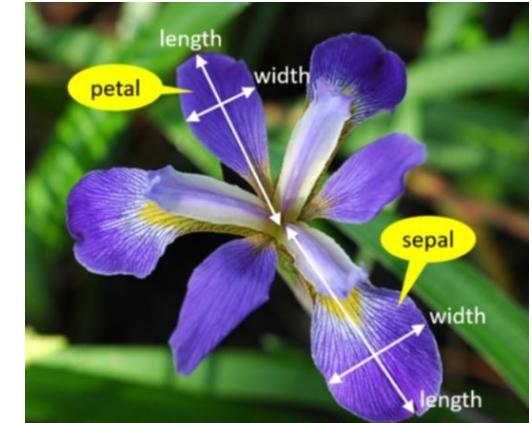


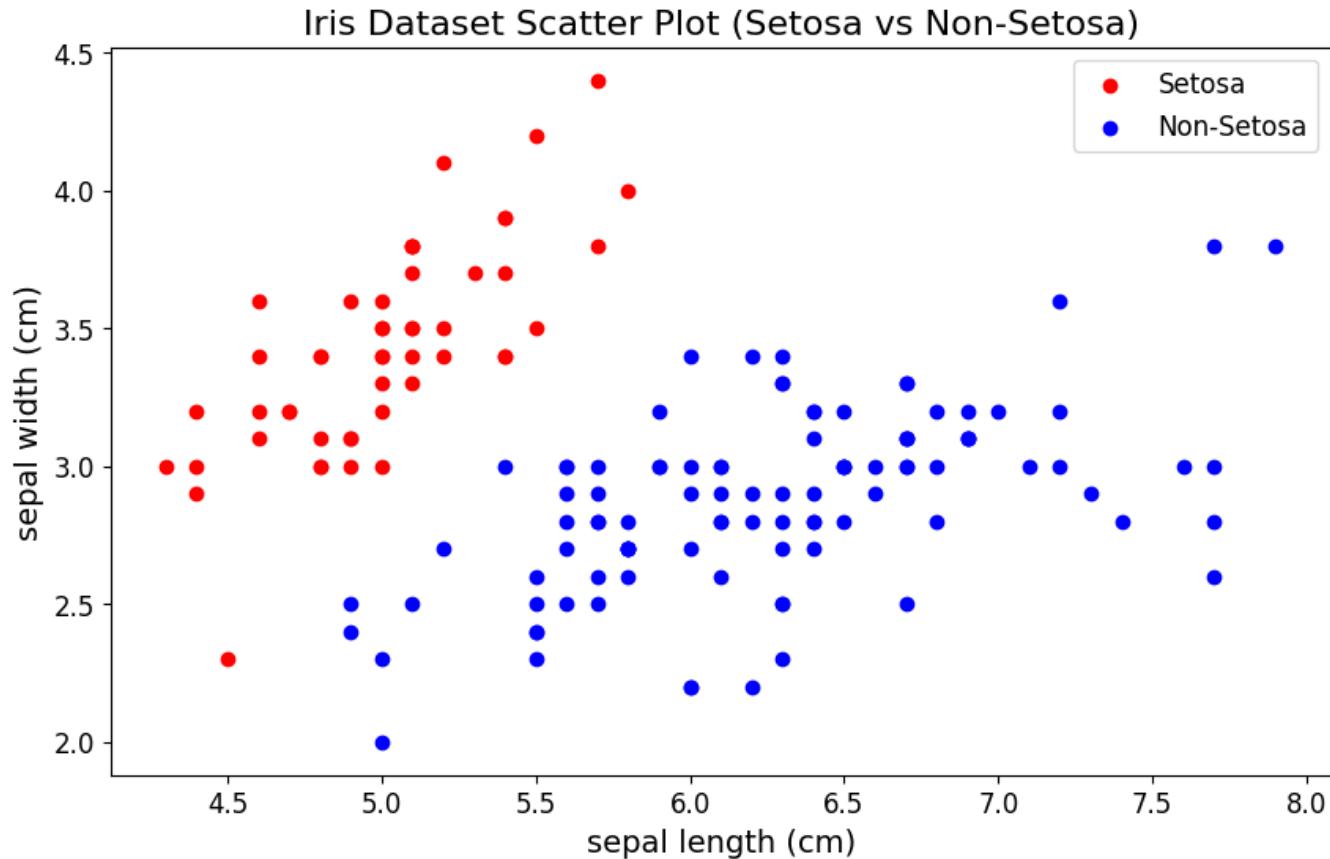
iris virginica



Features:

1. Sepal length
2. Sepal width



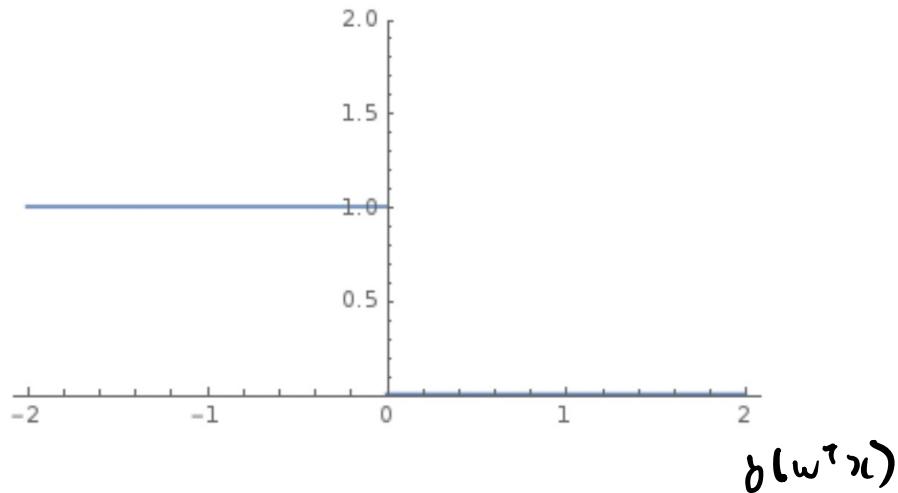


Choosing the loss function

Most common loss $\ell(f(x), y) = \mathbb{1}(f(x) \neq y) = \begin{cases} 1, & f(x) \neq y \\ 0, & \text{otw} \end{cases}$
Loss as a function of $y w^T x$

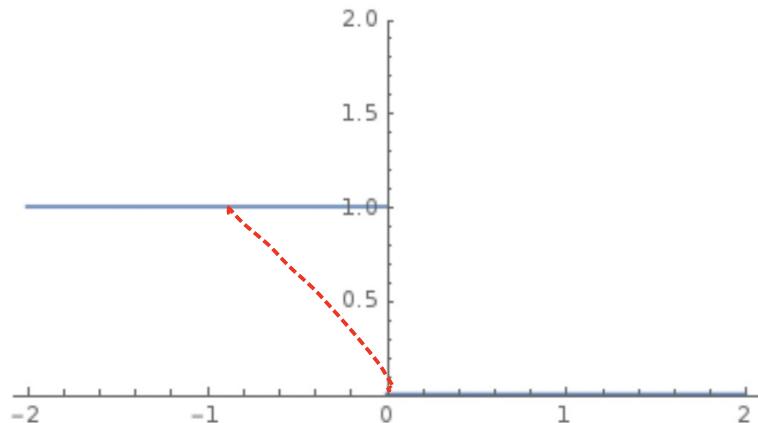
$$\ell_{0-1}(y w^T x) = \mathbb{1}(y w^T x \leq 0)$$

$$\ell_{0-1}(y w^T x)$$



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

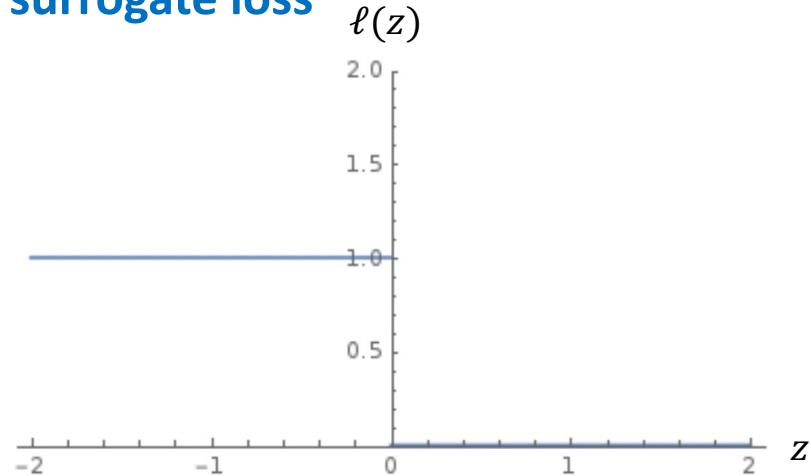
However, 0-1 loss is not convex.



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

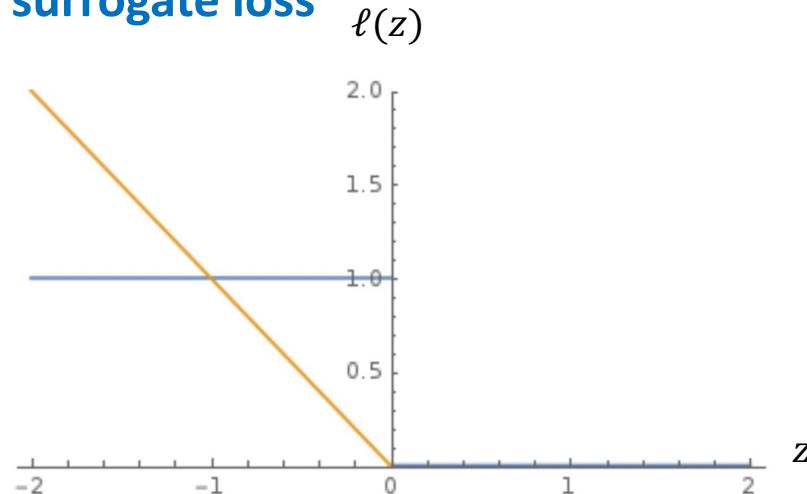
Choosing the loss function: **surrogate losses**

Solution: use a **convex surrogate loss**



Choosing the loss function: **surrogate losses**

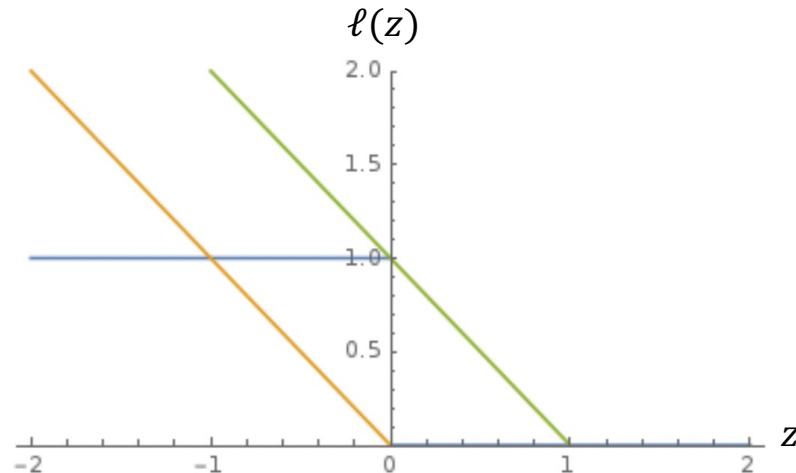
Solution: use a **convex surrogate loss**



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

Choosing the loss function: **surrogate losses**

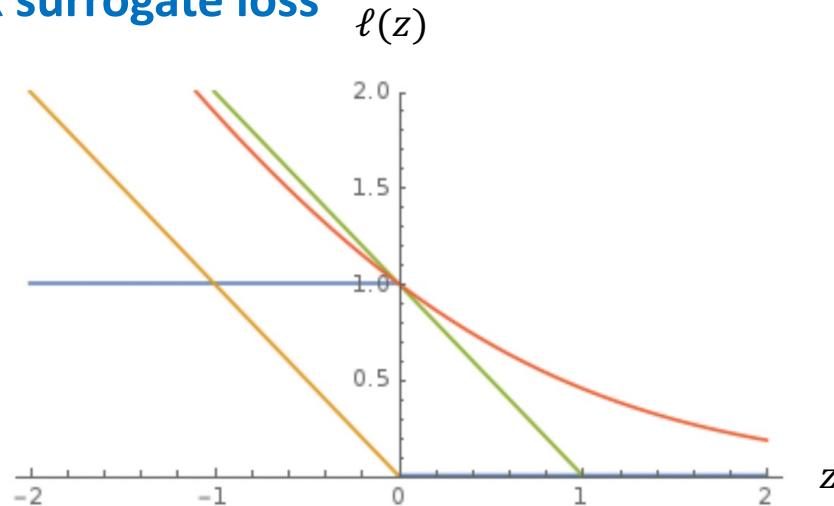
Solution: use a **convex surrogate loss**



- **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:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{n} \left(\sum_{i=1}^n \ell(y_i \mathbf{w}^\top \mathbf{x}_i) \right)$$

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

- 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

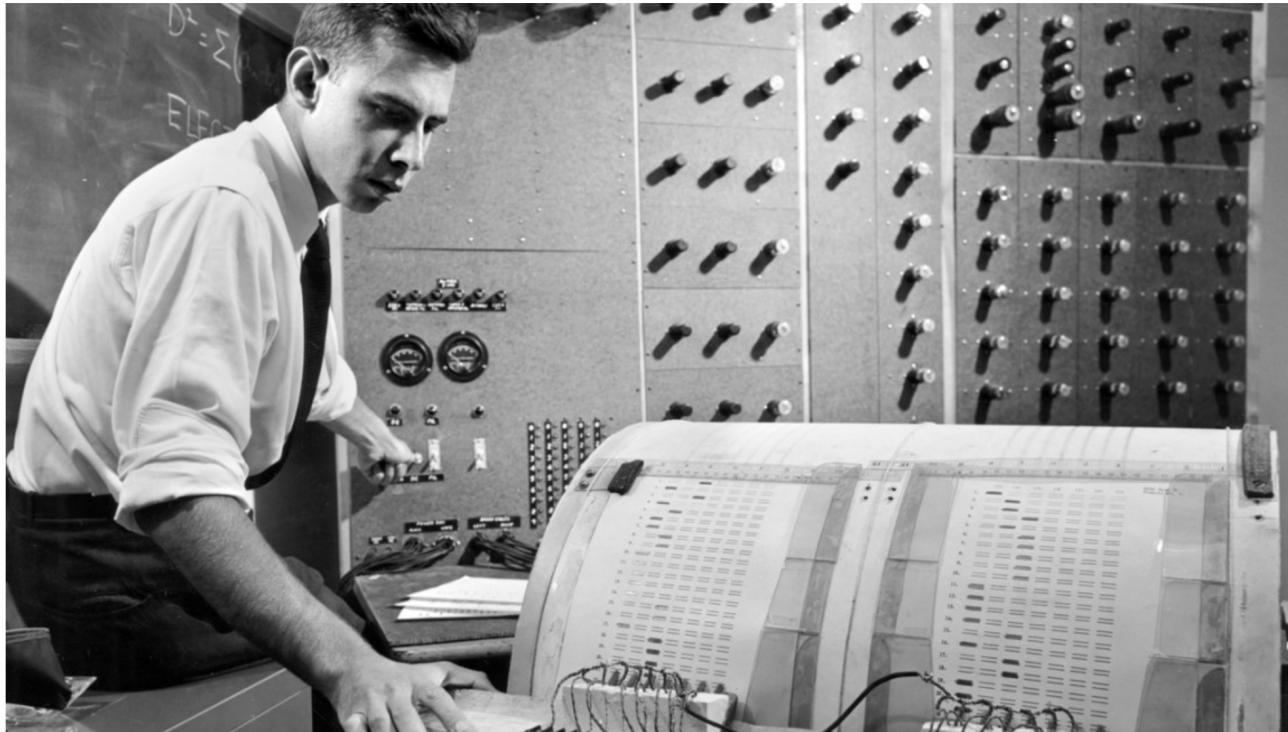
Psychologist Shows Embryo
of Computer Designed to
Read and Grow Wiser

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."*

The Perceptron



Frank Rosenblatt: Inventor of the perceptron

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

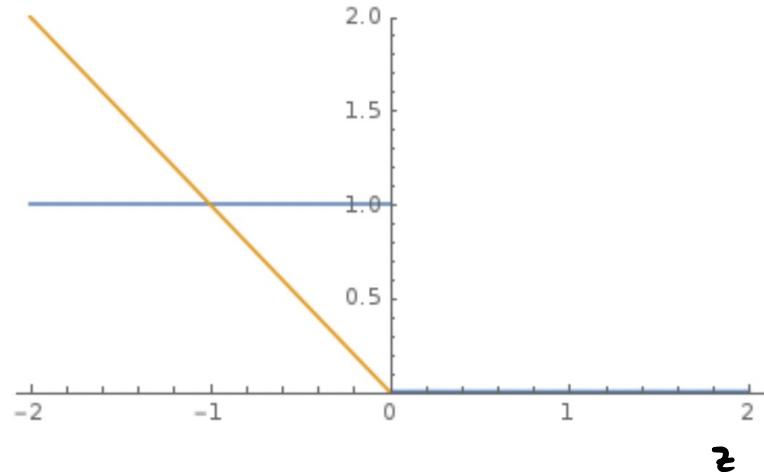
$$\begin{aligned} F(w) &= \frac{1}{n} \sum_{i=1}^n l_{\text{perceptr}}(y_i; w^\top x_i) \\ &= \frac{1}{n} \sum_{i=1}^n \max \{0, -y_i w^\top x_i\} \end{aligned}$$

What's the gradient?

$$0, \quad z > 0$$

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

$$l_{\text{perceptr}}(z)$$



Applying **GD** to perceptron loss

Gradient is

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

(only misclassified examples count)

$$GD : w_{\text{new}} = w_{\text{old}} + \frac{\eta}{n} \sum_{i=1}^n \mathbb{1}_{(y_i w^\top x_i \leq 0)} y_i x_i$$

Applying SGD to perceptron loss

How to get a stochastic gradient?

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

$$\tilde{\nabla} F(w^{(t)}) = -\mathbf{1}_{[y_i w^\top x_i \leq 0]} y_i x_i$$

Unbiased. why?

$$\begin{aligned} \mathbb{E}[\tilde{\nabla} F(w^{(t)})] &= \frac{1}{n} \sum_{i=1}^n -\mathbf{1}_{[y_i w^\top x_i \leq 0]} y_i x_i \\ &= \nabla F(w^{(t)}) \end{aligned}$$

SGD update: $w_{\text{new}} = w_{\text{old}} + \eta \mathbf{1}_{[y_i w^\top x_i \leq 0]} y_i x_i$

Perceptron algorithm

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

1. Initialize $\mathbf{w} = 0$
2. Repeat
 - Pick $\mathbf{x}_i \sim \text{Unif}(\mathbf{x}_1, \dots, \mathbf{x}_n)$
 - If $\text{sgn}(\mathbf{w}^T \mathbf{x}_i) \neq y_i$
$$\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$$

Perceptron algorithm: Intuition

Say that w makes mistake on (x_i, y_i)

$$y_i w^\top x_i < 0$$

Consider $w' = w + y_i x_i$

$$(w')^\top x = w^\top x + y_i x_i^\top x_i$$

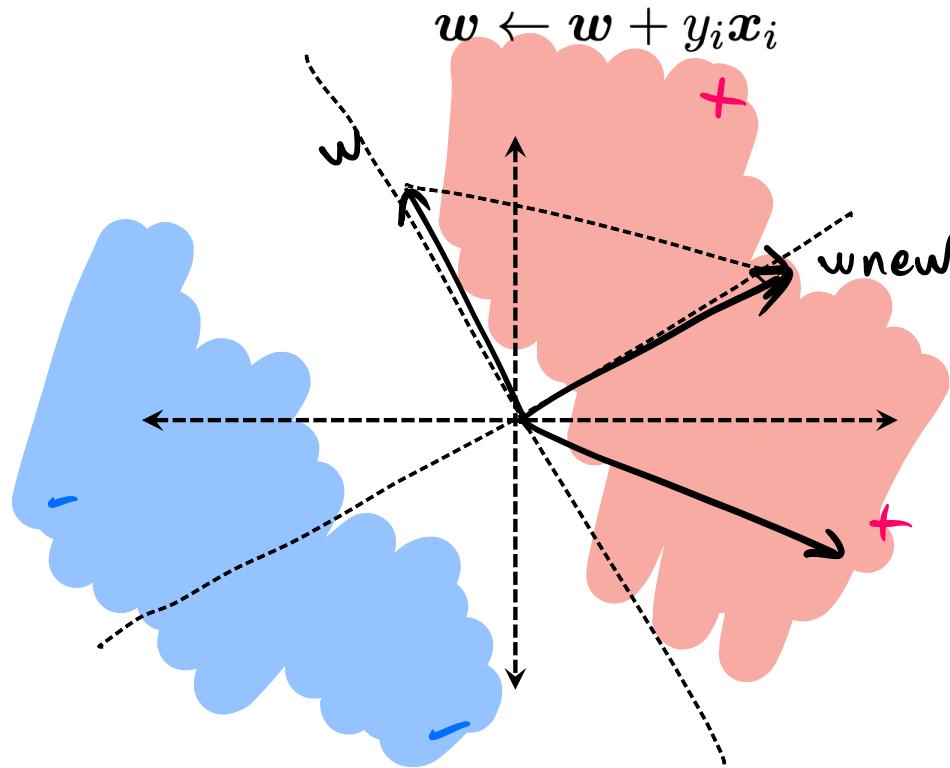
$$y_i (w')^\top x = y_i w^\top x + y_i^2 x_i^\top x_i$$

if $x_i \neq 0$, $y_i (w')^\top x > y_i w^\top x$

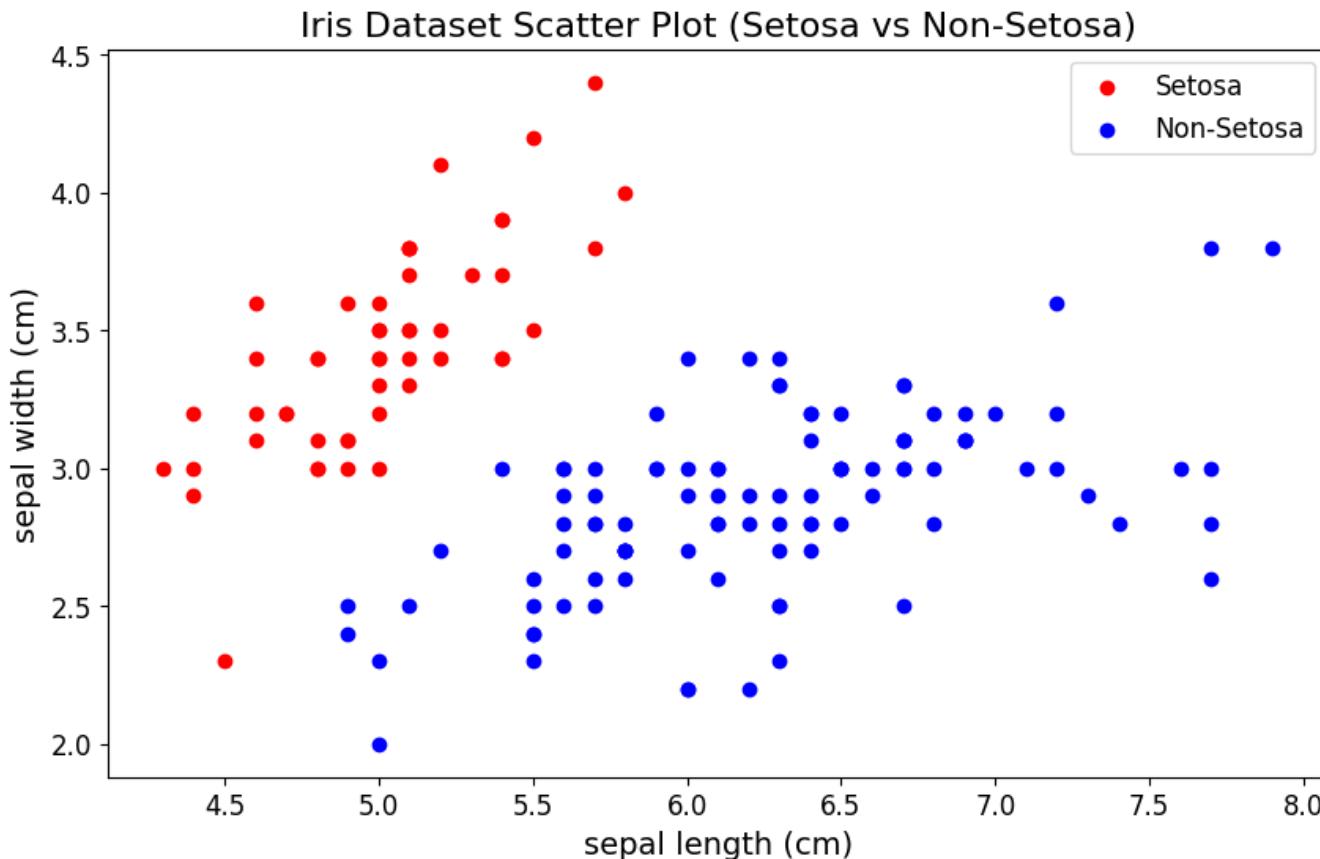
Perceptron algorithm: visually

Repeat:

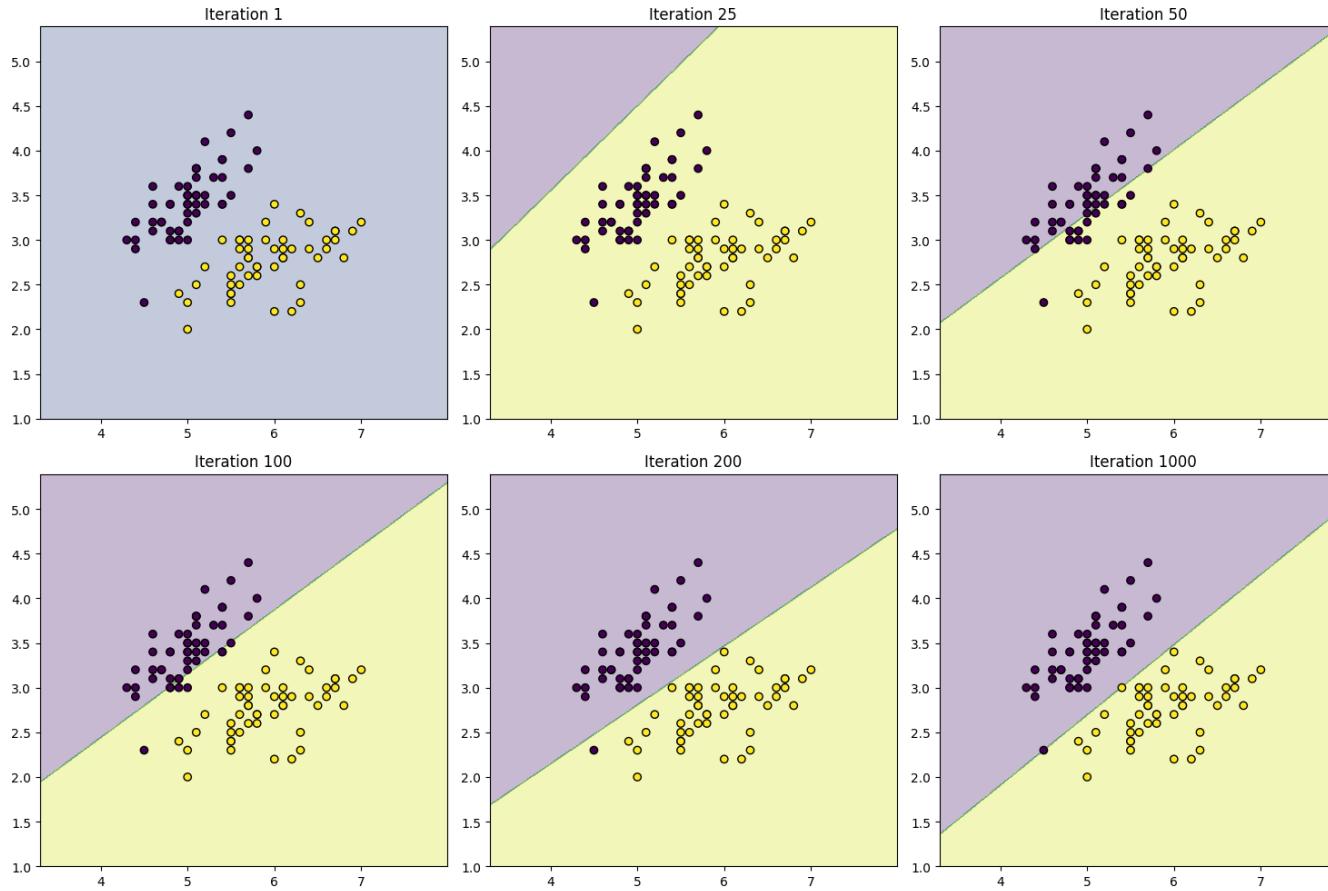
- Pick a data point x_i uniformly at random
- If $\text{sgn}(\mathbf{w}^T \mathbf{x}_i) \neq y_i$



Perceptron algorithm: Iris dataset



Perceptron algorithm: Iris dataset

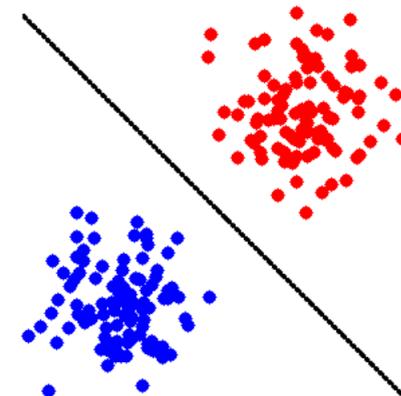


See demo in Google Colab

HW1: Theory for perceptron!

(HW 1) If training set is linearly separable

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



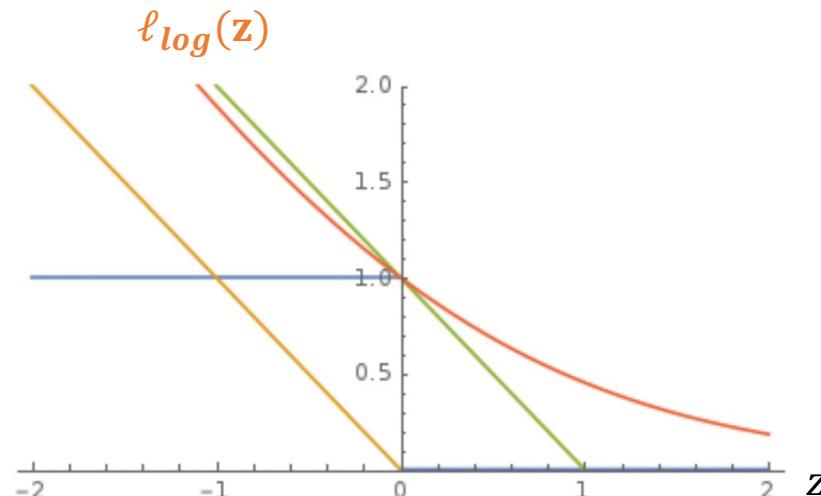
There are also guarantees when the data are not linearly separable.

Logistic regression



Logistic loss

$$F(w) = \frac{1}{n} \sum_{i=1}^n \ell_{\log}(y_i w^\top x_i)$$
$$= \frac{1}{n} \sum_{i=1}^n \log_e(1 + e^{-y_i w^\top x_i})$$



Predicting probabilities

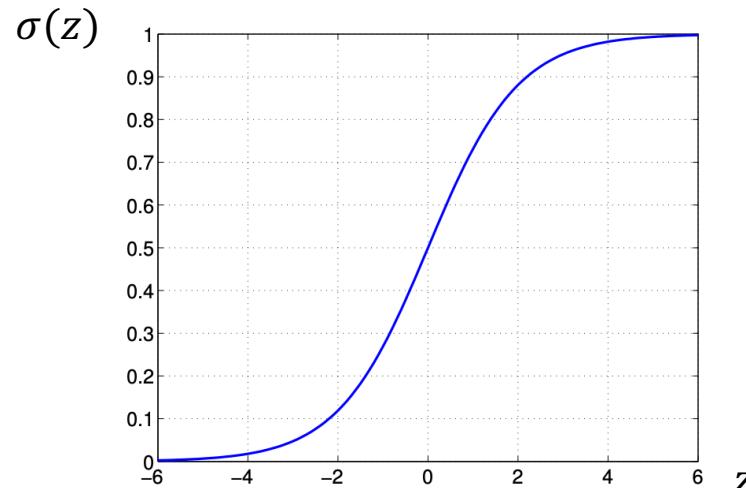
Instead of predicting the $\{\pm 1\}$ label, predict the probability (i.e. regression on probability).

Sigmoid + linear model:

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

where

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (\text{Sigmoid function})$$



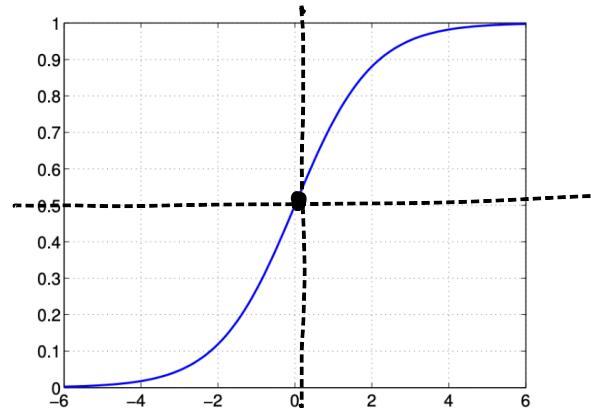
The sigmoid function

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

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

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

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



Maximum likelihood estimation

What we observe are labels, not probabilities.

Take a **probabilistic view**

- assume data is independently generated in this way by some w
- perform Maximum Likelihood Estimation (MLE)

Specifically, what is the probability of seeing labels y_1, \dots, y_n given x_1, \dots, x_n , as a function of some w ?

$$P(w) = \prod_{i=1}^n \mathbb{P}(y_i | x_i; w)$$

i.i.d assumption
all datapoints are independently & identically distributed

MLE: find w^* that **maximizes the probability** $P(w)$

Maximum likelihood solution

$$\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} P(\mathbf{w}) = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^n \mathbb{P}(y_i | \mathbf{x}_i; \mathbf{w})$$

$$= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^n \ln \mathbb{P}(y_i | \mathbf{x}_i; \mathbf{w})$$

$$= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n -\ln \mathbb{P}(y_i | \mathbf{x}_i; \mathbf{w})$$

$$\mathbb{P}(y_i | \mathbf{x}_i; \mathbf{w}) = \sigma(y_i \mathbf{w}^\top \mathbf{x}_i)$$

$$= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n \ln(1 + e^{-y_i \mathbf{w}^\top \mathbf{x}_i})$$

$$= \frac{1}{1 + e^{-y_i \mathbf{w}^\top \mathbf{x}_i}}$$

$$= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n \ell_{\text{logistic}}(y_i \mathbf{w}^\top \mathbf{x}_i)$$

$$= \operatorname{argmin}_{\mathbf{w}} F(\mathbf{w})$$

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

SGD to logistic loss

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \tilde{\nabla} F(\mathbf{w})$$

$$= \mathbf{w} - \eta \nabla_{\mathbf{w}} \ell_{\text{logistic}}(y_i \mathbf{w}^T \mathbf{x}_i)$$

$$= \mathbf{w} - \eta \left(\frac{\partial \ell_{\text{logistic}}(z)}{\partial z} \Big|_{z=y_i \mathbf{w}^T \mathbf{x}_i} \right) y_i \mathbf{x}_i \quad (\text{chain rule})$$

$$= \mathbf{w} - \eta \left(\frac{-e^{-z}}{1 + e^{-z}} \Big|_{z=y_i \mathbf{w}^T \mathbf{x}_i} \right) y_i \mathbf{x}_i$$

$$= \mathbf{w} + \eta \sigma(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \mathbf{x}_i$$

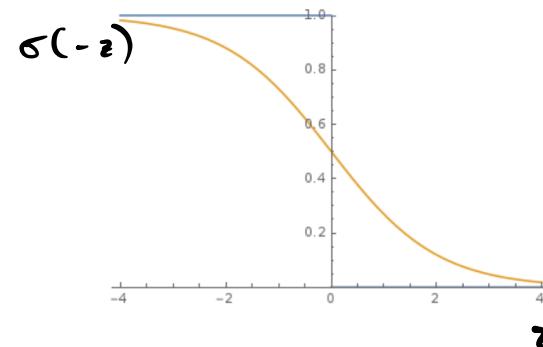
$$= \mathbf{w} + \eta \mathbb{P}(-y_i | \mathbf{x}_i; \mathbf{w}) y_i \mathbf{x}_i$$

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

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

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

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



This is a *soft version of Perceptron!*

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