Midterm review

CS 446
1. Lecture review
(Lec1.) Basic setting: supervised learning

Training data: labeled examples

\[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\]
(Lec1.) Basic setting: supervised learning

**Training data:** labeled examples

\[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\]

where

- each input \(x_i\) is a machine-readable description of an instance (e.g., image, sentence), and
Basic setting: supervised learning

**Training data:** labeled examples

\[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\]

where

- each input \(x_i\) is a machine-readable description of an instance (e.g., image, sentence), and
- each corresponding label \(y_i\) is an annotation relevant to the task—typically not easy to automatically obtain.
(Lec1.) Basic setting: supervised learning

**Training data:** labeled examples

\[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\]

where

- each **input** \(x_i\) is a machine-readable description of an instance (e.g., image, sentence), and
- each corresponding **label** \(y_i\) is an annotation relevant to the task—typically not easy to automatically obtain.

**Goal:** learn a function \(\hat{f}\) from labeled examples, that accurately “predicts” the labels of **new (previously unseen)** inputs. (**Note:** 0 training error is easy; test/population error is what matters.)

```
new (unlabeled) example
```

```
past labeled examples → learning algorithm → learned predictor
```

```
predicted label
```
(Lec2.) $k$-nearest neighbors classifier

**Given:** labeled examples $D := \{(x_i, y_i)\}_{i=1}^n$

**Predictor:** $\hat{f}_{D,k} : \mathcal{X} \rightarrow \mathcal{Y}$:

On input $x$,

1. Find the $k$ points $x_{i1}, x_{i2}, \ldots, x_{ik}$ among $\{x_i\}_{i=1}^n$ “closest” to $x$ (the $k$ nearest neighbors).
2. Return the plurality of $y_{i1}, y_{i2}, \ldots, y_{ik}$.

(Break ties in both steps arbitrarily.)
(Lec2.) Choosing $k$

The hold-out set approach

1. Pick a subset $V \subset S$ (hold-out set, a.k.a. validation set).
2. For each $k \in \{1, 3, 5, \ldots \}$:
   - Construct $k$-NN classifier $\hat{f}_{S \setminus V, k}$ using $S \setminus V$.
   - Compute error rate of $\hat{f}_{S \setminus V, k}$ on $V$ ("hold-out error rate").
3. Pick the $k$ that gives the smallest hold-out error rate.
The hold-out set approach

1. Pick a subset $V \subset S$ (hold-out set, a.k.a. validation set).
2. For each $k \in \{1, 3, 5, \ldots \}$:
   - Construct $k$-NN classifier $\hat{f}_{S \setminus V, k}$ using $S \setminus V$.
   - Compute error rate of $\hat{f}_{S \setminus V, k}$ on $V$ ("hold-out error rate").
3. Pick the $k$ that gives the smallest hold-out error rate.

(There are many other approaches.)
Directly optimize tree structure for good classification.

A decision tree is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, represented by a binary tree in which:

- Each tree node is associated with a splitting rule $g : \mathcal{X} \rightarrow \{0, 1\}$.
- Each leaf node is associated with a label $\hat{y} \in \mathcal{Y}$.
(Lec2.) Decision trees

Directly optimize tree structure for good classification.

A decision tree is a function \( f : \mathcal{X} \rightarrow \mathcal{Y} \), represented by a binary tree in which:

- Each tree node is associated with a splitting rule \( g : \mathcal{X} \rightarrow \{0, 1\} \).
- Each leaf node is associated with a label \( \hat{y} \in \mathcal{Y} \).

When \( \mathcal{X} = \mathbb{R}^d \), typically only consider splitting rules of the form

\[
g(\mathbf{x}) = 1 \{ x_i > t \}
\]

for some \( i \in [d] \) and \( t \in \mathbb{R} \).

Called axis-aligned or coordinate splits.

(Notation: \( [d] := \{1, 2, \ldots, d\} \).)
(Lec2.) Decision tree example

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

Decision tree example
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

$$x_1 > 1.7$$

$\hat{y} = 1$  $\hat{y} = 3$
(Lec2.) Decision tree example

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 = \text{ratio of sepal length to width}$
- $x_2 = \text{ratio of petal length to width}$

Decision tree:

1. $x_1 > 1.7$
2. $x_2 > 2.8$
3. $\hat{y} = 1$
Classifying irises by sepal and petal measurements

- \( \mathcal{X} = \mathbb{R}^2 \), \( \mathcal{Y} = \{1, 2, 3\} \)
- \( x_1 = \) ratio of sepal length to width
- \( x_2 = \) ratio of petal length to width

Decision tree example:

- \( x_1 > 1.7 \)
  - \( \hat{y} = 1 \)
  - \( x_2 > 2.8 \)
    - \( \hat{y} = 2 \)
    - \( \hat{y} = 3 \)
Today we covered two standard machine learning methods.

**Nearest neighbors.**
*Training/fitting*: memorize data.
*Testing/predicting*: find $k$ closest memorized points, return plurality label.
*Overfitting?* Vary $k$.

**Decision trees.**
*Training/fitting*: greedily partition space, reducing “uncertainty”.
*Testing/predicting*: traverse tree, output leaf label.
*Overfitting?* Limit or prune tree.

**Note**: both methods can output real numbers (*regression*, not *classification*); return median/mean of \{ neighbors, points reaching leaf \}. 
ERM setup for least squares.

- **Predictors/model**: \( \hat{f}(x) = w^T x; \)  
  a linear predictor/regressor.  
  (For linear classification: \( x \mapsto \text{sgn}(w^T x). \))

- **Loss/penalty**: the least squares loss  
  \[ \ell_{ls}(y, \hat{y}) = (y - \hat{y})^2. \]
  (Some conventions scale this by \( 1/2 \).)

- **Goal**: minimize least squares empirical risk  
  \[ \hat{R}_{ls}(\hat{f}) = \frac{1}{n} \sum_{i=1}^{n} \ell_{ls}(y_i, \hat{f}(x_i)) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2. \]

- Specifically, we choose \( w \in \mathbb{R}^d \) according to  
  \[ \arg \min_{w \in \mathbb{R}^d} \hat{R}_{ls}(x \mapsto w^T x) = \arg \min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^T x_i)^2. \]

- More generally, this is the ERM approach:  
  pick a model and minimize empirical risk over the model parameters.
ERM in general

- Pick a family of models/predictors $\mathcal{F}$.
  (For today, we use linear predictors.)
- Pick a loss function $\ell$.
  (For today, we chose squared loss.)
- Minimize the empirical risk over the model parameters.

We haven’t discussed: true risk and overfitting; how to minimize; why this is a good idea.
ERM in general

- Pick a family of models/predictors $\mathcal{F}$.
  (For today, we use linear predictors.)
- Pick a loss function $\ell$.
  (For today, we chose squared loss.)
- Minimize the empirical risk over the model parameters.

We haven't discussed: *true* risk and overfitting; *how* to minimize; *why* this is a good idea.

**Remark:** ERM is convenient in pytorch, just pick a model, a loss, an optimizer, and tell it to minimize.
Define $n \times d$ matrix $A$ and $n \times 1$ column vector $b$ by

$$A := \frac{1}{\sqrt{n}} \begin{bmatrix} \leftarrow & x_1^T & \rightarrow \\ \vdots \\ \leftarrow & x_n^T & \rightarrow \end{bmatrix}, \quad b := \frac{1}{\sqrt{n}} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}.$$
Define \( n \times d \) matrix \( A \) and \( n \times 1 \) column vector \( b \) by

\[
A := \frac{1}{\sqrt{n}} \begin{bmatrix}
\leftarrow x_1^T \rightarrow \\
\vdots \\
\leftarrow x_n^T \rightarrow 
\end{bmatrix}, \quad b := \frac{1}{\sqrt{n}} \begin{bmatrix}
y_1 \\
\vdots \\
y_n
\end{bmatrix}.
\]

Can write empirical risk as

\[
\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - x_i^T w \right)^2 = \| Aw - b \|_2^2.
\]
Define $n \times d$ matrix $A$ and $n \times 1$ column vector $b$ by

$$A := \frac{1}{\sqrt{n}} \begin{bmatrix} x_1^\top & \rightarrow & \vdots & \vdots & \leftarrow \ x_n^\top & \rightarrow \end{bmatrix}, \quad b := \frac{1}{\sqrt{n}} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}.$$ 

Can write empirical risk as

$$\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - x_i^\top w \right)^2 = \| A w - b \|_2^2.$$ 

Necessary condition for $w$ to be a minimizer of $\hat{R}$:

$$\nabla \hat{R}(w) = 0, \quad \text{i.e., } w \text{ is a critical point of } \hat{R}.$$
Define $n \times d$ matrix $A$ and $n \times 1$ column vector $b$ by

$$A := \frac{1}{\sqrt{n}} \begin{bmatrix} x_1^T \rightarrow \\ \vdots \\ x_n^T \rightarrow \end{bmatrix}, \quad b := \frac{1}{\sqrt{n}} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}.$$  

Can write empirical risk as

$$\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - x_i^T w \right)^2 = \| Aw - b \|_2^2.$$

Necessary condition for $w$ to be a minimizer of $\hat{R}$:

$$\nabla \hat{R}(w) = 0,$$

i.e., $w$ is a critical point of $\hat{R}$.

This translates to

$$(A^T A)w = A^T b,$$

a system of linear equations called the normal equations.
Define \( n \times d \) matrix \( A \) and \( n \times 1 \) column vector \( b \) by

\[
A := \frac{1}{\sqrt{n}} \begin{bmatrix}
\leftarrow x_1^\top \rightarrow \\
\vdots \\
\leftarrow x_n^\top \rightarrow 
\end{bmatrix}, \quad b := \frac{1}{\sqrt{n}} \begin{bmatrix}
y_1 \\
\vdots \\
y_n
\end{bmatrix}.
\]

Can write empirical risk as

\[
\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \left( y_i - x_i^\top w \right)^2 = \| A w - b \|_2^2.
\]

Necessary condition for \( w \) to be a minimizer of \( \hat{R} \):

\[
\nabla \hat{R}(w) = 0, \quad \text{i.e., } w \text{ is a critical point of } \hat{R}.
\]

This translates to

\[
(A^\top A)w = A^\top b,
\]

a system of linear equations called the normal equations.

In upcoming lecture we’ll prove every critical point of \( \hat{R} \) is a minimizer of \( \hat{R} \).
Given $M \in \mathbb{R}^{n \times d}$, let $M = USV^T$ denote the \textit{singular value decomposition} (SVD), where

- $U \in \mathbb{R}^{n \times n}$ is orthonormal, thus $U^TU = UU^T = I$,
- $V \in \mathbb{R}^{d \times d}$ is orthonormal, thus $V^TV = VV^T = I$,
- $S \in \mathbb{R}^{n \times d}$ has \textit{singular values} $s_1 \geq s_2 \geq \cdots \geq s_{\min(n,d)}$ along the diagonal and zeros elsewhere, where the number of positive singular values equals the rank of $M$.

Some facts:

- SVD is \textit{not unique} when the singular values are not distinct; e.g., we can write $I = UIV^T$ where $U$ is any orthonormal matrix.
- Pseudoinverse $S^+ \in \mathbb{R}^{d \times n}$ of $S$ is obtained by starting with $S^T$ and taking the reciprocal of each positive entry.
- Pseudoinverse of $M$ is $VS^+U^T$.
- If $M^{-1}$ exists, then $M^{-1} = M^+$. 

(Lec3-4.) Full (factorization) SVD (new slide)
Given $M \in \mathbb{R}^{n \times d}$, $(s, u, v)$ are a singular value with corresponding left and right singular vectors if $Mv = su$ and $M^Tu = sv$.

The thin SVD of $M$ is $M = \sum_{i=1}^{r} s_i u_i v_i^T$, where $r$ is the rank of $M$, and

- left singular vectors $(u_1, \ldots, u_r)$ are orthonormal (but we might have $r < \min\{n, d\}$) and span the column space of $M$,
- right singular vectors $(v_1, \ldots, v_r)$ are orthonormal (but we might have $r < \min\{n, d\}$) and span the row space of $M$,
- singular values $s_1 \geq \cdots \geq s_r > 0$.

Some facts:

- Pseudoinverse $M^+ = \sum_{i=1}^{r} \frac{1}{s_i} v_i u_i^T$.
- $(u_i)_{i=1}^{r}$ span t
Recall: we’d like to find \( w \) such that

\[
A^T A w = A^T b.
\]

If \( w = A^+ b \), then

\[
A^T A w = \left( \sum_{i=1}^{r} s_i v_i u_i^T \right) \left( \sum_{i=1}^{r} s_i u_i v_i^T \right) \left( \sum_{i=1}^{r} \frac{1}{s_i} v_i u_i^T \right) b
\]

\[
= \left( \sum_{i=1}^{r} s_i v_i u_i^T \right) \left( \sum_{i=1}^{r} u_i u_i^T \right) b = A^T b.
\]

Henceforth, define \( \hat{w}_{ols} = A^+ b \) as the OLS solution.
(OLS = “ordinary least squares”.)

**Note:** in general, \( A A^+ = \sum_{i=1}^{r} u_i u_i^T \neq I \).
Normal equations imply optimality

Consider $w$ with $A^T A w = A^T y$, and any $w'$; then

$$
\|A w' - y\|^2 = \|A w' - A w + A w - y\|^2 \\
= \|A w' - A w\|^2 + 2(A w' - A w)^T(A w - y) + \|A w - y\|^2.
$$

Since

$$(A w' - A w)^T(A w - y) = (w' - w)^T(A^T A w - A^T y) = 0,$$

then $\|A w' - y\|^2 = \|A w' - A w\|^2 + \|A w - y\|^2$. This means $w'$ is optimal.
Consider $w$ with $A^T Aw = A^T y$, and any $w'$; then
\[
\|Aw' - y\|^2 = \|Aw' - Aw + Aw - y\|^2
= \|Aw' - Aw\|^2 + 2(Aw' - Aw)^T (Aw - y) + \|Aw - y\|^2.
\]

Since
\[
(Aw' - Aw)^T (Aw - y) = (w' - w)^T (A^T Aw - A^T y) = 0,
\]
then \(\|Aw' - y\|^2 = \|Aw' - Aw\|^2 + \|Aw - y\|^2\). This means $w'$ is optimal.

Moreover, writing $A = \sum_{i=1}^r s_i u_i v_i^T$,
\[
\|Aw' - Aw\|^2 = (w' - w)^T (A^T A)(w' - w) = (w' - w)^T \left( \sum_{i=1}^r s_i^2 v_i v_i^T \right) (w' - w),
\]
so $w'$ optimal iff $w' - w$ is in the right nullspace of $A$. 

(We'll revisit all this with convexity later.)
(Lec3-4.) Normal equations imply optimality

Consider $w$ with $A^T A w = A^T y$, and any $w'$; then

$$
\| A w' - y \|^2 = \| A w' - A w + A w - y \|^2 \\
= \| A w' - A w \|^2 + 2 (A w' - A w)^T (A w - y) + \| A w - y \|^2.
$$

Since

$$
(A w' - A w)^T (A w - y) = (w' - w)^T (A^T A w - A^T y) = 0,
$$

then $\| A w' - y \|^2 = \| A w' - A w \|^2 + \| A w - y \|^2$. This means $w'$ is optimal.

Moreover, writing $A = \sum_{i=1}^{r} s_i u_i v_i^T$,

$$
\| A w' - A w \|^2 = (w' - w)^T (A^T A) (w' - w) = (w' - w)^T \left( \sum_{i=1}^{r} s_i^2 v_i v_i^T \right) (w' - w),
$$

so $w'$ optimal iff $w' - w$ is in the right nullspace of $A$.

(We’ll revisit all this with convexity later.)
Combine the two concerns: For a given $\lambda \geq 0$, find minimizer of

$$\hat{R}(w) + \lambda \|w\|_2^2$$

over $w \in \mathbb{R}^d$. 

Fact: If $\lambda > 0$, then the solution is always unique (even if $n < d$)!

This is called ridge regression. ($\lambda = 0$ is ERM / Ordinary Least Squares.)

Explicit solution: $(A^TA + \lambda I)^{-1}A^Tb$.

Parameter $\lambda$ controls how much attention is paid to the regularizer $\|w\|_2^2$ relative to the data fitting term $\hat{R}(w)$.

Choose $\lambda$ using cross-validation.

Note: in deep networks, this regularization is called "weight decay". (Why?)

Note: another popular regularizer for linear regression is $\ell_1$. 

Combine the two concerns: For a given $\lambda \geq 0$, find minimizer of

$$\hat{R}(w) + \lambda \|w\|_2^2$$

over $w \in \mathbb{R}^d$.

Fact: If $\lambda > 0$, then the solution is always unique (even if $n < d$)!
Combine the two concerns: For a given $\lambda \geq 0$, find minimizer of

$$\hat{R}(w) + \lambda \|w\|_2^2$$

over $w \in \mathbb{R}^d$.

Fact: If $\lambda > 0$, then the solution is always unique (even if $n < d$)!

- This is called *ridge regression*.
- ($\lambda = 0$ is ERM / Ordinary Least Squares.)

Explicit solution $(A^TA + \lambda I)^{-1}A^Tb$. 

Parameter $\lambda$ controls how much attention is paid to the regularizer $\|w\|_2^2$ relative to the data fitting term $\hat{R}(w)$.
Combine the two concerns: For a given $\lambda \geq 0$, find minimizer of

$$\hat{R}(w) + \lambda \|w\|^2_2$$

over $w \in \mathbb{R}^d$.

Fact: If $\lambda > 0$, then the solution is always unique (even if $n < d$)!

- This is called *ridge regression*.
  ($\lambda = 0$ is ERM / Ordinary Least Squares.)
  Explicit solution $(A^TA + \lambda I)^{-1}A^Tb$.

- Parameter $\lambda$ controls how much attention is paid to the regularizer $\|w\|^2_2$ relative to the data fitting term $\hat{R}(w)$. 

Choose $\lambda$ using cross-validation.

Note: In deep networks, this regularization is called "weight decay". (Why?)

Note: Another popular regularizer for linear regression is $\ell_1$. 

Combine the two concerns: For a given \( \lambda \geq 0 \), find minimizer of

\[
\hat{R}(w) + \lambda \|w\|_2^2
\]

over \( w \in \mathbb{R}^d \).

**Fact:** If \( \lambda > 0 \), then the solution is always unique (even if \( n < d \))!

- This is called *ridge regression*.
  
  \((\lambda = 0 \text{ is ERM / Ordinary Least Squares.})\)
  
  Explicit solution \((A^T A + \lambda I)^{-1} A^T b\).

- Parameter \( \lambda \) controls how much attention is paid to the *regularizer* \( \|w\|_2^2 \) relative to the *data fitting term* \( \hat{R}(w) \).

- Choose \( \lambda \) using cross-validation.

**Note:** in deep networks, this regularization is called "weight decay". (Why?)

**Note:** another popular regularizer for linear regression is \( \ell_1 \).
Combine the two concerns: For a given $\lambda \geq 0$, find minimizer of

$$\hat{R}(w) + \lambda \|w\|_2^2$$

over $w \in \mathbb{R}^d$.

Fact: If $\lambda > 0$, then the solution is always unique (even if $n < d$)!

- This is called ridge regression.
  ($\lambda = 0$ is ERM / Ordinary Least Squares.)
  Explicit solution $(A^T A + \lambda I)^{-1} A^T b$.

- Parameter $\lambda$ controls how much attention is paid to the regularizer $\|w\|_2^2$ relative to the data fitting term $\hat{R}(w)$.

- Choose $\lambda$ using cross-validation.

Note: in deep networks, this regularization is called “weight decay”. (Why?)

Note: another popular regularizer for linear regression is $\ell_1$. 
A hyperplane in $\mathbb{R}^d$ is a linear subspace of dimension $d-1$.

- A hyperplane in $\mathbb{R}^2$ is a line.
- A hyperplane in $\mathbb{R}^3$ is a plane.
- As a linear subspace, a hyperplane always contains the origin.

A hyperplane $H$ can be specified by a (non-zero) normal vector $w \in \mathbb{R}^d$.

The hyperplane with normal vector $w$ is the set of points orthogonal to $w$:

$$H = \left\{ x \in \mathbb{R}^d : x^T w = 0 \right\}.$$ 

Given $w$ and its corresponding $H$: $H$ splits the sets labeled positive $\{x : w^T x > 0\}$ and those labeled negative $\{x : w^T w < 0\}$. 

![Diagram of a hyperplane and normal vector](image-url)
Projection of $x$ onto $\text{span}\{w\}$ (a line) has coordinate $\|x\|_2 \cdot \cos(\theta)$ where $\cos(\theta) = \frac{x^T w}{\|w\|_2 \|x\|_2}$. (Distance to hyperplane is $\|x\|_2 \cdot |\cos(\theta)|$.)

Decision boundary is hyperplane (oriented by $w$): $x^T w > 0 \iff \|x\|_2 \cdot \cos(\theta) > 0 \iff x$ on same side of $H$ as $w$.

What should we do if we want hyperplane decision boundary that doesn’t (necessarily) go through origin?
Projection of \( \mathbf{x} \) onto \( \text{span}\{\mathbf{w}\} \) (a line) has coordinate

\[
\|\mathbf{x}\|_2 \cdot \cos(\theta)
\]

where

\[
\cos(\theta) = \frac{\mathbf{x}^\top \mathbf{w}}{\|\mathbf{w}\|_2 \|\mathbf{x}\|_2}.
\]

(Distance to hyperplane is \( \|\mathbf{x}\|_2 \cdot |\cos(\theta)| \).)
Projection of $x$ onto $\text{span}\{w\}$ (a line) has coordinate

$$\|x\|_2 \cdot \cos(\theta)$$

where

$$\cos(\theta) = \frac{x^T w}{\|w\|_2 \|x\|_2}.$$ 

(Distance to hyperplane is $\|x\|_2 \cdot |\cos(\theta)|$.)

**Decision boundary** is hyperplane (oriented by $w$):

$$x^T w > 0 \iff \|x\|_2 \cdot \cos(\theta) > 0 \iff x \text{ on same side of } H \text{ as } w$$
Projection of $x$ onto $\text{span}\{w\}$ (a line) has coordinate

$$\|x\|_2 \cdot \cos(\theta)$$

where

$$\cos(\theta) = \frac{x^T w}{\|w\|_2 \|x\|_2}.$$ 

(Distance to hyperplane is $\|x\|_2 \cdot |\cos(\theta)|$.)

**Decision boundary** is hyperplane (oriented by $w$):

$$x^T w > 0 \iff \|x\|_2 \cdot \cos(\theta) > 0 \iff x \text{ on same side of } H \text{ as } w$$

What should we do if we want hyperplane decision boundary that doesn’t (necessarily) go through origin?
Is it always possible to find \( \mathbf{w} \) with \( \text{sign}(\mathbf{w}^T \mathbf{x}_i) = y_i \)?
Is it always possible to find a hyperplane separating the data?
(Appending 1 means it need not go through the origin.)

Linearly separable.

Not linearly separable.
Cauchy-Schwarz inequality. \[ |a^T b| \leq \|a\| \cdot \|b\|. \]
Cauchy-Schwarz inequality. \( |a^T b| \leq \|a\| \cdot \|b\| \).

Proof. If \( \|a\| = \|b\| \),

\[
0 \leq \|a - b\|^2 = \|a\|^2 - 2a^T b + \|b\|^2 = 2\|a\| \cdot \|b\| - 2a^T b,
\]

which rearranges to give \( a^T b \leq \|a\| \cdot \|b\| \).

For the case \( \|a\| < \|b\| \), apply the preceding to \( \frac{\|b\|}{\|a\|} \left( a^T \left( \frac{a}{\|b\|} \right) \right) \).

For the absolute value, apply the preceding to \((a, -b)\). \(\square\)
Let’s state our classification goal with a generic margin loss $\ell$:

$$\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i w^T x_i);$$

the key properties we want:

- $\ell$ is continuous;
- $\ell(z) \geq c \mathbb{1}[z \leq 0] = c \ell_{zo}(z)$ for some $c > 0$ and any $z \in \mathbb{R}$, which implies $\hat{R}_\ell(w) \geq c \hat{R}_{zo}(w)$.
- $\ell'(0) < 0$ (pushes stuff from wrong to right).
Let’s state our classification goal with a generic margin loss $\ell$:

$$\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i w^T x_i);$$

the key properties we want:

- $\ell$ is continuous;
- $\ell(z) \geq c \mathbb{1}[z \leq 0] = c\ell_{zo}(z)$ for some $c > 0$ and any $z \in \mathbb{R}$, which implies $\hat{R}_\ell(w) \geq c\hat{R}_{zo}(w)$.
- $\ell'(0) < 0$ (pushes stuff from wrong to right).

Examples.

- **Squared loss**, written in margin form: $\ell_{ls}(z) := (1 - z)^2$; note $\ell_{ls}(y\hat{y}) = (1 - y\hat{y})^2 = y^2(1 - y\hat{y})^2 = (y - \hat{y})^2$.
- **Logistic loss**: $\ell_{log}(z) = \ln(1 + \exp(-z))$. 
Squared and logistic losses on linearly separable data I

Logistic loss.  Squared loss.
Squared and logistic losses on linearly separable data II

Logistic loss.  
Squared loss.
If there exists a perfect linear separator, empirical logistic risk minimization should find it.

**Theorem.**
If there exists a perfect linear separator, empirical logistic risk minimization should find it.

**Theorem.** If there exists $\bar{w}$ with $y_i \bar{w}^T x_i > 0$ for all $i$, then every $w$ with $\widehat{R}_{\log}(w) < \frac{\ln(2)}{2n} + \inf_v \widehat{R}_{\log}(v)$, also satisfies $y_i w^T x_i > 0$. 

If there exists a perfect linear separator, empirical logistic risk minimization should find it.

**Theorem.** If there exists $\bar{w}$ with $y_i \bar{w}^T x_i > 0$ for all $i$, then every $w$ with $\hat{R}_{\text{log}}(w) < \frac{\ln(2)}{2n} + \inf_v \hat{R}_{\text{log}}(v)$, also satisfies $y_i w^T x_i > 0$.

**Proof.** Omitted.
Least squares:

- Take gradient of $\|Aw - b\|^2$, set to 0; obtain normal equations $A^T Aw = A^T b$.
- One choice is minimum norm solution $A^+ b$. 

Logistic loss:

- Take gradient of $\hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(y_i w^T x_i))$ and set to 0 ???

Remark. Is $A^+ b$ a "closed form expression"?
Least squares:

- Take gradient of $\| Aw - b \|^2$, set to 0; obtain normal equations $A^T A w = A^T b$.
- One choice is *minimum norm solution* $A^+ b$.

Logistic loss:

- Take gradient of $\hat{R}_{\text{log}}(w) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + \exp(y_i w^T x_i))$ and set to 0.
Least squares:

- Take gradient of $\|Aw - b\|^2$, set to 0; obtain normal equations $A^T Aw = A^T b$.
- One choice is minimum norm solution $A^+ b$.

Logistic loss:

- Take gradient of $\hat{R}_{\log}(w) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 + \exp(y_i w^T x_i))$ and set to 0 ???

Remark. Is $A^+ b$ a “closed form expression”?
Given a function $F : \mathbb{R}^d \to \mathbb{R}$, gradient descent is the iteration

$$w_{i+1} := w_i - \eta_i \nabla w F(w_i),$$

where $w_0$ is given, and $\eta_i$ is a learning rate / step size.
Gradient descent

Given a function $F : \mathbb{R}^d \rightarrow \mathbb{R}$, gradient descent is the iteration

$$w_{i+1} := w_i - \eta_i \nabla_w F(w_i),$$

where $w_0$ is given, and $\eta_i$ is a learning rate / step size.

Does this work for least squares?
Given a function $F : \mathbb{R}^d \rightarrow \mathbb{R}$, gradient descent is the iteration

$$w_{i+1} := w_i - \eta_i \nabla w F(w_i),$$

where $w_0$ is given, and $\eta_i$ is a learning rate / step size.

Does this work for least squares?
Later we’ll show it works for least squares and logistic regression due to convexity.
All our methods so far handle multiclass:

- **$k$-nn and decision tree:** plurality label.
- **Least squares:**
  \[
  \arg \min_{\mathbf{W} \in \mathbb{R}^{d \times k}} \| \mathbf{AW} - \mathbf{B} \|_F^2 \quad \text{with} \quad \mathbf{B} \in \mathbb{R}^{n \times k};
  \]
  \[
  \mathbf{W} \in \mathbb{R}^{d \times k} \text{ is $k$ separate linear regressors in } \mathbb{R}^d.
  \]

How about **linear classifiers**?

- At prediction time, $x \mapsto \arg \max_y \hat{f}(x)_y$.
- As in binary case: interpretation $f(x)_y = \Pr[Y = y|X = x]$.

What is a good loss function?
Given two probability vectors $p, q \in \Delta_k = \{ p \in \mathbb{R}_{\geq 0}^k : \sum_i p_i = 1 \}$,

$$H(p, q) = -\sum_{i=1}^k p_i \ln q_i \quad \text{(cross-entropy)}.$$

- If $p = q$, then $H(p, q) = H(p)$ (entropy); indeed

$$H(p, q) = -\sum_{i=1}^k p_i \ln \left( p_i \frac{q_i}{p_i} \right) = H(p) + \text{KL}(p, q).$$

Since $\text{KL} \geq 0$ and moreover 0 iff $p = q$, this is the cost/entropy of $p$ plus a penalty for differing.

- Choose encoding $\tilde{y}_i = e_y$ for $y \in \{1, \ldots, k\}$, and $\hat{y} \propto \exp(f(x))$ with $f : \mathbb{R}^d \to \mathbb{R}^k$;

$$\ell_{\text{ce}}(\tilde{y}, f(x)) = H(\tilde{y}, \hat{y}) = -\sum_{i=1}^k \tilde{y}_i \ln \left( \frac{\exp(f(x)_i)}{\sum_{j=1}^k \exp(f(x)_j)} \right)$$

$$= -\ln \left( \frac{\exp(f(x)_y)}{\sum_{j=1}^k \exp(f(x)_j)} \right) = -f(x)_y + \ln \sum_{j=1}^k \exp(f(x)_j).$$

(In pytorch, use torch.nn.CrossEntropyLoss()(f(x), y).)
The zero-one loss for classification is
\[ \ell_{zo}(y_i, f(x)) = 1 \left[ y_i \neq \arg \max_j f(x)_j \right]. \]

In the multiclass case, can define margin as
\[ f(x)_y - \max_{j \neq y} f(x)_j, \]
interpreted as “the distance by which \( f \) is correct”. (Can be negative!)

Since \( \ln \sum_j z_j \approx \max_j z_j \), cross-entropy satisfies
\[ \ell_{ce}(\tilde{y}_i, f(x)) = -f(x)_y + \ln \sum_j \exp (f(x)_j) \]
\[ \approx -f(x)_y + \max_j f(x)_j, \]
thus minimizing cross-entropy maximizes margins.
These lectures will follow an **ERM perspective** on deep networks:

- Pick a **model/predictor class (network architecture)**.  
  (We will spend most of our time on this!)
- Pick a **loss/risk**.  
  (We will almost always use cross-entropy!)
- Pick an **optimizer**.  
  (We will mostly treat this as a black box!)

The goal is low **test error**, whereas above only gives low **training error**; we will briefly discuss this as well.
A self-contained expression is
\[ x \mapsto \sigma_L \left( W_L \sigma_{L-1} \left( \cdots (W_2 \sigma_1 (W_1 x + b_1) + b_2) \cdots \right) + b_L \right), \]
with equivalent “functional form”
\[ x \mapsto (f_L \circ \cdots \circ f_1)(x) \quad \text{where } f_i(z) = \sigma_i \left( W_i z + b_i \right). \]

Some further details (many more to come!):

- \( (W_i)_{i=1}^L \) with \( W_i \in \mathbb{R}^{d_i \times d_{i-1}} \) are the \textbf{weights}, and \( (b_i)_{i=1}^L \) are the \textbf{biases}.
- \( (\sigma_i)_{i=1}^L \) with \( \sigma_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i} \) are called \textbf{nonlinearities}, or \textbf{activations}, or \textbf{transfer functions}, or \textbf{link functions}.
- This is only the \textbf{basic setup}; many things can and will change, please ask many questions!
(Lec7-8.) Choices of activation

Basic form:

\[ \mathbf{x} \mapsto \sigma_L \left( W_L \sigma_{L-1} \left( \cdots W_2 \sigma_1 (W_1 \mathbf{x} + b_1) + b_2 \cdots \right) + b_L \right). \]

Choices of activation (univariate, coordinate-wise):

► **Indicator/step/heavyside/threshold** \( z \mapsto 1[z \geq 0] \).
  This was the original choice (1940s!).

► **Sigmoid** \( \sigma_s(z) := \frac{1}{1 + \exp(-z)} \).
  This was popular roughly 1970s - 2005?

► **Hyperbolic tangent** \( z \mapsto \tanh(z) \).
  Similar to sigmoid, used during same interval.

► **Rectified Linear Unit (ReLU)** \( \sigma_r(z) = \max\{0, z\} \).
  It (and slight variants, e.g., Leaky ReLU, ELU, ...) are the dominant choice now; popularized in “Imagenet/AlexNet” paper (Krizhevsky-Sutskever-Hinton, 2012).

► **Identity** \( z \mapsto z \); we’ll often use this as the last layer when we use cross-entropy loss.

► **NON-coordinate-wise choices:** we will discuss “softmax” and “pooling” a bit later.
Basic form:

\[ x \mapsto \sigma_L \left( W_L \sigma_{L-1} \left( \cdots W_2 \sigma_1 (W_1 x + b_1) + b_2 \cdots \right) + b_L \right) . \]

\(((W_i, b_i))_{i=1}^L\), the weights and biases, are the parameters.
Let’s roll them into \( \mathcal{W} := ((W_i, b_i))_{i=1}^L \),
and consider the network as a two-parameter function \( F_{\mathcal{W}}(x) = F(x; \mathcal{W}) \).

- The model or class of functions is \( \{F_{\mathcal{W}} : \text{all possible } \mathcal{W}\} \). \( F \) (both arguments unset) is also called an architecture.

- When we fit/train/optimize, typically we leave the architecture fixed and vary \( \mathcal{W} \) to minimize risk.
  (More on this in a moment.)
Standard ERM recipe:

- First we pick a class of functions/predictors; for deep networks, that means a $F(\cdot, \cdot)$.
- Then we pick a loss function and write down an empirical risk minimization problem; in these lectures we will pick cross-entropy:

$$\begin{align*}
\text{arg min}_{W} \frac{1}{n} \sum_{i=1}^{n} \ell_{ce}(y_i, F(x_i, W)) \\
= \text{arg min}_{W_1 \in \mathbb{R}^{d \times d_1}, b_1 \in \mathbb{R}^{d_1}} \ldots \\
\ldots \\
= \text{arg min}_{W_L \in \mathbb{R}^{d_{L-1} \times d_L}, b_L \in \mathbb{R}^{d_L}} \ldots \\
\ldots \\
= \text{arg min}_{W_1 \in \mathbb{R}^{d \times d_1}, b_1 \in \mathbb{R}^{d_1}} \ldots \\
\ldots \\
= \text{arg min}_{W_L \in \mathbb{R}^{d_{L-1} \times d_L}, b_L \in \mathbb{R}^{d_L}} \ldots \\
\text{arg min}_{W_{L-1} \in \mathbb{R}^{d_{L-2} \times d_{L-1}}, b_{L-1} \in \mathbb{R}^{d_{L-1}}} \\
\ldots \\
= \text{arg min}_{W_1 \in \mathbb{R}^{d \times d_1}, b_1 \in \mathbb{R}^{d_1}} \ldots \\
\ldots \\
= \text{arg min}_{W_L \in \mathbb{R}^{d_{L-1} \times d_L}, b_L \in \mathbb{R}^{d_L}} \ldots \\
\ldots \\
= \ldots
\end{align*}$$

- Then we pick an optimizer. In this class, we only use gradient descent variants. \textbf{It is a miracle that this works.}
(Lec7-8.) Sometimes, linear just isn’t enough

Linear predictor:
\[ x \mapsto w^T [x]. \]
Some blue points misclassified.

ReLU network:
\[ x \mapsto W_2 \sigma_r (W_1 x + b_1) + b_2. \]
0 misclassifications!
Classical “XOR problem” (Minsky-Papert-'69).
(Check wikipedia for “AI Winter”.)

*Theorem.* On this data, any linear classifier (with affine expansion) makes at least one mistake.

*Picture proof.* Recall: linear classifiers correspond to separating hyperplanes.
Classical "XOR problem" (Minsky-Papert-'69).
(Check wikipedia for "AI Winter".)

**Theorem.** On this data, any linear classifier (with affine expansion) makes at least one mistake.

**Picture proof.** Recall: linear classifiers correspond to separating hyperplanes.
  - If it splits the blue points, it’s incorrect on one of them.
(Lec7-8.) Classical example: XOR

Classical “XOR problem” (Minsky-Papert-’69).

(Check wikipedia for “AI Winter”.)

Theorem. On this data, any linear classifier (with affine expansion) makes at least one mistake.

Picture proof. Recall: linear classifiers correspond to separating hyperplanes.

- If it splits the blue points, it’s incorrect on one of them.
- If it doesn’t split the blue points, then one halfspace contains the common midpoint, and therefore wrong on at least one red point.
Theorem (Cybenko ’89, Hornik-Stinchcombe-White ’89, Funahashi ’89, Leshno et al ’92, ...). Given any continuous function $f : \mathbb{R}^d \to \mathbb{R}$ and any $\epsilon > 0$, there exist parameters $(W_1, b_1, W_2)$ so that

$$\sup_{x \in [0,1]^d} |f(x) - W_2 \sigma(W_1 x + b_1)| \leq \epsilon,$$

as long as $\sigma$ is “reasonable” (e.g., ReLU or sigmoid or threshold).
One layer was not enough. How about two?

**Theorem** (Cybenko ‘89, Hornik-Stinchcombe-White ‘89, Funahashi ‘89, Leshno et al ’92, . . .). Given any continuous function $f : \mathbb{R}^d \to \mathbb{R}$ and any $\epsilon > 0$, there exist parameters $(W_1, b_1, W_2)$ so that

$$
\sup_{x \in [0,1]^d} |f(x) - W_2 \sigma(W_1 x + b_1)| \leq \epsilon,
$$

as long as $\sigma$ is “reasonable” (e.g., ReLU or sigmoid or threshold).

**Remarks.**

- Together with XOR example, justifies using nonlinearities.
- Does *not* justify (very) deep networks.
- Only says these networks exist, not that we can optimize for them!
Classical graph-based perspective.

- Network is a directed acyclic graph; sources are inputs, sinks are outputs, intermediate nodes compute \( z \mapsto \sigma(w^Tz + b) \) (with their own \((\sigma, w, b)\)).
- Nodes at distance 1 from inputs are the first layer, distance 2 is second layer, and so on.

“Modern” graph-based perspective.

- Edges in the graph can be multivariate, meaning vectors or general tensors, and not just scalars.
- Edges will often “skip” layers; “layer” is therefore ambiguous.
- Diagram conventions differ; e.g., tensorflow graphs include nodes for parameters.
(Lec7-8.) 2-D convolution in deep networks (pictures)

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) 2-D convolution in deep networks (pictures)

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) 2-D convolution in deep networks (pictures)

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) 2-D convolution in deep networks (pictures)

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) Softmax

Replace vector input $z$ with $z' \propto e^z$, meaning

$$z \mapsto \left( \frac{e^{z_1}}{\sum_j e^{z_j}}, \ldots, \frac{e^{z_k}}{\sum_j e^{z_j}} \right).$$

- Converts input into a probability vector; useful for interpreting output network output as $\Pr[Y = y|X = x]$.
- We have baked it into our cross-entropy definition; last lectures networks with cross-entropy training had implicit softmax.
- If some coordinate $j$ of $z$ dominates others, then softmax is close to $e_j$. 
Max pooling

Often used together with convolution layers; shrinks/downsamples the input.

Another variant is average pooling.

Implementation: torch.nn.MaxPool2d.

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) Max pooling

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) Max pooling

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
(Lec7-8.) Max pooling

- Often used together with convolution layers; shrinks/downsamples the input.
- Another variant is average pooling.
- Implementation: `torch.nn.MaxPool2d`.

(Taken from https://github.com/vdumoulin/conv_arithmetic by Vincent Dumoulin, Francesco Visin.)
Define $G_j(\mathcal{W}) = \sigma_j(W_j \cdots \sigma_1(W_1x + b) \cdots)$.

The **multivariate chain rule** tells us

$$\nabla_W F(Wx) = J^T x^T,$$

and $J \in \mathbb{R}^{l \times k}$ is the **Jacobian matrix** of $F : \mathbb{R}^k \rightarrow \mathbb{R}^l$ at $Wx$, the matrix of all coordinate-wise derivatives.

$$\frac{dG_L}{dW_L} = J_L^T G_{L-1}(\mathcal{W})^T$$

$$\frac{dG_L}{db_L} = J_L^T,$$
Multivariate network single-example gradients

Define $G_j(W) = \sigma_j(W_j \cdots \sigma_1(W_1x + b) \cdots )$. The multivariate chain rule tells us

$$\nabla_W F(Wx) = J^T x^T,$$

and $J \in \mathbb{R}^{l \times k}$ is the Jacobian matrix of $F : \mathbb{R}^k \rightarrow \mathbb{R}^l$ at $Wx$, the matrix of all coordinate-wise derivatives.

$$\frac{dG_L}{dW_L} = J^T_L G_{L-1}(W)^T,$$

$$\frac{dG_L}{db_L} = J^T_L,$$

$$\vdots$$

$$\frac{dG_L}{dW_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T G_{j-1}(W)^T,$$

$$\frac{dG_L}{db_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T,$$
Define $G_j(\mathcal{W}) = \sigma_j(W_j \cdots \sigma_1(W_1x + b) \cdots)$.

The multivariate chain rule tells us

$$\nabla_W F(Wx) = J^T x^T,$$

and $J \in \mathbb{R}^{l \times k}$ is the Jacobian matrix of $F : \mathbb{R}^k \to \mathbb{R}^l$ at $Wx$, the matrix of all coordinate-wise derivatives.

$$\frac{dG_L}{dW_L} = J_L G_{L-1}(\mathcal{W})^T$$

$$\frac{dG_L}{db_L} = J_L^T,$$

$$\vdots$$

$$\frac{dG_L}{dW_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T G_{j-1}(\mathcal{W})^T,$$

$$\frac{dG_L}{db_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T,$$

with $J_j$ as the Jacobian of $\sigma_j$ at $W_j G_{j-1}(\mathcal{W}) + b_j$.

For example, with $\sigma_j$ that is coordinate-wise $\sigma : \mathbb{R} \to \mathbb{R}$,

$J_j$ is diag \( \sigma' \left( \left[ W_j G_{j-1}(\mathcal{W}) + b_j \right]_{1} \right), \ldots, \sigma' \left( \left[ W_j G_{j-1}(\mathcal{W}) + b_j \right]_{d_j} \right) \).
Recall

\[
\frac{dG_L}{dW_j} = (J_L W L J_{L-1} W_{L-1} \cdots J_j)^T G_{j-1}(W)^T.
\]

▶ What if we set \( W = 0 \)? What if \( \sigma = \sigma_r \) is a ReLU?
Recall
\[
\frac{dG_L}{dW_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T G_{j-1}(\mathcal{W})^T.
\]

- What if we set \( \mathcal{W} = 0 \)? What if \( \sigma = \sigma_r \) is a ReLU?
- What if we set two rows of \( W_j \) (two nodes) identically?
Recall

\[
\frac{dG_L}{dW_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T G_{j-1}(W)^T.
\]

- What if we set \( W = 0 \)? What if \( \sigma = \sigma_r \) is a ReLU?
- What if we set two rows of \( W_j \) (two nodes) identically?
- Resolving this issue is called symmetry breaking.
Recall
\[
\frac{dG_L}{dW_j} = (J_L W_L J_{L-1} W_{L-1} \cdots J_j)^T G_{j-1}(\mathcal{W})^T.
\]

- What if we set $\mathcal{W} = 0$? What if $\sigma = \sigma_r$ is a ReLU?
- What if we set two rows of $W_j$ (two nodes) identically?
- Resolving this issue is called symmetry breaking.
- Standard linear/dense layer initializations:

\[
\mathcal{N}(0, \frac{2}{d_{in}})
\]

“He et al.”

\[
\mathcal{N}(0, \frac{2}{d_{in} + d_{out}})
\]

Glorot/Xavier,

\[
\mathcal{U}(-\frac{1}{\sqrt{d_{in}}}, \frac{1}{\sqrt{d_{in}}})
\]

torch default.

(Convolution layers adjusted to have similar distributions.)

Random initialization is emerging as a key story in deep networks!
We used the linearity of gradients to write

$$\nabla_w \hat{R}(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(F(x_i; w), y_i).$$

What happens if we replace \(((x_i, y_i))_{i=1}^{n}\) with minibatch \(((x'_i, y'_i))_{i=1}^{b}\)?

- Random minibatch $\implies$ two gradients equal in expectation.
- Most torch layers take minibatch input:
  - `torch.nn.Linear` has input shape \((b, d)\), output \((b, d')\).
  - `torch.nn.Conv2d` has input shape \((b, c, h, w)\), output \((b, c', h', w')\).
- This is used heavily outside deep learning as well.
  It is an easy way to use parallel floating point operations (as in GPU and CPU).
- Setting batch size is black magic and depends on many things (prediction problem, gpu characteristics, \ldots).
A set $S$ is **convex** if, for every pair of points $\{x, x'\}$ in $S$, the line segment between $x$ and $x'$ is also contained in $S$. ($\{x, x'\} \in S \implies [x, x'] \in S$.)
A set $S$ is \textit{convex} if, for every pair of points $\{x, x'\}$ in $S$, the line segment between $x$ and $x'$ is also contained in $S$. ($\{x, x'\} \in S \implies [x, x'] \in S$)

\begin{align*}
\text{convex} & \quad \text{not convex} & \quad \text{convex} & \quad \text{convex} \\
\text{convex} & \quad \text{not convex} & \quad \text{convex} & \quad \text{convex}
\end{align*}

\textbf{Examples:}

- All of $\mathbb{R}^d$.
- Empty set.
- Half-spaces: $\{x \in \mathbb{R}^d : a^T x \leq b\}$.
- Intersections of convex sets.
- Polyhedra: $\left\{x \in \mathbb{R}^d : Ax \leq b\right\} = \bigcap_{i=1}^{m} \left\{x \in \mathbb{R}^d : a_i^T x \leq b_i\right\}$.
- Convex hulls:
  \[
  \text{conv}(S') := \left\{\sum_{i=1}^{k} \alpha_i x_i : k \in \mathbb{N}, \ x_i \in S, \ \alpha_i \geq 0, \ \sum_{i=1}^{k} \alpha_i = 1\right\}.
  \]
  (Infinite convex hulls: intersection of all convex supersets.)
The *epigraph* of a function $f$ is the area above the curve:

$$\text{epi}(f) := \{(x, y) \in \mathbb{R}^{d+1} : y \geq f(x)\}.$$ 

A function is convex if its epigraph is convex.
A function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is **convex** if for any $x, x' \in \mathbb{R}^d$ and $\alpha \in [0, 1],\n\quad f \left( (1 - \alpha)x + \alpha x' \right) \leq (1 - \alpha) \cdot f(x) + \alpha \cdot f(x').$
(Lec9-10.) Convex functions (standard definition)

A function \( f : \mathbb{R}^d \to \mathbb{R} \) is \textit{convex} if for any \( x, x' \in \mathbb{R}^d \) and \( \alpha \in [0, 1] \),

\[
 f((1 - \alpha)x + \alpha x') \leq (1 - \alpha) \cdot f(x) + \alpha \cdot f(x').
\]

Examples:

- \( f(x) = c^x \) for any \( c > 0 \) (on \( \mathbb{R} \))
- \( f(x) = |x|^c \) for any \( c \geq 1 \) (on \( \mathbb{R} \))
- \( f(x) = b^\top x \) for any \( b \in \mathbb{R}^d \).
- \( f(x) = \|x\| \) for any norm \( \|\cdot\| \).
- \( f(x) = x^\top A x \) for symmetric positive semidefinite \( A \).
- \( f(x) = \ln \left( \sum_{i=1}^d \exp(x_i) \right) \), which approximates \( \max_i x_i \).
Differentiable functions

If $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable, then $f$ is convex if and only if

$$f(x) \geq f(x_0) + \nabla f(x_0)^T(x - x_0)$$

for all $x, x_0 \in \mathbb{R}^d$.

Note: this implies increasing slopes:

$$(\nabla f(x) - \nabla f(y))^T(x - y) \geq 0.$$
Differentiable functions

If $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable, then $f$ is convex if and only if

$$f(x) \geq f(x_0) + \nabla f(x_0)^T (x - x_0)$$

for all $x, x_0 \in \mathbb{R}^d$.

**Note:** this implies *increasing slopes*:

$$(\nabla f(x) - \nabla f(y))^T (x - y) \geq 0.$$ 

Twice-differentiable functions

If $f : \mathbb{R}^d \to \mathbb{R}$ is twice-differentiable, then $f$ is convex if and only if

$$\nabla^2 f(x) \succeq 0$$

for all $x \in \mathbb{R}^d$ (i.e., the Hessian, or matrix of second-derivatives, is positive semi-definite for all $x$).
Standard form of a *convex optimization problem*:

\[
\min_{x \in \mathbb{R}^d} f_0(x) \\
\text{s.t.} \quad f_i(x) \leq 0 \quad \text{for all } i = 1, \ldots, n
\]

where \( f_0, f_1, \ldots, f_n : \mathbb{R}^d \rightarrow \mathbb{R} \) are *convex functions*.
Standard form of a *convex optimization problem*:

$$\min_{x \in \mathbb{R}^d} f_0(x)$$

subject to

$$f_i(x) \leq 0 \quad \text{for all } i = 1, \ldots, n$$

where $f_0, f_1, \ldots, f_n : \mathbb{R}^d \rightarrow \mathbb{R}$ are convex functions.

**Fact:** the feasible set

$$A := \left\{ x \in \mathbb{R}^d : f_i(x) \leq 0 \text{ for all } i = 1, 2, \ldots, n \right\}$$

is a convex set.

(SVMs next week will give us an example.)
If \( f : \mathbb{R}^d \to \mathbb{R} \) is convex, then \( \mathbb{E} f(X) \geq f(\mathbb{E}X) \).

**Proof.** Set \( y := \mathbb{E}X \), and pick any \( s \in \partial f(\mathbb{E}X) \). Then

\[
\mathbb{E} f(X) \geq \mathbb{E} \left( f(y) + s^T(X - y) \right) = f(y) + s^T \mathbb{E} (X - y) = f(y).
\]

**Note.** This inequality comes up often!
The solution $\hat{w}$ to the following mathematical optimization problem:

$$\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2} \| \mathbf{w} \|^2_2$$

subject to $y \mathbf{x}^T \mathbf{w} \geq 1$ for all $(\mathbf{x}, y) \in S$

gives the linear classifier with the largest minimum margin on $S$—i.e., the maximum margin linear classifier or support vector machine (SVM) classifier.
The solution $\hat{w}$ to the following mathematical optimization problem:

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|_2^2$$

s.t. $yx^T w \geq 1$ for all $(x, y) \in S$

gives the linear classifier with the largest minimum margin on $S$—i.e., the maximum margin linear classifier or support vector machine (SVM) classifier.

This is a convex optimization problem; can be solved in polynomial time.
(Lec11-12.) Maximum margin linear classifier

The solution \( \hat{w} \) to the following mathematical optimization problem:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2} \| \mathbf{w} \|_2^2 \\
\text{s.t.} \quad y\mathbf{x}^\top \mathbf{w} \geq 1 \quad \text{for all } (\mathbf{x}, y) \in S
\]

gives the linear classifier with the largest minimum margin on \( S \)—i.e., the maximum margin linear classifier or support vector machine (SVM) classifier.

This is a \textit{convex optimization problem}; can be solved in polynomial time.

If there is a solution (i.e., \( S \) is linearly separable), then the solution is \textit{unique}. 

(Lec11-12.) Maximum margin linear classifier

The solution $\hat{w}$ to the following mathematical optimization problem:

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|_2^2$$

s.t. $yx^T w \geq 1$ for all $(x, y) \in S$

gives the linear classifier with the largest minimum margin on $S$—i.e., the maximum margin linear classifier or support vector machine (SVM) classifier.

This is a convex optimization problem; can be solved in polynomial time.

If there is a solution (i.e., $S$ is linearly separable), then the solution is unique.

We can solve this in a variety of ways (e.g., projected gradient descent); we will work with the dual.
The solution \( \hat{w} \) to the following mathematical optimization problem:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \quad \frac{1}{2} \| \mathbf{w} \|^2_2 \\
\text{s.t.} \quad y \mathbf{x}^T \mathbf{w} \geq 1 \quad \text{for all} \quad (\mathbf{x}, y) \in S
\]

gives the linear classifier with the largest minimum margin on \( S \)—i.e., the maximum margin linear classifier or \textit{support vector machine (SVM) classifier}.

This is a \textit{convex optimization problem}; can be solved in polynomial time.

If there is a solution (i.e., \( S \) is linearly separable), then the solution is \textit{unique}.

We can solve this in a variety of ways (e.g., projected gradient descent); \textit{we will work with the dual}.

\textbf{Note}: Can also explicitly include affine expansion, so decision boundary need not pass through origin. \textit{We’ll do our derivations without it.}
Lagrangian

\[ L(w, \alpha) = \frac{1}{2} \|w\|_2^2 + \sum_{i=1}^{n} \alpha_i (1 - y_i x_i^T w). \]

Primal maximum margin problem was

\[ P(w) = \max_{\alpha \geq 0} L(w, \alpha) = \max_{\alpha \geq 0} \left[ \frac{1}{2} \|w\|_2^2 + \sum_{i=1}^{n} \alpha_i (1 - y_i x_i^T w) \right]. \]

Dual problem

\[ D(\alpha) = \min_{w \in \mathbb{R}^d} L(w, \alpha) = L \left( \sum_{i=1}^{n} \alpha_i y_i x_i, \alpha \right) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \left\| \sum_{i=1}^{n} \alpha_i y_i x_i \right\|_2^2 = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j. \]

Given dual optimum \( \hat{\alpha} \),

- Corresponding primal optimum \( \hat{w} = \sum_{i=1}^{n} \alpha_i y_i x_i \);
- Strong duality \( P(\hat{w}) = D(\hat{\alpha}) \);
- \( \hat{\alpha}_i > 0 \) implies \( y_i x_i^T \hat{w} = 1 \),
  and these \( y_i x_i \) are support vectors.
Unconstrained primal:

\[
\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \left[ 1 - y_i \mathbf{x}_i^T \mathbf{w} \right] + .
\]

Dual:

\[
\max_{\alpha \in \mathbb{R}^n} \left[ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \left\| \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \right\|^2 \right]
\]

Dual solution \( \hat{\alpha} \) gives primal solution \( \hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \).
Unconstrained primal:

$$\min_{w \in \mathbb{R}^d} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \left[ 1 - y_i x_i^T w \right]_+.$$ 

Dual:

$$\max_{\alpha \in \mathbb{R}^n} \left[ \sum_{i=1}^n \alpha_i - \frac{1}{2} \left\| \sum_{i=1}^n \alpha_i y_i x_i \right\|^2 \right]$$

Dual solution $\hat{\alpha}$ gives primal solution $\hat{w} = \sum_{i=1}^n \alpha_i y_i x_i$.

Remarks.

- Can take $C \to \infty$ to recover the separable case.
- Dual is a constrained convex quadratic (can be solved with projected gradient descent).
- Some presentations include bias in primal ($x_i^T w + b$); this introduces a constraint $\sum_{i=1}^n y_i \alpha_i = 0$ in dual.
- Some presentations replace $\frac{1}{2}$ and $C$ with $\frac{1}{2}$ and $\frac{1}{n}$, respectively.
Looking at the dual again

SVM dual problem only depends on $x_i$ through inner products $x_i^T x_j$.

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j.$$
SVM dual problem only depends on $x_i$ through inner products $x_i^T x_j$.

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j.$$ 

If we use feature expansion (e.g., quadratic expansion) $x \mapsto \phi(x)$, this becomes

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j).$$
SVM dual problem only depends on $x_i$ through inner products $x_i^T x_j$.

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j.$$ 

If we use feature expansion (e.g., quadratic expansion) $x \mapsto \phi(x)$, this becomes

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j).$$ 

Solution $\hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x_i)$ is used in the following way:

$$x \mapsto \phi(x)^T \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x)^T \phi(x_i).$$
SVM dual problem only depends on $x_i$ through inner products $x_i^T x_j$.

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j.$$  

If we use feature expansion (e.g., quadratic expansion) $x \mapsto \phi(x)$, this becomes

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j).$$

Solution $\hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x_i)$ is used in the following way:

$$x \mapsto \phi(x)^T \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x)^T \phi(x_i).$$

Key insight:

- Training and prediction only use $\phi(x)^T \phi(x')$, never an isolated $\phi(x)$;
- Sometimes computing $\phi(x)^T \phi(x')$ is much easier than computing $\phi(x)$.
(Lec11-12.) Quadratic expansion

▶ \( \phi: \mathbb{R}^d \rightarrow \mathbb{R}^{1+2d+\binom{d}{2}} \), where

\[
\phi(x) = \left( 1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_d, \ldots, \sqrt{2}x_{d-1}x_d \right)
\]

(Don’t mind the \( \sqrt{2}'s \ldots \))
(Lec11-12.) Quadratic expansion

- \( \phi : \mathbb{R}^d \rightarrow \mathbb{R}^{1+2d+d \choose 2} \), where

\[
\phi(x) = \left( 1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_d, \ldots, \sqrt{2}x_{d-1}x_d \right)
\]

(Don’t mind the \( \sqrt{2}’s \ldots \))

- **Computing** \( \phi(x)^T \phi(x') \) in \( O(d) \) time:

\[
\phi(x)^T \phi(x') = (1 + x^T x')^2.
\]
\(\phi: \mathbb{R}^d \to \mathbb{R}^{1+2d+\binom{d}{2}}\), where

\[
\phi(x) = \left(1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_d, \ldots, \sqrt{2}x_{d-1}x_d\right)
\]

(Don’t mind the \(\sqrt{2}\)'s…)

Computing \(\phi(x)^T \phi(x')\) in \(O(d)\) time:

\[
\phi(x)^T \phi(x') = (1 + x^T x')^2.
\]

Much better than \(d^2\) time.
Quadratic expansion

- \( \phi : \mathbb{R}^d \rightarrow \mathbb{R}^{1+2d+\binom{d}{2}} \), where

\[
\phi(x) = \left( 1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_d, \ldots, \sqrt{2}x_{d-1}x_d \right)
\]

(Don’t mind the \( \sqrt{2} \)'s...)

- **Computing** \( \phi(x)^T \phi(x') \) in \( O(d) \) time:

\[
\phi(x)^T \phi(x') = (1 + x^T x')^2.
\]

- Much better than \( d^2 \) time.

- What if we change exponent “2”?
\( \phi : \mathbb{R}^d \rightarrow \mathbb{R}^{1+2d+(d/2)} \), where

\[
\phi(x) = \left( 1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_1x_d, \ldots, \sqrt{2}x_{d-1}x_d \right)
\]

(Don’t mind the \( \sqrt{2} \)’s…)

\( \textbf{Computing } \phi(x)^\top \phi(x') \textbf{ in } O(d) \textbf{ time:} \)

\[
\phi(x)^\top \phi(x') = (1 + x^\top x')^2.
\]

\( \text{Much better than } d^2 \text{ time.} \)

\( \text{What if we change exponent “2”?} \)

\( \text{What if we replace additive “1” with 0?} \)
Infinite dimensional feature expansion

For any $\sigma > 0$, there is an infinite feature expansion $\phi: \mathbb{R}^d \to \mathbb{R}^\infty$ such that

$$\phi(x)^T \phi(x') = \exp \left( -\frac{\|x - x'\|_2^2}{2\sigma^2} \right),$$

which can be computed in $O(d)$ time.

(This is called the Gaussian kernel with bandwidth $\sigma$.)
A (positive definite) *kernel function* $K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function satisfying:

For any $x_1, x_2, \ldots, x_n \in \mathcal{X}$, the $n \times n$ matrix whose $(i, j)$-th entry is $K(x_i, x_j)$ is positive semidefinite.

(This matrix is called the *Gram matrix*.)
(Lec11-12.) Kernels

A (positive definite) kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function satisfying:

For any $x_1, x_2, \ldots, x_n \in \mathcal{X}$, the $n \times n$ matrix whose $(i,j)$-th entry is $K(x_i, x_j)$ is positive semidefinite.

(This matrix is called the Gram matrix.)

For any kernel $K$, there is a feature mapping $\phi : \mathcal{X} \to \mathbb{H}$ such that

$$\phi(x)^T \phi(x') = K(x, x').$$

$\mathbb{H}$ is a Hilbert space—i.e., a special kind of inner product space—called the Reproducing Kernel Hilbert Space corresponding to $K$. 
(Lec11-12.) Kernel SVMs (Boser, Guyon, and Vapnik, 1992)

Solve

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j).$$

\( \hat{w} \) is used in the following way:

\( x \mapsto \phi(x) \)

\( T \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i K(x, x_i) \).

▶ To represent classifier, need to keep support vector examples \((x_i, y_i)\) and corresponding \(\hat{\alpha}_i\)'s.

▶ To compute prediction on \(x\), iterate through support vector examples and compute \(K(x, x_i)\) for each support vector \(x_i\).

Very similar to nearest neighbor classifier:

predictor is represented using (a subset of) the training data.
(Lec11-12.) Kernel SVMs (Boser, Guyon, and Vapnik, 1992)

Solve

$$\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j).$$

Solution $\hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x_i)$ is used in the following way:

$$x \mapsto \phi(x)^\top \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i K(x, x_i).$$
Solve
\[
\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j).
\]

Solution \( \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x_i) \) is used in the following way:
\[
x \mapsto \phi(x)^\top \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i K(x, x_i).
\]

- To represent classifier, need to keep support vector examples \((x_i, y_i)\) and corresponding \(\hat{\alpha}_i\)'s.
Solve

\[
\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j).
\]

Solution \( \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x_i) \) is used in the following way:

\[
\begin{align*}
x \mapsto \phi(x)^{\top} \hat{w} &= \sum_{i=1}^{n} \hat{\alpha}_i y_i K(x, x_i).
\end{align*}
\]

- To represent classifier, need to keep support vector examples \((x_i, y_i)\) and corresponding \(\hat{\alpha}_i\)'s.
- To compute prediction on \(x\), iterate through support vector examples and compute \(K(x, x_i)\) for each support vector \(x_i\) ...
Solve

\[
\max_{\alpha_1, \alpha_2, \ldots, \alpha_n \geq 0} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j).
\]

Solution \( \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i \phi(x_i) \) is used in the following way:

\[
x \mapsto \phi(x)^T \hat{w} = \sum_{i=1}^{n} \hat{\alpha}_i y_i K(x, x_i).
\]

- To represent classifier, need to keep support vector examples \((x_i, y_i)\) and corresponding \(\hat{\alpha}_i\)'s.
- To compute prediction on \(x\), iterate through support vector examples and compute \(K(x, x_i)\) for each support vector \(x_i\) . . .

**Very similar to nearest neighbor classifier:**
predictor is represented using (a subset of) the training data.
Nonlinear support vector machines (again)

ReLU network.

Quadratic SVM.

RBF SVM (\(\sigma = 1\)).

RBF SVM (\(\sigma = 0.1\)).
The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$ 

Remarks.
Perceptron update (Rosenblatt ’58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$ 

Remarks.

- Can interpret algorithm as:
  - *either* we are correct with a margin ($yw^T x > 0$) and we do nothing,
  - *or* we are not and update $w \leftarrow w + yx$. 

Not obvious that Perceptron will eventually terminate! (We’ll return to this.)
Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter
\[ w \leftarrow w + 1[yw^T x \leq 0]yx. \]

Remarks.

- Can interpret algorithm as:
  *either* we are correct with a margin ($yw^T x > 0$) and we do nothing,
  *or* we are not and update $w \leftarrow w + yx$.

- Therefore: if we update, we do so by rotating towards $yx$. 
The Perceptron Algorithm

Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$  

Remarks.

- Can interpret algorithm as: 
  *either* we are correct with a margin ($yw^T x > 0$) *and* we do nothing,  
  *or* we are not and update $w \leftarrow w + yx$.

- Therefore: if we update, we do so by rotating towards $yx$.

- This makes sense: $(w + yx)^T(yx) = w^T(yx) + \|x\|^2$; i.e., we increase $w^T(yx)$.  

Not obvious that Perceptron will eventually terminate! (We'll return to this.)
Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter

$$w \leftarrow w + 1[yw^T x \leq 0]yx.$$ 

Remarks.

- Can interpret algorithm as:
  either we are correct with a margin ($yw^T x > 0$) and we do nothing, 
  or we are not and update $w \leftarrow w + yx$.
- Therefore: if we update, we do so by rotating towards $yx$.
- This makes sense: $(w + yx)^T(yx) = w^T(yx) + ||x||^2$; i.e., we increase $w^T(yx)$.

Scenario 1

Current vector $\hat{w}_t$ comparable to $x_t$ in length.
Lec13. The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize \( w := 0 \), and thereafter

\[
    w \leftarrow w + 1[yw^T x \leq 0]yx.
\]

Remarks.

► Can interpret algorithm as:

    either we are correct with a margin \((yw^T x > 0)\) and we do nothing, or we are not and update \( w \leftarrow w + yx \).

► Therefore: if we update, we do so by rotating towards \( yx \).

► This makes sense: \((w + yx)^T(yx) = w^T(yx) + \|x\|^2\); i.e., we increase \( w^T(yx) \).

Scenario 1

\[
    x^T \hat{w}_t \leq 0
\]

Updated vector \( \hat{w}_{t+1} \) now correctly classifies \((x_t, y_t)\).
Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter

$$w \leftarrow w + \mathbb{1}[yw^T x \leq 0]yx.$$ 

**Remarks.**

- Can interpret algorithm as: 
  *either we are correct with a margin ($yw^T x > 0$) and we do nothing, 
or we are not and update $w \leftarrow w + yx$. 
- Therefore: if we update, we do so by rotating towards $yx$. 
- This makes sense: $(w + yx)^T(yx) = w^T(yx) + \|x\|^2$; i.e., we increase $w^T(yx)$.

**Scenario 2**

Current vector $\hat{w}_t$ much longer than $x_t$. 

![Diagram of Scenario 2](image-url)
The Perceptron Algorithm

Perceptron update (Rosenblatt ’58): initialize \( \mathbf{w} := 0 \), and thereafter

\[
\mathbf{w} \leftarrow \mathbf{w} + 1[y\mathbf{w}^T\mathbf{x} \leq 0]y\mathbf{x}.
\]

Remarks.

- Can interpret algorithm as:
  
  \text{either we are correct with a margin (}y\mathbf{w}^T\mathbf{x} > 0\text{) and we do nothing, or we are not and update } \mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}.

- Therefore: if we update, we do so by rotating towards \( y\mathbf{x} \).

- This makes sense: \((\mathbf{w} + y\mathbf{x})^T(y\mathbf{x}) = \mathbf{w}^T(y\mathbf{x}) + ||\mathbf{x}||^2\); i.e., we increase \( \mathbf{w}^T(y\mathbf{x}) \).

Scenario 2

Updated vector \( \hat{\mathbf{w}}_{t+1} \) does not correctly classify \((\mathbf{x}_t, y_t)\).
Perceptron update (Rosenblatt '58): initialize $w := 0$, and thereafter
\[ w \leftarrow w + 1[yw^T x \leq 0]yx. \]

Remarks.
- Can interpret algorithm as: 
  *either* we are correct with a margin ($yw^Tx > 0$) and we do nothing, 
  *or* we are not and update $w \leftarrow w + yx$.
- Therefore: if we update, we do so by rotating towards $yx$.
- This makes sense: $(w + yx)^T(yx) = w^T(yx) + \|x\|^2$; i.e., we increase $w^T(yx)$.

Scenario 2

Updated vector $\hat{w}_{t+1}$ does not correctly classify $(x_t, y_t)$.

Not obvious that Perceptron will eventually terminate! (We’ll return to this.)
2. Homework review
Nice homework problems

- HW1-P3: SVD practice.
- HW2-P1: SVD practice, definition $\|A\|_2$ and $\|A\|_F$.
- HW3-P1: convexity practice.
- HW3-P2: deep network and probability practice.
- HW3-P4, HW3-P5: kernel practice.
3. Stuff added 3/12/2019
Belaboring Cauchy-Schwarz

Theorem (Cauchy-Schwarz). \( |a^T b| \leq \|a\| \cdot \|b\| \).
Belaboring Cauchy-Schwarz

**Theorem (Cauchy-Schwarz).** \(|a^\top b| \leq \|a\| \cdot \|b\|\).

**Proof (another one...).** If either \(a\) or \(b\) are zero, then \(a^\top b = 0 = \|a\| \cdot \|b\|\).

If both \(a\) and \(b\) are nonzero, note for any \(r > 0\) that

\[
0 \leq \left\|r a - \frac{b}{r}\right\|^2 = r^2 \|a\|^2 - 2 a^\top b + \frac{\|b\|^2}{r^2},
\]

which can be rearranged into

\[
a^\top b \leq \frac{r^2 \|a\|^2}{2} + \frac{\|b\|^2}{2r^2}.
\]

Choosing \(r = \sqrt{\|b\|/\|a\|}\),

\[
a^\top b \leq \frac{\|a\|^2}{2} \left(\frac{\|b\|}{\|a\|}\right) + \frac{\|b\|^2}{2} \left(\frac{\|a\|}{\|b\|}\right) = \|a\| \cdot \|b\|.
\]

Lastly, applying the bound to \((a, -b)\) gives

\[-a^\top b = a^\top (\ -b) \leq \|a\| \cdot \| -b\| = \|a\| \cdot \|b\|,
\]

so together it follows that \(\left|a^\top b\right| \leq \|a\| \cdot \|b\|\).
SVD again

1. SV triples: $(s, u, v)$ satisfies $Mv = s u$, and $M^T u = s v$.

2. Thin decomposition SVD: $M = \sum_{i=1}^{r} s_i u_i v_i^T$.

3. Full factorization SVD: $M = USV^T$.

4. "Operational" view of SVD: for $M \in \mathbb{R}^{n \times d}$, 

$$
\begin{bmatrix}
\uparrow & \uparrow \\
n & n \\
\downarrow & \downarrow
\end{bmatrix}
\cdot
\begin{bmatrix}
s_1 & 0 & \cdots & 0 \\
0 & s_r & \cdots & 0 \\
& & \ddots & \vdots \\
& & \cdots & s_k
\end{bmatrix}
\cdot
\begin{bmatrix}
\uparrow & \uparrow \\
r & d \\
\downarrow & \downarrow
\end{bmatrix}
\top.
$$

First part of $U$, $V$ span the col / row space (respectively), second part the left / right nullspaces (respectively).

Personally: I internalize SVD and use it to reason about matrices. E.g., "rank-nullity theorem".
SVD again

1. SV triples: \((s, u, v)\) satisfies \(Mv = su\), and \(M^T u = sv\).
SVD again

1. SV triples: \((s, u, v)\) satisfies \(M v = su\), and \(M^T u = sv\).
2. Thin decomposition SVD: \(M = \sum_{i=1}^{T} s_i u_i v_i^T\).
1. SV triples: \((s, u, v)\) satisfies \(Mv = su\), and \(M^T u = sv\).
2. Thin decomposition SVD: \(M = \sum_{i=1}^{r} s_i u_i v_i^T\).
3. Full factorization SVD: \(M = USV^T\).
1. SV triples: \((s, u, v)\) satisfies \(Mv = su\), and \(M^Tu = sv\).

2. Thin decomposition SVD: \(M = \sum_{i=1}^{r} s_i u_i v_i^\top\).

3. Full factorization SVD: \(M = USV^\top\).

4. “Operational” view of SVD: for \(M \in \mathbb{R}^{n \times d}\),

\[
\begin{bmatrix}
u_1 & \cdots & u_r & u_{r+1} & \cdots & u_n
\end{bmatrix}
\begin{bmatrix}
s_1 & 0 & 0 & \cdots & 0
\end{bmatrix}
\begin{bmatrix}
v_1 & \cdots & v_r & v_{r+1} & \cdots & v_d
\end{bmatrix}^\top
\]

First part of \(U\), \(V\) span the col / row space (respectively), second part the left / right nullspaces (respectively).

Personally: I internalize SVD and use it to reason about matrices. E.g., “rank-nullity theorem.”
SV triples: \((s, u, v)\) satisfies \(Mv = su\), and \(M^Tu = sv\).

2. Thin decomposition SVD: 
\[ M = \sum_{i=1}^{r} s_i u_i v_i^T. \]

3. Full factorization SVD: 
\[ M = U SV^T. \]

4. “Operational” view of SVD: for \(M \in \mathbb{R}^{n \times d}\),
\[
\begin{bmatrix}
\uparrow & \cdots & \uparrow \\
\downarrow & \cdots & \downarrow
\end{bmatrix}
\begin{bmatrix}
\uparrow \\
\downarrow
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
\hspace{2cm}
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
\uparrow & \cdots & \uparrow \\
\downarrow & \cdots & \downarrow
\end{bmatrix}
\begin{bmatrix}
\uparrow \\
\downarrow
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}
\hspace{2cm}
\end{bmatrix}
\end{bmatrix}^T.
\]

First part of \(U, V\) span the col / row space (respectively), second part the left / right nullspaces (respectively).
1. SV triples: \((s, u, v)\) satisfies \(Mv = su\), and \(M^Tu = sv\).

2. Thin decomposition SVD: \(M = \sum_{i=1}^{r} s_i u_i v_i^T\).

3. Full factorization SVD: \(M = USV^T\).

4. “Operational” view of SVD: for \(M \in \mathbb{R}^{n \times d}\),

\[
\begin{bmatrix}
\uparrow & \cdots & \uparrow \\
u_1 & \cdots & u_r \\
\downarrow & \cdots & \downarrow
\end{bmatrix}
\begin{bmatrix}
\uparrow & \cdots & \uparrow \\
u_{r+1} & \cdots & u_n \\
\downarrow & \cdots & \downarrow
\end{bmatrix}
\begin{bmatrix}
\begin{bmatrix}s_1 & \cdots & 0 \\0 & \cdots & 0 \\0 & \cdots & 0
\end{bmatrix} \\
\begin{bmatrix}v_1 & \cdots & v_r \\0 & \cdots & 0 \\0 & \cdots & 0
\end{bmatrix}
\end{bmatrix}^T
\]

First part of \(U, V\) span the col / row space (respectively), second part the left / right nullspaces (respectively).

Personally: I internalize SVD and use it to reason about matrices.
E.g., “rank-nullity theorem”.