STEVENS
INSTITUTE OF TECHNOLOGY
1870

Artificial Intelligence

Computer Science, CS541 - A

Jonggi Hong

Announcements

- HW #2
  - Update on exercise 3 (check the announcement in Canvas)
- Project midterm report
  - Due 11:59 pm, Tuesday, March 28
- HW #3
  - Due 11:59 pm, Tuesday, April 4

Midterm Course Review

- Good
  - Polls
  - Videos
  - Visualization
- Suggestions
  - Programming exercise
  - Examples in class
  - Difficulty of the contents
- Others
  - Grading
  - Random assignments for project groups
  - Reading assignments for extra credits
  - Difficulties in learning for remote students

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Recap

Perceptron, Decision Tree

Supervised Learning

- Given a training set of examples of the form (x, f(x))
  - x is the input, f(x) is the output.
- Return a function h that approximates f
  - h is the **hypothesis**.

Attribute

Sky	Air Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

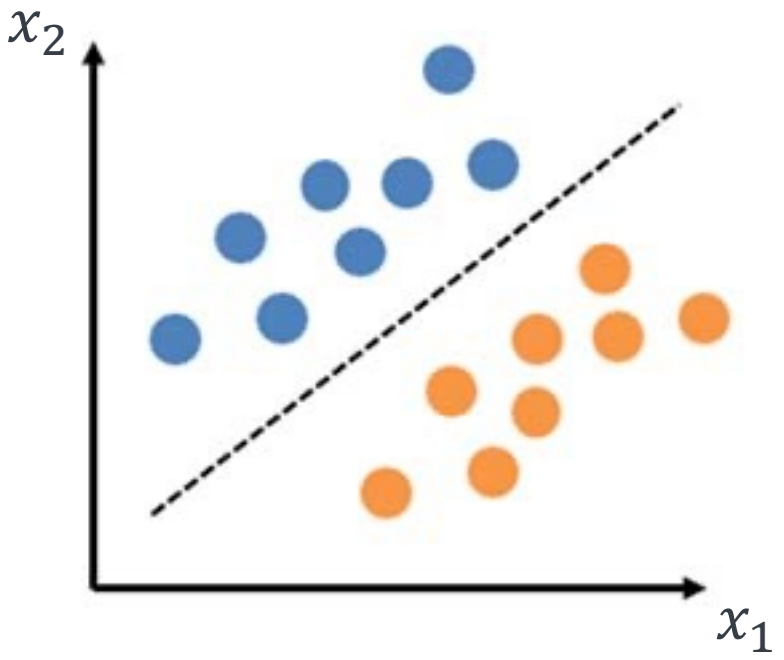
x

f(x)

Perceptron

Linear Threshold Classifiers

Imagine you have data of the form (**x**, f(**x**)) where **x** in $\mathbf{R}^n$ and f(x)=0 or 1

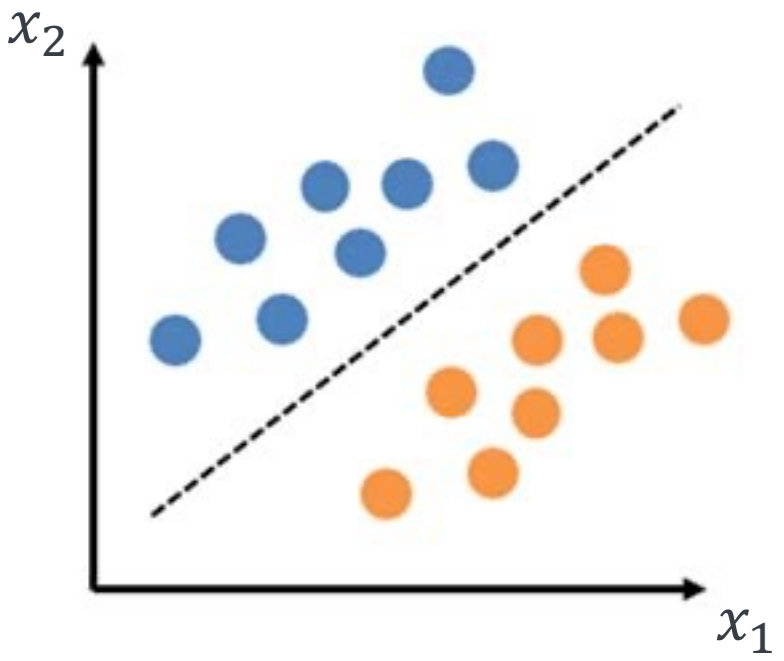


Linear Threshold Classifiers

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\boldsymbol{w} \cdot \boldsymbol{x} = w_0 + w_1 x_1 + w_2 x_2$$

(bias term)



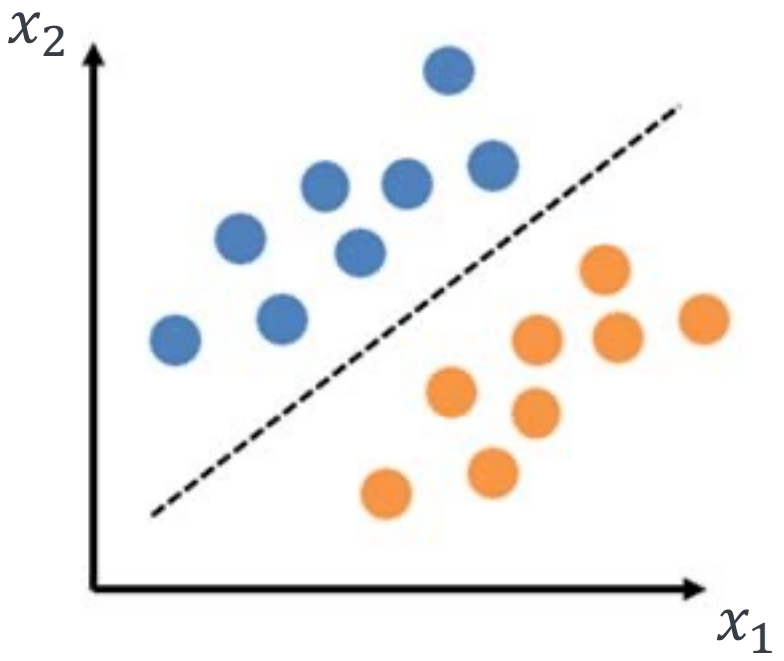
Linear Threshold Classifiers

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Learning problem:
Find weights, w, to minimize loss

$$Loss(h_{\boldsymbol{w}}) = L_2\big(y, h_{\boldsymbol{w}}(\boldsymbol{x})\big) = \sum_{j=1}^{N} \Big(y_j - h_{\boldsymbol{w}}\big(x_j\big)\Big)^2$$

$$w_i \leftarrow w_i + \alpha(y - h_{\boldsymbol{w}}(\boldsymbol{x}))x_i$$



Binary Perceptron

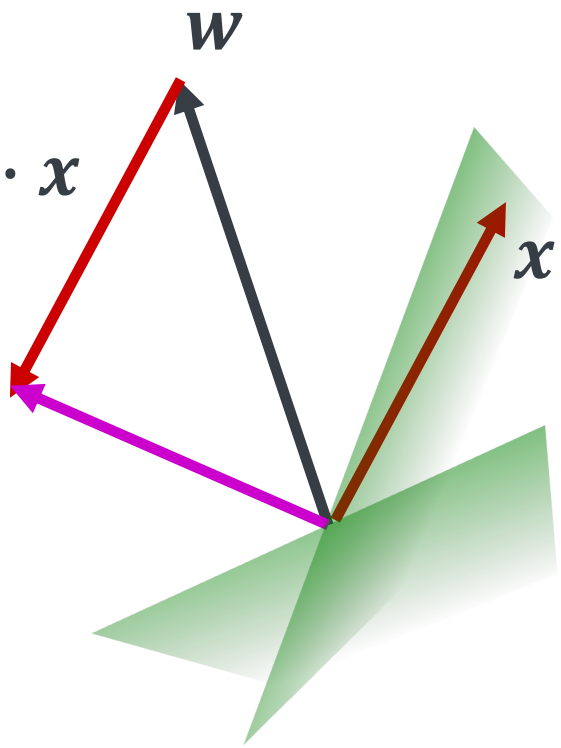
- Start with random weights
- For each training instance:
  - Classify with current weights

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

  - If correct (i.e., $y = h_{\boldsymbol{w}}(\boldsymbol{x})$), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector.

$$w_i \leftarrow w_i + (y - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot x_i$$

$$(\boldsymbol{y} - \boldsymbol{h_w}(\boldsymbol{x})) \cdot \boldsymbol{x}$$



Multiclass Perceptron

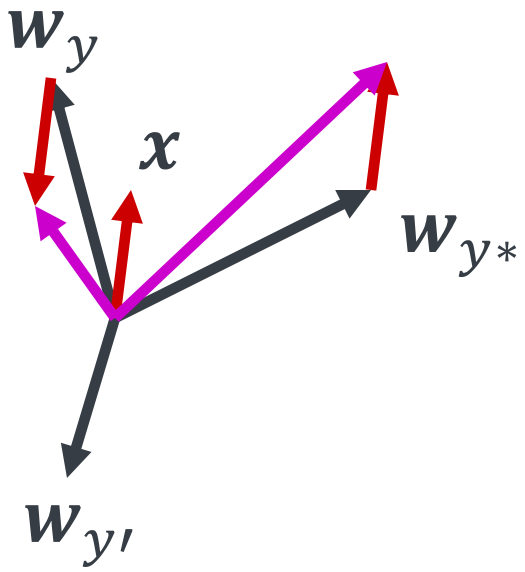
- Start with random weights
- Pick up training examples one by one
- Predict with current weights

$$y = \underset{y}{\operatorname{argmax}}\, \boldsymbol{w}_y \cdot \boldsymbol{x}$$

- If correct, no change!
- If wrong, lower score of wrong answer (y), raise score of right answer (y*)

$$\boldsymbol{w}_y = \boldsymbol{w}_y - \boldsymbol{x}$$

$$\boldsymbol{w}_{y^*} = \boldsymbol{w}_{y^*} + \boldsymbol{x}$$



Minimum Correcting Update

$$\min_{\boldsymbol{w}} \frac{1}{2} \sum_{y} \left\| \boldsymbol{w}_y - \boldsymbol{w}'_y \right\|^2$$

(**w**′ is the previous weights)

$$\boldsymbol{w}_{y^*} \cdot \boldsymbol{x} \geq \boldsymbol{w}_y \cdot \boldsymbol{x} + C$$

$$\boldsymbol{w}_y = \boldsymbol{w}'_y - \alpha \cdot \boldsymbol{x}$$
$$\boldsymbol{w}_{y^*} = \boldsymbol{w}'_{y^*} + \alpha \cdot \boldsymbol{x}$$

$$\min_{\alpha} \|\alpha \boldsymbol{x}\|^2$$
$$\boldsymbol{w}_{y^*} \cdot \boldsymbol{x} \geq \boldsymbol{w}_y \cdot \boldsymbol{x} + C$$

$$\left(\boldsymbol{w}'_{y^*} + \alpha \cdot \boldsymbol{x}\right) \cdot \boldsymbol{x} = \left(\boldsymbol{w}'_y - \alpha \cdot \boldsymbol{x}\right) \cdot \boldsymbol{x} + C$$

$$\alpha = \frac{\left(\boldsymbol{w}'_y - w'_{y*}\right) \cdot \boldsymbol{x} + C}{2\boldsymbol{x} \cdot \boldsymbol{x}}$$

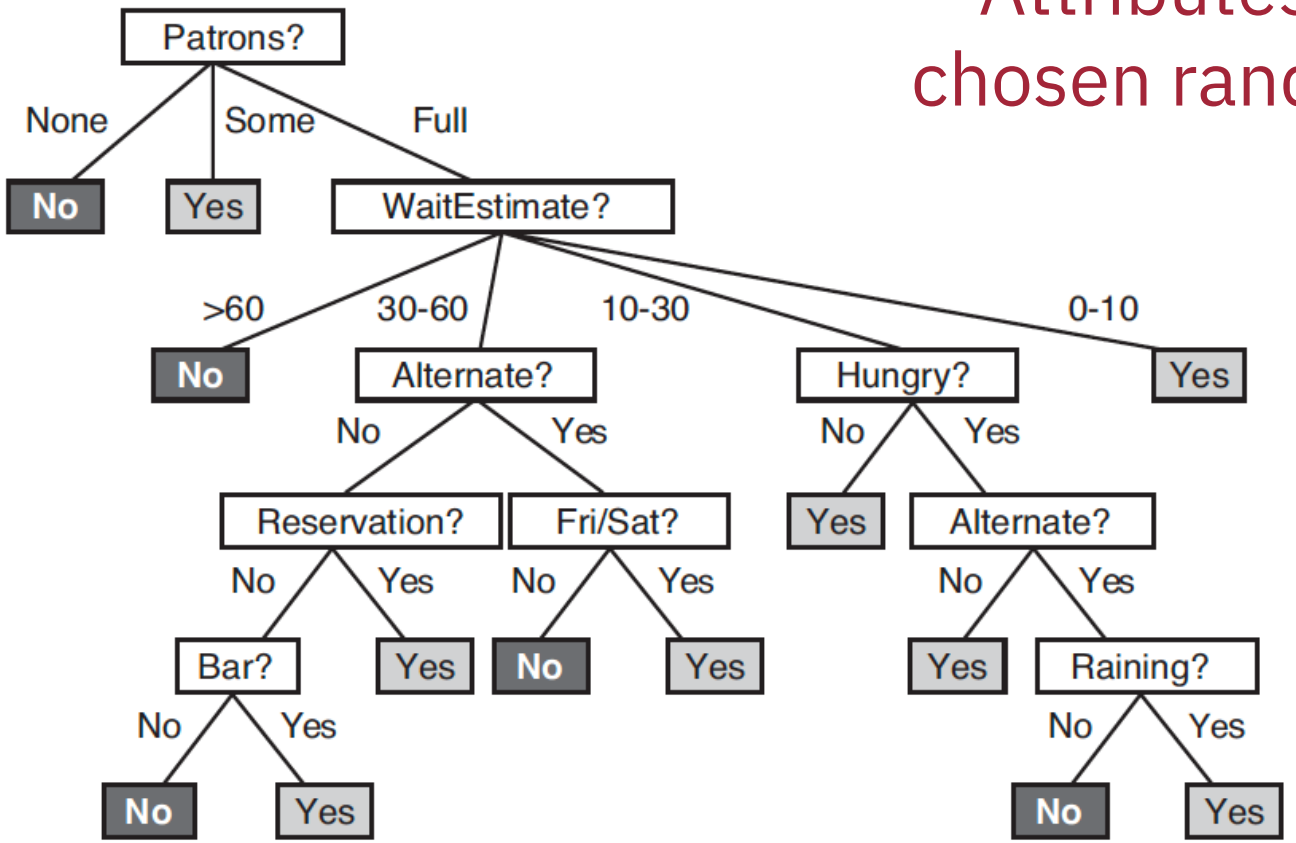
min of α will be where equality holds

Decision Tree

Example: Restaurants

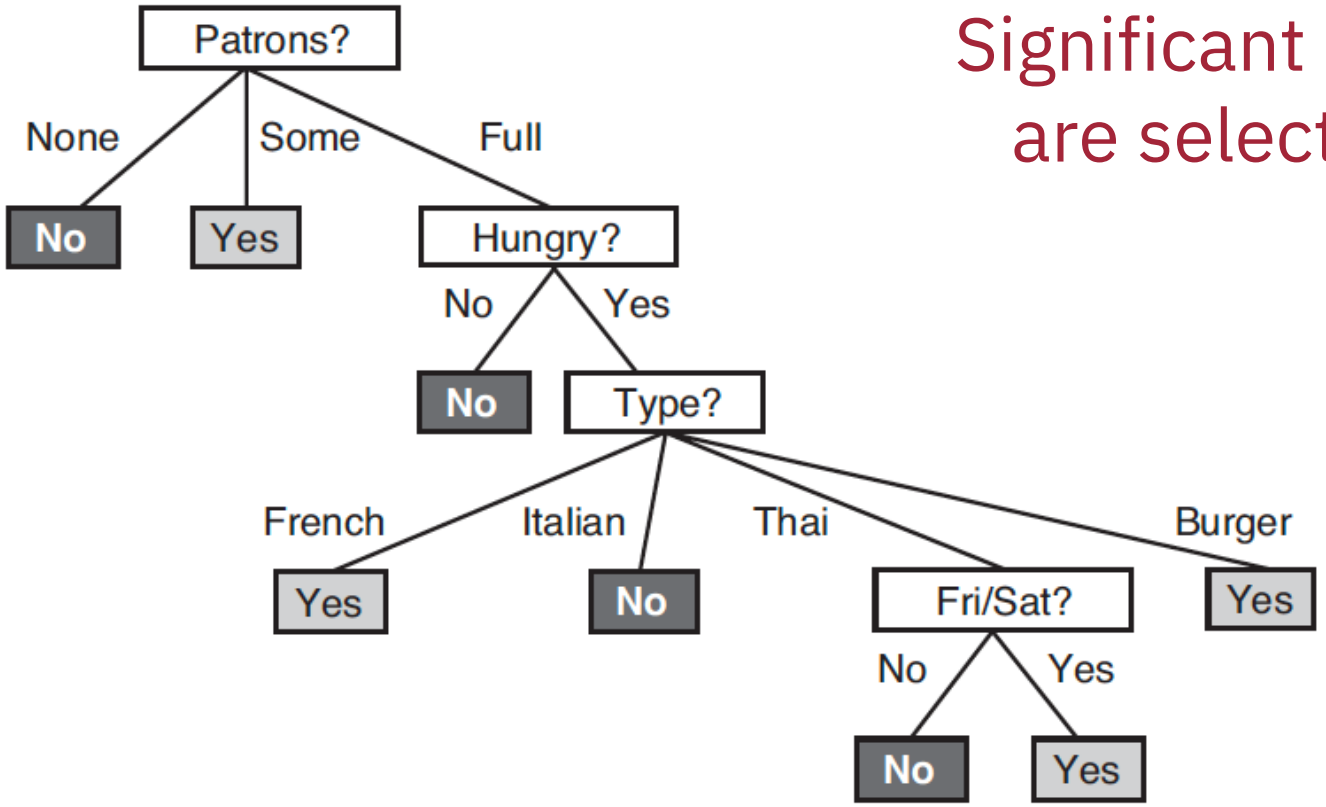
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Example: Restaurants



Attributes are chosen randomly.

Example: Restaurants



Significant attributes are selected first.

We can find a good approximate solution: a small (but not smallest) consistent tree.

Inducing a Decision Tree

- Aim: find a small tree consistent with the training examples
  - pick the attribute that goes as far as possible toward providing an **exact classification of the examples**
  - (recursively) choose the **most significant** attribute as root of (sub)tree

Using Information Theory

- Information content (entropy, a measure of the uncertainty of a random variable):

$$I\big(P(v_1), \ldots, P(v_2)\big) = \sum_i -P(v_i)\log_2 P(v_i)$$

- For training set containing p positive examples and n negative examples,

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information Gain

- Chosen attribute A divides the training set E into subsets E_1, …, E_v according to their values for A, where A has v distinct values.

$$remainder(A) = the\ expected\ entropy\ remaining\ after\ testing\ attribute\ A$$

$$= \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

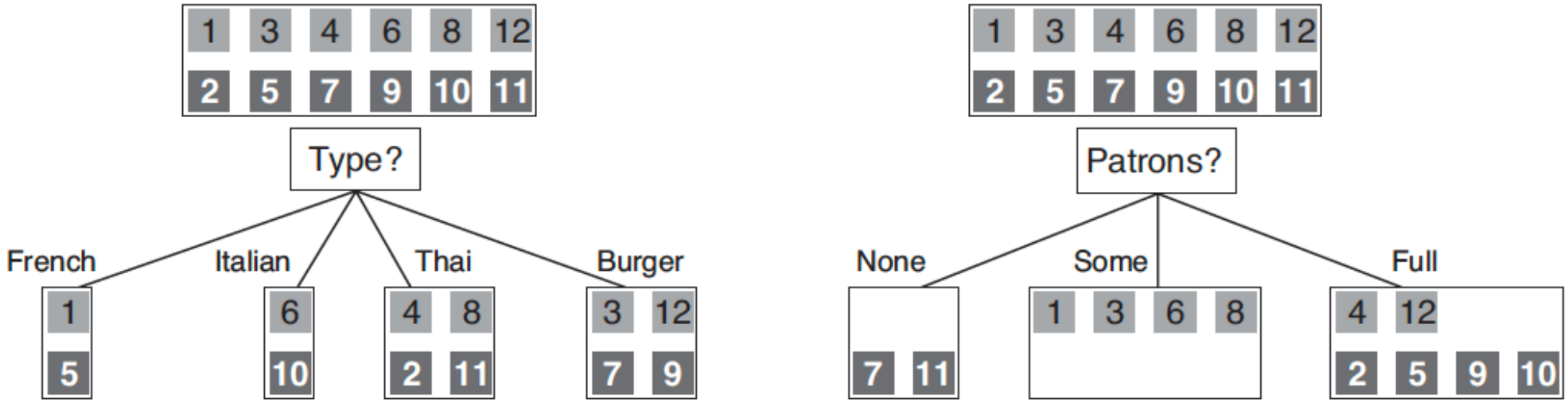
- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = Current\ entropy\ - remainder(A)$$

$$= I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

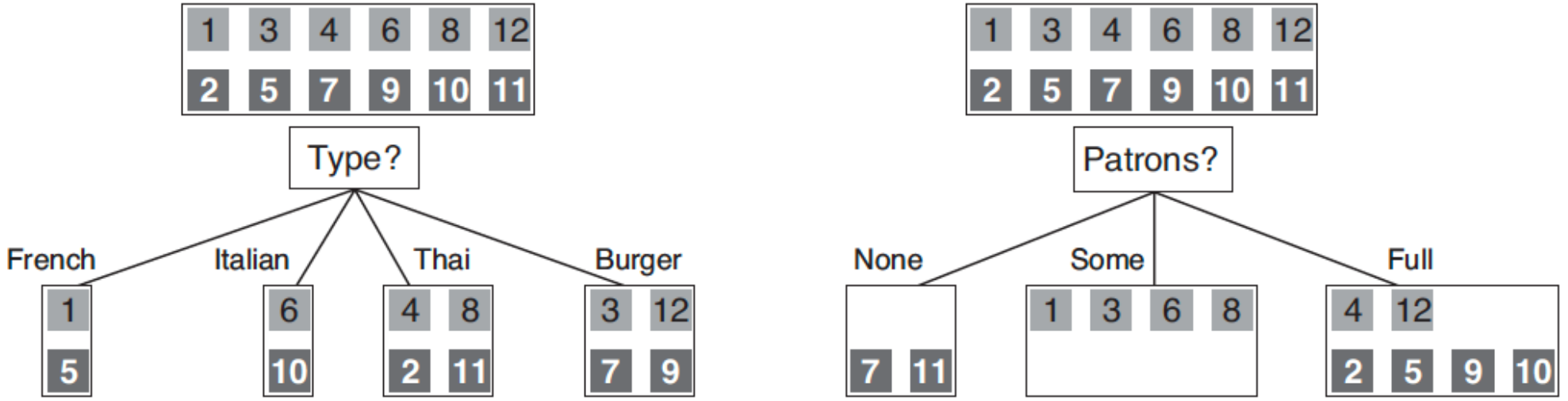
CHOOSE-ATTRIBUTE

- A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.



Choose the attribute with the highest information gain!

Which attribute is more significant?



$$IG(Type) = 1 - [\frac{2}{12} I\left(\frac{1}{2},\frac{1}{2}\right) + \frac{2}{12} I(\frac{1}{2},\frac{1}{2}) + \frac{4}{12} I(\frac{2}{4},\frac{2}{4}) + \frac{4}{12} I(\frac{2}{4},\frac{2}{4})] = 0$$

$$\leq$$

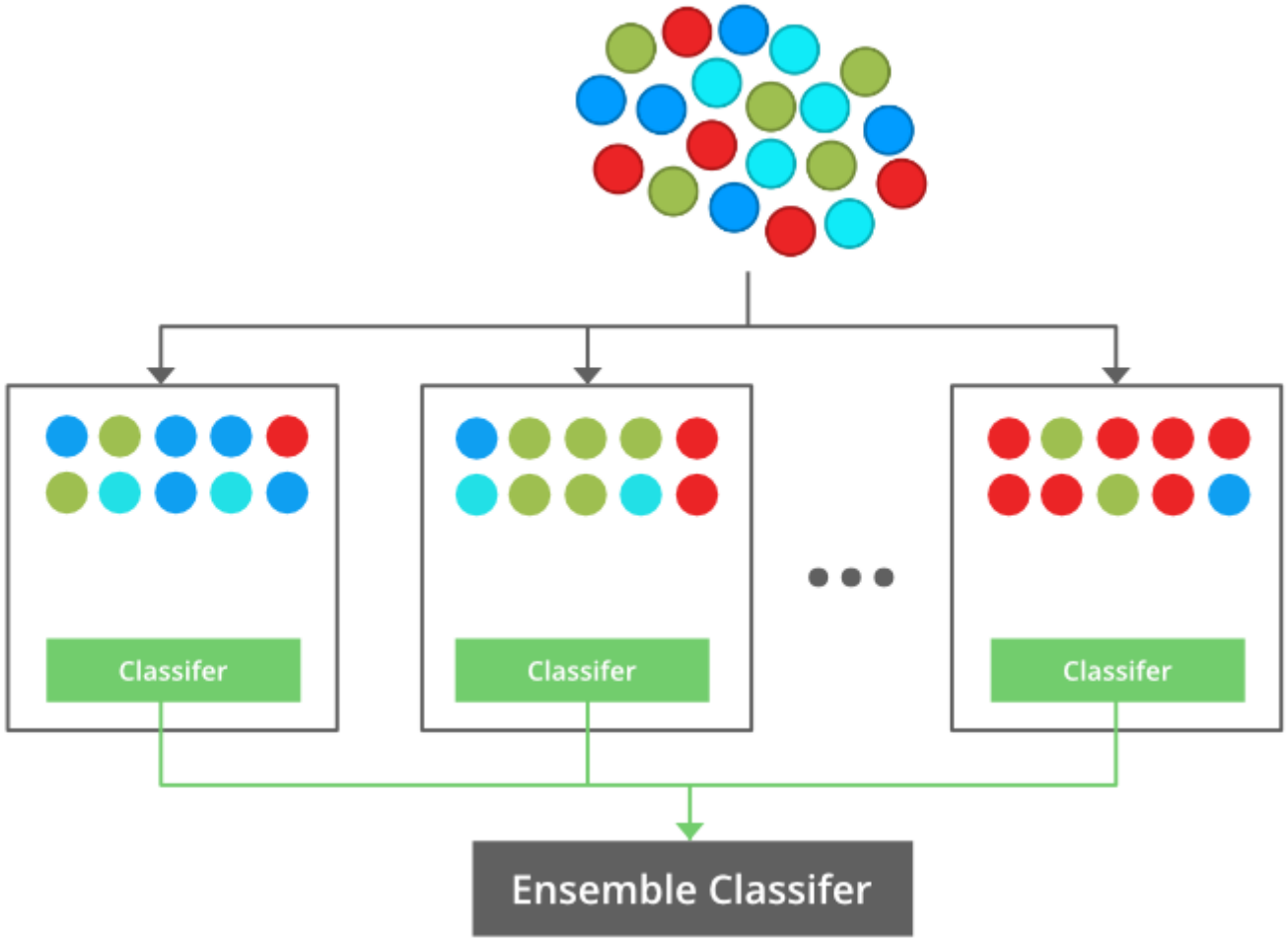
$$IG(Patrons) = 1 - [\frac{2}{12} I\left(\frac{0}{2},\frac{2}{2}\right) + \frac{4}{12} I(\frac{4}{4},\frac{0}{4}) + \frac{6}{12} I(\frac{2}{6},\frac{4}{6})] \approx 0.541$$

Ensemble Learning

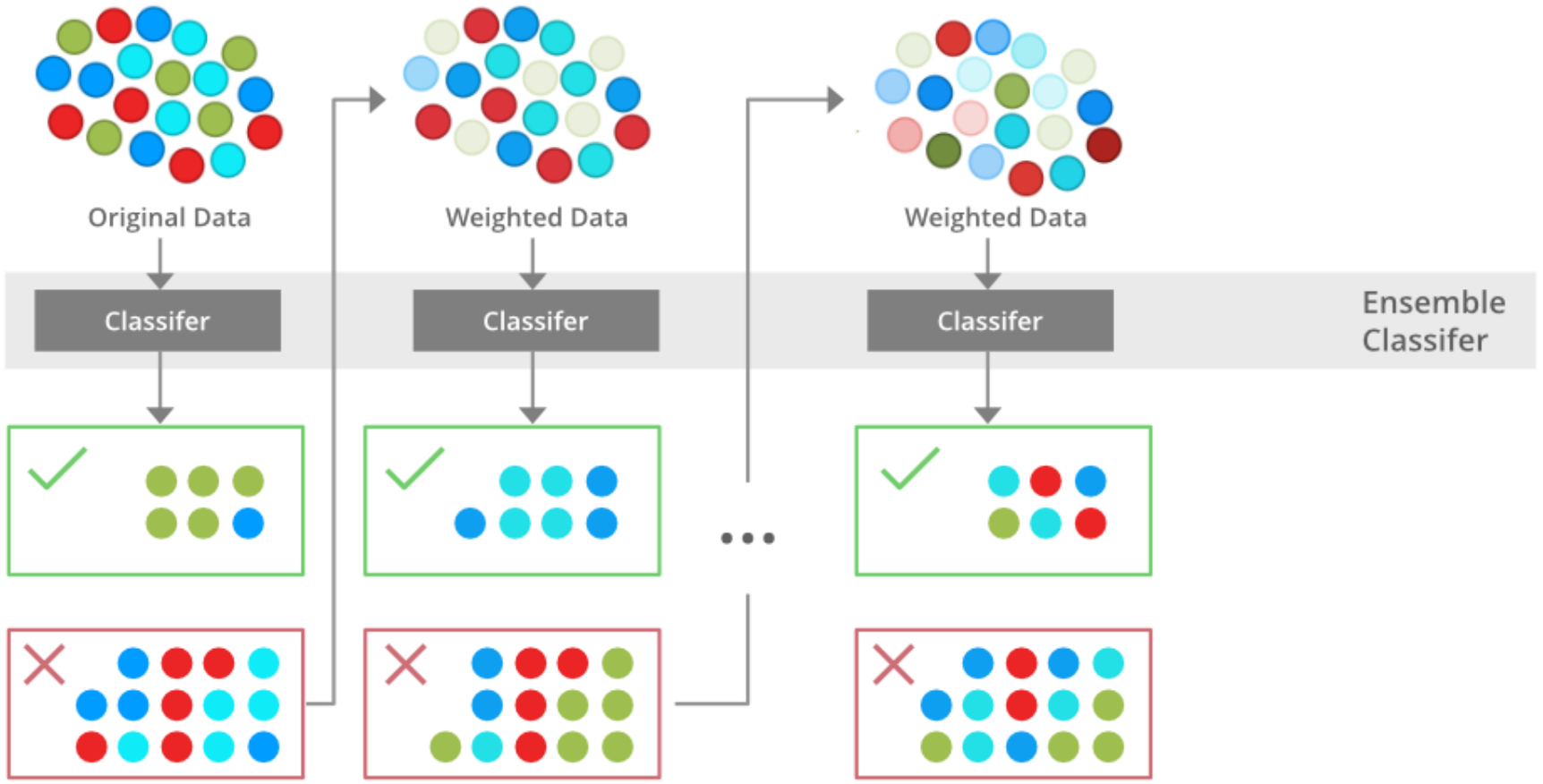
Ensemble Learning Techniques

- Bagging
- Boosting

Bagging

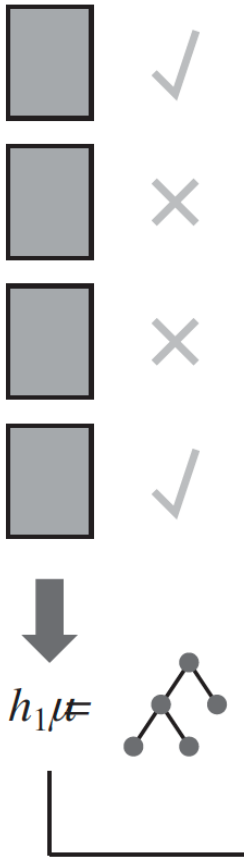


Boosting


Original Data
Weighted Data
Weighted Data
Classifer
Classifer
Classifer
Ensemble
Classifer

Visualizing the Process of Boosting

- **Shaded rectangle**: training example
  - **Height**: weight
  - **Checks and crosses**: whether the example was classified correctly by the current hypothesis.
- The size of the **decision tree**
  - weight of that hypothesis



AdaBoost

- AdaBoost with **K weak learning algorithms** returns a hypothesis that classifies the training data **perfectly** for large enough K.
  - **A week learning algorithm** always returns a hypothesis with accuracy on the training set that is slightly better than random guessing (i.e., 50%+ for Boolean classification).
- AdaBoost boosts the accuracy of the original learning algorithm on the training data.

AdaBoost

function ADABOOST($examples, L, K$) **returns** a weighted-majority hypothesis
  inputs: $examples$, set of N labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    K, the number of hypotheses in the ensemble
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of K hypotheses
    z, a vector of K hypothesis weights

  for $k = 1$ **to** K **do**
    $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE(**w**)
    $\mathbf{z}[k] \leftarrow \log(1 - error)/error$
  return WEIGHTED-MAJORITY(**h**, **z**)

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Support Vector Machine (SVM)

Chapter 18

Review of Vector Calculus

- Partial derivatives
  - For a multivariable function, like $f(x, y) = x^2 y$, computing partial derivatives looks something like this.

$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial}{\partial x} x^2 y}_{\text{Treat } y \text{ as constant; take derivative.}} = 2xy$$

$$\frac{\partial f}{\partial y} = \underbrace{\frac{\partial}{\partial y} x^2 y}_{\text{Treat } x \text{ as constant; take derivative.}} = x^2 \cdot 1$$

Review of Vector Calculus

- Chain rule
  - Given a multivariable function $f(x, y)$, and two single variable functions $x(t)$ and $y(t)$, here's what the multivariable chain rule says:

$$\underbrace{\frac{d}{dt} f(x(t), y(t))}_{\text{Derivative of composition function}} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

  - Written with vector notation, where $\boldsymbol{v}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$, this rule has a form in terms of the gradient of f and the vector-derivative of $\boldsymbol{v}(t)$.

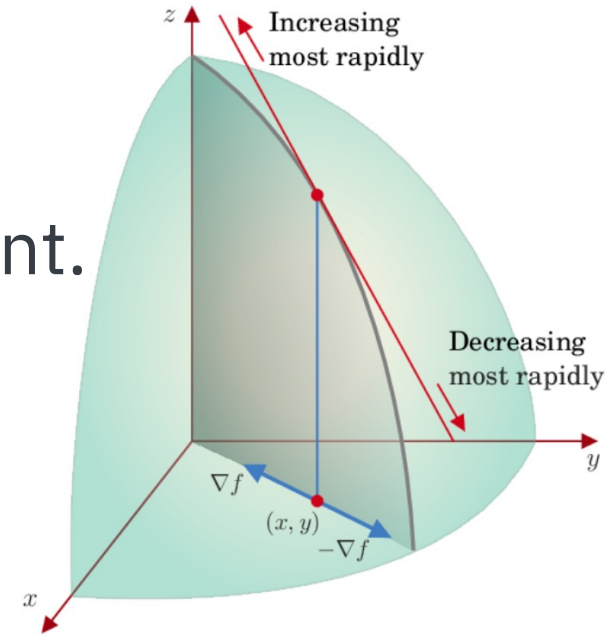
$$\underbrace{\frac{d}{dt} f(\vec{\mathbf{v}}(t))}_{\text{Derivative of composition function}} = \overbrace{\nabla f \cdot \vec{\mathbf{v}}'(t)}^{\text{Dot product of vectors}}$$

Review of Vector Calculus

- The gradient of a function f (denoted as ∇f)
  - Collection of all its partial derivatives into a vector

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial x}\right)$$

  - The gradient is the slope of the steepest ascent.



If f(x,y)=x^2-xy, which of the following represents ∇ f?

(2x-x, x^2-y)

(2x-y, -x)

(x^2, xy)

(x^2, -xy)

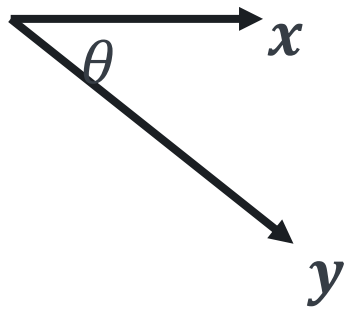
Review of Linear Algebra

- Matrix multiplication

$$x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$$

$$x^T y = x \cdot y = \|x\| \|y\| \cos\theta$$

(inner product)



- Transpose

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}^{\mathrm{T}}$$

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A + B)^T = A^T + B^T$

Review of Linear Algebra

- Methods to represent the magnitude of vectors

- L0 norm
  - Total number of nonzero elements in a vector
  - $\|x\|_0 = \|(3, -4)\|_0 = 2$

- L1 norm
  - The sum of the magnitudes of the elements (*i.e.*, Manhattan distance)
  - $\|x\|_1 = \|(3, -4)\|_1 = |3| + |-4| = 7$

- L2 norm
  - The shortest distance to go from one point to another (*i.e.*, Euclidean norm).
  - $\|x\|_2 = \|(3, -4)\|_2 = \sqrt{3^2 + (-4)^2} = 5$

- L-infinity norm
  - The largest magnitude among each element of a vector
  - $\|y\|_\infty = \|(-6, 4, 2)\|_\infty = 6$

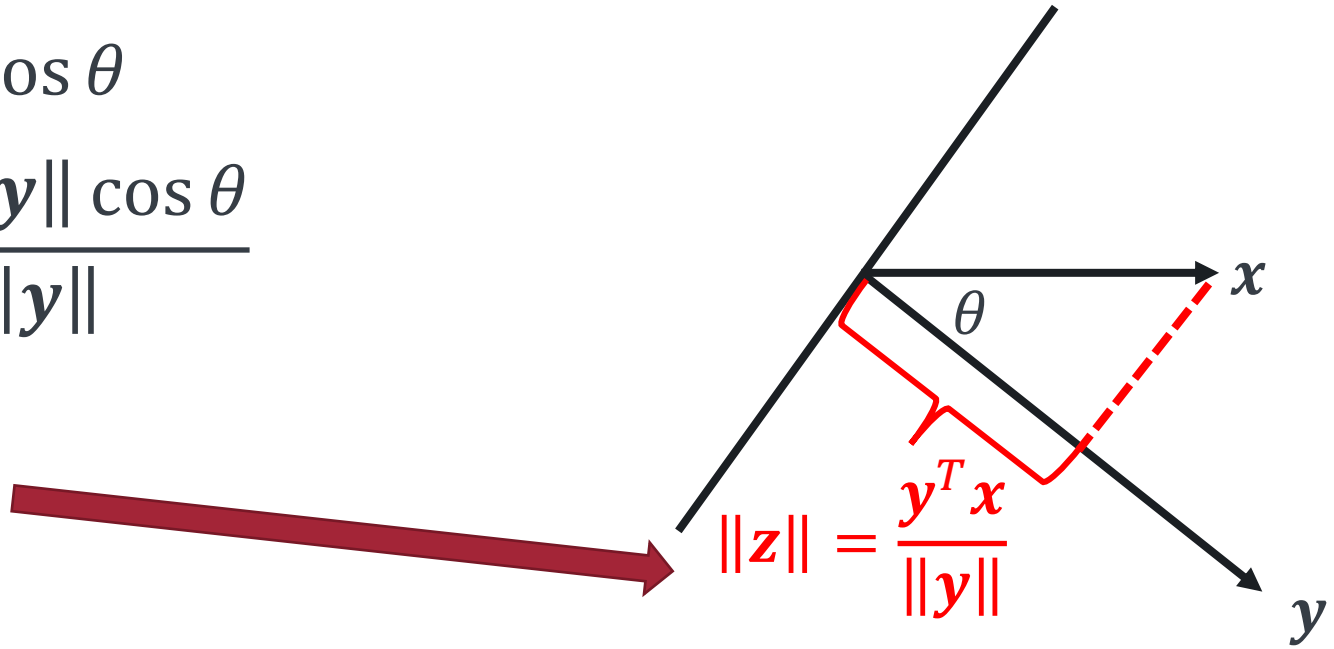
Review of Linear Algebra

- Projection

$$\|\boldsymbol{z}\| = \|\boldsymbol{x}\| \cos\theta$$

$$= \frac{\|\boldsymbol{x}\|\|\boldsymbol{y}\| \cos\theta}{\|\boldsymbol{y}\|}$$

$$= \frac{\boldsymbol{y}^T\boldsymbol{x}}{\|\boldsymbol{y}\|}$$

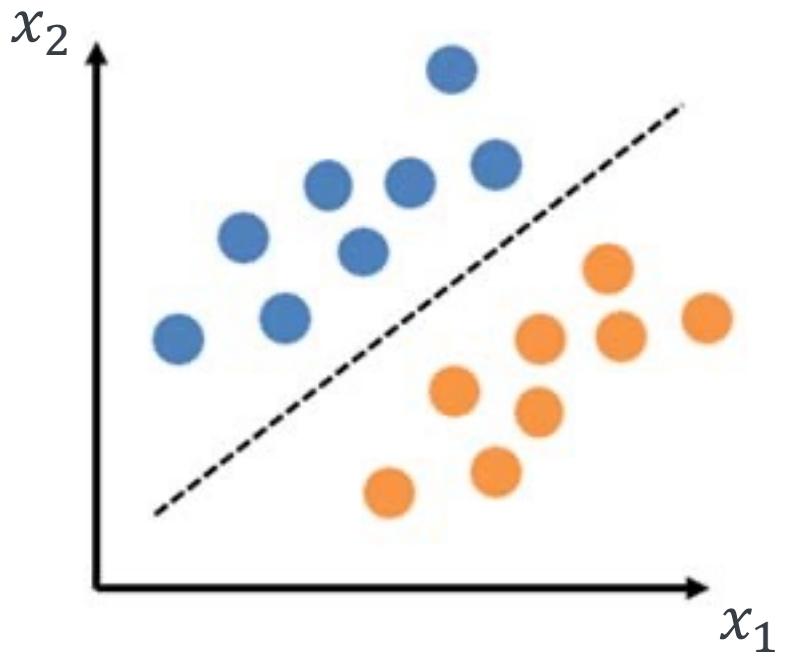


Linear Threshold Classifiers

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} = w_0 + w_1 x_1 + w_2 x_2$$

(bias term)



Optimize to find w!

What are the prediction results?

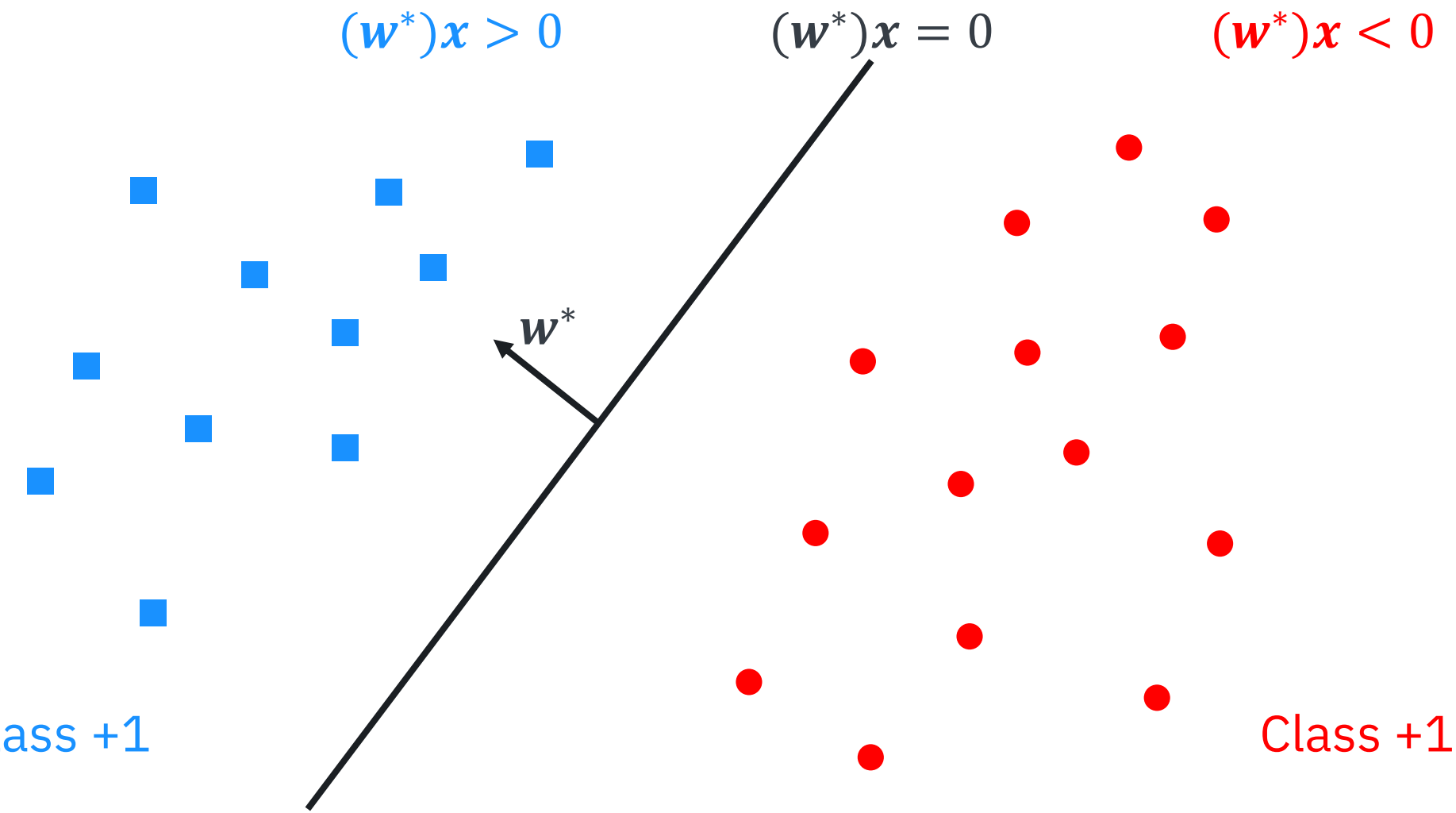
$$w_0 = 0, w_1 = 1, w_2 = -2$$

$$x = (x_1, x_2) = (1,0)$$

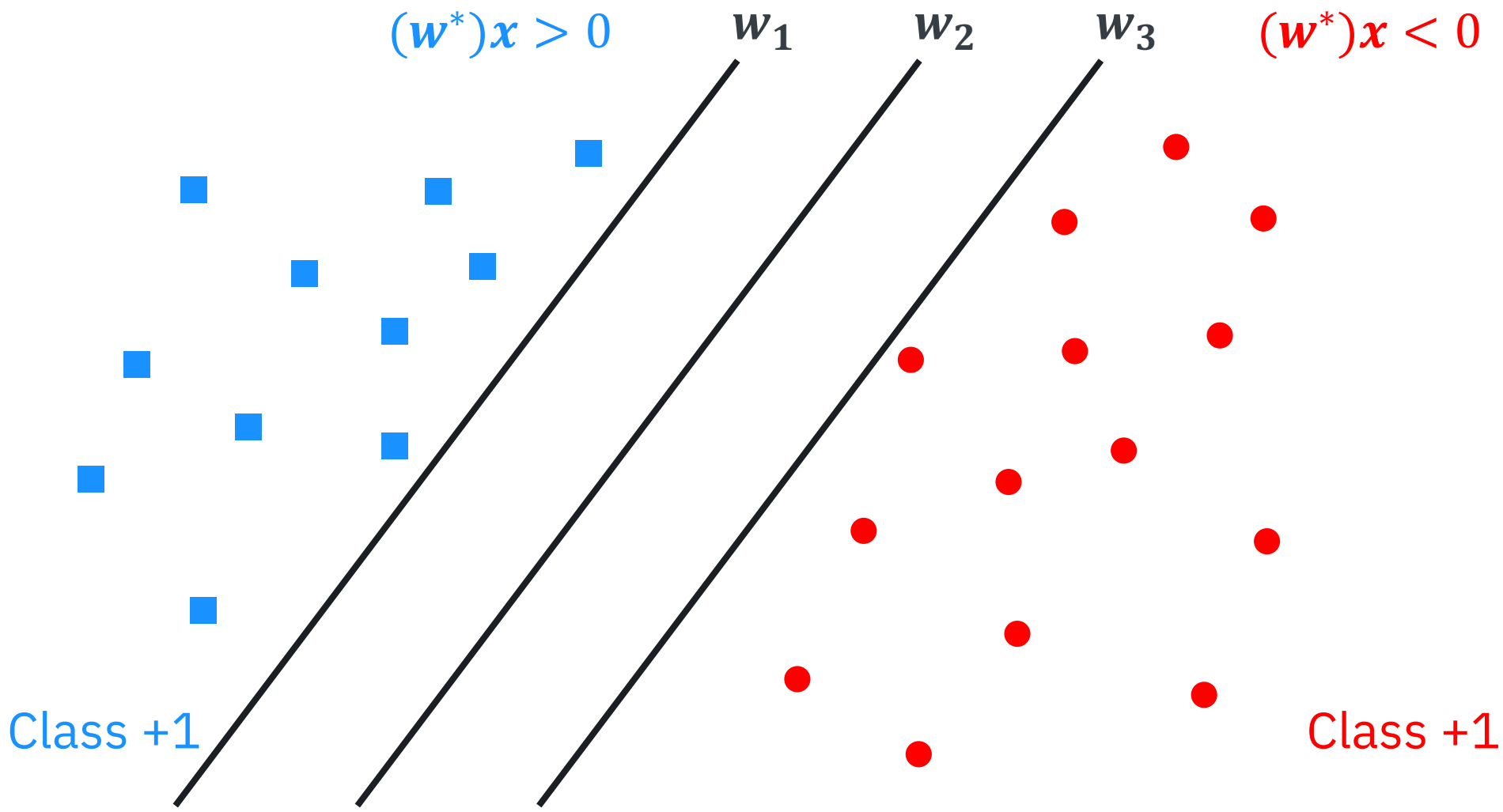
$$x' = (x'_1, x'_2) = (1,1)$$

- h(x)=1, h(x')=1
- h(x)=1, h(x')=-1
- h(x)=-1, h(x')=1
- h(x)=-1, h(x')=-1

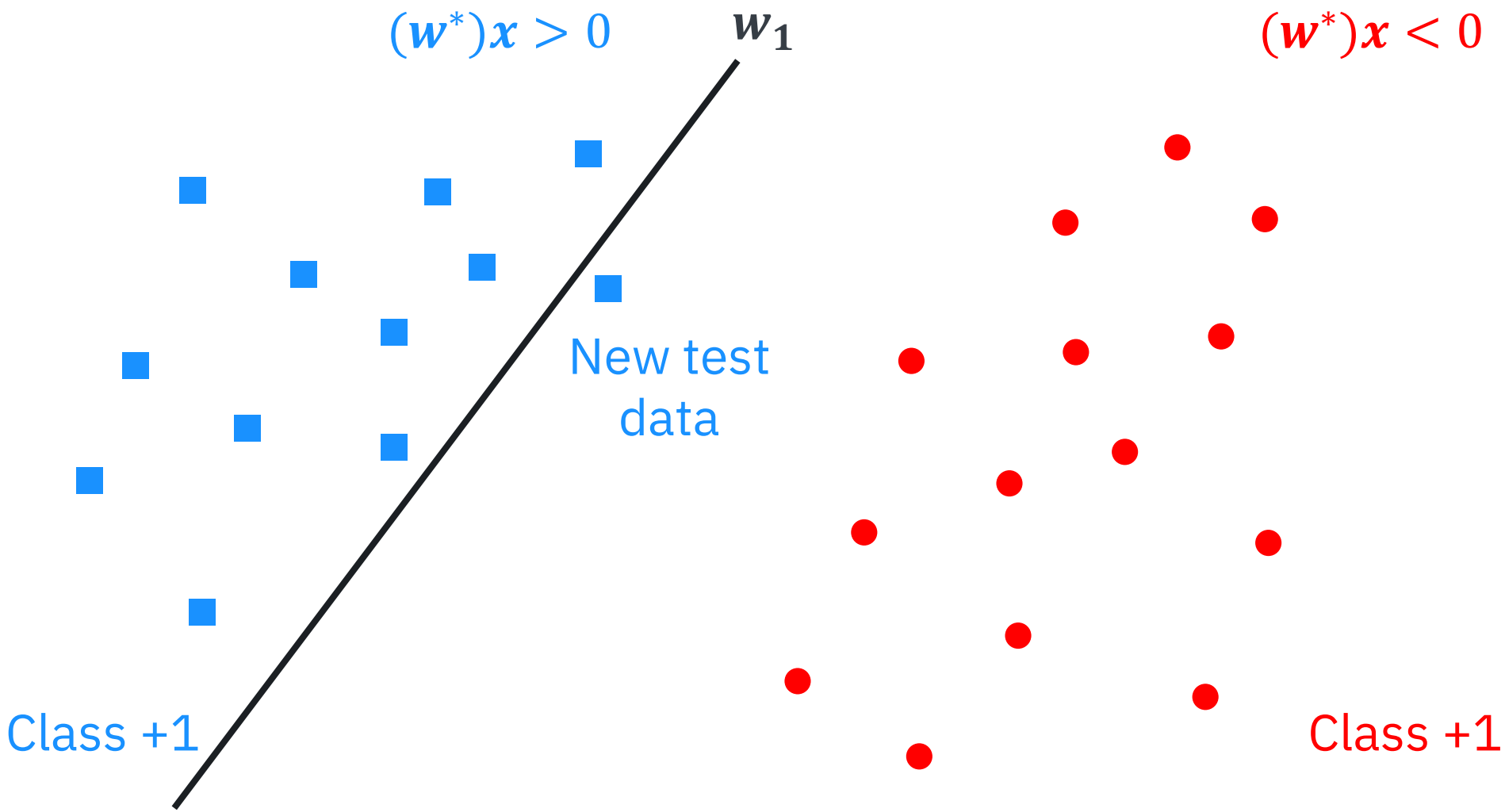
Linear Threshold Classifiers



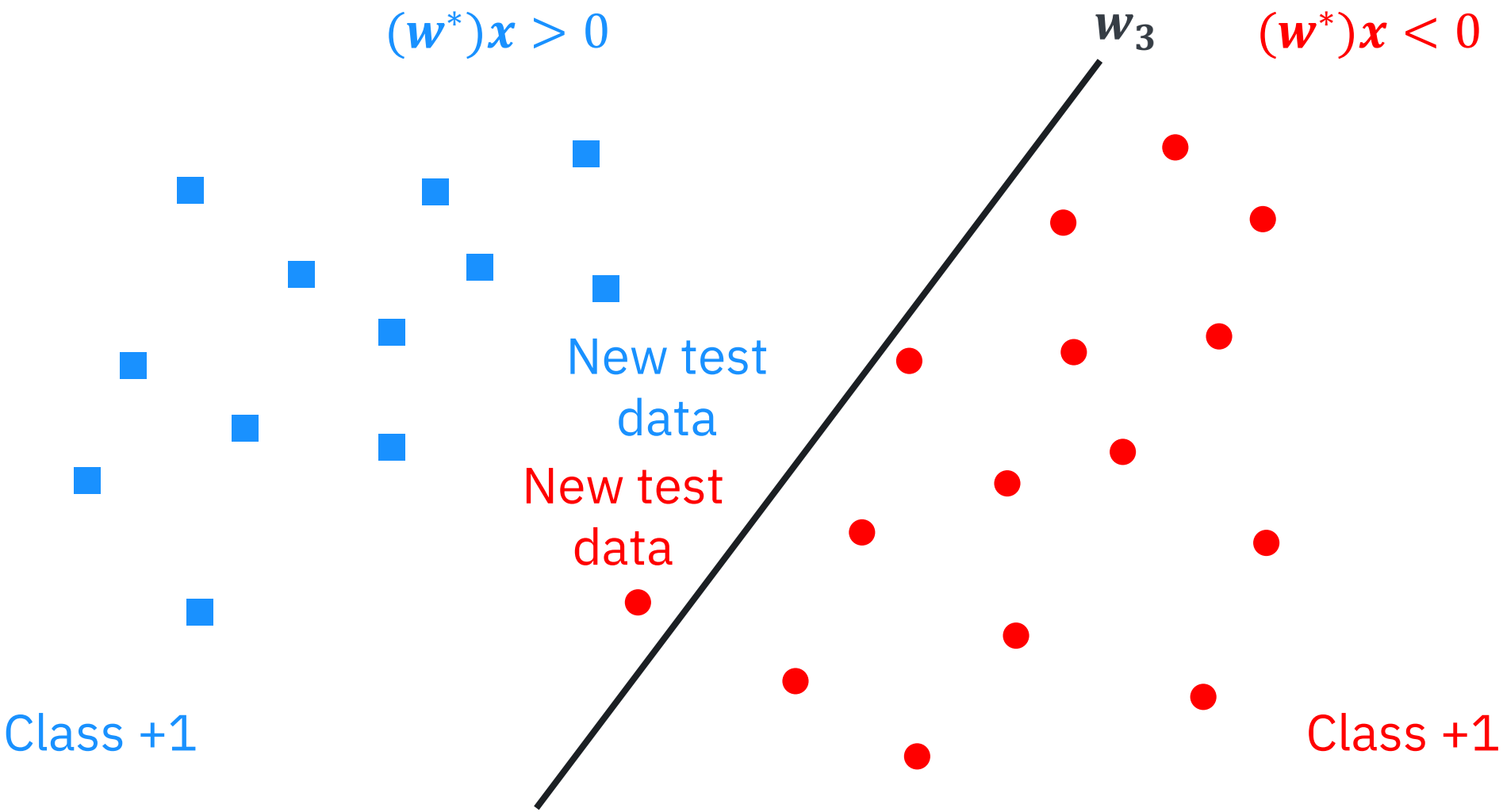
More Optimal Solutions?



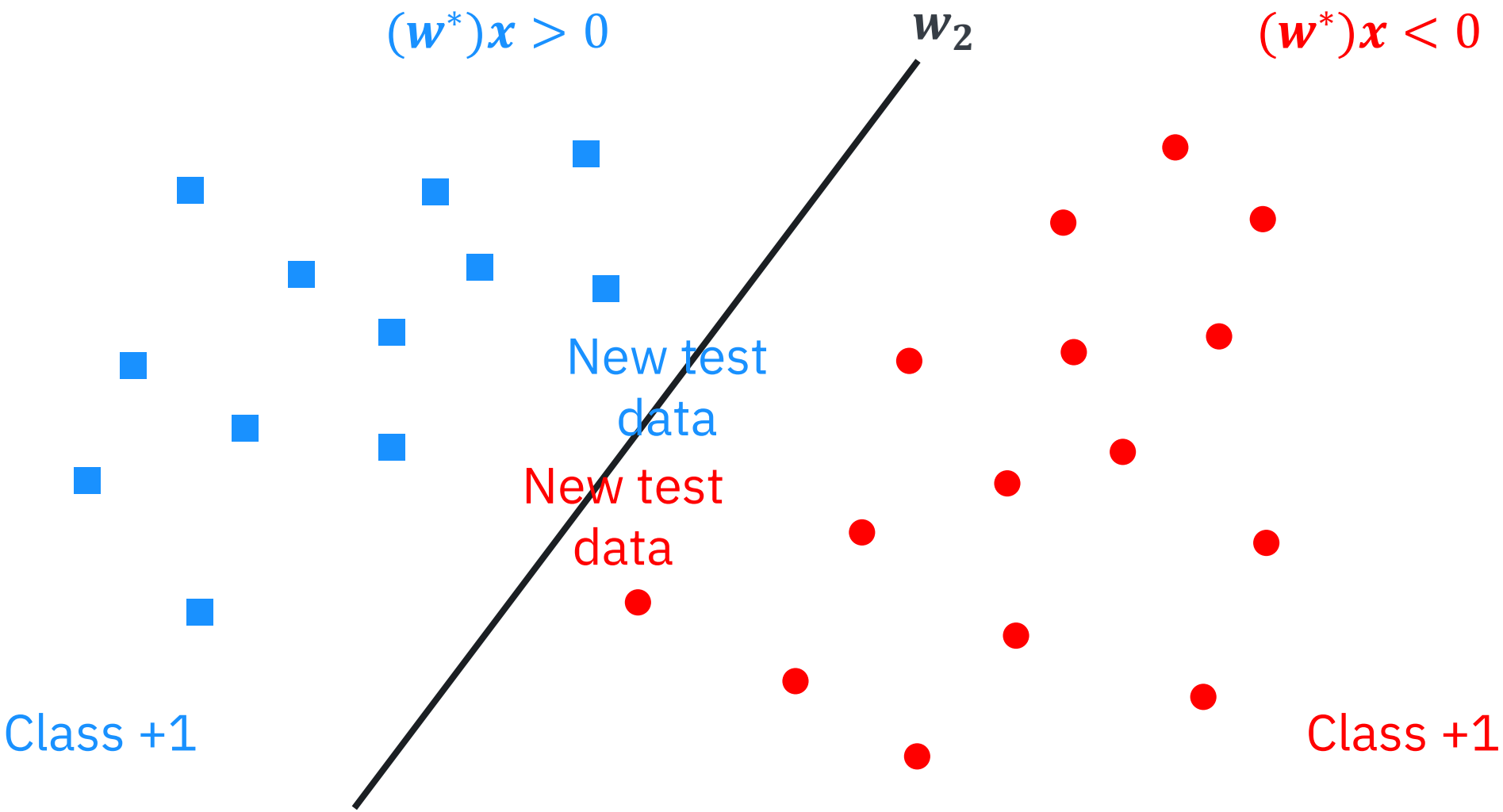
What About w_1?



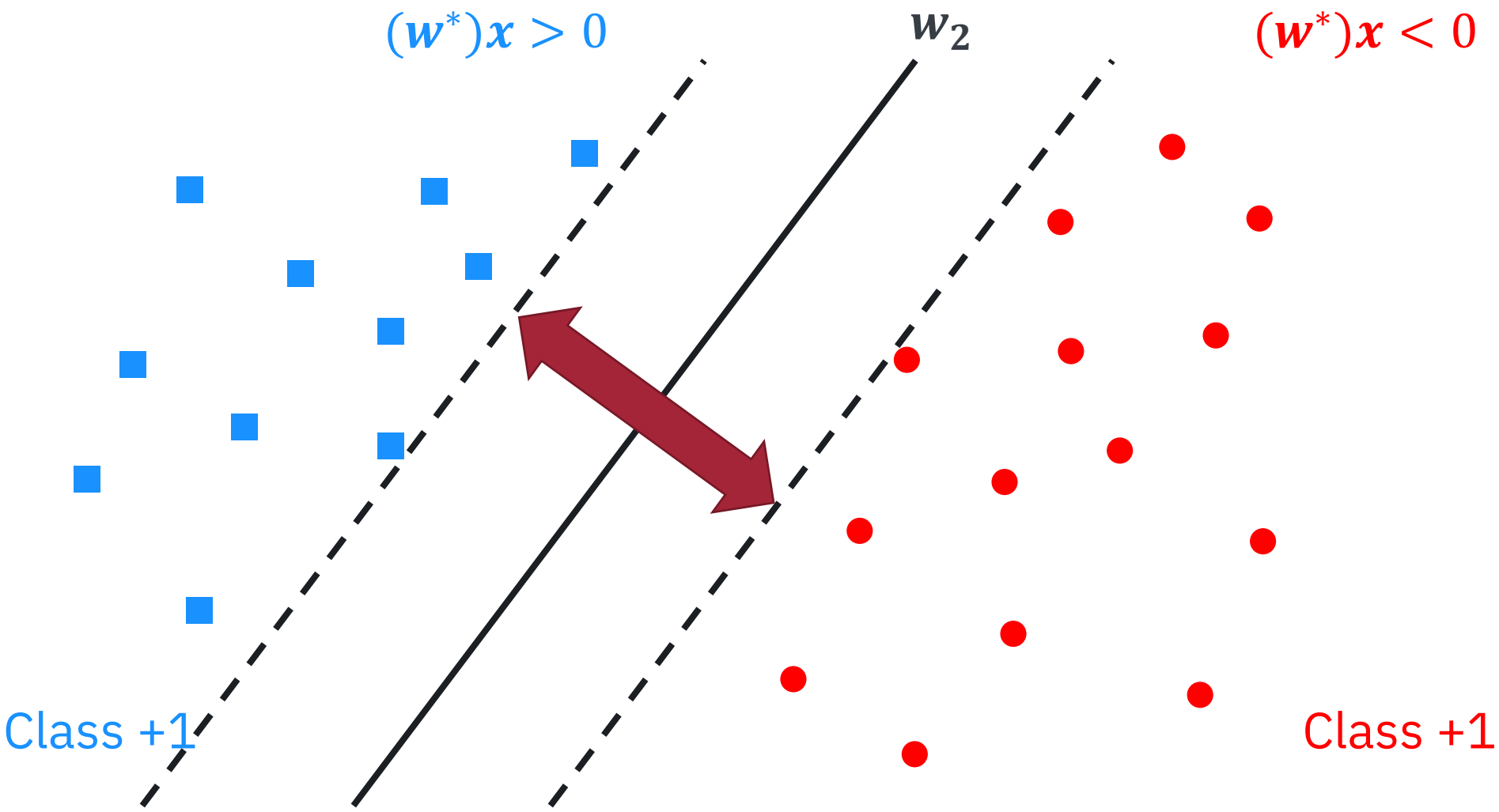
What About w_3?



Most Confident: w_2



Intuition: Margin



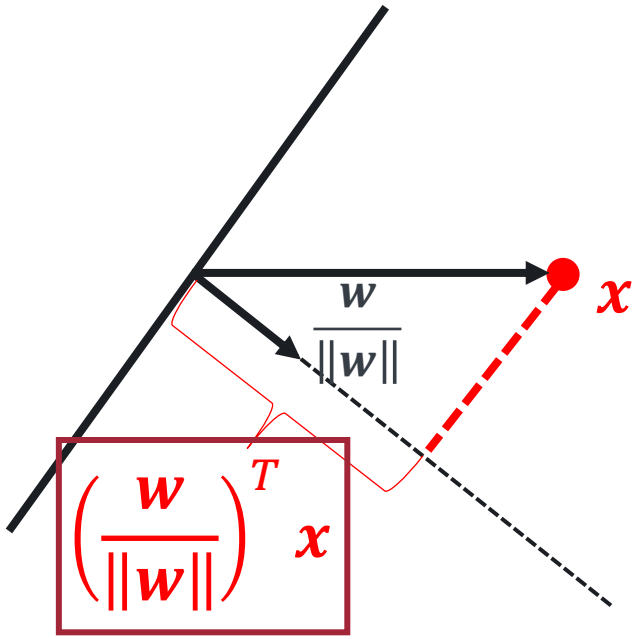

Support
Vector
Machine
In 2 min

Margin

$\boldsymbol{x}$ has distance $\frac{|f_{\boldsymbol{w}}(\boldsymbol{x})|}{\|\boldsymbol{w}\|}$ to the hyperplane $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} = 0$.

Proof:

- $\boldsymbol{w}$ is orthogonal to the hyperplane.
- The unit direction is $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$.
- The projection of $\boldsymbol{x}$ is $\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\right)^T \boldsymbol{x} = \frac{f_{\boldsymbol{w}}(\boldsymbol{x})}{\|\boldsymbol{w}\|}$.



SVM: Objective

- Margin over all training data points

$$\gamma = \min_i \frac{|f_{\boldsymbol{w}}(\boldsymbol{x}_i)|}{\|\boldsymbol{w}\|}$$

- Since only want correct $f_{\boldsymbol{w}}$, and recall $y_i \in \{+1, -1\}$, we have

$$\gamma = \min_i \frac{y_i f_{\boldsymbol{w}}(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

- If $f_{\boldsymbol{w}}$ incorrect on some $\boldsymbol{x}_i$, the margin is negative. Now, we assume that $f_{\boldsymbol{w}}$ is correct for all $\boldsymbol{x}_i$.

SVM: Objective

- Maximize margin over all training data points

$$\max_{\boldsymbol{w}} \gamma = \max_{\boldsymbol{w}} \min_{i} \frac{y_i f_{\boldsymbol{w}}(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|} = \max_{\boldsymbol{w}} \min_{i} \frac{y_i(\boldsymbol{w}^T \boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

- A bit complicated …

SVM: Simplified Objective

- Observation
  - When **w** is scaled by a factor *c*, the margin unchanged.

$$\frac{y_i(c\boldsymbol{w}^T\boldsymbol{x}_i)}{\|c\boldsymbol{w}\|} = \frac{y_i(\boldsymbol{w}^T\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

- Let's consider a fixed scale such that

$$y_{i^*}(\boldsymbol{w}^T\boldsymbol{x}_{i^*}) = 1$$

where $\boldsymbol{x}_{i^*}$ is the point closest to the hyperplane.

SVM: Simplified Objective

- Let's consider a fixed scale such that

$$y_{i^*}(\boldsymbol{w}^T \boldsymbol{x}_{i^*}) = 1$$

where $\boldsymbol{x}_{i^*}$ is the point closest to the hyperplane.

- Now we have for all data

$$y_{i^*}(\boldsymbol{w}^T \boldsymbol{x}_{i^*}) \geq 1$$

and at least for one i the equality holds.

- The margin is $\frac{1}{\|\boldsymbol{w}\|}$.

SVM: Simplified Objective

- Optimization simplified to

$$\min_{\boldsymbol{w}} \frac{1}{2} \|\boldsymbol{w}\|^2$$

(Using L2 norm instead of L1 norm because L2 norm is differentiable.)

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i) \geq 1, \forall i$$

- How to find the optimum $\hat{\boldsymbol{w}}^*$?

SVM: Optimization

- Optimization simplified to

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i) \geq 1, \forall i$$

- Solved by Lagrange multiplier method

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i [y_i \boldsymbol{w}^T x_i - 1]$$

Learning Using Kernels

- Conditions of optimal solution (Karush–Kuhn–Tucker conditions)

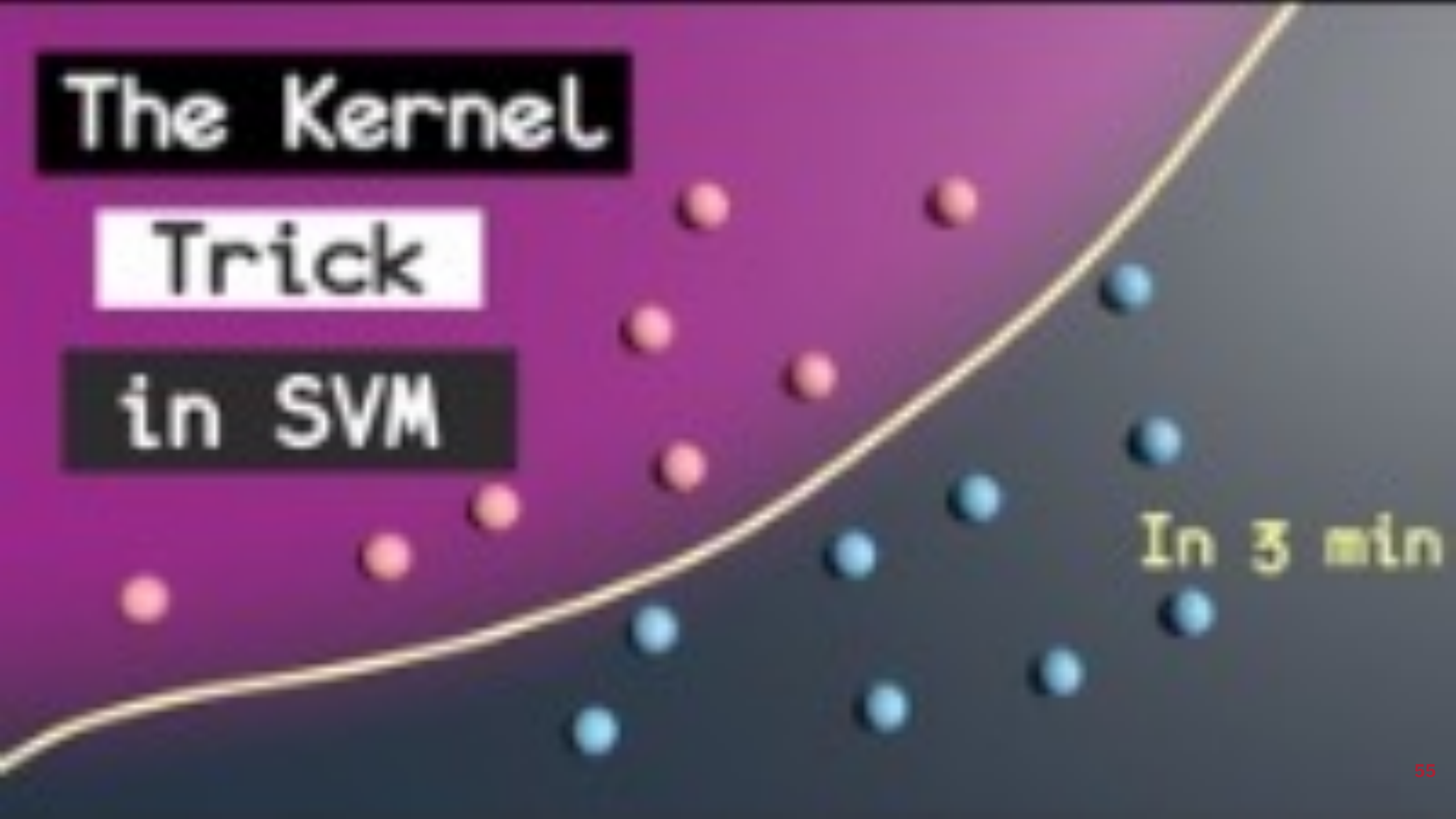
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0$$

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$$

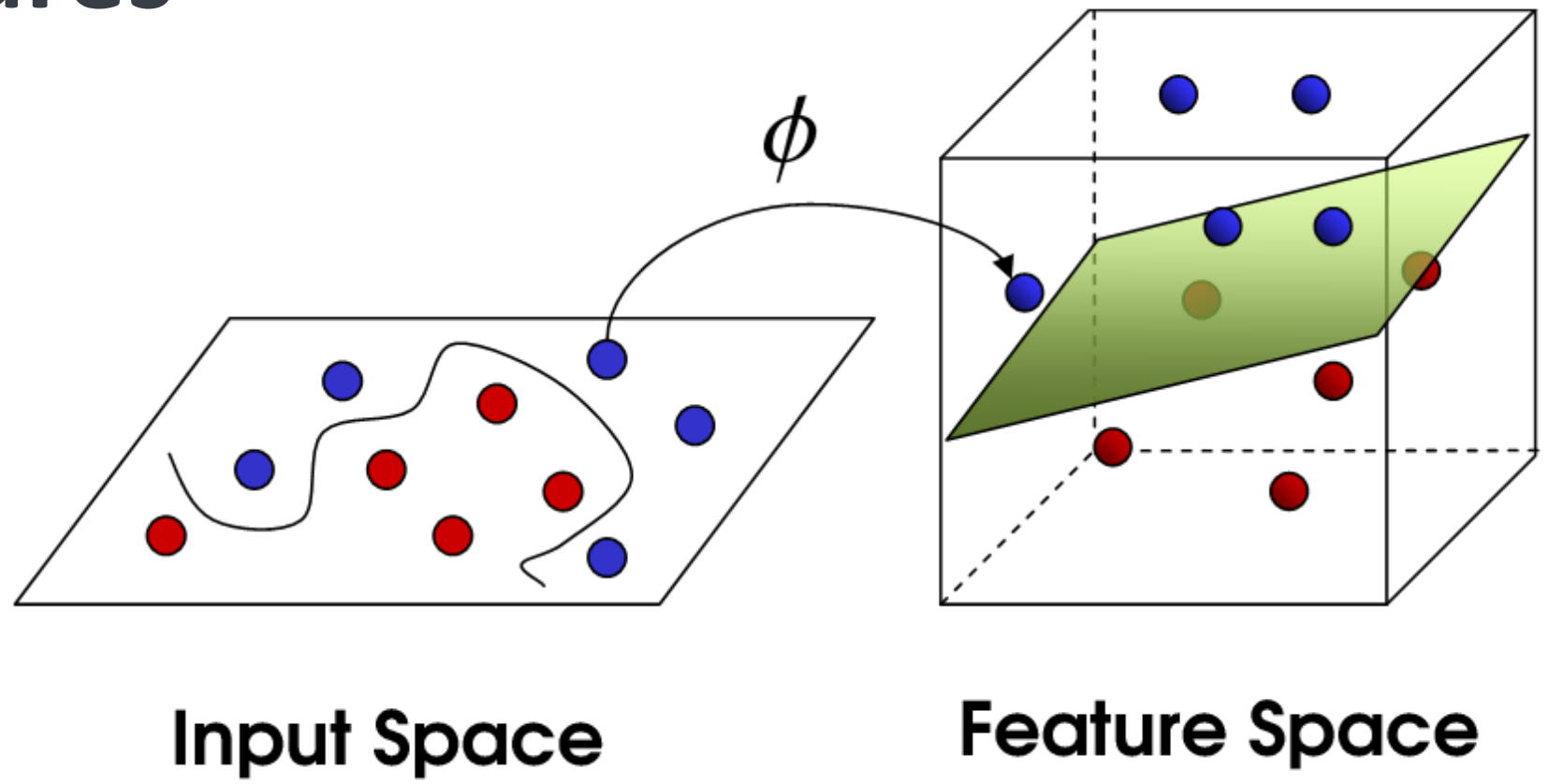
With $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha})$ and $f_{\boldsymbol{w}}(\boldsymbol{x})$, we can find the solution of the optimization problem.

(The mathematical derivation of the solution is beyond the scope of this class.)

Kernel Methods


The Kernel
Trick
in SVM
In 3 min

Features



$$\phi: (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Features

- Proper feature mapping can make non-linear to linear.
- **Problem: high computational load with many features!**

Features

- Kernel Method
  - Using SVM on the feature space $\{\phi(x_i)\}$: only need $\phi(x_i)^T\phi(x_j)$.
  - No need to design $\phi(x_i)$, only need to design

$$K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$$

Learning Using Kernels

- SVM optimization simplified to

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i) \geq 1, \forall i$$

- Solved by Lagrange multiplier method

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i [y_i \boldsymbol{w}^T x_i - 1]$$

Learning Using Kernels

- Conditions of optimal solution (Karush–Kuhn–Tucker conditions)

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0$$

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$$

Learning Using Kernels

- Plug into $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

- Plug into f:

$$f_{\boldsymbol{w}}(\boldsymbol{x}_i) = \boldsymbol{w}^T \boldsymbol{x}_i = \sum_j \alpha_j y_j \boldsymbol{x}_j^T \boldsymbol{x}_i$$

Only depend on inner product

Learning Using Kernels

- Kernel function

$$K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$$

- Both $\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha})$ and $f_{\boldsymbol{w}}(\phi(\boldsymbol{x}))$ depend on $\phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$ only.
  - You do not need to know $\phi(\boldsymbol{x})$ if you know $K(\boldsymbol{x}, \boldsymbol{x}')$ to solve the optimization problem!

Learning Using Kernels

- Plug into $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

- Plug into f:

$$f_{\boldsymbol{w}}(\boldsymbol{x}_i) = \boldsymbol{w}^T \boldsymbol{x}_i = \sum_j \alpha_j y_j \boldsymbol{x}_j^T \boldsymbol{x}_i$$

Only depend on inner product

Learning Using Kernels

- Plug into $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{\phi}(\boldsymbol{x}_i)^T \boldsymbol{\phi}(\boldsymbol{x}_j)$$

- Plug into f:

$$f_{\boldsymbol{w}}(\boldsymbol{\phi}(\boldsymbol{x}_i)) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) = \sum_j \alpha_j y_j \boldsymbol{\phi}\left(\boldsymbol{x}_j\right)^T \boldsymbol{\phi}(\boldsymbol{x}_i)$$

Learning Using Kernels

- Plug into $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

- Plug into f:

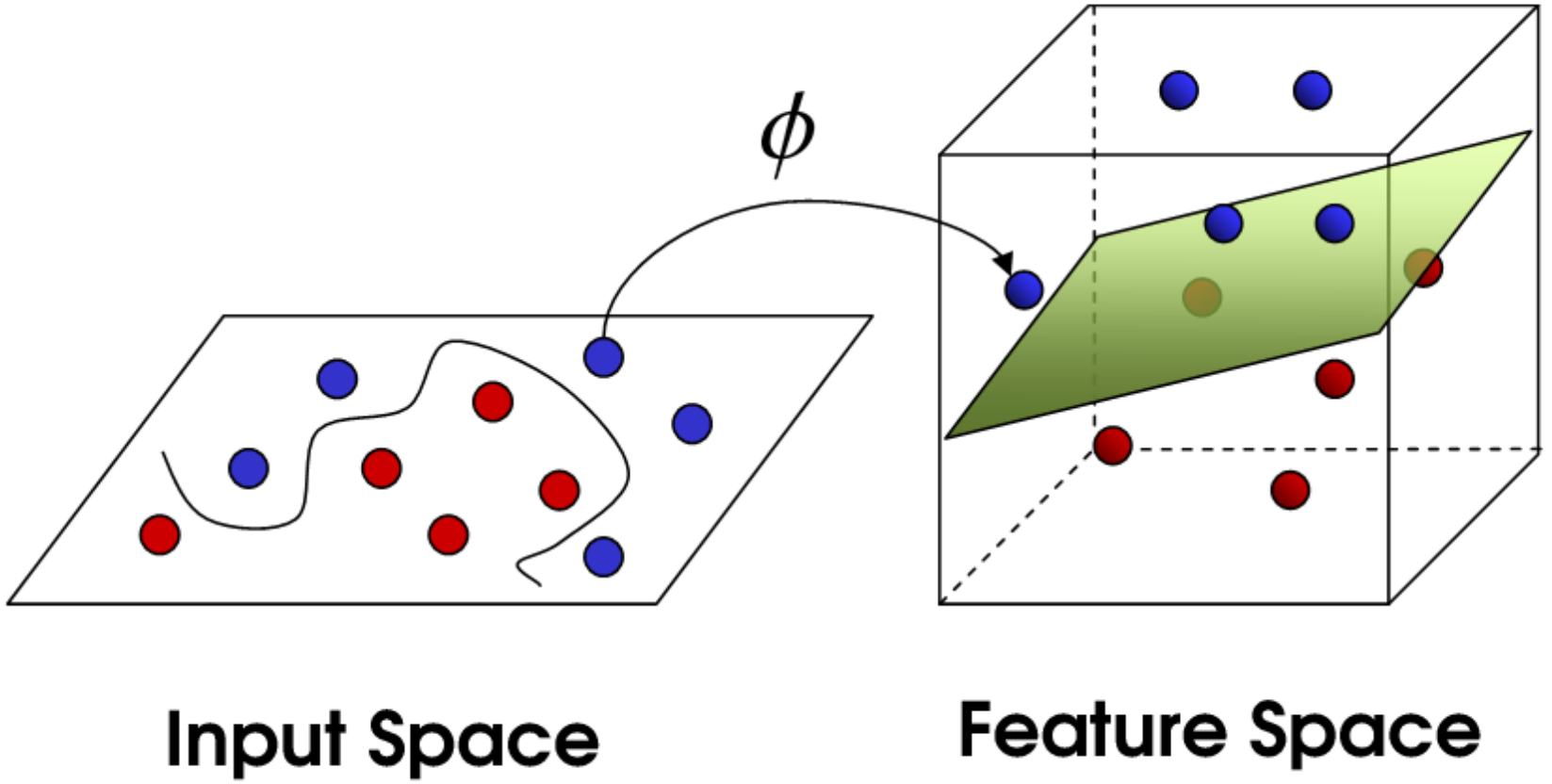
$$f_w(\phi(\boldsymbol{x}_i)) = \boldsymbol{w}^T \phi(\boldsymbol{x}_i) = \sum_j \alpha_j y_j K(x_j, x_i)$$

Learning Using Kernels

- Without a kernel function
  - Generate feature vectors $\phi(\boldsymbol{x})$ from samples $\boldsymbol{x}$.
  - The dimension of the feature vectors is usually **much higher** than the dimension of the original samples.
  - The computing load to find the solution of the optimization problem becomes higher.

- With a kernel function
  - If we can calculate the value of the kernel function, **we do not need to generate the feature vectors**.
  - It is much more efficient to find the solution of optimization problem.

Examples



Example

General inner product

$$\boldsymbol{x} = (x_1, x_2)$$

$$\phi(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)$$

$$K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}') = 1 + x_1 x'_1 + x_2 x'_2 + x_1^2 x_1'^2 + x_2^2 x_2'^2 + x_1 x'_1 x_2 x'_2$$

Kernel trick

$$K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x})^2 = (1 + x_1 x'_1 + x_2 x'_2)^2$$

$$= 1 + x_1 x'_1 + x_2 x'_2 + 2x_1^2 x_1'^2 + 2x_2^2 x_2'^2 + 2x_1 x'_1 x_2 x'_2$$

This is an inner product!

$$\phi(\boldsymbol{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2) \qquad \phi(\boldsymbol{x}') = (1, \sqrt{2}x'_1, \sqrt{2}x'_2, x'^2_1, x'^2_2, \sqrt{2}x'_1 x'_2)$$

Polynomial Kernel

- Polynomial Kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^k$$

Note that we did not generate feature vectors!

- Feature vector

$$K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^T \boldsymbol{x}')^k \quad \text{(assume } x_0 = 1\text{)}$$

$$= \left(\sum_{j=0}^{m} \boldsymbol{x}_j \boldsymbol{x}'_j\right)\left(\sum_{j=0}^{m} \boldsymbol{x}_j \boldsymbol{x}'_j\right) \cdots \left(\sum_{j=0}^{m} \boldsymbol{x}_j \boldsymbol{x}'_j\right)$$

$$= \sum_{J \in [0:m]^k} \prod_{i=1}^{k} \boldsymbol{x}_{J_i} \boldsymbol{x}'_{J_i} = \sum_{J \in [0:m]^k} \prod_{i=1}^{k} \boldsymbol{x}_{J_i} \cdot \prod_{i=1}^{k} \boldsymbol{x}'_{J_i}$$

Equivalent feature vector:

$$\phi(\boldsymbol{x}) = \left\{\prod_{i=1}^{k} \boldsymbol{x}_{J_i}\right\}_{J \in [0:m]^k}$$

Gaussian Kernel

- Gaussian Kernel (the original space is $\chi = \mathbb{R}$)

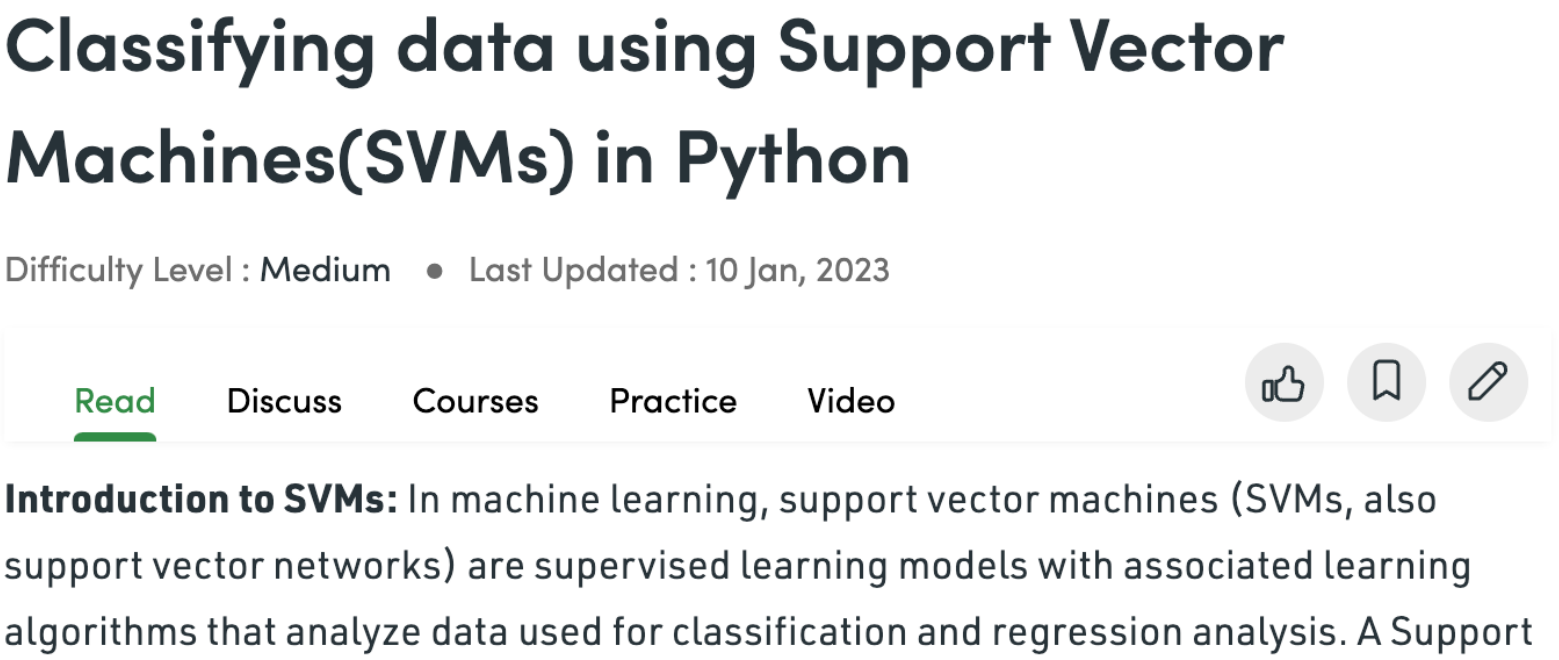
$$K(x, x') = \phi(x)^T \phi(x') = e^{-\frac{(x-x')^2}{2}}$$

- Feature vector

$$\phi(\boldsymbol{x}) = \left\{ \frac{1}{\sqrt{m!}} e^{-\frac{x^2}{2}} x^m \right\}_{m=0}^{\infty}$$

Programming Exercise

- GeeksForGeeks
  - https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/



Classifying data using Support Vector Machines(SVMs) in Python

Difficulty Level : Medium • Last Updated : 10 Jan, 2023

Read Discuss Courses Practice Video

Introduction to SVMs: In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. A Support

STEVENS
INSTITUTE OF TECHNOLOGY
1870

15 min. break

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Artificial Neural Network

Artificial Neural Network

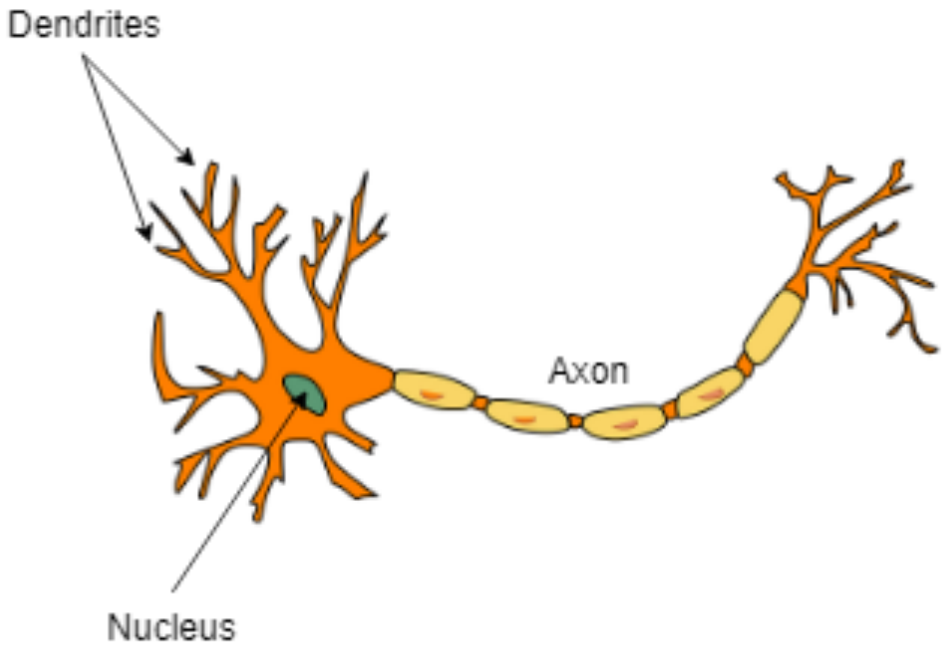
- Machine learning algorithms can be viewed as approximations of functions that describe the data.
- In practice, the relationships between input and output can be extremely complex.

Artificial Neural Network

- Idea
  - Humans can often learn complex relationships very well.
  - Probably we can simulate human learning.

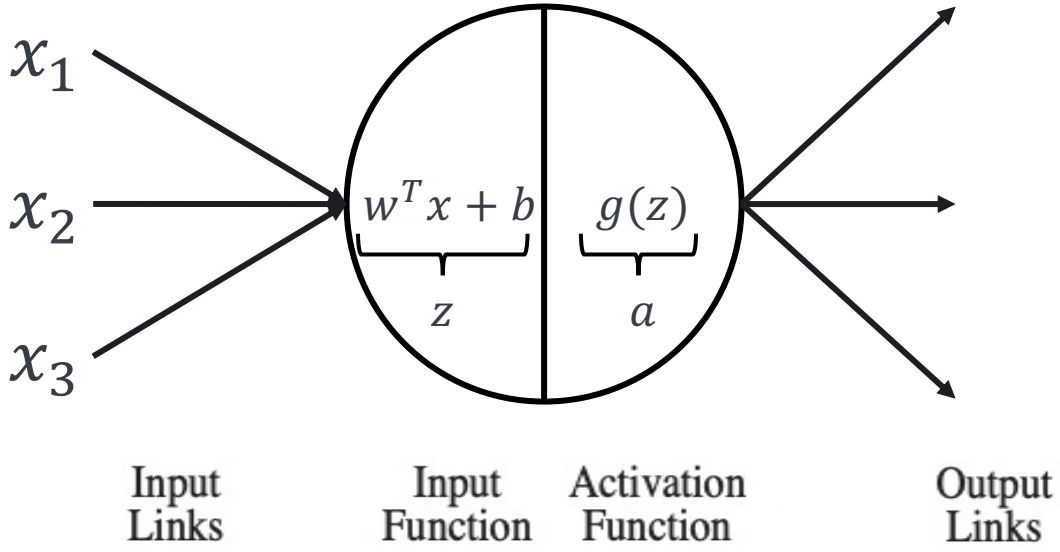
Human Brains

- A brain is a set of densely connected neurons.
- A neuron has several parts:
  - Dendrites: receiving inputs from other cells
  - Soma: controls activity of the neuron
  - Axon: sends output to other cells
  - Synapse: links between neurons



Artificial Brains?

- An artificial neuron (McCulloch and Pitts 1943)
  - Link~Synapse
  - Weight~Efficiency
  - Input Func.~Dendrite
  - Activation Fun.~Soma
  - Output=Fire or not

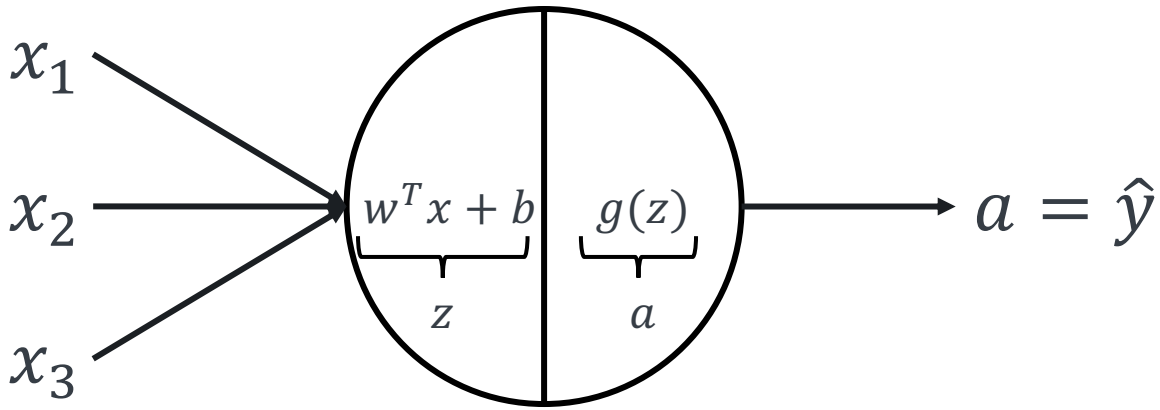


Neural Networks

Explained In A Minute

Artificial Neural Network

- Collection of simple artificial neurons
- Weights w denote strength of connection from a neuron i to j.
- Input function
  - $z = w^T x + b$
- Activation function
  - $a = g(z)$



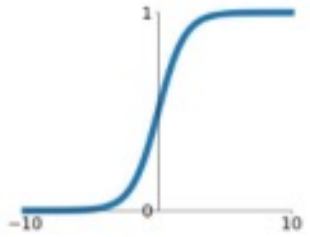
Activation Function

- Activation Function
  - $a = g(z)$
- Should be non-linear
  - Otherwise, we just have a linear equation.
- Should mimic firing in real neurons
  - Active ($a_i \sim 1$) when the “right” neighbors fire the right amounts
  - Inactive ($a_i \sim 0$) when fed “wrong” inputs

Common Activation Functions

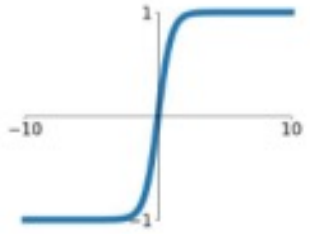
Sigmoid

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



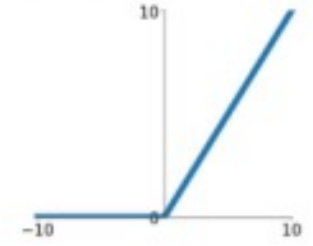
tanh

$$\tanh(x)$$



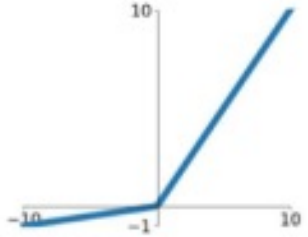
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

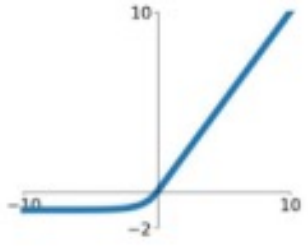


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



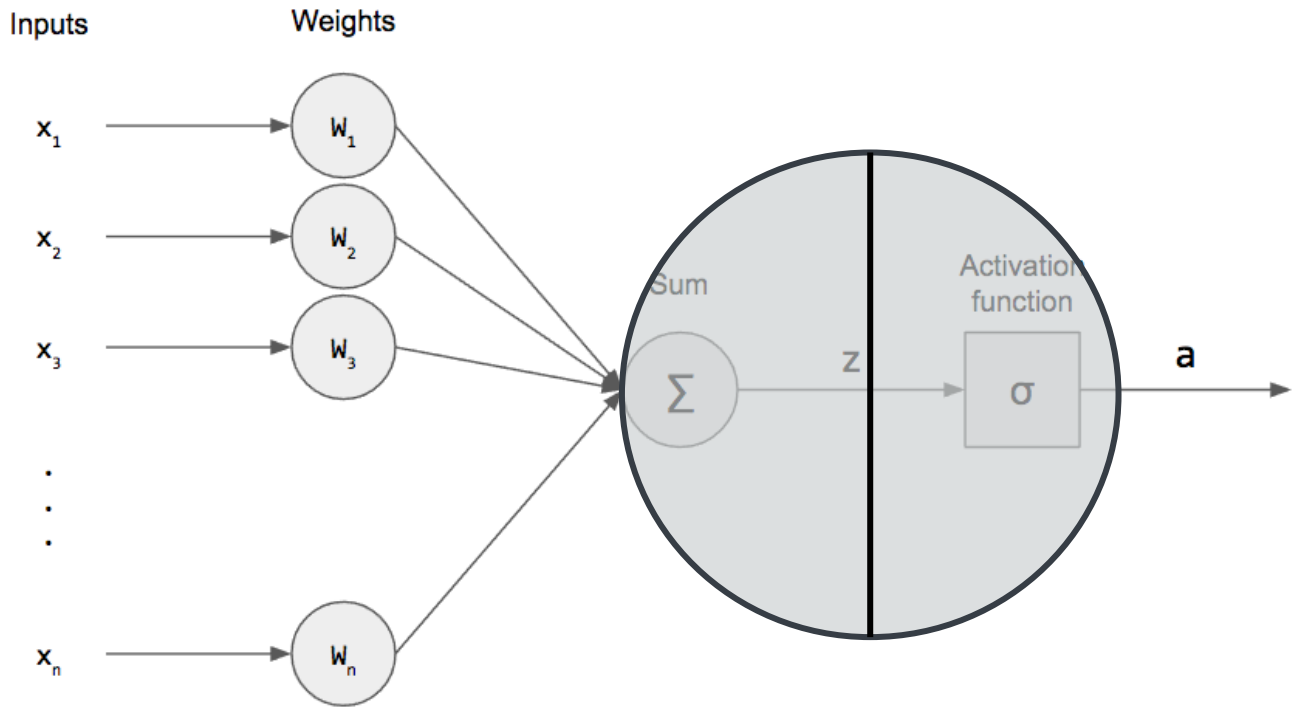
Network Structure

- Feed-forward ANN
  - Direct acyclic graph
  - **No internal state**: maps inputs to outputs
  - Commonly used when input and output are independent (*e.g.*, image recognition)

- Recurrent ANN
  - Directed cyclic graph
  - Dynamical system with an internal state
  - **Can remember information for future use**
  - Commonly used when input and output are dependent (*e.g.*, speech recognition)

Perceptron

- Single layer feed-forward network

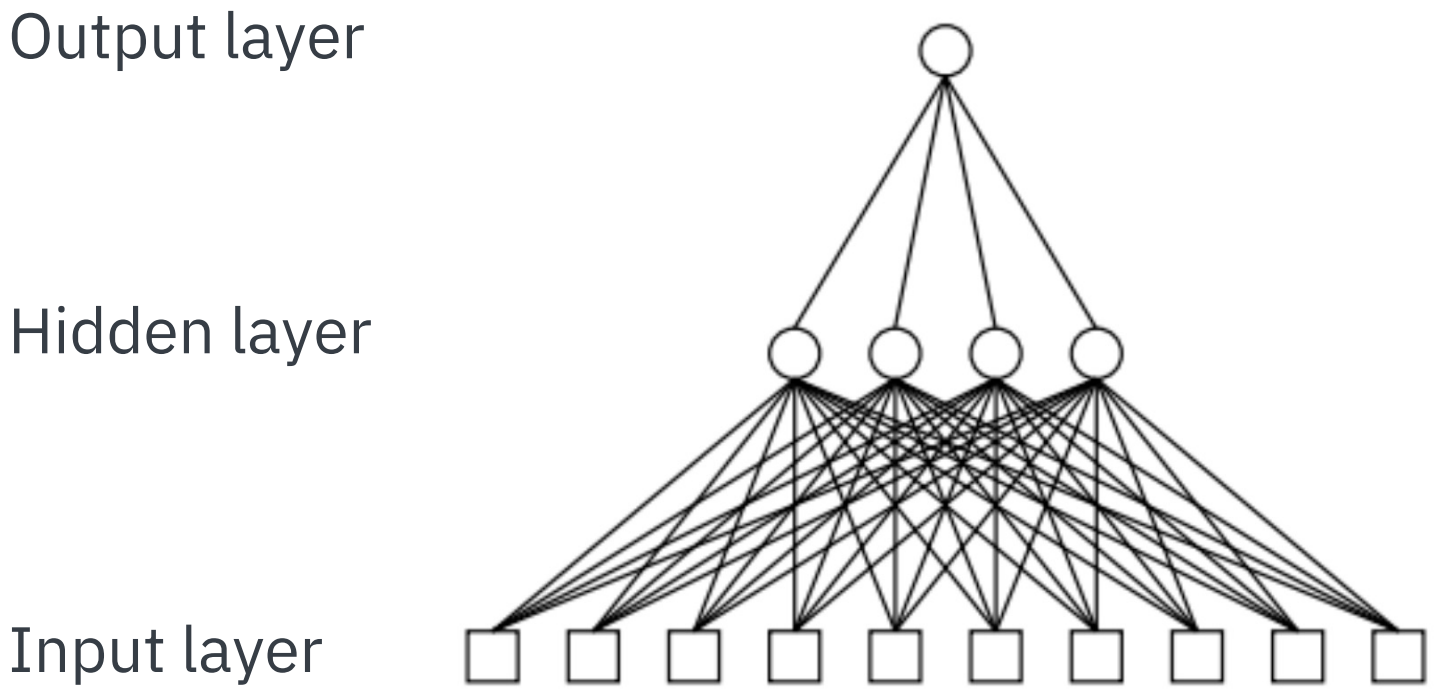


Multilayer Networks

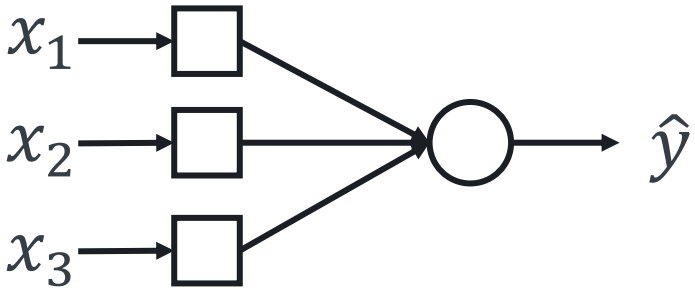
- Minsky's 1969 book *Perceptrons* showed perceptrons could not learn XOR.
- At the time, deeper networks were not usable practically.
- Most ANN research abandoned.

Multilayer Networks

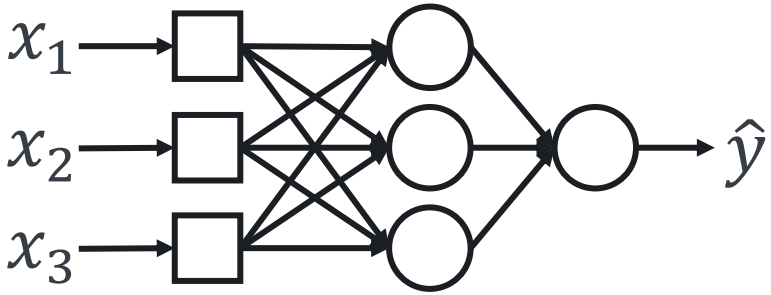
- Any continuous function can be learned by an ANN with just one hidden layer if the layer is large enough.



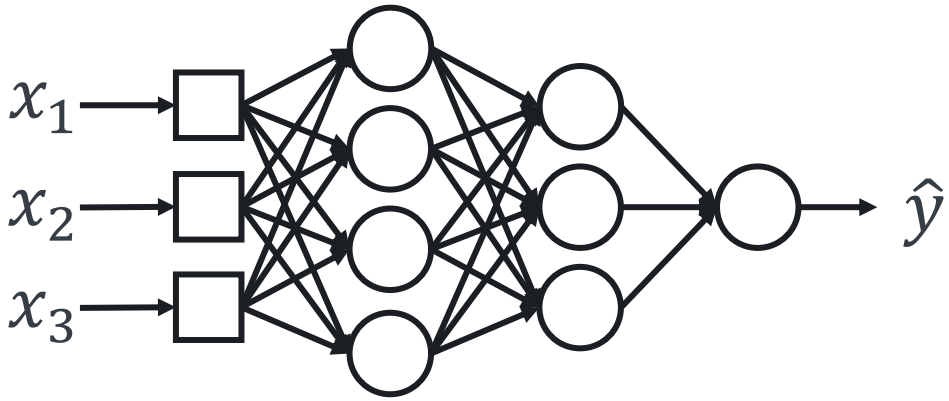
Multilayer Networks



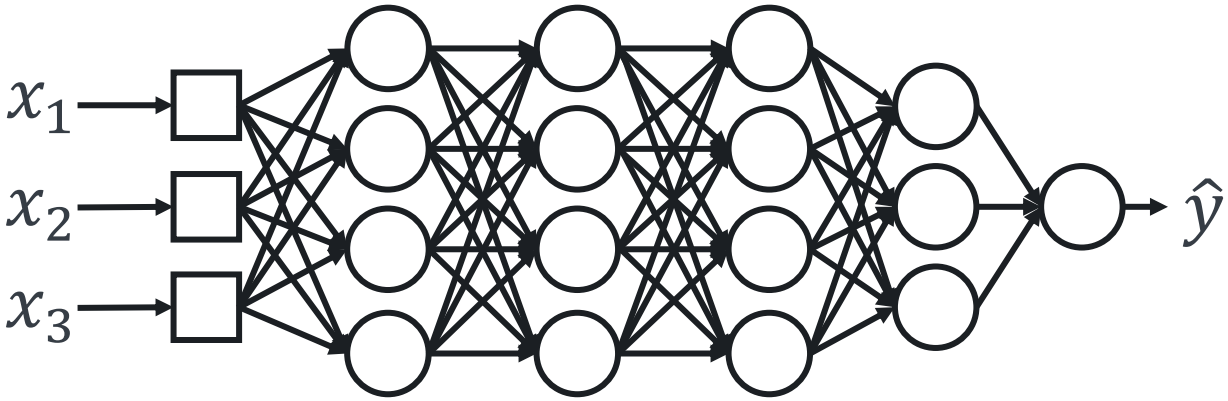
Perceptron



1 hidden layer

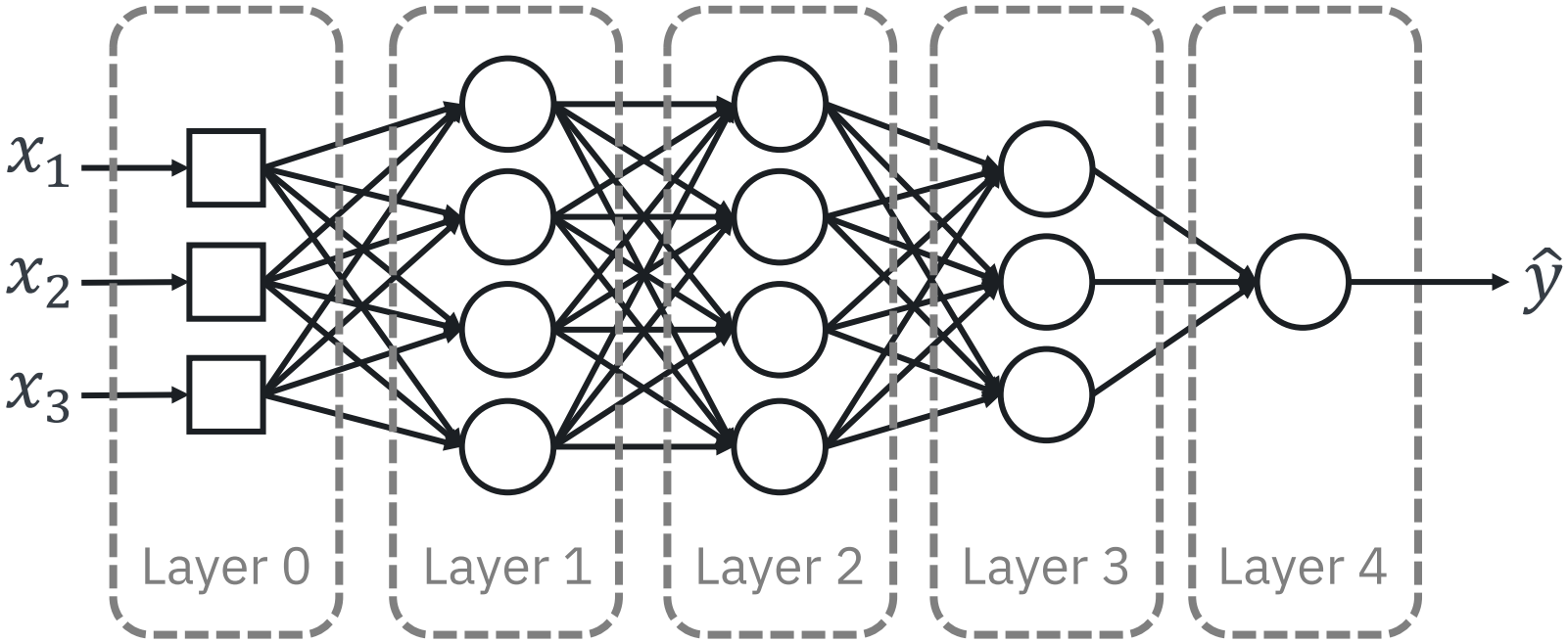


2 hidden layers



4 hidden layers

Deep Neural Network Notation



Number of layers

$L = 4$

Number of units at layer *l*

$n^{[l]}$

$n^{[0]} = 3$

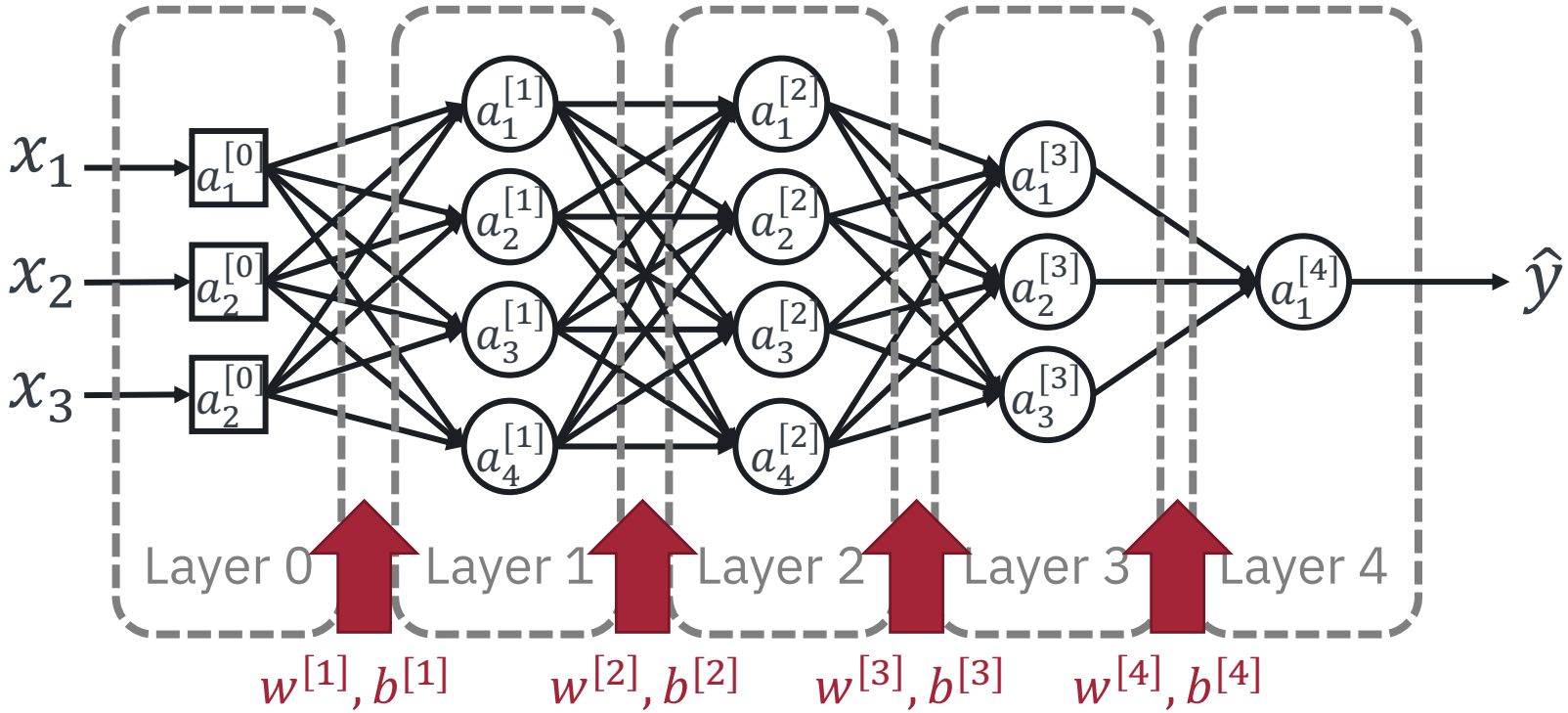
$n^{[1]} = 4$

$n^{[2]} = 4$

$n^{[3]} = 3$

$n^{[4]} = n^{[L]} = 1$

Deep Neural Network Notation



Activation in layer *l*

$a^{[l]} = g^{[l]}(z^{[l]})$

$z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$

$a^{[l]}$: $n^{[l]}$ x 1 matrix

$z^{[l]}$: $n^{[l]}$ x 1 matrix

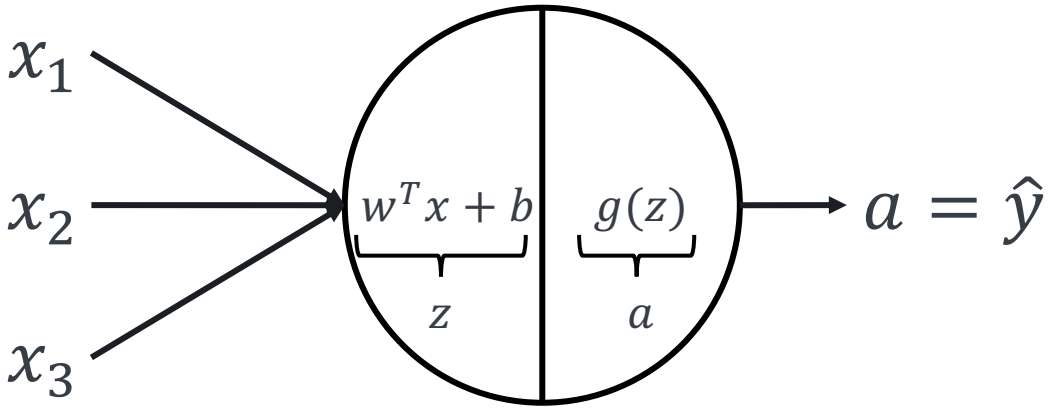
$w^{[l]}$: $n^{[l]}$ x $n^{[l-1]}$ matrix

$b^{[l]}$: $n^{[l]}$ x 1 matrix

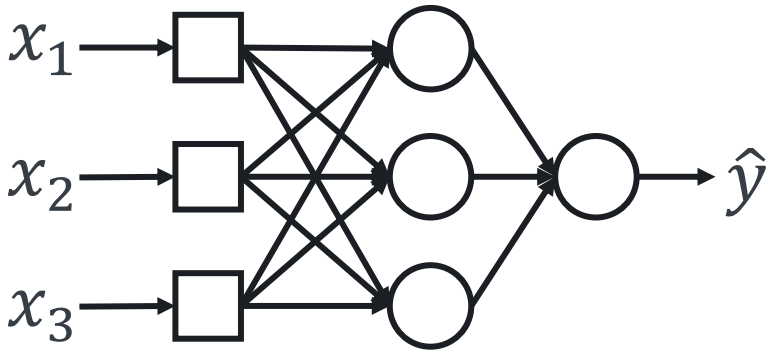
$x = a^{[0]}$

$\hat{y} = a^{[L]}$

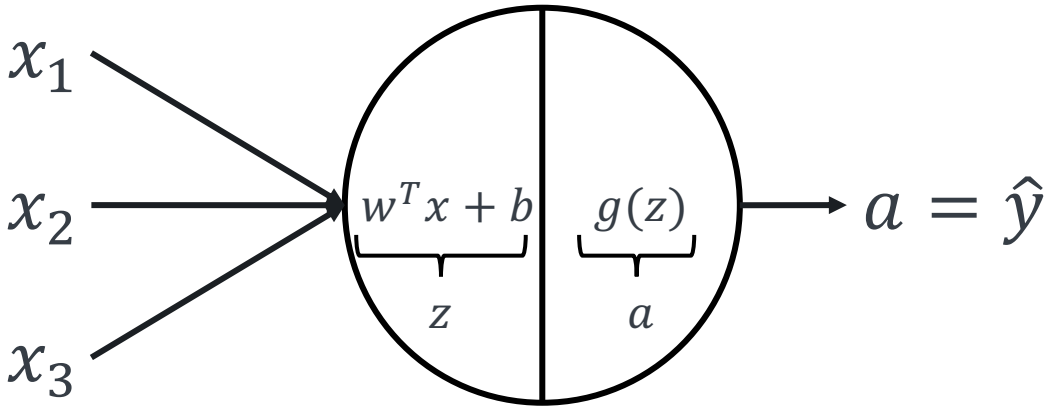
Forward Propagation



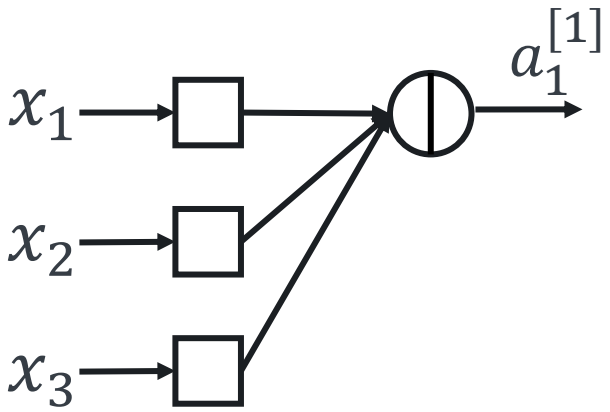
$$z = w^T x + b$$
$$a = g(z)$$



Forward Propagation

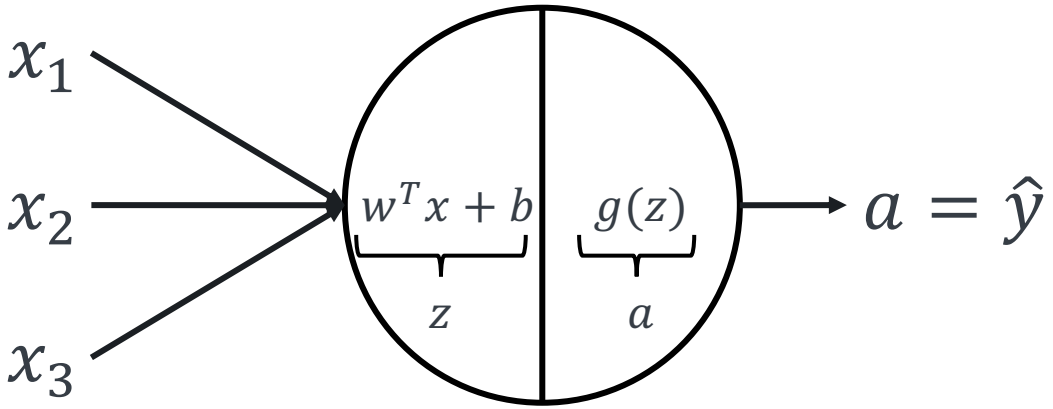


$$z = w^T x + b$$
$$a = g(z)$$

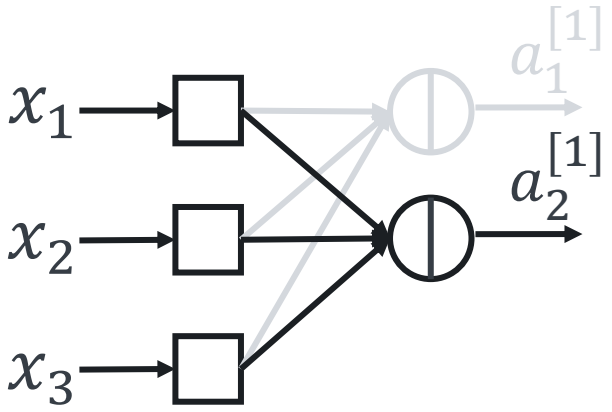


$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$
$$a_1^{[1]} = g(z_1^{[1]})$$

Forward Propagation

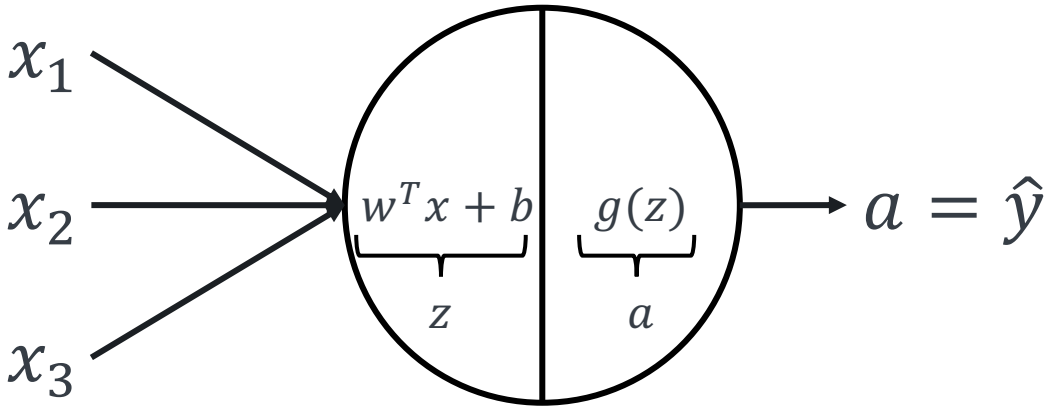


$$z = w^T x + b$$
$$a = g(z)$$

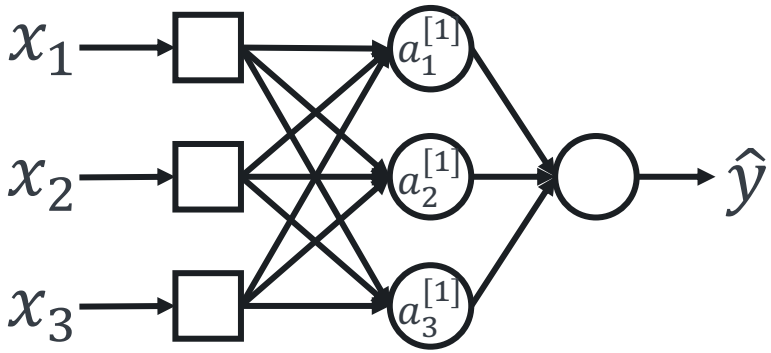


$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$
$$a_2^{[1]} = g(z_2^{[1]})$$

Forward Propagation



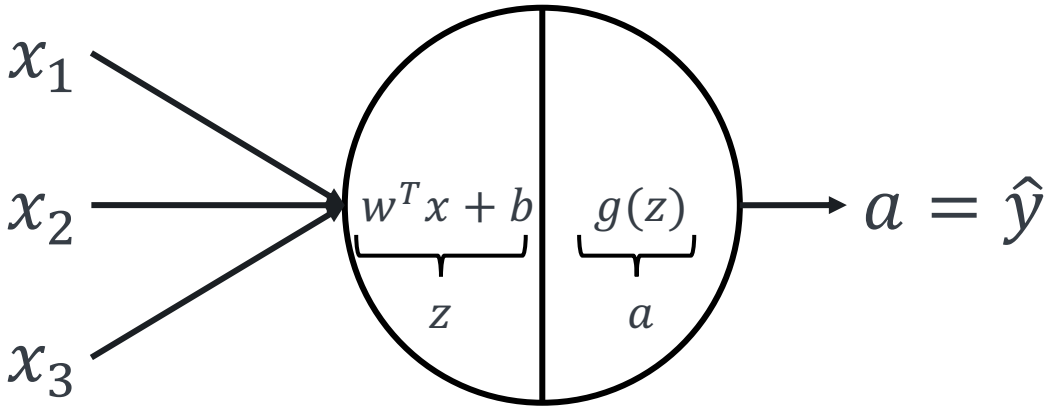
$$z = w^T x + b$$
$$a = g(z)$$



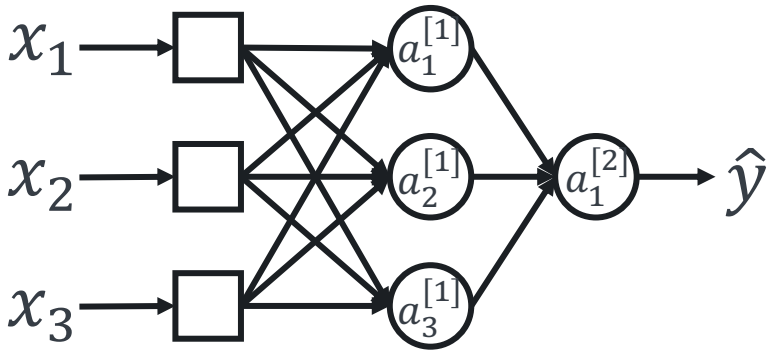
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]} \qquad a_1^{[1]} = g(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]} \qquad a_2^{[1]} = g(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]} \qquad a_3^{[1]} = g(z_3^{[1]})$$

$$z^{[1]} = w^{[1]T} x + b^{[1]} \qquad a^{[1]} = g(z^{[1]})$$

Forward Propagation



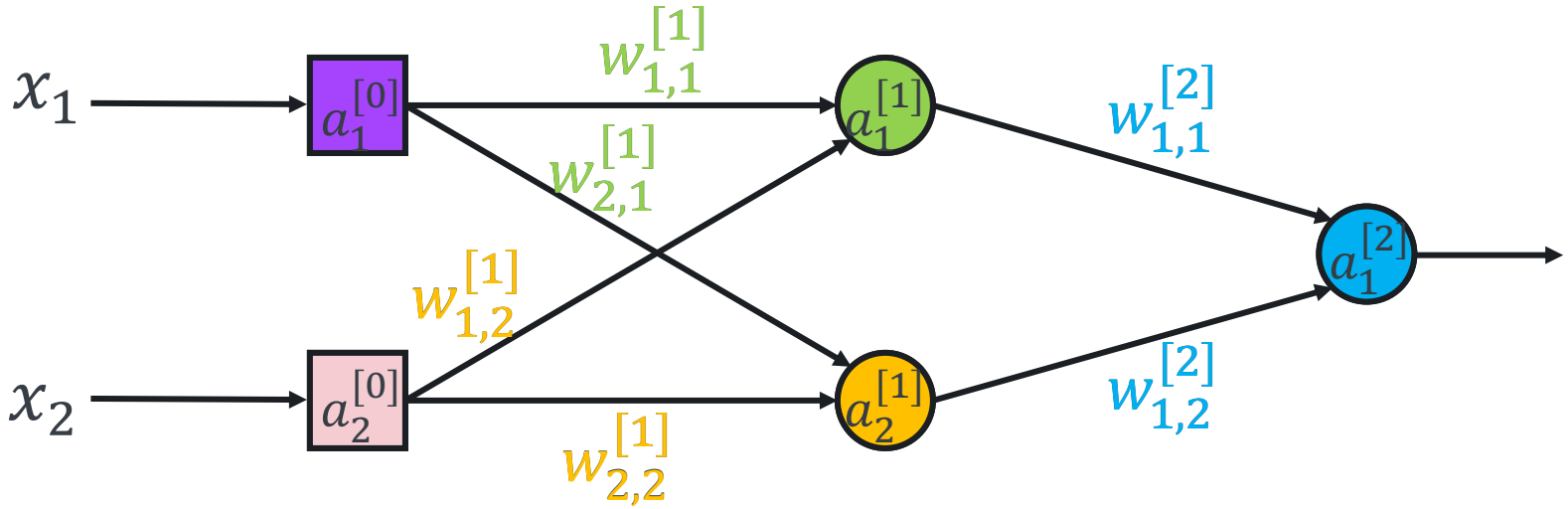
$$z = w^T x + b$$
$$a = g(z)$$



$$z^{[1]} = w^{[1]T} a^{[0]} + b^{[1]} \qquad a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = w^{[2]T} a^{[1]} + b^{[2]} \qquad a^{[2]} = g(z^{[2]})$$

$$z^{[l]} = w^{[l]T} a^{[l-1]} + b^{[l]} \qquad a^{[l]} = g(z^{[l]})$$

Forward Propagation: Example

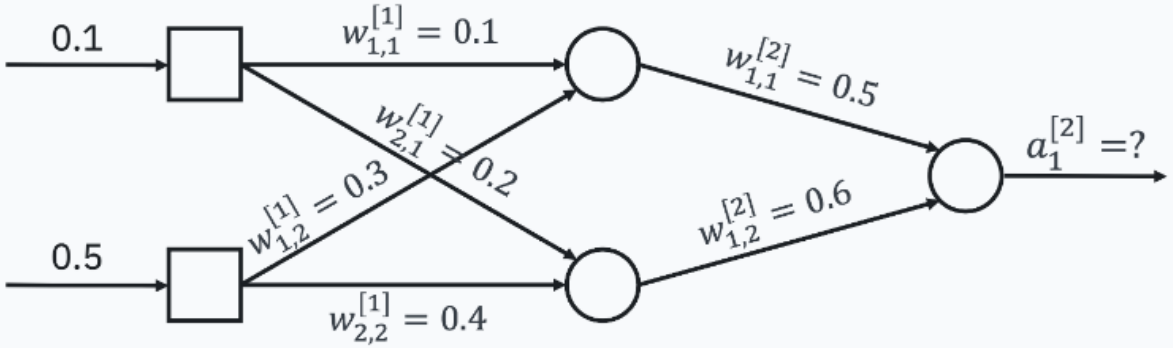


$$a_1^{[0]} = x_1 \qquad z_1^{[1]} = w_{1,1}^{[1]} a_1^{[0]} + w_{1,2}^{[1]} a_2^{[0]} + b_1^{[1]} \qquad a_1^{[1]} = g(z_1^{[1]})$$

$$a_2^{[0]} = x_2 \qquad z_2^{[1]} = w_{2,1}^{[1]} a_1^{[0]} + w_{2,2}^{[1]} a_2^{[0]} + b_1^{[1]} \qquad a_2^{[1]} = g(z_2^{[1]})$$

$$z_1^{[2]} = w_{1,1}^{[2]} a_1^{[1]} + w_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]} \qquad a_1^{[2]} = g(z_1^{[2]})$$

What is a[2]_1?



$b_1^{[1]} = 0.25 \quad b_2^{[1]} = 0.25 \quad b_1^{[2]} = 0.2$

g(0.41)=0.6 g(0.47)=0.62 g(0.872)=0.71 g(1.26)=0.78

0.6

0.62

0.71

0.78

Training Multilayer Nets

- Back propagation
- Gradient descent algorithm

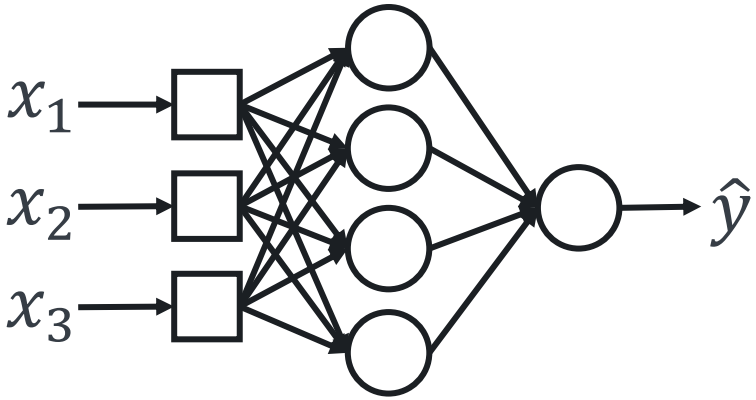
Network predicts again

Gradient Descent for Neural Networks

Parameters: $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}$

Cost function:

$$J\left(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}\right) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$$



Gradient descent

Repeat:

Compute prediction $\hat{y}_i \;\; (i = 1 \ldots m)$

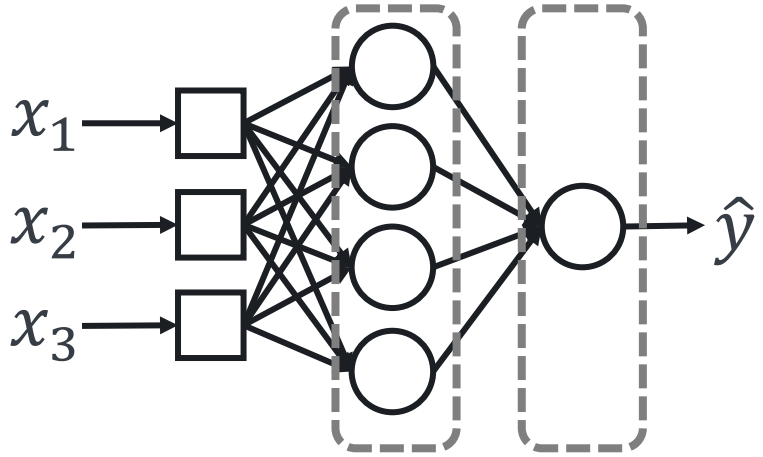
$$dw^{[l]} = \frac{dJ}{dw^{[l]}} \quad , \quad db^{[l]} = \frac{dJ}{db^{[l]}}$$

How do we get the derivatives?

$$w^{[2]} \leftarrow w^{[2]} - \alpha \cdot dw^{[2]} \qquad b^{[2]} \leftarrow b^{[2]} - \alpha \cdot db^{[2]}$$

$$w^{[1]} \leftarrow w^{[1]} - \alpha \cdot dw^{[1]} \qquad b^{[1]} \leftarrow b^{[1]} - \alpha \cdot db^{[1]}$$

Formulas for Computing Derivatives



Forward Propagation

$$z^{[1]} = w^{[1]T}x + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = w^{[2]T}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$

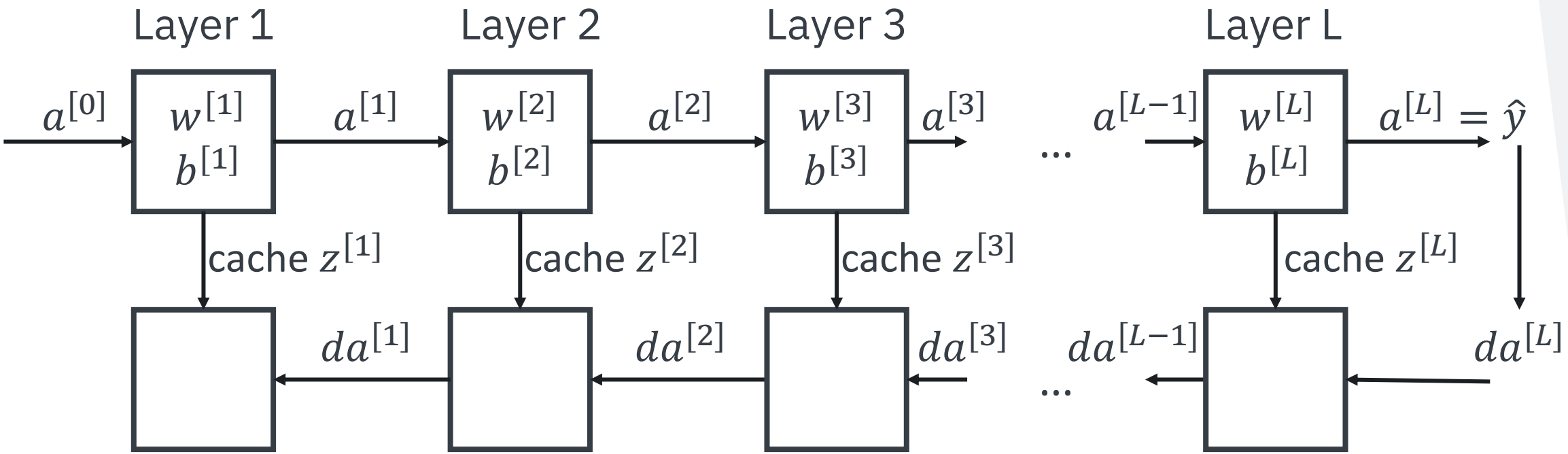
Backward Propagation

$$dz^{[2]} = da^{[2]} \cdot g'(z^{[2]})$$
$$dw^{[2]} = dz^{[2]} \cdot a^{[1]}$$
$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = da^{[1]} \cdot g'(z^{[1]})$$
$$dw^{[1]} = dz^{[1]} \cdot x$$
$$db^{[1]} = dz^{[1]}$$

Forward and Backward Propagation

$$z^{[l]} = w^{[l]T} a^{[l-1]} + b^{[l]} \qquad a^{[l]} = g(z^{[l]})$$



$$w^{[l]} \leftarrow w^{[l]} - \alpha \cdot dw^{[l]} \qquad b^{[l]} \leftarrow b^{[l]} - \alpha \cdot db^{[l]}$$

Forward and Backward Propagation

Forward Propagation

Input: $a^{[l-1]}$

Output: $a^{[l]}$, cache $(z^{[1]})$

$$z^{[l]} = w^{[l]T} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g(z^{[l]})$$

Backward Propagation

Input: $da^{[l]}$

Output: $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$

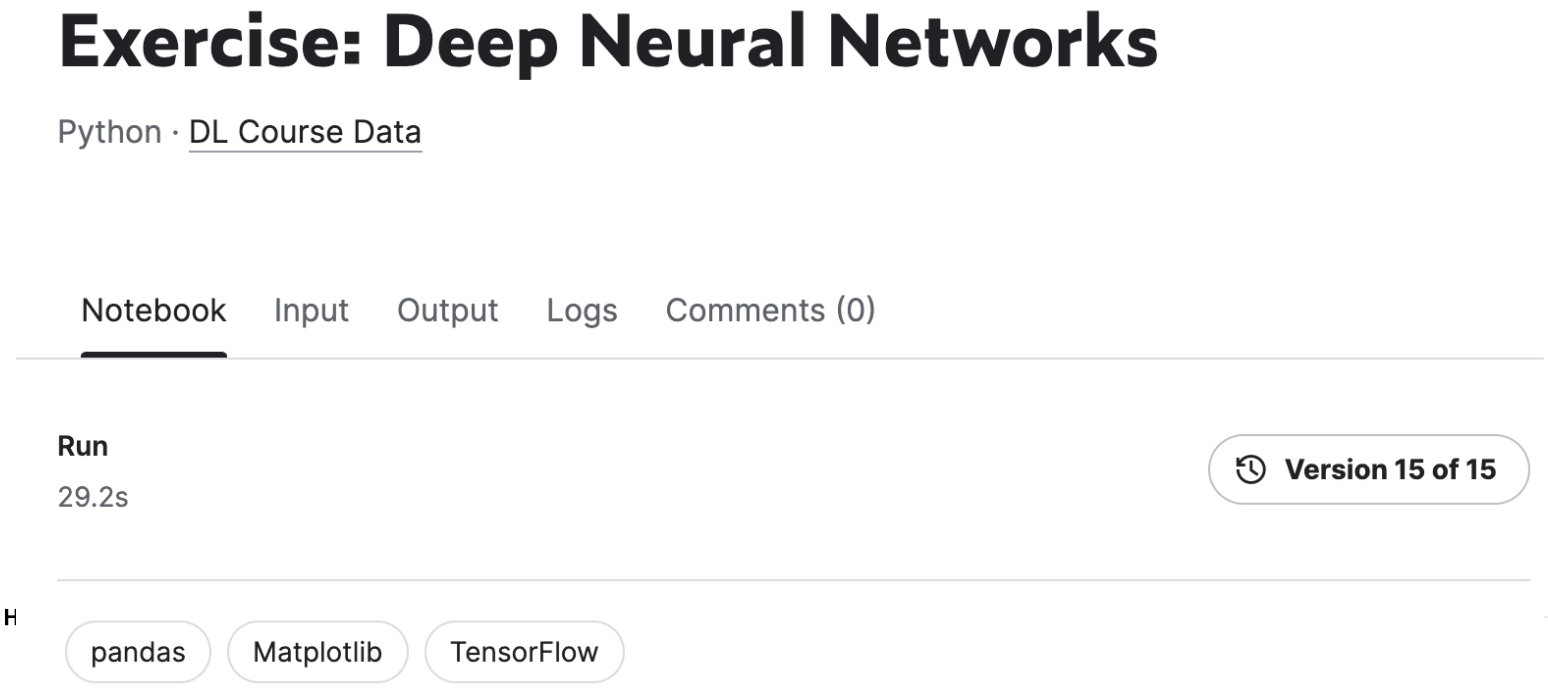
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$

$$db^{[l]} = dz^{[l]}$$

$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

Programming Exercise

- Kaggle
  - https://www.kaggle.com/code/ryanholbrook/exercise-deep-neural-networks



STEVENS
INSTITUTE OF TECHNOLOGY
1870

Questions?

Acknowledgement

Fahiem Bacchus, University of Toronto
Dan Klein, UC Berkeley
Kate Larson, University of Waterloo