



CMPT 413/713: Natural Language Processing

Classification - Logistic Regression

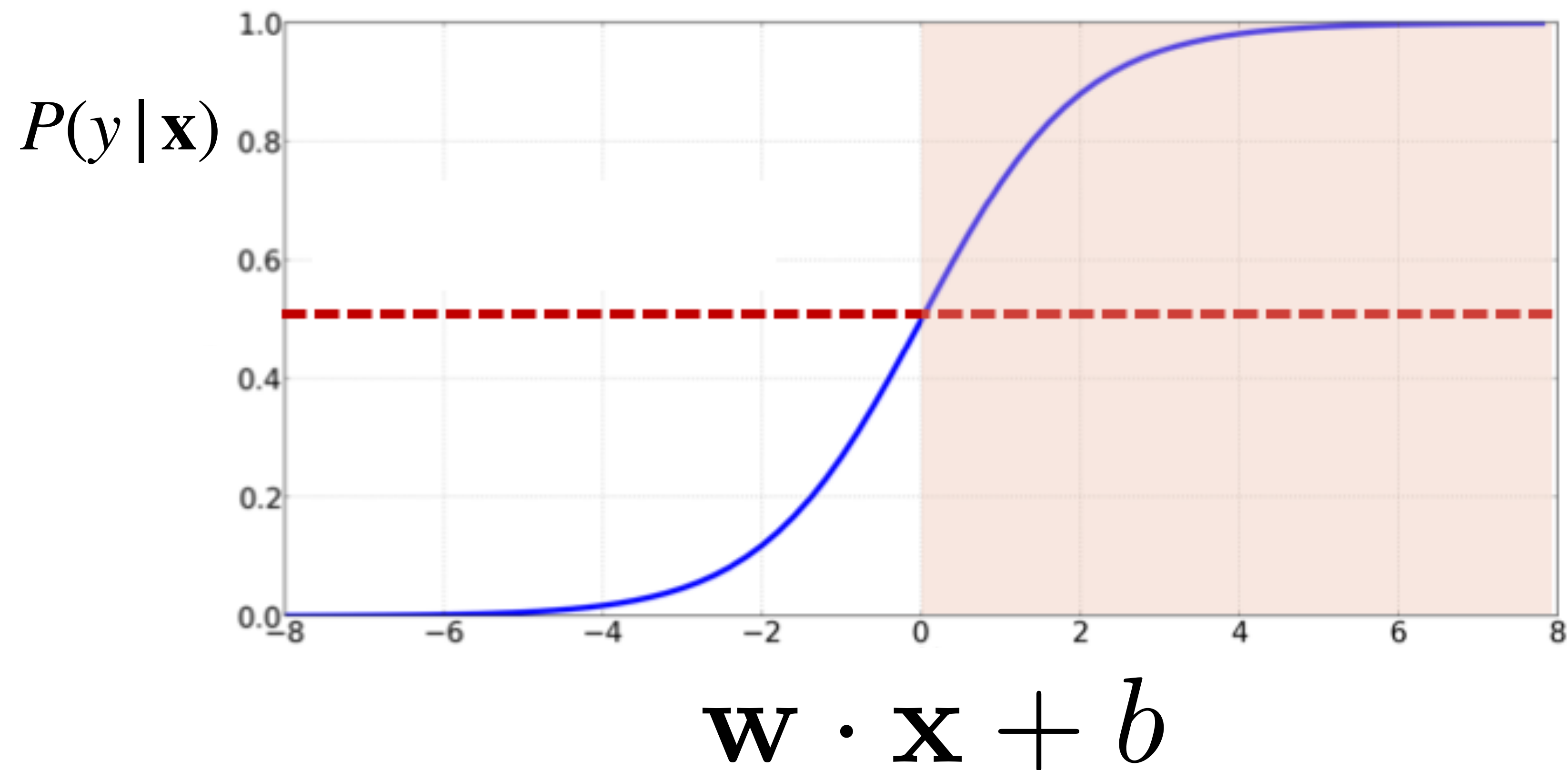
Spring 2026
2026-01-14

Adapted from slides from Danqi Chen, Karthik Narasimhan, and Anoop Sarkar

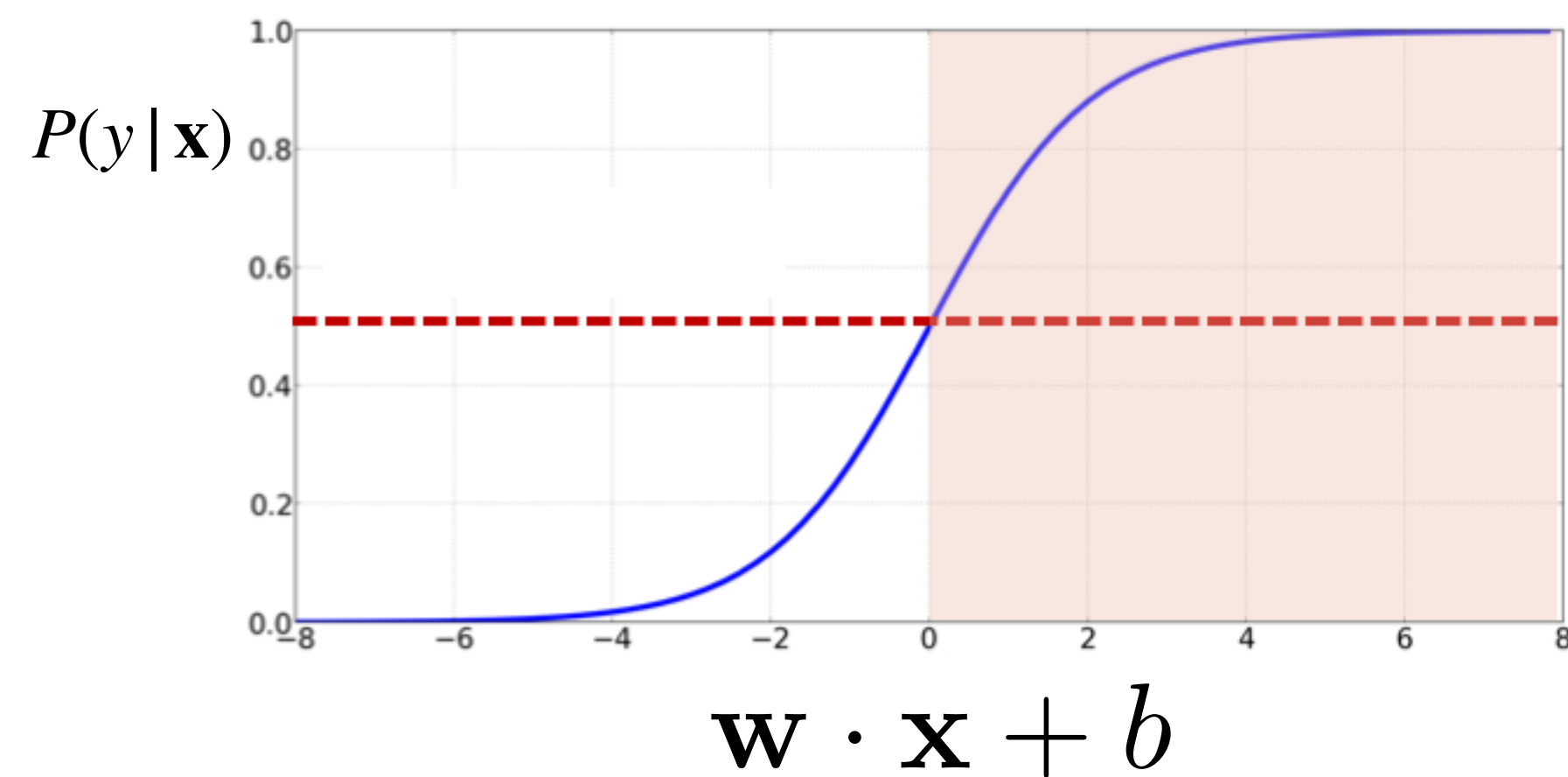
Logistic regression

Discriminative Model

- Logistic Regression: model $p(y | x)$ directly



Logistic Regression

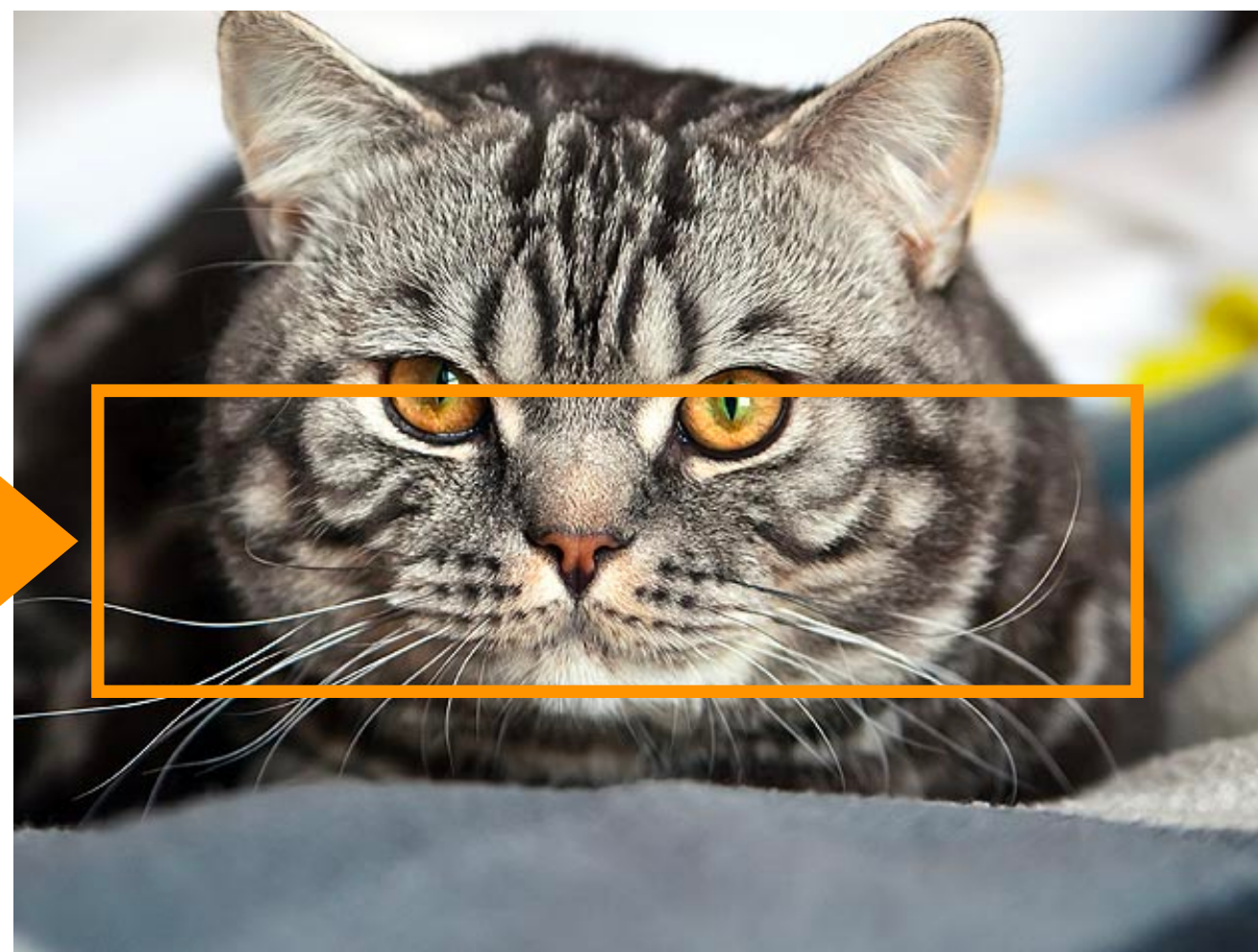


- Powerful supervised model
- Baseline approach to most NLP tasks
- Connections with neural networks
- Binary (two classes) or multinomial (>2 classes)

Discriminative Model

- *discriminative* model
Logistic Regression: $\hat{c} = \arg \max_c P(c | d)$ focus on discriminating features
- Naive Bayes: $\hat{c} = \arg \max_c P(c)P(d | c)$ can be used to generate the cat and the dog
generative model

cats have
whiskers



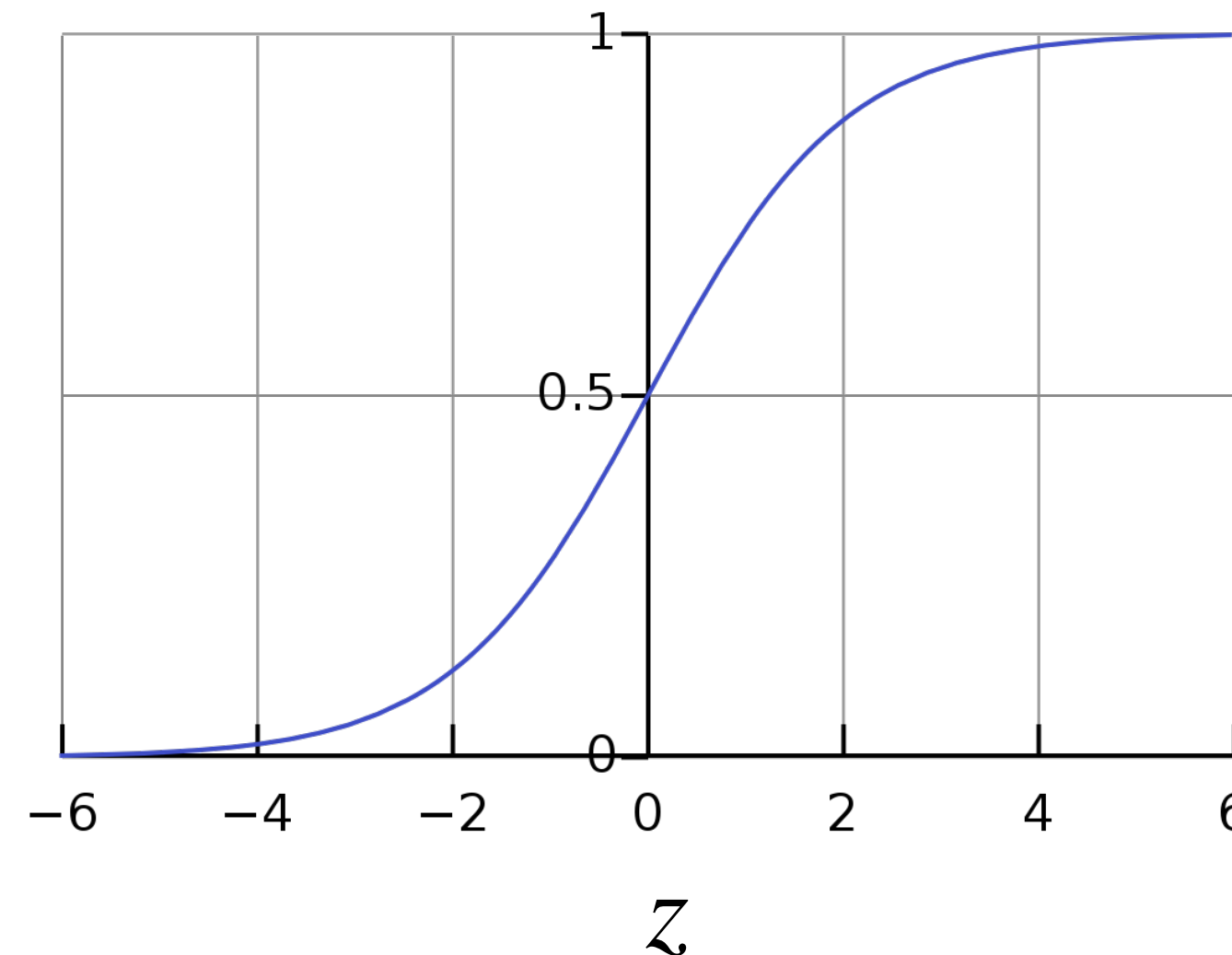
Classification function

- *Given:* Input feature vector $[x_1, x_2, \dots, x_d]$
- *Output:* $P(y = 1 | x)$ and $P(y = 0 | x)$ *(binary classification)*
- Use a *function*, $F : \mathbb{R}^d \rightarrow [0,1]$ to model the probability $P(y | x)$

- Sigmoid σ :

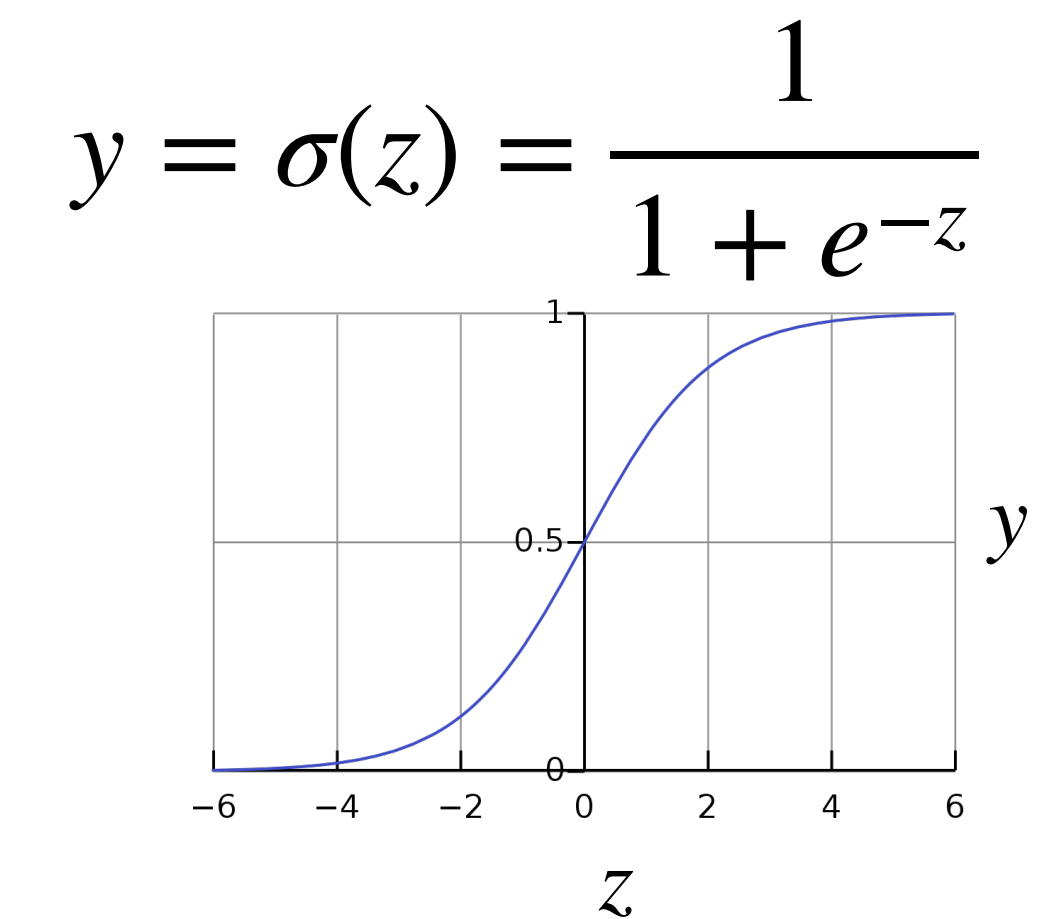
$$P(y|x) = \frac{1}{1 + e^{-z}}$$

$$z = \sum_{i=1}^d w_i x_i + b$$



$P(y|x)$

Why the sigmoid?



- Binary logistic regression uses the **log-odds** to determine the probability.

What is the probability p as a function of the log-odds?

- Log-odds:

$$\ell = \log \frac{p}{1 - p}$$

Value is also known
as **logit (logistic unit)**

$$\frac{p}{1 - p} = \exp(\ell)$$

$$p = \exp(\ell)(1 - p)$$

$$p(1 + \exp(\ell)) = \exp(\ell)$$

$$p = \frac{\exp(\ell)}{(1 + \exp(\ell))}$$

$$= \frac{1}{(1 + \exp(-\ell))} = \sigma(\ell)$$

- z is a linear estimate of the log-odds ℓ : $z = \sum_{i=1}^d w_i x_i + b$

Weights and bias

- *Which features are important* and *how much*?
- Learn a vector of **weights** and a **bias**
- Weights: Vector of real numbers, $\mathbf{w} = [w_1, w_2, \dots, w_d]$
- Bias: Scalar intercept, b

$$z = \sum_{i=1}^d w_i x_i + b$$

Vector form

$$z = \mathbf{w} \cdot \mathbf{x} + b = \mathbf{w}^T \mathbf{x} + b$$

Putting it together

Given $\mathbf{x}$, compute $z = \mathbf{w} \cdot \mathbf{x} + b$

Compute probabilities: $P(y = 1 | x) = \sigma(z) = \frac{1}{1 + e^{-z}}$

$$P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\begin{aligned} P(y = 0) &= 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) \\ &= 1 - \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \\ &= \frac{e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \end{aligned}$$

Decision boundary:

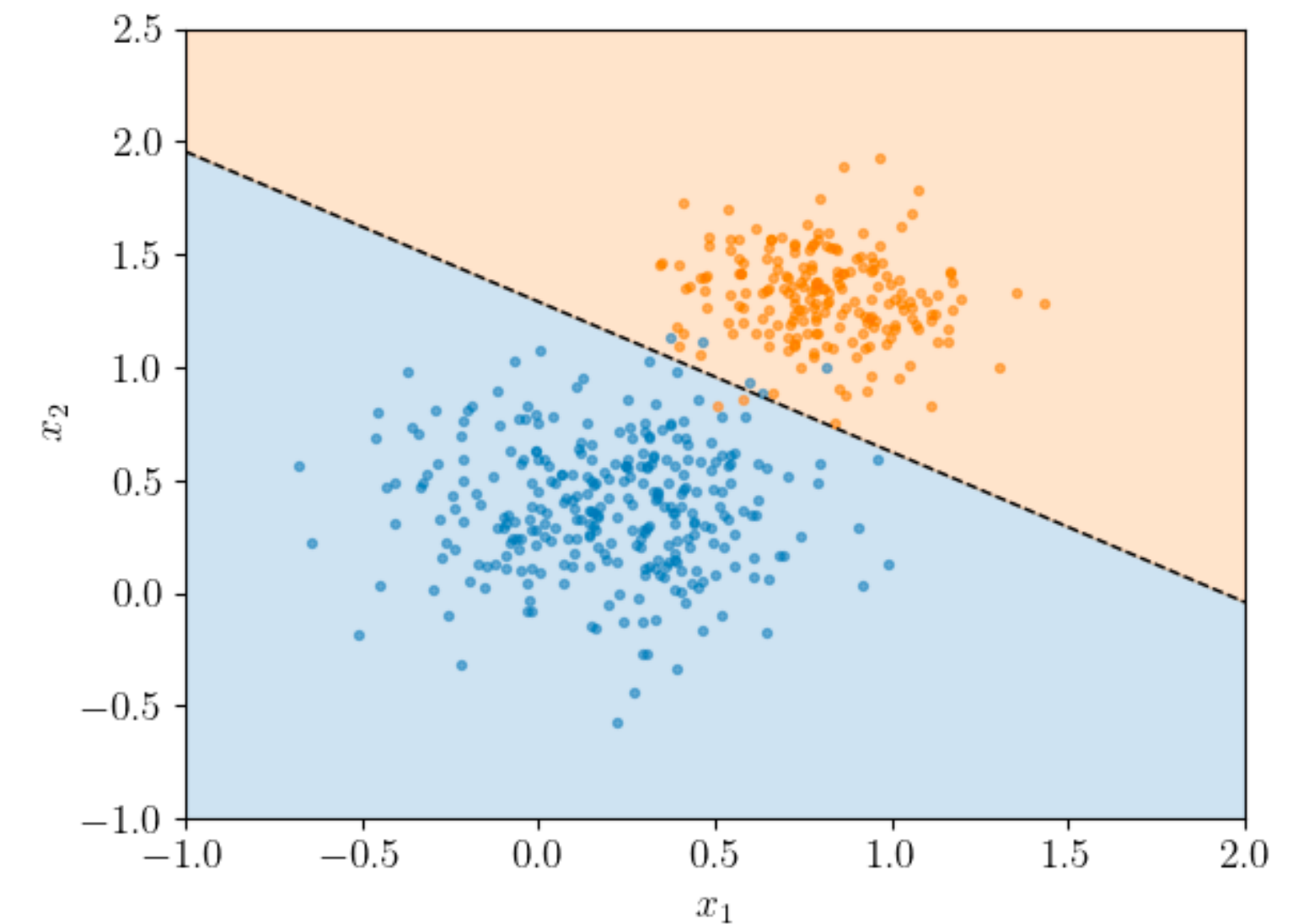
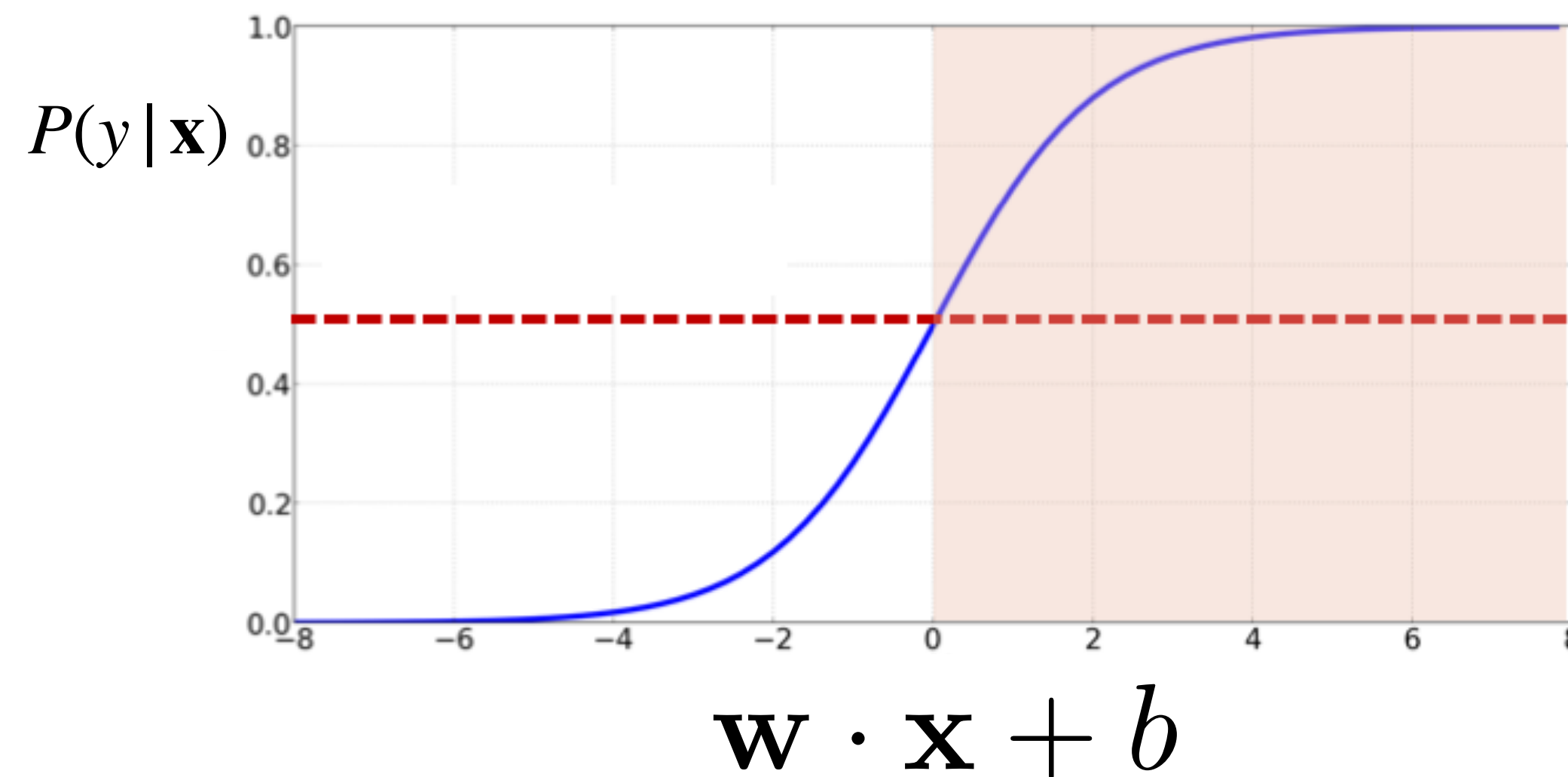
$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 | x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Decision boundary

Threshold can be set as
hyperparameter that is tuned

Decision boundary:

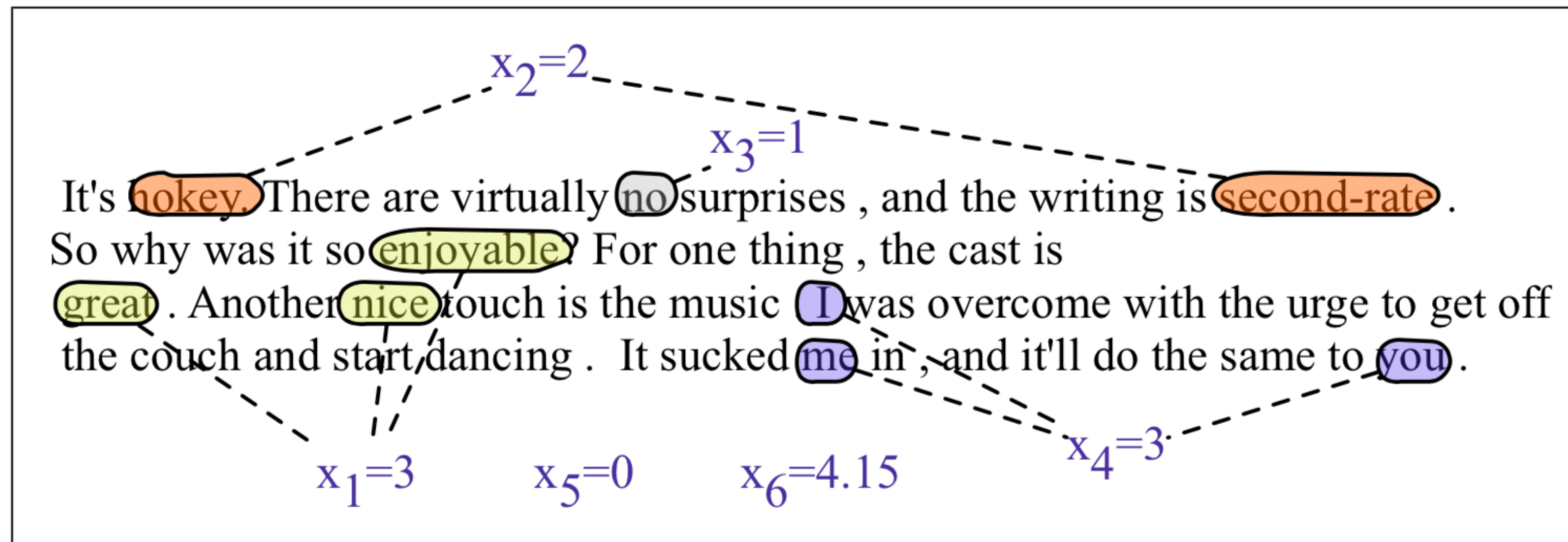
$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 | x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$
$$\begin{aligned} \mathbf{w} \cdot \mathbf{x} + b &> 0 \\ \mathbf{w} \cdot \mathbf{x} + b &\leq 0 \end{aligned}$$



Decision boundary is linear
function of features

What do we use as features?

Example: Sentiment classification



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no" } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!" } \in \text{ doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Example: Sentiment classification

Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$

$$\begin{aligned} p(+|x) = P(Y = 1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\ &= \sigma(.805) \\ &= 0.69 \end{aligned}$$

$$\begin{aligned} p(-|x) = P(Y = 0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.31 \end{aligned}$$

Feature design

- Most important rule: Data is *key*!

- Linguistic intuition (e.g. part of speech tags, parse trees)

- Complex combinations

$$x_1 = \begin{cases} 1 & \text{if “Case}(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if “}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if “}w_i = \text{St. \& Case}(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

- Feature templates

- Sparse representations, hash only seen features into index

- Ex. Trigram(*logistic regression classifier*)
= Feature #78

- Advanced: Representation learning (we will see this with neural networks!)

Multinomial Logistic Regression

Multinomial Logistic Regression

- What if we have more than 2 classes? (e.g. Part of speech tagging, named entity recognition)
- Need to model $P(y = c | x) \forall c \in C$
- Generalize **sigmoid** function to **softmax**

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}} \quad 1 \leq i \leq k$$

← Normalization

Softmax

- Similar to sigmoid, softmax squashes values towards 0 or 1
- If $z = [0,1,2,3,4]$, then
 - $\text{softmax}(z) = ([0.0117, 0.0317, 0.0861, 0.2341, 0.6364])$
- For multinomial LR,

$$P(y = c | x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^k e^{w_j \cdot x + b_j}}$$

Features in multinomial LR

- Features need to include both **input (x)** and **class (c)**
- Implicit in binary case

Var	Definition	Wt
$f_1(0, x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	-4.5
$f_1(+, x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	2.6
$f_1(-, x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1.3

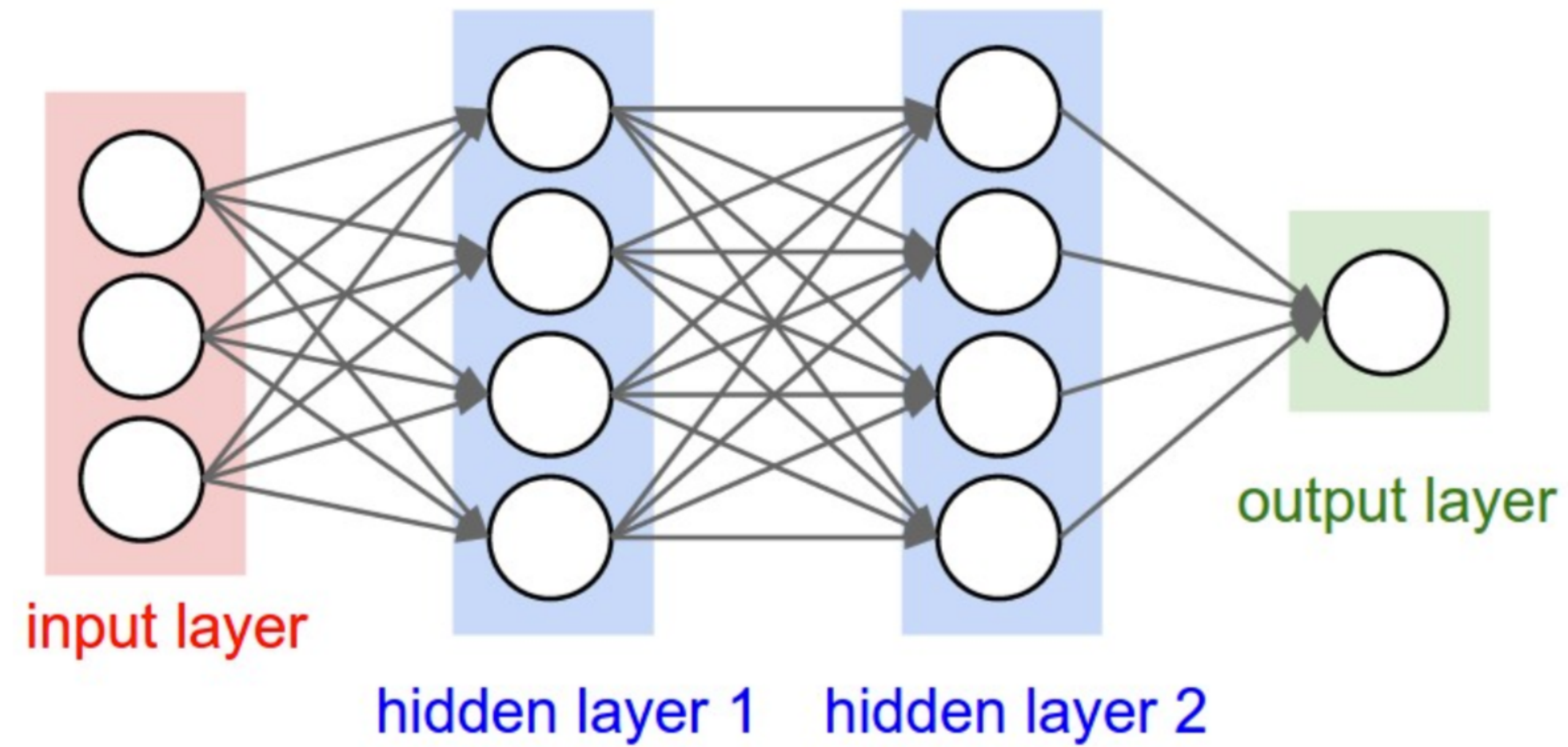
Logistic Regression: what's good and what's not

- More freedom in designing features
- No strong independence assumptions like Naive Bayes
- More robust to correlated features (“San Francisco” vs “Boston”) —LR is likely to work better than NB
- Can even have the same feature twice! (*why?*)
- **However:** not as good on small datasets (compared to Naive Bayes)
- Interpreting learned weights can be challenging

Relationship to neural networks

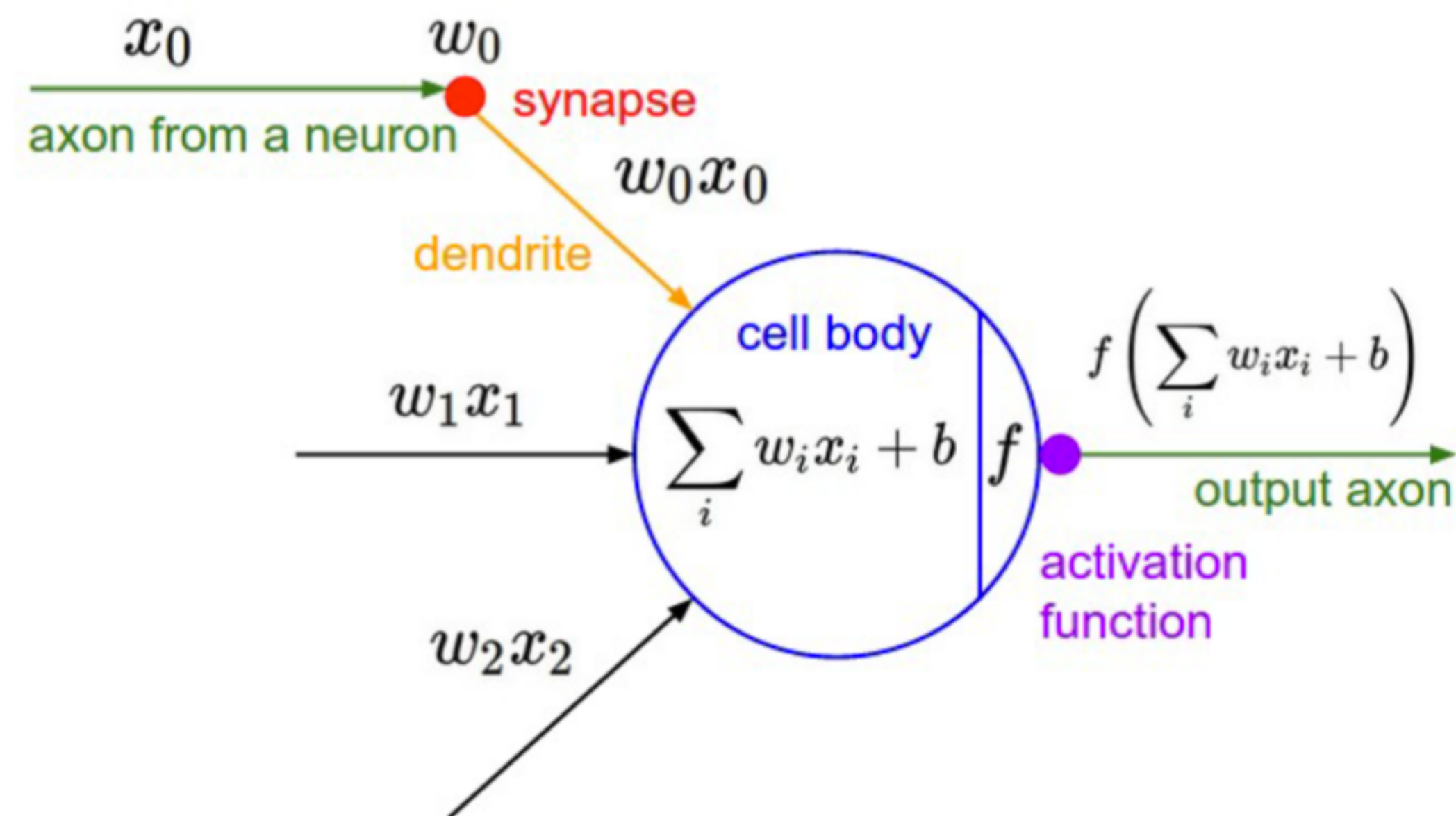
Feed-forward NNs

- Input: $x_1, \dots, x_d$
- Output: $y \in \{0,1\}$



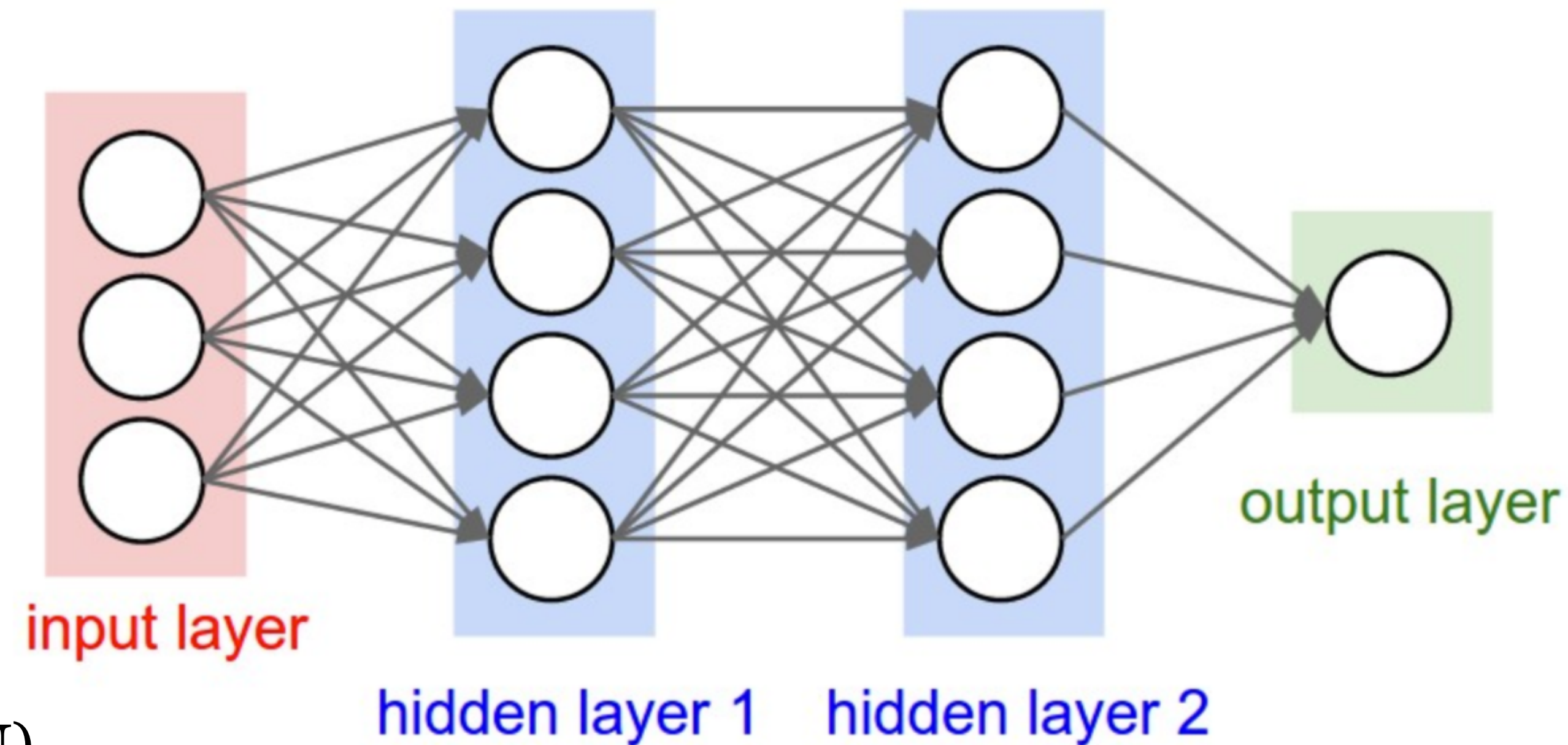
An artificial neuron

- A neuron is a computational unit that has scalar inputs and an output
- Each input has an associated weight.
- The neuron multiplies each input by its weight, sums them, applied a **nonlinear function** to the result, and passes it to its output.



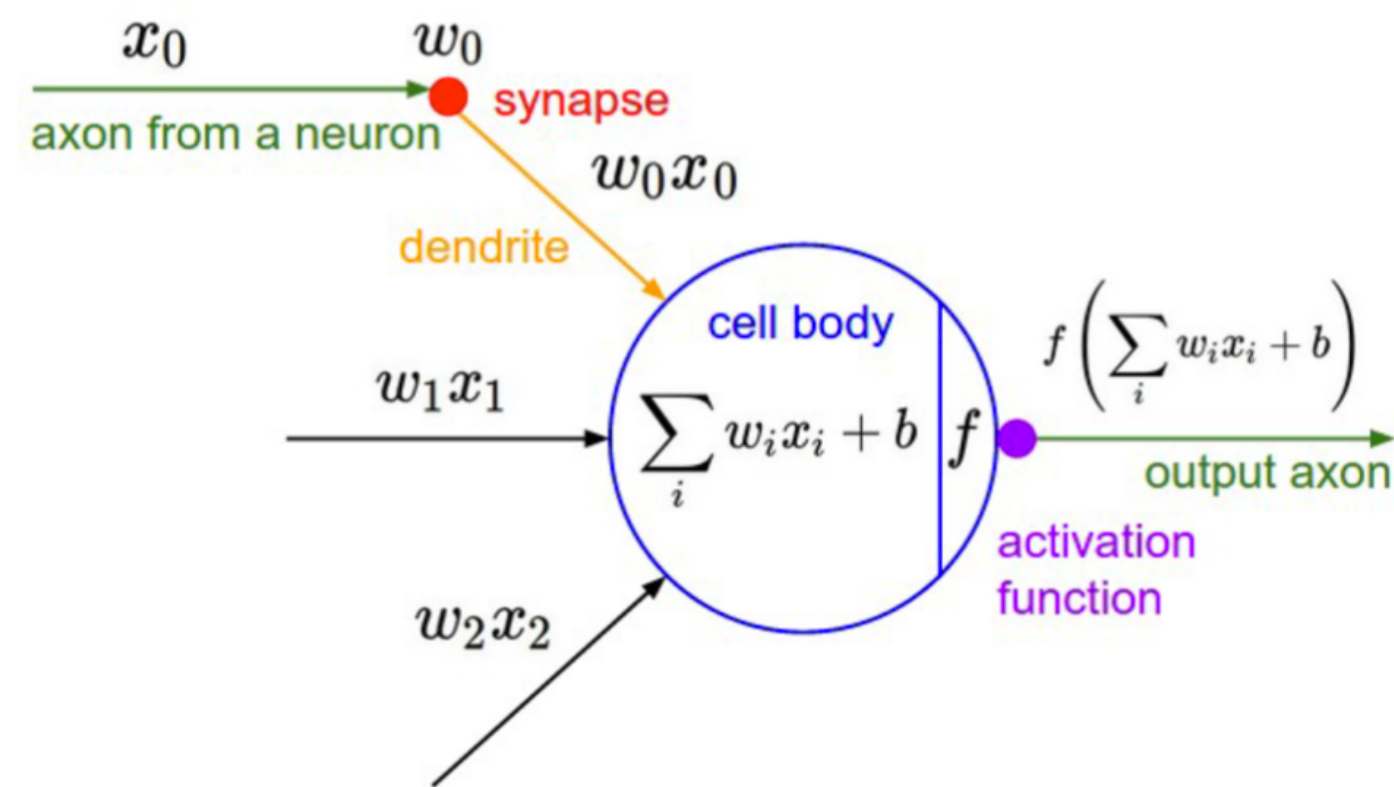
Neural networks

- The neurons are connected to each other, forming a **network**
- The output of a neuron may feed into the inputs of other neurons



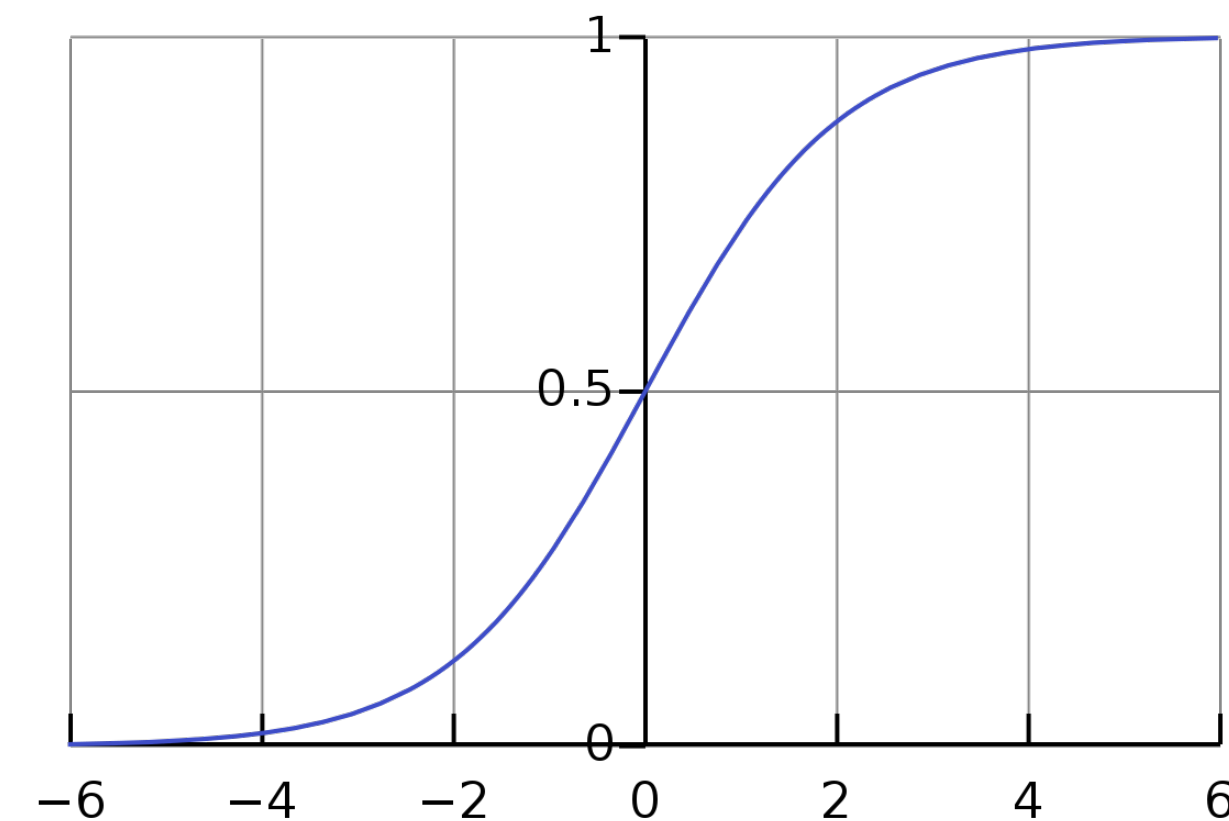
- Feed forward network (FFN)
- Fully connected network (FCN)

A neuron can be a binary logistic regression unit

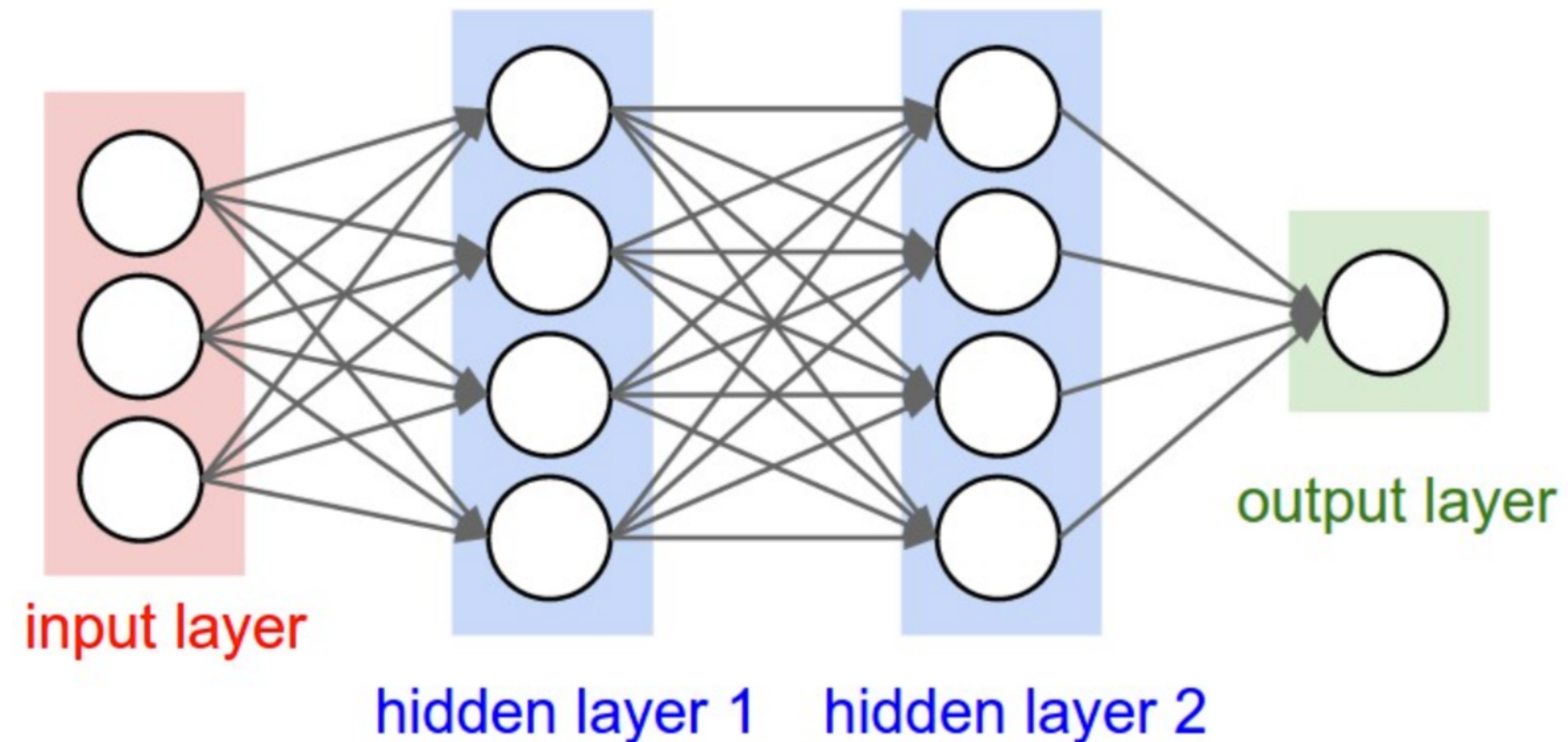


$$f(z) = \frac{1}{1 + e^{-z}}$$

$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^\top \mathbf{x} + b)$$



A neural network
= running several logistic regressions at the same time

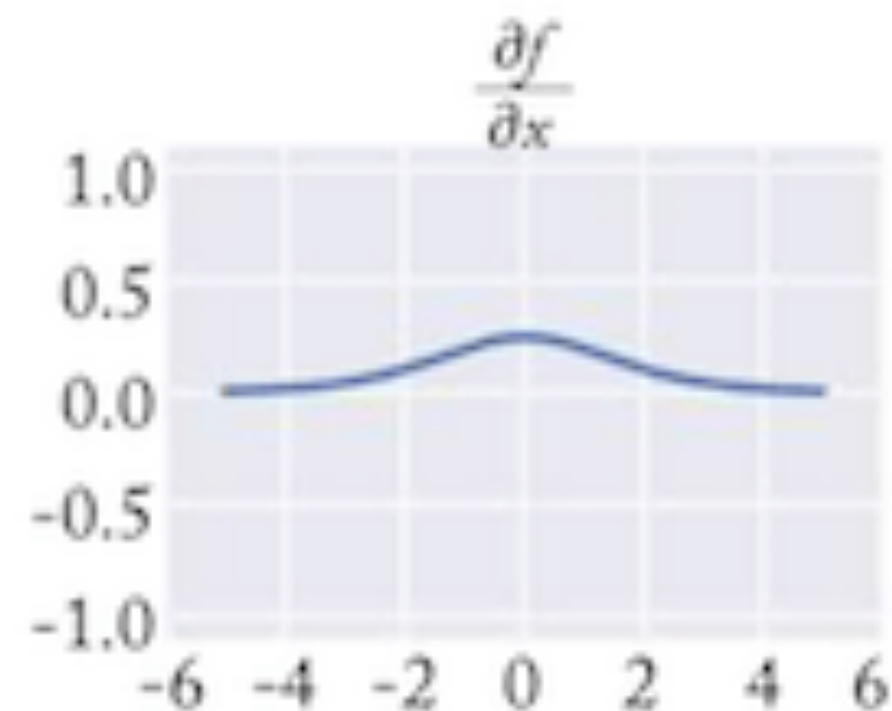
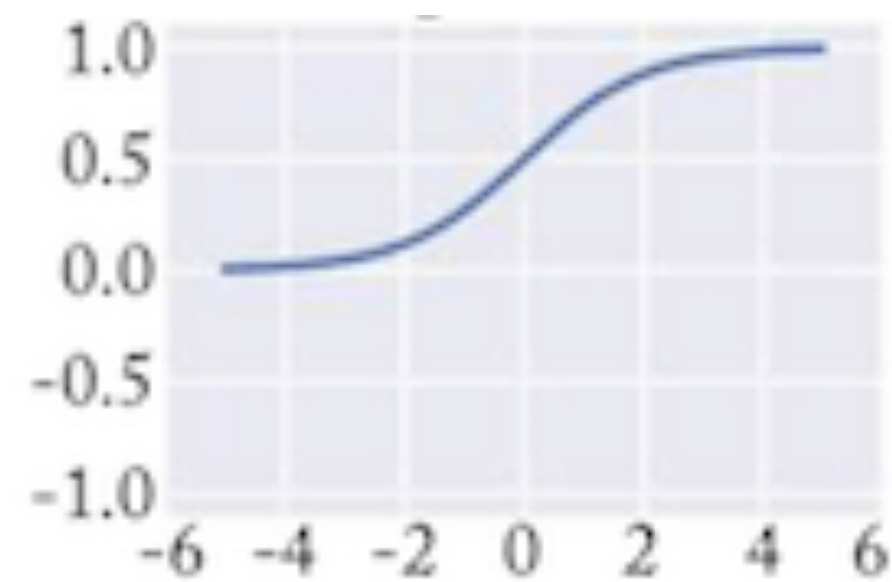


- If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs...
- which we can feed into another logistic regression function

NN has other activation functions

sigmoid

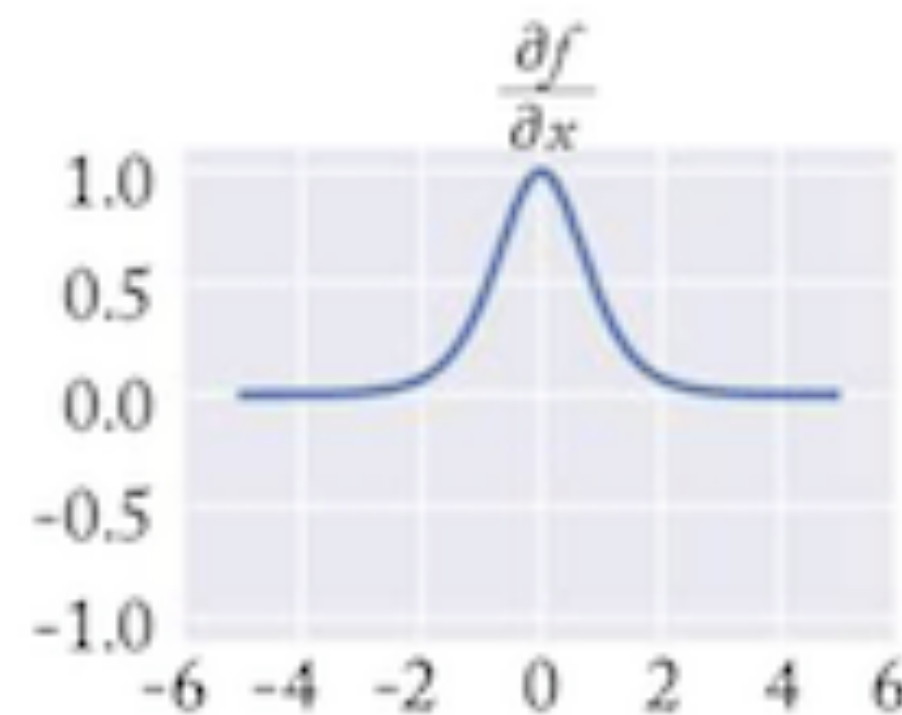
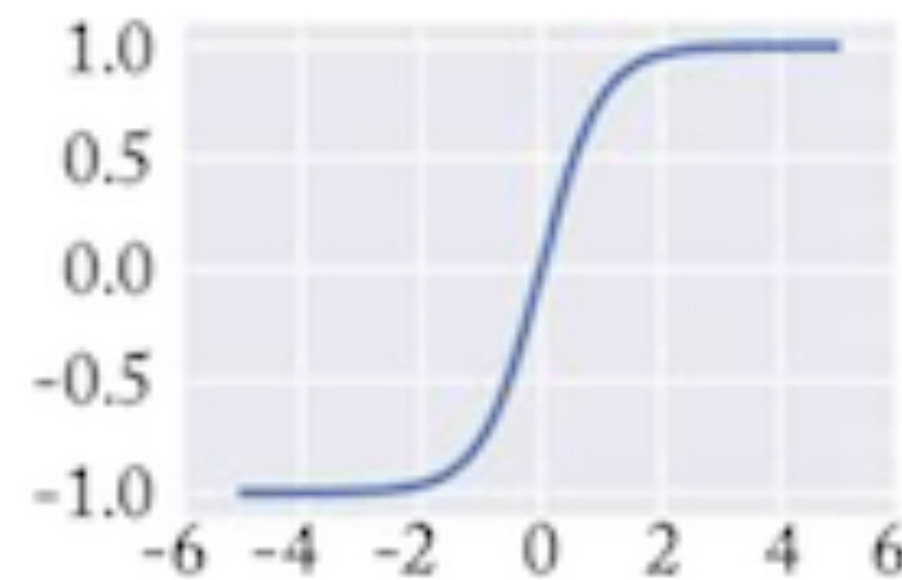
$$f(z) = \frac{1}{1 + e^{-z}}$$



$$f'(z) = f(z) \times (1 - f(z))$$

Zero centered
tanh

$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

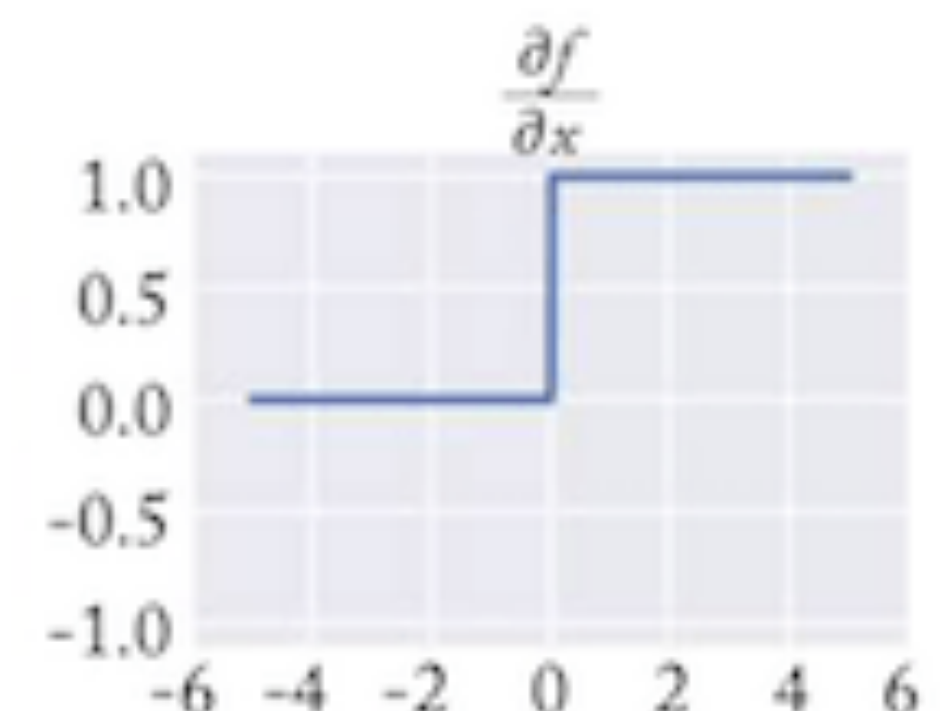
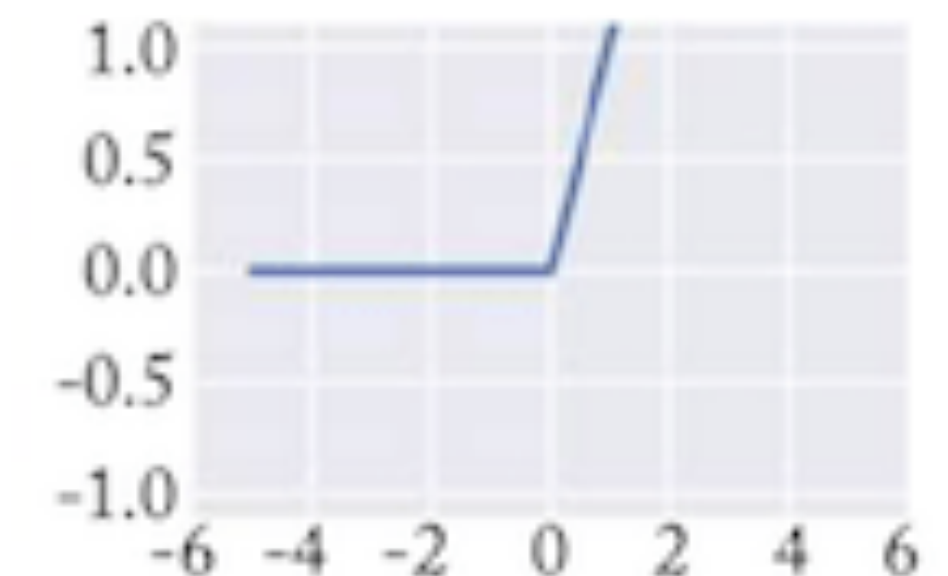


$$f'(z) = 1 - f(z)^2$$

Advantages of ReLU?

ReLU
(rectified linear unit)

$$f(z) = \max(0, z)$$



$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

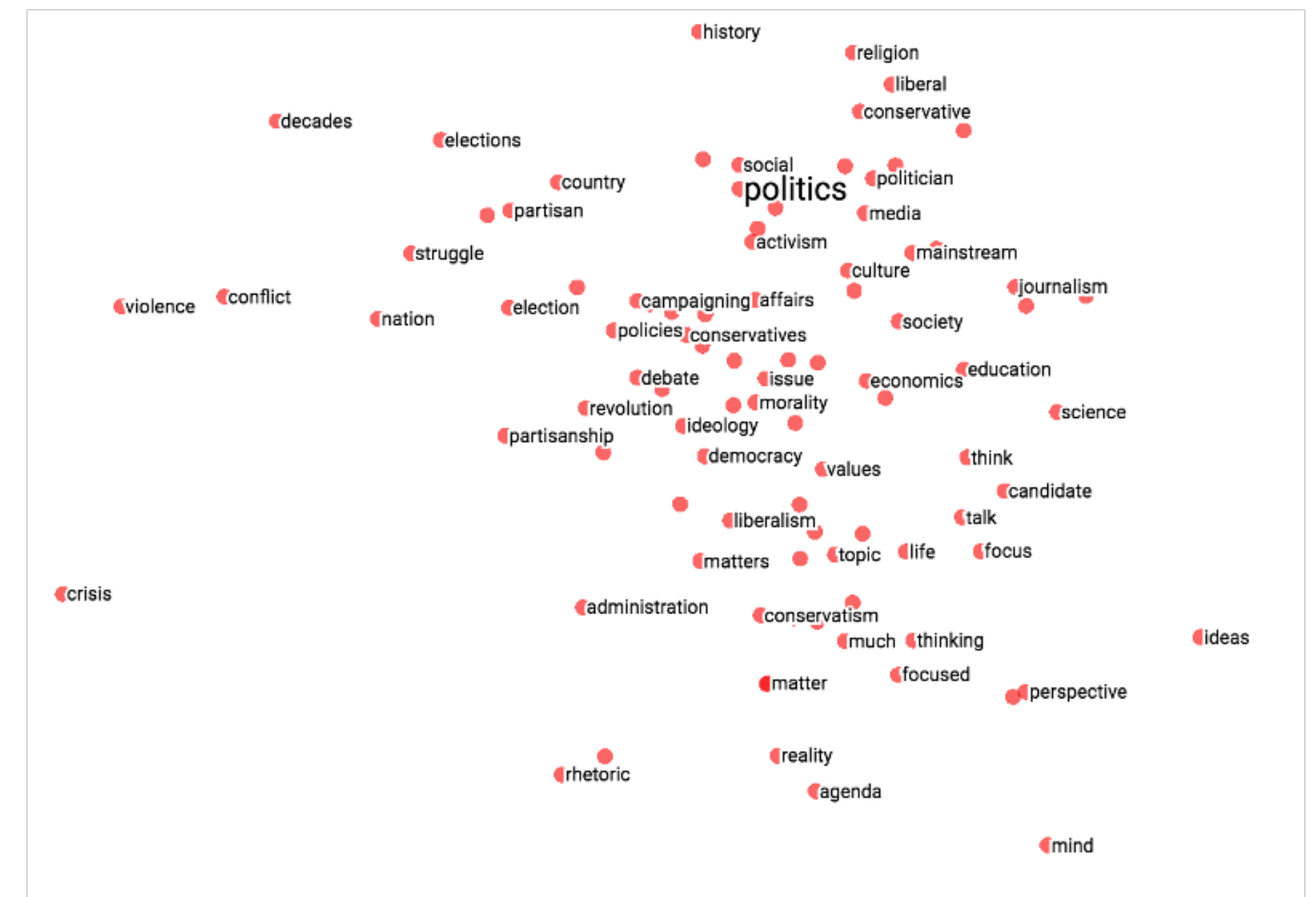
Advantages of neural networks

- Learn feature representations (no longer have to hand craft features)

Hand-crafted features

Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

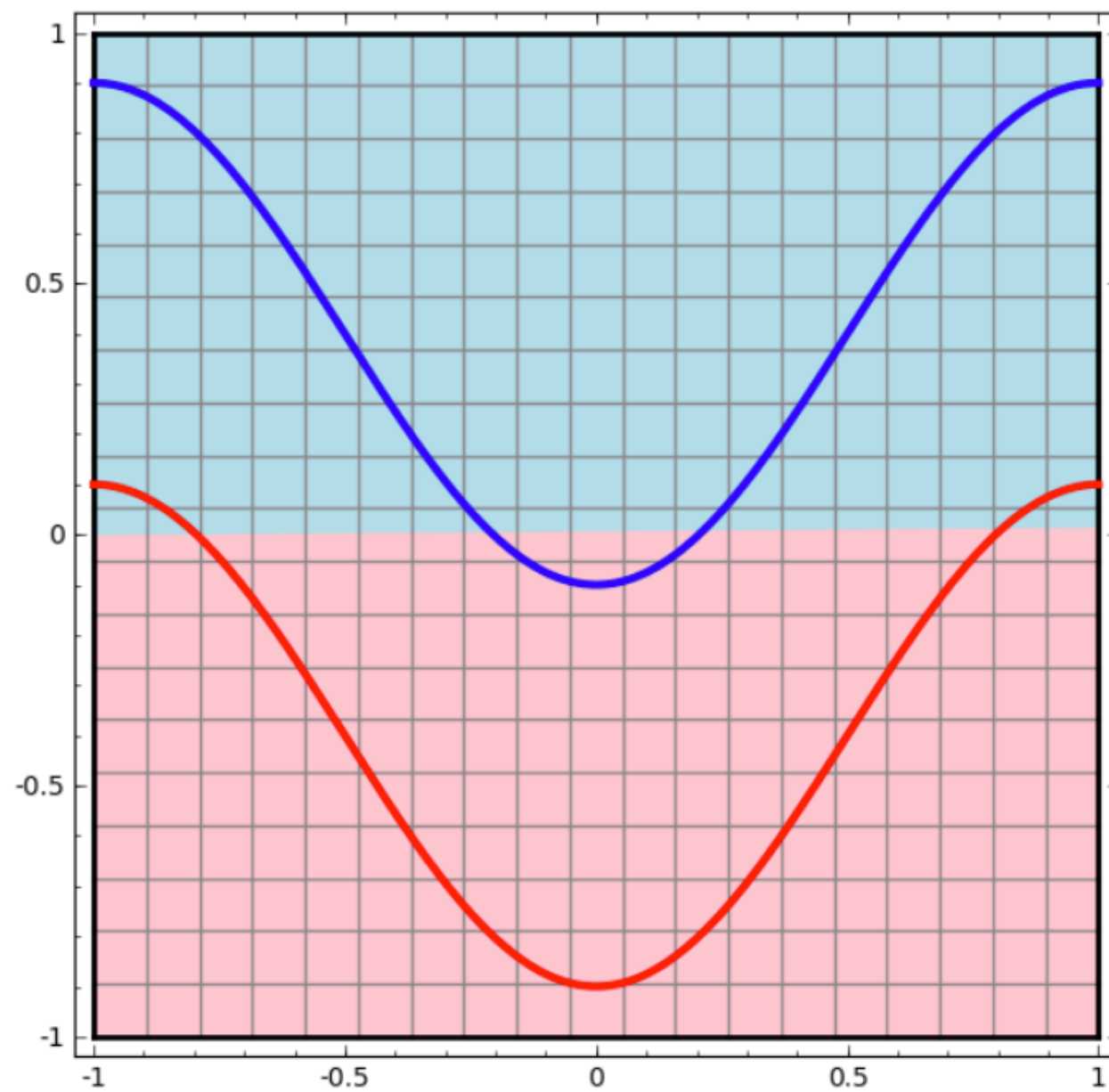
Learned representations



Advantages of neural networks

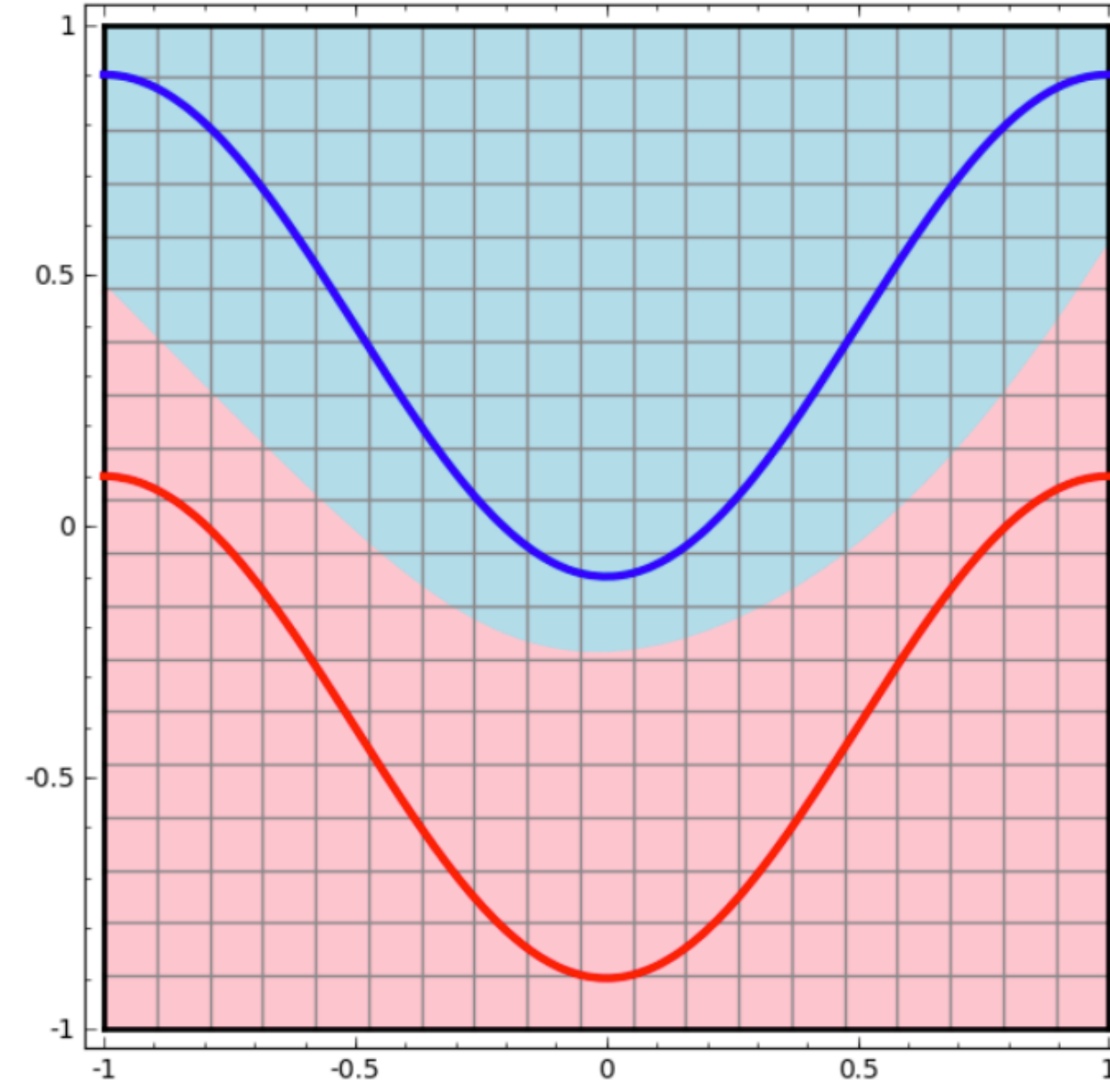
- Learn feature representations (no longer have to hand code features)
- Multiple layers allow for more complex functions with non-linear decision boundaries

Linear decision boundary



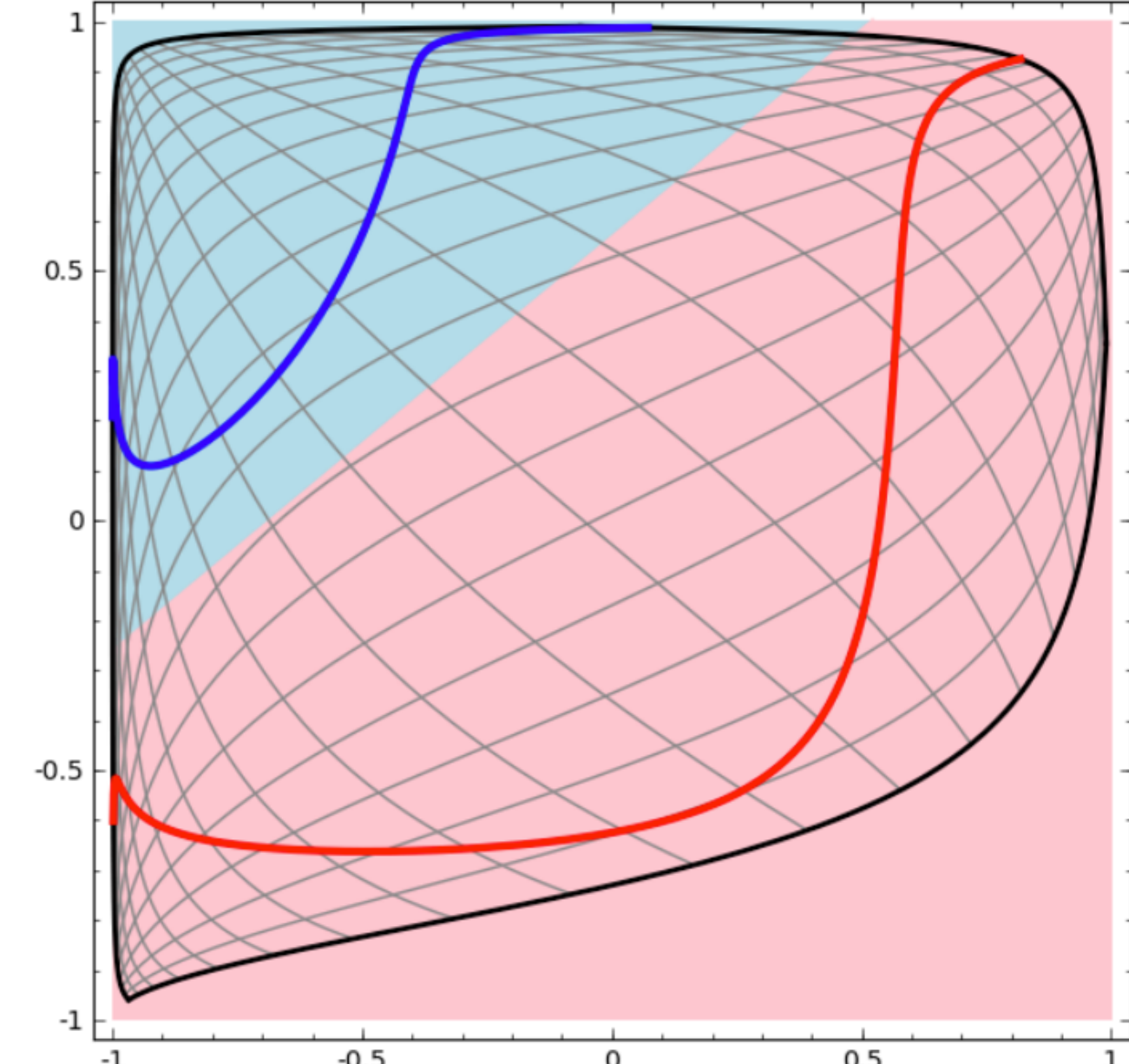
Linear classifier

Non-linear decision boundary



Neural Networks

Linear decision boundary
in transformed space



Hidden representations
are linearly separable!

Advantages of neural networks

- Learn feature representations (no longer have to hand code features)
- Multiple layers allow for more complex functions with non-linear decision boundaries
- Can build up complex models by connecting “neurons” into building blocks, and by connecting these building blocks together
- Optimization is mostly taken care of by auto-differentiation libraries
- Development of flexible deep learning libraries allow researchers and developers to focus more on modelling the problem

Back to Logistic Regression

How to learn the parameters?

Learning

- We have our **classification function** - how to assign weights and bias?
- **Goal:** predicted label $\hat{y}$ as close as possible to actual label y
- **Distance metric/Loss function** between $\hat{y}$ and y :
 $L(\hat{y}, y)$
- **Optimization algorithm** for updating weights

Loss function

- Assume $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
- $L(\hat{y}, y) =$ Measure of difference between $\hat{y}$ and y . But what form?
- **Maximum likelihood estimation** (conditional):
 - Choose $\mathbf{w}$ and b such that $\log P(y | x)$ is maximized for true labels y paired with input x
 - Similar to language models!
 - $\max \log P(w_t | w_{t-n}, \dots, w_{t-1})$ given a corpus

Cross Entropy loss

- Assume a single data point (x, y) and two classes
- Classifier probability: $P(y | x) = \hat{y}^y (1 - \hat{y})^{1-y}$
- Log probability: $\log P(y | x) = \log[\hat{y}^y (1 - \hat{y})^{1-y}]$
$$= y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$
- Loss: $-\log P(y | x) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$
$$y = 1 \Rightarrow -\log \hat{y} \qquad y = 0 \Rightarrow -\log(1 - \hat{y})$$

Example: Computing CE Loss

Var	Definition	Val
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$
- If $y = 1$ (positive sentiment), $L_{CE} = -\log(0.69) = 0.37$
- If $y = 0$ (negative sentiment), $L_{CE} = -\log(0.31) = 1.17$

Cross Entropy loss

- Assume n data points $(x^{(i)}, y^{(i)})$

- Classifier probability: $\prod_{i=1}^n P(y^{(i)} | x^{(i)}) = \prod_{i=1}^n \hat{y}^y (1 - \hat{y})^{1-y}$

- Loss: $-\log \prod_{i=1}^n P(y^{(i)} | x^{(i)}) = - \sum_{i=1}^n \log P(y^{(i)} | x^{(i)})$

$$L_{CE} = - \sum_{i=1}^n [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Maximizing likelihood = Minimizing cross entropy

Properties of CE Loss

- $L_{CE} = - \sum_{i=1}^n [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$
- Ranges from 0 (perfect predictions) to ∞
- Lower the value, better the classifier
- *Cross-entropy* between the true distribution $P(y | x)$ and predicted distribution $P(\hat{y} | x)$

$$L_{CE} = - \frac{n_{y=1}}{n} \log \hat{y} - \frac{n_{y=0}}{n} \log(1 - \hat{y}) = \sum_{y=0}^1 p(y | x) \log p(\hat{y} | x)$$

Learning (multinomial case)

- Generalize binary loss to **multinomial** CE loss:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= - \sum_{c=1}^k 1\{y = k\} \log P(y = k | x) \\ &= - \sum_{c=1}^k 1\{y = k\} \log \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^k e^{w_j \cdot x + b_j}} \end{aligned}$$

Binary CE loss

$$- \sum_{i=1}^n [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Optimization

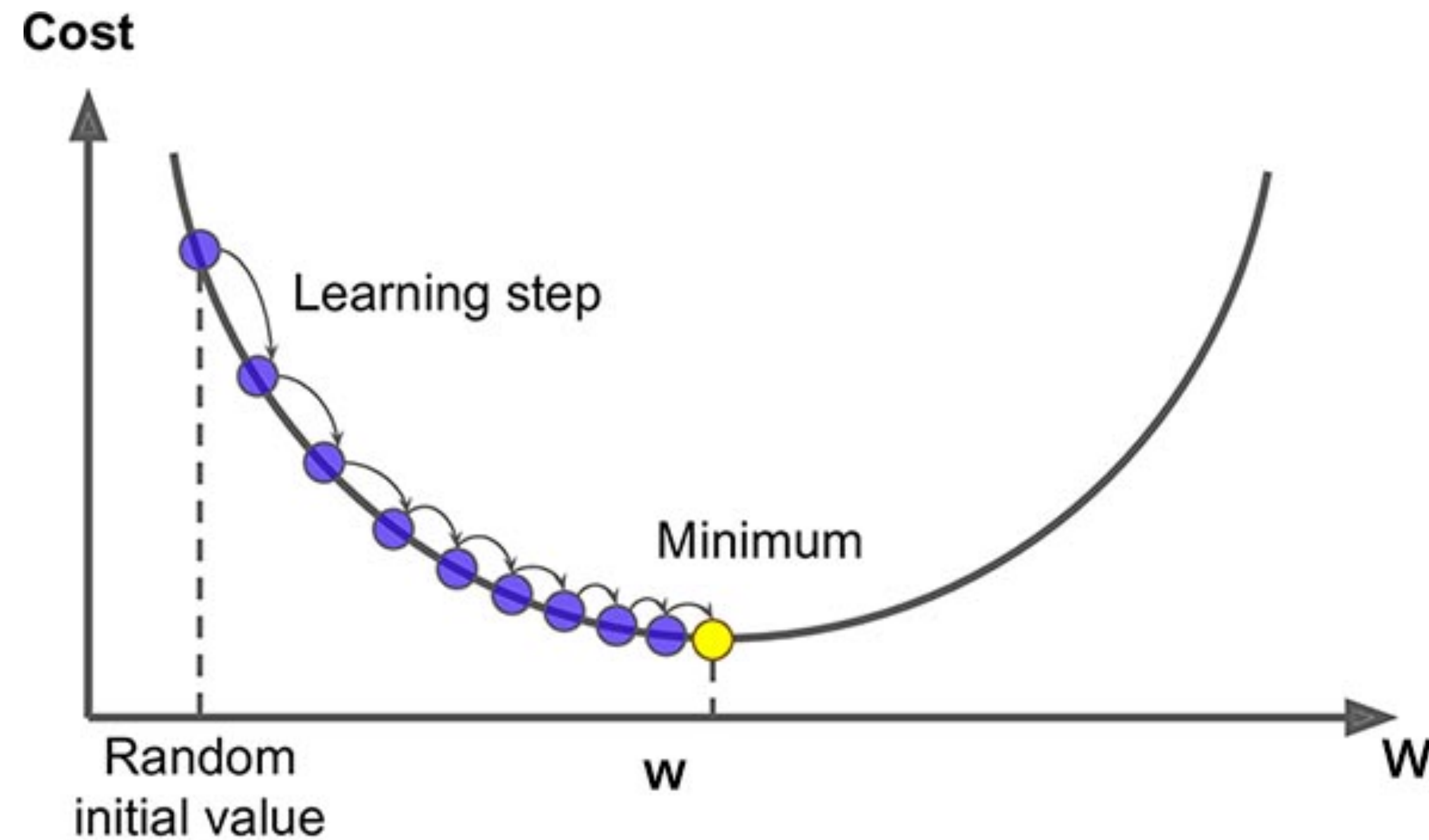
- We have our **classification function** and **loss function** - how do we find the best w and b ?

$$\theta = [w; b]$$

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- **No close form solution:** need to numerically determine optimal solution
- Gradient descent:
 - Find direction of steepest slope
 - Move in the opposite direction

Gradient descent (I-D)

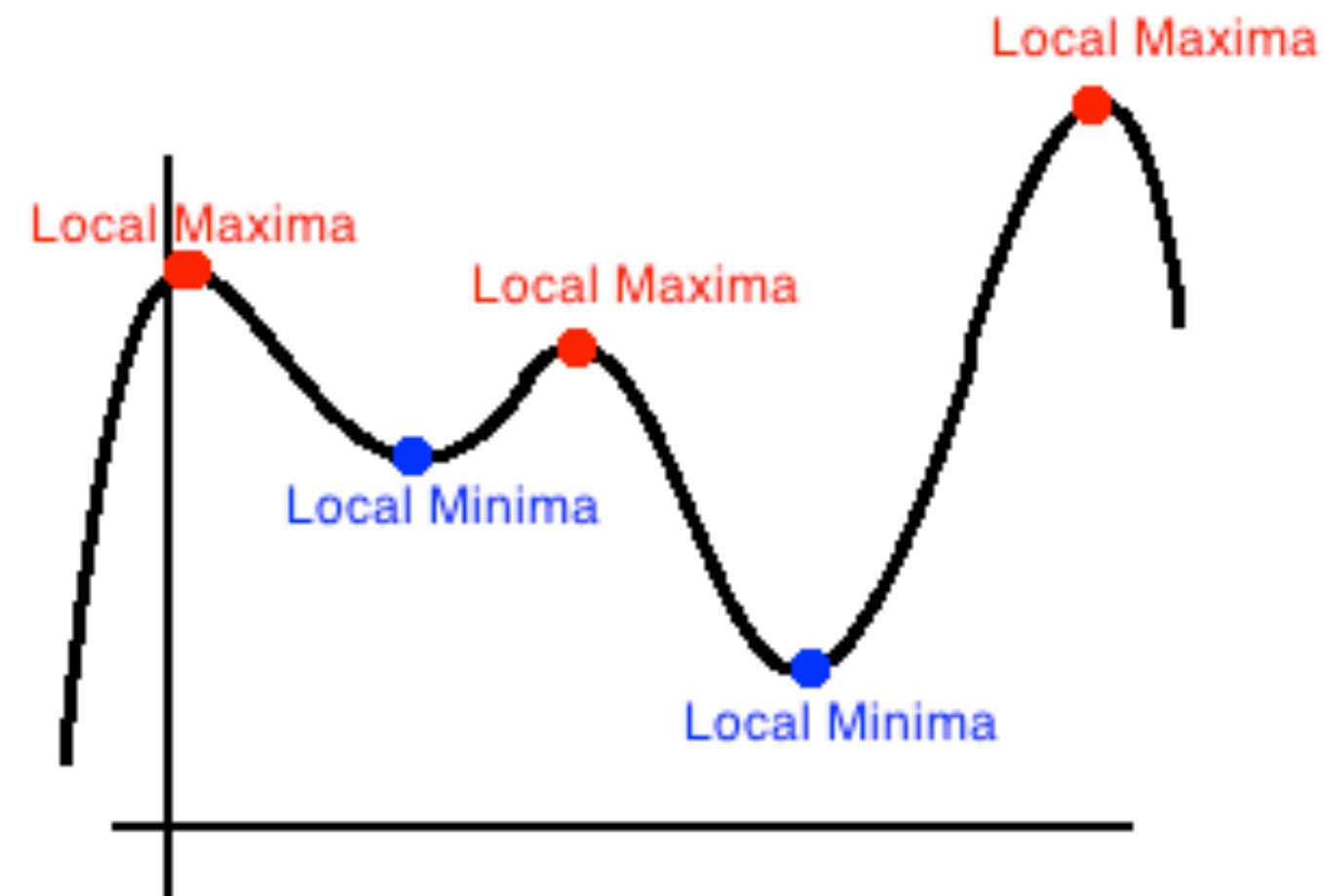


$$\theta^{t+1} = \theta^t - \eta \frac{d}{d\theta} L(f(x; \theta), y)$$

Loss function we are optimizing

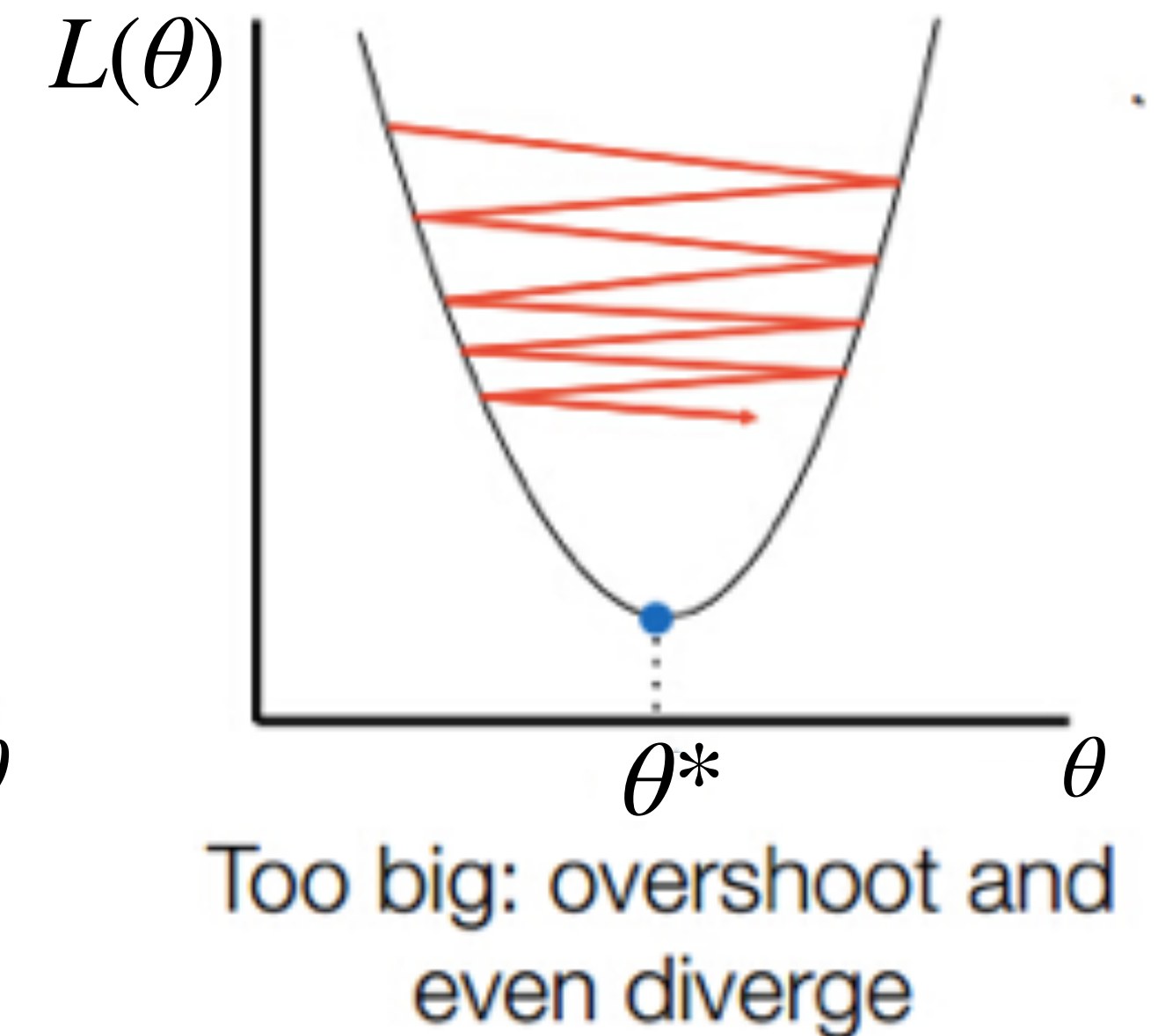
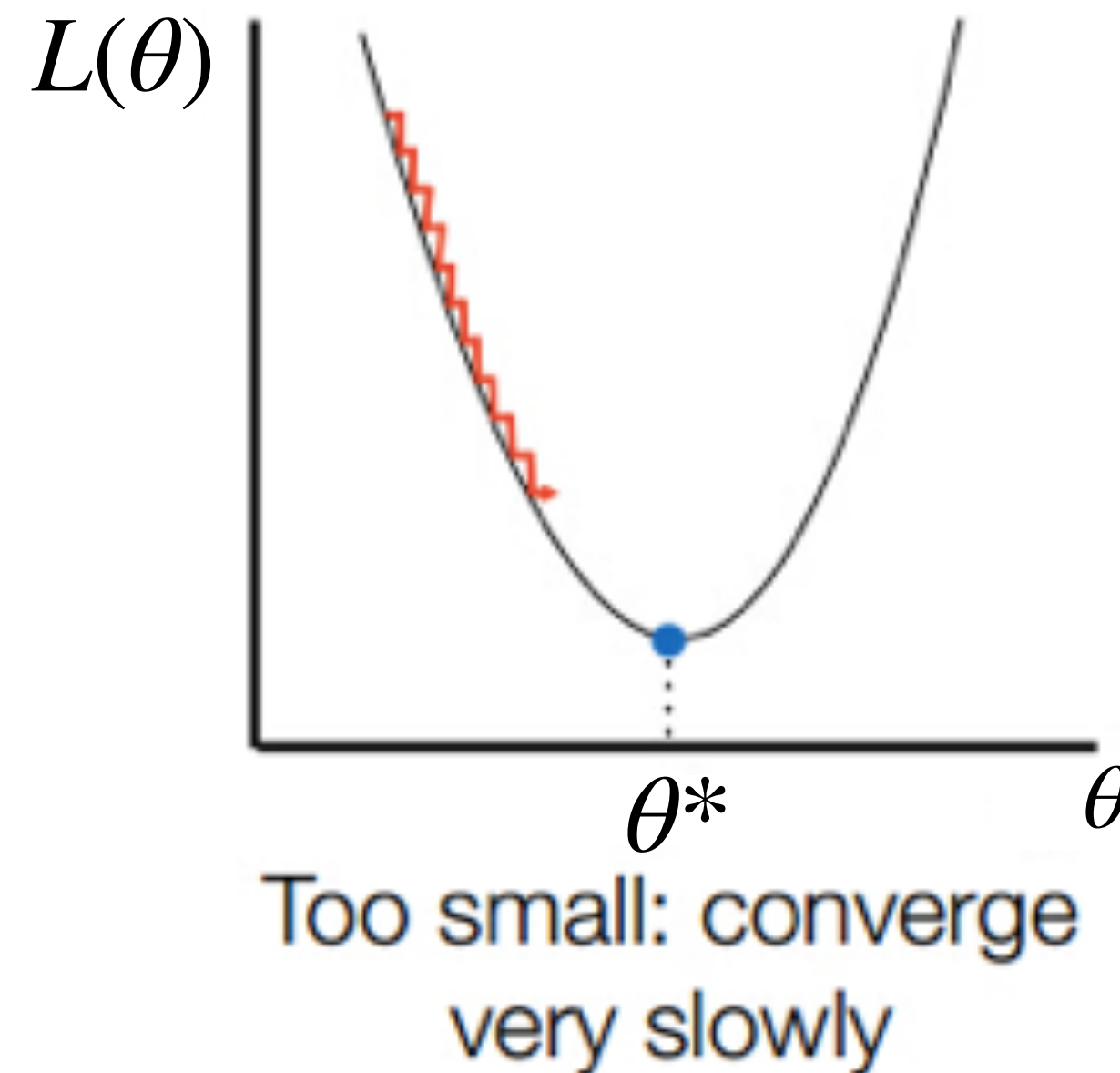
Gradient descent for LR

- Cross entropy loss for logistic regression is **convex** (i.e. has only one global minimum)
- No local minima to get stuck in
- Deep neural networks are not so easy
- Non-convex



Learning Rate

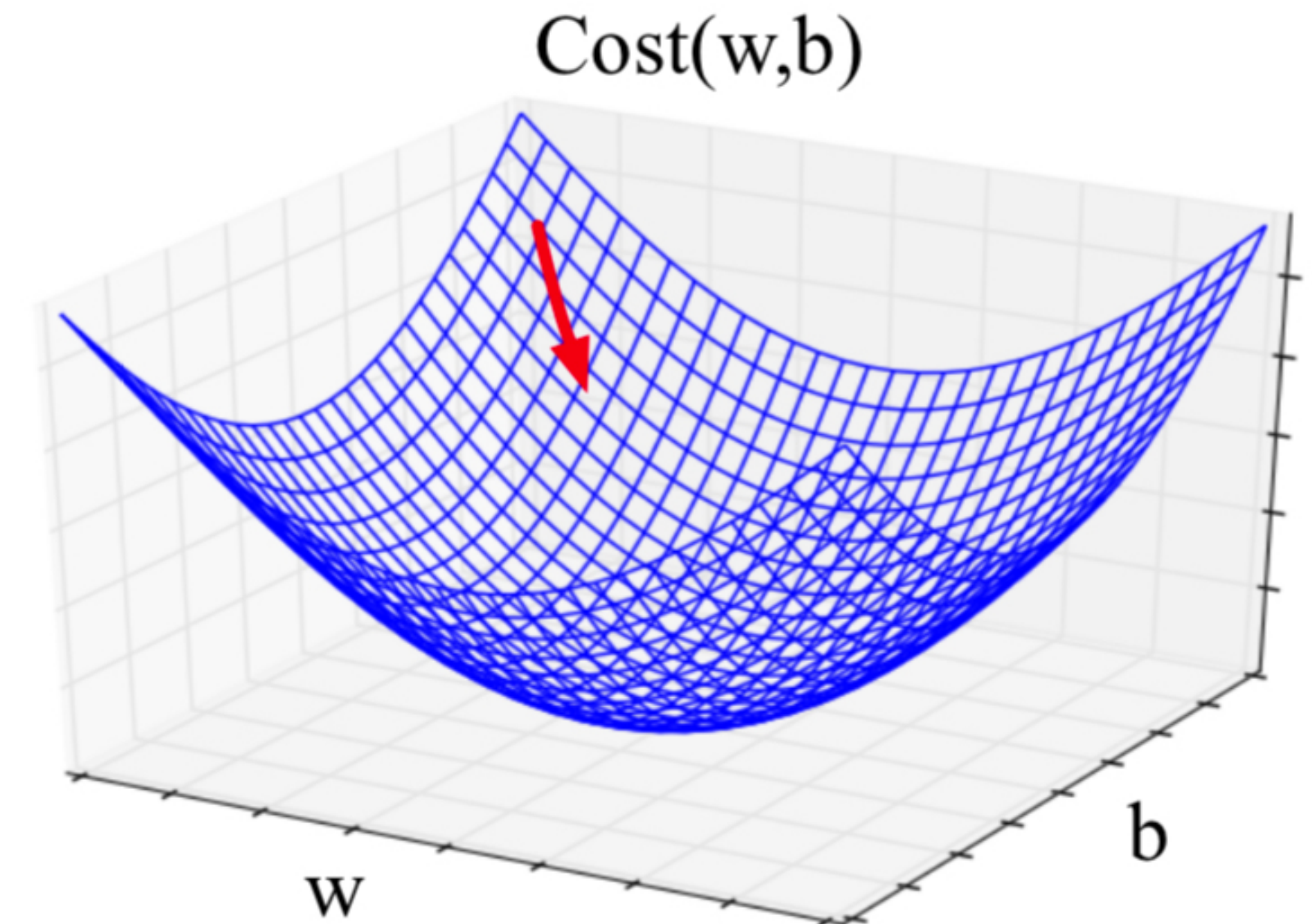
- Updates: $\theta^{t+1} = \theta^t - \eta \frac{d}{d\theta} L(f(x; \theta), y)$
- Magnitude of movement along gradient
- Higher/faster learning rate = larger updates to parameters



Gradient descent with vector weights

- In LR: weight w is a vector
- Express slope as a partial derivative of loss w.r.t each weight:

$$\nabla_{\theta} L(f(x; \theta), y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x; \theta), y) \\ \frac{\partial}{\partial w_2} L(f(x; \theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x; \theta), y) \end{bmatrix}$$



- Updates: $\theta^{(t+1)} = \theta^t - \eta \nabla L(f(x; \theta), y)$

Gradient for logistic regression

$$L_{CE} = - \sum_{i=1}^n [y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log(1 - \sigma(w \cdot x^{(i)} + b))]$$

$$\text{Gradient, } \frac{dL_{CE}(w, b)}{dw_j} = \sum_{i=1}^n [\underbrace{\sigma(w \cdot x^{(i)} + b) - y^{(i)}}_{\text{Diff between true } y \text{ and prediction}}] x_j^{(i)}$$

$$\frac{dL_{CE}(w, b)}{db} = \sum_{i=1}^n [\sigma(w \cdot x^{(i)} + b) - y^{(i)}]$$

input
feature
value

Properties of the logistic function

- Relationship to $\sigma(-z)$

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

$$\Rightarrow \sigma(-z) = 1 - \sigma(z)$$

- Derivative

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

$$\frac{d\sigma(z)}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \frac{1}{e^z + 1} = \sigma(z)\sigma(-z) = \sigma(z)(1 - \sigma(z))$$

Taking the gradient

$$L_{CE} = - \sum_{i=1}^n [y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log(1 - \sigma(w \cdot x^{(i)} + b))]$$

Consider cross-entropy loss for the i th instance / data point with $z = w \cdot x^{(i)} + b$

$$\frac{dL_{CE}^{(i)}}{dw_j} = \boxed{\frac{dL_{CE}^{(i)}}{dz}} \frac{dz}{dw_j} \qquad \frac{dL_{CE}^{(i)}}{db} = \boxed{\frac{dL_{CE}^{(i)}}{dz}} \frac{dz}{db}$$

$$\frac{dL_{CE}^{(i)}}{dz} = - \left[y^{(i)} \frac{\sigma'(z)}{\sigma(z)} + (1 - y^{(i)}) \frac{-\sigma'(z)}{1 - \sigma(z)} \right] \qquad \leftarrow \frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$

$$= - \left[y^{(i)} \frac{\sigma(z)(1 - \sigma(z))}{\sigma(z)} + (1 - y^{(i)}) \frac{-\sigma(z)(1 - \sigma(z))}{1 - \sigma(z)} \right]$$

$$= - [y^{(i)}(1 - \sigma(z)) + (1 - y^{(i)})(-\sigma(z))] = - [y^{(i)} - \sigma(z)] = \sigma(z) - y^{(i)}$$

$$\boxed{\frac{dL_{CE}^{(i)}}{dz} = \sigma(z) - y^{(i)}}$$

Taking the gradient

$$L_{CE} = - \sum_{i=1}^n [y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log(1 - \sigma(w \cdot x^{(i)} + b))]$$

$$\frac{dL_{CE}^{(i)}}{dz} = \sigma(z) - y^{(i)} \qquad z = w \cdot x^{(i)} + b$$

$$\frac{dL_{CE}^{(i)}}{dw_j} = \frac{dL_{CE}^{(i)}}{dz} \frac{dz}{dw_j} = (\sigma(w \cdot x^{(i)} + b) - y^{(i)}) x_j^{(i)} \qquad \frac{dz}{dw_j} = x_j^{(i)}$$

$$\frac{dL_{CE}^{(i)}}{db} = \frac{dL_{CE}^{(i)}}{dz} \frac{dz}{db} = (\sigma(w \cdot x^{(i)} + b) - y^{(i)}) \qquad \frac{dz}{db} = 1$$

Learning (multinomial case)

- Generalize binary loss to **multinomial** CE loss:

$$\begin{aligned} L_{CE}(\hat{y}, y) &= - \sum_{c=1}^k 1\{y = c\} \log P(y = c | x) \\ &= - \sum_{c=1}^k 1\{y = c\} \log \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^k e^{w_j \cdot x + b_j}} \end{aligned}$$

- Gradient for multinomial CE loss:

$$\begin{aligned} \frac{dL_{CE}}{dw_c} &= - (1\{y = c\} - P(y = c | x)) x_c \\ &= - \left(1\{y = c\} - \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^k e^{w_j \cdot x + b_j}} \right) x_c \end{aligned}$$

Binary CE loss

$$- \sum_{i=1}^n [y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Gradient of Binary CE loss

$$\frac{dL_{CE}}{dz} = \sigma(z) - y$$

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after each training example

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x , y) **returns** θ

where: L is the loss function

f is a function parameterized by θ

x is the set of training inputs $x^{(1)}, x^{(2)}, \dots, x^{(n)}$

y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \dots, y^{(n)}$

$\theta \leftarrow 0$

repeat til done # see caption

For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting): # How are we doing on this tuple?

Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?

Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?

2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?

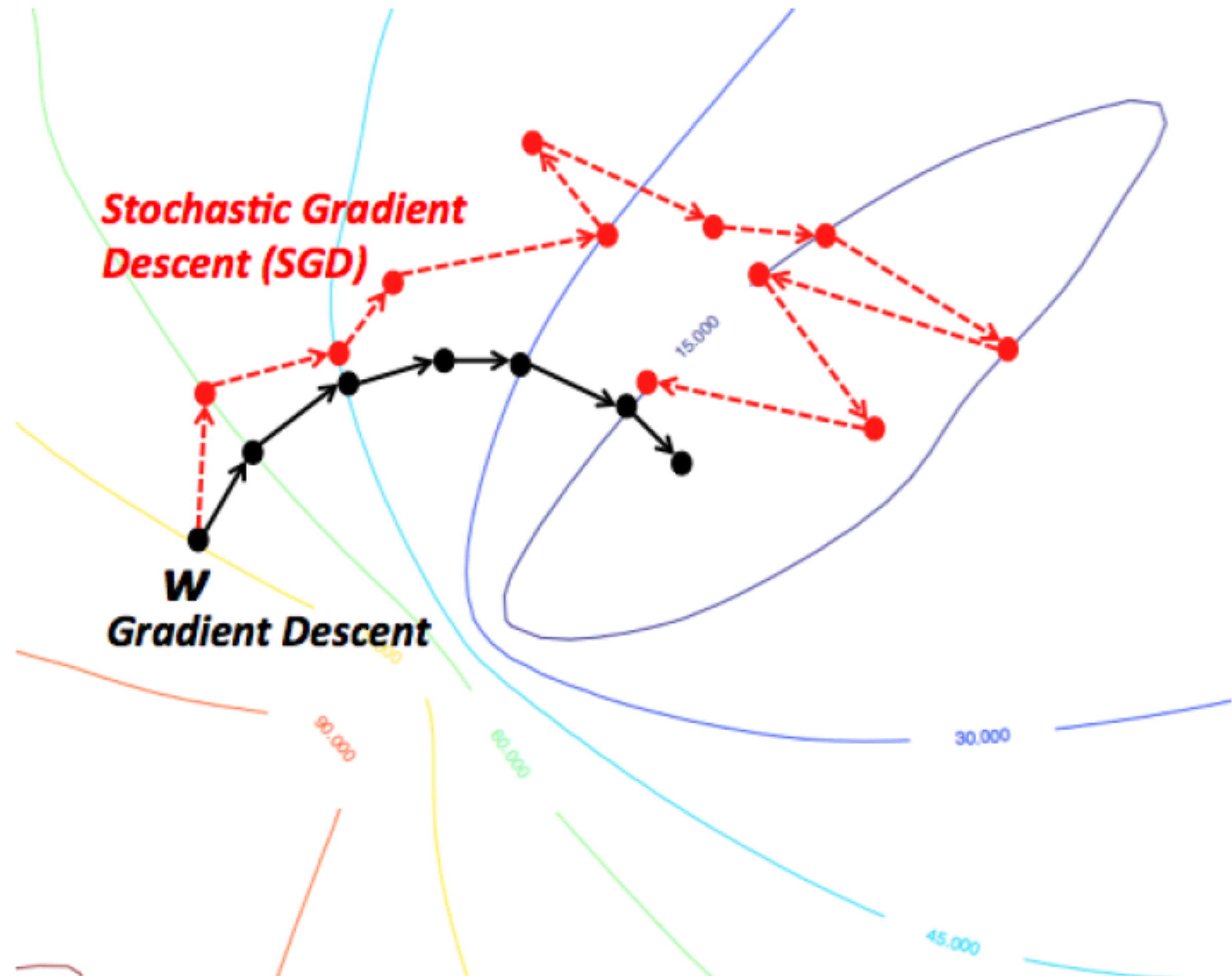
3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead

return θ

Per
Instance
Loss

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after each training example



Regularization

- Training objective: $\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^n \log P(y^{(i)} | x^{(i)})$
- This might fit the training set too well! (including noisy features)
- Poor generalization to the unseen test set — **Overfitting**

- **Regularization** helps prevent overfitting

$$\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^n \log P(y^{(i)} | x^{(i)}) - \alpha R(\theta)$$

Penalize
large
weights

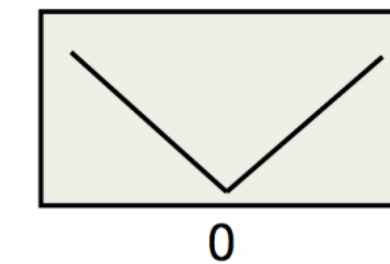
L2 regularization

- $R(\theta) = ||\theta||_2 = \sum_{j=1}^d \theta_j^2$
- **Euclidean distance** of weight vector θ from origin
- L2 regularized objective:

$$\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^n \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^d \theta_j^2$$

L1 Regularization

- $R(\theta) = ||\theta||_1 = \sum_{j=1}^d |\theta_j|$

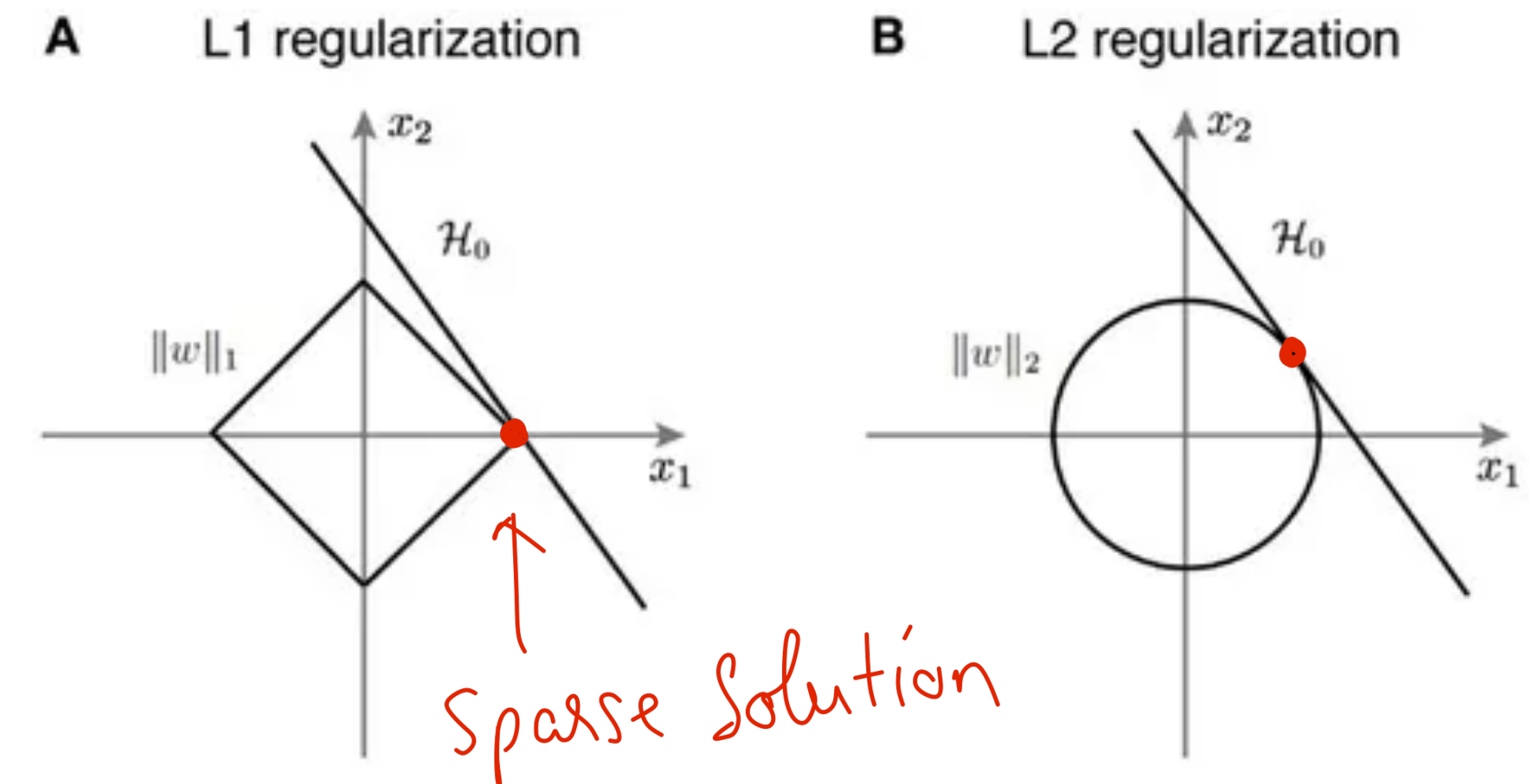


- **Manhattan distance** of weight vector θ from origin
- L1 regularized objective:

$$\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^n \log P(y^{(i)} | x^{(i)}) - \alpha \sum_{j=1}^d |\theta_j|$$

L2 vs L1 regularization

- L2 is easier to optimize - simpler derivation
- L1 is complex since the derivative of $|\theta|$ is not continuous at 0
- L2 leads to many small weights (due to θ^2 term)
- L1 prefers *sparse* weight vectors with many weights set to 0 (i.e. far fewer features used)



Summary

- Logistic regression - a discriminative classification model that models $P(c | d)$ directly
- Can have more flexible features than Naive Bayes
- Use optimizer to learn parameters
 - For logistic regression, maximizing the likelihood is the equivalent to minimizing the cross entropy loss
 - Likelihood function is convex, but MLE does not have closed form solution
 - Use gradient descent to determine parameters
- Connection to neural networks