Linear Regression. Logistic Regression. Overfitting and Regularization.

COMP5318 Machine Learning and Data Mining

semester 2, 2024, week 3

Nguyen Hoang Tran

Based on slides prepared by V. Smith (CMU)

THE UNIVERSITY OF SYDNEY

Outline

- Linear Regression
- Gradient Descent (GD), SGD, and Minibatch SGD
- Logistic Regression (Classification)

Terminology

EXAMPLE: SPAM DETECTION

Observations (n). Items or entities used for learning or evaluation, e.g., emails

Features (k). Attributes (typically numeric) used to represent an observation, e.g., length, date, presence of keywords

Labels. Values / categories assigned to observations, e.g., {*spam*, *not-spam*}

Training and Test Data. Observations used to train and evaluate a learning algorithm, e.g., a set of emails along with their labels

- Training data is given to the algorithm for training
- Test data is withheld at train time

Supervised learning

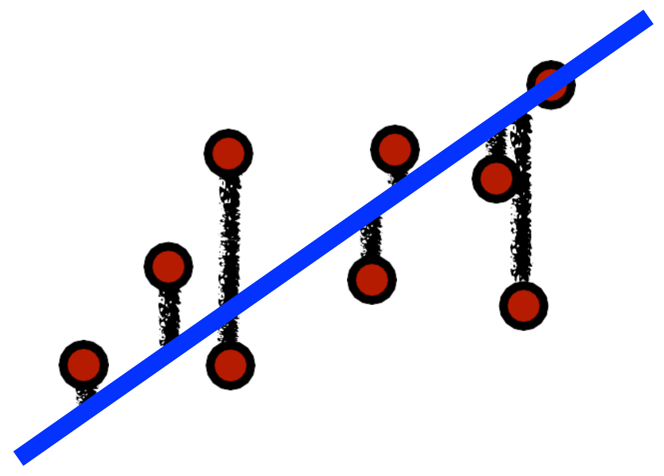
In *supervised learning*, you have access to input variables, *X*, and outputs, *Y*, and the goal is to predict an output given an input

- Examples:
  - Cat vs. dog (classification): predict whether a picture is a cat or a dog
  - Housing prices (regression): predict price of a house based on features (size, location, etc)

Outline

- Linear Regression
- Gradient Descent (GD), SGD, and Minibatch SGD
- Logistic Regression (Classification)

Linear regression



Example: Predicting house price from size, location, age

We can augment the feature vector to incorporate offset:

$$\mathrm{x}^T = [1 \quad x_1 \quad x_2 \quad x_3 \;]$$

We can then rewrite this linear mapping as scalar product:

$$y \approx \hat{y} = \sum_{i=0}^{3} w_i x_i = \mathrm{w}^{\mathrm{T}} \mathrm{x}$$

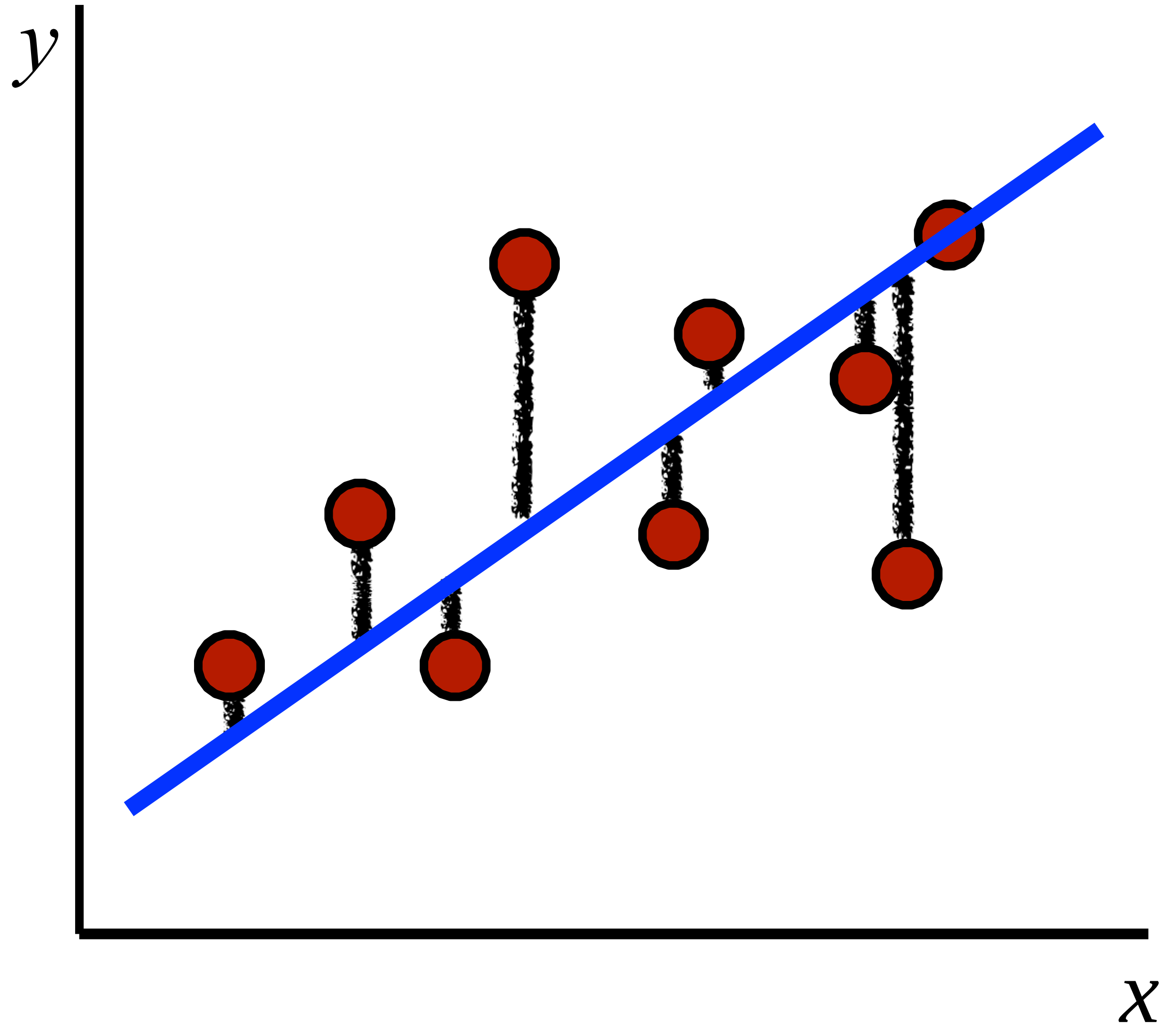
1D example

Goal: find the line of best fit

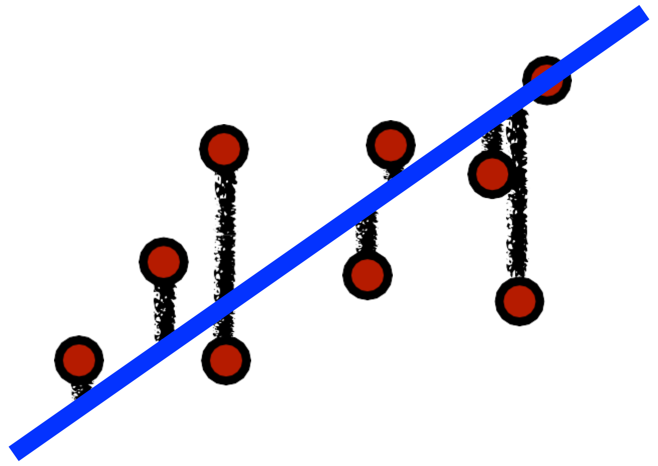
x coordinate: features

y coordinate: labels

$$y \approx \hat{y} = \underbrace{w_0}_{\text{Intercept / Offset}} + \underbrace{w_1}_{\text{Slope}} x$$



Evaluating predictions



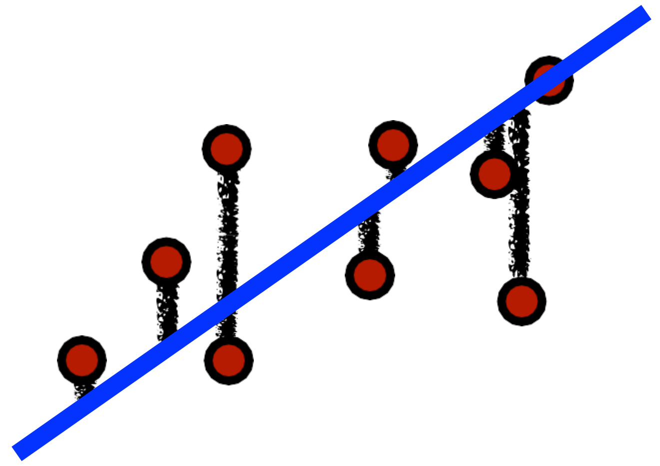
Can measure 'closeness' between label and prediction

- House price: better to be incorrect by $50 than $50,000
- Song year prediction: better to be off by a year than by 20 years

What is an appropriate evaluation metric or 'loss' function?

- Absolute loss: $|y - \hat{y}|$
- Square loss: $(y - \hat{y})^2$ ← Has nice mathematical properties

How can we learn model (w)?



Assume we have *n* training points, where $\mathrm{x}^{(i)}$ denotes the *i*th point

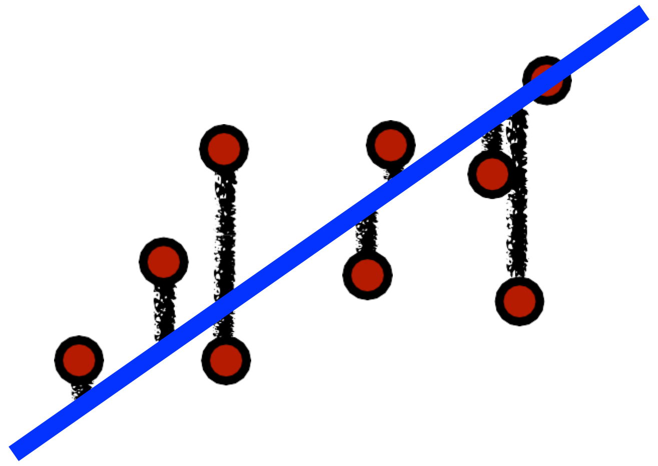
Recall two earlier points:

- *Linear* assumption: $\hat{y} = \mathrm{w}^T \mathrm{x}$
- We use *square loss*: $(y - \hat{y})^2$

Idea: Find **w** that minimizes square loss over training points:

$$\min_{\mathrm{w}} \sum_{i=1}^{n} (\underbrace{\mathrm{w}^{\mathrm{T}} \mathrm{x}^{(i)}}_{\hat{y}^{(i)}} - y^{(i)})^2$$

How can we learn model (w)?



Given n training points with k features, we define:

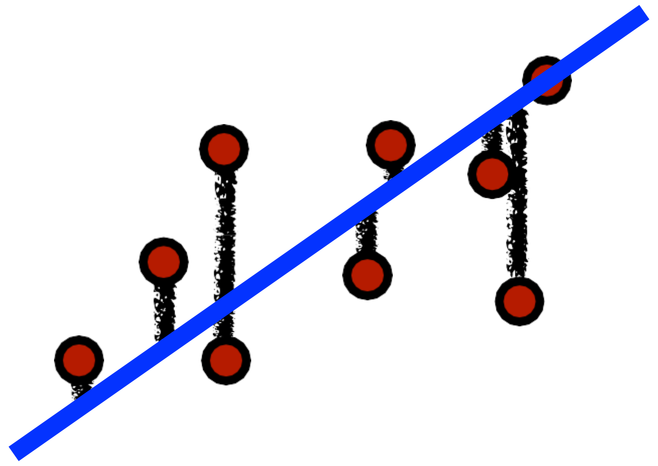
- $\mathbf{X} \in \mathbb{R}^{n \times k}$: matrix storing points
- $\mathbf{y} \in \mathbb{R}^{n}$: real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^{n}$: predicted labels, where $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^{k}$: regression parameters / model to learn

Least Squares Regression: Learn mapping ($\mathbf{w}$) from features to labels that minimizes residual sum of squares:

$$\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

Equivalent $\min_{\mathrm{w}} \sum_{i=1}^{n} (\mathrm{w}^{\mathrm{T}} \mathrm{x}^{(\mathrm{i})} - y^{(i)})^2$ by definition of Euclidean norm

How can we learn model (w)?



Find solution by setting derivative to zero

1D: $$f(w) = \|w\mathrm{x} - \mathrm{y}\|_2^2 = \sum_{i=1}^{n} (wx^{(i)} - y^{(i)})^2$$

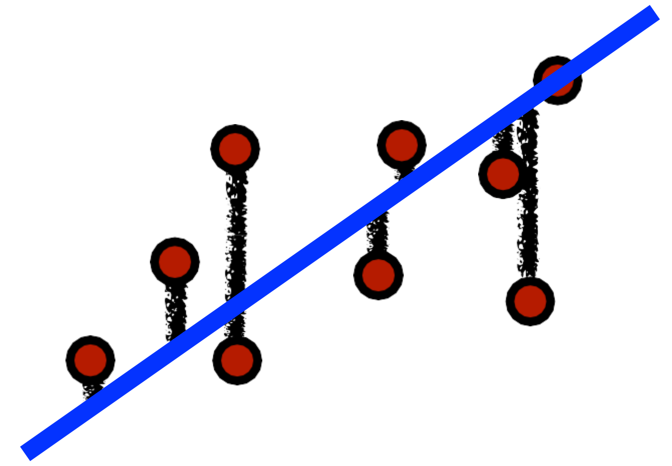
$$\frac{df}{dw}(w) = 2\sum_{i=1}^{n} x^{(i)}(wx^{(i)} - y^{(i)}) = 0$$

Least Squares Regression: Learn mapping from features to labels that minimizes residual sum of squares:

$$\min_{\mathbf{w}} ||\mathbf{Xw} - \mathbf{y}||_2^2$$

Closed form solution: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ (if inverse exists)

Given n training points with k features, we define:



- $\mathbf{X} \in \mathbb{R}^{n \times k}$: matrix storing points
- $\mathbf{y} \in \mathbb{R}^{n}$: real-valued labels
- $\hat{\mathbf{y}} \in \mathbb{R}^{n}$: predicted labels, where $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$
- $\mathbf{w} \in \mathbb{R}^{k}$: regression parameters / model to learn

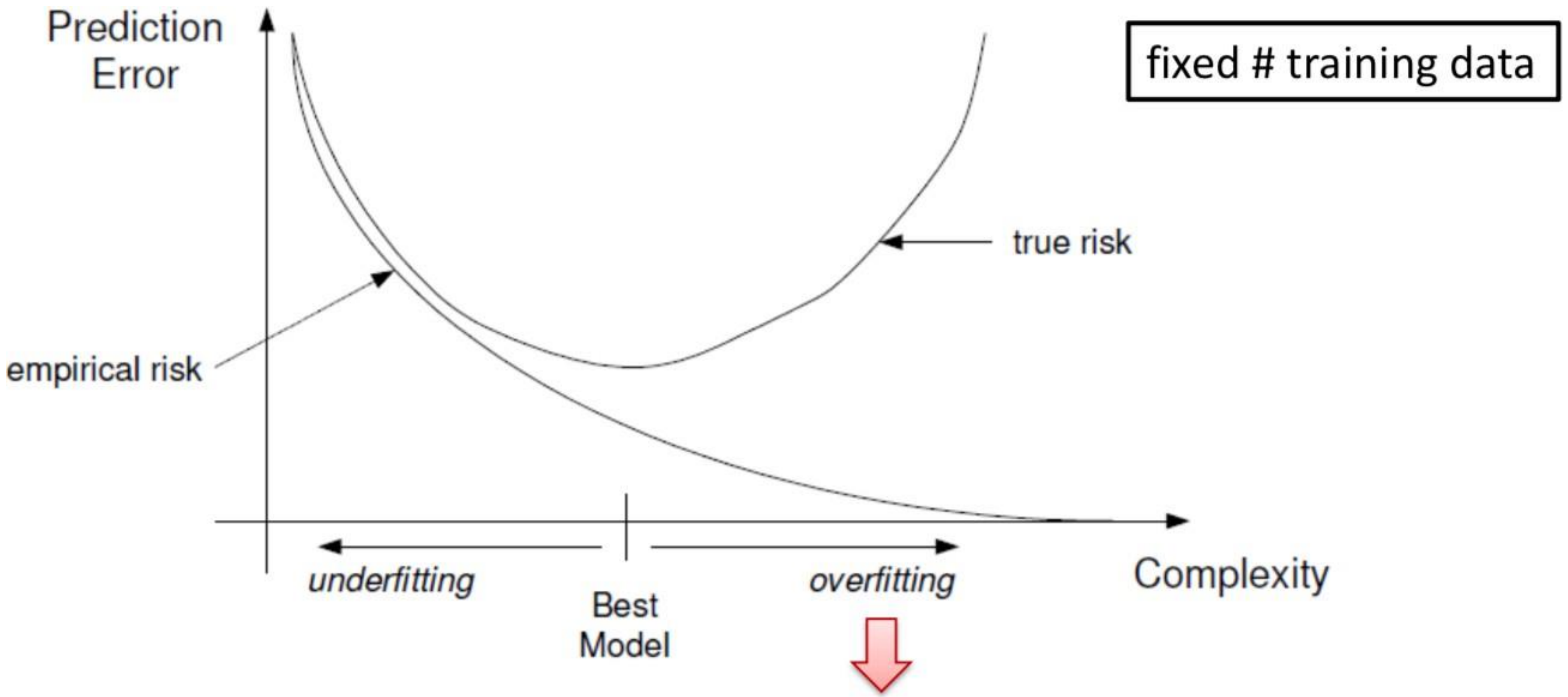
Ridge Regression: Learn mapping ($\mathbf{w}$) that minimizes residual sum of squares along with a regularization term:

$$\min_{\mathbf{w}} \overbrace{||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2}^{\text{Training Error}} + \overbrace{\lambda ||\mathbf{w}||_2^2}^{\text{Model Complexity}}$$

free parameter trades off between training error and model complexity

Closed-form solution: $\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_k)^{-1} \mathbf{X}^\top \mathbf{y}$

Overfitting and generalization



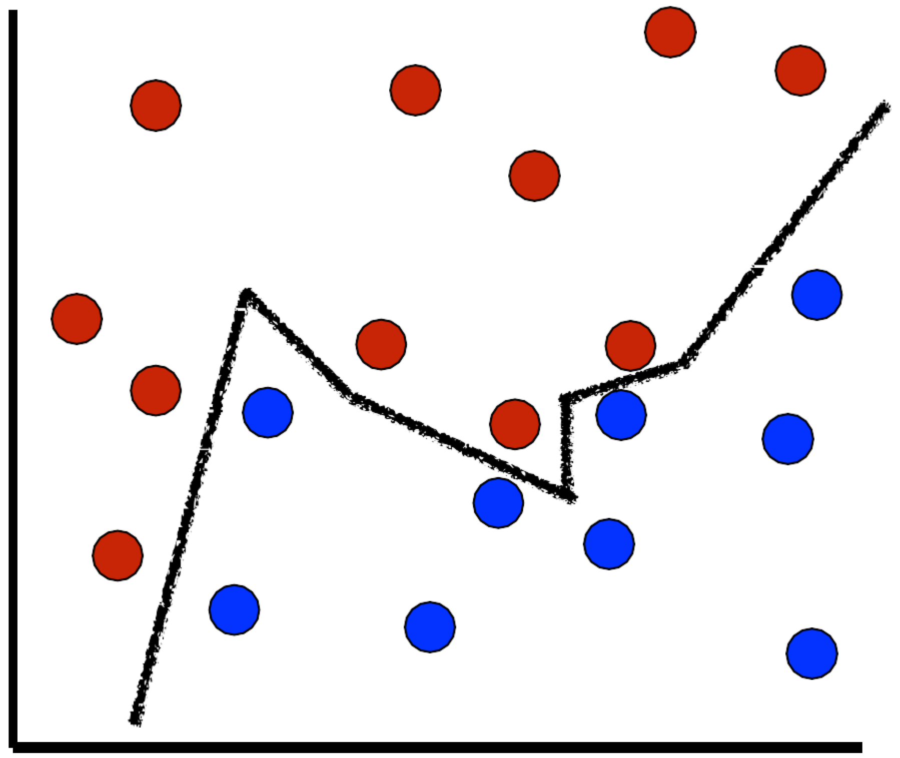
Empirical risk is no longer a good indicator of true risk

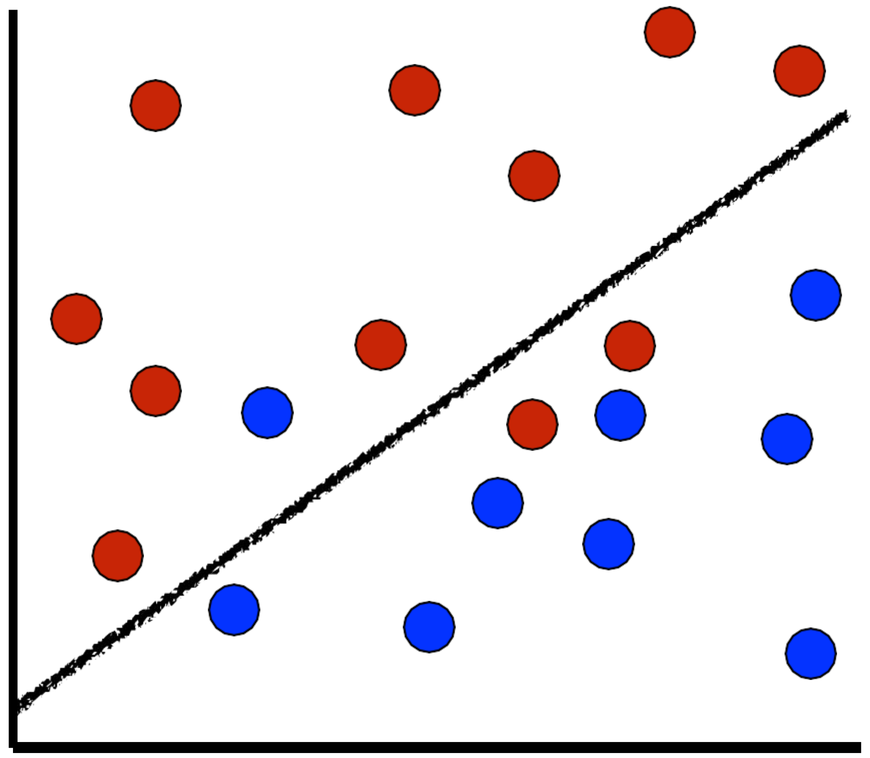
Overfitting and generalization

Fitting training data does not guarantee generalization

Left: perfectly fits training samples, but it is complex / may be overfitting.

Right: misclassifies a few points, but is simple / generalizes





More complex hypothesis class → greater risk of overfitting

Computing closed form solution

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Computation: $O(nk^2 + k^3)$ operations

Consider number of arithmetic operations (+, −, ×, /)

Computational bottlenecks:

- Matrix multiply of $\mathbf{X}^\top \mathbf{X}$: $O(nk^2)$ operations
- Matrix inverse: $O(k^3)$ operations

Storage requirements

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Computation: $O(nk^2 + k^3)$ operations

Storage: $O(nk + k^2)$ floats

Consider storing values as floats (8 bytes)

Storage bottlenecks:

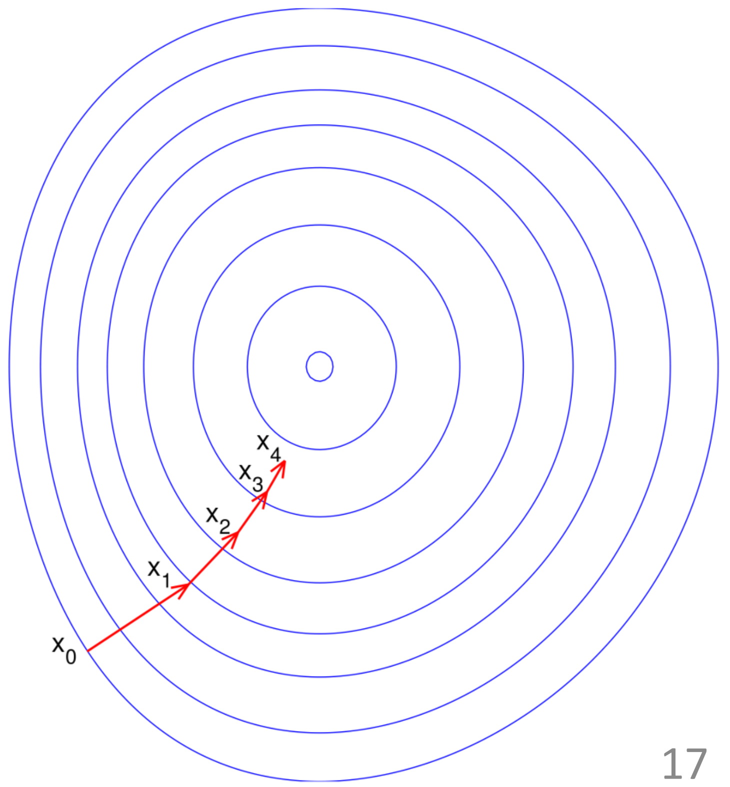
- $\mathbf{X}^\top \mathbf{X}$ and its inverse: $O(k^2)$ floats
- $\mathbf{X}$: $O(nk)$ floats

Large *n* and large *k*

We need methods that are linear in time and space

One idea:

- Gradient descent is an iterative algorithm that requires $O(nk)$ computation and $O(k)$ local storage per iteration



Outline

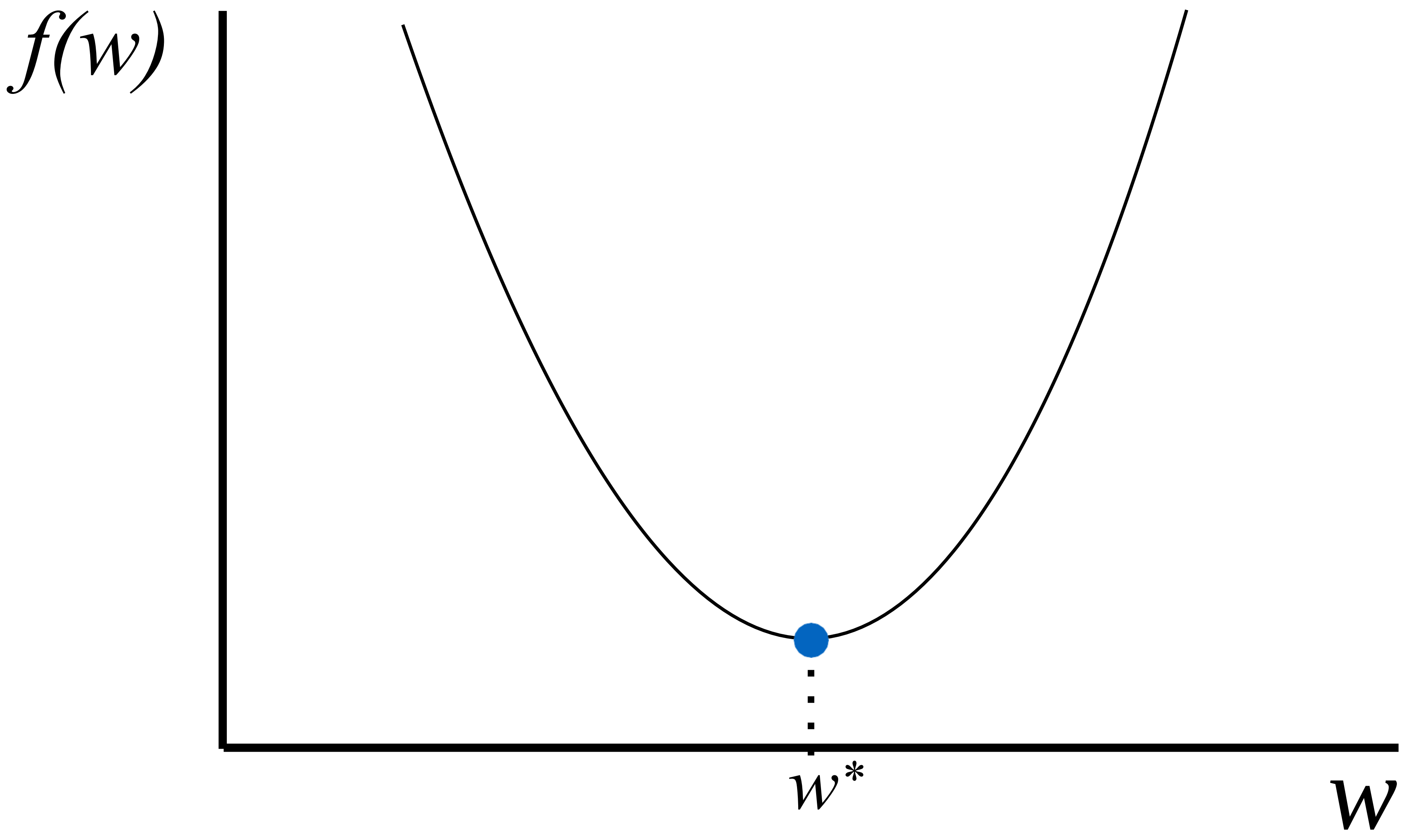
- Linear Regression
- Gradient Descent (GD), SGD, and Minibatch SGD
- Logistic Regression (Classification)

Linear Regression Optimization

Goal: Find w^* that minimizes

$$f(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

- Closed form solution exists
- Gradient Descent is iterative
  (Intuition: go downhill!)



Scalar objective: $f(w) = \|w\mathrm{x} - \mathrm{y}\|_2^2 = \sum_{j=1}^{n} (wx^{(j)} - y^{(j)})^2$

Gradient descent

Start at a random point

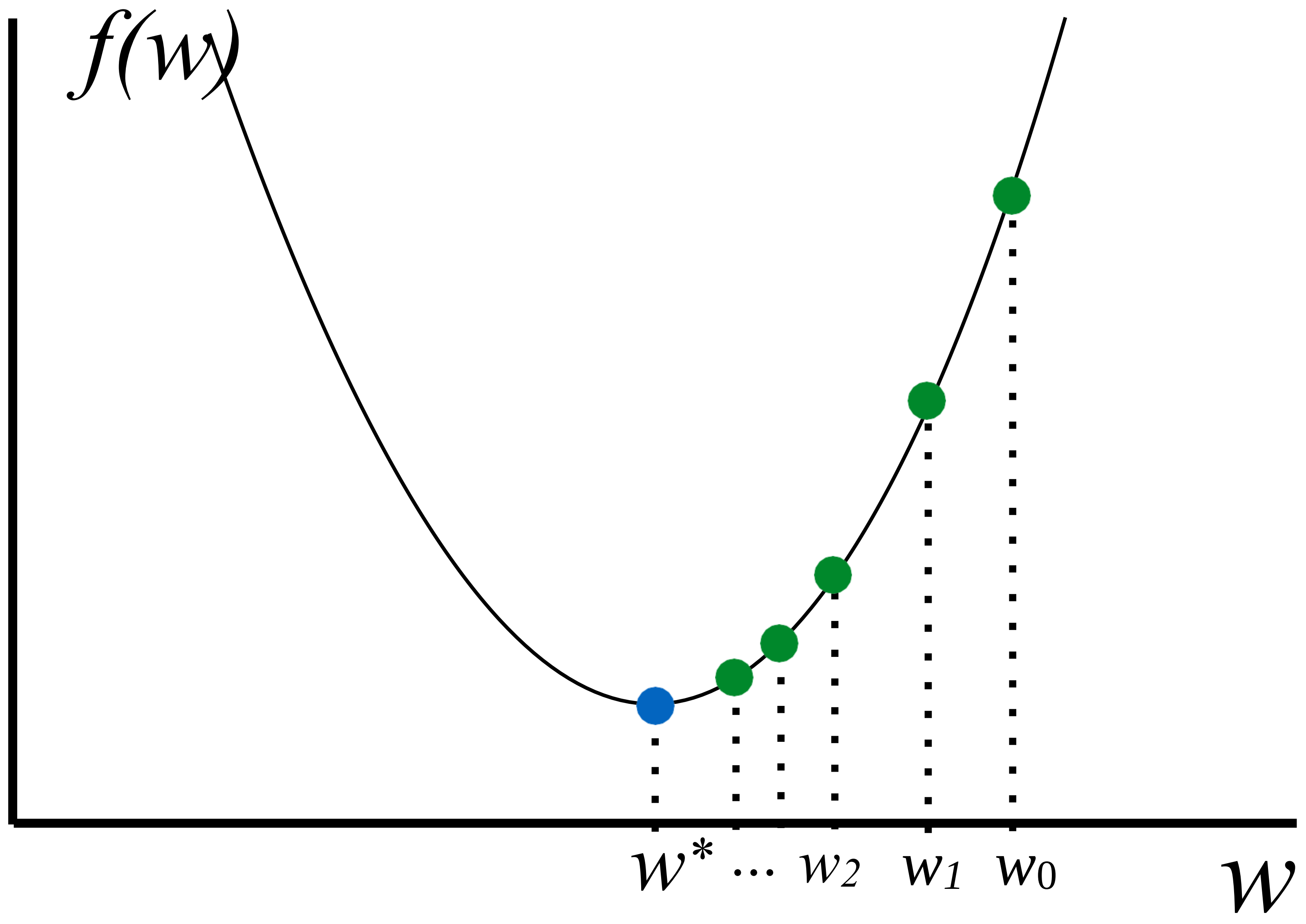
Repeat

Determine a descent direction

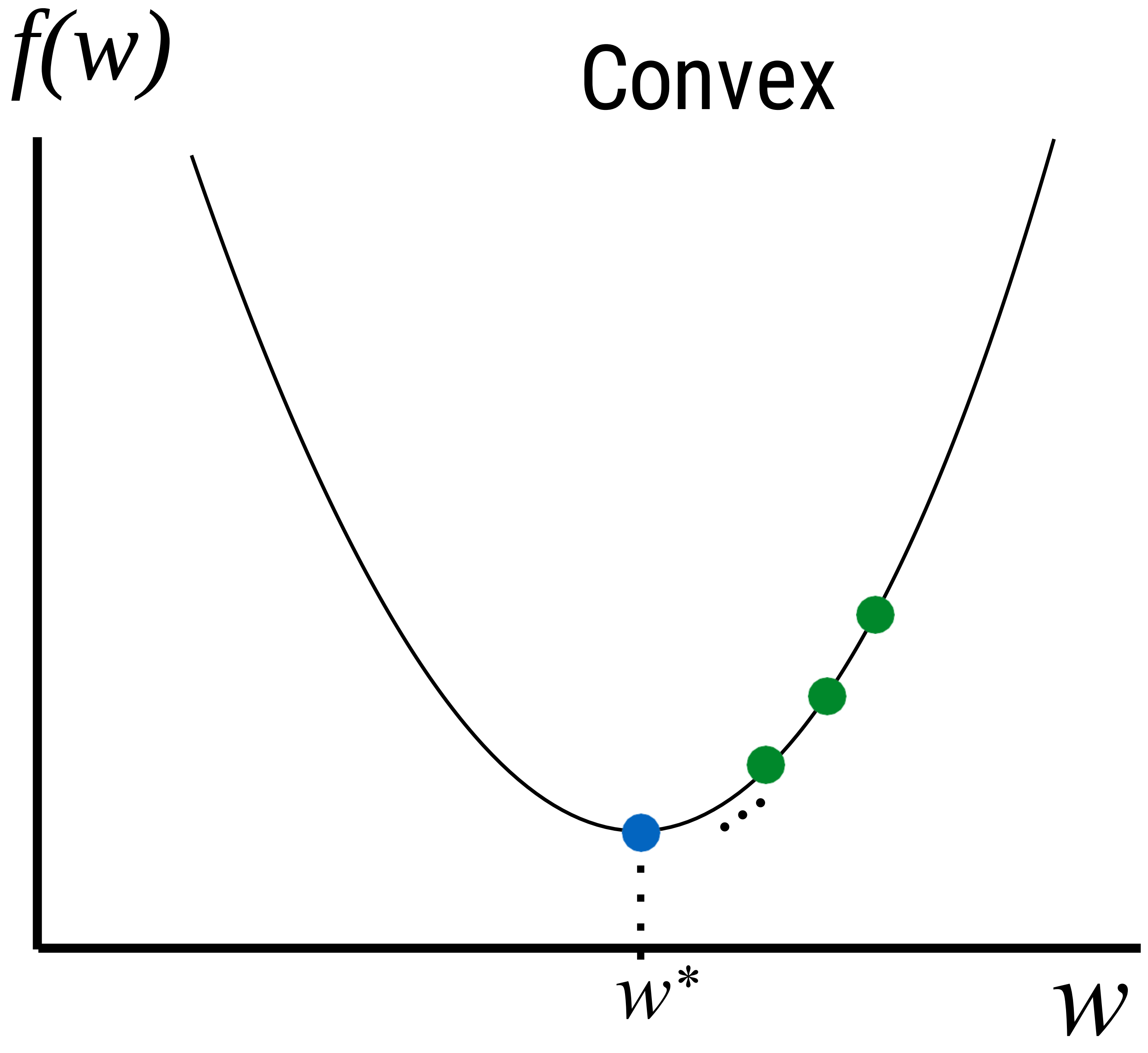
Choose a step size

Update

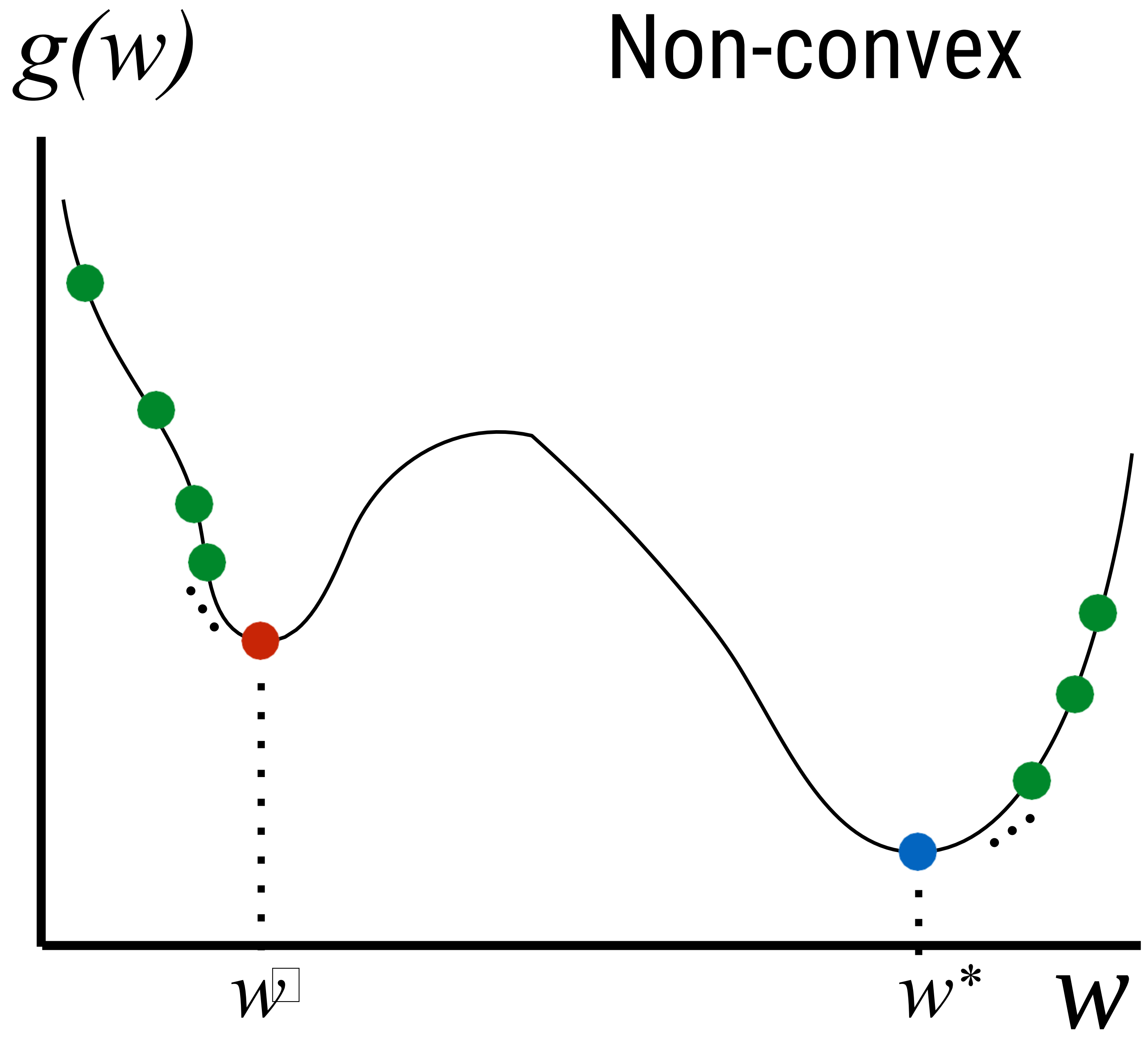
Until stopping criterion is satisfied



Where will we converge?



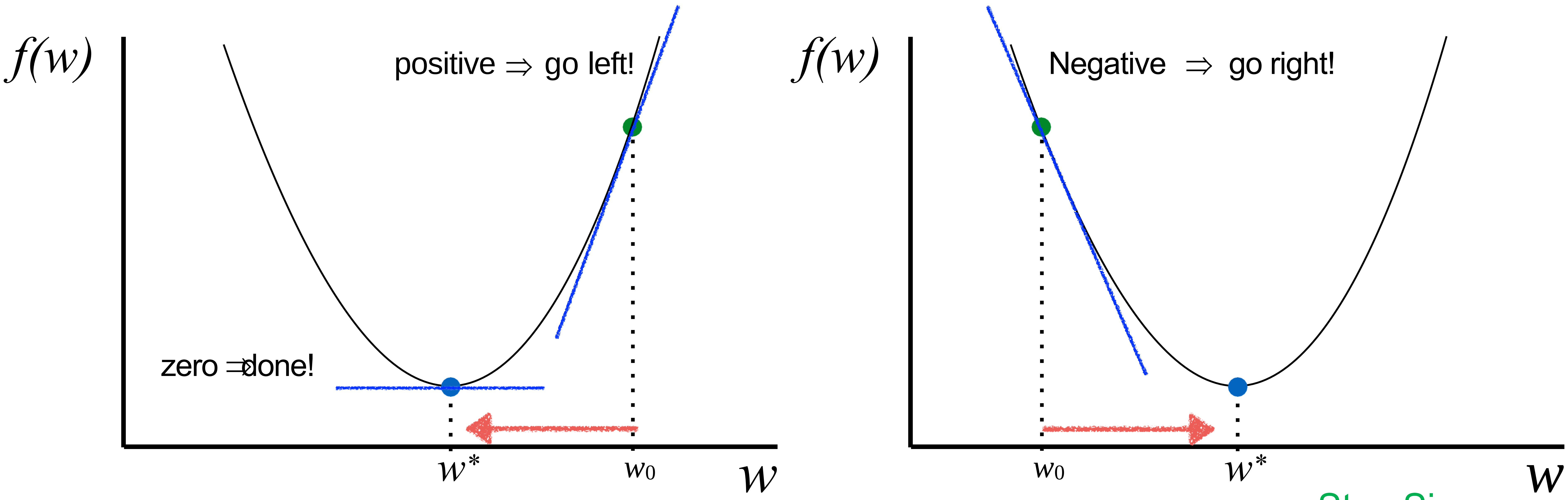
Any local minimum is a global minimum



Multiple local minima may exist

Least Squares, Ridge Regression, and Logistic Regression are all convex!

Choosing a descent direction (1D)



We can only move in two directions

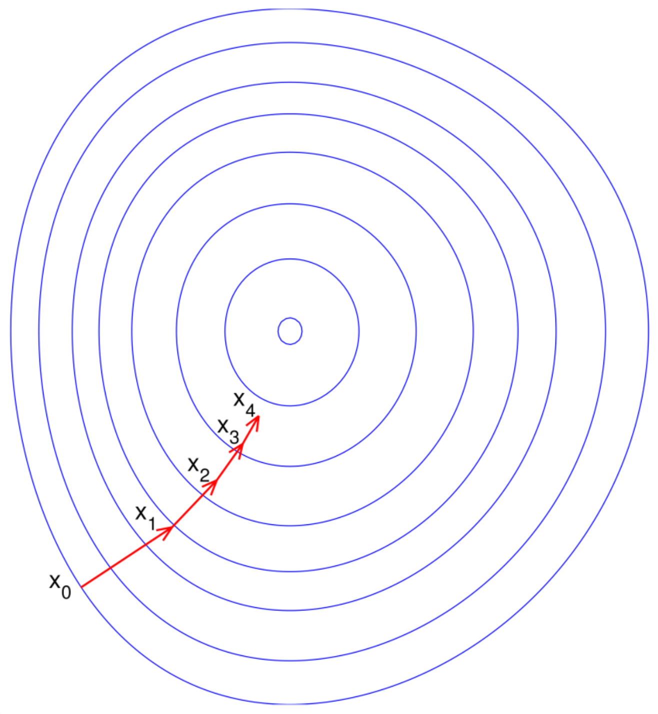
Negative slope is direction of descent!

Step Size

Update Rule: $$w_{i+1} = w_i - \alpha_i \frac{df}{dw}(w_i)$$

Negative Slope

Gradient descent for least squares



Update Rule: $w_{i+1} = w_i - \alpha_i \dfrac{df}{dw}(w_i)$

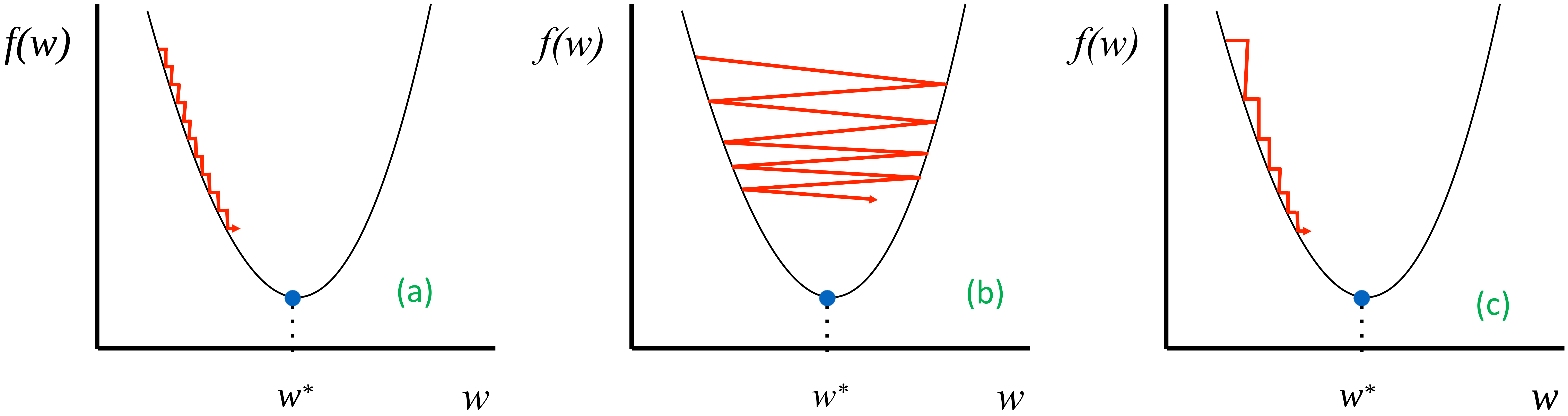
Scalar objective: $f(w) = \|w\mathrm{x} - \mathrm{y}\|_2^2 = \sum_{j=1}^{n} (wx^{(j)} - y^{(j)})^2$

Derivative: (chain rule) $\dfrac{df}{dw}(w) = 2\sum_{j=1}^{n} (wx^{(j)} - y^{(j)})x^{(j)}$

Scalar Update: (2 absorbed in α) $w_{i+1} = w_i - \alpha \sum_{j=1}^{n} (w_i x^{(j)} - y^{(j)})x^{(j)}$

Vector Update: $\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \sum_{j=1}^{n} (\mathrm{w}_i^{\mathrm{T}} \mathrm{x}^{(j)} - y^{(j)})\mathrm{x}^{(j)}$

Choosing a step size



Too small: converge very slowly

Too big: overshoot, can diverge

Reduce size over time

Theoretical convergence results for various step sizes

A common step size is

$$\alpha_i = \frac{\alpha}{n\sqrt{i}}$$

Constant

Iteration #

Training Points

Stochastic gradient descent

Gradient Descent Update: $$\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \sum_{j=1}^{n} \left(\mathrm{w}_i^{\mathrm{T}} \mathrm{x}^{(j)} - y^{(j)} \right) \mathrm{x}^{(j)}$$ O(nk) computation complexity

[FOR LINEAR REGRESSION]

How can we reduce computation?

*Idea: approximate full gradient with **just one observation***

Stochastic Gradient Descent (SGD) Update: $$\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \left(\mathrm{w}_i^{\mathrm{T}} \mathrm{x}^{(j)} - y^{(j)} \right) \mathrm{x}^{(j)}$$ O(k) computation complexity

the sum is gone!

Stochastic gradient descent

MORE GENERALLY

Objective we want to solve: $\min_{\mathrm{w}} f(\mathrm{w})$ where $f(\mathrm{w}) := \sum_{j=1}^{n} f_j(\mathrm{w})$

Gradient Descent Update: $\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \nabla f(\mathrm{w}_i)$

Stochastic Gradient Descent (SGD) Update: $\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \nabla f_j(\mathrm{w}_i)$

with j sampled at random

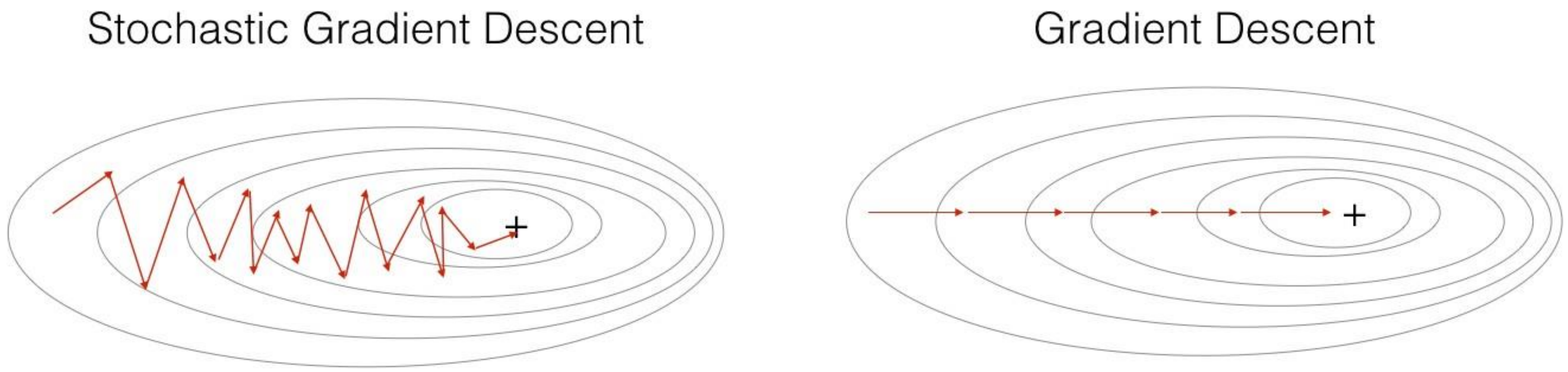
Stochastic gradient descent

What could go wrong?

- Gradient with respect to one random example might point in the *wrong direction*

Why would we expect it to work?

- On average, gradient of random examples will approximate the full gradient
- Method should converge in expectation



Stochastic gradient descent

Pros

- Less computation
- *n* times cheaper than gradient descent at each iteration
- This faster per-iteration cost might lead to faster overall convergence

Cons

- Less stable convergence than gradient descent
- In terms of iterations: slower convergence than gradient descent
- ***More iterations!***

Mini-batch SGD

Computation Complexity

Gradient Descent Update:

[FOR LINEAR REGRESSION]

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \sum_{j=1}^{n} \left(\mathbf{w}_i^{\mathrm{T}} \mathbf{x}^{(j)} - y^{(j)}\right) \mathbf{x}^{(j)}$$

$O(nk)$

Stochastic Gradient Descent (SGD) Update:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \left(\mathbf{w}_i^{\mathrm{T}} \mathbf{x}^{(j)} - y^{(j)}\right) \mathbf{x}^{(j)}$$

use a single observation

$O(k)$

Mini-batch SGD/GD Update:

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \sum_{j \in B_i}^{n} \left(\mathbf{w}_i^{\mathrm{T}} \mathbf{x}^{(j)} - y^{(j)}\right) \mathbf{x}^{(j)}$$

$O(bk)$

sum over a *mini-batch* of observations

MORE GENERALLY

Mini-batch SGD

Objective we want to solve: $\min_{\mathrm{w}} f(\mathrm{w})$ where $f(\mathrm{w}) := \sum_{j=1}^{n} f_j(\mathrm{w})$

Gradient Descent Update: $\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \nabla f(\mathrm{w}_i)$

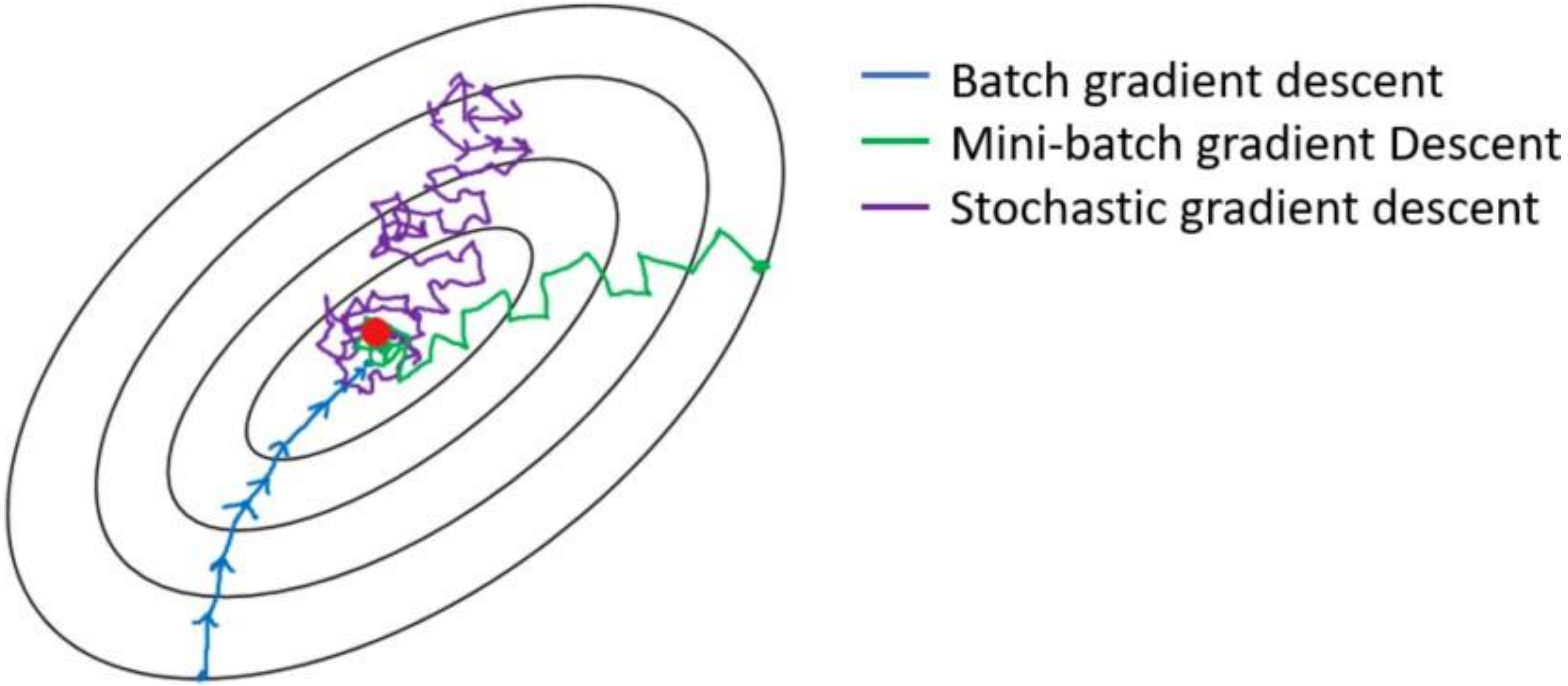
Stochastic Gradient Descent (SGD) Update: $\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \nabla f_j(\mathrm{w}_i)$

with j sampled at random

Mini-batch SGD Update: $\mathrm{w}_{i+1} = \mathrm{w}_i - \alpha_i \nabla f_{B_i}(\mathrm{w}_i)$

with mini-batch $\mathbf{B}_i \subseteq \{1, \ldots n\}$ *sampled at random*

Mini-batch SGD



https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Mini-batch SGD

Pros

- More computation than SGD
- Less computation than GD (might lead to faster overall convergence)
- Can tune computation vs. communication depending on batch size

Cons

- In terms of iterations: slower convergence than gradient descent
- Another parameter to tune (batch size)
- Still might be too much communication …

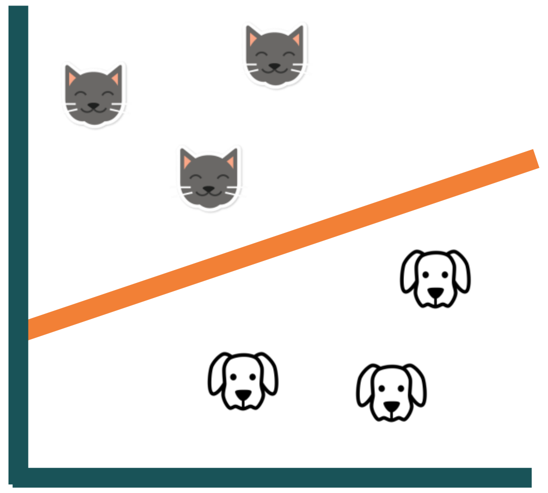
Outline

- Linear Regression
- Gradient Descent (GD), SGD, and Minibatch SGD
- Classification

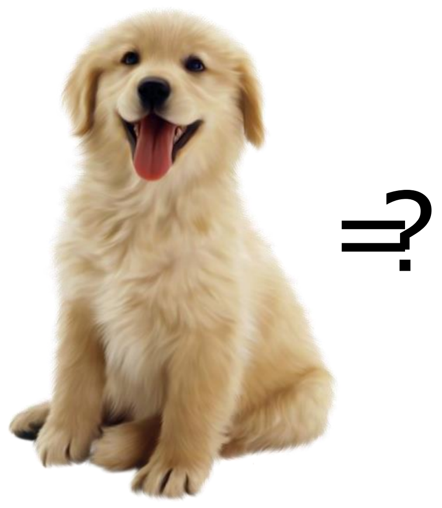
Classification

Cat or dog?

input: cats & dogs



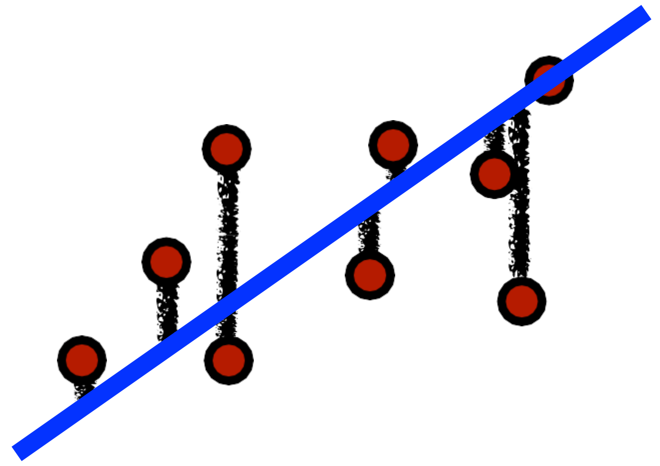
learn: x → y relationship



predict: y (categorical)

RECALL

Linear least squares regression



Example: Predicting house price from size, location, age

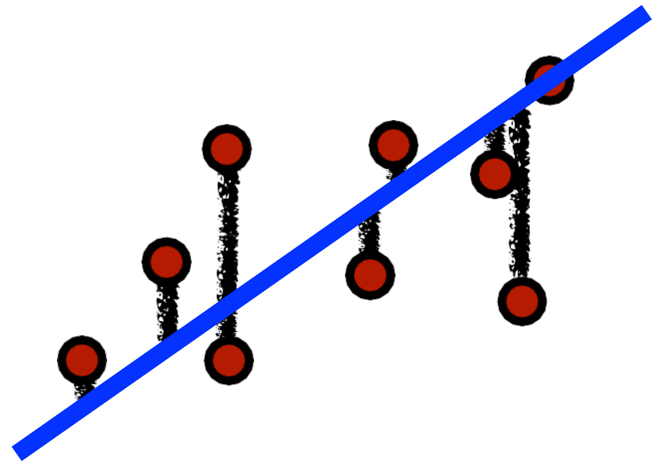
For each observation we have a feature vector, **x**, and label, y

$$\mathbf{x}^T = [\, x_1 \;\; x_2 \;\; x_3 \,]$$

We assume a *linear* mapping between features and label:

$$y \approx w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

Linear least squares regression



Example: Predicting house price from size, location, age

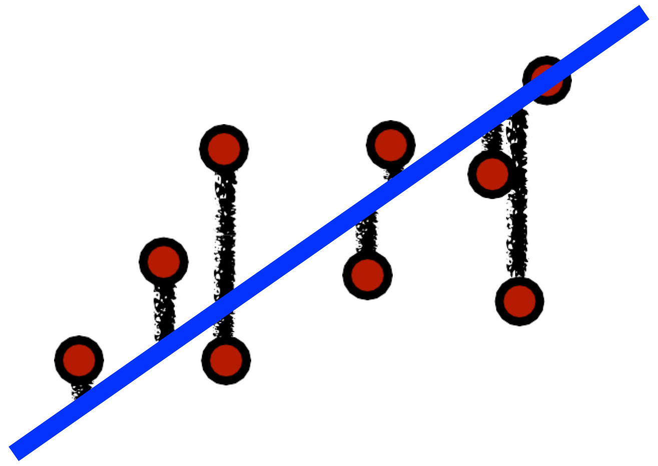
We can augment the feature vector to incorporate offset:

$$\mathbf{x}^T = [1 \quad x_1 \quad x_2 \quad x_3]$$

We can then rewrite this linear mapping as scalar product:

$$y \approx \hat{y} = \sum_{i=0}^{3} w_i x_i = \mathbf{w}^\top \mathbf{x}$$

Why a linear mapping?



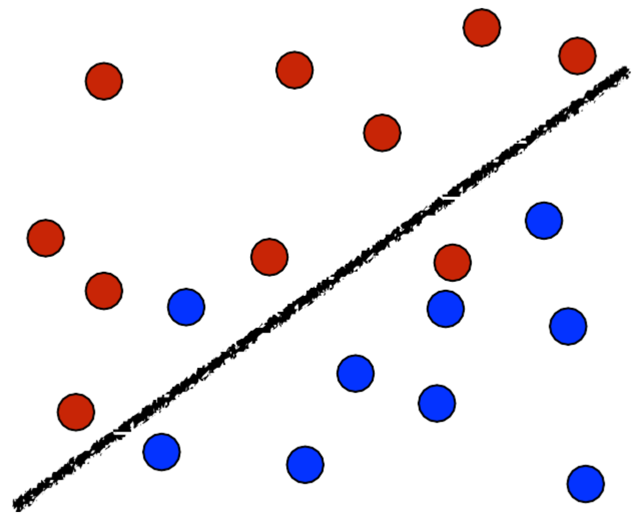
Simple

Often works well in practice

Can introduce complexity via feature extraction

Can we do something similar for classification?

Linear regression ⇒ linear classification



Example: Predicting rain from temperature, cloudiness, and humidity

Use the same feature representation: $\mathbf{x}^T = [1 \quad x_1 \quad x_2 \quad x_3]$

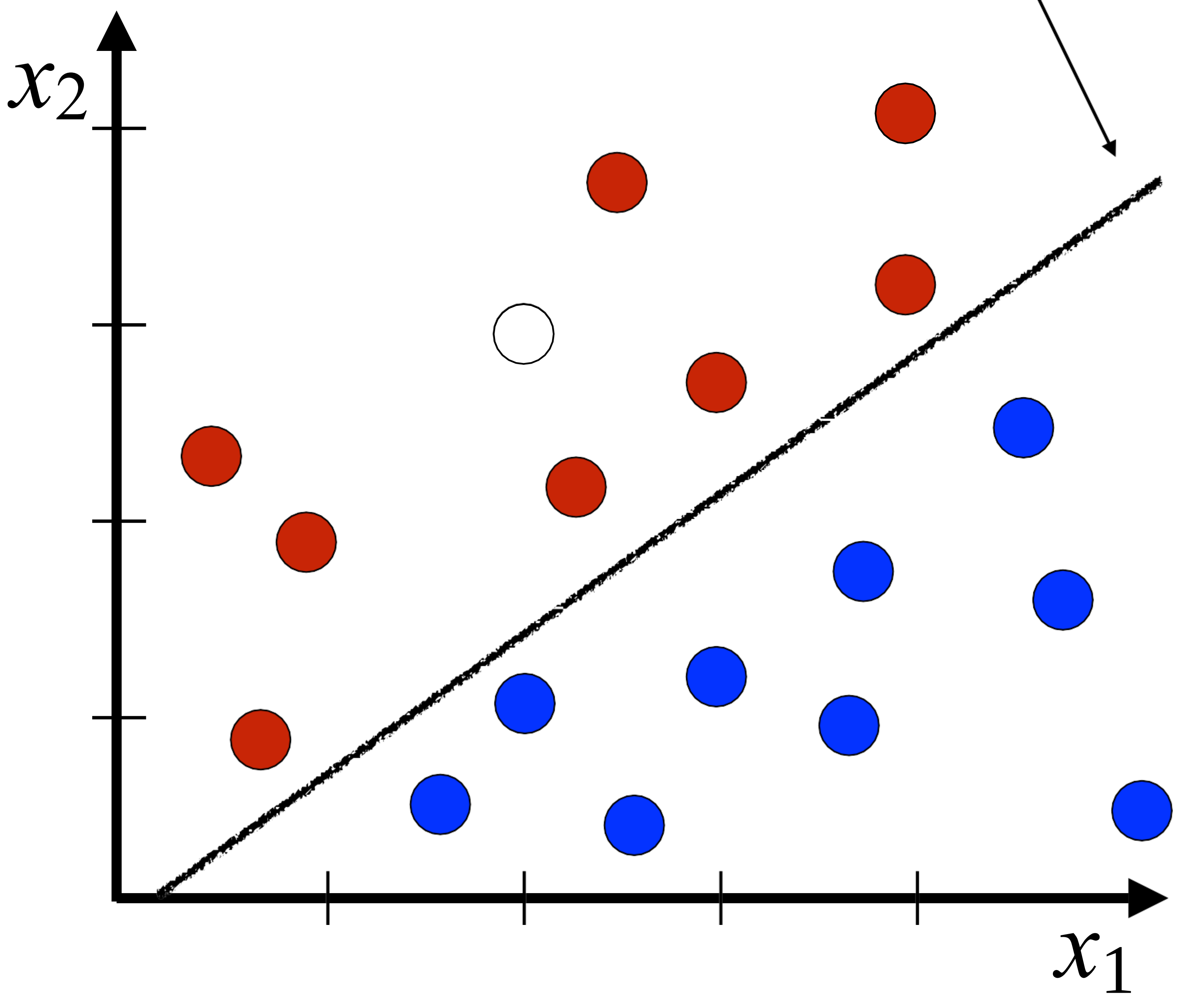
How can we make class predictions?

- {not-rain, rain}, {not-spam, spam}, {not-click, click}
- **Idea**: threshold by sign

$$\hat{y} = \sum_{i=0}^{3} w_i x_i = \mathrm{w}^T \mathrm{x} \Rightarrow \hat{y} = sign(\mathrm{w}^T \mathrm{x})$$

Linear classifier decision boundary

Decision Boundary $3x_1 - 4x_2 - 1 = 0$



Example: $w^T = [-1 \quad 3 \; -4]$

$x^T = [1 \quad 2 \quad 3]$

$x^T = [1 \quad 2 \quad 1] : \quad w^T x = 1$

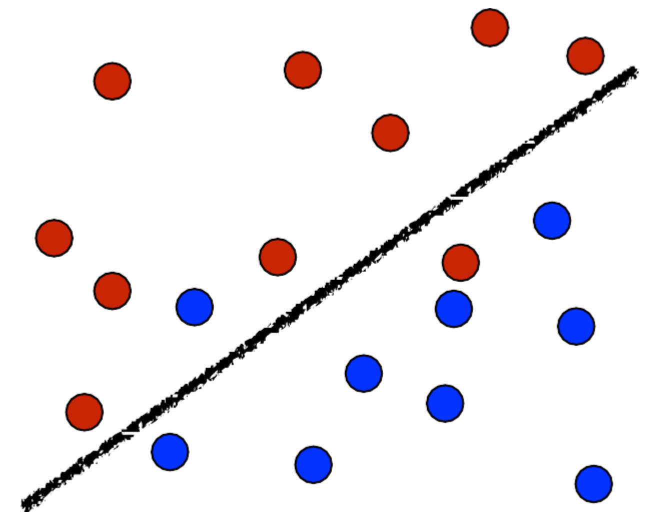
$x^T = [1 \quad 5 \quad .5] : \quad w^T x = 12$

$x^T = [1 \quad 3 \quad 2.5] : w^T x = -2$

Let's interpret this rule: $\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x})$

- $\hat{y} = 1 : \mathbf{w}^\top \mathbf{x} > 0$
- $\hat{y} = -1 : \mathbf{w}^\top \mathbf{x} < 0$
- Decision boundary: $\mathbf{w}^\top \mathbf{x} = 0$

Evaluating predictions



Regression: can measure 'closeness' between label and prediction

- House price prediction: better to be off by 10K than by 1or 0
- Squared loss: $(y-\hat{y})^2$

Classification: Class predictions are discrete

- 0-1 loss: Penalty is 0 for correct prediction, and 1 otherwise

Logistic regression with probabilistic interpretation

Probabilistic interpretation

Goal: Model conditional probability: $P[y = 1 \mid x]$

First thought: $P[y = 1 \mid \mathbf{x}] = w^T x$?

- Linear regression returns any real number, but probabilities are in range [0,1]

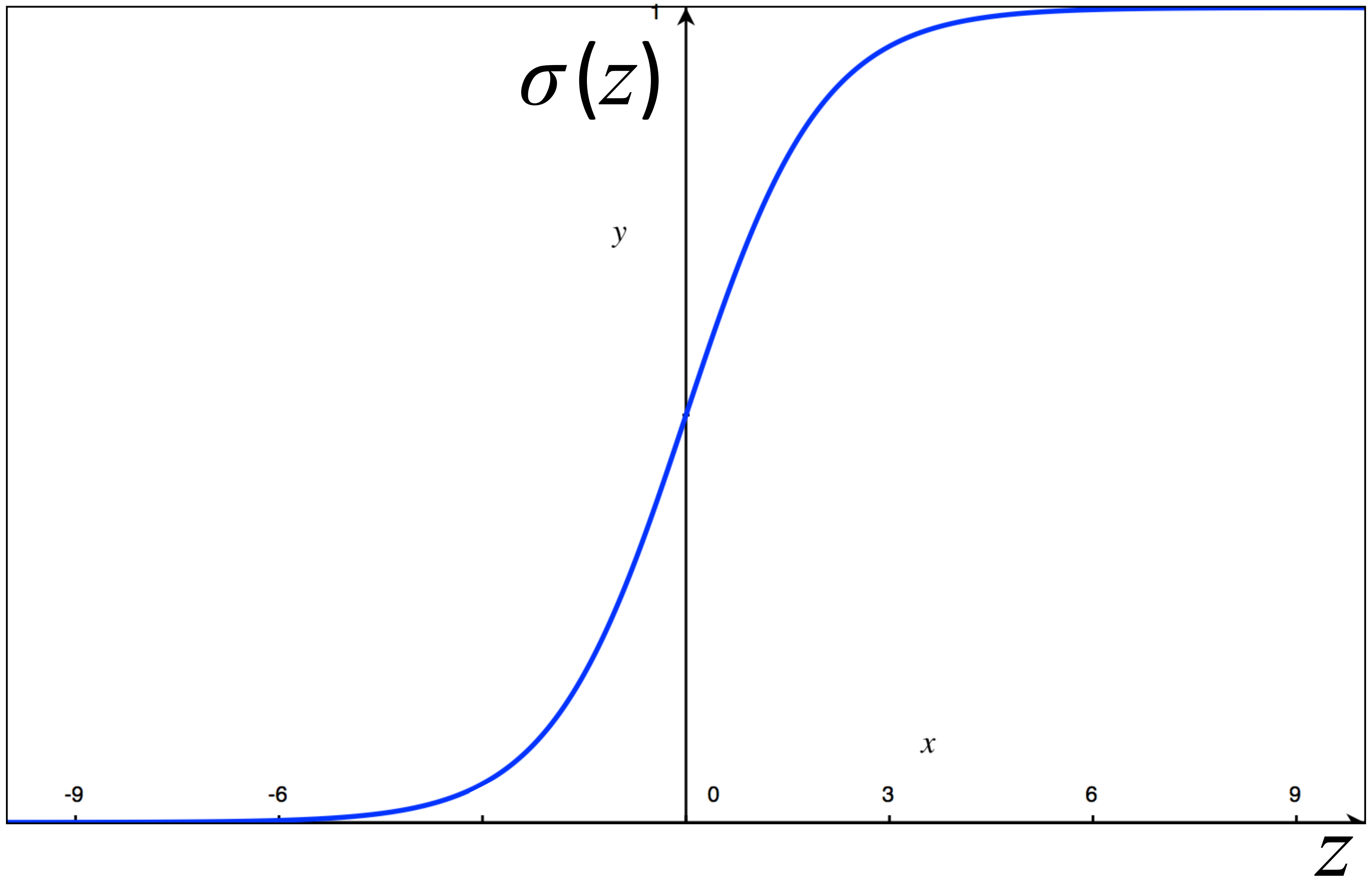
How can we transform its output?

- Use logistic (or sigmoid) function: $P[y = 1 \mid x] = \sigma(w^T x)$

Logistic function

Maps real numbers to $[0,1]$

- Large positive inputs $\Rightarrow 1$
- Large negative inputs $\Rightarrow 0$



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

Probabilistic interpretation

Goal: Model conditional probability: $P[\,y = 1 \mid x]$

Logistic regression uses logistic function to model this conditional probability

- $P[\,y = 1 \mid x\,] = \sigma(w^T x)$
- $P[\,y = 0 \mid x\,] = 1 - \sigma(w^T x)$

For notational convenience we now define $y \in \{0,1\}$

How do we use probabilities?

To make class predictions, we need to convert probabilities to values in $[0,1]$

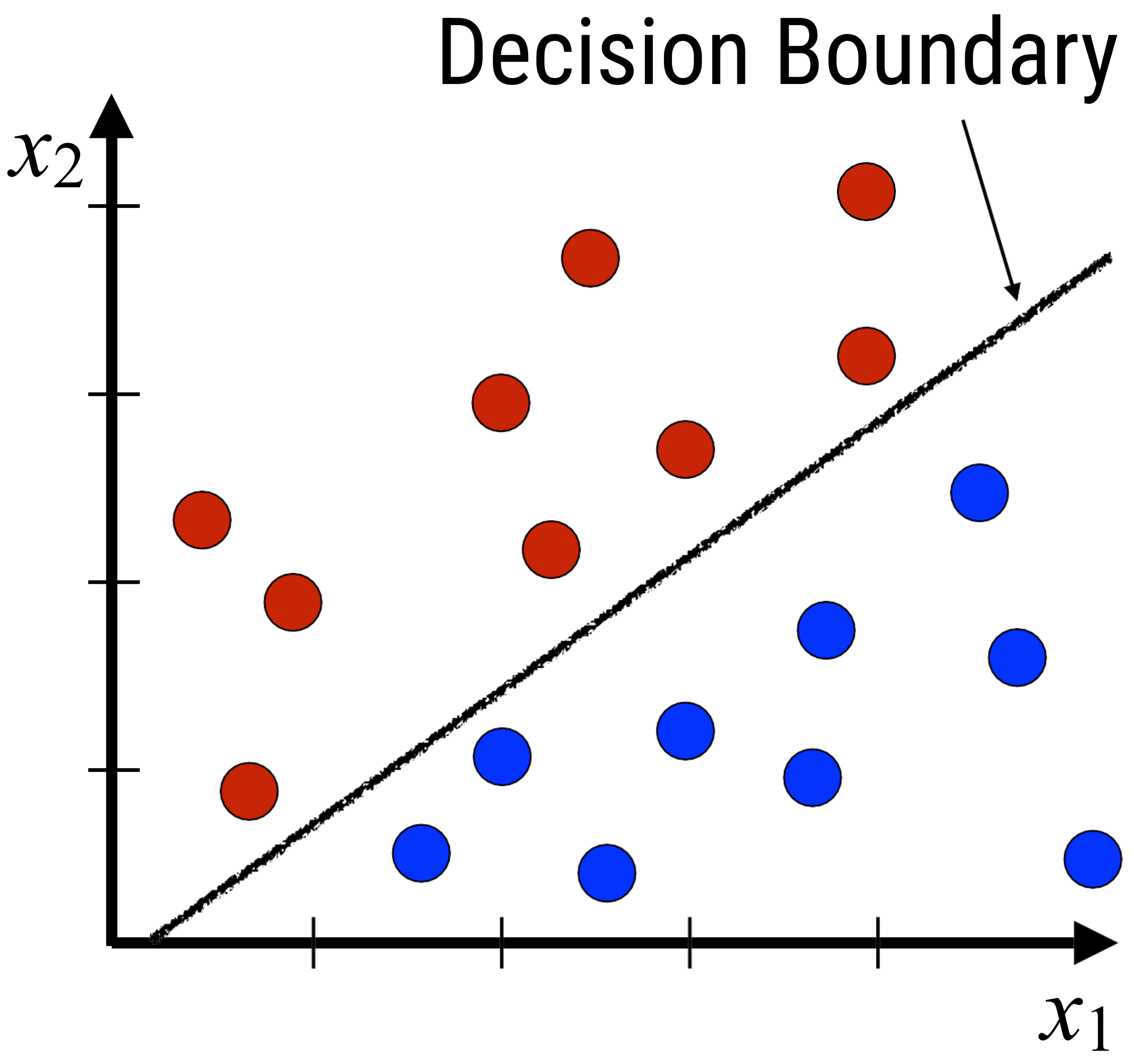
We can do this by setting a threshold on the probabilities

- Default threshold is 0.5
- $P[y = 1 \mid x] > 0.5 \Rightarrow \hat{y} = 1$

Example: Predict rain from temperature, cloudiness, humidity

- $P[y = \text{rain} \mid t = 14°\text{F},\ c = \text{LOW},\ h = 2\%] = .05 \quad \Rightarrow \quad \hat{y} = 0$
- $P[y = \text{rain} \mid t = 70°\text{F},\ c = \text{HIGH},\ h = 95\%] = .9 \quad \Rightarrow \quad \hat{y} = 1$

Connection with decision boundary?



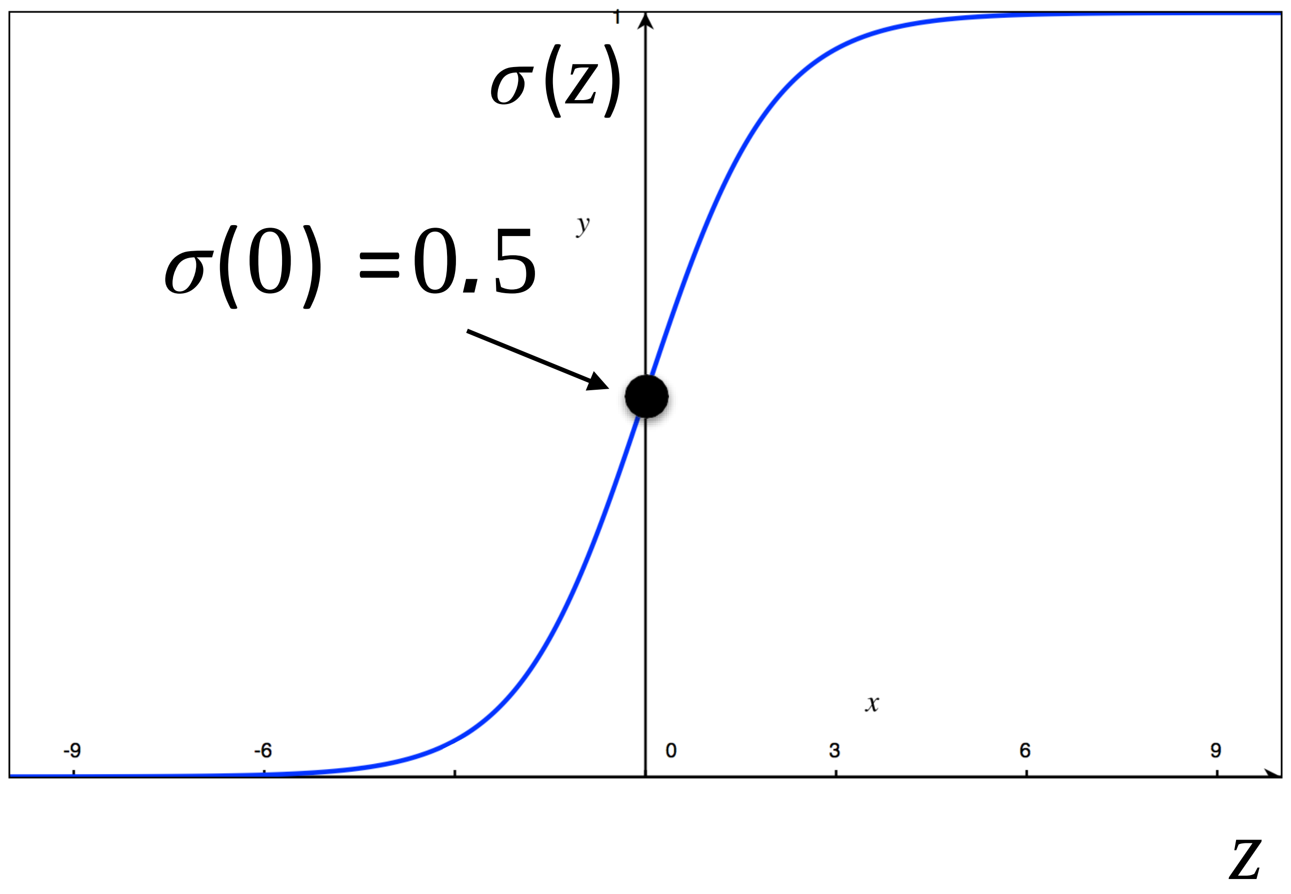
Threshold by sign to make class predictions: $\hat{y} = \mathrm{sign}(w^T x)$

- $\hat{y} = 1 : w^T x > 0$
- $\hat{y} = 0 : w^T x < 0$
- decision boundary: $w^T x = 0$

How does this compare with thresholding probability?

- $P[\,y = 1 \mid x\,] = \sigma(w^T x) > 0.5 \quad \Rightarrow \quad \hat{y} = 1$

Connection with decision boundary?



$w^T x = 0 \iff \sigma(w^T x) = 0.5$

Threshold by sign to make class predictions: $\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$

- $\hat{y} = 1$: $\mathbf{w}^T \mathbf{x} > 0$
- $\hat{y} = 0$: $\mathbf{w}^T \mathbf{x} < 0$
- decision boundary: $\mathbf{w}^T \mathbf{x} = 0$

How does this compare with thresholding probability?

- $P[y = 1 \mid x] = \sigma(w^T x) > 0.5 \Rightarrow \hat{y} = 1$
- With threshold of 0.5, the decision boundaries are identical!

Working directly with probabilities

Example: Predict click from ad's **h**istorical performance, user's click **f**requency, and publisher page's **r**elevance

- $\mathrm{P}[y = \text{click} \mid \text{h} = \text{GOOD},\ \text{f} = \text{HIGH},\ \text{r} = \text{HIGH}] = .6 \qquad \Rightarrow \hat{y} = 1$
- $\mathrm{P}[y = \text{click} \mid \text{h} = \text{BAD},\ \text{f} = \text{LOW},\ \text{r} = \text{LOW}] = .001 \qquad \Rightarrow \hat{y} = 0$

Probabilities provide more granular information

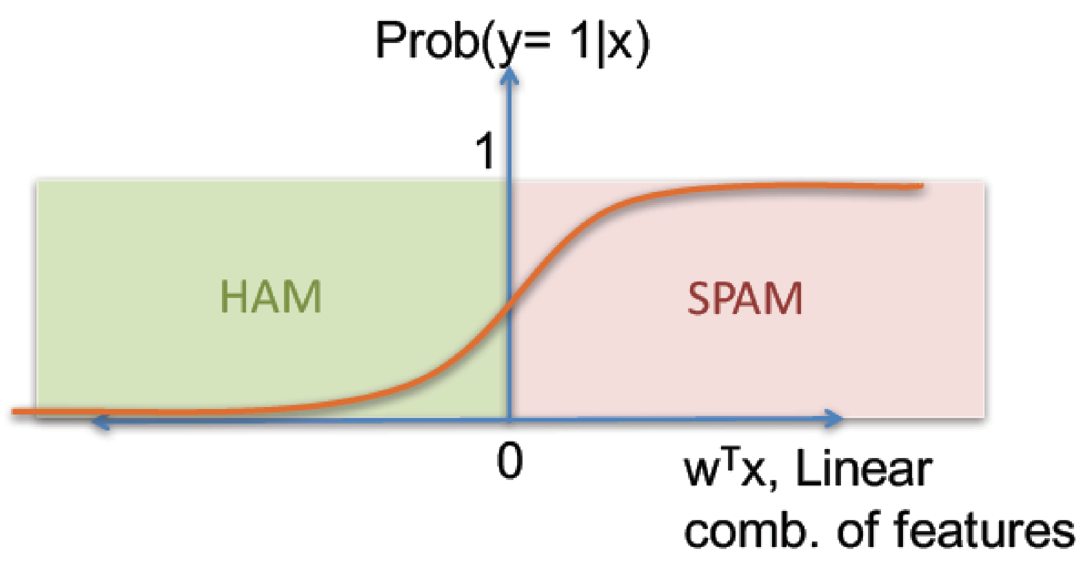
- Confidence of prediction
- Useful when combining predictions with other information

In such cases, we want to evaluate probabilities directly

- Logistic loss makes sense for evaluation!

Example: Spam classification

- **Class label**: binary
  - $y = \{$ spam, ham $\}$
- **Features**: word counts in the document (bag-of-words)
  - $\mathbf{x} = \{($'free', 100), ('lottery', 5), ('money', 10)$\}$
  - Each pair is in the format of $(w_i, \#w_i)$, namely, a unique word in the dictionary, and the number of times it shows up



Probability that predicted label is 1 (spam)

Key Problem: Finding optimal weights w that accurately predict this probability for a new email

Example: Spam classification

Suppose you see the following email:

```
CONGRATULATIONS!! Your email address have won you the
lottery sum of US$2,500,000.00 USD to claim your prize,
contact your office agent (Athur walter) via email
claims2155@yahoo.com.hk or call +44 704 575 1113
```

Keywords are [lottery, prize, office, email]
The given weight vector is $\mathbf{w} = [0.3, 0.3, -0.1, -0.04]^\top$

What is the probability that the email is spam?

$$\mathbf{x} = [1, 1, 1, 2]^\top$$

$$\mathbf{w}^\top \mathbf{x} = 0.3 * 1 + 0.3 * 1 - 0.1 * 1 - 0.04 * 2 = 0.42 > 0$$

$$\Pr(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-0.42}} = 0.603$$

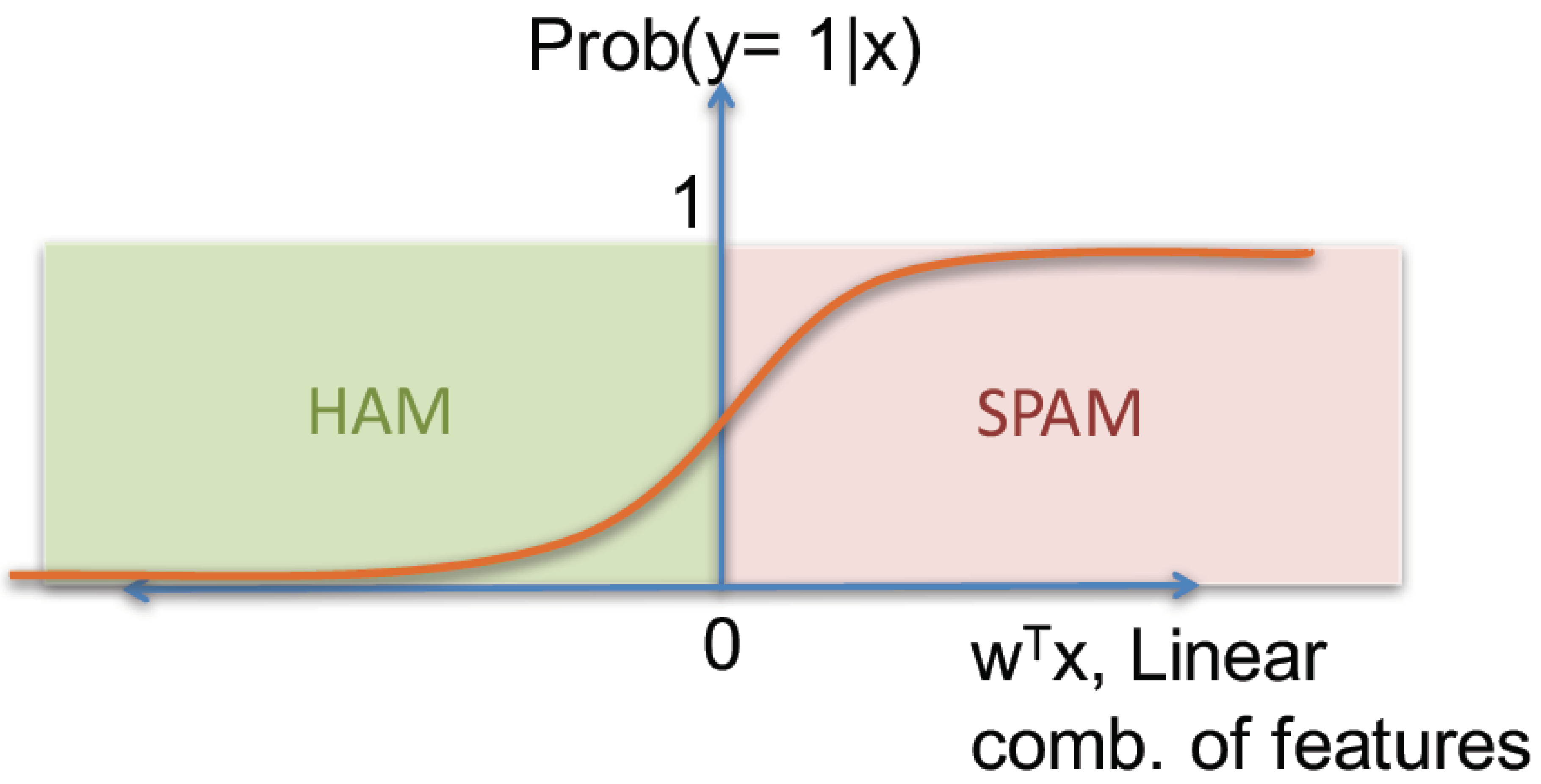
Likelihood function

Probability of a single training sample $(\boldsymbol{x}_n, y_n)$

$$p(y_n|\boldsymbol{x}_n; \boldsymbol{w}) = \begin{cases} \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n) & \text{if} \quad y_n = 1 \\ 1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n) & \text{otherwise} \end{cases}$$

Compact expression, exploring that y_n is either 1 or 0

$$p(y_n|\boldsymbol{x}_n; \boldsymbol{w}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)^{y_n}[1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n)]^{1-y_n}$$



Log likelihood or cross-entropy error

Log-likelihood of the whole training data $\mathcal{D}$

$$\log P(\mathcal{D}) = \sum_{i=1}^{n} \{ y_i \log \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) + (1 - y_i) \log[1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)] \}$$

It is convenient to work with its negation, which is called *cross-entropy error function*

$$f(\boldsymbol{w}) = -\sum_{i=1}^{n} \{ y_i \log \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) + (1 - y_i) \log[1 - \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)] \}$$

$$\nabla f(\boldsymbol{w}) = \sum_{i=1}^{n} \{ \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) - y_i \} \boldsymbol{x}_i$$

It's Gradient

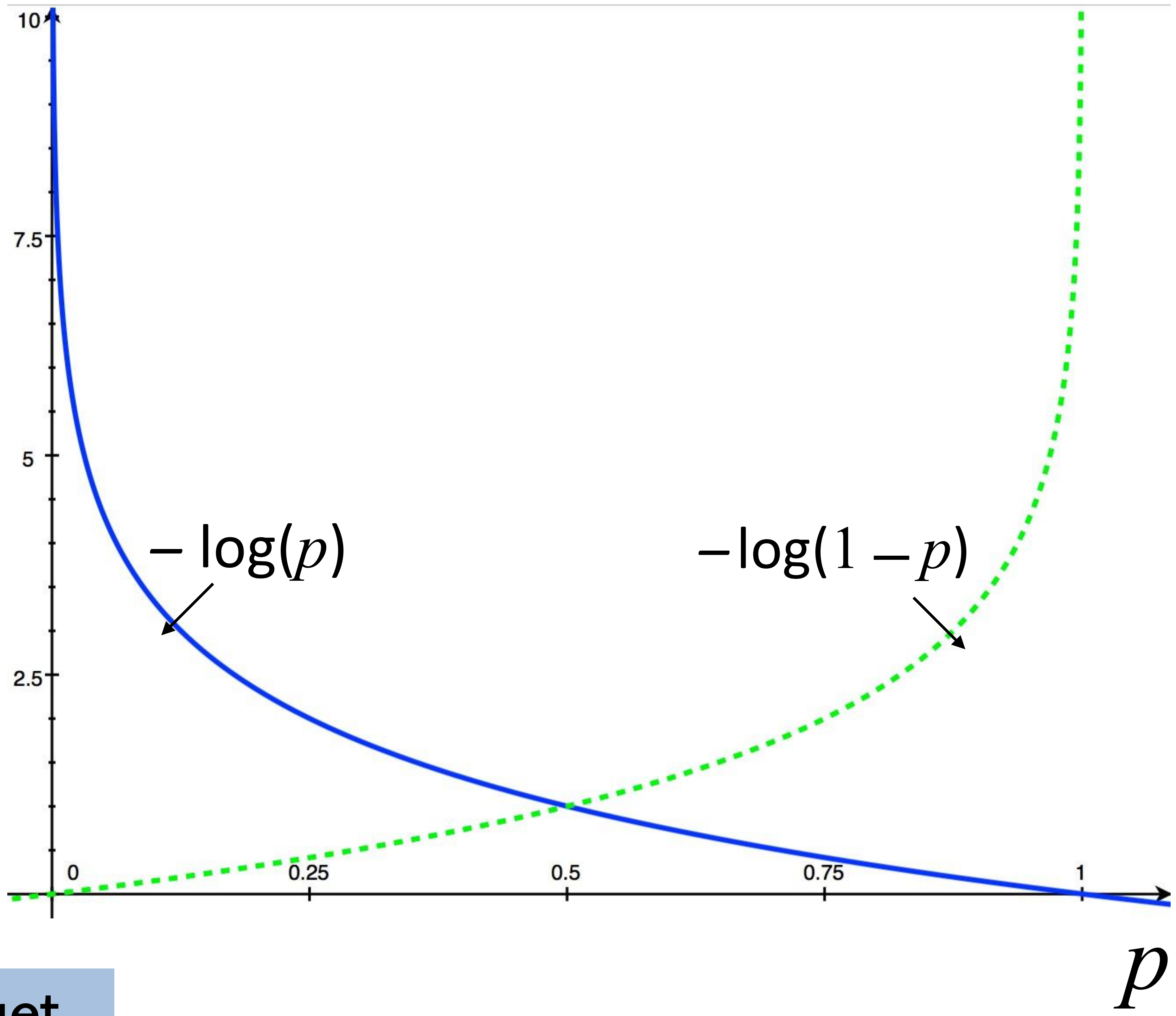
Log-likelihood loss

$$\ell_{log}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0 \end{cases}$$

When $y = 1$

- At p=1, cost = 0 when prediction is correct)
- Increasing penalty when p away from 1

Similar when $y = 0$



Captures intuition that that larger mistakes should get larger penalties

Multinomial Logistic Regression

- Model: For each class C_k, we have a parameter vector $\boldsymbol{w}_k$ and model the posterior probability as:

$$p(C_k|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_k^\top \boldsymbol{x}}}{\sum_{k'} e^{\boldsymbol{w}_{k'}^\top \boldsymbol{x}}} \qquad \leftarrow \qquad \textit{This is called softmax function}$$

- Decision boundary: Assign $\boldsymbol{x}$ with the label that is the maximum of posterior:

$$\arg\max_k P(C_k|\boldsymbol{x}) \rightarrow \arg\max_k \boldsymbol{w}_k^\top \boldsymbol{x}.$$

Log likelihood

$$\log P(\mathcal{D}) = \sum_n \log P(y_n|\boldsymbol{x}_n)$$

We will change y_n to $\boldsymbol{y}_n = [y_{n1}\ y_{n2}\ \cdots\ y_{nK}]^\top$, a K-dimensional vector using 1-of-K encoding.

$$y_{nk} = \begin{cases} 1 & \text{if } y_n = k \\ 0 & \text{otherwise} \end{cases}$$

Ex: if $y_n = 2$, then, $\boldsymbol{y}_n = [0\ 1\ 0\ 0\ \cdots\ 0]^\top$.

$$\Rightarrow \sum_n \log P(y_n|\boldsymbol{x}_n) = \sum_n \log \prod_{k=1}^{K} P(C_k|\boldsymbol{x}_n)^{y_{nk}} = \sum_n \sum_k y_{nk} \log P(C_k|\boldsymbol{x}_n)$$

Cross-entropy error function

Definition: negative log likelihood

$$f(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) = -\sum_n \sum_k y_{nk} \log P(C_k|\boldsymbol{x}_n)$$

$$= -\sum_n \sum_k y_{nk} \log \left(\frac{e^{\boldsymbol{w}_k^\top \boldsymbol{x}_n}}{\sum_{k'} e^{\boldsymbol{w}_{k'}^\top \boldsymbol{x}_n}} \right)$$

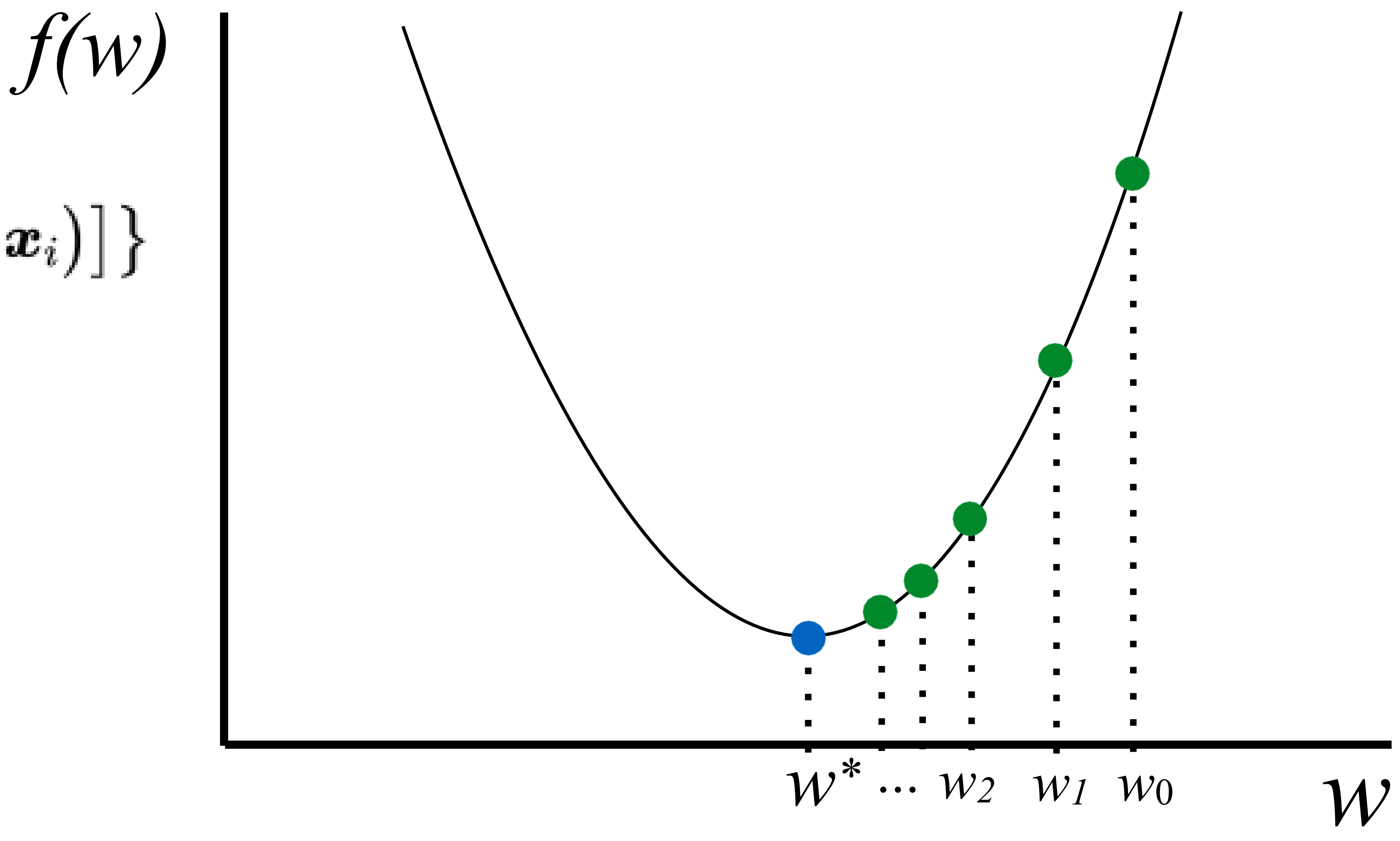
Properties

- Convex, therefore unique global optimum
- Optimization requires numerical procedures, analogous to those used for binary logistic regression

Logistic regression optimization

Goal: Find $\mathbf{w}^*$ that minimizes

$$f(\boldsymbol{w}) = -\sum_{i=1}^{n}\{y_i \log \sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) + (1-y_i)\log[1-\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)]\}$$



- Can solve via Gradient Descent

Step Size

Update Rule: $$\mathrm{w}_{t+1} = \mathrm{w}_t - \alpha_t \sum_{i=1}^{n}\left(\sigma\left(\mathrm{w}_t^{\mathrm{T}}\mathbf{x}^{(i)}\right) - y^{(i)}\right)\mathbf{x}^{(i)}$$

$$\nabla f(\boldsymbol{w}) = \sum_{i=1}^{n}\{\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i) - y_i\}\boldsymbol{x}_i$$

Gradient

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

Q&A