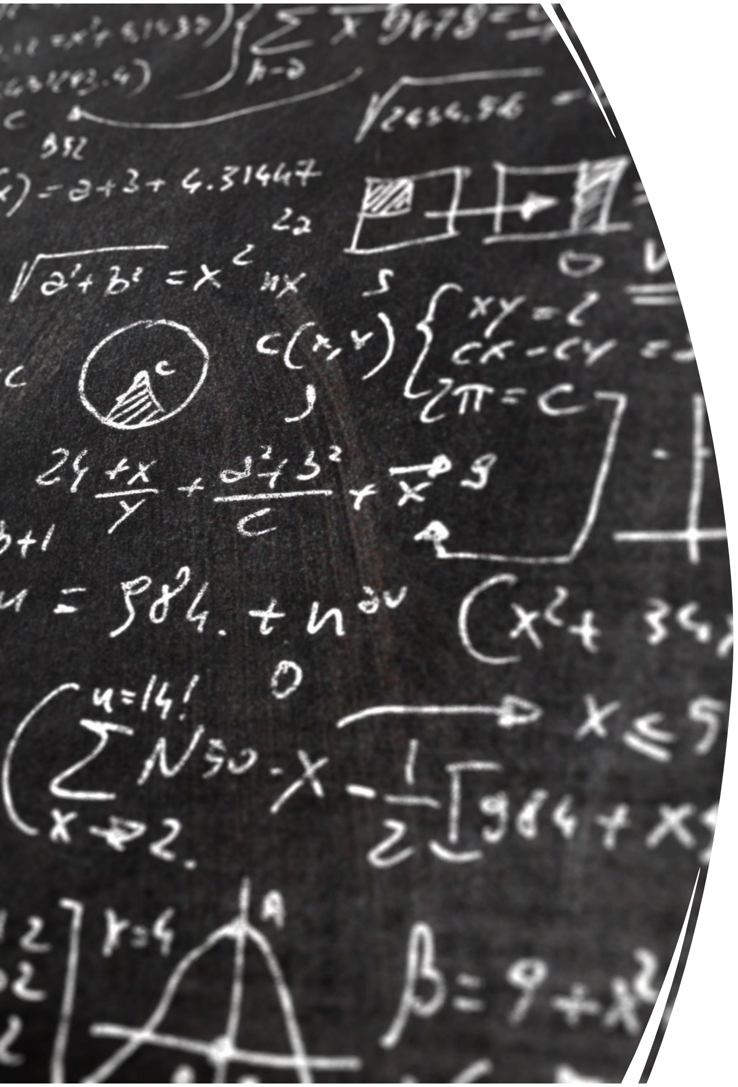


CS 383 – Computational Text Analysis

Lecture 9
Stochastic Gradient Descent, Feed Forward Networks

Adam Poliak

02/15/2023

Slides adapted Dan Jurafsky, Jordan Boyd-Graber, Nate Chambers

Announcements

- Reading 03 released Monday
  - Due Monday 02/13
- HW03 due Wednesday 02/15 (tonight)
- Reading 04
  - Due Monday 02/20 – Dictionary Methods
- HW04
  - Due next Friday
  - Will be released later today
- Reading 05 – will be released later this week
  - Due Monday 02/27 - CTA/TADA/CSS papers using Word Embeddings
- Office hours:
  - Thursday 3-4:45pm

Prediction Outline

- Linear Regression
- Logistic Regression
- Learning weights – SGD (today)
- Neural Networks (might start today)
  - Feed Forward Networks
    - Word2Vec
  - Recurrent Neural Networks
    - LSTMs
  - Transformers
    - Attention
    - BERT

Outline

Logistic Regression Examples

Learning Weights - SGD

Beyond Binary Classification

Implementation Tricks

Neural Networks

Idea of logistic regression

- We’ll compute $w \cdot x + b$
- And then we’ll pass it through the sigmoid function:

$$\sigma(w \cdot x + b)$$

- And we'll just treat it as a probability

Making probabilities with sigmoids

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

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

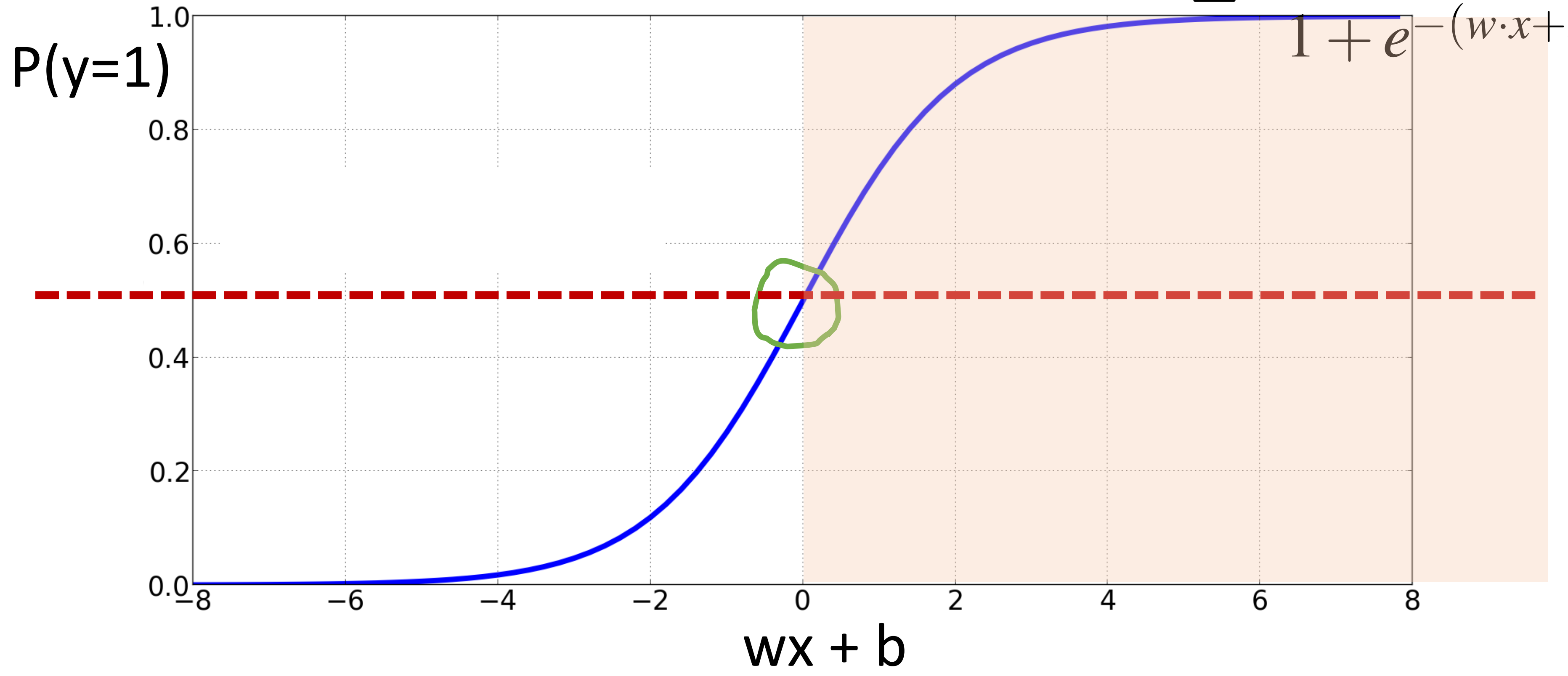
Turning a probability into a classifier

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

0.5 here is called the **decision boundary**

The probabilistic classifier

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



Turning a probability into a classifier

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

$\text{if } w \cdot x + b > 0$

$\text{if } w \cdot x + b \leq 0$

Examples

Feature	Coefficient	Weight
bias	β_0	0.1
"viagra"	β_1	2.0
"mother"	β_2	-1.0
"work"	β_3	-0.5
"nigeria"	β_4	3.0

Example 1: Empty Document
$X = \{\ \}$

$$P(Y=0) = \frac{1}{1+\exp(0.1)} = 0.48$$

$$P(Y=1) = \frac{\exp(0.1)}{1+\exp(0.1)} = 0.52$$

Bias β_0 represents the class priors

Examples

Feature	Coefficient	Weight
bias	β_0	0.1
"viagra"	β_1	2.0
"mother"	β_2	-1.0
"work"	β_3	-0.5
"nigeria"	β_4	3.0

Example 2:
$X = \{Mother, Nigeria\}$

$$P(Y=0) = \frac{1}{1+\exp(0.1\ -1.0+3.0)} = 0.11$$

$$P(Y=1) = \frac{\exp(0.1\ -1.0+3.0)}{1+\exp(0.1\ -1.0+3.0)} = 0.88$$

Examples

Feature	Coefficient	Weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

Example 3:
$X = \{\, Mother, Work, Nigeria, Mother\}$

$$P(Y=0) = \frac{1}{1+\exp(0.1\ -1.0+2.0+3.0\ -1.0)} \quad = 0.60$$

$$P(Y=1) = \frac{\exp(0.1\ -1.0+2.0+3.0\ -1.0)}{1+\exp(0.1\ -1.0+2.0+3.0\ -1.0)} \quad = 0.30$$

Logistic Regression

- Given a set of weights, β, compute conditional likelihood $P(y \,|\beta, x)$

- Find the weights that maximize the conditional likelihood on training data

- **Intuition:** higher weights implies corresponding feature is strongly indicative of the class for the observation

Outline

Process Learning Weights

1. Randomly initialize weights

2. Make predictions $\hat{y}$

3. Quantify how close $\hat{y}$ *and* y are

   We call this the ***distance*** aka Loss function

4. Update weights accordingly

   aka Optimization

5. Repeat 2-4

Process Learning Weights

1. Randomly initialize weights

2. Make predictions $\hat{y}$

3. Quantify how close $\hat{y}$ *and* y are
   We call this the ***distance*** aka Loss function

4. Update weights accordingly
   aka Optimization

5. Repeat 2-4

Distance between $\hat{y}$ and y

We want to know how far is the classifier output:

$\hat{y} = \sigma(w \cdot x + b)$

from the true output:

y [= either 0 or 1]

We'll call this difference our loss:

$L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

$L(\hat{y}, y)$ is a *loss function*

Process Learning Weights

1. Randomly initialize weights

2. Make predictions $\hat{y}$

3. Quantify how close $\hat{y}$ *and* y are

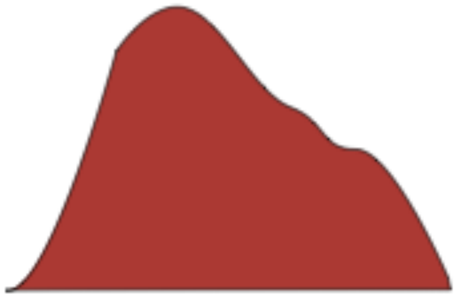
   We call this the ***distance*** aka Loss function

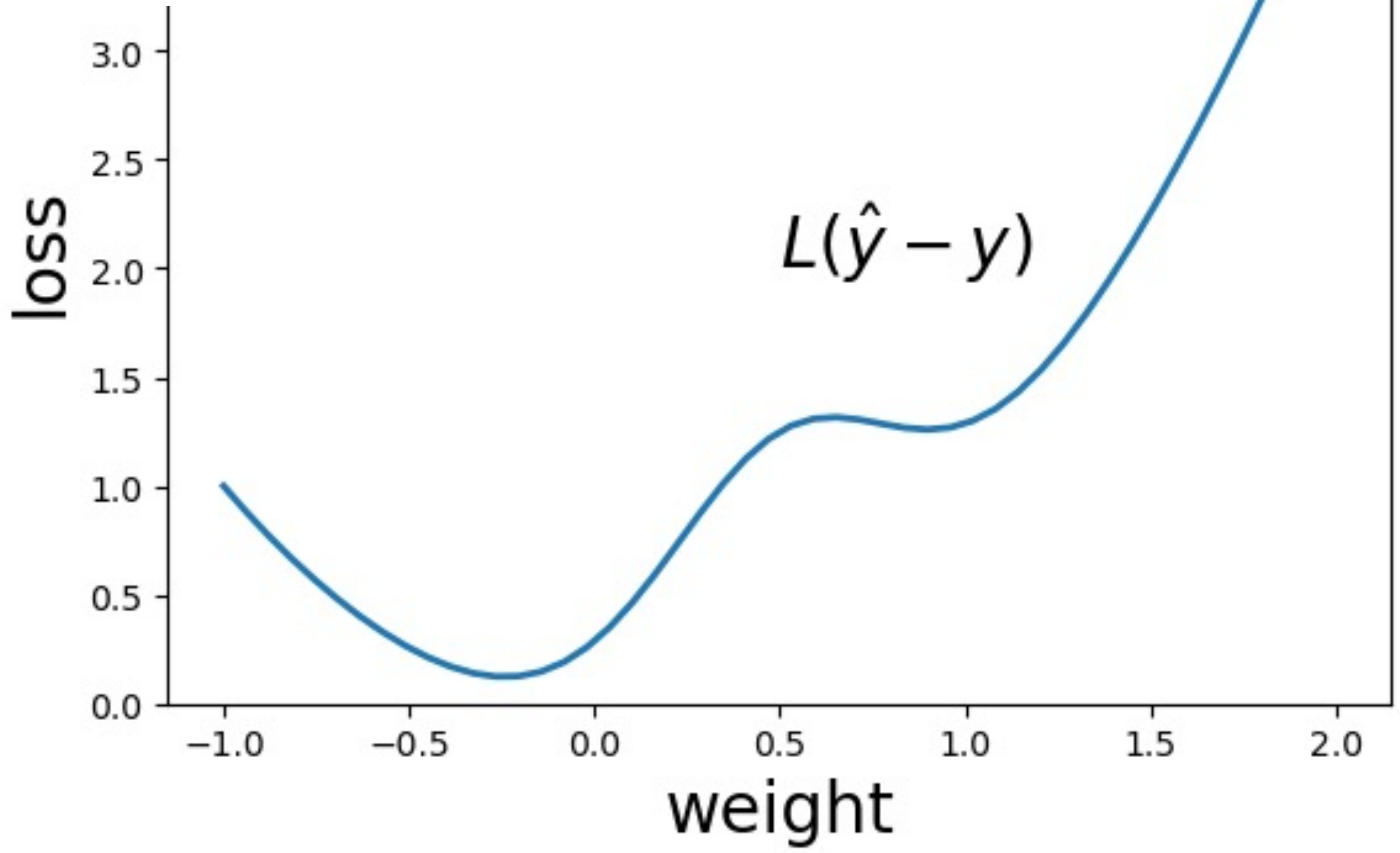
4. Update weights accordingly

   aka Optimization

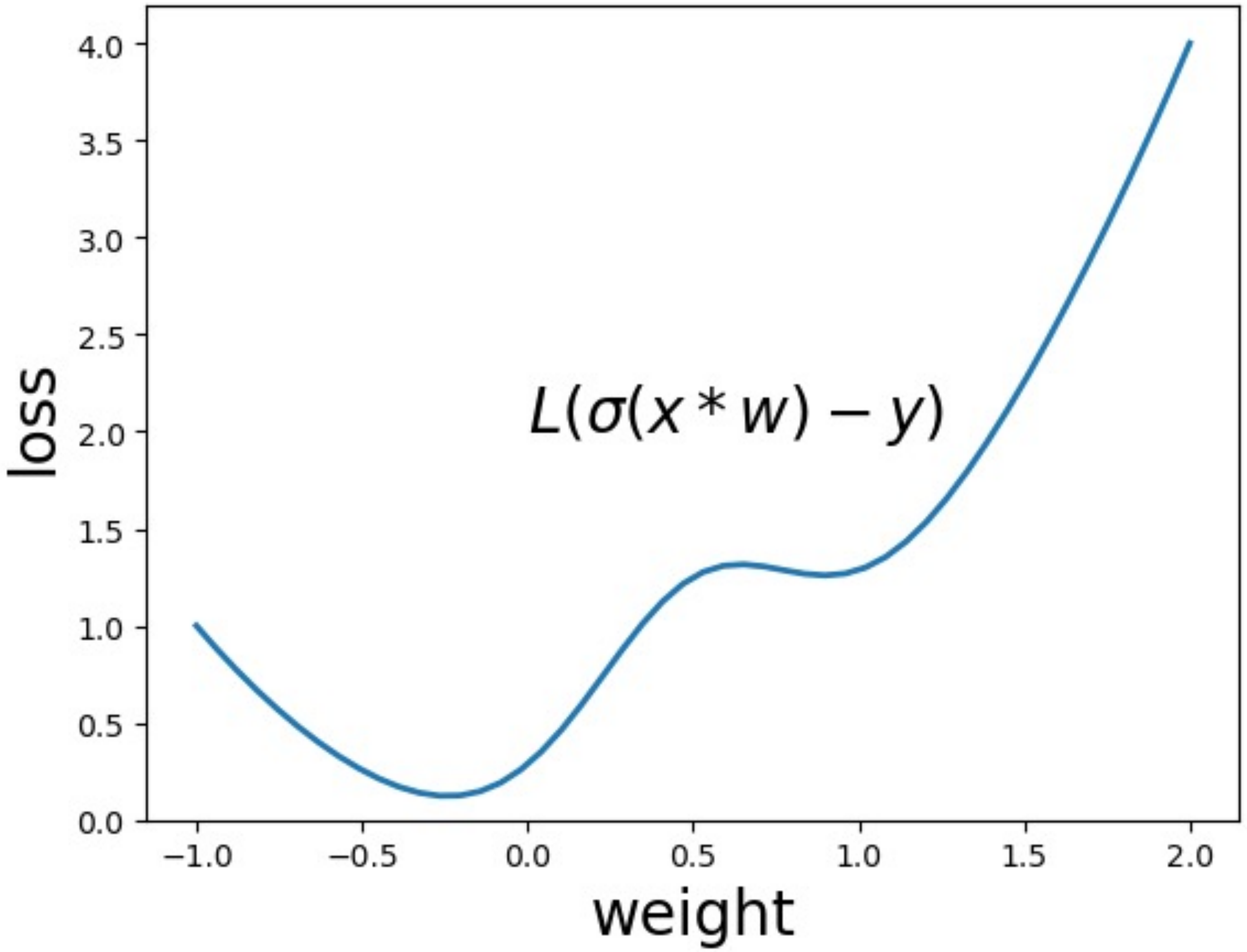
5. Repeat 2-4

hts that minimize the loss

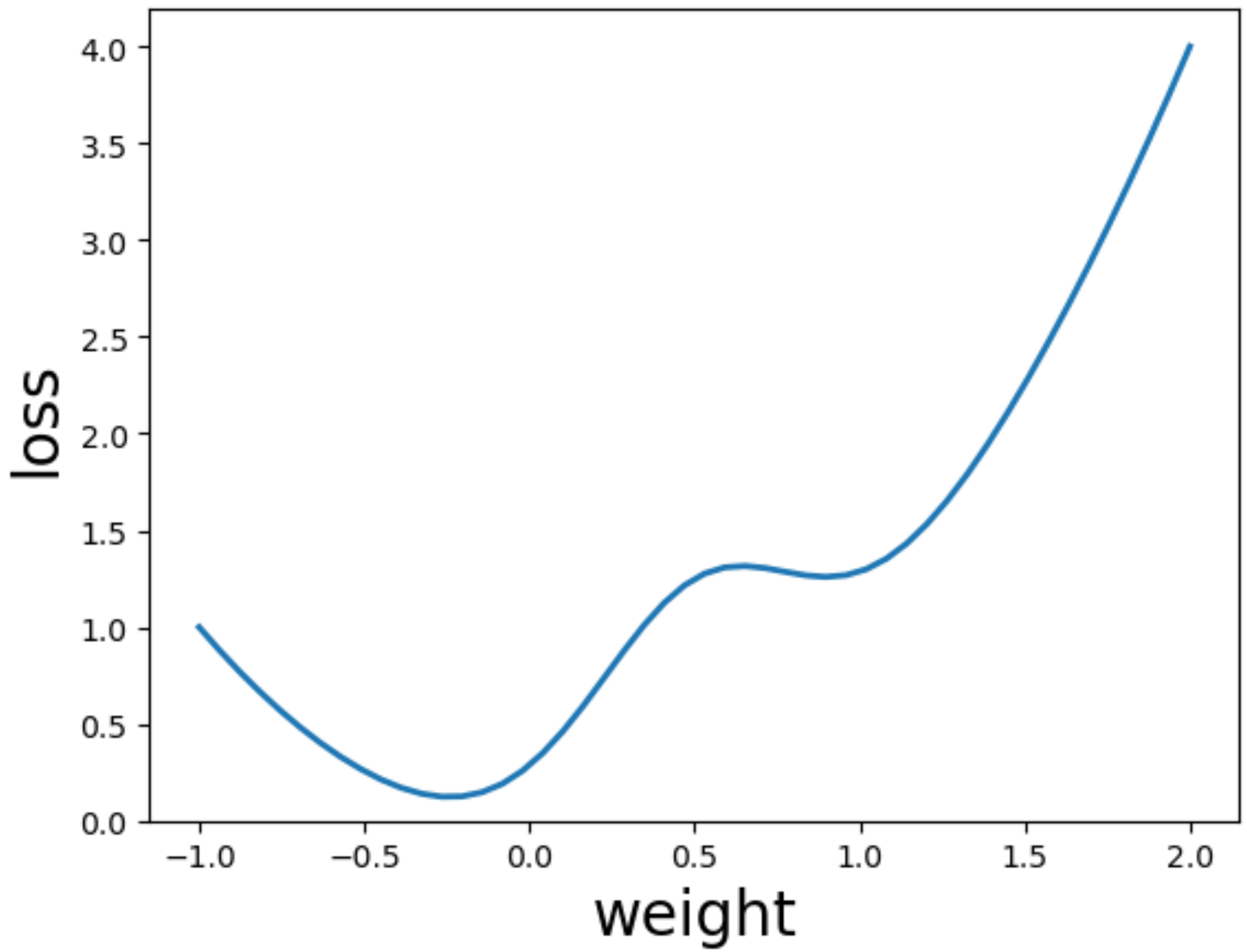




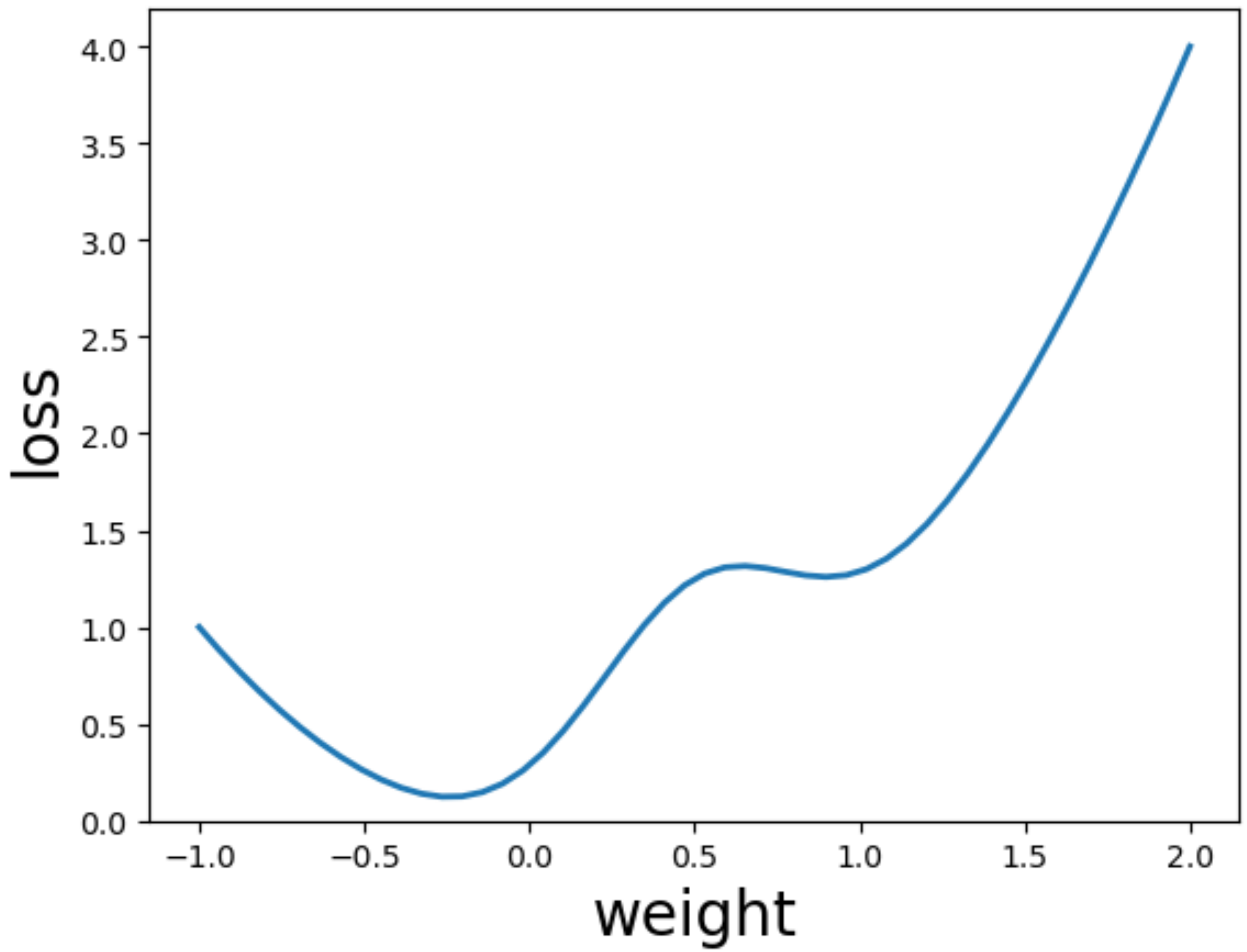
Find weights that minimize the loss



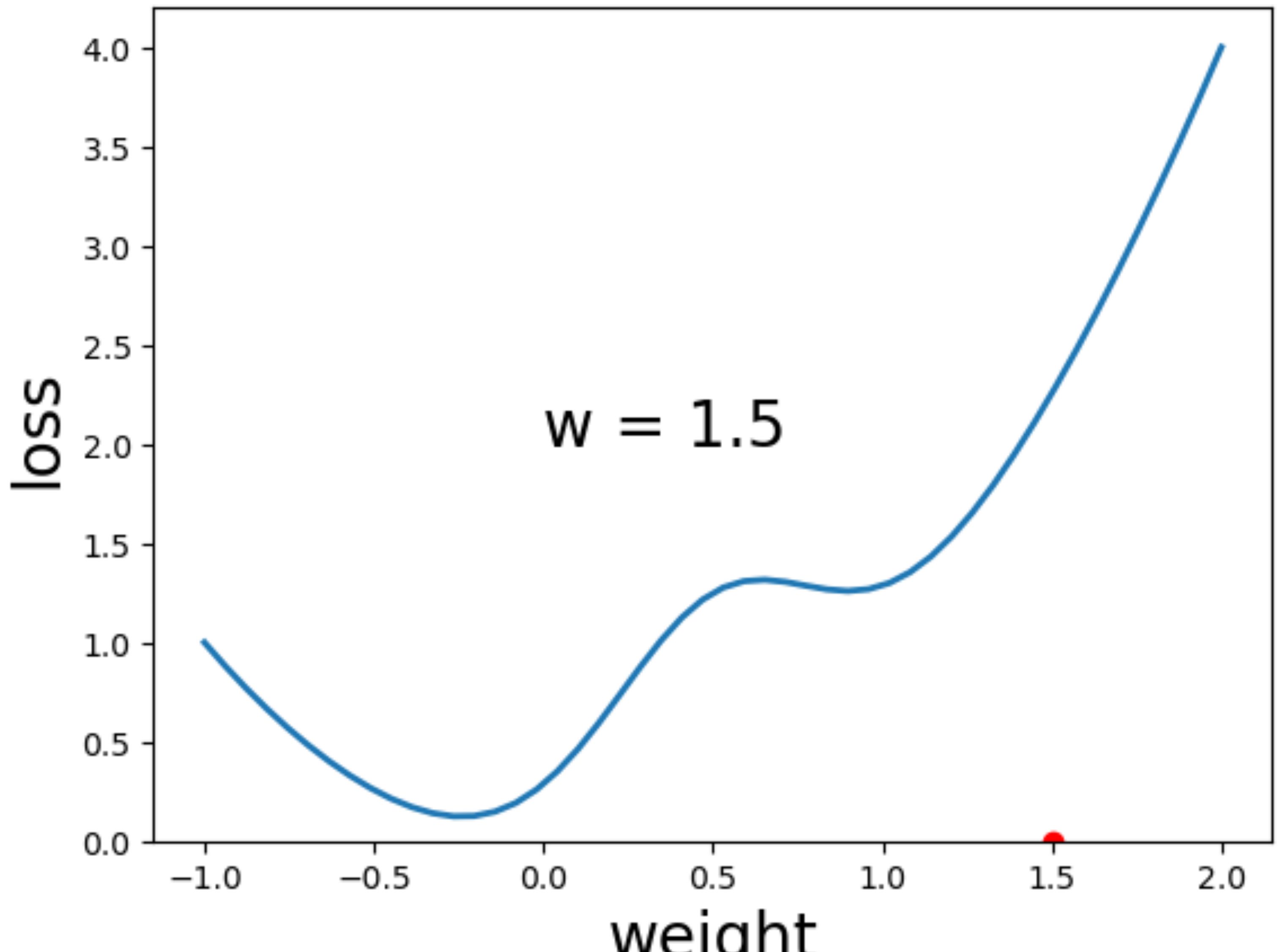
Find weights that minimize the loss



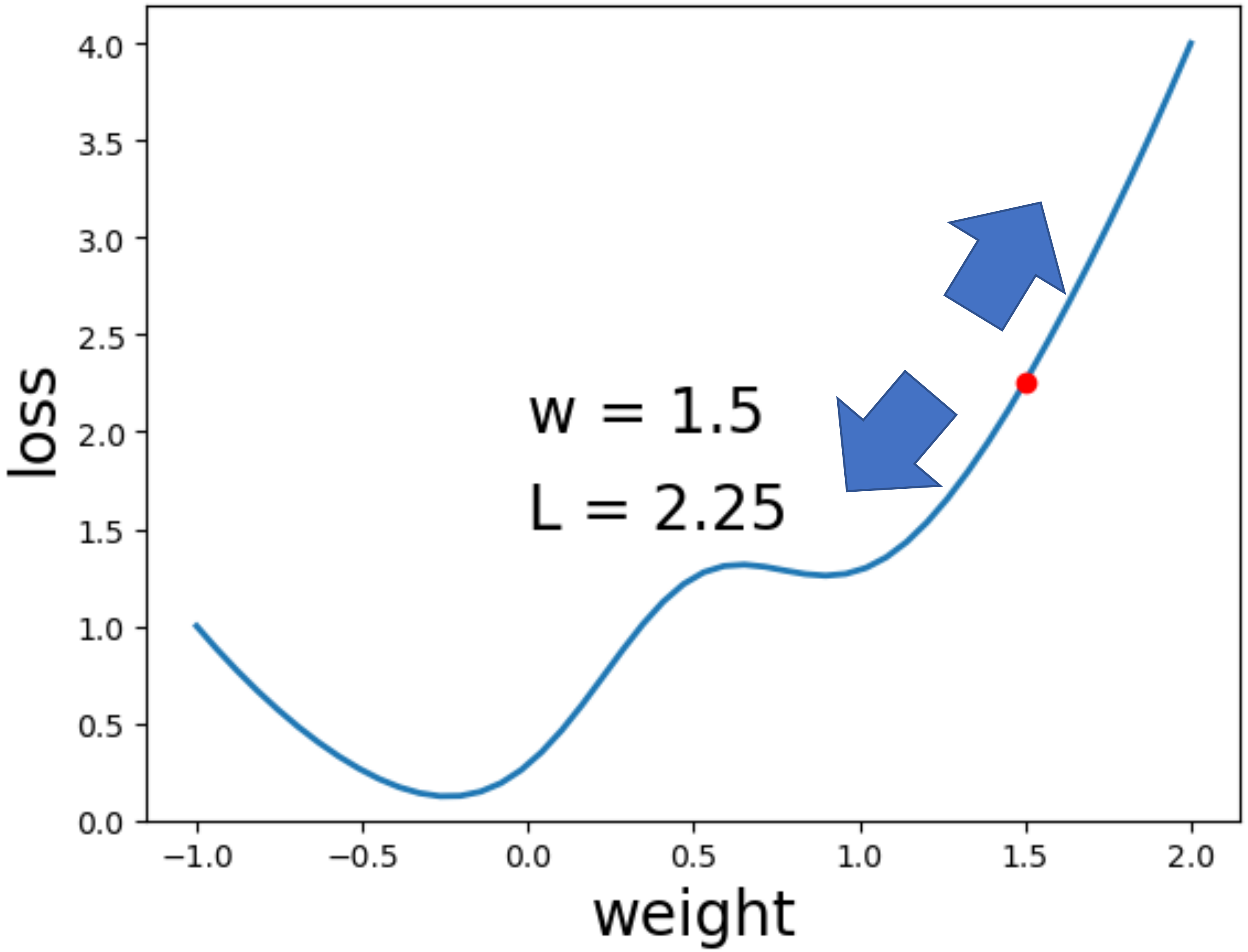
Find weights that minimize the loss



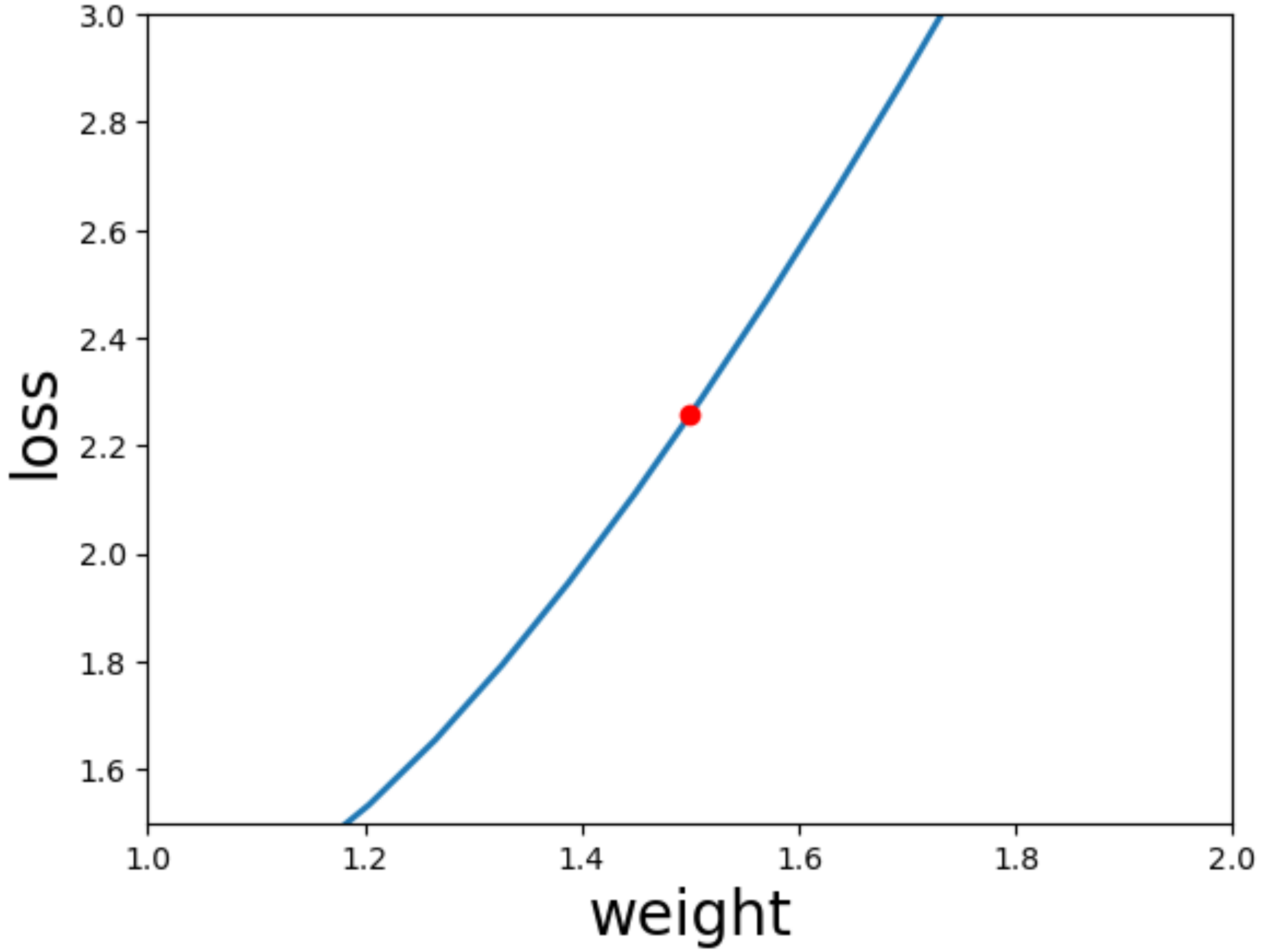
Find weights that minimize the loss



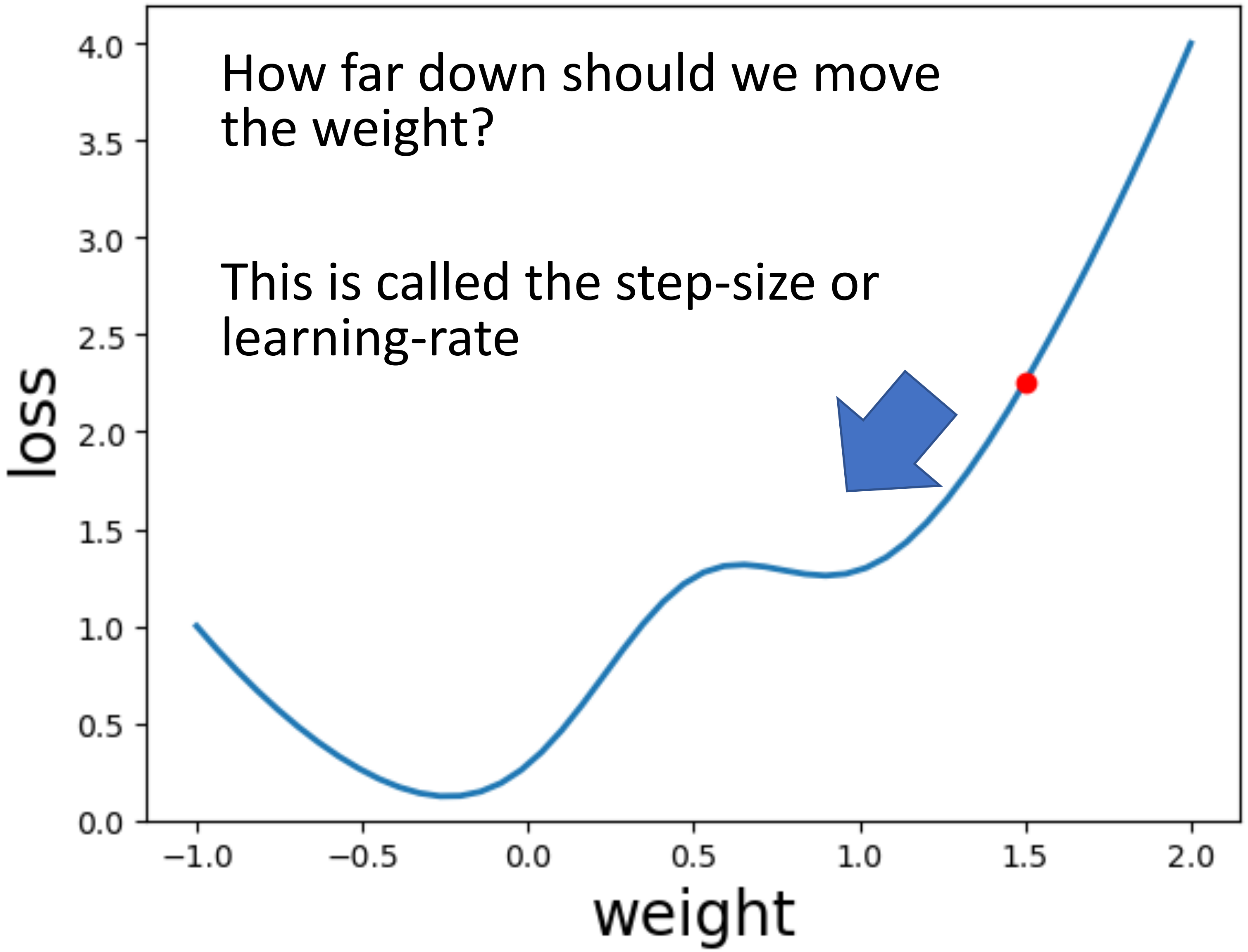
Find weights that minimize the loss



Find weights that minimize the loss



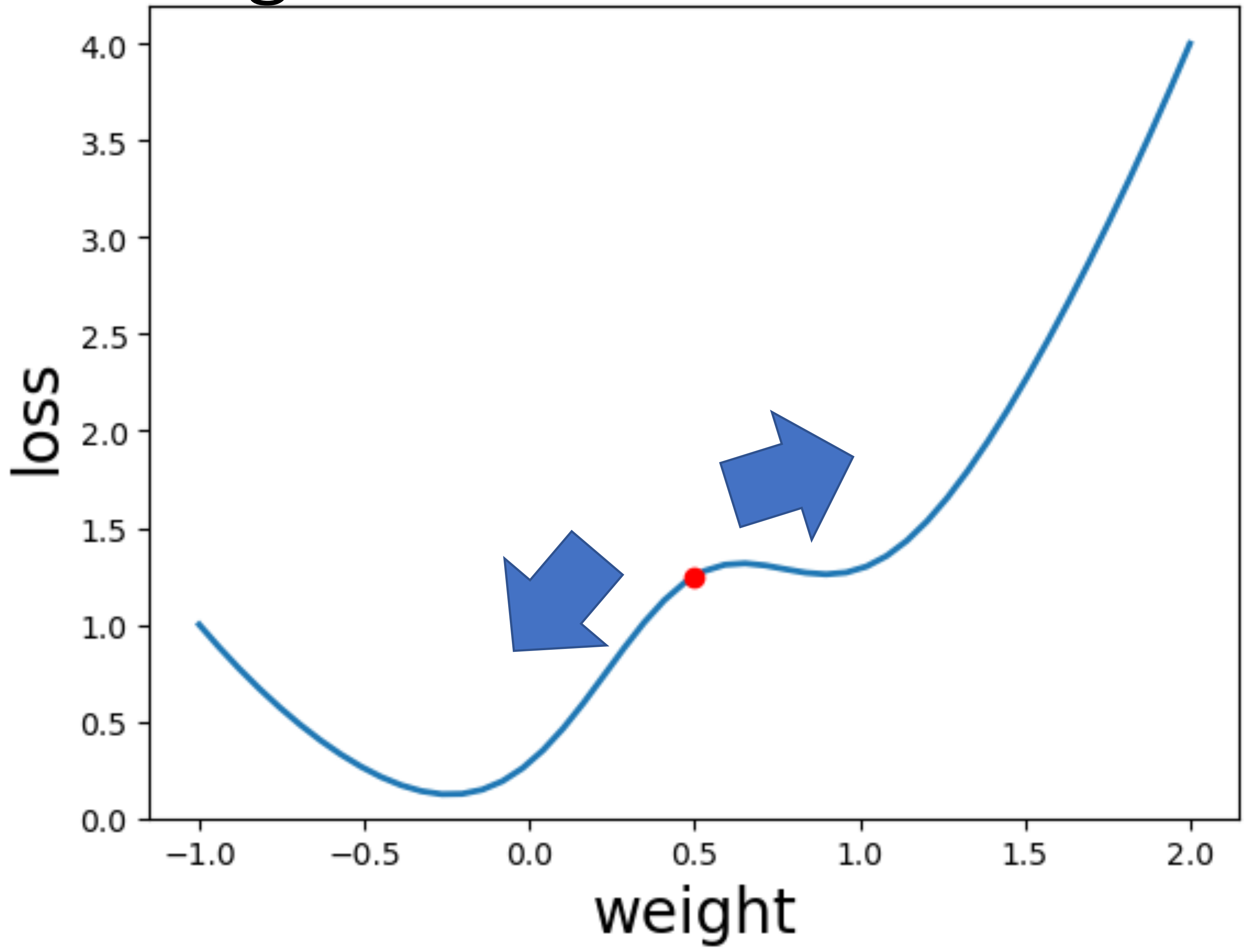
Find weights that minimize the loss



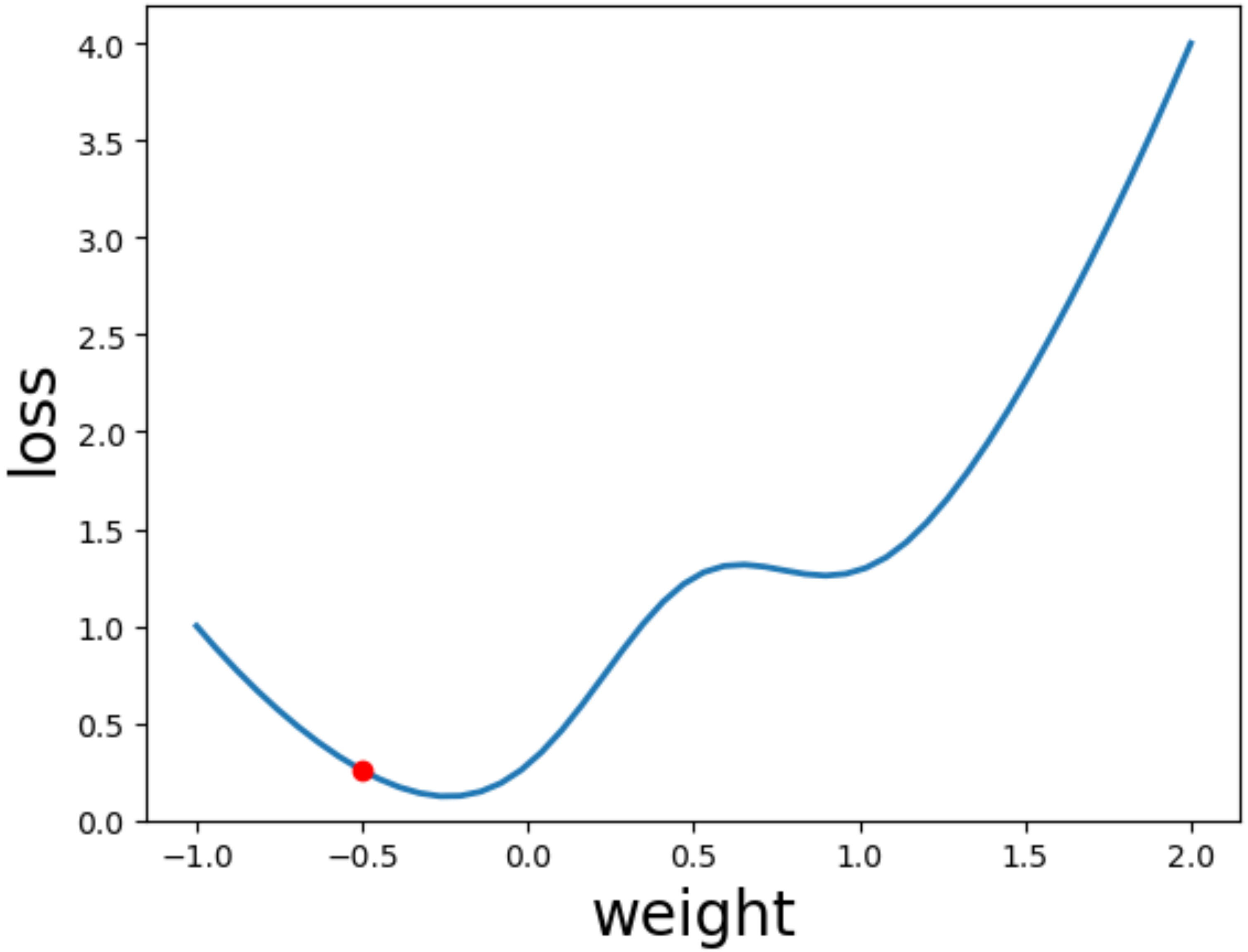
How far down should we move the weight?

This is called the step-size or learning-rate

Find weights that minimize the loss



Find weights that minimize the loss



How to update the weights

1. Find the direction of the derivative of the loss function

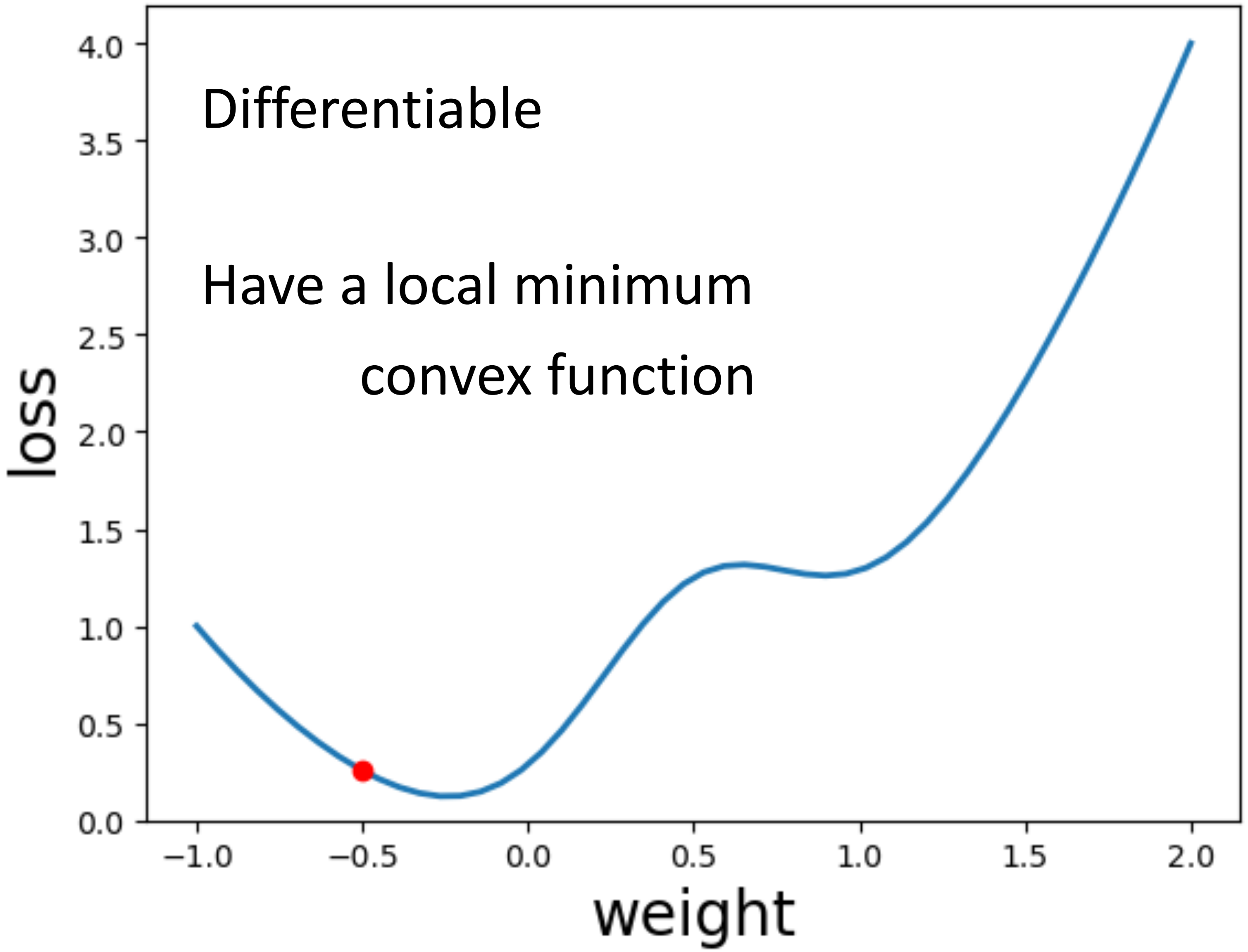
   aka gradient of the loss

2. Move the weight in that direction

3. Then make a prediction on a new $x_{\{i\}}, y_{\{i\}}$ pair, and repeat 1 and 2

4. Repeat this for every example in our training set

Loss function properties

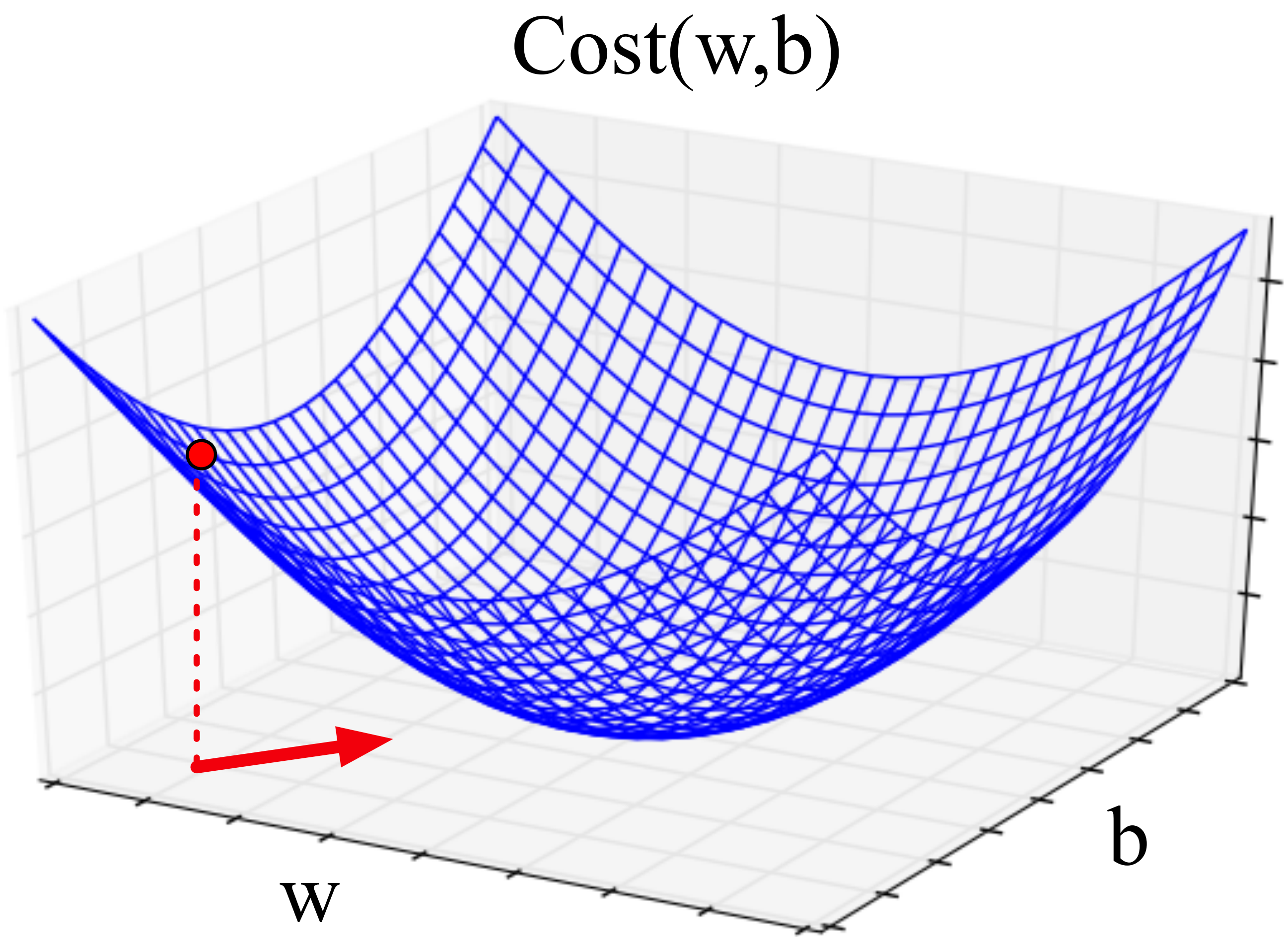


Differentiable

Have a local minimum

convex function

Moving to 2 weights



Process Learning Weights

1. Randomly initialize weights

2. Make predictions $\hat{y}$

Details!

3. Quantify how close $\hat{y}$ *and* y are

   We call this the ***distance*** aka Loss function

4. Update weights accordingly

   aka Optimization

5. Repeat 2-4

Loss function

This probability is the likelihood of the label given the data

Log likelihood

$$\mathcal{L}(\hat{y}, y) = \log(p\left(y_{\{i\}} \middle| x_{\{i\}}, \beta\right))$$

Computing this loss across our entire dataset

$$\mathcal{L}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_{i}^{n} \log(p\left(y_{\{i\}} \middle| x_{\{i\}}, \beta\right))$$

Recall

$P(y = 1) =$

$$\sigma\left(\sum_j^p \beta_{\{j\}} * x_{\{j\}}\right)$$

$P(y = 0) =$

$$1 - \sigma\left(\sum_j^p \beta_{\{j\}} * x_{\{j\}}\right)$$

Loss function

$P(y = 1) =$

$$\sigma(\boldsymbol{\beta} \, * \boldsymbol{x})$$

$P(y = 0) =$

$$1 - \sigma(\boldsymbol{\beta} \, * \boldsymbol{x})$$

Recall that $\sigma(\ldots)$ is used to create probabilities

$\mathcal{L}(\hat{y}, y) = \log(p\left(y_{\{i\}} \middle| x_{\{i\}}, \beta\right)) =$

$\log(\sigma(\boldsymbol{\beta} \, * \boldsymbol{x}))$ if y = 1

$\log(1 - \sigma(\boldsymbol{\beta} \, * \boldsymbol{x}))$ if y = 0

Loss function - examples

$\mathcal{L}(\hat{y}, y) =$

$$\log(\sigma(\boldsymbol{\beta} * \boldsymbol{x})) \quad \text{if y = 1}$$

$$\log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x})) \quad \text{if y = 0}$$

Gold label (y)	**Predicted ($\hat{y}$)**	$\mathcal{L}(\hat{y}, y)$

Loss function - examples

$\mathcal{L}(\hat{y}, y) =$

$\log(\sigma(\boldsymbol{\beta} * \boldsymbol{x}))$ if y = 1

$\log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x}))$ if y = 0

Gold label (y)	Predicted ($\hat{y}$)	$\mathcal{L}(\hat{y}, y)$
1	0.2	
0	0.01	
0	0.75	
1	0.75	

Loss function - examples

$\mathcal{L}(\hat{y}, y) =$

$\log(\sigma(\boldsymbol{\beta} * \boldsymbol{x}))$ if y = 1

$\log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x}))$ if y = 0

Gold label (y)	Predicted ($\hat{y}$)	$\mathcal{L}(\hat{y}, y)$
1	0.2	0.8
0	0.01	.99
0	0.75	0.75
1	0.75	0.25

Examples

Feature	Coefficient	Weight
bias	β_0	0.1
“viagra”	β_1	2.0
“mother”	β_2	-1.0
“work”	β_3	-0.5
“nigeria”	β_4	3.0

Example 2:
$X = \{ Mother, Nigeria \}$

$P(Y = 0) = 0.11$
$P(Y = 1) = 0.88$

What’s $\mathcal{L}(\hat{y}, y)$ if y = 0?
What’s $\mathcal{L}(\hat{y}, y)$ if y = 1?

Loss function

$$\mathcal{L}(\hat{y}, y) =$$

$$\begin{array}{ll} \log(\sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if y = 1} \\ \log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if y = 0} \end{array}$$

What about across our entire dataset?

$$\mathcal{L}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) =$$

$$= \sum_i^n \log(p\left(y_{\{i\}} \middle| x_{\{i\}}, \beta\right))$$

$$= \sum_i^n \begin{cases} \log(\sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if } y = 1 \\ \log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

Loss function

$$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) =$$

$$= \sum_i^n \log(p\left(y_{\{i\}} \middle| x_{\{i\}}, \beta\right))$$

$$= \sum_i^n \begin{cases} \log(\sigma(\boldsymbol{\beta} \ * \boldsymbol{x})) & \text{if y} = 1 \\ \log(1 \ - \ \sigma(\boldsymbol{\beta} \ * \boldsymbol{x})) & \text{if y} = 0 \end{cases}$$

Process Learning Weights

1. Randomly initialize weights

2. Make predictions $\hat{y}$

3. Quantify how close $\hat{y}$ *and* y are

   We call this the ***distance*** aka Loss function

4. Update weights accordingly

   aka Optimization

5. Repeat 2-4

Details!

Loss function

$$\mathcal{L}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) =$$

$$= \sum_i^n \log(p\left(y_{\{i\}} \middle| x_{\{i\}}, \beta\right))$$

$$= \sum_i^n \begin{cases} \log(\sigma(\boldsymbol{\beta} \ * \boldsymbol{x})) & \text{if y} = 1 \\ \log(1 - \sigma(\boldsymbol{\beta} \ * \boldsymbol{x})) & \text{if y} = 0 \end{cases}$$

How do we update the weights?

1. Compute the derivative/gradient of $\mathcal{L}$
2. Update the weights based on the direction of the derivative/gradient

Computing the gradient of $\mathcal{L}$

$\mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}) =$

$$= \sum_i^n \log(p(y_{\{i\}} | x_{\{i\}}, \beta))$$

What variable in the loss function can we update?

Answer: β

When we compute $\nabla\mathcal{L}$, what variable should use to take the derivative?

Answer: β take the derivate

Computing the gradient of $\mathcal{L}$

$$\nabla\mathcal{L} = \frac{d}{d\beta}L(\sigma(\beta * x), y)$$

How do we take the derivative of this?

Chain rule!

Chain Rule

If y is the result of

$$y = f\big(g(x)\big)$$

Then,

$$\frac{dy}{dx} = \frac{df}{dg}\,\frac{dg}{dx}$$

$$= f'[g(x)] * g'(x)$$

Example:

$$y = (x^2 + 1)^3$$

$$\frac{dy}{dx} =$$

$$= 3(x^2 + 1)^2 * 2x$$

$$= 6x(x^2 + 1)^2$$

Computing the gradient of $\mathcal{L}$

$$\nabla\mathcal{L} = \frac{d}{d\beta} L(\sigma(\beta \ * \ x), y)$$

What are our two functions here that depend on β?

1. σ
2. $\beta * x$

Computing the gradient of $\mathcal{L}$

$$\nabla\mathcal{L} = \frac{d}{d\beta}L(\sigma(\beta\ *\ x), y)$$

What are our two functions here that depend on β?

1. $\mathrm{f}\ =\ \sigma$
2. $\mathrm{g} =\ \beta * x$

Computing the gradient of $\mathcal{L}$

$$\nabla\mathcal{L} = \frac{d}{d\beta}L(\sigma(\beta \, * \, x), y)$$

What are our two functions here that depend on β?

1. $\mathrm{f} = \sigma$
2. $\mathrm{g} = \beta * x$

$$\begin{aligned}\nabla\mathcal{L} &= f'[g(x)] * g'(x) \\ &= \sigma'(\beta * x)\, g'(x)\end{aligned}$$

Computing the gradient of $\mathcal{L}$

1. $\mathrm{f} = \sigma$
2. $\mathrm{g} = \beta * x$

$$\begin{aligned}\nabla\mathcal{L} &= f'[g(x)] * g'(x) \\ &= \sigma'(\beta * x)\, g'(x)\end{aligned}$$

What's $\frac{d\sigma}{d\beta}$ and $\frac{d}{d\beta}\beta * x$?

$$\frac{d}{d\beta}\beta * x = x$$

$$\frac{d\sigma}{d\beta} = \sigma * (1 - \sigma)$$

Derivative of σ

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{(0)(1+e^{-x}) - (-e^{-x})(1)}{(1+e^{-x})^2} \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\frac{e^{-x}+(1-1)}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}}\left[\frac{(1+e^{-x})}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right] \\
&= \frac{1}{1+e^{-x}}\left[1 - \frac{1}{1+e^{-x}}\right] \\
&= \sigma(x)(1-\sigma(x))
\end{aligned}
$$

https://hausetutorials.netlify.app/posts/2019-12-01-neural-networks-deriving-the-sigmoid-derivative/

Computing the gradient of $\mathcal{L}$

$$\begin{aligned}\nabla\mathcal{L} &= f'[g(x)] * g'(x)\\ &= \sigma'(\beta * x)\, g'(x)\\ &= \sigma(\beta * x)(1 - \sigma(\beta * x))x\end{aligned}$$

But remember, what's $\mathcal{L}$?

$$\sum_i^n \begin{cases}\log(\sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if } \mathrm{y} = 1\\ \log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if } \mathrm{y} = 0\end{cases}$$

So, we need to apply the chain rule again

Computing the gradient of $\mathcal{L}$

So, we need to apply the chain rule again.
Facts:

1. $\frac{d}{dx}\log(\mathrm{x}) = \frac{1}{x}$

2. $\sum_i^n \begin{cases} \log(\sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if y} = 1 \\ \log(1 - \sigma(\boldsymbol{\beta} * \boldsymbol{x})) & \text{if y} = 0 \end{cases}$

3. $\nabla\mathcal{L} = f'[g(x)] * g'(x)$

Computing the gradient of $\mathcal{L}$

When $y = 1$

$$\nabla\mathcal{L} = log'(\sigma) * \sigma'$$

$$= \frac{1}{\sigma} * (\sigma * (1 - \sigma)x)$$

$$= (1 - \sigma)x$$

When $y = 0$

$$\nabla\mathcal{L} = log'(1 - \sigma) * - \sigma'$$

$$= \frac{1}{1 - \sigma}(-(\sigma * (1 - \sigma)x))$$

$$= \frac{-(\sigma * (1 - \sigma)x)}{1 - \sigma}$$

$$= -\sigma x$$

Computing the gradient of $\mathcal{L}$

$\nabla\mathcal{L} =$

$$\sum_i^n \begin{cases} (1 - \sigma)x & \text{if y} = 1 \\ -\sigma x & \text{if y} = 0 \end{cases}$$

$$\nabla\mathcal{L} = \begin{cases} (1 - \sigma)x & \text{if y} = 1 \\ -\sigma x & \text{if y} = 0 \end{cases}$$

Putting it into one equation:
$\nabla\mathcal{L} = (\mathrm{y} - \sigma)x$

Computing the gradient of $\mathcal{L}$

$$\nabla\mathcal{L} = (\mathrm{y} - \sigma)x$$

$$\begin{aligned}\frac{d\mathcal{L}}{d\beta} &= (\mathrm{y} - \sigma)x \\ &= \big(\mathrm{y} - \sigma(\beta * x)\big)x \\ &= \left(\mathrm{y} - \frac{1}{1+ e^{-(\beta * x)}}\right)x\end{aligned}$$

What if we have multiple β's?

Solution: partial derivatives!

Real gradients

- We have lots of weights/parameters
- For each parameter β_i, the gradient component *i* tells us the slope with respect to that variable.
  - "How much would a small change in β_i influence the total loss function $\mathcal{L}$?"
  - We express the slope as a partial derivative ∂ of the loss ∂ β_i
- The gradient is then defined as a vector of these partials.

Computing the gradient of $\mathcal{L}$ partial derivatives

$$\nabla_\beta \mathcal{L} = \begin{bmatrix} \frac{d\mathcal{L}}{d\beta_0} \\ \frac{d\mathcal{L}}{d\beta_1} \\ \cdots \\ \cdots \\ \frac{d\mathcal{L}}{d\beta_j} \end{bmatrix}$$

What can we do after we computed the gradients?

Updating weights based on gradients

$$\Delta\beta = \eta \nabla_{\beta} \mathcal{L}(\beta)$$

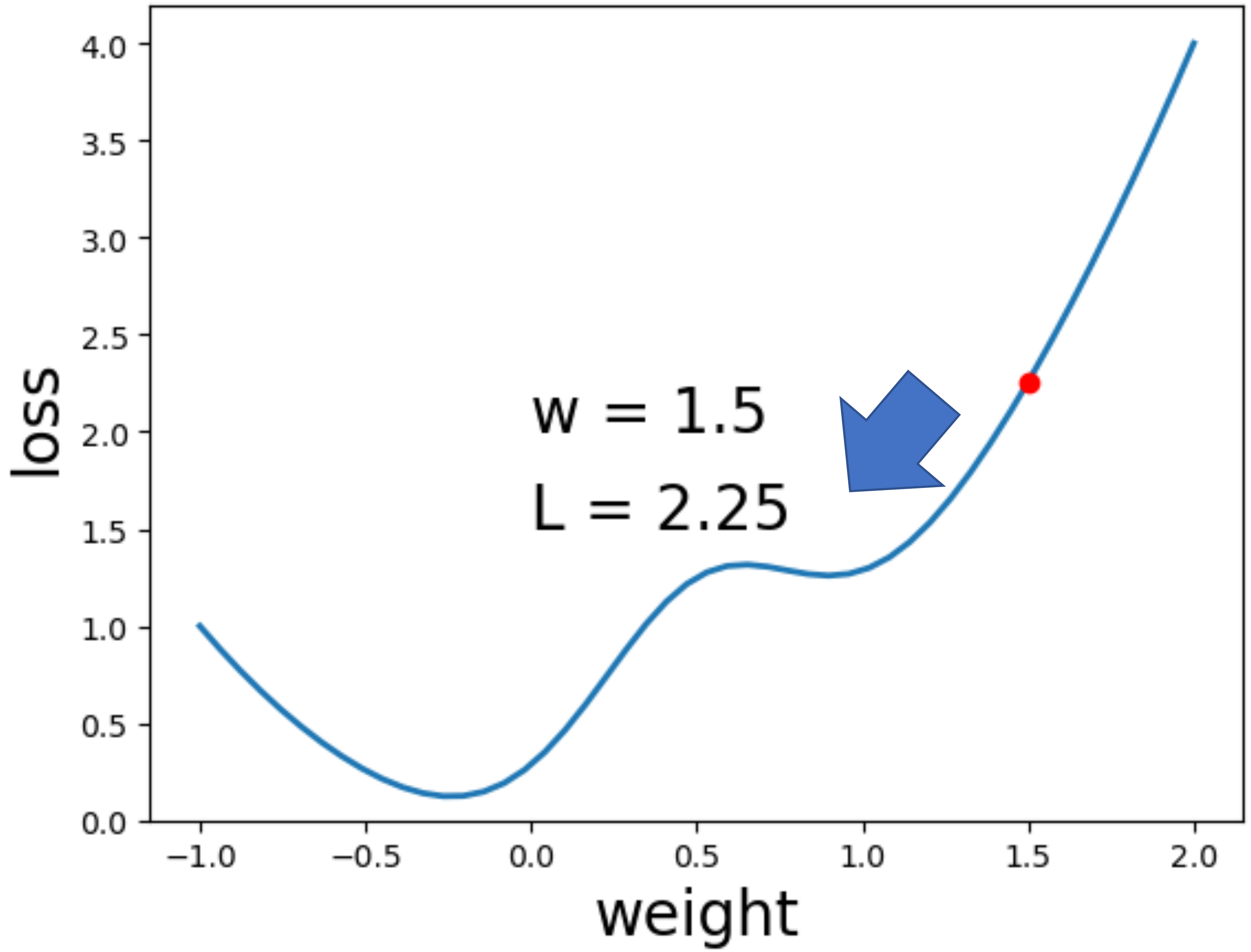
Update each individual weight:

$$\beta_i \leftarrow \beta_i - \eta \frac{d\mathcal{L}(\beta)}{d\beta_i}$$

If we want to perform gradient ascent, we …

$$\beta_i \leftarrow \beta_i + \eta \frac{d\mathcal{L}(\beta)}{d\beta_i}$$

Find weights that minimize the loss



Updating weights based on gradients

$$\Delta\beta = \eta\nabla_{\beta}\mathcal{L}(\beta)$$

Update each individual weight:

$$\beta_i \leftarrow \beta_i - \eta\frac{d\mathcal{L}(\beta)}{d\beta_i}$$

If we want to perform gradient ascent, we …

$$\beta_i \leftarrow \beta_i + \eta\frac{d\mathcal{L}(\beta)}{d\beta_i}$$

Step size

Process Learning Weights

1. Randomly initialize weights

2. Make predictions $\hat{y}$

3. Quantify how close $\hat{y}$ *and* y are

   We call this the ***distance*** aka Loss function

4. Update weights accordingly

   aka Optimization

5. Repeat 2-4

Stochastic Gradient Descent

```
1. Randomly initialize $\beta_i$
2. For every $\{x_i, y_i\}$ pair in our training set:
        Compute the gradient of the loss
            $\frac{d\mathcal{L}(\beta)}{d\beta_i}$
        Update each weights based on the gradients
            $\beta_i \leftarrow \beta_i - \eta \frac{d\mathcal{L}(\beta)}{d\beta_i}$

3. Repeat 2 until convergance (or max epochs)
4. return $\beta_i$
```

Hyperparameters

- Hyperparameters:
  - Briefly, a special kind of parameter for an ML model
  - Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.
- The learning rate η is a **hyperparameter**
  - too high: the learner will take big steps and overshoot
  - too low: the learner will take too long

SGD Example

Working through an example

- One step of gradient descent
- A mini-sentiment example, where the true y=1 (positive)
- Two features:

  $x_1 = 3$ (count of positive lexicon words)

  $x_2 = 2$ (count of negative lexicon words)

Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:

$w_1 = w_2 = b = 0$

$\eta = 0.1$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

- where $\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3;\ \ x_2 = 2$

- Update step for update θ is:

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

- where $\frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} \quad \end{bmatrix}$$

Example of gradient descent

w$_1$ = *w*$_2$ = *b* = 0;
x$_1$ = 3; *x*$_2$ = 2

- Update step for update θ is:

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

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_j} \;=\; [\sigma(w\cdot x+b)-y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w\cdot x+b)-y)x_1 \\ (\sigma(w\cdot x+b)-y)x_2 \\ \sigma(w\cdot x+b)-y \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

- where $\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} =$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

- where $\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} \; = \; [\sigma(w \cdot x + b) - y]x_j$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$\eta = 0.1$;

$$\theta^1 =$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

η = 0.1;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

η = 0.1;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$\eta = 0.1$;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Outline

Logistic Regression Examples

Learning Weights - SGD

Beyond Binary Classification

Implementation Tricks

Neural Networks

Prediction: NLP/ML vs CTA/TADA

NLP/ML:

- Make prediction about unseen data
  - Predict if a stock will go up or down based on social media posts

CTA/TADA/CSS:

- Learn something about different categories
  - Are there different terms/concepts used to describe male vs female professors on course reviews

Outline

Logistic Regression Examples

Learning Weights - SGD

Beyond Binary Classification

Implementation Tricks

Neural Networks

Multinomial Logistic Regression
aka softmax regression, multinomial logit

Softmax: a generalization of the sigmoid

- Takes a vector of *k* values
  - think scores for each class
- Outputs a probability distribution
  - each value in the range [0,1]
  - all the values summing to 1

$$softmax(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \qquad 1 \le i \le k$$

$$softmax(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{j=1}^{k} \exp(z_j)}, \frac{\exp(z_2)}{\sum_{j=1}^{k} \exp(z_j)}, \dots, \frac{\exp(z_3)}{\sum_{j=1}^{k} \exp(z_j)}\right]$$

Outline

Logistic Regression Examples

Learning Weights - SGD

Beyond Binary Classification

Implementation Tricks

Neural Networks

Mini-batch training

- Stochastic gradient descent chooses a single random example at a time.
- That can result in choppy movements
- More common to compute gradient over batches of training instances.
- **Batch training**: entire dataset
- **Mini-batch training**: *m* examples (512, or 1024)

Overfitting

- A model that perfectly match the training data has a problem.
- It will also **overfit** to the data, modeling noise
  - A random word that perfectly predicts *y* (it happens to only occur in one class) will get a very high weight.
  - Failing to generalize to a test set without this word.
- A good model should be able to **generalize**

Overfitting

Useful or harmless features

X1 = "this"
X2 = "movie
X3 = "hated"

X4 = "drew me in"

+

- This movie drew me in, and it'll do the same to you.

−

I can't tell you how much I hated this movie. It sucked.

4gram features that just "memorize" training set and might cause problems

X5 = "the same to you"
X7 = "tell you how much"

Overfitting

- 4-gram model on tiny data will just memorize the data
  - 100% accuracy on the training set
- But it will be surprised by the novel 4-grams in the test data
  - Low accuracy on test set
- Models that are too powerful can **overfit** the data
  - Fitting the details of the training data so exactly that the model doesn't generalize well to the test set
    - How to avoid overfitting?
      - Regularization in logistic regression
      - Dropout in neural networks

Regularization

- A solution for overfitting
- Add a regularization term *R*(θ) to the loss function
  (for now written as maximizing logprob rather than minimizing loss)

$$\hat{\theta} \;=\; \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) - \alpha R(\theta)$$

- Idea: choose an *R*(θ) that penalizes large weights
  - fitting the data well with lots of big weights not as good as fitting the data a little less well, with small weights

L2 Regularization (= ridge regression)

- The sum of the squares of the weights
- The name is because this is the (square of the) **L2 norm** $\|\theta\|_2$**, = Euclidean distance** of θ to the origin.

$$R(\theta) \;=\; ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

- L2 regularized objective function:

$$\hat{\theta} \;=\; \operatorname*{argmax}_{\theta} \left[\sum_{i=1}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} \theta_j^2$$

L1 Regularization (= lasso regression)

- The sum of the (absolute value of the) weights
- Named after the **L1 norm** $\|W\|_1$, = sum of the absolute values of the weights, = **Manhattan distance**

$$R(\theta) \;=\; ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

- L1 regularized objective function:

$$\hat{\theta} \;=\; \operatorname*{argmax}_{\theta} \left[\sum_{1=i}^{m} \log P(y^{(i)}|x^{(i)}) \right] - \alpha \sum_{j=1}^{n} |\theta_j|$$

Outline

Logistic Regression Examples

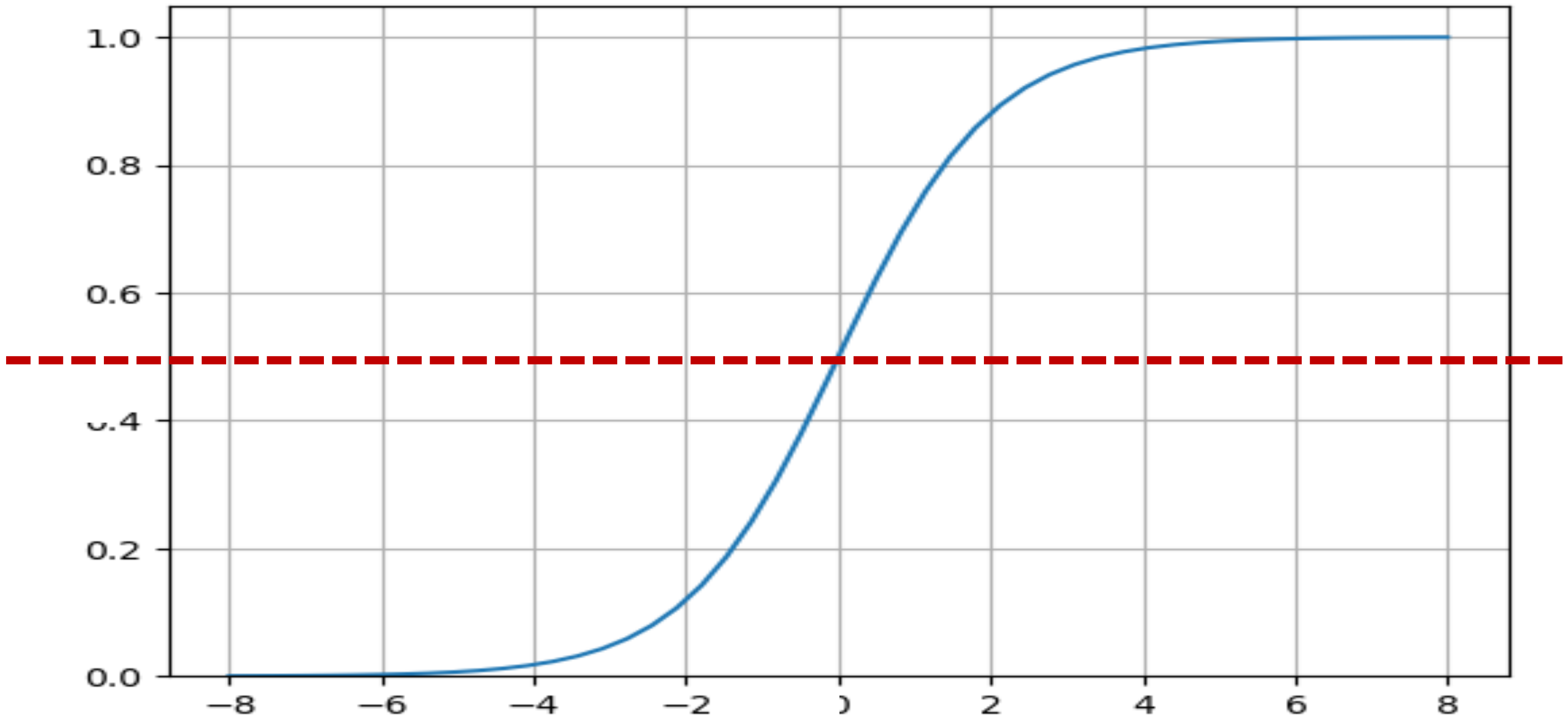
Learning Weights - SGD

Beyond Binary Classification

Implementation Tricks

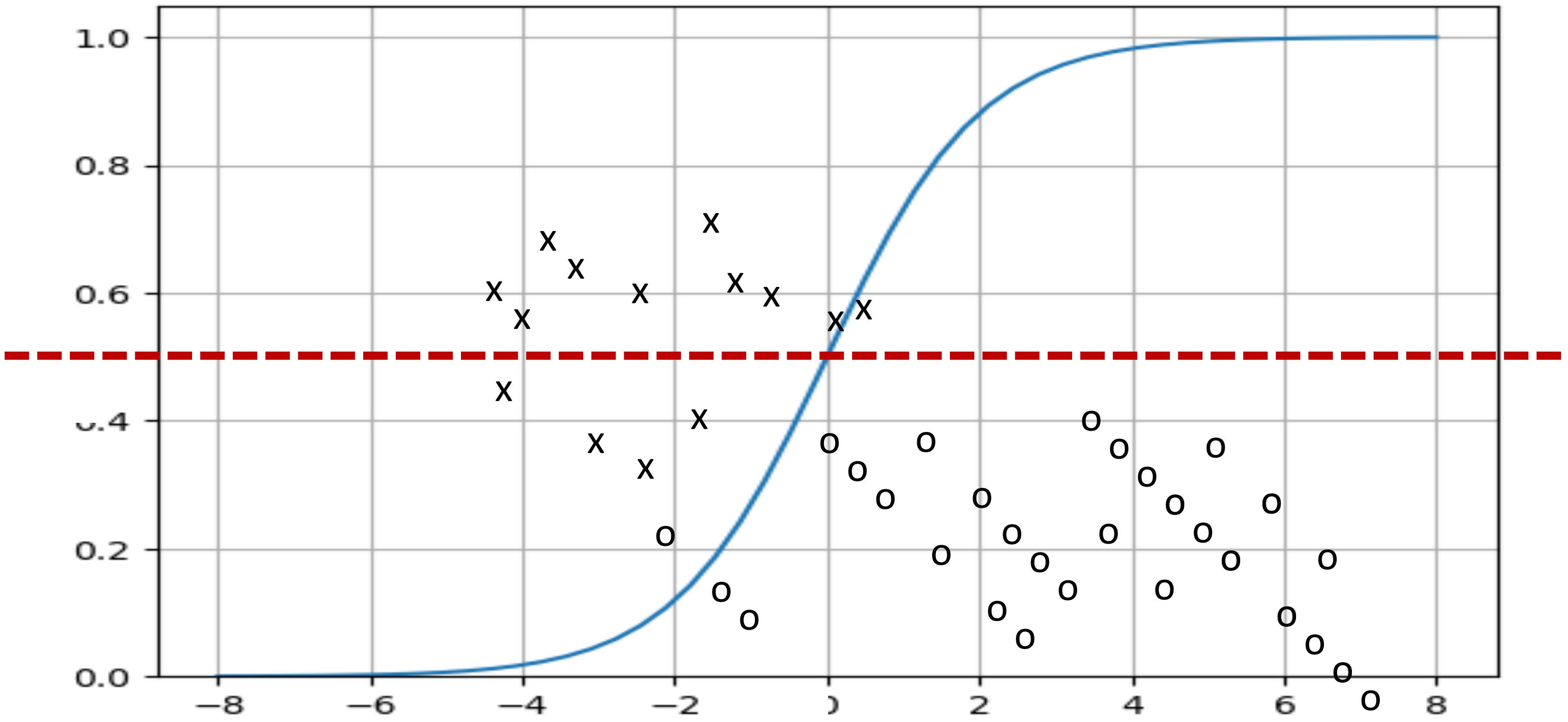
Neural Networks

Logistic Regression



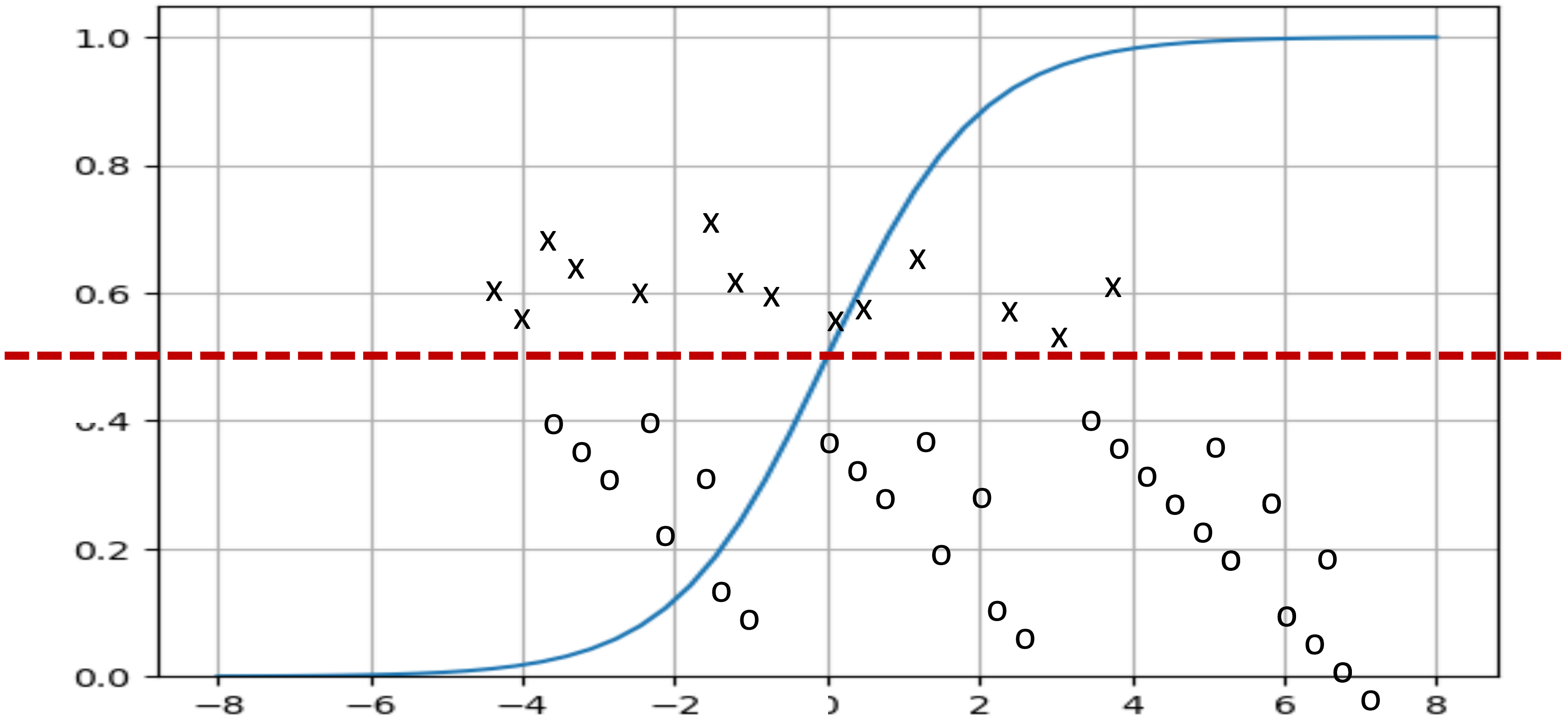
wx + b

Logistic Regression



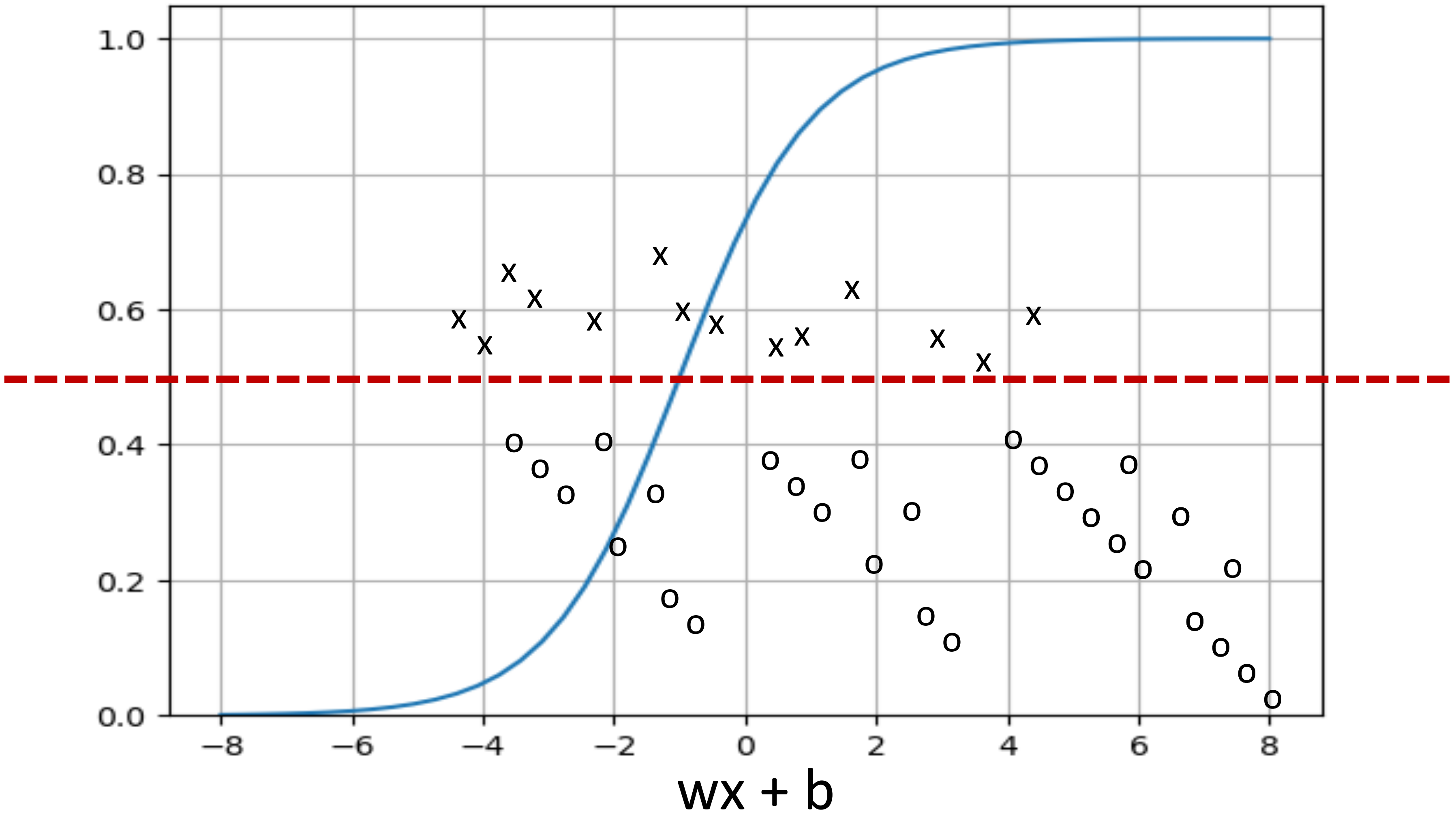
wx + b

Logistic Regression

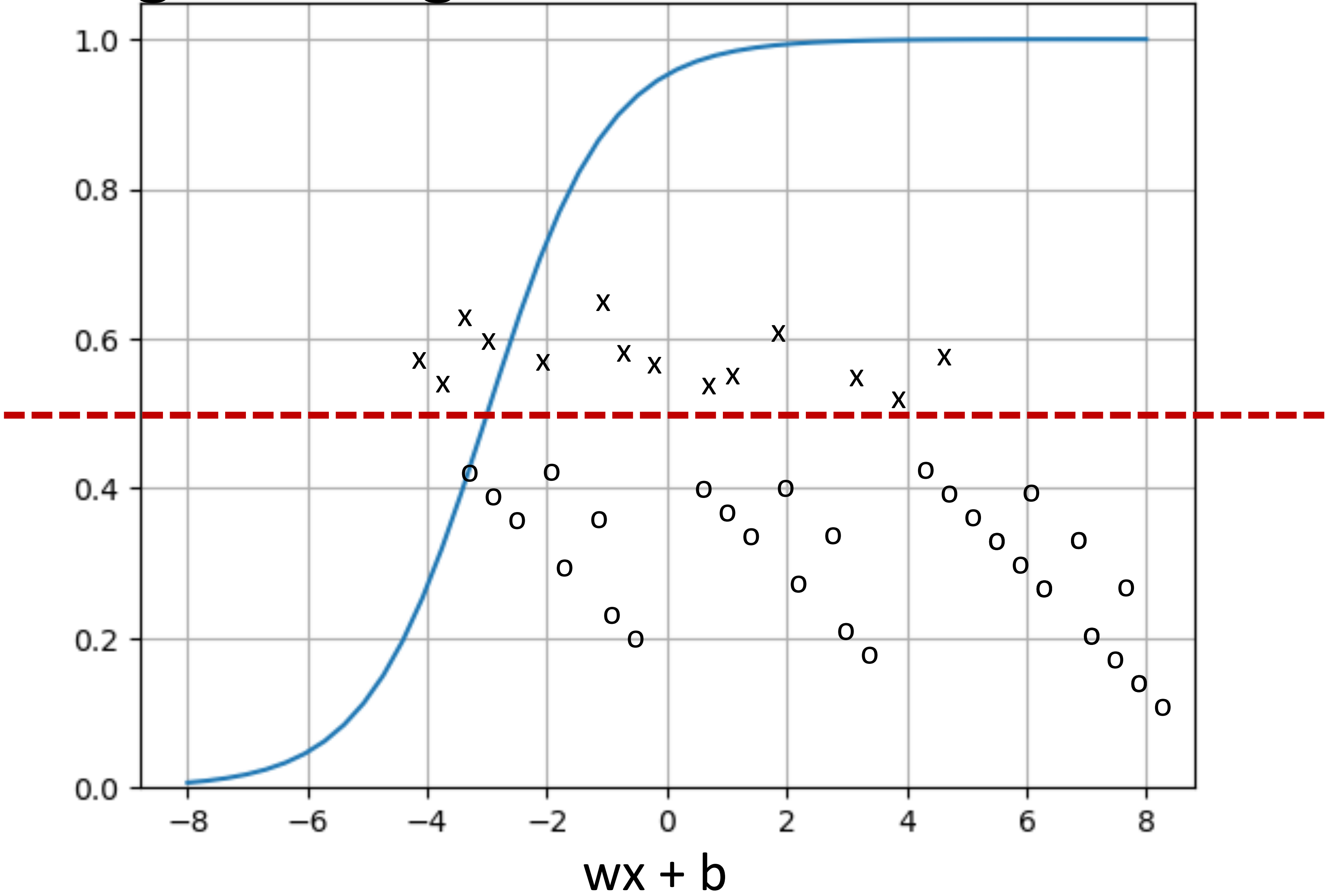


wx + b

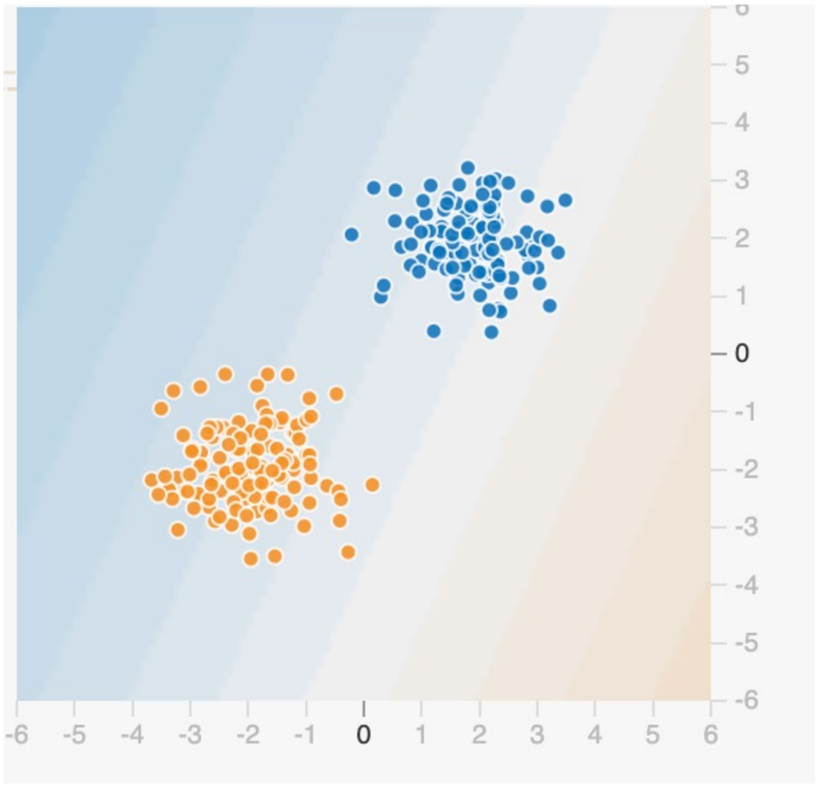
Logistic Regression

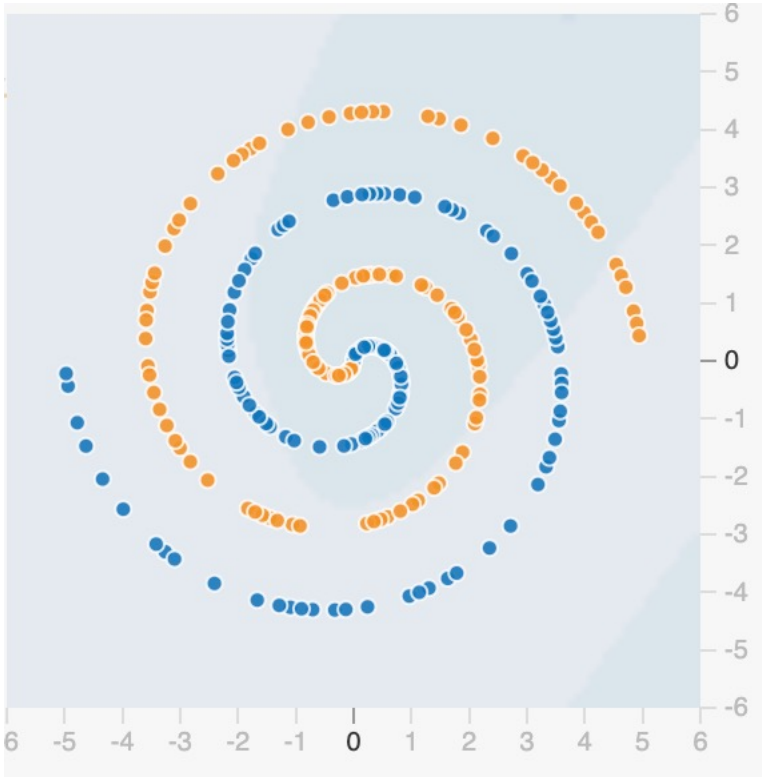


Logistic Regression

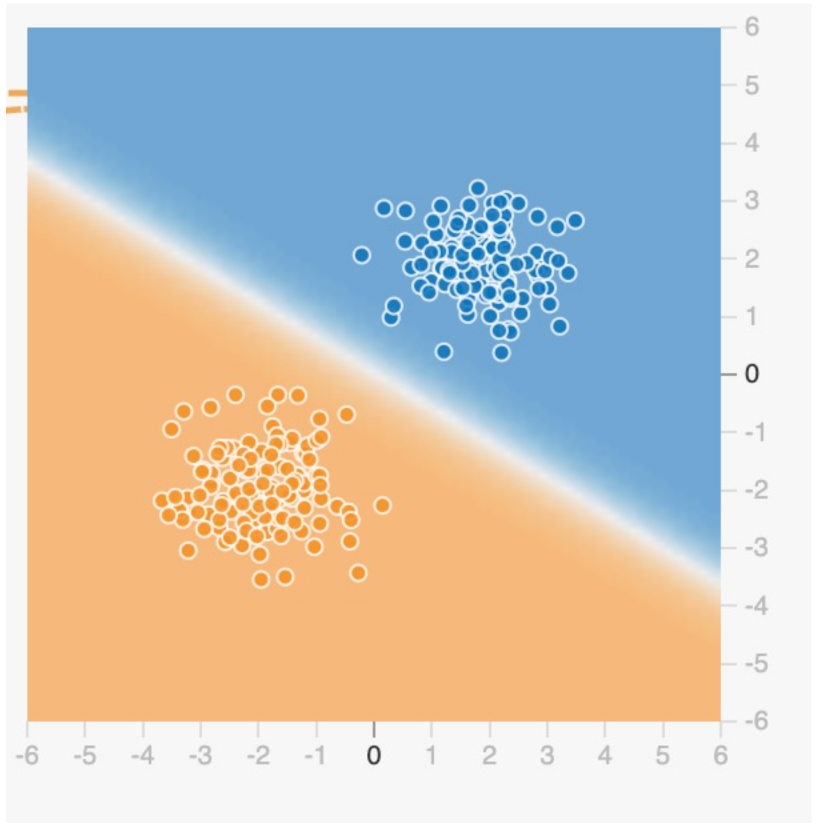


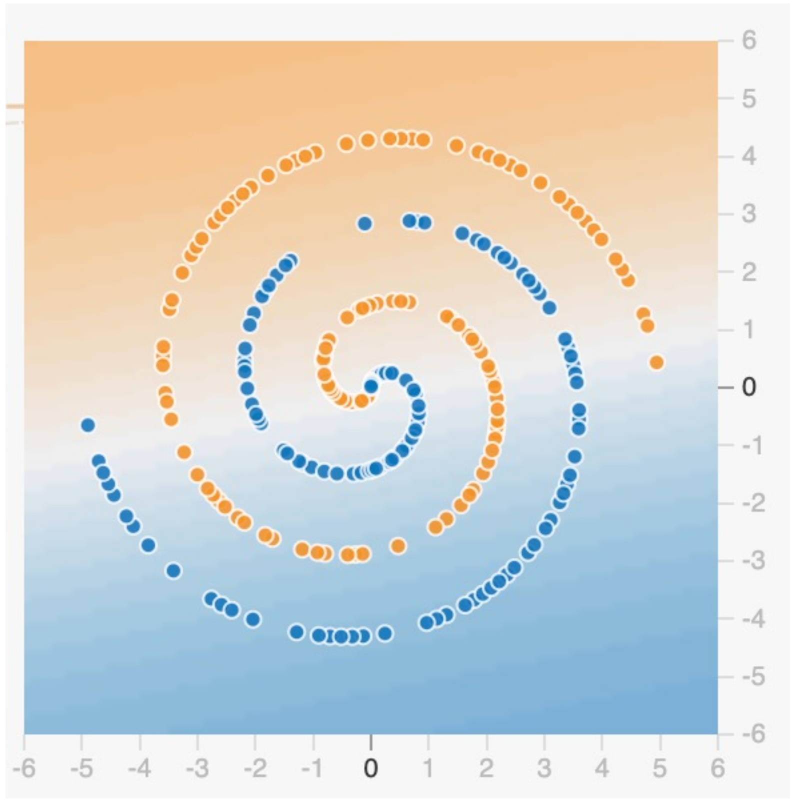
Could we train Logistic Regression on these two training sets?





Training Logistic Regression on these two training sets





Predicting with Logistic Regression

Given $\boldsymbol{x} = [x_1, \quad x_2, \quad x_{3,} \; \dots, x_j]$

Learn weights $\boldsymbol{\beta} = [\beta_1, \quad \beta_2, \quad \beta_{3,} \; \dots, \beta_j]$

Compute a dot product $\boldsymbol{x} \, * \boldsymbol{\beta}$

Dot product is a linear combination

We need to add some non-linearity

Predicting with Logistic Regression

Is sigmoid enough? Does adding sigmoid allows us to model non-linearly separable data?

https://playground.tensorflow.org/