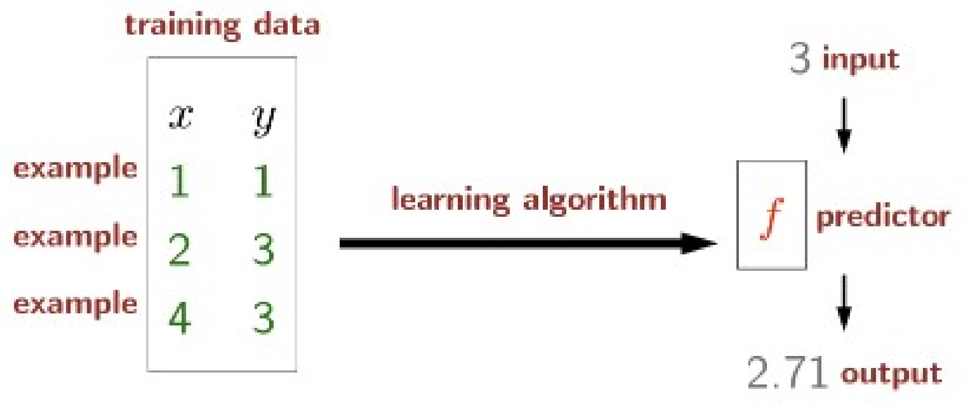
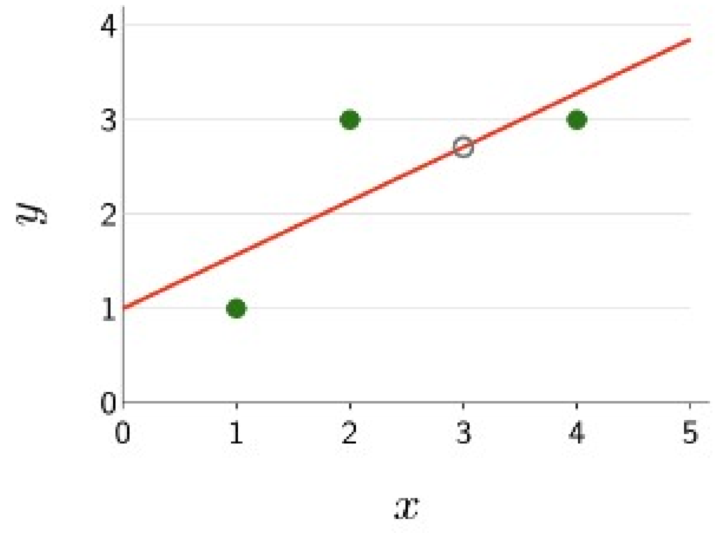
COSC 4368 Fundamentals of Artificial Intelligence

Linear Regression and Linear Classification
September 25th, 2023

Linear Regression





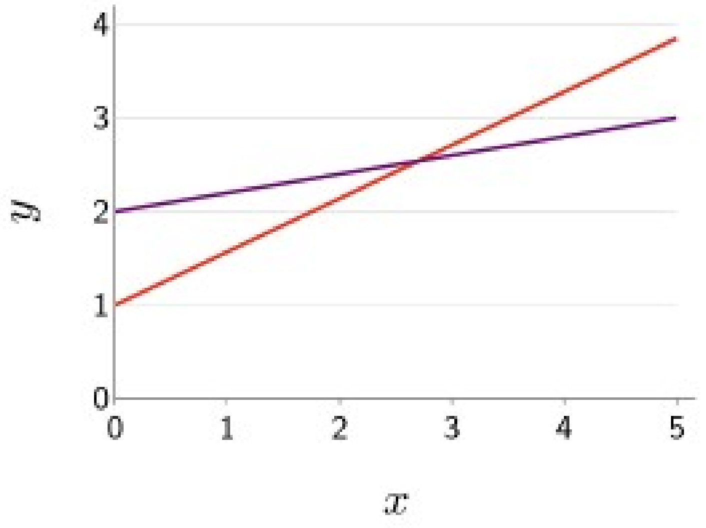
- Design decisions:
  - Which predictors are possible? Hypothesis space
  - How good is a predictor? Loss function
  - How do we compute the best predictor? Optimization algorithm

Hypothesis Space: Which Predictors?

$f(x) = 1 + 0.57x$

$f(x) = 2 + 0.2x$

$f(x) = w_1 + w_2 x$



- Vector notation: weight vector $\mathbf{w} = [w_1, w_2]$ feature extractor $\phi(x) = [1, x]$ feature vector

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$ score

$$f_{\mathbf{w}}(3) = [1, 0.57] \cdot [1, 3] = 2.71$$

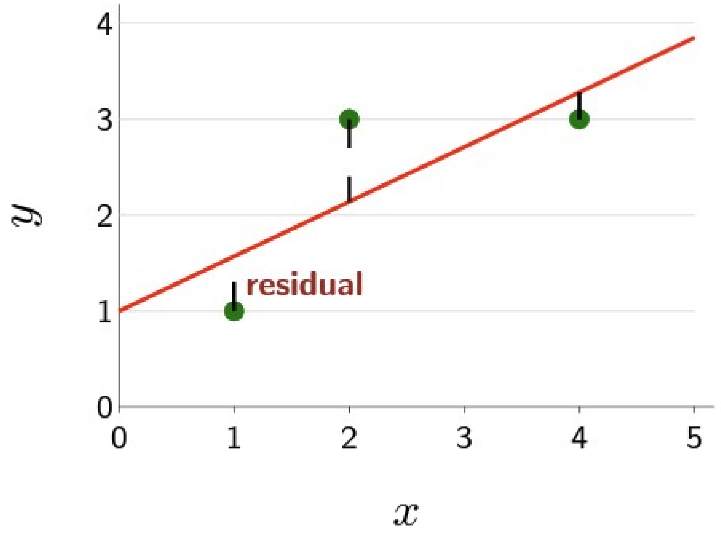
- Hypothesis space: linear functions $\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^2\}$
- Linear regression: the task of finding the best linear function (weights) that best fits the training data

Loss Function: How Good is A Predictor?

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$
$$\mathbf{w} = [1, 0.57]$$
$$\phi(x) = [1, x]$$

training data $\mathcal{D}_{\text{train}}$

x	y
1	1
2	3
4	3



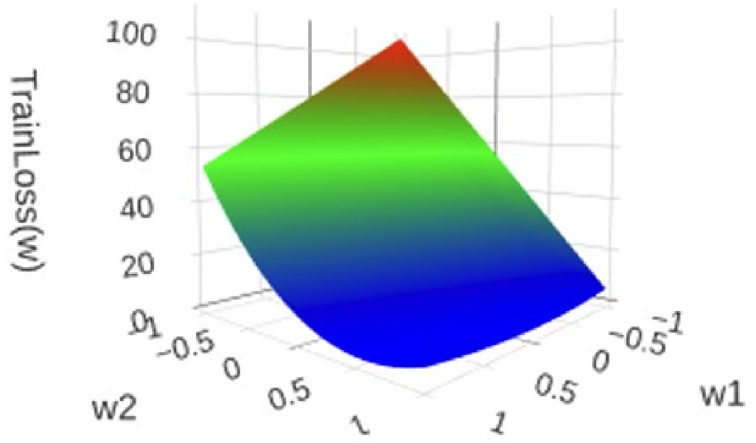
$$\text{Loss}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$$ squared loss

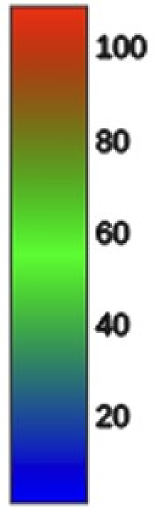
$$\text{Loss}(1, 1, [1, 0.57]) = ([1, 0.57] \cdot [1, 1] - 1)^2 = 0.32$$
$$\text{Loss}(2, 3, [1, 0.57]) = ([1, 0.57] \cdot [1, 2] - 3)^2 = 0.74$$
$$\text{Loss}(4, 3, [1, 0.57]) = ([1, 0.57] \cdot [1, 4] - 3)^2 = 0.08$$

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Loss Function: Visualization

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} (f_{\mathbf{w}}(x) - y)^2$$





Optimization problem:

$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

Closed Form Solution: How to Find the Best

Goal: $\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

is minimized when its partial derivatives w.r.t. and are 0

Suppose N data points in the training set

$$\frac{\partial}{\partial w_1}\sum_{j=1}^{N}(w_1+w_2x_j-y_j)^2=0 \qquad \frac{\partial}{\partial w_2}\sum_{j=1}^{N}(w_1+w_2x_j-y_j)^2=0$$

A unique solution:

$$w_2=\frac{N\left(\sum x_j y_j\right)-\left(\sum x_j\right)\left(\sum y_j\right)}{N\left(\sum x_j^2\right)-\left(\sum x_j\right)^2} \qquad w_1=\frac{\sum y_j-w_2\left(\sum x_j\right)}{N}$$

Optimization: How to Find the Best

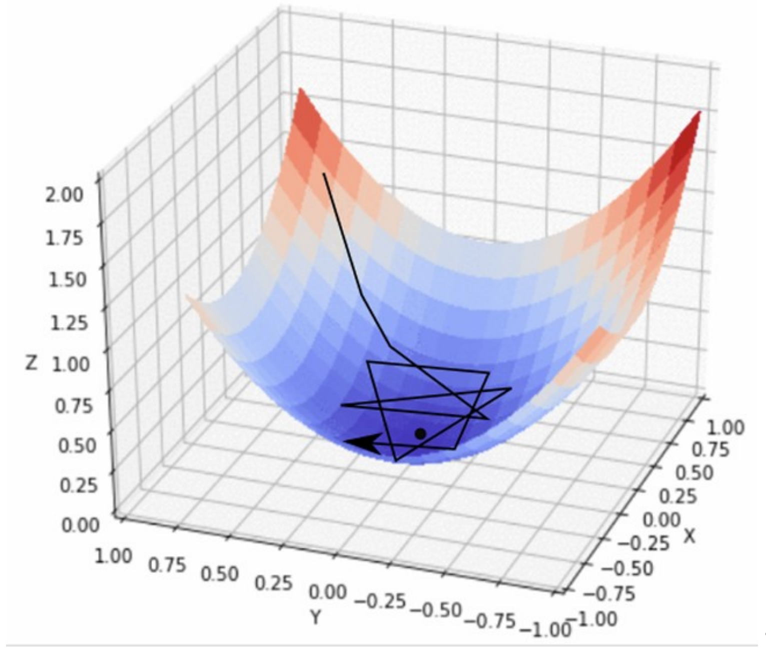
Goal: $\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Iterative algorithms can be used instead of directly deriving the closed form

The gradient is the direction that increases the training loss the most

Algorithm: gradient descent

```
Initialize $\mathbf{w} = [0, \ldots, 0]$
For $t = 1, \ldots, T$: epochs
    $\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$
```



Compute the Gradient

Objective function:

$$\mathsf{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\mathsf{train}}|} \sum_{(x,y)\in\mathcal{D}_{\mathsf{train}}} (\mathbf{w}\cdot\phi(x) - y)^2$$

Gradient (use chain rule):

$$\nabla_{\mathbf{w}}\mathsf{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\mathsf{train}}|} \sum_{(x,y)\in\mathcal{D}_{\mathsf{train}}} 2(\underbrace{\mathbf{w}\cdot\phi(x) - y}_{\mathsf{prediction}-\mathsf{target}})\phi(x)$$

Gradient Example

training data $\mathcal{D}_{\text{train}}$

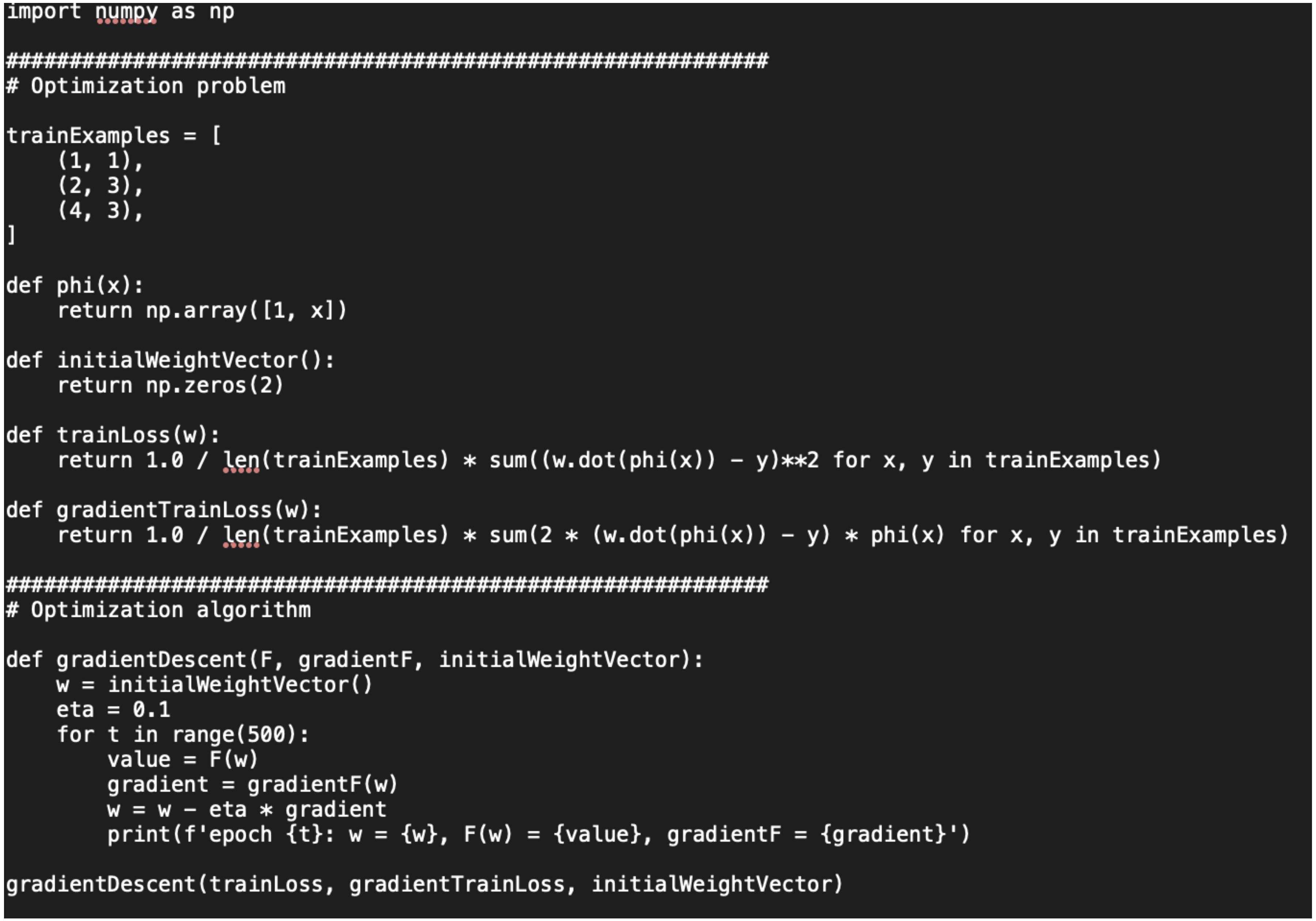
x	y
1	1
2	3
4	3

$$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} 2(\mathbf{w}\cdot\phi(x) - y)\phi(x)$$

Gradient update: $\mathbf{w} \leftarrow \mathbf{w} - 0.1\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

t	$\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$	$\mathbf{w}$
		$[0,0]$
1	$\underbrace{\frac{1}{3}(2([0,0]\cdot[1,1]-1)[1,1] + 2([0,0]\cdot[1,2]-3)[1,2] + 2([0,0]\cdot[1,4]-3)[1,4])}_{=[-4.67,-12.67]}$	$[0.47, 1.27]$
2	$\underbrace{\frac{1}{3}(2([0.47,1.27]\cdot[1,1]-1)[1,1] + 2([0.47,1.27]\cdot[1,2]-3)[1,2] + 2([0.47,1.27]\cdot[1,4]-3)[1,4])}_{=[2.18,7.24]}$	$[0.25, 0.54]$
...	...	...
200	$\underbrace{\frac{1}{3}(2([1,0.57]\cdot[1,1]-1)[1,1] + 2([1,0.57]\cdot[1,2]-3)[1,2] + 2([1,0.57]\cdot[1,4]-3)[1,4])}_{=[0,0]}$	$[1, 0.57]$

Gradient Decent in Python



```python
import numpy as np

############################################################
# Optimization problem

trainExamples = [
    (1, 1),
    (2, 3),
    (4, 3),
]

def phi(x):
    return np.array([1, x])

def initialWeightVector():
    return np.zeros(2)

def trainLoss(w):
    return 1.0 / len(trainExamples) * sum((w.dot(phi(x)) - y)**2 for x, y in trainExamples)

def gradientTrainLoss(w):
    return 1.0 / len(trainExamples) * sum(2 * (w.dot(phi(x)) - y) * phi(x) for x, y in trainExamples)

############################################################
# Optimization algorithm

def gradientDescent(F, gradientF, initialWeightVector):
    w = initialWeightVector()
    eta = 0.1
    for t in range(500):
        value = F(w)
        gradient = gradientF(w)
        w = w - eta * gradient
        print(f'epoch {t}: w = {w}, F(w) = {value}, gradientF = {gradient}')

gradientDescent(trainLoss, gradientTrainLoss, initialWeightVector)
```

Gradient Decent is Slow

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Algorithm: gradient descent

```
Initialize $\mathbf{w} = [0, \dots, 0]$
For $t = 1, \dots, T$: epochs
    $\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$
```

Every iteration requires going well all training samples ---- expensive for a large dataset

Stochastic Gradient Decent

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

Algorithm: stochastic gradient descent

```
Initialize $\mathbf{w} = [0, \ldots, 0]$
For $t = 1, \ldots, T$:
    For $(x, y) \in \mathcal{D}_{\text{train}}$:
        $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$
```

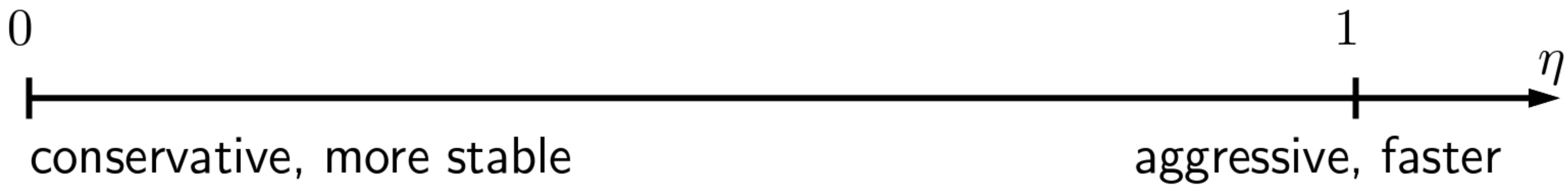
Update the weight based on the gradient with respect to one sample

MiniBatch SGD: each update consists of an averaged gradient over samples

Step Size

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

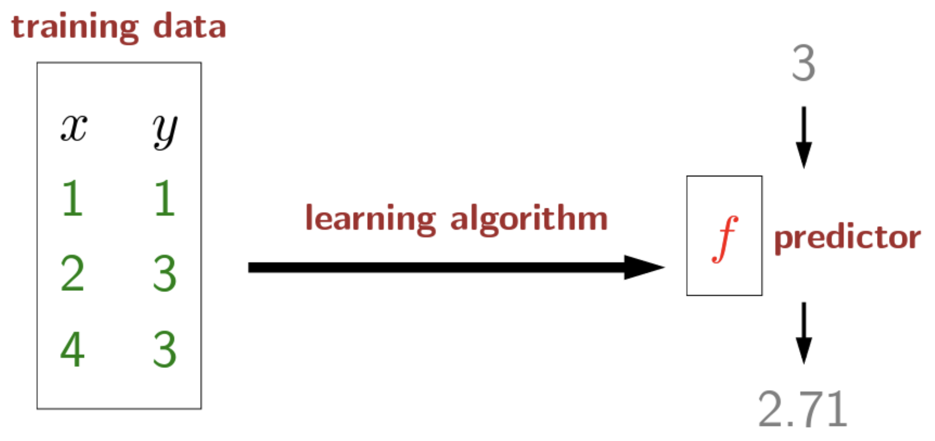
Question: what should η be?



Strategies:

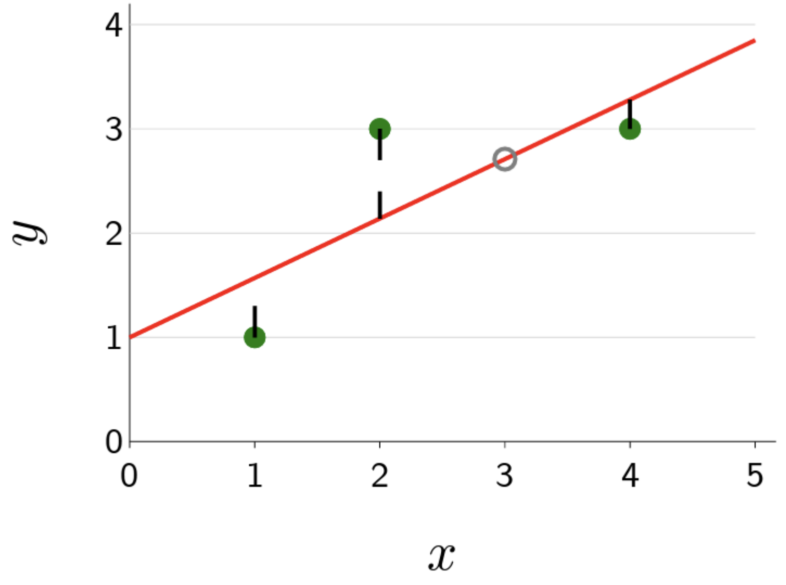
- Constant: $\eta = 0.1$
- Decreasing: $\eta = 1/\sqrt{\text{\# updates made so far}}$

Summary of Linear Regression



x	y
1	1
2	3
4	3

$$\text{training data} \xrightarrow{\text{learning algorithm}} \boxed{f} \text{ predictor}$$



Which predictors are possible? Linear functions

Hypothesis class $\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)\}, \phi(x) = [1, x]$

How good is a predictor? Squared loss

Loss function $\text{Loss}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$

How to compute best predictor? Gradient descent

Optimization algorithm $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \textbf{TrainLoss}(\mathbf{w})$

Linear Classification

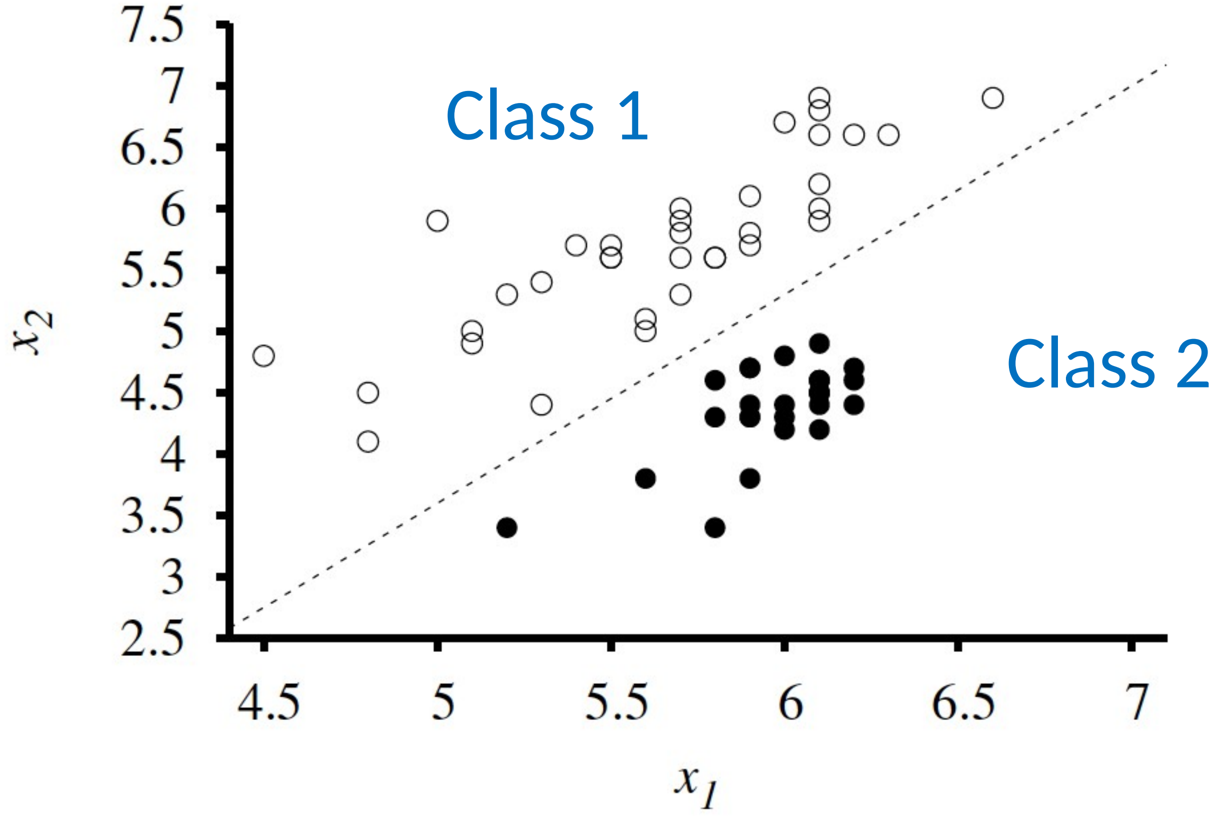
Example 1: image classification

Linear Classification

Example 2: spam detection

	#"$"	#"Mr."	#"sale"	…	Spam?
Email 1	2	1	1		Yes
Email 2	0	1	0		No
Email 3	1	1	1		Yes
…					
Email n	0	0	0		No
New email	0	0	1		??

Linear Classification

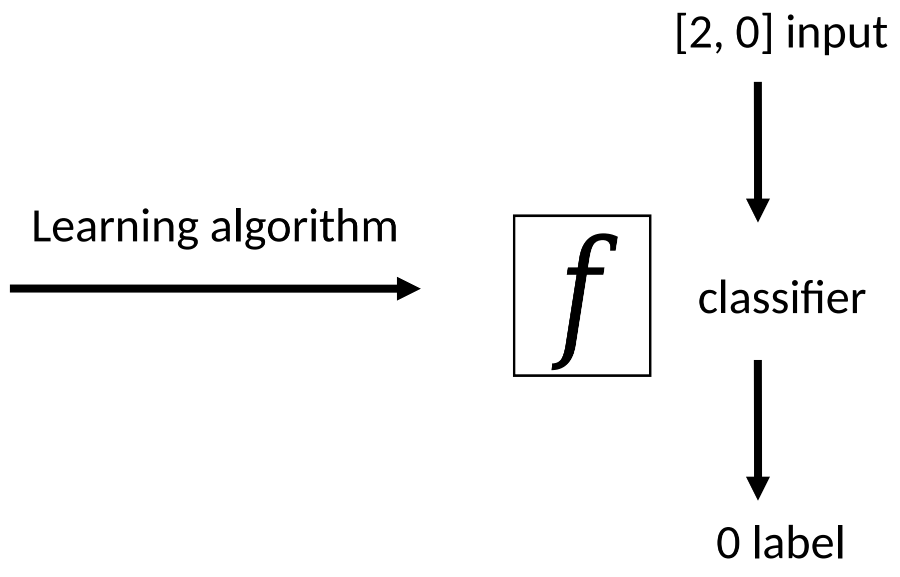


Decision boundary:
A line

Linear Classification

Training data

0	2	1
-2	0	1
1	-1	0



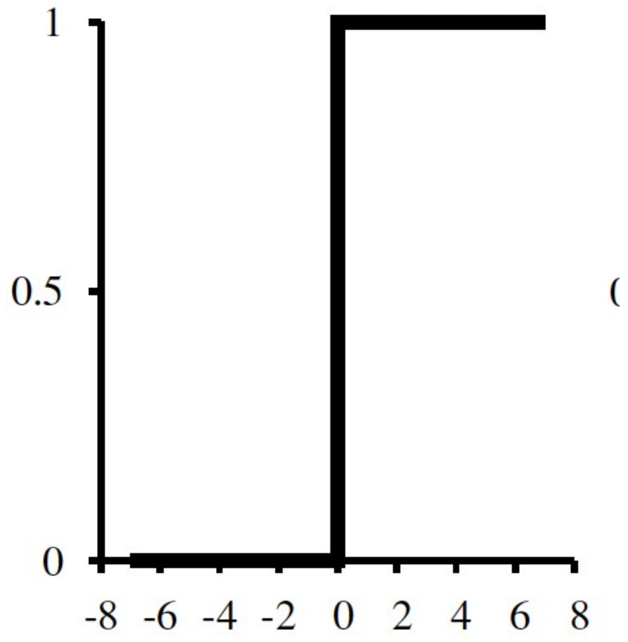
- Design decisions:
  - Which classifiers are possible? Hypothesis space
  - How good is a classifier? Loss function
  - How do we compute the best predictor? Optimization algorithm

Hypothesis Space

- Classification hypothesis:
    - if
    - if

- Think if as the result of passing the linear function through a threshold function:

    - if and 0 otherwise



Loss Function: 0-1 Loss

- Find to minimize

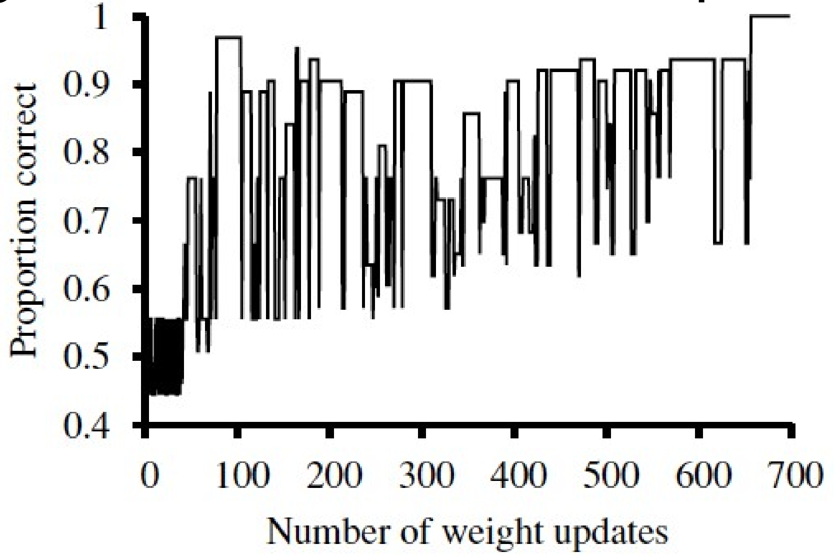
- Drawback: difficult to optimize
  - Gradient is zero almost everywhere in the weight space
  - Not differentiable

Loss Function: Mean Squared Error

- Find to minimize

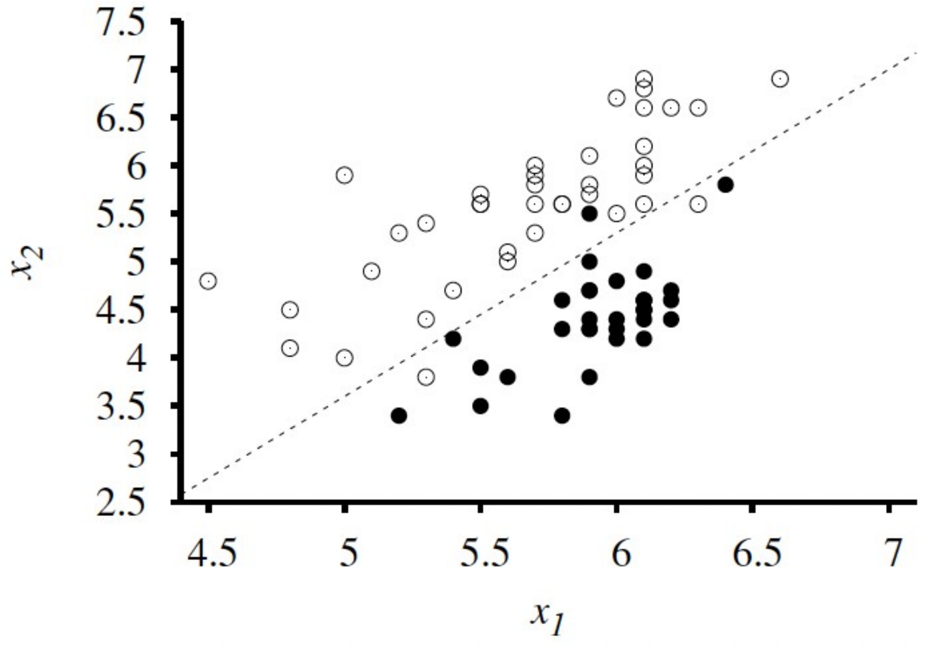
- Reduce to linear regression:
  - Ignore the fact
  - Run gradient descent or stochastic gradient descent

“converge” to a zero-error linear separator

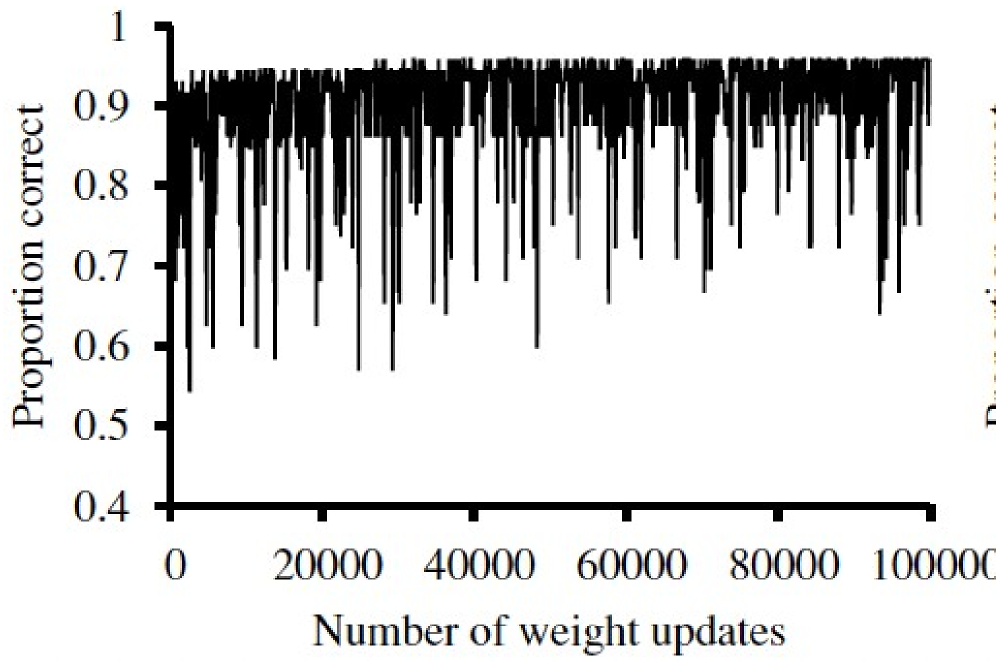


Loss Function: Mean Squared Error

- Works when the data points are linearly separable
- But not robust to "outliers" (not linearly separable)

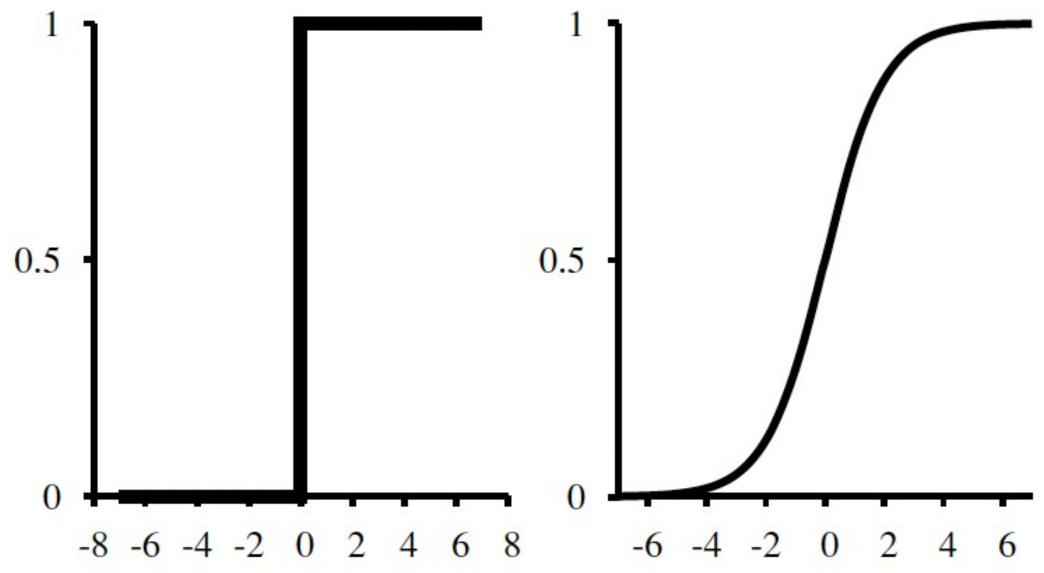


Not converge, keep changing the weights



Linear Classification with Logistic Regression

- Problems stem from the hard nature of the threshold function
- Solution: approximate the hard threshold with a continuous, differentiable function (soft thresholds)
- Logistic (sigmoid) function
  - Smooth



Properties of Sigmoid Function

- Bounded

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \in (0,1)$$

- Symmetric

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

- Gradient

$$\sigma'(a) = \frac{\exp(-a)}{(1 + \exp(-a))^2} = \sigma(a)(1 - \sigma(a))$$

Linear Classification with Logistic Regression

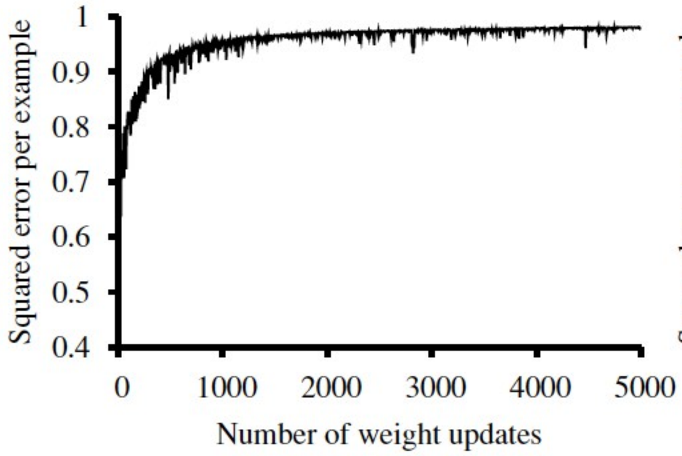
- Now

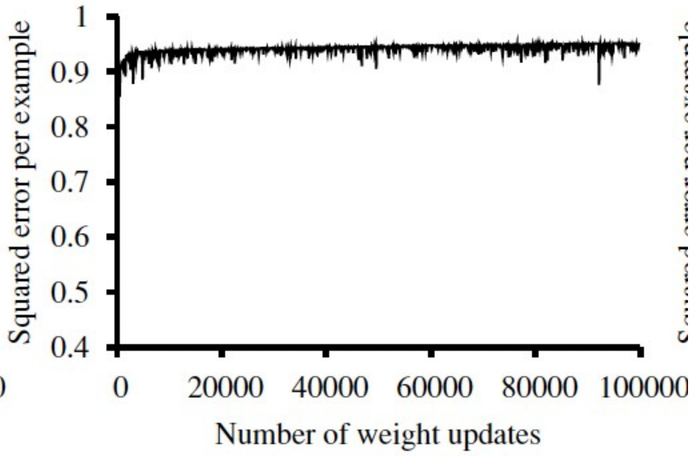
- Output can be interpreted as a probability of belonging to the class labeled 1
  - Gives a probability of 0.5 for any input at the center of the boundary region
  - Approaches 0 or 1 as we move away from the boundary

Linear Classification with Logistic Regression

- Logistic regression: find to minimize

- Run GD/SGD





Linearly separable:
slower

Non-separable:
Faster and more stable