Roadmap

Linear predictors

Loss minimization

Stochastic gradient descent

Application: spam classification

Input: $x =$ email message

```
From: pliang@cs.stanford.edu
Date: September 27, 2017
Subject: CS221 announcement

Hello students,
 I've attached the answers to homework 1...
```

```
From: a9k62n@hotmail.com
Date: September 27, 2017
Subject: URGENT

Dear Sir or maDam:
 my friend left sum of 10m dollars...
```

Output: $y \in \{\text{spam}, \text{not-spam}\}$

Objective: obtain a **predictor** f

$$x \longrightarrow \boxed{f} \longrightarrow y$$

Types of prediction tasks

Binary classification (e.g., email $\Rightarrow$ spam/not spam):

$$x \longrightarrow \boxed{f} \longrightarrow y \in \{-1, +1\}$$

Regression (e.g., location, year $\Rightarrow$ housing price):

$$x \longrightarrow \boxed{f} \longrightarrow y \in \mathbb{R}$$

Data

Example: specifies that y is the ground-truth output for x

$$(x, y)$$

Training data: list of examples

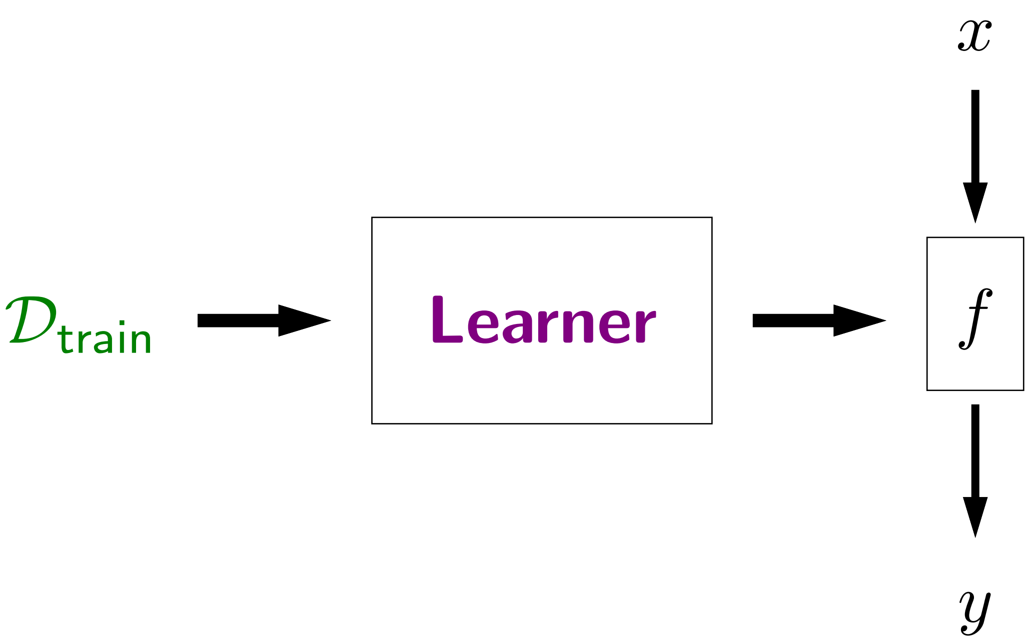
$$\mathcal{D}_{\text{train}} = [$$

("...10m dollars...",$+1$),

("...CS221...", -1),

]

Framework



Feature extraction

Example task: predict y, whether a string x is an email address

Question: what properties of x **might be** relevant for predicting y?

Feature extractor: Given input x, output a set of (feature name, feature value) pairs.

"abc@gmail.com" $\xrightarrow[\text{arbitrary!}]{\textbf{feature extractor}}$

length>10	: 1
fracOfAlpha	: 0.85
contains_@	: 1
endsWith_.com	: 1
endsWith_.org	: 0

Feature vector notation

Mathematically, feature vector doesn't need feature names:

length>10	: 1
fracOfAlpha	: 0.85
contains_@	: 1
endsWith_.com	: 1
endsWith_.org	: 0

$$\rightarrow \begin{bmatrix} 1 \\ 0.85 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Definition: feature vector

For an input x, its feature vector is:

$$\phi(x) = [\phi_1(x), \dots, \phi_d(x)].$$

Think of $\phi(x) \in \mathbb{R}^d$ as a point in a high-dimensional space.

Weight vector

Weight vector: for each feature j, have real number w_j representing contribution of feature to prediction

length>10	:-1.2
fracOfAlpha	:0.6
contains_@	:3
endsWith_.com	:2.2
endsWith_.org	:1.4
...	

Linear predictors

Weight vector $\mathbf{w} \in \mathbb{R}^d$

length>10	:-1.2
fracOfAlpha	:0.6
contains_@	:3
endsWith_.com	:2.2
endsWith_.org	:1.4

Feature vector $\phi(x) \in \mathbb{R}^d$

length>10	:1
fracOfAlpha	:0.85
contains_@	:1
endsWith_.com	:1
endsWith_.org	:0

Score: weighted combination of features

$$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi(x)_j$$

Example: $-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$

Linear predictors

Weight vector $\mathbf{w} \in \mathbb{R}^d$

Feature vector $\phi(x) \in \mathbb{R}^d$

For binary classification:

Definition: (binary) linear classifier

$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x)) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \phi(x) > 0 \\ -1 & \text{if } \mathbf{w} \cdot \phi(x) < 0 \\ ? & \text{if } \mathbf{w} \cdot \phi(x) = 0 \end{cases}$$

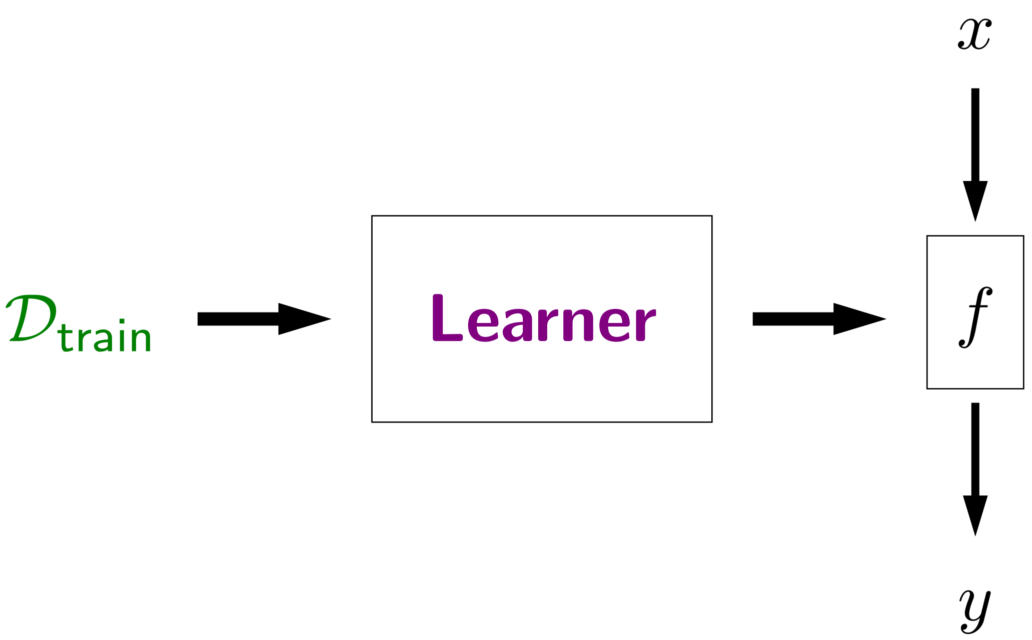
Roadmap

Linear predictors

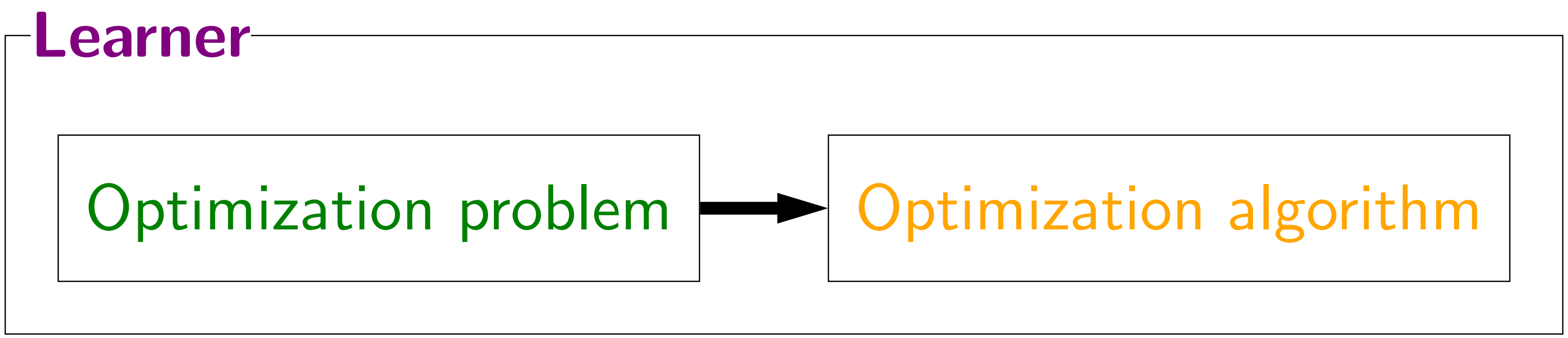
Loss minimization

Stochastic gradient descent

Framework



x
$\mathcal{D}_{\text{train}}$
Learner
f
y



Learner
Optimization problem
Optimization algorithm

Loss functions

Definition: loss function

A loss function $\text{Loss}(x, y, \mathbf{w})$ quantifies how unhappy you would be if you used $\mathbf{w}$ to make a prediction on x when the correct output is y. It is the object we want to minimize.

Score and margin

Correct label: y

Predicted label: $y' = f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$

Example: $\mathbf{w} = [2, -1], \phi(x) = [2, 0], y = -1$

Definition: score

The score on an example (x, y) is $\mathbf{w} \cdot \phi(x)$, how **confident** we are in predicting $+1$.

Definition: margin

The margin on an example (x, y) is $(\mathbf{w} \cdot \phi(x))y$, how **correct** we are.

cs221.stanford.edu/q

Question

When does a binary classifier err on an example?

- margin less than 0
- margin greater than 0
- score less than 0
- score greater than 0

Binary classification

Example: $\mathbf{w} = [2, -1], \phi(x) = [2, 0], y = -1$

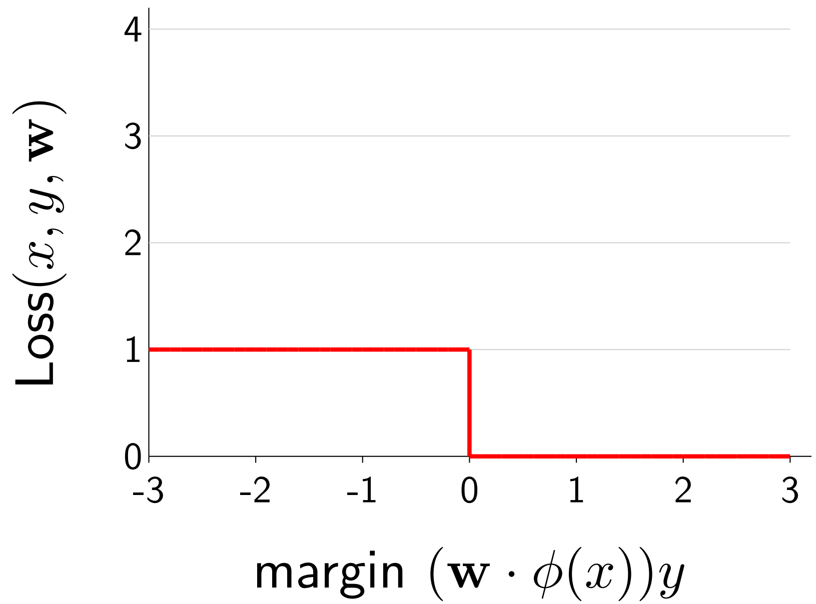
Recall the binary classifier:

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

Definition: zero-one loss

$$\begin{aligned}\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) &= \mathbf{1}[f_{\mathbf{w}}(x) \neq y] \\ &= \mathbf{1}[\underbrace{(\mathbf{w} \cdot \phi(x))y}_{\text{margin}} \leq 0]\end{aligned}$$

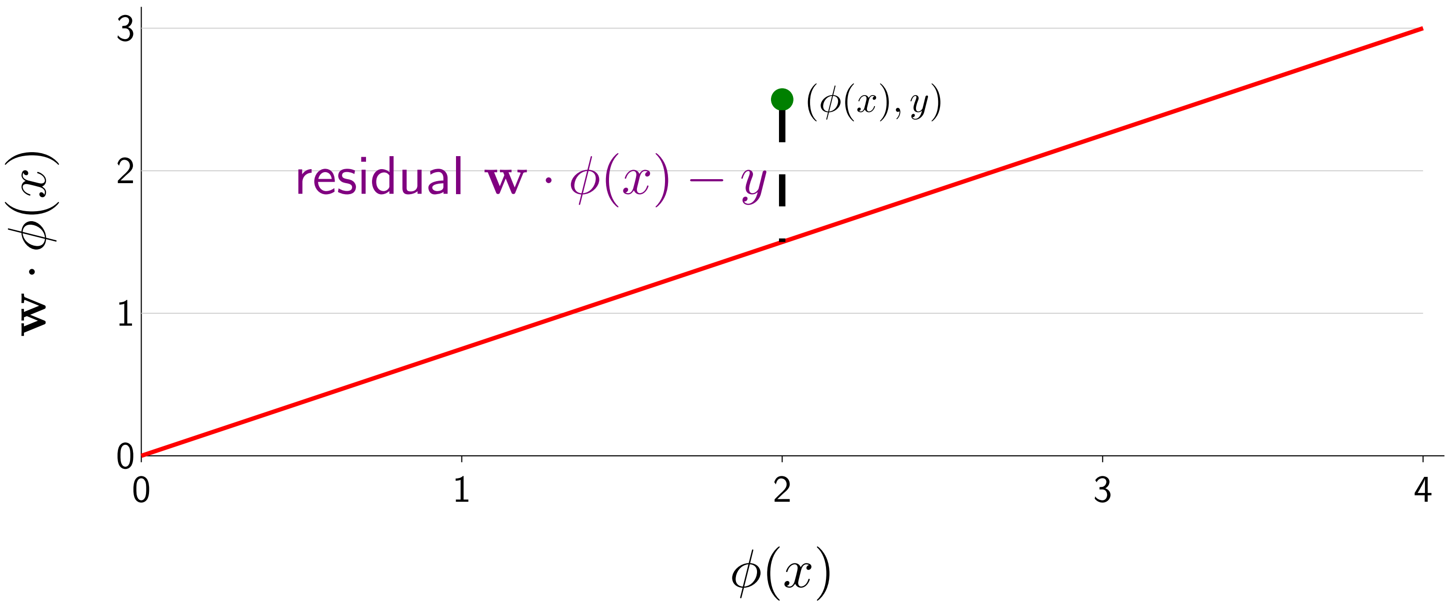
Binary classification



$$\text{Loss}_{\text{0-1}}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \le 0]$$

Linear regression

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



Definition: residual

The **residual** is $(\mathbf{w} \cdot \phi(x)) - y$, the amount by which prediction $f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$ overshoots the target y.

Linear regression

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

Definition: squared loss

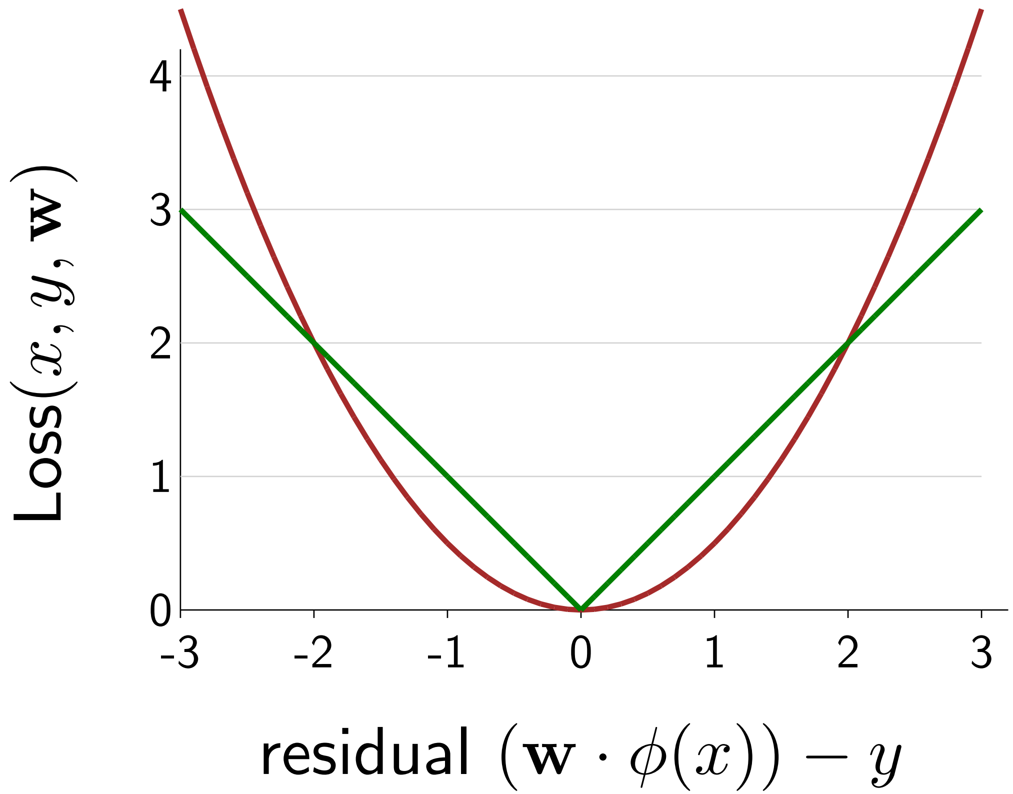
$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\underbrace{f_{\mathbf{w}}(x) - y}_{\text{residual}})^2$$

Example:

$$\mathbf{w} = [2, -1], \phi(x) = [2, 0], y = -1$$

$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = 25$$

Regression loss functions



$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\mathbf{w} \cdot \phi(x) - y)^2$$

$$\text{Loss}_{\text{absdev}}(x, y, \mathbf{w}) = |\mathbf{w} \cdot \phi(x) - y|$$

Loss minimization framework

So far: one example, $\text{Loss}(x, y, \mathbf{w})$ is easy to minimize.

Key idea: minimize training loss

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

$$\min_{\mathbf{w} \in \mathbb{R}^d} \text{TrainLoss}(\mathbf{w})$$

Key: need to set $\mathbf{w}$ to make global tradeoffs — not every example can be happy.

Which regression loss to use?

Example: $\mathcal{D}_{\text{train}} = \{(1,0),(1,2),(1,1000)\}$

For least squares (L_2) regression:

$$\text{Loss}_{\text{squared}}(x,y,\mathbf{w}) = (\mathbf{w}\cdot\phi(x) - y)^2$$

- $\mathbf{w}$ that minimizes training loss is **mean** y
- Mean: tries to accommodate every example, popular

For least absolute deviation (L_1) regression:

$$\text{Loss}_{\text{absdev}}(x,y,\mathbf{w}) = |\mathbf{w}\cdot\phi(x) - y|$$

- $\mathbf{w}$ that minimizes training loss is **median** y
- Median: more robust to outliers

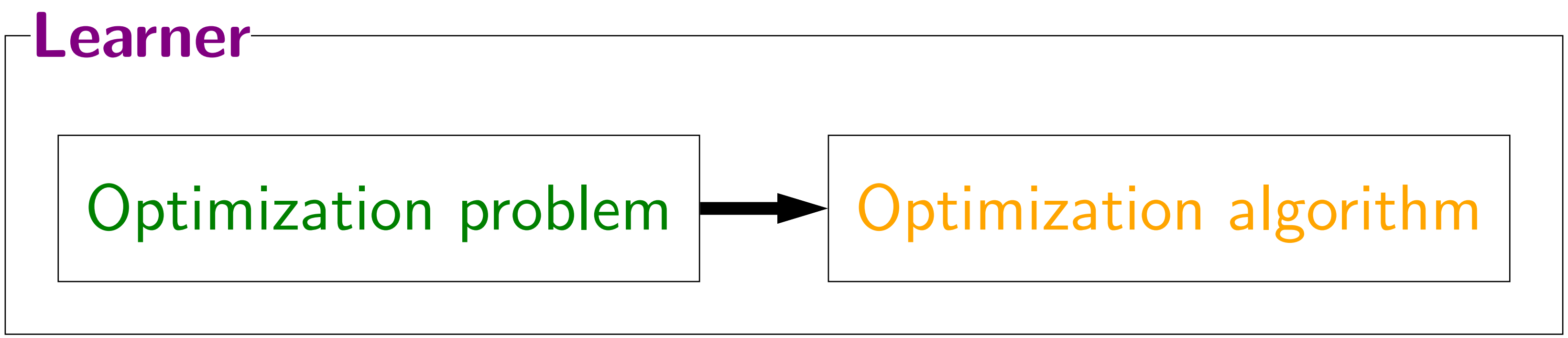
Roadmap

Linear predictors

Loss minimization

Stochastic gradient descent

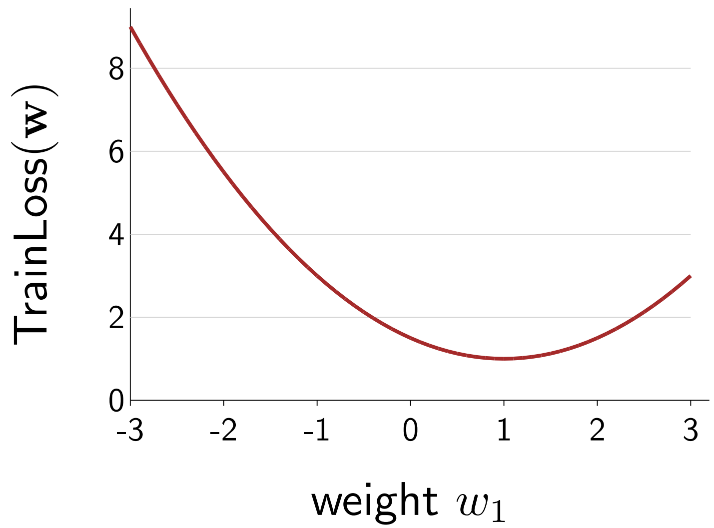
Learning as optimization



Optimization problem

Objective: $\min_{\mathbf{w} \in \mathbb{R}^d} \text{TrainLoss}(\mathbf{w})$

$\mathbf{w} \in \mathbb{R}$



$\mathbf{w} \in \mathbb{R}^2$

[gradient plot]

How to optimize?

Definition: gradient

The gradient $\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$ is the direction that increases the loss the most.

Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \dots, 0]$

For $t = 1, \dots, T$:

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

Least squares regression

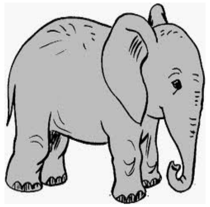
Objective function:

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

Gradient (use chain rule):

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

[semi-live solution]



Gradient descent is slow

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

Gradient descent:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

Problem: each iteration requires going over all training examples — expensive when have lots of data!

Stochastic gradient descent

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

Gradient descent (GD):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$$

Stochastic gradient descent (SGD):

For each $(x, y) \in \mathcal{D}_{\text{train}}$:

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

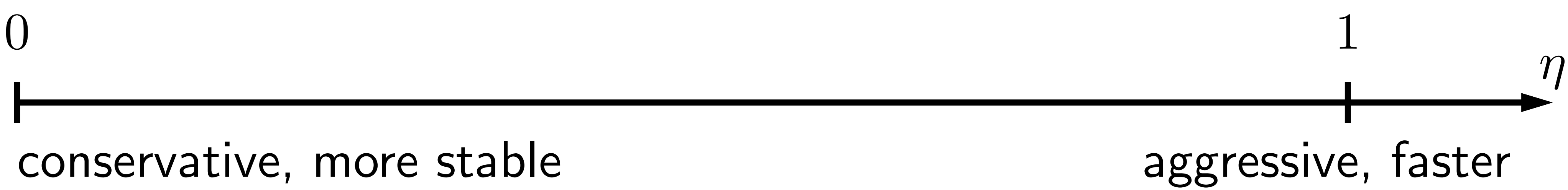
Key idea: stochastic updates

It's not about **quality**, it's about **quantity**.

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 so far

Linear predictors:

$$f_{\mathbf{w}}(x) \text{ based on score } \mathbf{w} \cdot \phi(x)$$

Loss minimization: learning as optimization

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

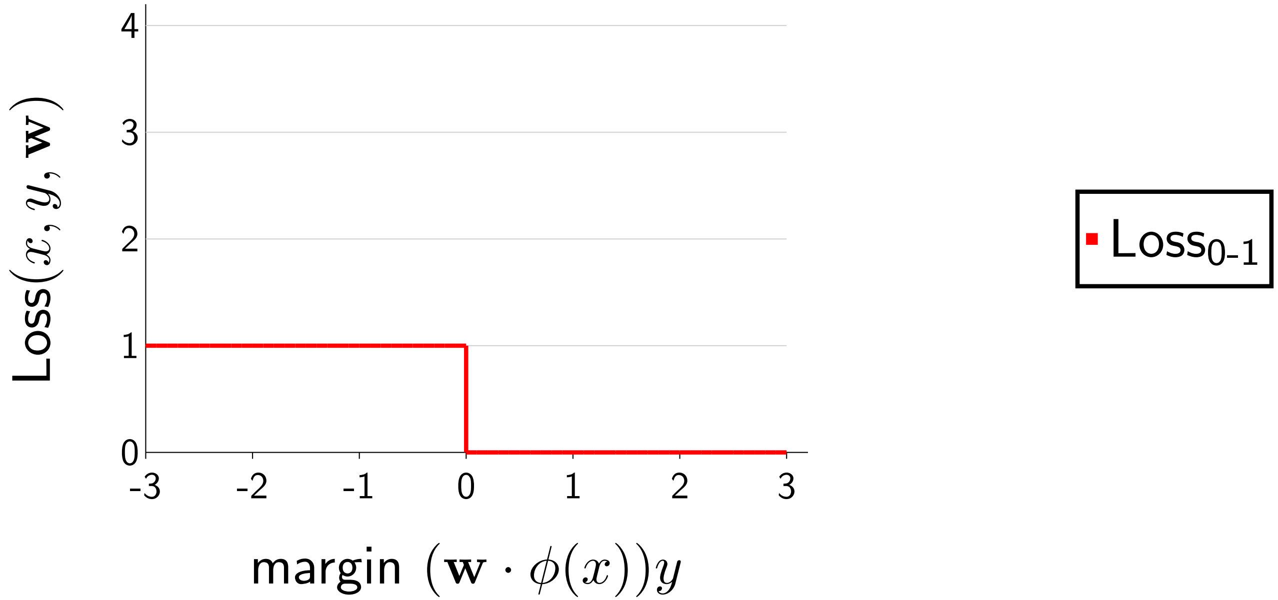
Stochastic gradient descent: optimization algorithm

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathsf{Loss}(x, y, \mathbf{w})$$

Done for linear regression; what about classification?

Zero-one loss

$$\text{Loss}_{0\text{-}1}(x, y, \mathbf{w}) = \mathbf{1}[(\mathbf{w} \cdot \phi(x))y \leq 0]$$



Problems:

- Gradient of $\text{Loss}_{0\text{-}1}$ is 0 everywhere, SGD not applicable
- $\text{Loss}_{0\text{-}1}$ is insensitive to how badly model messed up