Linear Predictors

Linear Predictors

- The simplest yet most practical tool
- Cover both
  - classification and
  - regression and
  - are examples of reflex models.
- by formulating an optimization problem based on the loss minimization framework.
- Finally, we will discuss gradient descent, an efficient algorithm for optimizing (that is, minimizing) the loss that’s tailored for machine learning

Roadmap

Linear predictors

Loss minimization

Stochastic gradient descent

Application: spam classification

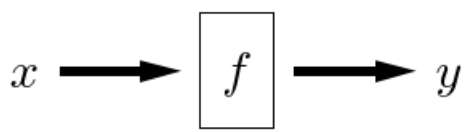
Input: x = email message

```
From:  pliang@cs.stanford.edu
Date:  September 25, 2019
Subject:  CS221 announcement

Hello students,
 Welcome to CS221!  Here's what...
```

```
From:  a9k62n@hotmail.com
Date:  September 25, 2019
Subject:  URGENT

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

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

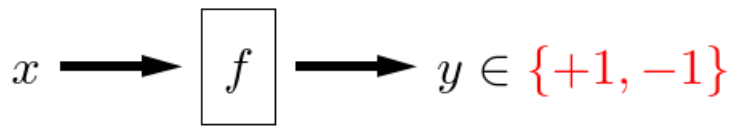
Objective: obtain a **predictor** f



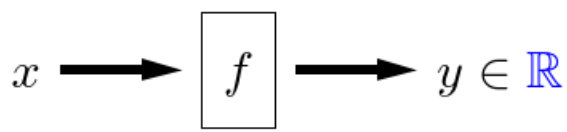
A predictor is a function f that maps an input x to an output y (also called a model or a hypothesis).

Types of prediction tasks

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

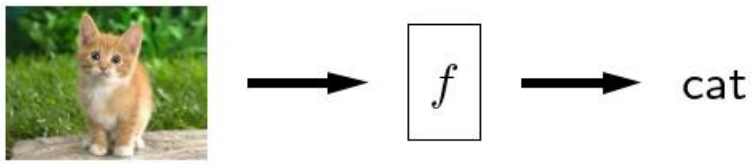


Regression (e.g., location, year $\Rightarrow$ housing price):

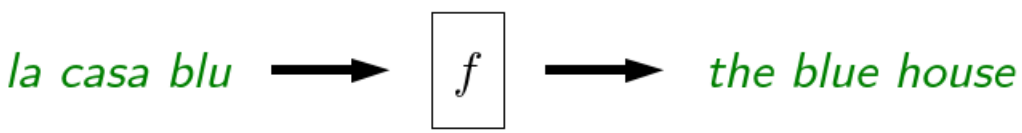


Types of prediction tasks

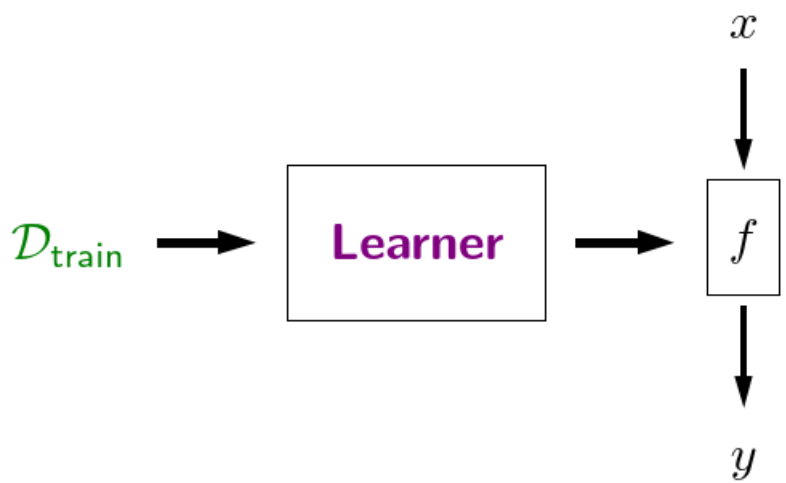
Multiclass classification: y is a category



Structured prediction: y is an object which is built from parts



Framework



Training Data (D_{train})

- The starting point of machine learning is the data.
- **supervised learning**, in which our **data provides both inputs and outputs**,
- in contrast to **unsupervised learning**, which **only provides inputs**.
  - Clustering
  - Association Rules
- A (supervised) **example** (also called a data point or instance) is simply an **input-output pair (x, y)**, which **specifies that y** is the ground-truth output for x.
- The training data D_{train} is a multiset of examples that forms a partial specification of the desired behavior of a predictor.

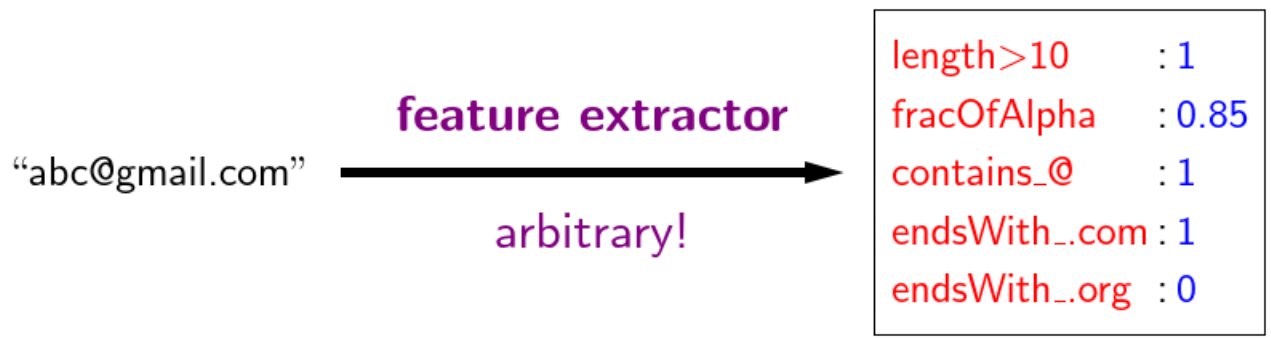
- Learning is about
  - taking the training data D_{train} and
  - producing a predictor f, which is a function that takes inputs x and tries to map them to outputs y = f(x).
- We want the predictor to approximately **work even for examples that we have not seen in D_{train}.**
- We will first focus on examining **what f is**, independent of how the learning works.
- Then we **will come back to learning f** based on data.

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.



Feature extraction

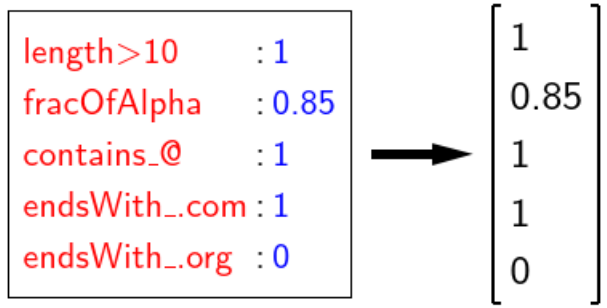
- We will consider **predictors f based on feature extractors**.

- **a bit of an art that requires intuition** about both
  - the task (what we are trying to learn) and also
  - what machine learning algorithms **are capable of**.

- The general principle is that features should **represent properties of x** which might be **relevant for predicting y**.
- It is **okay to add features which turn out to be irrelevant**, since the learning algorithm can sort it out (not all algorithms, NB and kNN do not)
- But it might require **more data** to do so.

Feature vector notation

Mathematically, feature vector doesn't need feature names:



Definition: feature vector

For an input x, its feature vector is:

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

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
...	

- So far, we have defined
  - a feature extractor ϕ that maps each input x to the feature vector $\phi(x)$.
  - A weight vector $w= [w_1, \ldots, w_d]$ (also called a parameter vector or weights) specifies the contributions of each feature vector to the prediction.
- In the context of binary classification with **binary features** ($\phi_j(x) \in \{0, 1\}$), the weights $w_j \in R$ have an **intuitive interpretation**.
  - **If w_j is positive**, then the presence of feature j ($\phi_j(x) = 1$) favors a positive classification.
  - Conversely, **if w_j is negative**, then the presence of feature j favors a negative classification.

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

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

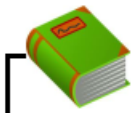
Example: $-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$

Linear predictors

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

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

For binary classification:

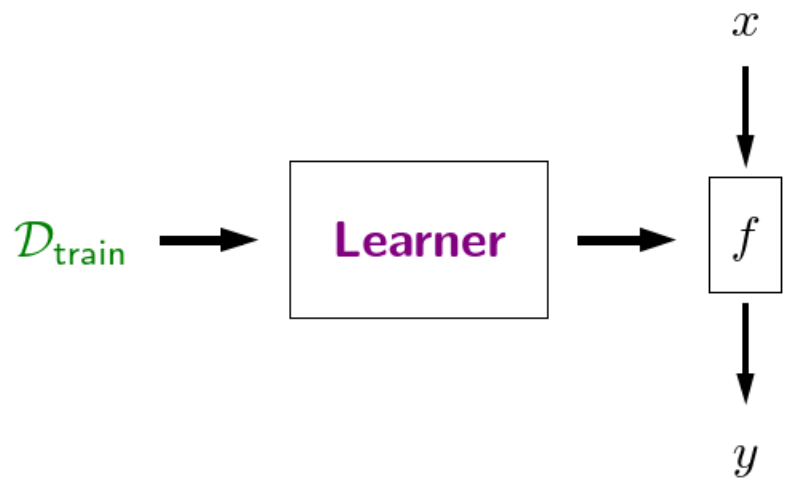


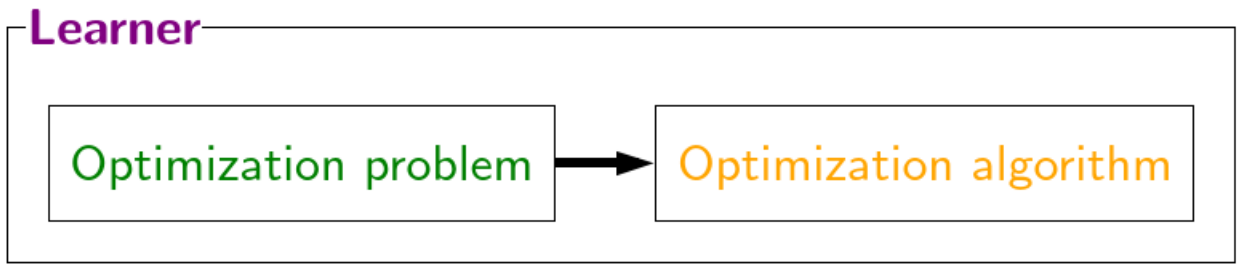
Definition: (binary) linear classifier

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

The case of $f_w(x) =?$ is a boundary case that isn't so important.
We can just predict +1 arbitrarily as a matter of convention.

Framework

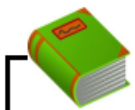

x
$\mathcal{D}_{\text{train}}$
Learner
f
y


Learner
Optimization problem
Optimization algorithm

Learning as Optimization

- So far we have talked about linear predictors f_w which are based on a feature extractor ϕ and a weight vector w.
- Now we turn to **the problem of estimating** w from training data
  - also known as **fitting or learning**.
- The loss minimization framework is to cast learning as an optimization problem.
- Note the theme of **separating our problem** into
  - a model (optimization problem) and
  - an algorithm (optimization algorithm).

Loss functions



Definition: loss function

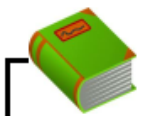
A loss function $\text{Loss}(x, y, \mathbf{w})$ quantifies how unhappy you would be if you used $\mathbf{w}$ to make a prediction on x when the correct output is y. It is the object we want to minimize.

Score and margin

Correct label: y

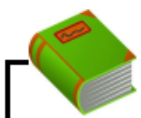
Predicted label: $y' = f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$

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.

Margin

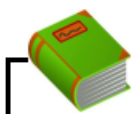
- Suppose the correct label is $y \in \{-1, +1\}$.
- The margin of an input x is $w \cdot \phi(x)\, y$
- measures **how correct the prediction that w makes is**.
- The larger the margin the better,
- and non-positive margins correspond to classification errors.

Binary classification

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

Recall the binary classifier:

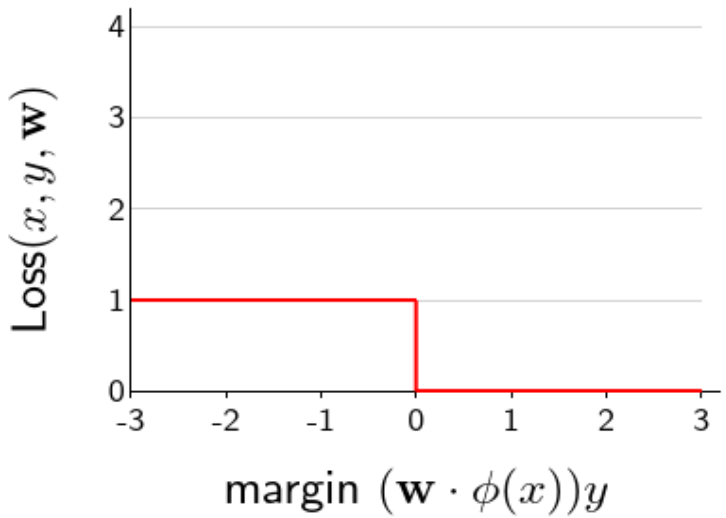
$$f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))$$



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



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

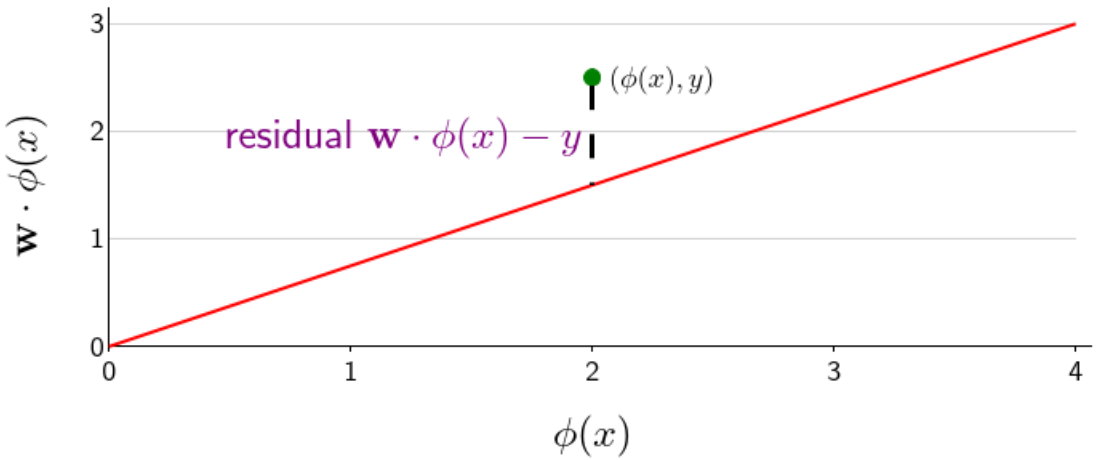
We can plot the loss as a function of the margin. From the graph, it is clear that the loss is 1 when the margin is negative and 0 when it is positive.

Regression

- Now let's turn for a moment to regression, where the output y is a real number rather than {−1, +1}.
- Here, the zero-one loss doesn't make sense, because it's unlikely that we're going to predict y exactly.
- Let's instead define the residual to measure how close the prediction $f_w(x)$ is to the correct y.
- The residual will play the analogous role of the margin for classification and will let us craft an appropriate loss function.

Linear regression

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



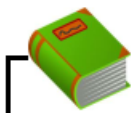
Definition: residual

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.

Here, the zero-one loss doesn't make sense, because it's unlikely that we're going to predict y exactly. So we will use a function based on residuals instead.

Linear regression

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



Definition: squared loss

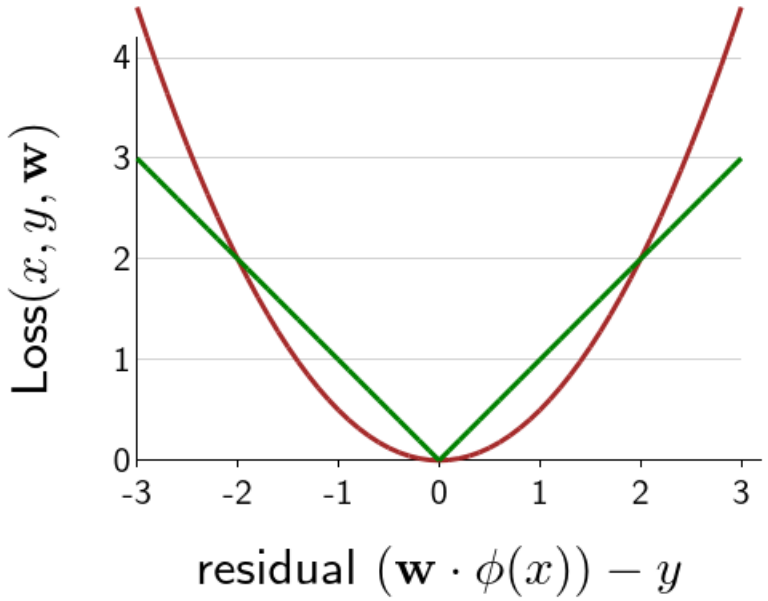
$$\text{Loss}_{\text{squared}}(x, y, \mathbf{w}) = (\underbrace{f_{\mathbf{w}}(x) - y}_{\text{residual}})^2$$

Example:

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

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

Regression loss functions



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

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

- A popular and convenient loss function to use in linear regression is the squared loss, which penalizes the residual of the prediction quadratically.
- If the predictor is off by a residual of 10, then the loss will be 100.
- An alternative to the squared loss is the absolute deviation loss, which simply takes the absolute value of the residual.

Loss minimization framework

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

Key idea: minimize training loss

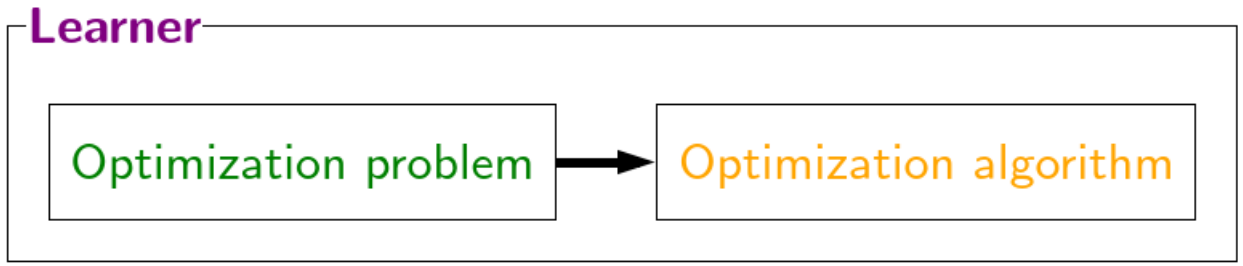
$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$

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

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

- Note that on one example, **both the squared and absolute deviation loss functions have the same minimum**,
  - so we cannot really appreciate the differences here.
- However, we are learning w based on a whole training set D_{train}, not just one example.
- We typically **minimize the training loss** also known as
  - the training error or
  - empirical risk.
- which is the average loss over all the training examples.
- Importantly, such an optimization problem **requires making trade offs across all the examples**.
  - i.e. we won't be able to set w to a single value that makes every example have low loss).

Learning as optimization

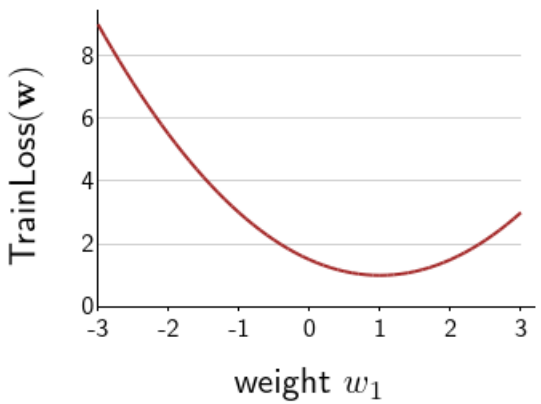


- The problem becomes a Local Search in Continuous Spaces
- ➔ The Gradient Descent algorithm can be used.

Optimization problem

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

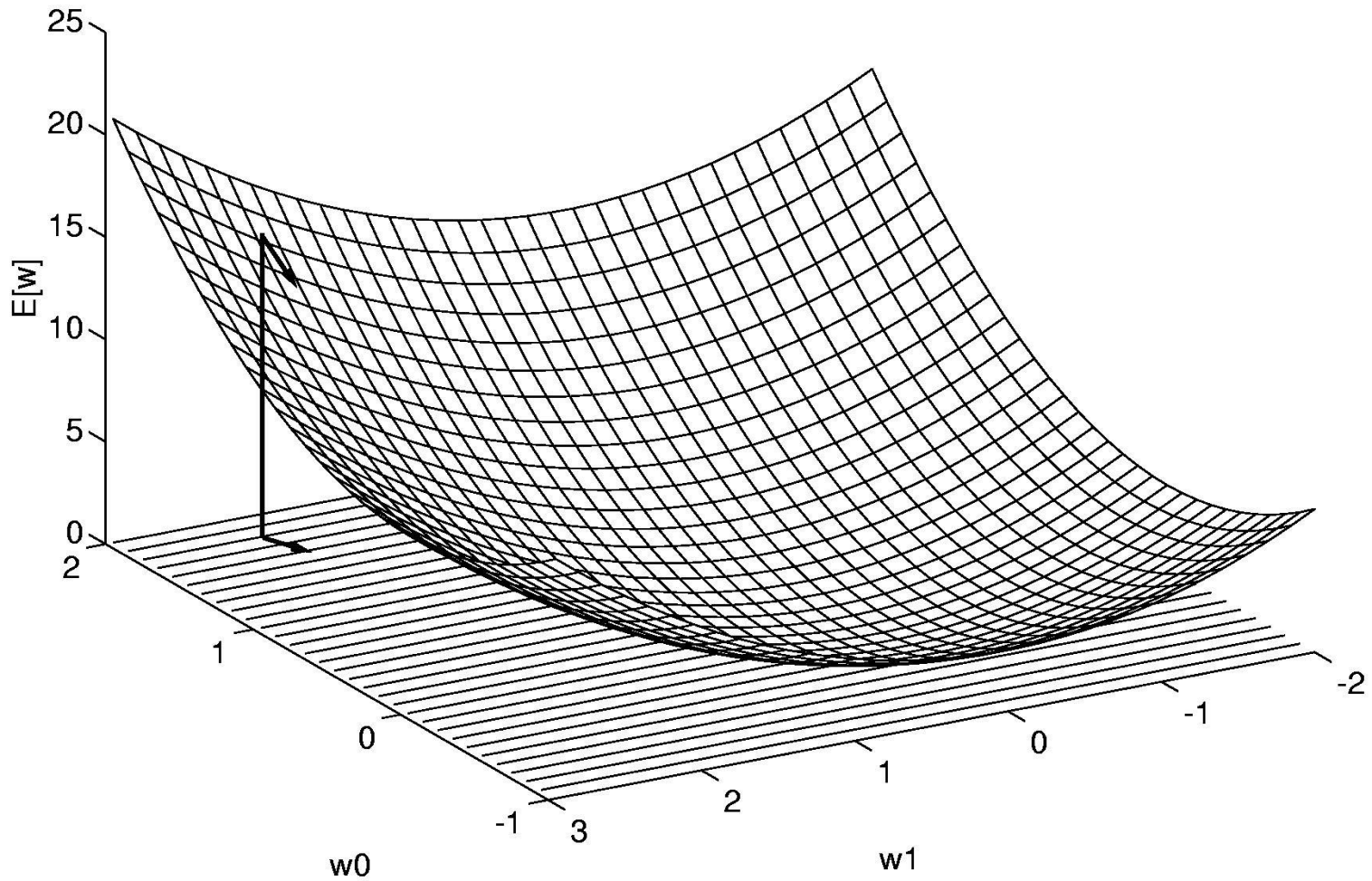
$\mathbf{w} \in \mathbb{R}$



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

[gradient plot]

Gradient Descent



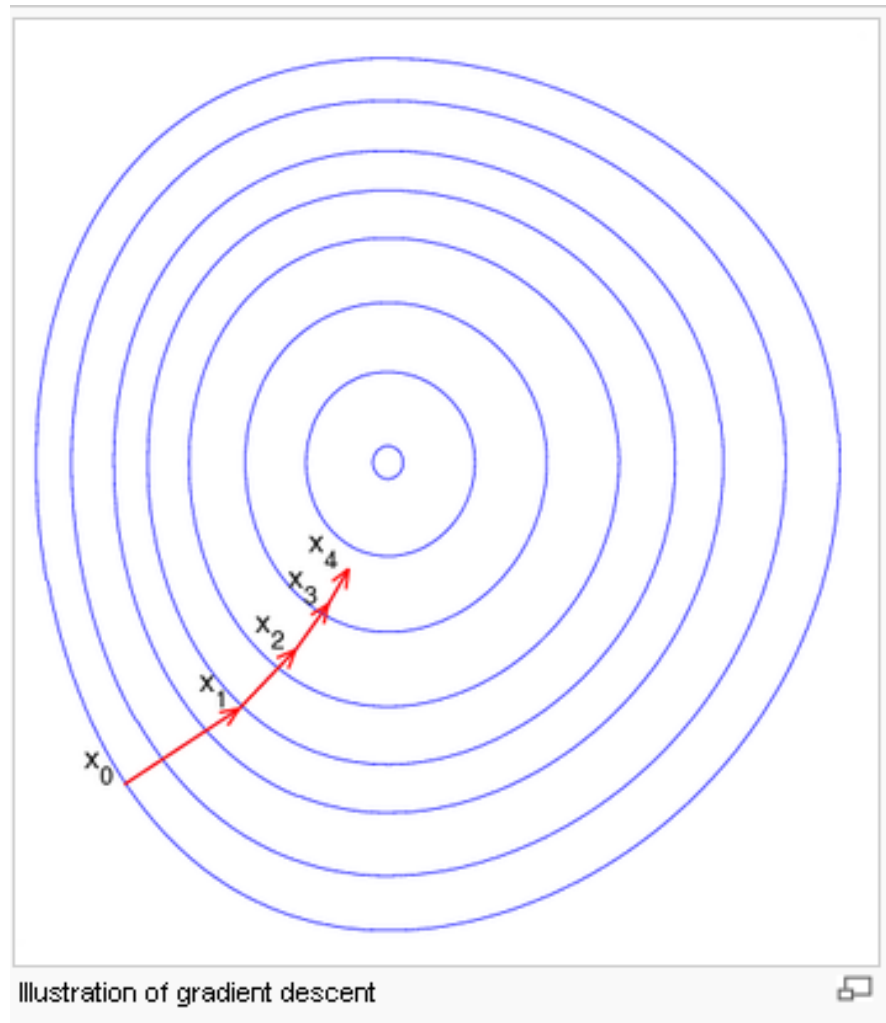


Illustration of gradient descent

Gradient Descent: the basic idea

Gradient descent is based on the observation that if the real-valued function $F(\mathbf{x})$ is defined and differentiable in a neighborhood of a point $\mathbf{a}$, then $F(\mathbf{x})$ decreases *fastest* if one goes from $\mathbf{a}$ in the direction of the negative gradient of F at $\mathbf{a}$, $-\nabla F(\mathbf{a})$. It follows that, if

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

for $\gamma > 0$ a small enough number, then $F(\mathbf{a}) \geq F(\mathbf{b})$. With this observation in mind, one starts with a guess $\mathbf{x}_0$ for a local minimum of F, and considers the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ such that

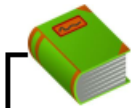
$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \ n \geq 0.$$

We have

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots,$$

so hopefully the sequence $(\mathbf{x}_n)$ converges to the desired local minimum. Note that the value of the *step size* γ is allowed to change at every iteration.

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 $\mathbf{w} = [0, \ldots, 0]$

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

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

Iterative Optimization

- A general approach is to use iterative optimization,
- Essentially starts at some starting point w (say, all zeros),
- and tries to tweak w so that the objective function value decreases.
- The gradient of the function tells us which direction to move in to decrease the objective the most.
- The gradient is a valuable piece of information, especially since we will often be optimizing in high dimensions (d on the order of thousands).
- This iterative optimization procedure is called gradient descent.

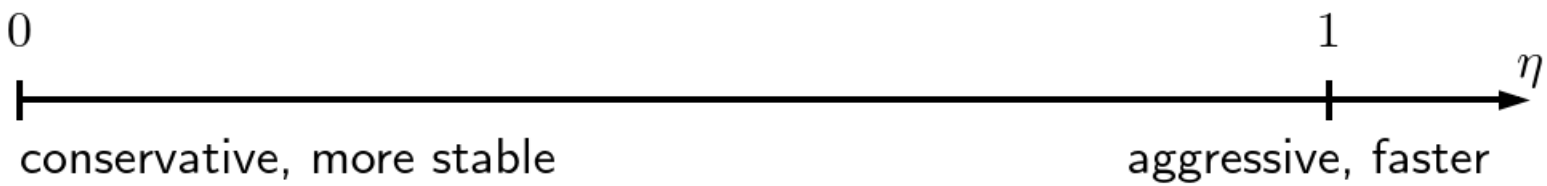
Hyperparameters

- Gradient descent has two hyperparameters,
  - the step size η (which specifies how aggressively we want to pursue a direction) and
  - the number of iterations T.

Step size

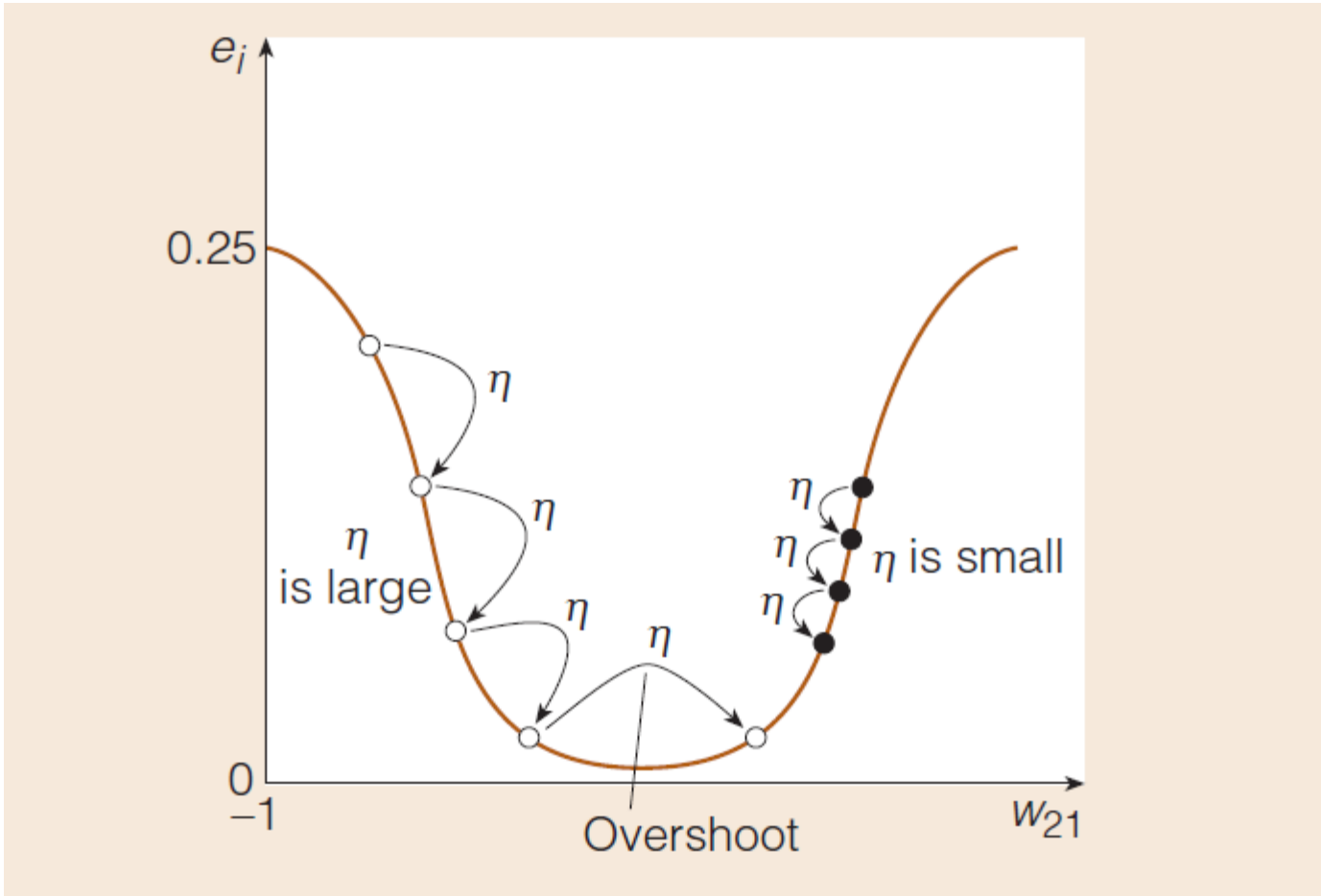
$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Question: what should η be?



Strategies:

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


e_i
0.25
0
−1
η
η
η is large
η
η
η
η
η
η
η is small
Overshoot
w_{21}

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):

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

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!

- If we have one million training examples (which is, by today's standards, only a modest number),
- then each gradient computation requires going through those one million examples, and this must happen before we can make any progress.
- **Can we make progress before seeing all the 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):

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

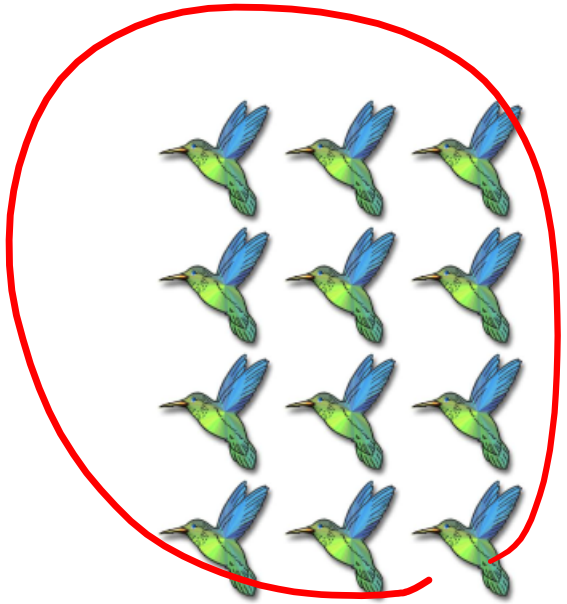
Stochastic gradient descent (SGD):

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**.



Algorithm: stochastic gradient descent

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

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

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

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

Stochastic Gradient Descent : SGD

- The answer is stochastic gradient descent (SGD). Rather than looping through all the training examples to compute a single gradient and making one step,
- SGD loops through the examples (x, y) and updates the weights w based on each example.
- **Each update is not as good**
  - because we're only looking at one example rather than all the examples,
  - but we can make many more updates this way.
- In practice, we often find that just performing one pass over the training examples with SGD, touching each example once, often performs comparably to taking ten passes over the data with GD.

Other Variants of SGD

- There are other variants of SGD. You can randomize the order in which you loop over the training data in each iteration, which is useful.
- Think about what would happen if you had all the positive examples first and the negative examples after that.

Summary so far

Linear predictors:

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

Loss minimization: learning as optimization

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

Stochastic gradient descent: optimization algorithm

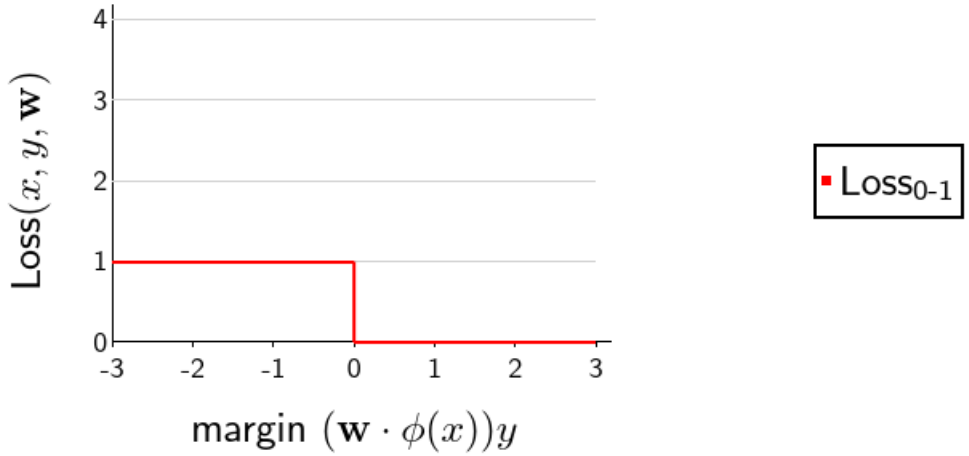
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Done for linear regression; what about classification?

Back to classification and the zero-one loss

Zero-one loss

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

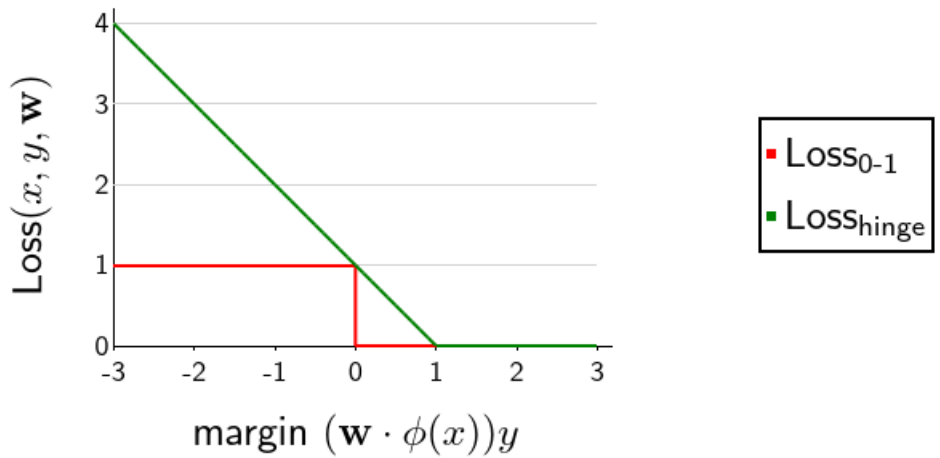


Problems:

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

Hinge loss (SVMs)

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

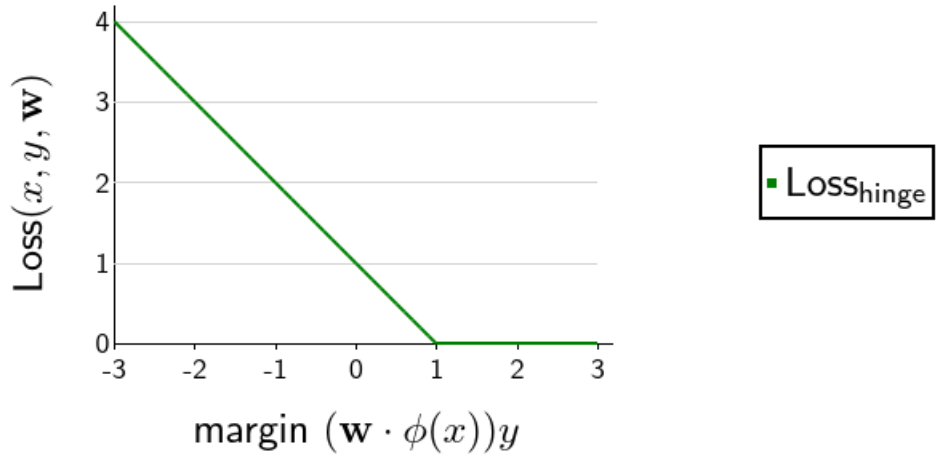


- Intuition: hinge loss upper bounds 0-1 loss, has non-trivial gradient
- Try to increase margin if it is less than 1

Advanced

- The hinge loss is an upper bound on the zero-one loss. Minimizing upper bounds are a general idea; the hope is that pushing down the upper bound leads to pushing down the actual function.
- The hinge loss corresponds to the Support Vector Machine (SVM) objective function with one important difference.
- The SVM objective function also includes a regularization penalty $||w||^2$, which prevents the weights from getting too large.
- Why should we penalize $||w||^2$?
- One answer is Occam's razor, which says to find the simplest hypothesis that explains the data.
- Here, simplicity is measured in the length of w.
- This can be made formal using statistical learning theory.

A gradient exercise



Problem: Gradient of hinge loss

Compute the gradient of

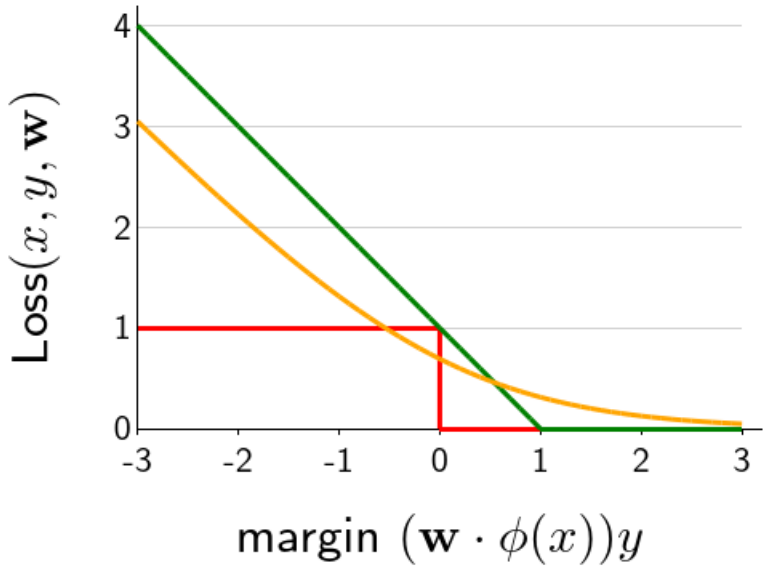
$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{1 - (\mathbf{w} \cdot \phi(x))y, 0\}$$

- You should try to "see" the solution before you write things down formally. Pictorially, it should be evident: when the margin is less than 1, then the gradient is the gradient of $1 - (\mathbf{w} \cdot \phi(x))y$, which is equal to $-\phi(x)y$. If the margin is larger than 1, then the gradient is the gradient of 0, which is 0. Combining the two cases: $\nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \begin{cases} -\phi(x)y & \text{if } \mathbf{w} \cdot \phi(x)y < 1 \\ 0 & \text{if } \mathbf{w} \cdot \phi(x)y > 1. \end{cases}$

- What about when the margin is exactly 1? Technically, the gradient doesn't exist because the hinge loss is not differentiable there. Fear not! Practically speaking, at the end of the day, we can take either $-\phi(x)y$ or 0 (or anything in between).

Logistic regression

$$\text{Loss}_{\text{logistic}}(x, y, \mathbf{w}) = \log(1 + e^{-(\mathbf{w} \cdot \phi(x))y})$$



- Intuition: Try to increase margin even when it already exceeds 1

- Another popular loss function used in machine learning is the logistic loss.
- The main property of the logistic loss is no matter how correct your prediction is, you will have non-zero loss, and so there is still an incentive (although a diminishing one) to push the margin even larger.
- This means that you'll update on every single example.

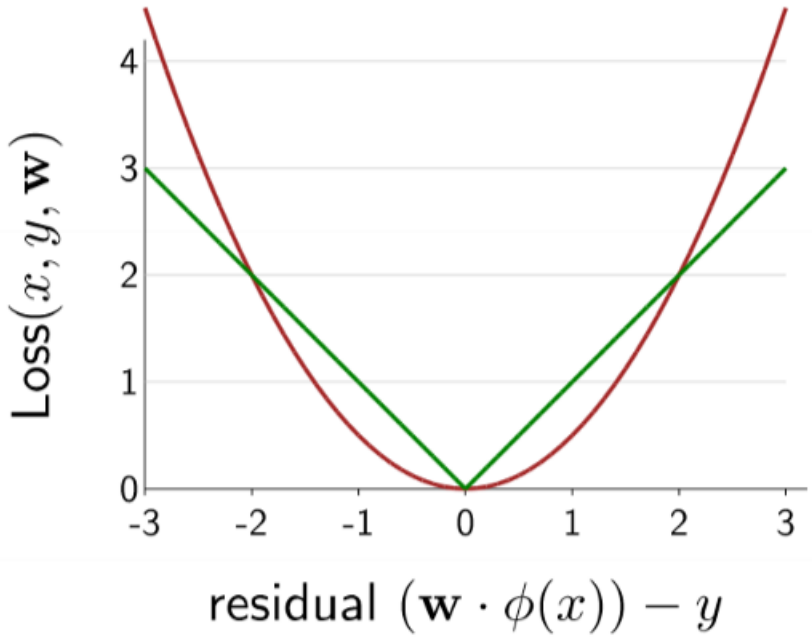
Summary so far

$$\underbrace{\mathbf{w} \cdot \phi(x)}_{\text{score}}$$

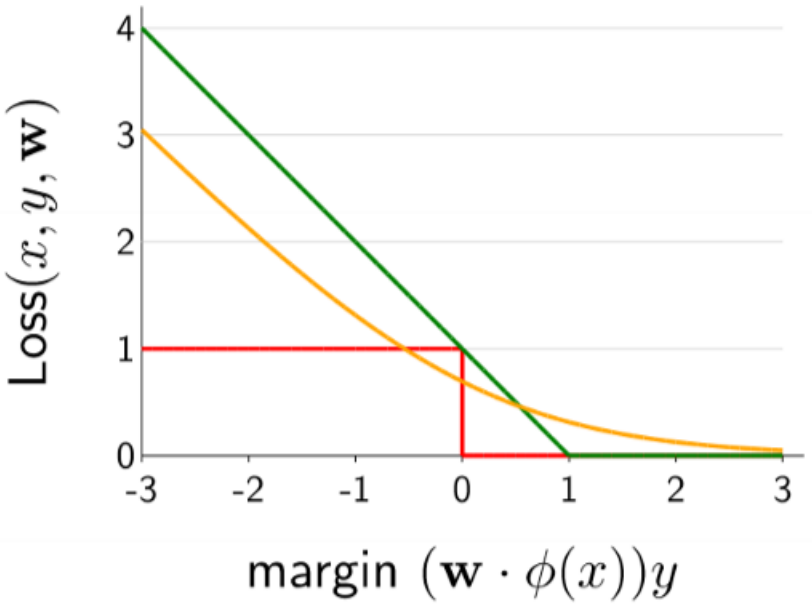
	Classification	Linear regression
Predictor $f_{\mathbf{w}}$	sign(score)	score
Relate to correct y	margin (score y)	residual (score $- y$)
Loss functions	zero-one hinge logistic	squared absolute deviation
Algorithm	SGD	SGD

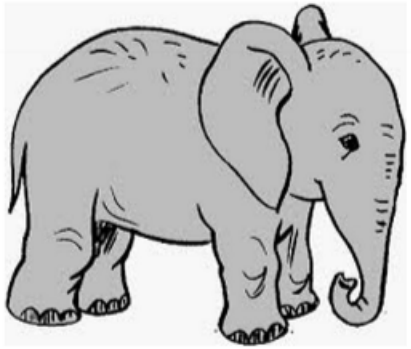
Review: loss functions

Regression



Binary classification



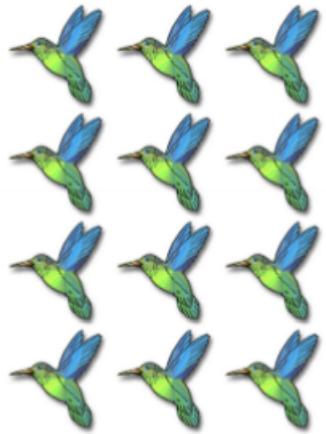


Algorithm: gradient descent

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

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

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



Algorithm: stochastic gradient descent

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

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

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

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

Batch (Gradient Descent) versus incremental (Stochastic Gradient Descent)

- The incremental version (sometimes called stochastic gradient) is faster
- But more sensitive to noise
- Better at avoiding local minima