CSCI567 Machine Learning (Fall 2017)

Prof. Fei Sha

U of Southern California

Lecture on Sept. 5, 2017

Outline

Outline

Administrative stuff

- Homework 1 will be released soon.
- TA office hours have been announced – location is outside SAL 126
- Course material (syllabus and lecture notes) will be uploaded to Piazza

Outline

General nonlinear basis functions

We can use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

where M is the dimensionality of the new feature/input $\boldsymbol{z}$ (or $\boldsymbol{\phi}(\boldsymbol{x})$). Note that M could be either greater than D or less than or the same.

Note that $\boldsymbol{z}$ is a vector

$$v_1 = \phi_1(\boldsymbol{x}), v_2 = \phi_2(\boldsymbol{x}), \cdots, v_M = \phi_M(\boldsymbol{x})$$

Nonlinear regression thru nonlinearly transformed features

With the new features – we call them nonlinear basis functions – we can apply our learning techniques:

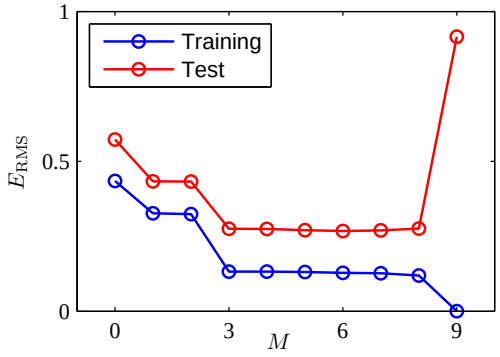
- linear methods: prediction is based on $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$
- more broadly, other methods: nearest neighbors, decision trees, etc

to minimize our errors on the transformed training data

Detecting overfitting

Plot model complexity versus objective function

As model becomes more complex, performance on training keeps improving while on test data improve first and deteriorate later.



- Horizontal axis: *measure of model complexity*
  In this example, we use the maximum order of the polynomial basis functions.
- Vertical axis:
  1. For regression, the vertical axis would be RSS or RMS (squared root of RSS)
  2. For classification, the vertical axis would be classification error rate or cross-entropy error function (more on the latter later)

How to make $\boldsymbol{w}$ small?

Regularized linear regression/Ridge Regression: a new error to minimize

$$\min \mathcal{E}(\boldsymbol{w}) = \min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where $\lambda > 0$. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

Intuitions

- If $\lambda \to +\infty$, then $\boldsymbol{w}$ approaches $\mathbf{0}$.

How to make $\boldsymbol{w}$ small?

Regularized linear regression/Ridge Regression: a new error to minimize

$$\min \mathcal{E}(\boldsymbol{w}) = \min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where $\lambda > 0$. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

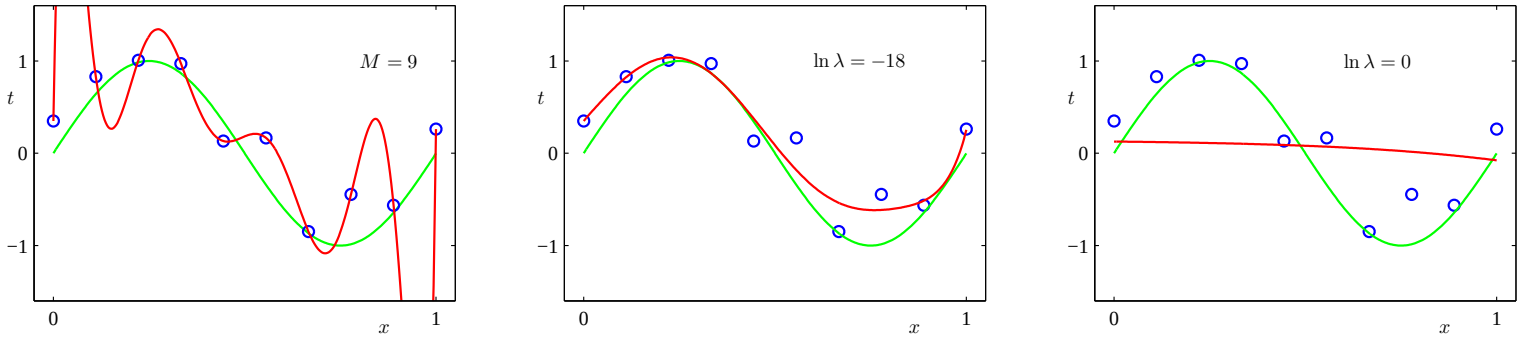
Intuitions

- If $\lambda \to +\infty$, then $\boldsymbol{w}$ approaches $\mathbf{0}$.
- If $\lambda \to 0$, then we approach the standard LMS solution

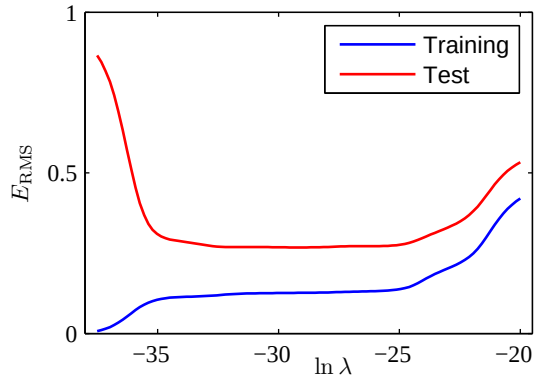
$$\arg\min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2$$

Overfitting in terms of λ

Overfitting is reduced from complex model to simpler one with the help of increasing regularizers



λ **vs. residual error** shows the difference of the model performance on training and testing dataset



Outline

Motivation

Multi-class classification

- Input (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- Learning goal: $y = f(\boldsymbol{x})$

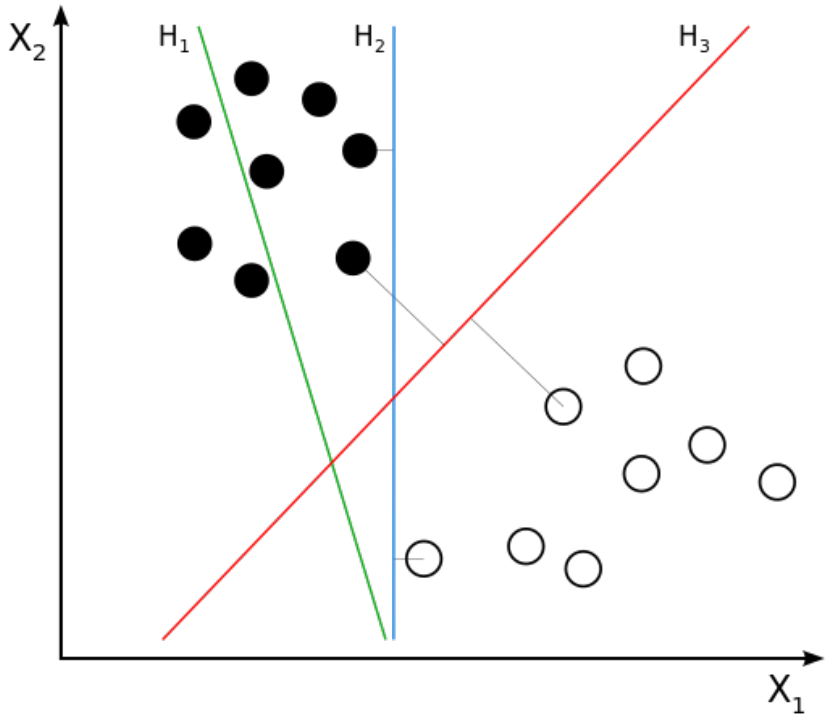
Special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$

Algorithms

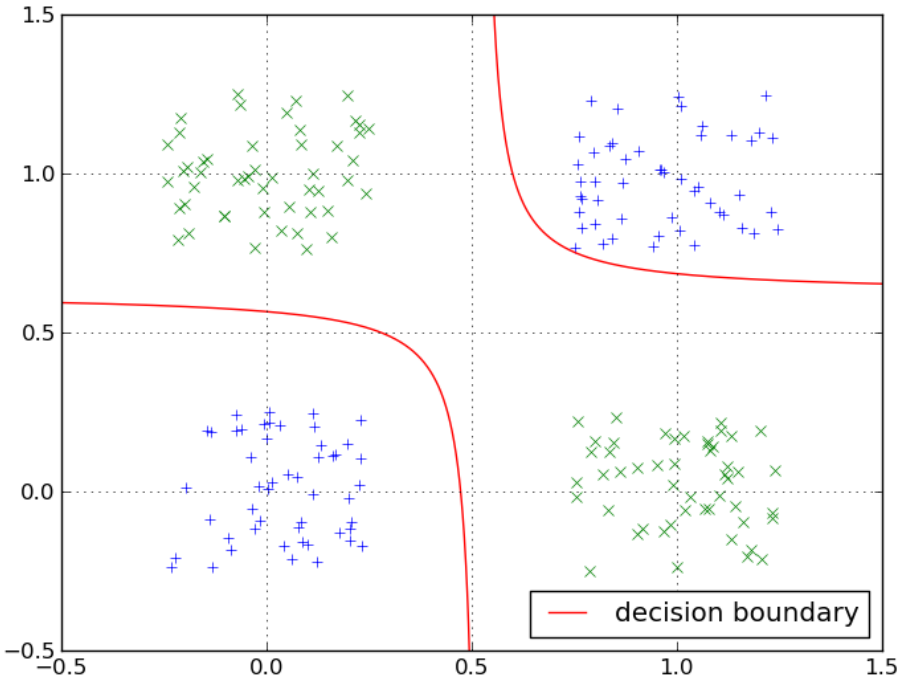
- Nearest neighbor classifier: nonparametric requiring carrying around the training dataset — can we have a parametric model similar to linear regression?

Use linear functions to separate data



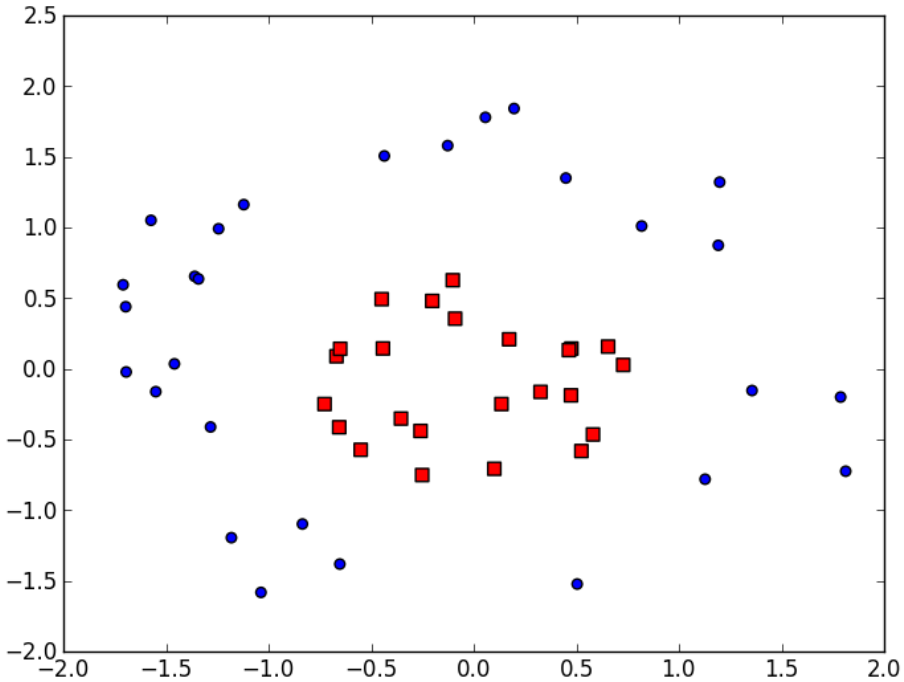
The two sets of (training) data can be separated by a line in 2D (or a plan in 3D, or a hyperplane in high-dimensional space). The line splits the space into two parts. We call the dataset is *linearly separable*

Linear functions are not always adequate



We cannot draw a single line to separate those points into two categories. The decision boundary is necessarily nonlinear - we can such datasets *not linearly separable*

Another example

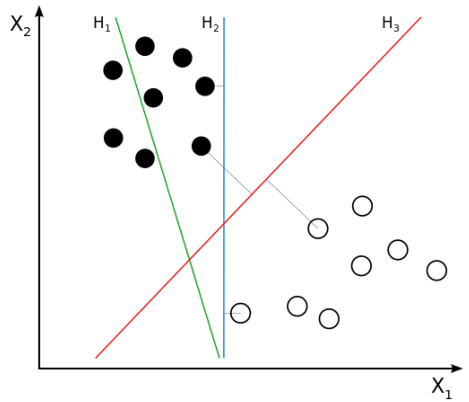


Our focus: linear classifier

Use nonlinear basis going linear to nonlinear

- deep learning approaches
- kernel methods
- ...

Definition of linear classifiers for binary classification



The decision boundary is a linear function of the input $\boldsymbol{x}$. The prediction is

$$y = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \begin{cases} +1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} > 0 \\ -1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \leq 0 \end{cases}$$

Sometimes, we use sgn for sign. Note that a correct classification happens *if and only if*

$$y^* \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} > 0$$

where y^* is the true label.

Outline

Main idea

Consider a linear model for binary classification

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$

is used to distinguish two classes $\{-1, +1\}$.
Our goal is to minimize the following

$$\varepsilon = \sum_{n} \mathbb{I}[y_n \neq \mathsf{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]$$

i.e., at least the errors on the training dataset are reduced.

Hard, but easy if we have only one training example

Suppose we have $\boldsymbol{x}_n$ and its correct label y_n. How to change $\boldsymbol{w}$ such that

$$y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)?$$

Two cases

- If $y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$, do nothing.
- If $y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$,

$$\boldsymbol{w}^{\mathrm{NEW}} \leftarrow \boldsymbol{w} + y_n\boldsymbol{x}_n$$

We are guaranteed that $\boldsymbol{w}^{\mathrm{NEW}}$ improves over $\boldsymbol{w}$.

Why would it work?

If $y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$, then (we had made a *mistake*)

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) < 0$$

Why would it work?

If $y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$, then (we had made a *mistake*)

$$y_n(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) < 0$$

What would happen if we change to new $\boldsymbol{w}^{\mathrm{NEW}} = \boldsymbol{w} + y_n\boldsymbol{x}_n$?

$$y_n[(\boldsymbol{w} + y_n\boldsymbol{x}_n)^{\mathrm{T}}\boldsymbol{x}_n] = y_n\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n + y_n^2\boldsymbol{x}_n^{\mathrm{T}}\boldsymbol{x}_n$$

We are adding a positive number, so it would be *possible for the new* $\boldsymbol{w}^{\mathrm{NEW}}$

$$y_n(\boldsymbol{w}^{\mathrm{NEW}\,\mathrm{T}}\boldsymbol{x}_n) > 0$$

i.e., *classify correctly!*

Perceptron

Iteratively improving one case at a time

- REPEAT
- Pick a data point $\boldsymbol{x}_n$ randomly
- Make a prediction $y = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$ using the *current* $\boldsymbol{w}$
- If $y = y_n$, do nothing. Else,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \boldsymbol{x}_n$$

- UNTIL converged.

Note that the new $\boldsymbol{w}$ is used to make prediction as soon as it is updated.

Properties

- If the training data is linearly separable, the algorithm stops in a finite number of steps.
- The parameter vector is always a linear combination of training instances.

What is our loss of misclassifying?

A loss function

$$L(f(\boldsymbol{x}), y) = L(\text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } y = f(x) \\ -y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} & \text{if } y \neq f(x) \end{cases}$$

The optimal w should minimize, given a training dataset $\{(\boldsymbol{x}_n, y_n)\}$

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_n L(f(\boldsymbol{x}_n), y_n)$$

How do we do that?

An overview of numerical methods

We describe one simple method

- Gradient descent (our focus in lecture): simple, especially effective for large-scale problems

Gradient descent is often referred to as an *first-order* method as it requires only to compute the gradients (i.e., the first-order derivative) of the function. This type of methods is hugely popular in practice.

Example: $\min f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

- We compute the gradients

$$\frac{\partial f}{\partial \theta_1} = 2(\theta_1^2 - \theta_2)\theta_1 + \theta_1 - 1 \tag{1}$$

$$\frac{\partial f}{\partial \theta_2} = -(\theta_1^2 - \theta_2) \tag{2}$$

- Use the following *iterative* procedure for *gradient descent*
  1. Initialize $\theta_1^{(0)}$ and $\theta_2^{(0)}$, and $t = 0$
  2. do

$$\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \eta \left[2({\theta_1^{(t)}}^2 - \theta_2^{(t)})\theta_1^{(t)} + \theta_1^{(t)} - 1 \right] \tag{3}$$

$$\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \eta \left[-({\theta_1^{(t)}}^2 - \theta_2^{(t)}) \right] \tag{4}$$

$$t \leftarrow t + 1 \tag{5}$$

  3. until $f(\boldsymbol{\theta}^{(t)})$ *does not change much*

Gradient descent

General form for minimizing $f(\boldsymbol{\theta})$

$$\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial f}{\partial \boldsymbol{\theta}}$$

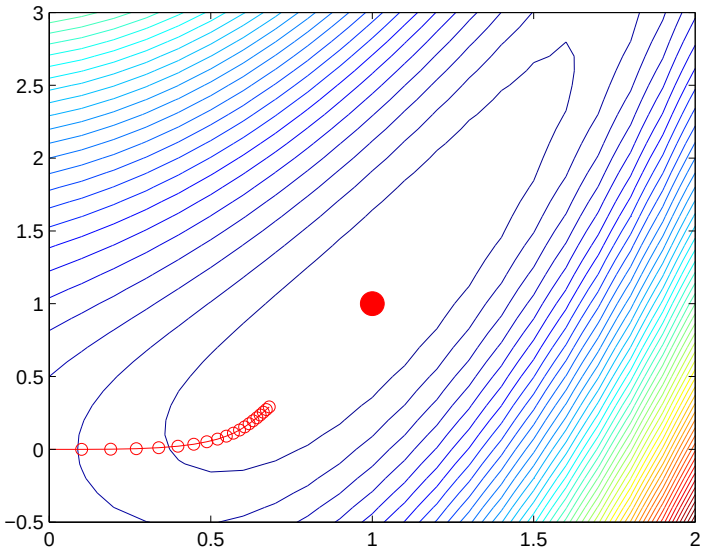
Remarks

- η is often called *step size* – literally, how far our update will go along the the direction of the negative gradient
- Note that this is for *minimizing* a function, hence the subtraction $(-\eta)$
- With a *suitable* choice of η, the iterative procedure converges to a stationary point where
$$\frac{\partial f}{\partial \boldsymbol{\theta}} = 0$$
- A stationary point is only necessary for being the minimum.

Seeing in action

Choose the right η is important

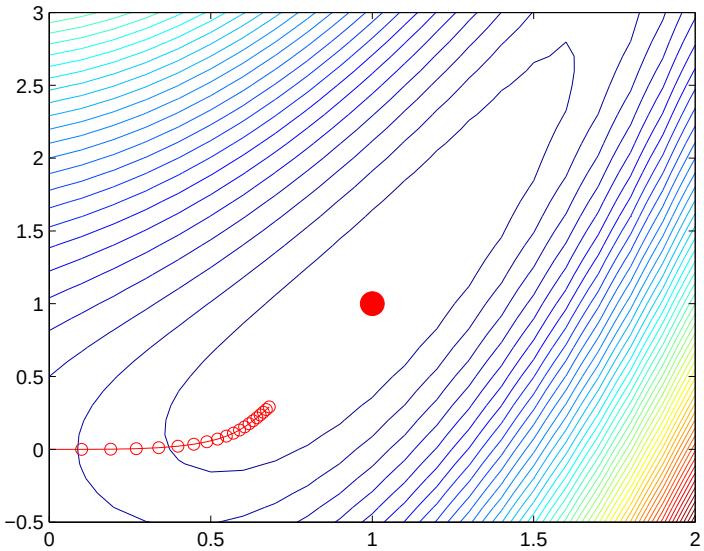
small η is too slow?



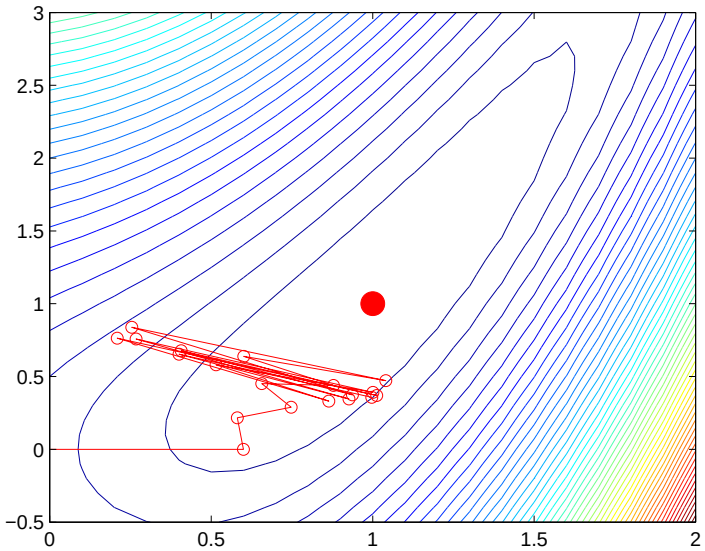
Seeing in action

Choose the right η is important

small η is too slow?



large η is too unstable?



Let us apply the trick to the perceptron

Loss function

$$l(\boldsymbol{w}) = \sum_n L(f(\boldsymbol{x}_n), y_n)$$

where

$$L(f(\boldsymbol{x}), y) = L(\text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } y = f(x) \\ -y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} & \text{if } y \neq f(x) \end{cases}$$

Let us apply the trick to the perceptron

Loss function

$$l(\boldsymbol{w}) = \sum_n L(f(\boldsymbol{x}_n), y_n)$$

where

$$L(f(\boldsymbol{x}), y) = L(\text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } y = f(x) \\ -y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} & \text{if } y \neq f(x) \end{cases}$$

Gradient is

$$\frac{\partial l(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{n: y_n \neq f(\boldsymbol{x}_n)} -y_n \boldsymbol{x}_n$$

Intuition: only those misclassified examples ($y_n \neq f(\boldsymbol{x}_n)$) contribute to gradients.

Update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \sum_{n: y_n \neq f(\boldsymbol{x}_n)} y_n \boldsymbol{x}_n$$

This is called *Batch Update*

But I am impatient!

Stochastic gradient descent: selecting and updating one sample at a time

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_n \boldsymbol{x}_n$$

if $y_n \neq f(\boldsymbol{x}_n)$

This is precisely the perceptron update rule if we set $\eta = 1$.

Stochastic gradient descent is the working horse of deep learning algorithm.

Stochastic gradient descent for linear regression

RSS Loss function

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2$$

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$

Stochastic gradient descent for linear regression

RSS Loss function

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2$$

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_n = (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_n) \tilde{\boldsymbol{x}}_n$$

Stochastic gradient descent for linear regression

RSS Loss function

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2$$

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_n = (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_n)\tilde{\boldsymbol{x}}_n$$

  3. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \boldsymbol{g}_n$

Stochastic gradient descent for linear regression

RSS Loss function

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n - y_n)^2$$

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_n$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_n = (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_n)\tilde{\boldsymbol{x}}_n$$

  3. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \boldsymbol{g}_n$
  4. $t \leftarrow t + 1$

Outline

Linear classifier

- Use a linear function of the input features to classify
- Perceptron
  - Mistake-driven: improve the linear function when it makes a mistake in classification
  - Iterative in nature: iterative numerical procedure