Machine Learning

CS 539

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

Project Teams

- Will Buchta, Sarah Semy, Deena Selitsk, and Humza Qureshi
- Kai Davidson, Sarah Kogan, Dominic Ranelli
- Bryce Lukens, Harrison Taylor, Gabriel Barbosa
- Axel Luca, Preet Patel, Xiaojun Liu
- Codey Battista, Avinash Bissoondial, Nick Chantre, Paul Crann
- John DeLeo, Jacob Donahue, Joe Molder, Iris Nycz
- Srisaranya Pujari, Kangjian Wu, Matthew Yeo, Yang Yi

Previous Class…

Linear Regression
Gradient Descent

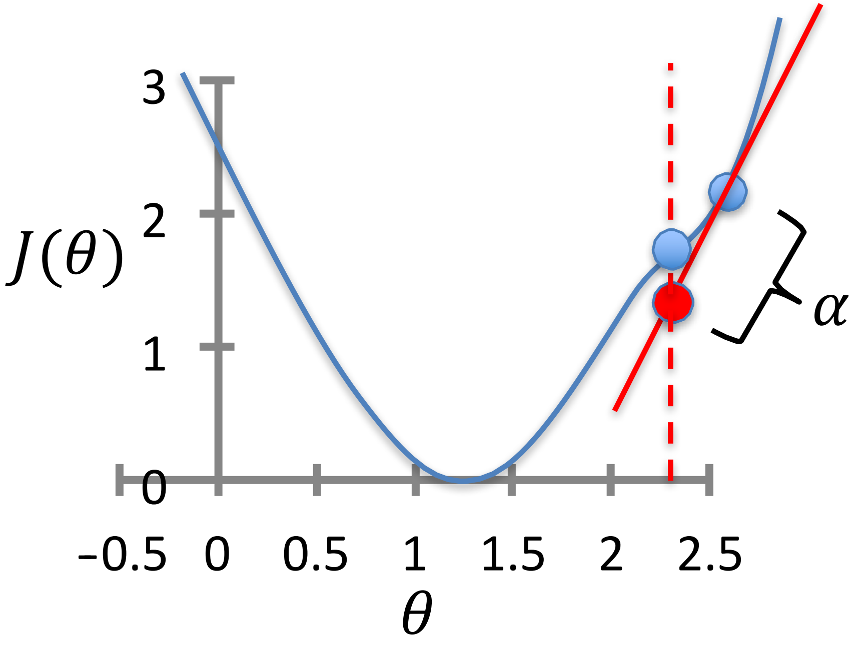
Gradient Descent

- Initialize θ
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

simultaneous update
for j = 0 … d

learning rate (small)
e.g., α = 0.05



Previous Class…

Linear Regression
Gradient Descent

Linear Basis Function
Models

Linear Basis Function Models

- Generally,

$$h_\theta(x) = \sum_{j=0}^{d} \theta_j \underbrace{\phi_j(x)}_{\text{basis function}}$$

- Typically, $\phi_0(x) = 1$ so that θ_0 acts as a bias
- In the simplest case, we use linear basis functions:

$$\phi_j(x) = x_j$$

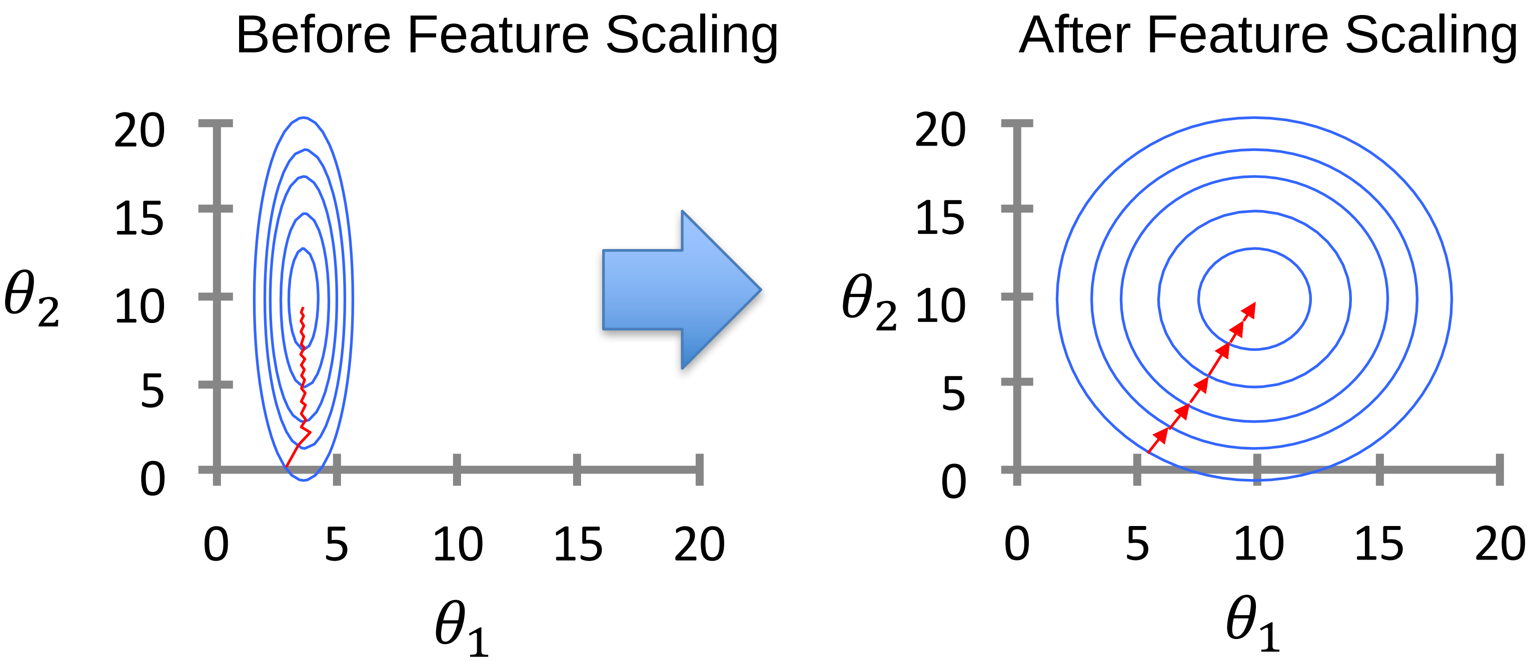
HW2

- https://canvas.wpi.edu/courses/58900/assignments/355666

Linear Regression

Improving Learning: Feature Scaling

- **Idea:** Ensure that feature have similar scales



Features have various value range
e.g., x1 = 1~2000 and x2 = 1~5

- Makes gradient descent converge *much* faster

Feature Standardization

- Rescales features to have zero mean and unit variance
  - Let μ_j be the mean of feature j: $$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$$
  - Replace each value with

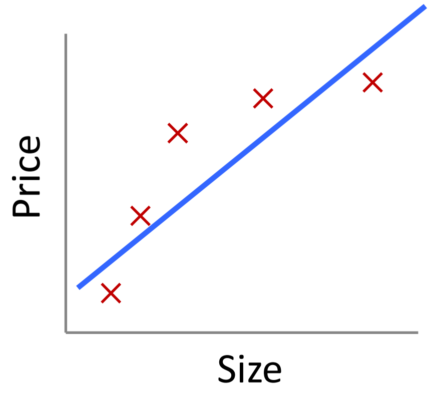
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

for j = 1...d
(not x_0!)

  - s_j is the standard deviation of feature j
  - Could also use the range of feature j ($\max_j$ – $\min_j$) for s_j

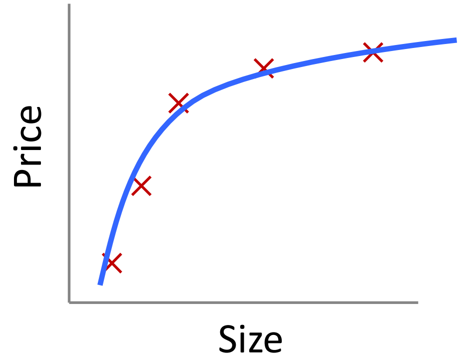
- Must apply the same transformation to instances for both training and prediction

Quality of Fit



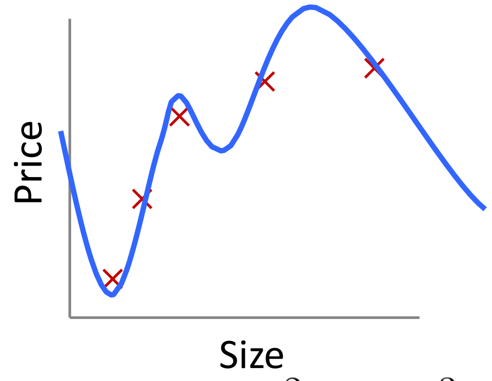
$\theta_0 + \theta_1 x$

Underfitting
(high bias)



$\theta_0 + \theta_1 x + \theta_2 x^2$

Correct fit



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting
(high variance)

- **Overfitting:**
  - The learned hypothesis may fit the training set very well ($J(\boldsymbol{\theta}) \approx 0$)
  - ...but fails to generalize to new examples

Regularization

- A method for automatically controlling the complexity of the learned hypothesis

- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label

- Can also address overfitting by eliminating features (either manually or via model selection)

Regularization (Ridge Regression)

- Linear regression objective function

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!
- Other regularization methods: Lasso and Elastic Net Regressions

https://jakevdp.github.io/PythonDataScienceHandbook/05.06-linear-regression.html#Gaussian-Basis
https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net

Understanding Regularization

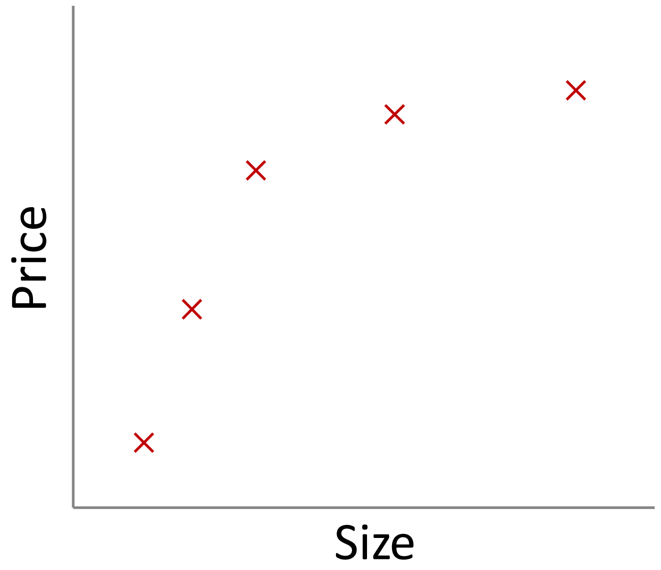
$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2$$

- Note that $$\sum_{j=1}^{d} \theta_j^2 = \|\boldsymbol{\theta}_{1:d}\|_2^2$$
  - This is the magnitude of the feature coefficient vector!

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?

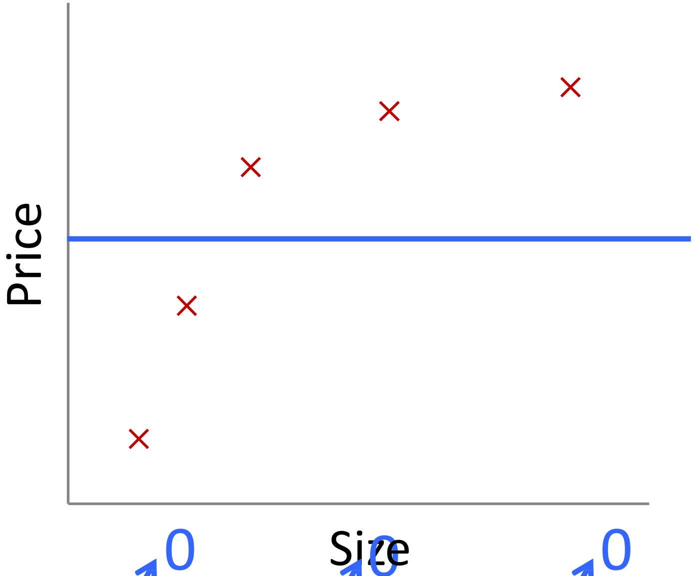


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \cancelto{0}{\theta_1} x + \cancelto{0}{\theta_2} x^2 + \cancelto{0}{\theta_3} x^3 + \cancelto{0}{\theta_4} x^4$$

Regularized Linear Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$$\underbrace{\frac{\partial}{\partial\theta_0}J(\theta)}\quad \theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\frac{\partial}{\partial\theta_j}J(\theta)\quad \theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \underbrace{- \alpha\lambda\theta_j}_{\text{regularization}}$$

Regularized Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \alpha\lambda\theta_j$$

- We can rewrite the gradient step as:

$$\theta_j \leftarrow \theta_j\left(1 - \alpha\lambda\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Regularization in Regression

- L2 Regularization (Ridge Regression)
  - ➔ Good for avoiding overfitting

- L1 Regularization (Lasso Regression)
  - ➔Sometimes perform as a feature selection method by making some coefficients 0.

References: https://jakevdp.github.io/PythonDataScienceHandbook/05.06-linear-regression.html#Gaussian-Basis
http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/

Demo

- The Jupyter notebook
  - A beautiful integrated development environment (IDE) for Python

- https://canvas.wpi.edu/courses/58900/files/6639043?module_item_id=1123451

Google Colab

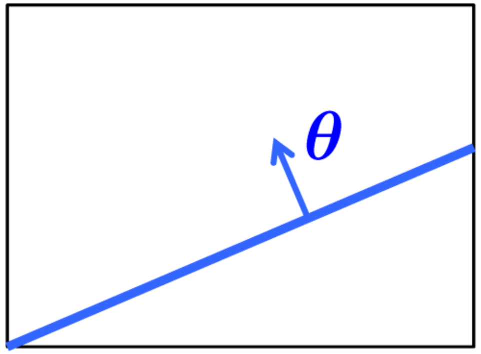
- **Google** Colaboratory is a free online cloud-based **Jupyter notebook** environment that allows us to train our machine learning and deep learning models on CPUs, GPUs, and TPUs.

https://www.youtube.com/watch?v=inN8seMm7UI

Linear Classification:
The Perceptron

Linear Classifiers

- A **hyperplane** partitions $\mathbb{R}^d$ into two half-spaces
  - Defined by the normal vector $\boldsymbol{\theta} \in \mathbb{R}^d$
    - $\boldsymbol{\theta}$ is orthogonal to any vector lying on the hyperplane



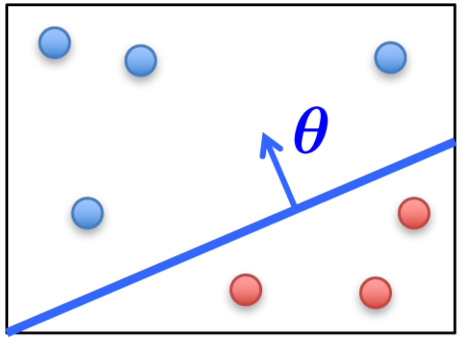
  - Assumed to pass through the origin
    - This is because we incorporated bias term θ_0 into it by $x_0 = 1$

- Consider classification with +1, -1 labels ...

Linear Classifiers

- **Linear classifiers**: represent decision boundary by hyperplane

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \ldots & x_d \end{bmatrix}$$

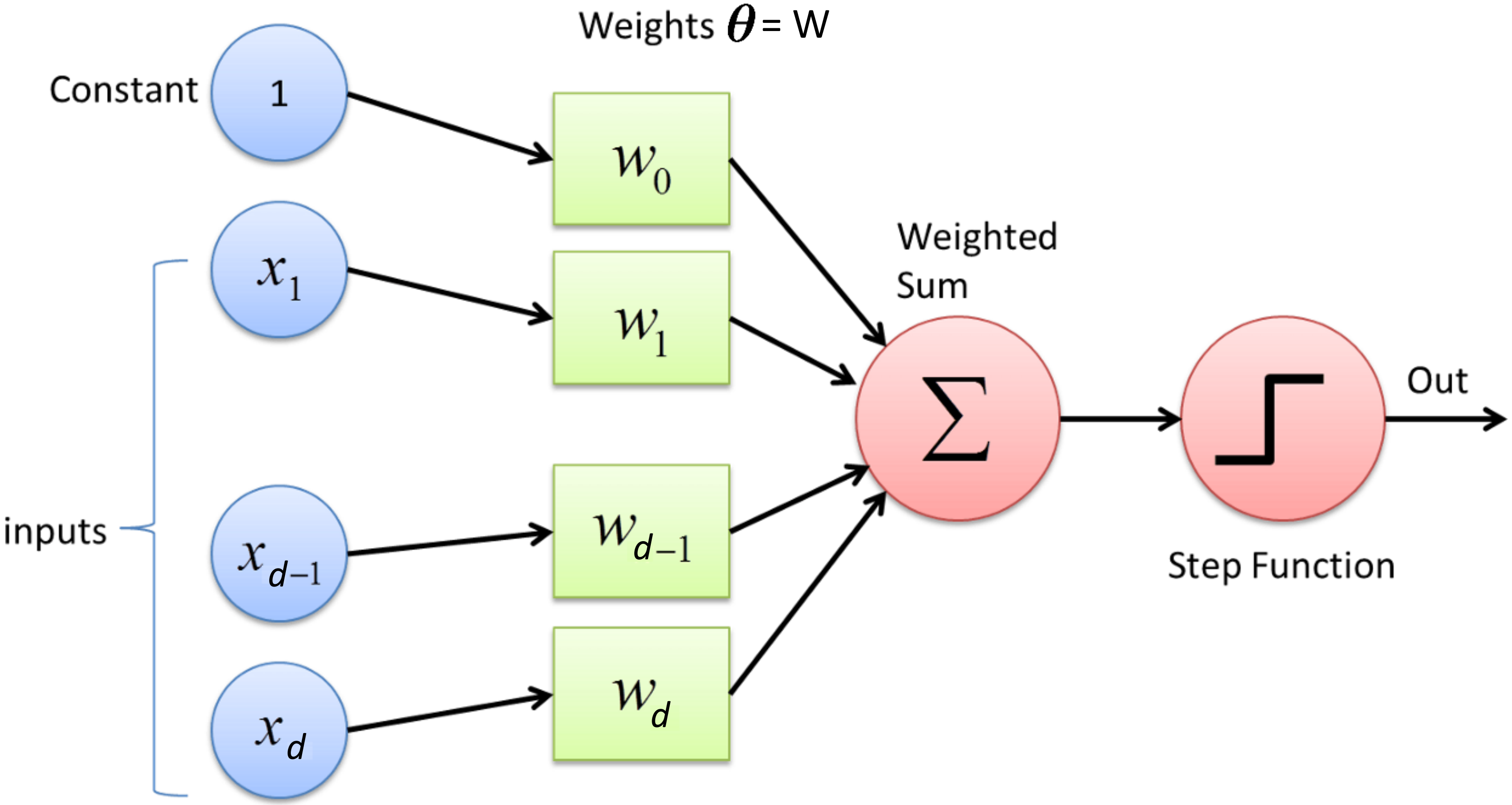


$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \text{ where } \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- Note that: $\boldsymbol{\theta}^\intercal \boldsymbol{x} > 0 \implies y = +1$

  $\boldsymbol{\theta}^\intercal \boldsymbol{x} < 0 \implies y = -1$

The Perceptron



- Perceptron is used to classify linearly separable classes
- Used for binary classification

The Perceptron

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{2} \underbrace{\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)}_{\text{either 2 or -2}} x_j^{(i)}$$

  - If the prediction matches the label, make no change
  - Otherwise, adjust $\boldsymbol{\theta}$

The Perceptron

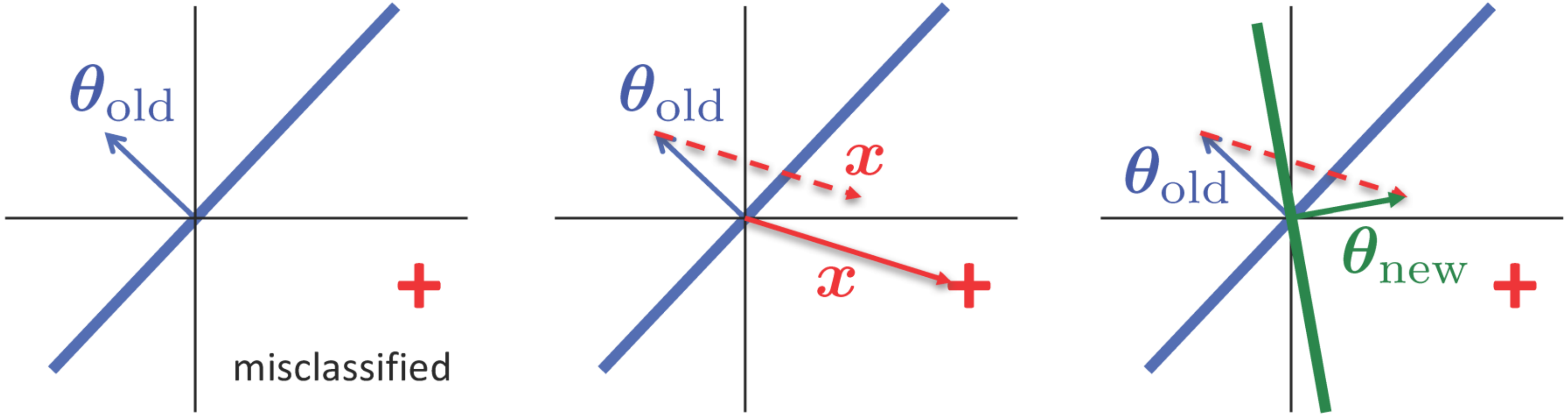
- The perceptron uses the following update rule each time it receives a new training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{2} \underbrace{\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)} \right)}_{\text{either 2 or -2}} x_j^{(i)}$$

- Re-write as $\theta_j \leftarrow \theta_j + \alpha y^{(i)} x_j^{(i)}$ (only upon misclassification)
  - Can eliminate α in this case, since its only effect is to scale $\boldsymbol{\theta}$ by a constant, which doesn't affect performance

Perceptron Rule: If $\boldsymbol{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \boldsymbol{x}^{(i)}$

Why the Perceptron Update Works



Why the Perceptron Update Works

- Consider the misclassified example (y = +1)
  - Perceptron wrongly thinks that $\boldsymbol{\theta}_{\text{old}}^{\mathsf{T}}\boldsymbol{x} < 0$

- Update:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + y\boldsymbol{x} = \boldsymbol{\theta}_{\text{old}} + \boldsymbol{x} \qquad (\text{since } y = +1)$$

- Note that

$$\begin{aligned} \boldsymbol{\theta}_{\text{new}}^{\mathsf{T}}\boldsymbol{x} &= (\boldsymbol{\theta}_{\text{old}} + \boldsymbol{x})^{\mathsf{T}}\boldsymbol{x} \\ &= \boldsymbol{\theta}_{\text{old}}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{x} \end{aligned}$$

$$\|\boldsymbol{x}\|_2^2 > 0$$

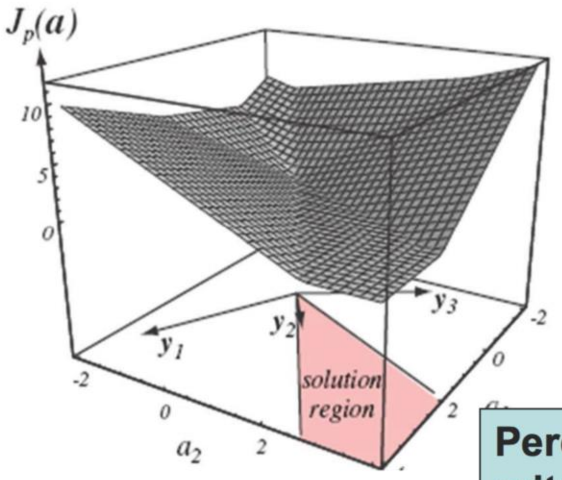
- Therefore, $\boldsymbol{\theta}_{\text{new}}^{\mathsf{T}}\boldsymbol{x}$ is less negative than $\boldsymbol{\theta}_{\text{old}}^{\mathsf{T}}\boldsymbol{x}$
  - So, we are making ourselves <u>more correct</u> on this example!

The Perceptron Cost Function

- The perceptron uses the following cost function

$$J_p(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \max(0, -y^{(i)}\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)})$$

  - $\max(0, -y^{(i)}\boldsymbol{\theta}^\intercal \boldsymbol{x}^{(i)})$ is 0 if the prediction is correct
  - Otherwise, it is the confidence in the misprediction



Nice gradient

Perceptron criterion

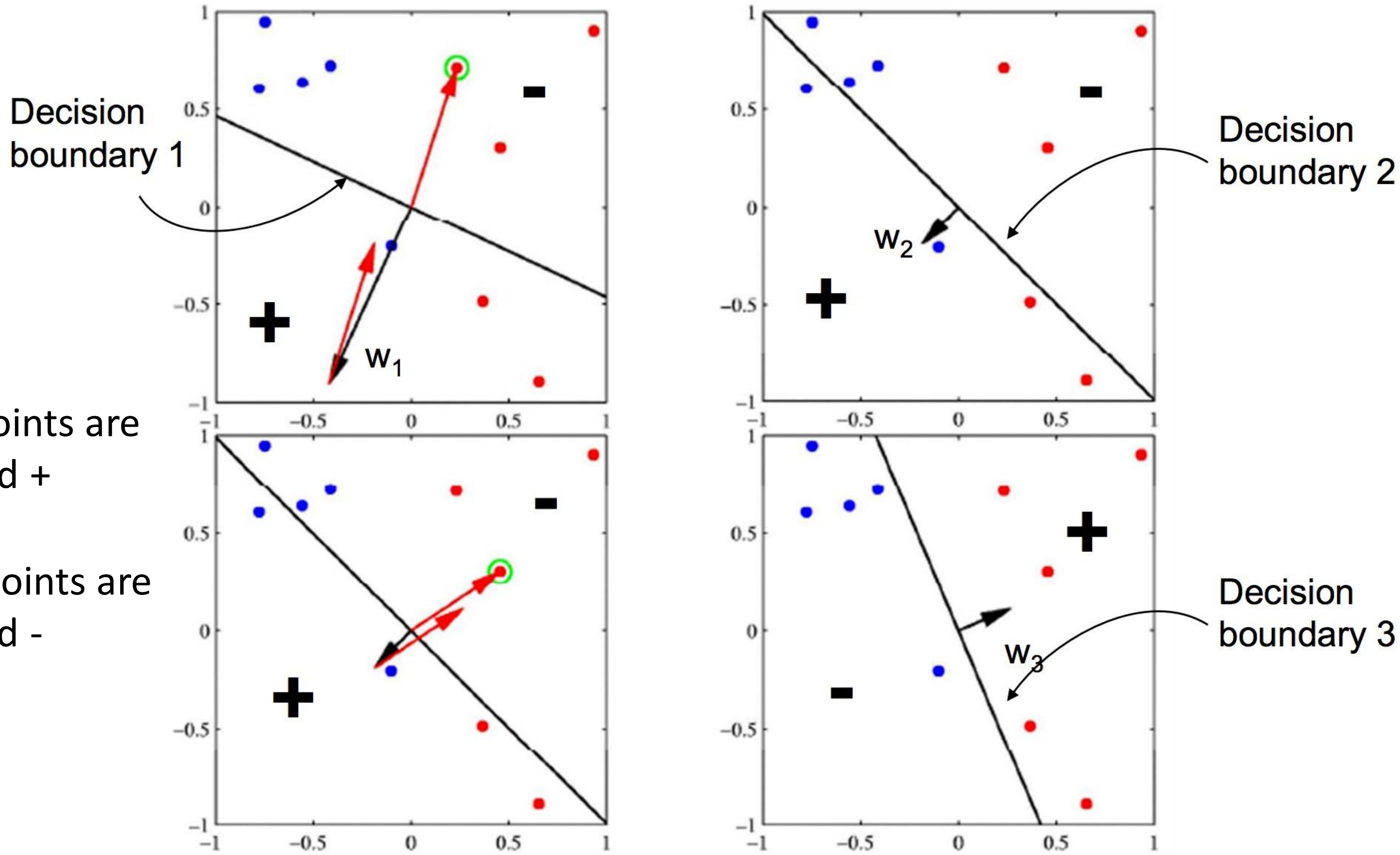
Online Perceptron Algorithm

Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
  Receive training example $(\boldsymbol{x}^{(i)}, y^{(i)})$
  if $y^{(i)} \boldsymbol{x}^{(i)} \boldsymbol{\theta} \leq 0$ // prediction is incorrect
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \boldsymbol{x}^{(i)}$

Online learning – the learning mode where the model update is performed each time a single observation is received

Batch learning – the learning mode where the model update is performed after observing the entire training set

Online Perceptron Algorithm



Red points are labeled +

Blue points are labeled -

See the perceptron in action: www.youtube.com/watch?v=vGwemZhPlsA

Batch Perceptron

Given training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
    Let $\boldsymbol{\Delta} \leftarrow [0, 0, \ldots, 0]$
    for $i = 1 \ldots n$, do
        if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} \leq 0$ // prediction for ith instance is incorrect
            $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} + y^{(i)}\boldsymbol{x}^{(i)}$
    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}/n$ // compute average update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\Delta}$
Until $\|\boldsymbol{\Delta}\|_2 < \epsilon$

- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Guaranteed to find a separating hyperplane if one exists

Improving the Perceptron

- The Perceptron produces many θ's during training
- The standard Perceptron simply uses the final θ at test time
  - This may sometimes not be a good idea!
  - Some other θ may be correct on 1,000 consecutive examples, but one mistake ruins it!

- **Idea:** Use a combination of multiple perceptrons
  - (i.e., neural networks!)
- **Idea:** Use the intermediate θ's
  - **Voted Perceptron**: vote on predictions of the intermediate θ's
  - **Averaged Perceptron**: average the intermediate θ's