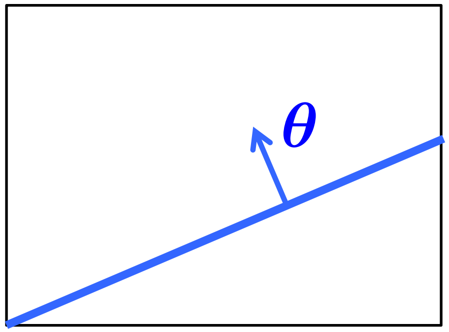
Linear Classification: The Perceptron

These slides were assembled by Eric Eaton, with grateful acknowledgement of the many others who made their course materials freely available online. Feel free to reuse or adapt these slides for your own academic purposes, provided that you include proper attribution. Please send comments and corrections to Eric.

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



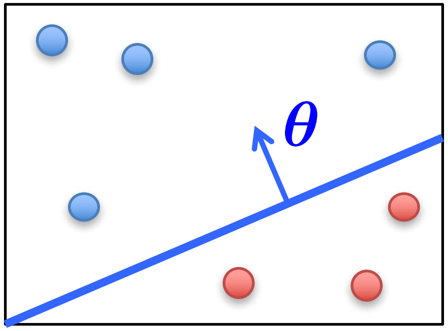
  - Assumed to pass through the origin
    - This is because we incorporated bias term θ_0 into it by $x_0 = 1$

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

Linear Classifiers

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

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



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

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

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

The Perceptron

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

- 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

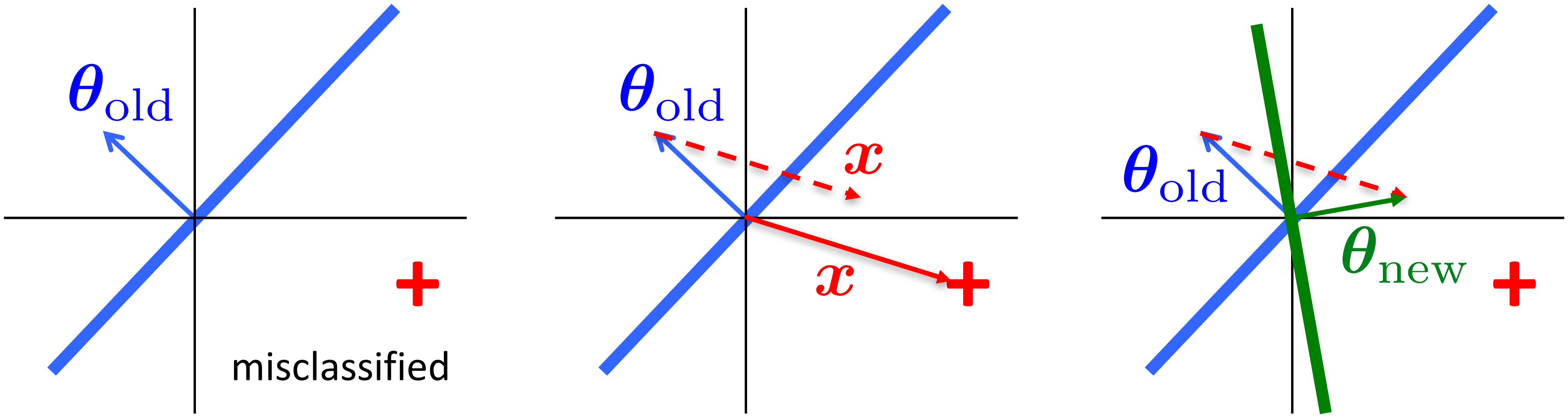
- The perceptron uses the following update rule each time it receives a new training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$

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

- 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 $\boldsymbol{\theta}_{\text{old}}^{\intercal}\boldsymbol{x} < 0$
- 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

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

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

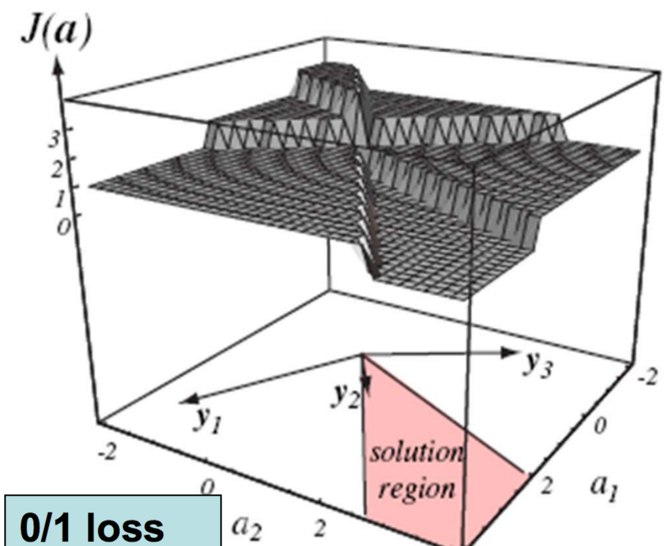
- Therefore, $\boldsymbol{\theta}_{\text{new}}^{\intercal}\boldsymbol{x}$ is less negative than $\boldsymbol{\theta}_{\text{old}}^{\intercal}\boldsymbol{x}$
  - So, we are making ourselves <u>more correct</u> on this example!

The Perceptron Cost Function

- Prediction is correct if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} > 0$
- Could have used 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(sign(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



0/1 loss

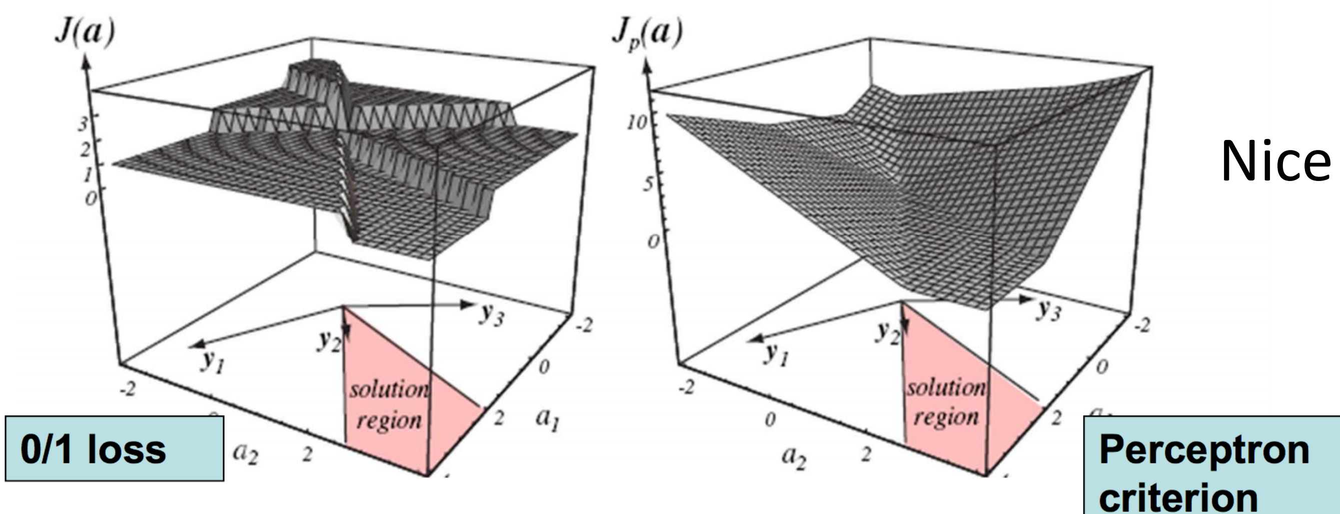
Doesn't produce a useful gradient

The Perceptron Cost Function

- The perceptron uses the following cost function

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

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



Nice gradient

Online Perceptron Algorithm

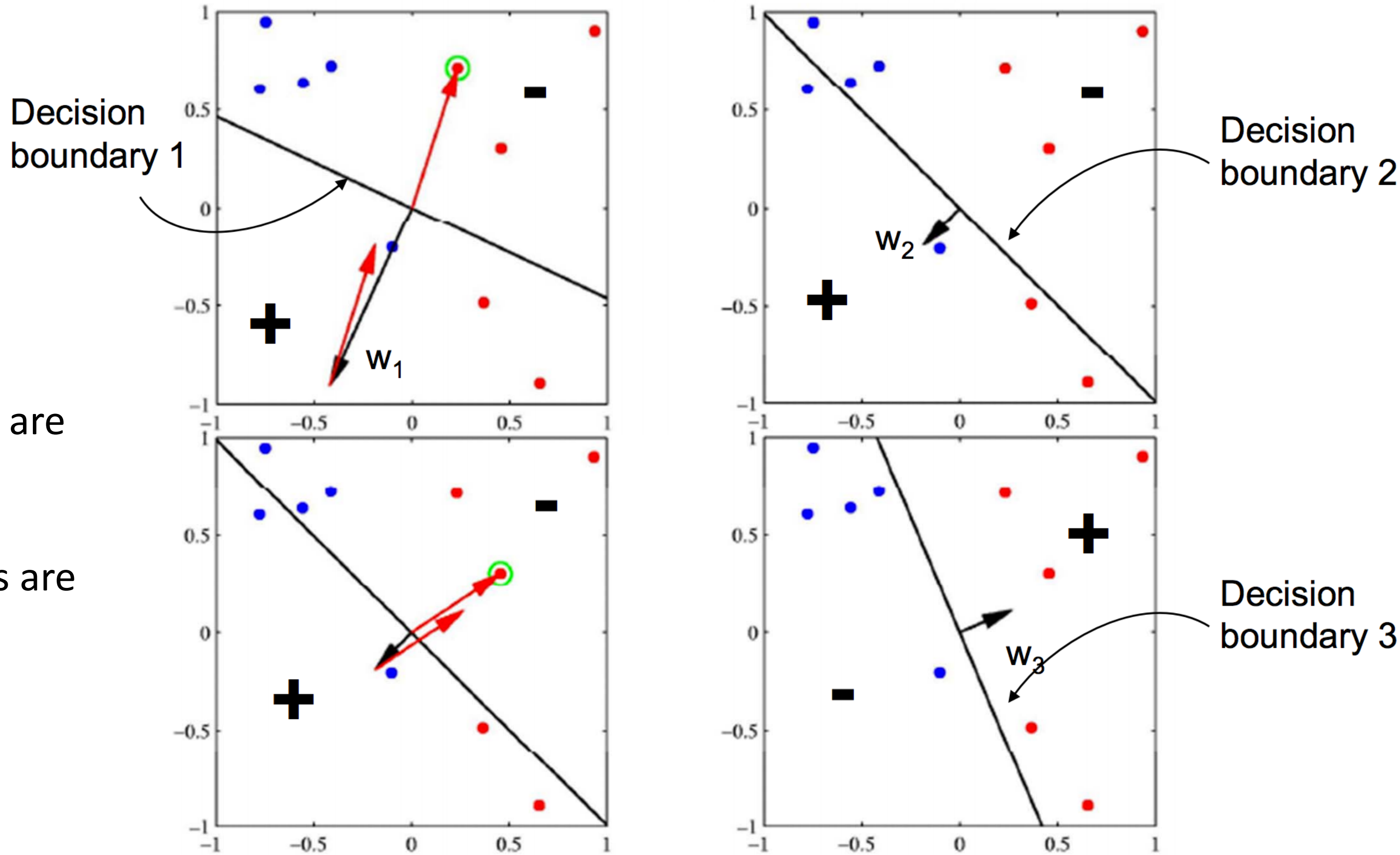
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
  Receive training example $(\boldsymbol{x}^{(i)}, y^{(i)})$
  if $y^{(i)} \boldsymbol{x}^{(i)} \boldsymbol{\theta} \leq 0$ // prediction is incorrect
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \boldsymbol{x}^{(i)}$

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

When an error is made, moves the weight in a direction that corrects the error



Red points are labeled +

Blue points are labeled -

Batch Perceptron

Given training data $\left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$
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 $\boldsymbol{\theta}$'s during training
- The standard Perceptron simply uses the final $\boldsymbol{\theta}$ at test time
  - This may sometimes not be a good idea!
  - Some other $\boldsymbol{\theta}$ 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 $\boldsymbol{\theta}$'s
  - **Voted Perceptron**: vote on predictions of the intermediate $\boldsymbol{\theta}$'s
  - **Averaged Perceptron**: average the intermediate $\boldsymbol{\theta}$'s