Linear Classification: The Perceptron
Linear Classifiers

• A hyperplane partitions $\mathbb{R}^d$ into two half-spaces
  – Defined by the normal vector $\theta \in \mathbb{R}^d$
    • $\theta$ is orthogonal to any vector lying on the hyperplane
  – Assumed to pass through the origin
    • This is because we incorporated bias term $\theta_0$ into it by $x_0 = 1$

• Consider classification with +1, -1 labels ...
Linear Classifiers

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

\[
\theta = \begin{bmatrix}
\theta_0 \\
\theta_1 \\
\vdots \\
\theta_d
\end{bmatrix}, \quad \mathbf{x}^T = \begin{bmatrix}
1 \\
x_1 \\
\vdots \\
x_d
\end{bmatrix}
\]

\[
h(\mathbf{x}) = \text{sign}(\theta^T \mathbf{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 
1 & \text{if } z \geq 0 \\
-1 & \text{if } z < 0
\end{cases}
\]

- Note that: 
  \[\theta^T \mathbf{x} > 0 \implies y = +1\]
  \[\theta^T \mathbf{x} < 0 \implies y = -1\]
The Perceptron

\[ h(\mathbf{x}) = \text{sign}(\theta^\top \mathbf{x}) \text{ where } \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases} \]

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

\[ \theta_j \leftarrow \theta_j - \frac{\alpha}{2} \left( h_\theta(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)} \]

- If the prediction matches the label, make no change
- Otherwise, adjust \(\theta\)
The Perceptron

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

\[
\theta_j \leftarrow \theta_j - \frac{\alpha}{2} \left( h_\theta (\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}
\]

either 2 or -2

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

Perceptron Rule: If \(\mathbf{x}^{(i)}\) is misclassified, do \(\theta \leftarrow \theta + y^{(i)} \mathbf{x}^{(i)}\)
Why the Perceptron Update Works

$\theta_{\text{old}}$

misclassified

Based on slide by Piyush Rai
Why the Perceptron Update Works

• Consider the misclassified example \((y = +1)\)
  – Perceptron wrongly thinks that \(\theta_{\text{old}}^T x < 0\)
• Update:
  \[
  \theta_{\text{new}} = \theta_{\text{old}} + yx = \theta_{\text{old}} + x \quad \text{(since } y = +1)\]
• Note that
  \[
  \theta_{\text{new}}^T x = (\theta_{\text{old}} + x)^T x \\
  = \theta_{\text{old}}^T x + x^T x \quad \|x\|_2^2 > 0
  \]
• Therefore, \(\theta_{\text{new}}^T x\) is less negative than \(\theta_{\text{old}}^T x\).
  – So, we are making ourselves more correct on this example!
The Perceptron Cost Function

• Prediction is correct if \( y^{(i)} x^{(i) \theta} > 0 \)

• Could have used 0/1 loss

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

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

Doesn’t produce a useful gradient
The Perceptron Cost Function

• The perceptron uses the following cost function

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

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

Nice gradient

Based on slide by Alan Fern
Online Perceptron Algorithm

Let $\theta \leftarrow [0, 0, \ldots, 0]$
Repeat:

1. Receive training example $(x^{(i)}, y^{(i)})$
2. if $y^{(i)} x^{(i)} \theta \leq 0$
   \hspace{1cm} // prediction is incorrect
   \hspace{1cm} $\theta \leftarrow \theta + y^{(i)} x^{(i)}$

**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 -

See the perceptron in action: [www.youtube.com/watch?v=vGwemZhPlsA](http://www.youtube.com/watch?v=vGwemZhPlsA)
Given training data \( \{ (x^{(i)}, y^{(i)}) \}_{i=1}^{n} \)
Let \( \theta \leftarrow [0, 0, \ldots, 0] \)
Repeat:
   Let \( \Delta \leftarrow [0, 0, \ldots, 0] \)
   for \( i = 1 \ldots n \), do
     if \( y^{(i)} x^{(i)} \theta \leq 0 \) \hspace{1cm} \text{// prediction for } i^{th} \text{ instance is incorrect}
     \( \Delta \leftarrow \Delta + y^{(i)} x^{(i)} \)
     \( \Delta \leftarrow \Delta / n \) \hspace{1cm} \text{// compute average update}
   \( \theta \leftarrow \theta + \alpha \Delta \)
Until \( \| \Delta \|_2 < \epsilon \)

- 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 $\theta$‘s during training
• The standard Perceptron simply uses the final $\theta$ at test time
  – This may sometimes not be a good idea!
  – Some other $\theta$ 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 $\theta$‘s
  – **Voted Perceptron:** vote on predictions of the intermediate $\theta$‘s
  – **Averaged Perceptron:** average the intermediate $\theta$‘s

Based on slide by Piyush Rai