The Perceptron

CMSC 422

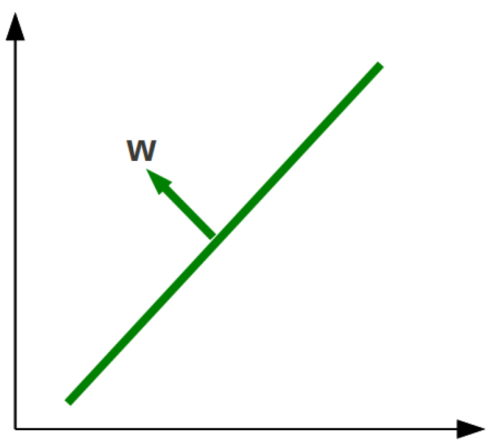
SOHEIL FEIZI
sfeizi@cs.umd.edu

Slides adapted from MARINE CARPUAT

This week

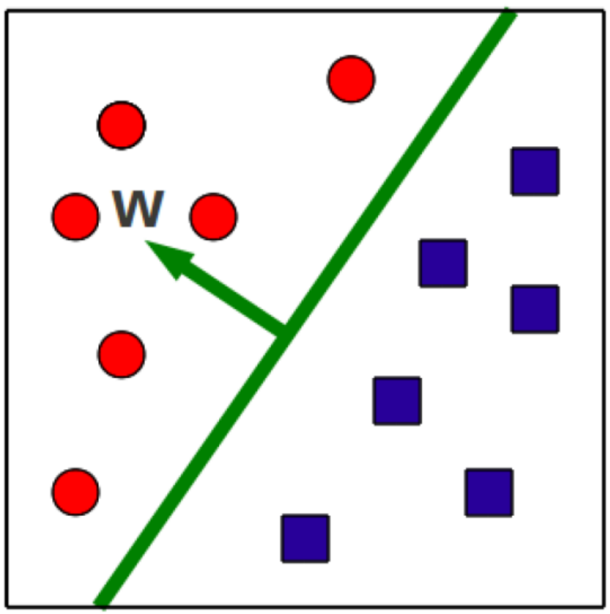
- A new model/algorithm
  - the perceptron
  - and its variants: voted, averaged
- Fundamental Machine Learning Concepts
  - Online vs. batch learning
  - Error-driven learning

Geometry concept: **Hyperplane**



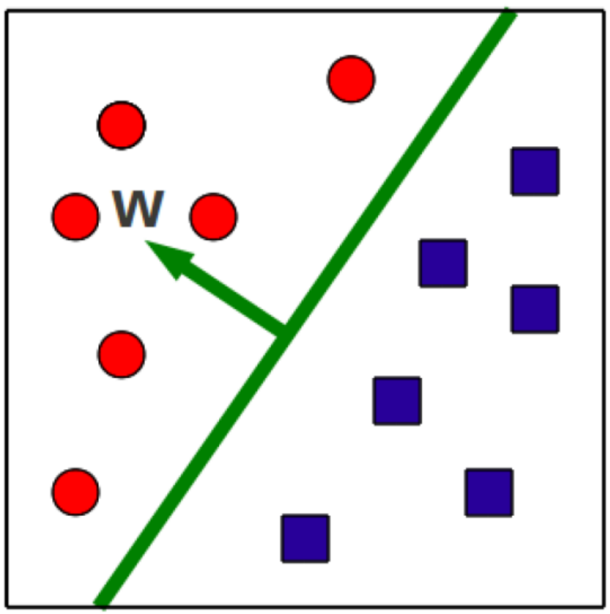
- Separates a D-dimensional space into two half-spaces
- Defined by an outward pointing normal vector $w \in \mathbb{R}^D$
  - w is **orthogonal** to any vector lying on the hyperplane
- Hyperplane passes through the origin, unless we also define a **bias** term b

Binary classification via hyperplanes



- Let's assume that the decision boundary is a hyperplane
- Then, training consists in finding a hyperplane w that separates positive from negative examples

Binary classification via hyperplanes



- At test time, we check on what side of the hyperplane examples fall

$$\hat{y} = sign(w^T x + b)$$

Function Approximation with Perceptron

Problem setting

- Set of possible instances X
  - Each instance $x \in X$ is a feature vector $x = [x_1, \dots, x_D]$
- Unknown target function $f: X \rightarrow Y$
  - Y is binary valued {-1; +1}
- Set of function hypotheses $H = \{h \mid h: X \rightarrow Y\}$
  - Each hypothesis h is a hyperplane in D-dimensional space

Input

- Training examples $\{(x^{(1)}, y^{(1)}), \dots (x^{(N)}, y^{(N)})\}$ of unknown target function f

Output

- Hypothesis $h \in H$ that best approximates target function f

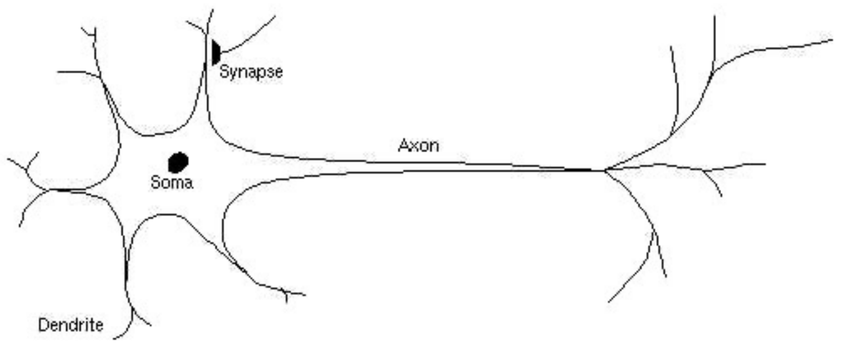
Perception: Prediction Algorithm

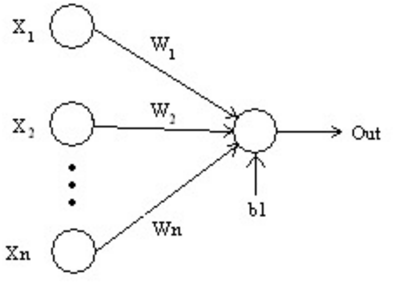
Algorithm 6 PerceptronTest($w_0, w_1, \dots, w_D, b, \hat{\boldsymbol{x}}$)

$a \leftarrow \sum_{d=1}^{D} w_d \, \hat{x}_d + b$ // compute activation for the test example

return sign(a)

Aside: biological inspiration





Analogy: the perceptron as a neuron

Perceptron Training Algorithm

Algorithm 5 PerceptronTrain(**D**, *MaxIter*)

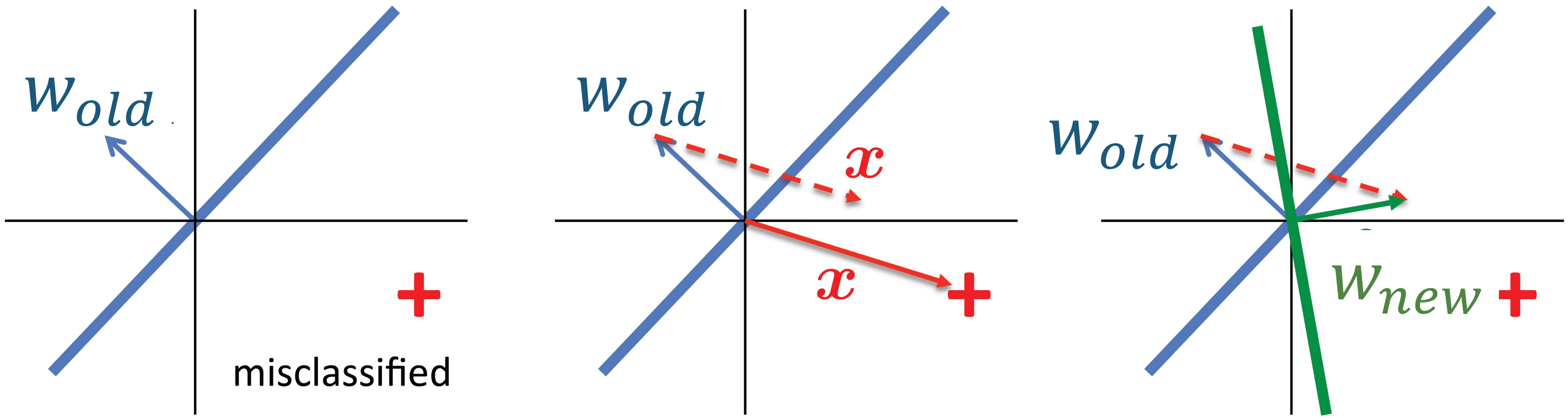
$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + y x_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Properties of the Perceptron training algorithm

- Online
  - We look at one example at a time, and update the model as soon as we make an error
  - As opposed to batch algorithms that update parameters after seeing the entire training set

- Error-driven
  - We only update parameters/model if we make an error

Perceptron update: geometric interpretation



Practical considerations

- The order of training examples matters!
  - Random is better
- Early stopping
  - Good strategy to avoid overfitting
- Simple modifications dramatically improve performance
  - voting or averaging

Standard Perceptron: predict based on final parameters

Algorithm 5 PerceptronTrain($\mathbf{D}$, $MaxIter$)

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $iter = 1 \dots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + yx_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Predict based on final + intermediate parameters

- The voted perceptron

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \text{sign}\left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

- The averaged perceptron

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

- Require keeping track of "survival time" of weight vectors $c^{(1)}, \ldots, c^{(K)}$

Averaged perceptron decision rule

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} c^{(k)} \left(\boldsymbol{w}^{(k)} \cdot \hat{\boldsymbol{x}} + b^{(k)}\right)\right)$$

can be rewritten as

$$\hat{y} = \text{sign}\left(\left(\sum_{k=1}^{K} c^{(k)} \boldsymbol{w}^{(k)}\right) \cdot \hat{\boldsymbol{x}} + \sum_{k=1}^{K} c^{(k)} b^{(k)}\right)$$

Can the perceptron always find a hyperplane to separate positive from negative examples?

Convergence of Perceptron

- The perceptron has converged if it can classify every training example correctly
  – i.e. if it has found a hyperplane that correctly separates positive and negative examples

- Under which conditions does the perceptron converge and how long does it take?

Convergence of Perceptron

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$,

Then **perceptron training converges after $\frac{R^2}{\gamma^2}$ errors** during training

(assuming ($||x|| < R$) for all x).

Margin of a data set *D*

$$margin(\mathbf{D}, w, b) = \begin{cases} \min_{(x,y)\in\mathbf{D}} y(w \cdot x + b) & \text{if } w \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases} \quad (4.8)$$

Distance between the hyperplane (w,b) and the nearest point in **D**

$$margin(\mathbf{D}) = \sup_{w,b} margin(\mathbf{D}, w, b) \quad (4.9)$$

Largest attainable margin on **D**

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$, then **perceptron training converges after $\frac{R^2}{\gamma^2}$ errors** during training (assuming ($||x|| < R$) for all x).

Proof:

- Margin of $\mathbf{w}_*$ on any *arbitrary example* $(\mathbf{x}_n, y_n)$: $\frac{y_n \mathbf{w}_*^T \mathbf{x}_n}{||\mathbf{w}_*||} = y_n \mathbf{w}_*^T \mathbf{x}_n \geq \gamma$
- Consider the $(k+1)^{th}$mistake: $y_n \mathbf{w}_k^T \mathbf{x}_n \leq 0$, and update $\mathbf{w}_{k+1} = \mathbf{w}_k + y_n \mathbf{x}_n$
- $\mathbf{w}_{k+1}^T \mathbf{w}_* = \mathbf{w}_k^T \mathbf{w}_* + y_n \mathbf{w}_*^T \mathbf{x}_n \geq \mathbf{w}_k^T \mathbf{w}_* + \gamma$ (why is this nice?)
- Repeating iteratively k times, we get $\mathbf{w}_{k+1}^T \mathbf{w}_* > k\gamma$ (1)
- $||\mathbf{w}_{k+1}||^2 = ||\mathbf{w}_k||^2 + 2y_n \mathbf{w}_k^T \mathbf{x}_n + ||\mathbf{x}||^2 \leq ||\mathbf{w}_k||^2 + R^2$ (since $y_n \mathbf{w}_k^T \mathbf{x}_n \leq 0$)
- Repeating iteratively k times, we get $||\mathbf{w}_{k+1}||^2 \leq kR^2$ (2)

Theorem (Block & Novikoff, 1962)

If the training data $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ is **linearly separable** with margin γ by a unit norm hyperplane w_* ($||w_*|| = 1$) with $b = 0$, then **perceptron training converges after $\frac{R^2}{\gamma^2}$ errors** during training (assuming ($||x|| < R$) for all x).

What does this mean?

- Perceptron converges quickly when margin is large, slowly when it is small
- Bound does not depend on number of training examples N, nor on number of features
- Proof guarantees that perceptron converges, but not necessarily to the max margin separator

Practical Implications

- Sensitivity to noise
  - if the data is not linearly separable due to noise, no guarantee of convergence or accuracy
- Linear separability in practice
  - Data may be linearly separable in practice
  - Especially when number of features >> number of examples
- Risk of overfitting mitigated by
  - Early stopping
  - Averaging

What you should know

- Perceptron concepts
  - training/prediction algorithms (standard, voting, averaged)
  - convergence theorem and what practical guarantees it gives us
  - how to draw/describe the decision boundary of a perceptron classifier
- Fundamental ML concepts
  - Determine whether a data set is linearly separable and define its margin
  - Error driven algorithms, online vs. batch algorithms

This week

- A new model/algorithm
  - the perceptron
  - and its variants: voted, averaged
- Fundamental Machine Learning Concepts
  - Online vs. batch learning
  - Error-driven learning