Slides adapted from Prof. Carpuat

CMSC 422 Introduction to Machine Learning

Lecture 8 The Perceptron

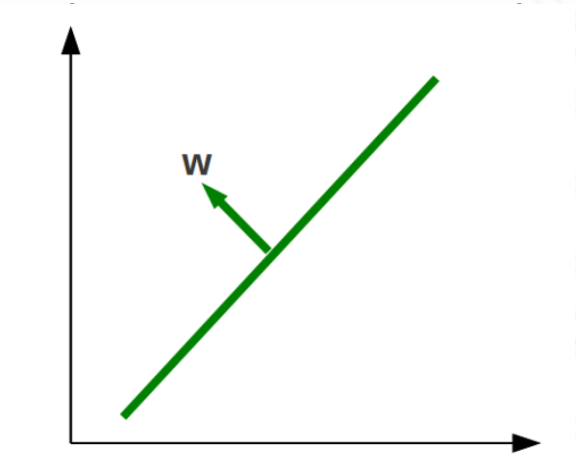
Furong Huang / furongh@umd.edu

UNIVERSITY OF MARYLAND

This week

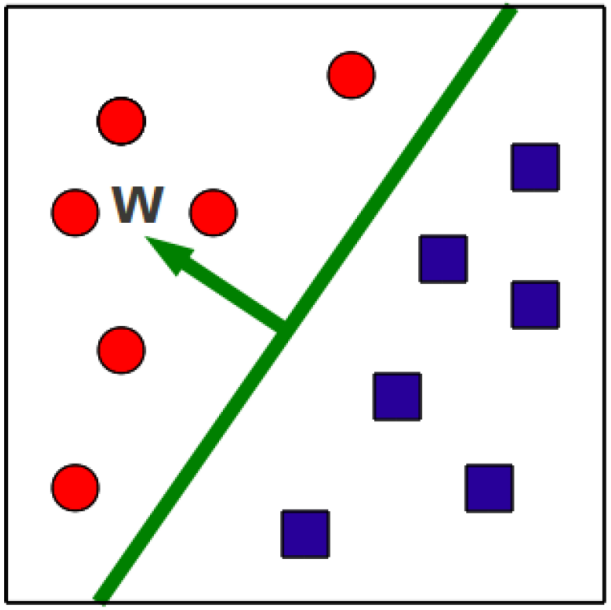
- A new model/algorithm
  - the perceptron
  - and its variants: voted, averaged
- Fundamental Machine Learning Concepts
  - Online vs. batch learning
  - Error-driven learning

Geometry concept: Hyperplane



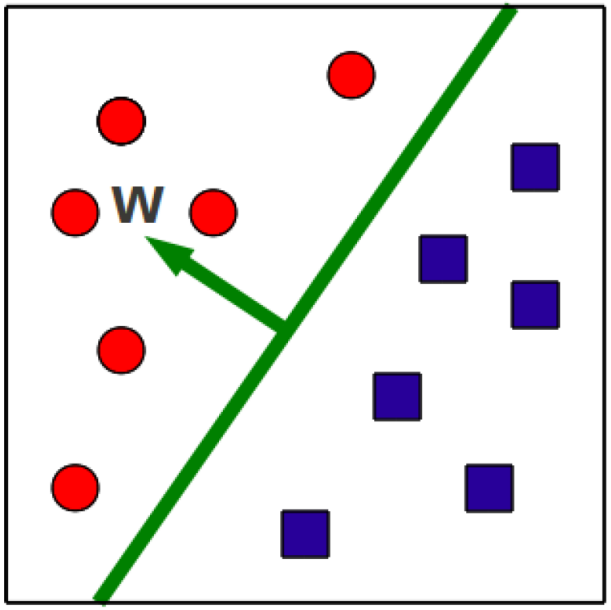
- Separates a D-dimensional space into two half-spaces
- Defined by an outward pointing normal vector $w \in \mathbb{R}^D$
  - w is **orthogonal** to any vector lying on the hyperplane
- Hyperplane passes through the origin, unless we also define a **bias** term b

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

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

Assuming labels are in $\{-1, +1\}$

Function Approximation with Perceptron

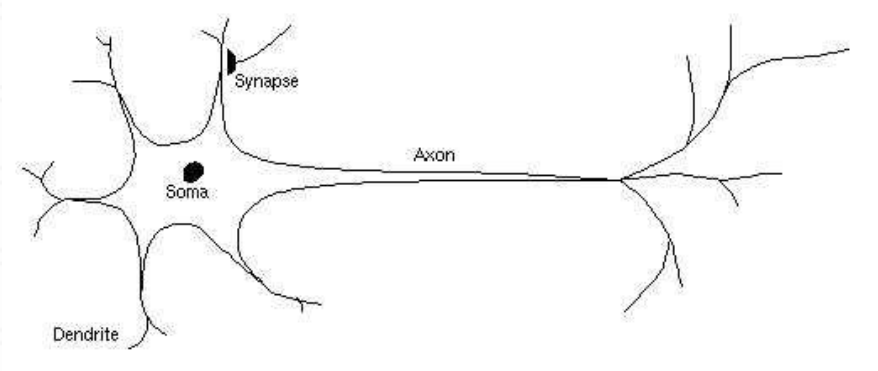
- Problem setting
- Set of possible instances X
  - Each instance $x \in X$ is a feature vector $x = [x_1, \ldots, x_D]$
- Unknown target function $f: X \rightarrow Y$
  - Y is binary valued {-1, +1}
- Set of function hypotheses $H = \{h \mid h: X \rightarrow Y\}$
  - **Each hypothesis h is a hyperplane in D-dimensional space**
- Input
- Training examples $\{(x^{(1)}, y^{(1)}), \ldots (x^{(N)}, y^{(N)})\}$ of unknown target function f
- Output
- Hypothesis $h \in H$ that best approximates target function f

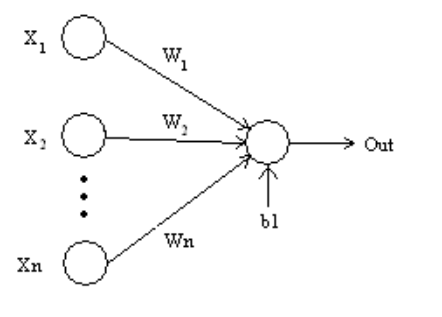
Perception: Prediction Algorithm

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

1: $a \leftarrow \sum_{d=1}^{D} w_d \, \hat{x}_d + b$ // compute activation for the test example
2: **return** SIGN(a)

Aside: biological inspiration





Analogy: the perceptron as a neuron

Learning

- Find algorithm that gives us w and b for a given data set D $(\boldsymbol{x}, y)$
  - Many algorithms possible
- Once we know w and b we can predict the class of a new data point $\boldsymbol{x}_i$ by evaluating

$$\hat{y} = sign(w^\top \boldsymbol{x}_i + b)$$

- We learned a particular way of finding these parameters– via the perceptron update rule
  - Iterative online algorithm–visits all the data over epochs

Perceptron Training Algorithm

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

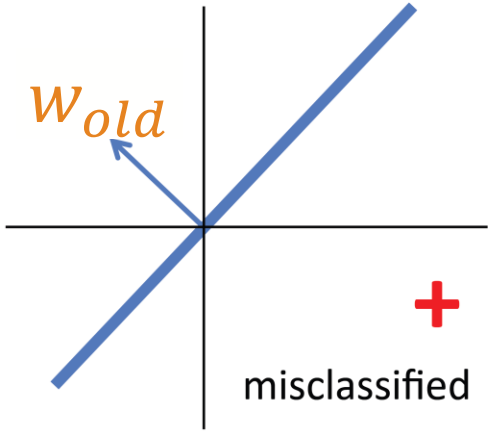
```
w_d ← 0, for all d = 1 ... D                          // initialize weights
b ← 0                                                 // initialize bias
for iter = 1 ... MaxIter do
    for all (x,y) ∈ D do
        a ← Σ_{d=1}^{D} w_d x_d + b                   // compute activation for this example
        if ya ≤ 0 then
            w_d ← w_d + y x_d, for all d = 1 ... D    // update weights
            b ← b + y                                 // update bias
        end if
    end for
end for
return w_0, w_1, ..., w_D, b
```

Perceptron update: geometric interpretation

A training example $(\boldsymbol{x}, y)$ is misclassified, i.e.,

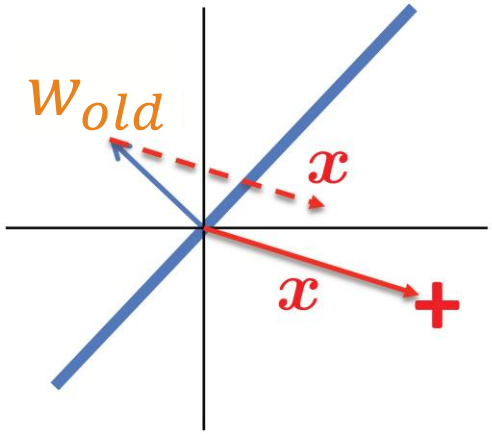
$$sign\left(\boldsymbol{w}_{old}^{\mathsf{T}}\boldsymbol{x} + b\right) \neq y$$

Let's say $y = +1$



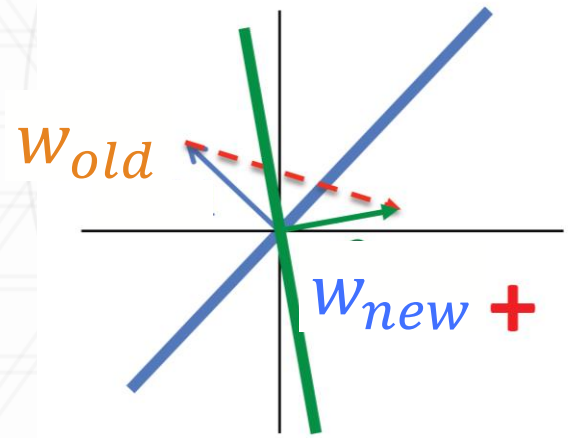
Perceptron update: geometric interpretation

Update: $\boldsymbol{w}_{new} \leftarrow \boldsymbol{w}_{old} + y\boldsymbol{x}$, i.e., $\boldsymbol{w}_{new} \leftarrow \boldsymbol{w}_{old} + \boldsymbol{x}$



Perceptron update: geometric interpretation

Update: $\boldsymbol{w}_{new} \leftarrow \boldsymbol{w}_{old} + y\boldsymbol{x}$, i.e., $\boldsymbol{w}_{new} \leftarrow \boldsymbol{w}_{old} + \boldsymbol{x}$



Standard Perceptron: predict based on final parameters

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

```
1:  w_d ← 0, for all d = 1 ... D                          // initialize weights
2:  b ← 0                                                  // initialize bias
3:  for iter = 1 ... MaxIter do
4:      for all (x,y) ∈ D do
5:          a ← Σ_{d=1}^{D} w_d x_d + b                    // compute activation for this example
6:          if ya ≤ 0 then
7:              w_d ← w_d + y x_d, for all d = 1 ... D     // update weights
8:              b ← b + y                                  // update bias
9:          end if
10:     end for
11: end for
12: return w_0, w_1, ..., 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

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

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 $c^{(1)}, \ldots, c^{(K)}$

How would you modify this algorithm for voted perceptron?

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

1: $w_d \leftarrow 0$, for all $d = 1 \ldots D$ // initialize weights
2: $b \leftarrow 0$ // initialize bias
3: **for** $iter = 1 \ldots MaxIter$ **do**
4: **for all** $(x,y) \in \mathbf{D}$ **do**
5: $a \leftarrow \sum_{d=1}^{D} w_d x_d + b$ // compute activation for this example
6: **if** $ya \leq 0$ **then**
7: $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ // update weights
8: $b \leftarrow b + y$ // update bias
9: **end if**
10: **end for**
11: **end for**
12: **return** $w_0, w_1, \ldots, w_D, b$

How would you modify this algorithm for averaged perceptron?

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

1: $w_d \leftarrow 0$, for all $d = 1 \ldots D$ // initialize weights
2: $b \leftarrow 0$ // initialize bias
3: **for** $iter = 1 \ldots MaxIter$ **do**
4: **for all** $(x,y) \in \mathbf{D}$ **do**
5: $a \leftarrow \sum_{d=1}^{D} w_d x_d + b$ // compute activation for this example
6: **if** $ya \leq 0$ **then**
7: $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ // update weights
8: $b \leftarrow b + y$ // update bias
9: **end if**
10: **end for**
11: **end for**
12: **return** $w_0, w_1, \ldots, w_D, b$

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

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

Averaged Perceptron: predict based on average of intermediate parameters

Algorithm 7 AVERAGEDPERCEPTRONTRAIN(**D**, *MaxIter*)

```
1:  w ← ⟨0, 0, ... 0⟩   ,   b ← 0                    // initialize weights and bias
2:  u ← ⟨0, 0, ... 0⟩   ,   β ← 0                    // initialize cached weights and bias
3:  c ← 1                                            // initialize example counter to one
4:  for iter = 1 ... MaxIter do
5:      for all (x,y) ∈ D do
6:          if y(w · x + b) ≤ 0 then
7:              w ← w + y x                          // update weights
8:              b ← b + y                            // update bias
9:              u ← u + y c x                        // update cached weights
10:             β ← β + y c                          // update cached bias
11:         end if
12:         c ← c + 1                                // increment counter regardless of update
13:     end for
14: end for
15: return w - (1/c) u, b - (1/c) β                  // return averaged weights and bias
```

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

Covered

A new model/algorithm

- the perceptron
- and its variants: voted, averaged

Fundamental Machine Learning Concepts

- Online vs. batch learning
- Error-driven learning

New

- The perception: a new model/algorithm
  - its variants: voted, averaged
  - **convergence proof**
- Fundamental Machine Learning Concepts
  - Online vs. batch learning
  - Error-driven learning
  - **Linear separability and margin of a dataset**

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?

Margin of a data set *D (IMPORTANT)*

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

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

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

Largest attainable margin on **D**

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).

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).

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)
- $||\mathbf{w}_{k+1}||^2 = ||\mathbf{w}_k||^2 + 2 y_n \mathbf{w}_k^T \mathbf{x}_n + ||\mathbf{x}||^2 \leq ||\mathbf{w}_k||^2 + R^2$ (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 $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ 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).

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 d
- **Proof guarantees that perceptron converges, but not necessarily to the max margin separator**

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

UNIVERSITY OF
MARYLAND

Furong Huang
4124 IRB, College Park, MD 20740
301.405.8010 / furongh@umd.edu