NLP 202: Structured Perceptron

Jeffrey Flanigan

Winter 2023

University of California Santa Cruz
jmflanig@ucsc.edu

- We have seen the simplest method for dependency parsing: *transition-based dependency parsing*
- Next simplest method: *graph-based dependency parsing with the structured perceptron*

  I first need to introduce a new learning algorithm: the **Perceptron algorithm**

The Perceptron

REPORT NO. 85-460-1

THE PERCEPTRON

A PERCEIVING AND RECOGNIZING AUTOMATON

(PROJECT PARA)

January, 1957

Prepared by: Frank Rosenblatt

Frank Rosenblatt,
Project Engineer

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

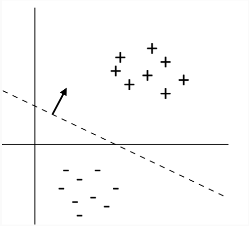
F. ROSENBLATT

Cornell Aeronautical Laboratory

The Perceptron algorithm

- Rosenblatt 1958
  - (Though there were some hints of a similar idea earlier, eg: Agmon 1954)
- The goal is to find a separating hyperplane
  - For separable data, guaranteed to find one
- An online algorithm
  - Processes one example at a time
- Several variants exist
  - We will see these briefly at towards the end

Learning setup: Binary classification



- Training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in R^n, y_i \in \{-1, 1\}$
- Make predictions: 1 if $\mathbf{w}^T\mathbf{x}_i \geq 0$, -1 otherwise Another way to write this: $\text{sign}(\mathbf{w}^T\mathbf{x})$
- Want to learn the weights $\mathbf{w}$
  Note: by include a constant feature of 1, the bias term can be "folded-in" to $\mathbf{w}$

Perceptron Algorithm

- **Decision rule**: $\text{sign}(\mathbf{w}^T \mathbf{x}_i)$
- **Learning rule**: If incorrect:
    - if positive, $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i$
      if negative, $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_i$
      Another way to write this: $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$, $y_i \in \{-1, 1\}$
- **Guaranteed to eventually correctly classify the data if the data are linearly separable**

Perceptron Algorithm

- **Decision rule**: $\text{sign}(\mathbf{w}^T\mathbf{x}_i)$
- **Learning rule**: If incorrect:
  - if positive, $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i$
    if negative, $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_i$
    Another way to write this: $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$, $y_i \in \{-1, 1\}$
- **Guaranteed to eventually correctly classify the data if the data are linearly separable**

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \neq \text{sign}(\mathbf{w}^T\mathbf{x})$ (shorthand: $y_i\mathbf{w}^T\mathbf{x}_i \leq 0$), then:
        Update $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$
3. Return $\mathbf{w}$

Perceptron Algorithm

- **Decision rule**: sign$(\mathbf{w}^T\mathbf{x}_i)$
- **Learning rule**: If incorrect:
  - if positive, $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i$
    if negative, $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_i$
    Another way to write this: $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$, $y_i \in \{-1, 1\}$
- **Guaranteed to eventually correctly classify the data if the data are linearly separable**

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \neq$ sign$(\mathbf{w}^T\mathbf{x})$ (shorthand: $y_i\mathbf{w}^T\mathbf{x}_i \leq 0$), then:
        Update $\mathbf{w} \leftarrow \mathbf{w} + ry_i\mathbf{x}_i$ Can also have a stepsize (learning rate) r. If constant, can set to 1
3. Return $\mathbf{w}$

Intuition behind the update

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Suppose we have made a mistake on a positive example

That is, $y = +1$ and $\mathbf{w}_t^{\mathrm{T}}\mathbf{x} \leq 0$

Call the new weight vector $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ (say r = 1)

The new dot product is $\mathbf{w}_{t+1}^{\mathrm{T}}\mathbf{x} = (\mathbf{w}_t + \mathbf{x})^{\mathrm{T}}\mathbf{x} = \mathbf{w}_t^{\mathrm{T}}\mathbf{x} + \mathbf{x}^{\mathrm{T}}\mathbf{x} \geq \mathbf{w}_{\mathbf{t}}^{\mathrm{T}}\mathbf{x}$

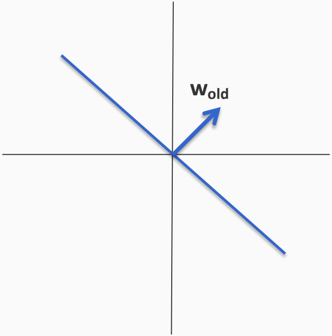
For a positive example, the Perceptron update will increase the score assigned to the same input

Similar reasoning for negative examples

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

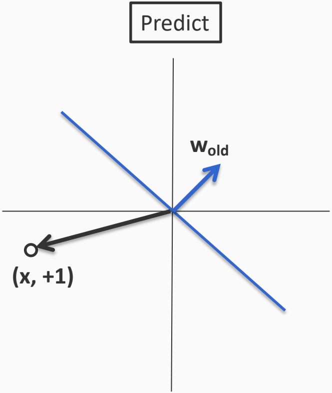
Geometry of the perceptron update

Predict



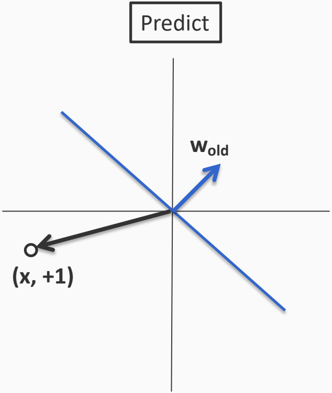
Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update



Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

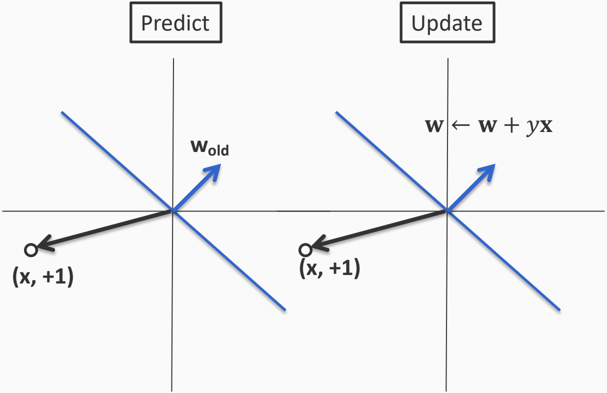
Geometry of the perceptron update



For a mistake on a positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

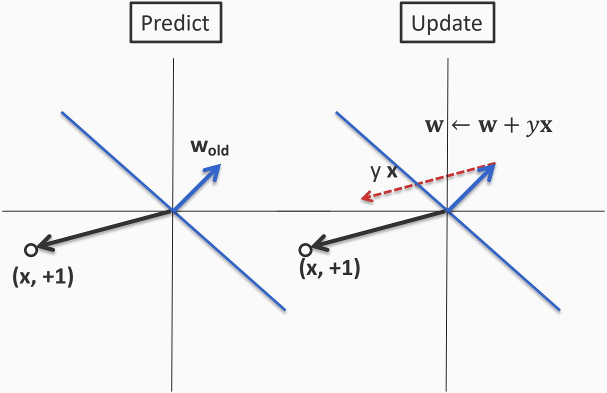
Geometry of the perceptron update



For a mistake on a positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

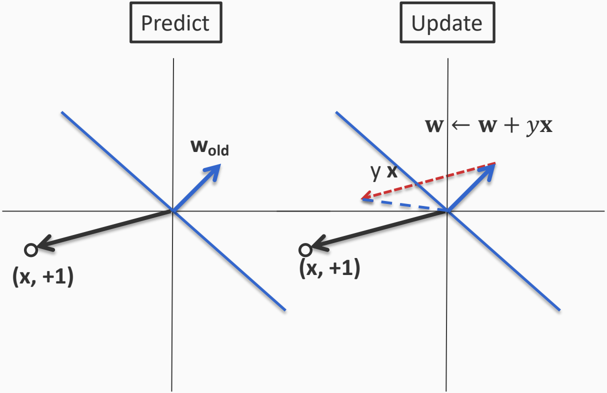
Geometry of the perceptron update



For a mistake on a positive example

Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

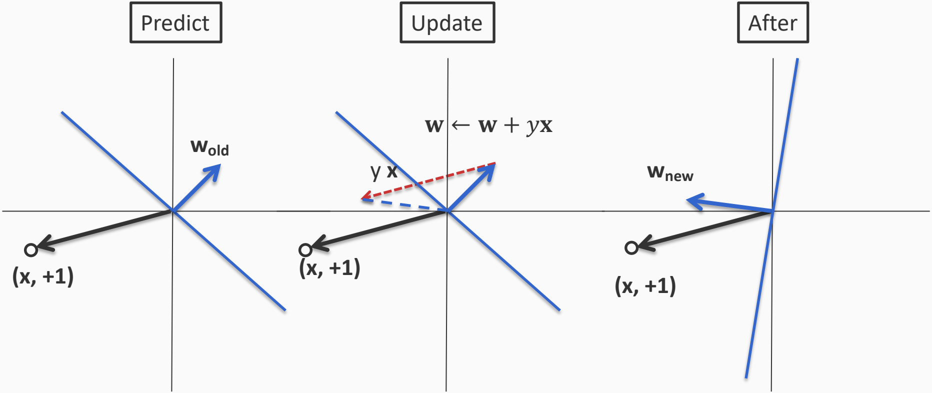
Geometry of the perceptron update



For a mistake on a positive example

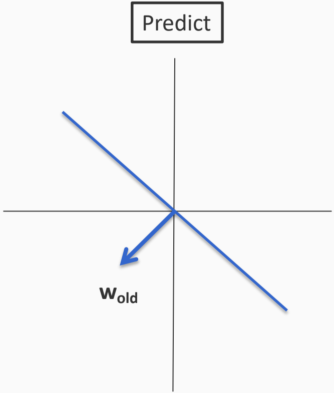
Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + r\mathbf{x}_i$
Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - r\mathbf{x}_i$

Geometry of the perceptron update

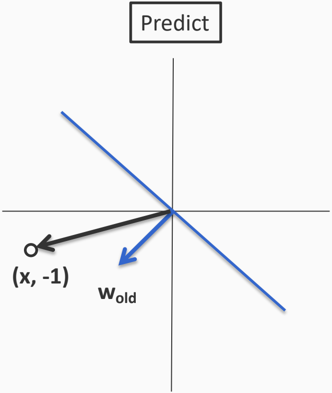


For a mistake on a positive example

Geometry of the perceptron update

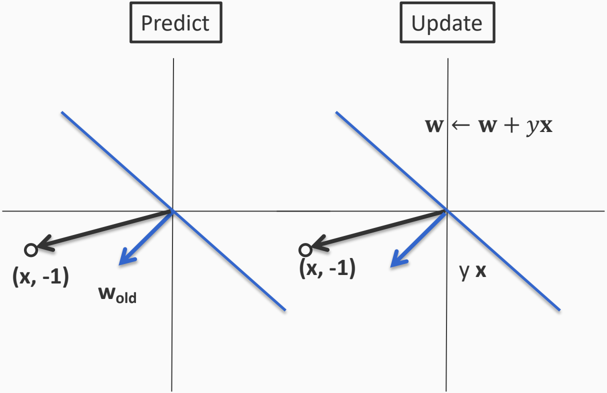


Geometry of the perceptron update



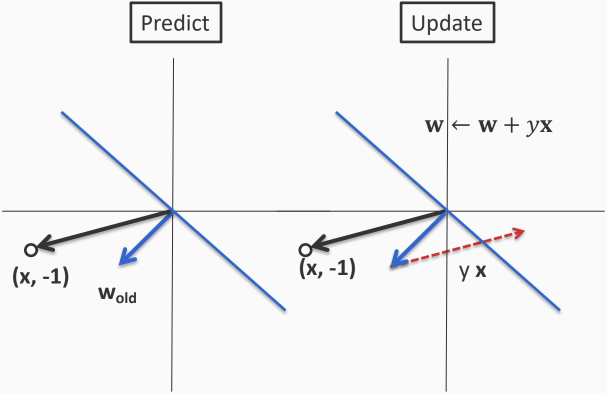
For a mistake on a negative example

Geometry of the perceptron update



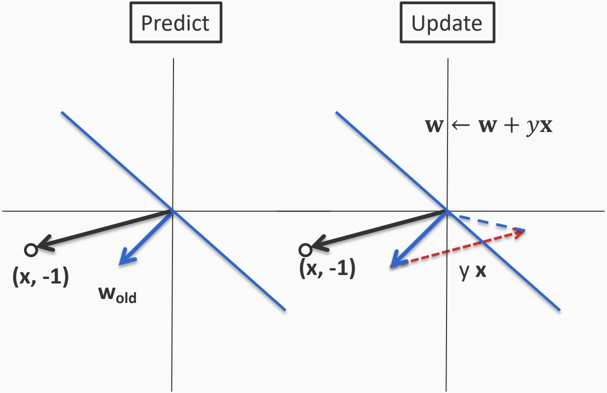
For a mistake on a negative example

Geometry of the perceptron update



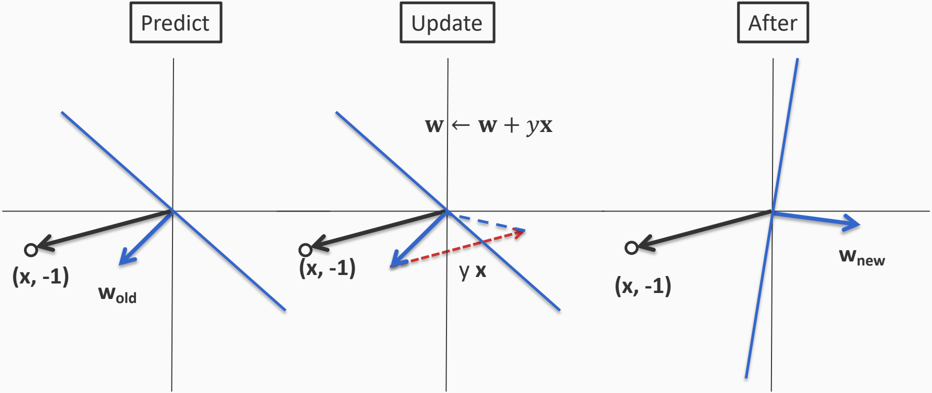
For a mistake on a negative example

Geometry of the perceptron update



For a mistake on a negative example

Geometry of the perceptron update



For a mistake on a negative example

Variants of the Perceptron: voting and averaging

1. The “standard” algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^n, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$
2. For epoch in $1 \cdots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
            – update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Prediction on a new example with features $\mathbf{x}$: $\text{sgn}(\mathbf{w}^T \mathbf{x})$

1. The "standard" algorithm

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^n, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$
2. For epoch in $1 \cdots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
            - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

T is a hyper-parameter to the algorithm

Another way of writing that there is an error

Prediction on a new example with features $\mathbf{x}$: $\text{sgn}(\mathbf{w}^T \mathbf{x})$

2. Voting and Averaging

- So far: We return the final weight vector

- Voted perceptron
  - Remember every weight vector in your sequence of updates.
  - At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
  - Comes with strong theoretical guarantees about generalization, impractical because of storage issues

2. Voting and Averaging

- So far: We return the final weight vector

- Voted perceptron
  - Remember every weight vector in your sequence of updates.
  - At final prediction time, each weight vector gets to vote on the label. The number of votes it gets is the number of iterations it survived before being updated
  - Comes with strong theoretical guarantees about generalization, impractical because of storage issues

- Averaged perceptron
  - Instead of using all weight vectors, use the average weight vector (i.e longer surviving weight vectors get more say)
  - More practical alternative and widely used

Averaged Perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^n, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$ and $\mathbf{a} = \mathbf{0} \in \Re^n$
2. For epoch in $1 \cdots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
            - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
        - $\mathbf{a} \leftarrow \boldsymbol{a} + \mathbf{w}$
3. Return $\mathbf{a}$

Prediction on a new example with features $\mathbf{x}$: $\text{sgn}(\mathbf{a}^T \mathbf{x})$

Averaged Perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^n, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$ and $\mathbf{a} = \mathbf{0} \in \Re^n$
2. For epoch in $1 \cdots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
        - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
      - $\mathbf{a} \leftarrow \boldsymbol{a} + \mathbf{w}$
3. Return $\mathbf{a}$

This is the simplest version of the averaged perceptron

There are some easy programming tricks to make sure that **a** is also updated only when there is an error

Prediction on a new example with features $\mathbf{x}$: $\mathrm{sgn}(\mathbf{a}^T \mathbf{x})$

Averaged Perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \Re^n, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in \Re^n$ $\boxed{\text{and } \mathbf{a} = \mathbf{0} \in \Re^n}$
2. For epoch in $1 \cdots T$:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
        - update $\mathbf{w} \leftarrow \mathbf{w} + r y_i \mathbf{x}_i$
      - $\boxed{\mathbf{a} \leftarrow \boldsymbol{a} + \mathbf{w}}$
3. Return $\mathbf{a}$

This is the simplest version of the averaged perceptron

There are some easy programming tricks to make sure that **a** is also updated only when there is an error

If you want to use the Perceptron algorithm, use averaging

Prediction on a new example with features $\mathbf{x}$: $\text{sgn}(\mathbf{a}^T \mathbf{x})$

Analysis: Perceptron

Perceptron Mistake Bound

Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$

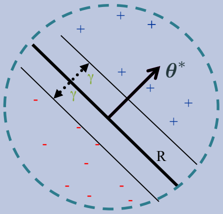


Figure from Nina Balcan

Standard Perceptron Algorithm (Binary Case)

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in R^n, y_i \in \{-1, 1\}$

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, then:
          Update $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$
3. Return $\mathbf{w}$

Prediction on a new example with features $\mathbf{x}$: sign$(\mathbf{w}^T \mathbf{x})$

What about multiclass?

Multiclass Perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in L$ and linear scoring function $\mathbf{w}^T \mathbf{f}(\mathbf{x}_i, y)$

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \neq \hat{y} = \mathrm{argmax}_y \, \mathbf{w}^T \mathbf{f}(\mathbf{x}_i, y)$, then:
          Update $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}_i, y_i) - \mathbf{f}(\mathbf{x}_i, \hat{y})$
3. Return $\mathbf{w}$

Prediction on a new example $\mathbf{x}$: $\hat{y} = \mathrm{argmax}_y \, \mathbf{w}^T \mathbf{f}(\mathbf{x}_i, y)$

Multiclass Perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in L$ and linear scoring function $\mathbf{w}^T\mathbf{f}(\mathbf{x}_i, y)$

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - If $y_i \neq \hat{y} = \mathrm{argmax}_y\, \mathbf{w}^T\mathbf{f}(\mathbf{x}_i, y)$, then (*this check is not necessary, because if same, then update cancels*):
          Update $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}_i, y_i) - \mathbf{f}(\mathbf{x}_i, \hat{y})$
3. Return $\mathbf{w}$

Prediction on a new example $\mathbf{x}$: $\hat{y} = \mathrm{argmax}_y\, \mathbf{w}^T\mathbf{f}(\mathbf{x}_i, y)$

Multiclass Perceptron

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in L$ and linear scoring function $\mathbf{w}^T\mathbf{f}(\mathbf{x}_i, y)$

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - $\hat{y} = \mathrm{argmax}_y\, \mathbf{w}^T\mathbf{f}(\mathbf{x}_i, y)$
        - Update $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}_i, y_i) - \mathbf{f}(\mathbf{x}_i, \hat{y})$
3. Return $\mathbf{w}$

Prediction on a new example $\mathbf{x}$: $\hat{y} = \mathrm{argmax}_y\, \mathbf{w}^T\mathbf{f}(\mathbf{x}_i, y)$

Can we use the perceptron for **structured outputs**?

For example, sequence labeling and dependency parsing?

Structured Perceptron (with linear scoring function)

Given a training set $D = \{(\mathbf{x}_i, y_i)\}$ where $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}(\mathbf{x}_i)$ (*output space depends on input*) and linear scoring function $\mathbf{w}^T \mathbf{f}(\mathbf{x}_i, \mathbf{y})$

1. Initialize $\mathbf{w} = \mathbf{0} \in R^n$
2. For epoch in $1 \ldots T$:
    1. Shuffle the data
    2. For each training example $(\mathbf{x}_i, y_i) \in D$:
        - Make prediction $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y}} \, \mathbf{w}^T \mathbf{f}(\mathbf{x}_i, \mathbf{y})$
        - Update $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{f}(\mathbf{x}_i, \mathbf{y}_i) - \mathbf{f}(\mathbf{x}_i, \hat{\mathbf{y}})$
3. Return $\mathbf{w}$

Prediction on a new example $\mathbf{x}$: $\hat{\mathbf{y}} = \mathrm{argmax}_{\mathbf{y}} \, \mathbf{w}^T \mathbf{f}(\mathbf{x}_i, \mathbf{y})$

Another view of the Perceptron

It turns out, the Perceptron algorithm is exactly stochastic (sub)-gradient descent on a loss function!

The Perceptron minimizes:

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, y') \right) - \mathbf{w} \cdot f(\boldsymbol{x}_i, y_i)$$

Minimizing the Perceptron Loss: The Perceptron Algorithm

Perceptron solves the following minization problem:

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, y') \right) - \mathbf{w} \cdot f(\boldsymbol{x}_i, y_i)$$

Stochastic subgradient descent (SSGD) with stepsize α on the above is the **Perceptron** algorithm!

- For $i \in \{1, \ldots, N\}$:
  - Shuffle the training data, and for each example i do the following update:
    - $\hat{y}_i \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, y)$
    - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(f(\boldsymbol{x}_i, \hat{y}) - f(\boldsymbol{x}_i, y_i) \right)$

Minimizing Perceptron Loss

$$\left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

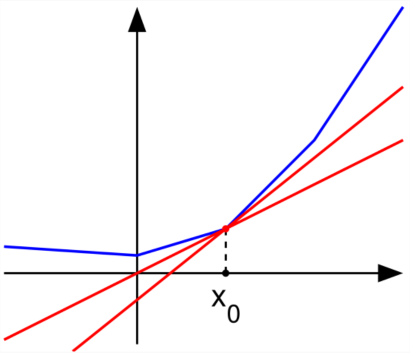
Minimizing Perceptron Loss

$$\left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

When two labels are *tied*, the function is not differentiable.

Solution: (stochastic) subgradient descent!

Subgradient



c is a **subgradient** of $f(x)$ at x_0 if $\forall x$:

$$f(x) - f(x_0) \geq c(x - x_0)$$

(Only defined for convex functions.)

Gradient of Perceptron Loss

For the ith data point the subgradient is:

$$\begin{aligned} g_i^j &= \frac{\partial}{\partial \mathbf{w}_j} \left[\left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(x_i, y') \right) - \mathbf{w} \cdot f(x_i, y_i) \right] \\ &= f^j(x_i, \hat{y}) - f^j(x_i, y_i) \end{aligned}$$

where

$$\hat{y} = \operatorname*{argmax}_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(x_i, y')$$

Minimizing the Perceptron Loss: The Perceptron Algorithm

Perceptron solves the following minization problem:

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, y') \right) - \mathbf{w} \cdot f(\boldsymbol{x}_i, y_i)$$

Stochastic subgradient descent (SSGD) with stepsize α on the above is the **Perceptron** algorithm!

- For $i \in \{1, \ldots, N\}$:
  - Shuffle the training data, and for each example i do the following update:
    - $\hat{y}_i \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}_i, y)$
    - $\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(f(\boldsymbol{x}_i, \hat{y}) - f(\boldsymbol{x}_i, y_i) \right)$

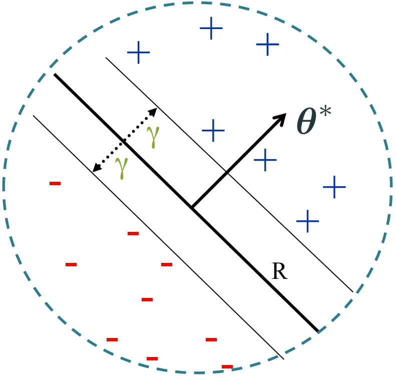
Proof the the Perceptron mistake bound (not covered in class).

Analysis: Perceptron

Perceptron Mistake Bound

Guarantee: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



Analysis: Perceptron

Perceptron Mistake Bound

Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$

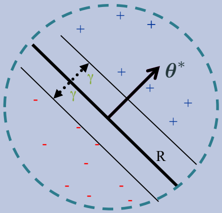


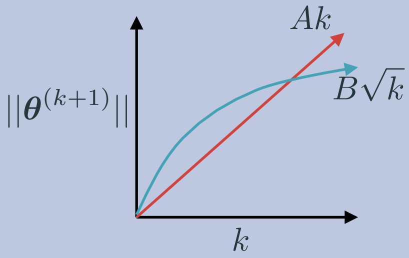
Figure from Nina Balcan

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

We will show that there exist constants A and B s.t.

$$Ak \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$$



Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 1: for some A, $Ak \leq ||\boldsymbol{\theta}^{(k+1)}||$

$$\boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* = (\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)})\boldsymbol{\theta}^*$$

by Perceptron algorithm update

$$= \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)})$$

$$\geq \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + \gamma$$

by assumption

$$\Rightarrow \boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* \geq k\gamma$$

by induction on k since $\theta^{(1)} = \mathbf{0}$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \geq k\gamma$$

since $\underbrace{||\mathbf{w}|| \times ||\mathbf{u}|| \geq \mathbf{w} \cdot \mathbf{u}}_{\text{Cauchy-Schwartz inequality}}$ and $||\theta^*|| = 1$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 2: for some B, $||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$

$$||\boldsymbol{\theta}^{(k+1)}||^2 = ||\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}||^2$$

by Perceptron algorithm update

$$= ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2 + 2y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)})$$

$$\leq ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2$$

since kth mistake $\Rightarrow y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$

$$= ||\boldsymbol{\theta}^{(k)}||^2 + R^2$$

since $(y^{(i)})^2||\mathbf{x}^{(i)}||^2 = ||\mathbf{x}^{(i)}||^2 = R^2$ by assumption and $(y^{(i)})^2 = 1$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}||^2 \leq kR^2$$

by induction on k since $(\theta^{(1)})^2 = 0$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 3: Combining the bounds finishes the proof.

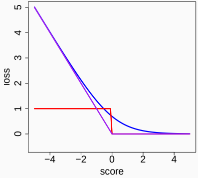
$$k\gamma \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

$$\Rightarrow k \leq \underbrace{(R/\gamma)^2}_{\text{The total number of mistakes must be less than this}}$$

Log Loss and Perceptron Loss for (x, y)

$$\text{log loss:} \quad \left(\log \sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$

$$\text{Perceptron loss:} \quad \left(\max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(\boldsymbol{x}, y')\right) - \mathbf{w} \cdot f(\boldsymbol{x}, y)$$



In purple is the hinge loss, in blue is the log loss; in red is the "zero-one" loss (error).