The Perceptron

Jiayu Zhou

[1]Department of Computer Science and Engineering
Michigan State University
East Lansing, MI USA

Case Study: Credit Approval

age	32 years
gender	male
salary	40,000
debt	26,000
years in job	1 year
years at home	3 years
...	...

Approve for credit?

Case Study: Credit Approval

age	32 years
gender	male
salary	40,000
debt	26,000
years in job	1 year
years at home	3 years
...	...

Approve for credit?

- Using salary, debt, years in residence, etc. approve for credit or not
- No magic credit approval formula
- Banks have lots of data
  - customer information: salary, debt, etc.
  - whether or not they defaulted on their credit.

Case Study: Credit Approval

age	32 years
gender	male
salary	40,000
debt	26,000
years in job	1 year
years at home	3 years
...	...

Approve for credit?

- Using salary, debt, years in residence, etc. approve for credit or not
- No magic credit approval formula
- Banks have lots of data
  - customer information: salary, debt, etc.
  - whether or not they defaulted on their credit.

A pattern exists. We don't know it. We have data to learn it.

The Key Players

- Salary, debt, years in residence
- Approve credit or not
- True relationship between $\mathbf{x}$ and y
- Data on customers

- input $\mathbf{x} \in \mathbb{R}^d \equiv \mathcal{X}$
- output $y \in \{-1, +1\} \equiv \mathcal{Y}$
- target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is fixed but unknown

The Key Players

- Salary, debt, years in residence
- Approve credit or not
- True relationship between $\mathbf{x}$ and y
- Data on customers

- input $\mathbf{x} \in \mathbb{R}^d \equiv \mathcal{X}$
- output $y \in \{-1, +1\} \equiv \mathcal{Y}$
- target function $f : \mathcal{X} \to \mathcal{Y}$
- $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is fixed but unknown

Goal: learn the function f from the data $\mathcal{D}$.

Learning

- Start with a set of candidate hypothese $\mathcal{H}$ which you think are likely to represent f.

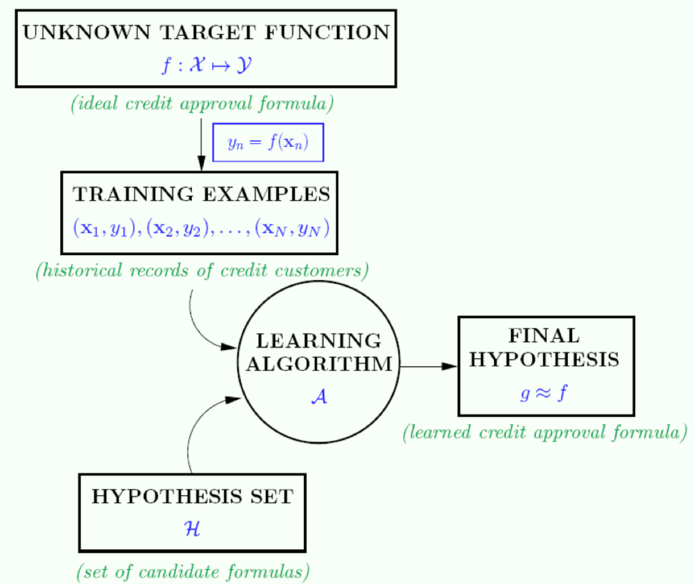
$$\mathcal{H} = \{h_1, h_2, \ldots, \}$$

  is called the hypothesis set or *model*.
- Select a hypothesis g from $\mathcal{H}$. The way we do this is called a *learning algorithm*.
- Use g for new customers. We hope $g \approx f$.

$\mathcal{X}, \mathcal{Y}$ and $\mathcal{D}$ are *given* by the learning problem;
The target f is **fixed but unknown**

We choose $\mathcal{H}$ and the learning algorithm

Summary of the Learning Setup



A Simple Learning Model

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

A Simple Learning Model

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, Give importance weights to the different features and compute a "Credit Score"

$$\text{“Credit Score”} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold}, (\text{“Credit score” is good})$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold}, (\text{“Credit score” is bad})$$

A Simple Learning Model

- Input vector $\mathbf{x} = [x_1, \ldots, x_d]^T$, Give importance weights to the different features and compute a "Credit Score"

$$\text{"Credit Score"} = \sum_{i=1}^{d} w_i x_i.$$

- Approve credit if the "Credit Score" is acceptable.

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold}, (\text{"Credit score" is good})$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold}, (\text{"Credit score" is bad})$$

- How to choose the importance weights w_i?

$$x_i \text{ is important} \Rightarrow \text{large weight}$$

$$x_i \text{ is beneficial for credit} \Rightarrow \text{positive weight}$$

$$x_i \text{ is detrimental for credit} \Rightarrow \text{negative weight}$$

A Simple Learning Model

$$\text{Approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold}, (\text{“Credit score” is good})$$

$$\text{Deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold}, (\text{“Credit score” is bad})$$

can be written formally as

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$$

The “bias weight” w_0 correspond to the threshold. (How?)

The Perceptron Hypothesis Set

We have defined a Hypothesis set $\mathcal{H}$

$$\mathcal{H} = \left\{ h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{d} w_i x_i + w_0\right) = \text{sign}(\mathbf{w}^T\mathbf{x}) \right\}$$

which is uncountably infinite.

The Perceptron Hypothesis Set

We have defined a Hypothesis set $\mathcal{H}$

$$\mathcal{H} = \left\{ h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{d} w_i x_i + w_0\right) = \text{sign}(\mathbf{w}^T\mathbf{x}) \right\}$$

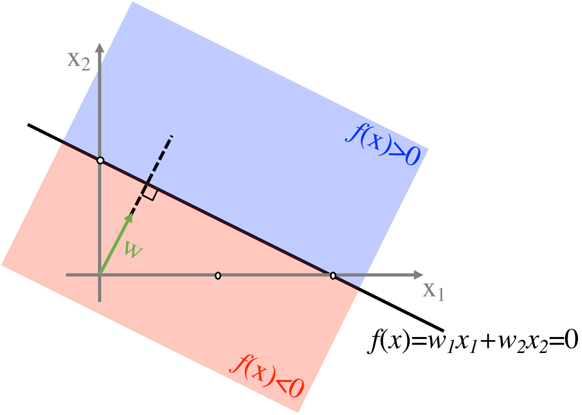
which is uncountably infinite. We have used the "dummy variable":

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} \in \mathbb{R}^{d+1}, \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_d \end{bmatrix} \in \{1\} \times \mathbb{R}^d$$

This hypothesis set is called the *perceptron* or *linear separator*

Geometry of The Perceptron

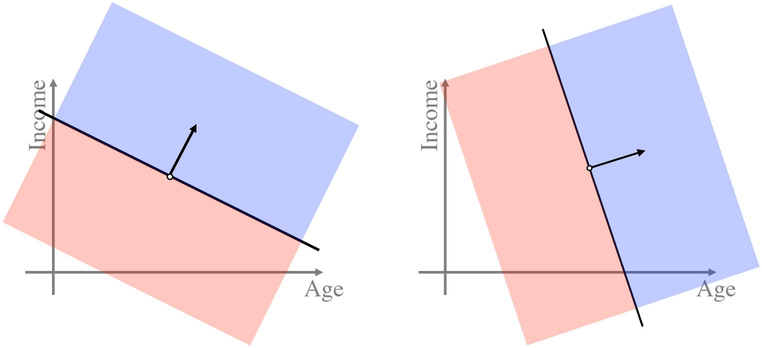
- In 2-d space, $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ defines a line that separates the space.



- In higher dimensional space, this corresponds to a hyperplane.

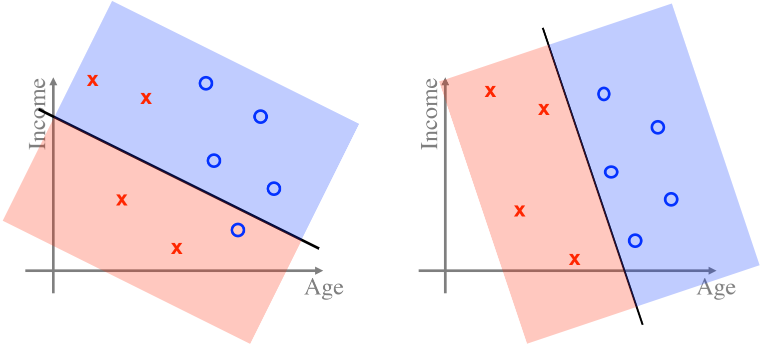
Geometry of The Perceptron

So many choices...



Which one should we pick?

Use the Data to Pick a Line



- A perceptron fits the data by using a line to separate the $+1$ from -1 data.
- **Fitting the data**: How to find a hyperplane that separates the data?

How to Learn a Final Hypothesis g from $\mathcal{H}$

- We want to select $g \in \mathcal{H}$ so that $g \approx f$.
- We certainly want $g \approx f$ on the data set $\mathcal{D}$. Ideally,

$$g(\mathbf{x}_n) = y_n$$

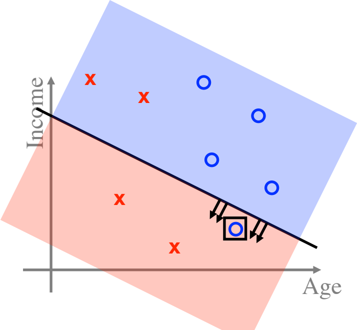
- How do we find such a g in the *infinite* hypothesis set $\mathcal{H}$, if it exists?

How to Learn a Final Hypothesis g from $\mathcal{H}$

- We want to select $g \in \mathcal{H}$ so that $g \approx f$.
- We certainly want $g \approx f$ on the data set $\mathcal{D}$. Ideally,

$$g(\mathbf{x}_n) = y_n$$

- How do we find such a g in the *infinite* hypothesis set $\mathcal{H}$, if it exists?
- **Idea!** Start with some weight vector and try to improve it.



The Perceptron Learning Algorithm (PLA)

A simple iterative method:

1. $\mathbf{w}(1) = 0$
2. for iteration $t = 1, 2, 3, \ldots$
3. the weight vector is $\mathbf{w}(t)$
4. From $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ pick any misclassified sample.
5. Call the misclassified example $(\mathbf{x}_*, y_*)$, i.e.,

$$\text{sign}(\mathbf{w}(t)^T \mathbf{x}_*) \neq y_*$$

6. Update the weight:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y_* \mathbf{x}_*$$

7. $t \leftarrow t + 1$

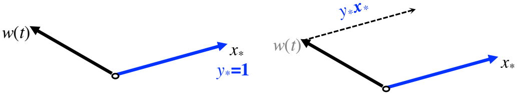
PLA implements our idea: start at some weights and try to improve: **"incremental learning" on a single sample at a time.**

The Perceptron Learning Algorithm (PLA)

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_* \mathbf{x}_*$
- For a positive sample

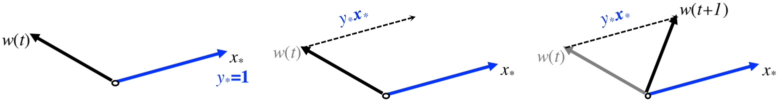
The Perceptron Learning Algorithm (PLA)

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- For a positive sample



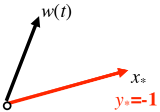
The Perceptron Learning Algorithm (PLA)

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- For a positive sample



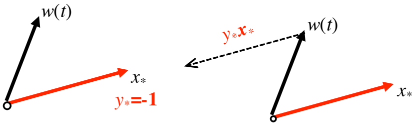
The Perceptron Learning Algorithm (PLA)

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- For a negative sample



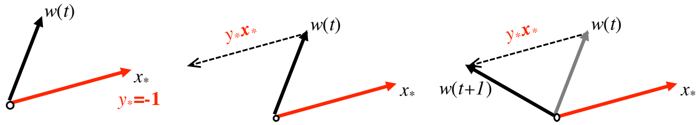
The Perceptron Learning Algorithm (PLA)

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- For a negative sample



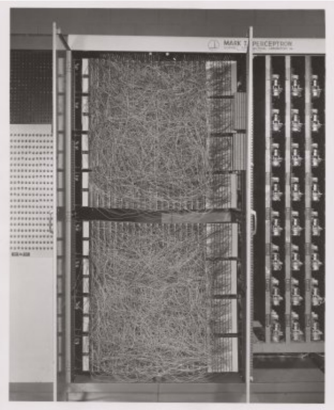
The Perceptron Learning Algorithm (PLA)

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- For a negative sample



The Perceptron Learning Algorithm (PLA)

- Invented in 1957 by Frank Rosenblatt, funded by ONR.
- The perceptron was intended to be a *machine*, rather than a program.
- Mark I perceptron was designed for image recognition: 400 photocells, weight updates during learning were performed by electric motors.



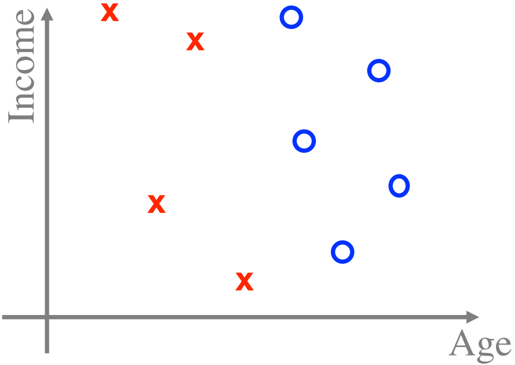
The Perceptron Learning Algorithm (PLA)

"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

— New York Times, 1958

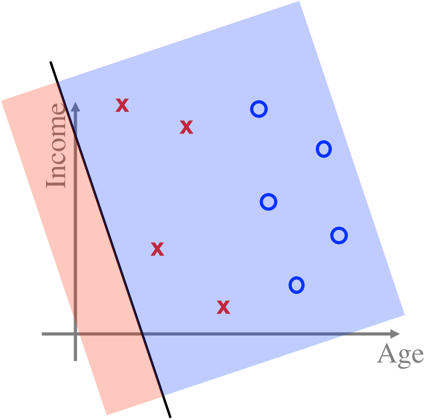
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



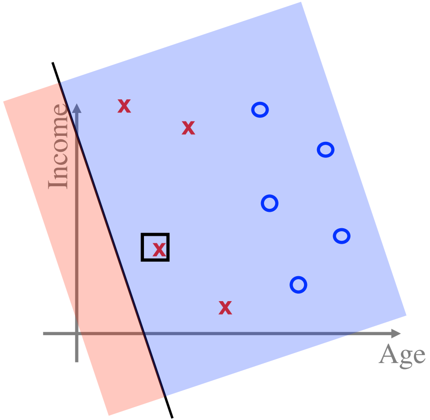
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



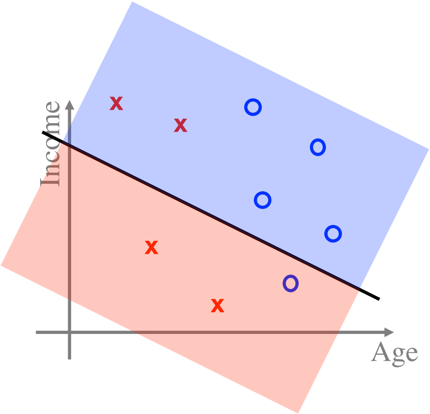
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



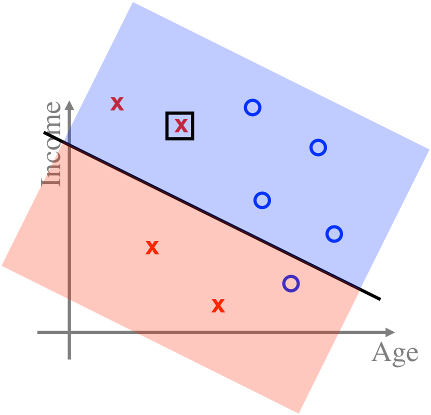
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



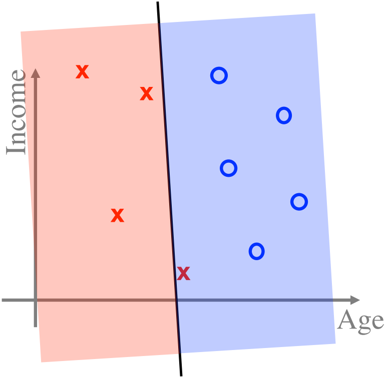
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



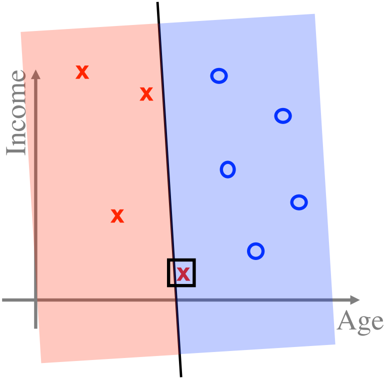
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



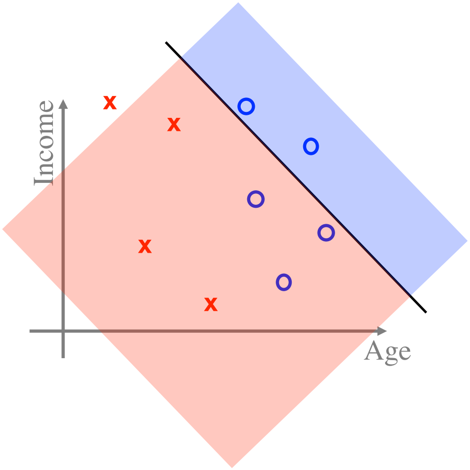
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



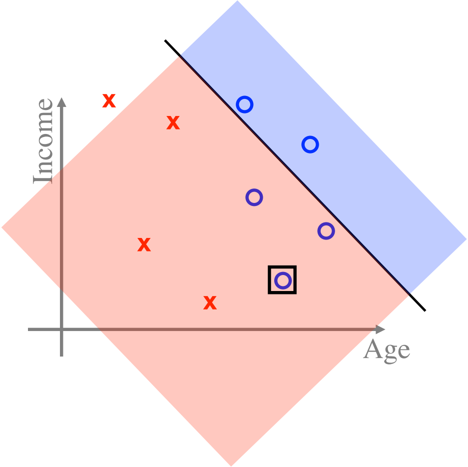
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



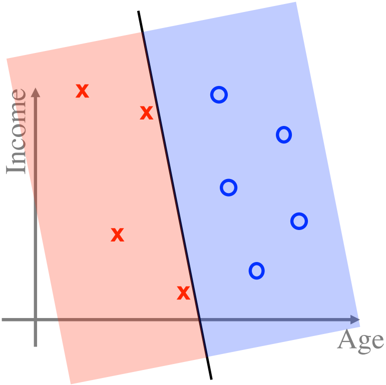
Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



Does PLA Work

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.



Why does PLA Work

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- We can show that the weight update rule above has the nice interpretation that it moves in the direction of classifying x_* correctly.
  - Show that $y_*\mathbf{w}(t)^T\mathbf{x}_* < 0$;

Why does PLA Work

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- We can show that the weight update rule above has the nice interpretation that it moves in the direction of classifying x_* correctly.
  - Show that $y_*\mathbf{w}(t)^T\mathbf{x}_* < 0$;
  - Show that $y_*\mathbf{w}(t+1)^T\mathbf{x}_* > y_*\mathbf{w}(t)^T\mathbf{x}_*$;

Why does PLA Work

- Identify misclassified example $(\mathbf{x}_*, y_*)$ and update $\mathbf{w}(t+1) = \mathbf{w}(t) + y_*\mathbf{x}_*$
- We can show that the weight update rule above has the nice interpretation that it moves in the direction of classifying x_* correctly.
  - Show that $y_*\mathbf{w}(t)^T\mathbf{x}_* < 0$;
  - Show that $y_*\mathbf{w}(t+1)^T\mathbf{x}_* > y_*\mathbf{w}(t)^T\mathbf{x}_*$;
  - Move from $\mathbf{w}(t)$ to $\mathbf{w}(t+1)$ is a move 'in the right direction' in terms of classifying $\mathbf{x}_*$

Issues with PLA

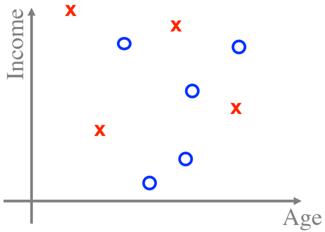
Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.

- Converge after how long?

Issues with PLA

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.

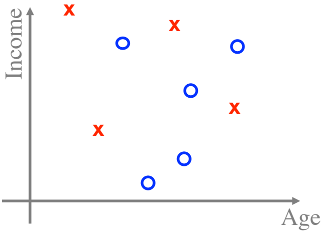
- Converge after how long?
- What if the data cannot be fit by a linear separator?



Issues with PLA

Theorem. If the data can be fit by a linear separator, then after some finite number of steps, PLA will find one.

- Converge after how long?
- What if the data cannot be fit by a linear separator?



Is it common? The XOR problem.

Converge after how long?

- Assume that there exists a hyperplane $\mathbf{w}^*$ that separates the data, i.e., exists δ, such that for $y = +1$,$(\mathbf{w}^*)^T \mathbf{x} > \delta$ and for $y = -1$, $(\mathbf{w}^*)^T \mathbf{x} < -\delta$. Or equivalently,

$$y(\mathbf{w}^*)^T \mathbf{x} > \delta, \forall (\mathbf{x}, y) \in \mathcal{D} \tag{1}$$

Converge after how long?

- Assume that there exists a hyperplane $\mathbf{w}^*$ that separates the data, i.e., exists δ, such that for $y = +1$,$(\mathbf{w}^*)^T\mathbf{x} > \delta$ and for $y = -1$, $(\mathbf{w}^*)^T\mathbf{x} < -\delta$. Or equivalently,

$$y(\mathbf{w}^*)^T\mathbf{x} > \delta, \forall(\mathbf{x}, y) \in \mathcal{D} \tag{1}$$

- We only update when we found a misclassified example, i.e., for any updated iteration i, we have $y(i)\mathbf{w}(i)^T\mathbf{x}(i) < 0$

Converge after how long?

- Assume that there exists a hyperplane $\mathbf{w}^*$ that separates the data, i.e., exists δ, such that for $y = +1$,$(\mathbf{w}^*)^T\mathbf{x} > \delta$ and for $y = -1$, $(\mathbf{w}^*)^T\mathbf{x} < -\delta$. Or equivalently,

$$y(\mathbf{w}^*)^T\mathbf{x} > \delta, \forall(\mathbf{x}, y) \in \mathcal{D} \tag{1}$$

- We only update when we found a misclassified example, i.e., for any updated iteration i, we have $y(i)\mathbf{w}(i)^T\mathbf{x}(i) < 0$
- Assume our algorithm ends after t iterations, all we need to show is that t is **upper bounded**.

Convergence of Perceptron

- When start from a zero vector $\mathbf{w}(0) = \mathbf{0}$, then we have

$$\mathbf{w}(t) = y(0)\mathbf{x}(0) + y(1)\mathbf{x}(1) + \cdots + y(t-1)\mathbf{x}(t-1)$$

Convergence of Perceptron

- When start from a zero vector $\mathbf{w}(0) = \mathbf{0}$, then we have

$$\mathbf{w}(t) = y(0)\mathbf{x}(0) + y(1)\mathbf{x}(1) + \cdots + y(t-1)\mathbf{x}(t-1)$$

- It follows that:

$$\begin{aligned}(\mathbf{w}^*)^T\mathbf{w}(t) &= y(0)(\mathbf{w}^*)^T\mathbf{x}(0) + \cdots + y(t-1)(\mathbf{w}^*)^T\mathbf{x}(t-1) \\ &\geq t\delta\end{aligned}$$

- According to *Cauchy-Schwartz* inequality:

$$\left((\mathbf{w}^*)^T\mathbf{w}(t)\right)^2 \leq \|\mathbf{w}^*\|_2^2\|\mathbf{w}(t)\|_2^2$$

$$\Rightarrow \|\mathbf{w}(t)\|_2^2 \geq \frac{\left((\mathbf{w}^*)^T\mathbf{w}(t)\right)^2}{\|\mathbf{w}^*\|_2^2} \geq \frac{t^2\delta^2}{\|\mathbf{w}^*\|_2^2}$$

Convergence of Perceptron

- Also we can show that

$$
\begin{aligned}
&\|\mathbf{w}(t)\|^2 = \mathbf{w}(t)^T\mathbf{w}(t) \\
&= (\mathbf{w}(t-1) + y(t-1)\mathbf{x}(t-1))^T(\mathbf{w}(t-1) + y(t-1)\mathbf{x}(t-1)) \\
&= \|\mathbf{w}(t-1)\|_2^2 + \|\mathbf{x}(t-1)\|_2^2 + 2y(t-1)\mathbf{w}(t-1)^T\mathbf{x}(t-1) \\
&\leq \|\mathbf{w}(t-1)\|_2^2 + \|\mathbf{x}(t-1)\|_2^2
\end{aligned}
$$

- It follows that

$$\|\mathbf{w}(t)\|_2^2 \leq \|\mathbf{x}(t-1)\|_2^2 + \cdots + \|\mathbf{x}(0)\|_2^2 \leq t \max \|\mathbf{x}(.)\|_2^2$$

Convergence of Perceptron

- Combine the two results.

$$\frac{t^2\delta^2}{\|\mathbf{w}^*\|_2^2} \leq \|\mathbf{w}(t)\|_2^2 \leq t \max \|\mathbf{x}(.)\|_2^2$$

Convergence of Perceptron

- Combine the two results.

$$\frac{t^2\delta^2}{\|\mathbf{w}^*\|_2^2} \leq \|\mathbf{w}(t)\|_2^2 \leq t \max \|\mathbf{x}(.)\|_2^2$$

- Equivalently:

$$t \leq \frac{\|\mathbf{w}^*\|_2^2 \cdot \max \|\mathbf{x}(.)\|_2^2}{\delta^2}$$

Convergence of Perceptron

- Combine the two results.

$$\frac{t^2\delta^2}{\|\mathbf{w}^*\|_2^2} \leq \|\mathbf{w}(t)\|_2^2 \leq t \max \|\mathbf{x}(.)\|_2^2$$

- Equivalently:

$$t \leq \frac{\|\mathbf{w}^*\|_2^2 \cdot \max \|\mathbf{x}(.)\|_2^2}{\delta^2}$$

- What does this bound tell us about the Perceptron?
  - δ measures how close the solution decision boundary is to the input patterns.

Convergence of Perceptron

- Combine the two results.

$$\frac{t^2\delta^2}{\|\mathbf{w}^*\|_2^2} \leq \|\mathbf{w}(t)\|_2^2 \leq t \max \|\mathbf{x}(.)\|_2^2$$

- Equivalently:

$$t \leq \frac{\|\mathbf{w}^*\|_2^2 \cdot \max \|\mathbf{x}(.)\|_2^2}{\delta^2}$$

- What does this bound tell us about the Perceptron?
  - δ measures how close the solution decision boundary is to the input patterns.
  - If the input classes are difficult to separate (are close to the decision boundary) it will take many iterations for the algorithm to converge.

Convergence of Perceptron

- Combine the two results.

$$\frac{t^2\delta^2}{\|\mathbf{w}^*\|_2^2} \leq \|\mathbf{w}(t)\|_2^2 \leq t \max \|\mathbf{x}(.)\|_2^2$$

- Equivalently:

$$t \leq \frac{\|\mathbf{w}^*\|_2^2 \cdot \max \|\mathbf{x}(.)\|_2^2}{\delta^2}$$

- What does this bound tell us about the Perceptron?
  - δ measures how close the solution decision boundary is to the input patterns.
  - If the input classes are difficult to separate (are close to the decision boundary) it will take many iterations for the algorithm to converge.
  - Additional assumption required: $\max \|\mathbf{x}(.)\|_2^2$ is bounded.

We can Fit the Data

- We are able to find a *h* that works from infinitely many (for the perceptron). So computationally, things seem good.

We can Fit the Data

- We are able to find a h that works from infinitely many (for the perceptron). So computationally, things seem good.
- Ultimately, our goal is to *predict*: we don't care about the training data, we care about "outside the training data".

We can Fit the Data

- We are able to find a h that works from infinitely many (for the perceptron). So computationally, things seem good.
- Ultimately, our goal is to *predict*: we don't care about the training data, we care about "outside the training data".

Can a limited data set reveal enough information to pin down an entire target function, so that we can predict outside the data?