Lecture 2: Learning to Answer Yes/No

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering

National Taiwan University
(國立台灣大學資訊工程系)
Roadmap

1. **When Can Machines Learn?**

   **Lecture 1: The Learning Problem**
   \[ \mathcal{A} \text{ takes } \mathcal{D} \text{ and } \mathcal{H} \text{ to get } g \]

   **Lecture 2: Learning to Answer Yes/No**
   - Perceptron Hypothesis Set
   - Perceptron Learning Algorithm (PLA)
   - Guarantee of PLA
   - Non-Separable Data

2. Why Can Machines Learn?
3. How Can Machines Learn?
4. How Can Machines Learn Better?
Credit Approval Problem Revisited

Applicant Information

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>23 years</td>
</tr>
<tr>
<td>gender</td>
<td>female</td>
</tr>
<tr>
<td>annual salary</td>
<td>NTD 1,000,000</td>
</tr>
<tr>
<td>year in residence</td>
<td>1 year</td>
</tr>
<tr>
<td>year in job</td>
<td>0.5 year</td>
</tr>
<tr>
<td>current debt</td>
<td>200,000</td>
</tr>
</tbody>
</table>

unknown target function $f : \mathcal{X} \rightarrow \mathcal{Y}$

(ideal credit approval formula)

training examples $\mathcal{D}: (x_1, y_1), \ldots, (x_N, y_N)$

(historical records in bank)

learning algorithm $\mathcal{A}$

final hypothesis $g \approx f$

('learned' formula to be used)

hypothesis set $\mathcal{H}$

(set of candidate formula)
Credit Approval Problem Revisited

Unknown target function $f: \mathcal{X} \rightarrow \mathcal{Y}$

Applicant Information

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>23 years</td>
</tr>
<tr>
<td>Gender</td>
<td>Female</td>
</tr>
<tr>
<td>Annual salary</td>
<td>NTD 1,000,000</td>
</tr>
<tr>
<td>Year in residence</td>
<td>1 year</td>
</tr>
<tr>
<td>Year in job</td>
<td>0.5 year</td>
</tr>
<tr>
<td>Current debt</td>
<td>200,000</td>
</tr>
</tbody>
</table>

Training examples $D: (x_1, y_1), \ldots, (x_N, y_N)$

Learning algorithm $\mathcal{A}$

Final hypothesis $g \approx f$

What hypothesis set can we use?
A Simple Hypothesis Set: the ‘Perceptron’

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>23 years</td>
</tr>
<tr>
<td>annual salary</td>
<td>NTD 1,000,000</td>
</tr>
<tr>
<td>year in job</td>
<td>0.5 year</td>
</tr>
<tr>
<td>current debt</td>
<td>200,000</td>
</tr>
</tbody>
</table>

- For \( \mathbf{x} = (x_1, x_2, \cdots, x_d) \) ‘features of customer’, compute a weighted ‘score’ and

  - approve credit if \( \sum_{i=1}^{d} w_i x_i > \text{threshold} \)
  - deny credit if \( \sum_{i=1}^{d} w_i x_i < \text{threshold} \)
A Simple Hypothesis Set: the ‘Perceptron’

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>age</td>
<td>23 years</td>
</tr>
<tr>
<td>annual salary</td>
<td>NTD 1,000,000</td>
</tr>
<tr>
<td>year in job</td>
<td>0.5 year</td>
</tr>
<tr>
<td>current debt</td>
<td>200,000</td>
</tr>
</tbody>
</table>

- For \( \mathbf{x} = (x_1, x_2, \cdots, x_d) \) ‘features of customer’, compute a weighted ‘score’ and
  
  \[
  \text{approve credit if } \sum_{i=1}^{d} w_i x_i > \text{threshold} \\
  \text{deny credit if } \sum_{i=1}^{d} w_i x_i < \text{threshold}
  \]

- \( \mathcal{Y} : \{+1(\text{good}), -1(\text{bad})\} \), 0 ignored—linear formula \( h \in \mathcal{H} \) are
  
  \[
  h(\mathbf{x}) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) - \text{threshold} \right)
  \]

called ‘perceptron’ hypothesis historically
Vector Form of Perceptron Hypothesis

\[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) - \text{threshold} \right) \]

\[ = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + \right) \]
Vector Form of Perceptron Hypothesis

\[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) - \text{threshold} \right) \]

\[ = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + \left( {-\text{threshold}} \right) \cdot \left( {+1} \right) \right) \]

Each 'tall' \( w \) represents a hypothesis \( h \) and is multiplied with 'tall' \( x \)—will use tall versions to simplify notation.
Vector Form of Perceptron Hypothesis

\[ h(x) = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) - \text{threshold} \right) \]

\[ = \text{sign} \left( \left( \sum_{i=1}^{d} w_i x_i \right) + (-\text{threshold}) \cdot (+1) \right) \]

\[ = \text{sign} \left( w_i x_i \right) \]
Vector Form of Perceptron Hypothesis

\[ h(x) = \text{sign} \left( \sum_{i=1}^{d} w_i x_i - \text{threshold} \right) \]

\[ = \text{sign} \left( \sum_{i=1}^{d} w_i x_i + (-\text{threshold}) \cdot (+1) \right) \]

\[ = \text{sign} \left( \sum_{i=0}^{d} w_i x_i \right) \]
Vector Form of Perceptron Hypothesis

\[
h(x) = \text{sign} \left( \sum_{i=1}^{d} w_i x_i - \text{threshold} \right)
\]

\[
= \text{sign} \left( \sum_{i=1}^{d} w_i x_i \right) + (-\text{threshold}) \cdot (+1)
\]

\[
= \text{sign} \left( \sum_{i=0}^{d} w_i x_i \right)
\]

\[
= \text{sign} \left( w_0 x_0 \right)
\]
Vector Form of Perceptron Hypothesis

Vector Form of Perceptron Hypothesis

\[ h(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^{d} w_i x_i - \text{threshold} \right) \]

\[ = \text{sign} \left( \sum_{i=1}^{d} w_i x_i + \left( -\text{threshold} \right) \cdot (1) \right) \]

\[ = \text{sign} \left( \sum_{i=0}^{d} w_i x_i \right) \]

\[ = \text{sign} \left( \mathbf{w}^T \mathbf{x} \right) \]

- each ‘tall’ \( \mathbf{w} \) represents a hypothesis \( h \) & is multiplied with ‘tall’ \( \mathbf{x} \) —will use tall versions to simplify notation
Vector Form of Perceptron Hypothesis

\[ h(x) = \text{sign} \left( \sum_{i=1}^{d} w_i x_i \right) - \text{threshold} \]

\[ = \text{sign} \left( \sum_{i=1}^{d} w_i x_i \right) + (-\text{threshold}) \cdot (1) \]

\[ = \text{sign} \left( \sum_{i=0}^{d} w_i x_i \right) \]

\[ = \text{sign} \left( w^T x \right) \]

- each ‘tall’ \( w \) represents a hypothesis \( h \) & is multiplied with ‘tall’ \( x \) — **will use tall versions to simplify notation**

what do perceptrons \( h \) ‘look like’?
Perceptrons in $\mathbb{R}^2$

$$h(x) = \text{sign} \left( w_0 + w_1 x_1 + w_2 x_2 \right)$$

- customer features $x$: points on the plane (or points in $\mathbb{R}^d$)
- labels $y$: $\circ (+1), \times (-1)$
- hypothesis $h$: lines (or hyperplanes in $\mathbb{R}^d$)
  —positive on one side of a line, negative on the other side
Perceptrons in $\mathbb{R}^2$

\[ h(x) = \text{sign} \left( w_0 + w_1 x_1 + w_2 x_2 \right) \]

- **customer features** $x$: points on the plane (or points in $\mathbb{R}^d$)
- **labels** $y$: ○ (+1), × (-1)
- **hypothesis** $h$: **lines** (or hyperplanes in $\mathbb{R}^d$)
  —positive on one side of a line, negative on the other side
- **different line** classifies customers differently

perceptrons $\iff$ **linear (binary) classifiers**
Consider using a perceptron to detect spam messages.

Assume that each email is represented by the frequency of keyword occurrence, and output $+1$ indicates a spam. Which keywords below shall have large positive weights in a good perceptron for the task?

1. coffee, tea, hamburger, steak
2. free, drug, fantastic, deal
3. machine, learning, statistics, textbook
4. national, Taiwan, university, coursera
Fun Time

Consider using a perceptron to detect spam messages.

Assume that each email is represented by the frequency of keyword occurrence, and output $+1$ indicates a spam. Which keywords below shall have large positive weights in a good perceptron for the task?

1. coffee, tea, hamburger, steak
2. free, drug, fantastic, deal
3. machine, learning, statistics, textbook
4. national, Taiwan, university, coursera

Reference Answer: 2

The occurrence of keywords with positive weights increase the ‘spam score’, and hence those keywords should often appear in spams.
Select $g$ from $\mathcal{H}$

$\mathcal{H} = \text{all possible perceptrons, } g =$?

- want: $g \approx f$ (hard when $f$ unknown)
Select $g$ from $\mathcal{H}$

$\mathcal{H} = \text{all possible perceptrons, } g =$

- want: $g \approx f$ (hard when $f$ unknown)
- almost necessary: $g \approx f$ on $\mathcal{D}$, ideally $g(x_n) = f(x_n) = y_n$
Select \( g \) from \( \mathcal{H} \)

\[ \mathcal{H} = \text{all possible perceptrons}, \quad g = ? \]

- want: \( g \approx f \) (hard when \( f \) unknown)
- almost necessary: \( g \approx f \) on \( D \), ideally
  \[ g(x_n) = f(x_n) = y_n \]
- difficult: \( \mathcal{H} \) is of **infinite** size
Select $g$ from $\mathcal{H}$

$\mathcal{H} = \text{all possible perceptrons, } g = ?$

- want: $g \approx f$ (hard when $f$ unknown)
- almost necessary: $g \approx f$ on $\mathcal{D}$, ideally
  $g(x_n) = f(x_n) = y_n$
- difficult: $\mathcal{H}$ is of infinite size
- idea: start from some $g_0$, and 'correct' its mistakes on $\mathcal{D}$
Select $g$ from $\mathcal{H}$

$\mathcal{H} = \text{all possible perceptrons}, \; g = ?$

- want: $g \approx f$ (hard when $f$ unknown)
- almost necessary: $g \approx f$ on $\mathcal{D}$, ideally $g(x_n) = f(x_n) = y_n$
- difficult: $\mathcal{H}$ is of infinite size
- idea: start from some $g_0$, and ‘correct’ its mistakes on $\mathcal{D}$

will represent $g_0$ by its weight vector $w_0$
Perceptron Learning Algorithm

start from some $w_0$ (say, 0), and ‘correct’ its mistakes on $D$

For $t = 0, 1, \ldots$

1. find a mistake of $w_t$ called $(x_{n(t)}, y_{n(t)})$

$$\text{sign} \left( w_t^T x_{n(t)} \right) \neq y_{n(t)}$$

That's it! — A fault confessed is half redressed. :-)

Hsuan-Tien Lin (NTU CSIE) Machine Learning Foundations
**Perceptron Learning Algorithm**

**start from some** $w_0$ (say, $0$), and ‘correct’ its mistakes on $\mathcal{D}$

For $t = 0, 1, \ldots$

1. find a **mistake** of $w_t$ called $(x_{n(t)}, y_{n(t)})$

   $$\text{sign} \left( w_t^T x_{n(t)} \right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

   $$w_{t+1} \leftarrow w_t + y_{n(t)}x_{n(t)}$$
For $t = 0, 1, \ldots$

1. find a **mistake** of $w_t$ called $(x_n(t), y_n(t))$

   \[ \text{sign} \left( w_t^T x_n(t) \right) \neq y_n(t) \]

2. (try to) correct the mistake by

   \[ w_{t+1} \leftarrow w_t + y_n(t)x_n(t) \]

   ... until no more mistakes

return last $w$ (called $w_{\text{PLA}}$) as $g$
Perceptron Learning Algorithm

start from some $w_0$ (say, 0), and ‘correct’ its mistakes on $D$

For $t = 0, 1, \ldots$

1. find a mistake of $w_t$ called $(x_n(t), y_n(t))$

$$\text{sign} \left( w_t^T x_n(t) \right) \neq y_n(t)$$

2. (try to) correct the mistake by

$$w_{t+1} \leftarrow w_t + y_n(t)x_n(t)$$

… until no more mistakes

return last $w$ (called $w_{\text{PLA}}$) as $g$

That’s it!

—A fault confessed is half redressed. :-)

Hsuan-Tien Lin (NTU CSIE)
Practical Implementation of PLA

start from some \( w_0 \) (say, \( 0 \)), and ‘correct’ its mistakes on \( D \)

**Cyclic PLA**

For \( t = 0, 1, \ldots \)

1. find the next mistake of \( w_t \) called \((x_{n(t)}, y_{n(t)})\)

\[
\text{sign}\left(w_t^T x_{n(t)}\right) \neq y_{n(t)}
\]

2. correct the mistake by

\[
w_{t+1} \leftarrow w_t + y_{n(t)} x_{n(t)}
\]

\( \ldots \) until a full cycle of not encountering mistakes

**next** can follow naïve cycle \((1, \cdots, N)\) or precomputed random cycle
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[ x_1 \]

\[ w(t+1) \]

update: 1

\[ x_9 \]

\[ w(t) \]

\[ w(t+1) \]

update: 2

\[ x_{14} \]

\[ w(t) \]

\[ w(t+1) \]

update: 3

\[ x_3 \]

\[ w(t) \]

\[ w(t+1) \]

update: 4

\[ x_9 \]

\[ w(t) \]

\[ w(t+1) \]

update: 5

\[ x_{14} \]

\[ w(t) \]

\[ w(t+1) \]

update: 6

\[ x_9 \]

\[ w(t) \]

\[ w(t+1) \]

update: 7

\[ x_{14} \]

\[ w(t) \]

\[ w(t+1) \]

update: 8

\[ x_9 \]

\[ w(t) \]

\[ w(t+1) \]

update: 9

PLA

finally

worked like a charm with < 20 lines!! (note: made \( x_i \gg x_0 = 1 \) for visual purpose)

Hsuan-Tien Lin (NTU CSIE)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[
\begin{align*}
\text{update: 1} & \\
\text{update: 2} & \\
\text{update: 3} & \\
\text{update: 4} & \\
\text{update: 5} & \\
\text{update: 6} & \\
\text{update: 7} & \\
\text{update: 8} & \\
\text{update: 9} & \\
\end{align*}
\]

Finally, the PLA worked like a charm with < 20 lines!!

(Hsuan-Tien Lin (NTU CSIE) Machine Learning Foundations 10/22)
Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[ w(t+1) = w(t) + x_9 \]

update: 2

Finally, PLA worked like a charm with < 20 lines!! (note: made \( x_i \gg x_0 = 1 \) for visual purpose)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[ x_1 \]

\[ w(t+1) \]

update: 1

\[ x \]

\[ w(t) \]

update: 2

\[ x \]

\[ w(t) \]

update: 3

\[ x \]

\[ w(t) \]

update: 4

\[ x \]

\[ w(t) \]

update: 5

\[ x \]

\[ w(t) \]

update: 6

\[ x \]

\[ w(t) \]

update: 7

\[ x \]

\[ w(t) \]

update: 8

\[ x \]

\[ w(t) \]

update: 9

PLA

finally

worked like a charm with < 20 lines!!

(note: made \[ x_i \gg x_0 = 1 \] for visual purpose)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

update: 4

\[ w(t+1) = w(t) + x \]

Finally, worked like a charm with <20 lines!! (note: made \( x_i \gg x_0 = 1 \) for visual purpose)
**Perceptron Learning Algorithm (PLA)**

**Seeing is Believing**

Learning to Answer Yes/No

**Initial Setup**
- \( w(t) \) is the weight vector at time \( t \).
- \( x \) is the input vector.
- The goal is to find a separating hyperplane.

**Update Rule**
- If the prediction is incorrect, update the weight vector as follows:
  
  
  \[
  w(t+1) = w(t) + x
  \]

**Progress**
- Each update moves the decision boundary closer to the correct classification.
- The algorithm terminates when all examples are correctly classified.

**Example**
- Consider a dataset with \( \mathbf{w}_{t+1} \) as the weight vector after \( t+1 \) updates.
- The final weight vector \( \mathbf{w}_{\text{PLA}} \) works like a charm with less than 20 lines.

\[
\text{(note: made } x_i \gg x_0 = 1 \text{ for visual purpose)}
\]

Hsuan-Tien Lin (NTU CSIE)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[ x_{14} \]

\[ w(t) \]

\[ w(t+1) \]

update: 6

\[ w \]

finally

worked like a charm with < 20 lines!! (note: made \( x_i \gg x_0 = 1 \) for visual purpose)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

update: 7

w(t)

w(t+1)

x_9

Finally, PLA worked like a charm with <20 lines!! (note: made x_i ≫ x_0 = 1 for visual purpose)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[
x_1 \quad w(t+1) \quad \text{update: 1}
\]

\[
x_9 \quad w(t) \quad \text{update: 2}
\]

\[
x_{14} \quad w(t) \quad \text{update: 3}
\]

\[
x_9 \quad w(t) \quad \text{update: 4}
\]

\[
x_{14} \quad w(t) \quad \text{update: 5}
\]

\[
x_9 \quad w(t) \quad \text{update: 6}
\]

\[
x_9 \quad w(t) \quad \text{update: 7}
\]

\[
x_{14} \quad w(t) \quad \text{update: 8}
\]

\[
x_9 \quad w(t) \quad \text{update: 9}
\]

\[
\text{finally}
\]

worked like a charm with \(< 20\) lines!!

(note: made \(x_i \gg x_0 = 1\) for visual purpose)

Hsuan-Tien Lin (NTU CSIE)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

worked like a charm with $< 20$ lines!!

(note: made $x_i \gg x_0 = 1$ for visual purpose)
Seeing is Believing

worked like a charm with $< 20$ lines!!
(note: made $x_i \gg x_0 = 1$ for visual purpose)
Some Remaining Issues of PLA

‘correct’ mistakes on $D$ until no mistakes

Algorithmic: halt (with no mistake)?

- naïve cyclic: ??
- random cyclic: ??
- other variant: ??
Some Remaining Issues of PLA

‘correct’ mistakes on $\mathcal{D}$ until no mistakes

Algorithmic: halt (with no mistake)?
- naïve cyclic: ??
- random cyclic: ??
- other variant: ??

Learning: $g \approx f$?
- on $\mathcal{D}$, if halt, yes (no mistake)
- outside $\mathcal{D}$: ??
- if not halting: ??
# Some Remaining Issues of PLA

`correct` mistakes on $\mathcal{D}$ **until no mistakes**

### Algorithmic: halt (with no mistake)?
- naïve cyclic: ??
- random cyclic: ??
- other variant: ??

### Learning: $g \approx f$?
- on $\mathcal{D}$, if halt, yes (no mistake)
- outside $\mathcal{D}$: ??
- if not halting: ??

[to be shown] if (...), after ‘enough’ corrections, any PLA variant halts
Let’s try to think about why PLA may work.

Let \( n = n(t) \), according to the rule of PLA below, which formula is true?

\[
\text{sign} \left( w_t^T x_n \right) \neq y_n, \quad w_{t+1} \leftarrow w_t + y_n x_n
\]

1. \( w_{t+1}^T x_n = y_n \)
2. \( \text{sign}(w_{t+1}^T x_n) = y_n \)
3. \( y_n w_{t+1}^T x_n \geq y_n w_t^T x_n \)
4. \( y_n w_{t+1}^T x_n < y_n w_t^T x_n \)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Fun Time

Let’s try to think about why PLA may work.

Let \( n = n(t) \), according to the rule of PLA below, which formula is true?

\[
\text{sign} \left( w_t^T x_n \right) \neq y_n, \quad w_{t+1} \leftarrow w_t + y_n x_n
\]

1. \( w_{t+1}^T x_n = y_n \)
2. \( \text{sign}(w_{t+1}^T x_n) = y_n \)
3. \( y_n w_{t+1}^T x_n \geq y_n w_t^T x_n \)
4. \( y_n w_{t+1}^T x_n < y_n w_t^T x_n \)

Reference Answer: 3

Simply multiply the second part of the rule by \( y_n x_n \). The result shows that the rule somewhat ‘tries to correct the mistake.’
Linear Separability

- **if** PLA halts (i.e. no more mistakes),
  (necessary condition) $\mathcal{D}$ allows some $w$ to make no mistake
- call such $\mathcal{D}$ **linear separable**
Linear Separability

- if PLA halts (i.e. no more mistakes), (necessary condition) $\mathcal{D}$ allows some $w$ to make no mistake
- call such $\mathcal{D}$ linear separable
Learning to Answer Yes/No  Guarantee of PLA

Linear Separability

- **if** PLA halts (i.e. no more mistakes),
  *(necessary condition)* $D$ allows some $\mathbf{w}$ to make no mistake
- call such $D$ **linear separable**

![Linear separable](image1)
![Not linear separable](image2)
![Not linear separable](image3)
Learning to Answer Yes/No

Guarantee of PLA

Linear Separability

- if PLA halts (i.e. no more mistakes),
  \( \text{(necessary condition)} \) \( \mathcal{D} \) allows some \( w \) to make no mistake
- call such \( \mathcal{D} \) linear separable

assume linear separable \( \mathcal{D} \),
does PLA always \textbf{halt}?
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$

- $w_f$ perfect hence every $x_n$ correctly away from line:

$$\min_n y_n w_f^T x_n > 0$$
PLA Fact: \( w_t \) Gets More Aligned with \( w_f \)

linear separable \( \mathcal{D} \iff \text{exists perfect } w_f \text{ such that } y_n = \text{sign}(w_f^T x_n) \)

- \( w_f \) perfect hence every \( x_n \) correctly away from line:

\[
y_n(t) w_f^T x_n(t) \geq \min_n y_n w_f^T x_n > 0
\]
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$

- $w_f$ perfect hence every $x_n$ correctly away from line:

$$y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0$$

by updating with any $(x_{n(t)}, y_{n(t)})$

$$w_f^T w_{t+1} = w_f^T (w_t + y_{n(t)} x_{n(t)})$$
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$

- $w_f$ perfect hence every $x_n$ correctly away from line:

$$y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0$$

by updating with any $(x_{n(t)}, y_{n(t)})$

$$w_f^T w_{t+1} = w_f^T (w_t + y_{n(t)} x_{n(t)})$$

$$w_f^T w_t$$
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$

- $w_f$ perfect hence every $x_n$ correctly away from line:

\[ y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0 \]

by updating with any $\left(x_{n(t)}, y_{n(t)}\right)$

\[ w_f^T w_{t+1} = w_f^T \left(w_t + y_{n(t)} x_{n(t)}\right) \]
\[ \geq w_f^T w_t + \min_n y_n w_f^T x_n \]
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$

- $w_f$ perfect hence every $x_n$ correctly away from line:

$$y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0$$

by updating with any $(x_{n(t)}, y_{n(t)})$

$$w_f^T w_{t+1} = w_f^T (w_t + y_{n(t)} x_{n(t)})$$

$$\geq w_f^T w_t + \min_n y_n w_f^T x_n$$

$$w_f^T w_t$$
PLA Fact: $w_t$ Gets More Aligned with $w_f$

linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$

- $w_f$ perfect hence every $x_n$ correctly away from line:

$$y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0$$

by updating with any $(x_{n(t)}, y_{n(t)})$

$$w_f^T w_{t+1} = w_f^T (w_t + y_{n(t)} x_{n(t)})$$

$$\geq w_f^T w_t + \min_n y_n w_f^T x_n$$

$$> w_f^T w_t + 0.$$
PLA Fact: $w_t$ Gets MoreAligned with $w_f$

**linear separable $\mathcal{D} \iff$ exists perfect $w_f$ such that $y_n = \text{sign}(w_f^T x_n)$**

- $w_f$ perfect hence every $x_n$ correctly away from line:
  
  $$y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0$$

- $w_f^T w_t$ ↑ by updating with any $(x_{n(t)}, y_{n(t)})$
  
  $$w_f^T w_{t+1} = w_f^T (w_t + y_{n(t)} x_{n(t)})$$
  
  $$\geq w_f^T w_t + \min_n y_n w_f^T x_n$$
  
  $$> w_f^T w_t + 0.$$ 

$w_t$ appears more aligned with $w_f$ after update (really?)
PLA Fact: \( w_t \) Does Not Grow Too Fast

\[ w_t \text{ changed only when mistake} \]

\[ \Leftrightarrow \text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)} \Leftrightarrow y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} \leq 0 \]
PLA Fact: $w_t$ Does Not Grow Too Fast

- $w_t$ changed only when mistake
  \[ \iff \text{sign} (w_t^T x_{n(t)}) \neq y_{n(t)} \iff y_{n(t)} w_t^T x_{n(t)} \leq 0 \]

\[
\begin{align*}
\|w_{t+1}\|^2 &= \|w_t + y_{n(t)} x_{n(t)}\|^2 \\
&= \|w_t\|^2 + \|y_{n(t)} x_{n(t)}\|^2 \\
\end{align*}
\]
PLA Fact: $w_t$ Does Not Grow Too Fast

$w_t$ changed only when mistake

$\Leftrightarrow \text{sign} \left( w_t^T x_{n(t)} \right) \neq y_{n(t)} \Leftrightarrow y_{n(t)} w_t^T x_{n(t)} \leq 0$

\[
\| w_{t+1} \|^2 = \| w_t + y_{n(t)} x_{n(t)} \|^2 \\
= \| w_t \|^2 + 2y_{n(t)} w_t^T x_{n(t)} + \| y_{n(t)} x_{n(t)} \|^2
\]
PLA Fact: $w_t$ Does Not Grow Too Fast

\[ w_t \text{ changed only when mistake } \iff \text{sign} \left( w_t^T x_{n(t)} \right) \neq y_{n(t)} \iff y_{n(t)} w_t^T x_{n(t)} \leq 0 \]

- mistake ‘limits’ $\|w_t\|^2$ growth,

\[
\|w_{t+1}\|^2 = \|w_t + y_{n(t)}x_{n(t)}\|^2 = \|w_t\|^2 + 2y_{n(t)}w_t^T x_{n(t)} + \|y_{n(t)}x_{n(t)}\|^2 \\
\leq \|w_t\|^2 + 0 + \|y_{n(t)}x_{n(t)}\|^2
\]
PLA Fact: $w_t$ Does Not Grow Too Fast

\[ w_t \text{ changed only when mistake} \]

\[ \iff \text{sign} \left( w_t^T x_{n(t)} \right) \neq y_{n(t)} \iff y_{n(t)} w_t^T x_{n(t)} \leq 0 \]

- mistake 'limits' $\|w_t\|^2$ growth,

\[
\begin{align*}
\|w_{t+1}\|^2 &= \|w_t + y_{n(t)}x_{n(t)}\|^2 \\
&= \|w_t\|^2 + 2y_{n(t)}w_t^T x_{n(t)} + \|y_{n(t)}x_{n(t)}\|^2 \\
&\leq \|w_t\|^2 + 0 + \|y_{n(t)}x_{n(t)}\|^2 \\
&\|w_t\|^2
\end{align*}
\]
PLA Fact: $w_t$ Does Not Grow Too Fast

- $w_t$ changed only when mistake
  \[ \iff \text{sign} (w_t^T x_{n(t)}) \neq y_{n(t)} \iff y_{n(t)} w_t^T x_{n(t)} \leq 0 \]

- mistake ‘limits’ $\|w_t\|^2$ growth, even when updating with ‘longest’ $x_n$

\[
\begin{align*}
\|w_{t+1}\|^2 &= \|w_t + y_{n(t)} x_{n(t)}\|^2 \\
&= \|w_t\|^2 + 2y_{n(t)} w_t^T x_{n(t)} + \|y_{n(t)} x_{n(t)}\|^2 \\
&\leq \|w_t\|^2 + 0 + \|y_{n(t)} x_{n(t)}\|^2 \\
&\leq \|w_t\|^2 + \max_n \|y_n x_n\|^2
\end{align*}
\]
PLA Fact: \( w_t \) Does Not Grow Too Fast

\( w_t \) changed only when mistake

\[ \iff \text{sign} (w_t^T x_{n(t)}) \neq y_{n(t)} \iff y_{n(t)} w_t^T x_{n(t)} \leq 0 \]

- mistake ‘limits’ \( \|w_t\|^2 \) growth, even when updating with ‘longest’ \( x_n \)

\[
\begin{align*}
\|w_{t+1}\|^2 &= \|w_t + y_{n(t)} x_{n(t)}\|^2 \\
&= \|w_t\|^2 + 2y_{n(t)} w_t^T x_{n(t)} + \|y_{n(t)} x_{n(t)}\|^2 \\
&\leq \|w_t\|^2 + 0 + \|y_{n(t)} x_{n(t)}\|^2 \\
&\leq \|w_t\|^2 + \max_n \|y_n x_n\|^2
\end{align*}
\]

start from \( w_0 = 0 \), after \( T \) mistake corrections,

\[
\frac{w_f^T}{\|w_f\|} \cdot \frac{w_T^T}{\|w_T\|} \geq \sqrt{T} \cdot \text{constant}
\]
Fun Time
Let’s upper-bound $T$, the number of mistakes that PLA ‘corrects’.

Define $R^2 = \max_n \|x_n\|^2$, $\rho = \min_n y_n \frac{w_f^T}{\|w_f\|} x_n$

We want to show that $T \leq \square$. Express the upper bound $\square$ by the two terms above.

1. $R/\rho$
2. $R^2/\rho^2$
3. $R/\rho^2$
4. $\rho^2/R^2$
Fun Time

Let’s upper-bound $T$, the number of mistakes that PLA ‘corrects’.

Define $R^2 = \max_n \|x_n\|^2$  $\rho = \min_n y_n \frac{w_f^T}{\|w_f\|}$ $x_n$

We want to show that $T \leq \Box$. Express the upper bound $\Box$ by the two terms above.

1. $R/\rho$
2. $R^2/\rho^2$
3. $R/\rho^2$
4. $\rho^2/R^2$

Reference Answer: 2

The maximum value of $\frac{w_f^T}{\|w_f\|} \frac{w_t}{\|w_t\|}$ is 1. Since $T$ mistake corrections increase the inner product by $\sqrt{T} \cdot \text{constant}$, the maximum number of corrected mistakes is $1/\text{constant}^2$. 
More about PLA

Guarantee

as long as linear separable and correct by mistake

- inner product of $w_f$ and $w_t$ grows fast; length of $w_t$ grows slowly
- PLA ‘lines’ are more and more aligned with $w_f \Rightarrow$ halts
More about PLA

**Guarantee**

as long as linear separable and correct by mistake

- inner product of $w_f$ and $w_t$ grows fast; length of $w_t$ grows slowly
- PLA ‘lines’ are more and more aligned with $w_f$ $\Rightarrow$ halts

**Pros**

simple to implement, fast, works in any dimension $d$
More about PLA

Guarantee

as long as linear separable and correct by mistake

- inner product of $w_f$ and $w_t$ grows fast; length of $w_t$ grows slowly
- PLA ‘lines’ are more and more aligned with $w_f \Rightarrow$ halts

Pros

simple to implement, fast, works in any dimension $d$

Cons

- ‘assumes’ linear separable $D$ to halt
  —property unknown in advance (no need for PLA if we know $w_f$)
More about PLA

**Guarantee**

as long as linear separable and correct by mistake

- inner product of $\mathbf{w}_f$ and $\mathbf{w}_t$ grows fast; length of $\mathbf{w}_t$ grows slowly
- PLA ‘lines’ are more and more aligned with $\mathbf{w}_f \Rightarrow$ halts

**Pros**

simple to implement, fast, works in any dimension $d$

**Cons**

- ‘assumes’ linear separable $\mathcal{D}$ to halt
  —property unknown in advance (no need for PLA if we know $\mathbf{w}_f$)
- not fully sure how long halting takes ($\rho$ depends on $\mathbf{w}_f$)
  —though practically fast
More about PLA

Guarantee
as long as linear separable and correct by mistake
- inner product of $w_f$ and $w_t$ grows fast; length of $w_t$ grows slowly
- PLA ‘lines’ are more and more aligned with $w_f \Rightarrow$ halts

Pros
simple to implement, fast, works in any dimension $d$

Cons
- ‘assumes’ linear separable $D$ to halt
  —property unknown in advance (no need for PLA if we know $w_f$)
- not fully sure how long halting takes ($\rho$ depends on $w_f$)
  —though practically fast

what if $D$ not linear separable?
Learning with **Noisy Data**

unknown target function
\[ f : \mathcal{X} \rightarrow \mathcal{Y} \]
+ noise

(ideal credit approval formula)

---

**training examples**
\[ \mathcal{D} : (x_1, y_1), \ldots, (x_N, y_N) \]
(historical records in bank)

---

**learning algorithm**
\[ \mathcal{A} \]

---

**final hypothesis**
\[ g \approx f \]
(‘learned’ formula to be used)

---

**hypothesis set**
\[ \mathcal{H} \]
(set of candidate formula)

---

**how to at least get**
\[ g \approx f \]
on **noisy** \( \mathcal{D} \)?
Line with Noise Tolerance

- assume ‘little’ noise: $y_n = f(x_n)$ usually
- if so, $g \approx f$ on $\mathcal{D} \iff y_n = g(x_n)$ usually
Learning to Answer Yes/No

Non-Separable Data

Line with Noise Tolerance

- assume ‘little’ noise: $y_n = f(x_n)$ usually
- if so, $g \approx f$ on $\mathcal{D} \iff y_n = g(x_n)$ usually
- how about

$$w_g \leftarrow \arg\min_w \sum_{n=1}^{N} \left[ y_n \neq \text{sign}(w^T x_n) \right]$$
Learning to Answer Yes/No
Non-Separable Data

Line with Noise Tolerance

- assume ‘little’ noise: $y_n = f(x_n)$ usually
- if so, $g \approx f$ on $\mathcal{D} \iff y_n = g(x_n)$ usually
- how about

$$w_g \leftarrow \text{argmin}_w \sum_{n=1}^{N} \left[ y_n \neq \text{sign}(w^T x_n) \right]$$

—NP-hard to solve, unfortunately
Learning to Answer Yes/No
Non-Separable Data

Line with Noise Tolerance

• assume ‘little’ noise: \( y_n = f(x_n) \) usually
• if so, \( g \approx f \) on \( D \) \( \iff \) \( y_n = g(x_n) \) usually
• how about

\[
\mathbf{w}_g \leftarrow \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \left[ y_n \neq \sign(\mathbf{w}^T \mathbf{x}_n) \right]
\]

—NP-hard to solve, unfortunately

---

can we modify PLA to get an ‘approximately good’ \( g \)?
Pocket Algorithm

modify PLA algorithm (black lines) by **keeping best weights in pocket**
Pocket Algorithm

modify PLA algorithm (black lines) by **keeping best weights in pocket**

**initialize pocket weights \( \hat{w} \)**

For \( t = 0, 1, \ldots \)

1. find a **(random)** mistake of \( w_t \) called \((x_{n(t)}, y_{n(t)})\)
2. (try to) correct the mistake by

\[
    w_{t+1} \leftarrow w_t + y_{n(t)} x_{n(t)}
\]

3. **if** \( w_{t+1} \) **makes fewer mistakes than** \( \hat{w} \), **replace** \( \hat{w} \) **by** \( w_{t+1} \)
modify PLA algorithm (black lines) by keeping best weights in pocket

initialize pocket weights \( \hat{w} \)
For \( t = 0, 1, \cdots \)
1. find a (random) mistake of \( w_t \) called \( (x_{n(t)}, y_{n(t)}) \)
2. (try to) correct the mistake by
\[
    w_{t+1} \leftarrow w_t + y_{n(t)}x_{n(t)}
\]
3. if \( w_{t+1} \) makes fewer mistakes than \( \hat{w} \), replace \( \hat{w} \) by \( w_{t+1} \)
...until enough iterations
**Pocket Algorithm**

modify PLA algorithm (black lines) by **keeping best weights in pocket**

**initialize pocket weights \( \hat{w} \)**

For \( t = 0, 1, \ldots \)

1. find a (random) mistake of \( w_t \) called \((x_{n(t)}, y_{n(t)})\)
2. (try to) correct the mistake by

\[
\begin{align*}
    w_{t+1} & \leftarrow w_t + y_{n(t)} x_{n(t)} \\
\end{align*}
\]

3. if \( w_{t+1} \) makes fewer mistakes than \( \hat{w} \), replace \( \hat{w} \) by \( w_{t+1} \)

...until **enough iterations**

return \( \hat{w} \) (**called** \( w_{POCKET} \)) as \( g \)
Pocket Algorithm

modify PLA algorithm (black lines) by keeping best weights in pocket

initialize pocket weights \( \hat{w} \)
For \( t = 0, 1, \ldots \)

1. find a (random) mistake of \( w_t \) called \((x_{n(t)}, y_{n(t)})\)
2. (try to) correct the mistake by

\[
 w_{t+1} \leftarrow w_t + y_{n(t)} x_{n(t)}
\]

3. if \( w_{t+1} \) makes fewer mistakes than \( \hat{w} \), replace \( \hat{w} \) by \( w_{t+1} \)

...until enough iterations
return \( \hat{w} \) (called \( w_{\text{POCKET}} \)) as \( g \)

a simple modification of PLA to find (somewhat) ‘best’ weights
Should we use pocket or PLA?

Since we do not know whether $\mathcal{D}$ is linear separable in advance, we may decide to just go with pocket instead of PLA. If $\mathcal{D}$ is actually linear separable, what’s the difference between the two?

1. Pocket on $\mathcal{D}$ is slower than PLA
2. Pocket on $\mathcal{D}$ is faster than PLA
3. Pocket on $\mathcal{D}$ returns a better $g$ in approximating $f$ than PLA
4. Pocket on $\mathcal{D}$ returns a worse $g$ in approximating $f$ than PLA
Should we use pocket or PLA?

Since we do not know whether $\mathcal{D}$ is linear separable in advance, we may decide to just go with pocket instead of PLA. If $\mathcal{D}$ is actually linear separable, what’s the difference between the two?

1. pocket on $\mathcal{D}$ is slower than PLA
2. pocket on $\mathcal{D}$ is faster than PLA
3. pocket on $\mathcal{D}$ returns a better $g$ in approximating $f$ than PLA
4. pocket on $\mathcal{D}$ returns a worse $g$ in approximating $f$ than PLA

Reference Answer: 1

Because pocket need to check whether $\mathbf{w}_{t+1}$ is better than $\hat{\mathbf{w}}$ in each iteration, it is slower than PLA. On linear separable $\mathcal{D}$, $\mathbf{w}_{\text{P}}$ is the same as $\mathbf{w}_{\text{PLA}}$, both making no mistakes.
1. **When Can Machines Learn?**

**Lecture 1: The Learning Problem**

**Lecture 2: Learning to Answer Yes/No**

- Perceptron Hypothesis Set
  - hyperplanes/linear classifiers in $\mathbb{R}^d$
- Perceptron Learning Algorithm (PLA)
  - correct mistakes and improve iteratively
- Guarantee of PLA
  - no mistake eventually if linear separable
- Non-Separable Data
  - hold somewhat ‘best’ weights in pocket

- **next: the zoo of learning problems**

2. Why Can Machines Learn?

3. How Can Machines Learn?

4. How Can Machines Learn Better?