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**
     \[ A \text{ takes } D \text{ and } H \text{ to get } g \]
   - **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

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)

Applicant Information

<table>
<thead>
<tr>
<th>Feature</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>

learning algorithm $\mathcal{A}$

final hypothesis $g \approx f$

('learned' formula to be used)

hypothesis set $\mathcal{H}$

(set of candidate formula)

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

- **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 $\sum_{i=1}^{d} w_i x_i < \text{threshold}$

- $\mathcal{Y}: \{+1(\text{good}), -1(\text{bad})\}$, 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

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

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

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

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

- 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’?
Learning to Answer Yes/No

Perceptron Hypothesis Set

**Perceptrons in \( \mathbb{R}^2 \)**

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

- **customer features** \( \mathbf{x} \): 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
- different line classifies customers differently

\[\text{perceptrons} \Leftrightarrow \text{linear (binary) classifiers}\]

Hsuan-Tien Lin (NTU CSIE)
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)
- 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 \( w_0 \)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

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

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

\( \ldots \) 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 \((x_{n(t)}, y_{n(t)})\)

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

2. correct the mistake by

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

\[\ldots \text{until a full cycle of not encountering mistakes}\]

\[\text{next} \text{ can follow naïve cycle } (1, \ldots, N)\]
\[\text{or precomputed random cycle}\]
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)
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)
Learning to Answer Yes/No

Perceptron Learning Algorithm (PLA)

Seeing is Believing

\[ x_1, w(t), x_2, w(t+1) \]

update: 3

\[ x_{14}, w(t), x_{15}, w(t+1) \]

update: 3

\[ x_9, w(t), x_{10}, w(t+1) \]

update: 5

\[ x_{14}, w(t), x_{15}, w(t+1) \]

update: 6

\[ x_9, w(t), x_{10}, w(t+1) \]

update: 7

\[ x_{14}, w(t), x_{15}, w(t+1) \]

update: 8

\[ x_9, w(t), x_{10}, w(t+1) \]

update: 9

\[ w(t) \]

\[ w(t+1) \]

\[ w_{PLA} \]

\[ w_{PLA} \]

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

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

\[ x_1, x_2, x_3, \ldots \]

update: 1

\[ x_9, x_14, x_3, x_9, x_14, x_9, x_14, x_9, x_14 \]

update: 2

update: 3

update: 4

update: 5

update: 6

update: 7

update: 8

update: 9

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: 5

Hsuan-Tien Lin (NTU CSIE)
Seeing is Believing

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

\[ x_1 \]

update: 1

\[ w(t) \]

update: 2

\[ x_9 \]

update: 3

\[ w(t) \]

update: 4

\[ x_{14} \]

update: 5

\[ w(t) \]

update: 6

\[ x_9 \]

update: 7

\[ w(t) \]

update: 8

\[ x_{14} \]

update: 9

\[ w(t+1) \]

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

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

\[ x_{14} \]

update: 8

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: 9

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

finally

\[ w_{PLA} \]

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 $\mathcal{D}$ 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. \( w^T_{t+1} x_n = y_n \)
2. \( \text{sign}(w^T_{t+1} x_n) = y_n \)
3. \( y_n w^T_{t+1} x_n \geq y_n w^T_t x_n \)
4. \( y_n w^T_{t+1} x_n < y_n w^T_t x_n \)

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)** $D$ allows some $w$ to make no mistake
- call such $D$ **linear separable**

(assuming $D$, does PLA always halt?)
PLA Fact: $w_t$ Gets More Aligned with $w_f$

- $w_f$ 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

$w_t$ 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,

\[
\frac{w_f^T}{\|w_f\|} \frac{w_f^T}{\|w_f\|} \geq \sqrt{T} \cdot \text{constant}
\]
Learning to Answer Yes/No

Guarantee of PLA

Fun Time

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

Define $R^2 = \max_n \|x_n\|^2$  \hspace{1cm} \rho = \min_n y_n \frac{w^T_f}{\|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 $\frac{w^T_f}{\|w_f\|} \frac{w^T_t}{\|w_t\|}$ 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 $w_f$ ⇒ 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 to Answer Yes/No

Non-Separable Data

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 \( D \) ⇔ \( y_n = g(x_n) \) usually
- how about

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

—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

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 \( w_{POCKET} \)) 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

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 $\mathcal{D}$, $\mathbf{w}_{\text{POCKET}}$ is the same as $\mathbf{w}_{\text{PLA}}$, both making no mistakes.
Summary

1. **When Can Machines Learn?**

   Lecture 1: The Learning Problem
   - 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?