Introduction of Perceptron in Python

Ko, Youngjoong

Dept. of Computer Engineering, Dong-A University
1. Basic Concepts
2. Bio-inspired Perceptron
3. Structure and Computation
4. Learning
5. Geometric Interpretation
6. Limitations of perceptron
7. Python Code and Practice
- "어떤 사과나무에 대해서 몇 년에 걸쳐 날짜 별로 사과들의 크기를 측정, 기록"
- 농부는 특정 크기가 넘을 때만 시장에 사과를 내다 팔 수 있다고 할 때,
- Q: 올해 Day-50에 사과를 내다 팔 수 있을까? 없을까?
Basic Concepts of Perceptron

- **Illustration Example (Apple Tree)**

  Default
  
  size = 5          size = 10          size = 15          size = 20          size = 25

  If size > 30, sell an apple!

  ![Illustration of an apple tree](image)

  Day 0 | Day 10 | Day 20 | Day 30 | Day 40 | Sell

  상황 1: 작년까지 이 사과나무는 위의 경향대로 사과 열매를 맛였다.
  조건: 사과의 크기가 30이 넘으면 팔 수 있다.

  Question: 예년에 사과를 팔 수 있음까?

  **Very Typical Regression Problem**
Basic Concepts of Perceptron

Illustration Example (Apple Tree)

If size > 30, sell an apple!

- Default size = 5
- size = 10
- size = 15
- size = 20
- size = 25

- Day 0
- Day 10
- Day 20
- Day 30
- Day 40

Regression

\[ y = ax + b \]

Size = 0.5*day + 5

Activation point to sell an apple!

\[ \rightarrow \text{learn the parameter 'a' and 'b' from the data} \]
**Basic Concepts of Perceptron**

- **Illustration Example (Apple Tree)**

\[
 y = ax + b 
\]

\[
 If \ y > 30 \rightarrow sell \ an \ apple \]

\[
 Y = WX + b 
\]

**Activation function**

- **Step Function**
Bio-inspired Learning

Our brains are made up of a bunch of little units, called neurons, that send electrical signals to one another.

- The rate of firing tells us how “activated” a neuron is.
- The incoming neurons are firing at different rates (i.e., have different activations).

The Goal is that we are going to think of our learning algorithm as a single neuron.
Bio-inspired Perceptron

- **Processing Unit**
  - Neuron vs. Node

- **Connection**
  - Synapse vs. Weight
Structure and Computation

- **Terminology for perceptron**
  - Layer, Node, Weight, Activation function and Learning

- **A simple example**
The neuron receives input from $D$-many other neurons
- One for each input feature
- The strength of these inputs are the feature values

Each incoming connection has a weight and the neuron simply sums up all the weighted inputs
- Based on this sum, it decides whether to “fire” or not
- Firing is interpreted as being a positive example and not Firing is a negative example
  - If the weighted sum is positive, it “fires” and otherwise it doesn’t fire
**Structure of Perceptron**

- Input layer: \((d+1)\) nodes (feature vector, \(\mathbf{x} = (x_1, \ldots, x_d)\))
- Output layer: 1 node (*binary* linear classifier)
The weights \((w = (w_0, ..., w_d))\) of these neurons are fairly easy to interpret.

- Suppose that a feature, for instance “is this a System’s class?” gets a zero weight
  - the activation is the same regardless of the value of this feature. So features with zero weight are ignored.

- Feature with positive weights are indicative of positive examples
  - Because they cause the activation to increase

- Feature with negative weights are indicative of negative examples
  - Because they cause the activation to decrease
Structure and Computation

- **Computation of Perceptron**
  - Input layer: *Just transfer*
  - Output layer: *summation and activation function*

\[
y = \tau(s) = \tau(\sum_{i=1}^{d} w_i x_i + b) = \tau(\mathbf{w}^T \mathbf{x} + b)
\]

\[
\phi(w) = \begin{cases} 
+1, & s \geq 0 \\
-1, & s < 0 
\end{cases}
\]

- Binary Linear Classifier

\[
d(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b > 0 \text{ 이면 } \mathbf{x} \in \omega_1 \\
d(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b < 0 \text{ 이면 } \mathbf{x} \in \omega_2
\]
Example of Perceptron Computation

- OR classification
- \(d(x) = x_1 + x_2 - 0.5\)
Perceptron Learning

- Training set: \( X = \{ (x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N) \}, t_i = 1 \text{ or } -1 \)
- Try to look for \( w = (w_0, \ldots, w_d) \) and \( b \)

Ex) And Problem

\[
\begin{align*}
a &= (0,0)^T, \quad b &= (1,0)^T, \quad c &= (0,1)^T, \quad d &= (1,1)^T \\
t_a &= -1, \quad t_b = -1, \quad t_c = -1, \quad t_d = 1
\end{align*}
\]
Learning of Perceptron

- **General Designing Steps for Learning in Pattern Recognition**
  - Step 1: Building up Classification Model
  - Step 2: Cost function, $J(\theta)$
  - Step 3: Finding $\theta$ to optimize $J(\theta)$

- This problem is changed into an Optimization Problem!
Learning of Perceptron

- **Step 1**
  - Parameter Set: $\theta = \{w, b\}$

- **Step 2**
  - Cost Function: $Y$ is a set of error training examples
    \[ J(\Theta) = \sum_{x_k \in Y} (-t_k)(w^T x_k + b) \]

- **Step 3**
  - Gradient Descent Method
  - Move $-\frac{\partial J}{\partial \theta}$ direction
  - Learning Rate:
Learning of Perceptron

- Sketch of algorithm
  - Setting up Initial Parameters for $\theta = \{w, b\}$

\[
\Theta(h + 1) = \Theta(h) - \rho(h) \frac{\partial J(\Theta)}{\partial \Theta}
\]

\[
\frac{\partial J(\Theta)}{\partial w} = \sum_{x_k \in Y} (-t_k) x_k
\]

\[
\frac{\partial J(\Theta)}{\partial b} = \sum_{x_k \in Y} (-t_k)
\]

\[
w(h + 1) = w(h) + \rho(h) \sum_{x_k \in Y} t_k x_k
\]

\[
b(h + 1) = b(h) + \rho(h) \sum_{x_k \in Y} t_k
\]

또는

\[
\hat{w}(h + 1) = \hat{w}(h) + \rho(h) \sum_{x_k \in Y} t_k \hat{x}_k
\]
Learning of Perceptron

- **Perceptron Learning in Batch Mode**

입력: 훈련 집합 \( X = \{ (x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N) \} \), 학습률 \( \rho \)

출력: 퍼셉트론 가중치 \( w, b \)

알고리즘:

1. \( w \)와 \( b \)를 초기화한다.
2. repeat {
   3. \( Y = \emptyset \); 
   4. for \( i = 1 \) to \( N \) {
      5. \( y = \tau( w^T x_i + b ) \); \hspace{1cm} // (4.2)로 분류를 수행함
      6. if \( ( y \neq t_i ) \) \( Y = Y \cup x_i \); \hspace{1cm} // 오분류된 샘플 수집
      7. }
   8. \( w = w + \rho \sum_{x_k \in Y} t_k x_k \); \hspace{1cm} // (4.7)로 가중치 갱신
   9. \( b = b + \rho \sum_{x_k \in Y} t_k \)
   10. } until \( ( Y = \emptyset ) \);
11. \( w \)와 \( b \)를 저장한다.
Learning of Perceptron

- Perceptron Learning in Pattern Mode

**Algorithm 5** \texttt{PERCEPTRONTRAIN}(D, MaxIter)

1. \( w_d \leftarrow 0 \), for all \( d = 1 \ldots D \)  
   \( b \leftarrow 0 \)  
   // initialize weights
2. for iter = 1 \ldots MaxIter do
3.   for all \( (x,y) \in D \) do
4.     \( a \leftarrow \sum_{d=1}^{D} w_d x_d + b \)  
5.     if \( ya \leq o \) then
6.       \( w_d \leftarrow w_d + yx_d \), for all \( d = 1 \ldots D \)  
7.       \( b \leftarrow b + y \)  
8.     end if
9.   end for
10. end for
11. return \( w_0, w_1, \ldots, w_D, b \)

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

1. \( a \leftarrow \sum_{d=1}^{D} w_d \hat{x}_d + b \)  
   // compute activation for the test example
2. return \( \text{SIGN}(a) \)
Learning of Perceptron

- An Example

\[ w(0) = (-0.5, 0.75)^T, \quad b(0) = 0.375 \]

\[ d(x) = -0.5x_1 + 0.75x_2 + 0.375 \]
\[ Y = \{a, b\} \]

\[ w(1) = w(0) + 0.4(t_a \cdot a + t_b \cdot b) = \begin{pmatrix} -0.5 \\ 0.75 \end{pmatrix} + 0.4 \begin{pmatrix} -0 \\ 0 \end{pmatrix} = \begin{pmatrix} -0.1 \\ 0.75 \end{pmatrix} \]
\[ b(1) = b(0) + 0.4(t_a + t_b) = 0.375 + 0.4 \cdot 0 = 0.375 \]

\[ d(x) = -0.1x_1 + 0.75x_2 + 0.375 \]
\[ Y = \{a\} \]

\[ w(2) = w(1) + 0.4(t_a \cdot a) = \begin{pmatrix} -0.1 \\ 0.75 \end{pmatrix} + 0.4 \begin{pmatrix} -0 \\ 0 \end{pmatrix} = \begin{pmatrix} -0.1 \\ 0.75 \end{pmatrix} \]
\[ b(2) = b(1) + 0.4(t_a) = 0.375 - 0.4 = -0.025 \]
Why this particular update achieves better job

- Some current set of parameters \( w, b \)
- An example \((x_i, t_i)\), suppose this is a positive example, so \( t_i = 1 \)
- Compute an activation \( a \), and make an error \( a < 0 \)
Geometric Interpretation

- **What does the decision boundary of a perceptron look like?**
  - The sign of the activation, \( a \), changes from -1 to +1
  - The set of points \( \mathbf{x} \) achieves zero activation
    - The points are not clearly positive nor negative

- **Consider the case where there is no “bias” term**
  - The decision boundary \( B \) is:
    \[
    B = \left\{ \mathbf{x} : \sum_{d} w_d x_d = 0 \right\}
    \]
  - If two vectors have a zero dot product, they are perpendicular
  - The decision boundary: the plane perpendicular to \( \mathbf{w} \)
The scale of the weight vector is irrelevant from the perspective of classification

- Work with normalized weight vector $\mathbf{w}$, $||\mathbf{w}|| = 1$

The value $\mathbf{w} \cdot \mathbf{x}$ is just the distance of $\mathbf{x}$ from the origin when projected onto the vector $\mathbf{w}$

This distance along $\mathbf{w}$ is exactly the activation of that example, with no bias
The role of the bias term

- Previously, the threshold would be at zero
- The bias simply moves this threshold
- Bias term $b$ is added to get the overall activation
  - The projection plus $b$ is then compared against zero

- From a geometric perspective, the role of the bias is to shift the decision boundary away from the origin, in the direction of $w$

- It is shifted exactly $b$ units
  - $b$ is positive, the boundary is shifted away from $w$
  - $b$ is negative, the boundary is shifted toward $w$

- A positive bias means that more examples should be classified positive
  - By moving the decision boundary in the negative direction, more space yields a positive classification
The perceptron update can also be considered geometrically.

Here, we have a current guess as to the hyperplane, and positive example comes in that is currently mis-classified.

The weights are updated: $w = w + xt$

- The weight vector is changed enough so this training example is now correctly classified.
Limitations of Perceptron

- The limitation is that its decision boundaries can only be linear
  - XOR problem

- You might ask is: “Do XOR-like problems exist in the real world?”
  - The answer is “YES.”

- Two alternative approaches to taking key ideas from the perceptron and generating classifiers with non-linear decision boundaries
  - Neural Networks: combine multi-layer perceptrons in a single framework
  - Kernels: find computationally efficient ways of doing feature mapping in a computationally and statistically efficient way
Python Code and Practice

- You should install Python 2.7 and Numpy
- Download from: [http://nlpmlir.blogspot.kr/2016/01/perceptron.html](http://nlpmlir.blogspot.kr/2016/01/perceptron.html)
- Homework
References

- 오일석. 패턴인식. 교보문고.


- [http://ciml.info/](http://ciml.info/)
Thank you for your attention!

http://web.donga.ac.kr/yjko/

고 영 중