Machine Learning I (DATS 6202)

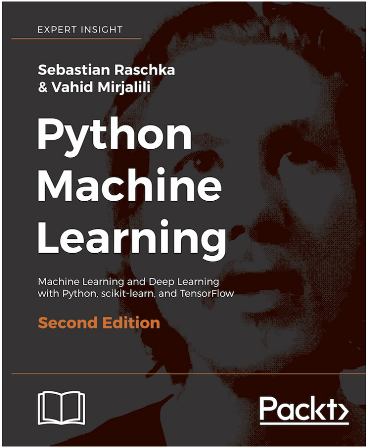
Perceptron and Adaline

Yuxiao Huang

Data Science, Columbian College of Arts & Sciences
George Washington University
yuxiaohuang@gwu.edu

September 25, 2018

Reference



Picture courtesy of the website of the book code repository and info resource

Reference

- This set of slices is an excerpt of the book by Raschka and Mirjalili, with some trivial changes by the creator of the slides
- Please find the reference to and website of the book below:
  - *Raschka S. and Mirjalili V. (2017). Python Machine Learning. 2nd Edition.*
  - `https://sebastianraschka.com/books.html`
- Please find the website of the book code repository and info resource below:
  - `https://github.com/rasbt/python-machine-learning-book-2nd-edition`

Overview

1. Perceptrons
2. Adaptive Linear Neuron
3. Batch VS Stochastic gradient descent

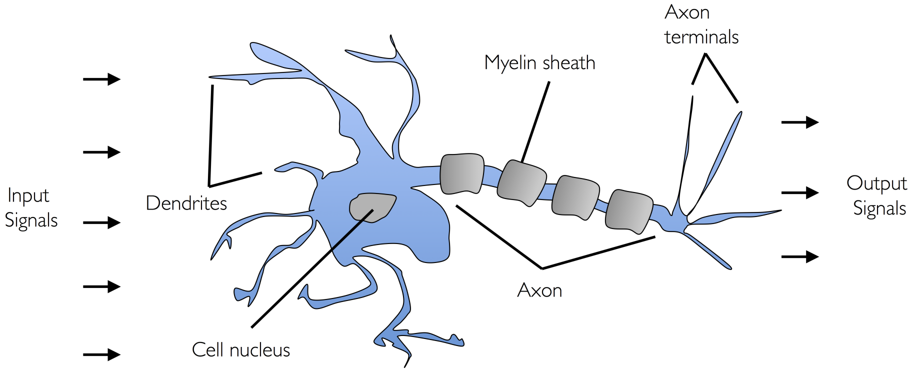
Neurons

- In simple terms, neurons take as input the chemical and electrical signals, integrate the signals, and output the integrated signals

$$\text{input} \rightarrow \text{something magical} \rightarrow \text{output}$$

- Figure 1 (see next page) illustrates a neuron

Figure 1

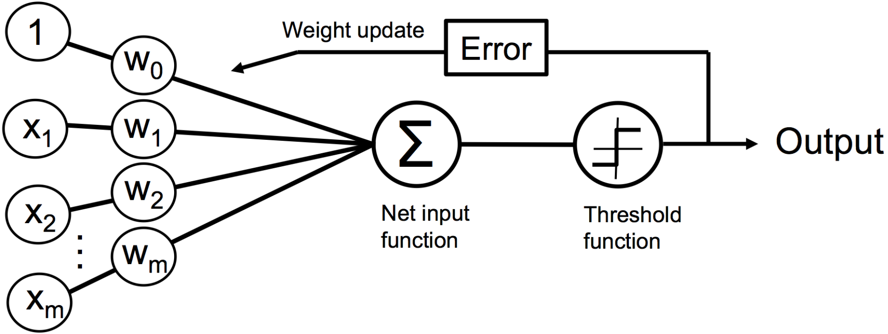


Input
Signals
Dendrites
Cell nucleus
Myelin sheath
Axon
terminals
Axon
Output
Signals

Perceptrons

- Perceptrons (a.k.a., artificial neurons) were proposed to mimic neurons
- Specifically, the "magical" part is simulated by:
  - adding weights (i.e., the parameters) to the input signals
  - integrating the signals using a net input function
  - predicting the output using a threshold function
  - updating the weights based on the errors
- Figure 2 (see next page) illustrates a perceptron

Figure 2



1
w_0
x_1
w_1
x_2
w_2
x_m
w_m
Weight update
Error
Σ
Net input function
Threshold function
Output

The net input function

- The net input function takes as input the signals and their weights, and outputs the weighted sum of the signals:

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{j=0}^{m} \mathbf{w_j x_j} = \mathbf{w}^T\mathbf{x}.$$

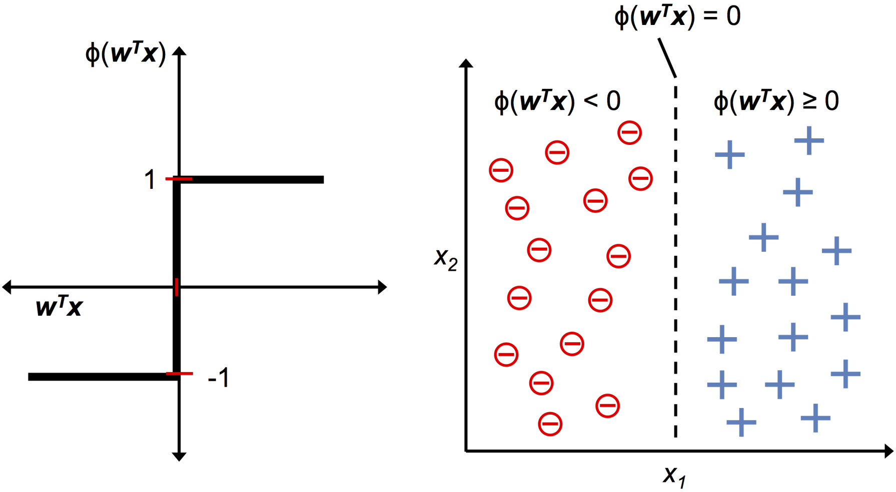
The threshold function

- The threshold function (a variant of the unit step function) takes as input the output of the net input function, and outputs a binary decision:

$$\hat{y} = \phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}.$$

- Figure 3 (see next page) illustrates how the threshold function works

Figure 3



$\phi(\boldsymbol{w}^T\boldsymbol{x})$
1
$\boldsymbol{w}^T\boldsymbol{x}$
-1
$\phi(\boldsymbol{w}^T\boldsymbol{x}) = 0$
$\phi(\boldsymbol{w}^T\boldsymbol{x}) < 0$
$\phi(\boldsymbol{w}^T\boldsymbol{x}) \geq 0$
x_2
x_1

The updating rule

- The weights are updated based on the difference between the predicted class and the actual one:

$$w_j = w_j + \Delta w_j$$

where w_j is the weight of feature x_j and

$$\Delta w_j = \eta\left(y^i - \hat{y}^i\right)x^i_j$$

- Here:
    - y^i is the actual class of sample i
    - $\hat{y}^i$ is the predicted class of sample i
    - x^i_j is feature x_j of sample i
    - η is the learning rate

The updating rule: why it works

- **Q:** Why does the updating rule work:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta\left(y^i - \hat{y}^i\right)x_j^i$$

The updating rule: why it works

- **Q:** Why does the updating rule work:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta \left(y^i - \hat{y}^i \right) x_j^i$$

- **A:** Because it pulls the predicted class ($\hat{y}$) closer to the real one (y)

The updating rule: why it works

- **Q:** Why does the updating rule work:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta\left(y^i - \hat{y}^i\right)x_j^i$$

- **A:** Because it pulls the predicted class ($\hat{y}$) closer to the real one (y)
  - Assume $y^i = 1$, $\hat{y}^i = -1$, and $x_j^i > 0$, then:
    1. $\Delta w_j > 0$
    2. $w_j \uparrow$
    3. $z = \mathbf{w}^T\mathbf{x} \uparrow$
    4. $P(z \geq 0) \uparrow$
    5. $P(\hat{y} = 1) \uparrow$

The updating rule: the problem

- **Q:** Does the updating rule have any problem:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta \left(y^i - \hat{y}^i \right) x_j^i$$

The updating rule: the problem

- **Q:** Does the updating rule have any problem:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta \left(y^i - \hat{y}^i \right) x_j^i$$

- **A:** It does not distinguish the distance between $z = \mathbf{w}^T \mathbf{x}$ and 0

The updating rule: the problem

- **Q:** Does the updating rule have any problem:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta\Big(y^i - \hat{y}^i\Big)x_j^i$$

- **A:** It does not distinguish the distance between $z = \mathbf{w}^T\mathbf{x}$ and 0
  - Assume $y^i = 1$ and $\hat{y}^i = -1$
  - Intuitively:
    - Δw_j should be much larger when z is much smaller than 0, say -100
    - Δw_j should be much smaller when z is much closer to 0, say -0.01
    - However, Δw_j is the same for both kinds of z:

$$\Delta w_j = 2\eta x_j^i$$

The updating rule: the solution

- **Q:** Is there a way to fix the problem?

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta\left(y^i - \hat{y}^i\right)x_j^i$$

The updating rule: the solution

- **Q:** Is there a way to fix the problem?

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta \Big(y^i - \hat{y}^i \Big) x_j^i$$

- **A:** Yes. We can simply replace $\hat{y}$ with z

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta \Big(y^i - z^i \Big) x_j^i$$

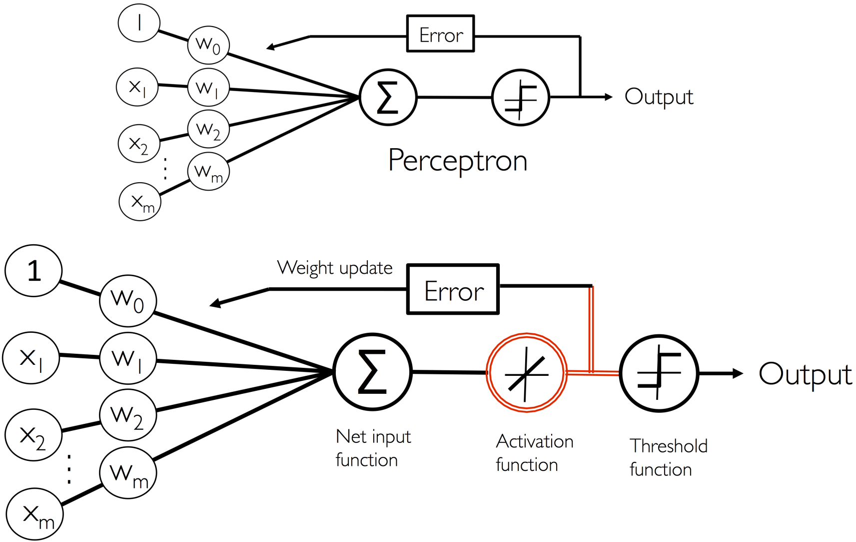
Here

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{j=0}^{m} \mathbf{w_j x_j} = \mathbf{w}^T \mathbf{x}.$$

Adaptive Linear Neuron

- The new updating rule leads to another kind of perceptron, Adaptive Linear Neuron (Adaline)
- The difference between perceptron and Adaline is shown in Figure 4 (see next page)

Figure 4


1
w_0
x_1
w_1
x_2
w_2
w_m
x_m
Error
Output
Perceptron
1
w_0
x_1
w_1
x_2
w_2
x_m
w_m
Weight update
Error
Output
Net input function
Activation function
Threshold function

The objective function

- Here, the objective function is the Sum of Squared Errors (SSE):

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i} \Big(y^i - \phi\big(z^i\big) \Big)^2,$$

where:

  - y^i is the class of sample i
  - $\phi(z)$, the activation function, is the identity function:

$$\phi(z) = z = \mathbf{w}^T \mathbf{x}.$$

Estimating the parameters

- The parameters, $\mathbf{w}$, can be estimated by minimizing the objective function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i}\left(y^{i} - \phi(z^{i})\right)^{2},$$

- Again, this can be done by gradient descent

The updating rule

- Using gradient descent, the updating rule can be written as

$$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta\nabla J(\mathbf{w}).$$

  Here:
  - η is the learning rate
  - $\nabla J(\mathbf{w})$ the gradient of $J(\mathbf{w})$
- **Q:** Why the negative sign?

The updating rule

- Using gradient descent, the updating rule can be written as

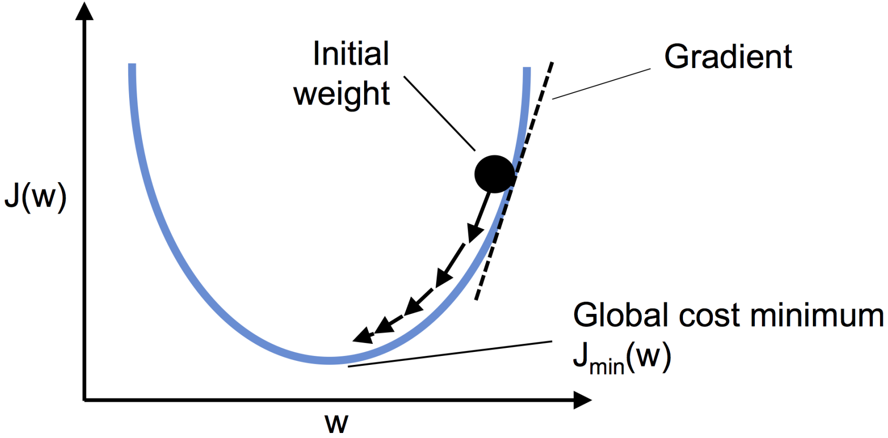
$$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta\nabla J(\mathbf{w}).$$

  Here:
  - η is the learning rate
  - $\nabla J(\mathbf{w})$ the gradient of $J(\mathbf{w})$
- **Q:** Why the negative sign?
- **A:** Two kinds of explanations

The visual explanation

- The negative sign updates the weights by taking a step in the opposite direction of the gradient
- Thus we can minimize, rather than maximize, the objective function
- This is also the reason why it is called gradient descent, rather than gradient assent
- Figure 5 (see next page) shows the idea

Figure 5



The mathematical explanation

- The updating rule for w_j is

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = -\eta \frac{\partial J}{\partial w_j}.$$

Here, $\frac{\partial J}{\partial w_j}$ is the partial derivative of J with respect to w_j:

$$\frac{\partial J}{\partial w_j} = -\sum_i \Big(y^i - \phi\big(z^i\big)\Big) x_j^i.$$

- Thus, the updating rule for w_j can be written as

$$w_j = w_j + \eta \sum_i \Big(y^i - \phi\big(z^i\big)\Big) x_j^i.$$

- **Q:** Does this look familiar?

Perceptron review: why it works

- **Q:** Why does the updating rule work:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta\left(y^i - \hat{y}^i\right)x_j^i$$

- **A:** Because it pulls the predicted class ($\hat{y}$) closer to the actual one (y)
- Assume $y^i = 1$, $\hat{y}^i = -1$, and $x_j > 0$, then:
  1. $\Delta w_j > 0$
  2. $w_j \uparrow$
  3. $z = \mathbf{w}^T\mathbf{x} \uparrow$
  4. $P(z \geq 0) \uparrow$
  5. $P(\hat{y} = 1) \uparrow$

Perceptron review: the problem

- **Q:** Does the updating rule have any problem:

$$w_j = w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = \eta\Big(y^i - \hat{y}^i\Big)x_j^i$$

- **A:** It does not distinguish the distance between $z = \mathbf{w}^T\mathbf{x}$ and 0
- Assume $y^i = 1$ and $\hat{y}^i = -1$. Intuitively:
  - Δw_j should be much larger when z is much smaller than 0, say -100
  - Δw_j should be much smaller when z is much closer than 0, say -0.01
  - However, Δw_j is the same for both kinds of z:

$$\Delta w_j = 2\eta x_j^i$$

Perceptron review: the solution

- **Q:** Is there a way to fix the problem?

$$w_j = w_j + \eta\Big(y^i - \hat{y}^i\Big)x_j^i$$

- **A:** Yes. We can simply replace $\hat{y}$ with z

$$w_j = w_j + \eta\Big(y^i - z^i\Big)x_j^i$$

Here

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{j=0}^{m} \mathbf{w_j x_j} = \mathbf{w}^T\mathbf{x}.$$

Adaline review: the mathematical explanation

- Thus, the updating rule for w_j can be written as

$$w_j = w_j + \eta \sum_i \Big(y^i - \phi(z^i)\Big) x_j^i.$$

- **Q:** Does this look familiar?

Adaline review: the mathematical explanation

- Thus, the updating rule for w_j can be written as

$$w_j = w_j + \eta \sum_i \Big(y^i - \phi(z^i) \Big) x_j^i.$$

- **Q:** Does this look familiar?
- **A:** It is almost the same as the updating rule we just proposed:

$$w_j = w_j + \eta \Big(y^i - z^i \Big) x_j^i.$$

Batch VS Stochastic gradient descent

- The only difference between the two updating rules is that, in each epoch:
  - Adaline updates the weights for the whole training set
  - Perceptron updates the weights for each sample
- This difference leads to two kinds of gradient descent
  - Batch gradient descent: updating the parameters for the whole training set
  - Stochastic gradient descent: updating the parameters for each sample

Stochastic gradient descent

- Since stochastic gradient descent updates parameters more frequently:
  - it usually reaches convergence faster
  - it usually escapes local minimum easier
- To obtain good results:
  - randomize the order of the data in the training set
  - shuffle the data in the training set each epoch