Lecture 5: Adaline algorithm

Dr. Huiping Cao

Outline

- Adaline model
- Gradient descent algorithm
- Implementation

Adaline algorithm

- **ADAptive LInear NEuron (Adaline)**[*]: single layer neural network
- An improvement of the perceptron algorithm
- The key concepts of **defining and minimizing continuous cost functions**.

[*] *An Adaptive "Adaline" Neuron using chemical "Memistors"*, Technical Report Number 1553-2, B. Widow and others, Stanford Electron Labs, Stanford, CA, October 1960

Difference of Adaline and Perceptron

- Perceptron: weights are updated using a unit step function

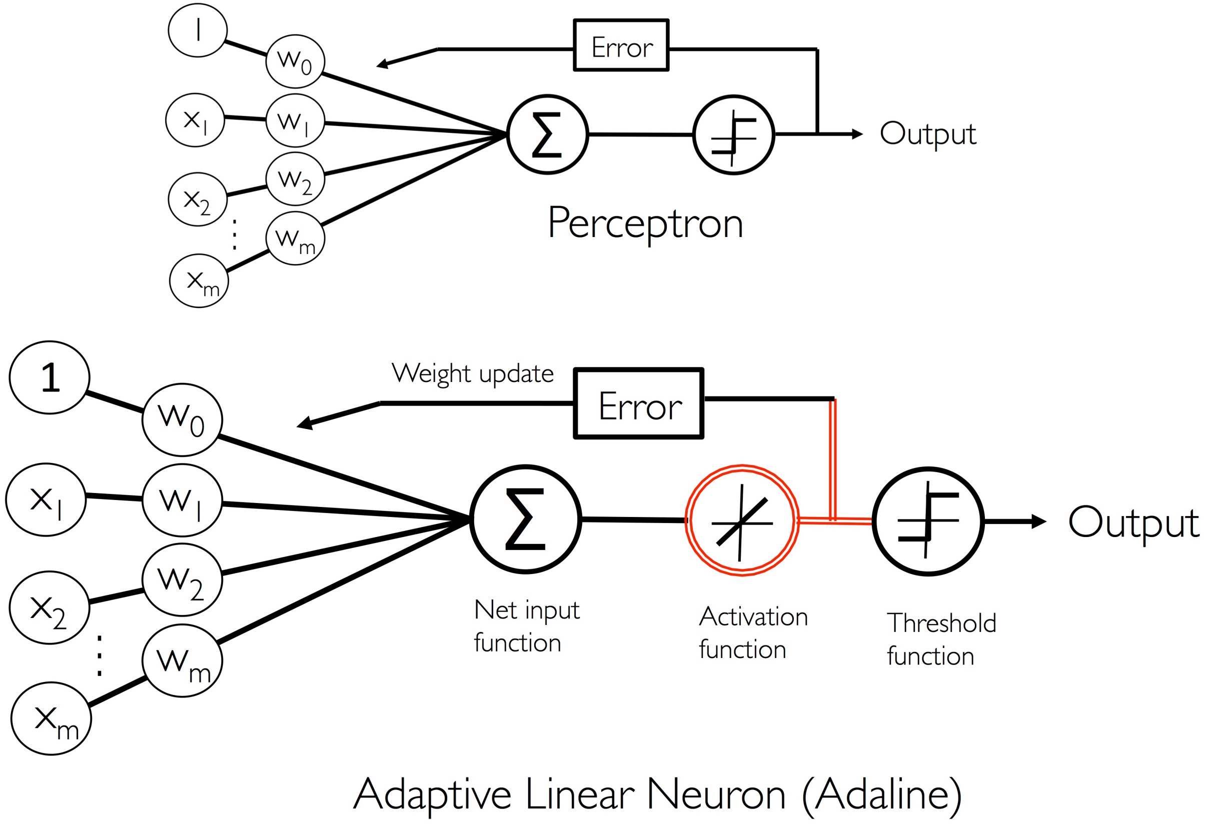
$$\varphi(z) = \begin{cases} 1 \; if \; z \geq 0 \\ -1 \; otherwise \end{cases}$$

- Adaline: weights are updated based on a linear activation function

$$\varphi(z) = \phi(\boldsymbol{w}^{\intercal}\boldsymbol{x}) = \boldsymbol{w}^{\intercal}\boldsymbol{x}$$

  - This activation function is the identity function of the net input.

Difference of Adaline and Perceptron



Objective function

- Objective function: cost function, $J(\boldsymbol{w})$
- **Adaline cost function**: Sum Squared Errors (SSE) between the calculated outcome and the true class label

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

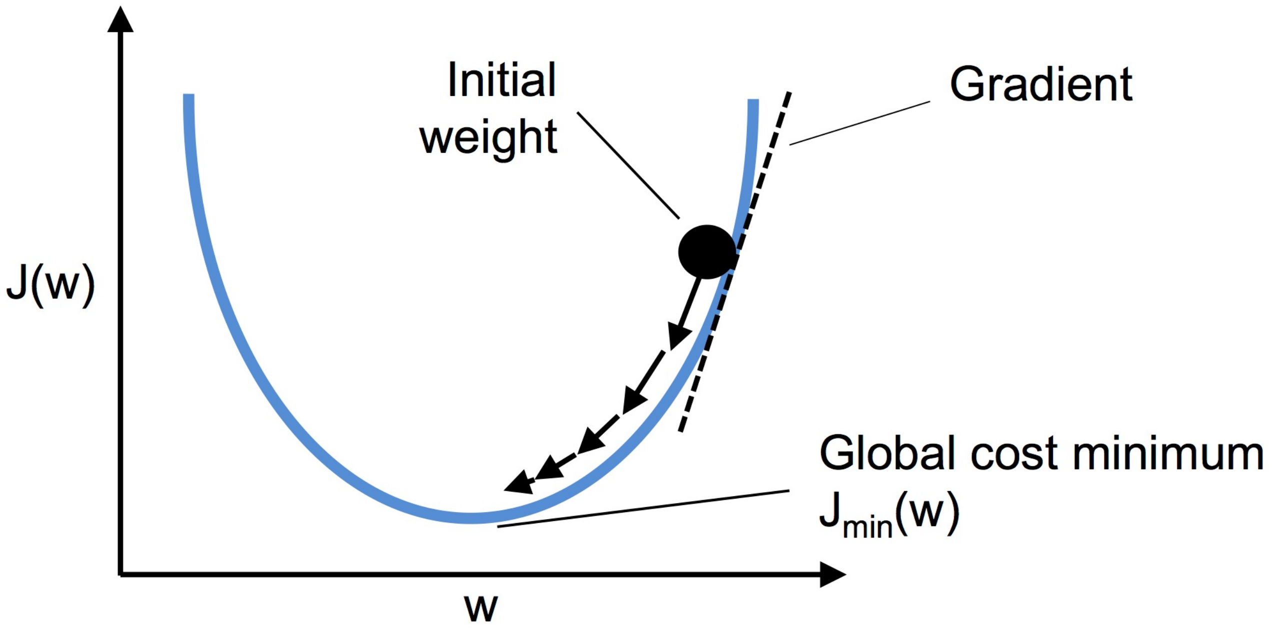
  - Term 1/2
  - Properties: differential, convex.

Gradient descent

- **Idea of gradient descent**: *climbing down a hill* until a local/global optimum is reached. In each iteration, we take a step in the **opposite direction of the gradient**. The step size is determined by a learning rate η and the slope of the gradient.

Move in the opposite direction of the gradient

- Each coordinate of the gradien tells us the slope if you were tc increase along that direction.
- If the slope along that directior is negative, then you (i) get a decrease as you move forward along that direction, (ii) get a greater increase if you move backward along the direction.



Weight updates using gradient descent

- Update the weights

$$\boldsymbol{w} = \boldsymbol{w} + \Delta\boldsymbol{w}$$

- The **weight change** Δw
  - Let $\nabla J(w)$ be the gradient of *J*(**w**)
  - A step in the opposite direction of the the gradient of *J*(**w**)

$$\Delta w = -\eta \nabla J(w)$$

$\nabla J(w)$ calculation

- The gradient of the cost function is computed using the **partial derivative** of the cost function w.r.t. each weight w_j.

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_i (y^{(i)} - \varphi(z^{(i)}))^2 \ldots (1)$$

$$\varphi(z) = \phi(\boldsymbol{w}^\intercal \boldsymbol{x}) = \boldsymbol{w}^\intercal \boldsymbol{x} \qquad \ldots (2)$$

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2$$

$$= \frac{1}{2}\frac{\partial}{\partial w_j}\sum_i (y^{(i)} - \phi(z^{(i)}))^2$$

$\nabla J(w)$ calculation

Rule 1

$$\frac{d}{dx}x^n = nx^{n-1}$$

Implicit differentiation rule (rule 2)

$$\frac{df}{dx} = \frac{df}{dq} \times \frac{dq}{dx}$$

E.g., $x = q^2$

$$\frac{d}{dx}(x) = \frac{d}{dx}(q^2) = \frac{d}{dq}(q^2)\frac{dq}{dx} = 2q\frac{dq}{dx}$$

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j}\frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\ &= \frac{1}{2}\frac{\partial}{\partial w_j}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\ &= \sum_i (y^{(i)} - \phi(z^{(i)}))\frac{\partial}{\partial w_j}(y^{(i)} - \phi(z^{(i)}))\end{aligned}$$

$\nabla J(w)$ calculation

$$\varphi(z) = \phi(\boldsymbol{w}^\top \boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$$

$$\boldsymbol{w}^T \boldsymbol{x} = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m$$

$$\frac{\partial}{\partial w_j}(y^{(i)} - \textstyle\sum_j w_j\, x_j{}^{(i)}) = -x_j{}^{(i)}$$

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\
&= \frac{1}{2} \frac{\partial}{\partial w_j} \sum_i (y^{(i)} - \phi(z^{(i)}))^2 \\
&= \sum_i (y^{(i)} - \phi(z^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - \phi(z^{(i)})) \\
&= \sum_i (y^{(i)} - \phi(z^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - \boldsymbol{w}^\top \boldsymbol{x}^{(i)}) \\
&= \sum_i (y^{(i)} - \phi(z^{(i)})) \frac{\partial}{\partial w_j} (y^{(i)} - \sum_j w_j\, x_j{}^{(i)}) \\
&= \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) (-x_j{}^{(i)}) \\
&= -\left(\sum_i \left(y^{(\mathrm{i})} - \phi\left(z^{(i)}\right)\right) x_j{}^{(i)}\right)
\end{aligned}$$

Δw_j

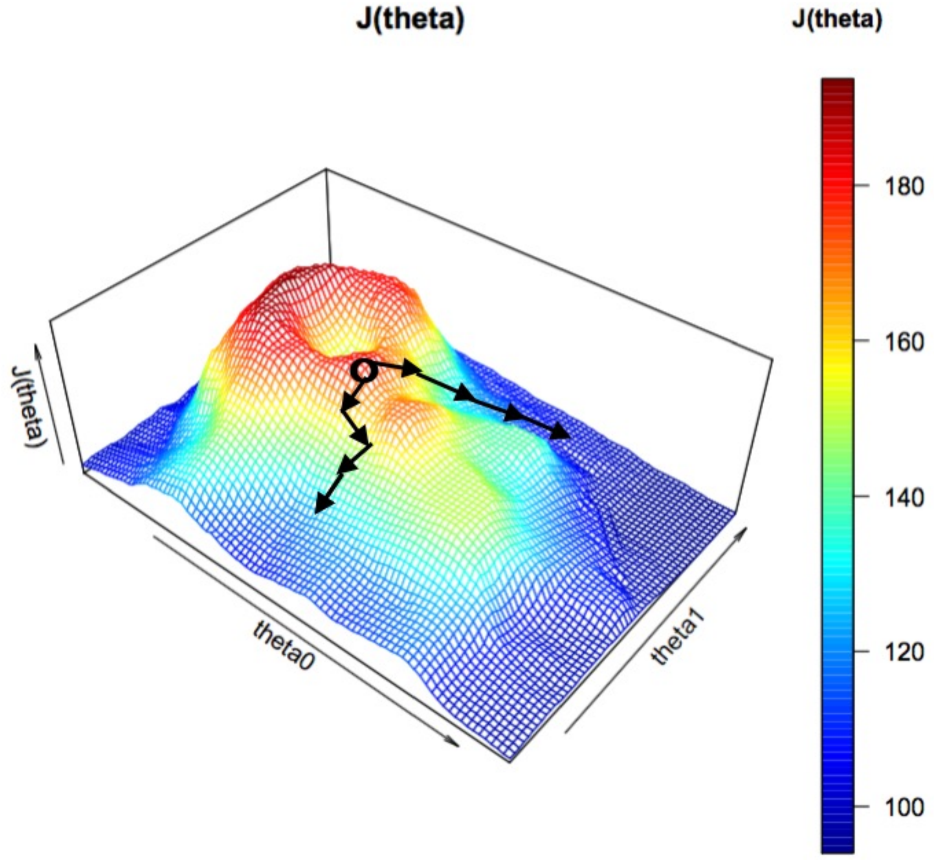
- Weight update

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = -\eta \left(-\left(\sum_i \left(y^{(\mathrm{i})} - \phi\left(z^{(i)}\right) \right) x_j{}^{(i)} \right) \right) = \eta \left(\sum_i \left(y^{(\mathrm{i})} - \phi\left(z^{(i)}\right) \right) x_j{}^{(i)} \right)$$

Batch gradient descent

- **Idea**: consider an easy case $minimize_{w_0, w_1} J(w_{0,} w_1)$
  - Start with some $w_{0,} w_1$ values (say $w_0 = 0.1, w_1 = 0.2$)
  - Keep changing $w_{0,} w_1$ to reduce $J(w_{0,} w_1)$ until we hopefully end up at a minimum
- In the general case: $minimize_{w_0, w_1, \ldots, w_m} J\left(w_{0,} w_1, \ldots, w_m\right)$
- The weight update is calculated based on **all** samples in the training set, this approach is also called **batch gradient descent**.

$$minimize_{\theta_0,\theta_1} J(\theta_0,\theta_1)$$



Gradient descent algorithm

- Repeat until convergence (or run a given number of iterations)
  - Generate net inputs and calculate prediction error
  - $w_j = w_j + \Delta w_j = w_j - \eta \frac{\partial}{\partial w_j} J(w_0, w_1, \ldots, w_m)$ for j = 0, …., m

Not for each training sample

- w_js need to be updated simultaneously. Consider $J(w_0, w_1)$
  - Correct

$$temp_0 = w_0 - \eta \frac{\partial}{\partial w_0} J(w_0, w_1)$$

$$temp_1 = w_1 - \eta \frac{\partial}{\partial w_1} J(w_0, w_1)$$

$$w_0 = temp_0$$

$$w_1 = temp_1$$

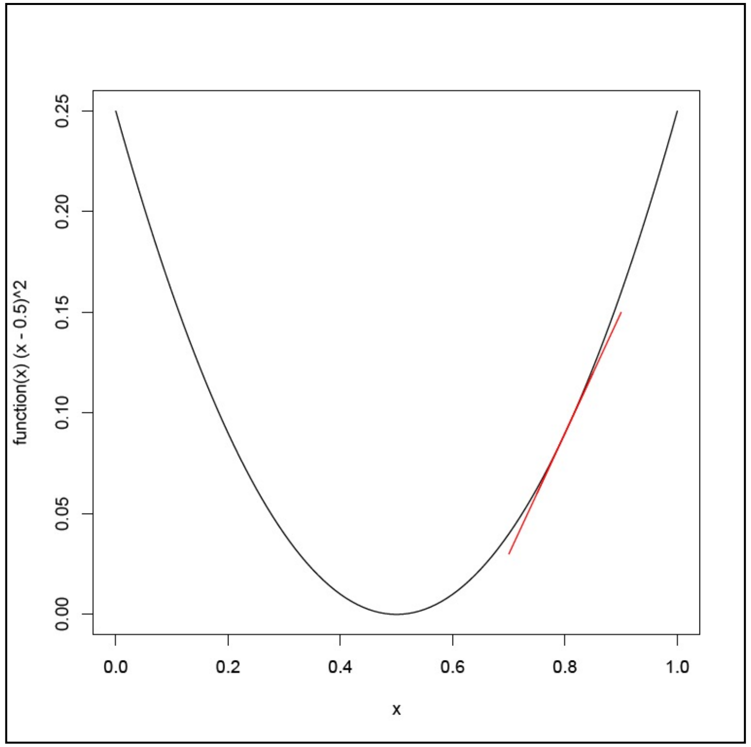
Wrong:

$$w_0 = w_0 - \eta \frac{\partial}{\partial w_0} J(w_0, w_1)$$

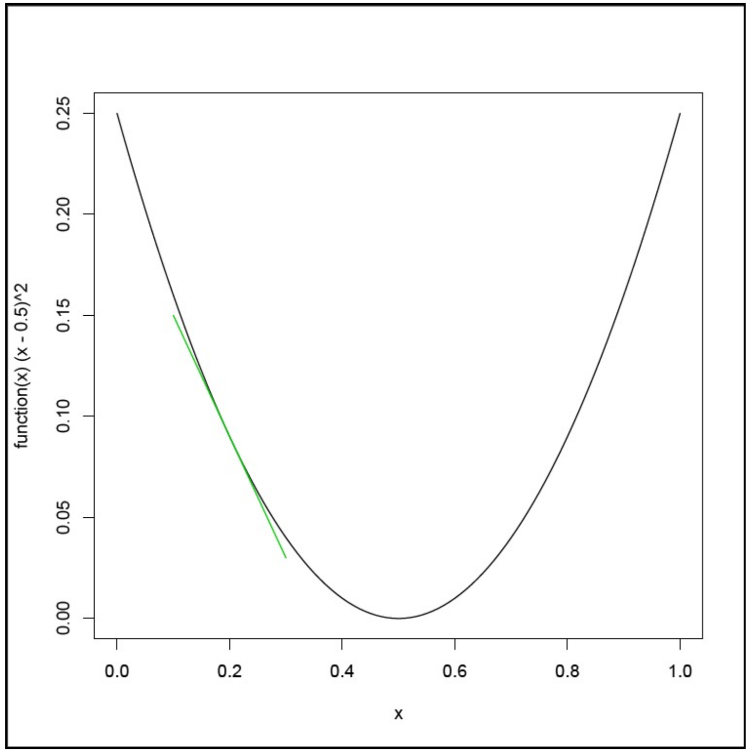
$$w_1 = w_1 - \eta \frac{\partial}{\partial w_1} J(w_0, w_1)$$

Gradient descent algorithm - derivative

$w_j = w_j - \eta \times$ positive number



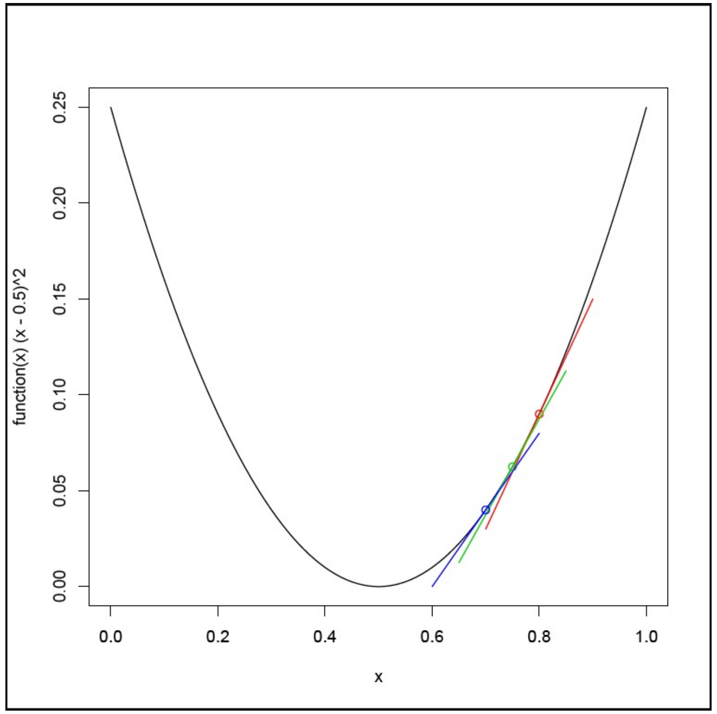
$w_j = w_j - \eta \times$ negative number



Gradient descent algorithm - derivative

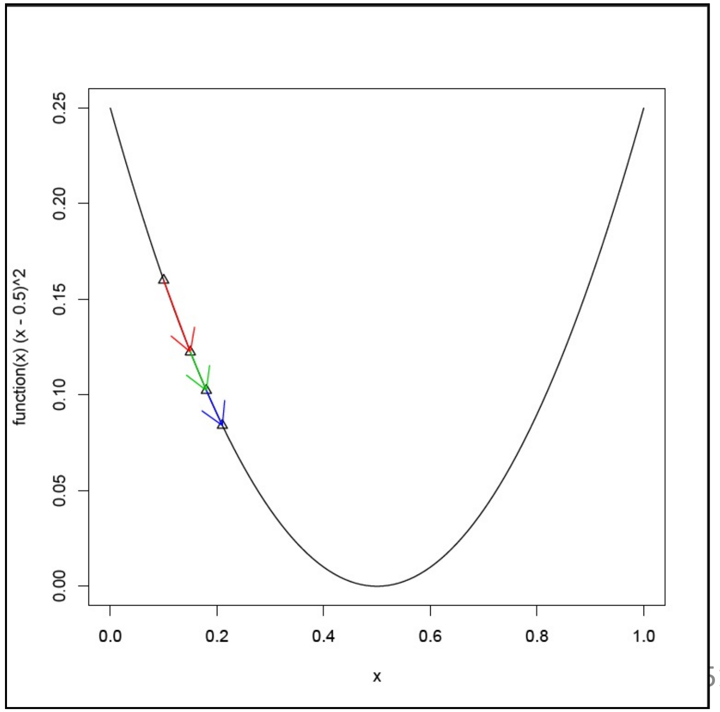
- $w_j = w_j - \eta \cdot derivative = w_j - \eta \cdot 0 = w_j$

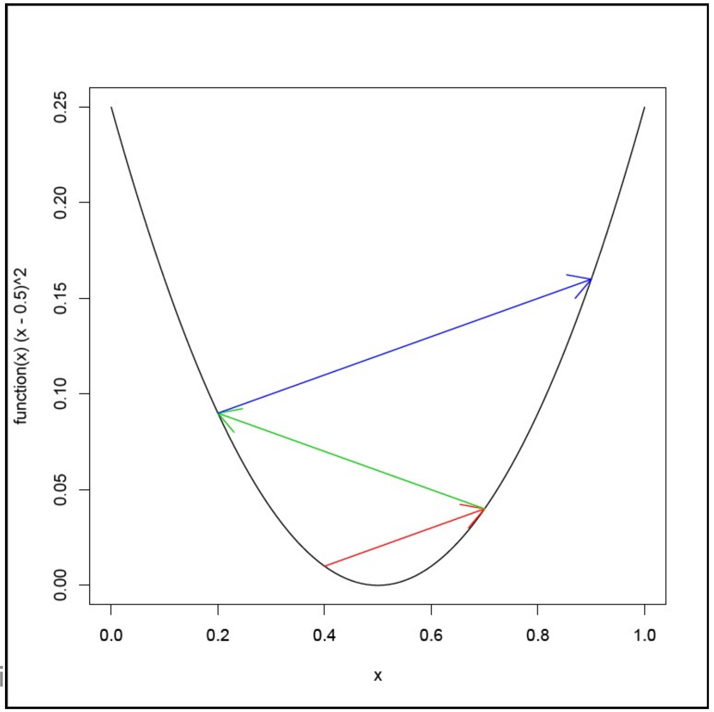
- Derivative keeps changing



Gradient descent algorithm – learning rate

- If η is too small, gradient descent can converge very slowly.
- If η is too large, gradient descent can overshoot the minimum, it may fail to converge, or even diverge.





Implementation – adaline class

```
def fit(self, X, y):
    rgen = np.random.RandomState(self.random_state)
    self.w_ = rgen.normal(loc=0.0, scale=0.01, size=1 + X.shape[1])
    self.cost_ = []

    for _ in range(self.n_iter):
        errors = 0
        for xi, target in zip(X, y):
            update = self.eta * (target - self.predict(xi))
            self.w_[1:] += update * xi
            self.w_[0] += update
            errors += int(update != 0.0)
        self.errors.append(errors)
    return self
```

```
net_input = self.net_input(X)
output = net_input
errors = (y - output)
self.w_[1:] += self.eta * X.T.dot(errors)
self.w_[0] += self.eta * errors.sum()
cost = (errors**2).sum() / 2.0
self.cost_.append(cost)
```

Explanation to the fit function

```
net_input = self.net_input(X)
output = net_input
errors = (y - output)
self.w_[1:] += self.eta * X.T.dot(errors)
self.w_[0] += self.eta * errors.sum()
cost = (errors**2).sum() / 2.0
self.cost_.append(cost)
```

net_input = $\mathbf{w}^{\intercal}\mathbf{x}^{(\mathbf{i})} + w_0$ (next slide)

errors: a length-*n* vector; The *i*-th value = $y^{(i)} - \phi(z^{(i)})$.

$errors.sum() = \sum_i (y^{(i)} - \phi(z^{(i)}))$, which is Δw_0.

$$(errors ** 2).sum()/2 = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2 = J(\mathbf{w})$$

Calculation of net input

```
def net_input(self, X):
        return np.dot(X, self.w_[1:]) + self.w_[0]

def predict(self, X):
        return np.where(net_input(X) >= 0.0, 1, -1)
```

net_input$_i$ = $\mathbf{w}^\intercal \mathbf{x}^{(\mathbf{i})} + w_0$

The return value of the net_input function

$$\begin{pmatrix} \mathbf{w}^\intercal \mathbf{x}^{(\mathbf{1})} + w_0 \\ \mathbf{w}^\intercal \mathbf{x}^{(\mathbf{2})} + w_0 \\ \dots \\ \mathbf{w}^\intercal \mathbf{x}^{(\mathbf{n})} + w_0 \end{pmatrix}$$

Update self.w_[1:]

- self.w_[1:] += self.eta * X.T.dot(errors)

$$X.T.dot(erros) = \begin{bmatrix} {x_1}^{(1)} & {x_1}^{(2)} & \dots & {x_1}^{(n)} \\ {x_2}^{(1)} & {x_2}^{(2)} & \dots & {x_2}^{(n)} \\ \dots & \dots & \dots & \dots \\ {x_m}^{(1)} & {x_m}^{(2)} & \dots & {x_m}^{(n)} \end{bmatrix} \cdot \begin{bmatrix} y^{(1)} - \phi(z)^{(1)} \\ y^{(2)} - \phi(z)^{(2)} \\ \dots \\ y^{(n)} - \phi(z)^{(n)} \end{bmatrix}$$

$$= \begin{bmatrix} {x_1}^{(1)}(y^{(1)} - \phi(z^{(1)})) + {x_1}^{(2)}(y^{(2)} - \phi(z^{(2)})) + & \dots & + {x_1}^{(n)}(y^{(n)} - \phi(z^{(n)})) \\ {x_2}^{(1)}(y^{(1)} - \phi(z^{(1)})) + {x_2}^{(2)}(y^{(2)} - \phi(z^{(2)})) + & \dots & + {x_2}^{(n)}(y^{(n)} - \phi(z^{(n)})) \\ & \dots & \\ {x_m}^{(1)}(y^{(1)} - \phi(z^{(1)})) + {x_m}^{(2)}(y^{(2)} - \phi(z^{(2)})) + & \dots & + {x_m}^{(n)}(y^{(n)} - \phi(z^{(n)})) \end{bmatrix}$$

Update self.w_[1:]: Δw_j

- Look at the j-th row

$$x_j^{(1)}(y^{(1)} - \phi(z^{(1)})) + x_j^{(2)}(y^{(2)} - \phi(z^{(2)})) + \ \dots + x_j^{(n)}(y^{(n)} - \phi(z^{(n)}))$$

$$= \sum_i x_j^{(i)}(y^{(i)} - \phi(z^{(i)}))$$

$$= \sum_i (y^{(i)} - \phi(z^{(i)}))\, x_j^{(i)}$$

$$= \Delta w_j / \eta$$

$$\Delta w_j = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right) x_j^{(i)}$$

$$X.T.dot(erros) = \begin{bmatrix} \Delta w_1/\eta \\ \Delta w_2/\eta \\ \dots \\ \Delta w_m/\eta \end{bmatrix}$$

Feature scaling

- Data range for different features are different.
- Normalization $x'_j = \dfrac{x_j - \mu_j}{\sigma_j}$
- Implementation

```
X_std = np.copy(X)
X_std[:, 0] = (X[:, 0] - X[:, 0].mean()) / X[:, 0].std()
X_std[:, 1] = (X[:, 1] - X[:, 1].mean()) / X[:, 1].std()
X_std[:, 2] = (X[:, 2] - X[:, 2].mean()) / X[:, 2].std()
```

Summary

- Adaline model
- Gradient descent algorithm
- Adaline implementation
- Difference between Adaline and perceptron