Gradient Descent Rule

- The objective is to minimize the following error:

$$E(\mathrm{w}) = \frac{1}{2}\sum_{e}(y_e - o_e)^2$$

- The training is a process of minimizing the error E(w) in the steepest direction (most rapid decrease), that is *in direction opposite to the gradient:*

$$\nabla E(w) = [\partial E / \partial w_0, \partial E / \partial w_1, \dots, \partial E / \partial w_d]$$

which leads to the **gradient descent training rule**:

$$\boxed{w_i = w_i - \eta \partial E / \partial w_i}$$

注：$o_e = w_i x_{ie}$

The weight update can be derived as follows:

$$\begin{aligned}
\partial E / \partial w &= \partial(\frac{1}{2}\sum_e (y_e - o_e)^2) / \partial w_i \\
&= \frac{1}{2}\sum_e \partial (y_e - o_e)^2 / \partial w_i \\
&= \frac{1}{2}\sum_e 2(y_e - o_e)\partial(y_e - o_e) / \partial w_i \\
&= \sum_e (y_e - o_e)\partial(y_e - w_i x_{ie}) / \partial w_i \\
&= \sum_e (y_e - o_e)(-x_{ie})
\end{aligned}$$

where x_{ie} denotes the *i-th* component of the example *e*.

The gradient descent training rule becomes:

$$\boxed{w_i = w_i + \eta \sum_e (y_e - o_e) x_{ie}}$$

Gradient Descent Learning Algorithm

Initialization: *Examples* $\{(x_e, y_e)\}_{e=1}^{N}$ *, initial weights* w_i *set to small random values, learning rate parameter* η

Repeat

for each training example (x_e*,* y_e *)*

- calculate the network output: $o_e = \sum_{i=0}^{d} w_i x_{ie}$

- if the Perceptron does not respond correctly, compute weight corrections:

$$\boxed{\Delta w_i = \Delta w_i + \eta (y_e - o_e) x_{ie}}$$

update the weights with the accumulated error from all examples

$$\boxed{w_i = w_i + \Delta w_i}$$

Gradient Descent Rule

until *termination condition is satisfied.*

注意：这里 Δw 需要积累，即要把所有 training examples 的都考虑进去。这是 gradient descent 的特点，而之后的 increasement gradient descent 就不用。

Example:

- Suppose an example of Perceptron which accepts two inputs x_1 and x_2, with weights w_1 = 0.5 and w_2 = 0.3 and w_0 = -1, learning rate = 1.
- Let the example is given: $x_1 = 2$, $x_2 = 1$, $y = 0$
  The network output of the Perceptron is :

$$o = 2 * 0.5 + 1 * 0.3 - 1 = 0.3$$

- The weight updates according to the **gradient descent algorithm** will be:

$$\Delta w_1 = (0 - 0.3) * 2 = - 0.6$$
$$\Delta w_2 = (0 - 0.3) * 1 = - 0.3$$
$$\Delta w_0 = (0 - 0.3) * 1 = - 0.3$$

注：x_0 恒为 1

- Let another example is given: $x_1 = 1$, $x_2 = 2$, $y = 1$
- The network output of the Perceptron is :

$$o = 1 * 0.5 + 2 * 0.3 - 1 = 0.1$$

  The weight updates according to the gradient descent algorithm will be:

$$\Delta w_1 = - 0.6 + (1 - 0.1) * 1 = 0.3$$
$$\Delta w_2 = - 0.3 + (1 - 0.1) * 2 = 1.5$$
$$\Delta w_0 = - 0.3 + (1 - 0.1) * 1 = 0.6$$

  If there are no more examples, the weights will be modified as follows:

$$w_1 = 0.5 + 0.3 = 0.8$$
$$w_2 = 0.3 + 1.5 = 1.8$$
$$w_0 = - 1 + 0.6 = 1.6$$

注意，上面对 Δw 进行了计算（蓝字），先把所有的 Δw 都考虑到，最后再根据最终的 Δw 进行权重更新。

Incremental gradient descent

梯度下降规则在实践中面临两个困难：

- 它收敛非常缓慢
- 如果在 error surface 有多个 local minima，那么不能保证它会找到 global minimum

因此，为了克服这些困难，开发了一种 **stochastic version** 名为 **incremental gradient descent rule**。

gradient descent rule 在计算所有 training examples 中累积的错误后更新权重；而 incremental gradient descent rule 通过在每个 training example 后更新权重来使梯度下降的误差减少。

Incremental gradient descent is implemented

$$w_i = w_i + \eta(y_e - o_e)x_{ie} \qquad where \qquad o_e = \sum_{i=0}^{d} w_i x_{ie}$$

Incremental Gradient Descent Learning Algorithm

Initialization: *Examples* $\{(x_e, y_e)\}_{e=1}^{N}$, *initial weights* w_i *set to small random values, learning rate parameter* η

Repeat

for each training example (x_e, y_e)

- *calculate the network output:*

$$o_e = \sum_{i=0}^{d} w_i x_{ie}$$

- *if the Perceptron does not respond correctly update the weights:*

$$w_i = w_i + \eta\ (y_e - o_e)\ x_{ie}$$

until *termination condition is satisfied.*

注：这里就没有对 Δw 进行累积，因为 incremental gradient descent 只考虑单个点。

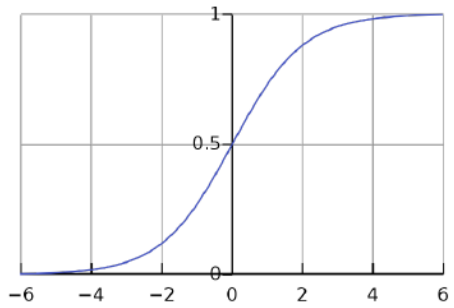
Sigmoidal Perceptrons

The sigmoidal Perceptron produces output:

$$o = \sigma(S) = \frac{1}{1+e^{-S}},$$

where:

$$S = \sum_{i=0}^{d} w_i x_x$$



- The gradient descent rule for training sigmoidal Perceptrons is again:

$$\boxed{w_i = w_i - \eta \partial E / \partial w_i}$$

- The difference is in the error derivative $\partial E / \partial w_i$ which due to the use of the sigmoidal function σ(s) becomes:

$$\partial E/\partial w_i = \partial ((½)\Sigma_e (y_e - o_e)^2)/\partial w_i$$

$$= (½)\Sigma_e \partial (y_e - o_e)^2/\partial w_i$$

$$= (½)\Sigma_e 2 (y_e - o_e)\partial (y_e - o_e)/\partial w_i$$

$$= \Sigma_e (y_e - o_e)\partial (y_e - \sigma(s))/\partial w_i$$

$$= \Sigma_e (y_e - o_e)\, \sigma'(s)\, (-x_{ie})$$

where x_{ie} *denotes the i-th component of the example*

- The *Gradient descent training rule* for training sigmoidal Perceptrons is:

$$w_i = w_i + \eta \sum_e (y_e - o_e)\sigma'(S)x_{ie}$$

where:

$$\sigma'(S) = \sigma(S)(1 - \sigma(S))$$

Gradient Descent Learning Algorithm for Sigmoidal Perceptrons

- *Initialization: Examples $\{(x_e, y_e)\}_{e=1}^N$, initial weights w_i set to small random values, learning rate parameter η*
- *Repeat*

  for each training example (x_e, y_e)

  - calculate the network output: $o = \sigma(s)$ *where* $s=\sum_{i=0}^{d} w_i x_{ie}$

  - if the Perceptron does not respond correctly compute weight corrections:

  $$\Delta w_i = \Delta w_i + \eta(y_e - o_e)\sigma(s)(1 - \sigma(s))x_{ie}$$

  update the weights with the accumulated error from all examples $w_i = w_i + \Delta w_i$

  until termination condition is satisfied. 22

Example:

- Suppose an example of Perceptron which accepts two inputs x_1 and x_2, with weights $w_1 = 0.5$ and $w_2 = 0.3$ and $w_0 = -1$, learning rate = 1.
- Let the following example is given: $x_1 = 2$, $x_2 = 1$, $y = 0$

  The output of the Perceptron is :

$$O = \sigma(-1+2 * 0.5 + 1 * 0.3) = \sigma (0.3)=0.5744$$

- The weight updates according to the gradient descent algorithm will be:

$$\Delta w_0 = (0 - 0.5744) * 0.5744 * (1 -0.5744) * 1 = - 0.1404$$
$$\Delta w_1 = (0 - 0.5744) * 0.5744 * (1 -0.5744) * 2 = - 0.2808$$
$$\Delta w_2 = (0 - 0.5744) * 0.5744 * (1 -0.5744) * 1 = - 0.1404$$

Let another example is given: $x_1 = 1$, $x_2 = 2$, $y = 1$

The output of the Perceptron is :

$$O = \sigma(-1+1 * 0.5 + 2 * 0.3) = \sigma (0.1)=0.525$$

The weight updates according to the gradient descent algorithm will be:

$$\Delta w_0 = - 0.1404 + (1 - 0.525) * 0.525 * (1 - 0.525) * 1 = -0.0219$$
$$\Delta w_1 = - 0.2808 + (1 - 0.525) * 0.525 * (1 - 0.525) * 1 = -0.1623$$
$$\Delta w_2 = - 0.1404 + (1 - 0.525) * 0.525 * (1 - 0.525) * 2 = 0.0966$$

If there are no more examples in the batch, the weights will be modified as follows:

$$w_0 = - 1 + (-0.0219) = -1.0219$$
$$w_1 = 0.5 + (-0.1623) = 0.3966$$
$$w_2 = 0.3 + 0.0966 = 0.3966$$

Perceptron vs. Gradient Descent

Gradient descent finds the decision boundary which minimizes the **sum squared error** of the (target -net) value rather than the (target -output) value.

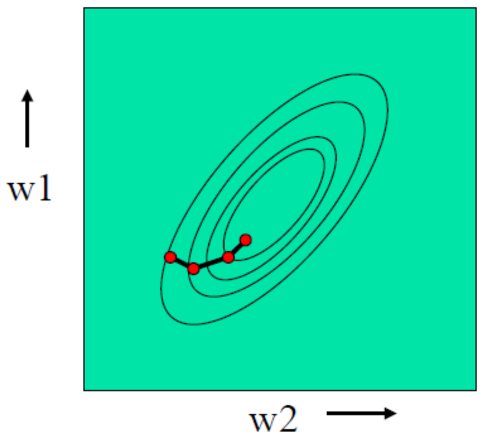
- Perceptron rule will find the decision boundary which minimizes the classification error – if the problem is linearly separable

- Gradient descent decision boundary may leave more instances misclassified as compared to the perceptron rule: could have a higher misclassification rate than with the perceptron rule

Perceptron rule (target - thresholded output) guaranteed to converge to a separating hyperplane if the problem is linearly separable.

Batch vs incremental learning

- Batch learning does steepest descent on the error surface



- Incremental learning zig-zags around the direction of steepest descent