How do you derive the Gradient Descent rule for Linear Regression and Adaline?

Linear Regression and Adaptive Linear Neurons (Adalines) are closely related to each other. In fact, the Adaline algorithm is a identical to linear regression except for a threshold function $\phi(\cdot)_T$ that converts the continuous output into a categorical class label

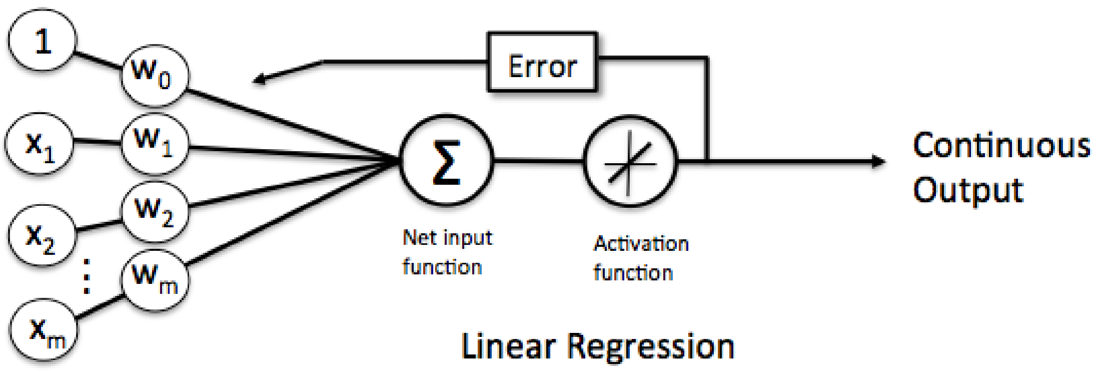
$$\phi(z)_T = \begin{cases} 1 & if \; z \geq 0 \\ 0 & if \; z < 0 \end{cases},$$

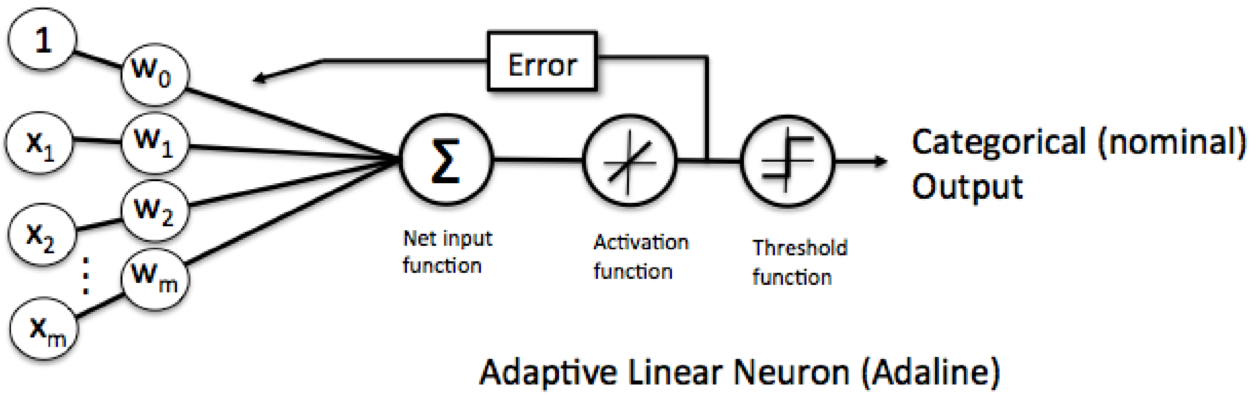
where z is the net input, which is computed as the sum of the input features **x** multiplied by the model weights **w**:

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

(Note that x_0 refers to the bias unit so that $x_0 = 1$.)

In the case of linear regression and Adaline, the activation function $\phi(\cdot)_A$ is simply the identity function so that $\phi(z)_A = z$.





Now, in order to learn the optimal model weights **w**, we need to define a cost function that we can optimize. Here, our cost function $J(\cdot)$ is the sum of squared errors (SSE), which we multiply by $\frac{1}{2}$ to make the derivation easier:

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

where $y^{(i)}$ is the label or target label of the *i*th training point $x^{(i)}$.

(Note that the SSE cost function is convex and therefore differentiable.)

In simple words, we can summarize the gradient descent learning as follows:

1. Initialize the weights to 0 or small random numbers.
2. For *k* epochs (passes over the training set)
    1. For each training sample $x^{(i)}$
        - Compute the predicted output value $\hat{y}^{(i)}$
        - Compare $\hat{y}^{(i)}$ to the actual output $y^{(i)}$ and Compute the “weight update” value
        - Update the “weight update” value
    2. Update the weight coefficients by the accumulated “weight update” values

Which we can translate into a more mathematical notation:

1. Initialize the weights to 0 or small random numbers.
2. For *k* epochs
    1. For each training sample $x^{(i)}$
        - $\phi(z^{(i)})_A = \hat{y}^{(i)}$
        - $\Delta w_{(t+1),\, j} = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$ (where *η* is the learning rate);
        - $\Delta w_j := \Delta w_j + \Delta w_{(t+1),\, j}$
    2. $\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$

Performing this global weight update

$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}$,

can be understood as "updating the model weights by taking an opposite step towards the cost gradient scaled by the learning rate *η*"

$$\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}),$$

where the partial derivative with respect to each w_j can be written as

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

To summarize: in order to use gradient descent to learn the model coefficients, we simply update the weights **w** by taking a step into the opposite direction of the gradient for each pass over the training set – that's basically it. But how do we get to the equation

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

Let's walk through the derivation step by step.

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