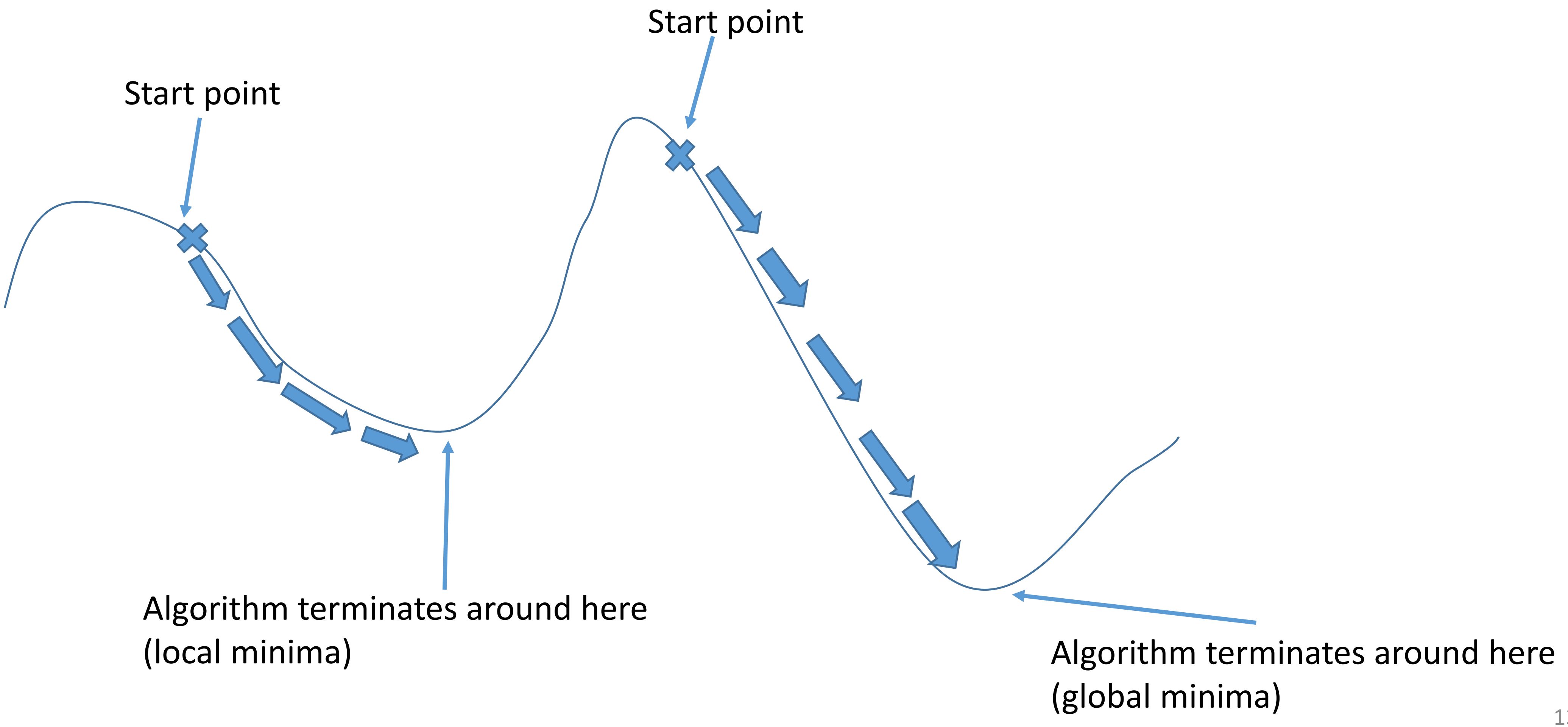
Gradient decent – Convergence is to local optima



Gradient decent – multivariate situation

- In practice our NN will have multiple weights – so the error will be in terms of multiple variables.
- This means we will be taking partial derivatives with respect to each weight.
- Multivariate gradient decent:
- $w_{j+1} = w_j - \alpha \frac{\partial E\ (w_1, w_2, \ldots, w_N)}{\partial w_j}$ where E is the error function in terms of the N weights w_j. The derivative is wrt to the single weight w_j which is to be adjusted

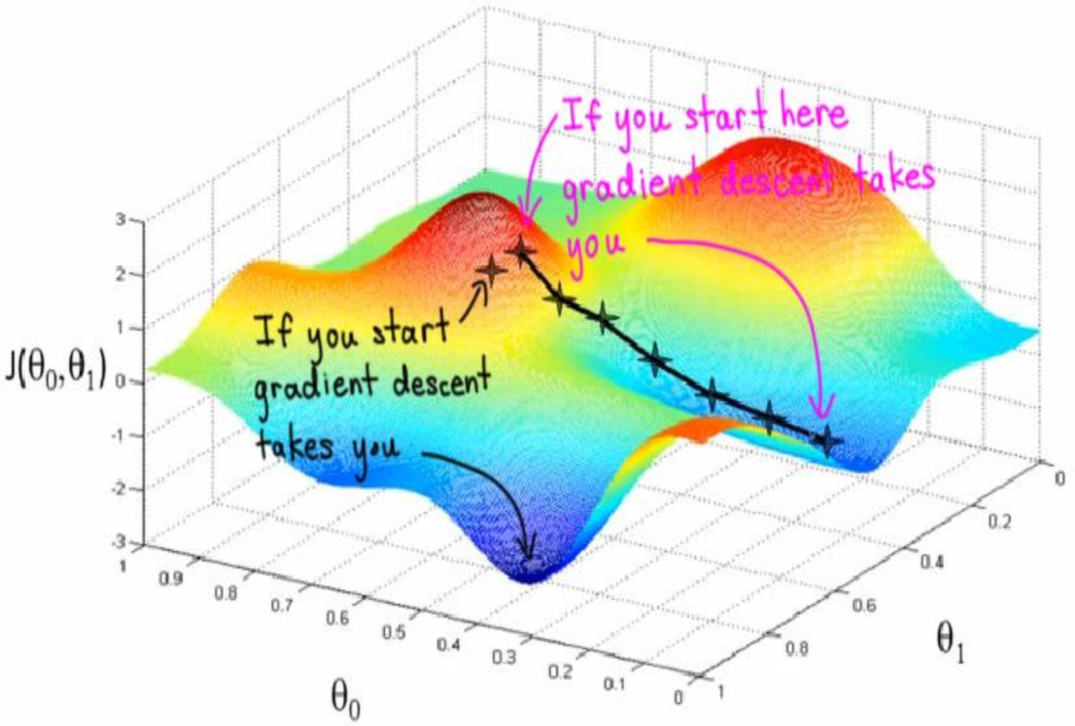


Image by quinnliu.
See [1] for full citation.

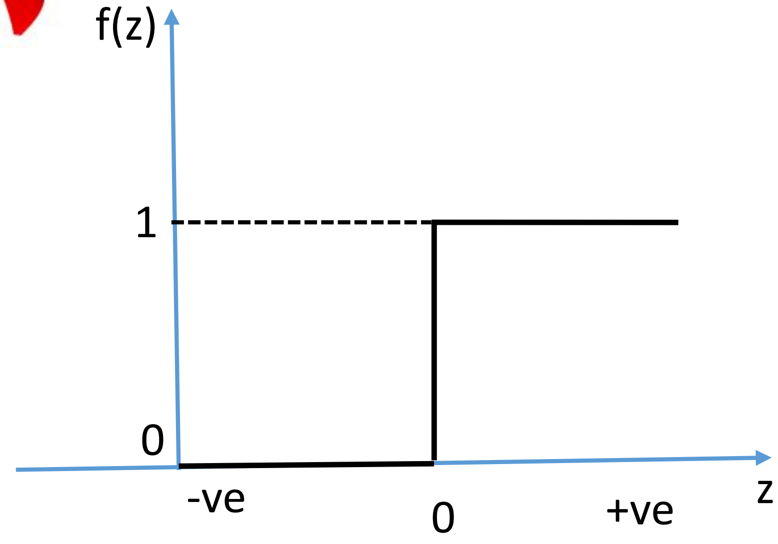
Gradient decent – Implementation Options

- Problem defn: We have M feature vectors each of which has N components.
- Batch Mode: After inputting all M feature vectors, compute the gradient with respect to each of the N weights, and update each weight accordingly. Input the M feature vectors again, recompute weights, and repeat the process until convergence
- Online Mode/Stochastic Gradient Decent: After inputting a single feature vector, compute the gradient with respect to each of the N weights and update each weight accordingly. Repeat the process for each input feature vector until all M vectors have been used. Before the next epoch, randomly shuffle the vectors, and repeat the process until convergence

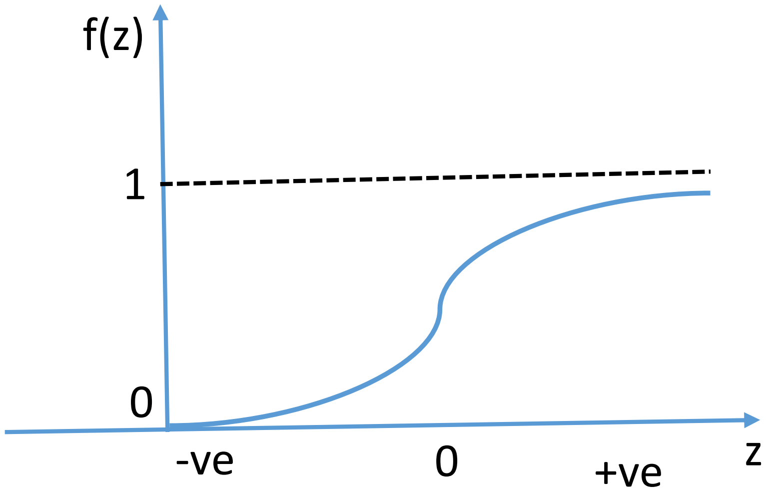
Gradient Decent – more requirements

- Example Activation functions

Step function



Sigmoid function



Preliminaries – Sigmoid derivative

- Let the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}} = y$
- $Let\ z = e^{-x} \Rightarrow y = \frac{1}{1+z} = (1+z)^{-1}$

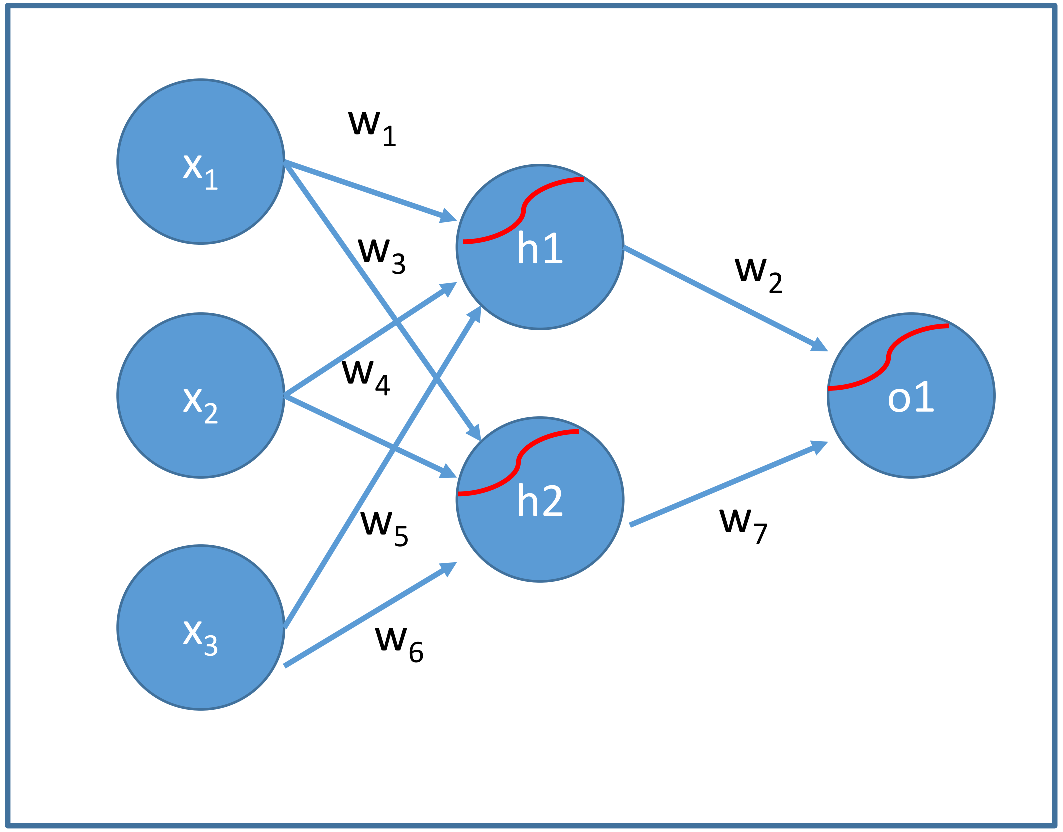
$$\frac{dy}{dx} = \frac{dy}{dz}.\frac{dz}{dx} = -(1+z)^{-2}.-e^{-x} = e^{-x}(1+e^{-x})^{-2} = \frac{e^{-x}}{(1+e^{-x})^2}$$

Rearrangement gives: $\frac{dy}{dx} = \frac{1}{(1+e^{-x})}.\frac{e^{-x}}{(1+e^{-x})} = \frac{1}{(1+e^{-x})}.\frac{e^{-x}+1-1}{(1+e^{-x})}$

$$= \frac{1}{(1+e^{-x})}\left[\frac{e^{-x}+1}{(1+e^{-x})} - \frac{1}{(1+e^{-x})}\right] = \frac{1}{(1+e^{-x})}\left[1 - \frac{1}{(1+e^{-x})}\right]$$

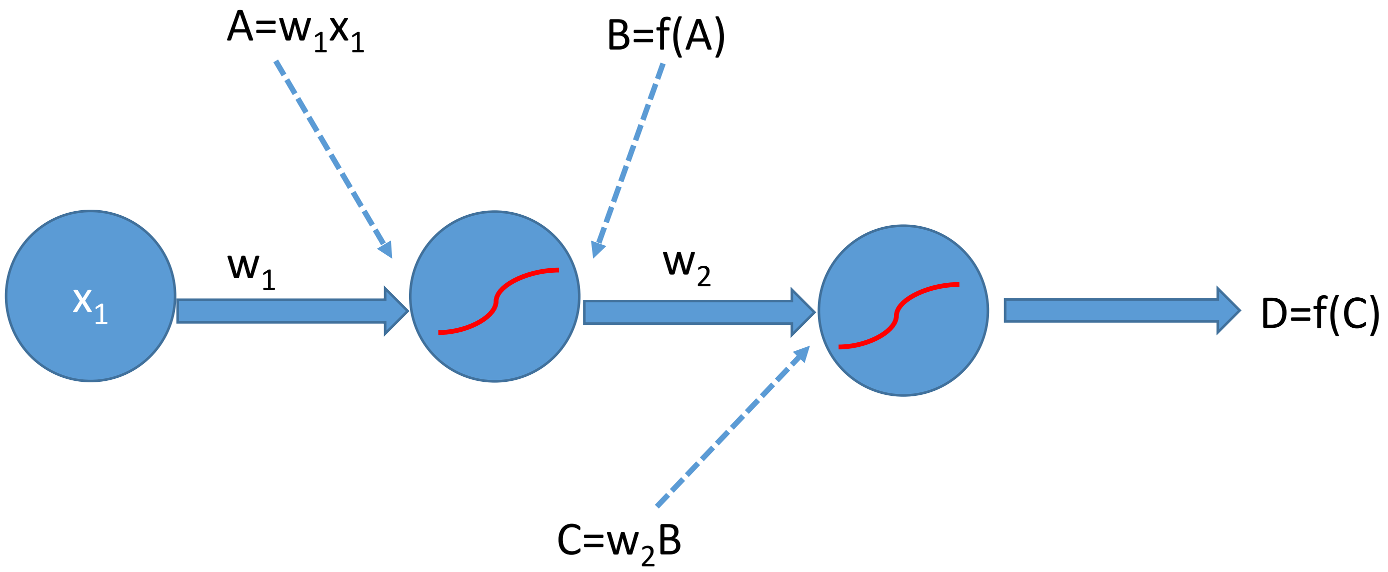
$$\Rightarrow \boldsymbol{\sigma'(x) = \sigma(x)[1-\sigma(x)]}$$

Multi-layer perceptron



Feed-Forward Multilayer Perceptron

Gradient Decent + Chain Rule = Back Propagation!



f= Activation function, e.g., Sigmoid
D=Final output
f(A)≡sigmoid(A)
f(C)≡sigmoid(C)

Back Propagation

- Error, $E = \frac{1}{2}(t_p - y_p)^2$.
- $t_p = Expected\ ouput;\ y_p = Output\ obtained\ from\ network$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial D} . \frac{\partial D}{\partial C} . \frac{\partial C}{\partial w_2}$$

Error can be rewritten as $E = \frac{1}{2}(t_p - D)^2 \Rightarrow \frac{\partial E}{\partial D} = -1.\left(t_p - D\right) = D - t_p$.

Assuming a sigmoid squashing function, we can recall from our sigmoid derivative that if D=Sigmoid(C), then, $\frac{\partial D}{\partial C} = Sigmoid(C)[1 - Sigmoid(C)]$

$$\Rightarrow \frac{\partial D}{\partial C} = \sigma(C)[1 - \sigma(C)] = D(1 - D)$$

$$Lastly\ \frac{\partial C}{\partial w_2} = B$$

Back Propagation

$$\Rightarrow \frac{\partial E}{\partial w_2} = \left(D - t_p\right) * D(1 - D) * \mathrm{B}$$

$$\Rightarrow \boldsymbol{w_2^* = w_2 - \alpha \frac{\partial E}{\partial w_2}}$$

Back Propagation

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial D} \cdot \frac{\partial D}{\partial C} \cdot \frac{\partial C}{\partial B} \cdot \frac{\partial B}{\partial A} \cdot \frac{\partial A}{\partial w_1}$$

$$\frac{\partial A}{\partial w_1} = x_1$$

$$\frac{\partial B}{\partial A} = \boldsymbol{\sigma(A)[1 - \sigma(A)] = B(1 - B)}$$

$$\frac{\partial C}{\partial B} = w_2$$

$$\Rightarrow \boldsymbol{\frac{\partial E}{\partial w_1} = \left(D - t_p\right).D(1 - D).w_2.B(1 - B).x_1}$$

$$\boldsymbol{w_1^* = w_1 - \alpha \frac{\partial E}{\partial w_1}}$$

Refrences

- **[1]** https://github.com/quinnliu/machineLearning/tree/master/supervisedLearning/linearRegressionInMultipleVariables