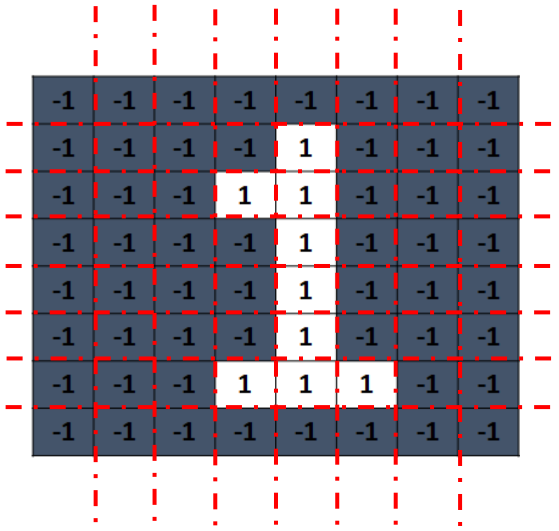
Deep Learning (CS 470, CS 570)

Module 3, Lecture 2: Gradient Descent, Backpropagation

Example: How ANN Works

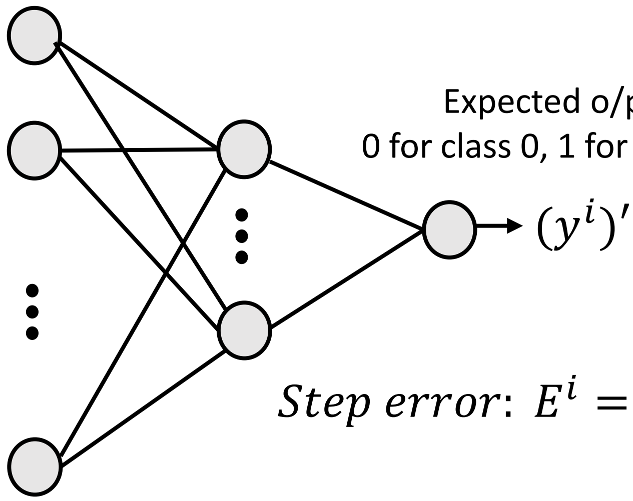
$\boldsymbol{For\ i = 1, N}$ $(N\ is\ number\ of\ training\ samples)$



-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

Resolution: $m \times n$

$$\xrightarrow{\text{Feature vect.}} \boldsymbol{x}^i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m\times n} \end{bmatrix}^i$$



$$Step\ error:\ E^i = Er\left(y^i - (y^i)'\right)$$

As mentioned before, for each training sample, a feature vector is formed and passed to the MLP. The network connection weights are initialized randomly and during the training process network learn the weights so that it can predict the output classes of the feature vectors. Every time the MLP produces an output, the error is calculate as the absolute difference of the expected and predicted outputs. The error is a function of the weight values. The objective of the training is to minimize the expected error for any sample.

$$Total\ error:\ E = \sum_i E^i$$

$E\ depends\ on\ \boldsymbol{x}^i s\ and\ \boldsymbol{w}\ but\ training\ set\ \boldsymbol{x}^i s$ are constant

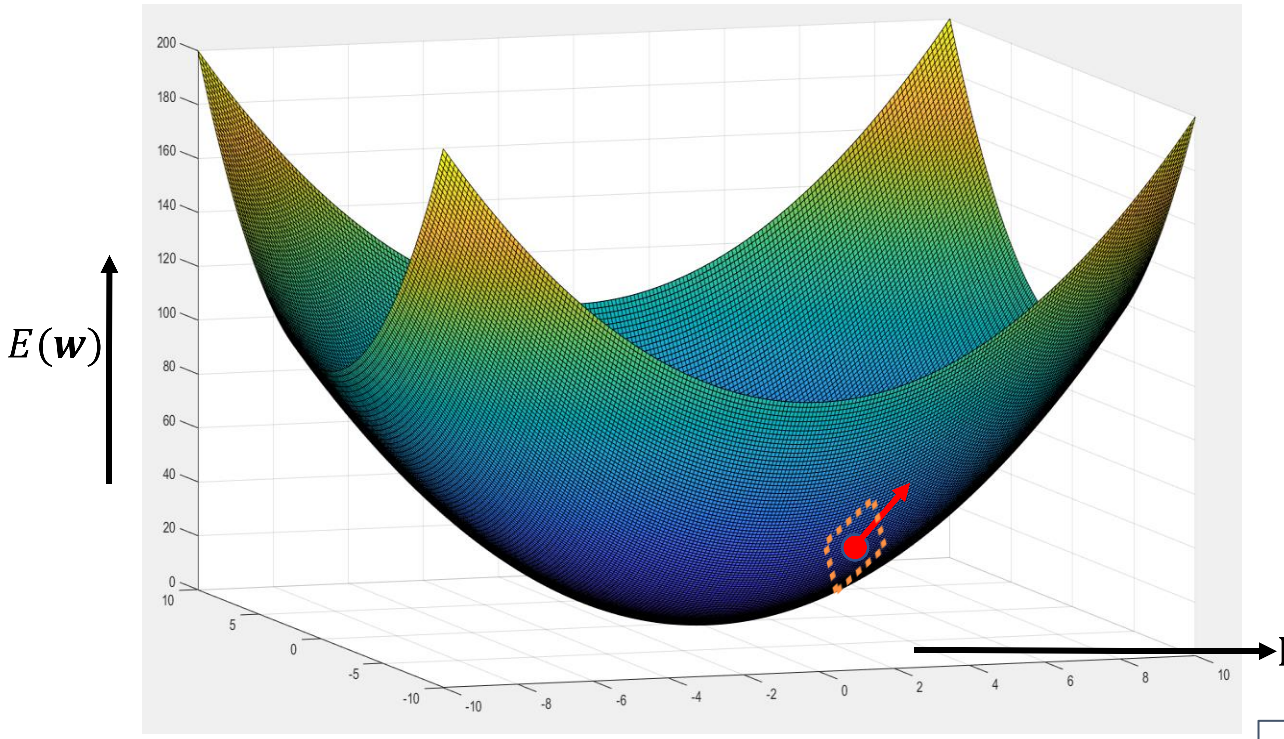
$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{N}} (y^i - h(\boldsymbol{x}^i))^2 \quad , \qquad h(\boldsymbol{x}^i)\ is\ a\ complex\ function\ of\ \boldsymbol{w}$$

Optimization: Gradient Descent

Gradient: $\nabla f(x,y) = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\rangle$

Magnitude: $\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

Direction: $\tan^{-1}\left(\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right)$



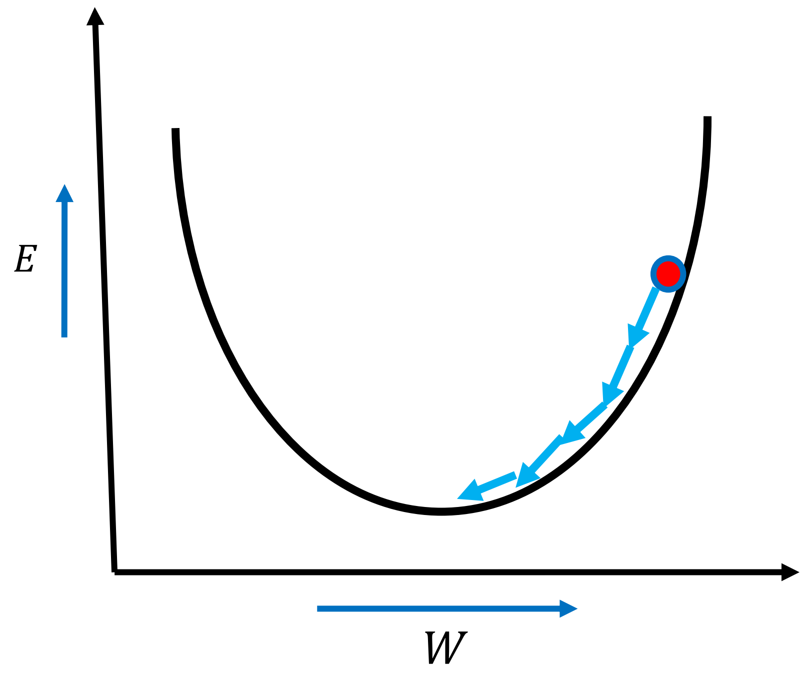
High dimensional $\boldsymbol{w}$ space

- Small jump along the gradient direction: increase of E
- Small jump opposite to gradient direction: decrease of E

→	Gradient direction
●	$\mathrm{E}(\mathbf{w}^t)$ is the error at a particular weight value $\mathbf{w}^t$
⬚	A very small region on the error surface that can be approximated as a plane

As mentioned the objective of training is to minimize the expected error. This is achieved through gradient descent optimization technique.
For most of the error functions, shape of the error surface w.r.t. **w** is unknown. Therefore, starting with an initial weight vector (**w**) the slope of the function needs to be calculated locally and values of **w** need to be changed such that the function moves to the direction of highest descent of error function E()

Optimization: Gradient Descent



Error function, where y^i is the targeted output (ground truth)

$$E = \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{N}}(y^i - h(\boldsymbol{x}^i))^2$$

Gradient descent algorithm:

- ➢ Update weight in the opposite direction of the gradient (i.e. descent)
- ➢ $W^{t+1} = W^t - \alpha\Delta E(W)$,where $\Delta E(W) = \left\langle \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_d} \right\rangle$
- ➢ Iteratively, follow the direction of highest slope

Variation of Gradient descent: batch gradient descent, stochastic gradient descent, mini-batch gradient descent

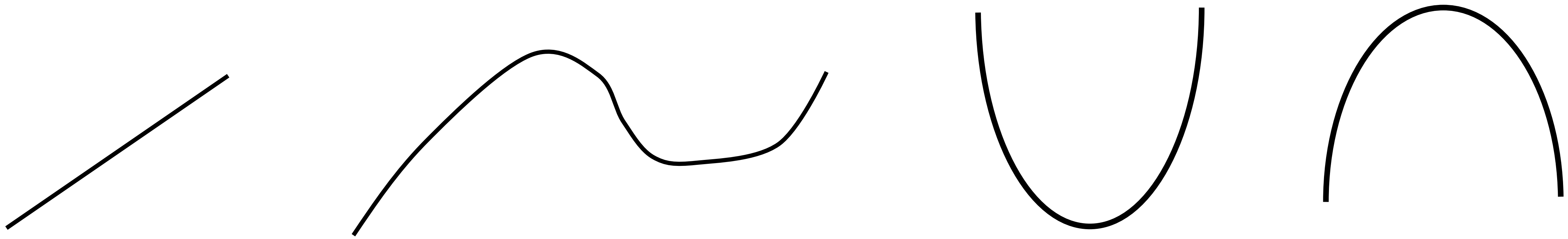
Check this article for details.

Convex Function

A convex function is a continuous function whose value at the midpoint of every interval in its domain does not exceed the arithmetic mean of its values at the ends of the interval.

A function $f(x)$ is convex on an interval $[a, b]$ if for any two points x_1 and x_2 in $[a, b]$ and any λ where $0 < \lambda < 1$

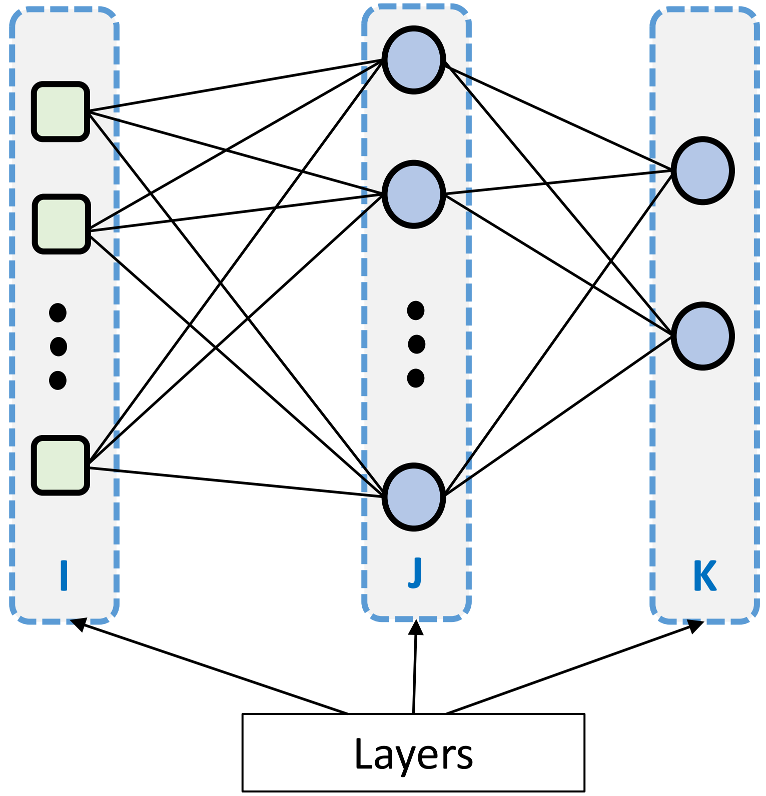
$$\boxed{f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda) f(x_2)}$$



Identify the convex functions

For a convex function, gradient descent is guaranteed to converge on a global minimum.

Optimization: Back-propagation Algorithm



Notations:

$w_{i,j}^{l}$	Weight of connection between node j of layer l and node i of layer l-1
t_k	Target output of node k of the output layer
y_k^l	Output of node k of layer l
z_k^l	Input of node k of layer l
θ_k^l	Bias of node k of layer l

Prerequisites:

Input of node k of layer l is the linear combination of outputs of previous layer

$$z_k^l = \sum_{j \in l-1} w_{j,k}^l y_j^{l-1} \qquad \textcircled{1}$$

Derivative of sigmoid function

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right)$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = \sigma(z)(1-\sigma(z)) \qquad \textcircled{2}$$

Expression of output of node k of layer l

$$y_k^l = \sigma\left(\sum_{j \in l-1} w_{j,k}^l y_j^{l-1}\right) = \sigma(z_k^l) \qquad \textcircled{3}$$

Optimization: Back-propagation Algorithm

Given a training data point with target values and output values at node k as t_k and y_k^K respectively, the error function can be defined as:

$$E = \frac{1}{2}\sum_{\kappa \in K} (y_\kappa^K - t_\kappa)^2$$

Note: we are assuming stochastic gradient descent for the time being hence E is calculated for each data point. We are omitting the index of the training points for the simplicity of the notation

We consider two cases, **1)** a node in the output layer and **2)** a node in the hidden layer

1) k_{th} node in the output layer

We calculate derivative of error with respect to weight w_{jk}^K

$$\frac{\partial E}{\partial w_{jk}^K} = \frac{\partial}{\partial w_{jk}^K}\left(\frac{1}{2}\sum_{\kappa \in K}(y_\kappa^K - t_\kappa)^2\right)$$

$$= \sum_{\kappa \in K}(y_\kappa^K - t_\kappa)\frac{\partial}{\partial w_{jk}^K} y_\kappa^K = (y_k^K - t_k)\frac{\partial}{\partial w_{jk}^K} y_k^K = (y_k^K - t_k)\frac{\partial}{\partial w_{jk}^K}\sigma(z_k^K) \tag{4}$$

As only y_k^K is dependent on w_{jk}^K, summation can be removed

Using 3

$$= (y_k^K - t_k)\sigma(z_k^K)(1-\sigma(z_k^K))\frac{\partial}{\partial w_{jk}^K} z_k^K = (y_k^K - t_k)y_k^K(1-y_k^K)y_j^J \tag{5}$$

Using 2, and applying chain rule of derivative

Using 1, and taking derivative. y_j^J Only dependent on w_{jk}^K

This completes the derivation of error with respect to the weight term. For the compactness of notation, we represent it as:

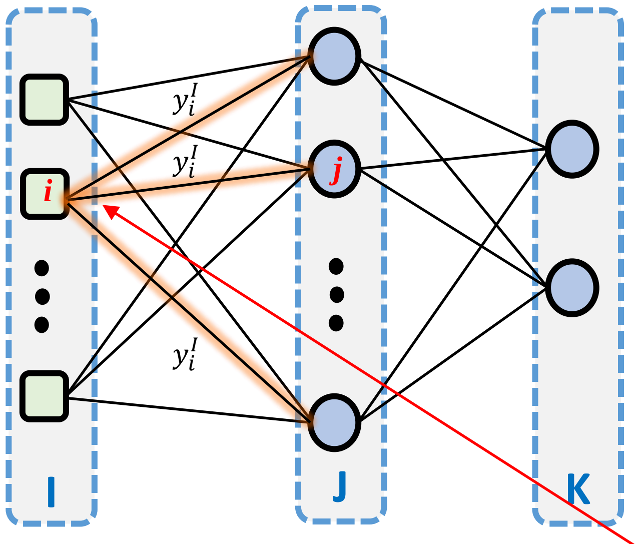
$$\frac{\partial E}{\partial w_{jk}^K} = y_j^J \delta_k$$

where, $\delta_k = (y_k^K - t_k)y_k^K(1-y_k^K)$

Optimization: Back-propagation Algorithm

2) j_{th} node node in any hidden layer J

First approach: compute gradient of error E with respect to a weight w_{ij}^J when the gradient in the next layer 'K' is already computed.



$$\frac{\partial E}{\partial w_{ij}^J} = \frac{\partial z_j^J}{\partial w_{ij}^J}\frac{\partial E}{\partial z_j^J} = y_i^I \frac{\partial y_j^J}{\partial z_j^J}\frac{\partial E}{\partial y_j^J} = y_i^I y_j^J(1-y_j^J)\boxed{\frac{\partial E}{\partial y_j^J}} \qquad (6)$$

This should come from the calculation of layer K

Using (1), and taking derivative. y_i^I Only dependent on w_{ij}^J

Using (2)

y_i^I Affects inputs of all the nodes in layer J hence the summation is needed

$$\boxed{\frac{\partial E}{\partial y_i^I}} = \sum_{j\in J}\frac{\partial E}{\partial z_j^J}\frac{dz_j^J}{\partial y_i^I} = \sum_{j\in J} w_{ij}^J \frac{\partial E}{\partial z_j^J} = \sum_{j\in J} w_{ij}^J y_j^J(1-y_j^J)\frac{\partial E}{\partial y_j^J} \qquad (7)$$

This should be forwarded to the calculation of layer I

Expanded similar to (6)

Thus the whole calculation can be done using dynamic programming starting the calculation from the output layer and moving towards the inner layers. In each layer, calculation from previous layer can be used for gradient estimation and same can be passed to the next layer.

Additional Readings

Backpropagation:

- Intuitive explanations (video)
- Mathematical explanation (video)
- Article