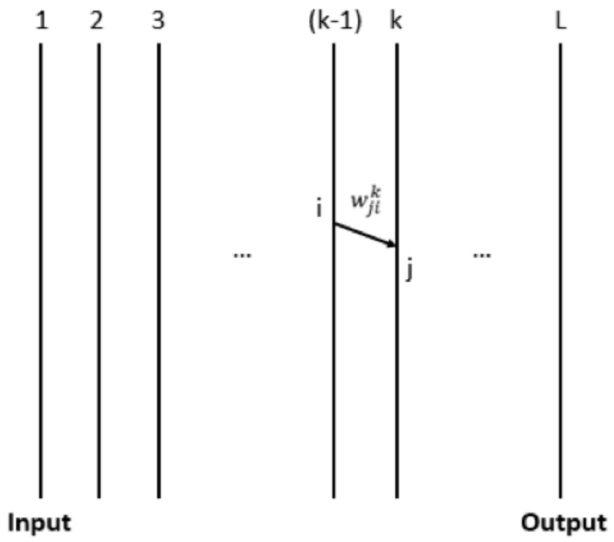
CSE 4502/5717 **with Dr. Sanguthevar Rajasekaran**

Lecture #18 4/04/2018 **notes from Chenyu Zhang**

A General Neural Network:



Let n_k be the number of neurons in level k, $1 \leq k \leq L$.

There could be a connection from every node in level $(k-1)$ to every node in level k, $1 \leq k \leq L$.

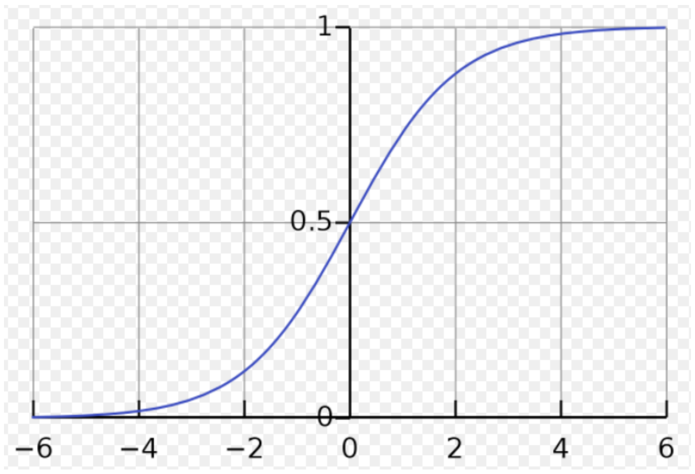
Let the weight of the edge from node *i* of level $(k-1)$ to node *j* of level *k* be denote as w_{ji}^k.

Let the output of node *j* in level *k* be denoted as a_j^k. $a_j^k = \sigma\left(\sum_{i=1}^{n_{l-1}} w_{ji}^k a_i^{k-1} + b_j^k\right)$, where

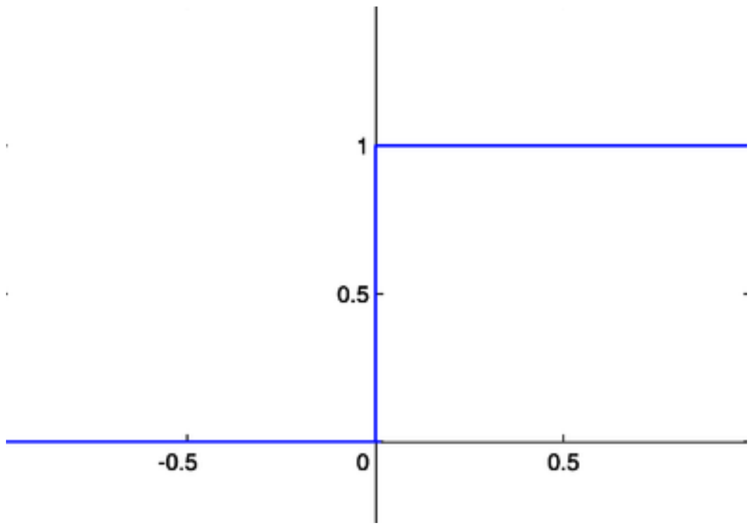
$\sigma \rightarrow Activation\ Function$. Let $z_j^k = \sum_{i=1}^{n_{l-1}} w_{ji}^k a_i^{k-1} + b_j^k$. z_j^k is the weighted input for the node *j* in level *k*.

<u>Possible Activation Functions:</u>

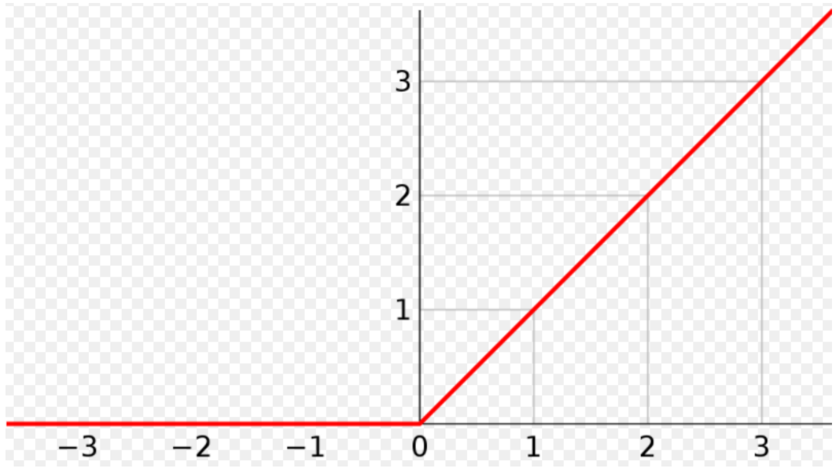
<u>1.Sigmoid Function:</u> $\sigma(x) = \frac{1}{1+e^{-x}}$



The sigmoid function can be thought of as an approximation to the step function :



2.Rectilinear Function: $\sigma(x) = \max\{0, x\}$



3. Softmax Function:

$$\boldsymbol{softmax(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}, where\ z\ is\ a\ vector.}$$

We compute optimal values for the network parameters using gradient descent.

Let *C* be any cost function on $x_1, x_2, x_3, \dots x_N$. If we change the parameter values by $\Delta x_1, \Delta x_2, \dots\ \Delta x_N$, the change in $C = \Delta C = \frac{\partial C}{\partial x_1}\Delta x_1 + \frac{\partial C}{\partial x_2}\Delta x_2 + \cdots \frac{\partial C}{\partial x_N}\Delta x_N$. Let $\boldsymbol{x} = (x_1, x_2, .. x_N)^{\mathrm{T}}$.

$$\nabla_{\boldsymbol{x}} C = \begin{bmatrix} \frac{\partial C}{\partial x_1} \\ \vdots \\ \frac{\partial C}{\partial x_N} \end{bmatrix}, \Delta C = (\Delta x_1\ \Delta x_2\ \dots\ \Delta x_N)\,.\nabla_{\boldsymbol{x}} C$$

If $\Delta\boldsymbol{x} = -\alpha\nabla_{\boldsymbol{x}} C$, then $\Delta C = -\alpha(\nabla_{\boldsymbol{x}} C)^{\mathrm{T}}(\nabla_{\boldsymbol{x}} C) = -\alpha\|\nabla_{\boldsymbol{x}} C\|_2^2$ which will always be negative. The idea of gradient descent is to choose this value for $\Delta\boldsymbol{x}$.

We can think of an iterative algorithm for minimizing *C*:

1. Start with some initial value for $\boldsymbol{x}$; Let this value be $\boldsymbol{x_0}$;
2. Compute $C(x_0)$ and $\nabla_{\boldsymbol{x}} C$, let $\boldsymbol{x_1} = \boldsymbol{x_0} - \alpha\nabla_{\boldsymbol{x}} C$
3. Repeat step 2 until the gradient becomes zero or close to zero.
   α is called the $learning\ rate$;

In the case of a Neural Network, let $w_1, w_2, \dots, w_N$ be the list of parameters, *C* could be the MSE.

We'll use gradient decent to update each parameter, i.e., $w_i = w_i - \alpha\frac{\partial C}{\partial w_i}, \forall\ i$.

Let C_x be the cost for example *x*, we can compute *C* as $\frac{1}{m}\sum_x C_x$. We can also compute ∇C as $\frac{1}{m}\sum_x \nabla C_x$.

If *m* is large, it will take a very long time before the parameters are updated. An alternative is to use Stochastic Gradient Descent. We pick a random sample *S* of examples & the gradient can be computed using the sample. Once the gradient is computed, we can update the parameters.

We'll compute *C* as $\frac{1}{|S|}\sum_{x\in S} C_x$ and $\nabla_x C$ as $\frac{1}{|S|}\sum_{x\in S} \nabla C_x$. Make updates using these values. We'll repeat this for other samples from the input.

Each subset is a mini batch.

When we end up using all the examples in the input once, we have completed an epoch. In fact, we can, in any epoch, shuffle the input examples randomly and partition the input into several mini batches. In the epoch, we will process each of the mini batches exactly once.

We may repeat the epochs.

When $|S| = 1$, we call it as online or incremental learning.

Training:

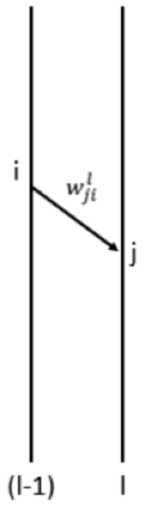
An Epoch:

For every Mini Batch do:

Compute the activation values for each node in each layer starting from the input layer and ending at the output. This is one forward propagation. Followed by this, compute $\frac{\partial C}{\partial w_i}$ for every parameter w_i.

Compute ∇C & C, and update the parameter values.

Forward propagation:



Assume that we have already computed a_j^{l-1} for $1 \leq j \leq n_{l-1}$. I.e., we have computed the activation values for every node in level (l-1).

We'll see how to compute the activation values for every node in level l. I.e., we'll see how to compute a_j^l for $1 \leq j \leq n_l$.

1. Compute $z_i^l = \sum_{j=1}^{n_{l-1}} w_{ji}^l\, a_j^{l-1} + b_i^l$. For every $1 \leq i \leq n_l$.
2. Compute $a_i^l = \sigma(z_i^l)$

Let $\boldsymbol{W^l} = \begin{bmatrix} w_{11}^l & \cdots & w_{1n_{l-1}}^l \\ \vdots & \ddots & \vdots \\ w_{n_l 1}^l & \cdots & w_{n_l n_{l-1}}^l \end{bmatrix}$ and $\boldsymbol{z^l} = (z_1^l\ z_2^l\ \cdots\ z_{n_l}^l)^{\mathrm{T}}$

We can see that $z^l = \boldsymbol{W}^l \cdot \boldsymbol{a}^{l-1} + \boldsymbol{b}^l. where\ \boldsymbol{a}^{l-1} = \left(a_1^{l-1}\ \ a_2^{l-1} \ldots\ a_{n_{l-1}}^{l-1}\right)^{\mathrm{T}}$ & $\boldsymbol{b}^l = \left(b_1^l\ b_2^l \ldots\ b_{n_l}^l\right)^{\mathrm{T}}$.

Computation of $\boldsymbol{z}^l$ involves the computation of a matrix of size $(n_l \times n_{l-1})$ and a vector of size $(n_{l-1} \times 1)$.

As a result, the time spent at level l to compute $\boldsymbol{z}^l = O(n_{l-1}n_l + n_l + n_l) = O(n_{l-1}n_l)$.

This means that the total time taken for forward propagation = $O\left(\sum_{l=2}^{L} n_l \, n_{l-1}\right)$.

If $n_l = n \ \forall l,$ then the forward propagation time is $O(n^2 L)$

Let the indegree of the node *i* at level *l* be D_i^l.

The time to compute $\boldsymbol{z}_i^l$ is $O\left(D_i^l\right)$; the time to compute a_i^l is $O\left(D_i^l\right)$.

Therefore, the time to compute $\boldsymbol{a}^l$ is $O\left(\sum_{i=1}^{n_l} D_i^l\right) = O\left(E^l\right),$ where $E^l =$ the number of edges coming into level *l*.

Total time for forward propagation is $O(|E| + |V|)$, where *G*(*V,E*) is the Neural Network.

The next step is to compute the gradient.

Let *C* be any function to be optimized and $\boldsymbol{w}$ be the vector of parameters. We can approximate the partial derivative of *C* with respect to any parameter w_i as $\frac{\partial C}{\partial w_i} \approx \frac{C(\boldsymbol{w}+\epsilon \boldsymbol{e}_i) - C(\boldsymbol{w})}{\epsilon}$,

where $\boldsymbol{e}_i = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{bmatrix}$ (the value of the i^{th} entry is one).

This takes one forward propagation for every w_i and hence the time taken will be too much!

<u>**FACT:**</u> We can compute the gradient in one backward propagation through the network starting from level L & progressing one level at a time. This will be shown in the next lecture.