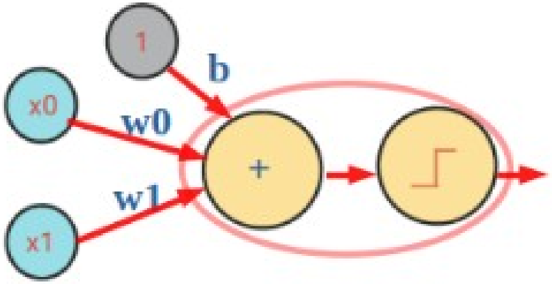
Learning logic function XOR

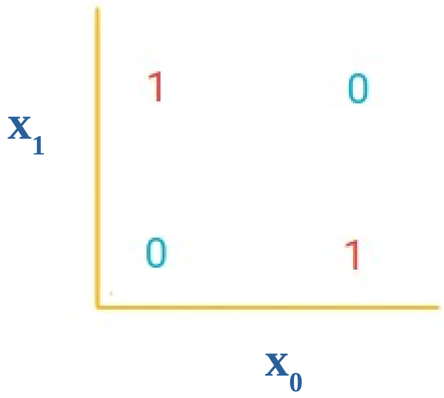
- If we consider the input (x_0, x_1) as a point on a plane



x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	0

- Perceptron equation will be

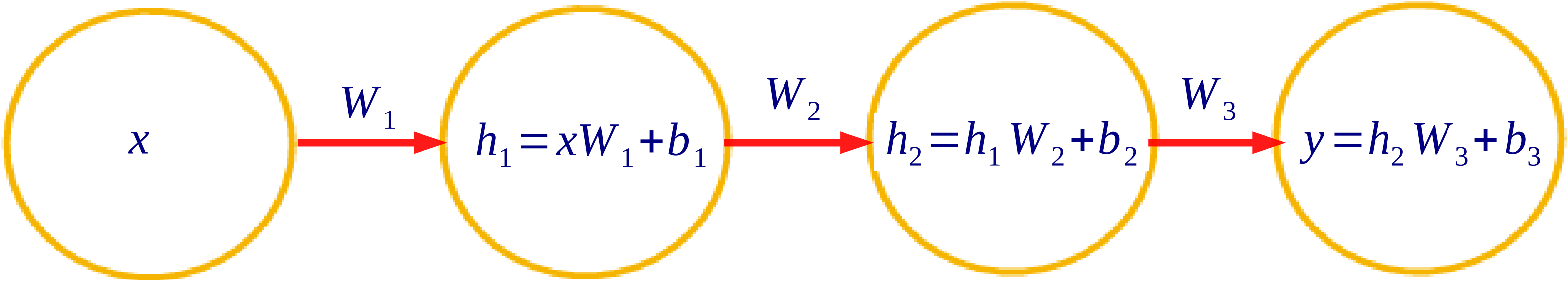
$$w_0 x_0 + w_1 x_1 + b = 0$$



Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Network with Multiple Neurons

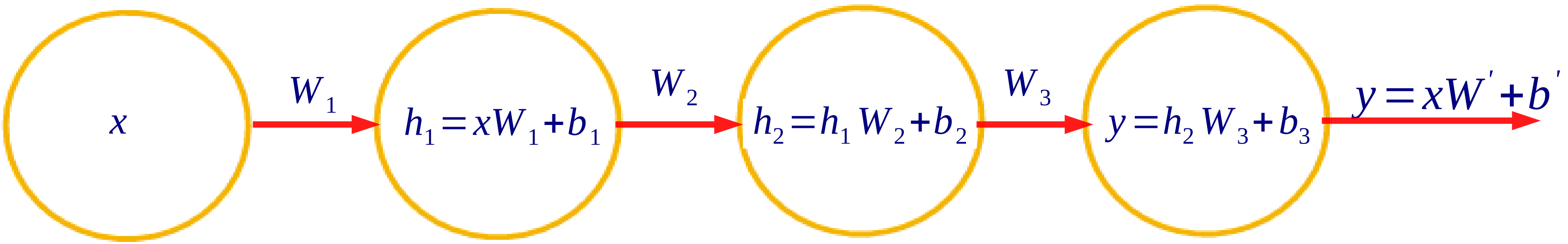
- Neural network with linear activation and any number of hidden layers is equivalent to just a linear neural neural network with no hidden layer.



$$y = h_2 W_3 + b_3$$
$$y = (h_1 W_2 + b_2) W_3 + b_3$$
$$y = h_1 W_2 W_3 + b_2 W_3 + b_3$$
$$y = (xW_1 + b_1) W_2 W_3 + b_2 W_3 + b_3 = xW_1 W_2 W_3 + b_1 W_2 W_3 + b_2 W_3 + b_3$$
$$y = xW' + b'$$

Network with Multiple Neurons

- Combination of several linear transformations can be replaced with one transformation.
- Combination of several bias term is just a single bias out come is same even if we add linear activation



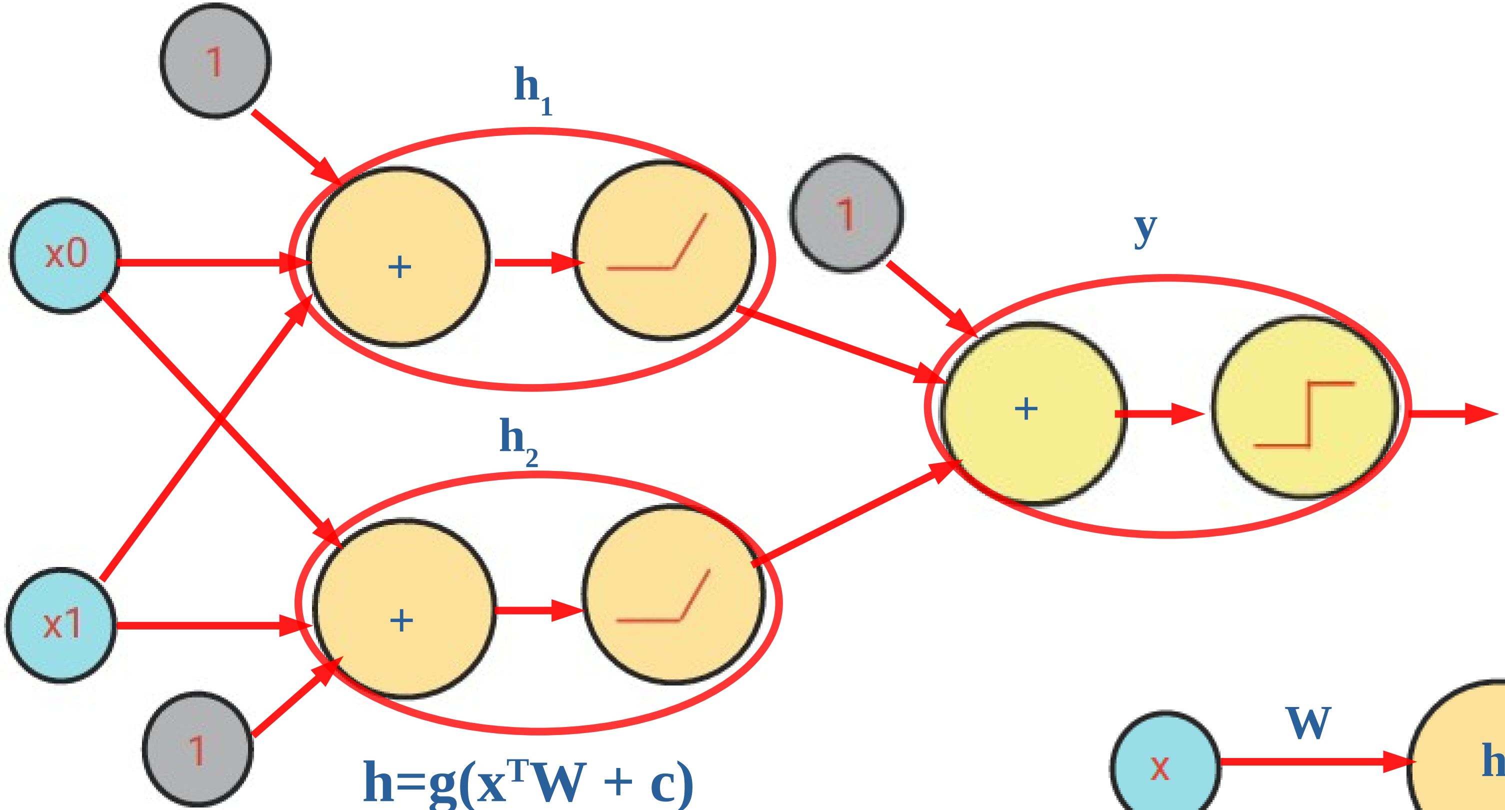
- Adding new layers does not increase the approximation power of linear neural network at all.
- We need non linear activation function to approximate non linear functions

Non Linear Models

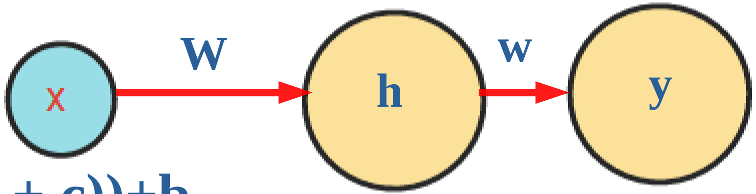
- The simplest way of modelling a nonlinear relationship is to transform the variables before estimating a regression model.
- Below regression equation is linear in parameters. But which can fit the curve

$$y = b + w_0 x_0 + w_1 x_0^2$$

Learning logic function XOR



x_0	x_1	y
0	0	0
0	1	1
1	0	1
1	1	0



$h=g(x^TW + c)$

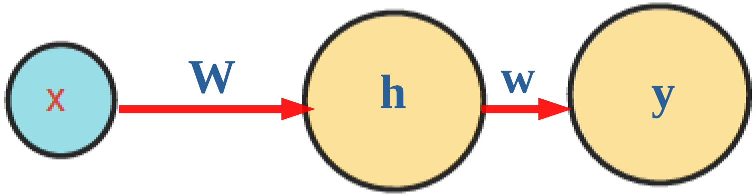
$f(x;W, c, w,b)=w^T \max(0,(W^Tx + c))+b$

Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Learning logic function XOR

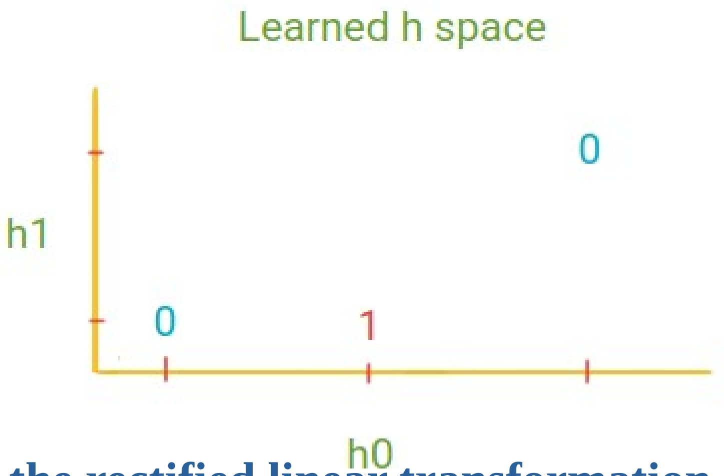
- Let $W=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$ $c=\begin{bmatrix}0\\ -1\end{bmatrix}$ $w=\begin{bmatrix}1\\ -2\end{bmatrix}$ and $b=0$

$$X=\begin{bmatrix}0 & 0\\ 0 & 1\\ 1 & 0\\ 1 & 1\end{bmatrix}$$



$f(x;W, c, w,b)=w^T \max(0,(W^Tx + c))+b$

$$XW=\begin{bmatrix}0 & 0\\ 1 & 1\\ 1 & 1\\ 2 & 2\end{bmatrix} \quad XW+c=\begin{bmatrix}0 & -1\\ 1 & 0\\ 1 & 0\\ 2 & 1\end{bmatrix} \quad h=\begin{bmatrix}0 & 0\\ 1 & 0\\ 1 & 0\\ 2 & 1\end{bmatrix}$$



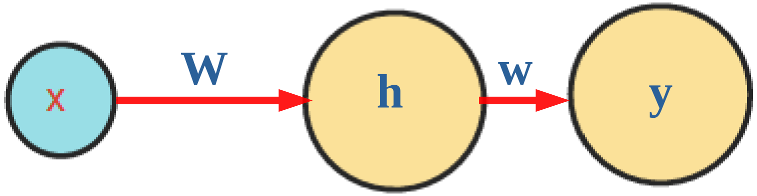
To finish computing the value of h for each example, we apply the rectified linear transformation

Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Learning logic function XOR

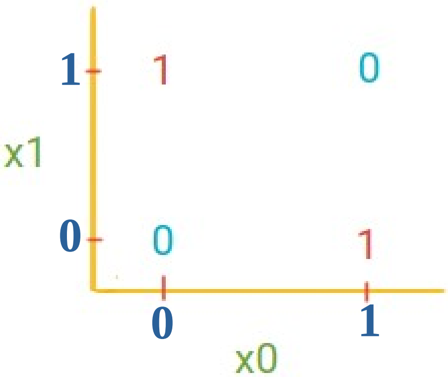
- Let $W=\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$ $c=\begin{bmatrix}0\\ -1\end{bmatrix}$ $w=\begin{bmatrix}1\\ -2\end{bmatrix}$ and $b=0$

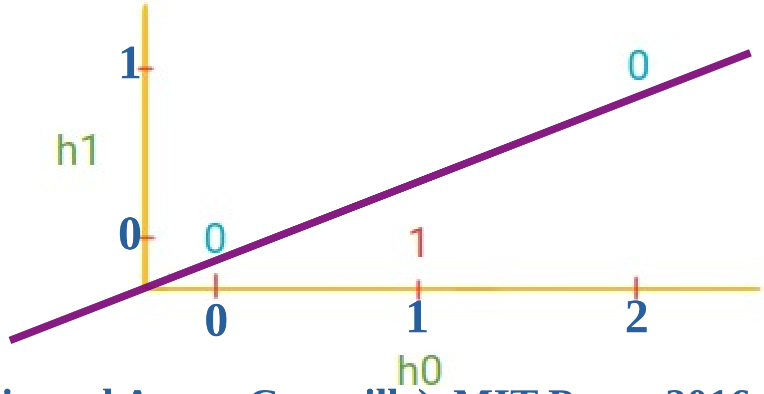
$$X=\begin{bmatrix}0 & 0\\ 0 & 1\\ 1 & 0\\ 1 & 1\end{bmatrix}$$



$$\begin{bmatrix}0\\ 1\\ 1\\ 0\end{bmatrix}$$

f(x;W, c, w,b)=w^T max(0,(W^Tx + c))+b





Reference: Deep Learning (Ian J. Goodfellow, Yoshua Bengio and Aaron Courville), MIT Press, 2016.

Activation Function

- Activation Function activates the neuron(fire or not)

  1. Activation function adds the **non linearity** to neural network by performing fixed mathematical operation on output from summation operation.

  2. It will **normalize the output** from summation operation.

Properties of Activation Function

- Activation functions have different mathematical properties

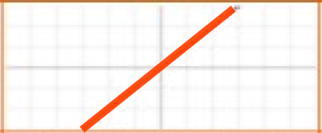
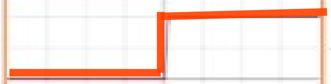
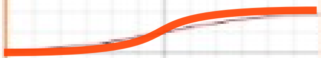
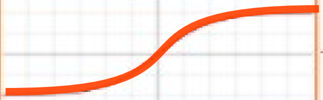
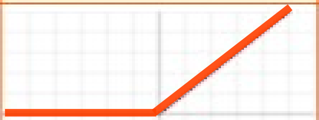
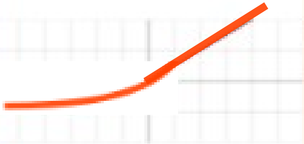
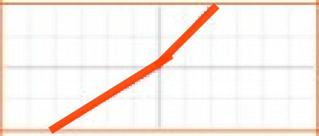
  1. Nonlinear
     - When the activation function is non-linear, then a two-layer neural network can be proven to be a **universal function approximator**
  2. Range
     - When the **range** of the activation function **is finite**, gradient-based **training** methods tend to be more **stable**, because pattern presentations significantly affect only limited weights

Properties of Activation Function

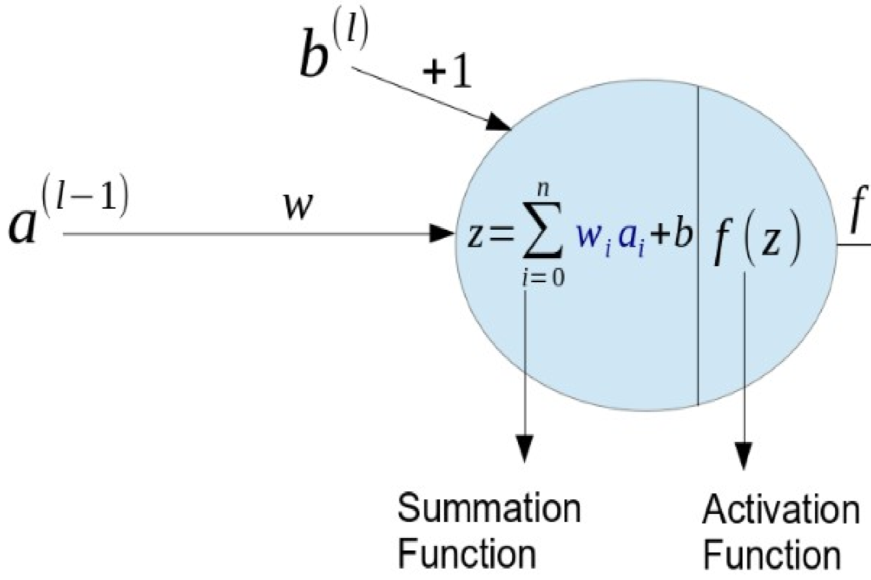
- Activation functions have different mathematical properties

    3. Continuously differentiable

    - The binary step activation function is not differentiable at 0, and it differentiates to 0 for all other values, so gradient-based methods can **make no progress** with it.

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range
Identity		x	1	$(-\infty, \infty)$
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1 + e^{-x}}$	$f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$
Rectified linear unit (ReLU)		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Exponential linear unit (ELU)		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$
Leaky ReLU		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Softmax		$\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	$f_i\left(\vec{x}\right)\left(\delta_{ij} - f_j\left(\vec{x}\right)\right)$	$(0, 1)$

Activation Function

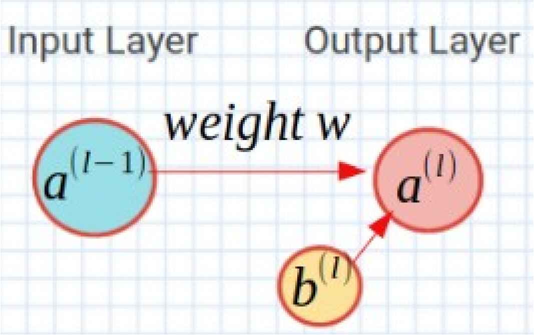


$$\underbrace{f(b^{(l)}+w a^{(l-1)})}_{} = a^{(l)}$$

$$z = b^{(l)}+w a^{(l-1)}$$

$$(a^{(l)}-y)^2 = J$$

y



$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w}$$

$$(a^{(l)}-y)^2 = J$$

Find how much a change of the $a^{(l)}$ affects the total error

Then find how much a change of the z affects the $a^{(l)}$

Next find how much a change of the w affects the z

$$w_{new} = w - \alpha \frac{\partial J}{\partial w}$$

Linear Activation Function

- If a linear activation function is used, the derivative of the cost function is a constant with respect to (w.r.t) input
- so the value of input (to neurons) does not affect the updating of weights.
- This means that we can not figure out which weights are most effective in creating a good result and therefore we are forced to change all weights equally.

Linear Activation Function

$$\frac{\partial J}{\partial a^{(l)}} = 2(a^{(l)} - y)$$

$$\frac{\partial a^{(l)}}{\partial z^{(l)}} = \frac{\partial f(z^{(l)})}{\partial z^{(l)}} = f'(z^{(l)})$$

$$\frac{\partial z^{(l)}}{\partial w} = \frac{\partial (b + a^{(l-1)} w)}{\partial w} = a^{(l-1)}$$

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w} = 2(a^{(l)} - y) f'(z^{(l)}) a^{(l-1)} = 2(a^{(l)} - y) * 1 * a^{(l-1)}$$

Step Activation Function

- Heaviside step function is non-differentiable at z = 0
- It has 0 derivative elsewhere.
- This means that gradient descent won't be able to make a progress in updating the weights.

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w} = 2\left(a^{(l)} - y\right) f'\left(z^{(l)}\right) a^{(l-1)} = 2\left(a^{(l)} - y\right) * 0 * a^{(l-1)}$$

Sigmoid Activation Function

- It takes a real-valued number and “squashes” it into range between 0 and 1.
- In particular, large negative numbers become 0 and large positive numbers become 1.
- The sigmoid function has seen frequent use historically
- Since it has a nice interpretation as the firing rate of a neuron: from not firing at all (0) to fully-saturated firing at an assumed maximum frequency (1)

Refrence: https://cs231n.github.io/neural-networks-1/

Sigmoid Activation Function

- It has two major drawbacks
- 1. Sigmoids saturate and kill gradients
  - when the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero.
  - during backpropagation, this (local) gradient will be multiplied to the gradient of this gate's output for the whole objective.
  - Therefore, if the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data

Refrence: https://cs231n.github.io/neural-networks-1/

Sigmoid Activation Function

- It has two major drawbacks
- 1. Sigmoids saturate and kill gradients
  - if the initial weights are too large then most neurons would become saturated and the network will barely learn

$$\frac{\partial J}{\partial w}=\frac{\partial J}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial w}=2\left(a^{(l)}-y\right)\sigma'\left(z^{(l)}\right)a^{(l-1)}=2\left(a^{(l)}-y\right)z^{(l)}\left(1-z^{(l)}\right)a^{(l-1)}$$

0.99(1-0.99) = 0.0099 **0.01(1-0.01) = 0.0099**

Refrence: https://cs231n.github.io/neural-networks-1/

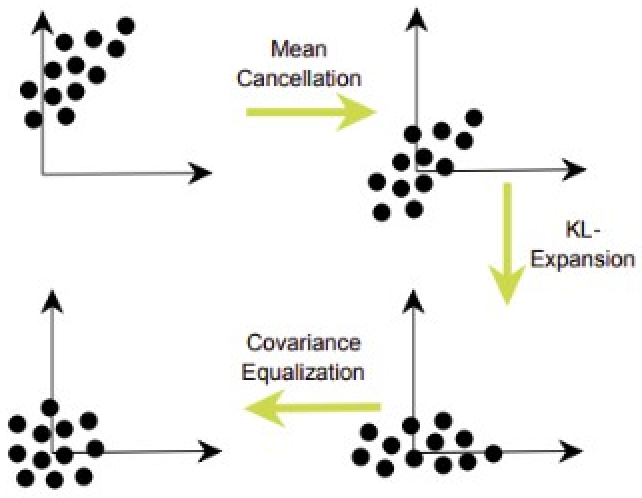
Sigmoid Activation Function

- It has two major drawbacks
- 2. Sigmoid outputs are not zero-centered
  - Convergence is usually faster if the average of each input variable over training set is close to zero
  - Any shifting of average input away from zero will bias the updates in particular direction and thus slow down learning
  - It is good to shift the inputs so the average over the training set is close to zero

Refrence: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

Sigmoid Activation Function

- It has two major drawbacks
- 2. Sigmoid outputs are not zero-centered



Refrence: http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf

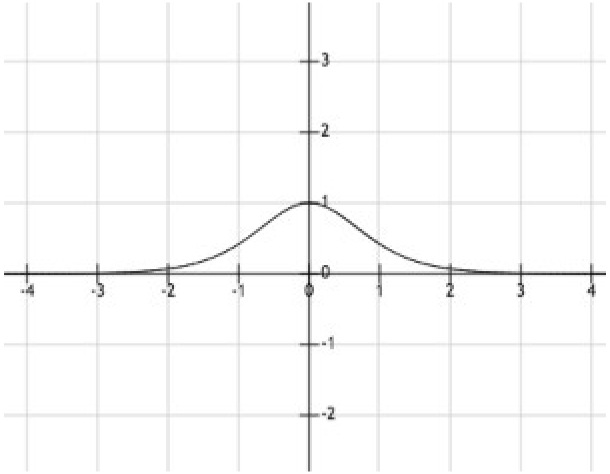
Tanh Activation Function

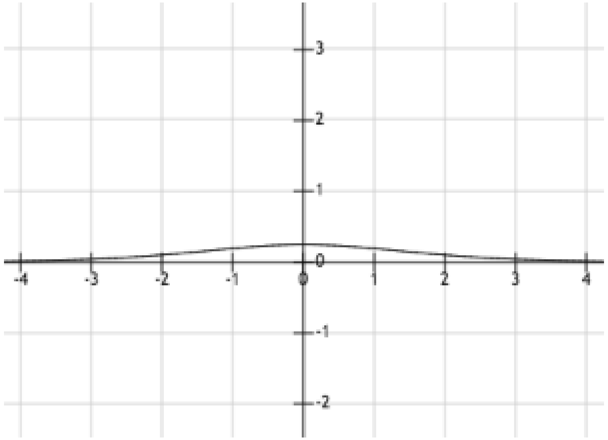
- Tanh squashes a real-valued number to the range [-1, 1].
- Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered.
- Therefore, in practice the tanh non-linearity is always preferred to the sigmoid nonlinearity.
- tanh neuron is simply a scaled sigmoid neuron, in particular the following holds: $\tanh(x)=2\sigma(2x)-1$

Refrence: https://cs231n.github.io/neural-networks-1/

Tanh Activation Function

- Since data is centered around 0, the derivatives are higher





For input between [-1,1], we have derivative between [0.42, 1].

For input between [0,1], we have derivative between [0.20, 0.25]

Refrence: https://stats.stackexchange.com/questions/101560/tanh-activation-function-vs-sigmoid-activation-function

ReLU Activation Function

- The Rectified Linear Unit has become very popular in the last few years.

  f(x)=max(0,x)

- In other words, the activation is simply thresholded at zero.
- There are several pros and cons to using the ReLUs

Refrence: https://cs231n.github.io/neural-networks-1/

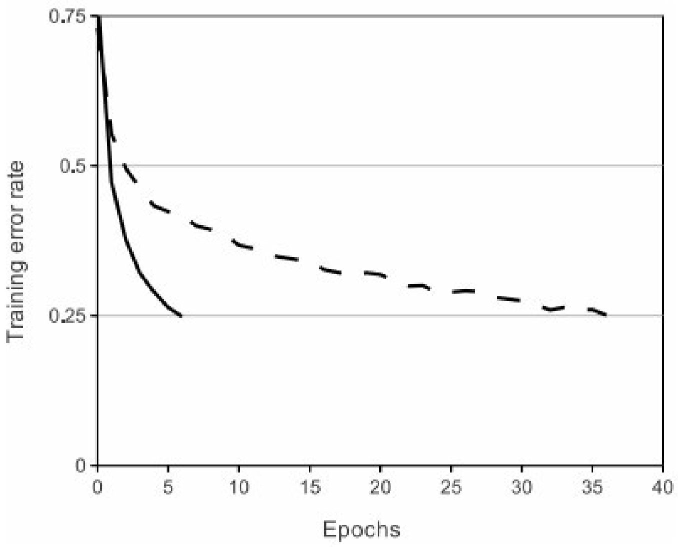
ReLU Activation Function

- Pros
  - Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.),
  - the ReLU can be implemented by simply thresholding a matrix of activations at zero.
  - It was found to greatly accelerate (e.g. a factor of 6) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions.
  - It is argued that this is due to its linear, non-saturating form.

Refrence: https://cs231n.github.io/neural-networks-1/

ReLU Activation Function

- Pros
  - It was found to greatly accelerate (e.g. a factor of 6) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions.



ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line).

Refrence: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

ReLU Activation Function

- Cons
  - **Non-differentiable at zero:** however, it is differentiable anywhere else, and the value of the derivative at zero can be arbitrarily chosen to be 0 or 1.
  - Not zero-centered.
  - Unbounded.
  - **Dying ReLU problem:** ReLU (Rectified Linear Unit) neurons can sometimes be pushed into states in which they become inactive for essentially all inputs.
  - In this state, no gradients flow backward through the neuron, and so the neuron becomes stuck in a perpetually inactive state and "dies". This is a form of the vanishing gradient problem.

Refrence: https://cs231n.github.io/neural-networks-1/

Leaky ReLU Activation Function

- In this state, no gradients flow backward through the neuron, and so the neuron becomes stuck in a perpetually inactive state and "dies". This is a form of the vanishing gradient problem.
- In some cases, large numbers of neurons in a network can become stuck in dead states, effectively decreasing the model capacity.
- This problem typically arises when the learning rate is set too high.
- Leaky ReLUs are one attempt to fix the “dying ReLU” problem.
- Instead of the function being zero when $x < 0$, a leaky ReLU will instead have a small positive slope (of 0.01, or so).

Refrence: https://cs231n.github.io/neural-networks-1/

Softmax Activation Function

- The softmax function, also known normalized exponential function, is a generalization of the logistic function to multiple dimensions
- It is used in multinomial logistic regression and is often used as the last activation function of a neural network
- it normalize the output of a network to a probability distribution over predicted output classes
- The softmax function takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.

Refrence: https://en.wikipedia.org/wiki/Softmax_function

Softmax Activation Function

- In simple words, it applies the standard exponential function to each element z_i of the input vector z and normalizes these values by dividing by the sum of all these exponentials;

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}} \text{ for } i = 1, \ldots\ldots, K \land z = (z_1, \ldots\ldots, z_k)$$

- this normalization ensures that the sum of the components of the output vector is 1.

Refrence: https://en.wikipedia.org/wiki/Softmax_function

Softmax Activation Function

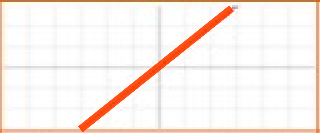
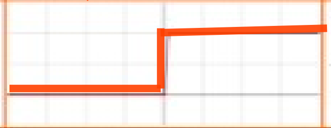
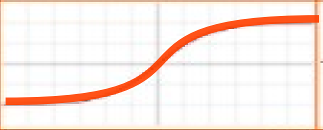
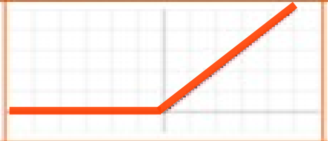
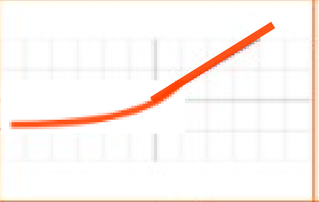
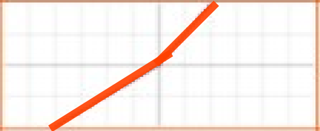
```
[6]  import numpy as np
```

```
[7]  z = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
     np.exp(z) / np.sum(np.exp(z))

     array([0.02364054, 0.06426166, 0.1746813 , 0.474833  , 0.02364054,
            0.06426166, 0.1746813 ])
```

```
[8]  np.exp(z) / (1+ np.exp(z))

     array([0.73105858, 0.88079708, 0.95257413, 0.98201379, 0.73105858,
            0.88079708, 0.95257413])
```

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}} \, for \; i = 1, \ldots\ldots, K \wedge z = (z_1, \ldots\ldots, z_k)$$

Refrence: https://en.wikipedia.org/wiki/Softmax_function

Name	Plot	Function, $f(x)$	Derivative of f, $f'(x)$	Range
Identity		x	1	$(-\infty, \infty)$
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Logistic, sigmoid, or soft step		$\sigma(x) = \dfrac{1}{1 + e^{-x}}$	$f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic tangent (tanh)		$\tanh(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - f(x)^2$	$(-1, 1)$
Rectified linear unit (ReLU)		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Exponential linear unit (ELU)		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$
Leaky ReLU		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Softmax		$\dfrac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	$f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$	$(0, 1)$