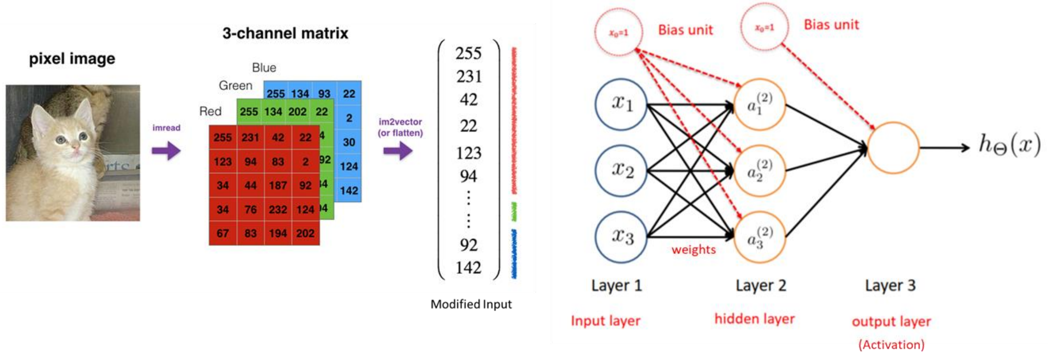
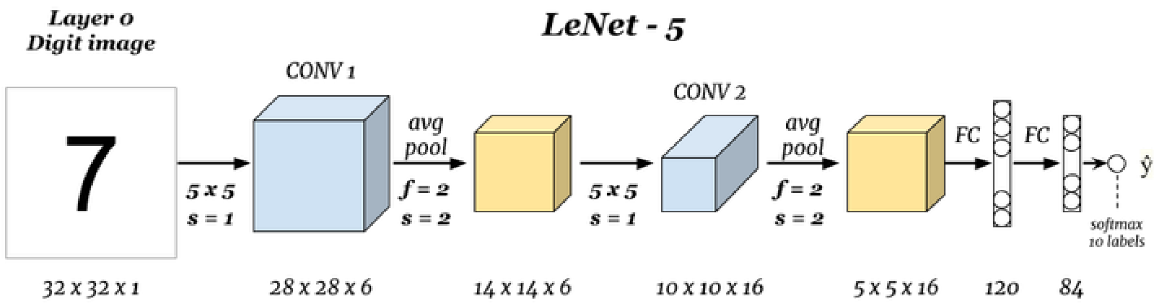
Lecture 20
Parameter Learning in NN

Disclosure: The slides are adopted from Stanford CS231n and Andrew Ng's online Machine Learning Course.

Quick recap on what we learned so far

- Fully Neural Network
- Convolutional Neural Network
- An NN architecture
  - How many layers?
  - How many neurons? and
  - How the neurons are connected?
- The parameters values (weights!)
- Components
  - Filters:
    - Convolutional Filter
    - Pooling Filters
  - Activation Functions
    - Sigmoid
    - ReLu etc.

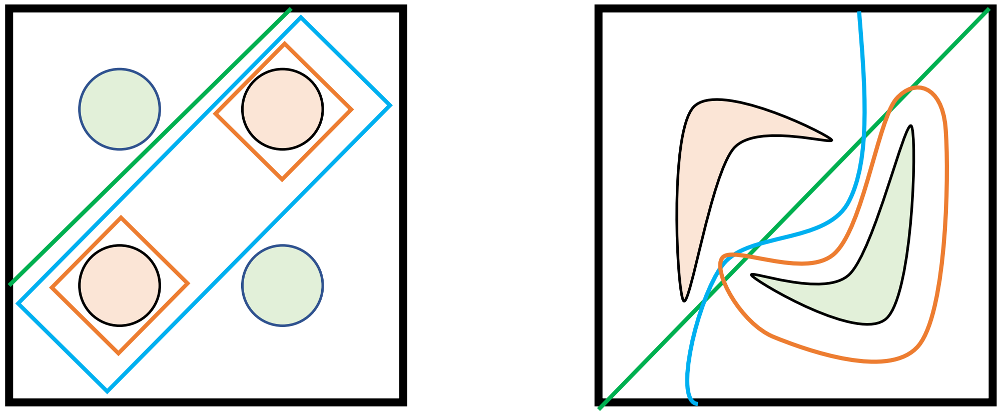




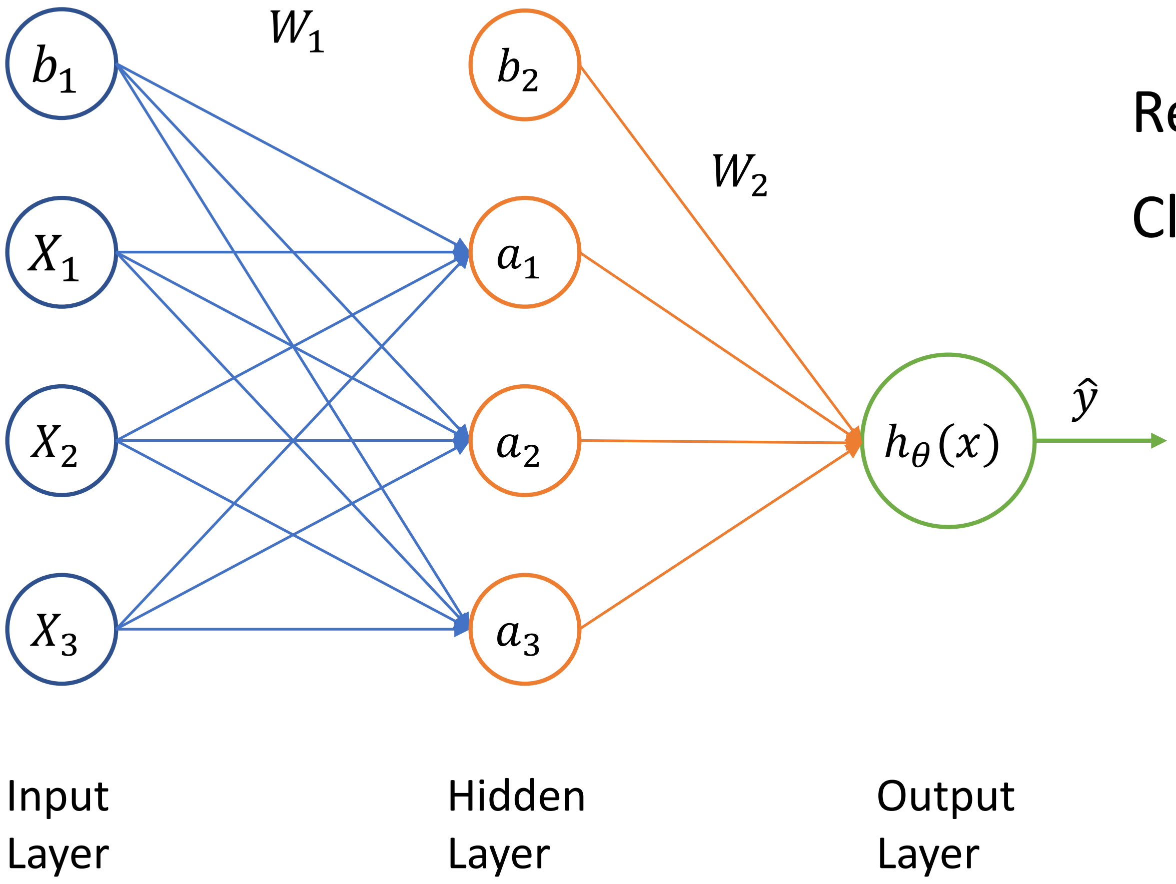
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

Quick recap on what we learned so far

- How many layers should we use?
  - Theoretically:
    - A NN with one Hidden layer is a universal function approximator (*Cybenko, 1989*)
  - Empirically:
    - Before year 2006 : There should be at least 2 – 6 hidden layers.
    - After year 2006 : There should be at least 5+ hidden layers.
- How the decision boundary works?
  - 0 Hidden Layer
  - 1 Hidden Layer
  - 2 Hidden Layer



Quick recap on what we learned so far



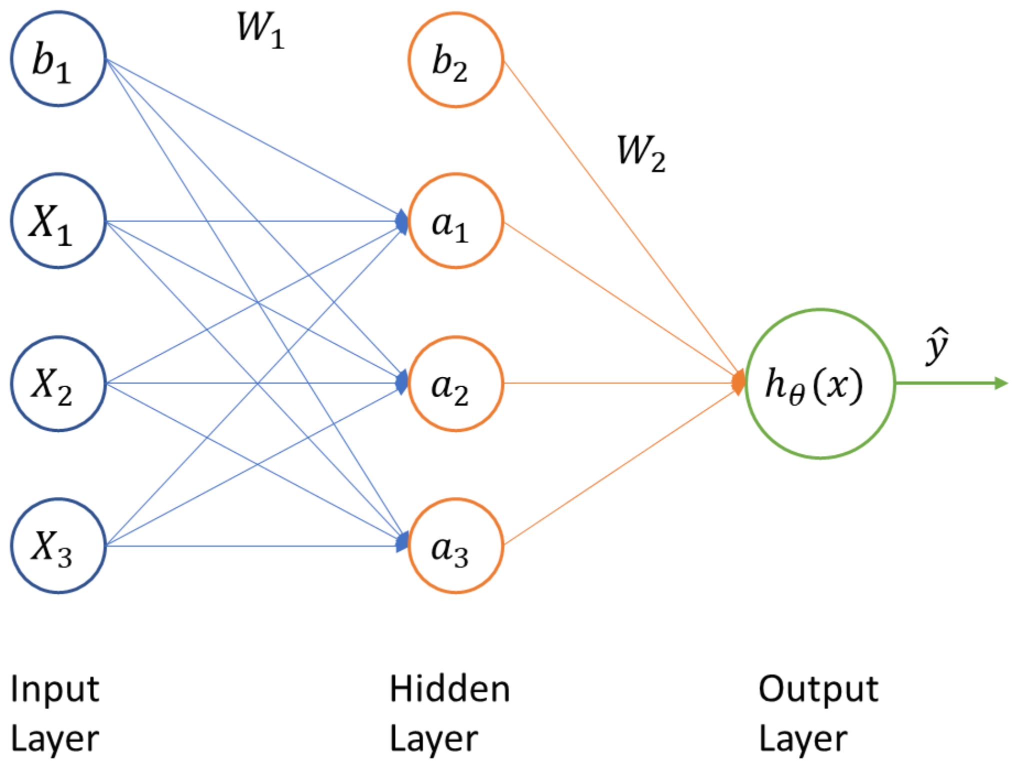
Regression Problem, $\widehat{y_r} = W_2^T a_i + b_2$

Classification Problem, $\widehat{y_c} = \sigma(W_2^T a_i + b_2)$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

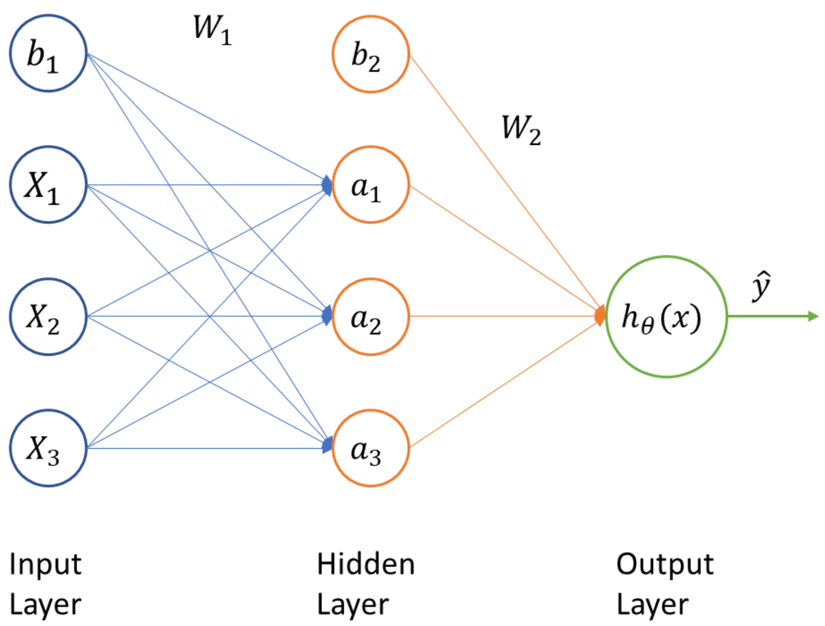
Learning from Network

- The two Components:
  - Architecture
  - Parameters (W_1, W_2)
- Learning Steps:
  1. Initialize the ***weights,*** $\boldsymbol{W_n}$
  2. Calculate the ***forward propagation***
  3. Calculate the loss function
  4. Perform ***backpropagation***
  5. Update the parameters
  6. Repeat until convergence!



Step 1: Weight Initialization

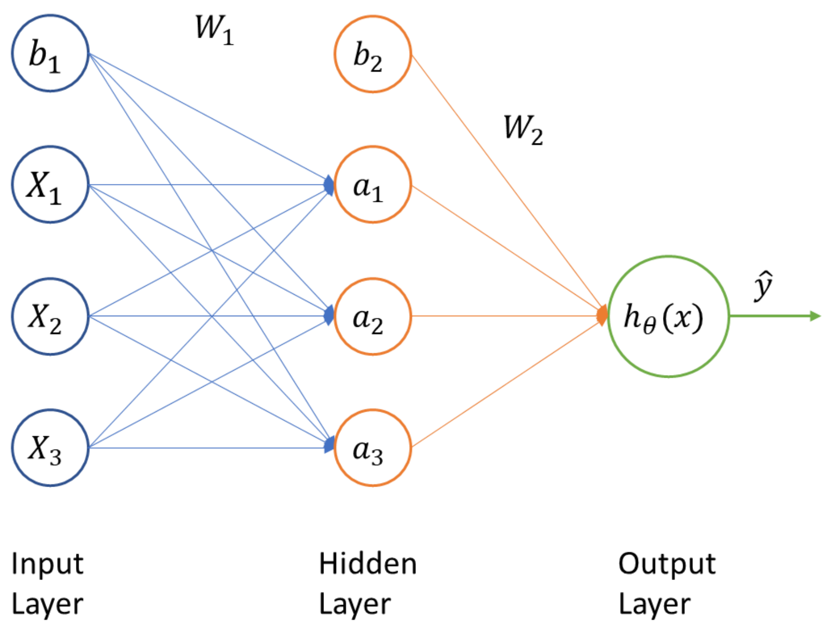
- Q1. How to initialize the ***weights, W_n*** ?
  - Randomly initialize the parameters to small values (e.g., normally distributed round zero; $\mathcal{N}\ (0, 0.1)$).
- Q2. What will happen if we initialize the weights as **zero**?
  - The first layer will always be the same since, $W^{[1]}x^i + b^{[1]} = 0^{3\times1}x^i + 0^{3\times1}$ where, $0^{n\times m}$ denotes a matrix of size $n \times m$ filled with zeros.
  - Again, the output of the network has a sigmoid function, thus no matter what input we provide we will get an output probability of 0.5 i.e. $\sigma(0) = 0.5$.



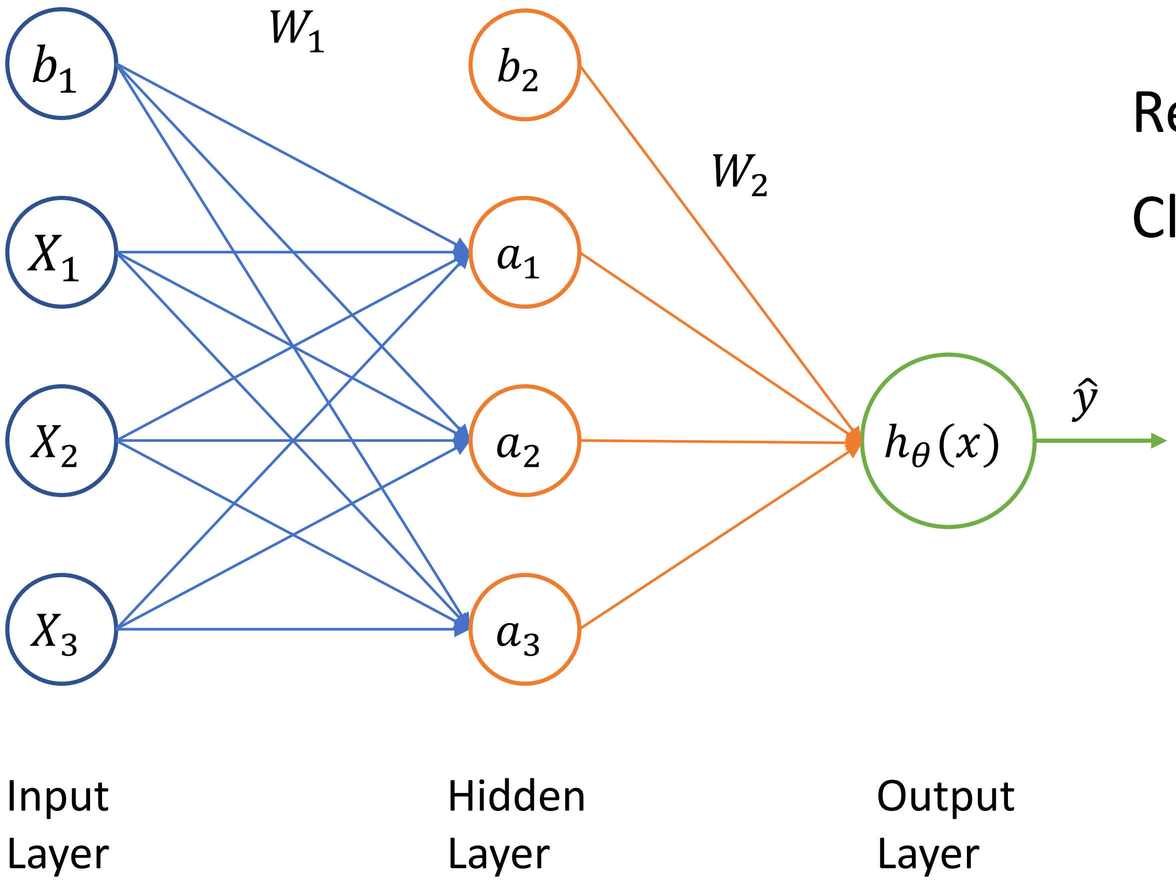
Step 1: Weight Initialization

- Q3. What if we initialized all parameters to be the same non-zero value?
  - Each element of the activation vector will be the same (because $W^{[1]}$ contains all the same values). This behavior will occur at all layers of the neural network. As a result, when we compute the gradient, all neurons in a layer will be equally responsible for anything contributed to the final loss
  - **Summary**: *All neurons will learn the same thing.*
- Q4. Is there a better approach than randomly initializing?
  - Yes, there is one! It's called Xavier/He initialization.

$$w^{[\ell]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[\ell]} + n^{[\ell-1]}}}\right)$$



Step 2: Forward Propagation



Regression Problem, $\widehat{y_r} = W_2^T a_i + b_2$

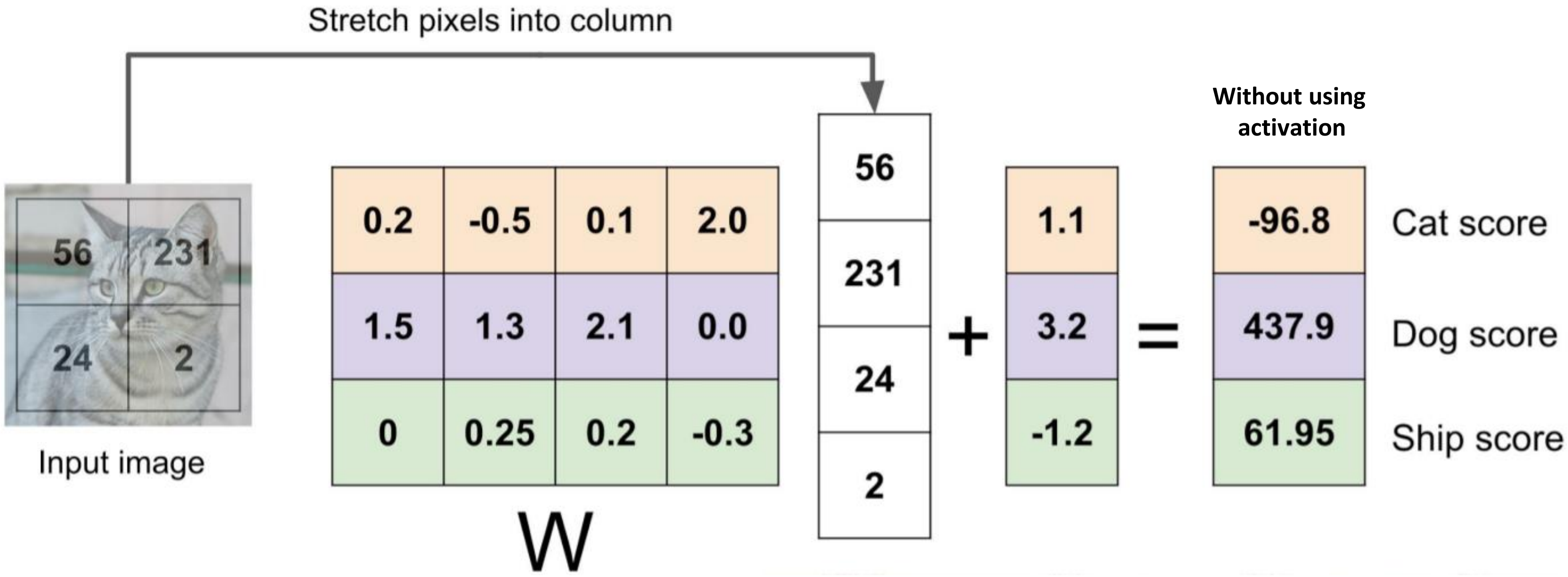
Classification Problem, $\widehat{y_c} = \sigma(W_2^T a_i + b_2)$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Hidden Layer Calculation (with activation),

$$a_i = \sigma(W_1^T X + b_1)$$

Step 2: Forward Propagation



Step 3: Loss Function

- Loss function is defined as –

$$\mathcal{L}(\underbrace{y}_{\text{True Outcome}}, \underbrace{\hat{y}}_{\text{Predicted outcome}}, \overbrace{X}^{\text{Inputs Variable}}) = \frac{1}{2}(\, y \;-\hat{y}\,)^2$$

- In terms of log-loss, the loss function is defined as –

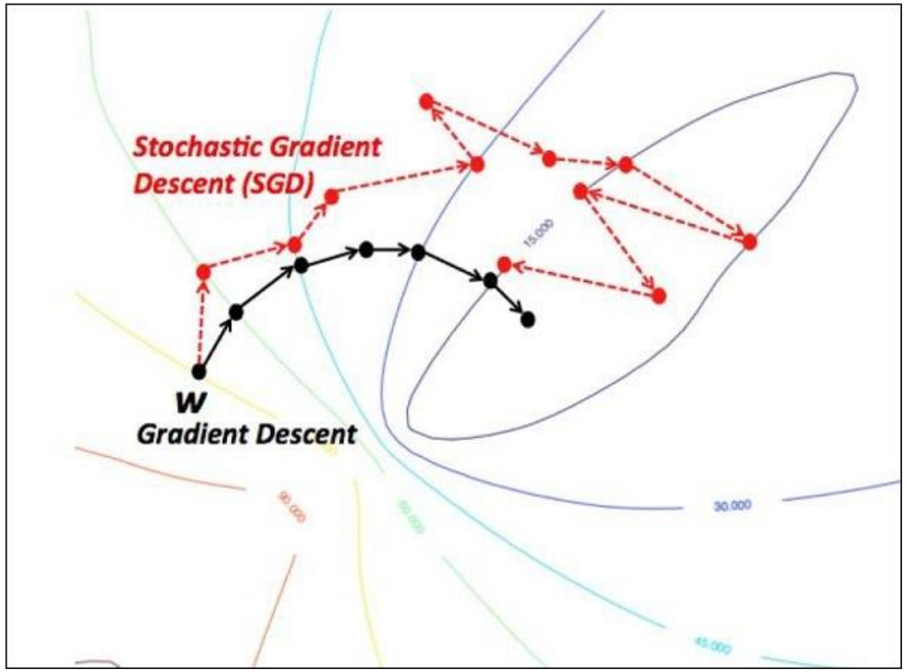
$$\mathcal{L}(y, \hat{y}, X) = -[(1 - y)\log(1 - \hat{y}) + y\ log\ \hat{y}]$$

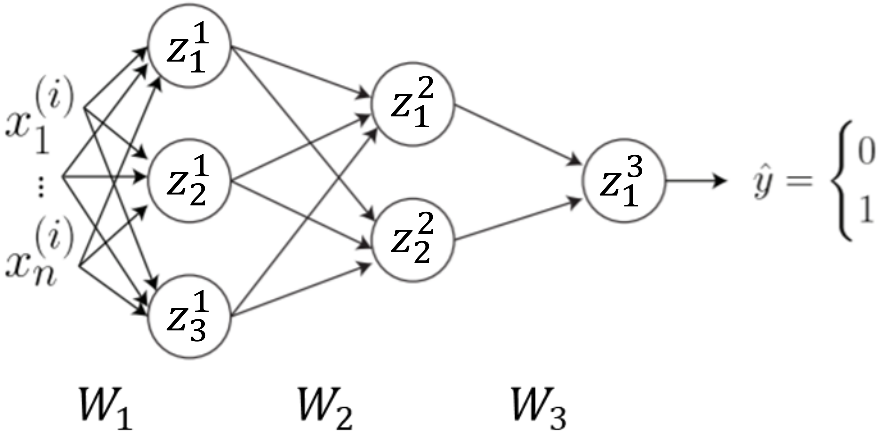
Step 4:Back Propagation

- The partial derivative of $\mathcal{L}$ can be decomposed as a sum of partial derivatives of individual losses:

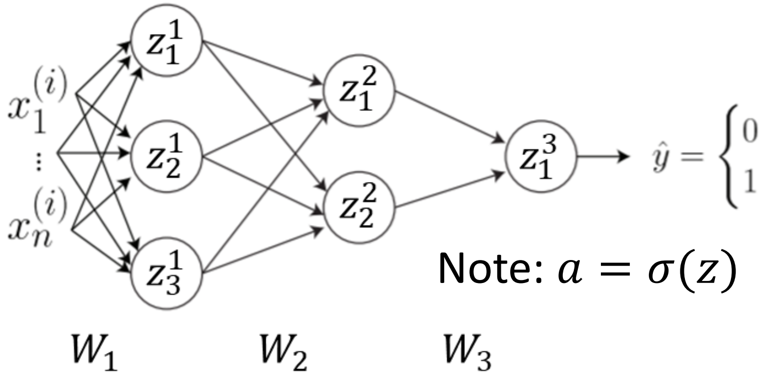
$$\frac{\partial \mathcal{L}}{\partial W^l} = \sum_{k=1}^{n} \frac{\partial \mathcal{L}(y, \hat{y})}{\partial W^l}$$

- Say, we have the network parameters, $W^1, W^2, W^3, b^1, b^2, b^3$.
- We will first calculate the gradient with respect to W^3 as its due to influence of W^1 on output is complex than that of W^3.





Step 4: Updating the Parameters (Back Propagation)



Note: $a = \sigma(z)$

- Thus we have,

$$\frac{\partial \mathcal{L}}{\partial W^3} = -\frac{\partial}{\partial W^3}\left((1-y)\log(1-\hat{y}) + y\log\hat{y}\right)$$

$$= -(1-y)\frac{\partial}{\partial W^3}\log\left(1 - \sigma(1 - W^3a^2 + b^3)\right) - y\frac{\partial}{\partial W^3}\log\left(\sigma(W^3a^2 + b^3)\right)$$

$$= (1-y)\sigma(W^3a^2 + b^3){a^2}^T - y\left(1 - \sigma(W^3a^2 + b^3)\right){a^2}^T$$

$$= (1-\mathrm{y})\,\mathrm{a}^3{\mathrm{a}^2}^{\mathrm{T}} - \mathrm{y}(1-\mathrm{a}^3){\mathrm{a}^2}^{\mathrm{T}}$$

$$= (a^3 - y)\,{a^2}^T$$

- To compute the gradient with respect to W^2:

$$\frac{\partial \mathcal{L}}{\partial W^2} = \underbrace{\frac{\partial \mathcal{L}}{\partial a^3}\frac{\partial a^3}{\partial z^3}}_{a^3 - y}\underbrace{\frac{\partial z^3}{\partial a^2}}_{W^3}\underbrace{\frac{\partial a^2}{\partial z^2}}_{\sigma'(z^2)}\underbrace{\frac{\partial z^2}{\partial W^2}}_{a^1}$$

Note: $\sigma' = \sigma(1-\sigma)$

Step 4.1: Regularization (if applicable)

- Similar to what we covered in Lecture 13. (Remember for linear regression)
- Is used to reduce the overfitting of neural networks.

Hypothesis: $h_{\beta}(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$

Cost Function: $J(\beta) = \frac{1}{2n}\left[\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1^{(i)} - \cdots - \beta_p x_p^{(i)})^2 + \lambda \sum_{j=1}^{p} \beta_{j=1}^2\right]$

Parameters: $\beta_0, \beta_1, \ldots, \beta_p$

$$\min_{\beta} \mathrm{J}(\beta)$$

Gradient descent:

Repeat until convergence{

$$\beta_0 := \beta_0 + \alpha \frac{1}{n}\left[\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1^{(i)} - \cdots - \beta_p x_p^{(i)})\right]$$

$$\beta_1 := \beta_1 + \alpha \frac{1}{n}\left[\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1^{(i)} - \cdots - \beta_p x_p^{(i)})\, x_1^{(i)} - \lambda \beta_1\right]$$

...

$$\beta_p := \beta_p + \alpha \frac{1}{n}\left[\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1^{(i)} - \cdots - \beta_p x_p^{(i)})\, x_p^{(i)} - \lambda \beta_p\right]$$

}

NOTE:

$$\beta_p := \beta_p\left(1 - \alpha\frac{\lambda}{n}\right) + \alpha \frac{1}{n}\left[\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_1^{(i)} - \cdots - \beta_p x_p^{(i)})\, x_p^{(i)}\right]$$

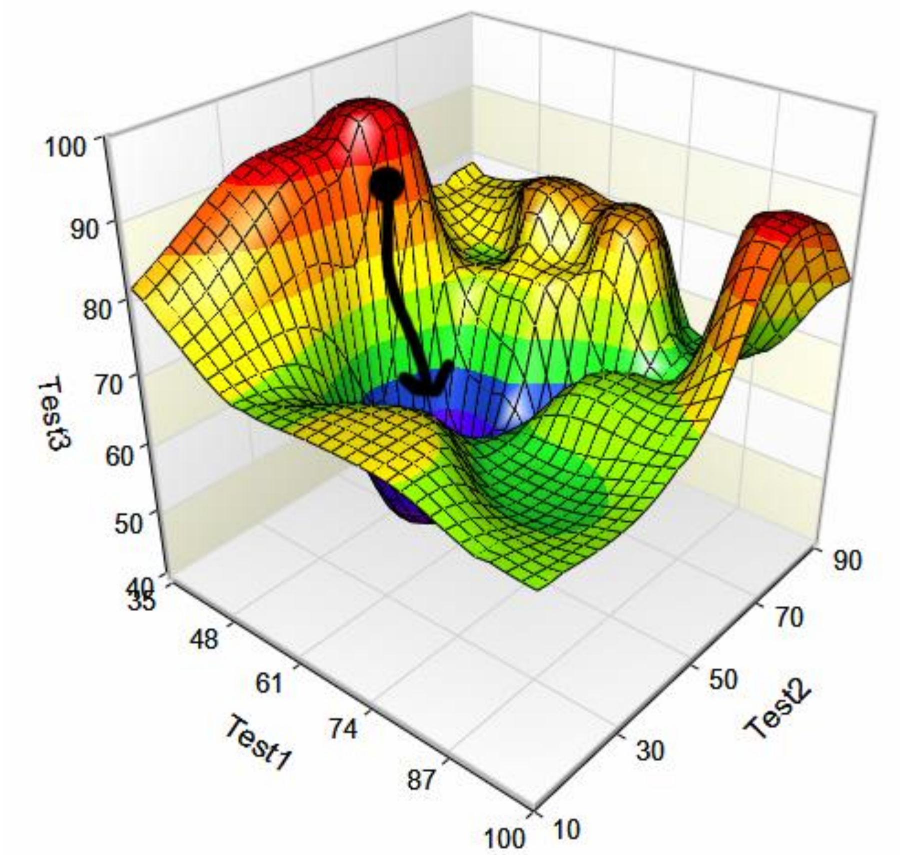
Step 5: Updating the Parameters

- Optimize/update the parameters by using gradient descent optimization. For a given number of layer, l:

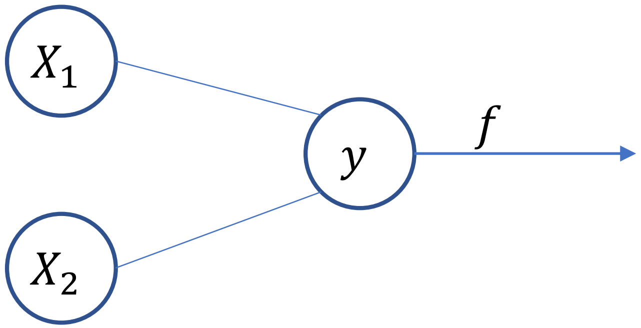
$$W^l = W^l - \alpha \frac{\partial \mathcal{L}}{\partial W^l}$$

$$b^l = b^l - \alpha \frac{\partial \mathcal{L}}{\partial b^l}$$

- This computation is non-trivial at hidden layers ($l < L$ (Output Layer)) but has complex chain of influence via activation values at subsequent layers.
- This problem is already addressed by **backpropagation.**

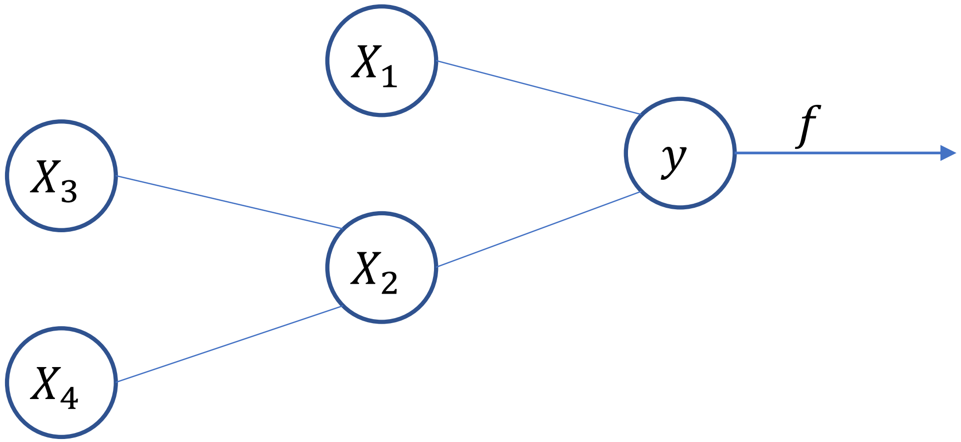


Example: Backpropagation



$$f = X_1 - X_2$$

$$\frac{\partial f}{\partial X_1} = 1 \qquad \frac{\partial f}{\partial X_2} = 1$$



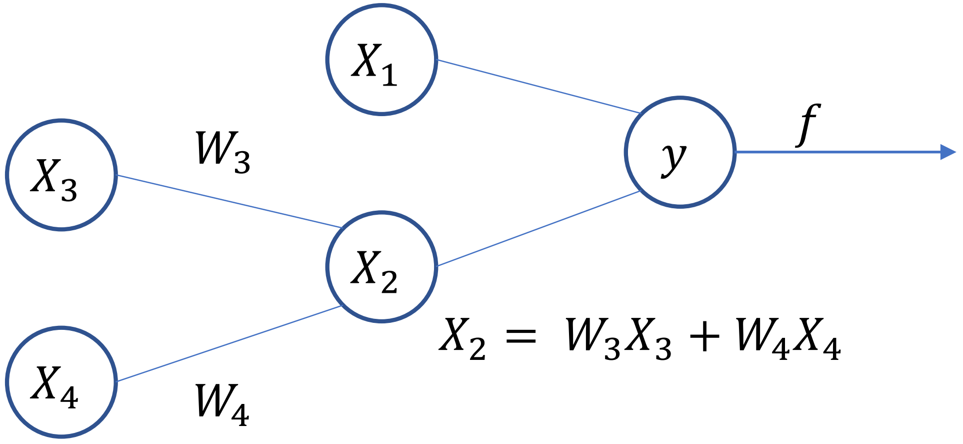
$$f = X_1 - X_2 = X_1 - (X_3 + X_4)$$

$$\frac{\partial X_2}{\partial X_3} = 1 \qquad \frac{\partial X_2}{\partial X_4} = 1$$

$$\frac{\partial f}{\partial X_3} = ?$$

$$\frac{\partial f}{\partial X_3} = \frac{\partial f}{\partial X_2}\frac{\partial X_2}{\partial X_3} = -1 \times 1 = -1$$

Example: Backpropagation

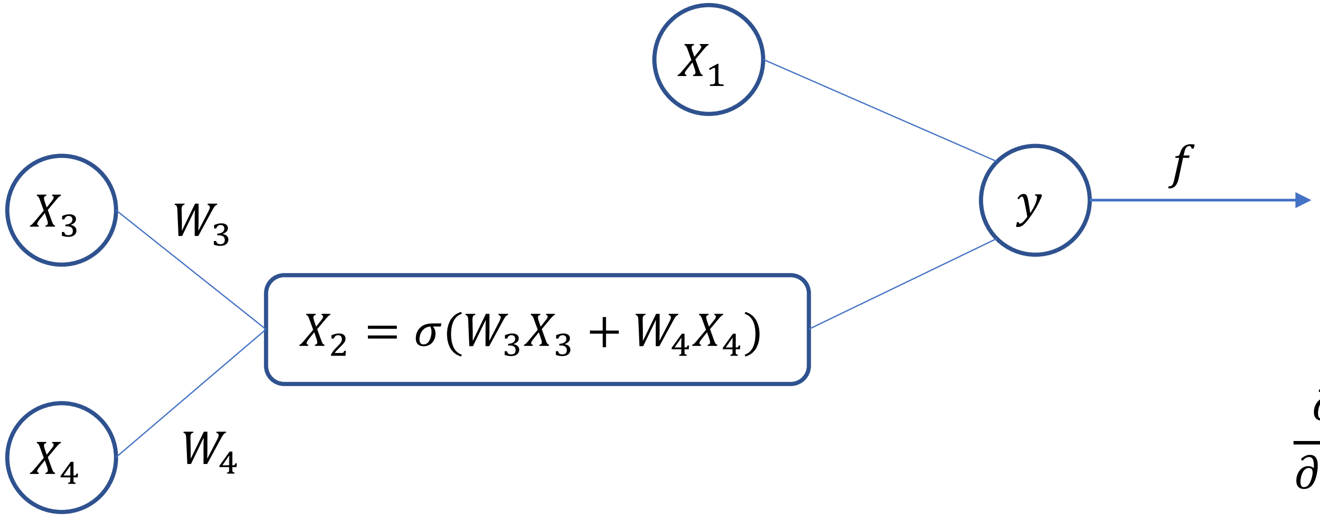


$$f = X_1 - X_2 = X_1 - (W_3X_3 + W_4X_4)$$

$$\frac{\partial X_2}{\partial X_3} = 1 \qquad \frac{\partial X_2}{\partial X_4} = 1$$

$$\frac{\partial f}{\partial W_3} = ?$$

$$\frac{\partial f}{\partial W_3} = \frac{\partial f}{\partial X_2} \times \frac{\partial X_2}{\partial W_3} = -1 \times X_3 = -X_3$$

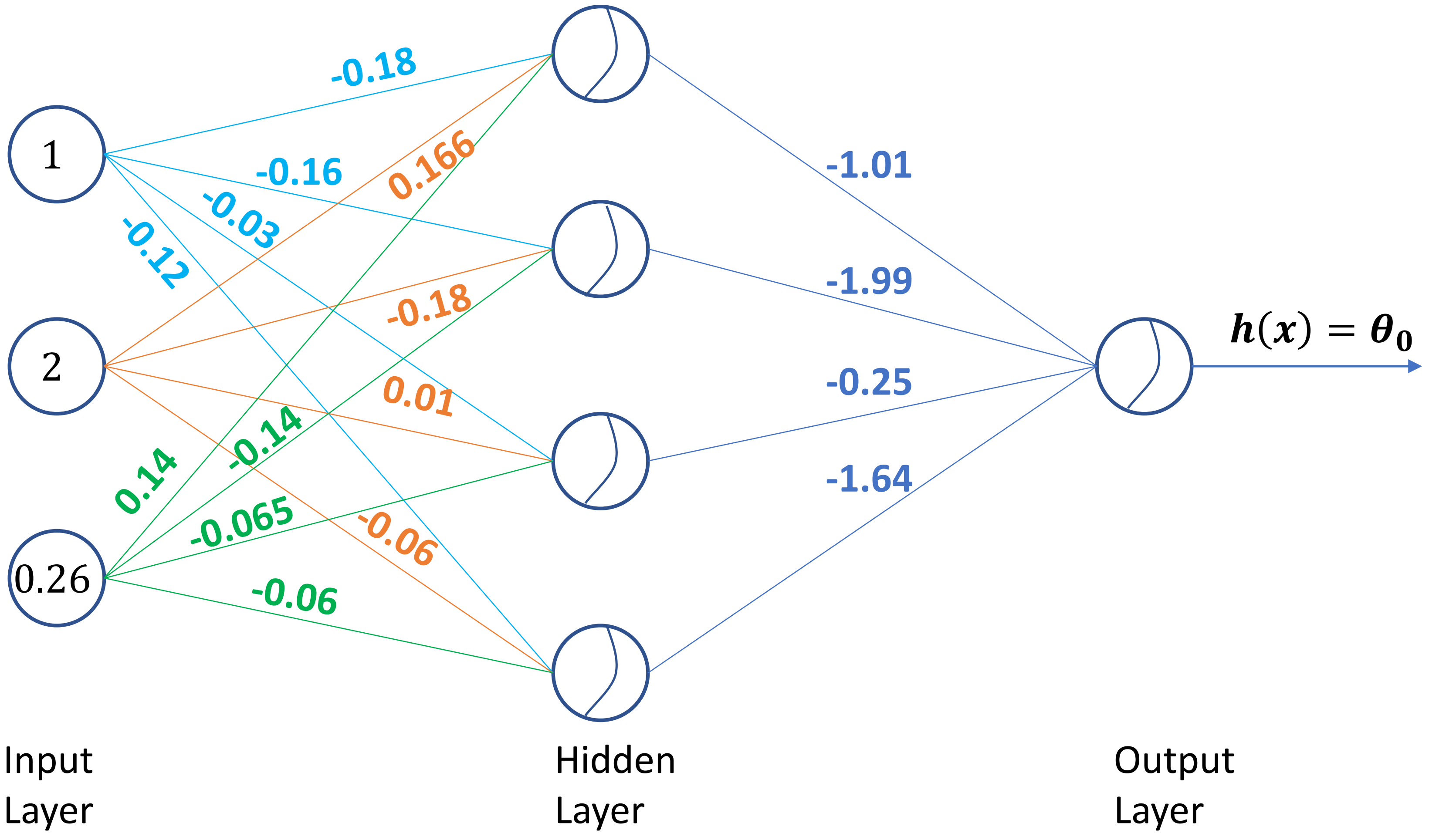


$$X_2' \rightarrow W_3X_3 + W_4X_4$$

$$\frac{\partial f}{\partial W_3} = ?$$

$$\frac{\partial f}{\partial W_3} = \frac{\partial f}{\partial X_2} \times \frac{\partial X_2}{\partial X_2'} \times \frac{\partial X_2'}{\partial W_3} = -1 \times \sigma' \times X_3 = -\sigma' X_3$$

Example: Learning a Neural Network



Illustrated Example: https://hmkcode.com/ai/backpropagation-step-by-step/

Learning the Convolution filter by backpropagation:

Remember the convolution step:

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

Input

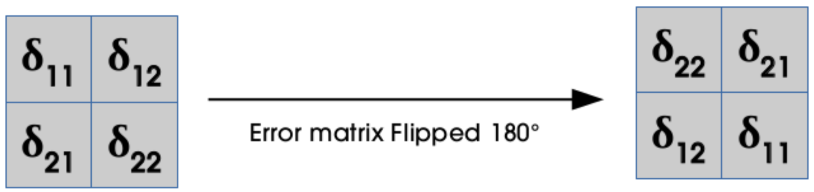
W_{11}	W_{12}
W_{21}	W_{22}

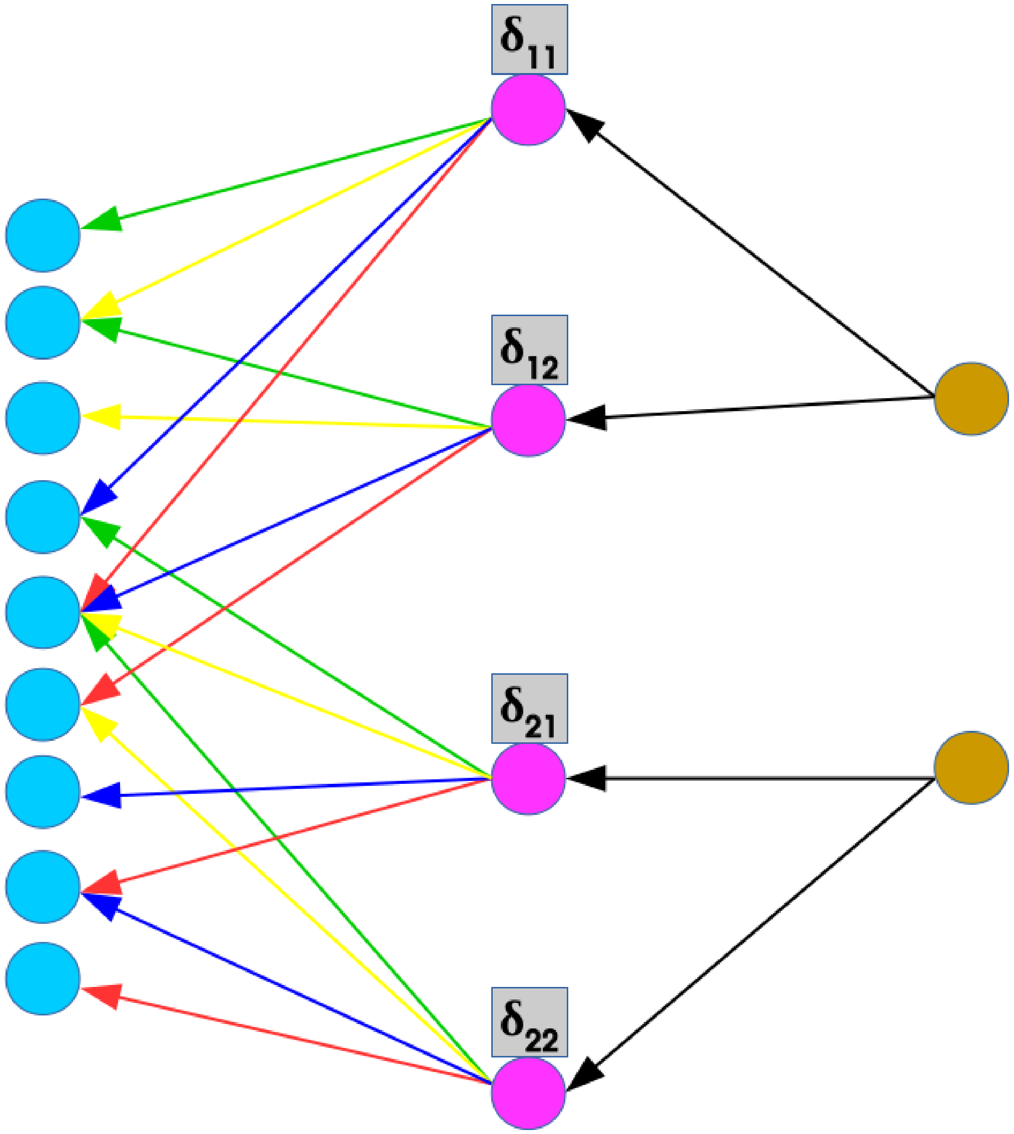
Filter

Output

https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Learning the Convolution filter by backpropagation:





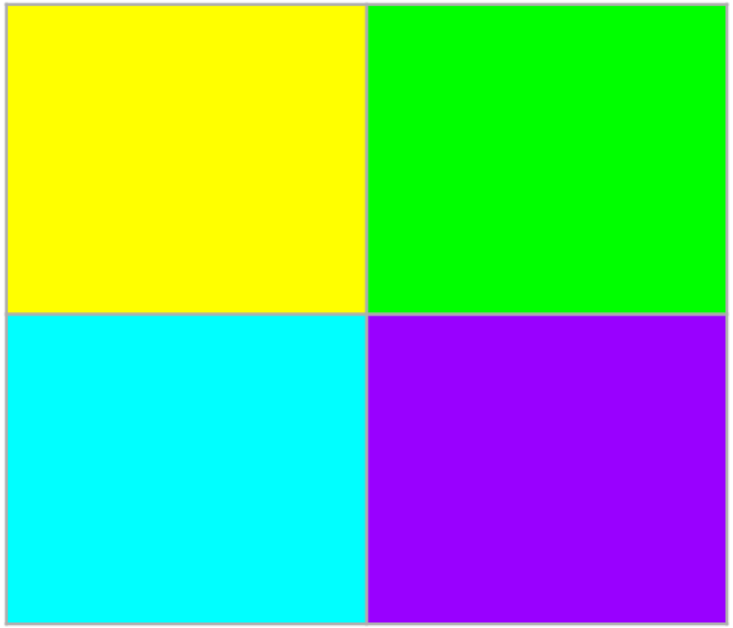
(For detailed derivation check blackboard)

https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Learning the Convolution filter by backpropagation:

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

Input



Filter

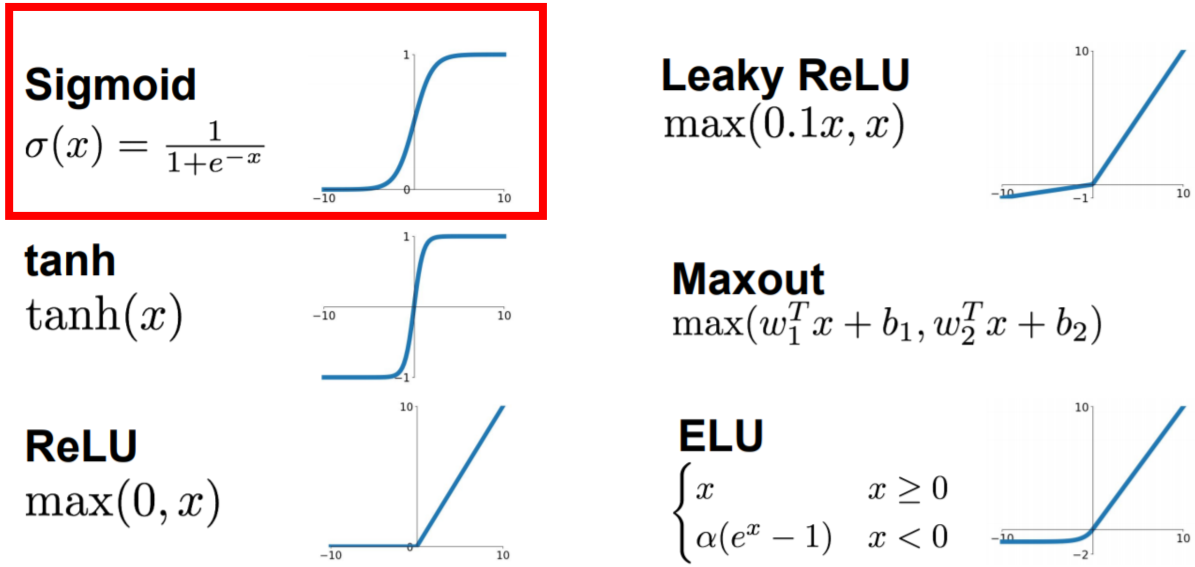
∂h_{11}	∂h_{12}
∂h_{21}	∂h_{22}

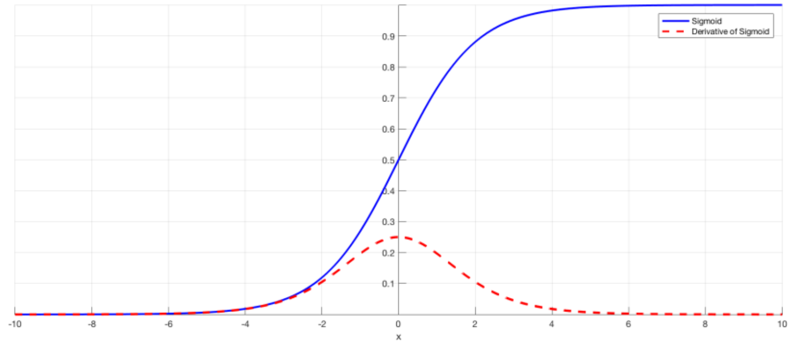
Error

(For detailed derivation check blackboard)

https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Selection of Activation Function

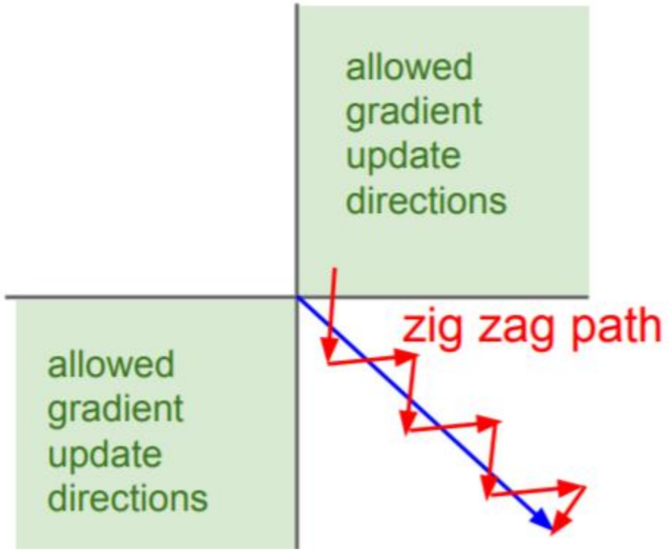




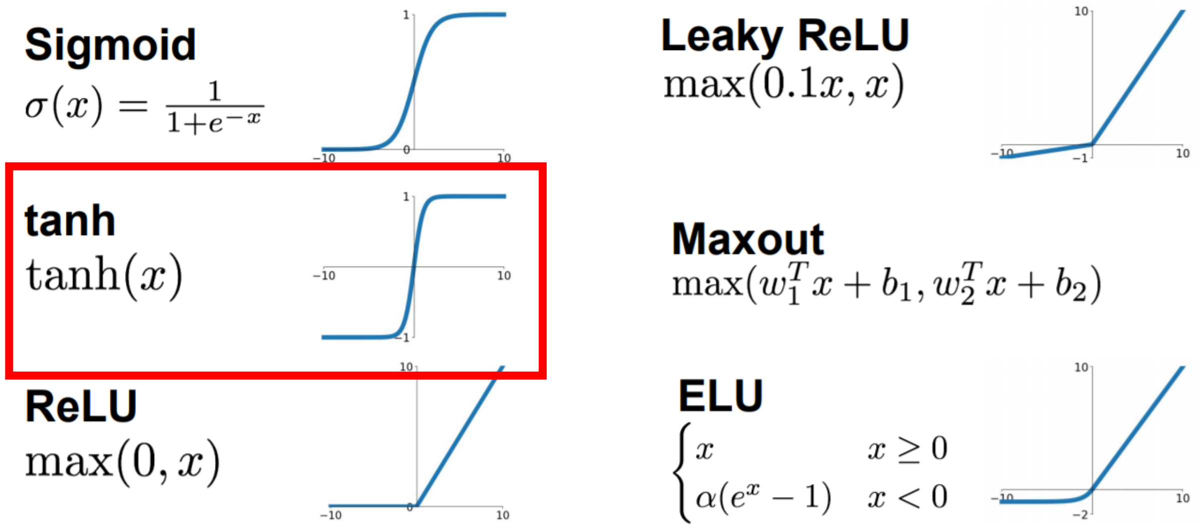
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

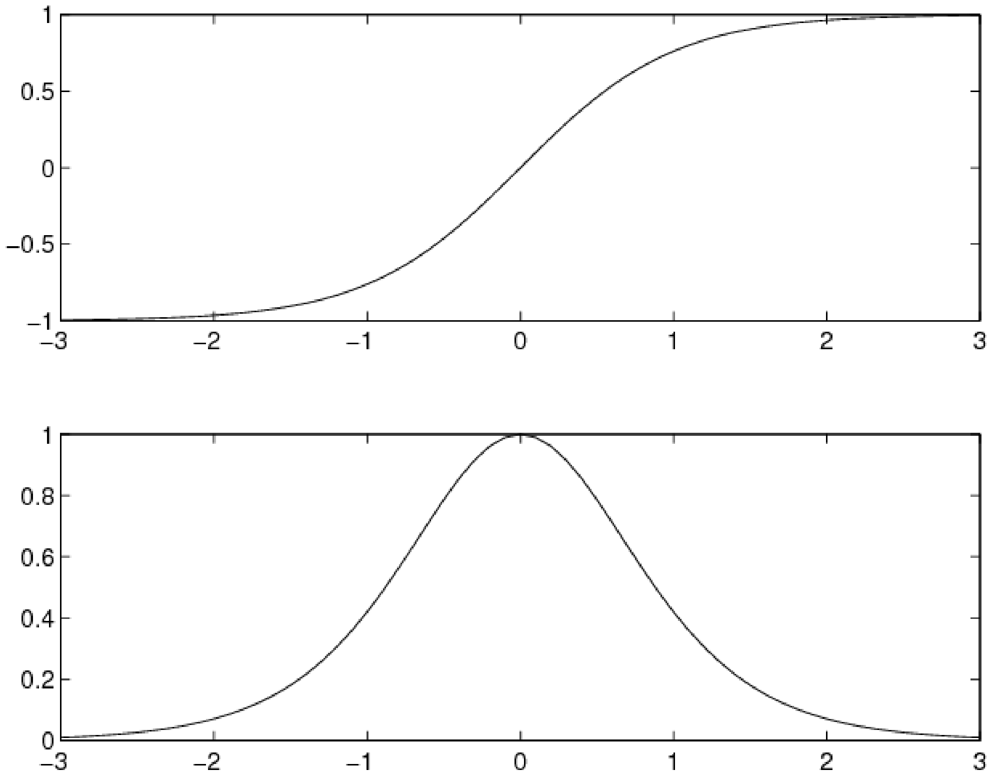
- Converted range: [0 1]

- Saturated activation causes gradients to vanish (over the chain of multiplication)
- Outputs are not zero centered
- Calculating "exponential" is computationally expensive.



Selection of Activation Function





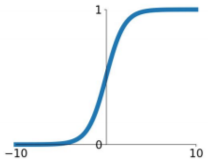
$\tanh(x)$

- Converted range: [-1 1]
- Zero Centered

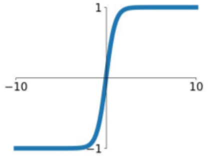
- Saturated activation causes gradients to vanish (over the chain of multiplication)

Selection of Activation Function

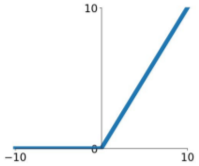
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



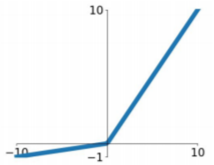
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

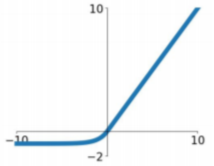


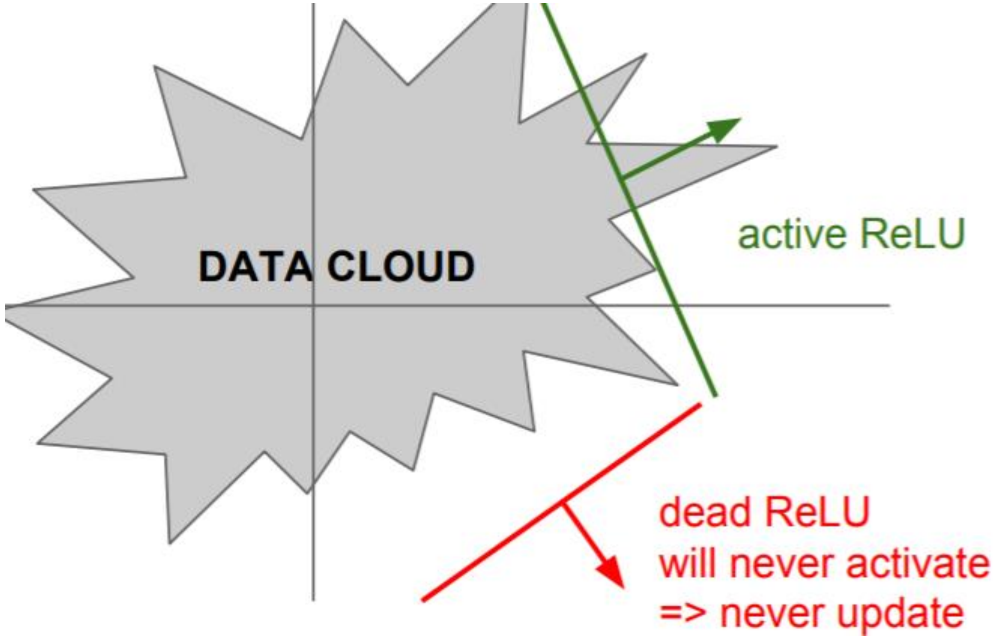
Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



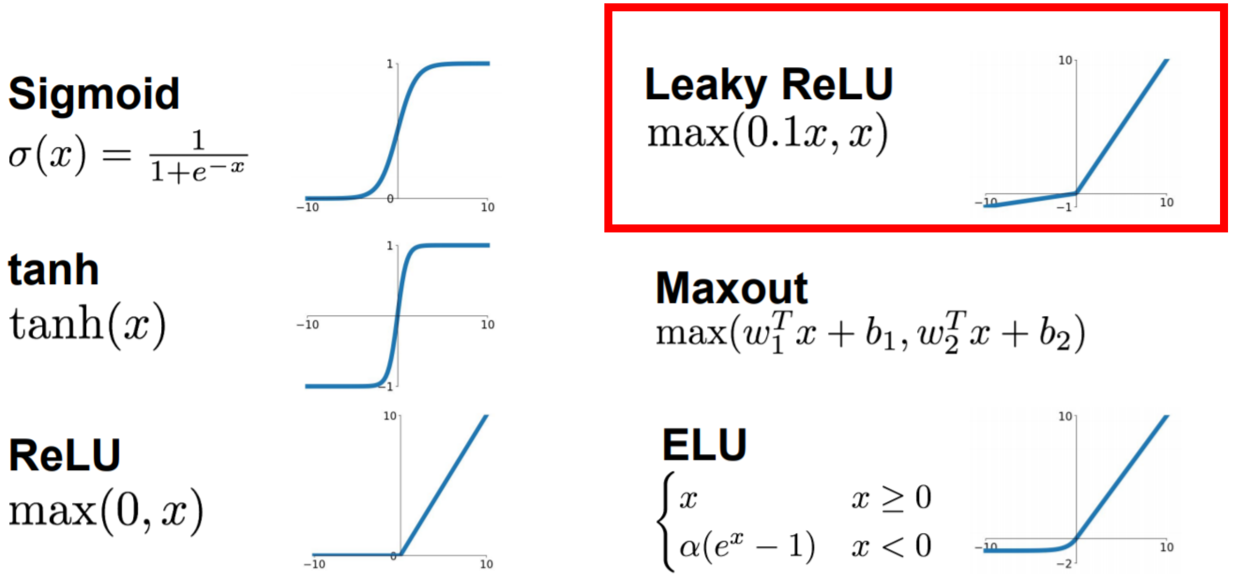


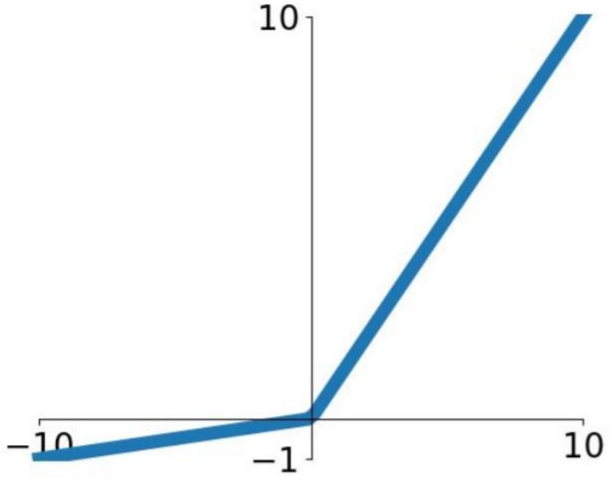
$$f(x) = \max(0, x)$$

- Doesn't Saturate (+ve Region only)
- Converges faster than the previous two

- Not zero centered
- Neurons with negative values will never activate!

Selection of Activation Function



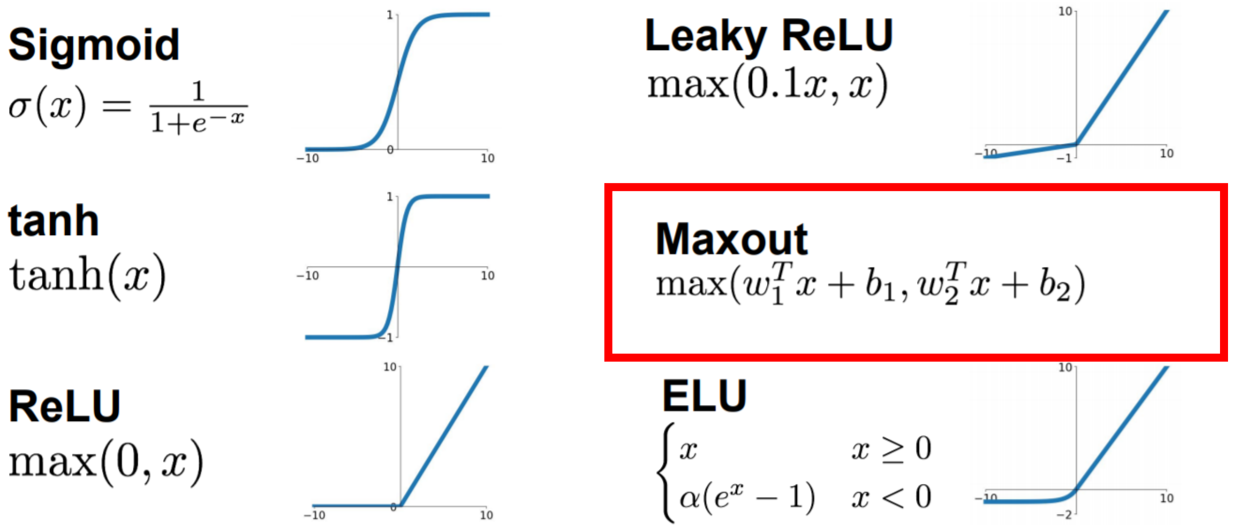


$f(x) = \max(0.01x, x)$

- Doesn’t Saturate
- Computationally efficient
- Converges faster than the previous two

- Not zero centered
- Neurons with negative values will never activate!

Selection of Activation Function



$f(x)$
$= \max(w_1^T x + b_1, w_2^T x + b_2)$

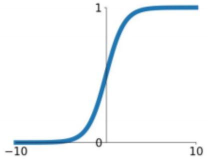
- Doesn't saturate the activation.
- Generalizes ReLU and Leaky ReLU.

- Increased number of parameters!

Selection of Activation Function

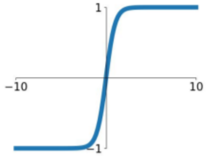
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



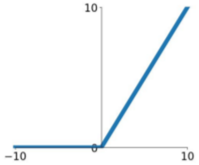
tanh

$\tanh(x)$



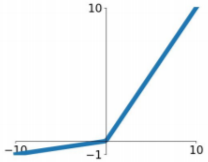
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

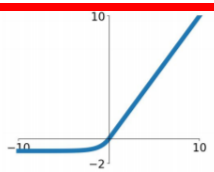


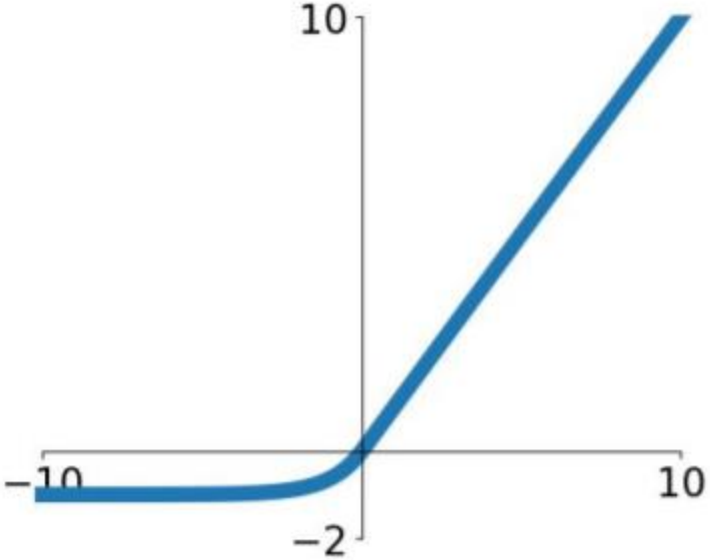
Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$





$$f(x) = \begin{cases} x \ \ if\ x > 0 \\ \alpha\ (\exp(x) - 1\ \ if\ x \leq 0 \end{cases}$$

- Everything ReLU
- Closer to Zero mean
- Adds some robustness to noise.

- Calculating “exponential” is computationally expensive

Few more things

- Regularization
  - Normalization / Batch Normalization
  - Dropout
- Optimizers
  - SGD
  - SGD + Momentum
  - AdaGrad
  - RMSProp
  - Adam
- Hyper-parameter optimization
  - Grid Search
  - Bayesian Optimization
  - DoE
- Transfer Learning