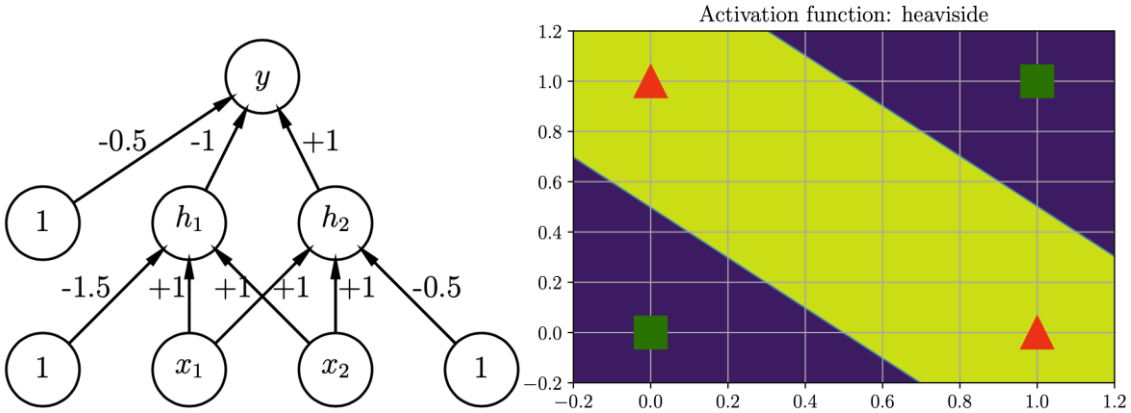
Neural Network

Reference: Yangfeng Ji, Neural Network, lecture notes

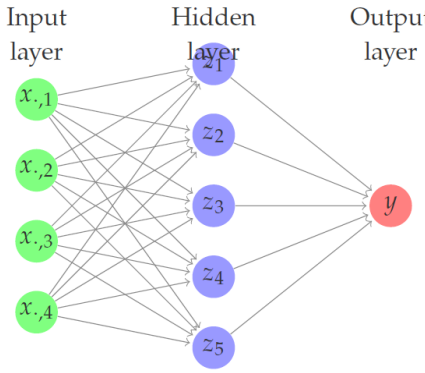
https://aman.ai/cs229/neural-networks/

1 Overview

- **Motivation**: simple Multi-Layer Perceptron (MLP) (linear classification) can achieve complicated non-linear classification.



- 2-layer Neural network with sigmoid active function



$$z = \sigma(\mathbf{W}^{(1)}x) \quad (13)$$

$$P(y = +1 \mid x) = \sigma((w^{(o)})^{\mathsf{T}}z) \quad (14)$$

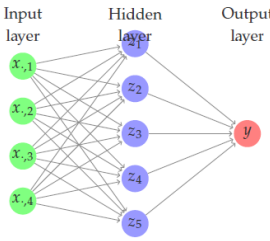
where $\mathbf{W}^{(1)} \in \mathbb{R}^{K\times d}$ and $\mathbf{w}^{(o)} \in \mathbb{R}^{K}$

- Parameters:
  a) Input x: d-dimension
  b) The hidden layer contains k units
  c) W^1 and w^0: the coefficient we want to learn.

2 Hypothesis space

- **Hypothesis space**: defined by the architecture of the network

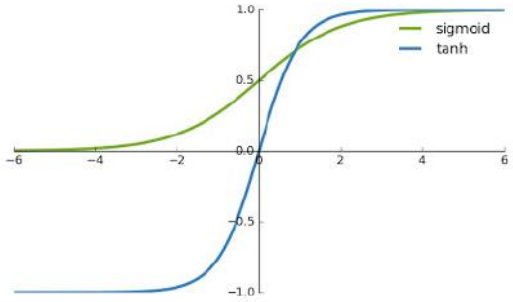
- the nodes in the network,
- the connections in the network, and
- the activation function (e.g., σ)

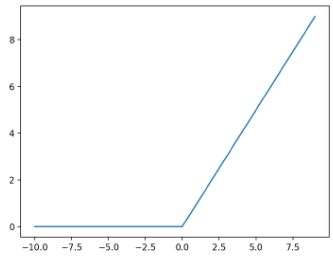


- **Active functions**: mainly three different active functions, i.e., sigmoid, ReLU, and tanh

https://aman.ai/cs229/neural-networks/

$$g(z) = \frac{1}{1+e^{-z}} \quad \text{(sigmoid)}$$
$$g(z) = \max(z, 0) \quad \text{(ReLU)}$$
$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \quad \text{(tanh)}$$





(c) ReLU function [Jarrett et al., 2009]

3 Question: how to choose suitable active functions?

https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

- ReLU: the most popular common function used for hidden layers.
    a) Advantages: i) it is less susceptible to vanishing gradients in back-propagation.

ii) Computational simplicity by max().
iii) Sparsity since the output can be 0
iv) Linear behavior

b) Disadvantages: i) may suffer from saturated or "dead" units.
c) Application tips: i) Use ReLU with MLPs, CNNs, but Probably Not RNNs.
ii) a good practice to use a "He Normal" or "He Uniform" weight initialization and scale input data to the range [0,1] (normalize) prior to training.
iii) Add weight penalty (L1-norm or L2-norm).

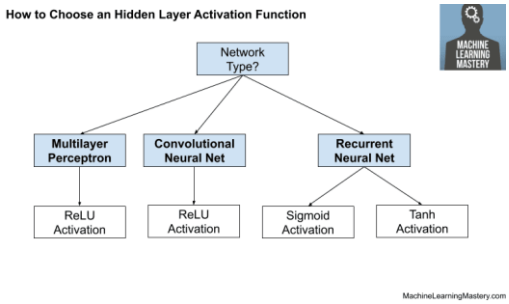
https://medium.com/analytics-vidhya/activation-functions-why-tanh-outperforms-logistic-sigmoid-3f26469ac0d1#:~:text=Both%20sigmoid%20and%20tanh%20are,lies%20between%201%20and%20%2D1.

- Sigmoid with output [0,1]: S-shape, scale the input data into the range of $[0,1]$.
- Tanh function with output [-1,1]: S-shape, scale the input data into the range of $[-1,1]$. It outperforms the Sigmoid function.

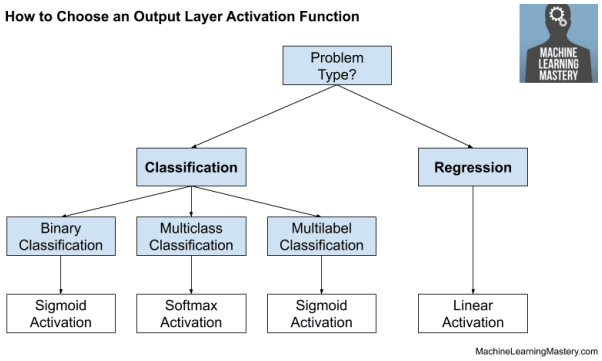
Conclusion:

1. tanh and logistic sigmoid are the most popular activation functions in the '90s but because of their Vanishing gradient problem and sometimes Exploding gradient problem (because of weights), they aren't mostly used now.
2. These days ***Relu*** activation function is widely used. Even though, it sometimes gets into vanishing gradient problems, variants of Relu help solve such cases.
3. tanh is preferred to sigmoid for faster convergence BUT again, this might change based on data. Data will also play an important role in deciding which activation function is best to choose.

- Hidden layers



- Output layers



https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

4 Loss function

- **Cross entropy**: of the prediction distribution p and the empirical distribution q is

$$\begin{aligned} H(q,p) &= -q(Y=+1 \mid \boldsymbol{x}) \log p(Y=+1 \mid \boldsymbol{x}) \\ &\quad -q(Y=-1 \mid \boldsymbol{x}) \log p(Y=-1 \mid \boldsymbol{x}) \end{aligned}$$

- Since q is defined with a Delta function, Depending on y, we have

$$H(q,p) = \begin{cases} -\log p(Y=+1 \mid \boldsymbol{x}) & Y=+1 \\ -\log p(Y=-1 \mid \boldsymbol{x}) & Y=-1 \end{cases}$$

  Pay attention to the situation that $y \in [-1,+1]$.

- The **empirical risk minimization (ERM)** of a dataset $S = \{(x_i, y_i)\}_1^m$ is

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{m} \log p(y_i \mid \boldsymbol{x}_i)$$

- **Solution**: Gradient descendant method (back-propagation algorithm)

5 Initialize parameters

- Initializing weights: **uniform**, **normal** distribution with small values $N(0,0.1)$, or **Xavier/He initialization**

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

  - where $n^{[\ell]}$ is the number of neurons in layer ℓ. This acts as a mini-normalization technique. For a single layer, consider the variance of the input to the layer as $\sigma^{(in)}$ and the variance of the output (i.e., activations) of a layer to be $\sigma^{(out)}$. Xavier/He initialization encourages $\sigma^{(in)}$ to be similar to $\sigma^{(out)}$

- Input data: scale to the range $[-1,1]$ or $[0,1]$ based on the active function you have selected.

6 Back-propagation algorithm

- Based on the Gradient descendant method, we update parameters

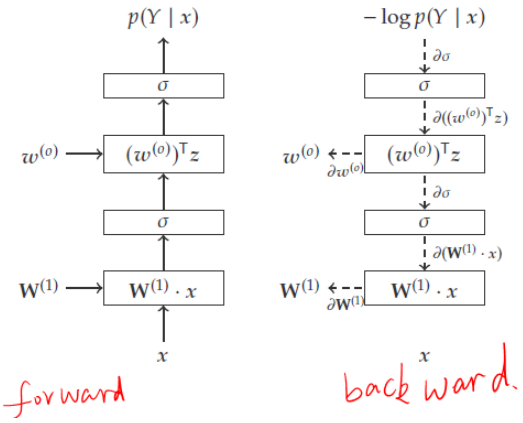
$$\theta^{(\text{new})} \leftarrow \theta^{(\text{old})} - \eta \cdot \left.\frac{\partial L(\theta)}{\partial \theta}\right|_{\theta=\theta^{(\text{old})}}$$

- **Calculate gradient**: based on the chain rule, we obtain the gradient.

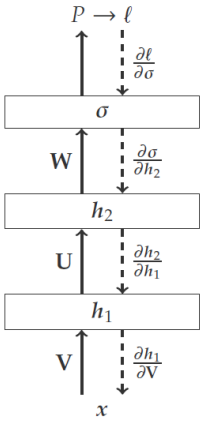
$$\begin{aligned}\frac{\partial L(\theta)}{\partial w^{(o)}} &= -\frac{\partial \log \sigma(\cdot)}{\partial \sigma(\cdot)} \cdot \frac{\partial \sigma\left((w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x)\right)}{\partial (w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x)} \cdot \frac{\partial (w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x)}{\partial w^{(o)}} \\ &= -\left\{1 - \sigma\left((w^{(o)})^{\mathsf{T}} \sigma(\mathbf{W}^{(1)} x)\right)\right\} \cdot \sigma(\mathbf{W}^{(1)} x)\end{aligned}$$

- It contains **4 different units** in the back-propagation network

- $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
- $\frac{\partial a^{\mathsf{T}} x}{\partial x} = a$
- $\frac{\partial \log(x)}{\partial x} = \frac{1}{x}$
- $\frac{\partial \mathbf{W} x}{\partial x} = \begin{bmatrix} x^{\mathsf{T}} \\ \vdots \\ x^{\mathsf{T}} \end{bmatrix}$



$$\frac{\partial \ell}{\partial \mathbf{V}} = \frac{\partial \ell}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial \mathbf{V}}$$



- Before ReLU active function, cumbersome training schemes is applied to achieve layer-wise training.

5 Question: how to improve the model accuracy

At this point, we have initialized the parameters and have optimized the parameters. Suppose we evaluate the trained model and observe that it achieves 96% accuracy on the training set but only 64% on the testing set.

Some solutions include collecting more data, employing regularization, or making the model shallower.

https://aman.ai/cs229/neural-networks/