ONLY GOD

1402-2023

mansoori@shirazu.ac.ir

Neural Network & Deep Learning

Deep MLP

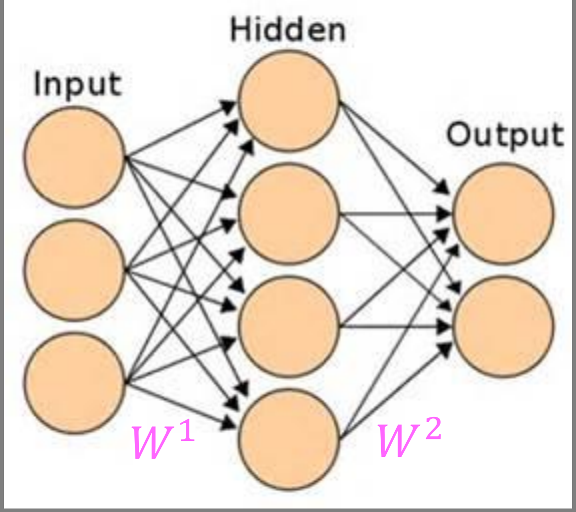
CSE & IT Department
School of ECE
Shiraz University

Why need deep architectures?

- Theoretician's dilemma:
  - We can approximate any function with shallow architecture
  - Why would we need deep ones?

- An NN with single hidden layer is universal approximator:

$$\vec{y} = f^2(W^2 . f^1(W^1 . \vec{x}))$$

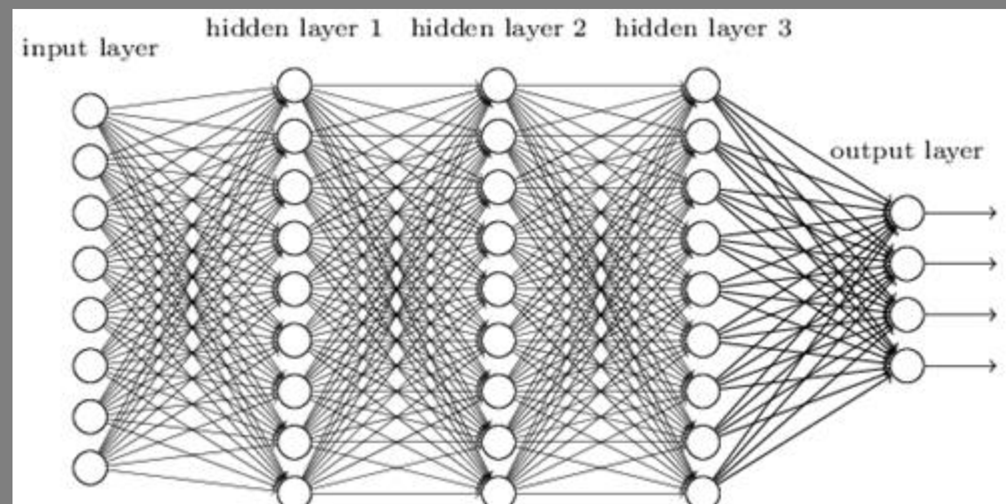


Deep MLP

mansoori@shirazu.ac.ir

- A deep network with many $(K-1)$ hidden layers

$$\vec{y} = f^{K}(W^{K}.f^{K-1}(W^{K-1}.f^{K-2}(\ldots f^{1}(W^{1}.\vec{x})\ldots)))$$



- Deep networks are more efficient for representing certain classes of functions, particularly those involved in visual recognition

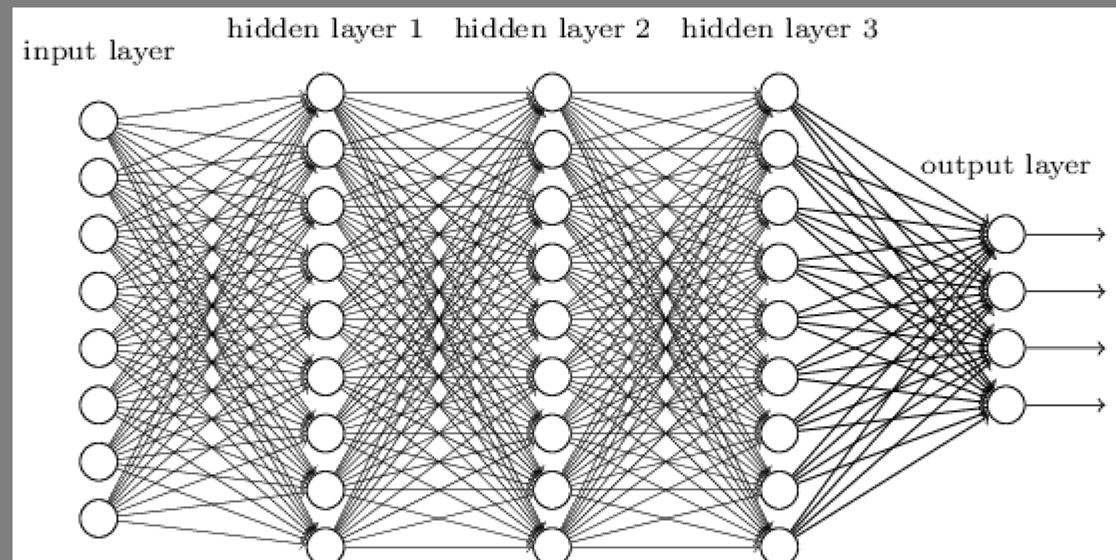
- They can represent more complex functions with less hardware

Deep MLP

mansoori@shirazu.ac.ir

Input data is a column matrix ($d \times n$) for n data samples with d dimensions

$$X = \begin{bmatrix} | & | & \cdots & | \end{bmatrix}_{d \times n}$$



$d{=}p_0$ p_1 p_2 p_3 p_4

Dimension of each layer

- Layer l of network has a weight matrix W^l
- It is a $p_l \times p_{l-1}$ dimensional row matrix
- ith row of W^l is the weights of ith neuron of layer l

$$W^l = \begin{bmatrix} \text{---} \\ \text{---} \\ \vdots \\ \text{---} \end{bmatrix}_{p_l \times p_{l-1}}$$

Bias vector of layer l

$$b^l = \begin{bmatrix} | \end{bmatrix}_{p_l \times 1}$$

Outputs of layer l

$$A^l = \begin{bmatrix} | & | & \cdots & | \end{bmatrix}_{p_l \times n}$$

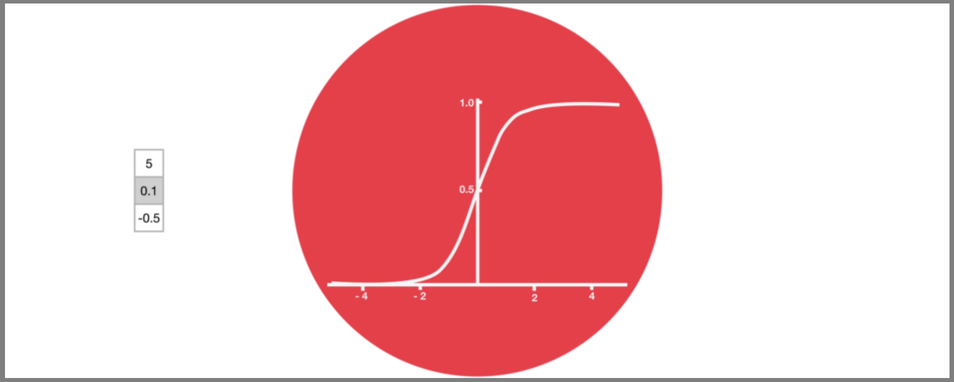
Deep MLP Training

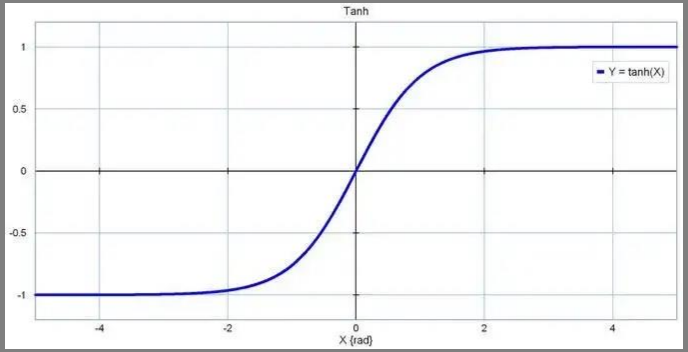
mansoori@shirazu.ac.ir

- Main learning algorithm is Back-propagation
  - Randomly initialize the weights and biases
  - Train it supervisedly, by applying gradient descent method

Binary/Bipolar sigmoid activation function:

$$f(x) = \mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}} \qquad f(x) = \tanh(x) = \frac{1-e^{-x}}{1+e^{-x}}$$

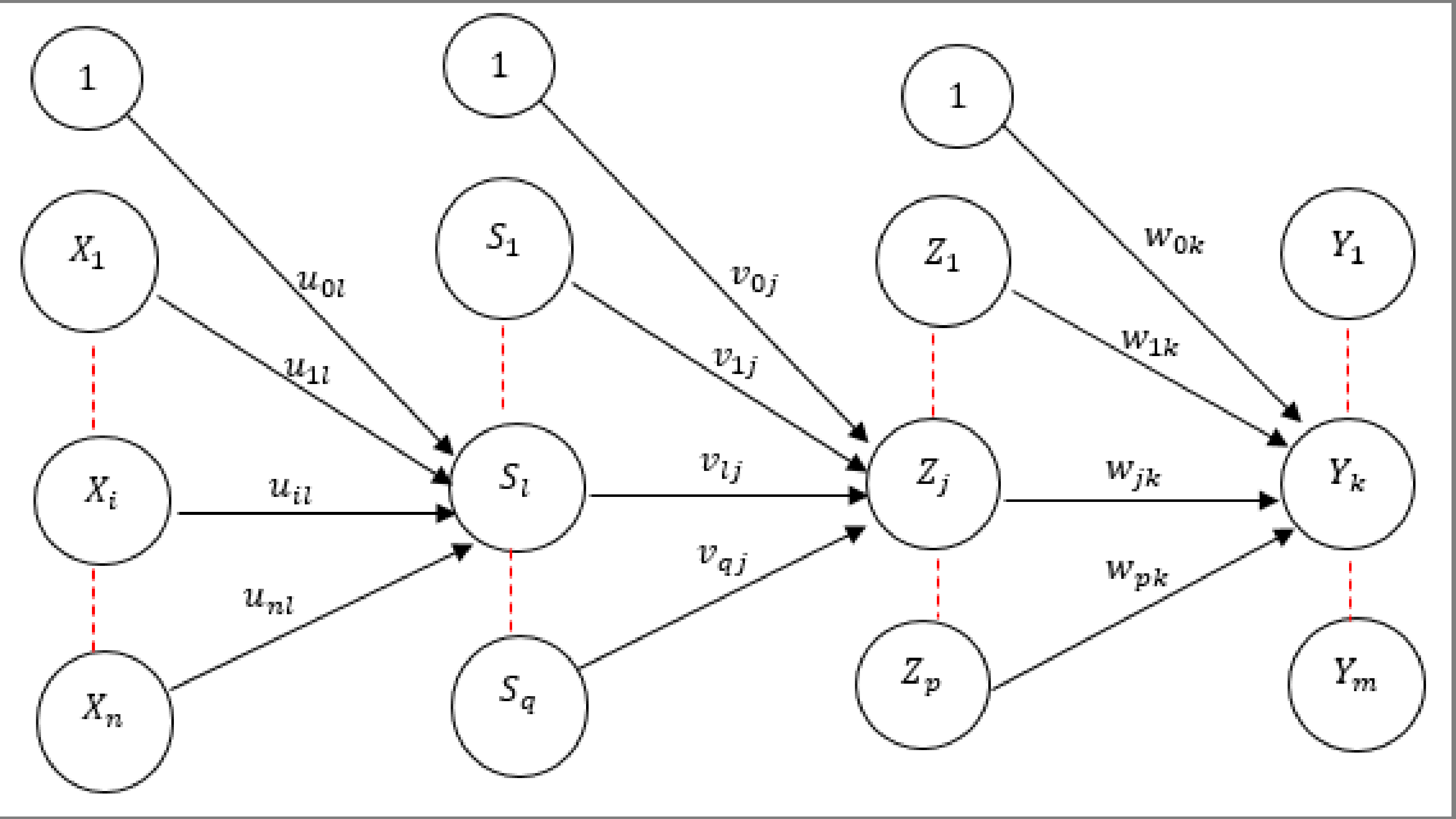




Commonly used in MLPs

Deep MLP Training

mansoori@shirazu.ac.ir



$U = \{u_{il}\}\ (i = 0, \dots, n\ , \qquad l = 1, \dots, q), \qquad u_{0l}$: biases

$V = \{v_{lj}\}\ (l = 0, \dots, q\ , \qquad j = 1, \dots, p), \qquad v_{0j}$: biases

$W = \{w_{jk}\}\ (j = 0, \dots, p\ , \qquad k = 1, \dots, m), \qquad w_{0k}$: biases

Deep MLP Training

mansoori@shirazu.ac.ir

$$s\_in_l = u_{0l} + \sum_{i=1}^{n} x_i\, u_{il} \;, \quad s_l = f^{\mathrm{H}_1}(s\_in_l) \;, \quad (l = 1, \dots, q)$$

$$z\_in_j = v_{0j} + \sum_{l=1}^{q} s_l\, v_{lj} \;, \quad z_j = f^{\mathrm{H}_2}\left(z\_in_j\right) \;, \quad (j = 1, \dots, p)$$

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j\, w_{jk} \;, \quad y_k = f^{\mathrm{O}}(y\_in_k) \;, \quad (k = 1, \dots, m)$$

$$\delta_k^{\mathrm{O}} = f^{\mathrm{O}\prime}(y\_in_k)\{-(t_k - y_k)\}, \qquad (k = 1, \dots, m)$$

$$\delta_j^{\mathrm{H}_2} = f^{\mathrm{H}_2\prime}\left(z\_in_j\right)\left(\sum_{k=1}^{m} \delta_k^{\mathrm{O}}\, w_{jk}\right), \;\; (j = 1, \dots, p)$$

$$\delta_l^{\mathrm{H}_1} = f^{\mathrm{H}_1\prime}(s\_in_l)\left(\sum_{j=1}^{p} \delta_j^{\mathrm{H}_2}\, v_{lj}\right), \;\; (l = 1, \dots, q)$$

$$\Delta w_{jk} = -\alpha\, \delta_k^{\mathrm{O}}\, z_j \;, \qquad \Delta w_{0k} = -\alpha\, \delta_k^{\mathrm{O}}, \qquad (j = 1, \dots, p\;;\;\; k = 1, \dots, m)$$

$$\Delta v_{lj} = -\alpha\, \delta_j^{\mathrm{H}_2}\, s_l \;, \qquad \Delta v_{0j} = -\alpha\, \delta_j^{\mathrm{H}_2}, \qquad (l = 1, \dots, q\;;\;\; j = 1, \dots, p)$$

$$\Delta u_{il} = -\alpha\, \delta_l^{\mathrm{H}_1}\, x_i \;, \qquad \Delta u_{0l} = -\alpha\, \delta_l^{\mathrm{H}_1}, \qquad (i = 1, \dots, n\;;\;\; l = 1, \dots, q)$$

mansoori@shirazu.ac.ir

Back-propagation in Practice (to avoid some difficulties)

1. Use reLU non-linearity
   - Bipolar and logistic sigmoid are falling out of favor

2. Use stochastic gradient descent on mini-batches
   - Mini-batch: divide data into k smaller fractions and pass them simultaneously to network during the learning phase

3. Shuffle the training samples

4. Normalize the input data to zero mean and unit variance

5. Schedule to decrease the learning rate

Back-propagation in Practice (to avoid some difficulties)

mansoori@shirazu.ac.ir

6. Use L1 or L2 regularization on the weights
   - The combination of L1 and L2 may be used
   - It is best to turn it on after a couple of epochs

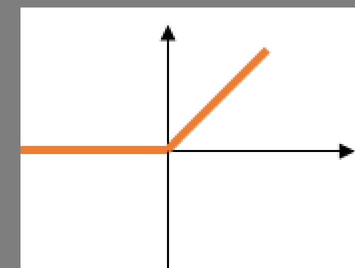
7. Use dropout for regularization
   - A technique for reducing over-fitting in NNs
   - Turning off the output of some neurons (e.g. 50% each time) in each layer
   - Hidden neurons will not co-adapt to other neurons and the model will be more general

Rectifier Linear Unit (reLU)

mansoori@shirazu.ac.ir

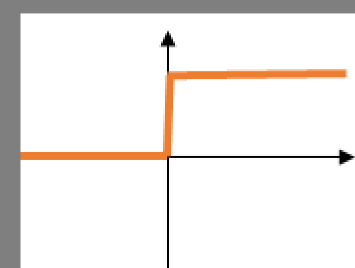
- reLU is a modern activation function which is popular in deep NNs:

$$\mathrm{reLU}(x) = \max(x, 0)$$



- Its derivative can be defined by step function:

$$\frac{\partial}{\partial x}\mathrm{reLU}(x) = \mathrm{step}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

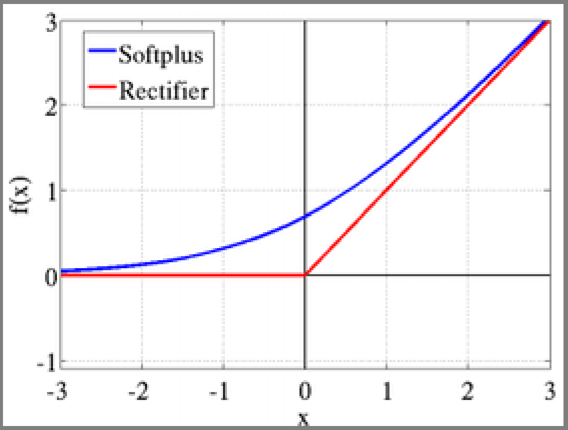


Soft Approximation of reLU

mansoori@shirazu.ac.ir

- A smooth approximation to reLU is softplus function:

$$\text{softplus}(x) = \ln(1 + e^{x})$$



- Its derivative is binary sigmoid function:

$$\frac{\partial}{\partial x}\text{softplus}(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$