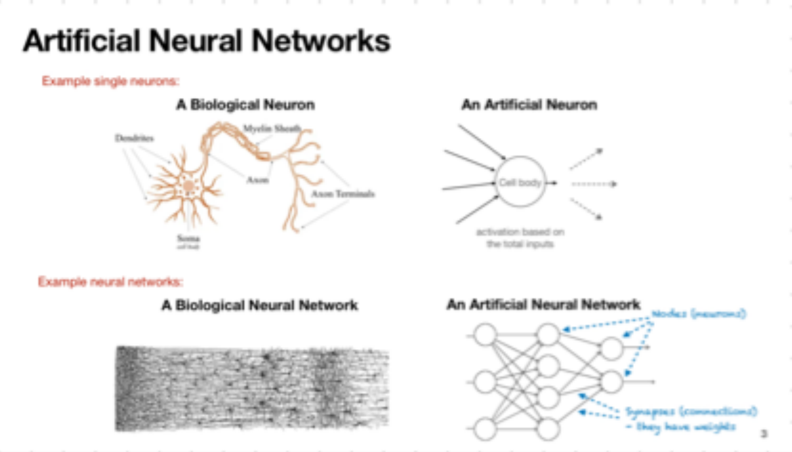
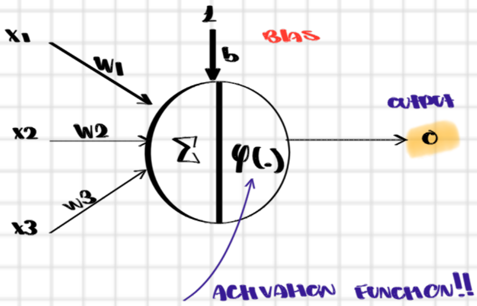
Lecture 20 – 12/11/24



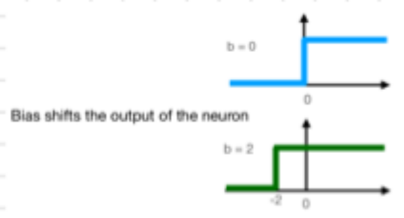
- Highly inspired by human & animals
- Acquired popularity as deep learning!

Artificial Neuron



$$o = \varphi\left([x_1 x_2 \dots x_p]\begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_p \end{bmatrix} + b\right)$$

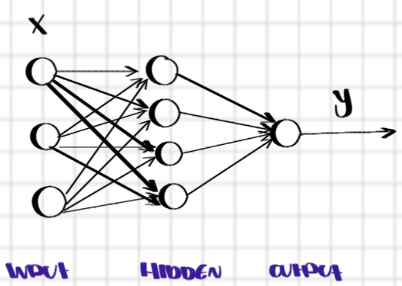
$$o = \varphi(x \cdot w + b)$$



- Currently the most popular activation function is ReLU (rectified linear unit)

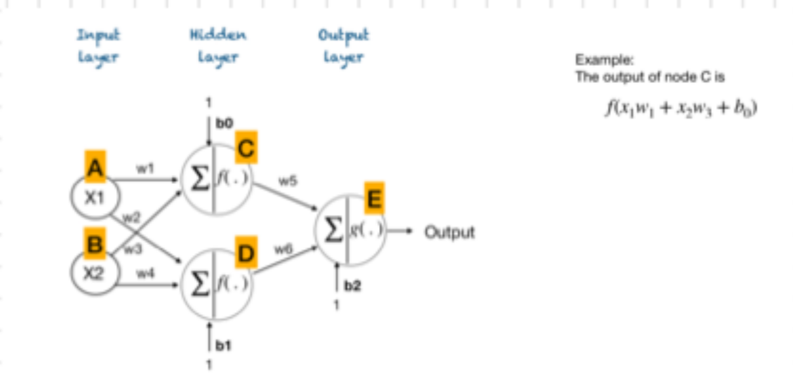
Feedforward Artificial Neural Network

- A network of connected neurons with several layers:
  - information flows from first layer toward last one!



could be more than one!!

- Non-linear activation functions allow neural networks to perform more complex tasks!
- Each neuron is a non-linear trasformation of inputs!

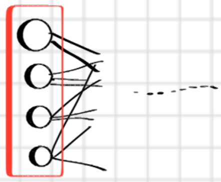


- Example of a 2-layer feedforward
- We can see how X1 feeds both C & D
- Example output of node D:

$$f(x_2 w_4 + x_1 w_2 + b_1)$$

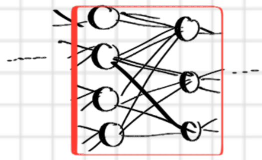
Input Layer

- Neurons in input directly output the input values
- No ~~activation function~~



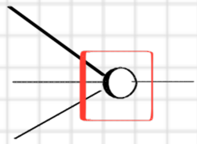
Hidden Layers

- must use a non-linear activation function
  - otherwise network = just linear function!
- Some famous ones:
  - Sigmoid
  - tanh
  - ReLU



Output Layer

- defines range of the outputs
  - Regression → Linear func
  - Binary classification → Sigmoid
  - Multi-class classification → Softmax



Forward Pass

- Calculating the lost / cost
- $\forall$ sample (x, y):
  - Present x at the input layer
  - Compute $z^{(l)}$ & $a^{(l)}$ $\forall$ layers
  - Compute output of the network & the value of the loss function for sample
- If u sum up the loss of all the samples, we get cost (total loss)

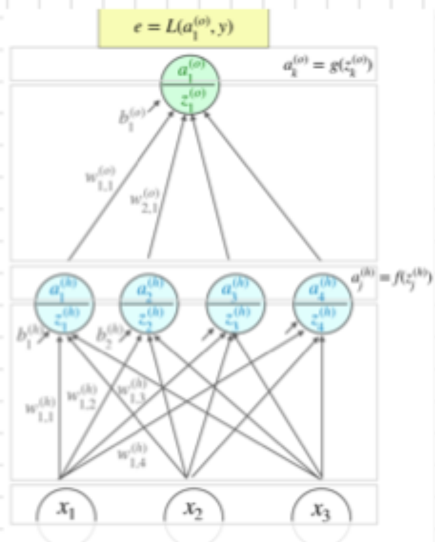
i.e.:

$$z_1^{(h)} = x_1 W_{1,1}^{(h)} + x_2 W_{2,1}^{(h)} + x_3 W_{3,1}^{(h)} + b_1^{(h)} \qquad a_1^{(h)} = f(z_1^{(h)})$$
$$z_2^{(h)} = x_1 W_{1,2}^{(h)} + x_2 W_{2,2}^{(h)} + x_3 W_{3,2}^{(h)} + b_2^{(h)}$$
$$z_3^{(h)} = x_1 W_{1,3}^{(h)} + x_2 W_{2,3}^{(h)} + x_3 W_{3,3}^{(h)} + b_3^{(h)}$$
$$z_4^{(h)} = x_1 W_{1,4}^{(h)} + x_2 W_{2,4}^{(h)} + x_3 W_{3,4}^{(h)} + b_4^{(h)} \qquad a_4^{(h)} = f(z_4^{(h)})$$

now this are hidden layer input

$$z_1^{(o)} = a_1^{(h)} W_{1,1}^{(o)} + a_2^{(h)} W_{2,1}^{(o)} + a_3^{(h)} W_{3,1}^{(o)} + a_4^{(h)} W_{4,1}^{(o)} + b_1^{(o)}$$

$$e = L(a_1^{(o)}, y)$$



All of the above is done for a single sample

Training ANN

- We have to train all the weights (including biases)
- We can use some variants of Gradient Descent
  - Initialize W & b randomly ∀ layers
  - Choose a learning rate α.
  - Repeat until convergence
- Take Gradient Step & update parameters:
  - Compute partial derivative of the cost function wrt to weights & biases
  - Update weights & biases

$$\boxed{W_{(t+1)} = W_{(t)} - \alpha \nabla L} \qquad \boxed{b_{(t+1)} = b_{(t)} - \alpha \nabla L}$$

- How to find partial derivatives considering multiple layers?

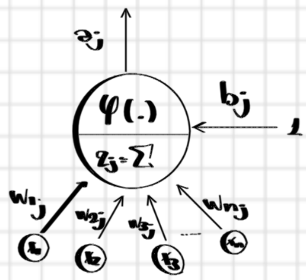
  The answer is: Back-Propagation:

  - It is a backward pass that calculates partial derivatives starting from output moving towards input

    ↳ calculate locally & combine using chain rule

Simple Neuron

- Loss function: $L(y, a_j) = \frac{1}{2}(y - a_j)^2$
- First use a data point:
  - Do forward pass & get a_j
- Then do back-propagation



A simple Neuron

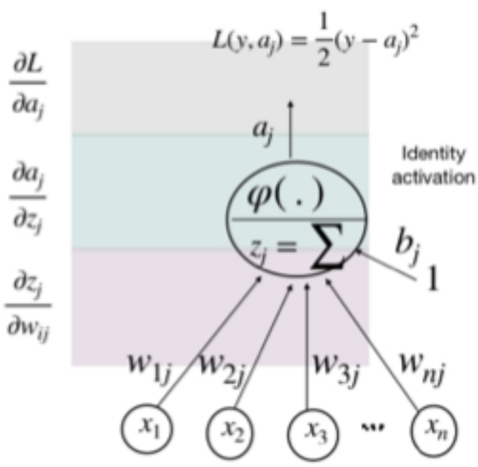
Back-propagation

Depends on your loss function: $\frac{\partial L}{\partial a_j} = a_j - y$

Depends on your activation function: $\frac{\partial a_j}{\partial z_j} = 1$

Always holds (doesn't depend to anything):

$$\frac{\partial z_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj}x_k + b_j\right) = \frac{\partial}{\partial w_{ij}} w_{ij}x_i = x_i \qquad \frac{\partial z_j}{\partial b_j} = 1$$



$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot x_i = (a_j - y) \cdot x_i$$

$$\frac{\partial L}{\partial b_j} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_j} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} = (a_j - y)$$

4

For the training: same steps explained above!