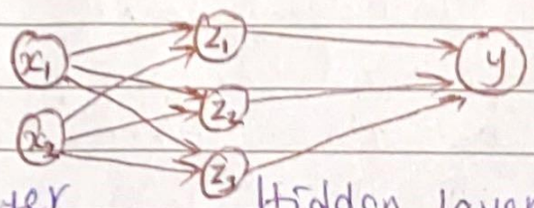
Neural networks

- A neural network is a learning machine that can be represented as a network of neurons.
  - can be arbitrarily expressive
    - (by varying their size and structure)
  - and used in both the classification & regression setups.

Example:

- A simple neural network for regression.
  - The associated learning machine is $F_{231} : \mathbb{R}^2 \times \Lambda \to \mathbb{R}$, with $\Lambda \in \mathbb{R}^2$



→ circles & lines represent (visible & hidden) neurons and their connections.
→ the components of x, z, and y are the values of the neurons.
→ $x = [x_1, x_2]^T$ is the object attribute
→ y is the (continuous) label
→ $z = [z_1, z_2, z_3]^T$ is the value of the hidden neurons.

Note that: each neuron may have several inputs & outputs.

Neurons & connections

- Each neuron has n_{in} inputs associated with
  - $n_{in} + 1$ free parameters, $w_0, w_1, \ldots, w_{n_{in}}$ called weights
  - an activation function, $\phi : \mathbb{R} \to \mathbb{R}$ for

Given an input $z_{in} = [[z_{in}]_1, \ldots, [z_{in}]_{n_{in}}]^T \in \mathbb{R}^{n_{in}}$

- the neurons output is

$$z_{out} = \phi(w_0 + w^T z_{in}) = \phi\left(w_0 + \sum_{i=1}^{n_{in}} w_i z_i\right)$$

The output neuron

- Except for the output neuron,
  - ϕ can be any function
  - and the z_{out} may be sent to many other neurons.

- There are some restrictions on the choice of the activation function of the output neuron,
  - depending on the regression/ classification setup

  - Regression networks : $\phi : \mathbb{R} \to \mathbb{R}$
  - classification networks : $\phi : \mathbb{R} \to [0,1]$
    - to be interpreted as the probability of the object to belong to a given class
    - (as for logistic regression models)

Activation functions

- Let the neuron value, i.e. its weighted input, be $s = w_0 + w^T z_{in}$
- The output depends on the neuron activation function, e.g. you can have
  - Linear activation: $\phi(s) = s$
  - Step function : $\phi(s) = \theta(s) = \mathbb{1}[s > 0]$
  - Sigmoid : $\phi(s) = \sigma(s) = \frac{1}{1+e^{-s}}$
  - ReLU : $\phi(s) = \text{ReLU}(s) = \max(0, s)$
  - SoftPlus : $\phi(s) = \text{softplus}(s) = \log(1 + e^s)$

- θ & ReLU are non-differentiable functions & are often approximated using the σ & softplus.

Neural network modelling for data analysis

- You are given a labelled data set, $\mathcal{D}$ &
  - a data analysis task,

  e.g. predicting the price of a house, $Y \in \mathcal{Y}$,
  - given its attribute, $X \in \mathcal{X}$

- How to use a neural network to solve the task?

  You need to:

  1. Define the input and output spaces & a learning machine
     $F : \mathcal{X} \times \Lambda \rightarrow \mathcal{Y}$

  2. Train F, i.e. find

$$\hat{f}(X) = F(X, \hat{\lambda}), \quad \hat{\lambda} = \arg\min_{\lambda \in \Lambda} \sum_{(x,y) \in \mathcal{D}} J(y, F(x, \lambda))$$

  3. Evaluate the trained network on the known attribute of a new house, x_0
     - to predict its unknown price

Choosing the learning machine

- choosing the right neural network, F, for your task is a hard problem.
- Data driven approaches,

  e.g. neural architecture search,
  - are computationally expensive and unstable.

- The degrees of freedom, of F

  i.e. the flexibility of the model,
  - depends on the size of the parameter space, $|\Lambda|$.
- The performance of the model depends also on
  - the structure of Λ

  e.g. the number and shape the network hidden layers.
  - the neuron activation function, &
  - the efficiency of the training process.

Example

- A fully connected network with
  - n_0 inputs
  - n_h hidden layers
  - of n_i neurons, $i = 1, \dots, n_h$
  - and 1 output has

$$(n_{n_h} + 1) + \sum_{i=1}^{n_h} n_i (n_{i-1} + 1)$$

- as each layer of n_i neurons have
  - $n_i(n_{i-1} + 1)$ parameters.

Exercise

Compute the degrees of freedom of F_{231}

Training

Training a network consists of

- Choosing a cost function,

  e.g.
  $$J(Y, F(X, \lambda)) = \|Y - F(X, \lambda)\|^2$$

  For
  - a regression learning machine or
  - a classification model

- Solving the corresponding minimization problem, i.e. finding

$$\hat{\lambda} = \arg \min_{(x,y) \in D} \sum J(y, F(x, \lambda)).$$

- In general,
  - the objective function is non-convex in λ, and
  - the minimization problem cannot be solved globally on Λ
- A popular gradient-based strategy to search for a locally optimal solution is
  - the gradient descent algorithm

where you,

1. start from an initial guess, $\lambda^{(0)}$,
2. update it through a series of small updates, e.g.

$$\lambda^{(t+1)} = \lambda^{(t)} - \eta \sum_{(x,y)\in D} \nabla F(x, \lambda^{(t)})$$

where;

- $\eta > 0$ is the learning rate,
- $t = 1, \ldots, T$ and
- T is the training epochs.

3. let $\hat{\lambda} = \lambda^{(T)}$

Prediction

- Given $\hat{\lambda} = \lambda^{(T)}$
  - the trained network is $\hat{f}(x) = F(x, \hat{\lambda})$ and
  - can be used to define a prediction rule
  - for guessing a new unknown output
  - $\hat{y}$ given a new previously unseen input $x_0 \in \mathcal{X}$ and $x_0 \notin D$.

- Regression network
  - As for a linear regression model,
  - $\hat{f}$ is used directly to make prediction by letting

$$\hat{y} = \hat{f}(x_0)$$

- Binary classification networks
  - As for a logistic regression model,
  - the output of the network
    - is the probability of an object to belong to a specified class,

i.e.

$$\hat{f}(x) = \text{Prob}(Y = y_i) \quad \text{for } y_i \in \{y_i\}_{i=1}^{|Y|}$$

and the prediction rule is

$$\hat{y} = \arg\max \{F(x_0, \hat{\lambda}), 1 - F(x_0, \hat{\lambda})\}$$

A regression neural network

- To see how the training & prediction steps work in practice, focus on the regression case & let

1. The network architecture be the one on page 17,

2. The output of the hidden neurons be

$$z_i = \text{softPlus}(s), \quad s = w_{i0}^h + \sum_{j=1}^{2} w_{ij}^h x_j = (w_i^h)^T [1, x_1, x_2]^T$$

3. The output of the last neuron be

$$y = s, \quad s = (w^o)^T [1, z_1, z_2, z_3]^T$$

More explicitly

- Equivalently, let your learning machine be

$$F(x, [w^h, w^o]) = w^o + \sum_{i=1}^{3} w_i^o \, \text{softPlus}\left(w_{i0}^h + \sum_{j=1}^{2} w_{ij}^h x_j\right)$$

or

$$F(x, [w^h, w^o]) = w^o + \sum_{i=1}^{3} w_i^o \, \text{softPlus}(z_i^h)$$

$$z_i^h = w_{i0}^h + \sum_{j=1}^{2} w_{ij}^h x_j$$

RSS training

- The training data set is

$$\mathcal{D} = \{(x_i, y_i), \; x_i \in \mathbb{R}^3, \; y_i \in \mathbb{R}, \; i = 1, \ldots, n\}.$$

and we find a model estimate that minimizes

$$\mathcal{L}(w, \mathcal{D}) = n^{-1} \sum_{i=1}^{n} \mathcal{J}(w, (x_i, y_i)), \quad \mathcal{J}(w, (x, y)) = (y - \hat{y})^2$$

where:

- $\hat{y} = F(x, [w^h, w^o])$
- To compute a gradient descent update

$$w^{(t+1)} = w^{(t)} - \eta \nabla \mathcal{L}[w^{(t)}, \mathcal{D}]$$

we need

1. A forward pass, i.e. to evaluate $(y - \hat{y})^2$

2. A backward pass

i.e. to evaluate $\nabla \mathcal{J}(w, (x, y))^2 = -2(y - \hat{y}) \nabla \hat{y}$

- for all $(x, y) \in \mathcal{D}$ & all $t = 1, \ldots, T$

Computing the network output (forward pass)

- The network evaluation
  - is called forward propagation as you proceed from the input to the output, i.e.

1. Given an input x,
   - you evaluate the first layer of neurons, i.e. you compute z_i^h for all $i = 1, 2, 3$

2. Given z_i^h,
   - you evaluate the network output, i.e. you compute

$$\hat{y} = (w^o)^T [1, \text{softPlus}(z_1^h), \dots, \text{softPlus}(z_3^h)]^T$$

Computing the network gradient (backward pass)

- Evaluating the network gradient
  - is called back-propagation as you proceed from the output to the input layers
  - and use the estimates obtained via the forward pass.
- A network
  - is the composition of a series of multi-variated functions
  - and the backpropagation technique follows from the chain rule

$$\frac{d}{dx}(h \circ g \circ f)(x) = \frac{d}{dx} h(g(f(x))) = h'(g(f(x)))\, g'(f(x))\, f'(x)$$

More explicitly

$$\partial_{w_i^o} \hat{y} = z_i^h$$

$$\partial_{w_{ij}^h} \hat{y} = w_i^o \, softPlus'(z_i^h) \, \partial_{w_{ij}^h} z_i^h = w_i^o \, softPlus'(z_i^h) x_j$$

The process is going backward because

1. The first factor, $softplus'(z_i^h)$
   - is evaluated on the first hidden layer

2. The second factor, $x_j = \partial_{w_{ij}^o} (w_i^o)^T x$
   - is evaluated on the input layer

Exercise

Universality of neural networks

- Consider a regression neural network
  - with the structure of the network on page 22
  - but with n_h hidden neurons.
- If the activation function of all hidden neurons
  - is σ and the activation of the output neuron
  - is the linear activation $\phi(x) = x$,

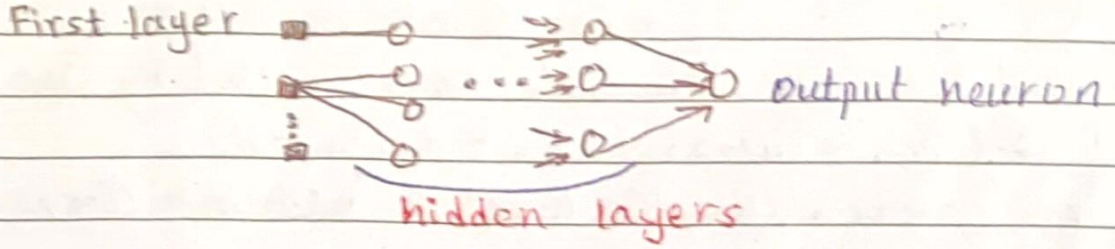
then,

$$F(x, w) = w_0^o + \sum_{i=1}^{n_h} \sigma((w_i^h)^T [0, x_1, x_2]).$$

- The obtained neural network
  - is universal in the sense of being able to
  - approximate any continuous function.

Neural Networks

- A neural network is a learning machine of a particular kind.



- input variables - each of them taking values 0 or 1
- neurons - hidden layers
- the output neurons - contains one or more neurons

* each neuron has several inputs and an output

- Neuron
  - For each neuron there are
    - → k weights $w_1, w_2, \ldots, w_k$

    where; k = is the number of arrows entering that neuron
    - → b threshold

- If the neuron receives signals $s_1, s_2, \ldots, s_k$
  - from the neurons (or input variables)
  - It sends the signal

$$\theta(w_1 s_1 + w_2 s_2 + \ldots + w_k s_k - b)$$

  - to the neuron above it.

- θ is the step function

$$\theta(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases}$$

Learning machine

The neural net functions as follows:

→ The input variables : send their values to the level 1 neurons above them

→ level 1 neurons compute their output by the formula

$$\theta(w_1 s_1 + w_2 s_2 + \dots + w_k s_k - b)$$

and send it to the level 2 neurons above them

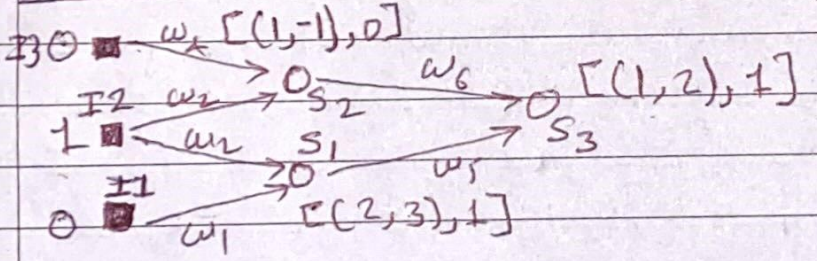
→ ...

→ The output neurons compute their outputs, and the vector of outputs is the overall output of the net.

Training

- Training a network consists of setting
  - the weights and
  - thresholds for all the neurons.
  - and possibly changing the topology of the network.

Example: What is the output of this network?



$[(w_1, \dots, w_k), b]$ represent weights $w_1, \dots, w_k$ & threshold b.

$$s_1 = \theta((w_1 \times I1) + (w_2 \times I2) + b = \theta((2 \times 0) + (3 \times 1) + 1 = \theta(4) = 0.98$$

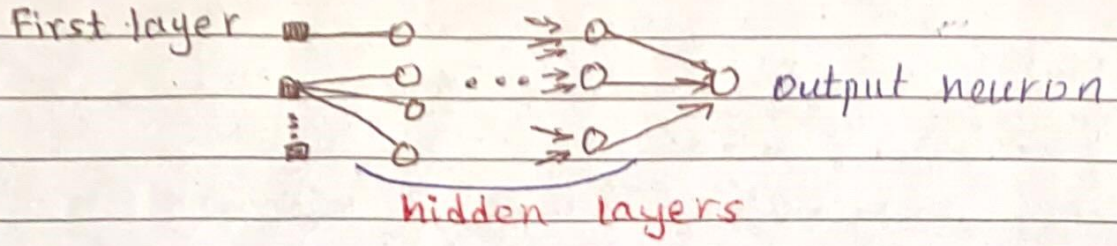
$$\therefore \theta(4) = \frac{1}{1+e^{-4}} = 0.98$$

$$s_2 = \theta((1 \times 1) + (0 \times -1) + 0) = \theta(1) = 0.73$$

$$s_3 = \theta((0.98 \times 1) + (0.73 \times 2) + 1) = \theta(3.44) = 0.97$$

Neural Networks

- A neural network is a learning machine of a particular kind.



- input variables - each of them taking values 0 or 1
- neurons - hidden layers
- the output neurons - contains one or more neurons.

* each neuron has several inputs and an output

- Neuron
  - For each neuron there are
    - → k weights $w_1, w_2, \ldots, w_k$

    where; k = is the number of arrows entering that neuron
    - → b threshold

  - If the neuron receives signals $s_1, s_2, \ldots, s_k$
    - from the neurons (or input variables)
    - it sends the signal

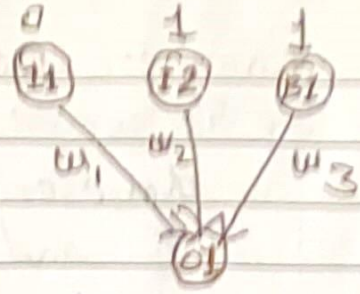
$$\theta(w_1 s_1 + w_2 s_2 + \ldots + w_k s_k - b)$$

    - to the neuron above it.

  - θ is the step function

$$\theta(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{otherwise} \end{cases}$$

Example:



$w_1 = I1 \to O1 = 0.5$
$w_2 = I2 \to O1 = 0.6$
$w_3 = B1 \to O1 = 0.7$

$O1 = \Theta((I1 \times w1) + (I2 \times w2) + w_3)$
$O1 = \Theta((0 \times 0.5) + (1 \times 0.6) + 0.7 = \Theta(1.3) = 0.79$

Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$ i.e. $\sigma(1.3) = \frac{1}{1+e^{-1.3}} = 0.79$

$\Theta(1.3) = \frac{1}{1+e^{-1.3}} = 0.79$ ✓

- Instead of Θ (discontinuous function)
  - we can take the sigmoid (logistic) function
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

- Another learning machine
Regression neural net
  - At the bottom, we have our standard input variables (attributes) $X_1, \ldots, X_p$
  - The hidden layer consists of the unit $Z_1, \ldots, Z_m$
  - The output nodes are $Y_1, \ldots, Y_k$

Parameter (weights)

- The complete set λ of weights consists of:
  $\alpha_{m,0}$ and $\alpha_{m,l}$ for $m = 1, \ldots, M$ and $l = 1, \ldots, p$
  β_0 and β_m for $m = 1, \ldots, M$.

⇒ $\alpha_{m,0}$ and β_0 play the role of thresholds

⇒ There are $M(p+1) + (M+1)$ weights overall

- convenient notation:

$\alpha_m = (\alpha_{m,1}, \ldots, \alpha_{m,p})'$ $\quad \beta = (\beta_1, \ldots, \beta_M)'$

$X = (X_1, \ldots, X_p)'$ $\quad Z = (Z_1, \ldots, Z_M)'$

Details of the regression neural net

- The learning machine works as follows:

$$Z_m = \sigma(\alpha_{m0} + \alpha_m' X) \quad m = 1, \ldots, M$$
$$Y = \beta_0 + \beta' Z$$

- A standard algorithm for training it: back-propagation

Why σ?

- without σ, it would have been just linear regression
- with σ, the prediction rules that we can get are sophisticated non-linear functions.

Introduction to back-propagation

- we do not really want to use the ERM principle
  - minimizing the training error is not computationally feasible
  - and even if it we, we could overfit

- Instead we do something similar
  - slowly move against the gradient of the training MSE
  - until some stopping condition is met
  - Namely:
    - at each step we add $-\eta$ times the gradient,
      where; η - the learning rate
      - is a small positive constant.

Computing the gradient

- As usual, the training set is $(x_1, y_1), \ldots, (x_n, y_n)$;
  - it is fixed
- The training MSE is
  $$\frac{1}{n}\sum_{i=1}^{n} R_i \quad \text{where;} \quad R_i = (y_i - Y_i)^2$$

- For given parameters, we can compute the values of the nodes:
  - Z_m & $Z_m = \sigma(\alpha_{m0} + \alpha_m' X)$
  - Y $\quad Y = \beta_0 + \beta' Z$

for each observation $i = 1, \ldots, n$;
denote them Z_{im} and Y_i, respectively.

- set $x_{i0} = 1$ and $Z_{i0} = 1$ for all observations

$R_i = (y_i - \hat{y}_i)^2$

The standard rules give:

$$\frac{\partial R_i}{\partial \beta_m} = -2(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial \beta_m} = -2(y_i - \hat{y}_i) z_{im}$$

(where $m = 0$ is allowed) and

$$\frac{\partial R_i}{\partial \alpha_{mj}} = -2(y_i - \hat{y}_i)\frac{\partial \hat{y}_i}{\partial \alpha_{mj}} = -2(y_i - \hat{y}_i)\beta_m \frac{\partial z_{im}}{\partial \alpha_{mj}}$$

$$\frac{\partial R_i}{\partial \alpha_{mj}} = -2(y_i - \hat{y}_i)\beta_m \sigma'(\alpha_{m0} + \alpha'_m x_i) x_{ij}$$

(where $j = 0$ is allowed)

Let us rewrite the derivatives as

$$\frac{\partial R_i}{\partial \beta_m} = -\delta_i z_{im} \qquad \frac{\partial R_i}{\partial \alpha_{mj}} = -s_{im} x_{ij}$$

where the "errors" δ and s are defined by

$$\delta_i = 2(y_i - \hat{y}_i) \qquad s_{im} = \beta_m \sigma'(\alpha_{m0} + \alpha'_m x_i)\delta_i$$

(The latter is called the back-propagation equation)

The overall gradient of the training MSE is given by

$$\frac{\partial MSE}{\partial \beta_m} = -\frac{1}{n}\sum_{i=1}^{n} \delta_i z_{im} \quad \text{and} \quad \frac{\partial MSE}{\partial \alpha_{mj}} = -\frac{1}{n}\sum_{i=1}^{n} s_{im} x_{ij}$$

The algorithm

- start from random weight β_m $(m = 0, 1, \ldots, M)$ and α_{mj} $(m = 1, \ldots, M,\ j = 0, 1, \ldots, p)$

1. Forward pass:

- Compute the variable $Z_m = Z_{im}$ $(m = 1, \ldots, M)$ on $X = X_i$ for the i^{th} training observation $(i = 1, \ldots, n)$

$$Z_m = \sigma(\alpha_{m0} + \alpha'_m X) \quad \text{and} \quad Y = \beta_0 + \beta' Z$$

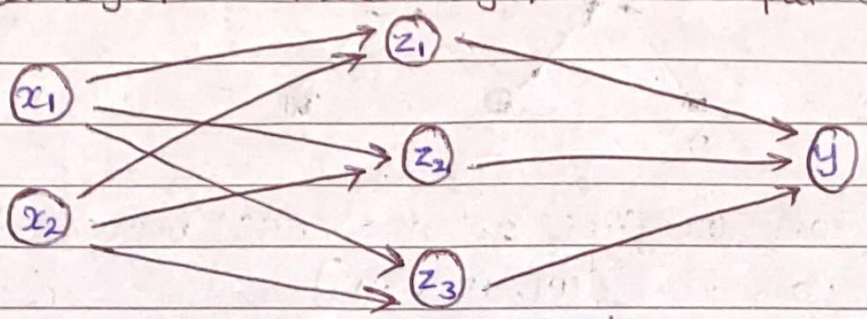
2. Backward pass:

- compute the "errors"

$$\delta_i = 2(y_i - x_i)$$

Week 4: Quiz

Q1. Consider the following binary classification neural network, where $y \in [0,1]$ is the probability of an object to be of class "yes".



Let $w^o = [0,1,1,-1]^T$ and $w_1^h = [1,0,0]^T$, $w_2^h = [0,1,0]^T$ and $w_3^h = [0,0,1]^T$ and $x = [1,-1]^T$.

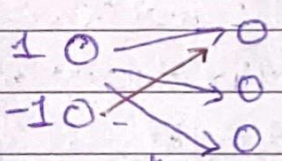
What is the class predicted by the network?

Assume that

- the activation function of the output neuron is $\sigma(t)$
- and the activation functions of all hidden neurons is

$$\text{Relu}(x) = \max\{x, 0\}$$

Answer: "yes"



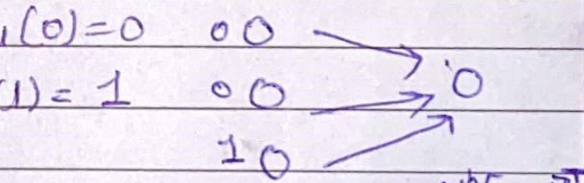
$z_1 = \text{Relu}(($

$z_1 = \text{Relu}((1 \times 1) + (-1 \times 1) + 0) = \text{Relu}(0)$

$\therefore \text{Relu}(0) = 0$

$z_1 = \text{Relu}((1 \times 0) + (-1 \times 1) + 1) = \text{Relu}(0) = 0$

$z_3 = \text{Relu}((1 \times 0) + (-1 \times 0) + 1) = \text{Relu}(1) = 1$

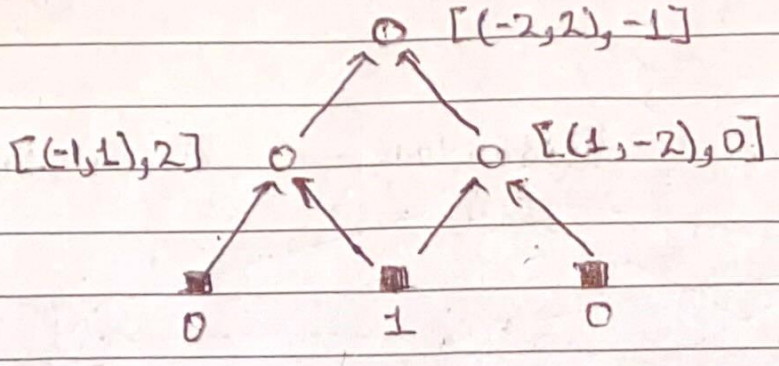


$y = \sigma((0 \times 0) + (0 \times 0) + (1 \times 1) + 1) = \sigma(2)$

$\therefore \sigma(2) = \frac{1}{1+e^{-2}} = 0.880$

So, since $\sigma(2) > 0.5$, the Answer is "yes"

Q5. What is the output of the following neural network associated with the indicated input [0, 1, 0]?



- The network weights, are the values indicated at each node in the format $[(w_1, w_2), w_0]$
- and you should assume that all all activation functions are step functions defined as $\Theta(x) = 1[x > 0]$

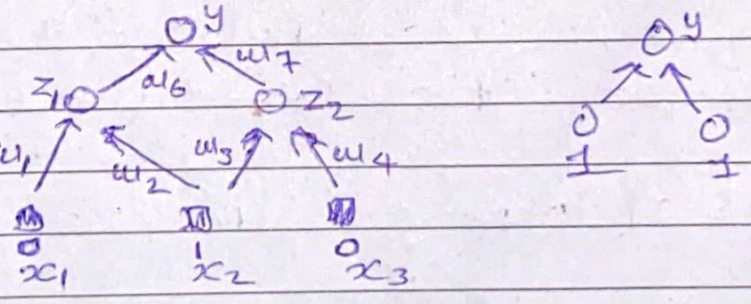
• Step function

$$\Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Note: the hidden neurons have only two inputs even if the network input is $x \in \mathbb{R}^3$.

The first weight,
i.e. w_1, always multiplies the input on the left and the second weight,
i.e. the input on the right.

For example, the input to the hidden neuron on the left is
$0 \times w_1 + 1 \times w_2 + w_0 = w_2 + w_0 = 3$ and its output is $\Theta(3) = 1$



$z_1 = \Theta((x_1 \times w_1) + (x_2 \times w_2) + w_0) = \Theta((0 \times -1) + (1 \times 1) + 2) = \Theta(3) = 1$

$z_2 = \Theta((x_2 \times w_3) + (x_3 \times w_4) + w_0) = \Theta((1 \times 1) + (0 \times -2) + 0) = \Theta(1) = 1$

$y = \Theta((z_1 \times w_6) + (z_2 \times w_7) + w_0) = \Theta((1 \times -2) + (1 \times 2) + (-1)) = \Theta(-1) = 0$

∴ Answer: 0

Q2. Compute the value of $\binom{N}{K}$ for $N=10$ & $K=4$.

$$\binom{N}{K} = \frac{N!}{K!(N-K)!} = \frac{10!}{4!(10-4)!} = \frac{3628800}{4!\,6!} = 210$$

Q3. Week 4 lab describes how to fit a regression tree
- to predict the real labels {y ε medv} given the other attributes.
- use the same technique to predict another set of real variable {y ε ptratio}, from the three attributes medv, age, rm,

i.e. using attributes in the form $x = \{x_1, x_2, x_3\}$ with $x_1 \varepsilon$ medv, $x_2 \varepsilon$ age, and $x_3 \varepsilon$ rm.
- Split the data set into a training and a test data sets of equal size using.

```
> set.seed(1234)
> train <- sample(1:nrow(Boston), nrow(Boston)/2)
```

Compare the performance of the fitted tree on the training and testing data by computing

$$\Delta_{RSS} = \frac{|RSS(\mathcal{D}_{train}) - RSS(\mathcal{D}_{test})|}{RSS(\mathcal{D}_{test})}$$

where RSS($\mathcal{D}$) is the Residual Sum of Squares
- of the prediction of the model on data set $\mathcal{D}$.

Write down the value of Δ_{RSS} (± 0.05)

Answer: 3.95

Q4. Let $F: \mathcal{X} \times \Lambda \to \mathcal{Y}$ be a learning machine for
- linear logistic regression & let the object space be $\mathcal{X} = \mathbb{R}^d$. Select the correct

b. $\mathcal{Y} = [0,1]$.

Week 4: Quiz

5.

$S_1 = (-1 \times 0) + (1 \times 1) + 2 = 3$

$\therefore \theta(x) = 1[x > 0]$ so, $\theta(3) = 1[3 > 0] = 1$

$S_2 = (1 \times 1) + (0 \times -2) + 0 = 1$

$\therefore$

$\theta(1) = 1[1 > 0] = 1.$

$S_3 = (-2 \times 1) + (2 \times 1) + (-1) = -2 + 2 - 1 = -1$

$\therefore \theta(-1) = 1[-1 < 0] = 0$