Introduction to Neural Networks

Neural Networks

Cathy Poliak, Ph.D.
cpoliak@central.uh.edu

Department of Mathematics
University of Houston

Neural Networks: Introduction

We live in the era of technology with computers taking over vast majority of the operation.

Computers have by far surpassed humans in the domains of

- numerical computations, and
- symbol manipulation

Remember such things as

- travel agencies?
- Or newspapers (made of.. paper)?
- Or those things for cashiers to count?

Neural Networks

Nonetheless, there are still domains where humans reign supreme:

- pattern recognition,
- noise reduction,
- certain optimization tasks.

https://www.youtube.com/watch?v=ad79nYk2keg

Example. A toddler can recognize his/her mom in a huge crowd, but a computer with a centralized architecture wouldn't be able to do the same.

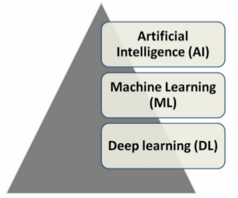
Example. Simply check this priceless video of robots playing soccer:

`https://www.youtube.com/watch?v=KfNRXTS55nY`.

Improving on computers' capability in dealing with those tasks is the field of Artificial Intelligence (AI).

Artificial Intelligence (AI) and Machine Learning (ML)

- **Artificial Intelligence (AI)** systems attempt creating machines that imitate parts of human intelligence mechanisms.
- **Machine learning (ML)** is a branch of AI which helps computers to program themselves based on the input data.
- **Deep learning (DL)** is complex set of neural networks with more layers of processing, which develop high levels of abstraction.
- The hierarchy of those critical concepts looks like this:



Artificial Neural Networks (ANN)

A decisive step in the improvement of such machines came from the use of so-called **Artificial Neural Networks (ANNs)**.

ANNs are an example of machine learning algorithms that try to

- emulate the neuronal structure of human brain, and
- reproduce the human thinking process for certain tasks.

Those tasks include

- image & object recognition,
- image classification,
- fraud detection,
- hand writing identification,
- video analysis.

Inspiration for Neural Networks

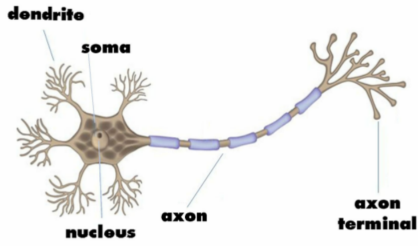
- The **human brain** is the central processing unit for all the functions performed by us as humans.
- It constitutes a complex network of neurons, most of which process and transmit information they obtain either from:
  - sensors (e.g. eyes or ears),
  - or other neurons.
- Weighing only 1.5 kilos, it has around 86 billion neurons.

Biological Structure of a Neuron

Neurons are the nodes of the brain network that receive, process and transmit signals.

The major components of each neuron are:

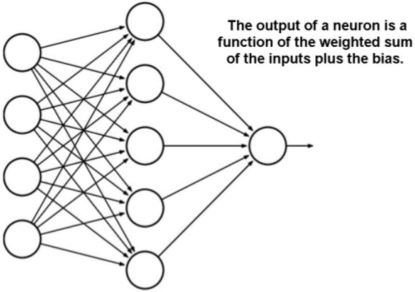
- **Dendrites**: Entry points in each neuron which take input from other neurons in the network in form of electrical impulses.
- **Cell Body**: Calculates a function (or "processes the signals") of dendrite inputs.
- **Axons**: Transmit calculated outputs as signals to next neuron.



How do neural netWORKs WORK?

Similar to the biological neuron structure, ANNs define the neuron as a:

"Central processing unit that performs a mathematical operation to generate output from a set of inputs."



Essentially, ANN is a set of mathematical function approximations.

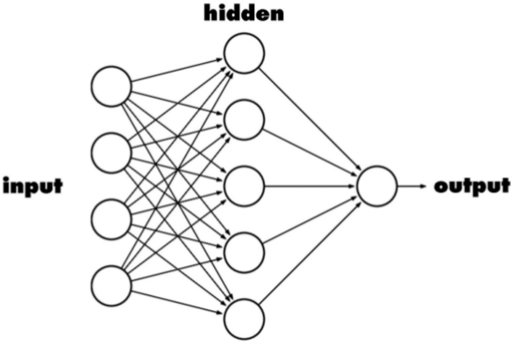
Neural Network terminology.

We would now be introducing new terminology associated with ANNs:

- Input layer
- Hidden layer
- Output layer
- Weights
- Bias
- Activation functions

Layered Structure.

Most neural network processing frameworks look as follows:

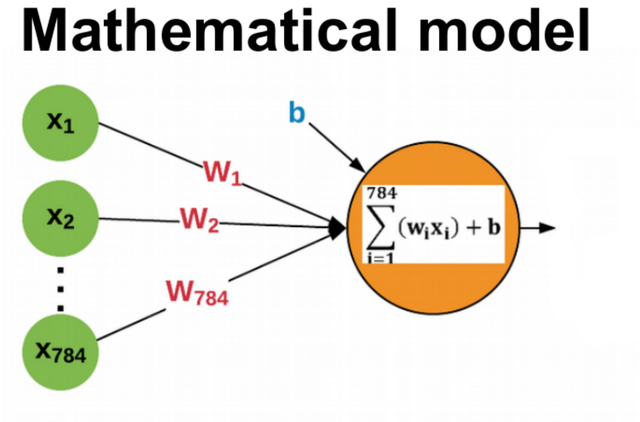


It consists of

- Input layer - the data inputs,
- Hidden layer(s) - processing "unit",
- Output layer - the neural network output.

Mathematical Model of a (Linear) Neuron

For a linear neuron, mathematical model would look like:



Neuron Model: Weights and Bias

Let y - neuron output, then

$$y = b + \sum_{i=1}^{p} w_i x_i$$

Remotely reminds you of something?

Neuron Model: Weights and Bias

Let y - neuron output, then

$$y = b + \sum_{i=1}^{p} w_i x_i$$

Remotely reminds you of something? Recall multiple linear regression:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

Only now the parameters we need to learn are called

- **weights** (instead of coefficients), and
- **bias** (instead of intercept)

and denoted as

- w_i (instead of β_i), $i = 1, \ldots, p$,
- b (instead of β_0)

Neuron Model: Weights and Bias

Let y - neuron output, then

$$y = b + \sum_{i=1}^{p} w_i x_i$$

Remotely reminds you of something? Recall multiple linear regression:

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

Only now the parameters we need to learn are called

- **weights** (instead of coefficients), and
- **bias** (instead of intercept)

and denoted as

- w_i (instead of β_i), $i = 1, \ldots, p$,
- b (instead of β_0)

Our task is to estimate weights w_i and bias b, where

- weights determine how strongly one neuron affects the other,
- bias off-sets some of the effects.

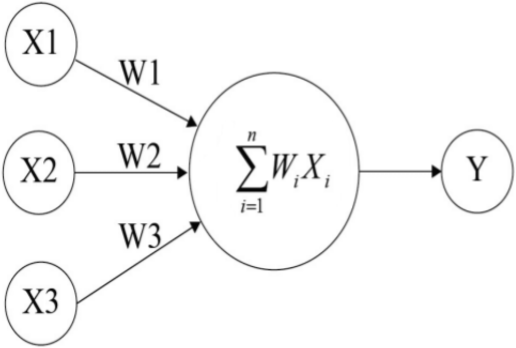
Weights and Biases: Example.

If a neuron has three inputs x_1, x_2, x_3, then

- the weights applied to those inputs are denoted w_1, w_2, w_3,
- the output is

$y = \hat{f}(x) + \varepsilon$

$$y = \underline{f(x_1, x_2, x_3)} = \sum_{i=1}^{3} w_i x_i$$



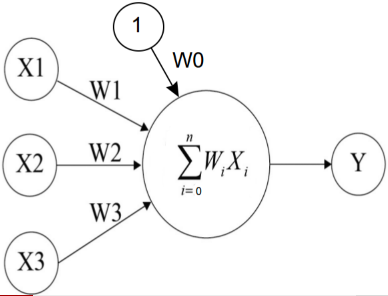
Weights and Biases: Example.

In most scenarios we need bias term as well, which is added by introducing

- an "artificial input" $x_0 \equiv 1$, and
- its corresponding weight w_0

For our previous three input example, the output now is

$$y = f(1, x_1, x_2, x_3) = \sum_{i=0}^{3} w_i x_i$$

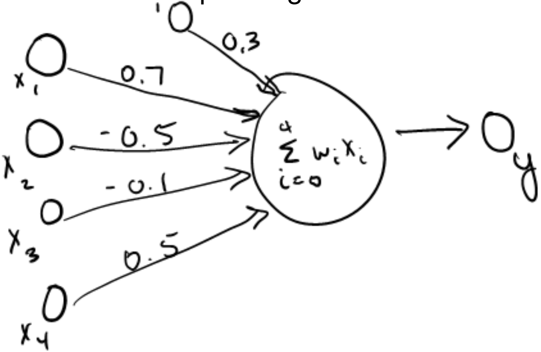


Example

Assume we have a linear neuron with

- inputs x_1, x_2, x_3, x_4
- weights $w_1 = 0.7, w_2 = -0.5, w_3 = -0.1, w_4 = 0.5$
- for bias node, $b = 0.3$.

Draw the corresponding neuron structure.



$$\hat{y} = b + \sum_{i=1}^{4} w_i x_i$$

$$\tilde{y} = 0.3 + 0.7x_1 - 0.5x_2 - 0.1x_3 + 0.5x_4$$

Lab Questions

1. Calculate the output of the previous linear neuron for $x_1 = 3$, $x_2 = 7$, $x_3 = -3$, and $x_4 = 10$.

   a) 0.3
   b) 10
   c) 17
   d) 4.2

$$\hat{y} = b + \sum_{i=1}^{4} w_i x_i$$

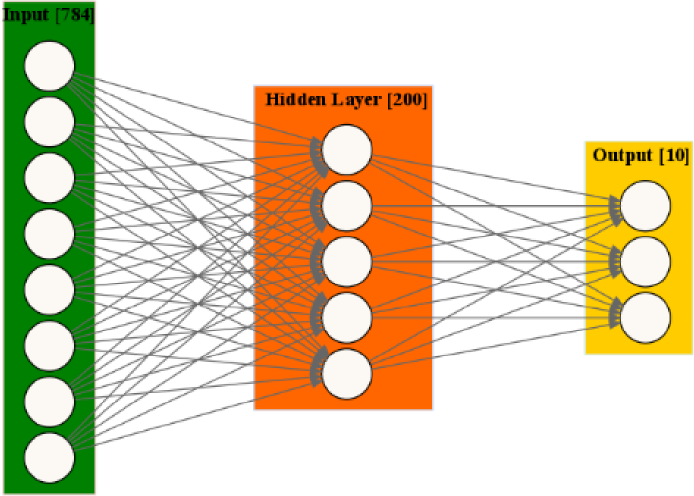
$$= 0.3 + 0.7x_1 - 0.5x_2 - 0.1x_3 + 0.5x_4$$

$$= 0.3 + 0.7(3) - 0.5(7) - 0.1(-3) + 0.5(10)$$

$$= 4.2$$

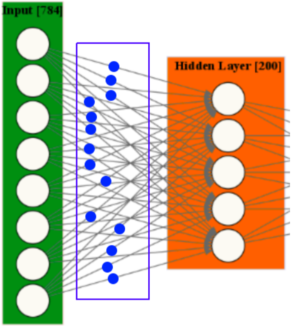
Artificial Neural Networks

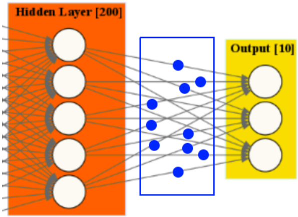
Easy so far? Well, typically we don't deal with just a single neuron, but rather with a network of neurons:



Each neuron in the hidden layer brings about:

- weight parameters, that determine either
  - how strongly it is affected by other neurons (e.g. inputs),
  - or how strongly it affects other neurons (e.g. outputs).





- and a bias term to offset some of the effects.

Training ANN. Universal Function Approximator.

Training is the act of

- presenting the ANN with some sample data, and
- modifying the weights to better approximate the desired function.

Training ANN. Universal Function Approximator.

Training is the act of

- presenting the ANN with some sample data, and
- modifying the weights to better approximate the desired function.

In general, ANN is a universal function approximator:

- given input data $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$, and
- corresponding labels $y_1, \ldots, y_n \in \mathbb{R}$ of those inputs,
- it approximates function $f(.)$ such that

$$f(\mathbf{x}_i) = y_i, \; i = 1, \ldots, n$$

Training Methods

There are two main types of training:

- Supervised learning:
  1. ANN is supplied with both inputs and desired outputs
  2. Weights are modified to reduce the difference between the predicted and desired outputs
- Unsupervised learning:
  1. ANN is only supplied with inputs
  2. Weights are adjusted for similar inputs to generate similar outputs
  3. ANN identifies the patterns and differences in the inputs

Training ANN: Forward Propagation

To obtain a neural network output, we :

input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

which is called forward propagation.

At each neuron of a hidden layer,

1. $\sum_i weight_i \times input_i + bias$ is applied, where *input* comes from previous layer (could be an input or previous hidden layer).

Training ANN: Forward Propagation

To obtain a neural network output, we :

$$\text{input layer} \rightarrow \text{hidden layer(s)} \rightarrow \text{output layer}$$

which is called forward propagation.

At each neuron of a hidden layer,

1. $\sum_i weight_i \times input_i + bias$ is applied, where *input* comes from previous layer (could be an input or previous hidden layer).
2. Activation function f is applied to $\sum_i weight_i \times input_i + bias$ (more on activation function later, but for linear neuron it's identity)

Training ANN: Forward Propagation

To obtain a neural network output, we :

input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

which is called forward propagation.

At each neuron of a hidden layer,

1. $\sum_i weight_i \times input_i + bias$ is applied, where *input* comes from previous layer (could be an input or previous hidden layer).
2. Activation function f is applied to $\sum_i weight_i \times input_i + bias$ (more on activation function later, but for linear neuron it's identity)
3. Value $f(\sum_i weight_i \times input_i + bias)$ is propagated to the next layer (could be the next hidden layer or the output layer)

Backpropagation.

Once the output is arrived at after completion of forward propagation:

- we compute the error $\hat{y} - y$ (predicted output − true output),
- use this error to correct the weights and biases used in forward propagation.

How exactly is the error used?

1. Derivative of error is taken.
2. Amount of weight that has to be changed is determined by gradient descent.
3. The gradient suggests how steeply the error will be reduced or increased for a change in the weight.
4. The backpropagation keeps changing the weights until there is greatest reduction in errors by an amount known as the learning rate.

This process of

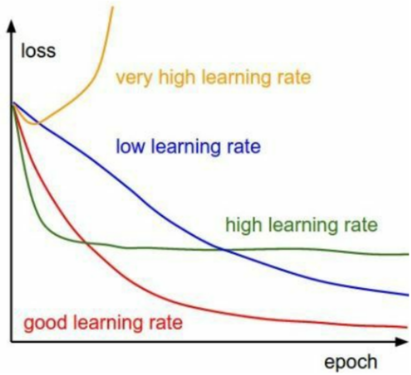
compute error $\rightarrow$ take derivative $\rightarrow$ change weights/bias via gradient descent

is known as backpropagation.

Backpropagation: Learning Rate.

The backpropagation keeps changing the weights until there is greatest reduction in errors by an amount known as the learning rate.

Learning rate is a scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments to reduce the errors faster.



Linear ANN: Example

Example. Presume we have an ANN of linear neurons with

- Input layer of three neurons: x_1, x_2, x_3,
- Fully-connected hidden layer of two neurons: h_1, h_2,
- One output neuron y.

Linear ANN: Example

Example. Presume we have an ANN of linear neurons with

- Input layer of three neurons: x_1, x_2, x_3,
- Fully-connected hidden layer of two neurons: h_1, h_2,
- One output neuron y.

Presume the following **weight matrices** for

input & **hidden** layer:

Input \ Hidden	h_1	h_2
1 (bias)	0.5	0.3
x_1	0.2	0.5
x_2	0.5	-0.2
x_3	-0.4	0.7

Linear ANN: Example

Example. Presume we have an ANN of linear neurons with

- Input layer of three neurons: x_1, x_2, x_3,
- Fully-connected hidden layer of two neurons: h_1, h_2,
- One output neuron y.

Presume the following **weight matrices** for

input & **hidden** layer:

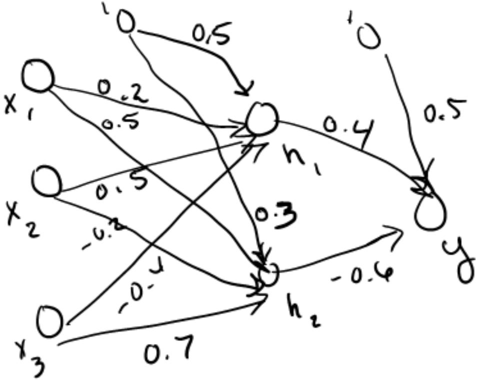
Input \ Hidden	h_1	h_2
1 (bias)	0.5	0.3
x_1	0.2	0.5
x_2	0.5	-0.2
x_3	-0.4	0.7

hidden & **output** layer:

Hidden \ Output	y
1 (bias)	0.5
h_1	0.4
h_2	-0.6

Example (cont'd). Proceed to

- Draw this ANN, and
- Given $x_1 = 5, x_2 = 10, x_3 = -2$, calculate the resulting output



$$\hat{y} = 0.5 + 0.4\hat{h}_1 - 0.6\hat{h}_2$$

$$\hat{h}_1 = 0.5 + 0.2x_1 + 0.5x_2 - 0.4x_3$$

$$\hat{h}_2 = 0.3 + 0.5x_1 - 0.2x_2 + 0.7x_3$$

Lab Question

2. What is the predicted output $\hat{y}$ for the previous example?
   a) -0.6
   b) 7.3
   c) 3.78
   d) 0.5

$x_1 = 5 \quad x_2 = 10 \quad x_3 = -2$

$\hat{h}_1 = 0.5 + 0.2(5) + 0.5(10) - 0.4(-2) = 7.3$

$\hat{h}_2 = 0.3 + 0.5(5) - 0.2(10) + 0.7(-2) = -0.6$

$\hat{y} = 0.5 + 0.4\hat{h}_1 - 0.6\hat{h}_2$

$= 0.5 + 0.4(7.3) - 0.6(-0.6)$

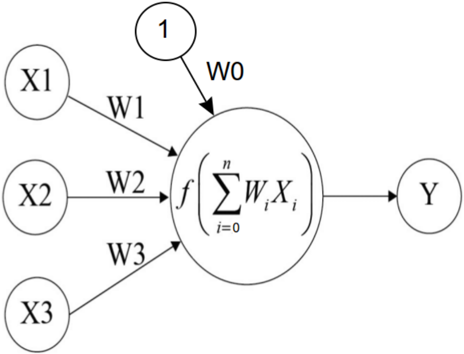
$= 3.78$

Epoch

- One iteration or pass through the process of providing the network with an input and updating the network's weights is called an epoch.
- It is a full run of feed-forward and backpropagation for update of weights. It is also one full read through of the entire dataset.
- Typically, many epochs, in the order of tens of thousands at times, are required to train the neural network efficiently.

ANN: Activation Functions.

The full potential of neural networks is mainly achieved through activation function f: it helps transform the linear combinations of inputs, e.g. $\sum_i w_i x_i$, for the purpose of approximating a wide array of functions.



Without activation functions, the neural networks are only able to approximate linear functions (those of form $\sum_i a_i x_i$).

ANN: Activation Functions.

A linear function is a polynomial of degree one, a straight line without any curves:

$$y = f(x) = 3 + 5x$$

However, most of the problems the neural networks try to solve are nonlinear and complex in nature.

To achieve the nonlinearity, the activation functions are used, which could be:

- high-degree polynomial functions

$$y = f(x) = x^4 + 3x^3 + 4x$$

- & others

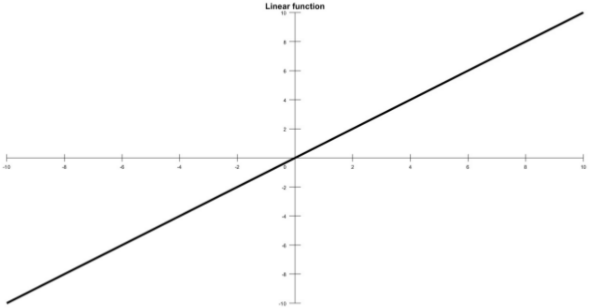
$$y = f(x) = sin(x)$$

Activation functions:

- give the nonlinearity property to neural networks and
- make them true universal function approximators

Activation Functions: Linear Function.

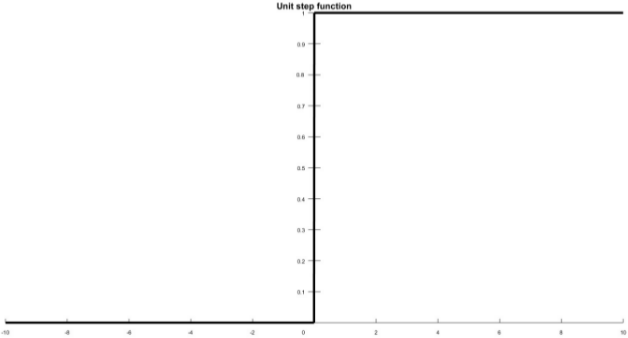
1. Linear function: $y = f(x) = x, range(f) = (-\infty, +\infty)$



Activation Functions: Unit-step Function.

1. Unit-step activation function:

$$y = f(x) = \begin{cases} 0, x < 0 \\ 1, x \geq 0 \end{cases} \quad , range(f) = \{0, 1\}$$

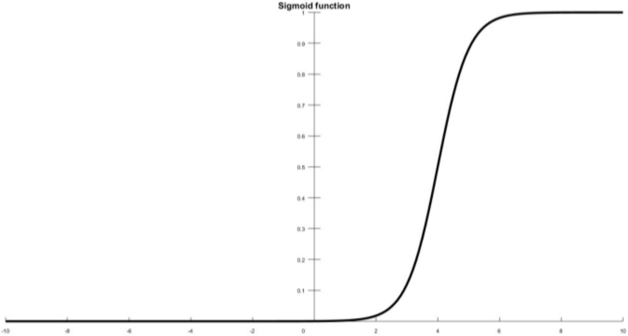


These types of activation functions are useful classify an input model in one of two groups.

Activation Functions: Sigmoid.

1. Sigmoid function:

$$y = f(x) = \frac{1}{1 + e^{-x}}, \ range(f) = (0, 1)$$

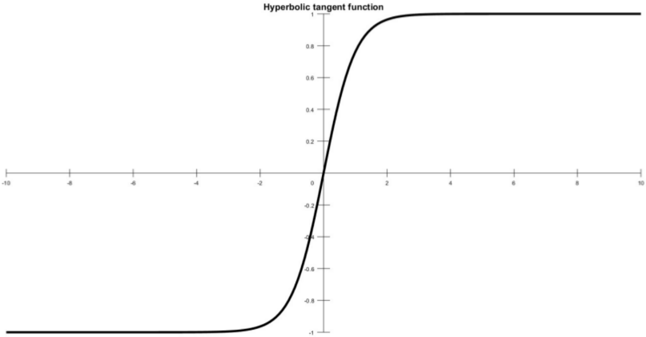


Just as for logistic regression, sigmoid function calculates probabilities of a binary outcome.

Activation Functions: Hyperbolic Tangent.

1. Hyperbolic Tangent function:

$$y = f(x) = tanh(x), \ range(f) = (-1, 1)$$

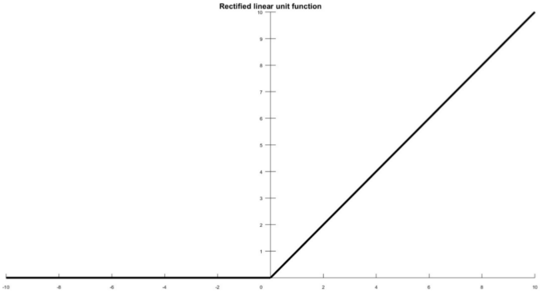


It is a scaled sigmoid function, with gradient being stronger (derivatives more steep) for hyperbolic tangent.

Activation Functions: Rectified Linear Unit (RELU).

1. Rectified Linear Unit (ReLU) function:

$$y = f(x) = \begin{cases} 0, x < 0, \\ x, x \geq 0 \end{cases}, \quad range(f) = [0, \infty)$$



ReLU is the most used activation function since 2015. It is a simple condition and has advantages over the other functions. ReLU finds applications in computer vision and speech recognition using deep neural nets.

Activation Functions: Which one to use?

When picking activation function to use, consider the following:

- It should be differential; we will see why we need differentiation in backpropagation.
- It should not cause gradients to vanish.
- It should be simple and fast in processing.
- It should be zero-centered.

Activation Functions: Which one to use Sigmoid.?

The sigmoid is the most used activation function, but it suffers from the following setbacks:

- Since it uses logistic model, the computations are time consuming and complex
- It causes gradients to vanish and no signals pass through the neurons at some point of time
- It is slow in convergence
- It is not zero-centered

Activation Functions: Which one to use? ReLU.

These drawbacks are solved by ReLU:

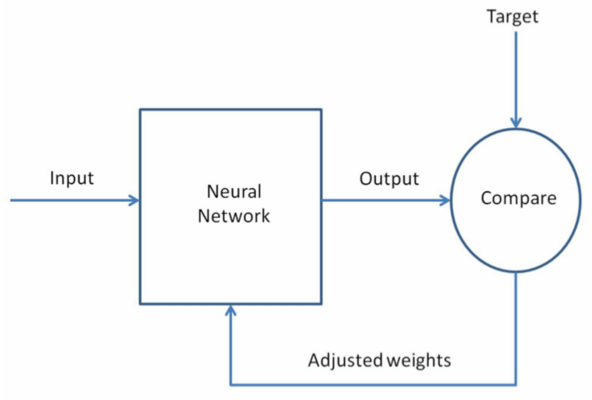
- ReLU is simple and is faster to process.
- It does not have the vanishing gradient problem.
- It has shown vast improvements compared to the sigmoid and tanh functions.
- ReLU is the most preferred activation function for neural networks and DL problems.

ReLU is used for hidden layers, while the output layer can use a

- softmax function for classification problems, and
- a linear function of regression problems.

Step-by-step ANN training.

Let us analyze in detail, step by step, all the operations to be done for network training:



Step-by-step ANN training.

Let us analyze in detail, step by step, all the operations to be done for network training:

1. Initialize the weights w^0 and biases b^0 with random values (one time).
2. Repeat the steps 3 to 5 for each training pattern (presented in random order), until the error is minimized or a stopping criteria is reached.
3. Conduct forward propagation for ANN with weights w^0 and biases b^0:

   input layer $\rightarrow$ hidden layer(s) $\rightarrow$ output layer

4. Conduct backpropagation:

   compute error $\rightarrow$ take derivative $\rightarrow$ adjust weights/biases

5. Set w^0 and b^0 equal to adjusted weights/biases, back to step 3.

The complete pass back and forth is called a training cycle or epoch. The updated weights and biases are used in the next cycle. We keep recursively training until the error is very minimal.

Simple Example in R

- The package for neural networks is called `nuerualnet`
- Consider a simple data set of squared numbers, which will be used to train a `nerualnet` function in R.
- Type and run the following in `R`

```
install.packages("neuralnet")
library(neuralnet)
input = c(0,1,2,3,4,5,6,7,8,9,10)
output = input^2
mydata = data.frame(cbind(input,output))
names(mydata) = c("input","output")
mydata
set.seed(1)
model = neuralnet(output ~ input,  data = mydata,
                   hidden = 10, threshold = 0.01)
model
plot(model)
```

3. In the results matrix, what is the value of the error?

a) 10 b) 0.0104 c) 0.02 d) 1

Type and run the following in R

```
yhat = as.data.frame(model$net.result)
final_output = cbind(input, output,yhat)
colnames(final_output) = c("Input","Expected output", "Neural Net
print(final_output)
sum((output - yhat)^2/2)
```

4. What is the predicted output if $x = 5$?

   a) 24.9927
   b) 5
   c) -0.0612
   d) 0.0104

5. What is the result of $\sum \frac{(y-yhat)^2}{2}$?

   a) 0.0012
   b) -0.0612
   c) 0.0104
   d) 10