

Introduction to multi-layer perceptron

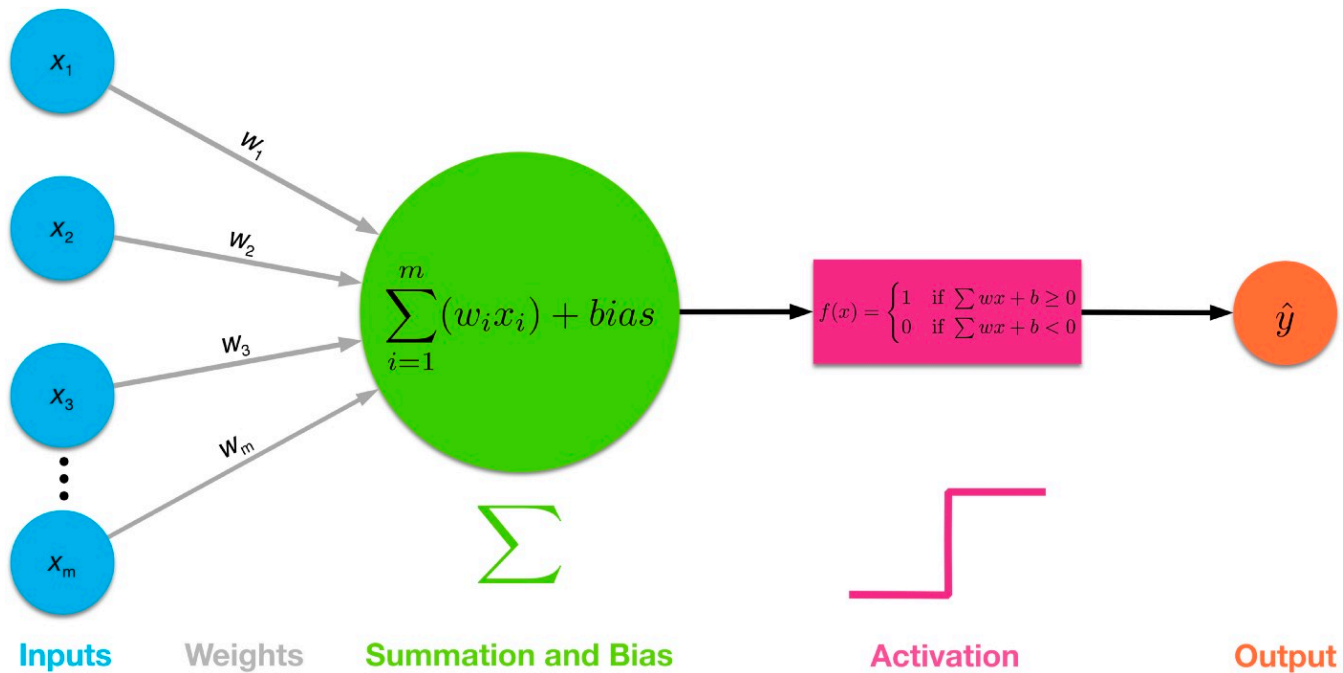
Yulia Newton, Ph.D.

CS156, Introduction to Artificial Intelligence

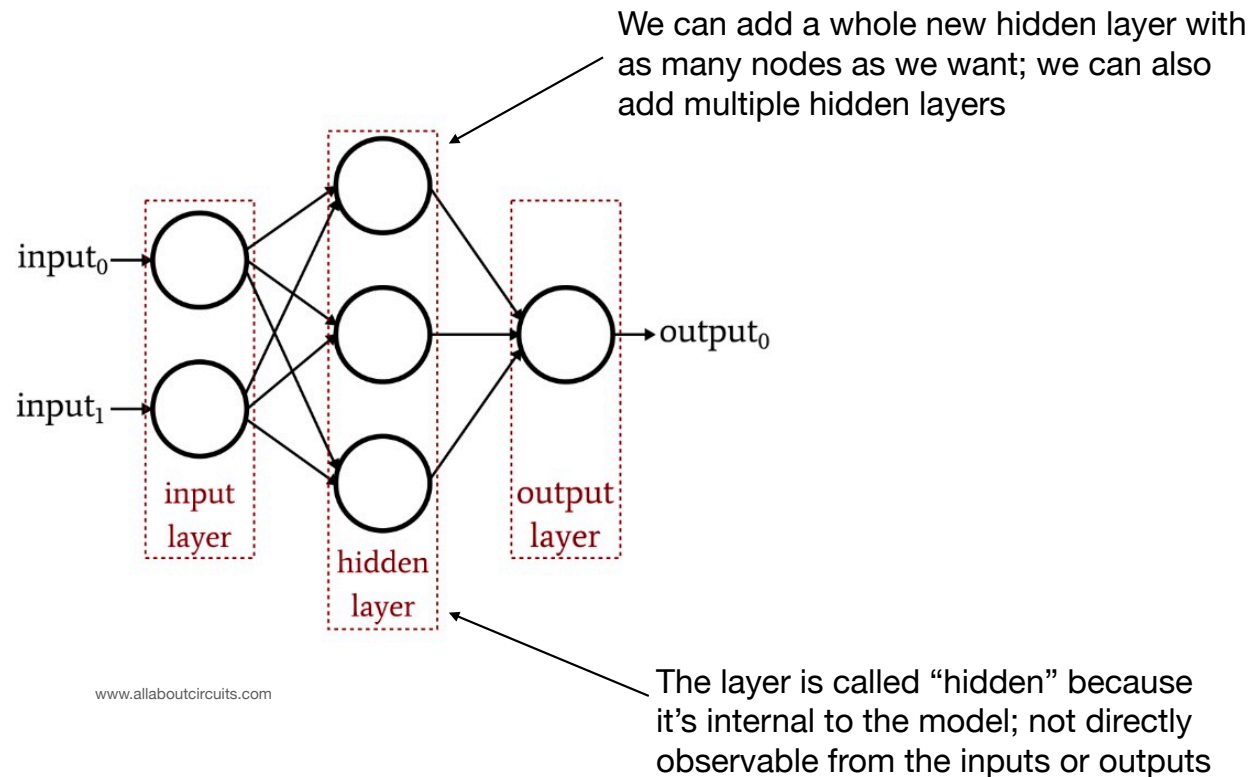
San Jose State University

Spring 2021

Single layer perceptron model review



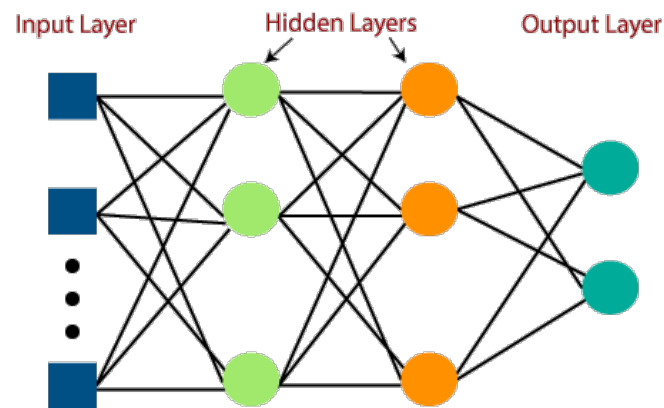
We can expand this model to more nodes and more layers



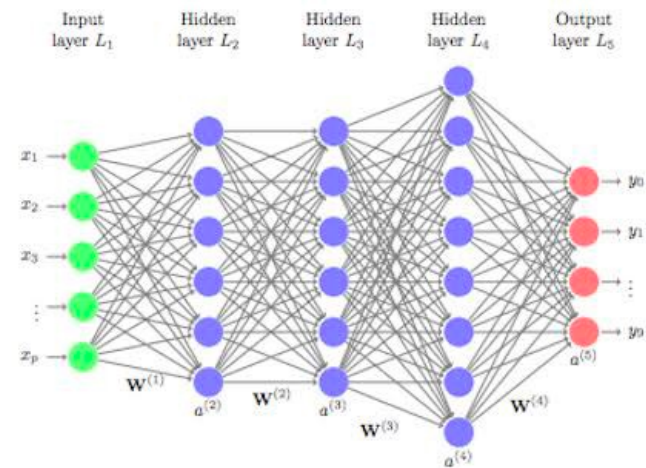
Multi-layer perceptrons

- Multi-layer perceptron (MLP) models
- MLP is a type of an artificial neural network (ANN)
 - Everything we learn about MLPs is applicable to general ANNs
- Hidden layers in MLP models are fully connected
 - Also called “dense” layers
 - We will discuss other types of neural networks later

MLPs with multiple hidden layers



The outputs of one layer become inputs for the next layer



The structure of our ANN model (number of inputs, number of hidden layers, number of neuron nodes in each hidden layer, the choice of activation functions) are referred to as “architecture” of a particular ANN model

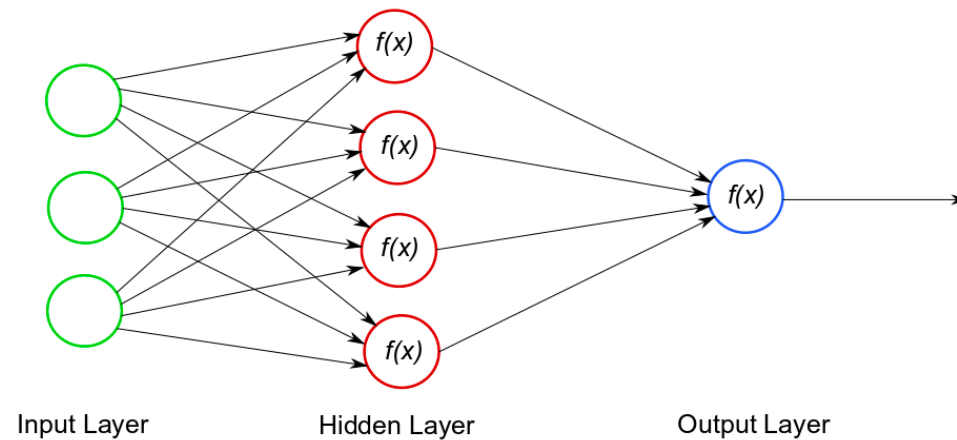
Different types of nodes in ANNs

- Input nodes - provide information from the outside to the ANN model
 - No computations are performed in the input nodes
 - Pass on the input information to the nodes in the next layer
- Hidden nodes - these nodes have no connection to the outside world
 - Perform computations
 - Transfer transformed information from the input nodes to the output nodes
 - An ANN model can have zero, one or many hidden layers
- Output nodes - responsible for presenting computed information to the outside world
 - Present the results of the ANN model prediction

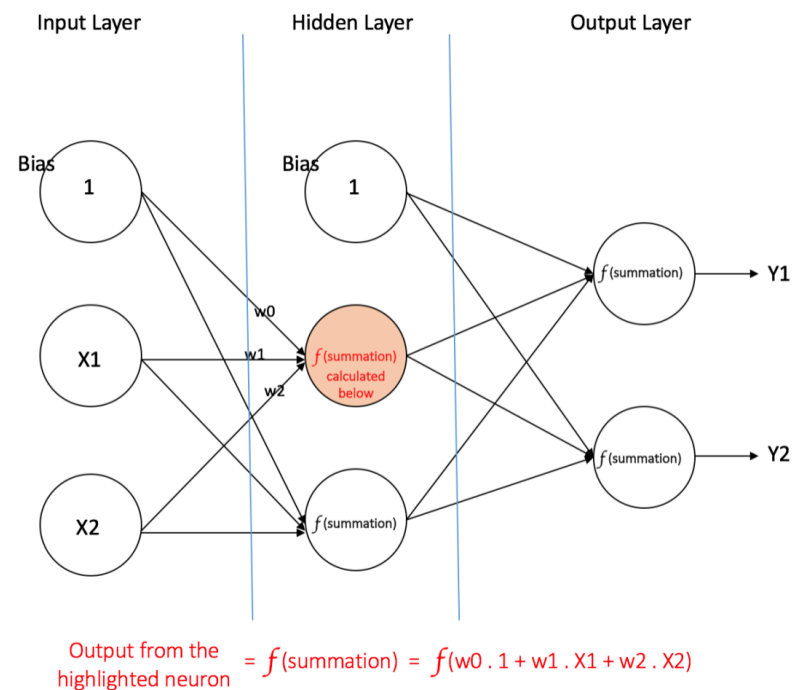
What is hidden by the hidden layer?

- The neurons of the hidden layer and their outputs cannot be directly observed
- No way to know what the desired output of the hidden layer is
- The inside structure of a neural network model is a black box to the users
 - Think of commercial ANN-based predictive models, we observe inputs and outputs, nothing inside the model's hidden layer
 - Black box

Hidden layer neurons have activation functions attached to them as well



Activation functions act on the weighted sum computed for each neuron node



Let's revisit activation functions

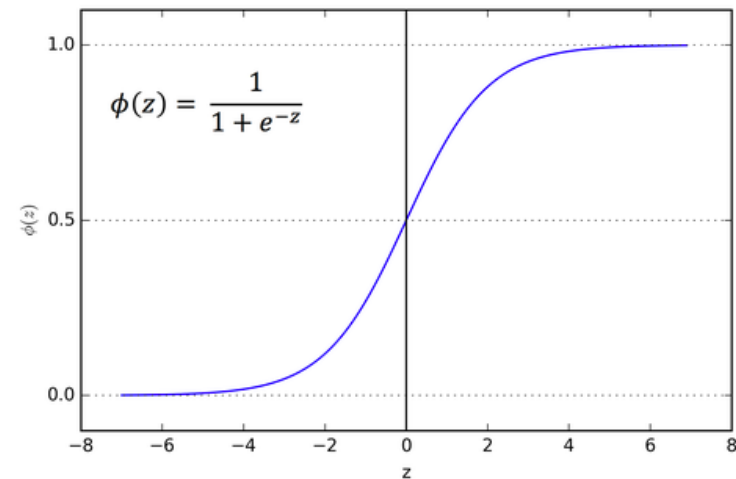
- Activation functions allow mapping input values to a desired scale/value range/value set/distribution
- An activation function allows an artificial neuron in our model to activate/fire or not
 - Similar to how the biological neurons work in the brain
- Activation functions used in ANNs are non-linear functions
- In an ANN:
 - An given activation function can be used for a neuron or for the whole layer
 - The activation function is applied to the weighted sum of the inputs
 - The output of the neuron is the transformed by activation function weighted sum

Activation functions most often used in ANNs

- Sigmoid function
 - Tanh function
 - ReLU function
 - Leaky ReLU function
-
- Let's look at these functions and understand them

Sigmoid function

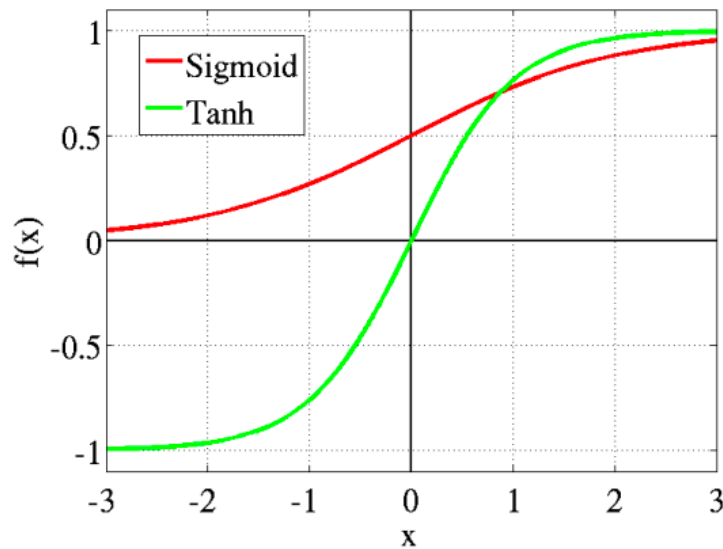
- You should be familiar with the sigmoid function from the logistic regression module
- Transforms a real value into a probability
- Softmax is a generalized version of sigmoid function used for multi-class problems
- Usually used in the output layer



<https://towardsdatascience.com>

Tanh function

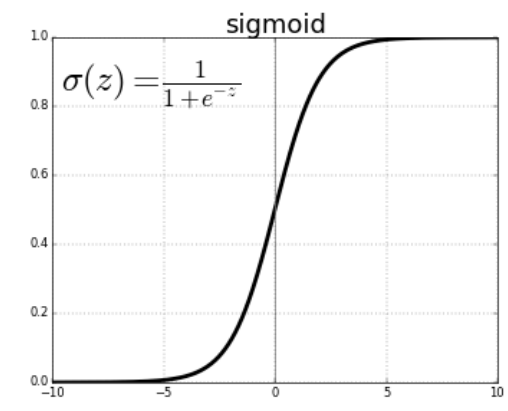
- Similar to sigmoid function but transforms real values to $[-1,1]$ range



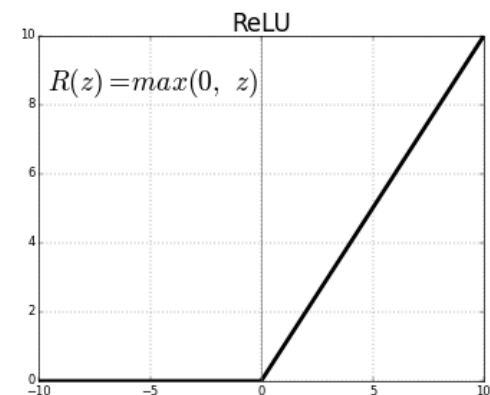
<https://towardsdatascience.com>

ReLU function

- Rectified Linear Unit (ReLU)
- Most widely used activation function
- Transforms real values to $[0, \text{infinity})$ range
- Both the function and its derivative are monotonic
- The biggest advantage of this activation function is that it does not activate all the neurons

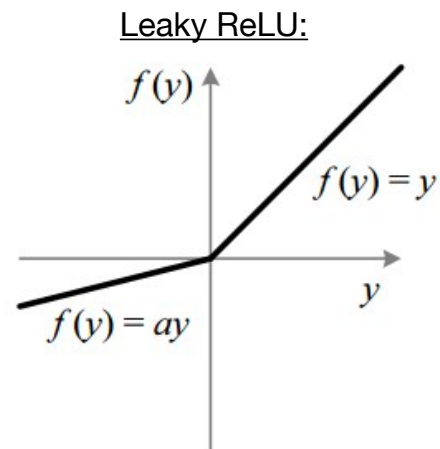
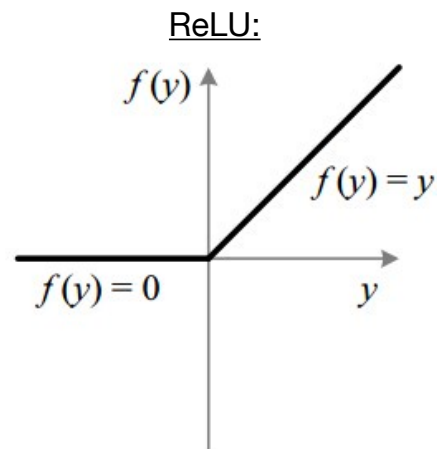


<https://towardsdatascience.com>



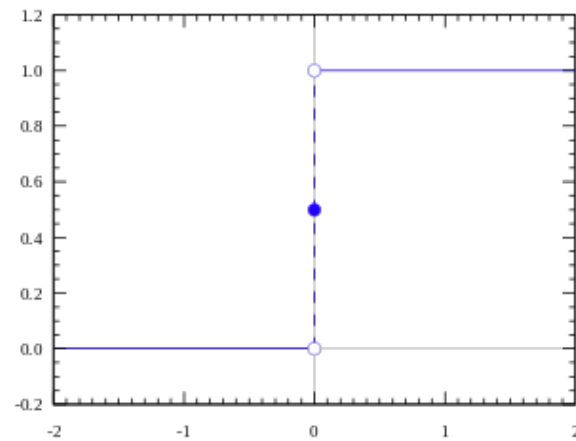
Leaky ReLU function

- An attempt to solve the problem with ReLU function assigning zero to any negative input
 - “Neuronal death”
- Transforms input values to $(-\infty, \infty)$ range



Step function

- Occasionally you will see the step function used as well



<https://medium.com>

Requirements for the activation function

- An activation function must be non-linear and continuously differentiable
- This requirement is because we use gradient descent technique in fitting/learning the network weights

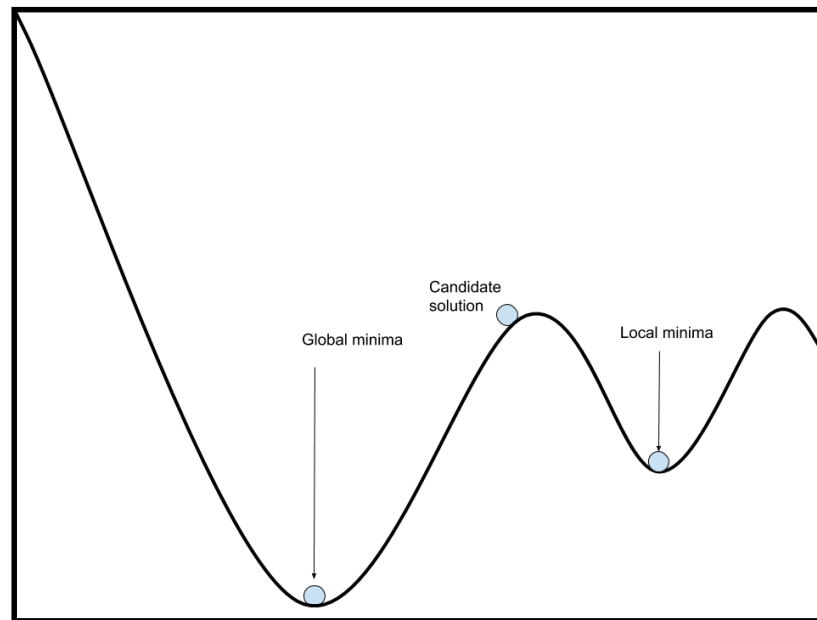
How to choose which activation function to use?

- An activation function is meant to approximate the function you are trying to learn with your model
 - E.g. sigmoid function works well for two class classifier problems
 - If you do not know what function the patterns in your data form, choose ReLU
- Choosing an appropriate activation function will lead to faster training and better performing model
- You can use a custom activation function as long as satisfies the requirements of an activation function
 - An activation function must be non-linear and continuously differentiable

Training MLP models

- By training we mean fitting the weights for all the connections in the network that achieve the best model performance
- Training in MLP is similar to training a single layer perceptron
 - A set of training examples and initial weights are presented to model, predictions are computed, error is calculated, then weights are adjusted
- Because the loss function for each neuron is differentiable, the overall loss function that combines all the neurons is also differentiable (chain rule)
 - ANN model is trained using back-propagation with gradient descent
 - Caution: MLP loss function might have a number of local minima
 - Learning with multiple restarts might help

Training MLP models (cont'd)



Loss function curves often exhibit non-convex shapes

What is back-propagation?

- Weights in an MLP model are adjusted using backpropagation method
 - A type of an optimization method
- The technique used is gradient descent
 - Weight adjustment during back-propagation uses differential of the loss function
 - The weight are adjusted to reduce the loss/error

Training steps for MLP models

- Steps:

- Initial weights get assigned for the connections to the first hidden layer
- Forward propagation:
 - The weights and the input values are used to calculate the outputs of the nodes of the first hidden layer
 - These outputs are used as inputs into the next hidden layer
 - And so the outputs of the previous hidden layer are propagated further as the inputs into the next hidden layer
 - Until we reach the output layer
- Back propagation:
 - Perform weight update backwards, from output layer to input layer
 - Using gradient descent procedure
- Repeat forward propagation and back propagation until some stopping condition

Model initialization

- It's a common practice for the initial weights to be randomly initialized

Forward propagation

- The predictions are computed using current model weights (initial weights at the start)
- The outputs of the previous layer are used as input into the next layer
- An activation function is used in the output layer
- Predictions are compared to true labels

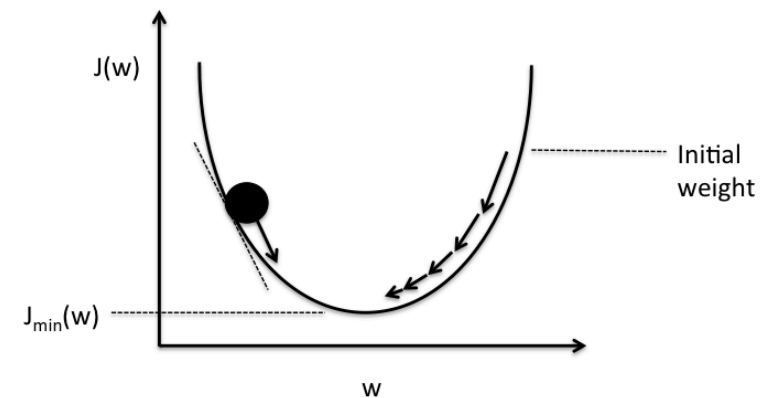
Loss function

- A loss function is a performance metric of how closely our model predicts the output variable
 - Are the predicted values as close as possible to the actual values?
- A loss function indicates how much precision we lost using the current model (weights)
- Most often used loss function in ANNs is sum of squares
 - Small errors have much less influence/penalty than large errors

Gradient descent of the loss function

- Gradient descent of the loss function is the most commonly used technique for optimizing weights
 - Other techniques have been experimented on in the past (e.g. genetic algorithms, grid search, etc.)
- Visualize gravity
- Learning rate tells how far we step

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

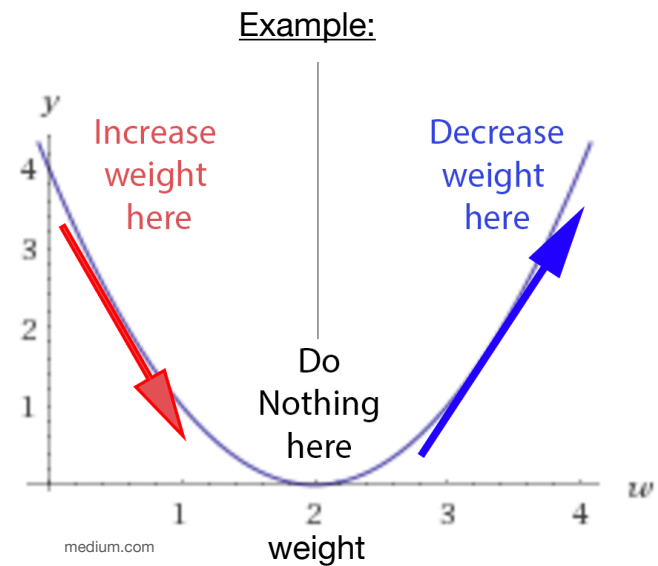


medium.com

Schematic of gradient descent.

Loss function derivative

- If the derivative is positive then loss will decrease if we decrease the weight
- If the derivative is negative then loss will decrease if we increase the weight
- If the derivative is zero then we do nothing





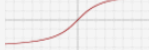






Loss function derivatives

		Propagation	Back-propagation
Square		$y = \frac{1}{2}(x - d)^2$	$\frac{\partial E}{\partial x} = (x - d)^T \frac{\partial E}{\partial y}$
Log	$c = \pm 1$	$y = \log(1 + e^{-cx})$	$\frac{\partial E}{\partial x} = \frac{-c}{1 + e^{cx}} \frac{\partial E}{\partial y}$
Hinge	$c = \pm 1$	$y = \max(0, m - cx)$	$\frac{\partial E}{\partial x} = -c \mathbb{I}\{cx < m\} \frac{\partial E}{\partial y}$
LogSoftMax	$c = 1 \dots k$	$y = \log(\sum_k e^{x_k}) - x_c$	$\left[\frac{\partial E}{\partial x}\right]_s = (e^{x_s} / \sum_k e^{x_k} - \delta_{sc}) \frac{\partial E}{\partial y}$
MaxMargin	$c = 1 \dots k$	$y = \left[\max_{k \neq c} \{x_k + m\} - x_c\right]_+$	$\left[\frac{\partial E}{\partial x}\right]_s = (\delta_{sk^*} - \delta_{sc}) \mathbb{I}\{E > 0\} \frac{\partial E}{\partial y}$

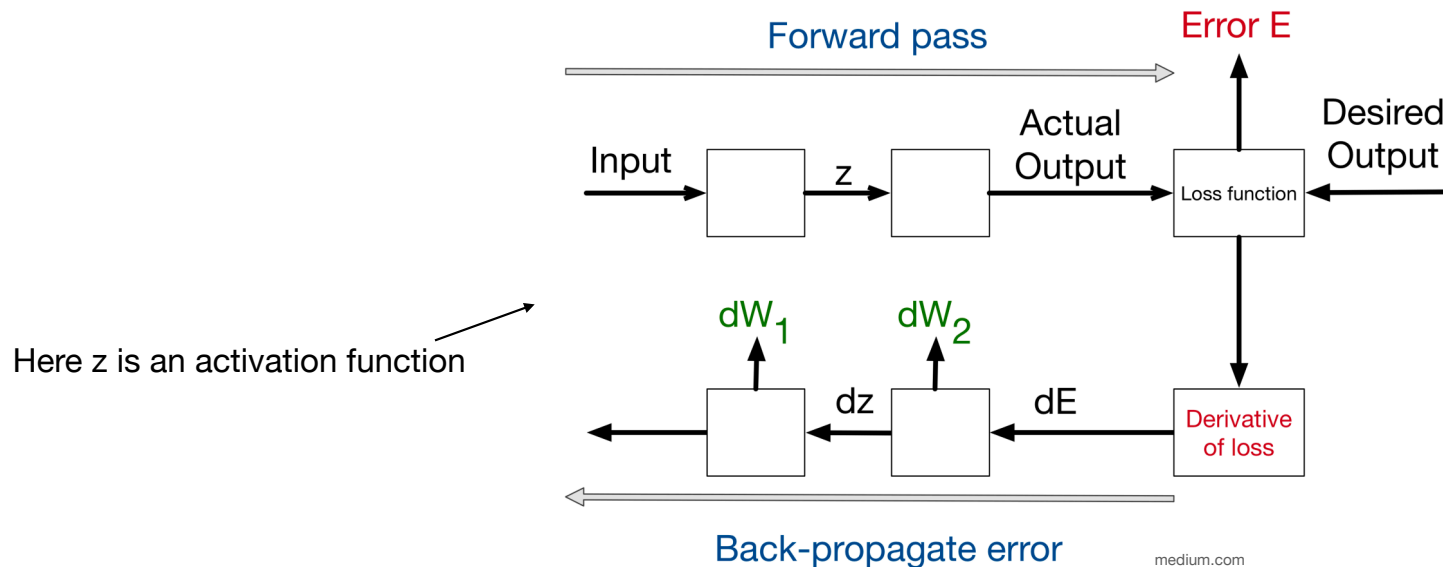
<https://erogol.com>

Activation function derivatives

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Back-propagation

- During back-propagation phase we update the weights in the direction opposite to the gradient of the loss function
- Using ReLU activation function makes back-propagation a lot more efficient as some weights do not need to be updated



Forward vs. back-propagation

- We apply the activation function directly to the inputs during forward propagation
- We apply the derivative of the loss/activation function during back-propagation

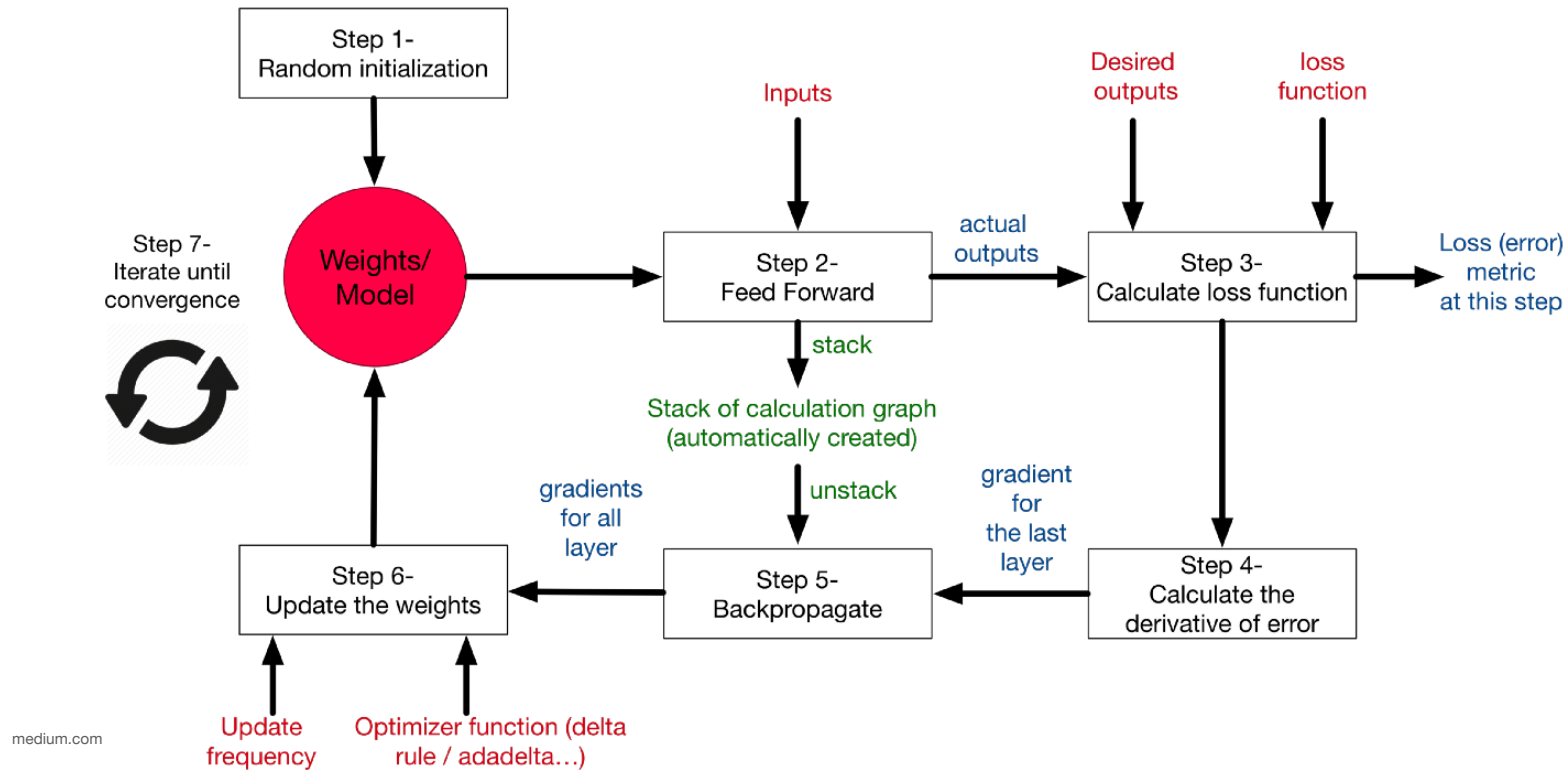
Tricks to make back-propagation efficient

- ANN learning usually heavily utilizes matrix manipulations
- We create a library of all the activation and loss derivatives
 - Most libraries already do this for us

Stopping conditions for training ANN

- The training of the ANN iterates
 - For a predetermined number of epochs
 - For a predetermined number of iterations
 - Until the convergence
 - The loss is small enough
 - Or the weight updates become small enough

A full picture of the ANN learning process



Let's watch a few great video explaining neural networks to better understand ANNs

- <https://www.youtube.com/watch?v=aircAruvnKk>
- <https://www.youtube.com/watch?v=IHZwWFHWA-w>
- <https://www.youtube.com/watch?v=llg3gGewQ5U>
- <https://www.youtube.com/watch?v=tleHLnjs5U8&t=197s>
- The deep learning book mentioned in the second video:
 - <http://neuralnetworksanddeeplearning.com/>

Let's play around with some network architectures

- <http://playground.tensorflow.org/>

A few thoughts about neural networks

- These are robust models for complex non-linear ML problems
- Work well for capturing associations and discovering regular patterns in the data
- No assumption of normality, linearity, and independence of input variables in the data
- Analytical alternative to some other ML methods
- Potential pitfalls
 - Rely on back-propagation to optimize weights
 - Not directly observable
 - Slow to train
 - Potential for memorizing data rather than learning a rule

Some additional resources for understanding neural networks

- Step by step calculations for a toy example
 - Includes both forward and back propagation calculations
 - Includes python code for coding the toy neural network used in the example
 - <https://victorzhou.com/blog/intro-to-neural-networks/>

Let's look at some code

- Classifying hand written digits using multi-layer perceptron
 - *MLP.MNIST.ipynb*