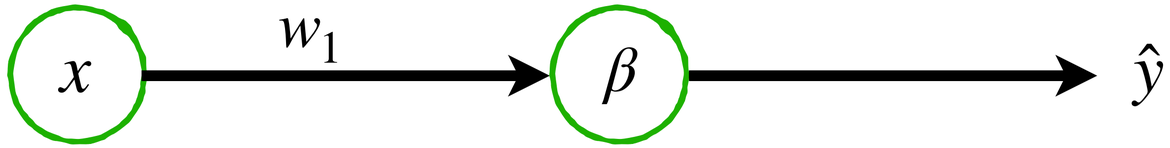
Neural Networks

Activation & Cost Functions

Rahul Singh
rsingh@arrsingh.com

Neural Networks

$$\hat{y} = \beta + w_1 x$$

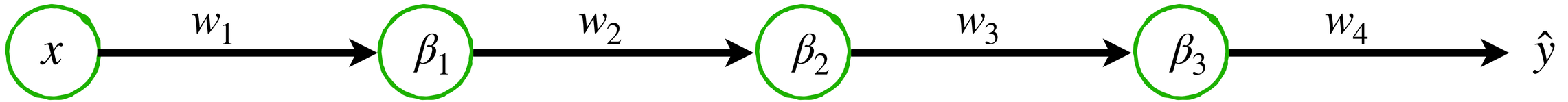


This is the simplest possible neural network

Multiply the input (x) with the Weight (w_1)
and add the Bias (β) to compute the output ($\hat{y}$)

Neural Networks

With multiple layers, the output at each Layer is the input to the next layer



$$z_1 = \beta_1 + w_1 x$$
$$z_2 = \beta_2 + w_2 z_1$$
$$z_3 = \beta_3 + w_3 z_2$$
$$\hat{y} = \beta_4 + w_4 z_3$$

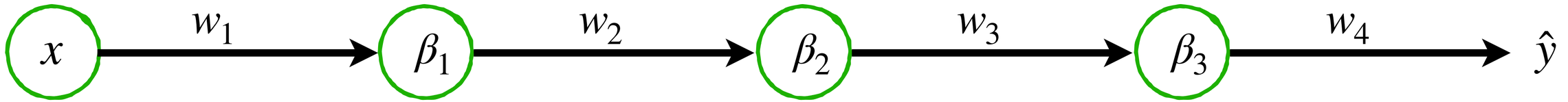
Multiply the input (x) with the Weight (w_1)
and add the Bias (β) to compute the output ($\hat{y}$)

Neural Networks

With multiple layers, the output at each Layer is the input to the next layer

However, adding multiple layers has no effect if we simply compute the linear combination of variables and weights at each layer

All the layers can be reduced to a single linear transformation

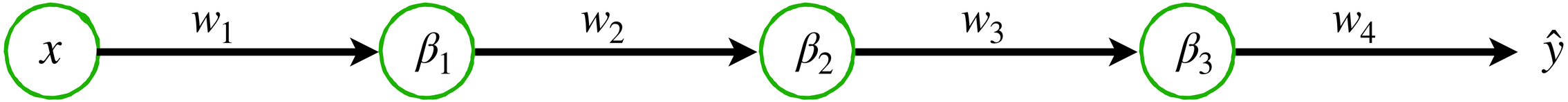


$$z_1 = \beta_1 + w_1 x$$
$$z_2 = \beta_2 + w_2 z_1$$
$$z_3 = \beta_3 + w_3 z_2$$
$$\hat{y} = \beta_4 + w_4 z_3$$

Multiply the input (x) with the Weight (w_1)
and add the Bias (β) to compute the output ($\hat{y}$)

Neural Networks

These two neural networks are equivalent



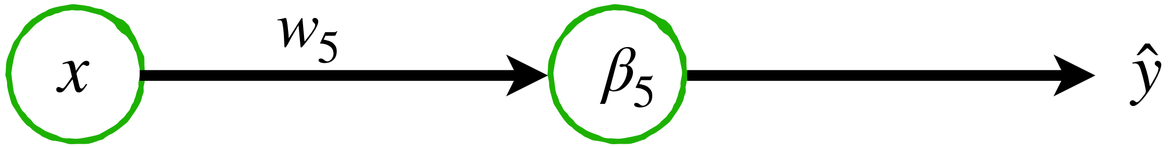
$$z_1 = \beta_1 + w_1 x$$

$$z_2 = \beta_2 + w_2 z_1$$

$$z_3 = \beta_3 + w_3 z_2$$

$$\hat{y} = \beta_4 + w_4 z_3$$

All the layers can be reduced to a single linear transformation



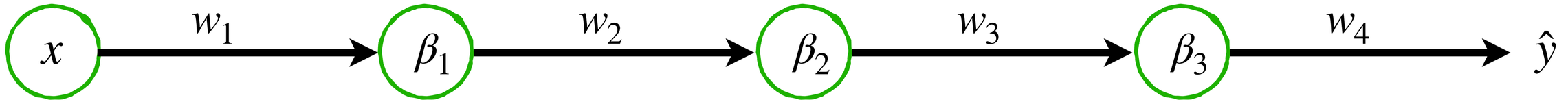
$$\hat{y} = \beta_5 + w_5 x$$

$$w_5 = w_1 w_2 w_3 w_4$$

$$\beta_5 = \beta_4 + w_4 \beta_3 + w_4 w_3 \beta_2 + w_4 w_3 w_2 \beta_1$$

Neural Networks

These two neural networks are equivalent



$$z_1 = \beta_1 + w_1 x$$

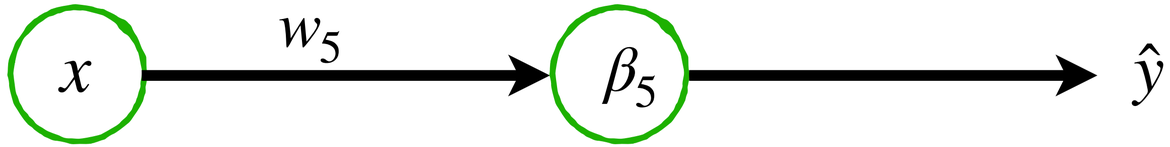
$$z_2 = \beta_2 + w_2 z_1$$

$$z_3 = \beta_3 + w_3 z_2$$

$$\hat{y} = \beta_4 + w_4 z_3$$

All the layers can be reduced to a single linear transformation

Simple Linear transformations can only fit simple lines and planes to data.
(lines in 2D, planes in 3D and hyper planes in higher dimensions)



$$\hat{y} = \beta_5 + w_5 x$$

$$w_5 = w_1 w_2 w_3 w_4$$

$$\beta_5 = \beta_4 + w_4 \beta_3 + w_4 w_3 \beta_2 + w_4 w_3 w_2 \beta_1$$

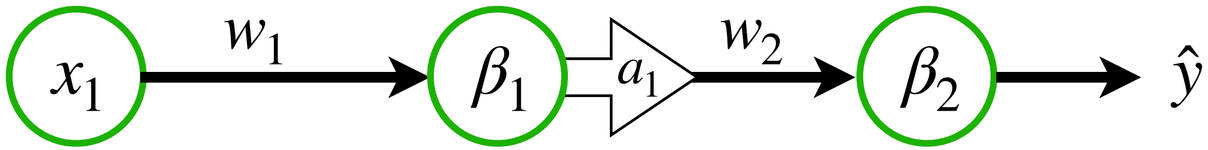
Neural Networks

To enable Neural Networks to learn complex patterns we have to introduce non - linearity

We need an activation function on each neuron…

Activation functions introduce non-linearity by converting the linear combinations of inputs into non-linear outputs.

A Simple Three Layer Neural Network



$n_0 = 1$ $n_1 = 1$ $n_2 = 1$

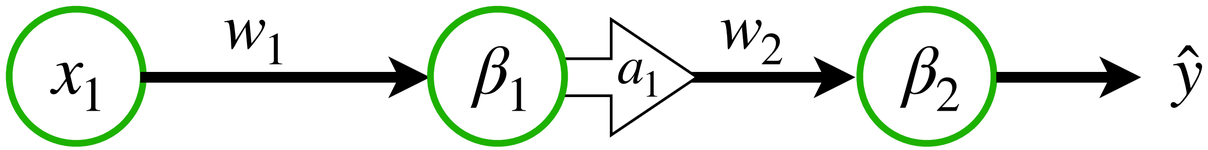
Input Layer (L_0) Hidden Layer (L_1) Output Layer (L_2)

$$z_1 = x_{11} w_1 + \beta_1$$
$$a_1 = f(z_1)$$
$$z_2 = a_1 w_2 + \beta_2$$
$$\hat{y} = f(z_2)$$

$f(x)$ is the activation function

Rather than simply multiplying the output at each layer and adding the bias, we first pass the input to an activation function f

A Simple Three Layer Neural Network



$n_0 = 1$ $n_1 = 1$ $n_2 = 1$

Input Layer (L_0) Hidden Layer (L_1) Output Layer (L_2)

$$z_1 = x_{11} w_1 + \beta_1$$
$$a_1 = f(z_1)$$
$$z_2 = a_1 w_2 + \beta_2$$
$$\hat{y} = f(z_2)$$

$f(x)$ is the activation function

Rather than simply multiplying the output at each layer and adding the bias, we first pass the input to an activation function f

Every neuron has its own activation function, however all neurons in a given layer have the same activation function

Neural Networks

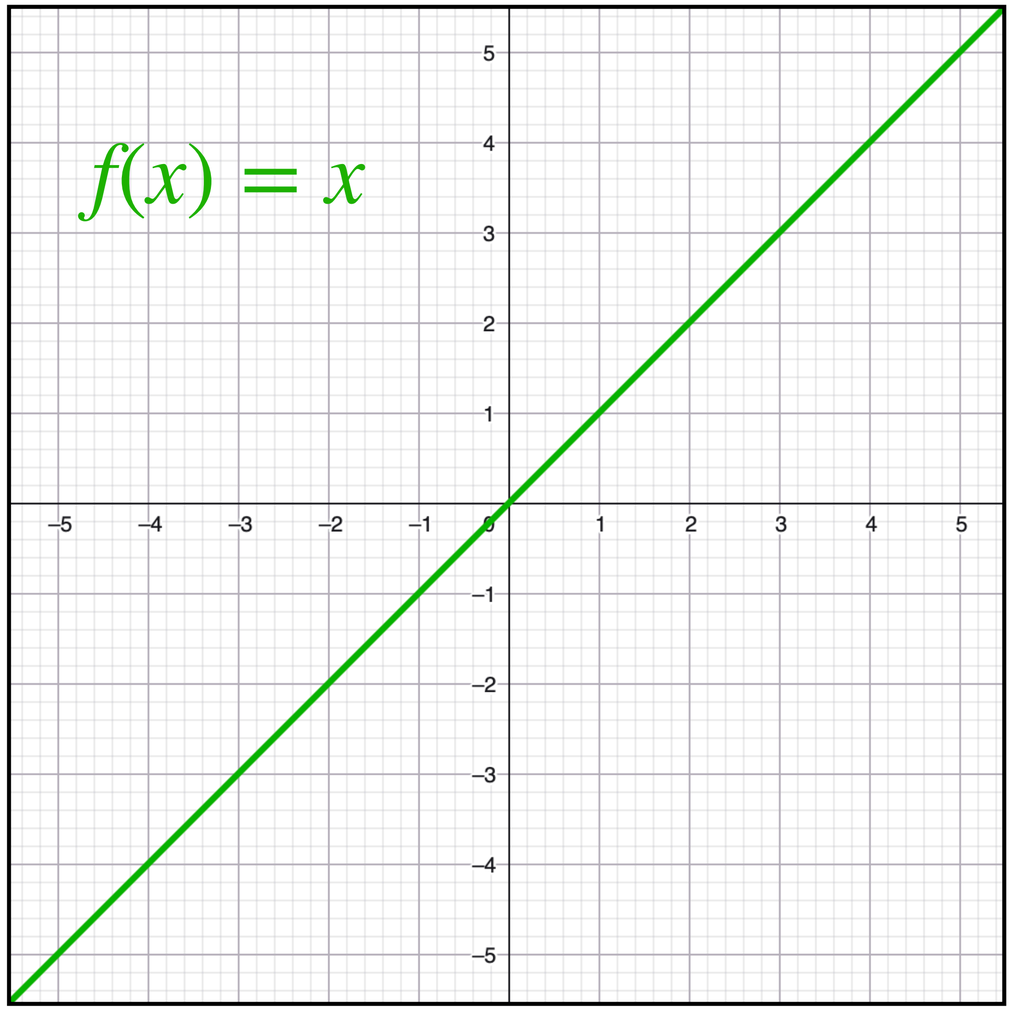
Lets go through the most common activation functions

Linear Activation Function

$$f(x) = x$$

The Linear function (also known as the identity function) doesn't transform the input. Its equivalent to having no activation.

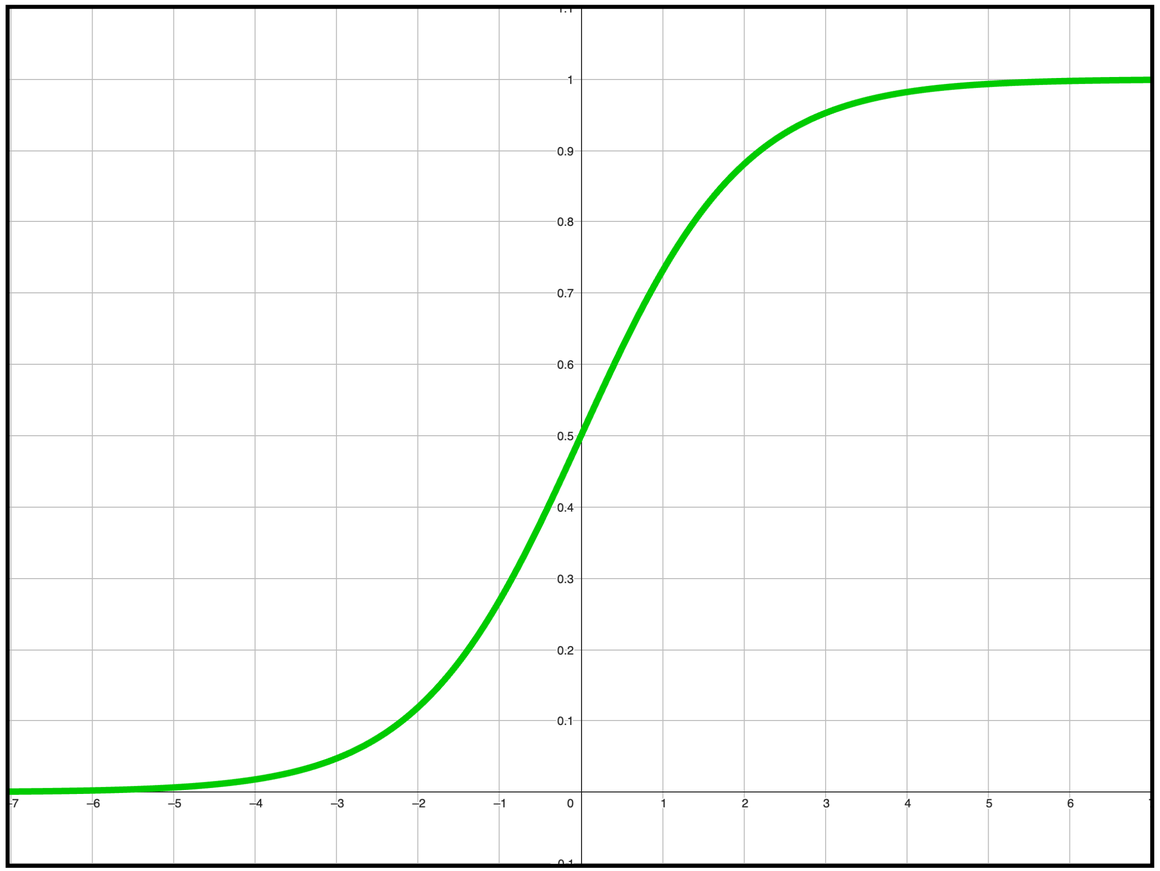
Commonly used in Linear Regression
(as we've seen in the previous slides)



Neural Networks

Sigmoid Activation Function

$$f(x) = \frac{1}{1 + e^{-x}}$$



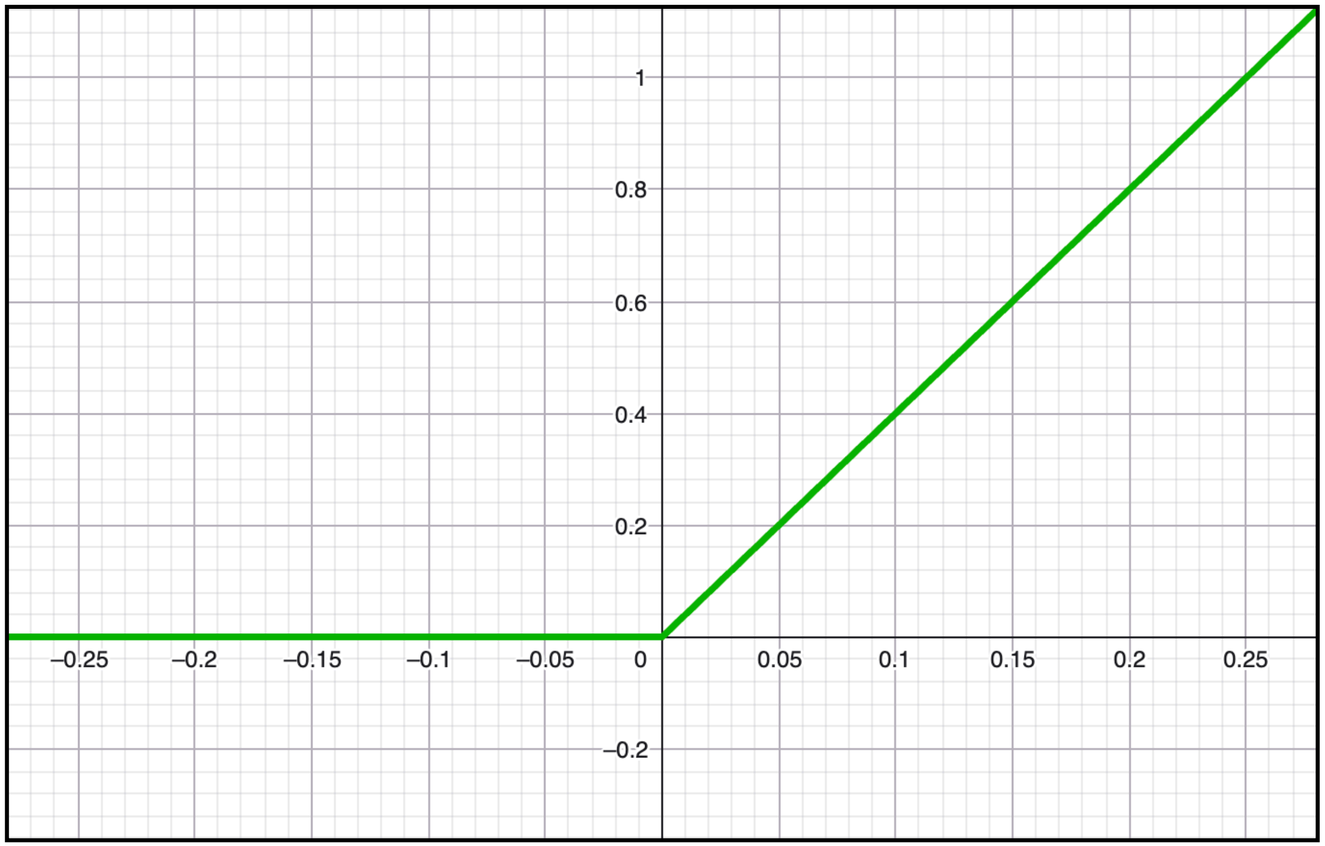
The Logistic function transforms the input to a range between 0 and 1.

Commonly used in the output layer for binary classification (also used in Logistic Regression)

ReLU Activation Function

Rectified Linear Unit

$$f(x) = max(0,x)$$



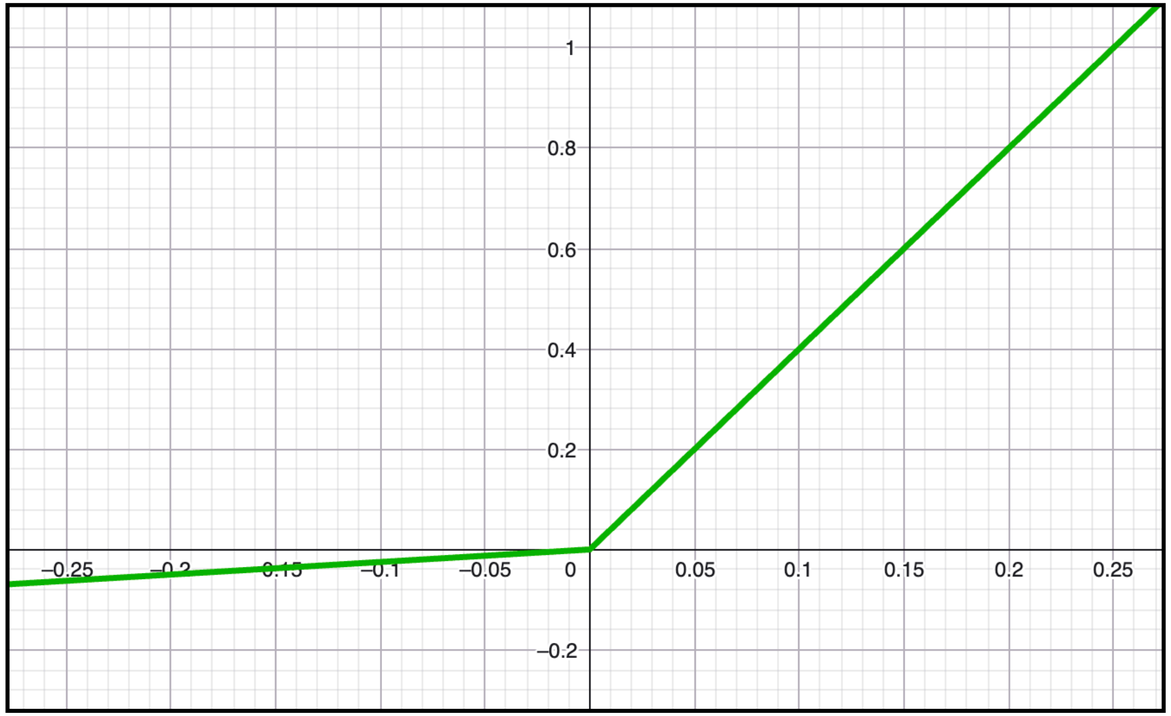
The ReLU function returns 0 if x < 0 and x otherwise. Transforms the input to a range between 0 and ∞

The ReLU function is typically used in hidden layers and is less computationally expensive than other activation functions (sigmoid, tanh)

Leaky ReLU Activation Function

Leaky Rectified Linear Unit

$$f(x) = \begin{cases} x, & if\, x > 0 \\ \alpha x, & if\, x \leq 0 \end{cases}$$



The Leaky ReLU function returns αx (instead of 0) if x < 0 which provides a small slope for negative values of x. Returns x for positive values of x

Helps provide a small gradient for negative values of x, which helps solve for the Dying ReLU problem where the gradient falls to zero and the neuron doesn't activate anymore and back propagation fails to update gradients any more.

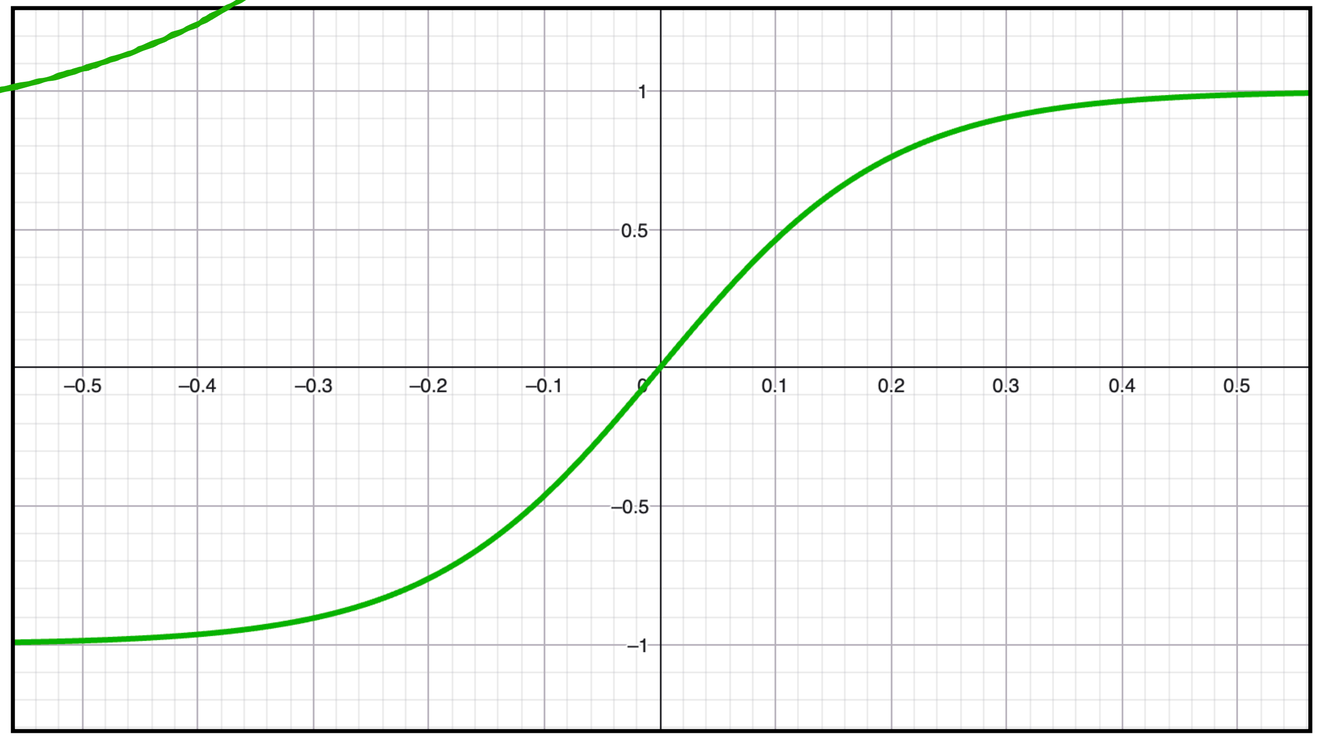
Tanh Activation Function

Hyperbolic Tangent

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

First derivative of $tanh(x)$

$$\frac{\partial}{\partial x} tanh(x) = 1 - tanh^2(x)$$



The Logistic function transforms the input to a range between -1 and +1.

Commonly used in the hidden layers given its zero centered output, which makes mapping the output to strong negative, neutral or strong positive

Neural Networks

Softmax Activation Function

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

K is the total number of classes

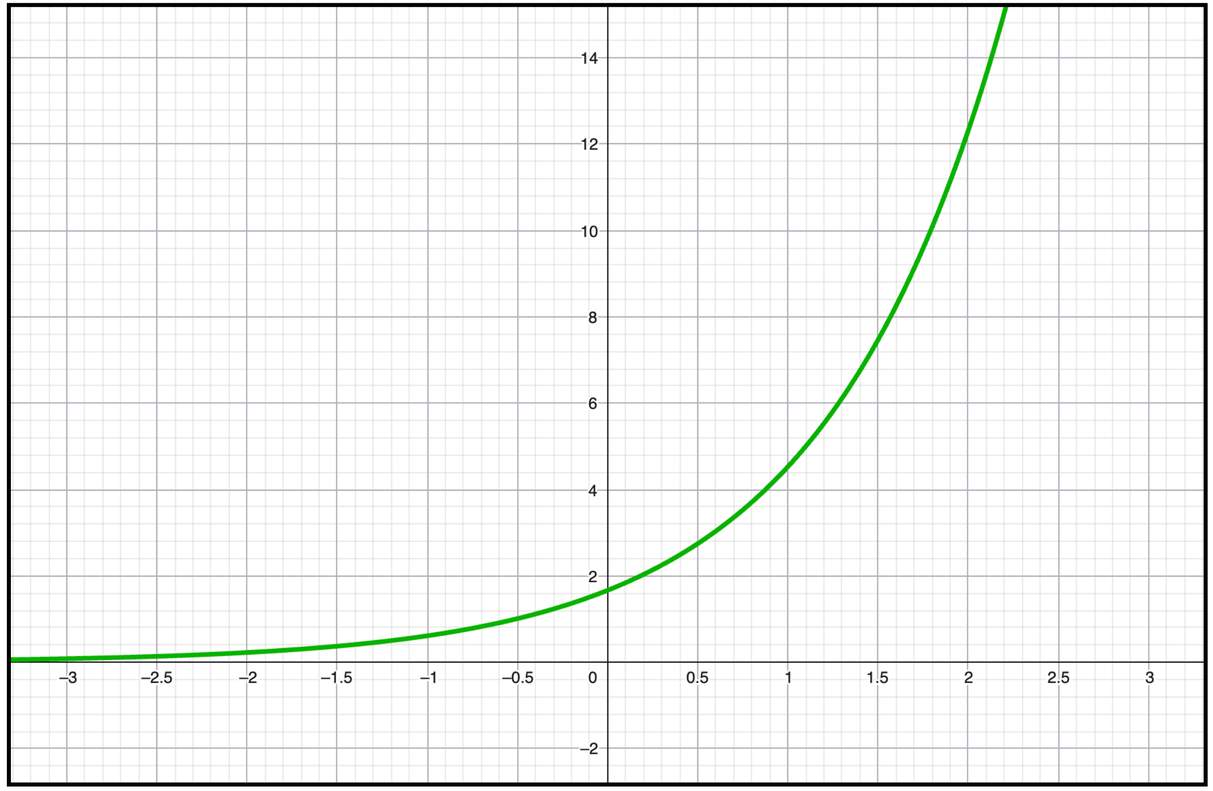
z_i is the output for the i^{th} class

e^{z_i} is the exponent of the i^{th} output

$\sum_{j=1}^{K} e^{z_j}$ is the sum of the exponents

All outputs are between 0 and 1 and sum to 1.0

The softmax function converts a tuple of K real numbers into a probability distribution over K



Typically used in the output layers for multi class classification

Neural Networks

Softmax Activation Function

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

K is the total number of classes

z_i is the output for the i^{th} class

e^{z_i} is the exponent of the i^{th} output

$\sum_{j=1}^{K} e^{z_j}$ is the sum of the exponents

All outputs are between 0 and 1 and sum to 1.0

Given a Neural Network with 3 output neurons for multi class classification (i.e. 3 classes)

Neural Networks

Softmax Activation Function

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

K is the total number of classes

z_i is the output for the i^{th} class

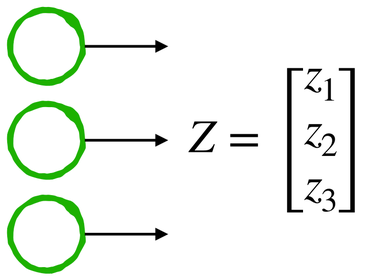
e^{z_i} is the exponent of the i^{th} output

$\sum_{j=1}^{K} e^{z_j}$ is the sum of the exponents

All outputs are between 0 and 1 and sum to 1.0

Given a Neural Network with 3 output neurons for multi class classification (i.e. 3 classes)

z_1, z_2, z_3 are the raw values from the outputs



Neural Networks

Softmax Activation Function

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

K is the total number of classes

z_i is the output for the i^{th} class

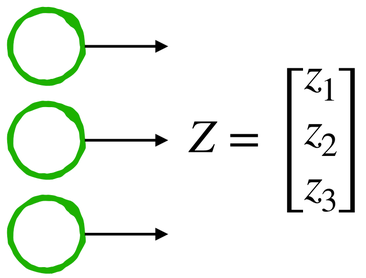
e^{z_i} is the exponent of the i^{th} output

$\sum_{j=1}^{K} e^{z_j}$ is the sum of the exponents

All outputs are between 0 and 1 and sum to 1.0

Given a Neural Network with 3 output neurons for multi class classification (i.e. 3 classes)

z_1, z_2, z_3 are the raw values from the outputs



Compute e^{z_i} for each z_i

Compute the Sum of the exponents:

$$\sum_{j=1}^{K} = e^{z_1} + e^{z_2} + e^{z_3}$$

Softmax: Divide each e^{z_i} by $\sum_{j=1}^{K}$

Neural Networks

Softmax Activation Function

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

K is the total number of classes

z_i is the output for the i^{th} class

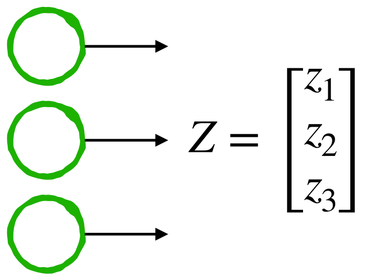
e^{z_i} is the exponent of the i^{th} output

$\sum_{j=1}^{K} e^{z_j}$ is the sum of the exponents

All outputs are between 0 and 1 and sum to 1.0

Given a Neural Network with 3 output neurons for multi class classification (i.e. 3 classes)

z_1, z_2, z_3 are the raw values from the outputs



$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$$softmax(Z) = \begin{bmatrix} \frac{e^{z1}}{e^{z1} + e^{z2} + e^{z3}} \\ \frac{e^{z2}}{e^{z1} + e^{z2} + e^{z3}} \\ \frac{e^{z3}}{e^{z1} + e^{z2} + e^{z3}} \end{bmatrix}$$

Compute e^{z_i} for each z_i

Compute the Sum of the exponents:

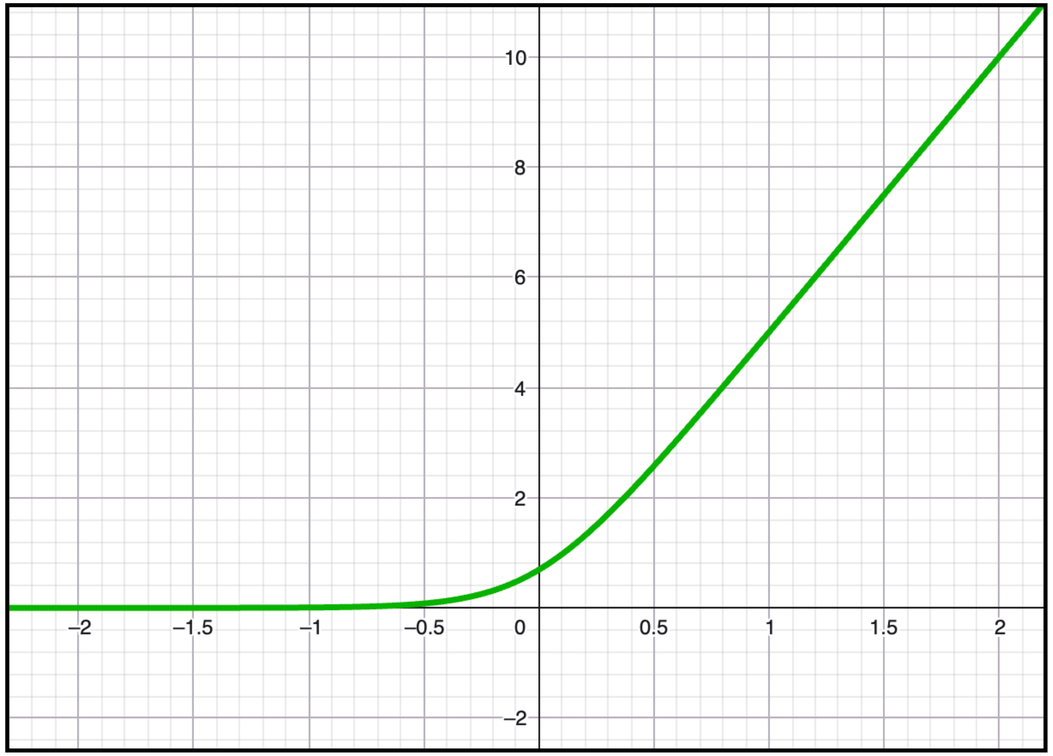
$$\sum_{j=1}^{K} = e^{z_1} + e^{z_2} + e^{z_3}$$

Softmax: Divide each e^{z_i} by $\sum_{j=1}^{K}$

Softplus Activation Function

$$f(x) = log_e(1 + e^x)$$

The Softplus function transforms the input to a range between 0 and ∞. The output is always positive and differentiable at all points (unlike ReLU which is not differentiable at $x = 0$)



Softmax is typically used in hidden layers and avoids the hard transition of ReLU (at $x = 0$)

Swish Activation Function

$$f(x) = \frac{x}{1 + e^{-\beta x}}$$

Continuous and differentiable at every point. Transforms the input to a range between 0 and ∞.

Swish is typically used in hidden layers and also avoids the hard transition of ReLU (at $x = 0$)

Neural Networks

In addition to activation functions we also need a cost function

Different types of problems, require different cost functions:

Lets look at the different types of problems and the cost functions for each

Linear Regression

Neural networks used for regression, predict continuous values and have one neuron in the output layer for each continuous value being predicted

Cost function is the **Mean Squared Error (MSE):**

$$\frac{1}{n}\| Y - X\beta \|^2$$

Sometimes **Mean Absolute Error** is also used

$$\frac{1}{n}\| Y - X\beta \|$$

For more details see the tutorial on Multiple Regression

Binary Classification

Neural networks used for binary classification, predict one of two discrete values and have one neuron in the output layer with a sigmoid activation function

Cost function is the **Binary Cross Entropy:**

$$-\frac{1}{n}\sum_{i=1}^{n} y\, log_e \hat{y} + (1-y)\, log_e(1-\hat{y})$$

For more details see the tutorial on Logistic Regression

Multi - Class Classification

Neural networks used for multi class classification, predict one of several classes and have one neuron in the output layer for each class

Cost function is the **Categorical Cross Entropy:**

$$-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{K} y_{ij}\, log_e \hat{y}_{ij}$$

Softmax is typically used as the activation function in the output layer

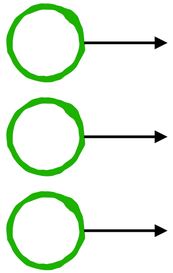
y_{ij} is the observed label in the one hot representation of the j^{th} class in the target vector for the i^{th} observation
$\hat{y}_{ij}$ is the predicted probability for the j^{th} class for the i^{th} observation
m is the total number of observations
K is the total number of classes

Multi - Class Classification

Neural networks used for multi class classification, predict one of several classes and have one neuron in the output layer for each class

Cost function is the **Categorical Cross Entropy:**

$$-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{K} y_{ij}\, log_e \hat{y}_{ij}$$



$$Y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \hat{Y} = \begin{bmatrix} 0.1 \\ 0.65 \\ 0.3 \end{bmatrix} \quad y_{1j} log(\hat{y}_{1j}) = \begin{bmatrix} 0 \times log(0.1) \\ 1 \times log(0.65) \\ 0 \times log(0.3) \end{bmatrix}$$

Negate and average over m samples

y_{ij} is the observed label in the one hot representation of the j^{th} class in the target vector for the i^{th} observation

$\hat{y}_{ij}$ is the predicted probability for the j^{th} class for the i^{th} observation

m is the total number of observations

K is the total number of classes

Note: log of a number between 0 and 1 is a negative number so we negate to ensure that loss is positive.

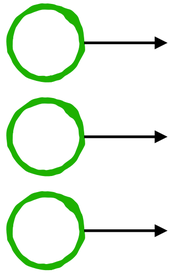
Neural Networks

Multi - Class Classification

Neural networks used for multi class classification, predict one of several classes and have one neuron in the output layer for each class

Cost function is the **Categorical Cross Entropy:**

$$-\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{K} y_{ij}\, log_e \hat{y}_{ij}$$



$$\hat{Y} = softmax(Z) = \begin{bmatrix} \frac{e^{z1}}{e^{z1}+e^{z2}+e^{z3}} \\ \frac{e^{z2}}{e^{z1}+e^{z2}+e^{z3}} \\ \frac{e^{z3}}{e^{z1}+e^{z2}+e^{z3}} \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \hat{Y} = \begin{bmatrix} 0.1 \\ 0.65 \\ 0.3 \end{bmatrix} \quad y_{1j} log(\hat{y}_{1j}) = \begin{bmatrix} 0 \times log(0.1) \\ 1 \times log(0.65) \\ 0 \times log(0.3) \end{bmatrix}$$

Negate and average over m samples

y_{ij} is the observed label in the one hot representation of the j^{th} class in the target vector for the i^{th} observation

$\hat{y}_{ij}$ is the predicted probability for the j^{th} class for the i^{th} observation

m is the total number of observations

K is the total number of classes

Note: log of a number between 0 and 1 is a negative number so we negate to ensure that loss is positive.

Related Tutorials & Textbooks

Neural Networks

An introduction to Neural Networks starting from a foundation of linear regression, logistic classification and multi class classification models along with the matrix representation of a neural network generalized to *l* layers with *n* neurons

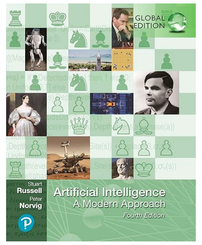
Forward and Back Propagation in Neural Networks

A deep dive into how Neural Networks are trained using Gradient Descent. Output predictions, are compared to observations to calculate loss and Backward propagation then computes gradients by working backward through the network

Gradient Descent for Multiple Regression

Gradient Descent algorithm for multiple regression and how it can be used to optimize k + 1 parameters for a Linear model in multiple dimensions.

Recommended Textbooks



Artificial Intelligence: A Modern Approach
by Peter Norvig, Stuart Russell

For a complete list of tutorials see:
https://arrsingh.com/ai-tutorials