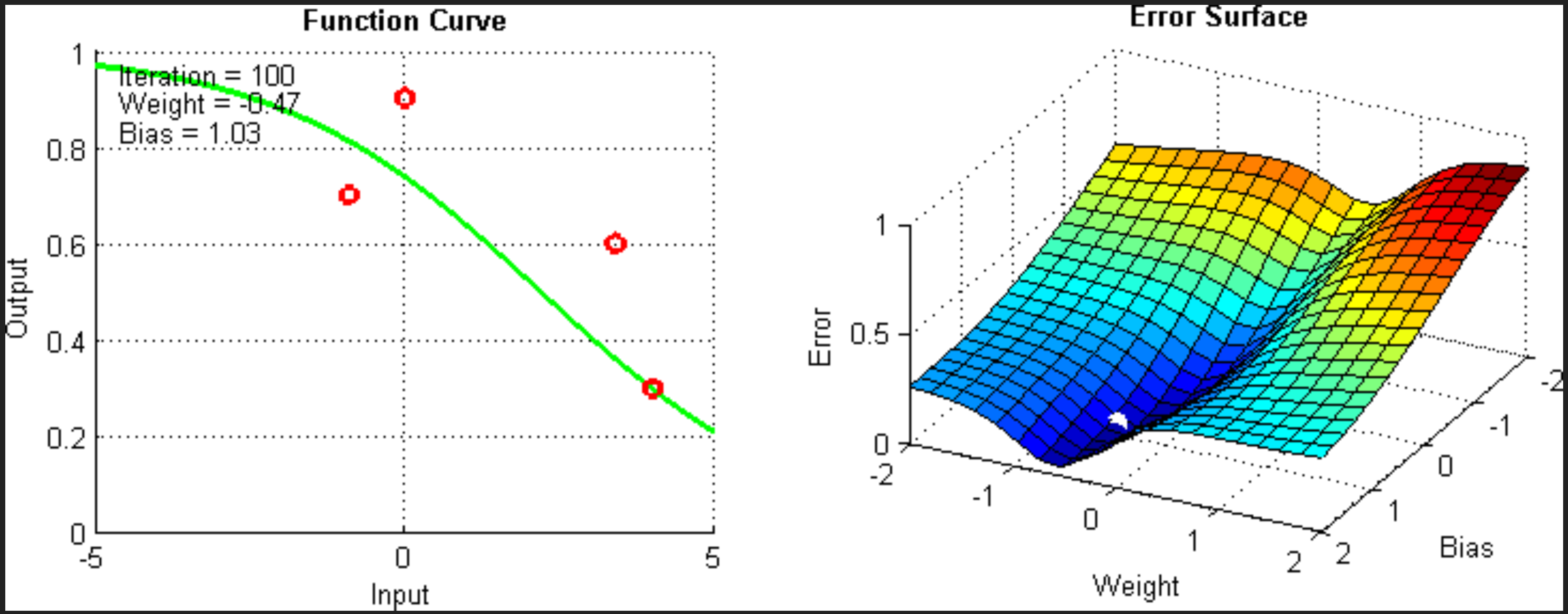
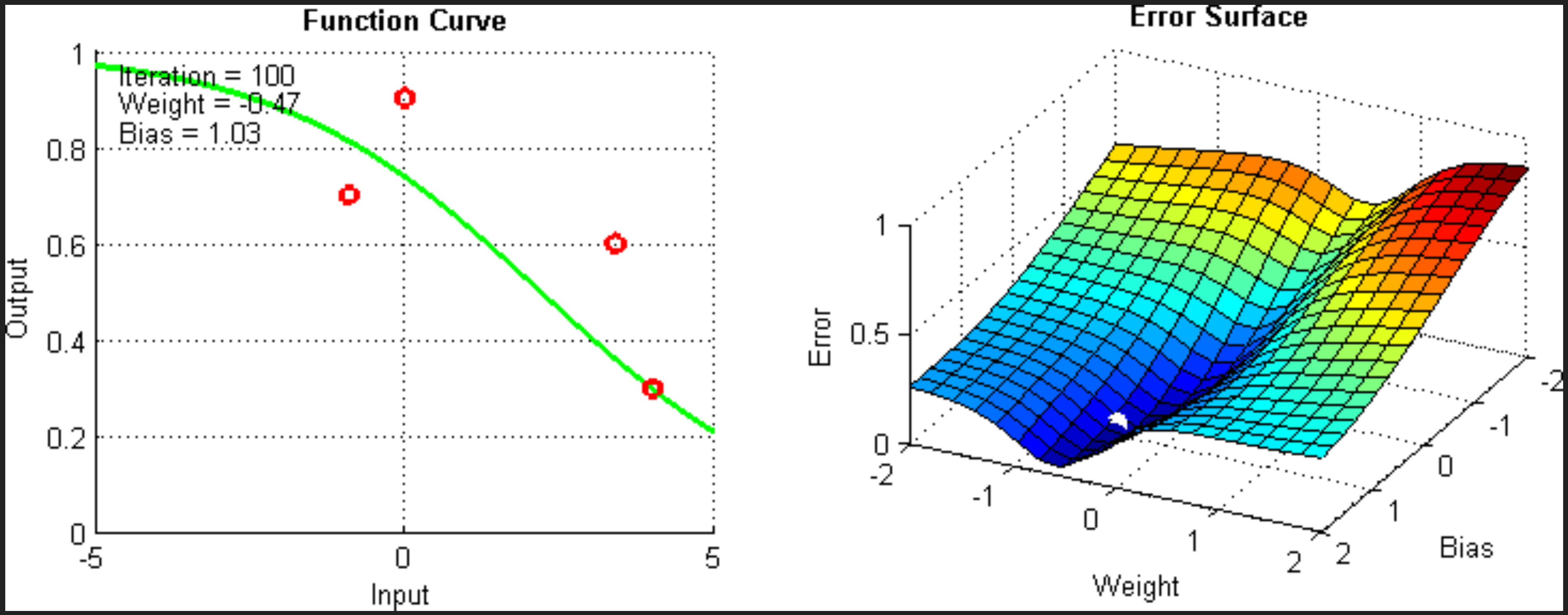


LOSSES, OPTIMISERS & ACTIVATIONS



LOSSES, OPTIMISERS & ACTIVATIONS

LOSS FUNCTIONS

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

- L2 Loss - Least Square Errors

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

- L2 Loss - Least Square Errors

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

- L2 Loss - Least Square Errors

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

- Binary Cross Entropy Loss (Sigmoid Loss)

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

- L2 Loss - Least Square Errors

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

- Binary Cross Entropy Loss (Sigmoid Loss)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

- L2 Loss - Least Square Errors

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

- Binary Cross Entropy Loss (Sigmoid Loss)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

- Cross Entropy Loss

LOSS FUNCTIONS

- L1 Loss - Least Absolute Deviations

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

- L2 Loss - Least Square Errors

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

- Binary Cross Entropy Loss (Sigmoid Loss)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))]$$

- Cross Entropy Loss

$$H(p,q) = -\sum_{x} p(x)\,\log q(x)$$

USECASES

USECASES

- L1 Loss: Dataset with lots of outliers

USECASES

- L1 Loss: Dataset with lots of outliers
- L2 Loss: Complex Function Mapping e.g. Regression

USECASES

- L1 Loss: Dataset with lots of outliers
- L2 Loss: Complex Function Mapping e.g. Regression
- BCE Loss: Classification with target variable having only two classes

USECASES

- L1 Loss: Dataset with lots of outliers
- L2 Loss: Complex Function Mapping e.g. Regression
- BCE Loss: Classification with target variable having only two classes
- CE Loss: Classification with target variable having multiple classes

OPTIMISERS

VANILLA GRADIENT DESCENT

VANILLA GRADIENT DESCENT

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

VANILLA GRADIENT DESCENT

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$
$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

- ALL samples at once

VANILLA GRADIENT DESCENT

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$
$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

- ALL samples at once
- Iterative

VANILLA GRADIENT DESCENT

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

- ALL samples at once
- Iterative
- First Order Differentiation

VANILLA GRADIENT DESCENT

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

- ALL samples at once
- Iterative
- First Order Differentiation
- Also called Batch Gradient Descent

VANILLA GRADIENT DESCENT

Function:

$$J(\theta_1, \theta_2) = {\theta_1}^2 + {\theta_2}^2$$

Objective:

$$\min_{\theta_1, \theta_2} J(\theta_1, \theta_2)$$

Update rules:

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1, \theta_2)$$

$$\theta_2 := \theta_2 - \alpha \frac{d}{d\theta_2} J(\theta_1, \theta_2)$$

Derivatives:

$$\frac{d}{d\theta_1} J(\theta_1, \theta_2) = \frac{d}{d\theta_1} {\theta_1}^2 + \frac{d}{d\theta_1} {\theta_2}^2 = 2\theta_1$$

$$\frac{d}{d\theta_2} J(\theta_1, \theta_2) = \frac{d}{d\theta_2} {\theta_1}^2 + \frac{d}{d\theta_2} {\theta_2}^2 = 2\theta_2$$

- ALL samples at once
- Iterative
- First Order Differentiation
- Also called Batch Gradient Descent
- Running through ALL samples can be very computationally expensive!

STOCHASTIC GRADIENT DESCENT

STOCHASTIC GRADIENT DESCENT

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$
Require: Initial parameter $\boldsymbol{\theta}$
$k \leftarrow 1$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
$k \leftarrow k + 1$
end while

STOCHASTIC GRADIENT DESCENT

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

- Mini-batch of m examples

STOCHASTIC GRADIENT DESCENT

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

- Mini-batch of m examples
- Iterative

STOCHASTIC GRADIENT DESCENT

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

- Mini-batch of m examples
- Iterative
- First Order Differentiation

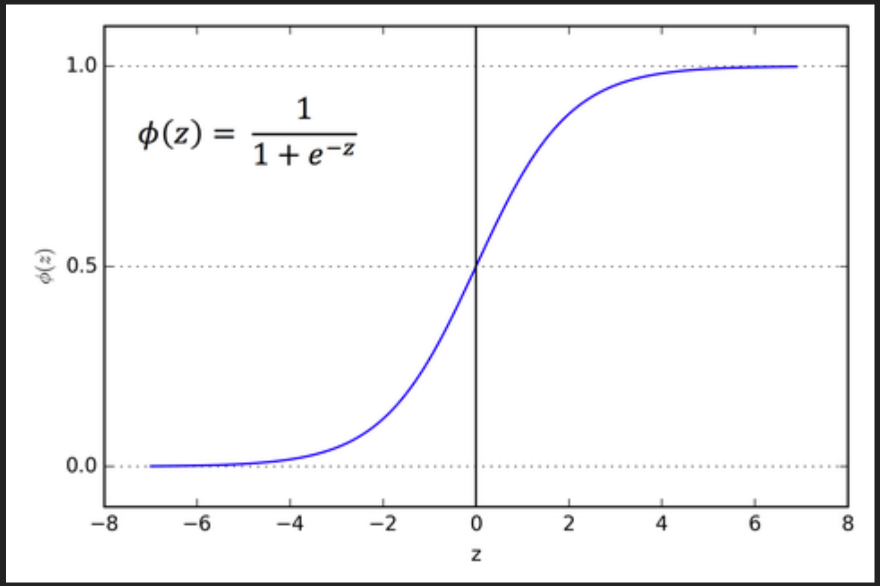
STOCHASTIC GRADIENT DESCENT

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

- Mini-batch of m examples
- Iterative
- First Order Differentiation
- Regularisation effect due to mini-batch

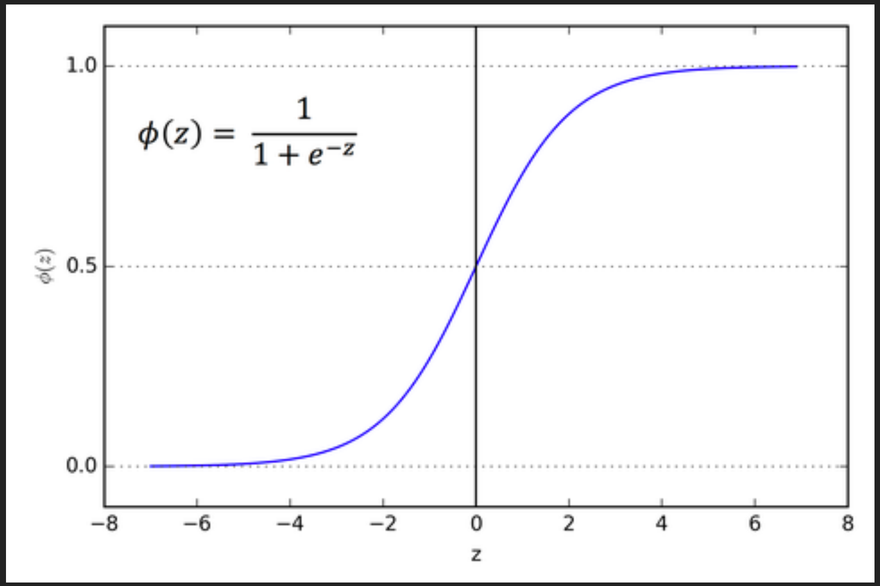
ACTIVATIONS

ACTIVATIONS – SIGMOID

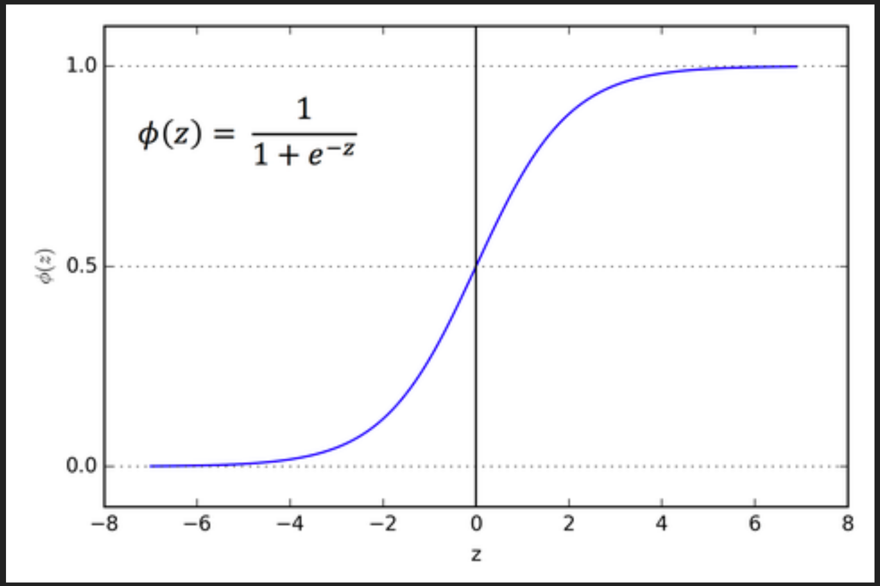


ACTIVATIONS - SIGMOID

- Good for Binary Classification

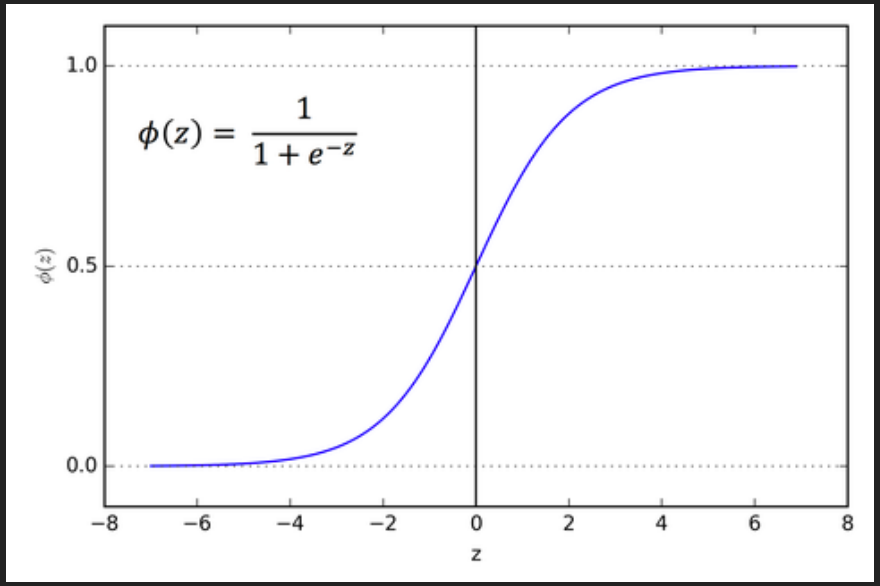


ACTIVATIONS – SIGMOID



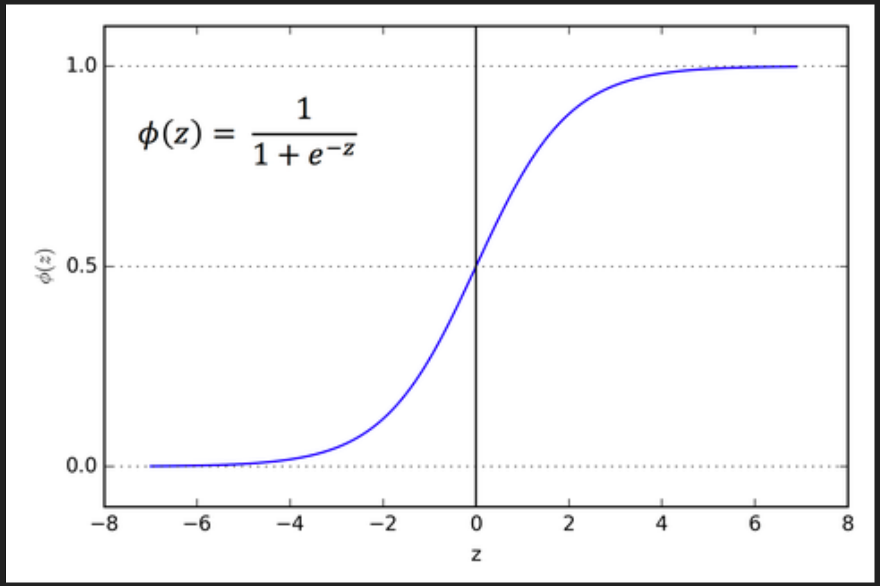
- Good for Binary Classification
- Softmax - Extension to Multi Class Classification

ACTIVATIONS – SIGMOID



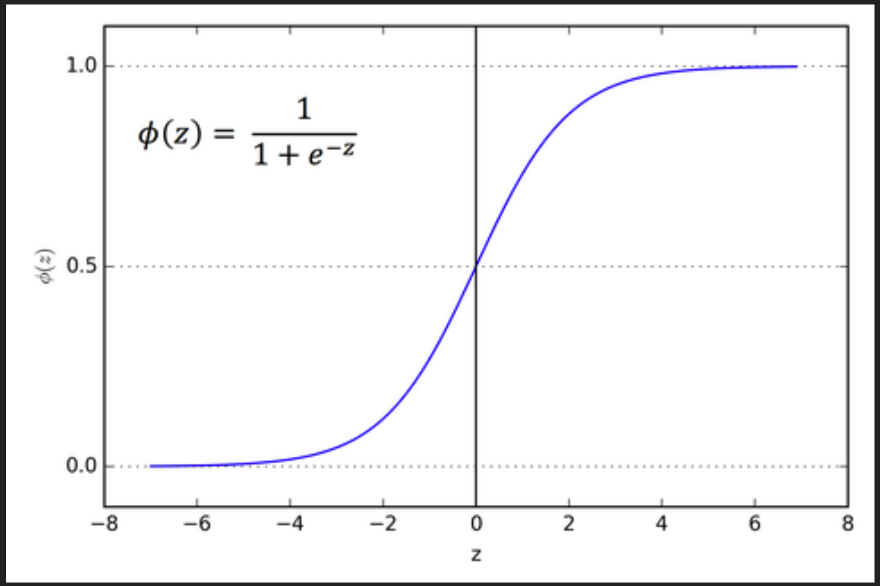
- Good for Binary Classification
- Softmax - Extension to Multi Class Classification
- Saturated Neurons - Kills gradients

ACTIVATIONS – SIGMOID



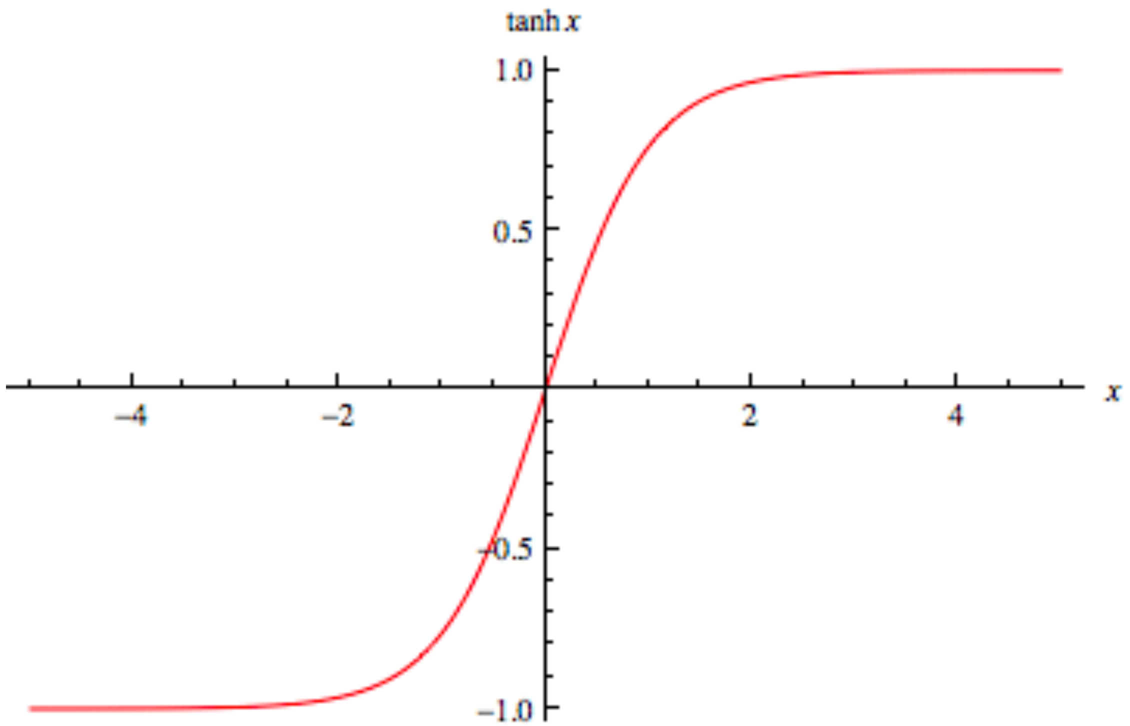
- Good for Binary Classification
- Softmax - Extension to Multi Class Classification
- Saturated Neurons - Kills gradients
- Not zero centered

ACTIVATIONS – SIGMOID



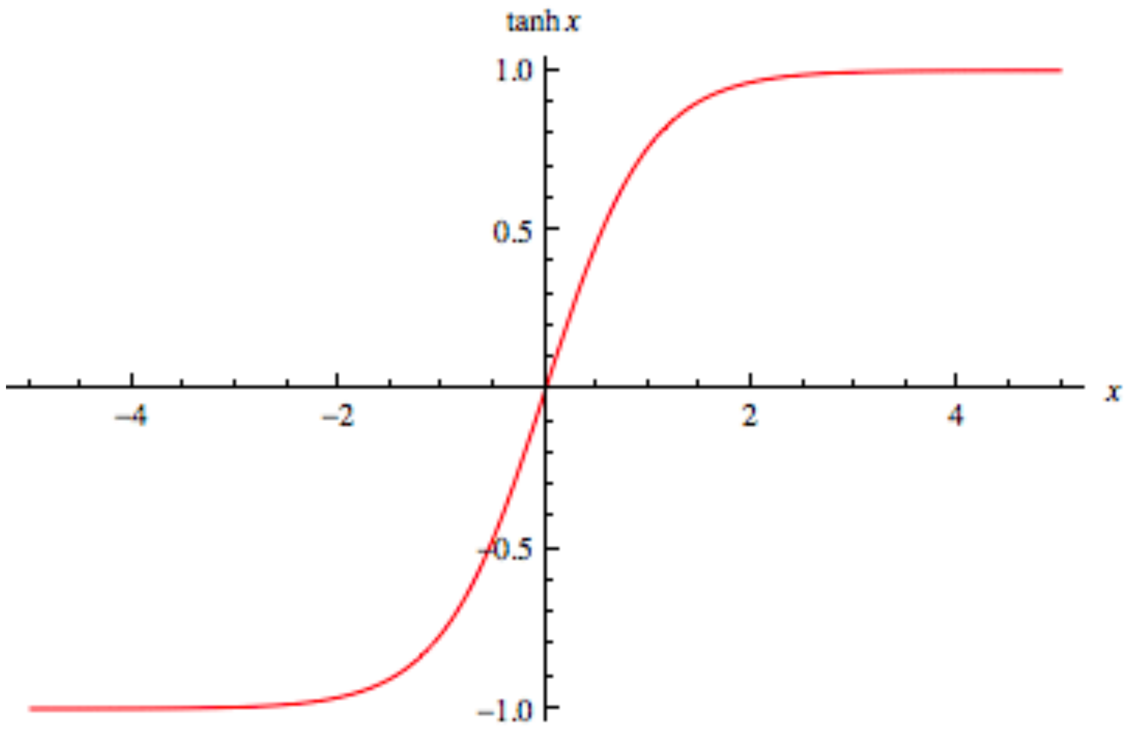
- Good for Binary Classification
- Softmax - Extension to Multi Class Classification
- Saturated Neurons - Kills gradients
- Not zero centered
- Exp() - Computationally expensive

ACTIVATIONS – TANH



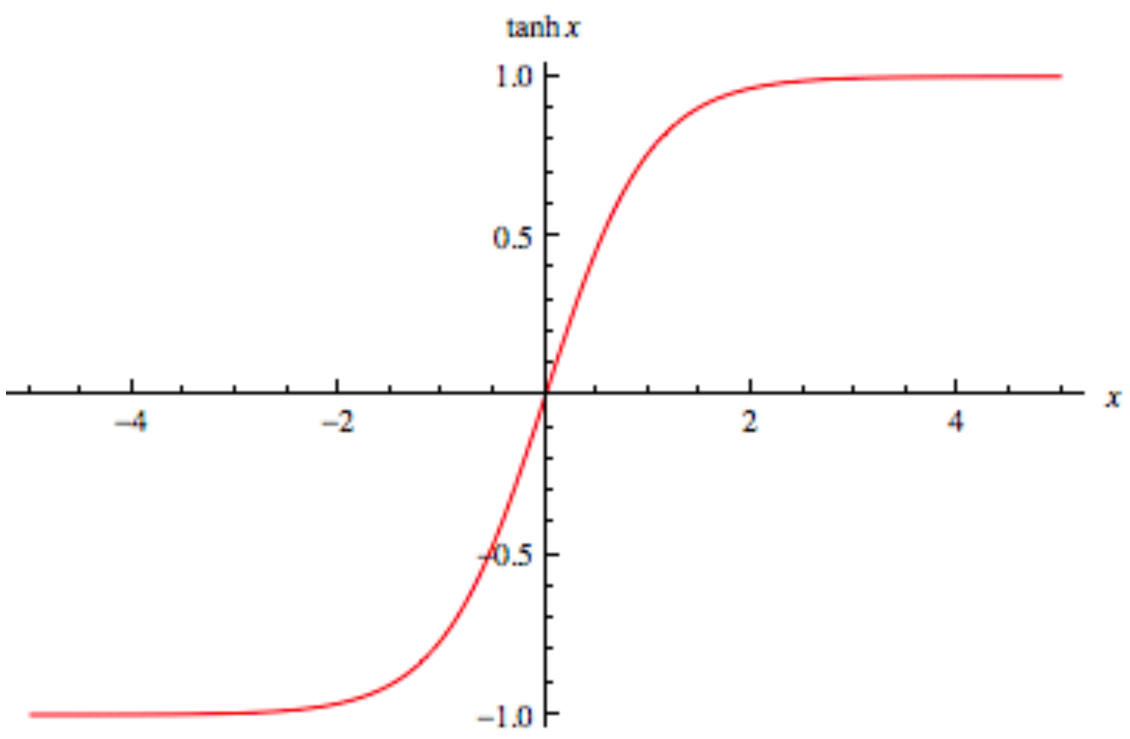
ACTIVATIONS – TANH

- Zero - centered



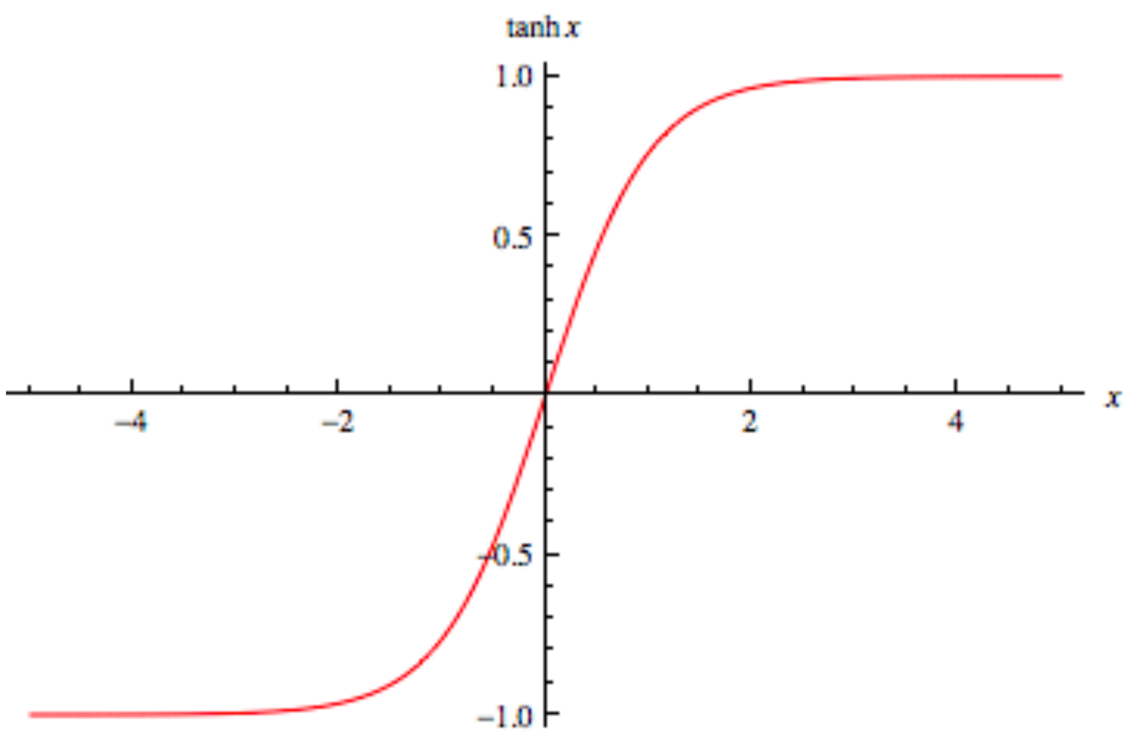
ACTIVATIONS – TANH

- Zero - centered
- Used in Recurrent Neural Networks

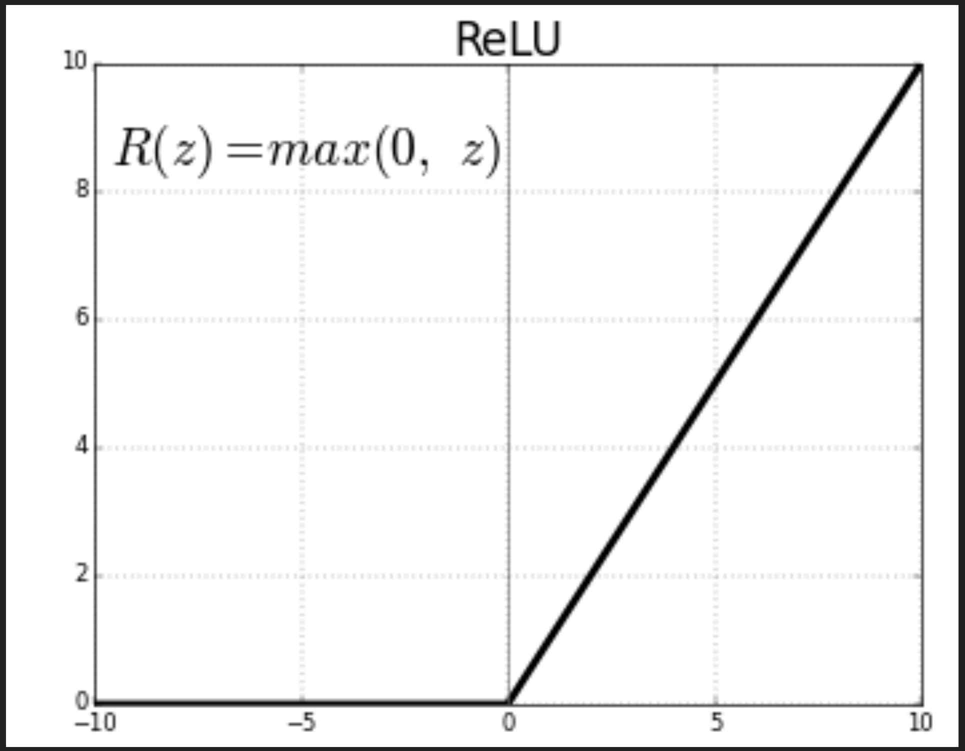


ACTIVATIONS – TANH

- Zero - centered
- Used in Recurrent Neural Networks
- Saturated Neurons - Kill gradients

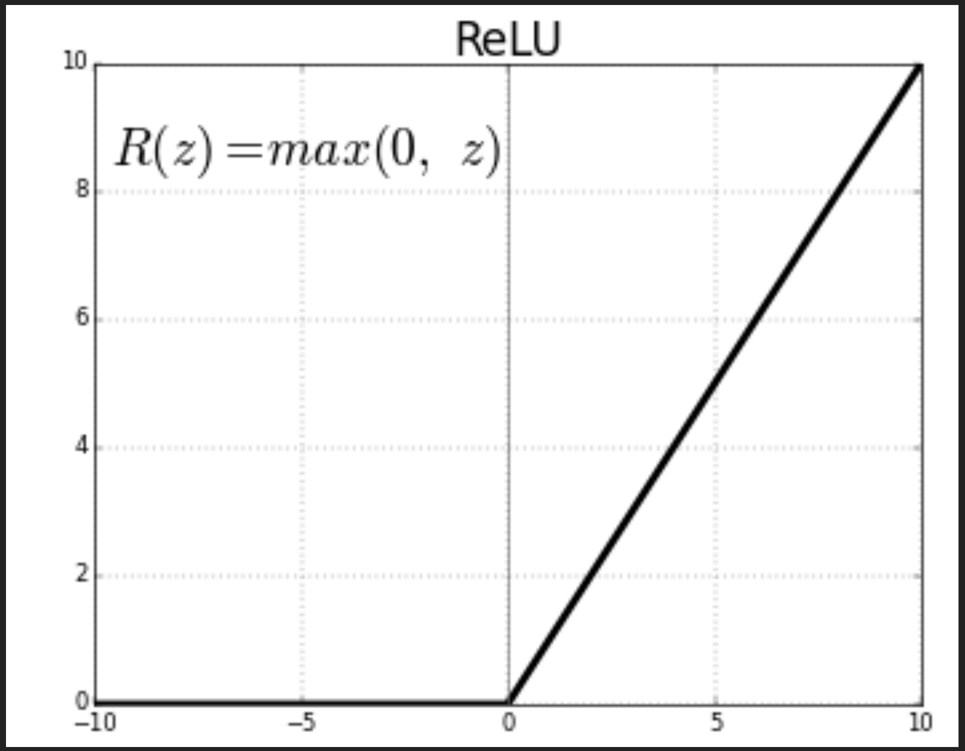


ACTIVATIONS – RECTIFIED LINEAR UNIT(RELU)



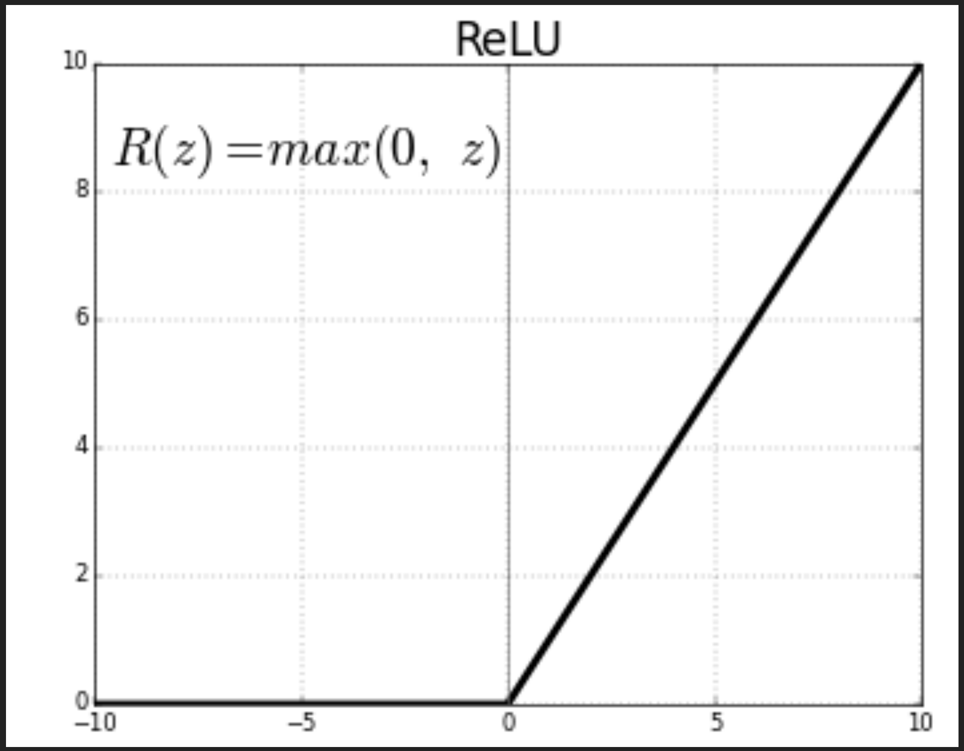
ACTIVATIONS – RECTIFIED LINEAR UNIT(RELU)

- Neurons do not get saturated. Faster Convergence. Computationally efficient.



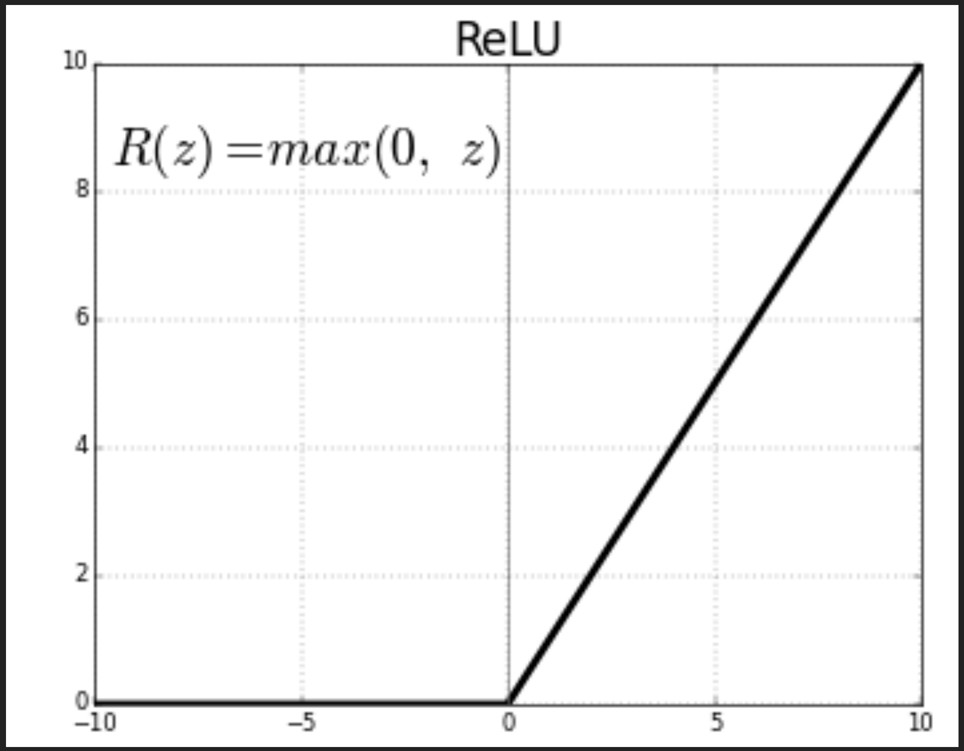
ACTIVATIONS – RECTIFIED LINEAR UNIT(RELU)

- Neurons do not get saturated. Faster Convergence. Computationally efficient.
- Not zero-centered



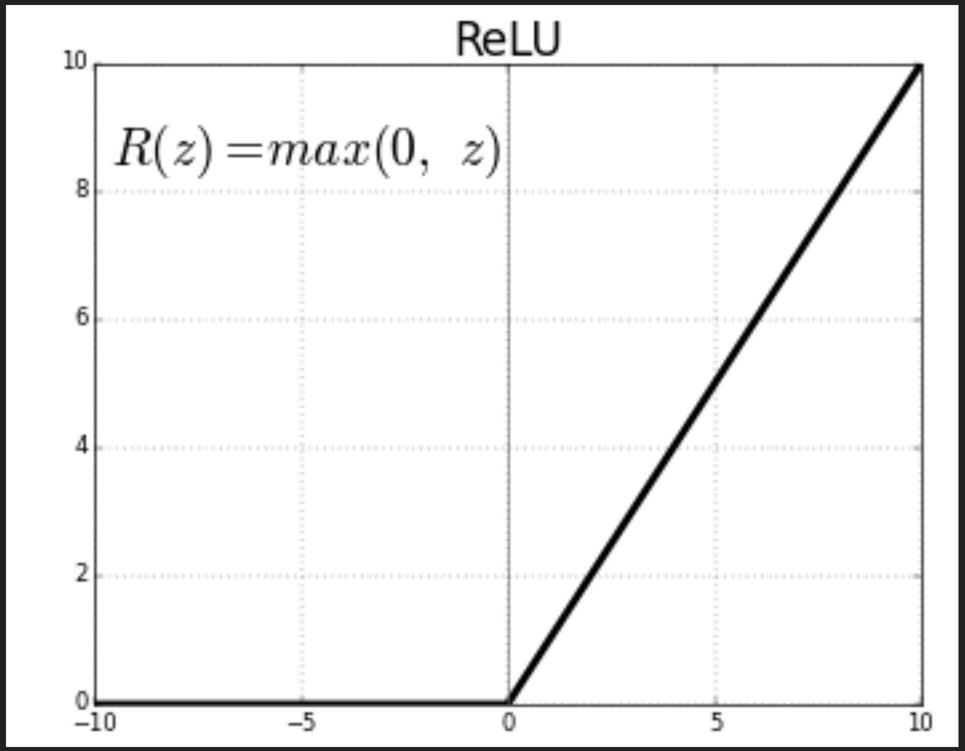
ACTIVATIONS – RECTIFIED LINEAR UNIT(RELU)

- Neurons do not get saturated. Faster Convergence. Computationally efficient.
- Not zero-centered
- Gradient is 0 for x<0



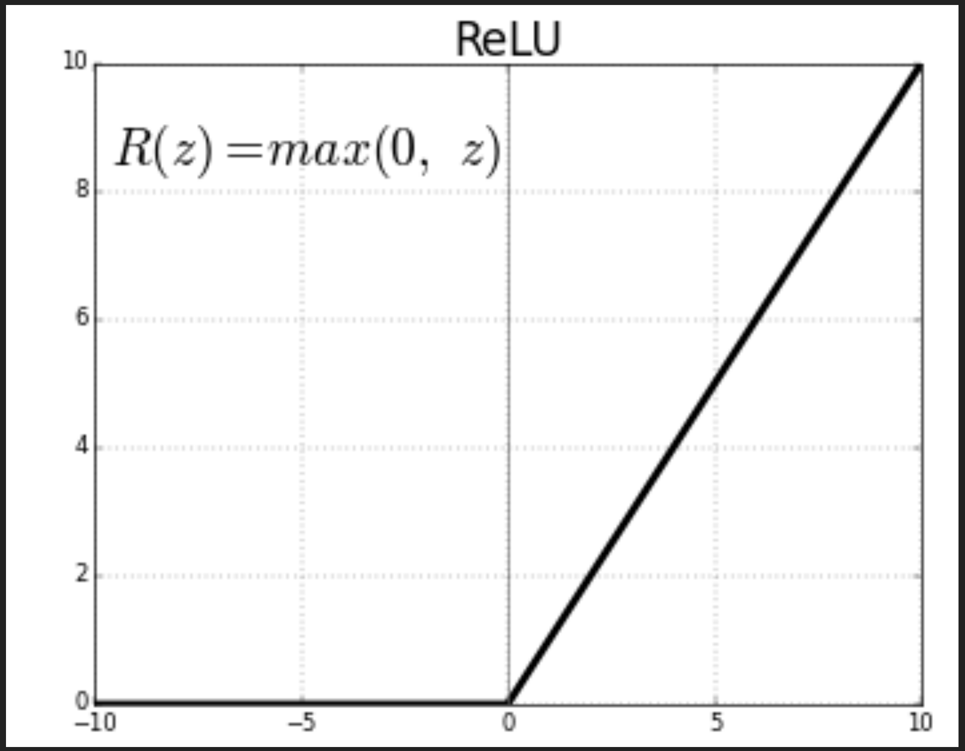
ACTIVATIONS - RECTIFIED LINEAR UNIT(RELU)

- Neurons do not get saturated. Faster Convergence. Computationally efficient.
- Not zero-centered
- Gradient is 0 for x<0
- Regularisation effect

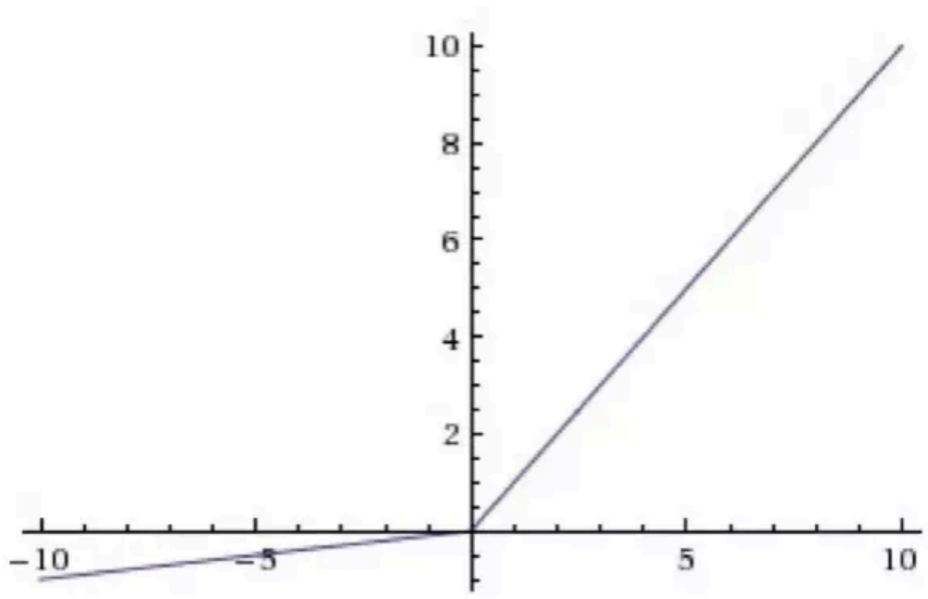


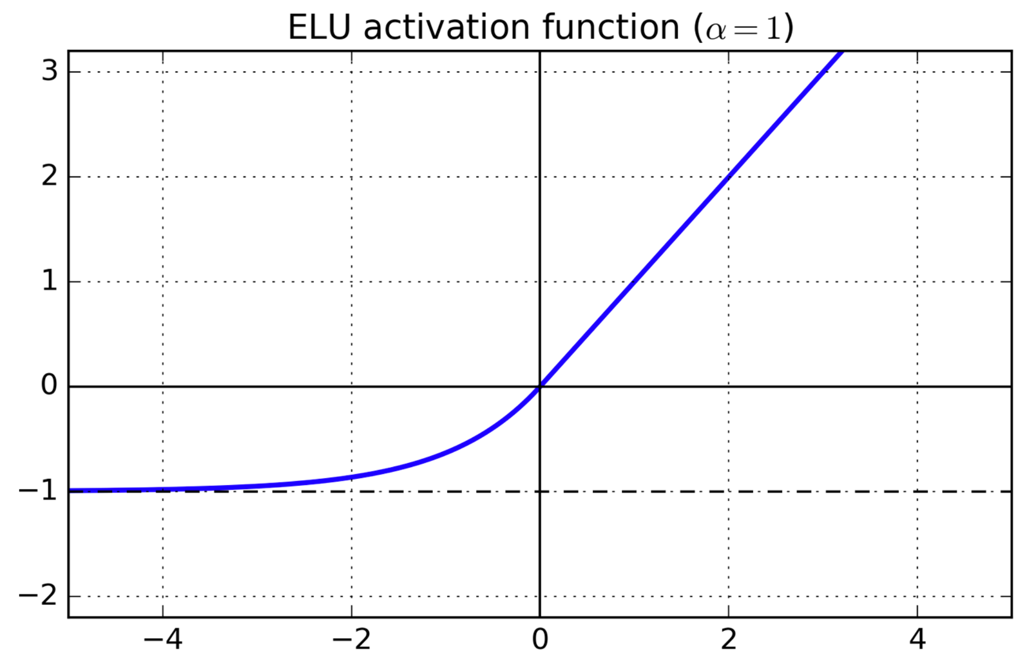
ACTIVATIONS - RECTIFIED LINEAR UNIT(RELU)

- Neurons do not get saturated. Faster Convergence. Computationally efficient.
- Not zero-centered
- Gradient is 0 for x<0
- Regularisation effect
- More biologically plausible than Sigmoid



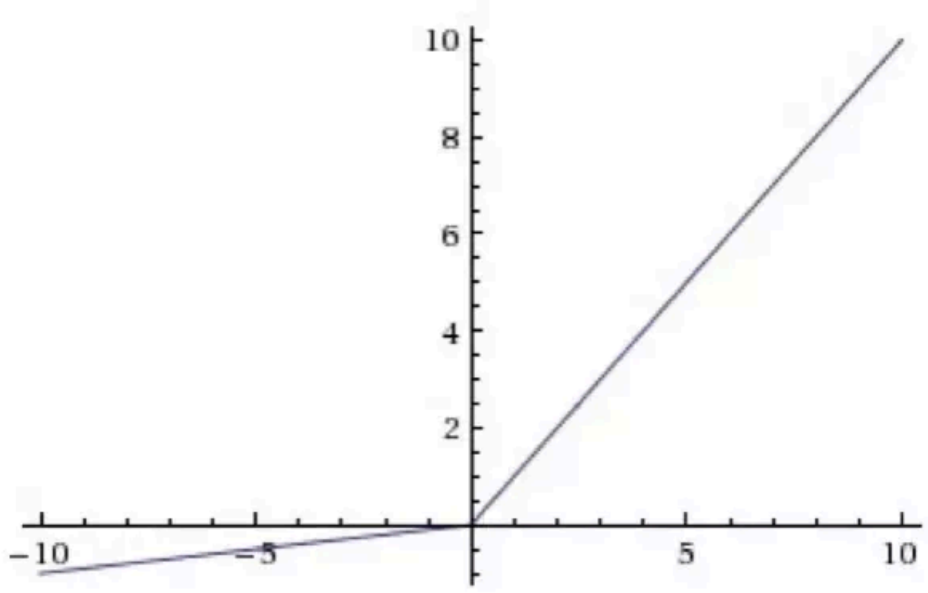
ACTIVATIONS – LEAKY RELU AND ELU

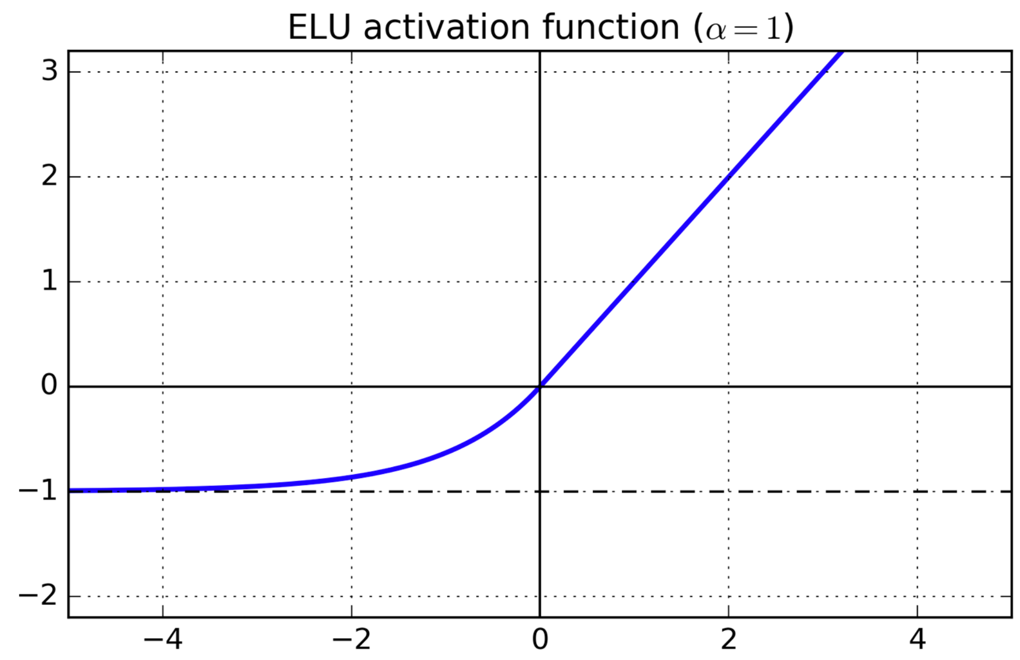




ACTIVATIONS - LEAKY RELU AND ELU

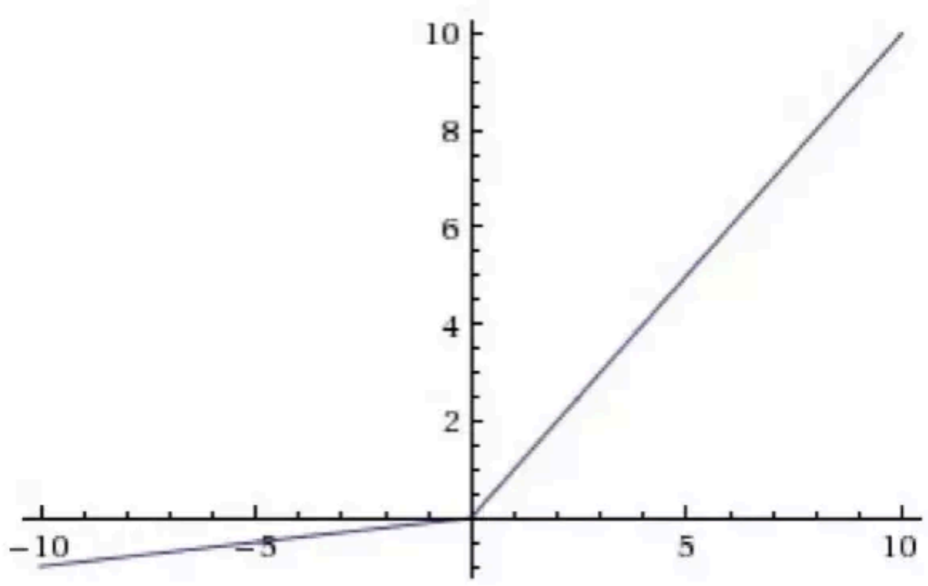
- No Saturation.
  Computationally efficient

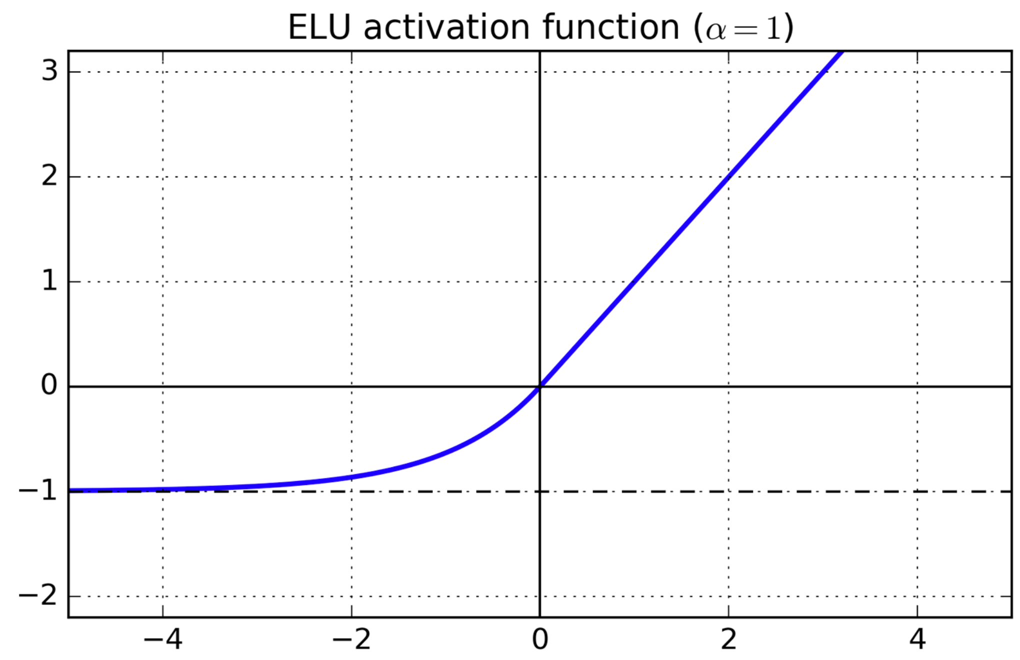




ACTIVATIONS - LEAKY RELU AND ELU

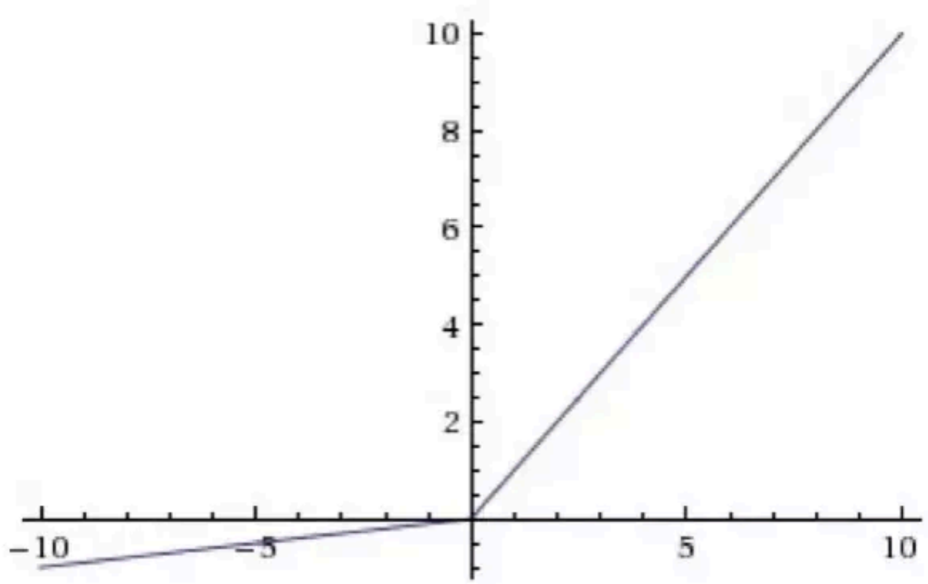
- No Saturation.
  Computationally efficient
- Gradient is not 0 for x<0

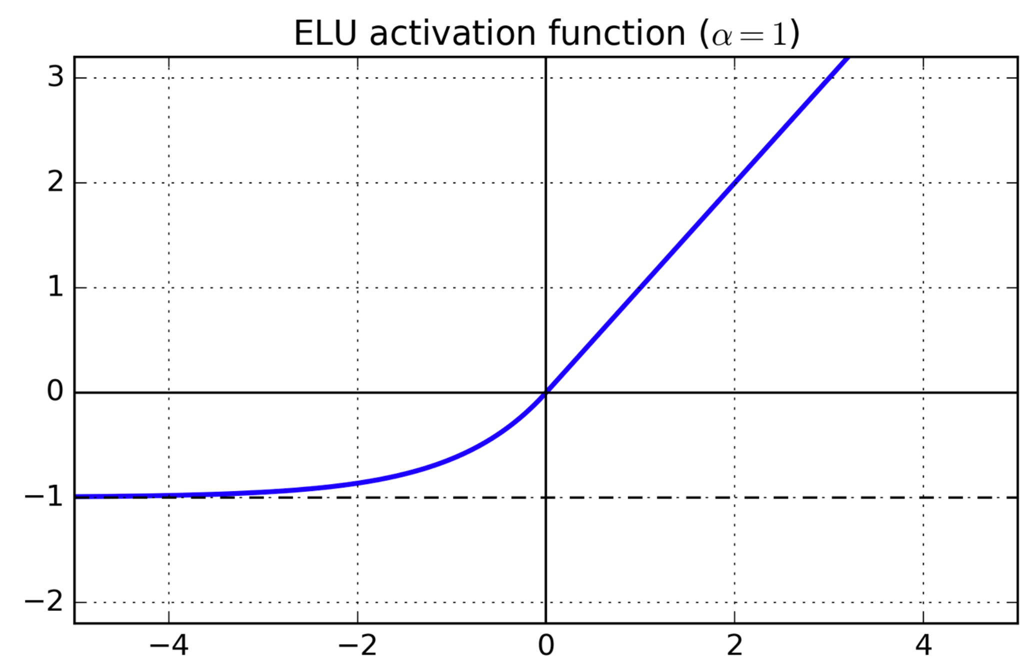




ACTIVATIONS – LEAKY RELU AND ELU

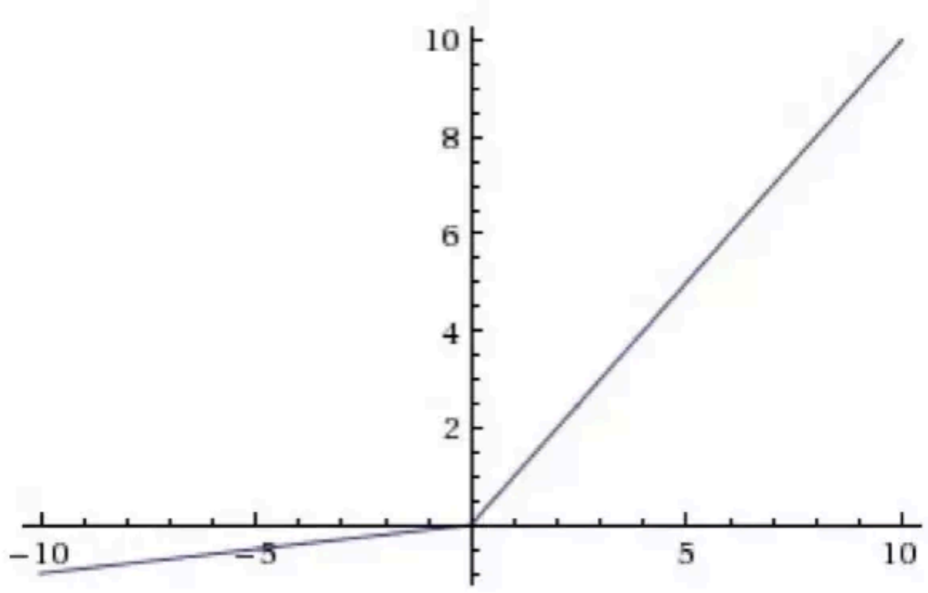
- No Saturation. Computationally efficient
- Closer to zero-entered than other activations
- Gradient is not 0 for x<0





ACTIVATIONS – LEAKY RELU AND ELU

- No Saturation. Computationally efficient
- Gradient is not 0 for x<0



- Closer to zero-entered than other activations
- Negative Saturation adds noise hence robustness