An Introduction to Deep Neural Networks

Olac Fuentes
Associate Professor
Computer Science Department
UTEP

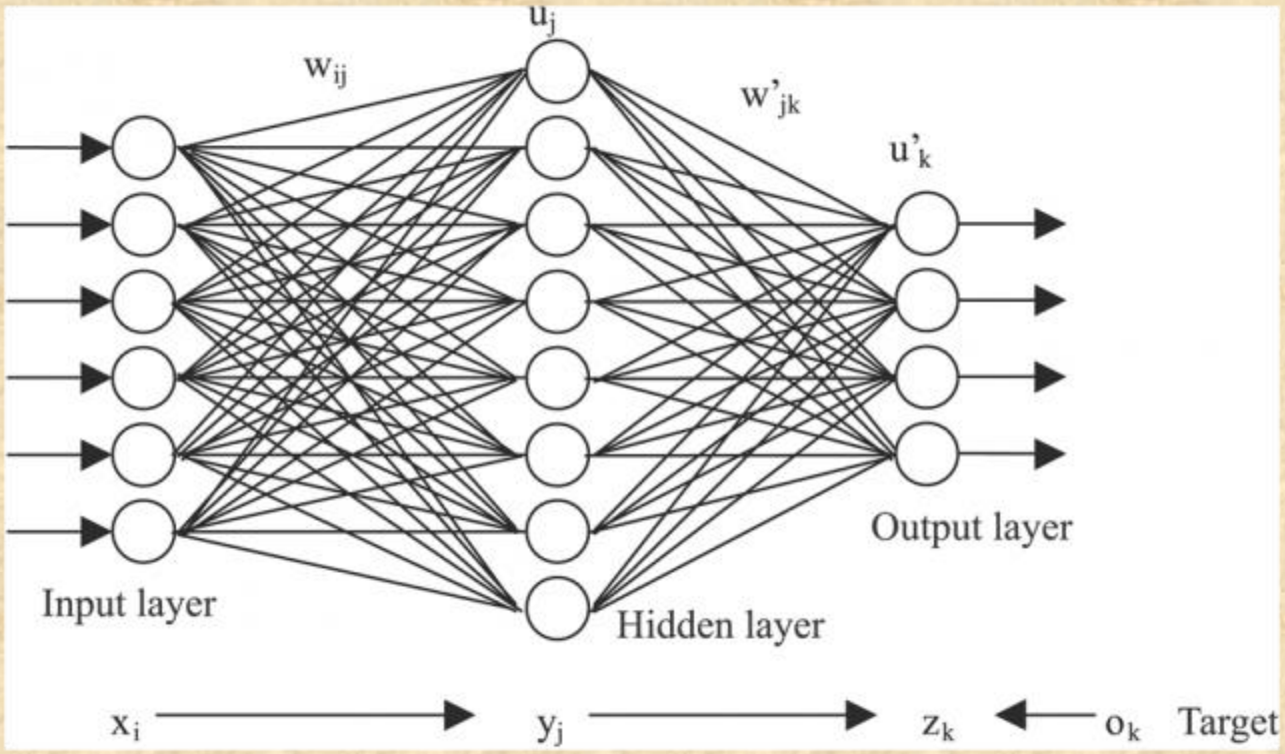
Neural Networks - History

- Perceptron – (Rosenblatt, 1957)
- “Perceptrons” – (Minsky and Papert, 1969)
- PDP (Rumelhart, Hinton & Williams, 1986)
- Convolutional neural networks (LeCun 1992)
- The deep learning revolution – 2006 to date

What is a Neural Network?

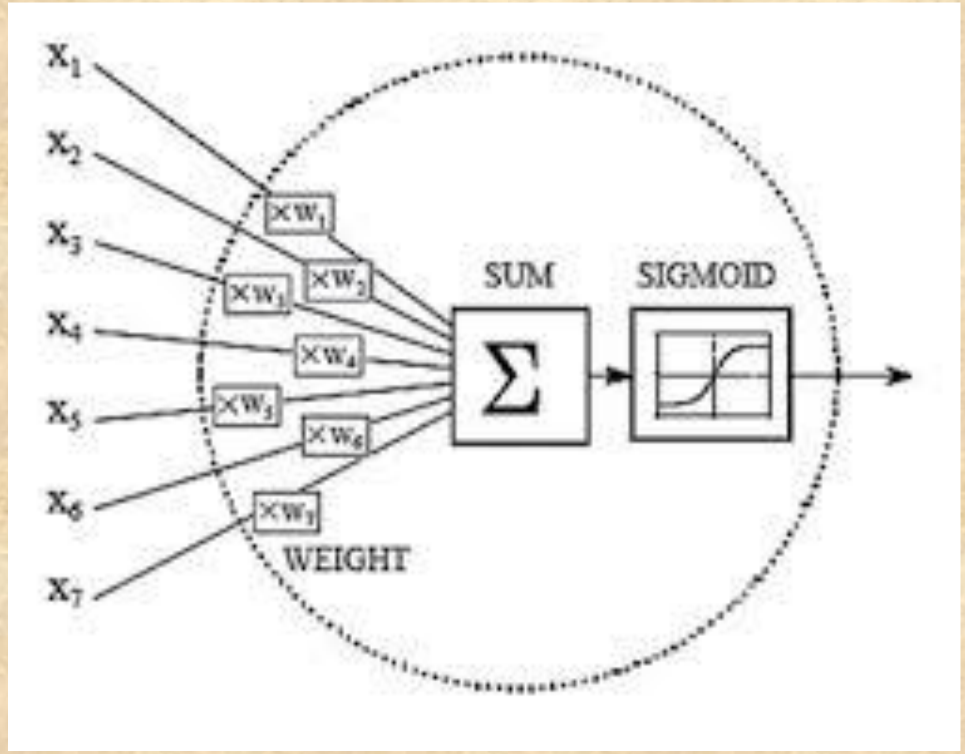
- Artificial neural networks are composed of sets of simple units densely interconnected
- Each unit (or artificial neuron) takes as input several real-valued numbers (which are possibly the outputs of another neuron) and produces a real-valued output.

What is a Neural Network?

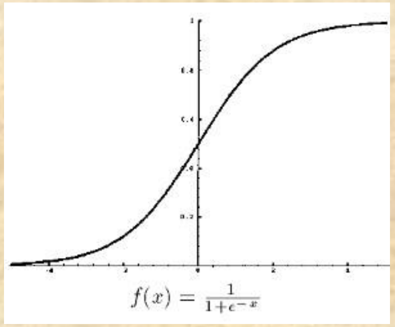


A neural network

What is a Neural Network?



An individual neuron using the sigmoid activation function



Learning in a Neural Network

- Suppose we have a set of *training instances* $x_0,..,x_n$, with target function values $f(x_1),..,f(x_n)$.

- Examples

x

$f(x)$	lion	dog	elephant	gorilla

- Optimize the weights ***W*** of the network such that

$f(x_i) = \Theta(\mathbf{W},x_i)$, for $1 \leq i \leq n$

Learning in a Neural Network

- Optimize the weights ***W*** of the network such that

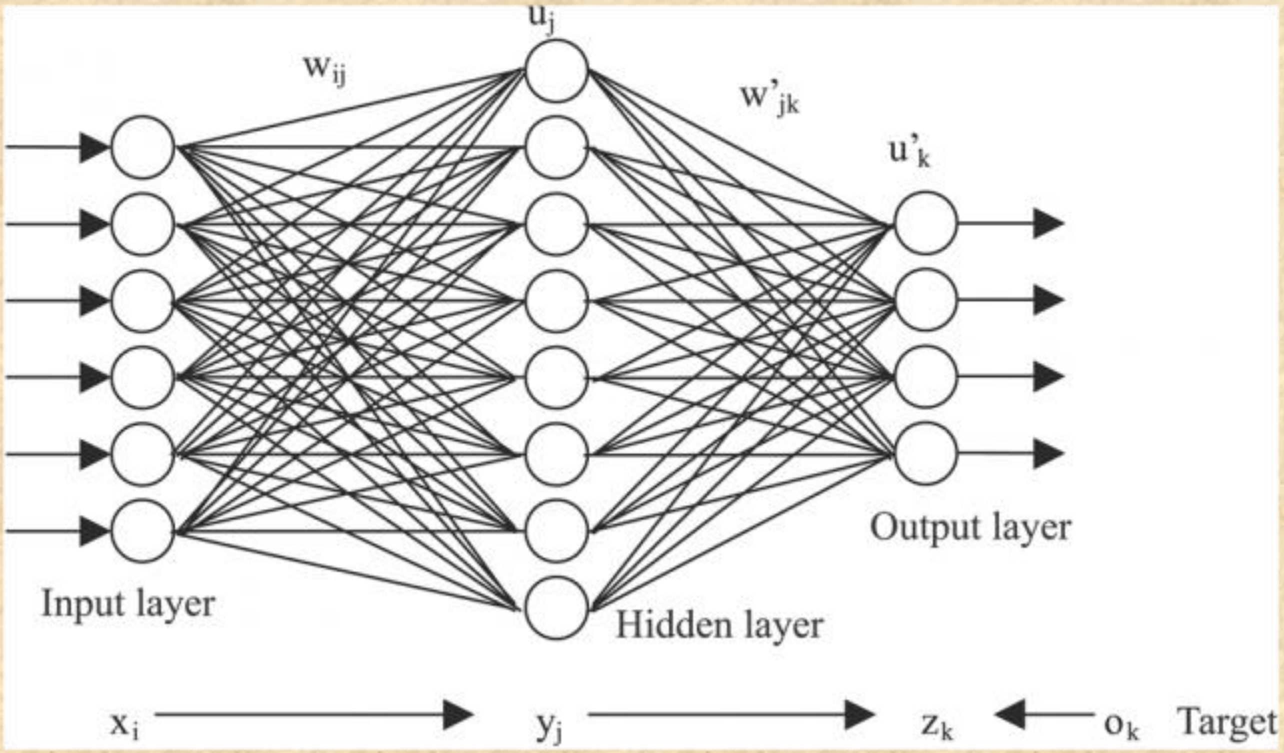
  $f(x_i) = \Theta(W,x_i)$, for $1 \leq i \leq n$
- In practice, we minimize:

  $J(w) = \Sigma (f(x_i) - \Theta(W,x_i))^2$

  using a suitable optimization algorithm.

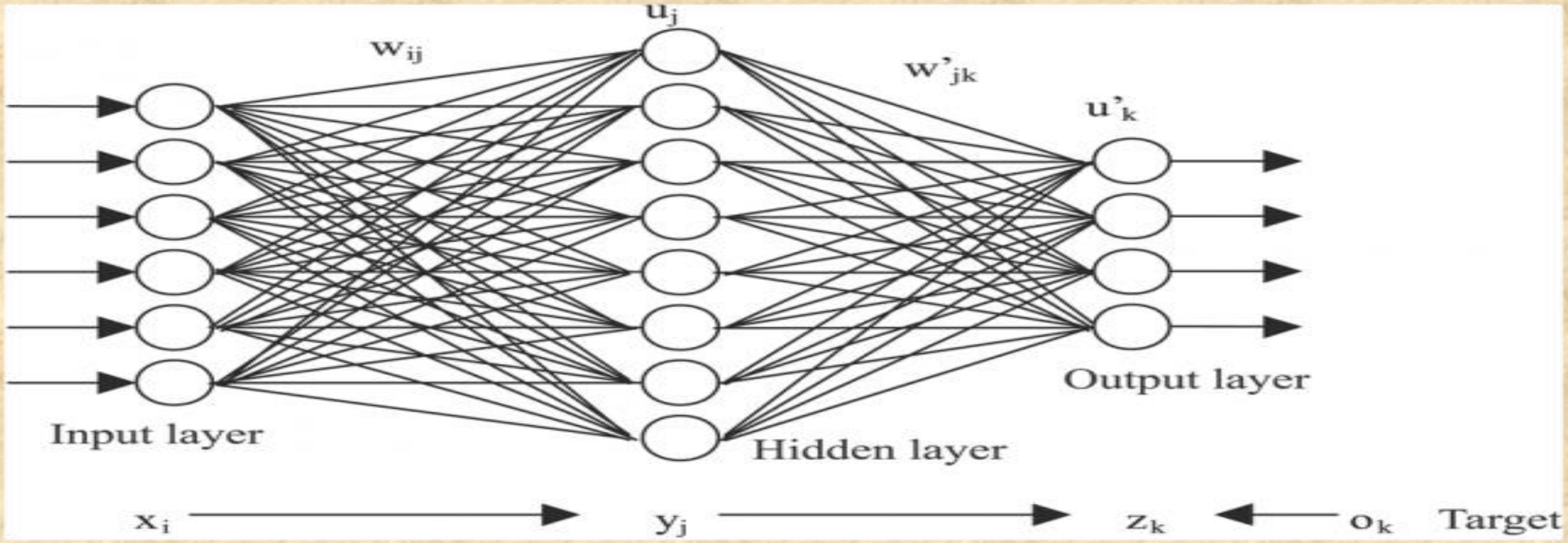
  Find $\Delta J(w)$ using calculus, change weights iteratively in the direction of decreasing $J(w)$
- First algorithm to do this efficiently: Backpropagation (Rumelhart, Hinton & Williams, 1986).

A Simple 3-layer feed-forward neural network



Let x be the input (a training example), let y* be the output (the target function predicition), let h be the hidden layer activation

A Simple 3-layer feed-forward neural network



Each layer implements a linear function

$h = x\,W_0$ (x is [1 x m], W_0 is [m x n], h is [1 x n])

$y^* = h\,W_1$ (h is [1 x n], W_1 is [n x o], y* is [1 x o])

A Simple 3-layer feed-forward neural network

Each layer implements a linear function

$h = x W_0$ (x is [1 x m], W_0 is [m x n], h is [1 x n])

$y^* = h W_1$ (h is [1 x n], W_1 is [n x o], y* is [1 x o])

A Simple 3-layer feed-forward neural network

Each layer implements a linear function

$h = x W_0$ (x is [1 x m], W_0 is [m x n], h is [1 x n])

$y^* = h W_1$ (h is [1 x n], W_1 is [n x o], y^* is [1 x o])

Let (x,y) be a training example

The quadratic error is is given by:

$e = 1/2(y^* - y)^2$

A Simple 3-layer feed-forward neural network

Each layer implements a linear function

$h = x W_0$ (x is [1 x m], W_0 is [m x n], h is [1 x n])

$y^* = h W_1$ (h is [1 x n], W_1 is [n x o], y* is [1 x o])

Let (x,y) be a training example

The quadratic error is is given by:

$e = 1/2(y^* - y)^2$

We can compute

$\delta e/\delta W_0$ and $\delta e/\delta W_1$ and find W_0 and W_1 using gradient descent and gradient descent

$\delta e/\delta W_1 = (\delta e/\delta y^*) (\delta y^*/\delta W_1)$

$\delta e/\delta W_0 = (\delta e/\delta y^*) (\delta y^*/\delta h) (\delta h/\delta W_0)$

A Simple 3-layer feed-forward neural network

Each layer implements a linear function

$h = x W_0$ (x is [1 x m], W_0 is [m x n], h is [1 x n])

$y = h W_1$ (h is [1 x n], W_1 is [n x o], y^* is [1 x o])

Then:

$y^* = x W_0 W_1$

We can define a matrix $W = W_0 W_1$

$y^* = x W$

thus a linear 3-layer neural network is equivalent to linear regression

Adding a nonlinearity

$h = f_0(x W_0)$ (x is [1 x m], W_0 is [m x n], h is [1 x n],and f_0 is a nonlinear function)

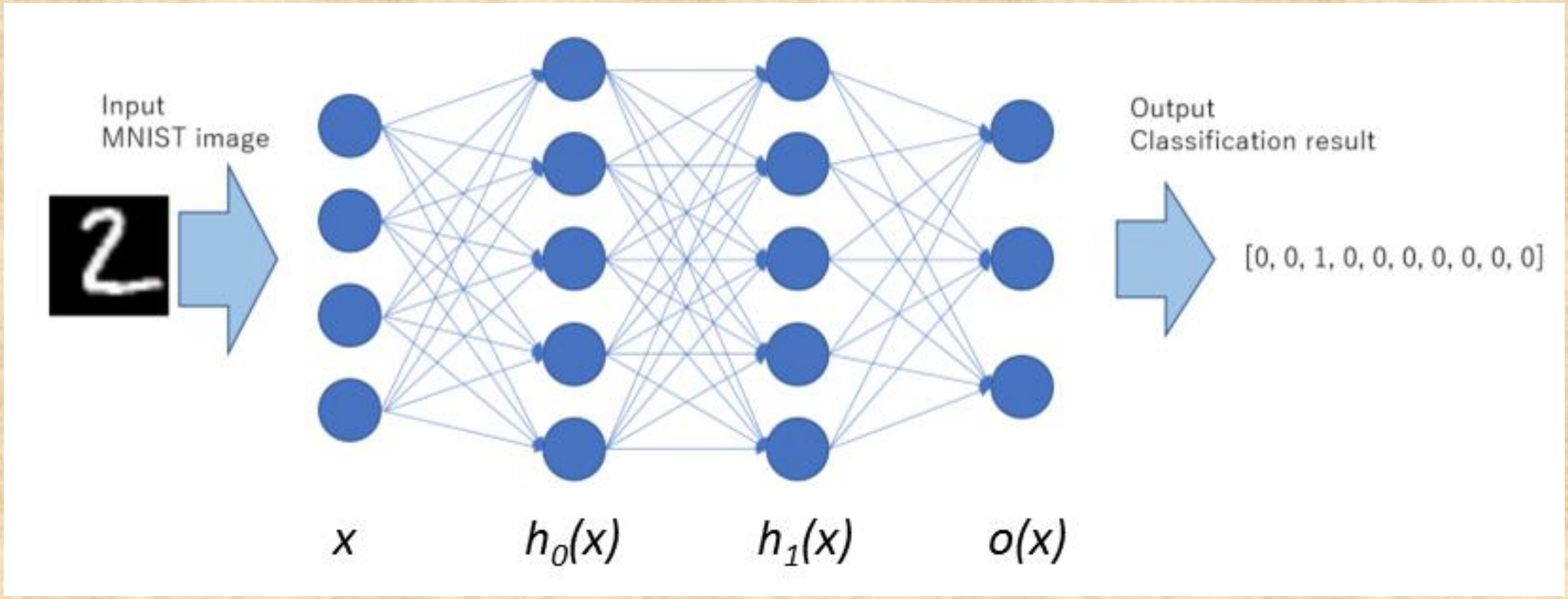
$y = f_1(h W_1)$ (h is [1 x n], W_1 is [n x o], y^* is [1 x o], and f_1 is a nonlinear function)

Then:

$y^* = f_1(f_0(x W_0)W_1)$

For several types on non-linear **activation** functions, this ‘stacking’ of non-linearities allows the network to learn very complex functions

Learning in neural networks



How do neural networks learn?

Apply gradient descent:

$$w_{i,j,k} = w_{i,j,k} - \lambda \frac{\partial J}{\partial w_{i,j,k}}$$

where J is the error or cost function

Learning in neural networks

Thus for every parameter $w_{i,j,k}$, we need to answer:

How does the error change in response to (small) changes in $w_{i,j,k}$?

How do we do that?

Basic calculus

We need to find derivatives of error, activation functions and matrix products and combine them using the chain rule.

Learning in neural networks

Thus for every parameter $w_{i,j,k}$, we need to answer:

How does the error change in response to (small) changes in $w_{i,j,k}$?

How do we do that?

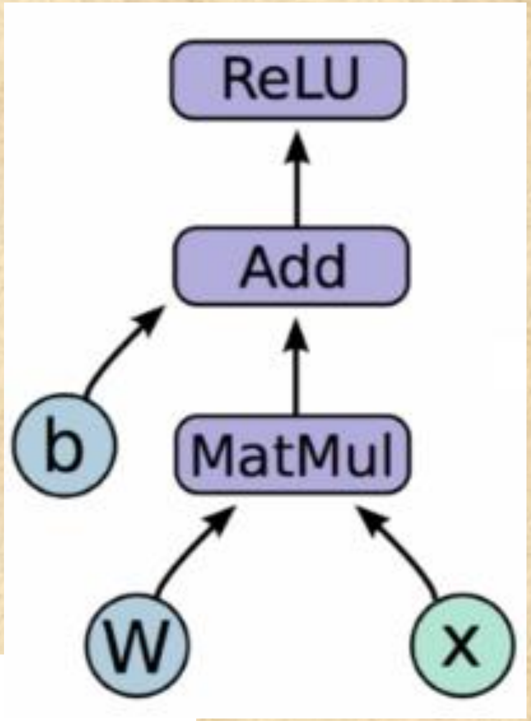
Basic calculus

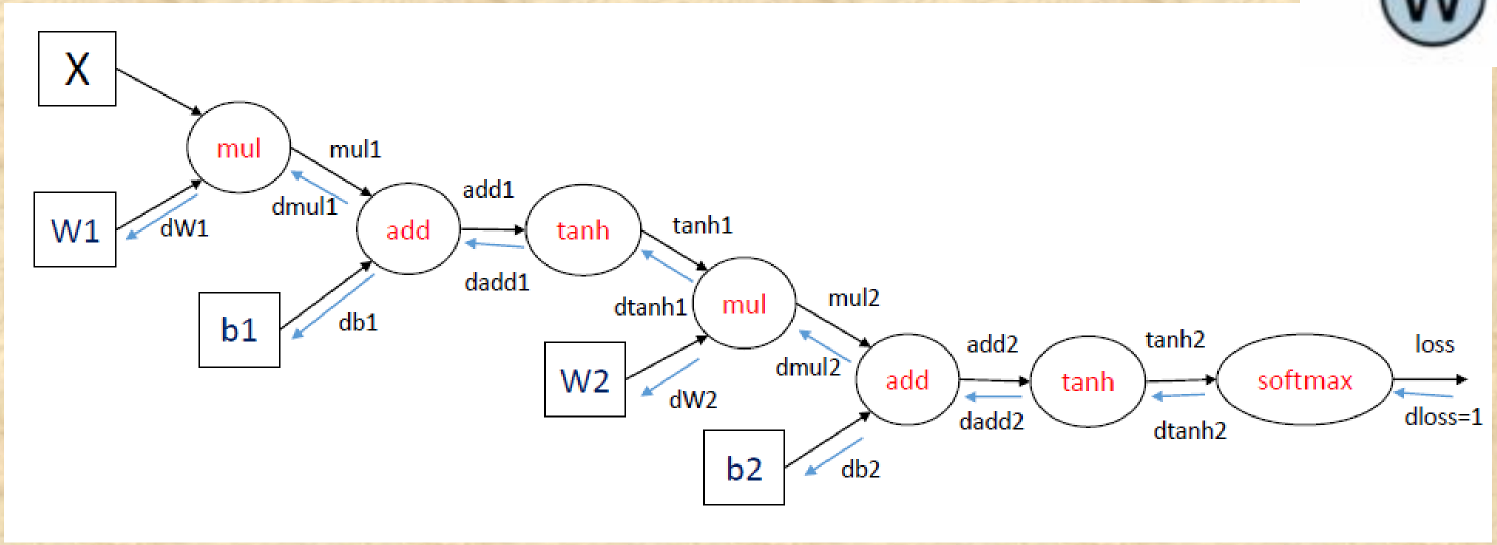
We need to find derivatives of error, activation functions, and matrix products and combine them using the chain rule.

Learning in neural networks

It's easy to find the derivative of each variable with respect to each of its inputs.

Applying the chain rule we can find the derivative of the error with respect to the network parameters w and b.





Activation Functions

Normally the output of a neuron is given by:

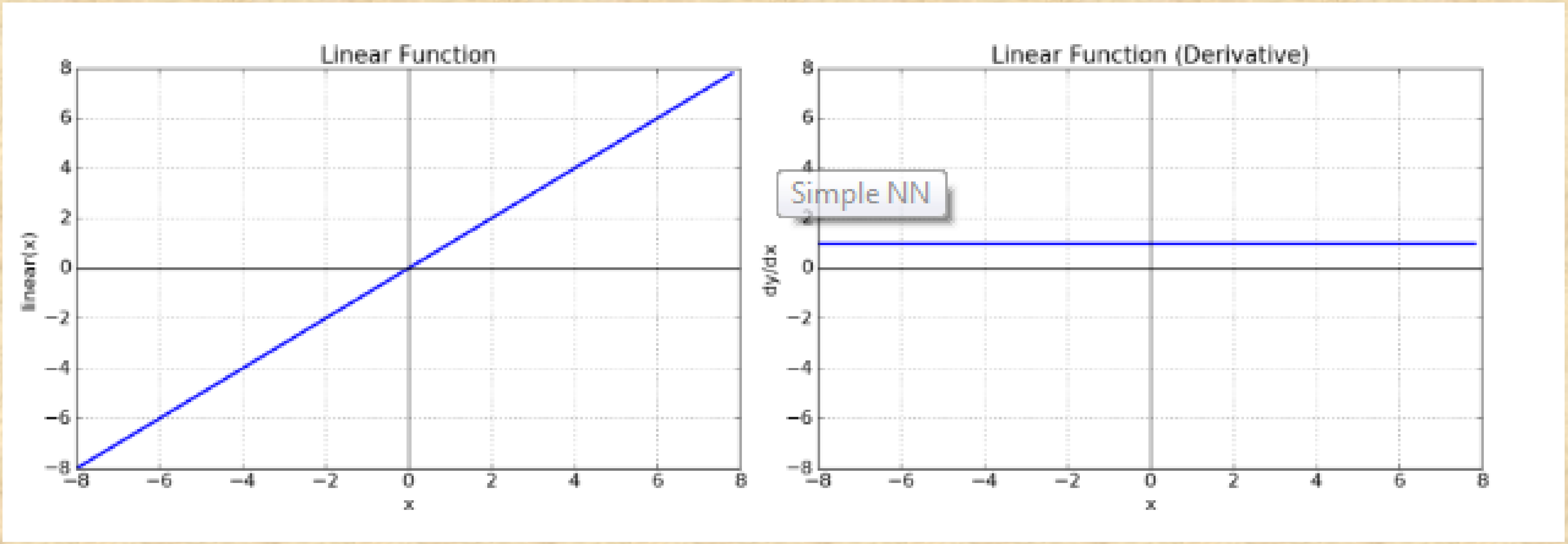
$f(xW\ )$

where x is the input vector (or matrix), W is the weight matrix, b is the bias vector, and f is the **activation function**.

In order to be able to learn, the activation function must be differentiable (i.e. it must have a derivative).

Activation Functions

Linear function:



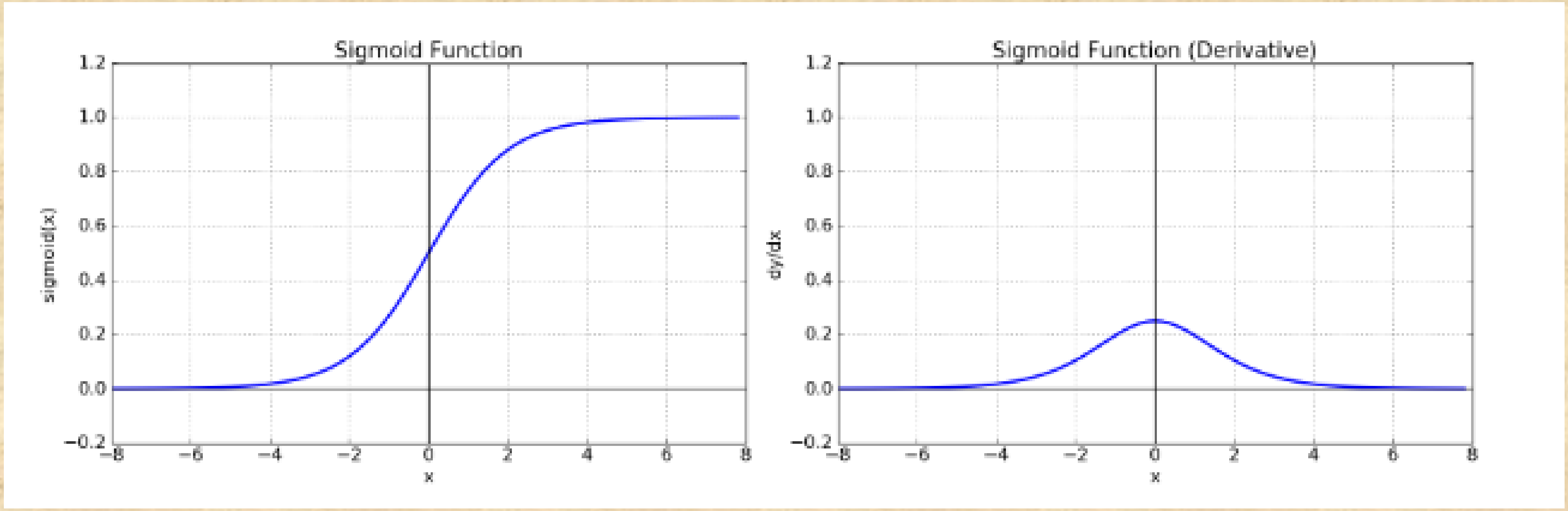
$z = xW$

$f(z) = z$

$f'(z) = 1$

Activation Functions

Sigmoid function:



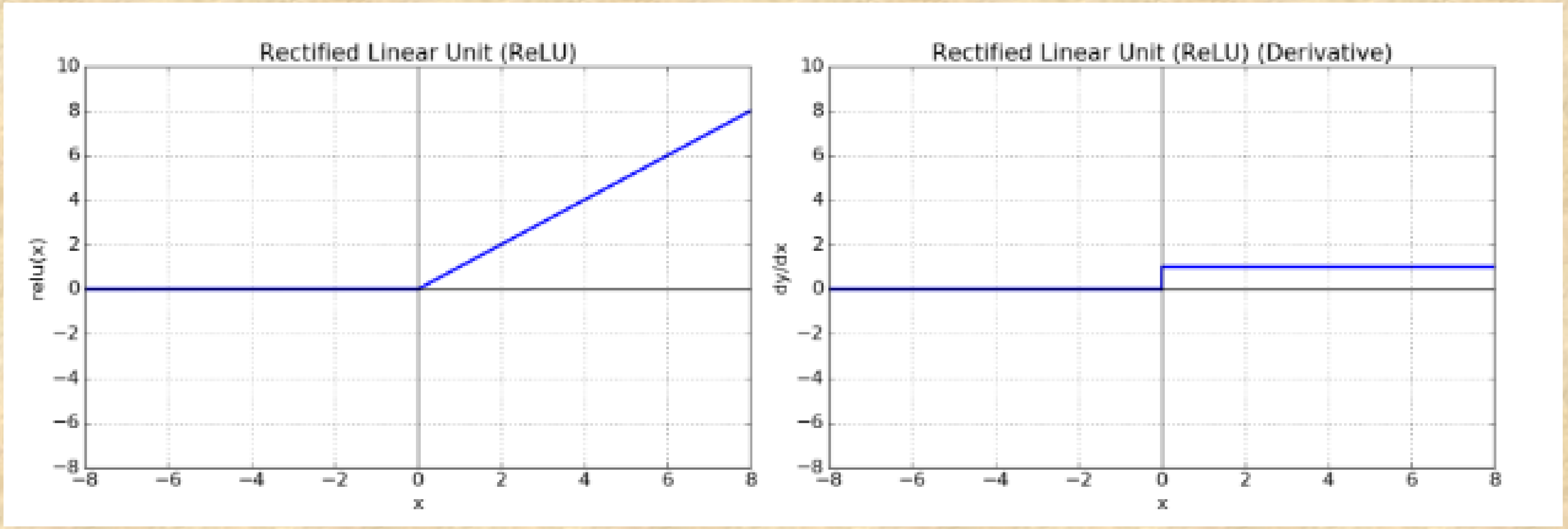
$z = xW$

$f(z) = 1/(1+e^{-z})$

$f'(z) = e^{-z}/(1+e^{-z})^2 = f(z)(1 - f(z))$

Activation Functions

Rectified linear unit (RELU):



$z = xW$

$f(z) = z$ if $z>0$

$f(z) = 0$ otherwise

$f'(z) = 1$ if $f(z)>0$

$f'(z) = 0$ otherwise

$f'(z) = sign(z)$

Activation Functions

Softmax:

More complicated, as it depends on activations of other units in the same layer

Useful as a final layer for classification

$f_i(z) = e^{zi} / \Sigma\, e^{zj}$

$f'_i(z_j) = f_i(z_j)(1 - f_i(z_j)) \quad if\ i==j$

$f'_i(z_j) = -f_i(z_i)\, f_i(z_j) \quad if\ i\ !=j$