Deep Learning
-Activation, Regularization

Artificial Intelligence

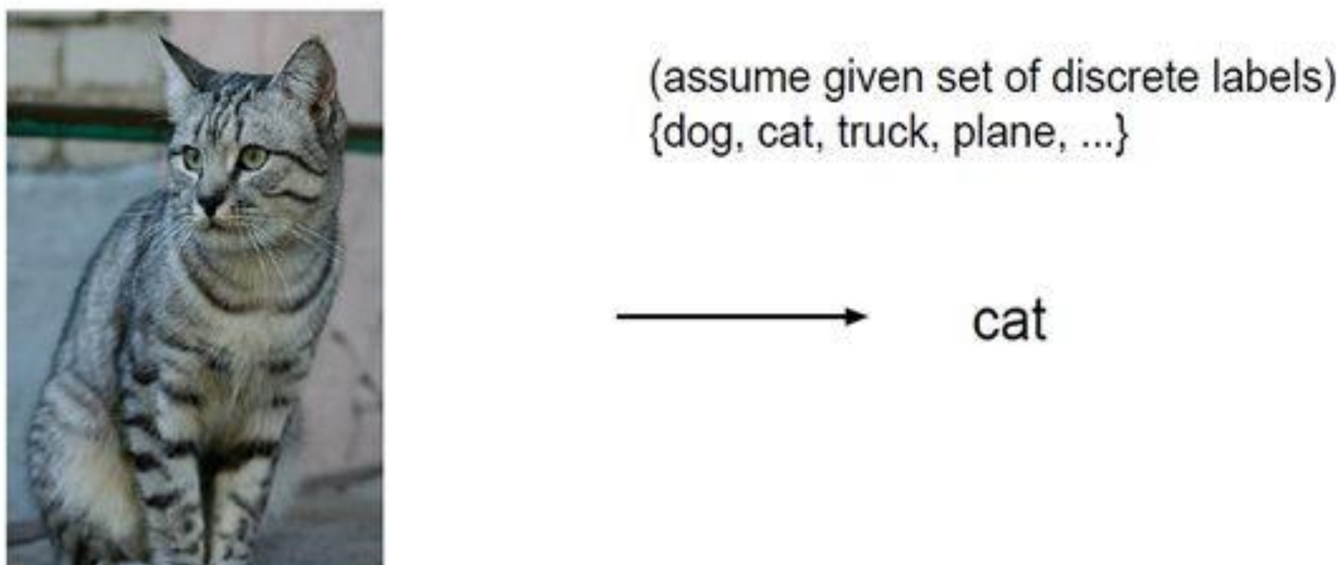
School of Computer Science
The University of Adelaide

Visual Learning

- Learning the mapping function.

$$f(\mathbf{X}) \rightarrow \mathbf{Y}$$

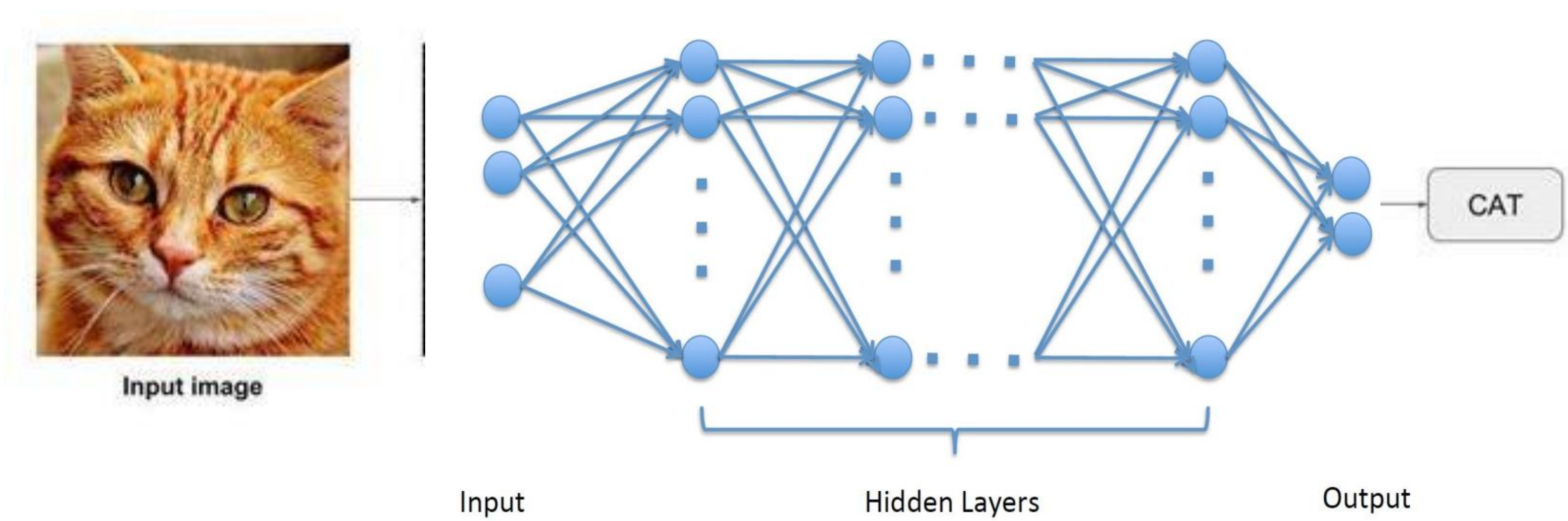
- For example, Image or Text to category, video to action



Deep Neural Network

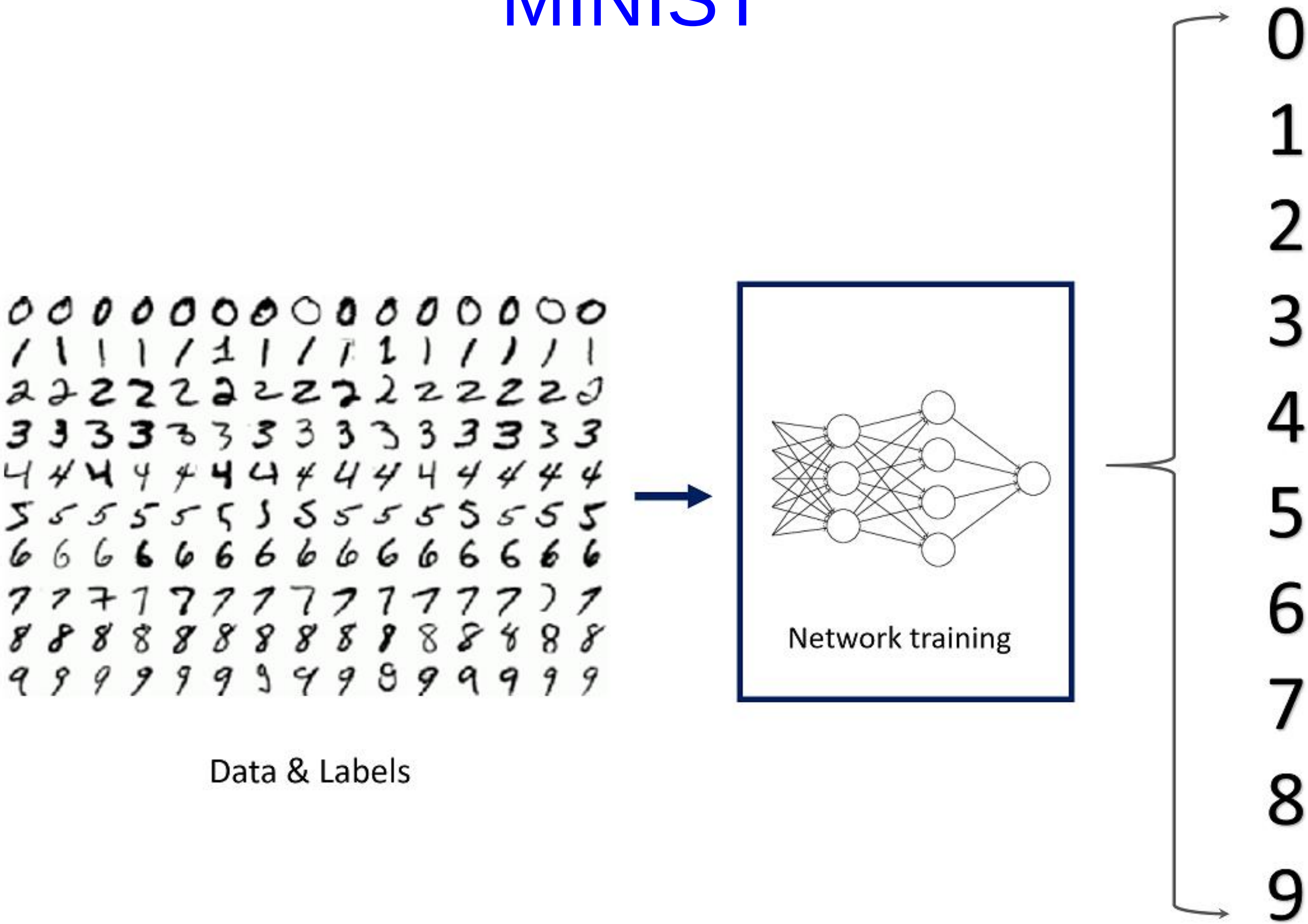
- Decompose the problem into multiple parts:

$$\mathbf{Z}_1 = f_1(\mathbf{X})$$
$$\mathbf{Z}_2 = f_2(\mathbf{Z}_1)$$
$$\cdots$$
$$\mathbf{Z}_t = f_t(\mathbf{Z}_{t-1})$$
$$\mathbf{Y} = f_y(\mathbf{Z}_t)$$



- Both features and classifier are learned
- Researchers have hypothesized that the number of layers in an MLP correlates well with high-level information.

MINIST



MLP Based Deep Learning

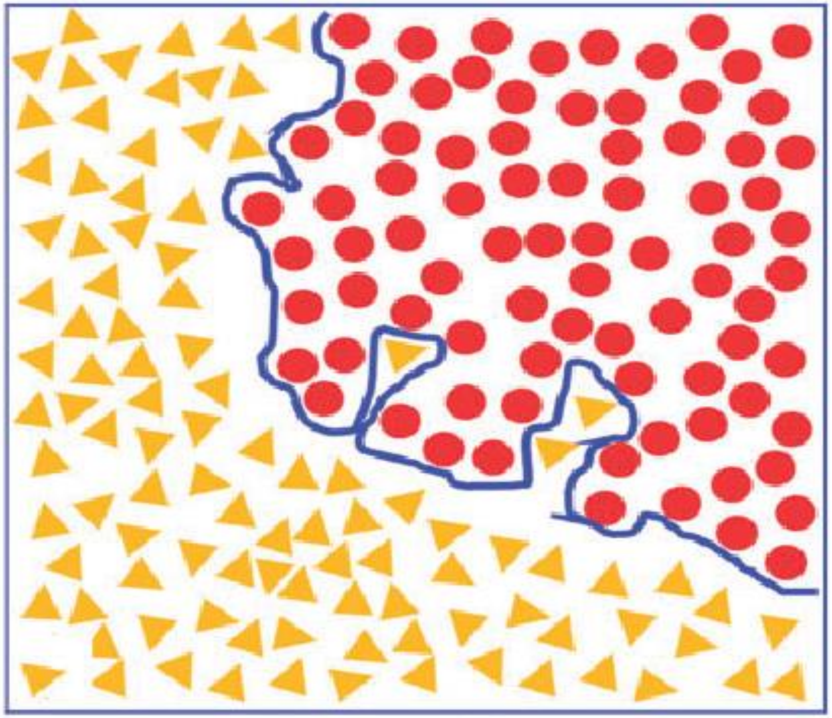
- Unmanageable number of parameters
  - For example, let's take MNIST problem with 28*28=784 input images
  - 1 hidden layer with 1000 nodes and an output layer with 10 nodes: #weights = 784*1000+1000*10=79400 weights
  - 2 hidden layers with 1000 nodes and an output layer with 10 nodes: #weights = 784*1000+1000*1000+100*10=1794000weights

- Gradient descent didn't work beyond a couple of hidden layers
  - Magnitude kept reducing as the gradient flowed back to the input layer (vanishing gradient problem [Hochreiter91])
  - Convergence issues

Learning Neural Networks

- Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}}$$



Too many Parameters

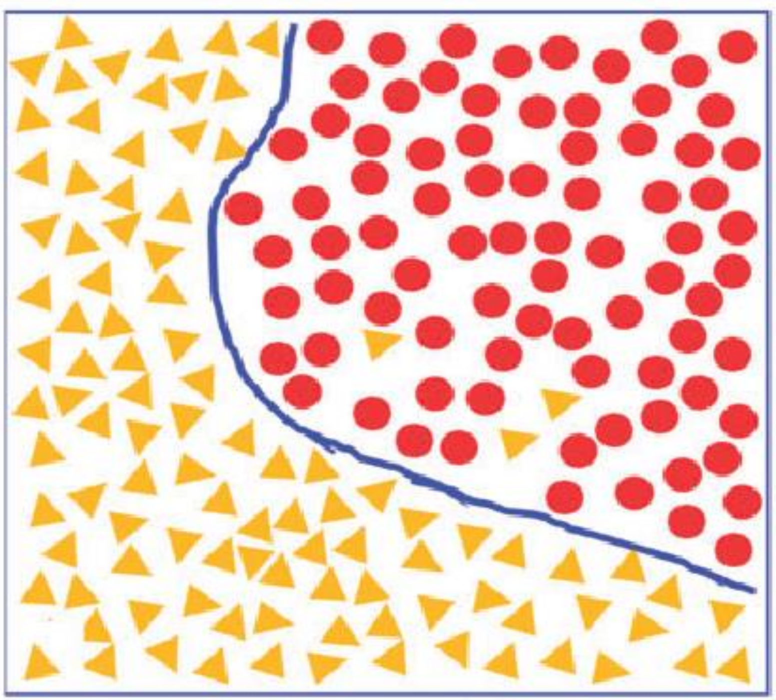
=

Overfitting!!!

Learning of Neural Network

- Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}} + \underbrace{\lambda R(W)}_{\text{Choose the simplest model}}$$



Overfitting

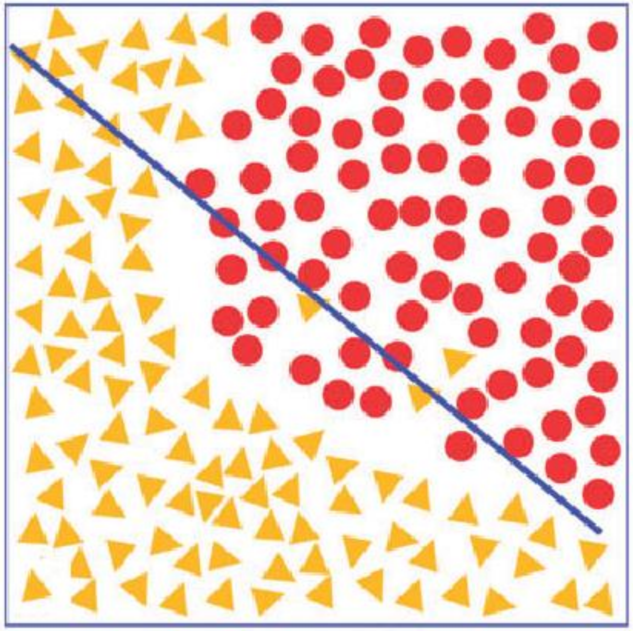
Remember Occam's Razor !!!

Regularizing decision boundaries in this manner help avoiding overfitting.

Regularization

- Optimizing a loss function to learn parameters

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Fitting to data}} + \underbrace{\lambda R(W)}_{\text{Choose the simplest model}}$$



underfitting

Learning of Neural Network

- Commonly-used regularizers
  - L2-regularization (Lasso): $R_{L_2}(w) \triangleq ||W||_2^2$
  - L1-regularization (Ridge): $R_{L_1}(w) \triangleq \sum_{k=1}^{Q} ||W||_1$
  - Drop-out: it randomly selects some nodes and removes them along with all of their incoming and outgoing connections as shown below.
  - Early stopping: keep one part of the training set as the validation set. When we see that the performance on the validation set is getting worse, we immediately stop the training on the model. This is known as early stopping.

L2-Norm

- L2-Norm: L2 regularization is also called *Weight decay*.

$$\|\mathrm{W}\|_2 \equiv \sqrt{\sum_{i=1}^{m} |w_i|^2}$$

$$L = L' + \frac{\lambda}{2n} \sum_w w^2$$

$$\frac{\partial L}{\partial w} = \frac{\partial L'}{\partial w} + \frac{\lambda}{n} w$$

$$w \to w - \eta \frac{\partial L'}{\partial w} - \frac{\eta \lambda}{n} w$$

$$= \underline{\left(1 - \frac{\eta \lambda}{n}\right)} w - \eta \frac{\partial L'}{\partial w}$$

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

L1-Norm

- L1-Norm:

$$\|\mathrm{W}\|_1 \equiv \sum_{i=1}^{m} |W_i|$$

$$L = L' + \frac{\lambda}{n}\sum_{w}|w|$$

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$$

$$\frac{\partial L}{\partial w} = \frac{\partial L'}{\partial w} + \frac{\lambda}{n} sgn(w)$$

$$w \rightarrow w - \frac{\eta\lambda}{n} sgn(w) - \eta\frac{\partial L'}{\partial w}$$

L1 vs L2

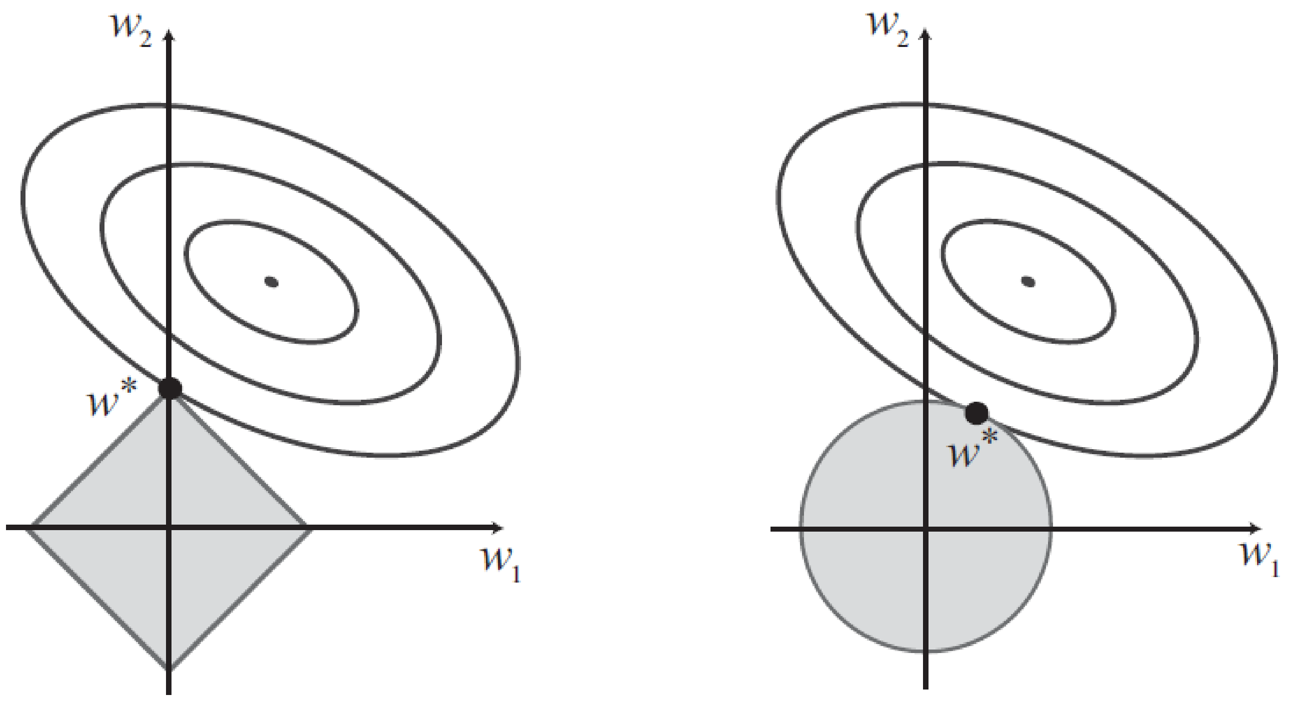


Figure 18.14 Why L_1 regularization tends to produce a sparse model. (a) With L_1 regularization (box), the minimal achievable loss (concentric contours) often occurs on an axis, meaning a weight of zero. (b) With L_2 regularization (circle), the minimal loss is likely to occur anywhere on the circle, giving no preference to zero weights.

MLP Based Deep Learning

- Unmanageable number of parameters
  - For example, let's take MNIST problem with 28*28=784 input images
  - 1 hidden layer with 1000 nodes and an output layer with 10 nodes: #weights = 784*1000+1000*10=79400 weights
  - 2 hidden layers with 1000 nodes and an output layer with 10 nodes: #weights = 784*1000+1000*1000+100*10=1794000weights

- Gradient descent didn't work beyond a couple of hidden layers
  - Magnitude kept reducing as the gradient flowed back to the input layer (vanishing gradient problem [Hochreiter91])
  - Convergence issues

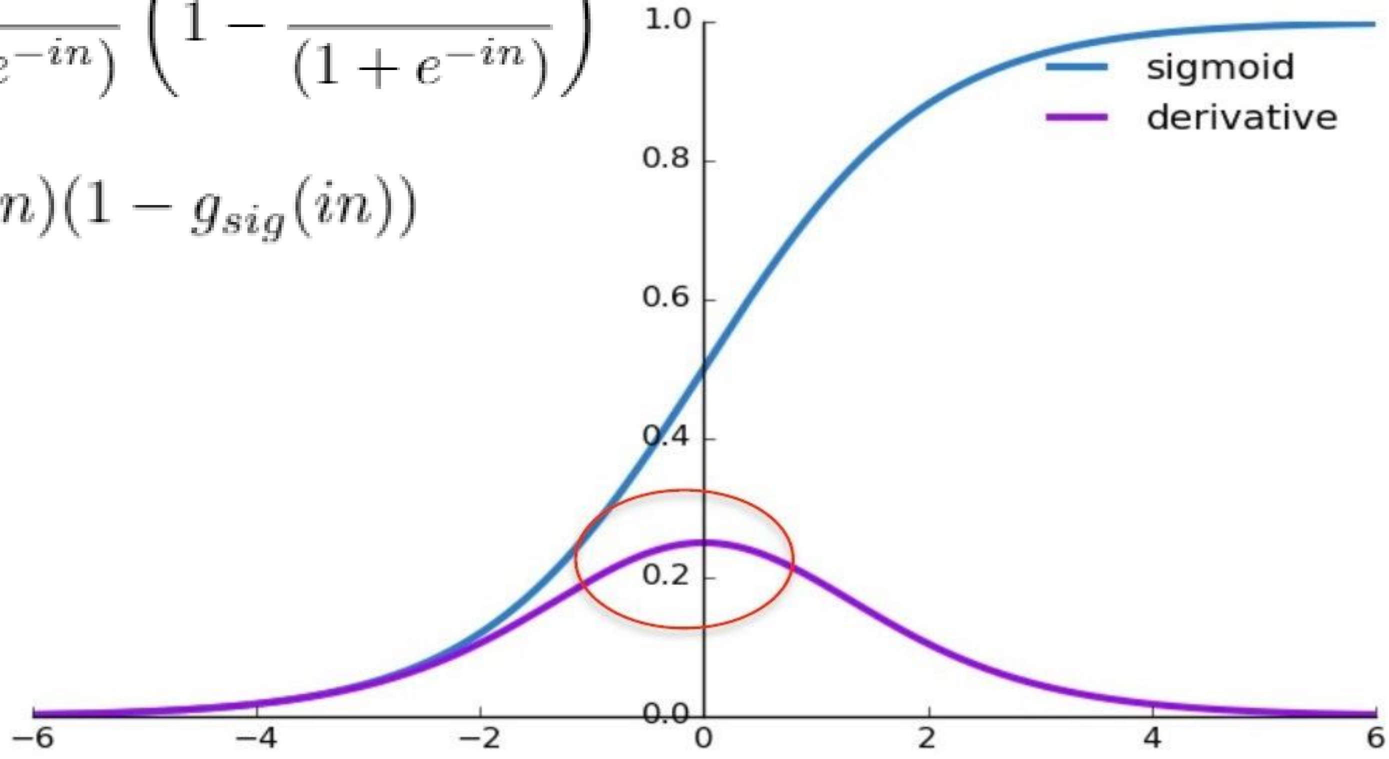
Activation Function: Sigmoid

- Non-linearity based on sigmoid.

$$g_{sig}(in) = \frac{1}{1 + e^{-in}}$$

$$g'_{sig}(in) = \frac{1}{(1 + e^{-in})}\left(1 - \frac{1}{(1 + e^{-in})}\right)$$

$$= g_{sig}(in)(1 - g_{sig}(in))$$



Activation Function: Sigmoid

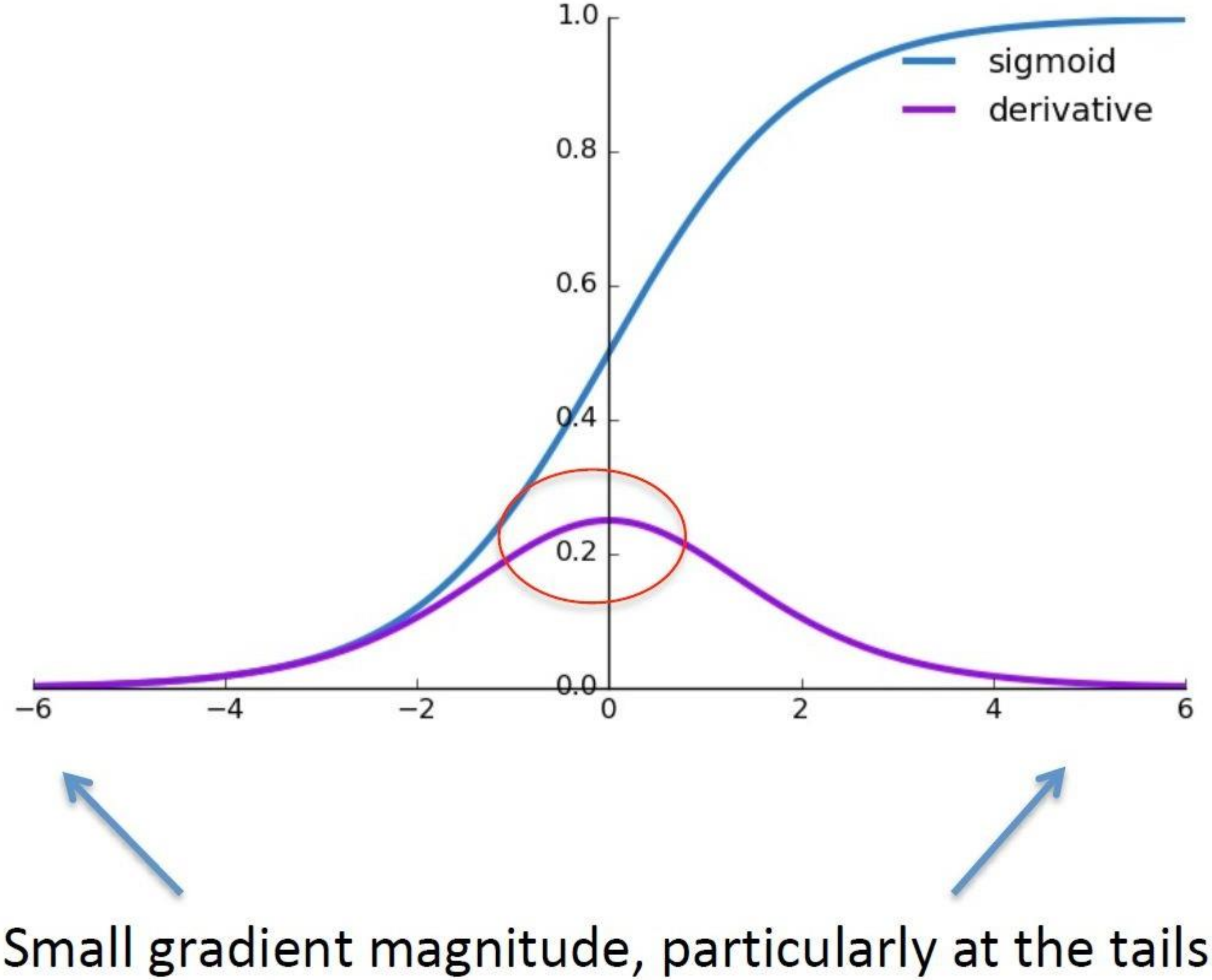
- **Advantages**
  - **Smooth gradient**, preventing “jumps” in output values.
  - **Output values bound** between 0 and 1, normalizing the output of each neuron.
  - **Clear predictions**—For x above 2 or below -2, tends to bring the y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.
- **Disadvantages**
  - **Vanishing gradient**—for very high or very low values of x, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
  - **Computationally expensive**

Sigmoid: Vanishing Gradient Problem

- After several multiplications over the layers, the gradient magnitude will become insignificant.

$$\frac{\partial E}{\partial w_{k,j}} = \Delta_j * a_k$$

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$



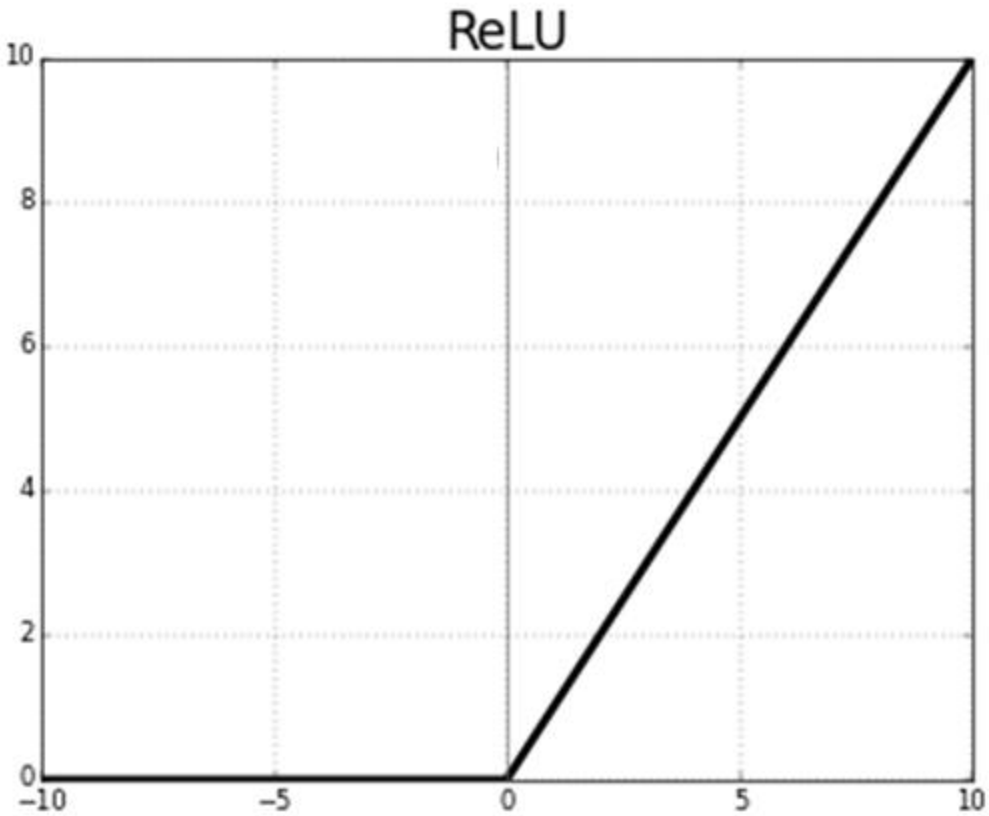
Rectified Linear Units (RELU)

Nair & Hinton (2010)

- Maximum gradient magnitude is 1
- Still non-linear
- Gradient shape?

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

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$



Rectified Linear Units (RELU)

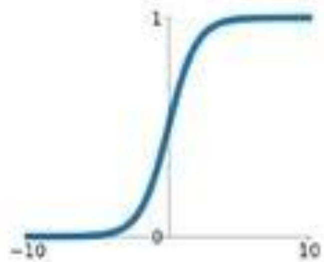
- **Advantages**
  - Computationally efficient—allows the network to converge very quickly
  - Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation

- **Disadvantages**
  - The Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

Some Prevalent Activation Functions

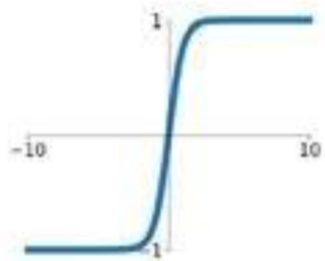
Sigmoid

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



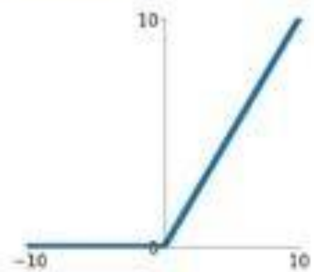
tanh

$\tanh(x)$



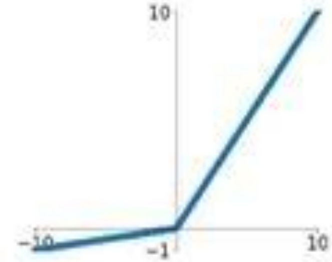
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$



Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$