# CS 4248
# Natural Language Processing

**Professor NG Hwee Tou**

**Department of Computer Science**
**School of Computing**
**National University of Singapore**
**nght@comp.nus.edu.sg**

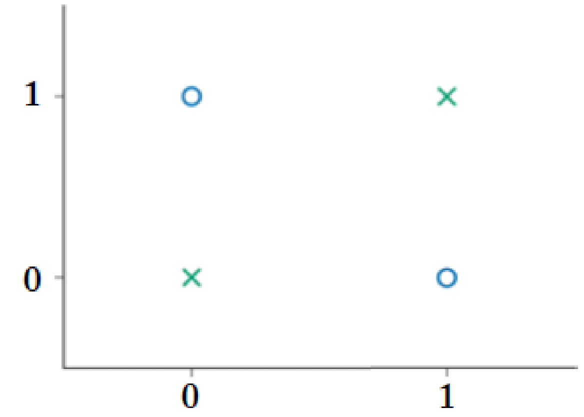
# Materials

- NNM4NLP Chapter 3, 4

# Limitations of Linear Models

- XOR problem
- No parameters $\boldsymbol{w} \in \mathbb{R}^2, b \in \mathbb{R}$ such that

$$(0,0) \cdot \boldsymbol{w} + b < 0$$
$$(1,1) \cdot \boldsymbol{w} + b < 0$$
$$(0,1) \cdot \boldsymbol{w} + b \geq 0$$
$$(1,0) \cdot \boldsymbol{w} + b \geq 0$$

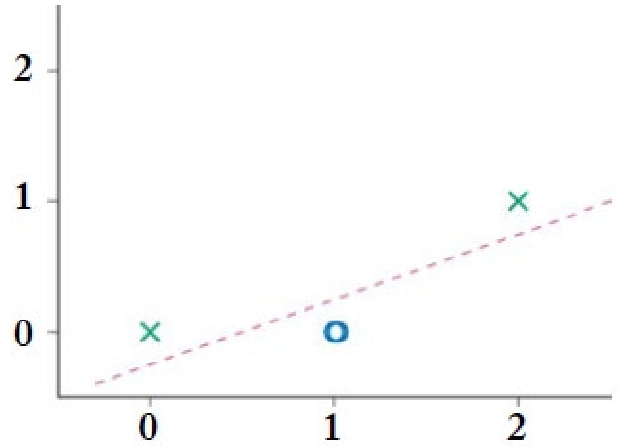



- No straight line can separate the two classes

# Limitations of Linear Models

Apply a nonlinear input transformation:

$$\phi(x_1, x_2) = [x_1 + x_2, x_1 \times x_2]$$




$\phi$ maps the data into a representation suitable for linear classification

$$\hat{y} = \phi(\boldsymbol{x})\boldsymbol{W} + b$$

# Limitations of Linear Models

Another effective mapping function:

$$\phi(\boldsymbol{x}) = g(\boldsymbol{x}\boldsymbol{W}' + \boldsymbol{b}')$$

$$\boldsymbol{W}' = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

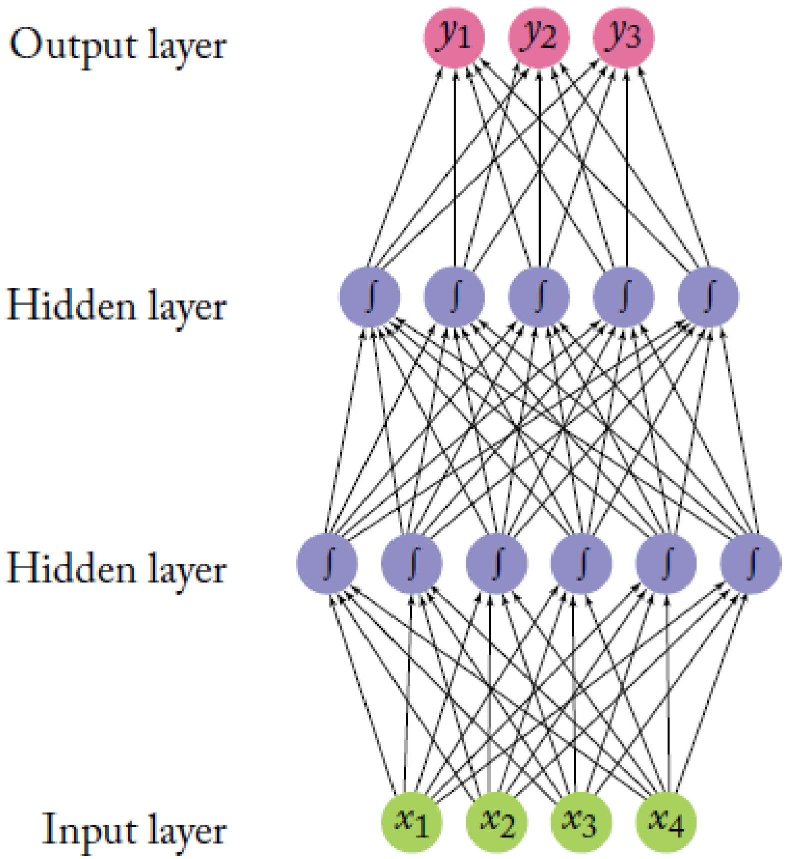
$$\boldsymbol{b}' = \begin{pmatrix} 0 & -1 \end{pmatrix}$$

$g(x) = \max(0, x) = \mathrm{ReLU}(x)$
applied to each dimension

# Feed-forward Neural Networks

- Many neuron-like threshold units
- Many weighted connections among units
- Highly parallel and distributed processing
- Automatic weight tuning

# Feed-forward Neural Networks



# Perceptron

A linear model

$$\mathrm{NN}_{\mathrm{Perceptron}}(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$

Parameters:

$\boldsymbol{W}$: weight matrix

$\boldsymbol{b}$: bias term

# Feed-forward Neural Networks

Multilayer perceptron (MLP) with one hidden layer

$$\mathrm{NN}_{\mathrm{MLP1}}(\boldsymbol{x}) = g\left(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}}\right)\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}}$$

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad \boldsymbol{W}^{\mathbf{1}} \in \mathbb{R}^{d_{in} \times d_1}$ $\quad \boldsymbol{b}^{\mathbf{1}} \in \mathbb{R}^{d_1}$

$\boldsymbol{W}^{\mathbf{2}} \in \mathbb{R}^{d_1 \times d_2}$ $\quad \boldsymbol{b}^{\mathbf{2}} \in \mathbb{R}^{d_2}$

$g$: nonlinear activation function

# Feed-forward Neural Networks

Multilayer perceptron with one hidden layer

$$\mathrm{NN}_{\mathrm{MLP1}}(\boldsymbol{x}) = g\left(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}}\right)\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}}$$

- First layer transforms the data into a good representation $g\left(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}}\right)$
- Second layer applies a linear classifier to that representation

# Feed-forward Neural Networks

Multilayer perceptron with two hidden layers

$$\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) = (g^2(g^1(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}})\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}}))\boldsymbol{W}^{\mathbf{3}} + \boldsymbol{b}^{\mathbf{3}}$$

Equivalently,

$$\boldsymbol{h}^{\mathbf{1}} = g^1(\boldsymbol{x}\boldsymbol{W}^{\mathbf{1}} + \boldsymbol{b}^{\mathbf{1}})$$

$$\boldsymbol{h}^{\mathbf{2}} = g^2(\boldsymbol{h}^{\mathbf{1}}\boldsymbol{W}^{\mathbf{2}} + \boldsymbol{b}^{\mathbf{2}})$$

$$\boldsymbol{y} = \boldsymbol{h}^{\mathbf{2}}\boldsymbol{W}^{\mathbf{3}} + \boldsymbol{b}^{\mathbf{3}}$$

$$\mathrm{NN}_{\mathrm{MLP2}}(\boldsymbol{x}) = \boldsymbol{y}$$

# Feed-forward Neural Networks

- NNs with many hidden layers: “deep” NNs (“deep” learning)
- Output of a NN: $d_{out}$ dimensional vector
- $d_{out} = 1$: regression (scoring)
- Binary classification: $\mathrm{sign}() = 1 \text{ or } -1$
- $d_{out} = k$: *k*-class classification (find the dimension (class) with the maximal value)
- $d_{out} = k$: softmax() transforms output vector into a probability distribution

# Representation Power

- MLP1 is a universal approximator – it can approximate with any desired non-zero amount of error a family of functions that includes all continuous functions on a closed and bounded subset of $\mathbb{R}^n$, and any function mapping from any finite dimensional discrete space to another.

# Representation Power

- The theory states that a representation exists but does not say how easy or hard it is to set the parameters
- Does not guarantee that a training algorithm will find the correct function
- Does not state how large the hidden layer should be
  - There exist NNs with many layers of bounded size that cannot be approximated by NNs with fewer layers unless these layers are exponentially large

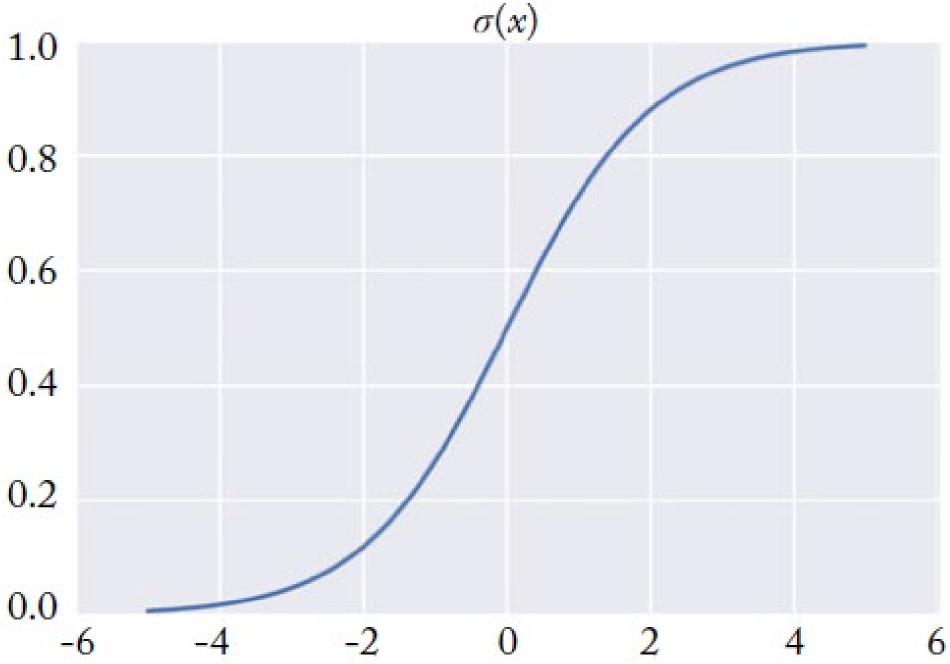
# Representation Power

- In practice, it is beneficial to use more complex architectures than MLP1

# Activation Functions

- Sigmoid (logistic)
- tanh
- Hard tanh
- Rectified linear unit (ReLU)

# Sigmoid Function




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

# Sigmoid function

- Nice property:

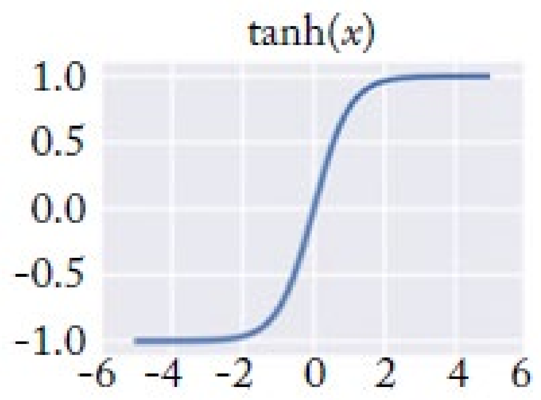
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot (1 - \frac{1}{1 + e^{-x}})$$

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

# Hyperbolic Tangent (tanh)

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



# Hyperbolic Tangent (tanh)

- Nice property:

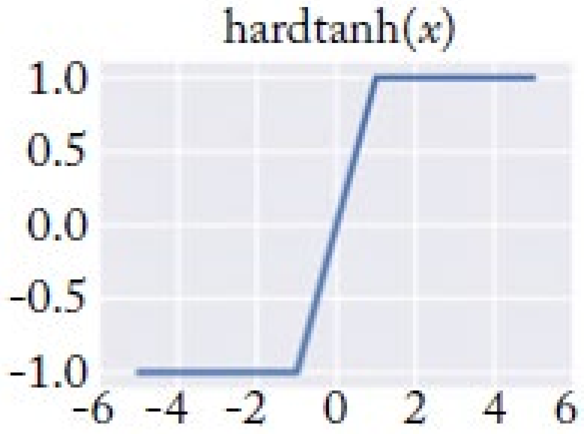
$$\tanh(\mathrm{x}) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{d}{dx}\tanh(\mathrm{x}) = \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

$$= 1 - \left(\frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2 = 1 - [\tanh(x)]^2$$

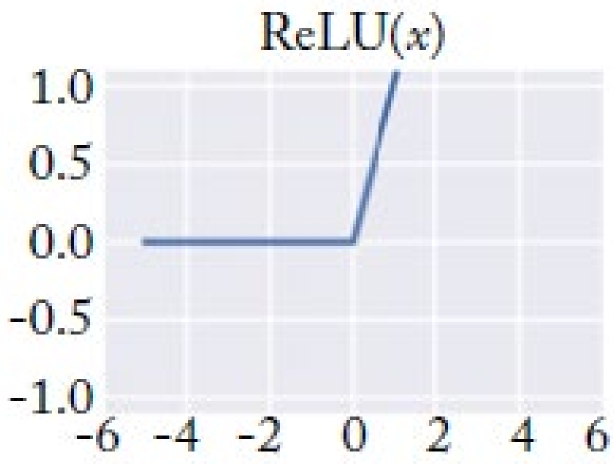
# Hard Hyperbolic Tangent

$$\text{hardtanh(x)} = \begin{cases} -1 & x < -1 \\ 1 & x > 1 \\ x & \text{otherwise} \end{cases}$$

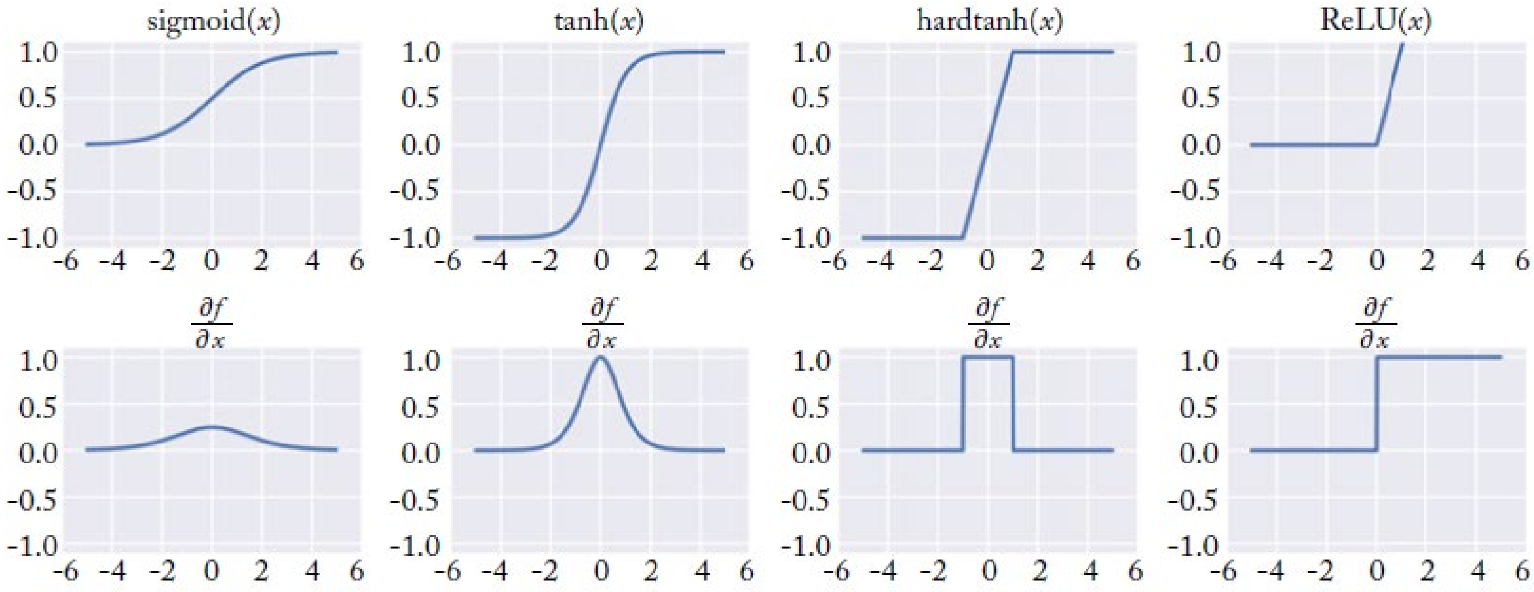


# Rectified Linear Unit (ReLU)

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & \text{otherwise} \end{cases}$$



# Activation Functions



# Dropout Regularization for NNs

- Prevent a NN from overfitting the training data (prevent it from learning to rely on specific weights)
- Randomly dropping (setting to 0) some of the neurons in the NN during stochastic gradient descent

# Dropout Regularization for NNs

$$\boldsymbol{h}^1 = g^1(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1)$$

$$\boldsymbol{m}^1 \sim \text{Bernoulli}(r^1)$$

$$\widetilde{\boldsymbol{h}}^1 = \boldsymbol{m}^1 \odot \boldsymbol{h}^1$$

$$\boldsymbol{h}^2 = g^2(\widetilde{\boldsymbol{h}}^1\boldsymbol{W}^2 + \boldsymbol{b}^2)$$

$$\boldsymbol{m}^2 \sim \text{Bernoulli}(r^2)$$

$$\widetilde{\boldsymbol{h}}^2 = \boldsymbol{m}^2 \odot \boldsymbol{h}^2$$

$$\boldsymbol{y} = \widetilde{\boldsymbol{h}}^2\boldsymbol{W}^3 + \boldsymbol{b}^3$$

$$\text{NN}_{\text{MLP2}}(\boldsymbol{x}) = \boldsymbol{y}$$

Without dropout:

$$\boldsymbol{h}^1 = g^1(\boldsymbol{x}\boldsymbol{W}^1 + \boldsymbol{b}^1)$$
$$\boldsymbol{h}^2 = g^2(\boldsymbol{h}^1\boldsymbol{W}^2 + \boldsymbol{b}^2)$$
$$\boldsymbol{y} = \boldsymbol{h}^2\boldsymbol{W}^3 + \boldsymbol{b}^3$$
$$\text{NN}_{\text{MLP2}}(\boldsymbol{x}) = \boldsymbol{y}$$

$\boldsymbol{m}^1, \boldsymbol{m}^2$: random masking vectors (elements $= 0$ or $1$)

$\odot$: element−wise multiplication