# DS 4400

## Machine Learning and Data Mining I
## Spring 2022

Alina Oprea
Associate Professor
Khoury College of Computer Science
Northeastern University

April 4 2022

# Announcements

- HW 4 is due on April 8
  - Decision trees, ensembles, Naïve Bayes
- Project milestone is due on April 13
  - Template in Gradescope
  - We would like to see at least one trained ML model
  - Discuss any challenges
- Experential AI Institute opening on Wed, April 6
  - Poster session 12-4pm, needs registration
  - I will present on AI in Cybersecurity at 3pm
  - PhD student Giorgio Severi will give the lecture and tutorial on language models

# Outline

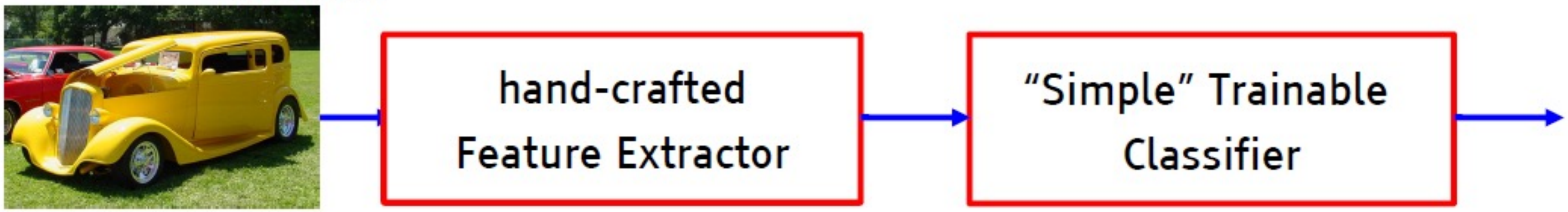
- Deep learning
  - Learning feature representations
  - Perceptron and limitations
- Feed-forward neural networks
  - Activations
  - Forward propagation
  - Vectorization
  - Softmax classifier

# References

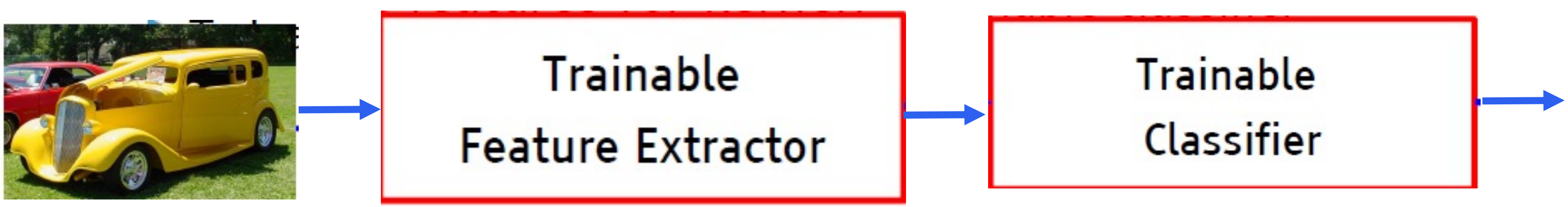
- Deep Learning books
  - https://d2l.ai/ (D2L)
  - https://www.deeplearningbook.org/ (advanced)
- Stanford notes on deep learning
  - http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

# Deep Learning

- **The traditional model of pattern recognition (since the late 50's)**
  - Fixed/engineered features (or fixed kernel) + trainable classifier

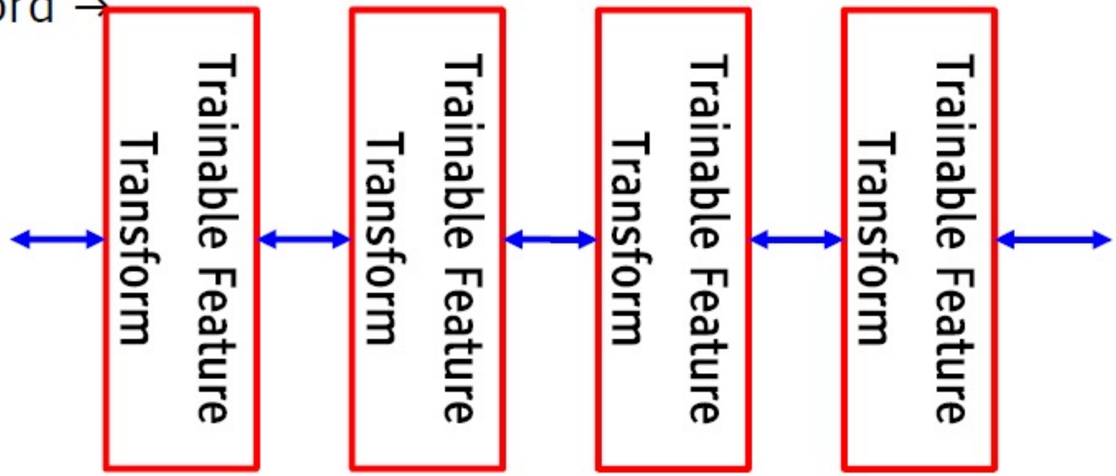



- End-to-end learning / Feature learning / Deep learning

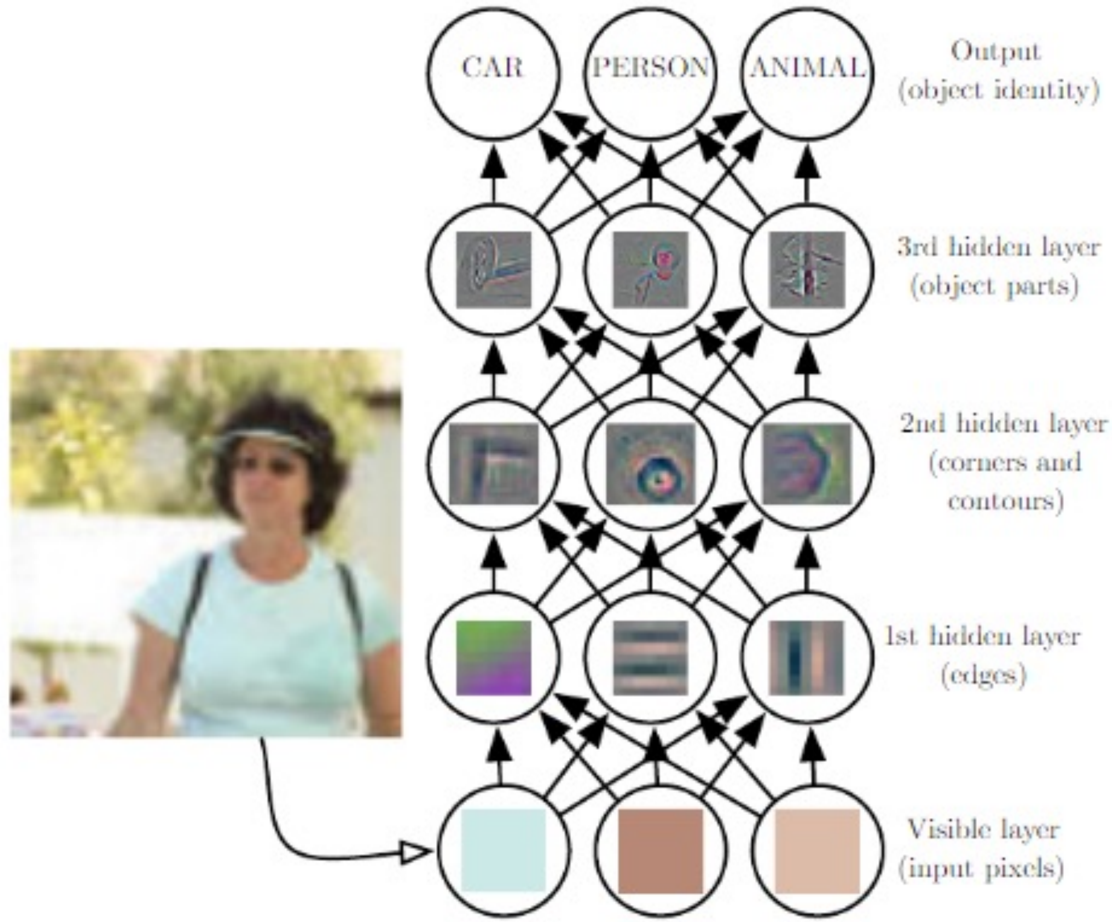


# Trainable Feature Hierarchy

- **Hierarchy of representations with increasing level of abstraction**
- **Each stage is a kind of trainable feature transform**
- **Image recognition**
  - Pixel → edge → texton → motif → part → object
- **Text**
  - Character → word → word group → clause → sentence → story
- **Speech**
  - Sample → spectral band → sound → … → phone → phoneme → word →



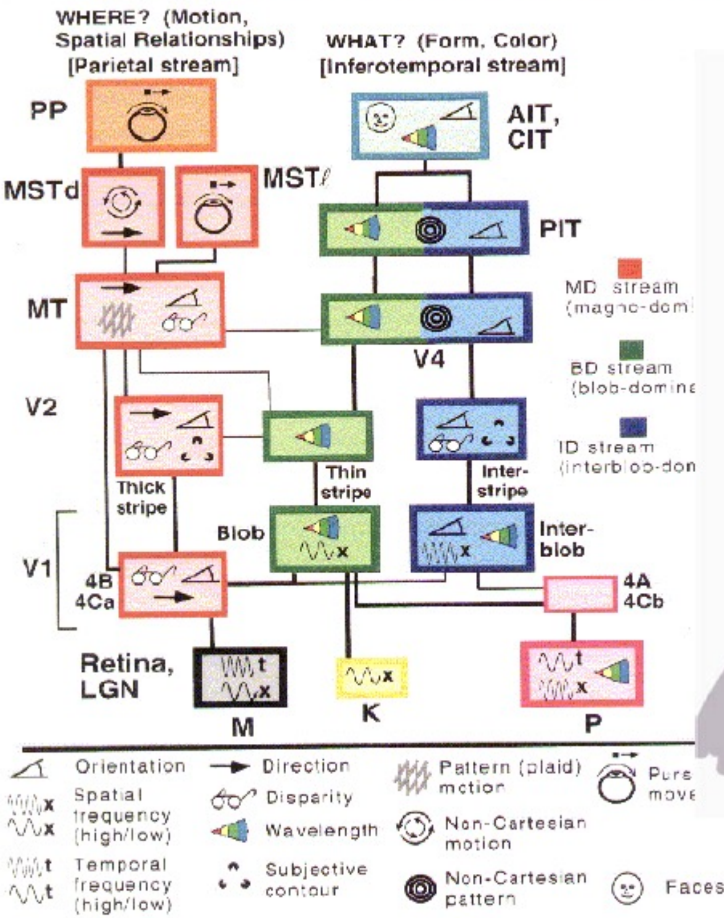
# Learning Representations

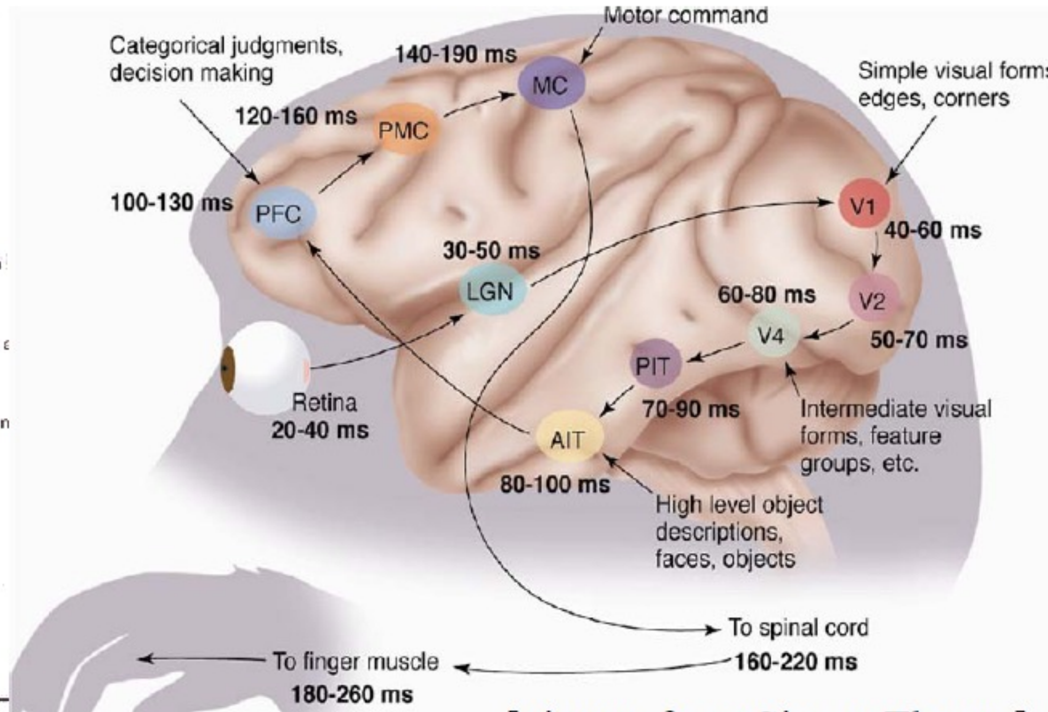



Deep Learning addresses the problem of learning hierarchical representations

# The Visual Cortex is Hierarchical

- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



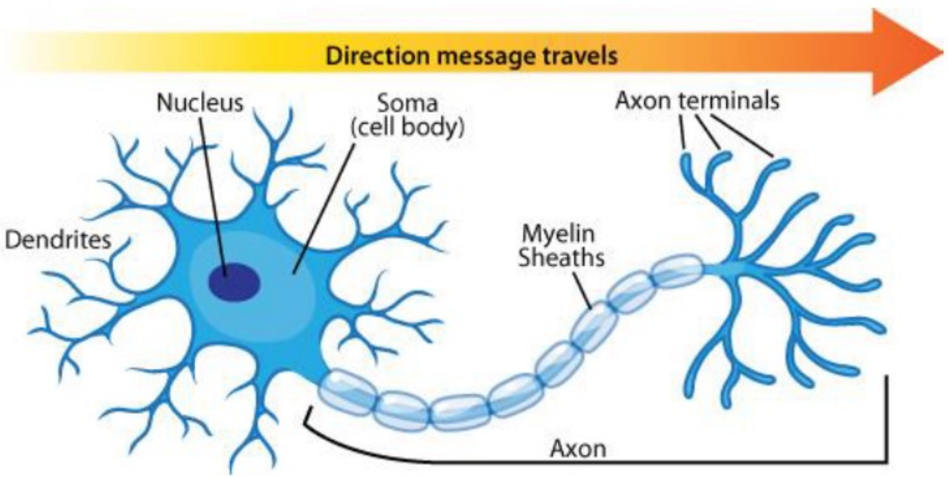



[picture from Simon Thorpe]

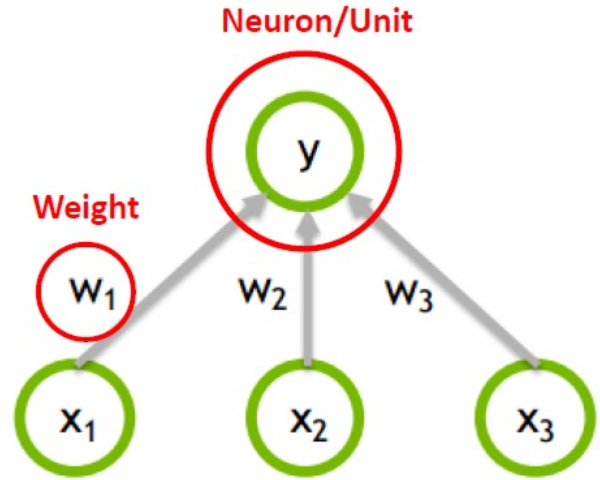
[Gallant & Van Essen]

# Analogy to Human Brain

## Human Brain




Biological Neuron




$$y=F(w_1x_1+w_2x_2+w_3x_3)$$

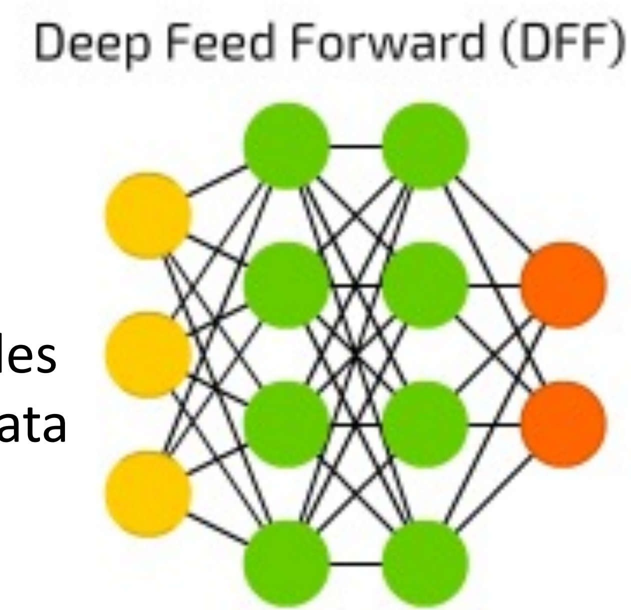
$$F(x)=\max(0,x)$$

Artificial Neuron

# Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

# Neural Networks



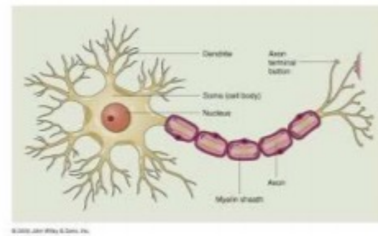

Input nodes
Training data

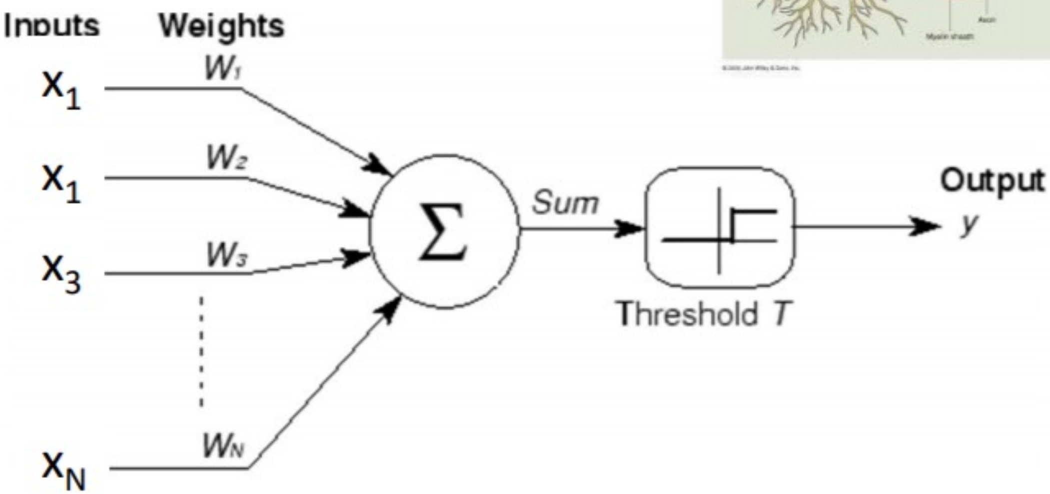
Output nodes

Hidden units
(Learned during training)

- Neural networks: made up from nodes connected by links
- Each link has a weight and activation function
- Feed-forward neural networks
  - Training data is input to left, prediction are output on right

# Perceptron






$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

- A threshold unit
  - "Fires" if the weighted sum of inputs exceeds a threshold

# The Perceptron

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance $(x_i, y_i)$

$$\boxed{\theta_j \leftarrow \theta_j - \frac{1}{2}\left(h_\theta(x_i) - y_i\right)x_{ij}}$$

  - If the prediction matches the label, make no change
  - Otherwise, adjust $\boldsymbol{\theta}$

# The Perceptron

- The perceptron uses the following update rule each time it receives a new training instance $(x_i, y_i)$

$$\boxed{\theta_j \leftarrow \theta_j - \frac{1}{2}\left(h_\theta(x_i) - y_i\right)x_{ij}}$$

# The Perceptron

- The perceptron uses the following update rule each time it receives a new training instance $(x_i, y_i)$

$$\boxed{\theta_j \leftarrow \theta_j - \frac{1}{2}\underbrace{\left(h_\theta(x_i) - y_i\right)}_{\text{either 2 or -2}} x_{ij}}$$

- Re-write as $\boxed{\theta_j \leftarrow \theta_j + y_i x_{ij}}$ (only upon misclassification)

Perceptron Rule: If $x_i$ is misclassified, do

$$\theta \leftarrow \theta + y_i\, x_i$$

# Online Perceptron

Let $\theta \leftarrow [0,0,\ldots,0]$
Repeat:
  Receive training example $(x_i, y_i)$
  If $y_i \theta^T x_i \leq 0$  // prediction is incorrect
    $\theta \leftarrow \theta + y_i\, x_i$
Until stopping condition

**Online learning** – the learning mode where the model update is performed each time a single observation is received

**Batch learning** – the learning mode where the model update is performed after observing the entire training set

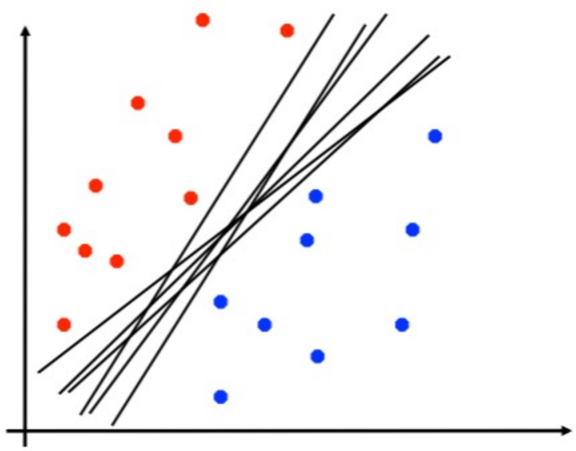
# Batch Perceptron

Let $\theta \leftarrow$[0,0,…,0]
Repeat:
$\Delta = [0,0, \ldots, 0]$
For $i = 1$ to $N$ // Consider all training examples
If $y_i\theta^T x_i \leq 0$ // Prediction is incorrect
$\Delta \leftarrow \Delta + y_i\, x_i.$ // Accumulate all errors
$\theta \leftarrow \theta + \frac{\Delta}{N}$ // Parameter update rule
Until stopping condition

- Guaranteed to find separating hyperplane if data is linearly separable

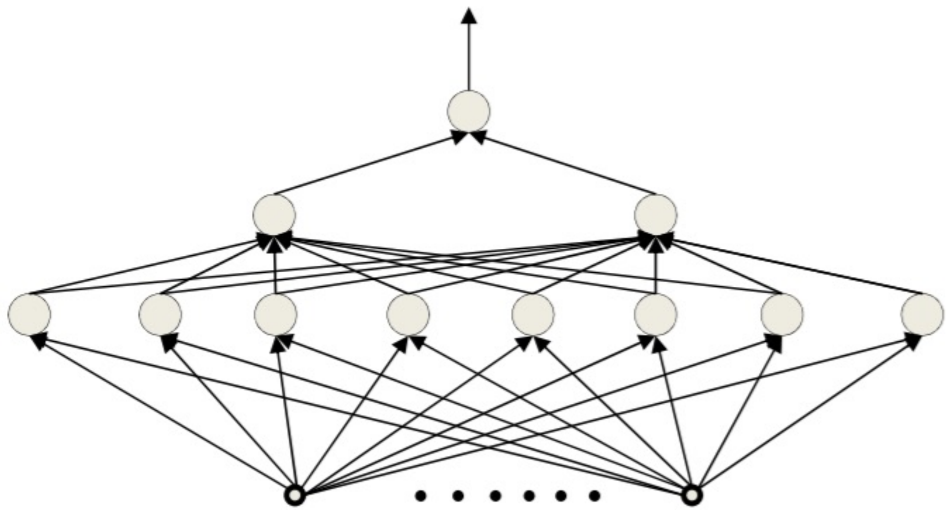
# Perceptron Limitations

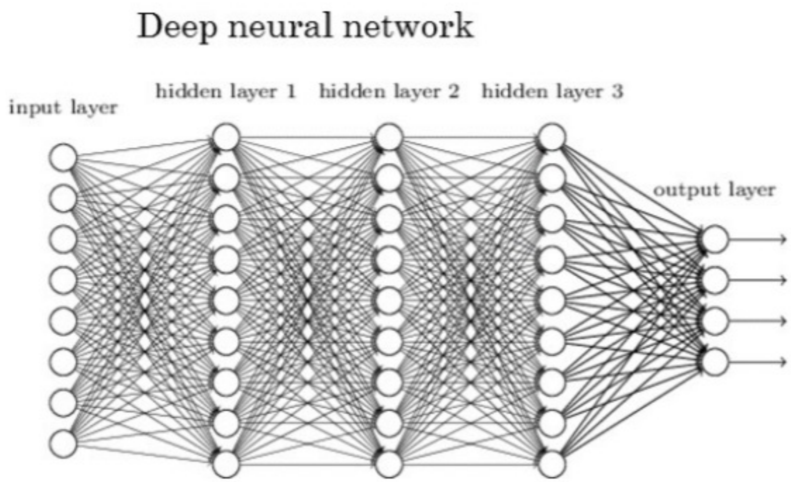
- Is dependent on starting point
- It could take many steps for convergence
- Perceptron can overfit
  - Move the decision boundary for every example
- It is a linear model



Which of this is optimal?

# Multi-Layer Perceptron



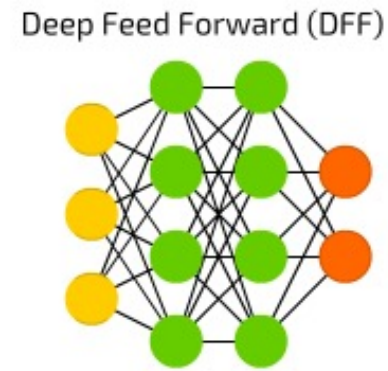



- A network of perceptrons
  - Generally "layered"

# Neural Network Architectures

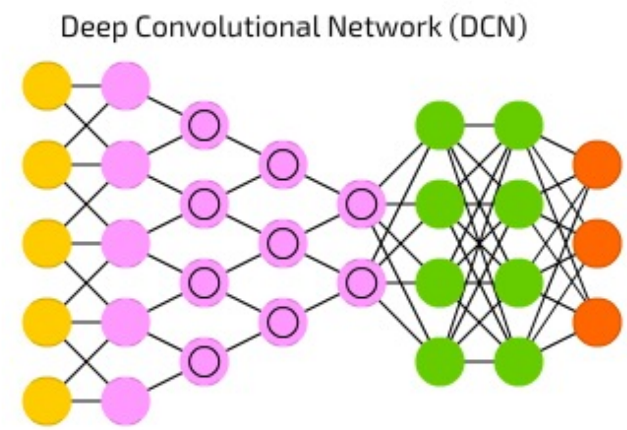
## Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



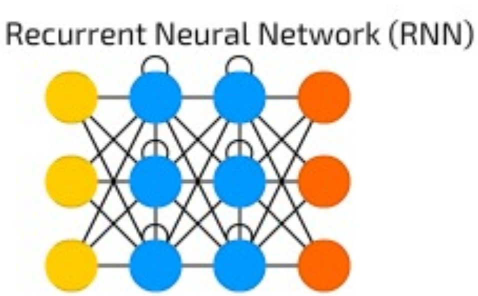

## Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

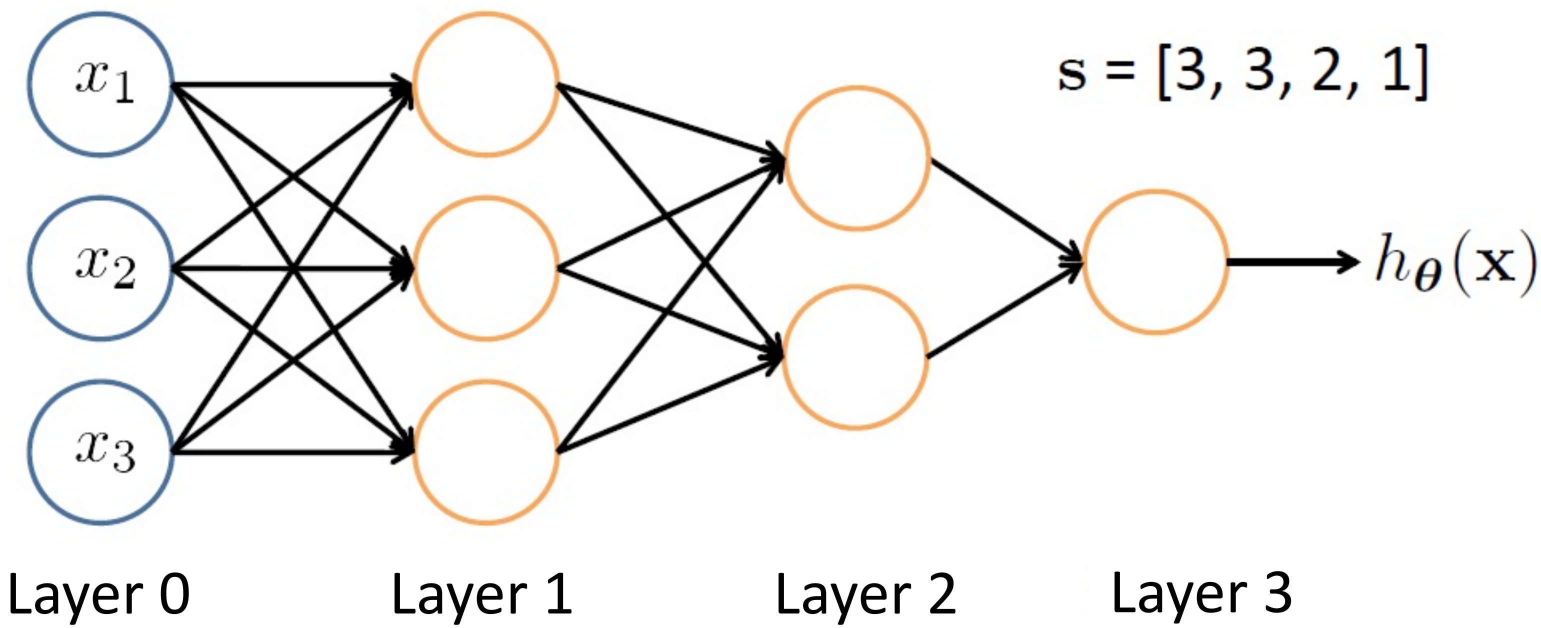



## Recurrent Networks

- Keep hidden state
- Have cycles in computational graph



# Feed-Forward Networks




$L$ denotes the number of layers

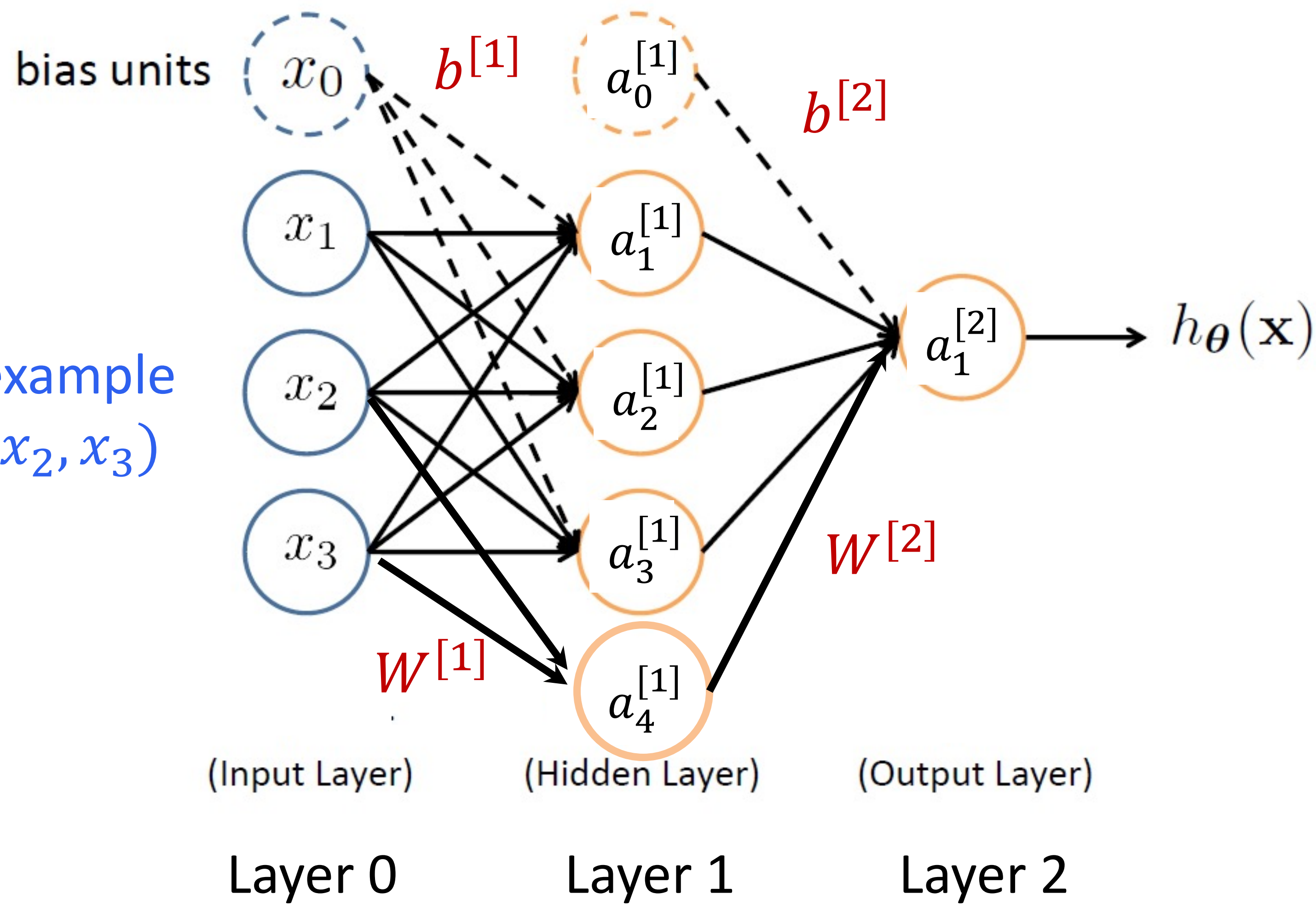
$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L\text{-}1}=K$ (# classes)

# Feed-Forward NN

- Hyper-parameters
  - Number of layers
  - Architecture (how layers are connected)
  - Number of hidden units per layer
  - Number of units in output layer
  - Activation functions
- Other
  - Initialization
  - Regularization

# Feed-Forward Neural Network




Training example
$x = (x_1, x_2, x_3)$

No cycles

$\theta = (b^{[1]}, W^{[1]}, b^{[2]}, W^{[2]})$

# Vectorization: First Layer

$$
\begin{array}{ccc}
z_1^{[1]} = W_1^{[1]} \, x + b_1^{[1]} & \text{and} & a_1^{[1]} = g(z_1^{[1]}) \\
\vdots & \vdots & \vdots \\
z_4^{[1]} = W_4^{[1]} \, x + b_4^{[1]} & \text{and} & a_4^{[1]} = g(z_4^{[1]})
\end{array}
$$

$$
\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} - & W_1^{[1]} & - \\ - & W_2^{[1]} & - \\ & \vdots & \\ - & W_4^{[1]} & - \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}
$$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

Linear

$$a^{[1]} = g(z^{[1]})$$

Non-Linear

# Vectorization: Second Layer

Output layer

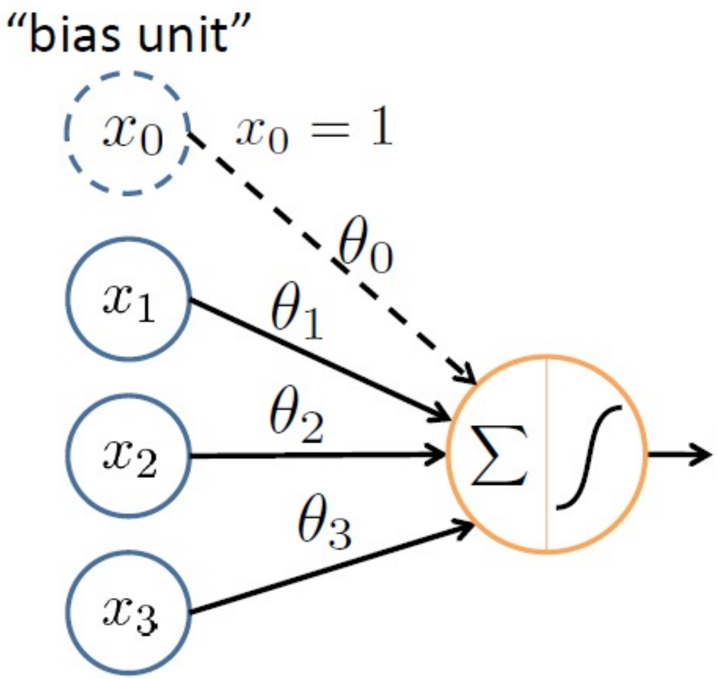
$$z_1^{[2]} = W_1^{[2]^T} a^{[1]} + b_1^{[2]} \quad \text{and} \quad a_1^{[2]} = g(z_1^{[2]})$$

$$\boxed{\underbrace{z^{[2]}}_{1\times1} = \underbrace{W^{[2]}}_{1\times4} \underbrace{a^{[1]}}_{4\times1} + \underbrace{b^{[2]}}_{1\times1} \quad \text{and} \quad \underbrace{a^{[2]}}_{1\times1} = g(\underbrace{z^{[2]}}_{1\times1})}$$

# Hidden Units

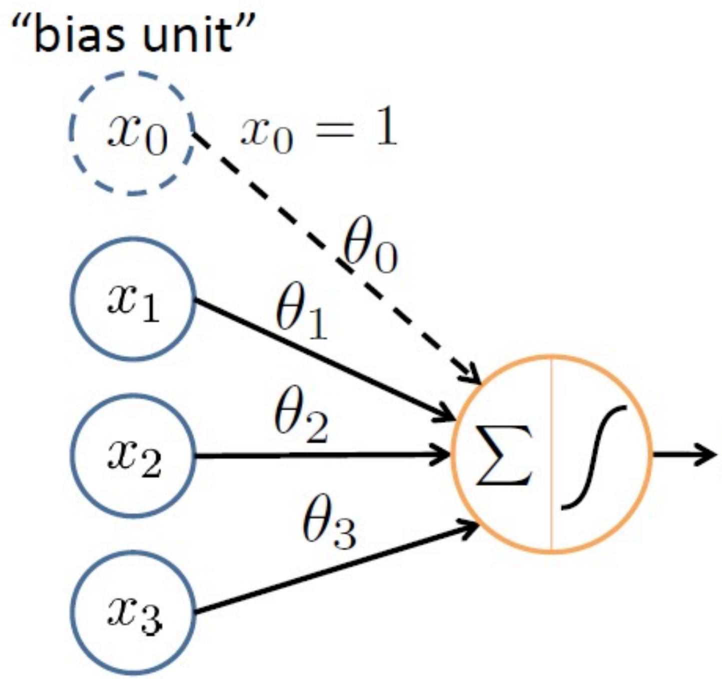
- Layer 1
  - First hidden unit:
    - Linear: $z_1^{[1]} = \mathrm{W}_1^{[1]\,\mathrm{T}} x + \mathrm{b}_1^{[1]}$
    - Non-linear: $a_1^{[1]} = \mathrm{g}(\mathrm{z}_1^{[1]})$
  - …
  - Fourth hidden unit:
    - Linear: $z_4^{[1]} = \mathrm{W}_4^{[1]\,\mathrm{T}} x + \mathrm{b}_4^{[1]}$
    - Non-linear: $a_4^{[1]} = \mathrm{g}(\mathrm{z}_4^{[1]})$
- Terminology
  - $a_i^{[j]}$ - Activation of unit i in layer j
  - $\mathrm{g}$ - Activation function
  - $W^{[j]}$ - Weight matrix controlling mapping from layer j-1 to j
  - $b^{[j]}$ - Bias vector from layer j-1 to j

# Logistic Unit: A simple NN




$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

# Logistic Unit: A simple NN




$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\intercal \mathbf{x}\right) \quad = g(w^T x + b)$$

$$= \frac{1}{1 + e^{-\boldsymbol{\theta}^\intercal \mathbf{x}}}$$

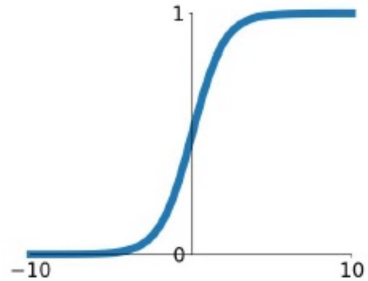
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

No hidden layers

# Non-Linear Activation Functions

**Sigmoid**

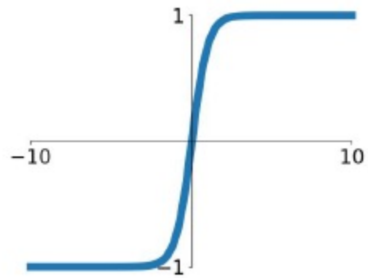
$$\sigma(x) = \frac{1}{1+e^{-x}}$$




Binary Classification

**tanh**

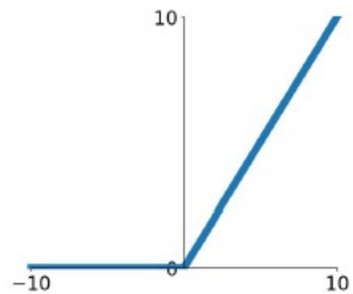
$$\tanh(x)$$




Regression

**ReLU**

$$\max(0, x)$$




Intermediary layers

# Training Neural Networks

- Input training dataset D
  - Number of features: d
  - Labels from K classes
- First layer has d+1 units (one per feature and bias)
- Output layer has K units
- Training procedure determines parameters that optimize loss function
  - Backpropagation
  - Learn optimal $W^{[i]}, b^{[i]}$ at layer *i*
- Evaluation of a point done by forward propagation

# Forward Propagation

- The input neurons first receive the data features of the object. After processing the data, they send their output to the first hidden layer.
- The hidden layer processes this output and sends the results to the next hidden layer.
- This continues until the data reaches the final output layer, where the output value determines the object's classification.
- This entire process is known as Forward Propagation, or Forward prop.