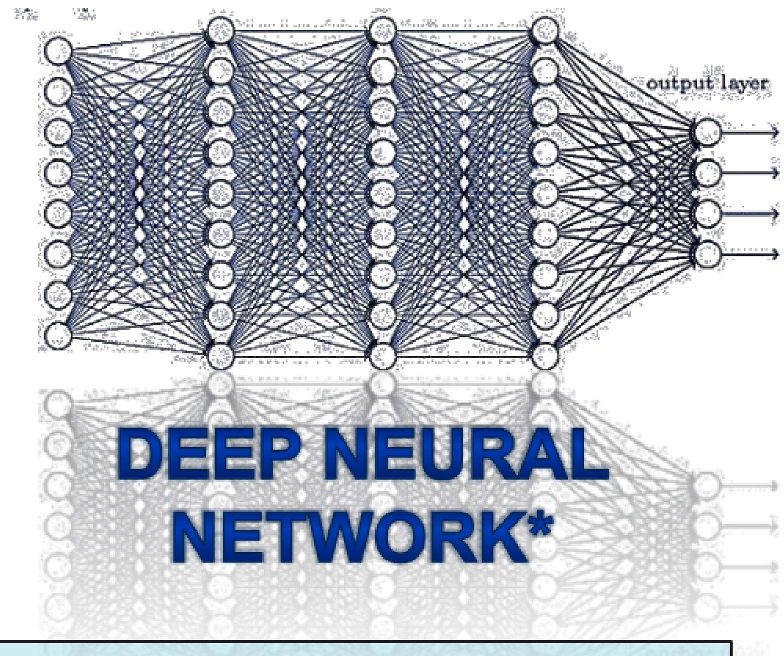
Deep Learning

- **Success of Deep Learning techniques attributable to concurrence of big data sets, scalable hardware, and high-level software**



Data

Software

NVIDIA CUDA

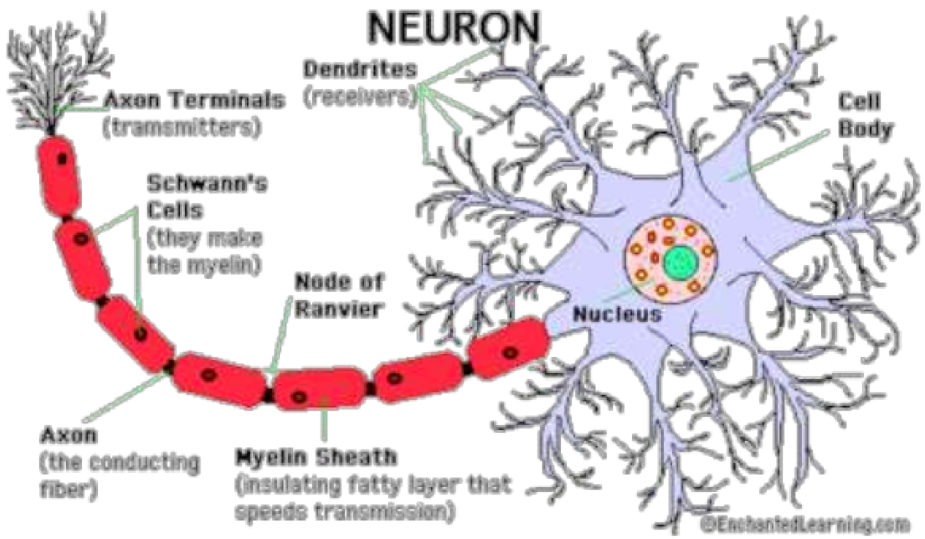
theano

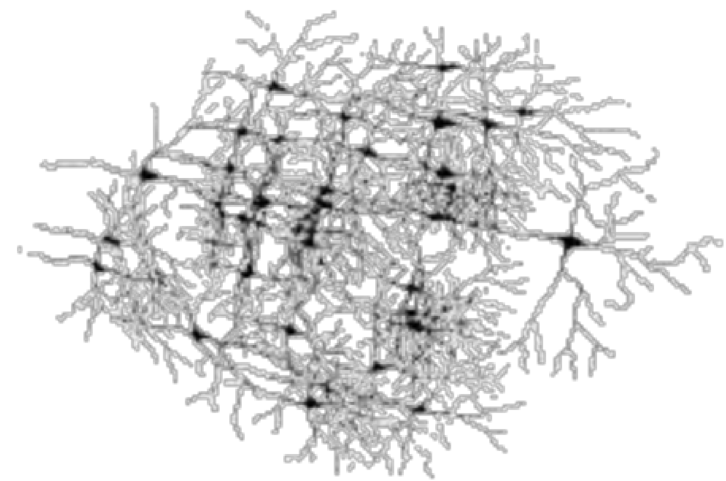
TensorFlow

Hardware

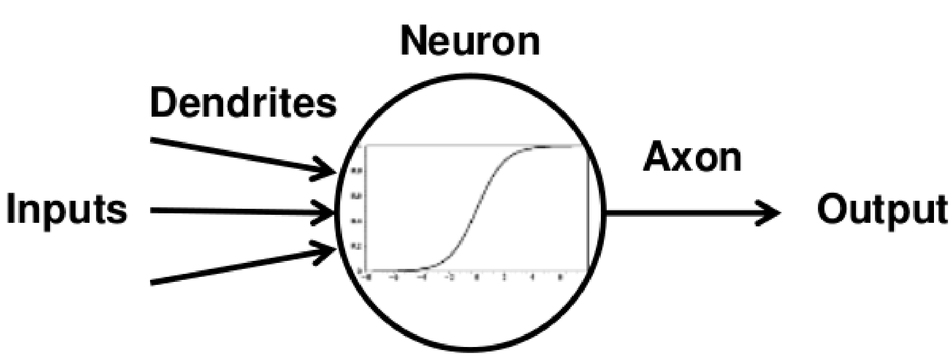
(Deep) Neural Network Overview

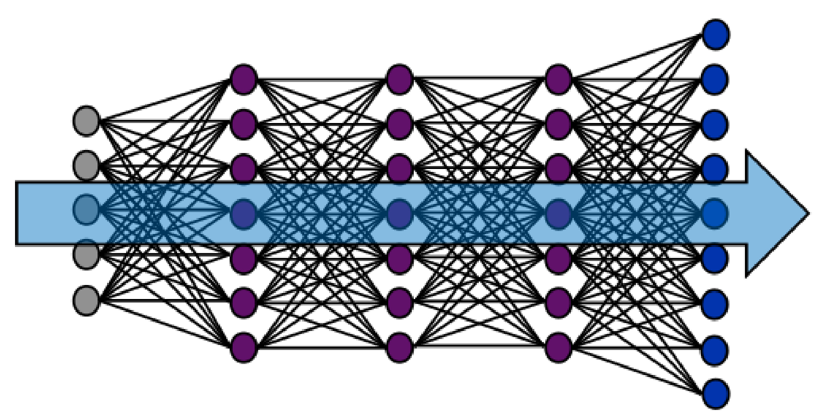
- **Neural networks were originally motivated by the brain**





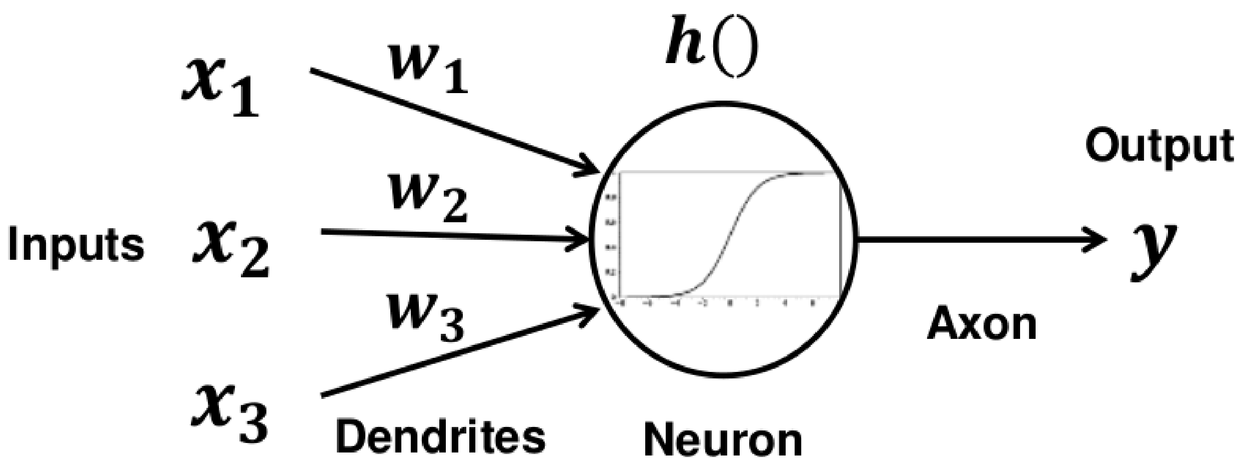
- **Artificial neural nets – neurons in layers (multi-layer perceptron)**





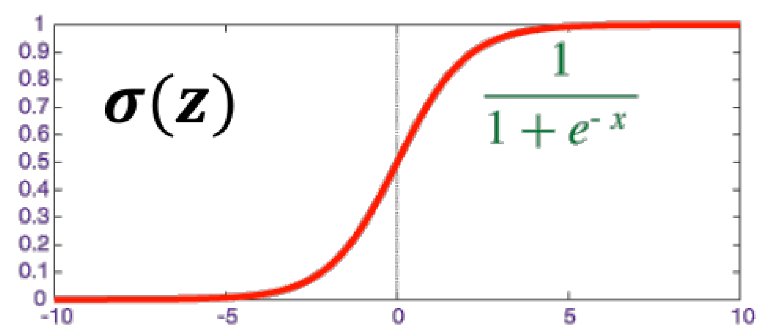
(Deep) Neural Network Overview

- The input from other neurons are “activations”



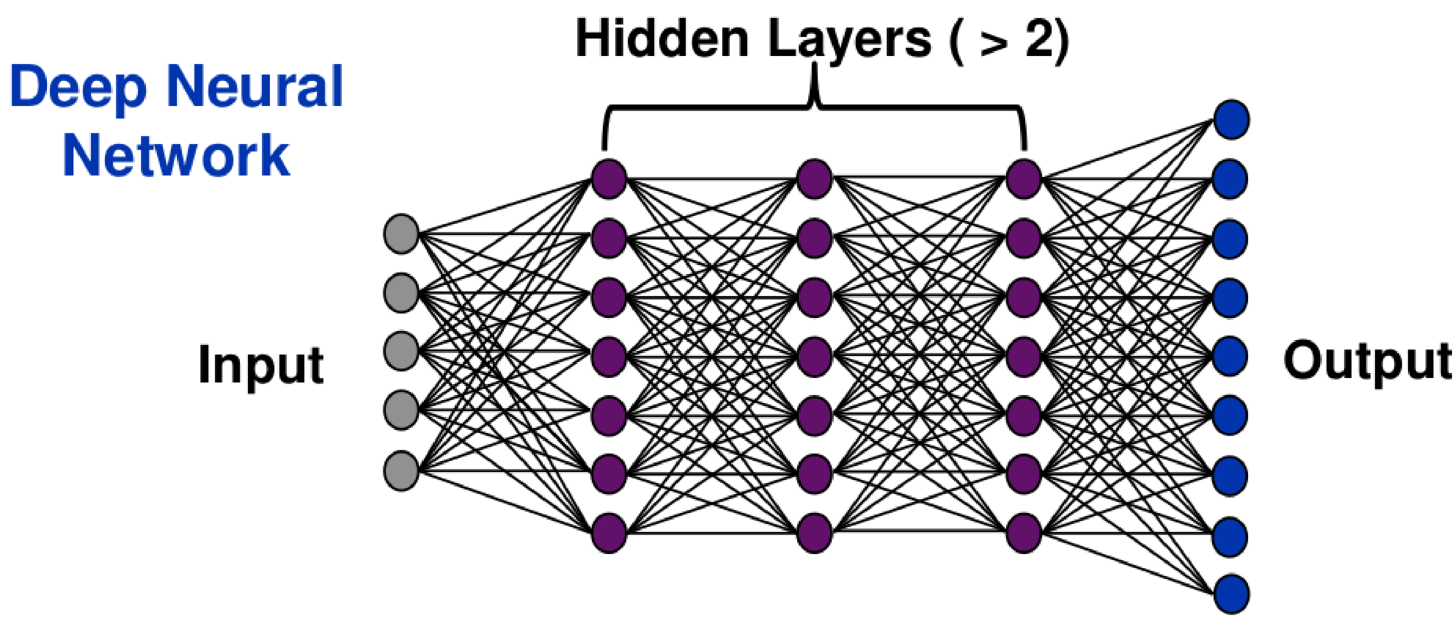
- $h()$ is the activation function: $y = h(w_1x_1 + w_2x_2 + w_3x_3)$

- A typical $h()$ is the sigmoid:



(Deep) Neural Network Overview

- Neural networks are <u>not</u> accurate models of the brain
- But they are statistical models*
- Prior to 2006 it was difficult to train deep neural networks
  - ("deep" means > 2 hidden layers)



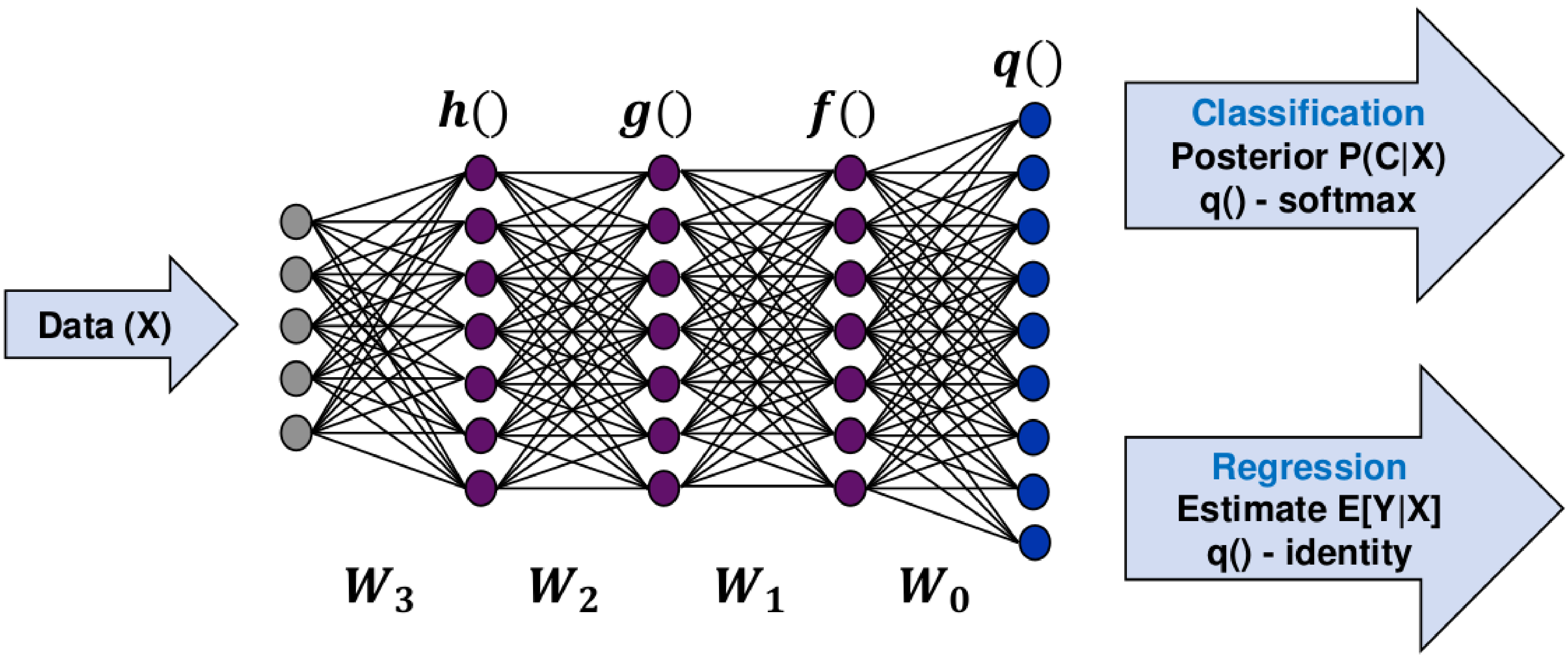
*Mike D. Richard and Richard P. Lippmann, "Neural Network Classifiers Estimate Bayesian a Posteriori Probabilities," *Neural Computation,* 3, 461-483,1992

(Deep) Neural Network Overview

- **Neural networks are a composition of functions:**

$$q(W_0 f(W_1 \boldsymbol{g}(\boldsymbol{W_2 h}(\boldsymbol{W_3 x}))))$$

- **Last layer activation $q()$ depends on task:**



(Deep) Neural Network Overview

Task	Classification	Regression
Output	Posterior $p(c\|x)$	Mean $E[y\|x]$
Final activation	Soft-max	Identity
Objective function	Normalized cross entropy (NCE)	Mean squared error (MSE)
Data requirements	Class labels (c)	Targets (y)

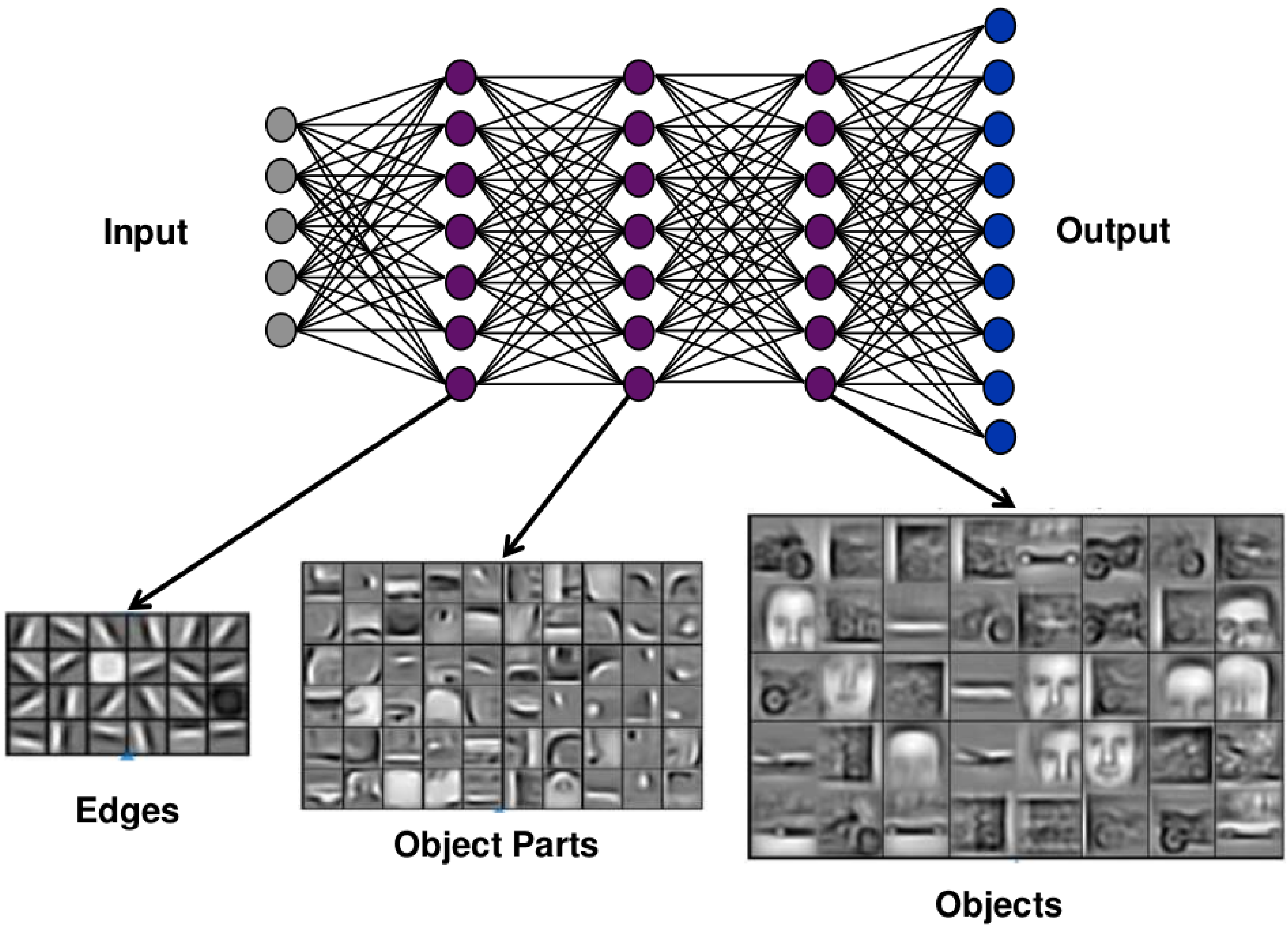
- (D)NN Training Algorithm – Stochastic Gradient Descent:
  1. Initialize network weights (random or use “pretraining”)
  2. Compute gradient using “back propagation” (chain rule)
  3. Update weights using the gradient and a learning rate
  4. Stop when error rate goes up on held-out data
- Note: for N nodes and M layers, # param is MN^2

DNNs for Classification

- Two major approaches when applying DNN systems to classification tasks
- End-to-end (direct)
  - DNN does feature extraction and final classification
  - Mainly applicable for tailored, closed-set classification with lots of data per class
  - Can be tricky to modify classes and adapt
- Representation learning (indirect)
  - DNN is for feature extraction only
  - Typically done via bottleneck features learned with a end-to-end DNN using proxy classes (also referred to as transfer learning)
  - Separation of feature generation and final classifier allows for more flexible classifier (classes and compensation/adaptation)

DNN Representation Learning

- **DNNs learn more abstract representation at deeper layers**



Bottleneck Features

- We can take advantage of the abstraction:
  - A narrow bottleneck can be used to reduce dimensionality
- First part of network is used for feature extraction
- Last part of network is a simple classifier