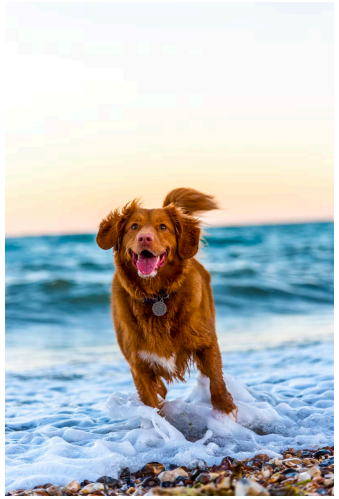
CS M146: Introduction to Machine Learning
Neural Networks - Part 1

Aditya Grover

https://aditya-grover.github.io/ @adityagrover_

Curse of Dimensionality

- Real-world is high-dimensional



To get an A+ in machine learning, you need to know a few things.

1) To understand why machine learning works the way it does.

2) To be able to analyze the underlying mechanisms of a model.

3) To have a deep understanding of the mathematics behind neural networks.

4) To be able to read papers in top tier machine learning journals.

5) To be able to code in the languages that are best suited for machine learning.

Should our hypothesis class remain overly simple?

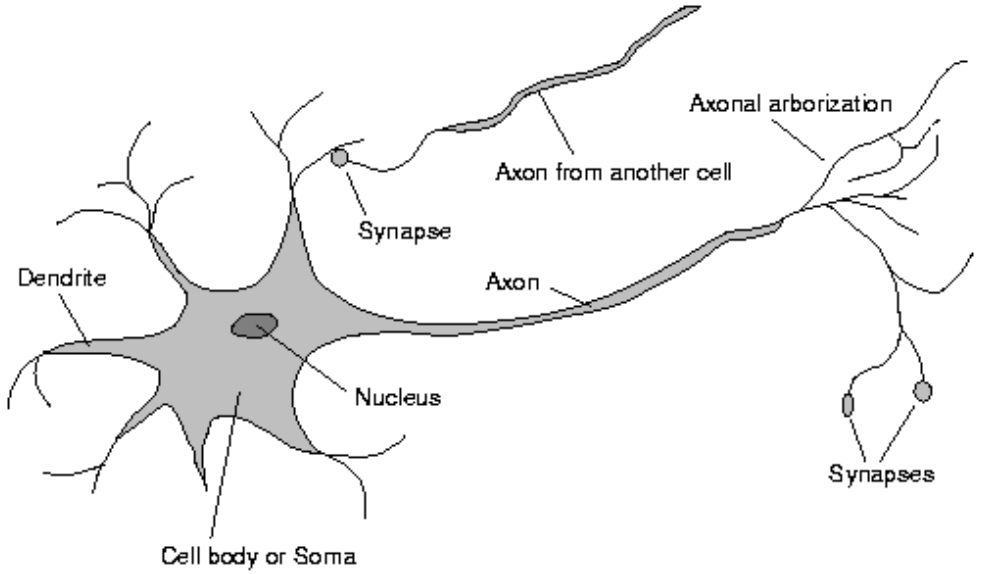
From Simple to Complex Models

- **Strategy 1:** Use **kernels**
  - Project inputs to a higher dimensional basis
  - Hypothesis is linear in $\boldsymbol{\theta}$
    - For classification, only the final discretization step (e.g., taking the sign function, or applying the logistic function) is non-linear w.r.t. $\boldsymbol{\theta}$
  - Can "kernelize" any linear model, such as linear regression, perceptrons, logistic regression, SVMs.

- **Strategy 2:** Use **neural networks**

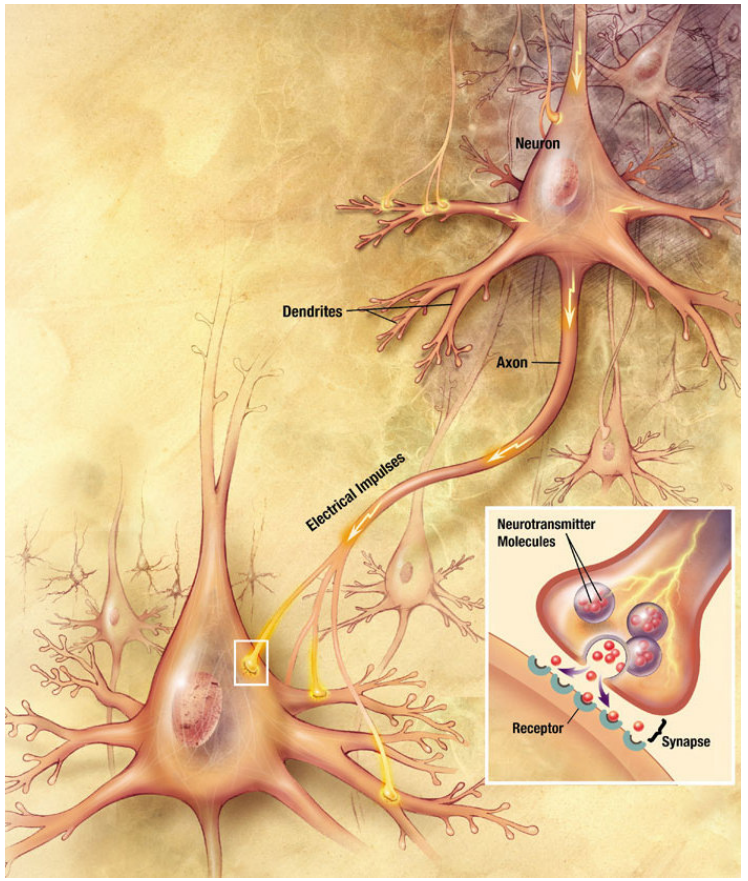
What is a neural network?

An **artificial neural network** represents a hypothesis class that is inspired by the structure and functioning of **biological neurons**



Biological Neural Networks

- Brain function (thought) occurs as the result of the firing of **neurons**
- Neurons connect to each other through **synapses**, which propagate potential (electrical impulses) by releasing neurotransmitters
  - Synapses can be excitatory (potential-increasing) or inhibitory (potential-decreasing), and have varying **activation thresholds**
  - Learning occurs as a result of the synapses' **plasticity**: They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and 10^{14} synapses in the human brain!

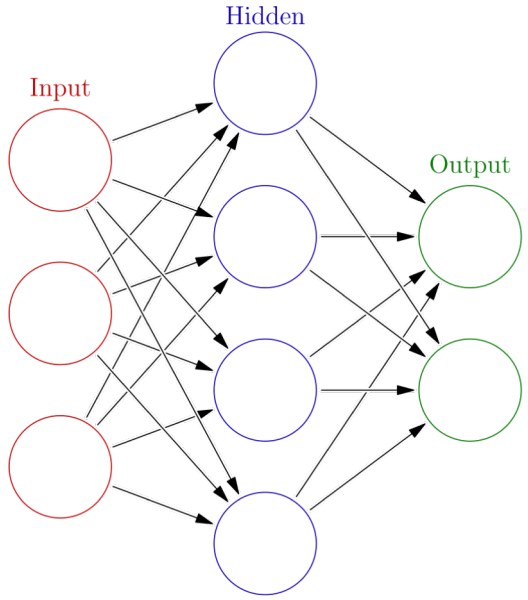


Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Artificial Neural Networks

An **artificial neural network** (ANN) is a representation of hypothesis class with 3 special features:

- **Neurons:** Artificial neurons (nodes) that process and transmit information.
- **Layers:** Neurons are organized into layers, including an input layer, one or more hidden layers, and an output layer.
- **Connections:** Neurons are interconnected through weighted connections that transmit signals.



Logistic Regression for Binary Classification

3 Steps:

- Representation: $\boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}$

  Linear functions: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{sigmoid}(\boldsymbol{w}^T\boldsymbol{x} + b)$

- Define a loss function

  Logistic loss:

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n}[y^{(i)}\log f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)})\log(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})]$$

- Optimize loss to find best hypothesis

  Gradient descent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha(f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})\boldsymbol{x}^{(i)}$

For all neural net lectures, we will use $f_{\boldsymbol{\theta}}$ to denote our hypothesis. [Why? h is used for denoting something different]

Example: Image Classification

Input: image

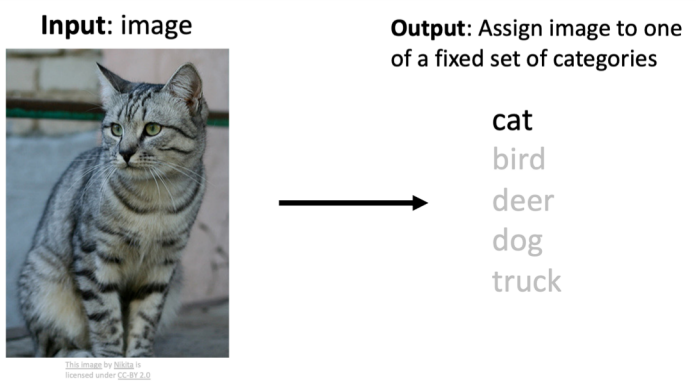
Output: Assign image to one of a fixed set of categories

cat
bird
deer
dog
truck

Array of **32x32x3** numbers
(3072 numbers total)

Multi-Class Classification

- Input attributes: $\boldsymbol{x} \in \mathbb{R}^d$
- Labels are represented as one-hot vectors: $\boldsymbol{y} \in \mathbb{R}^C$
- Assign an index i to each class:
  - cat: 1
  - bird: 2
  - deer: 3
  - dog: 4
  - truck: 5
- For any input $\boldsymbol{x}$, set entries of $\boldsymbol{y}$ as
  - $y_c = 1$ if true class is c and 0 otherwise
  - Note: we are assuming vector indices start from 1 (and not 0)



$\boldsymbol{y}$ =[1,0,0,0,0]

Logistic Regression for Multi-Class Classification

- Inputs: $\boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{y} \in \mathbb{R}^C$
- Representation: $W \in \mathbb{R}^{C \times d}, \boldsymbol{b} \in \mathbb{R}^C$

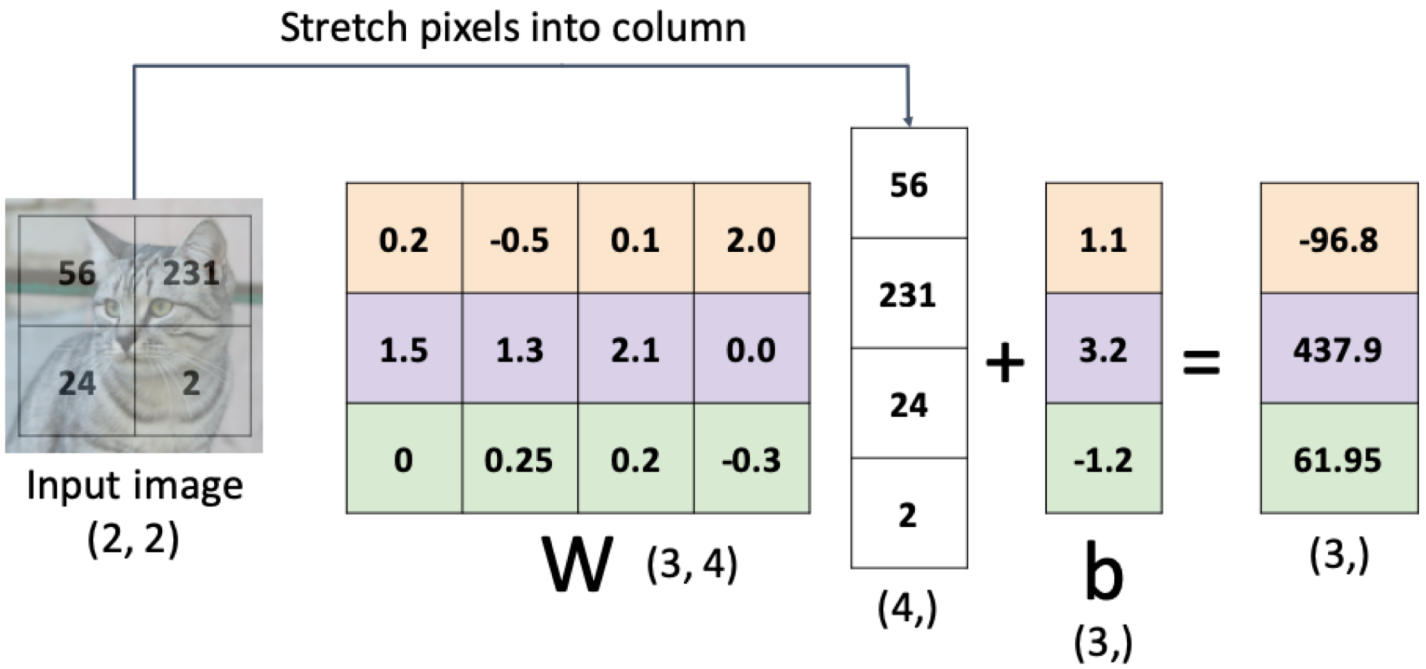
  score vector $\boldsymbol{s} = W\boldsymbol{x} + \boldsymbol{b}$

  hypothesis $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathrm{softmax}(\boldsymbol{s}) = \frac{\exp \mathbf{s}}{\sum_j \exp s_j}$

- Multi-class Logistic loss:

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \sum_{c=1}^{C} y_c^{(i)} \log f_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right)$$

Example: 2x2 image, 3 classes



From Logistic Regression to Neural Nets

Inputs: $\boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{y} \in \mathbb{R}^C$

Output: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{s})$

- **Logistic Regression**

$$\boldsymbol{s} = W\boldsymbol{x} + \boldsymbol{b}$$

  - Parameters: $W \in \mathbb{R}^{C \times d}, \boldsymbol{b} \in \mathbb{R}^C$

- **1-hidden layer neural net**

$$\boldsymbol{s} = W_2 \max(0, W_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

  - Parameters: $W_1 \in \mathbb{R}^{H \times d}, W_2 \in \mathbb{R}^{C \times H}, \boldsymbol{b}_1 \in \mathbb{R}^H, \boldsymbol{b}_2 \in \mathbb{R}^C$

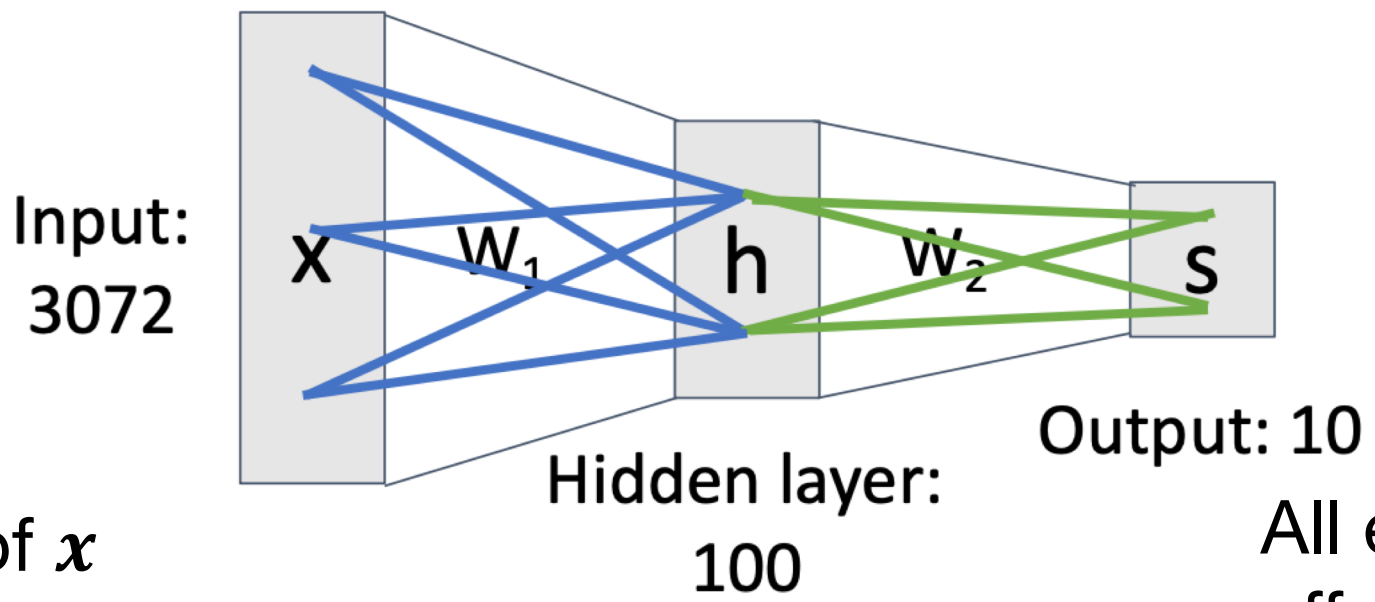
1-Hidden Layer Neural Network

C = 10

$$\boldsymbol{h} = \max(0, W_1 \boldsymbol{x} + \boldsymbol{b}_1)$$
$$\boldsymbol{s} = W_2 \boldsymbol{h} + \boldsymbol{b}_2$$

Element (i, j) of W_1 gives the effect on h_i from x_j

Element (i, j) of W_2 gives the effect on s_i from h_j



All elements of $\boldsymbol{x}$ affect all elements of $\boldsymbol{h}$

All elements of $\boldsymbol{h}$ affect all elements of $\boldsymbol{s}$

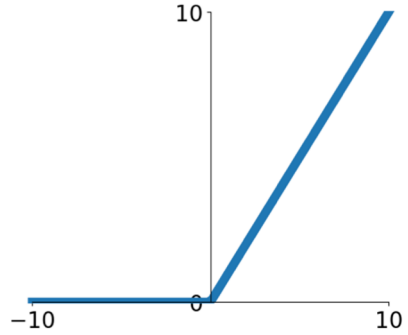
Fully-connected neural network. A.k.a. "Multi-Layer Perceptron" (MLP)

Activation Functions

$$\boldsymbol{s} = W_2 \max(0, W_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2$$

Example of an **activation function** called rectified linear units (ReLU)

$$\text{ReLU}(z) = \max(0, z)$$



Activation function is any non-linearity that excites a neuron.

What if there is no activation function?

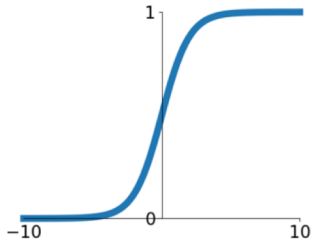
$\boldsymbol{s} = W_2(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 = W\boldsymbol{x} + \boldsymbol{b}$ where $W = W_2W_1$, $\boldsymbol{b} = W_2\boldsymbol{b_2} + \boldsymbol{b_1}$

Linear classifier!

Activation Functions

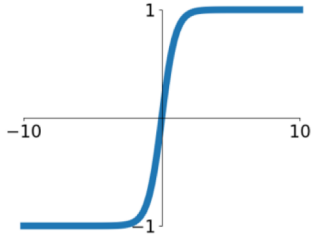
Sigmoid

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



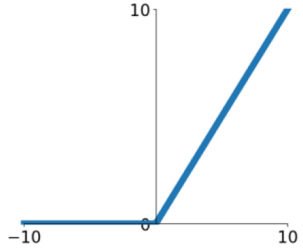
tanh

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



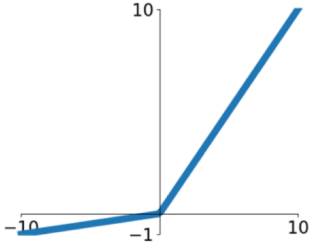
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.2x, x)$$

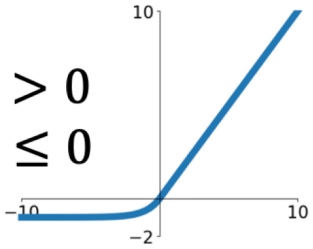


Softplus

$$\log(1 + \exp(x))$$

ELU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$$

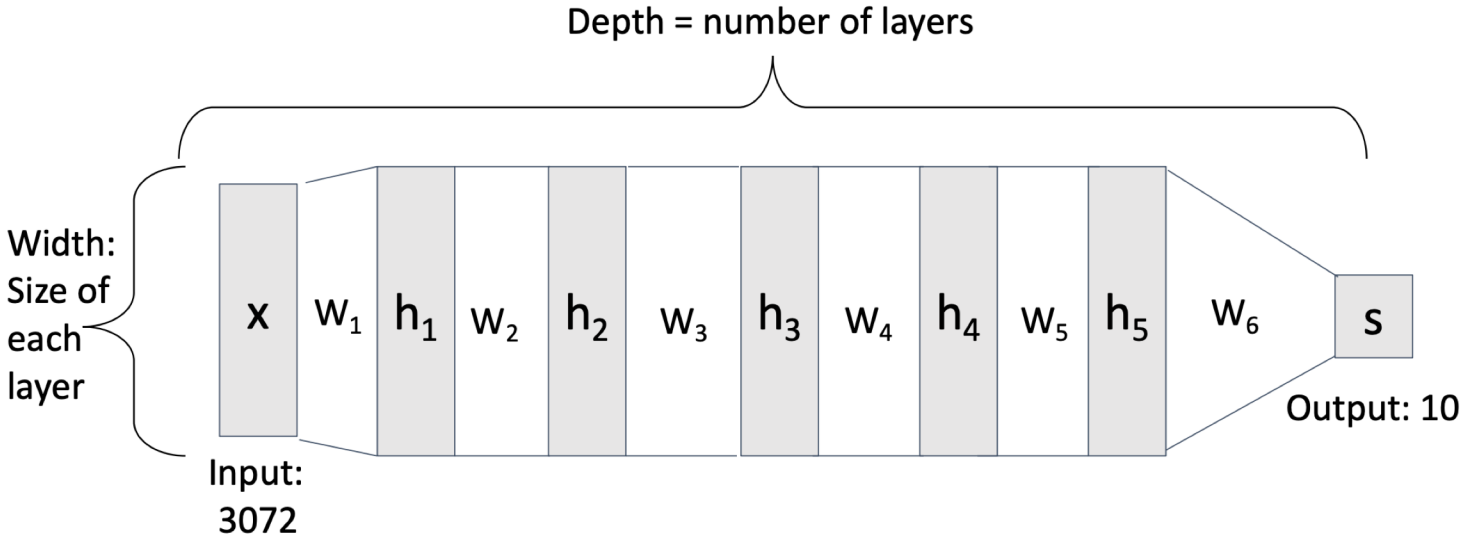


Deep Neural Network

- **2-hidden layer neural net**

$$\boldsymbol{s} = W_3 \max(0,\ W_2 \max(0, W_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3$$

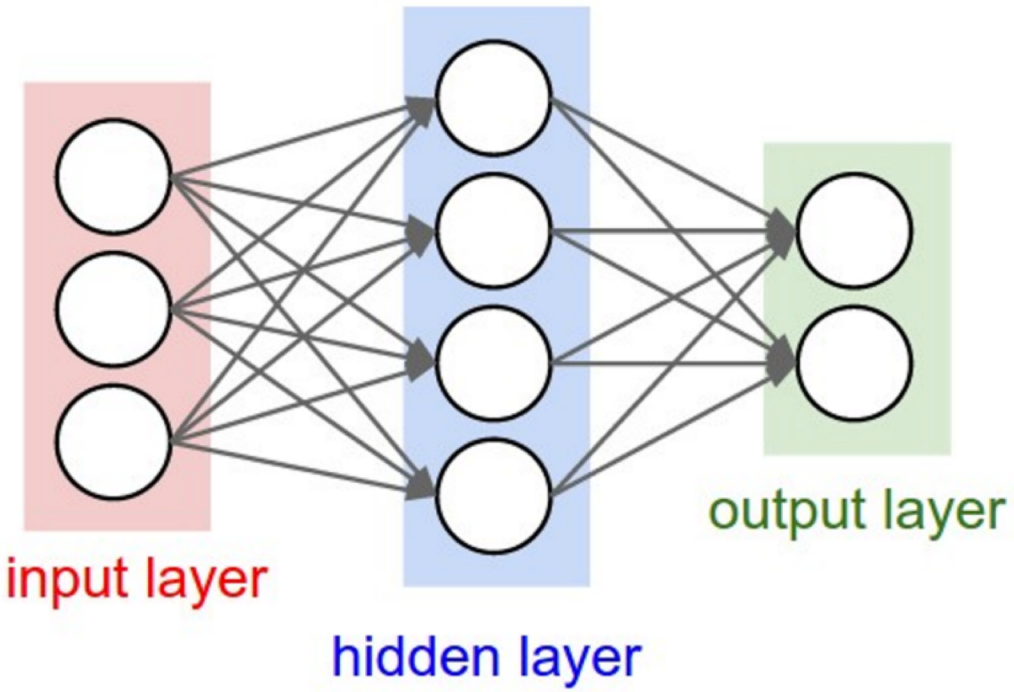
- **5-hidden layer neural net**



$$\boldsymbol{s} = W_6 \max(W_5 \max(W_4 \max(W_3\ \max(0,\ W_2\ \max(0, W_1\ \boldsymbol{x})))))$$

Network depth, activation function, layer widths are all hyperparameters

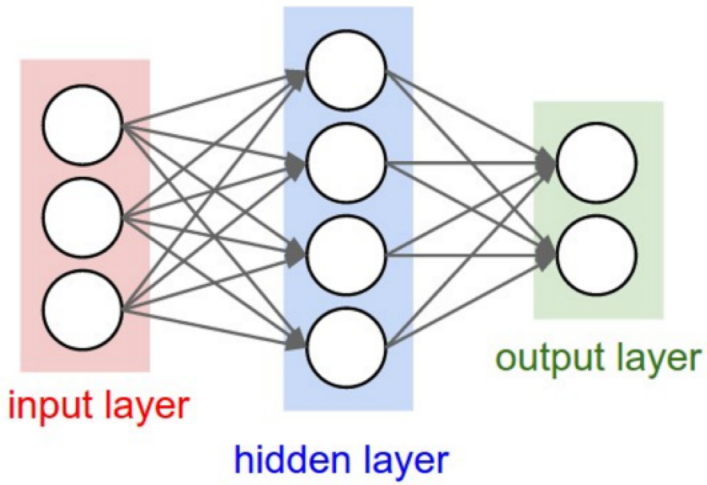
Neural Net in <20 lines!




```
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
for t in range(10000):
  h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  dy_pred = 2.0 * (y_pred - y)
  dw2 = h.T.dot(dy_pred)
  dh = dy_pred.dot(w2.T)
  dw1 = x.T.dot(dh * h * (1 - h))
  w1 -= 1e-4 * dw1
  w2 -= 1e-4 * dw2
```

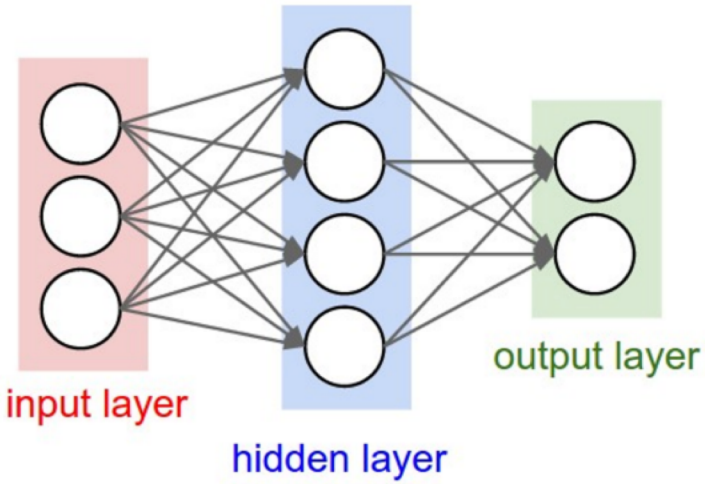
Neural Net in <20 lines!



Initialize weights and data

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Neural Net in <20 lines!

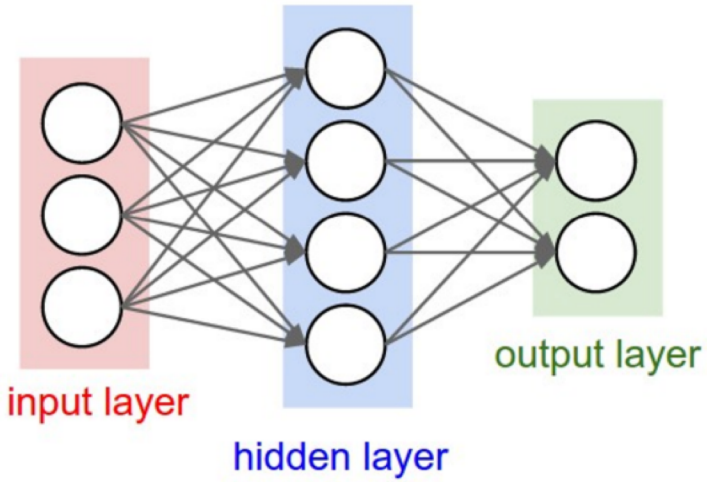


Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Neural Net in <20 lines!



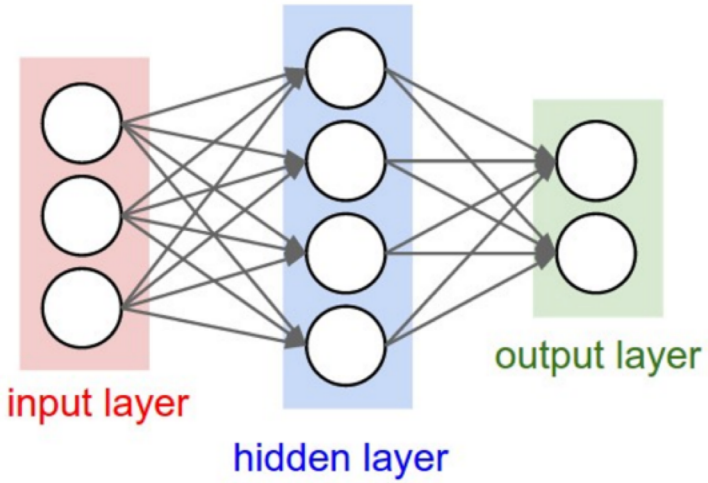
Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

Compute gradients

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Neural Net in <20 lines!



Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

Compute gradients

SGD step

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Are Artificial NNs same as Biological NNs?

- No. Be careful with brain analogies!
- Unlike artificial neurons, biological neurons:
  - Have many different types
  - Dendrites which can also perform complex non-linear computations
  - Synapses are not a single weight but a complex nonlinear dynamical system
- Biological neural nets take much less power than GPU-powered artificial neural nets today

Why Neural Nets?

Universal Approximation Theorem (without proof): A neural network with one hidden layer can approximate any multivariate function with arbitrary precision*

- Universal approximation tells us:
  - Neural nets can represent any function
- Universal approximation DOES NOT tell us:
  - Whether we can actually learn any function with SGD
  - How much data we need to learn a function
- kNN is also a universal approximator but very different from NNs in theory and practice

*Many technical conditions: Only holds on compact subsets of input domain; function must be continuous; need to define “arbitrary precision”; etc

Summary

- Neural Networks
  - Biologically **inspired** models for machine learning

- Key properties
  - Compute non-linear functions using activation functions
  - Flexible architecture design (multiple layers, layer widths, activations, multiple outputs)
  - Universal approximators