

---

---

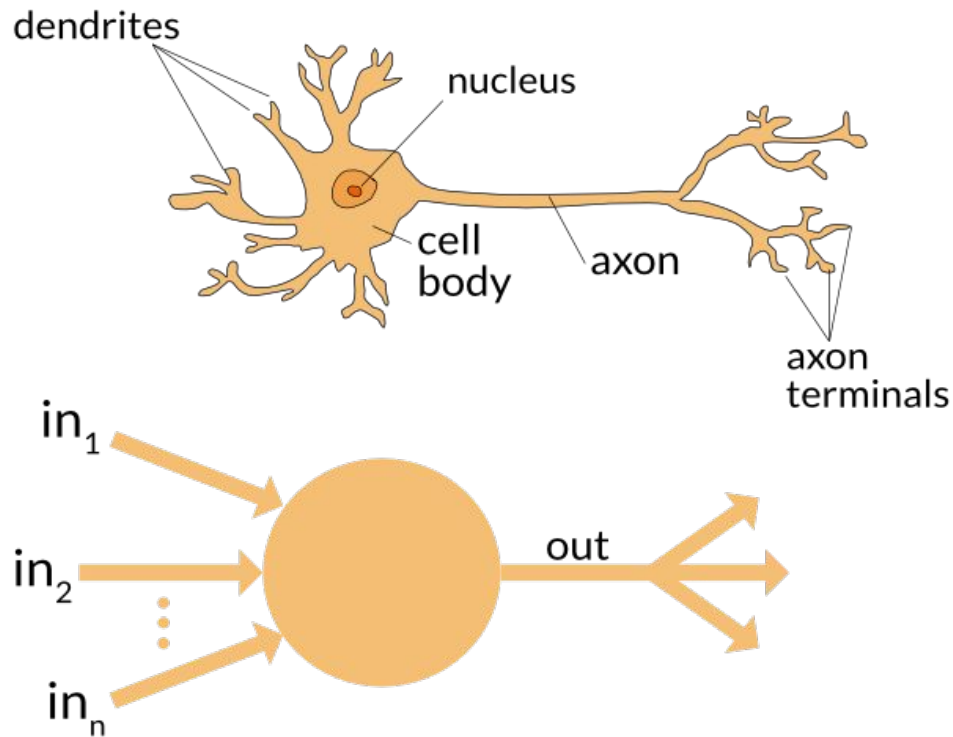
# MSSP 608 Recitation 5

Deep Learning Review

---

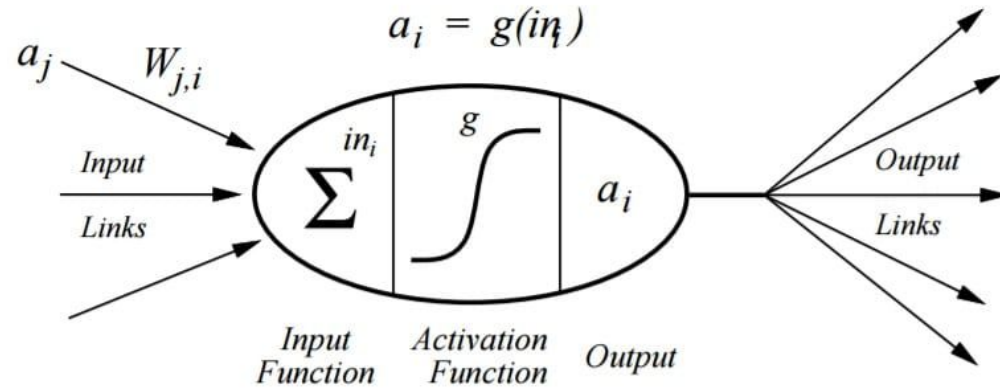
---

# Early Deep Learning (Perceptron)



---

# Perceptron Math

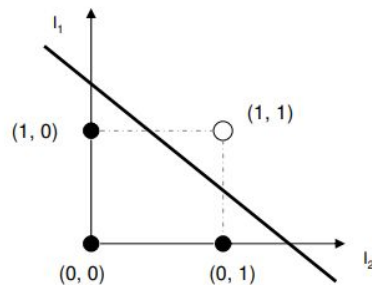


$$a_i = g\left(\sum_j W_{j,i} a_j\right)$$

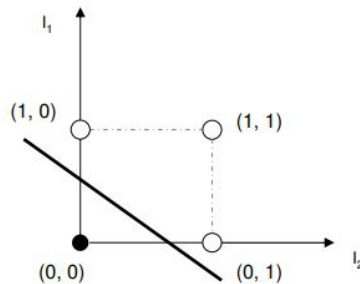
---

# First AI Winter

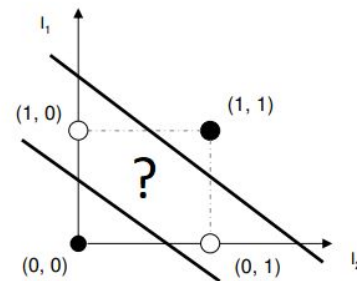
AND		
$I_1$	$I_2$	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
$I_1$	$I_2$	out
0	0	0
0	1	1
1	0	1
1	1	1

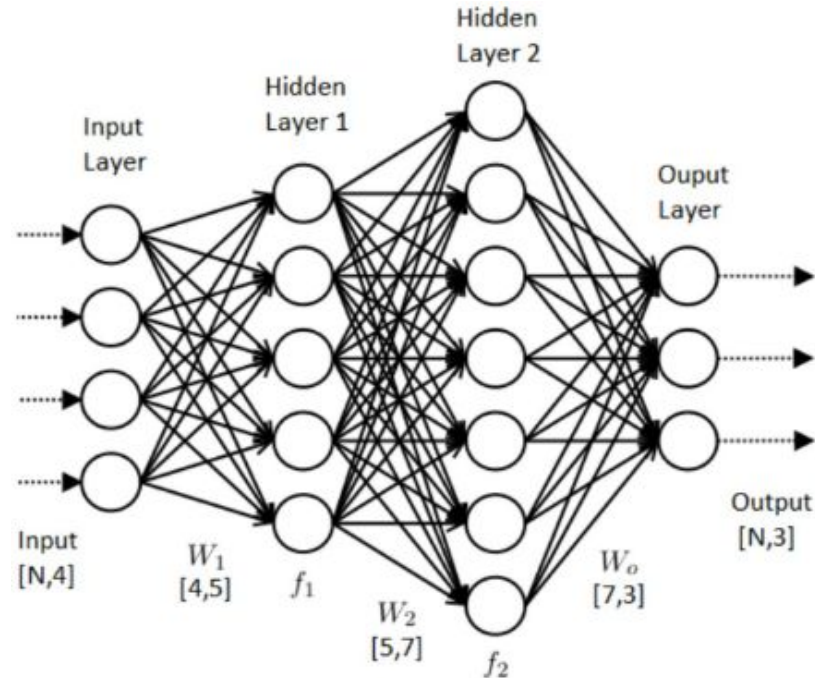


XOR		
$I_1$	$I_2$	out
0	0	0
0	1	1
1	0	1
1	1	0



---

# Solution - Multilayer Perceptrons



---

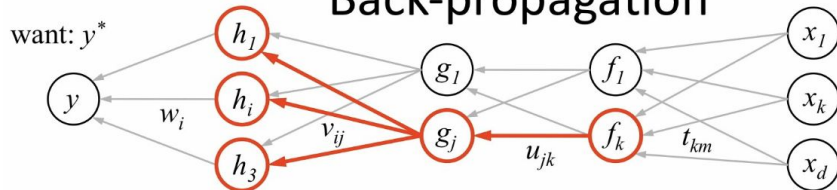
# How to train MLP quickly?

- Step 1: Initialize the weights of the network
  - Step 2: For epoch in epochs:
    - (a): Get a batch of data
    - (b): Put the batch of data through the network
    - (c): Compute the value of the loss function
    - (d): Use gradient descent + **backpropagation** to compute gradients
    - (e): Update the weights to be  $W - c * dW$  where  $c$  is the learning rate
  - Step 3: Save the model
- 
- Modern Deep Learning works well because it also can take good advantage of **powerful GPU hardware**
-

# What is backpropagation

Basically, a clever way to compute the chain rule using dynamic programming

## Back-propagation



1. receive new observation  $\mathbf{x} = [x_1 \dots x_d]$  and target  $y^*$
2. **feed forward:** for each unit  $g_j$  in each layer  $1 \dots L$  compute  $g_j$  based on units  $f_k$  from previous layer:  $g_j = \sigma \left( u_{j0} + \sum_k u_{jk} f_k \right)$
3. get prediction  $y$  and error  $(y - y^*)$
4. **back-propagate error:** for each unit  $g_j$  in each layer  $L \dots 1$

(a) compute error on  $g_j$

$$\frac{\partial E}{\partial g_j} = \sum_i \underbrace{\sigma'(h_i)}_{\text{should } g_j \text{ be higher or lower?}} \underbrace{v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each  $u_{jk}$  that affects  $g_j$

(i) compute error on  $u_{jk}$

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \underbrace{\sigma'(g_j)}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

## Taking Derivatives Using The Chain Rule

$$F(x) = (x^2 + 1)^{1/2}$$

## The Chain Rule

$$[f(g(x))]' = f'(g(x)) \cdot g'(x)$$

$$F'(x) = \frac{x}{\sqrt{x^2 + 1}}$$

---

# Why GPU?

- The GPU is very good at parallelizable operations
- Can have thousands of cores rather than just 2-64

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 \end{bmatrix}$$



---

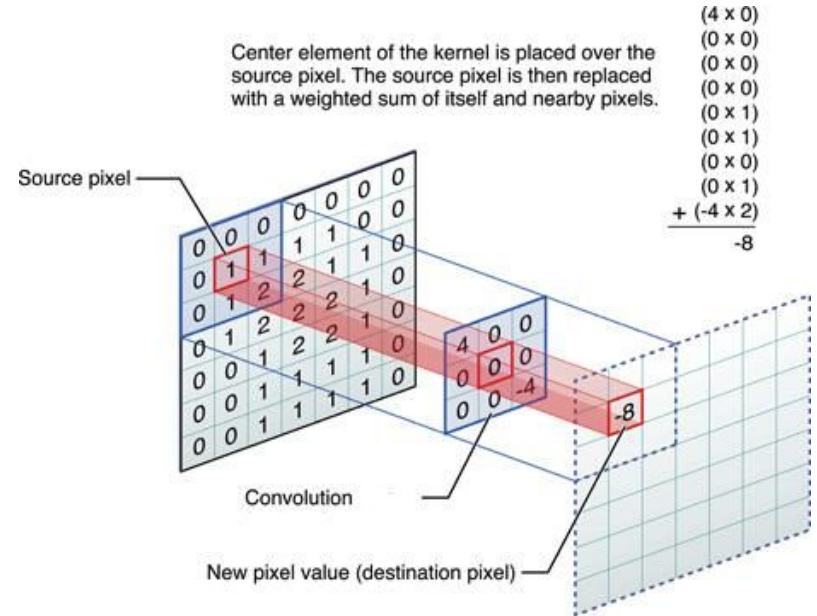
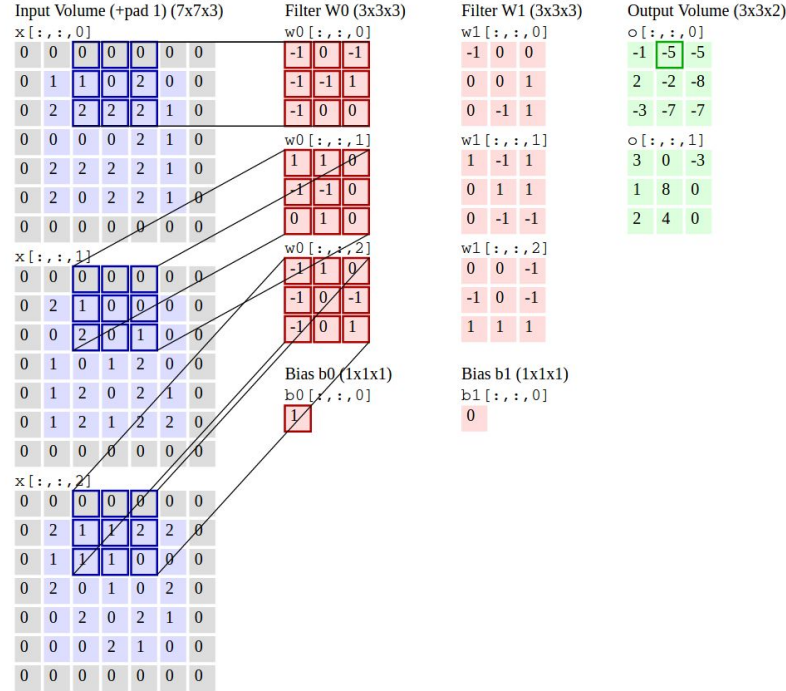
# More advanced Networks (Convolution)

We can incorporate knowledge we know to be true about the data through the network architecture

One piece of knowledge we know for images -- If you shift an image left or right it still has the same object in it

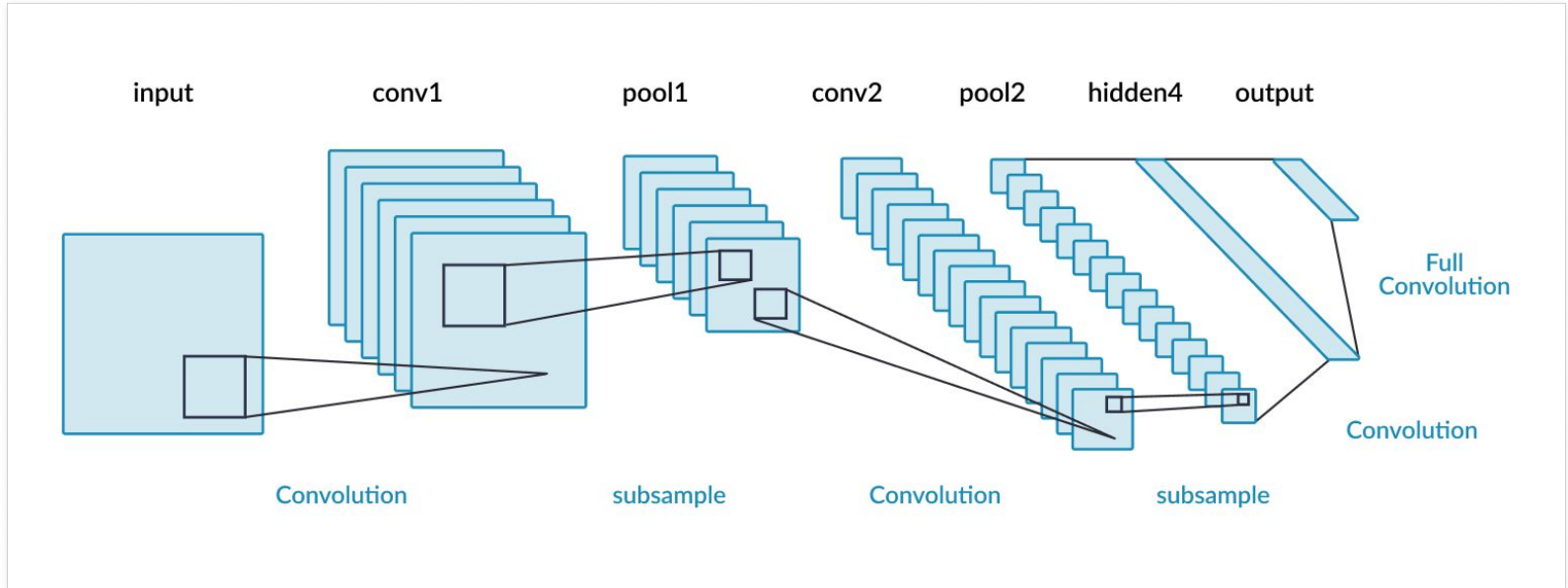
---

# Convolutional Networks (CNNs)



---

# Convolutional Networks (CNNs)



---

# More Deep Learning

- There is a lot more to deep learning
    - Other network architectures and elements than MLP or Convolutional networks
    - Other (better) variants of gradient descent for optimization
    - Other activation functions
    - Advanced methods for initialization of networks
    - Data augmentation
    - Much more
  - Here are some resources I like:
    - 3Blue1Brown visual introduction course on YouTube
    - Fast.ai deep learning for coders course
-

---

# Why I recommend fast.ai

5 lines of code gets you

- State of the art architecture
- Good optimizer and initialization
- Transfer learning
- Training loop already programmed
- Also has a full course on how to get more out of the library and as an introduction + deep dive on deep learning

```
1 from fastai.vision import *
2 path = untar_data(URLs.MNIST_SAMPLE)
3 data = ImageDataBunch.from_folder(path)
4 learn = cnn_learner(data, models.resnet50, metrics=accuracy)
5 learn.fit(5)
```

---