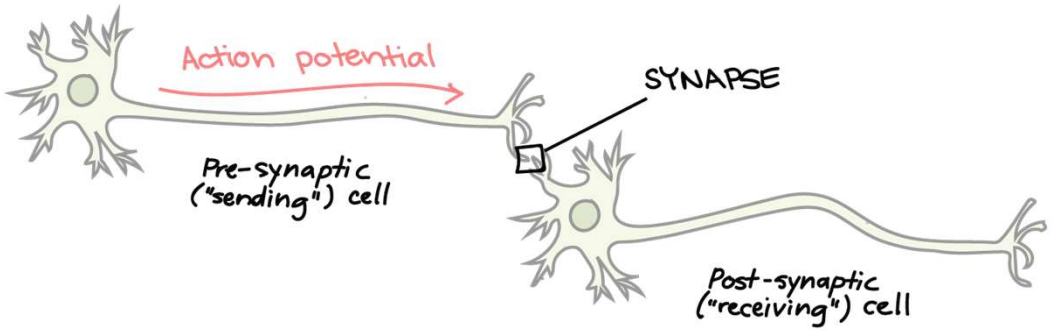
NEURAL NETWORKS I

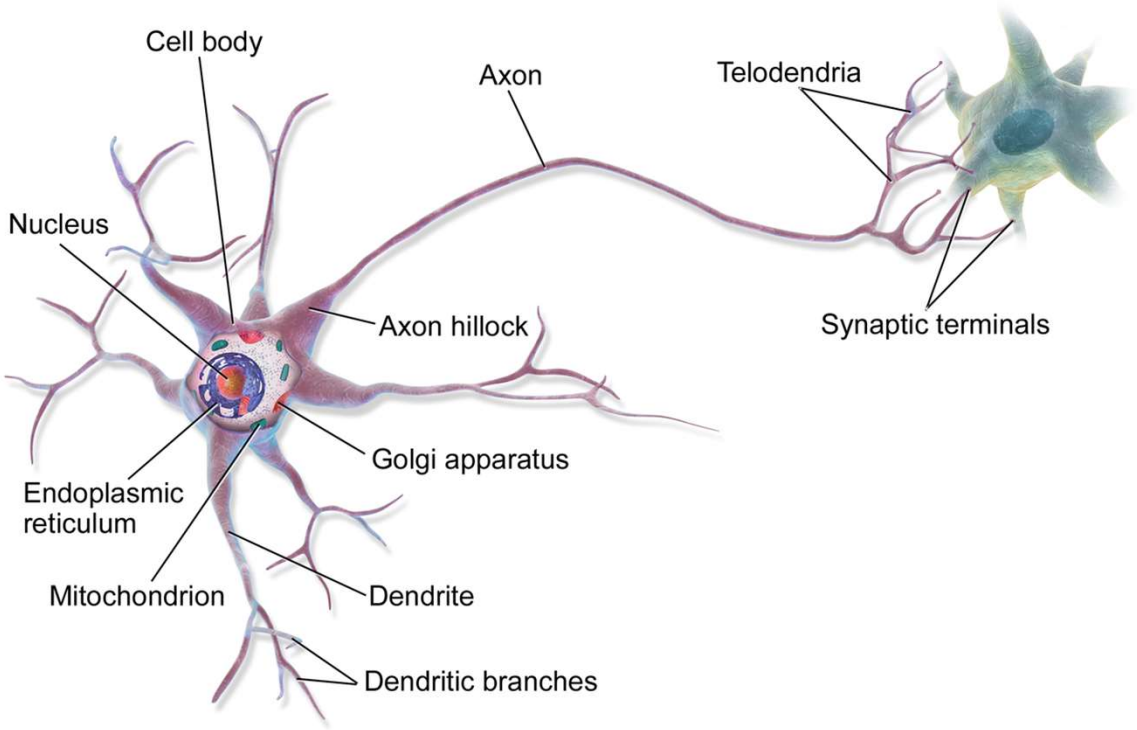
DATA/MSML 603: Principles of Machine Learning

Neural Function



- Brain function (thought) occurs as the result of the firing of **neurons**
- Neurons connect to each other through **synapses**, which propagate **action 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 (causes structural and functional change in nervous system)
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain!

Biology of A Neuron



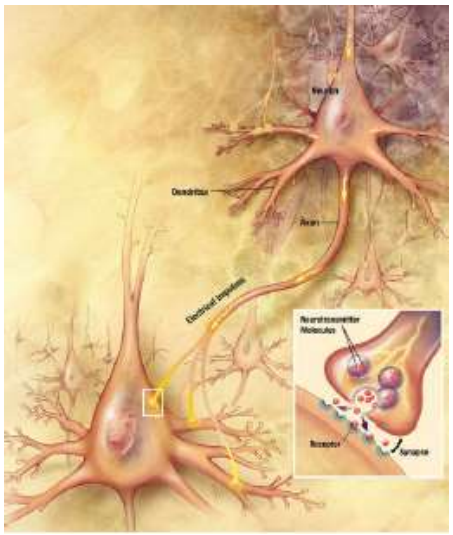
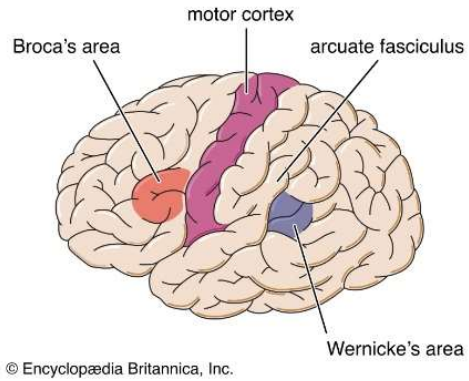


Image source: Bruce Blausen

Brain Structure



- Different areas of the brain have different functions
  - Some areas seem to have the same function in all humans (e.g., Broca's region for motor speech); the overall layout is generally consistent
  - Some areas are more plastic, and vary in their function; also, the lower-level structure and function vary greatly
- We do not know how different functions are "assigned" or acquired
  - Partly the result of the physical layout / connection to inputs (sensors) and outputs (effectors)
  - Partly the result of experience (learning)

Comparison of Computing Power

INFORMATION CIRCA 2012	Computer	Human Brain
Computation Units	10-core Xeon: 10^9 Gates	10^{11} Neurons
Storage Units	10^9 bits RAM, 10^{12} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-9} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec

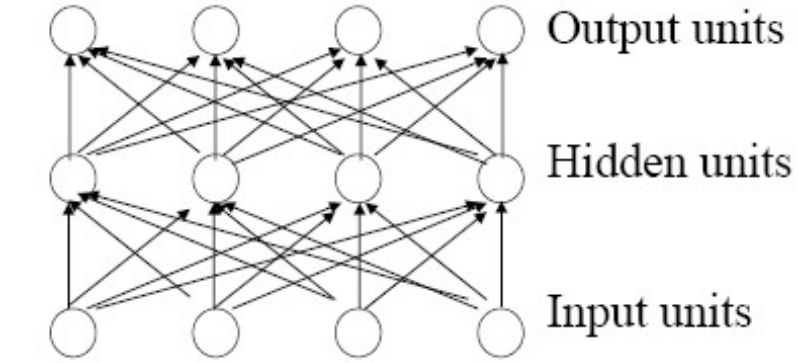
- (Artificial) neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

(Artificial) 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 (2/2)

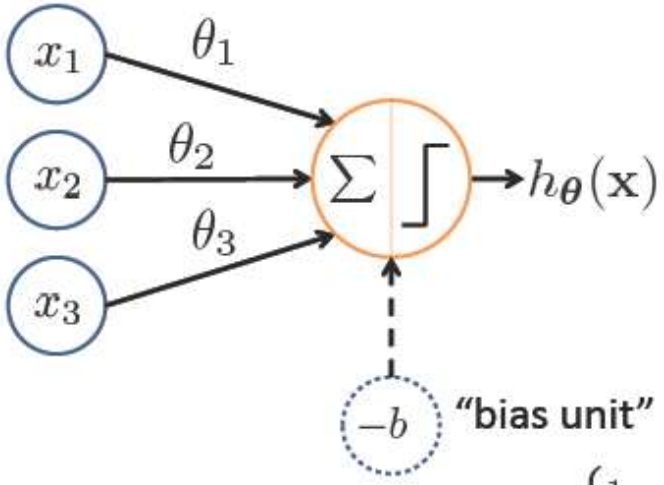
- Computers are way faster than neurons…
- But, there are a lot more neurons than we can reasonably model in modern digital computers, and they all fire in parallel
- Neural networks are designed to be massively parallel
- The brain is effectively a billion times faster

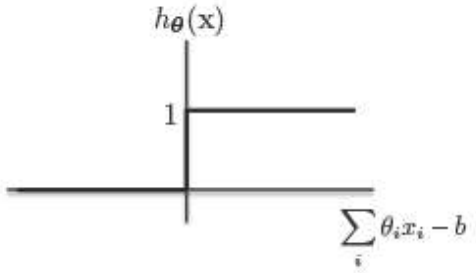


Layered feed-forward network

Neuron Model: Threshold Unit

input $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$ weights

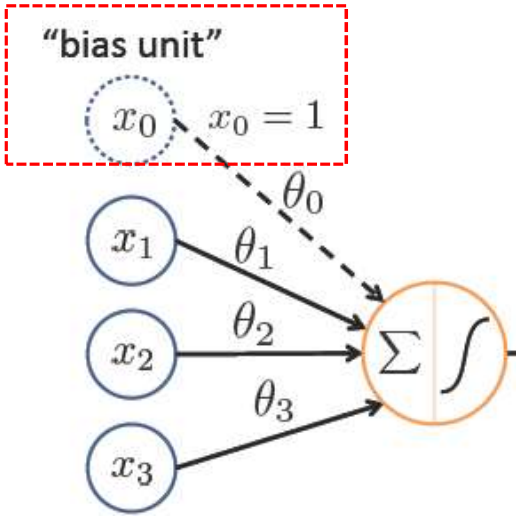




Activate if weighted sum of inputs is larger than "threshold"

Step activation function: $h_{\boldsymbol{\theta}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_i \theta_i x_i \geq b \\ 0 & \text{otherwise} \end{cases}$

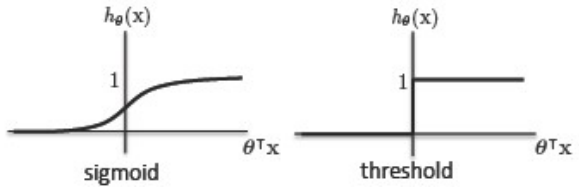
Neuron Model: Logistic Unit



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

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

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

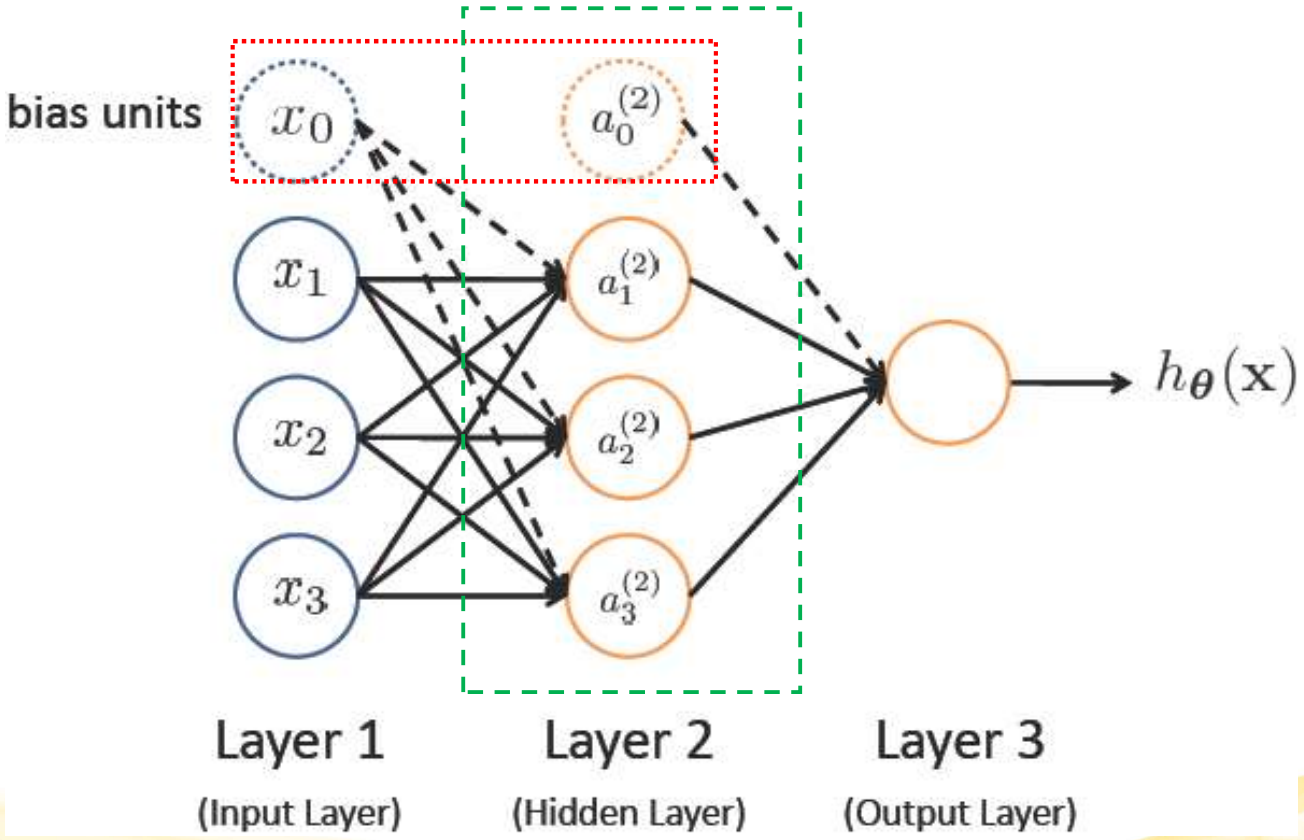


Neuron Model: Other Activation Functions

- Tanh or hyperbolic tangent activation function
- Rectified Linear Unit (ReLU) activation function
- Leaky ReLU

- We will come back to these later…

Neural Network

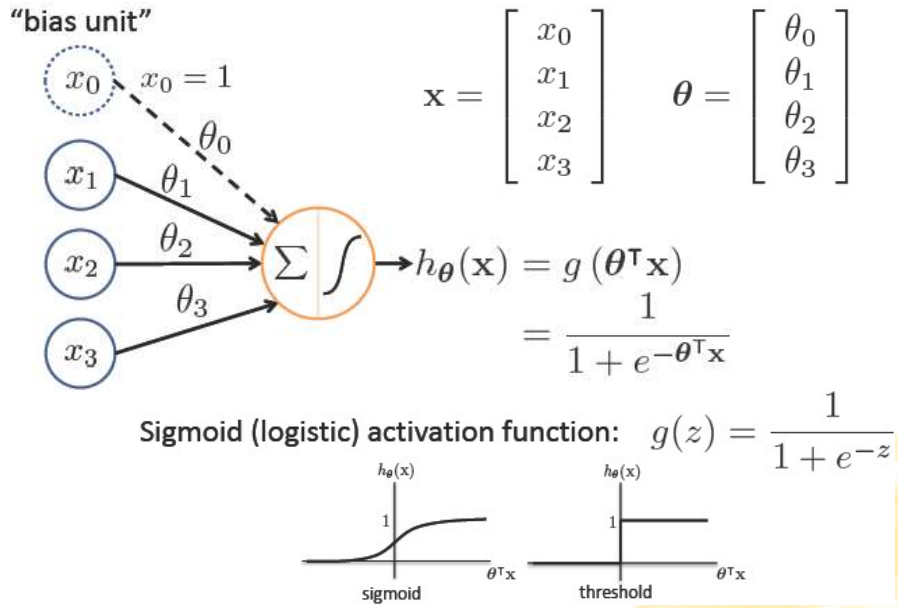


Feed-forward Process (1/2)

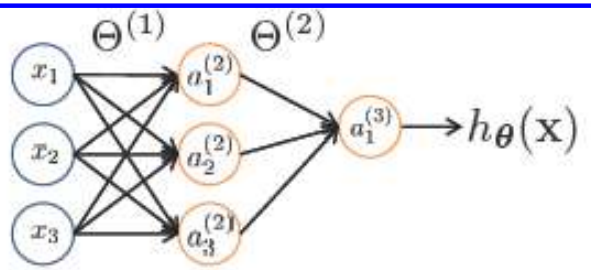
- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node

Feed-forward Process (2/2)

- The **activation function** transforms this input function (value) into a final value
  - Typically this is a **nonlinear** function, often a sigmoid function corresponding to the threshold of that node



Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

Hidden units

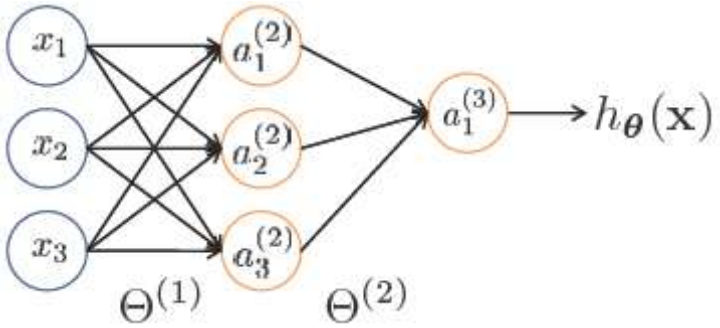
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$ [includes bias].

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$
$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$

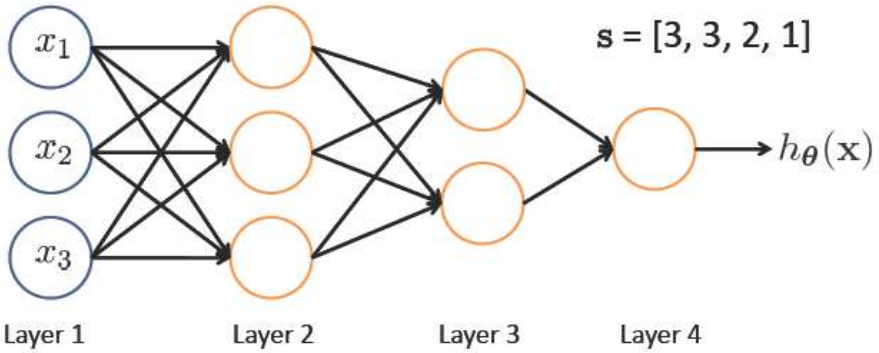


Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$
$$\text{Add } a_0^{(2)} = 1$$
$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

bias

Other Network Architectures



- L denotes the number of layers
- $\mathrm{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer
  - Not counting bias units
  - Typically, $s_0 = d$ (# input features) and $s_{L-1} = K$ (# classes)

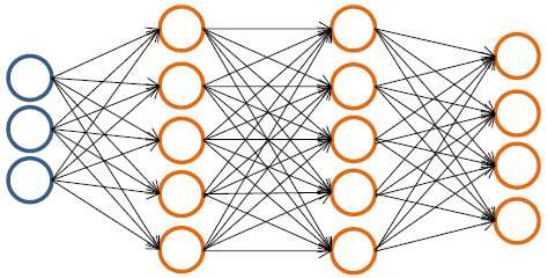
Multiple Output Units: One-vs-Rest


Pedestrian


Car


Motorcycle


Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

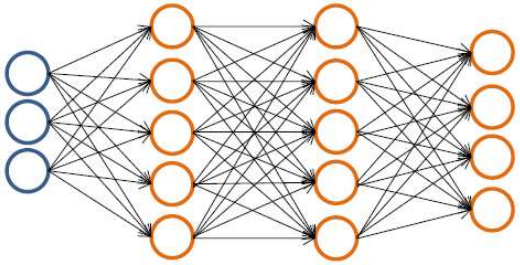
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

Multiple Output Units: One-vs-Rest



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

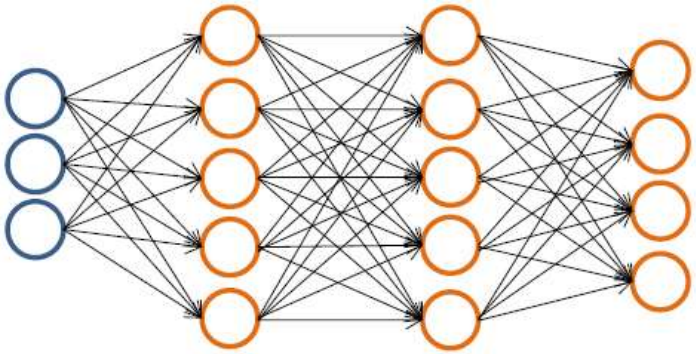
We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when pedestrian | when car | when motorcycle | when truck

- Given $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$
- Must convert labels to 1-of-K representation
  - e.g., $\mathbf{y}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $\mathbf{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

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

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Multi-class classification (K classes)

$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units $(s_{L-1} = K)$

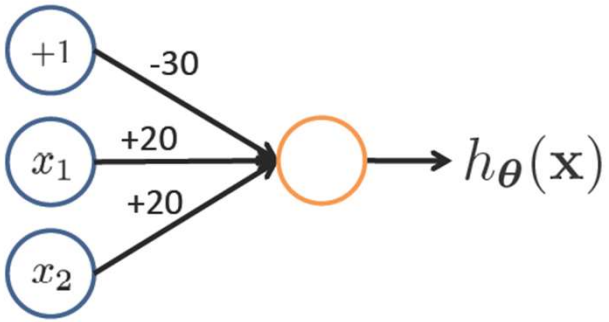
Understanding Representations

Representing Boolean Functions

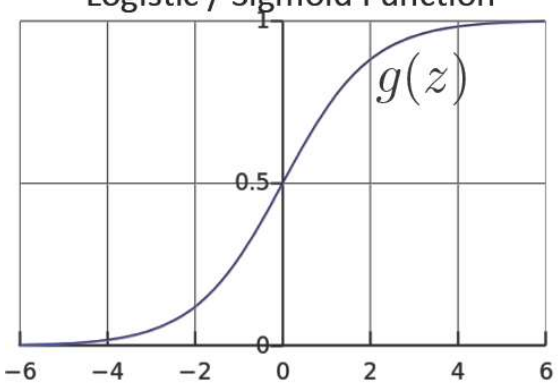
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



Logistic / Sigmoid Function



x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

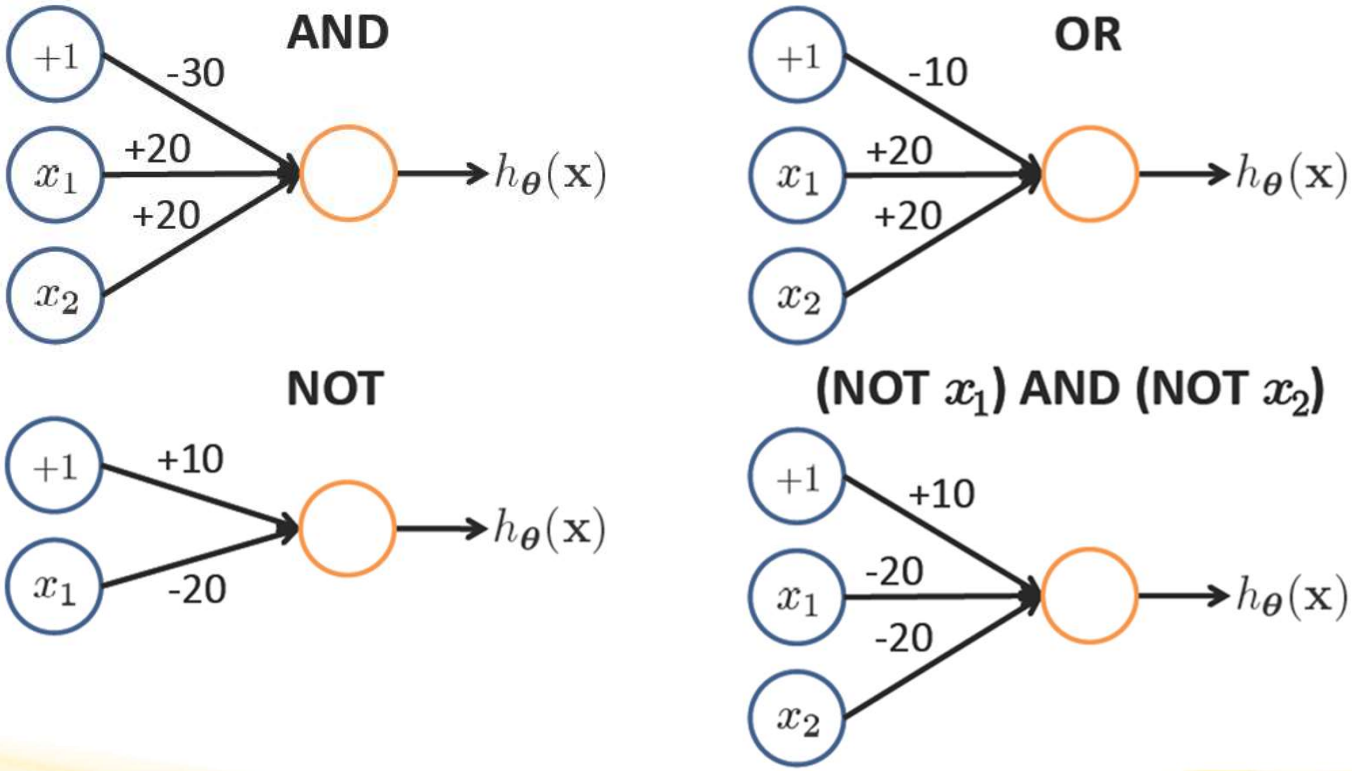
$$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

$$= \frac{1}{1 + e^{-(-30+20x_1+20x_2)}}$$

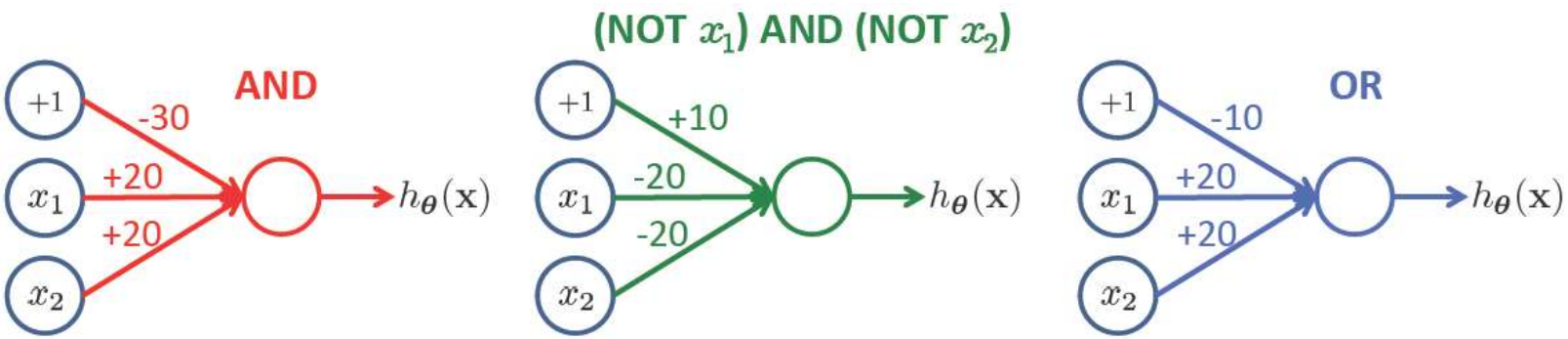
$$g(-30) = 9.36 \cdot 10^{-14}, \; g(-10) = 4.54 \cdot 10^{-5}$$

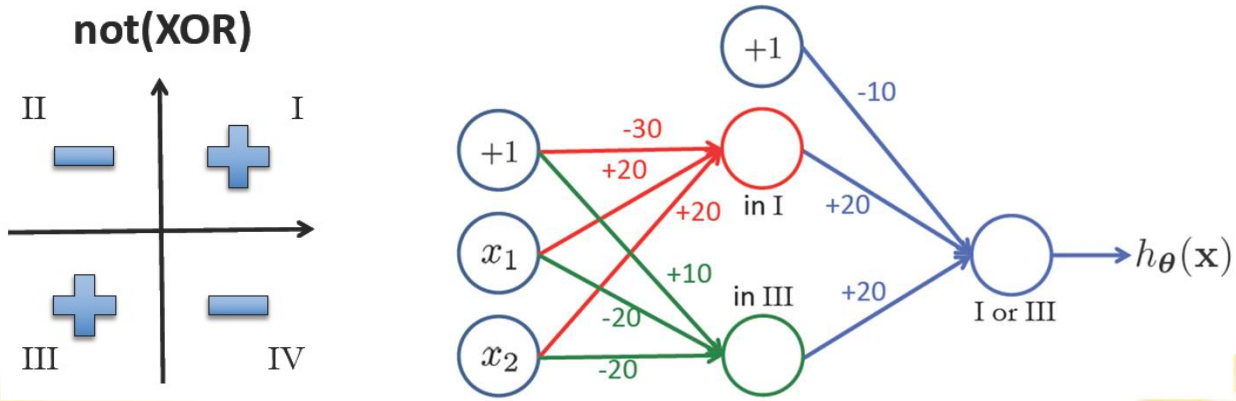
$$g(-30) = 9.36 \cdot 10^{-14}, \; g(-10) = 4.54 \cdot 10^{-5}$$

Representing Boolean Functions

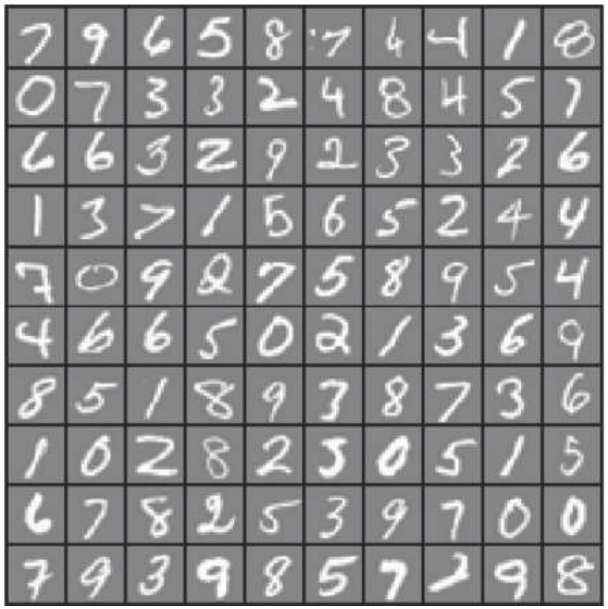


Combining Representations

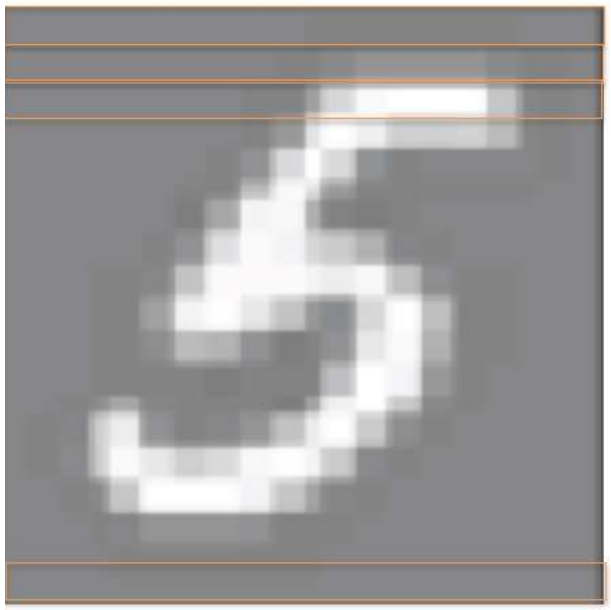




Layering Representations



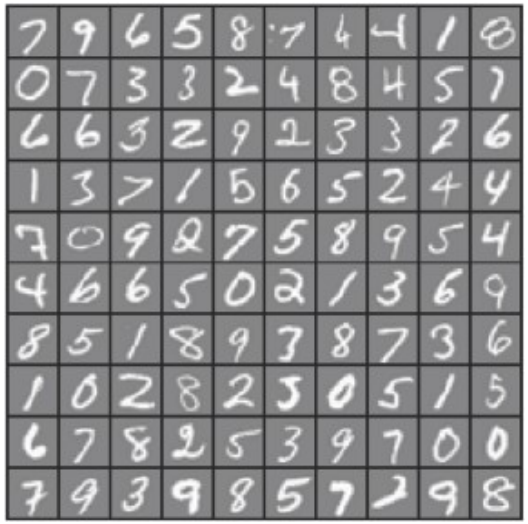
20x20 pixel images
d = 400, 10 classes

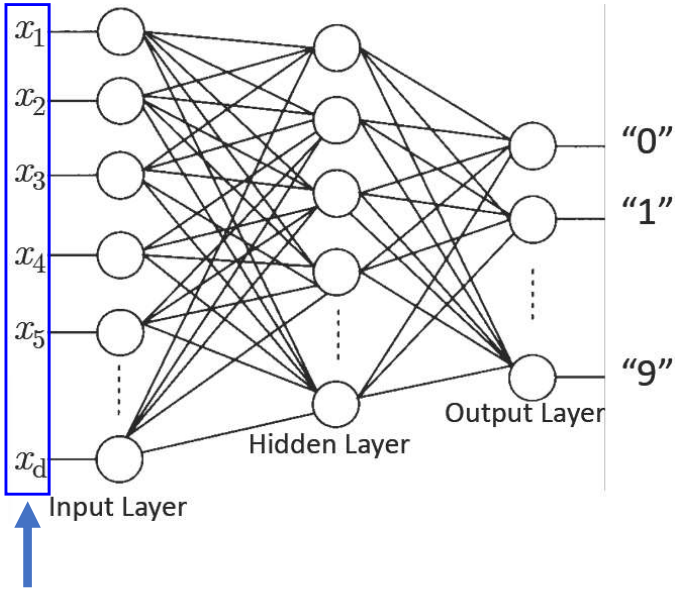


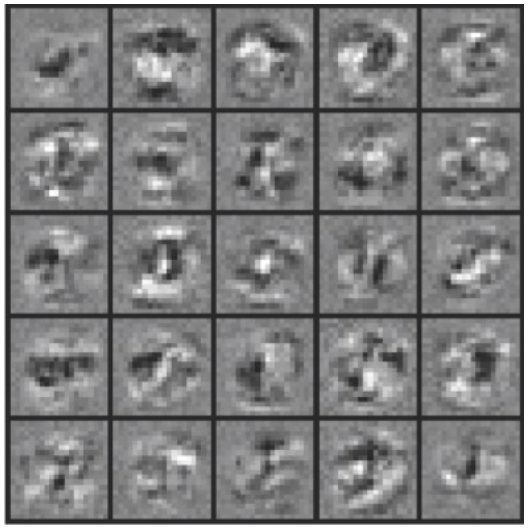
$$x_1 \dots x_{20}$$
$$x_{21} \dots x_{40}$$
$$x_{41} \dots x_{60}$$
$$\vdots$$
$$x_{381} \dots x_{400}$$

Vectorization

Layering Representations (2/2)







Visualization of Hidden Layer

Vectorization

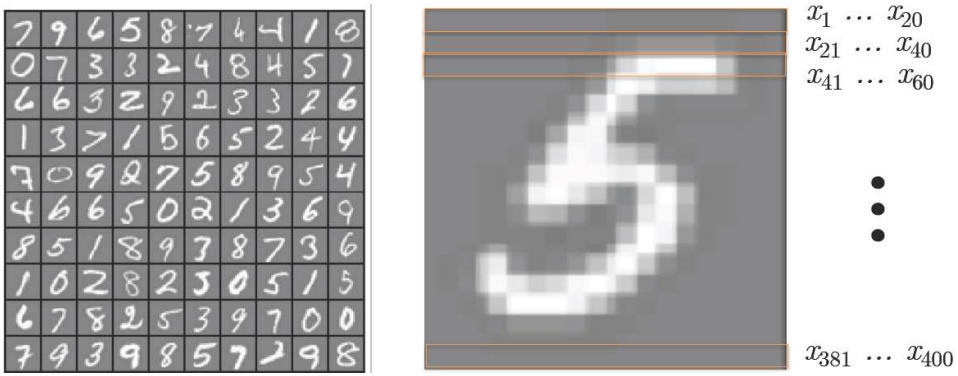
Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- Originally introduced for problems where input data has a grid-like structure
  - Image – pixels reside on a two-dimensional grid
  - Audio waveform – one-dimensional grid
  - Volumetric data, e.g., computer tomography (CT) scans or video data – three-dimensional grid

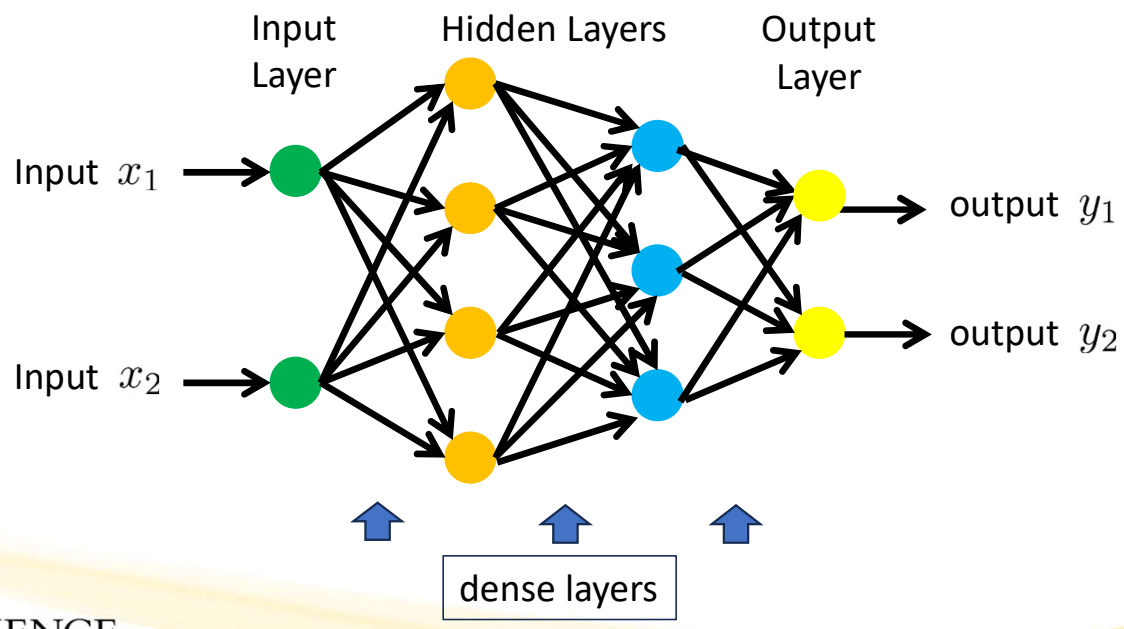
Convolutional Neural Networks (CNNs)

- Putting all input variables representing the image pixels in a long vector causes loss of a lot of structure present in the image
  - For instance, two pixels close to each other typically have more in common than two pixels farther apart, which is destroyed by vectorization



Convolutional Layer

- CNNs tries to preserve this information by representing both input variables and hidden layers as matrices



Dense layer: each input variable connected to all hidden units in subsequent layer, and each such connection has a unique parameter associated to it

- Provide too much flexibility for images

Convolutional layer exploits the structure present in images to find a more efficiently parameterized model

Convolutional Layer

- Convolutional layer leverages two key concepts

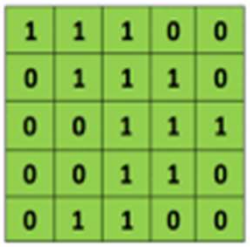
1. Sparse interactions: Most of the parameters in a corresponding dense layer are forced to be zero
   - A hidden unit in a convolutional layer only depends on the pixels in a small region of the image and not on all pixels
2. Parameter sharing: In a dense layer, each link between an input variable and a hidden unit has its own unique parameter
   - In a convolutional layer, the set of parameters (filters), for different hidden units are all the same
   - Instead of learning separate sets of parameters, learn only one set of parameters

Convolutional Layer

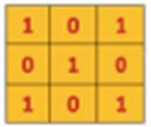
- Sparse interactions and parameter sharing in a convolutional layer makes CNNs relatively invariant to translations of objects in images
  - If the parameters in a filter are sensitive to a certain detail (e.g., a corner, an edge), a hidden unit will react to this detail regardless of where in the image that detail is present
- A convolutional layer uses significantly fewer parameters compared to a dense layer

Convolution

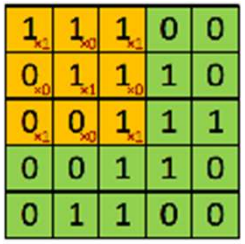
- Loosely mimics neuronal responses
- To compute:
  - Slide small kernel (also called a filter) over input
  - Element-wise multiplication and sum



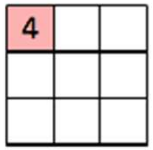
Input matrix



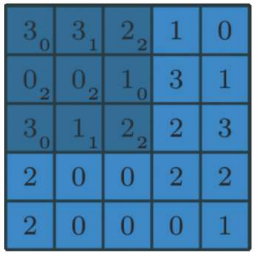
2D Convolutional
3x3 filter



Image

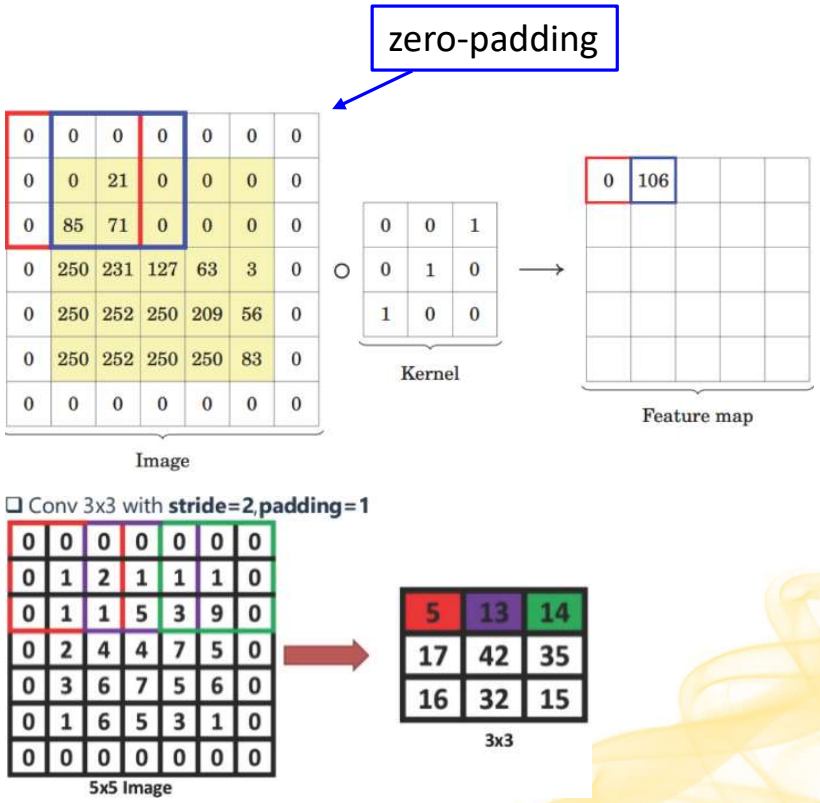


Convolved
Feature



Convolution Summary

- Layers structured as matrices, not vectors
- Convolution of matrices by small (e.g. 3 x 3) matrices called **kernels**
- The kernels are the **learned weights**
- Depending on **padding** and **stride**, image may decrease slightly in size
- Intuitively, convolutional layers learn how to relate neighboring pixels
- Image layers in a CNN typically have many **features/hidden units**
  - Organized as **"channels"**



Motivation: Conv. for Edge Detection

-1	-1	-1
2	2	2
-1	-1	-1

Horizontal lines

-1	2	-1
-1	2	-1
-1	2	-1

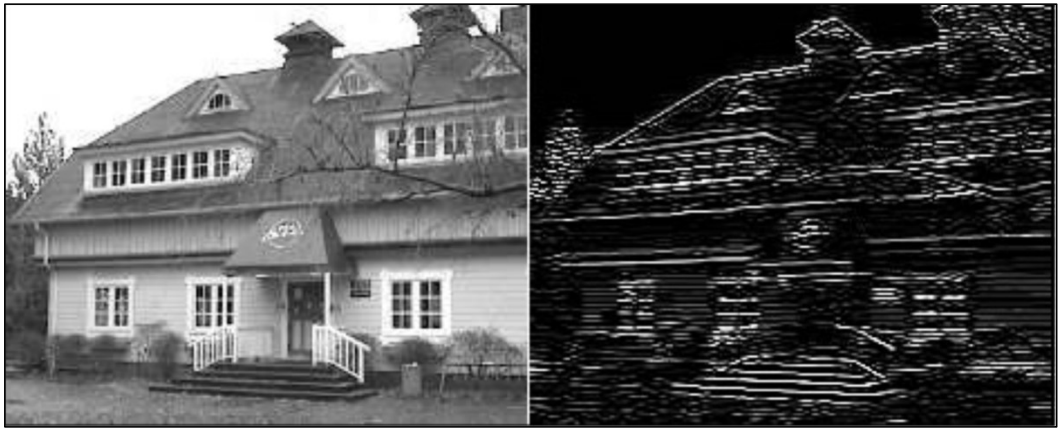
Vertical lines

-1	-1	2
-1	2	-1
2	-1	-1

45 degree lines

2	-1	-1
-1	2	-1
-1	-1	2

135 degree lines

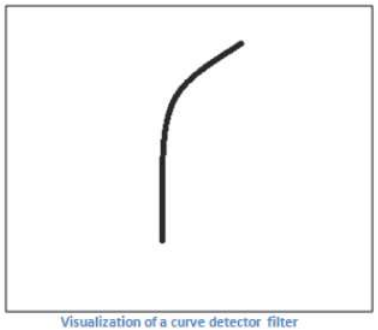


Horizontal lines

Cross-correlation in Two Dimensions

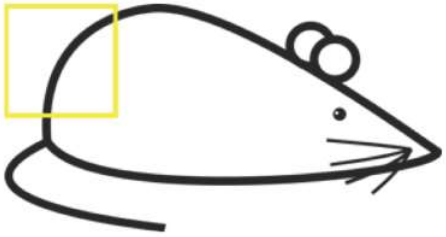
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

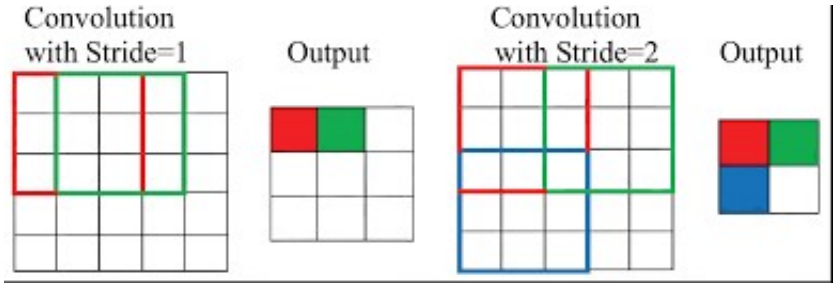
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Convolutional Layer with Strides

- With zero-padding, in a convolutional layer we saw earlier, the number of hidden units is equal to the number of pixels inn an image
- We often want to reduce the number of hidden units and store only the most important information from the previous layers
- Stride controls how many pixels the filter/kernel shifts over at each step



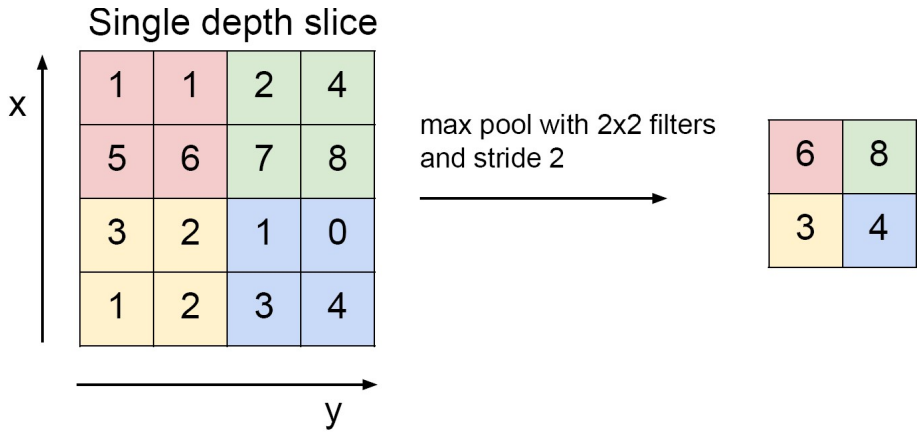
[Source: https://svitla.com/blog/math-at-the-heart-of-cnn/]

Pooling Layer

- Another way of summarizing information in previous layers
- Additional layer after a convolutional layer
  - Similar to convolutional layer, depends only on a small region of pixels
  - Does not require extra trainable parameters
  - Also use strides to condense the information
- Most common pooling methods
  - Max pooling (MaxPool)
  - Average pooling

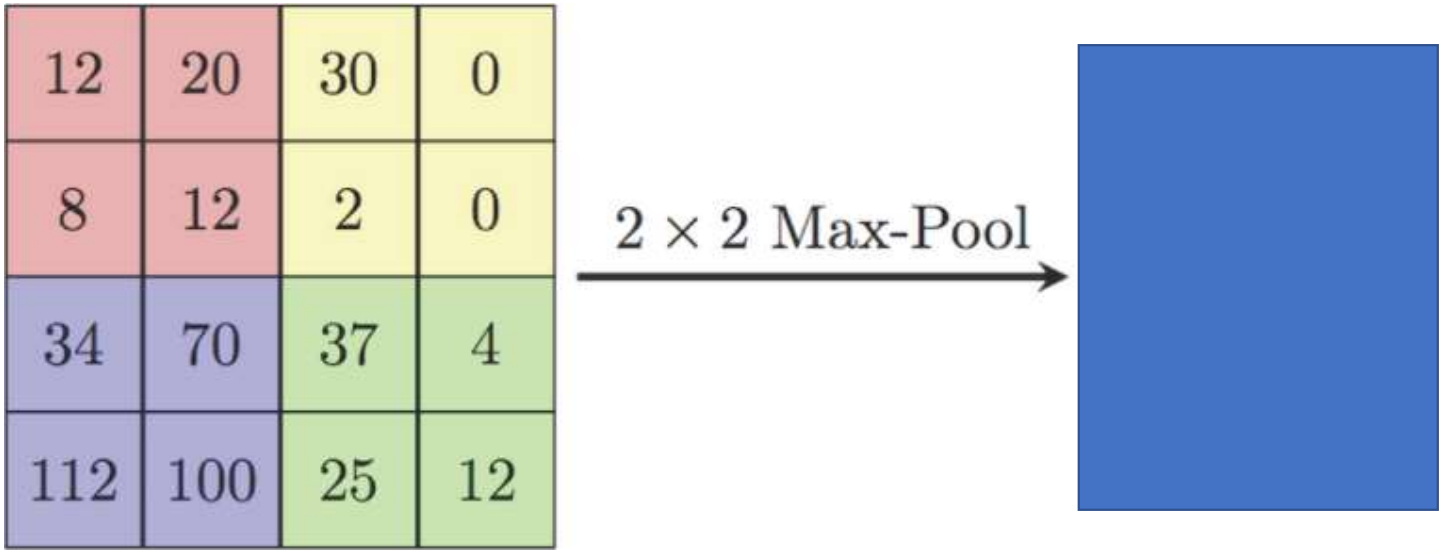
Max Pool (1/2)

- Intuitively, MaxPool is used because nearby pixels will give very similar output
- Over the course of a network, pooling shifts layers from spatial dimensions to feature



Max Pool (2/2)

- Practice

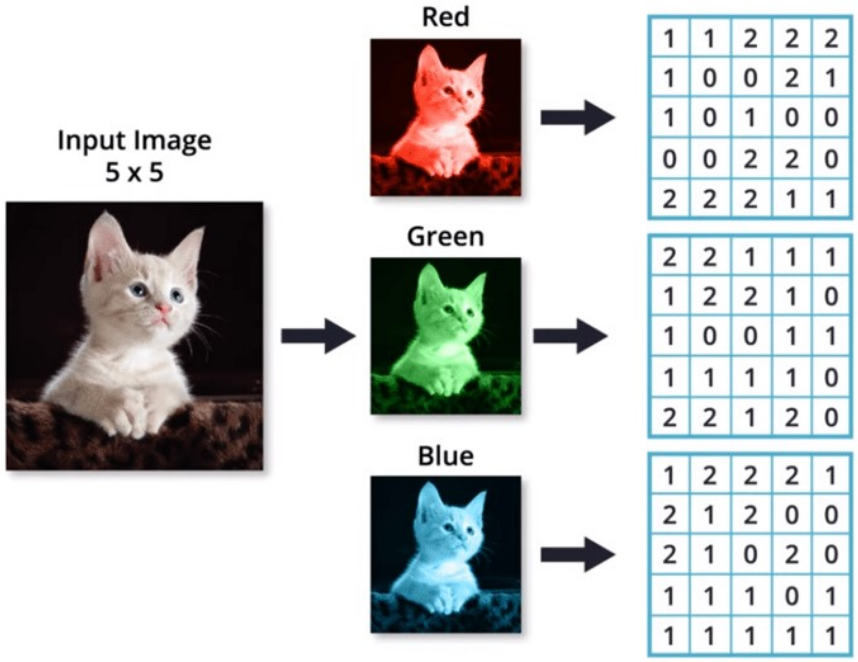


Channels

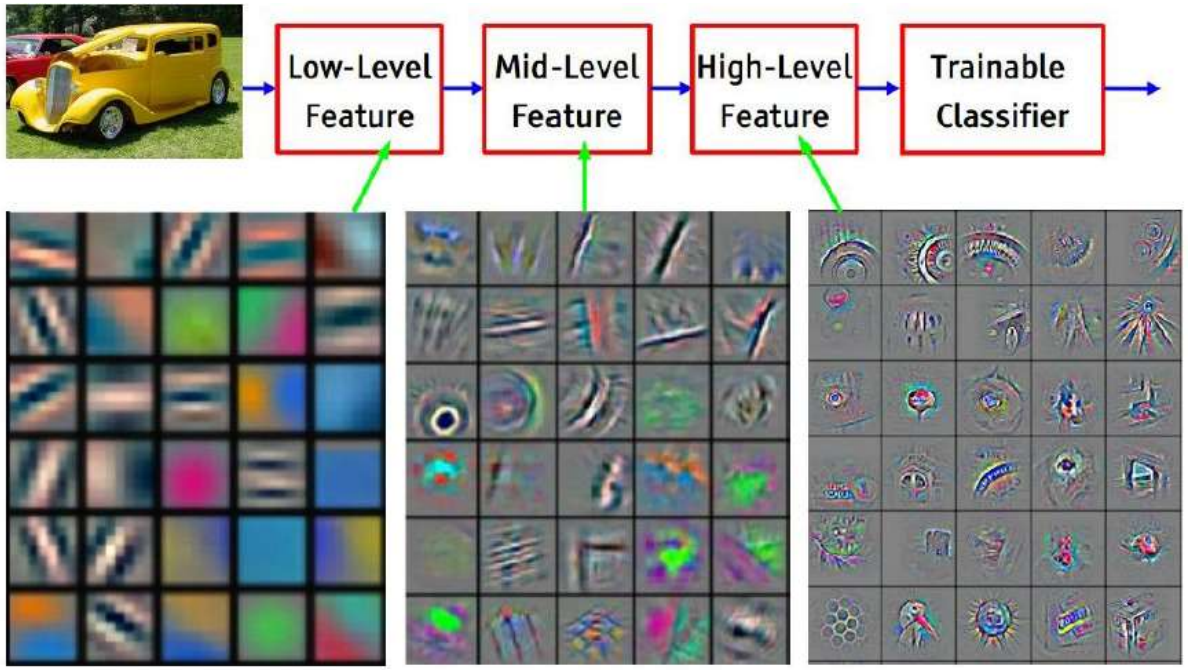
- Using a single filter/kernel in each convolutional layer often not sufficient to encode all interesting properties of images
- Add multiple filters, each with own set of parameters to be trained
  - Each filter produces its own set of hidden units, called a “channel”, using the same convolution operation
  - Each layer of hidden units in a CNN organized into a tensor with dimension (rows x columns x channels)
- When stacking convolutional layers, each filter depends on all the channels in the previous layer
  - Each filter is a tensor of dimension (filter rows x filter columns x input channels)

Color Images

- Color (RGB) image represented as a tensor (h x w x 3)
- 3D filters
  - Depth of each filter chosen to match the number of color channels
  - Each channel convolved with own 2D filter
  - Add the values from the convolution with 2D filters
- Each 3D filter produces a separate 2D matrix with hidden units

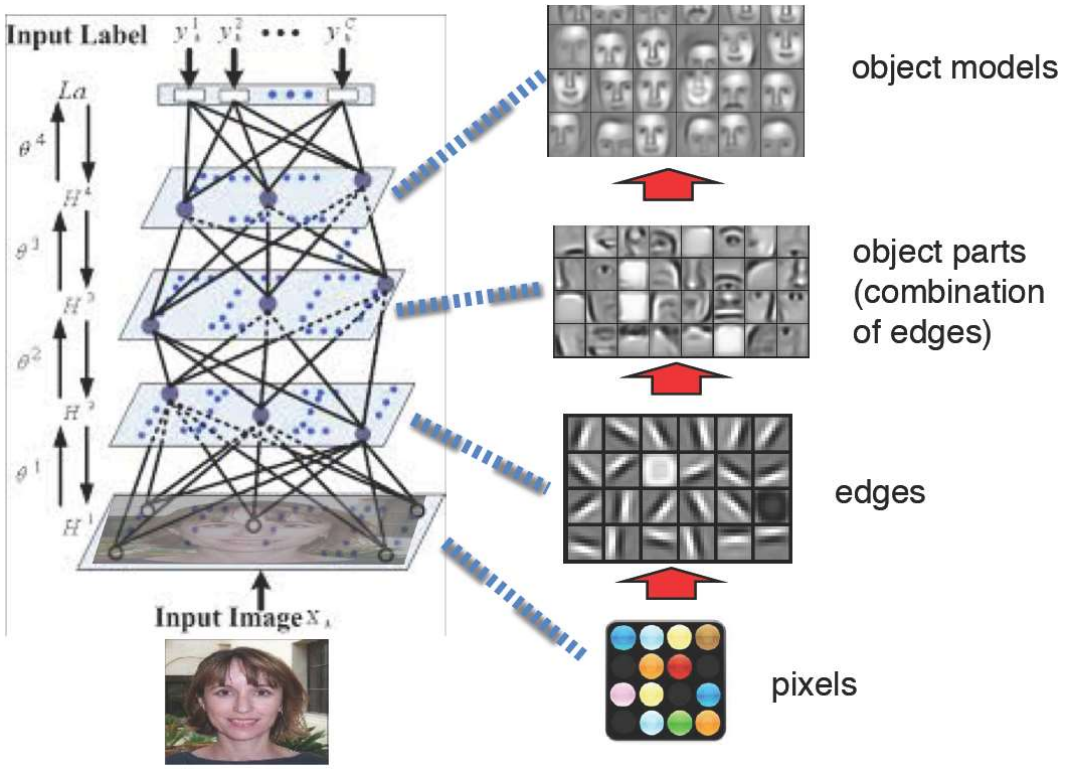


What CNNs Learn

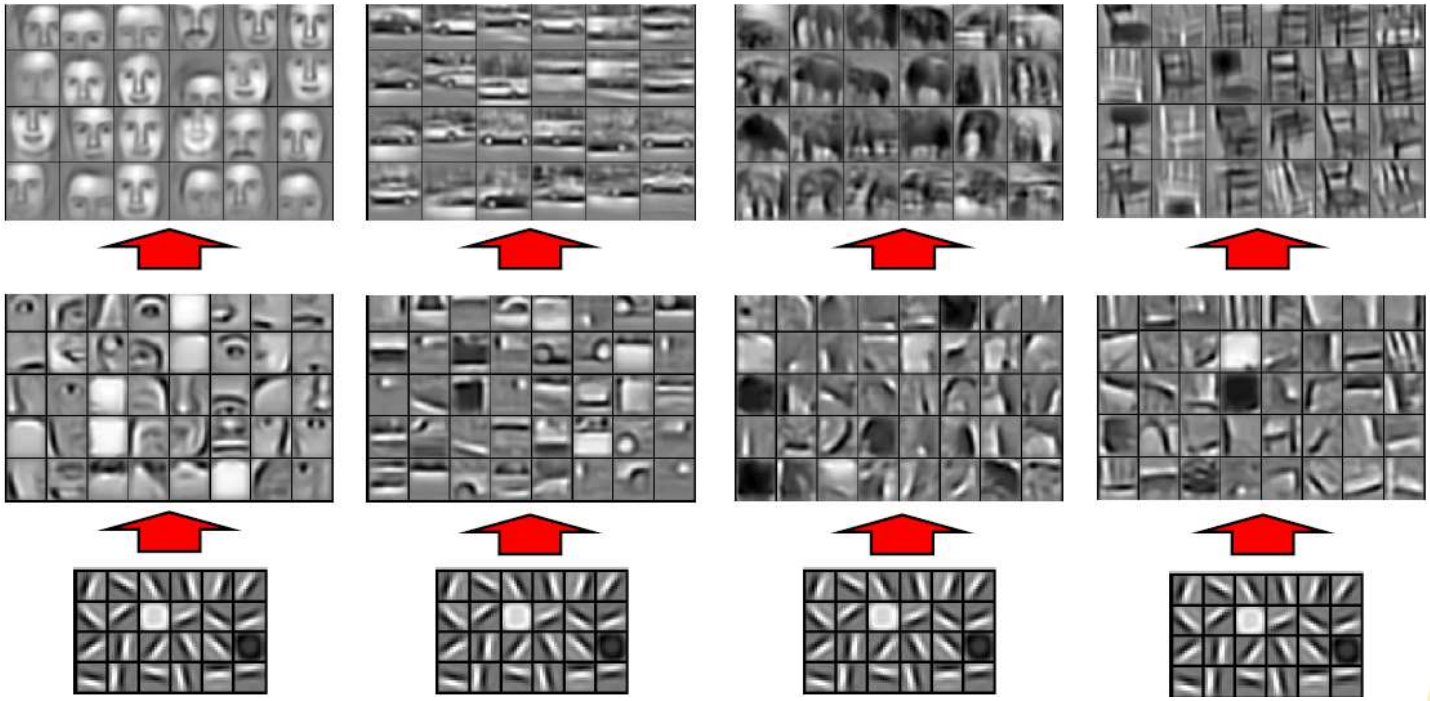


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

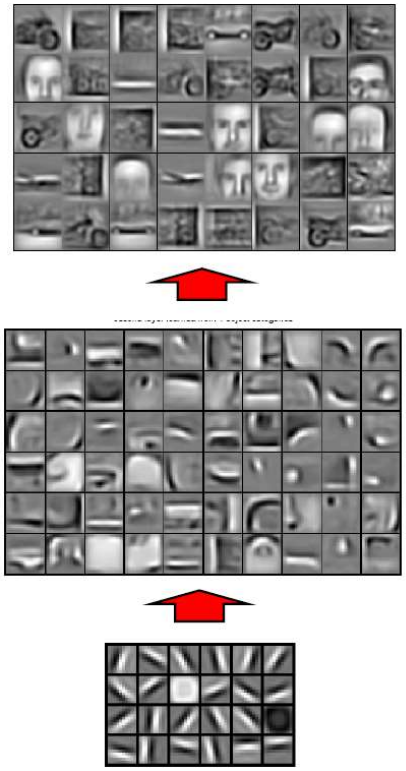
Deep Belief Net on Face Images



Learning of Object Parts



Training on Multiple Objects



Trained on 4 classes (cars, faces, motorbikes, airplanes).

Second layer: Shared-features and object-specific features.

Third layer: More specific features.