DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

November 1 2018

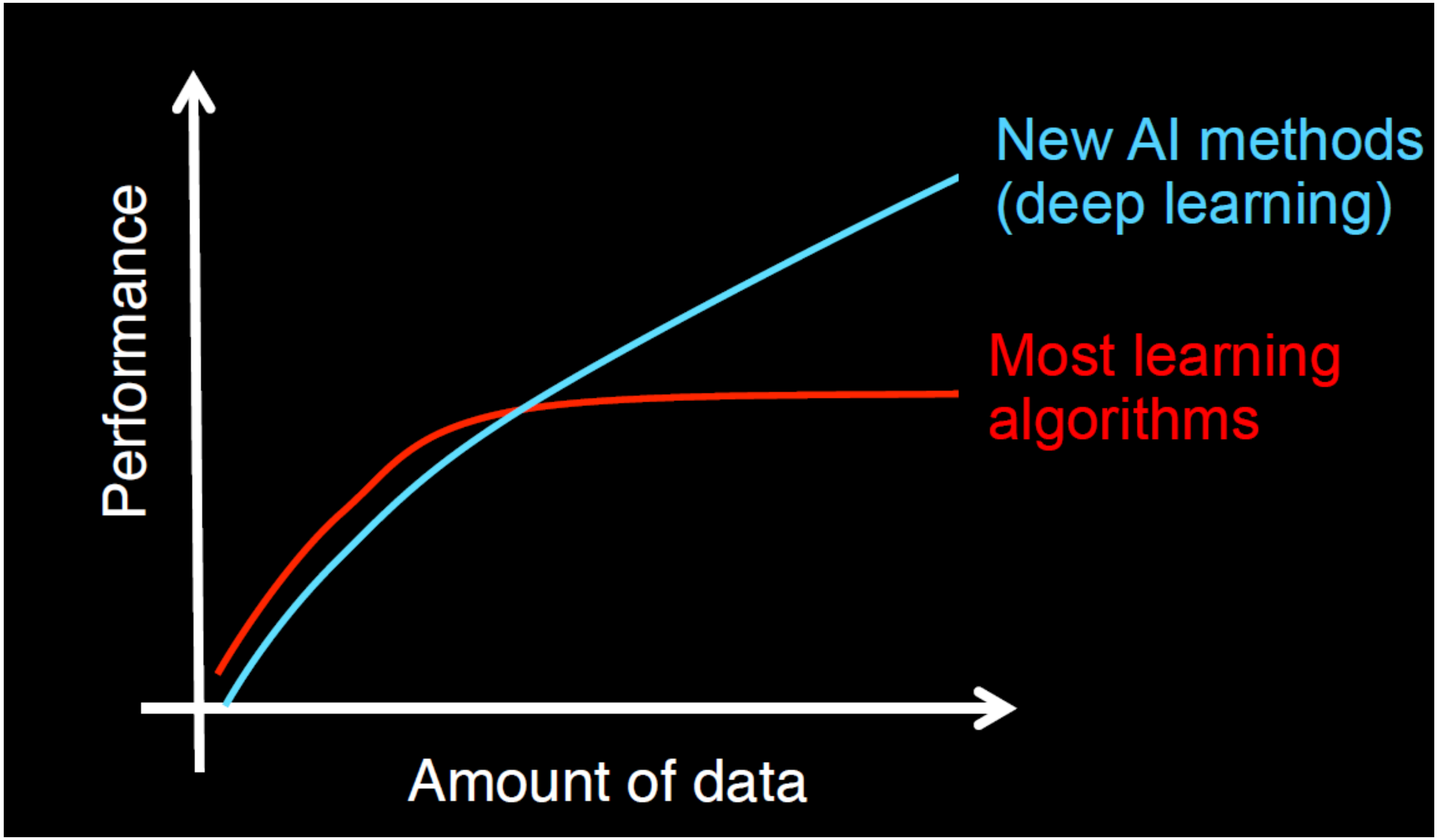
Review

- Deep Learning has the ability to learn hierarchy of features
  - Performs better with more training data
  - End-to-end learning
- Feed-Forward architectures
  - Neurons from one layer
  - At each layer linear operation followed by non-linear activation function
  - Activation functions: sigmoid, ReLU, tanh (for regression)

Outline

- Feed-Forward architectures
  - Non-linear activations
  - Multi-Layer Perceptron
  - Multi-class classification (softmax unit)
  - Representing Boolean functions
- Convolutional Neural Networks
  - Convolution layer
  - Max pooling layer

Performance of Deep Learning

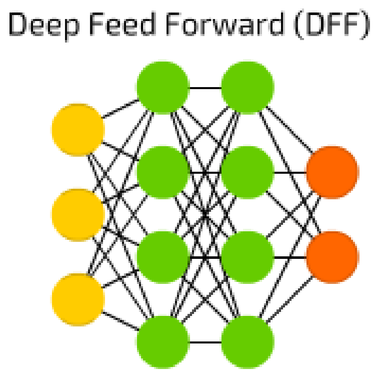


Deep Learning Applications

Neural Network Architectures

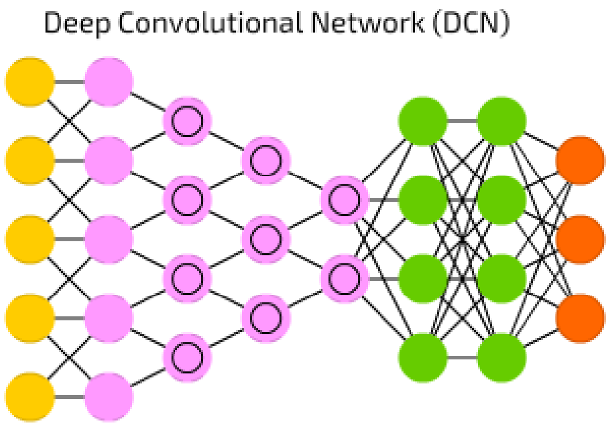
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



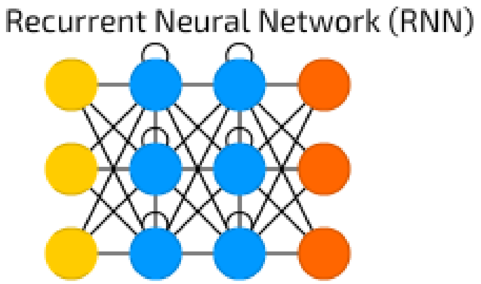
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

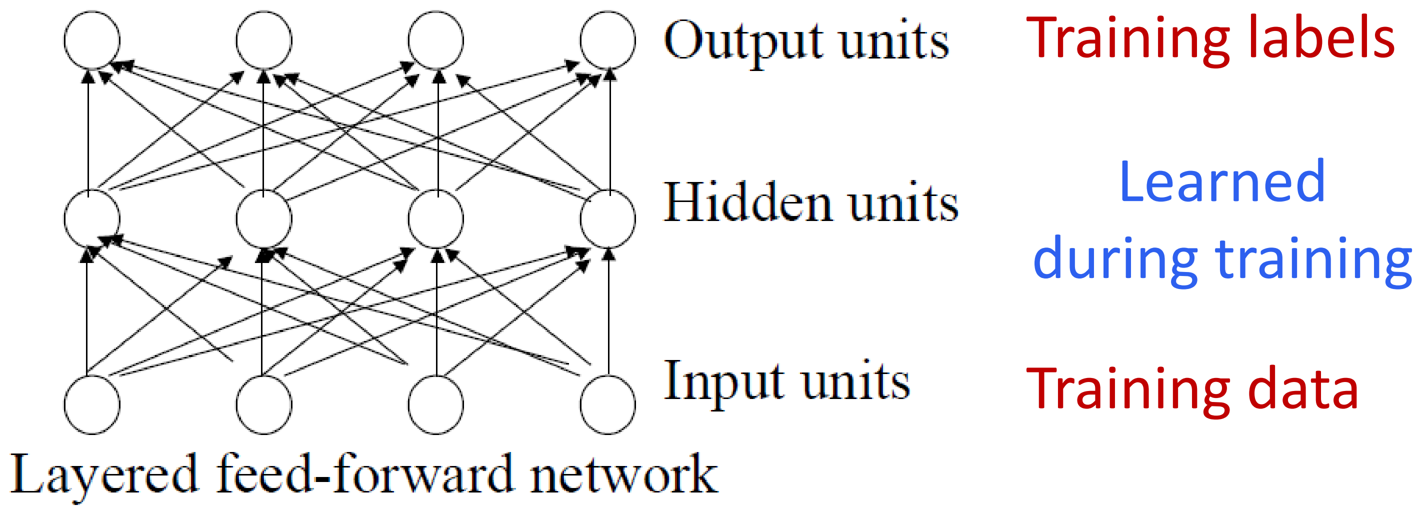


Recurrent Networks

- Keep hidden state
- Have cycles in computational graph

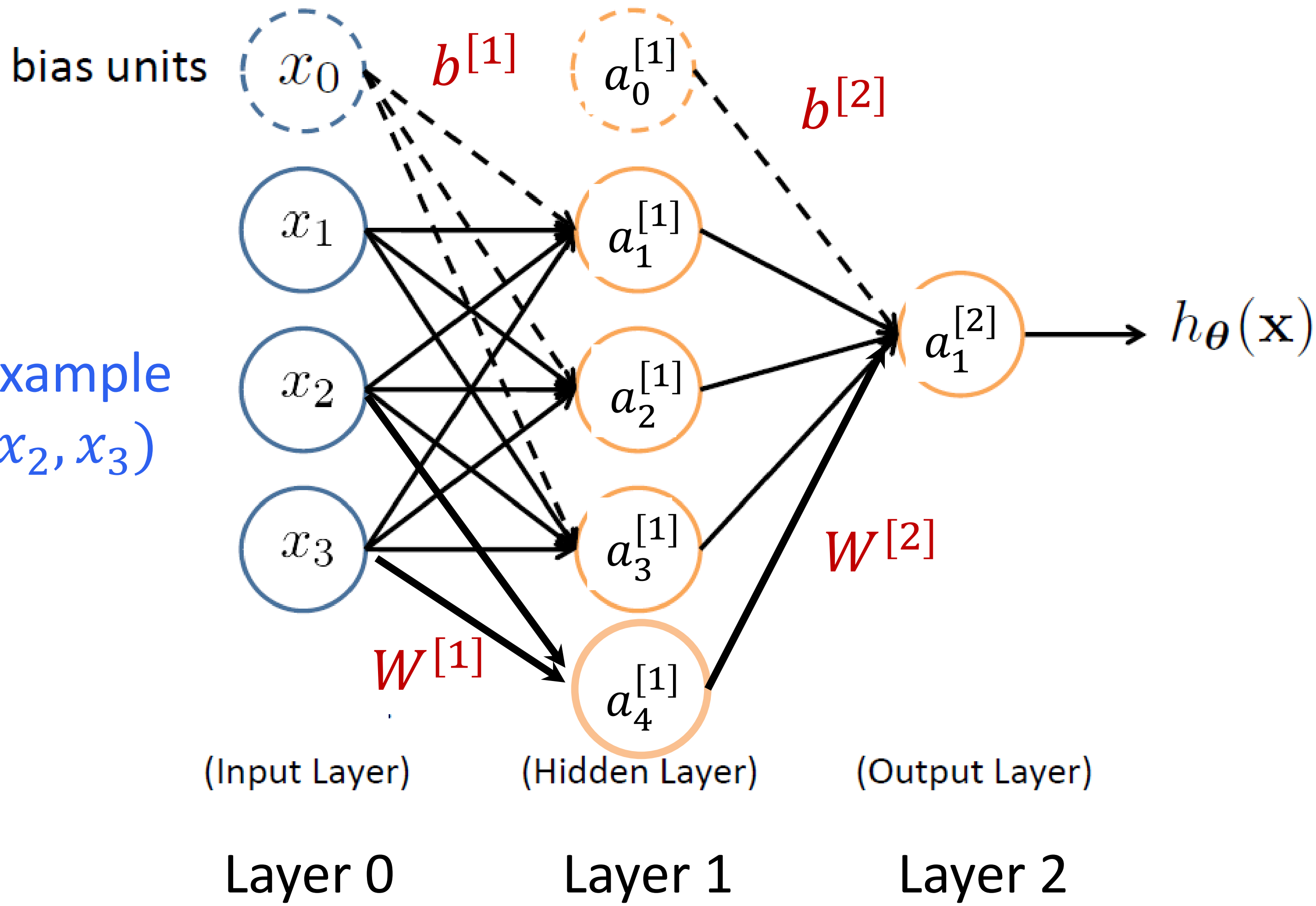


Feed-Forward Neural Networks



- 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**

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

$$z_1^{[1]} = W_1^{[1]T} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g(z_1^{[1]})$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots$$
$$z_4^{[1]} = W_4^{[1]T} x + b_4^{[1]} \quad \text{and} \quad a_4^{[1]} = g(z_4^{[1]})$$

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

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

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

Vectorization

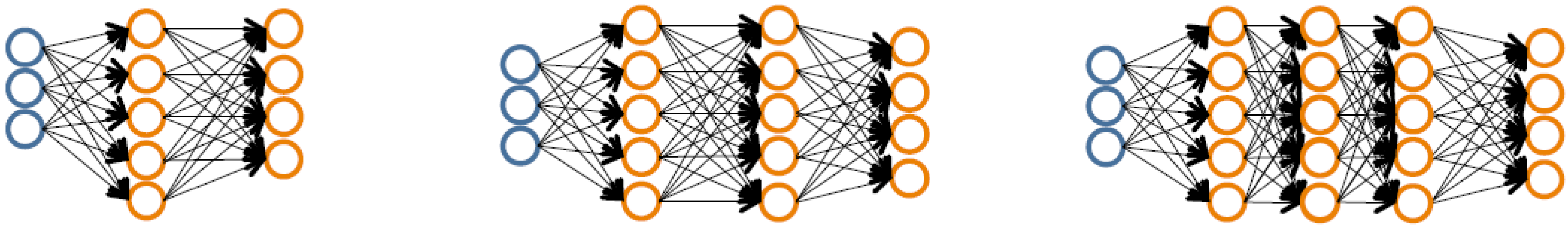
Output layer

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

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

Training Neural Networks

Pick a network architecture (connectivity pattern between nodes)



- # input units = # of features in dataset
- # output units = # classes

Reasonable default: 1 hidden layer

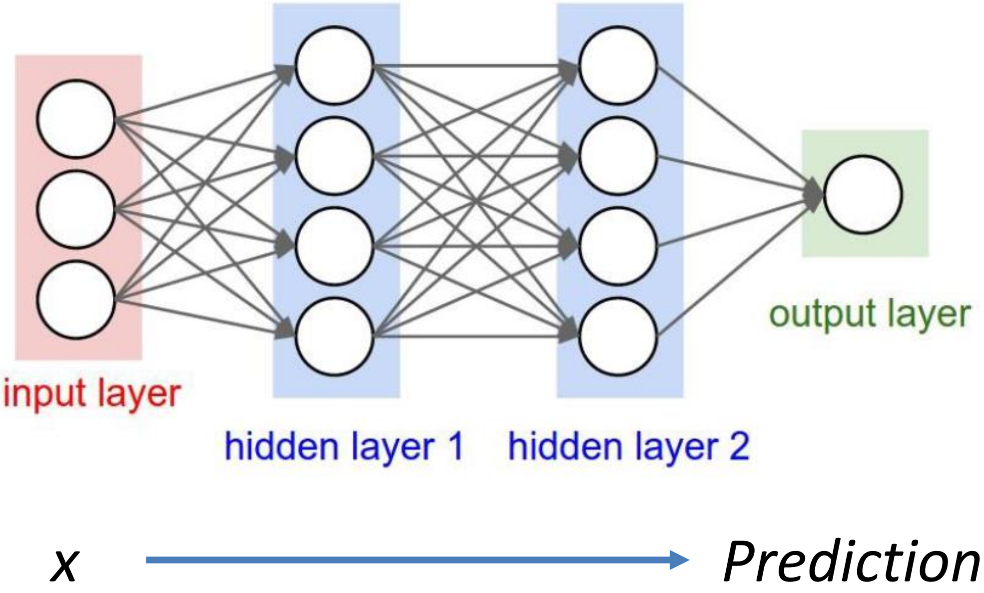
- or if >1 hidden layer, have same # hidden units in every layer (usually the more the better)

Training Neural Networks

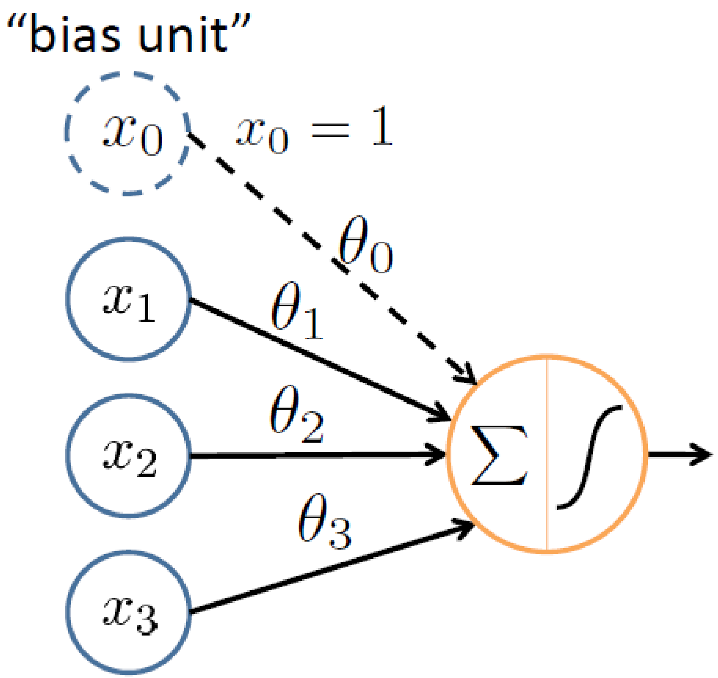
- Input training dataset D
  - Number of features: d
  - Labels from K classes
- First layer has d+1 units (one per feature and bias)
- Output layer has K units
- Training procedure determines parameters that optimize loss function
  - Backpropagation
  - Learn optimal $W^{[i]}, b^{[i]}$ at layer *i*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



Logistic Unit: A simple NN



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

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

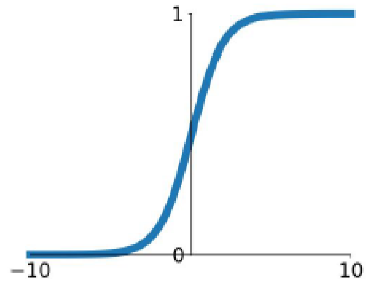
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

No hidden layers

Activation Functions

Sigmoid

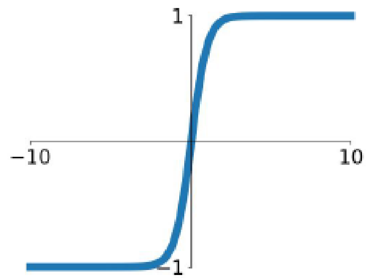
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Positive Classification

tanh

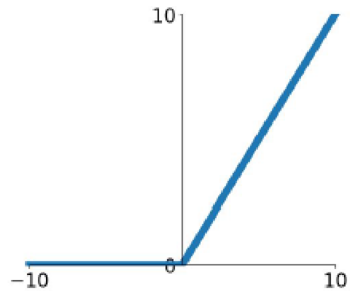
$$\tanh(x)$$



Regression

ReLU

$$\max(0, x)$$



Positive Classification

Why Non-Linear Activations?

- Assume g is linear: $\mathrm{g}(z) = Uz$
  - At layer 1: $z^{[1]} = \mathrm{W}^{[1]\,\mathrm{T}}x + \mathrm{b}^{[1]}$
  - $a^{[1]} = \mathrm{g}\left(\mathrm{z}^{[1]}\right) = \mathrm{Uz}^{[1]} = \mathrm{UW}^{[1]\mathrm{T}}\mathrm{x} + \mathrm{Ub}^{[1]}$
- Layer 2:
  - $a^{[2]} = \mathrm{g}\left(\mathrm{z}^{[2]}\right) = \mathrm{Uz}^{[2]} = \mathrm{UW}^{[2]\mathrm{T}}\mathrm{a}^{[1]} + \mathrm{Ub}^{[2]} = \mathrm{UW}^{[2]\mathrm{T}}\,\mathrm{UW}^{[1]\mathrm{T}}x + \mathrm{UW}^{[2]\mathrm{T}}\mathrm{Ub}^{[1]} + \mathrm{Ub}^{[2]}$
- Last layer
  - Output is linear in input!
  - Then NN will only learn linear functions

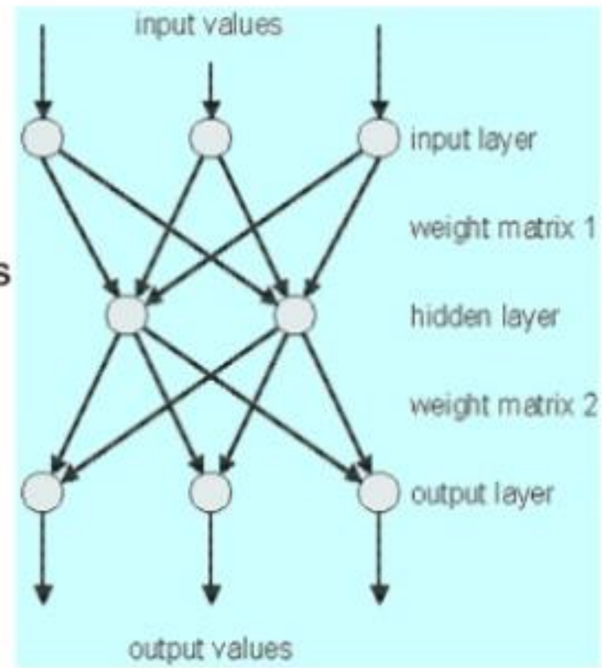
Multi-Layer Perceptron (MLP)

The Multi-Layer-Perceptron was first introduced by M. Minsky and S. Papert in 1969

Type:
Feedforward

Neuron layers:
1 input layer
1 or more hidden layers
1 output layer

Learning Method:
Supervised



- Neurons from one layer are fully connected to neurons in next layer

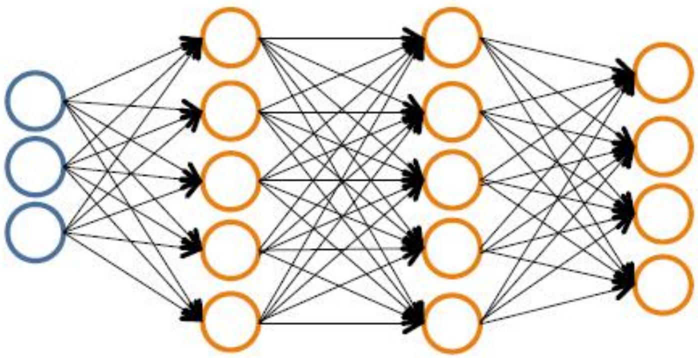
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



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

We want:

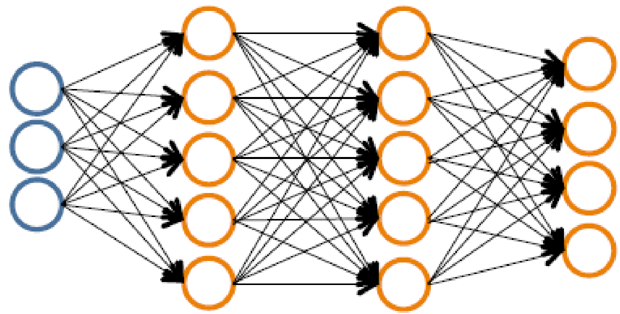
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
when pedestrian

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

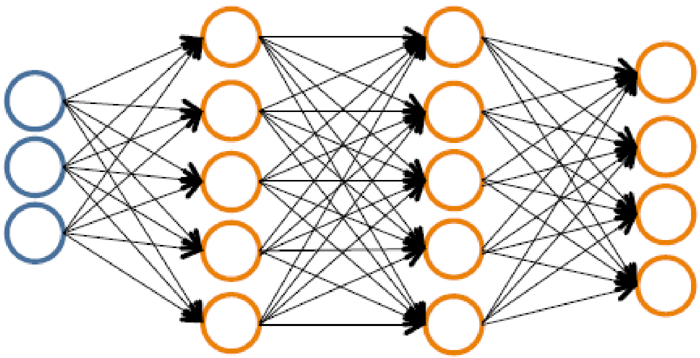
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\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

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.

Based on slide by Andrew Ng

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)

<u>Binary classification</u>

y = 0 or 1

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

Sigmoid

<u>Multi-class classification</u> (K classes)

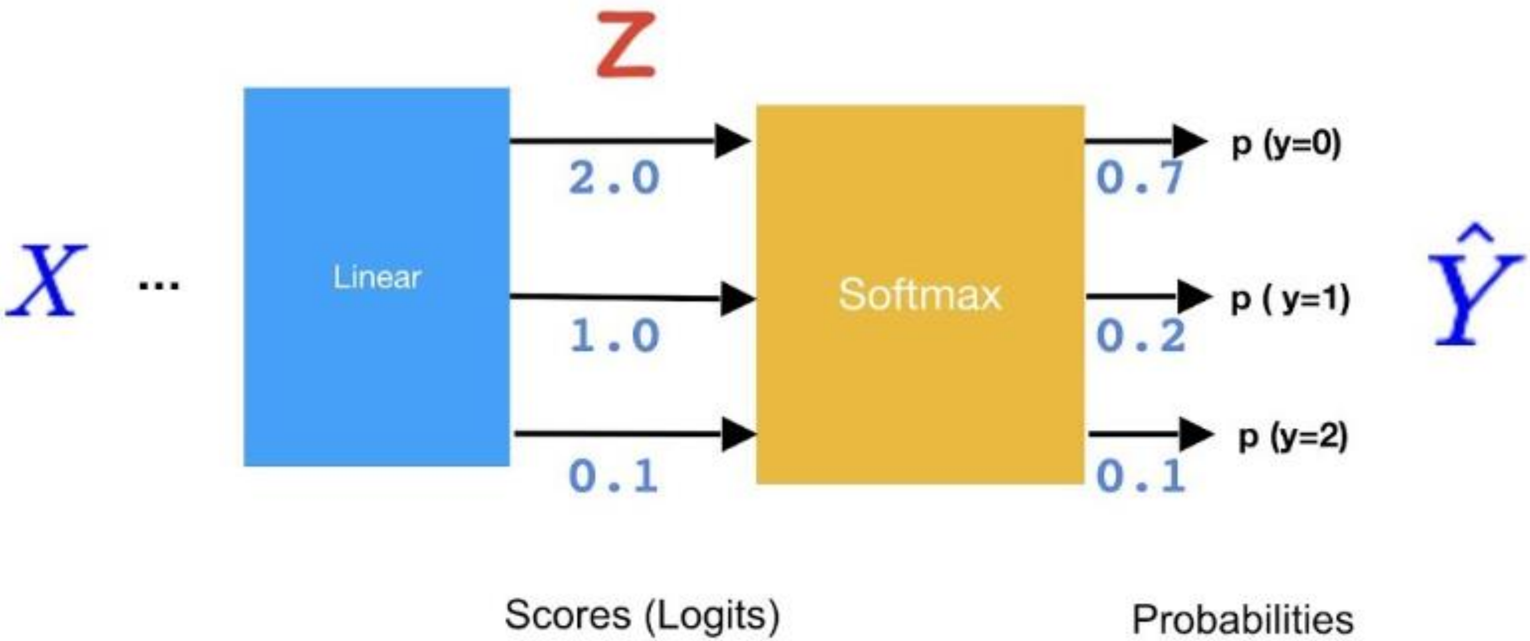
$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

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

Softmax

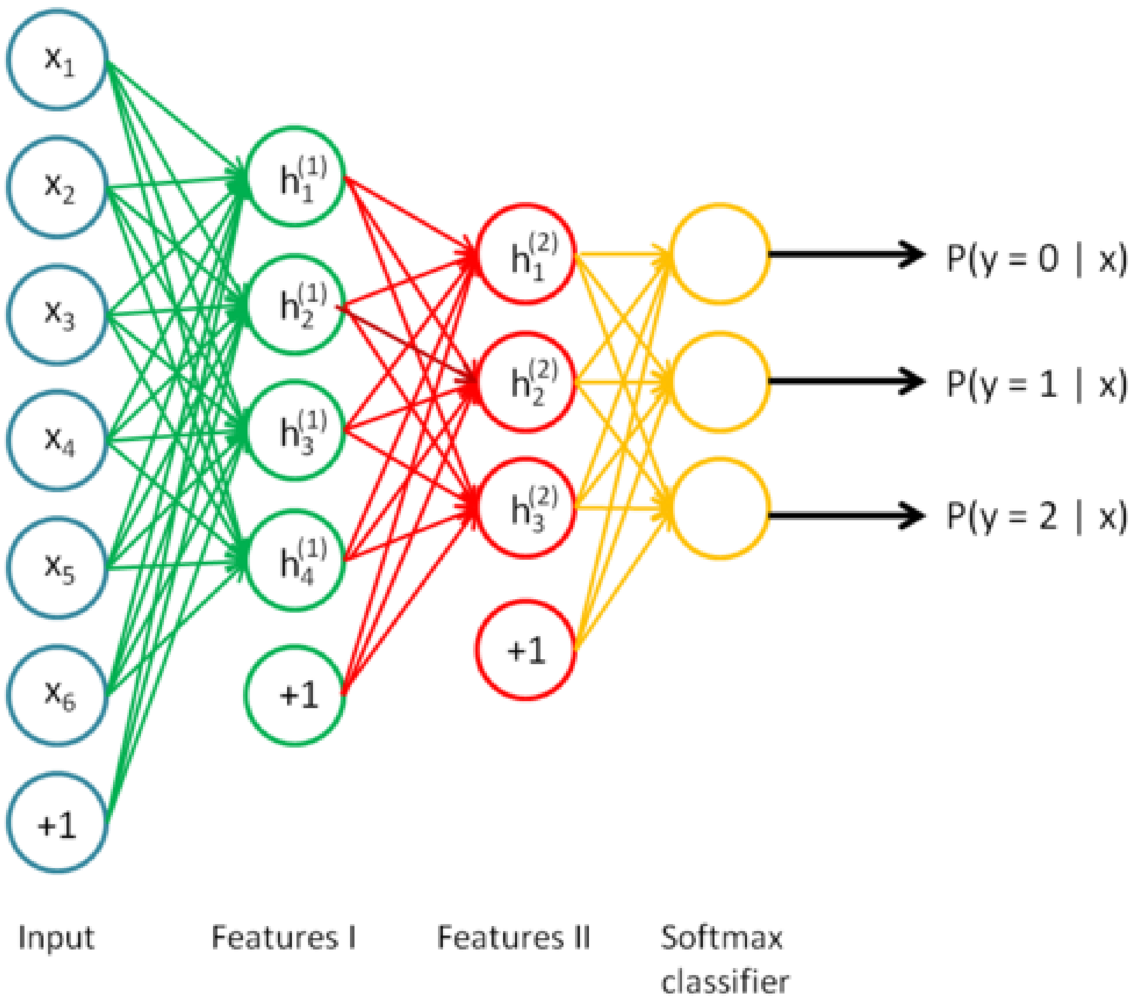
Softmax classifier



$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Predict the class with highest probability
- Generalization of sigmoid/logistic regression to multi-class

Multi-class classification

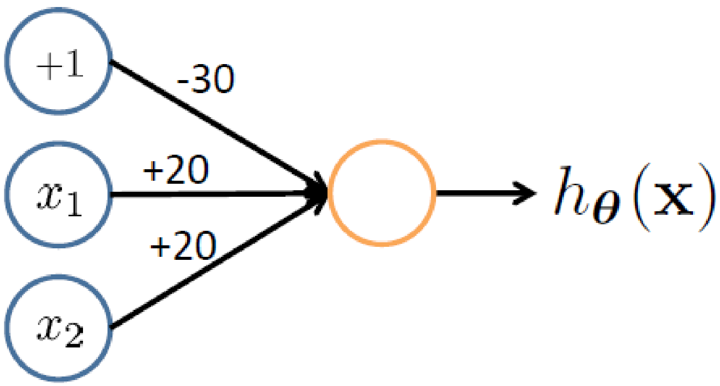


Representing Boolean Functions

Simple example: AND

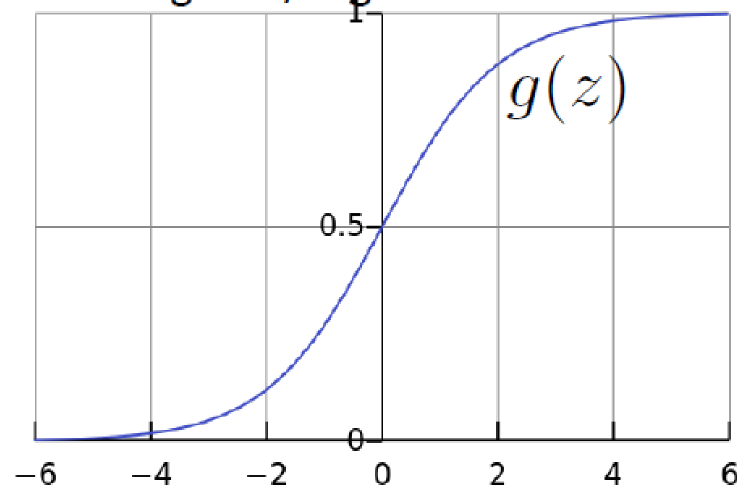
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



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

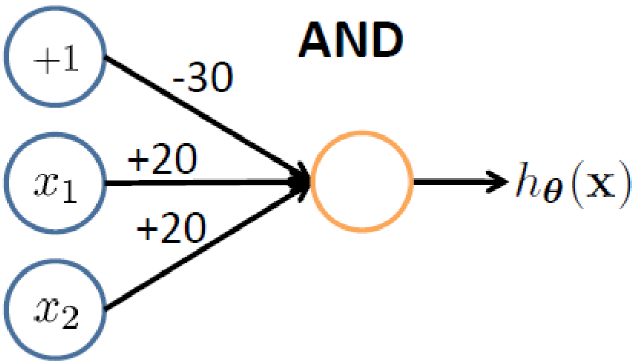
Logistic / Sigmoid Function



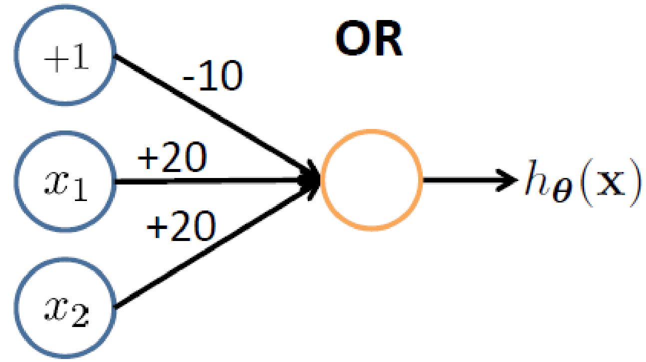
x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Representing Boolean Functions

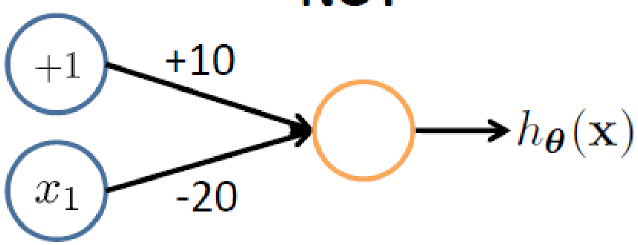
AND



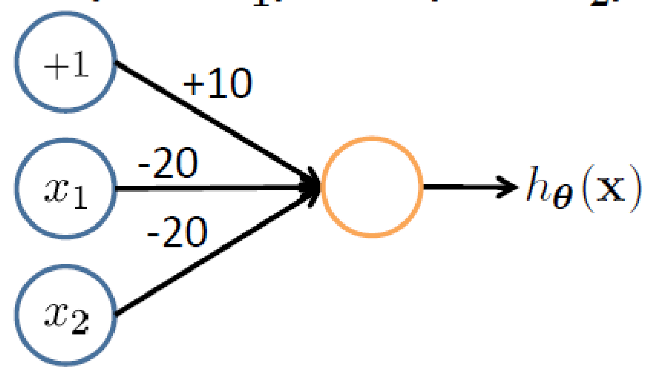
OR



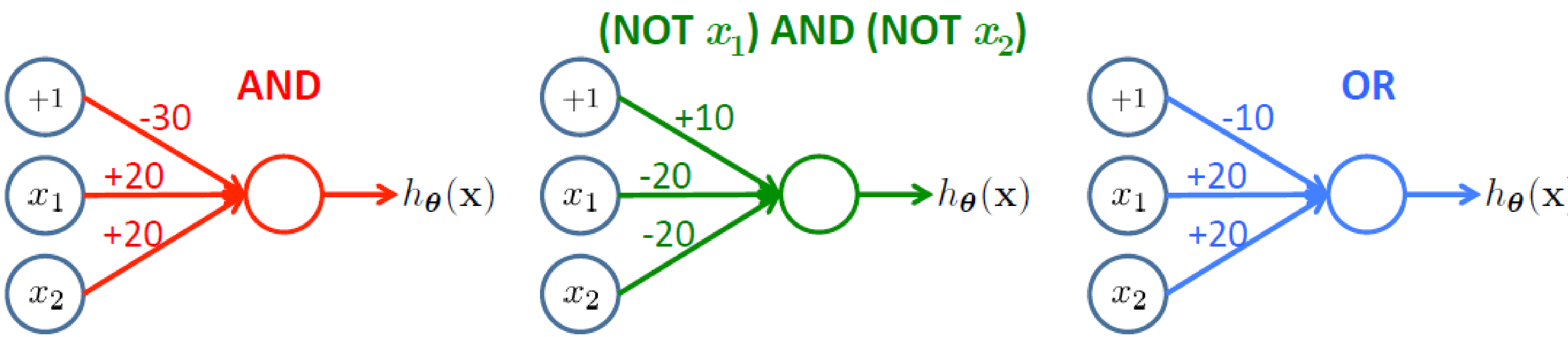
NOT

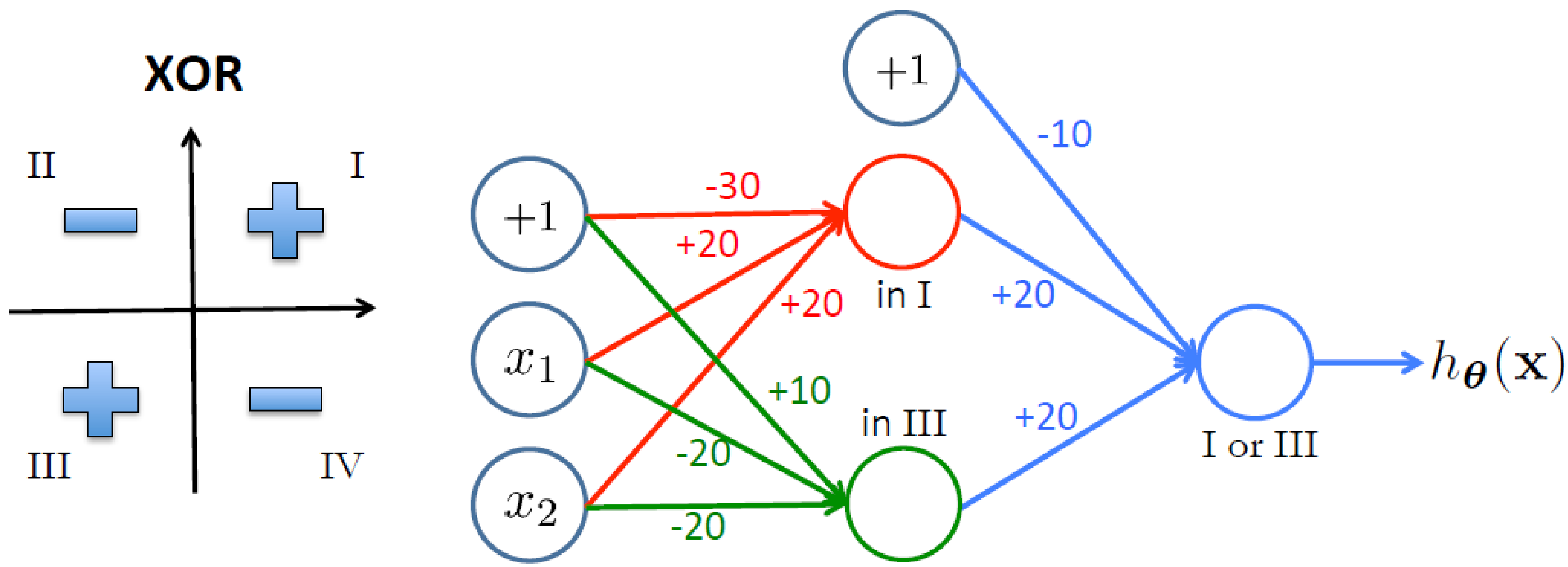


(NOT x_1) AND (NOT x_2)



Combining Representations



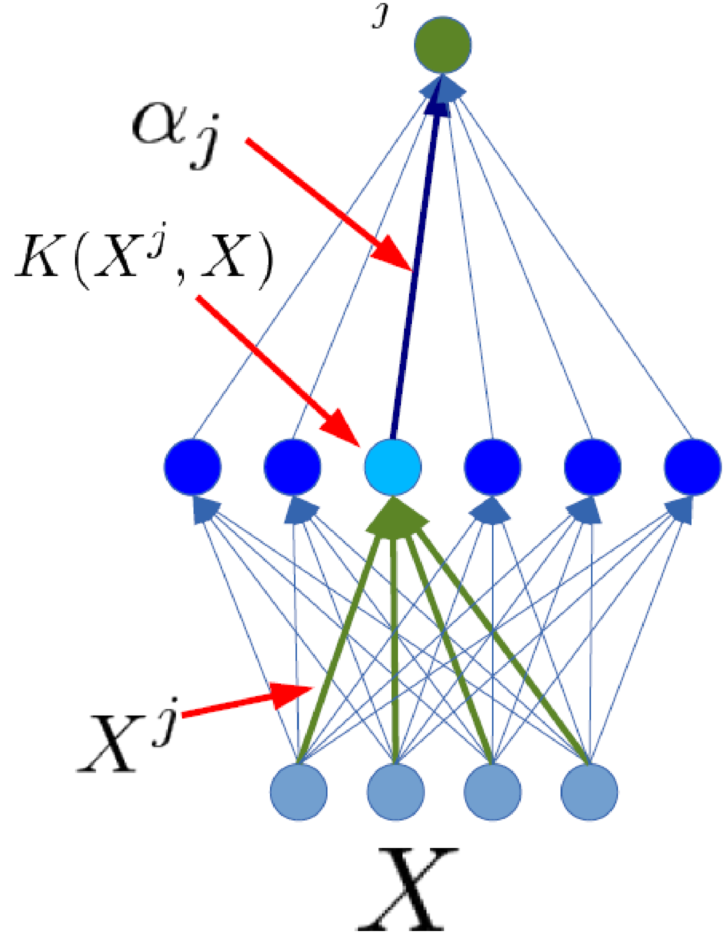


XOR is non-linear!

Which models are deep?

- **2-layer models are not deep (even if you train the first layer)**
  - Because there is no feature hierarchy
- **Neural nets with 1 hidden layer are not deep**
- **SVMs and Kernel methods are not deep**
  - Layer1: kernels; layer2: linear
  - The first layer is "trained" in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- **Classification trees are not deep**
  - No hierarchy of features. All decisions are made in the input space

$$G(X, \alpha) = \sum_j \alpha_j K(X^j, X)$$



Outline

- Feed-Forward architectures
  - Non-linear activations
  - Multi-Layer Perceptron
  - Multi-class classification (softmax unit)
  - Representing Boolean functions
- Convolutional Neural Networks
  - Convolution layer
  - Max pooling layer

Convolutional Neural Networks

First strong results

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

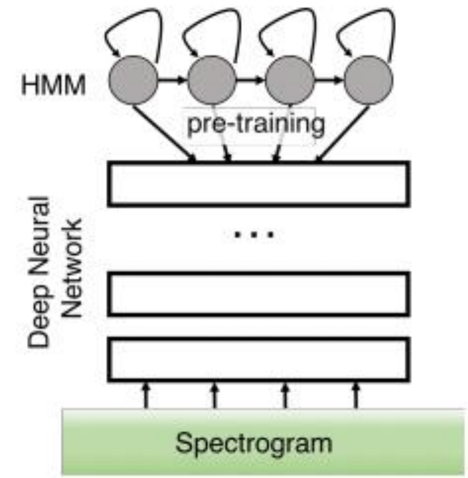
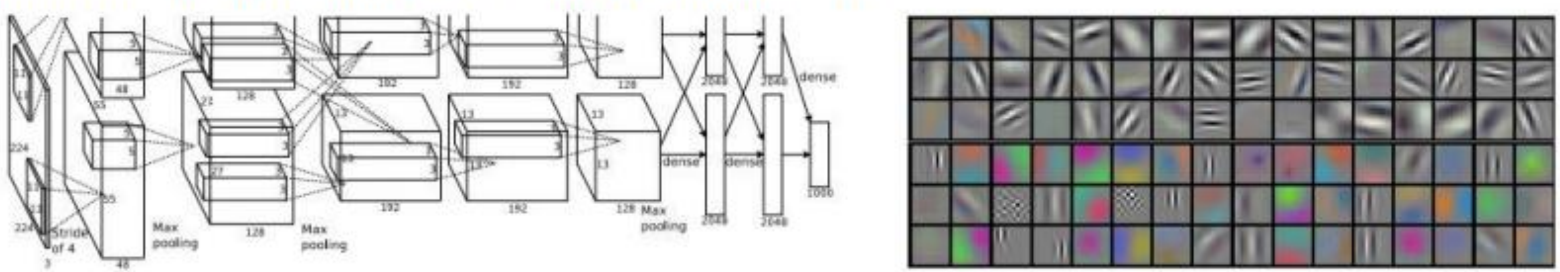


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

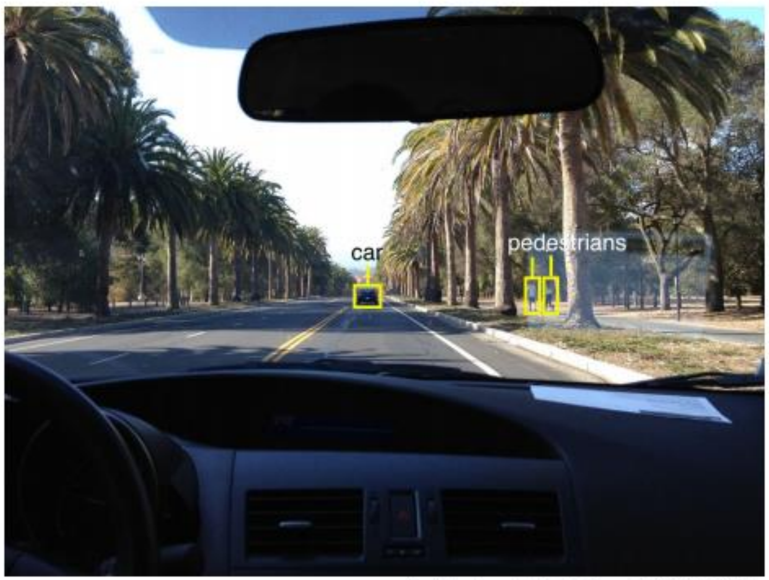
Convolutional Nets

Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

Convolutional Nets



self-driving cars

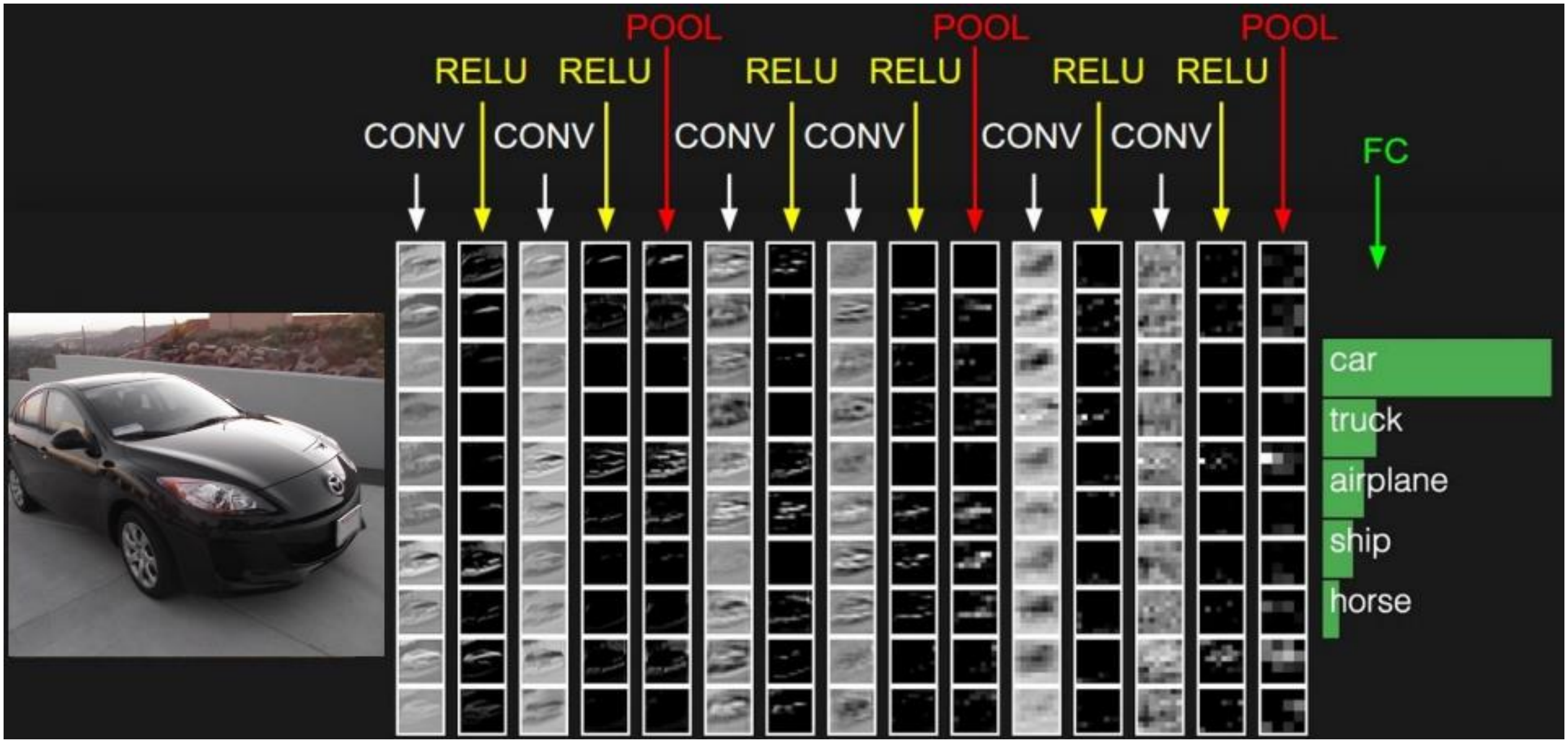
- Object recognition
- Steering angle prediction
- Assist drivers in making decisions

- Image captioning

Convolutional Nets

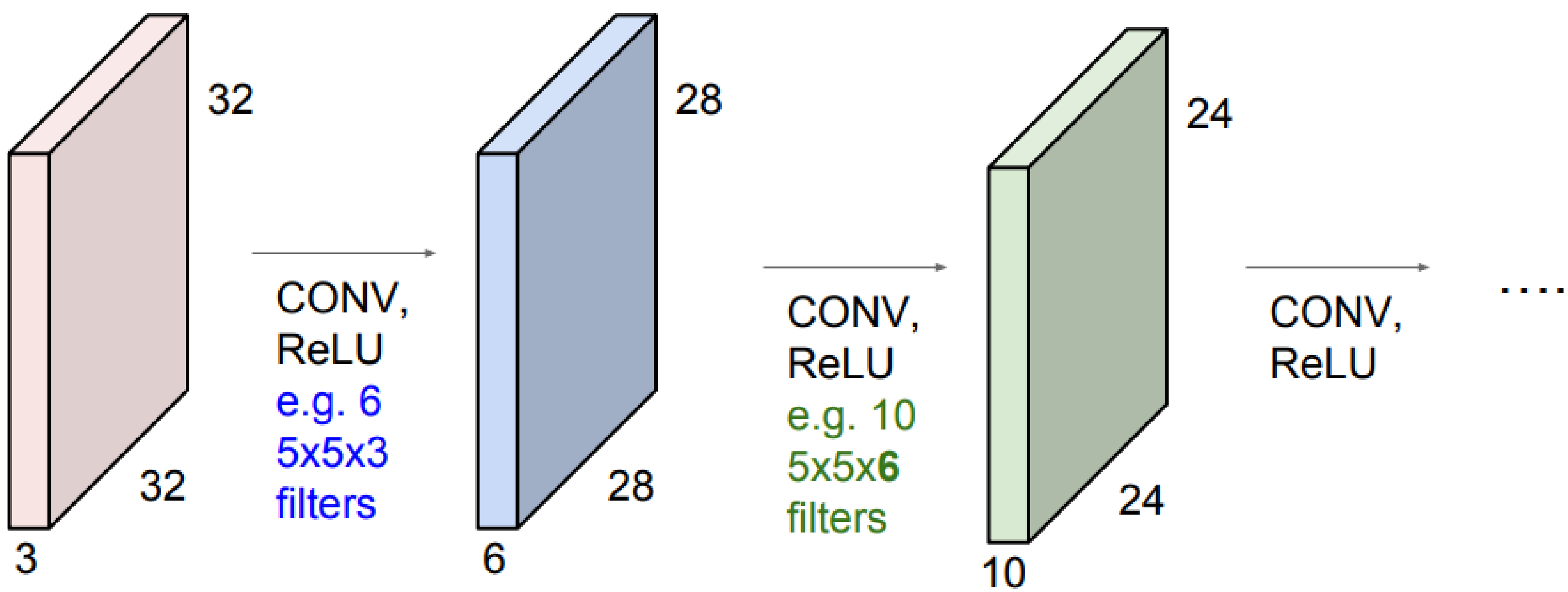
- Particular type of Feed-Forward Neural Nets
  - Invented by [LeCun 89]
- Applicable to data with natural grid topology
  - Time series
  - Images
- Use convolutions on at least one layer
  - Convolution is a linear operation
  - Also use pooling operation
  - Used for dimensionality reduction and learning hierarchical feature representations

Convolutional Nets



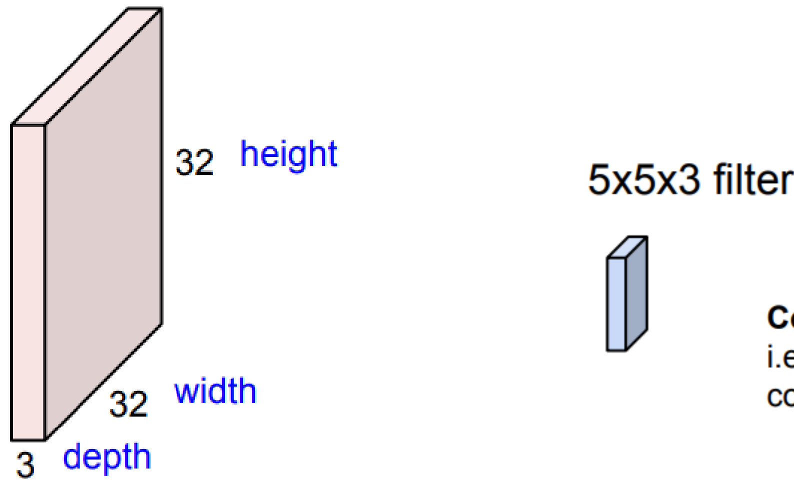
Convolutional Nets

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolution Layer

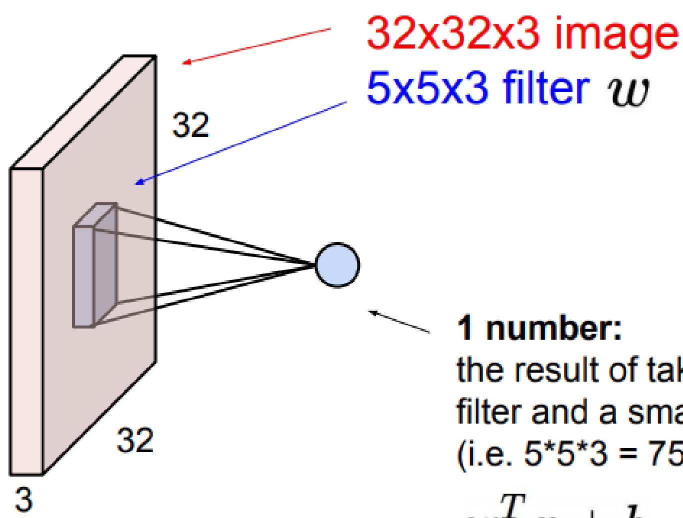
32x32x3 image -> preserve spatial structure



Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

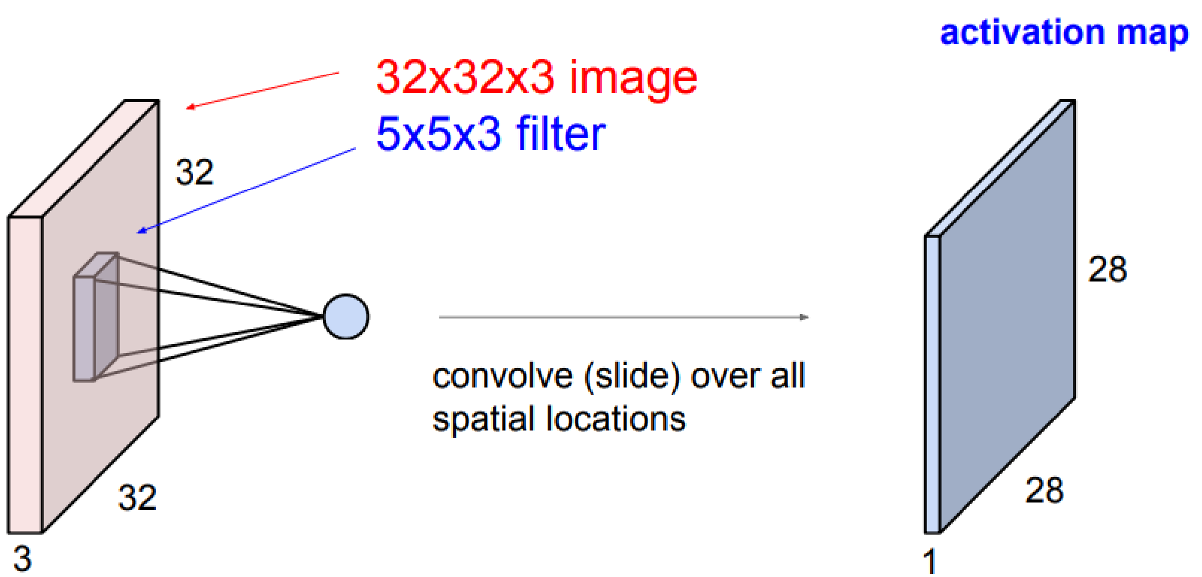
- Depth of filter always depth of input

Convolution Layer

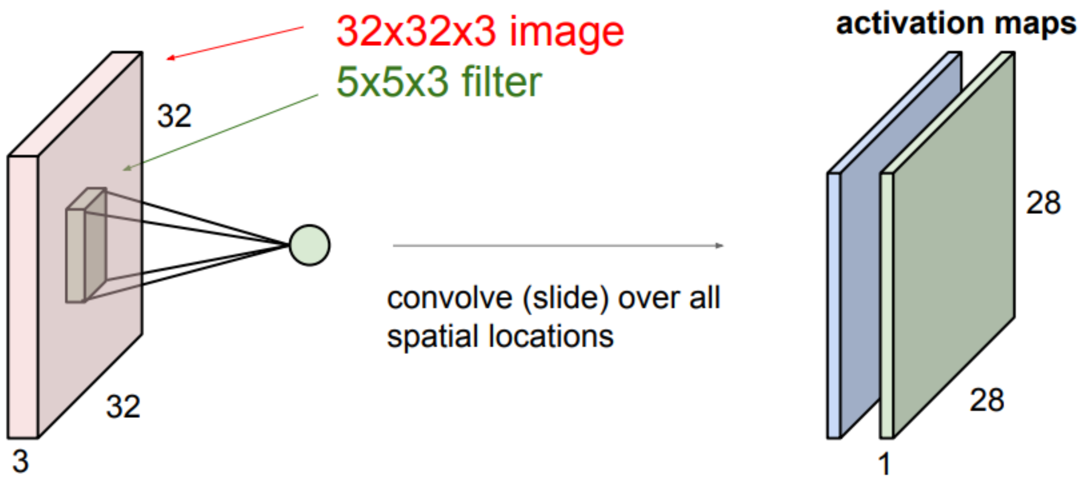


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

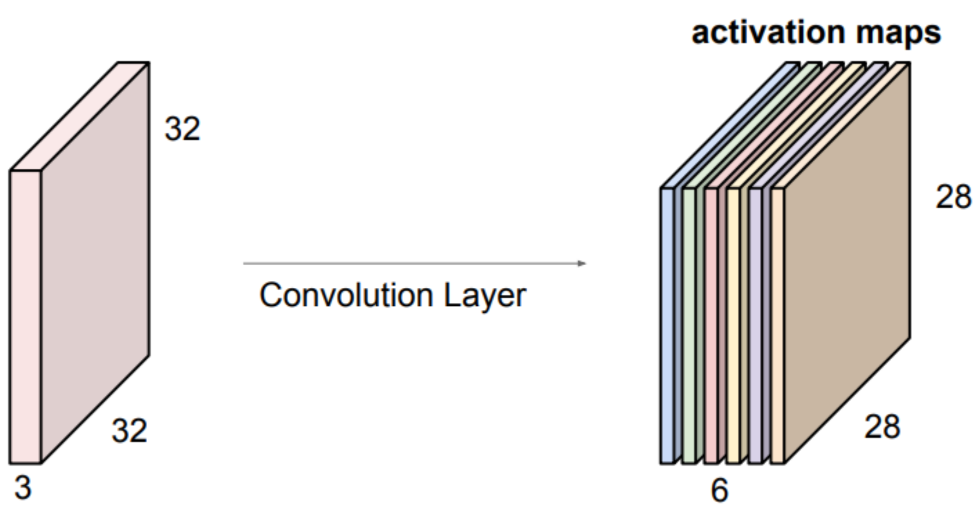
$$w^T x + b$$



Convolution Layer



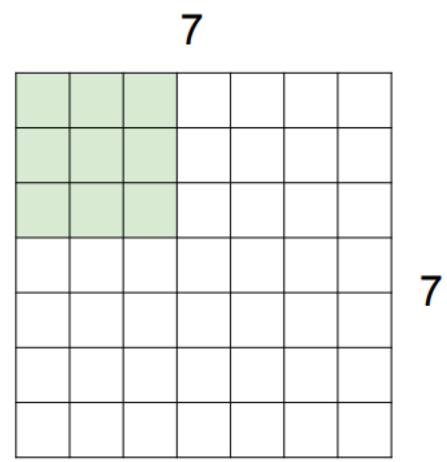
Second, green filter



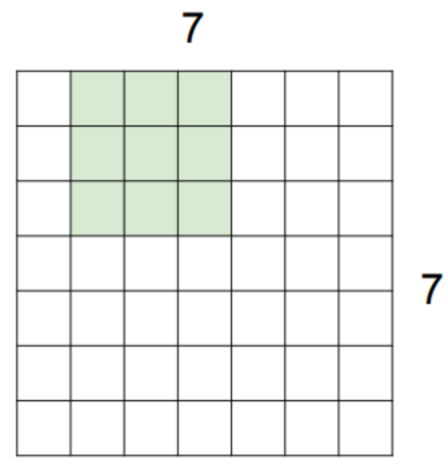
6 filters

Convolutions

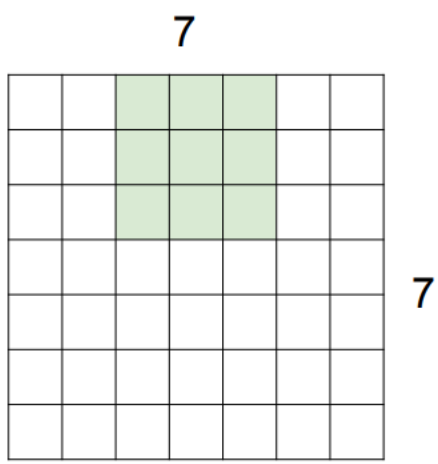
A closer look at spatial dimensions:

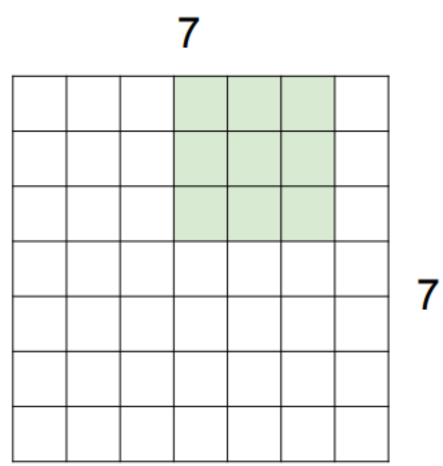


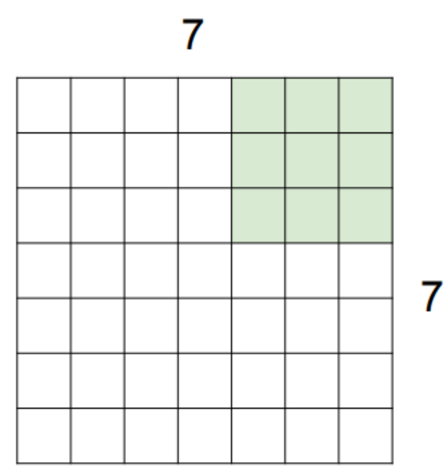
7x7 input (spatially)
assume 3x3 filter



=> 5x5 output

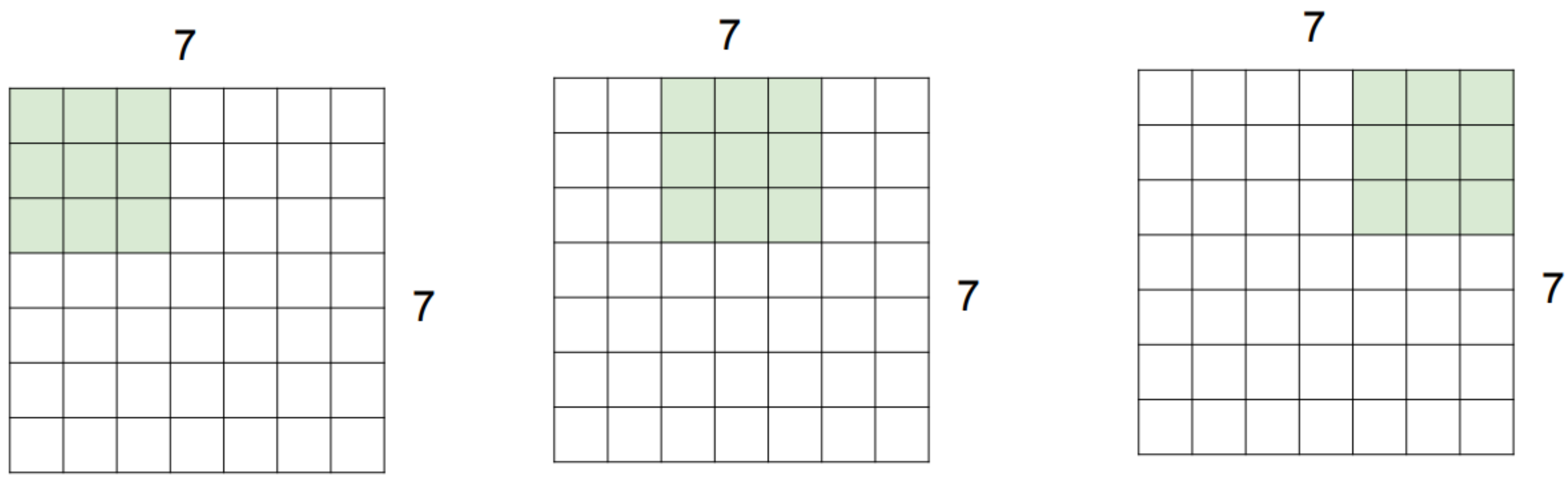






Convolutions with stride

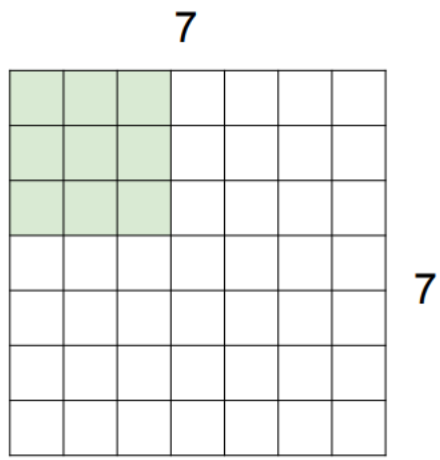
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



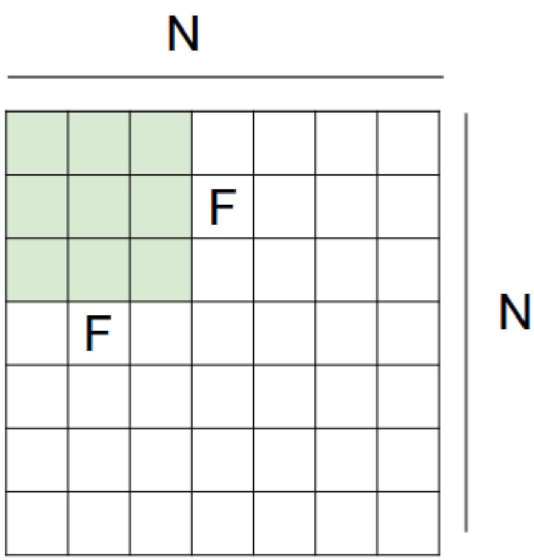
=> 3x3 output!

Convolutions with stride

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**



doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

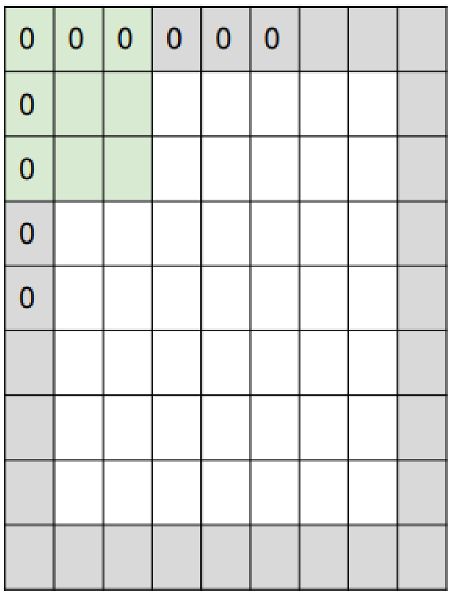


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Padding

In practice: Common to zero pad the border

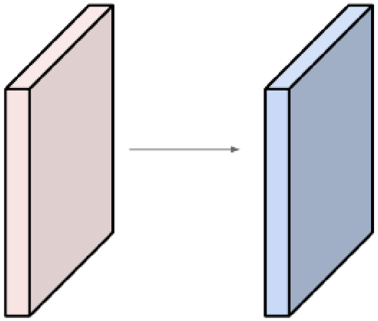


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Examples

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Summary: Convolutional Layer

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Acknowledgements

- Slides made using resources from:
  - Yann LeCun
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
- Thanks!