DS 4400

Machine Learning and Data Mining I

Alina Oprea
Associate Professor, CCIS
Northeastern University

November 17 2020

Announcements

- HW 4 is due Saturday at midnight
- Final exam will start at 2:30pm on Thursday, Nov 19
  - It will be up for 24 hours
  - You can pick up a time frame of 3 hours
  - No class lecture on Nov 19
- Project milestone: Nov 25
- Ethics of AI: Tue, Dec 1, by Kevin Mills

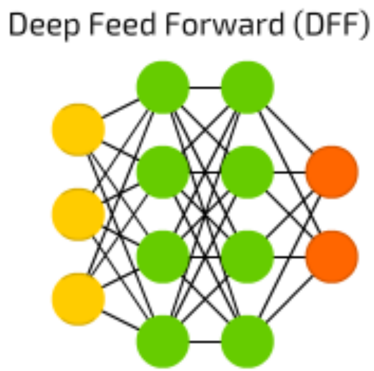
Outline

- Convolutional Neural Networks
  - Max pooling
  - Famous architectures
- Backpropagation algorithm
  - Example for 2-layer neural network

Neural Network Architectures

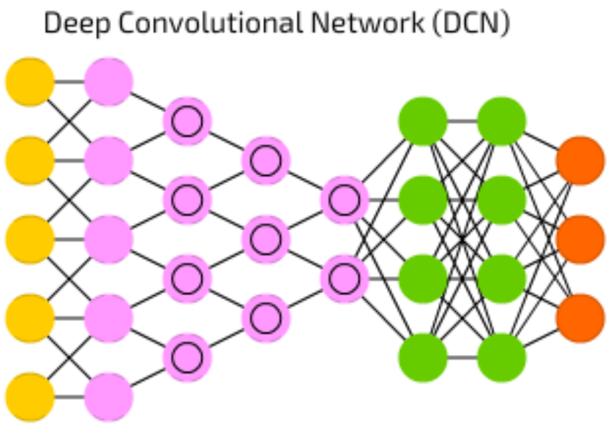
Feed-Forward Networks

- Neurons from each layer connect to neurons from next layer



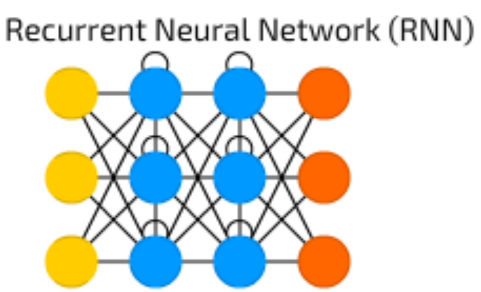
Convolutional Networks

- Includes convolution layer for feature reduction
- Learns hierarchical representations

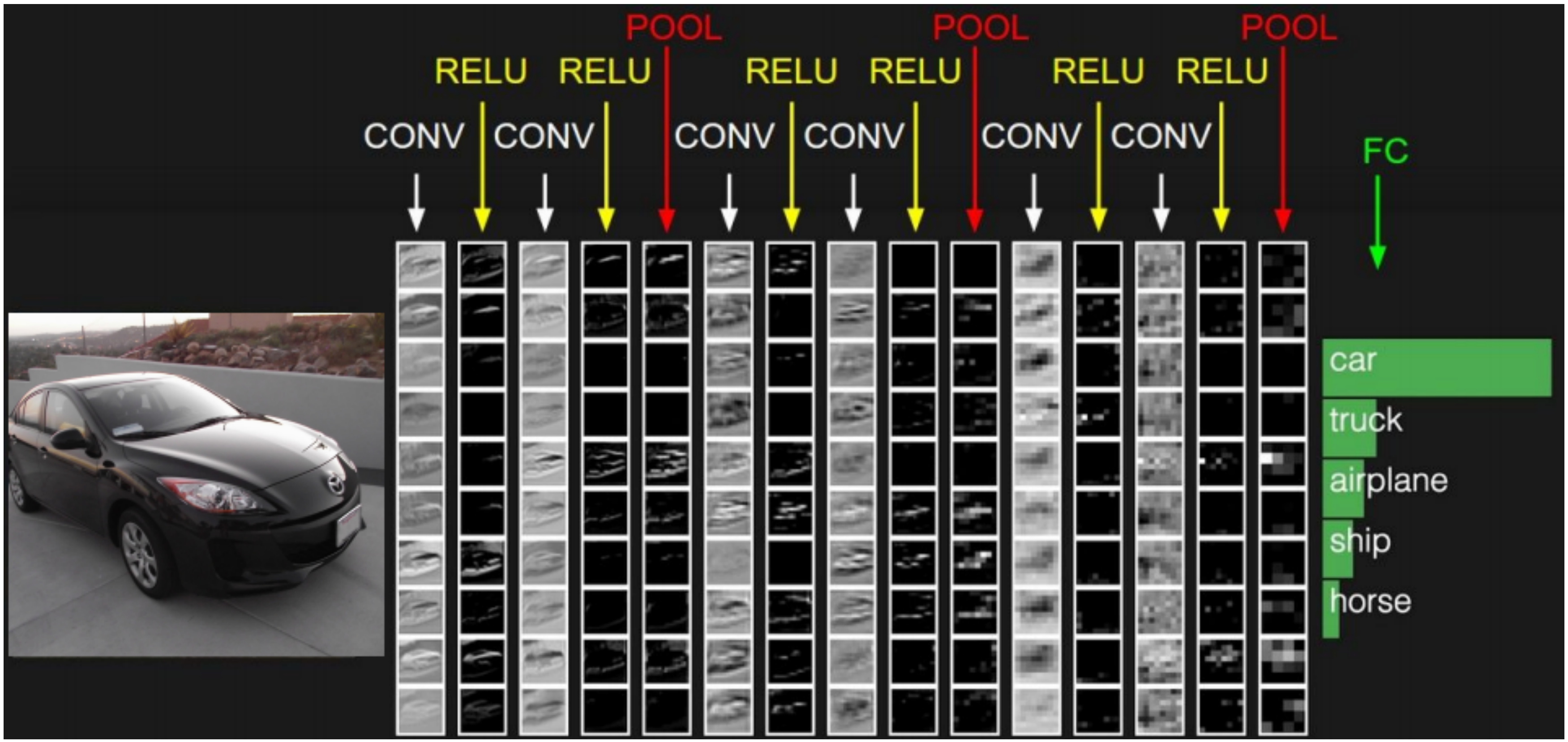


Recurrent Networks

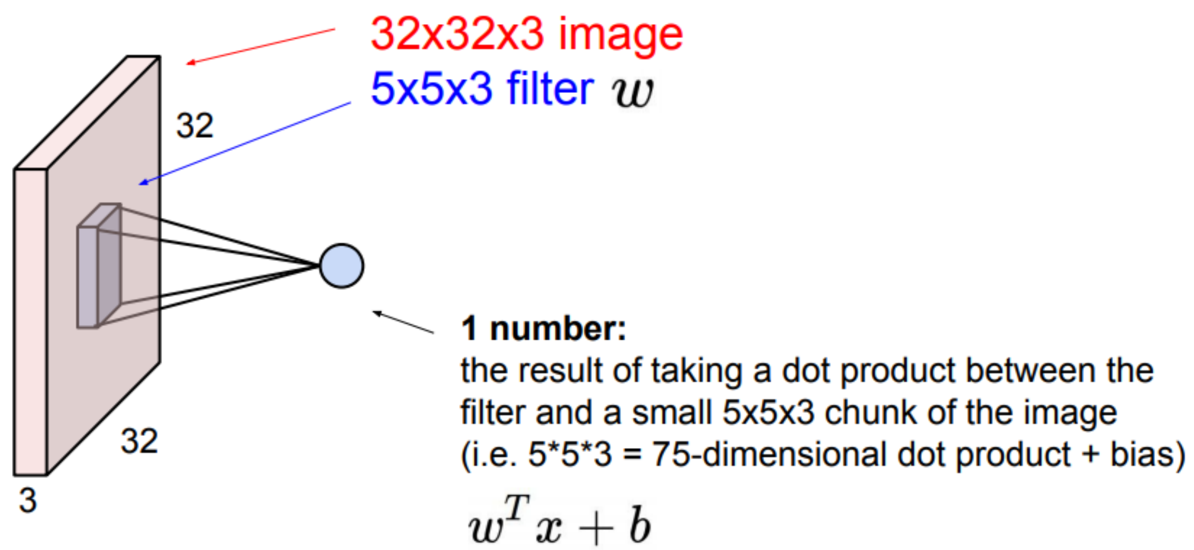
- Keep hidden state
- Have cycles in computational graph

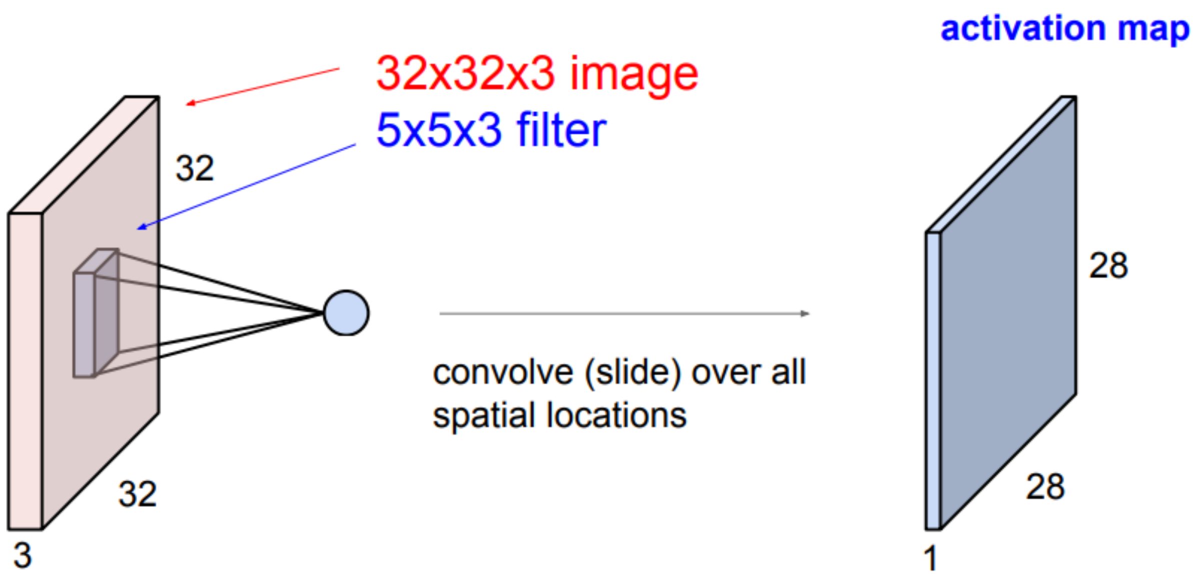


Convolutional Nets

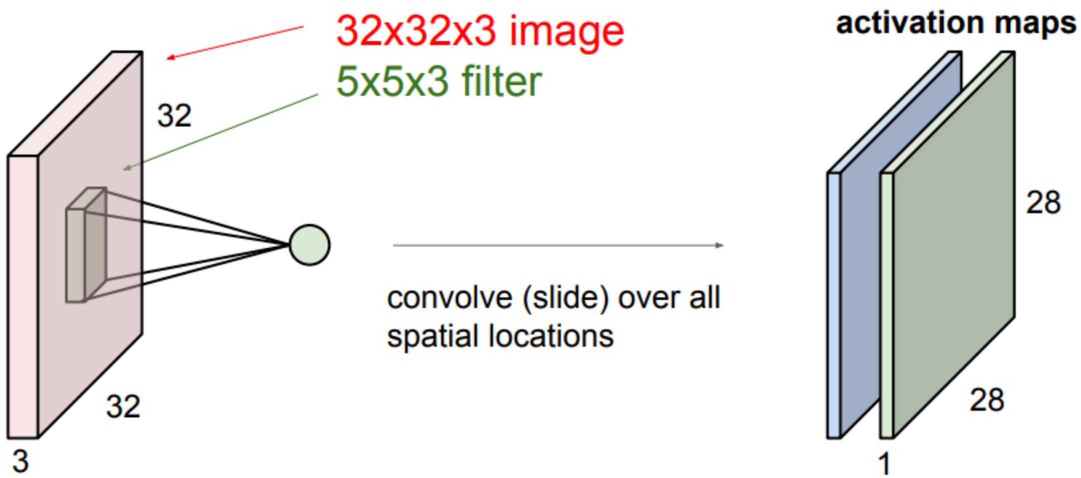


Convolution Layer

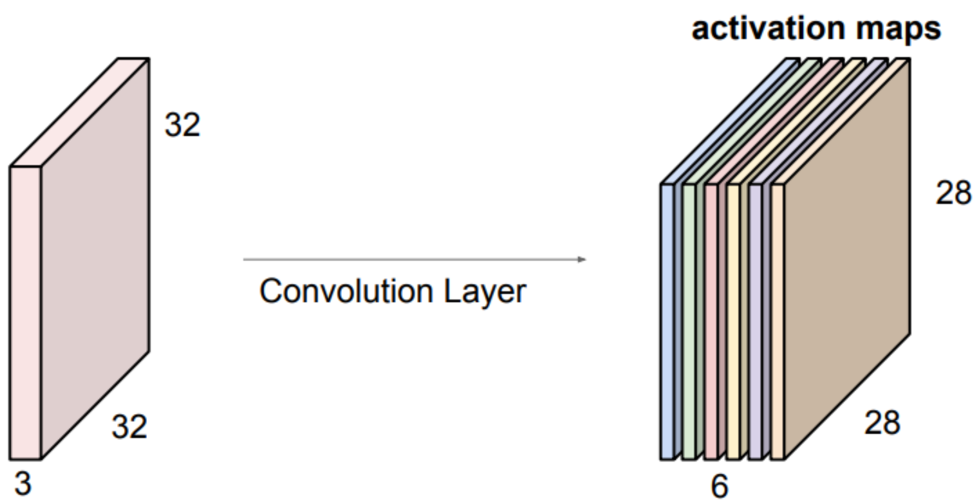




Convolution Layer



Second, green filter

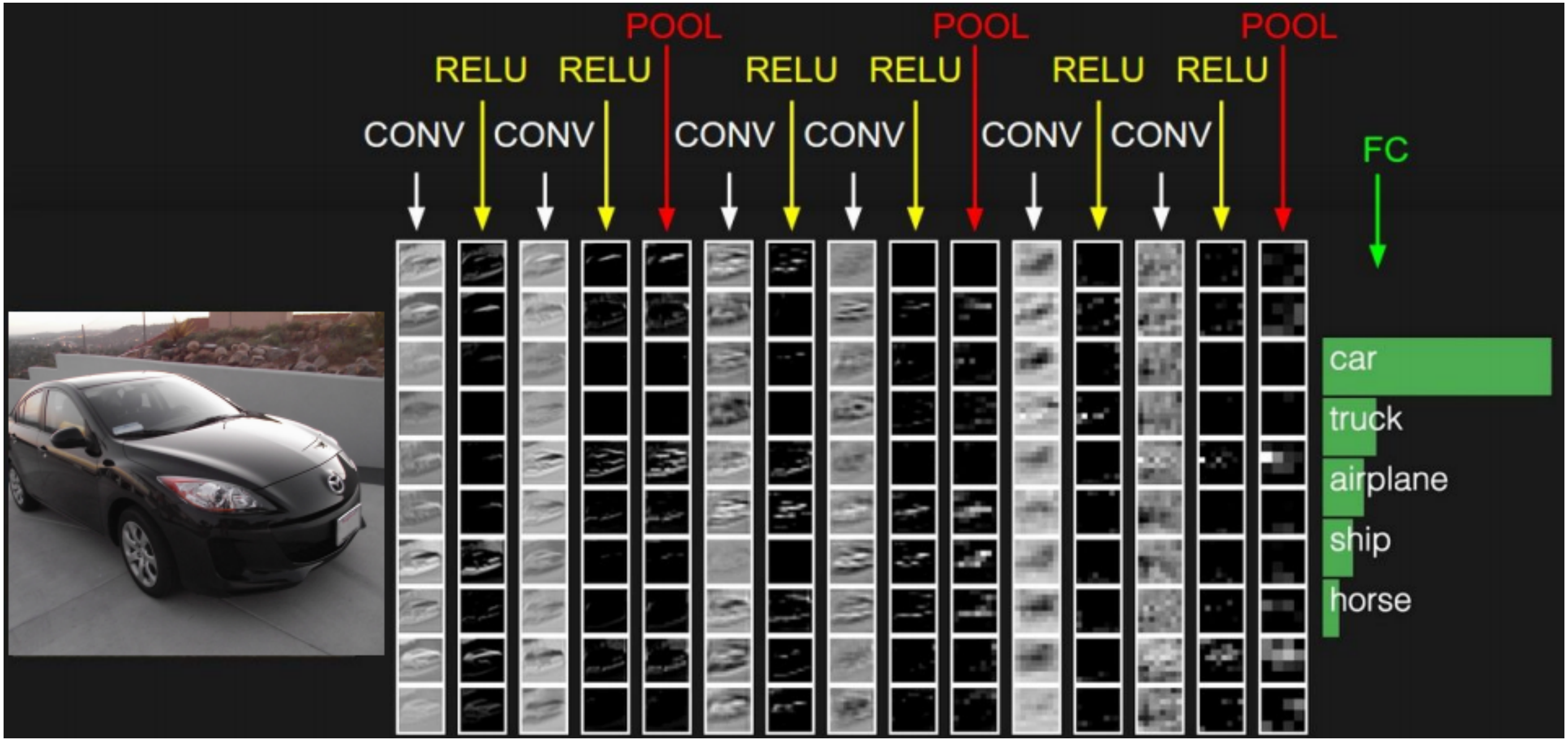


6 filters

Convolution layer: Takeaways

- Convolution is a linear operation
  - Reduces parameter space of Feed-Forward Neural Network considerably
  - Capture locality of pixels in images
  - Smaller filters need less parameters
  - Multiple filters in each layer (computation can be done in parallel)
- Convolutions are followed by activation functions
  - Typically ReLU

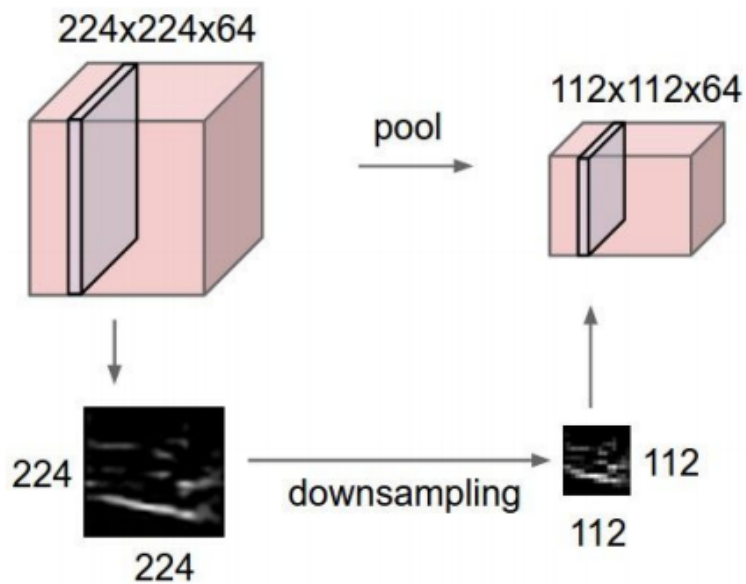
Convolutional Nets



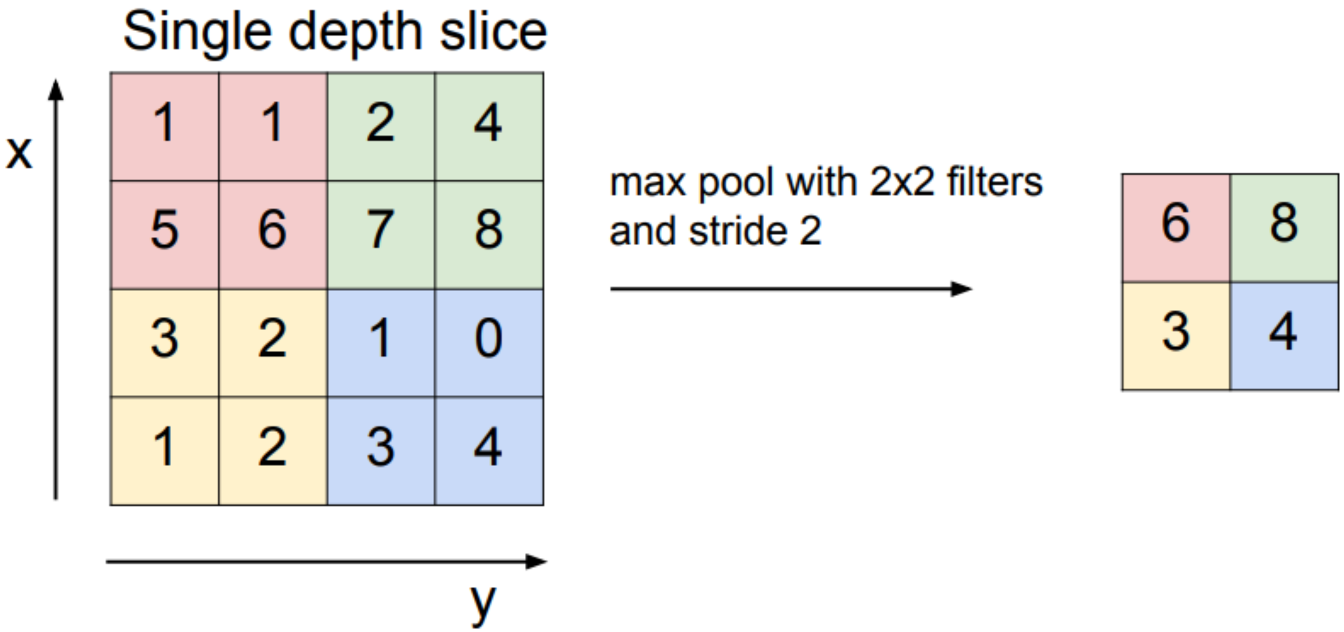
Pooling layer

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling

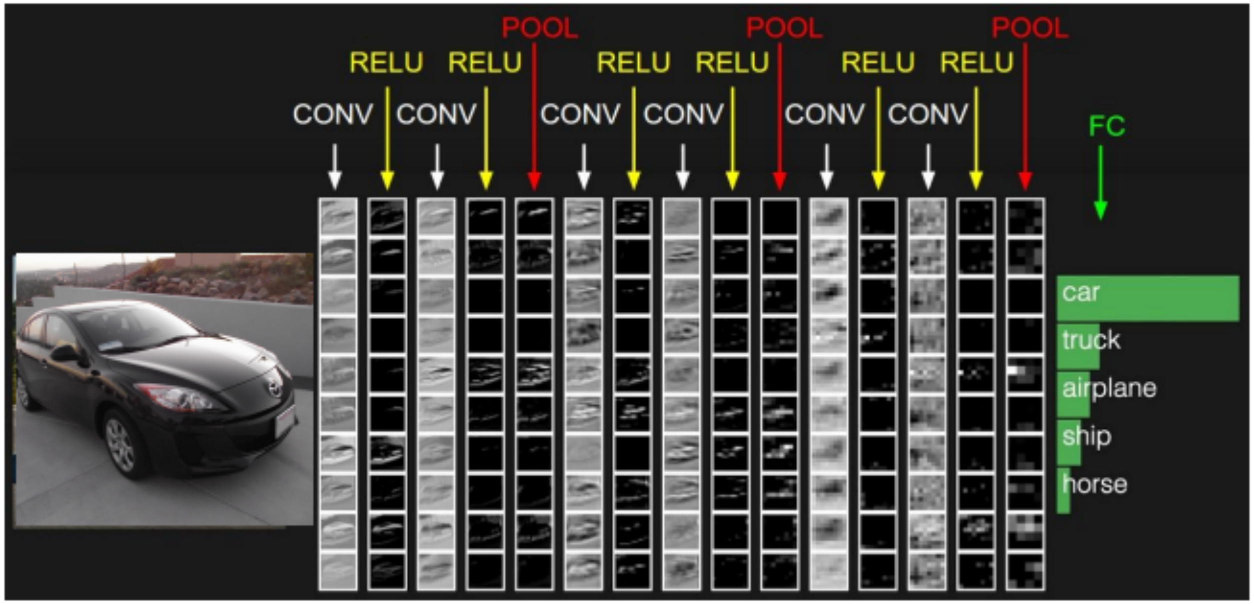


- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
    - their spatial extent F,
    - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F)/S + 1$
    - $H_2 = (H_1 - F)/S + 1$
    - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Convolutional Nets

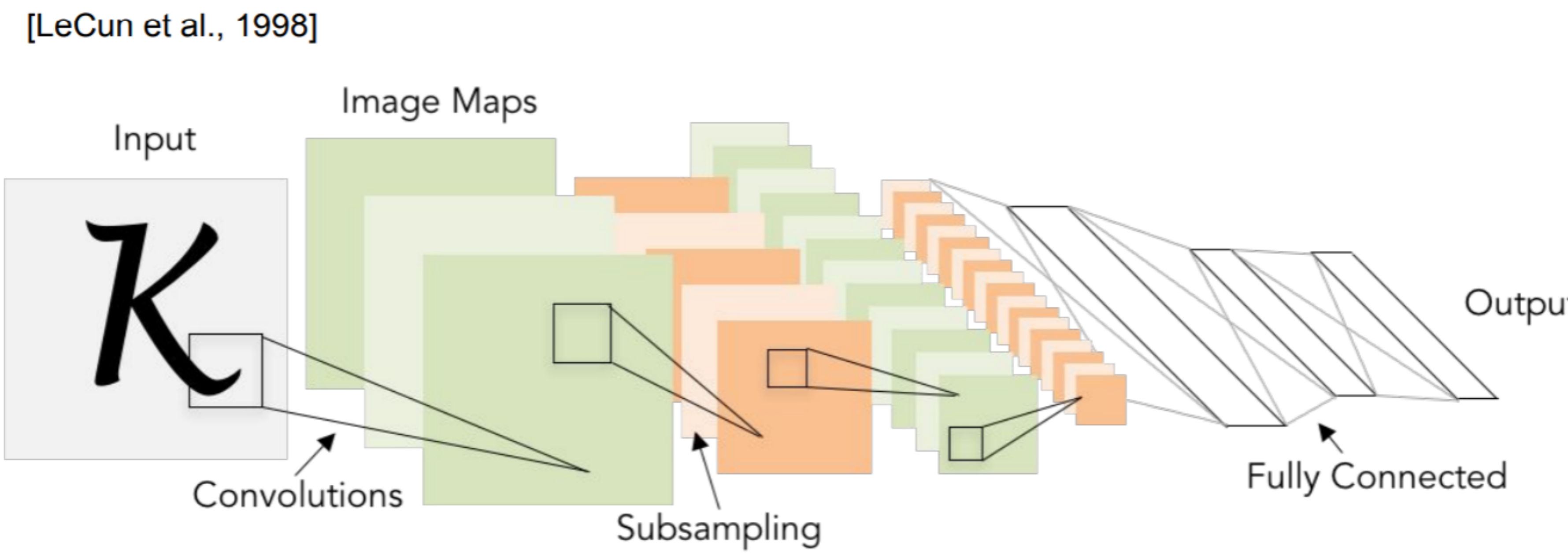
Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



- FC layers are usually at the end, after several Convolutions and Pooling layers

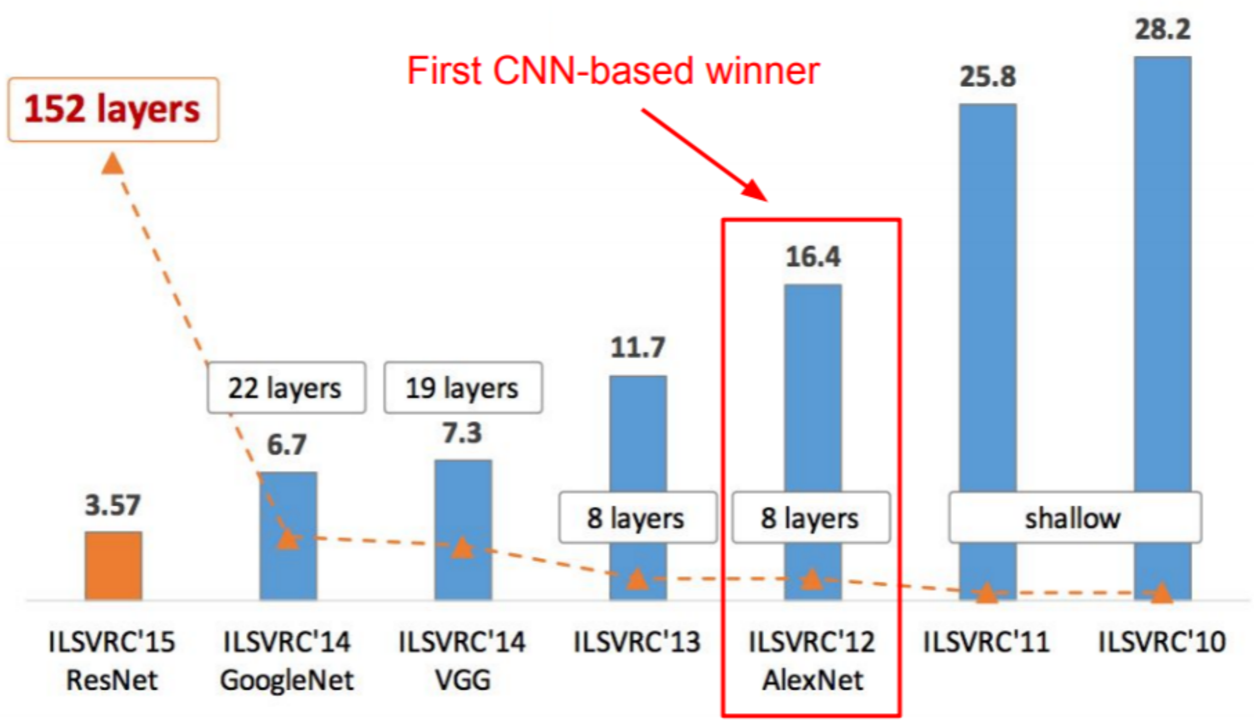
LeNet 5



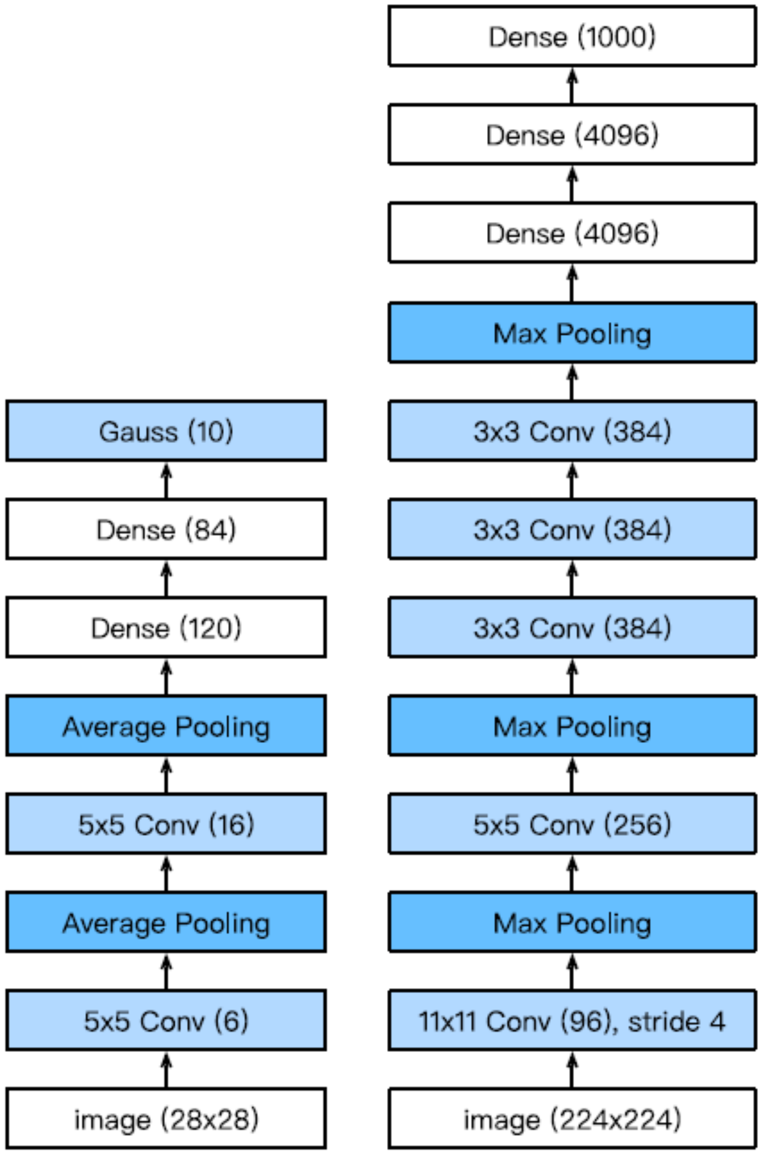
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]

History

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



LeNet (left) and AlexNet (right)



Main differences

- Deeper
- Wider layers
- ReLU activation
- More classes in output layer
- Max Pooling instead of Avg Pooling

VGGNet

Case Study: VGGNet

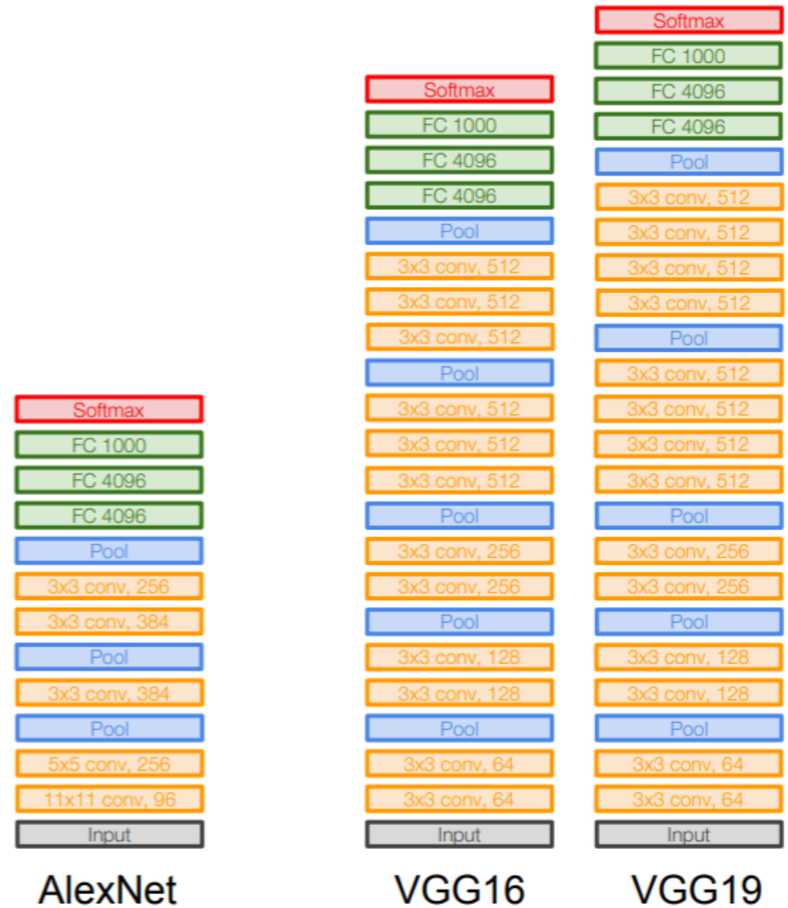
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14



138 million
parameters

Summary CNNs

- Convolutional Nets are Feed-Forward Networks with at least one convolution layer and optionally max pooling layers
- Convolutions enable dimensionality reduction, are translation invariant and exploit locality
- Much fewer parameters relative to Feed-Forward Neural Networks
  - Deeper networks with multiple small filters at each layer is a trend
- Fully connected layer at the end (fewer parameters)
- Learn hierarchical feature representations
  - Data with natural grid topology (images, maps)
- Reached human-level performance in ImageNet in 2014

How to train Neural Networks?

- Backpropagation algorithm
- David Rumelhart, Geoffrey Hinton, Ronald Williams. "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. 1986
- Applicable to both FFNN and CNN
- Extension of Gradient Descent to multi-layer neural networks

Reminder: Logistic Regression

$$J(\theta) = -\sum_{i=1}^{N}[y_{\mathrm{i}}\log h_\theta(x_i) + (1 - y_i)\log\left(1 - h_\theta(x_i)\right)]$$

- Cost of a single instance:

$$\text{cost}\,(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

- Can re-write objective function as

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} \underbrace{\text{cost}\left(h_\theta(x_i), y_i \right)}_{\text{Cross-entropy loss}}$$

Gradient Descent

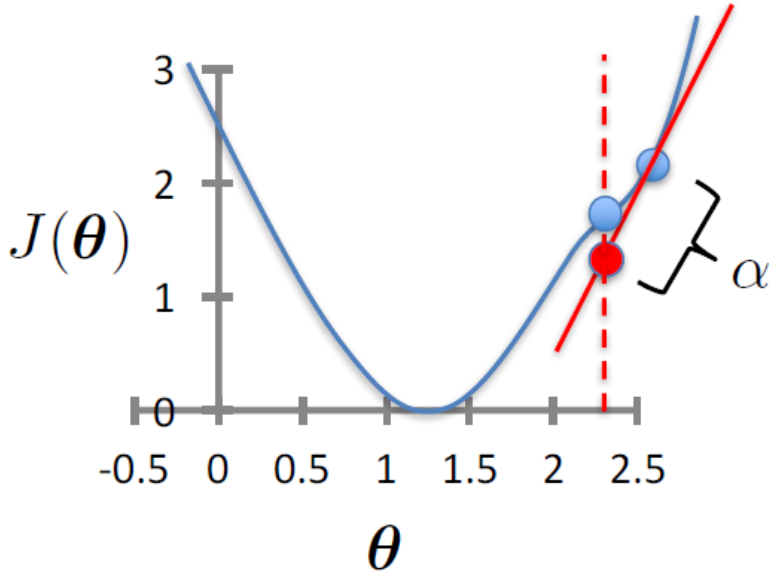
- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$\boldsymbol{\theta} = (W, b)$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- Converges for convex objective
- Could get stuck in local minimum for non-convex objectives

Training Neural Networks

- Training data $x_1, y_1, \ldots x_N, y_N$
- One training example $x_i = (x_{i1}, \ldots x_{id})$, label y_i
- One forward pass through the network
  - Compute prediction $\hat{y}_i = h(x_i)$
- Loss function for one example
  - $L(\hat{y}, y) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$

Cross-entropy loss

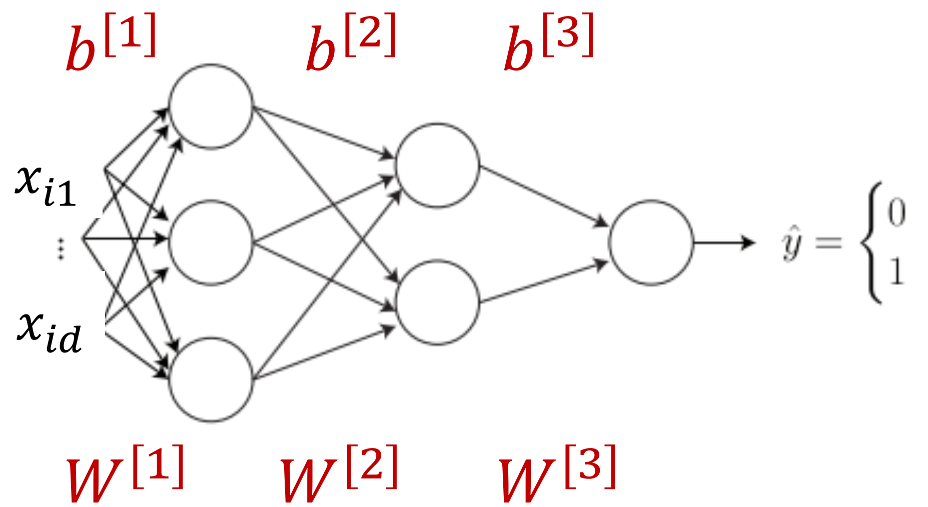
- Loss function for training data
  - $J(W, b) = \frac{1}{N}\sum_i L\left(\hat{y}_i, y_i\right) + \lambda R(W, b)$

GD for Neural Networks

- Initialization
  - For all layers ℓ
    - Initialize $W^{[\ell]}, b^{[\ell]}$
- Backpropagation
  - Fix learning rate α
  - For all layers ℓ (starting backwards)
    - $W^{[\ell]} = W^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$
    - $b^{[\ell]} = b^{[\ell]} - \alpha \sum_{i=1}^{N} \frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$

Example 2 Hidden Layers

Training data
Dimension d

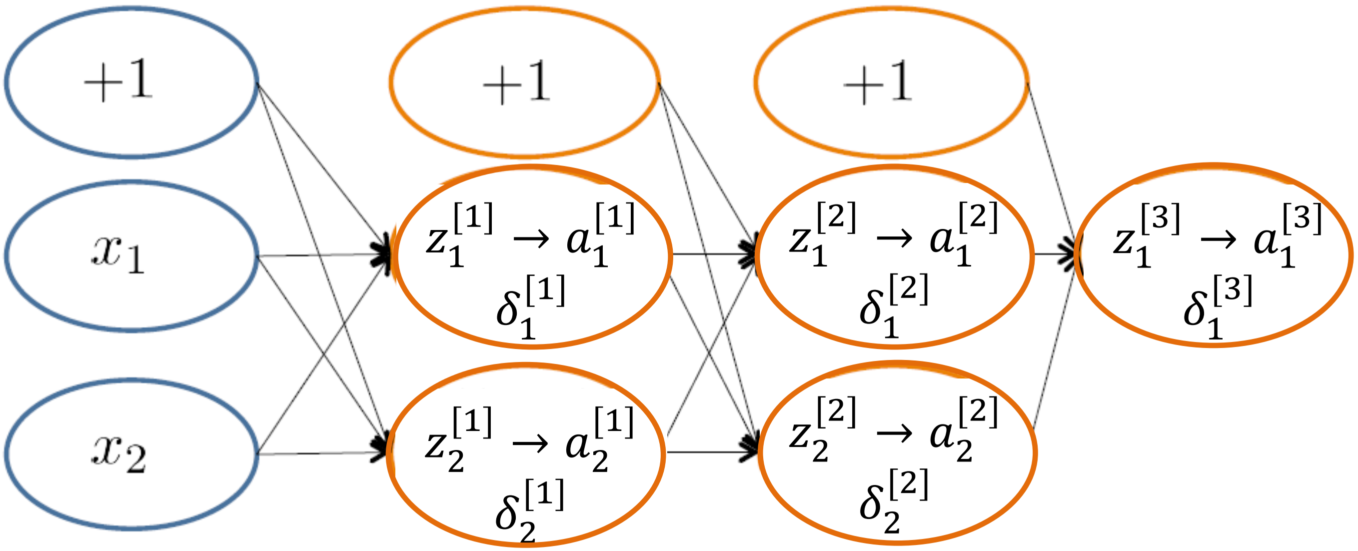


$$z^{[1]} = W^{[1]} x_i + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$
$$z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$$
$$\hat{y}^{(i)} = a^{[3]} = g(z^{[3]})$$

Parameter Initialization

- How about we set all W and b to 0?
- First layer
  - $z^{[1]} = \mathrm{W}^{[1]} x + \mathrm{b}^{[1]}$= (0,…0)
  - $a^{[1]} = \mathrm{g}\left(\mathrm{z}^{[1]}\right) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$
- Second layer
  - $z^{[2]} = \mathrm{W}^{[2]} x + \mathrm{b}^{[2]}$= (0,…0)
  - $a^{[2]} = \mathrm{g}\left(\mathrm{z}^{[2]}\right) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$
- Third layer
  - $z^{[3]} = \mathrm{W}^{[3]} x + \mathrm{b}^{[3]}$= (0,…0)
  - $a^{[3]} = \mathrm{g}\left(\mathrm{z}^{[3]}\right) = \left(\frac{1}{2}, \dots, \frac{1}{2}\right)$ does not depend on *x*
- Initialize with random values instead!

Backpropagation Intuition

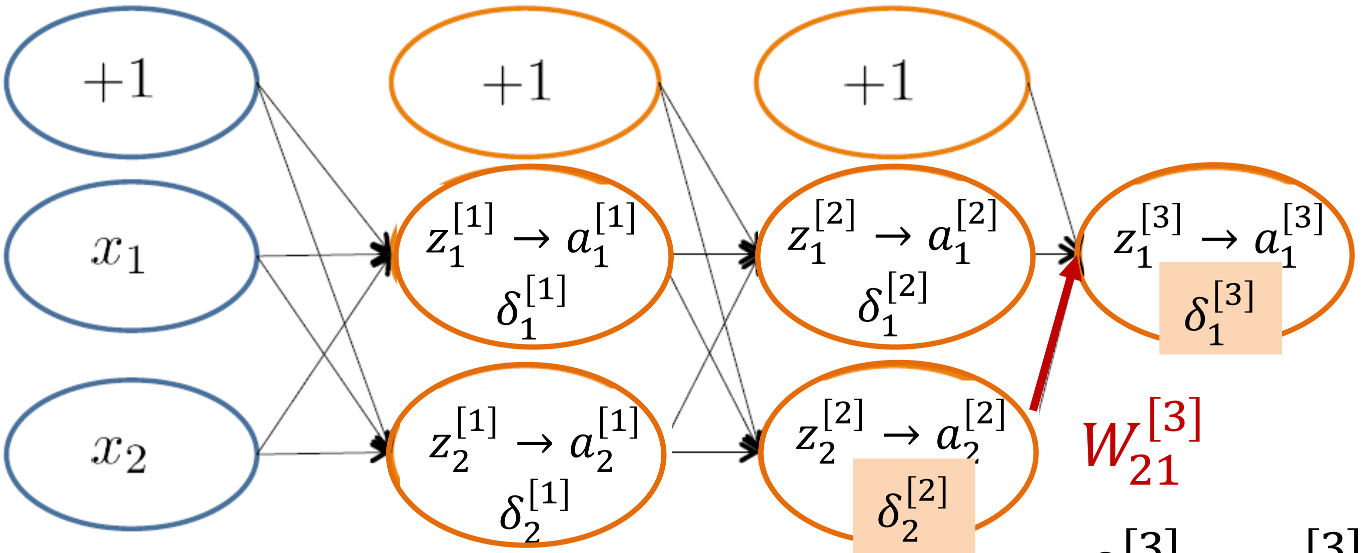


$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

where $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Backpropagation Intuition



$\delta_1^{[3]} \approx a_1^{[3]} - y$

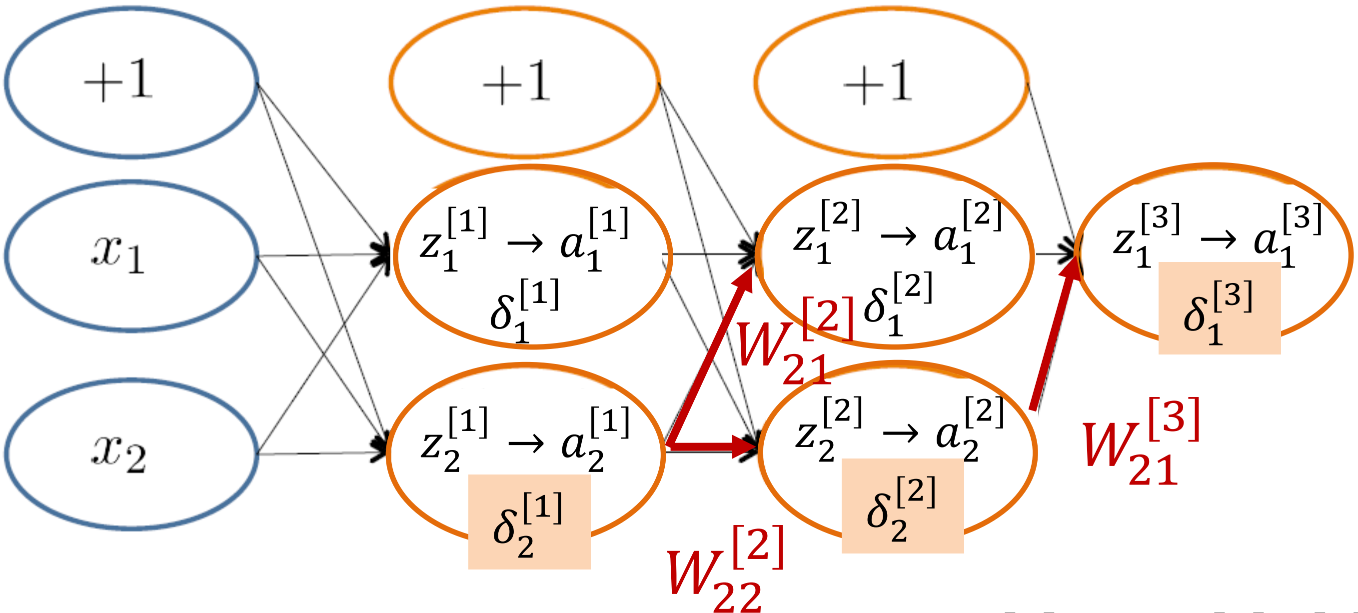
$\delta_2^{[2]} \approx \delta_1^{[3]} W_{21}^{[3]}$

$\delta_j^{(l)}$ = "error" of node j in layer l

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$$

Backpropagation Intuition



$$\delta_2^{[1]} \approx W_{21}^{[2]}\delta_1^{[2]} + W_{22}^{[2]}\delta_2^{[2]}$$

$\delta_j^{(l)}$ = "error" of node j in layer l

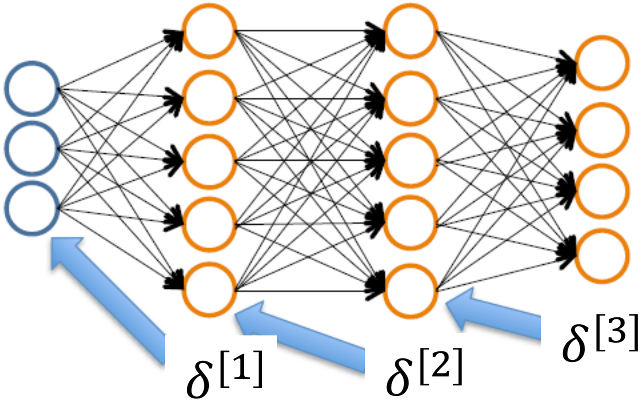
Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

$$\text{where } \text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i)\log(1 - h_\Theta(\mathbf{x}_i))$$

Backpropagation

Let $\delta_j^{(l)}$ = "error" of node j in layer l

$$L(y,\hat{y}) = -[(1-y)\log(1-\hat{y}) + y\log\hat{y}]$$



Definitions

- $z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}, a^{[\ell]} = g\left(z^{[\ell]}\right)$
- $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}}$; Output $\hat{y} = a^{[L]} = g\left(z^{[L]}\right)$

1. For last layer L: $\delta^{[L]} = \frac{\partial L(\hat{y},y)}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{[L]}} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}} g'\left(z^{[L]}\right)$
2. For layer ℓ: $\delta^{[\ell]} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell+1]}} \frac{\partial z^{[\ell+1]}}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial z^{[\ell]}} = \delta^{[\ell+1]} W^{[\ell+1]} g'(z^{[\ell]})$
3. Compute parameter gradients

- $\frac{\partial L(\hat{y},y)}{\partial W^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial W^{[\ell]}} = \delta^{[\ell]} a^{[\ell-1]T}$
- $\frac{\partial L(\hat{y},y)}{\partial b^{[\ell]}} = \frac{\partial L(\hat{y},y)}{\partial z^{[\ell]}} \frac{\partial z^{[\ell]}}{\partial b^{[\ell]}} = \delta^{[\ell]}$

Backpropagation

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)

For each training instance $(\mathbf{x}_i, y_i)$:

Set $\mathbf{a}^{(1)} = \mathbf{x}_i$

Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation

Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$

Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$

Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Average gradient is $\frac{\Delta_{ij}^{[\ell]}}{N}$

Training NN with Backpropagation

Given training set $(x_1, y_1), \dots, (x_N, y_N)$
Initialize all parameters $W^{[\ell]}, b^{[\ell]}$ randomly, for all layers ℓ
Loop

Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
For each training instance $(\mathbf{x}_i, y_i)$:
Set $\mathbf{a}^{(1)} = \mathbf{x}_i$
Compute $\{\mathbf{a}^{(2)}, \dots, \mathbf{a}^{(L)}\}$ via forward propagation
Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$
Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \dots, \boldsymbol{\delta}^{(2)}\}$
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

EPOCH

Update weights via gradient step

- $W_{ij}^{[\ell]} = W_{ij}^{[\ell]} - \alpha \frac{\Delta_{ij}^{[\ell]}}{N}$
- Similar for $b_{ij}^{[\ell]}$

Until weights converge or maximum number of epochs is reached

Training Neural Networks

- Randomly initialize weights
- Implement forward propagation to get prediction $\widehat{y_i}$ for any training instance x_i
- Compute loss function $L(\hat{y}_i, y_i)$
- Implement backpropagation to compute partial derivatives $\frac{\partial L(\hat{y}_i, y_i)}{\partial W^{[\ell]}}$ and $\frac{\partial L(\hat{y}_i, y_i)}{\partial b^{[\ell]}}$
- Use gradient descent with backpropagation to compute parameter values that optimize loss
- Can be applied to both feed-forward and convolutional nets

Materials

- Stanford tutorial on training Multi-Layer Neural Networks
  - http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/
- Notes on backpropagation by Andrew Ng
  - http://cs229.stanford.edu/notes-spring2019/backprop.pdf
- Deep learning notes by Andrew Ng
  - http://cs229.stanford.edu/notes2020spring/cs229-notes-deep_learning.pdf

Acknowledgements

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