Neural Networks

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence from 10s: state-of-the-art techniques for many applications:
  - Computer Vision
  - Natural language processing
  - Speech recognition
  - Decision-making / control problems (AlphaGo, Dota, robots)
- Limited theory:
  - Non-convexity
  - Model are complex but generalization error is small

Neural Networks

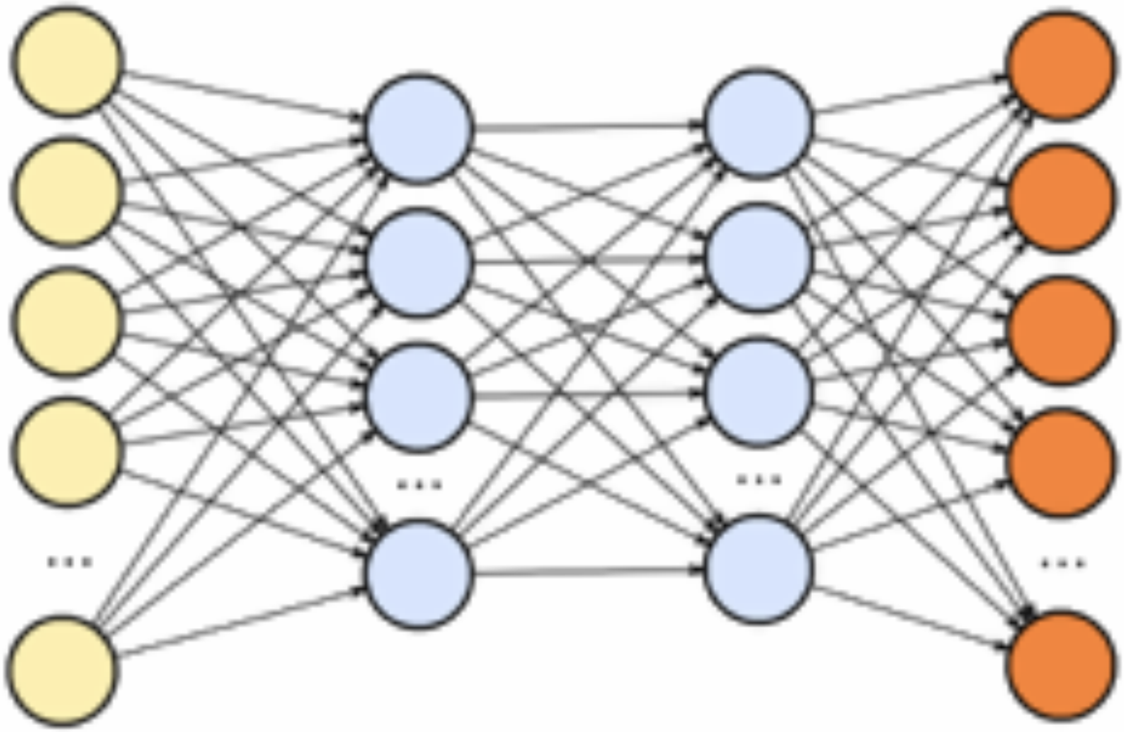
This week:

1. Definitions of neural networks

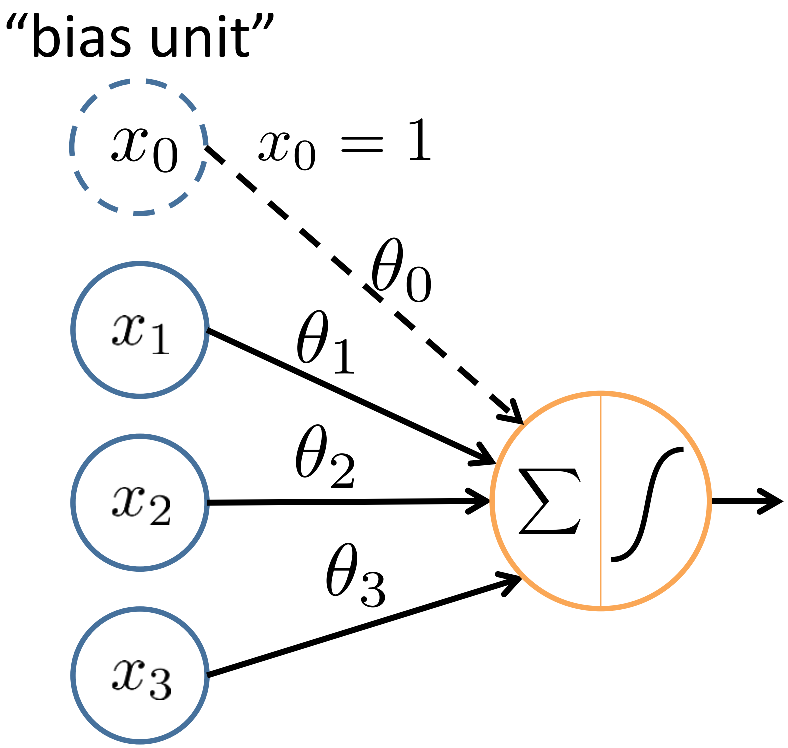
2. Training neural networks:
   1. Algorithm: back propagation
   2. Putting it to work

3. Neural network architecture design:
   1. Convolutional neural network

Neural Networks



Single Node



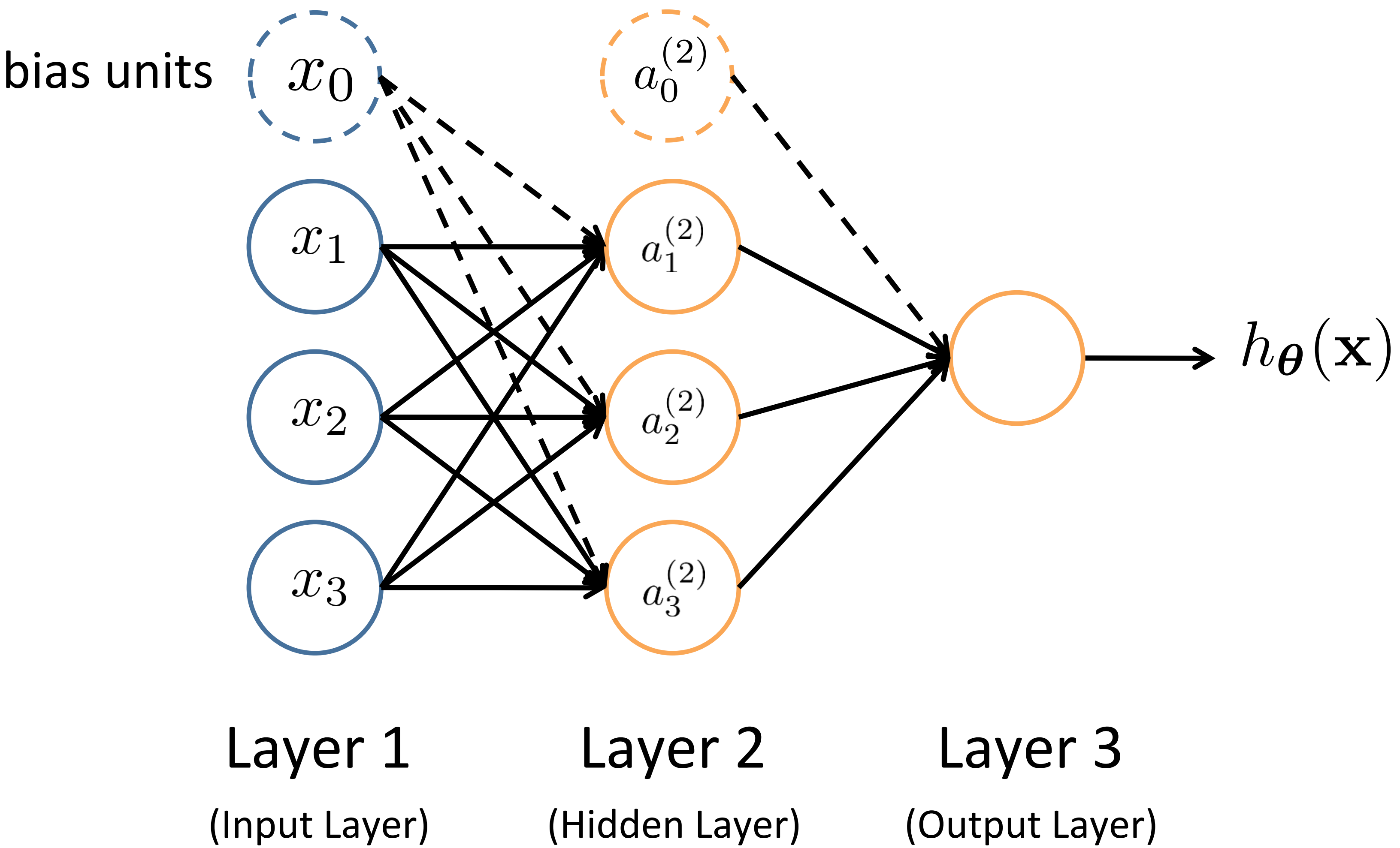
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

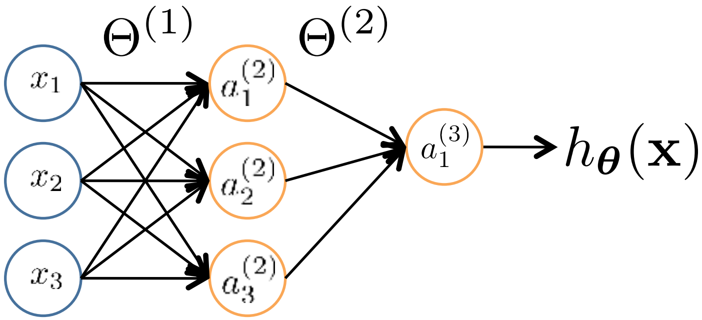
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\intercal}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\intercal}\mathbf{x}}}$$

Binary Logistic Regression

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

Neural Network





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

$\Theta^{(j)}$ = weight matrix stores parameters from layer j to layer $j + 1$

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

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

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

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$

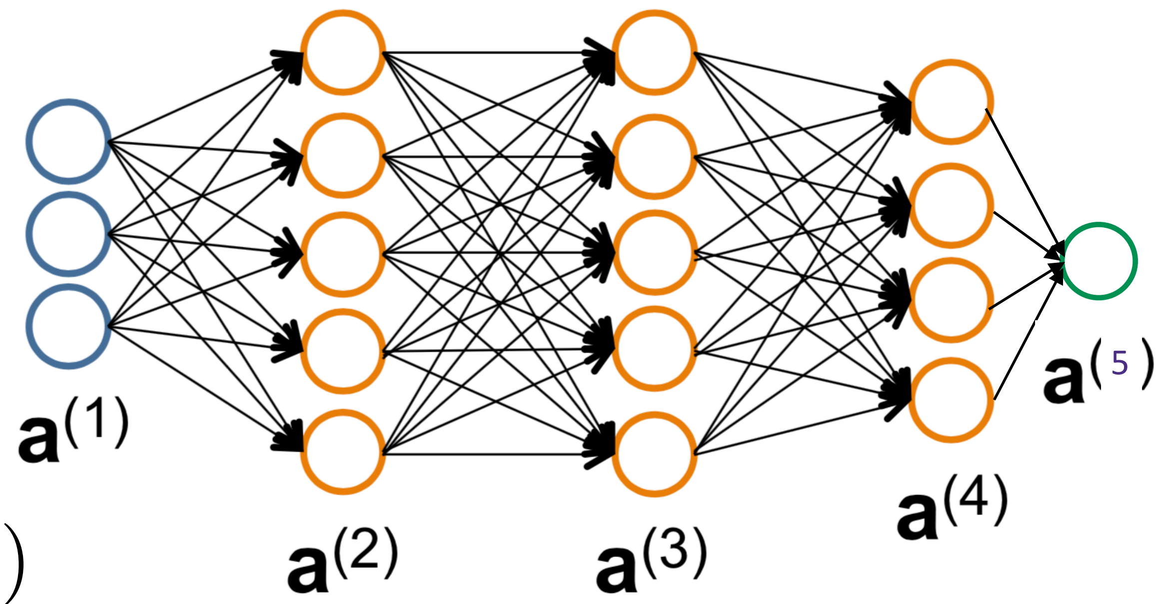
$$a^{(2)} = g(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

$$a^{(l+1)} = g(\Theta^{(l)} a^{(l)})$$

$$\vdots$$

$$\hat{y} = g(\Theta^{(L)} a^{(L)})$$



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

$$g(z) = \frac{1}{1 + e^{-z}}$$

Binary
Logistic
Regression

Multi-layer Neural Network - Binary Classification

$$a^{(1)} = x$$

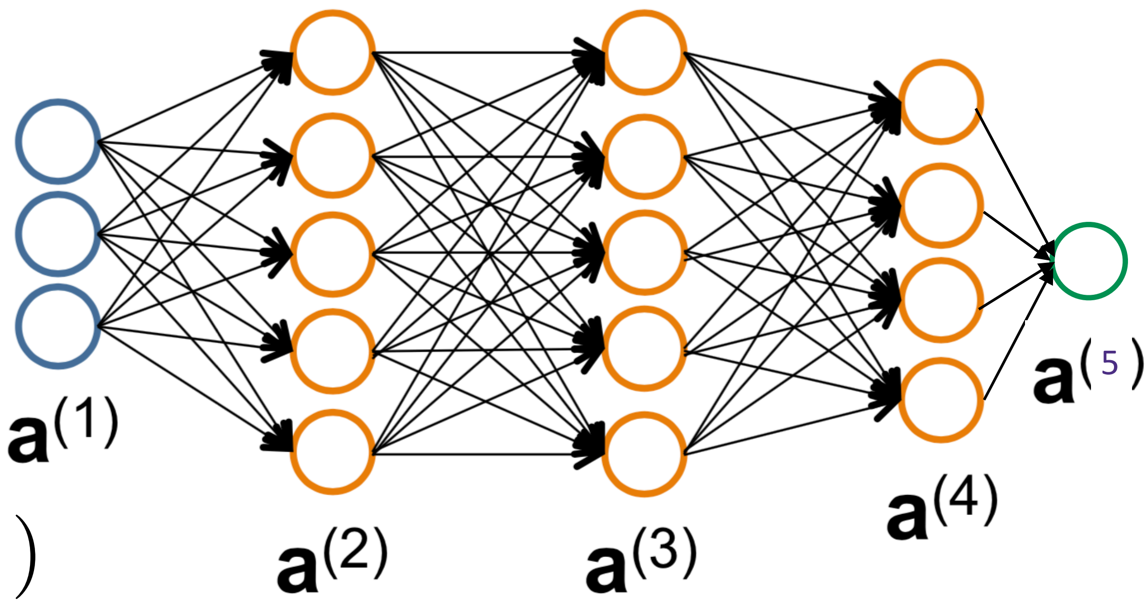
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

$$\vdots$$

$$\widehat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \widehat{y}) = y\log(\widehat{y}) + (1-y)\log(1-\widehat{y})$$

$$\sigma(z) = \max\{0, z\} \quad g(z) = \frac{1}{1+e^{-z}}$$

Binary Logistic Regression

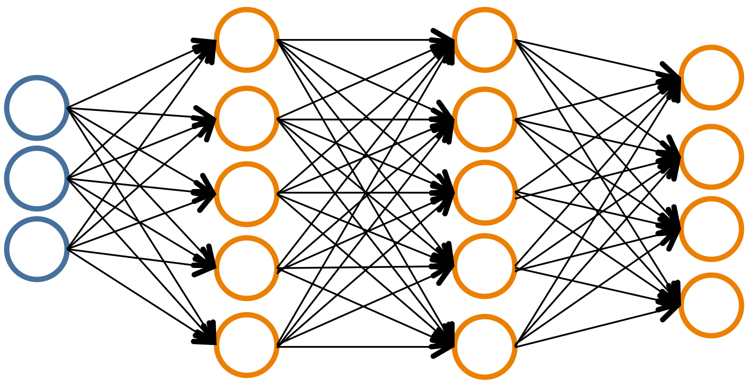
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



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

Multi-class
Logistic
Regression

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

Multi-layer Neural Network - Regression

$$a^{(1)} = x$$

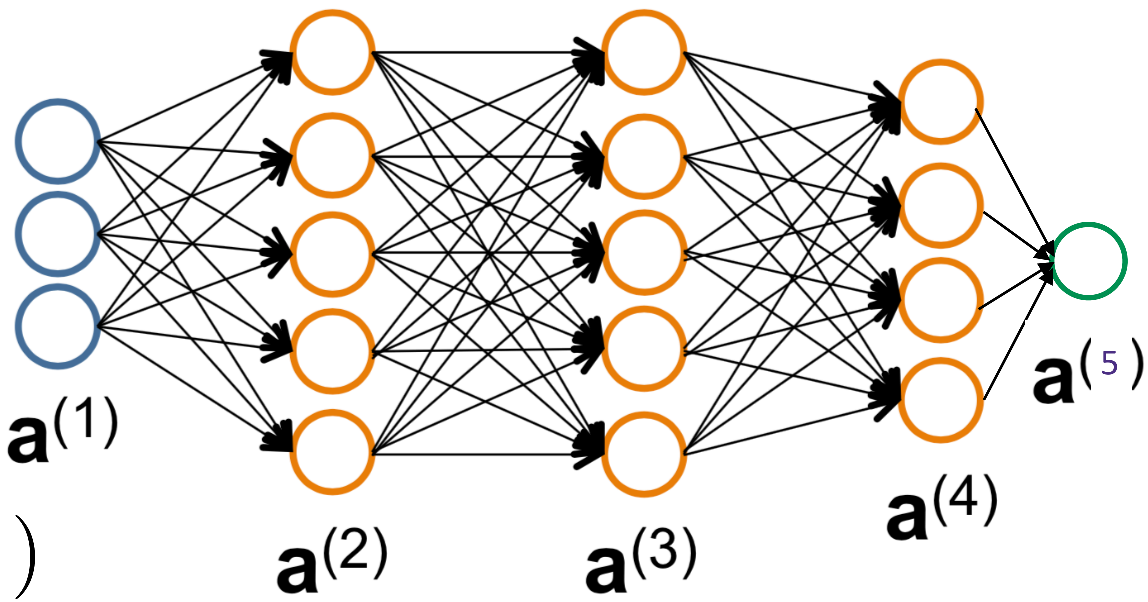
$$a^{(2)} = \sigma(\Theta^{(1)} a^{(1)})$$

$$\vdots$$

$$a^{(l+1)} = \sigma(\Theta^{(l)} a^{(l)})$$

$$\vdots$$

$$\widehat{y} = \Theta^{(L)} a^{(L)}$$



$$L(y, \widehat{y}) = (y - \widehat{y})^2$$

$$\sigma(z) = \max\{0, z\}$$

Regression

Neural Networks are arbitrary function approximators

Theorem 10 (Two-Layer Networks are Universal Function Approximators). *Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer neural network $\hat{F}$ with a finite number of hidden units that approximate F arbitrarily well. Namely, for all $\boldsymbol{x}$ in the domain of F, $\left|F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x})\right| < \epsilon$.*

Cybenko, Hornik (theorem reproduced from CIML, Ch. 10)

Training Neural Networks

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

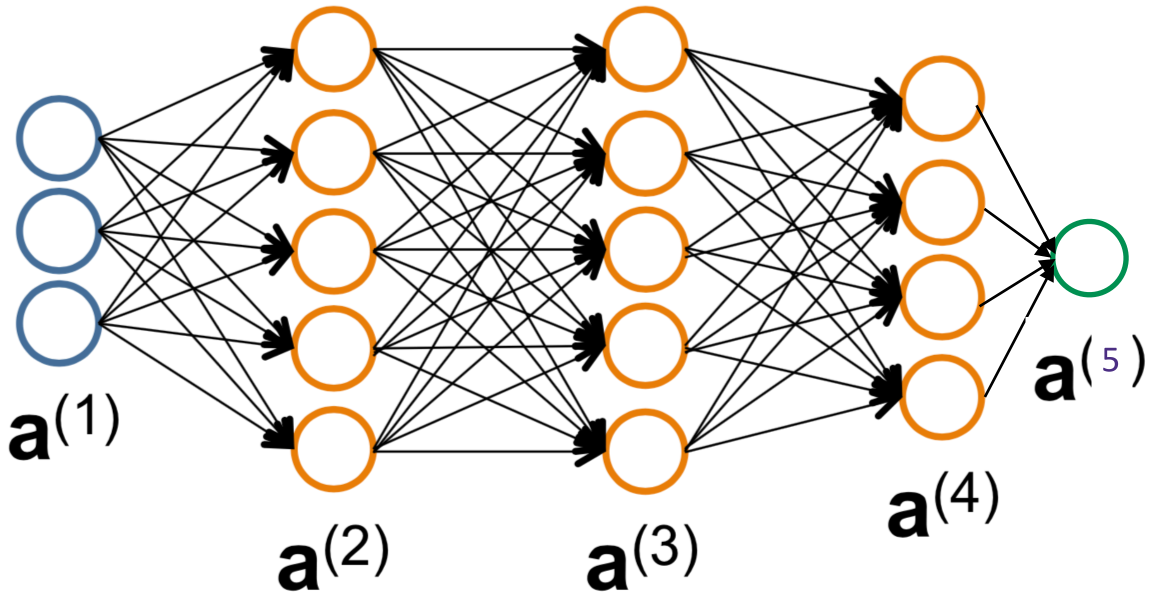
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = g(\Theta^{(L)} a^{(L)})$$



$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \qquad \forall l$

Gradient Descent: $\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \qquad \forall l$

Seems simple enough, why are packages like PyTorch, Tensorflow, Theano, Cafe, MxNet synonymous with deep learning?

1. Automatic differentiation

Gradient Descent: $$\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \qquad \forall l$$

Seems simple enough, why are packages like PyTorch, Tensorflow, Theano, Cafe, MxNet synonymous with deep learning?

1. Automatic differentiation

2. Convenient libraries

Gradient Descent:

Seems simple enough, why are libraries like PyTorch, Tensorflow, Theano, Cafe, MxNet s

1. Automatic differ

2. Convenient libra

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 3x3 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 3)
        self.conv2 = nn.Conv2d(6, 16, 3)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 6 * 6, 120)  # 6*6 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

Gradient Descent: $$\Theta^{(l)} \leftarrow \Theta^{(l)} - \eta \nabla_{\Theta^{(l)}} L(y, \widehat{y}) \qquad \forall l$$

Seems simple enough, why are packages like PyTorch, Tensorflow, Theano, Cafe, MxNet synonymous with deep learning?

1. Automatic differentiation

2. Convenient libraries

3. GPU support

Common training issues

Neural networks are **non-convex**

- For large networks, **gradients** can **blow up** or **go to zero**. This can be helped by **batchnorm** or ResNet architecture

- **Stepsize**, **batchsize**, **momentum** all have large impact on optimizing the training error *and* generalization performance

- Fancier alternatives to SGD (Adagrad, Adam, LAMB, etc.) can significantly improve training

-Overfitting is common and not undesirable: typical to achieve 100% training accuracy even if test accuracy is just 80%

- Making the network *bigger* may make training *faster!*

Common training issues

Training is too slow:

- Use larger step sizes, develop step size reduction schedule
- Use GPU resources
- Change batch size
- Use momentum and more exotic optimizers (e.g., Adam)
- Apply batch normalization
- Make network larger or smaller (# layers, # filters per layer, etc.)

Test accuracy is low

- Try modifying all of the above, plus changing other hyperparameters

Intuition

https://playground.tensorflow.org/

Back Propagation

Forward Propagation

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

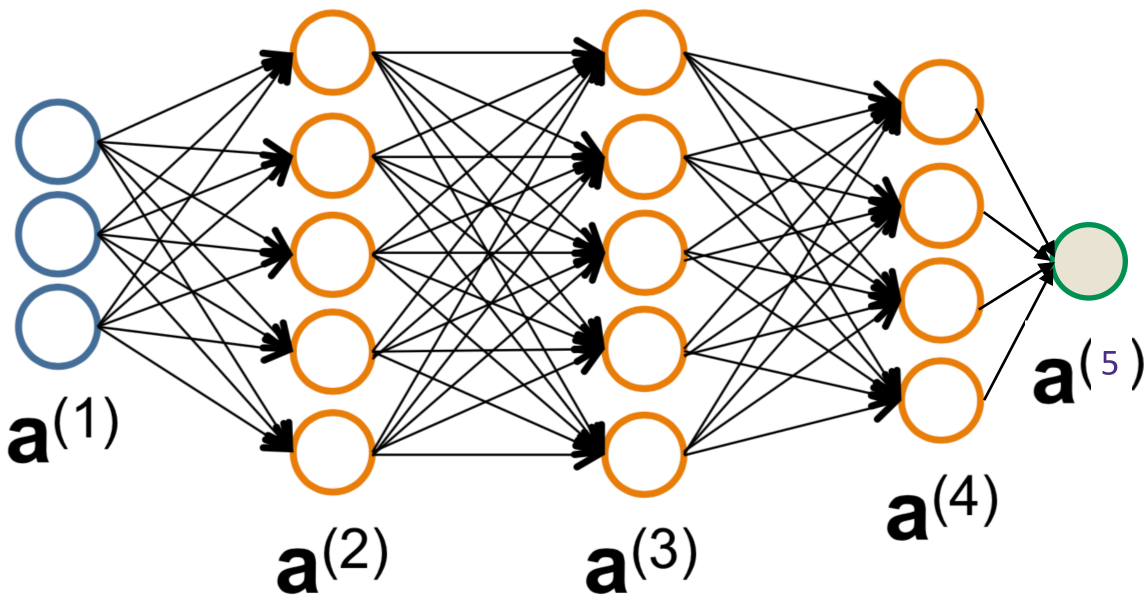
$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$



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

$$g(z) = \frac{1}{1 + e^{-z}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}}$$

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

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

Train by Stochastic Gradient Descent:

$$\Theta_{i,j}^{(l)} \leftarrow \Theta_{i,j}^{(l)} - \eta \frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \widehat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\boxed{\frac{\partial L(y, \widehat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}}$$

$$\delta_i^{(l)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \widehat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$

$$= \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}\, g'(z_i^{(l)})$$

$$= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta^{(l)}_{k,i}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1 - y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g\left(z^{(l)}\right)$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\widehat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \widehat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\begin{aligned}
\delta_i^{(L+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(L+1)}} &= \frac{\partial}{\partial z_i^{(L+1)}} \left[y \log(g(z^{(L+1)})) + (1-y)\log(1 - g(z^{(L+1)})) \right] \\
&= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1-y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \\
&= y - g(z^{(L+1)}) = y - a^{(L+1)}
\end{aligned}$$

$$L(y, \widehat{y}) = y \log(\widehat{y}) + (1-y)\log(1 - \widehat{y})$$

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \widehat{y})}{\partial z_i^{(l+1)}}$$

Backprop

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

$$\vdots$$

$$a^{(l)} = g(z^{(l)})$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)}$$

$$\delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

$$\delta^{(L+1)} = y - a^{(L+1)}$$

Recursive Algorithm!

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

$$g(z) = \frac{1}{1 + e^{-z}} \qquad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}$$

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

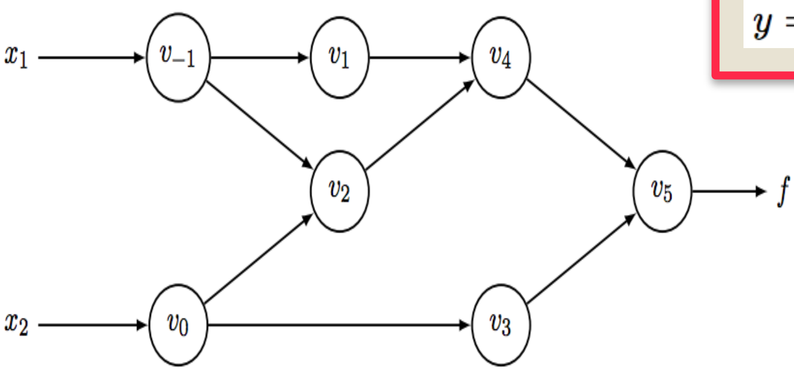
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$

D$^{(l)}$ is the matrix of partial derivatives of $J(\Theta)$

Autodiff

Backprop for this simple network architecture is a special case of ***reverse-mode auto-differentiation:***



$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$

Forward Primal Trace

$v_{-1} = x_1$		$= 2$
$v_0 = x_2$		$= 5$
$v_1 = \ln v_{-1}$		$= \ln 2$
$v_2 = v_{-1} \times v_0$		$= 2 \times 5$
$v_3 = \sin v_0$		$= \sin 5$
$v_4 = v_1 + v_2$		$= 0.693 + 10$
$v_5 = v_4 - v_3$		$= 10.693 + 0.959$
$y = v_5$		$= 11.652$

Reverse Adjoint (Derivative) Trace

$\bar{\boldsymbol{x}}_{\mathbf{1}} = \bar{\boldsymbol{v}}_{\mathbf{-1}}$			$= \mathbf{5.5}$
$\bar{\boldsymbol{x}}_{\mathbf{2}} = \bar{\boldsymbol{v}}_{\mathbf{0}}$			$= \mathbf{1.716}$
$\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$		$= 5.5$
$\bar{v}_0 = \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$		$= 1.716$
$\bar{v}_{-1} = \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$		$= 5$
$\bar{v}_0 = \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$		$= -0.284$
$\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$		$= 1$
$\bar{v}_1 = \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$		$= 1$
$\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$		$= -1$
$\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$		$= 1$
$\bar{v}_5 = \bar{y}$	$= 1$		

This is the special sauce in Tensorflow, PyTorch, Theano, …

Feature extraction given Image data

Multi-layer Neural Network

$$a^{(1)} = x$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$

$$a^{(2)} = g\left(z^{(2)}\right)$$

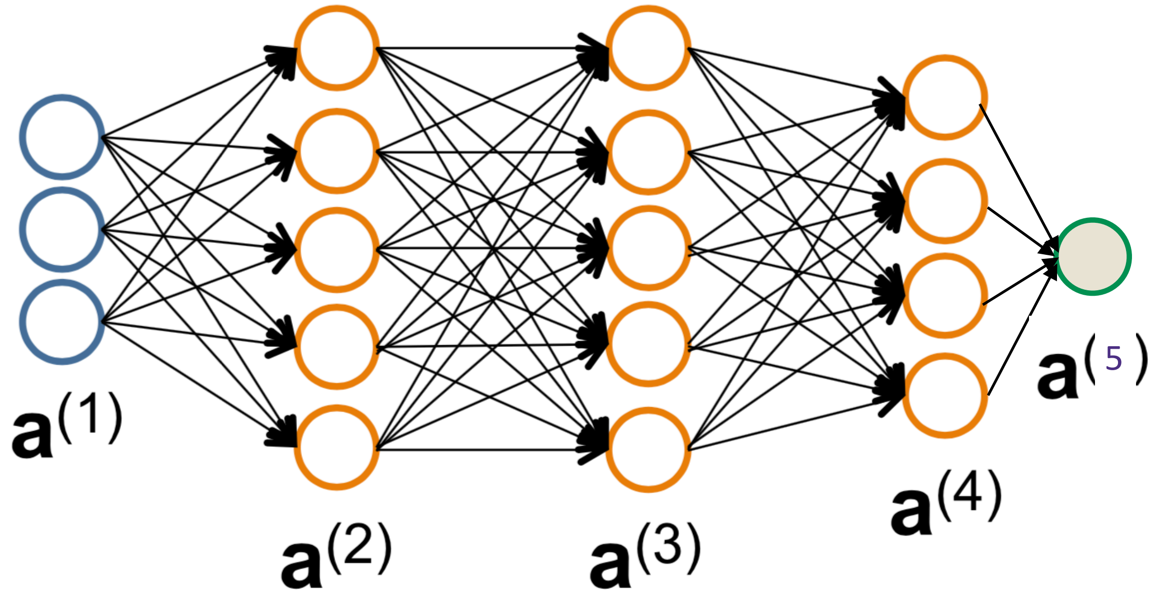
$$\vdots$$

$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$

$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$

$$\vdots$$

$$\hat{y} = a^{(L+1)}$$



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

$$g(z) = \frac{1}{1+e^{-z}}$$

Binary
Logistic
Regression

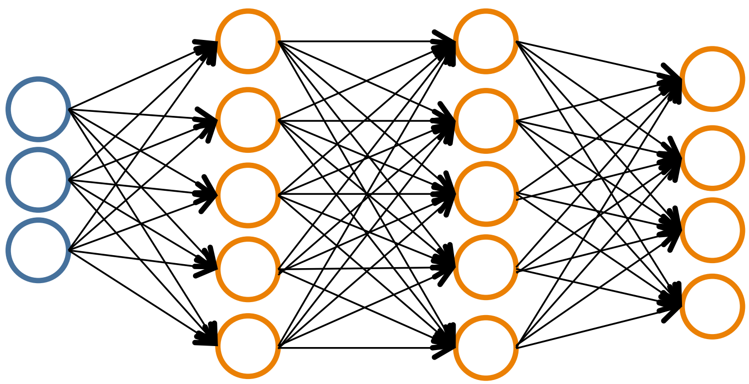
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



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

Multi-class
Logistic
Regression

We want:

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

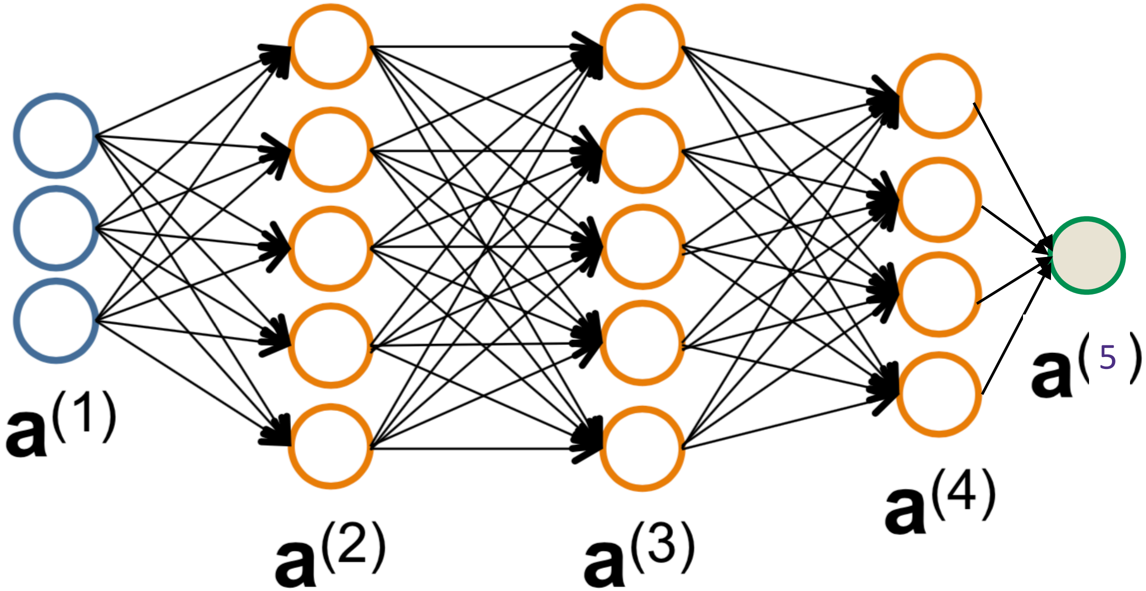
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$
when car

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

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

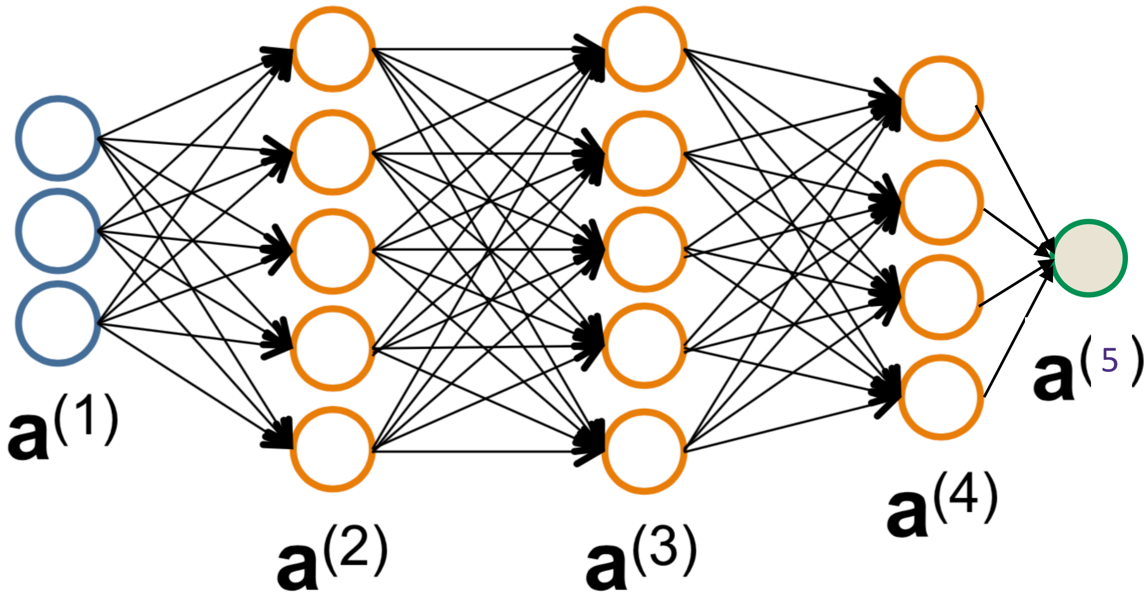
Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.



Neural Network Architecture

The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by **allowable edges**.



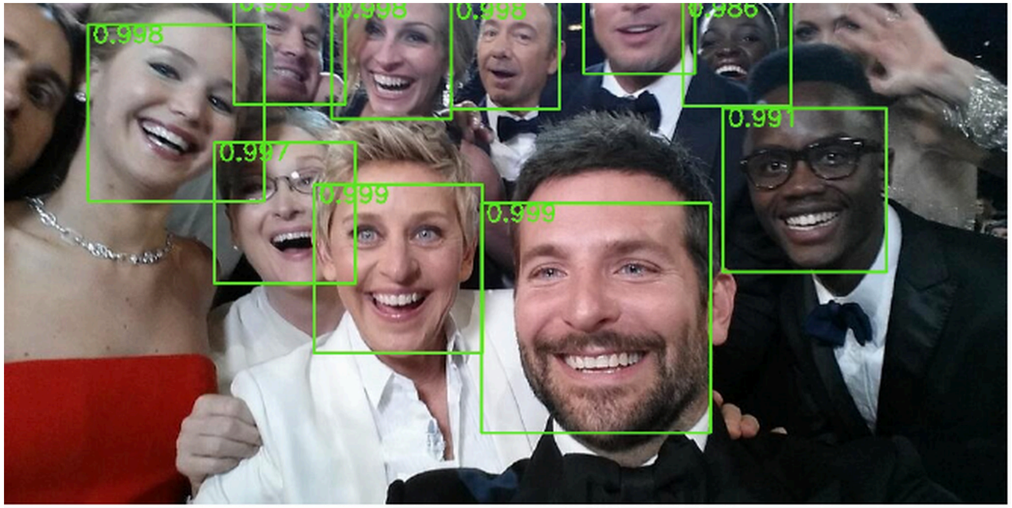
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

$$\mathbf{a}^{(k+1)} = g(\Theta \mathbf{a}^{(k)}) \quad \text{for any } \Theta \in \mathbb{R}^{n_{k+1} \times n_k}$$

A lot of parameters!! $n_1 n_2 + n_2 n_3 + \cdots + n_L n_{L+1}$

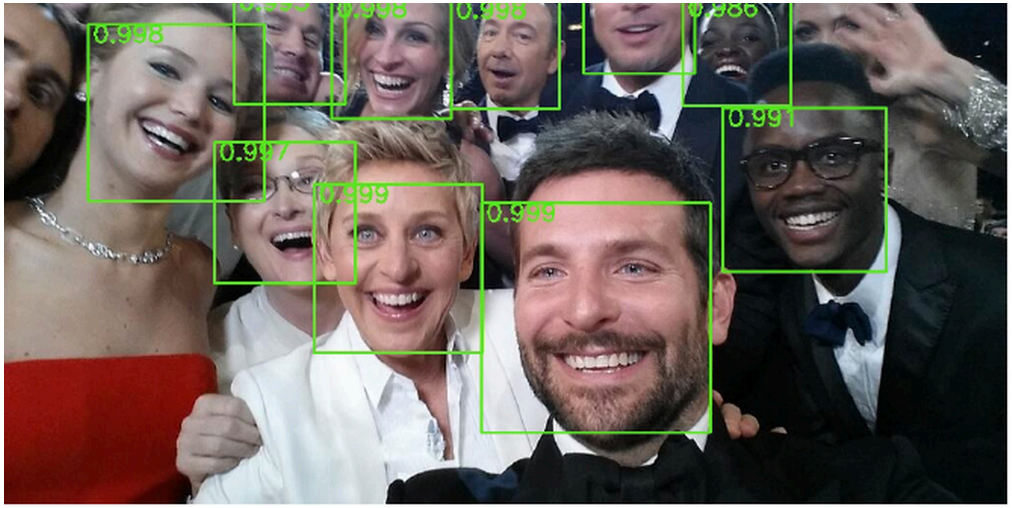
Neural Network Architecture

Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.

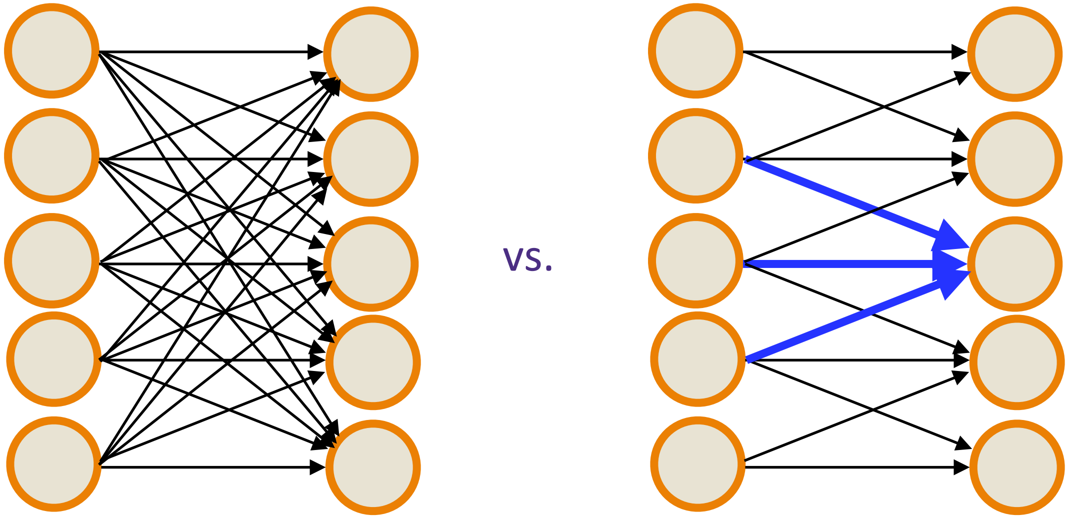


Neural Network Architecture

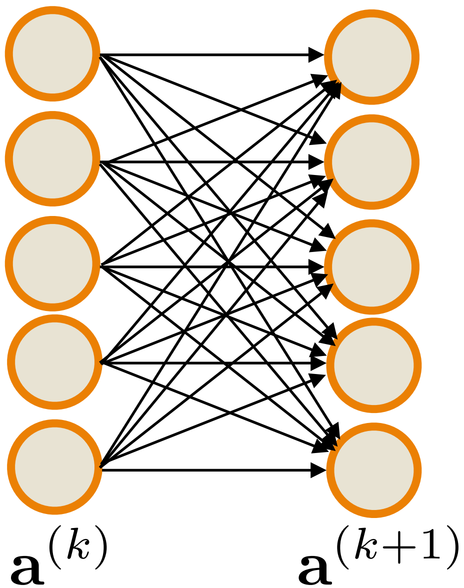
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



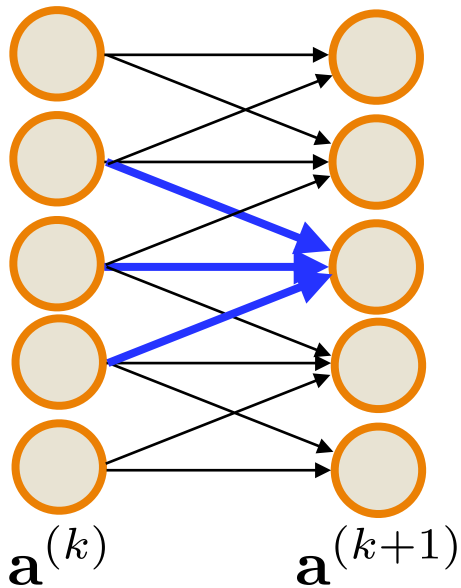
Similarly, to identify edges or other local structure, it makes sense to only look at **local information**



Neural Network Architecture



vs.

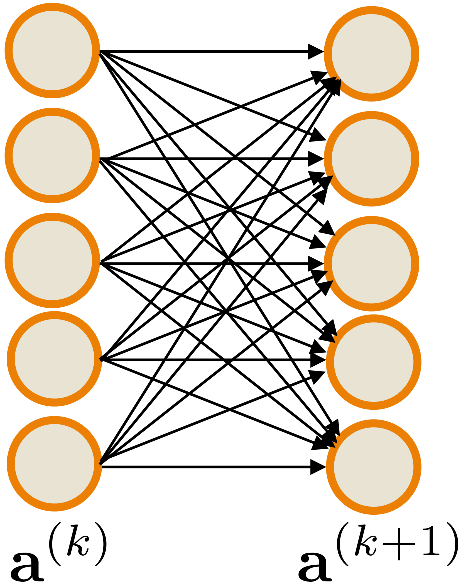


$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \qquad \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

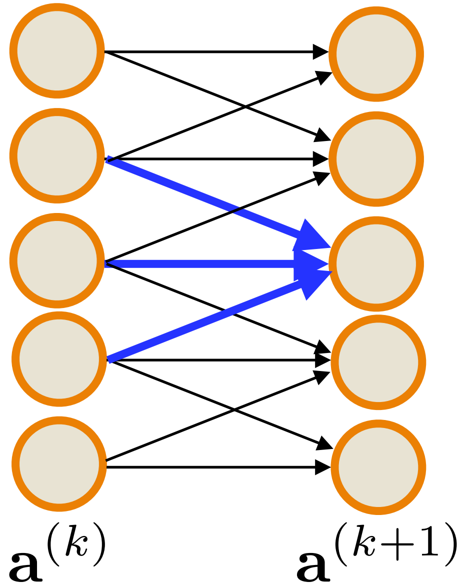
Parameters: n^2 $\qquad 3n-2$

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j}\mathbf{a}_j^{(k)}\right)$$

Neural Network Architecture



vs.



Mirror/share local weights everywhere (e.g., structure equally likely to be anywhere in image)

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \quad \begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & 0 & 0 & 0 \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & 0 & 0 \\ 0 & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 \\ 0 & 0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ 0 & 0 & 0 & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix} \quad \begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

Parameters: n^2 $\quad 3n-2$ $\quad 3$

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j}\mathbf{a}_j^{(k)}\right) \qquad \mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)}\right)$$

Neural Network Architecture

Fully Connected (FC) Layer

$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} \\ \Theta_{4,0} & \Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} \end{bmatrix}$$

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{n-1} \Theta_{i,j}\mathbf{a}_j^{(k)}\right)$$

Convolutional (CONV) Layer (1 filter)

$$\begin{bmatrix} \theta_1 & \theta_2 & 0 & 0 & 0 \\ \theta_0 & \theta_1 & \theta_2 & 0 & 0 \\ 0 & \theta_0 & \theta_1 & \theta_2 & 0 \\ 0 & 0 & \theta_0 & \theta_1 & \theta_2 \\ 0 & 0 & 0 & \theta_0 & \theta_1 \end{bmatrix}$$

m=3

$$\mathbf{a}_i^{(k+1)} = g\left(\sum_{j=0}^{m-1} \theta_j \mathbf{a}_{i+j}^{(k)}\right) = g([\theta * \mathbf{a}^{(k)}]_i)$$

Convolution*

$\theta = (\theta_0, \ldots, \theta_{m-1}) \in \mathbb{R}^m$ is referred to as a “filter”

Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

1	1	1	0	0

Input $x \in \mathbb{R}^n$

1	0	1

Filter $\theta \in \mathbb{R}^m$

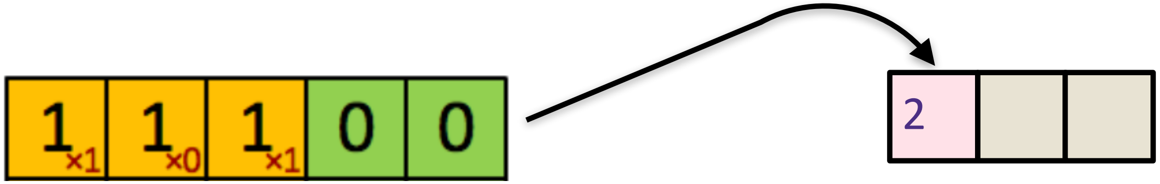
Output $\theta * x$

Example (1d convolution)

Input $x \in \mathbb{R}^n$

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

Filter $\theta \in \mathbb{R}^m$



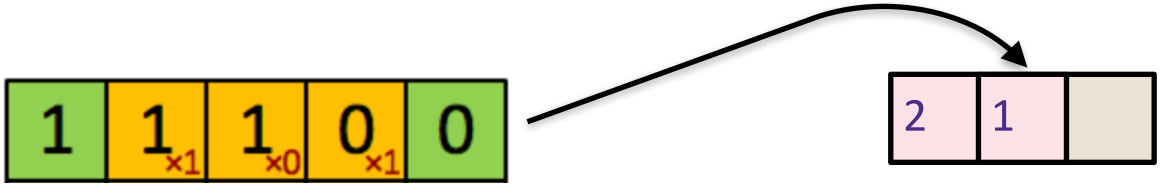
Output $\theta * x$

Example (1d convolution)

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$

Input $x \in \mathbb{R}^n$

Filter $\theta \in \mathbb{R}^m$

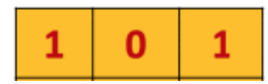


Output $\theta * x$

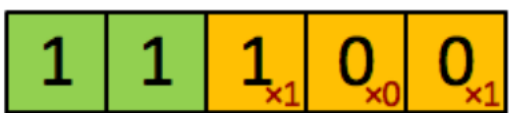
Example (1d convolution)

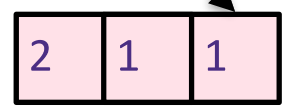
Input $x \in \mathbb{R}^n$

$$(\theta * x)_i = \sum_{j=0}^{m-1} \theta_j x_{i+j}$$



Filter $\theta \in \mathbb{R}^m$

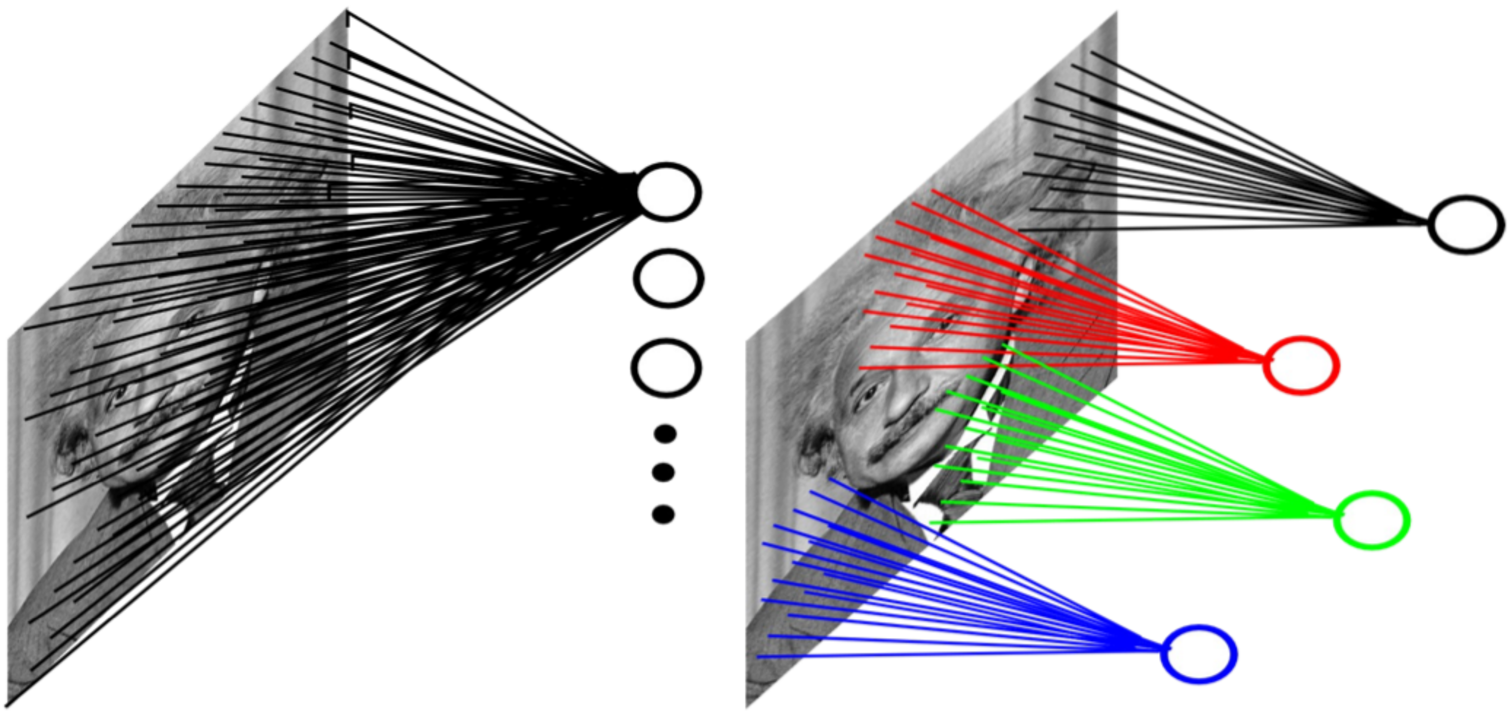




Output $\theta * x$

2d Convolution Layer

- **Example: 200x200 image**
  - Fully-connected, 400,000 hidden units = 16 billion parameters
  - Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
  - Local connections capture local dependencies



Convolution of images (2d convolution)

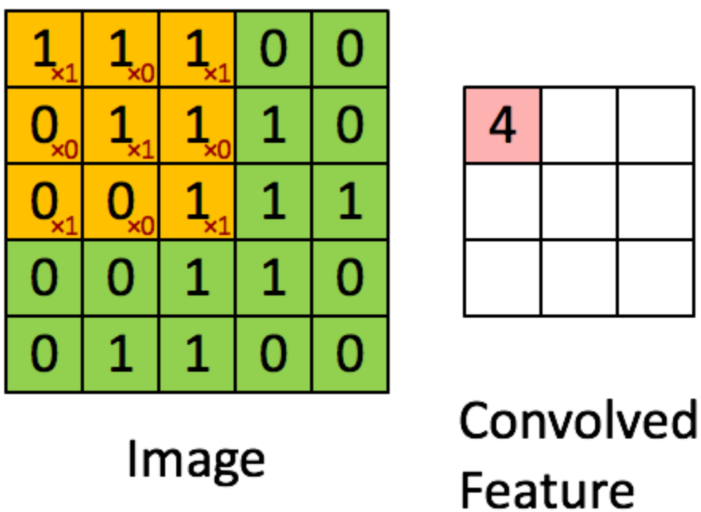
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image I

1	0	1
0	1	0
1	0	1

Filter K

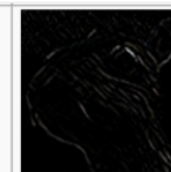
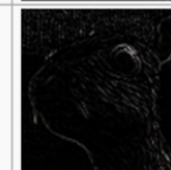
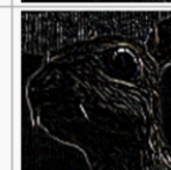
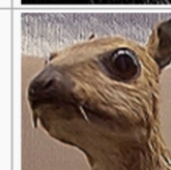
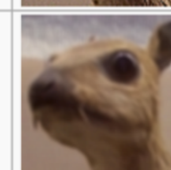
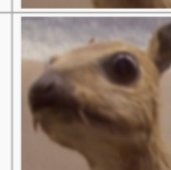


$I * K$

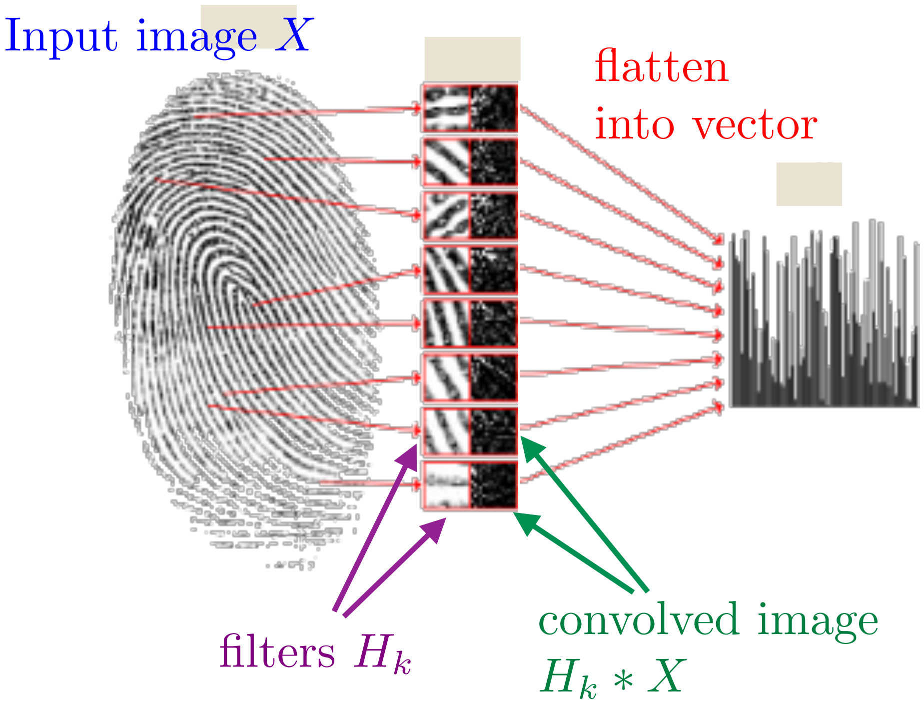
Convolution of images

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Image I

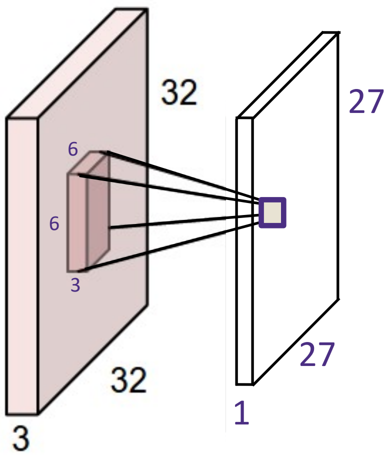
Operation	Filter K	Convolved Image $I * K$
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolution of images



$$\begin{bmatrix} \mathrm{vec}(H_1 * X) \\ \mathrm{vec}(H_2 * X) \\ \vdots \end{bmatrix}$$

3d Convolution

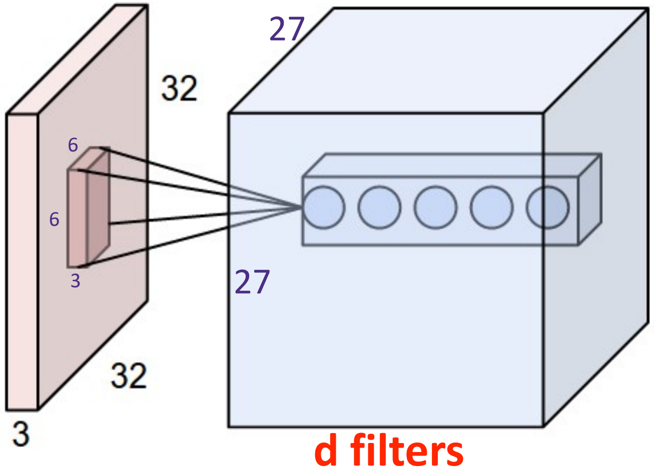


$$\Theta \in \mathbb{R}^{m \times m \times r}$$

$$x \in \mathbb{R}^{n \times n \times r}$$

$$(\Theta * x)_{s,t} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{k=0}^{r-1} \Theta_{i,j,k} x_{s+i,t+j}$$

Stacking convolved images



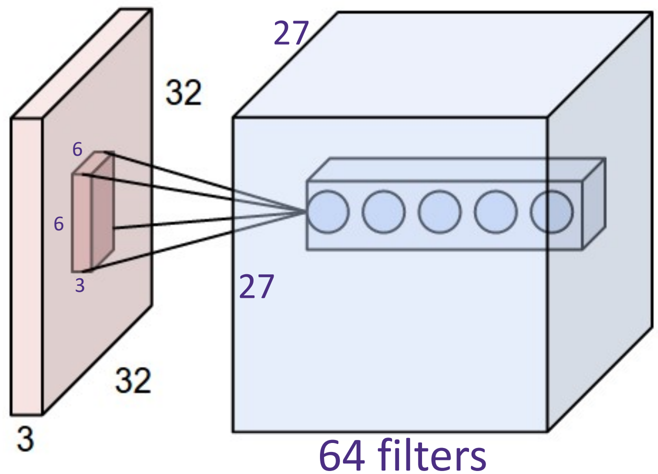
Repeat with d filters!

$$\Theta \in \mathbb{R}^{m \times m \times r}$$

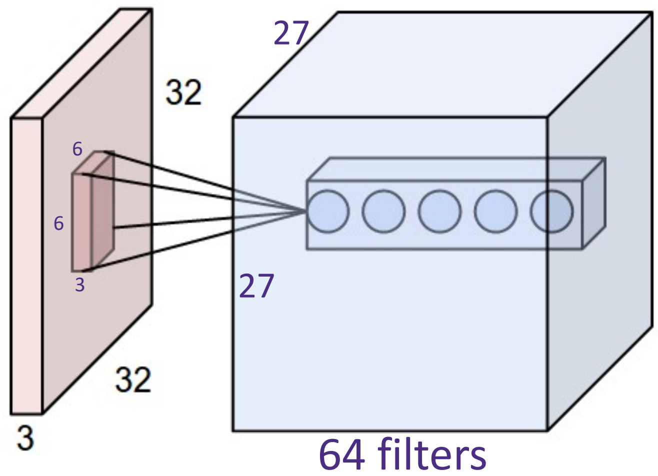
$$x \in \mathbb{R}^{n \times n \times r}$$

$$(\Theta * x)_{s,t} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \sum_{k=0}^{r-1} \Theta_{i,j,k} x_{s+i,t+j}$$

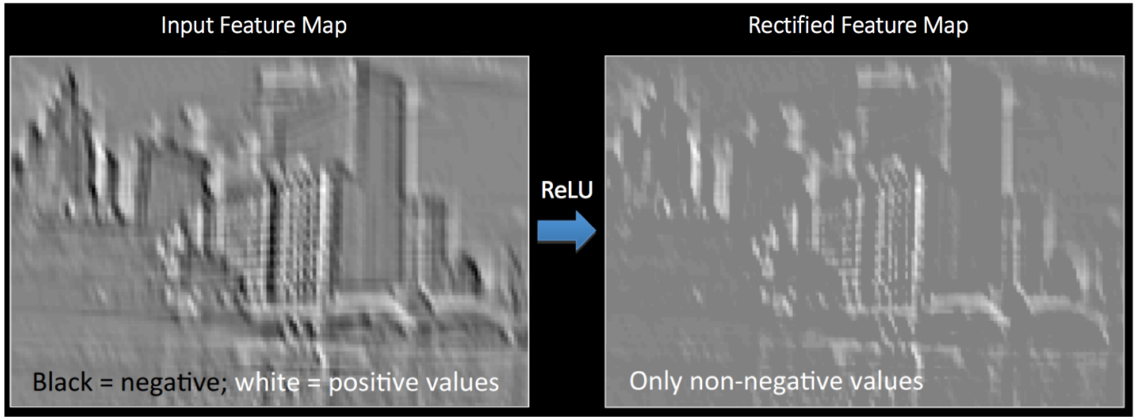
Stacking convolved images



Stacking convolved images



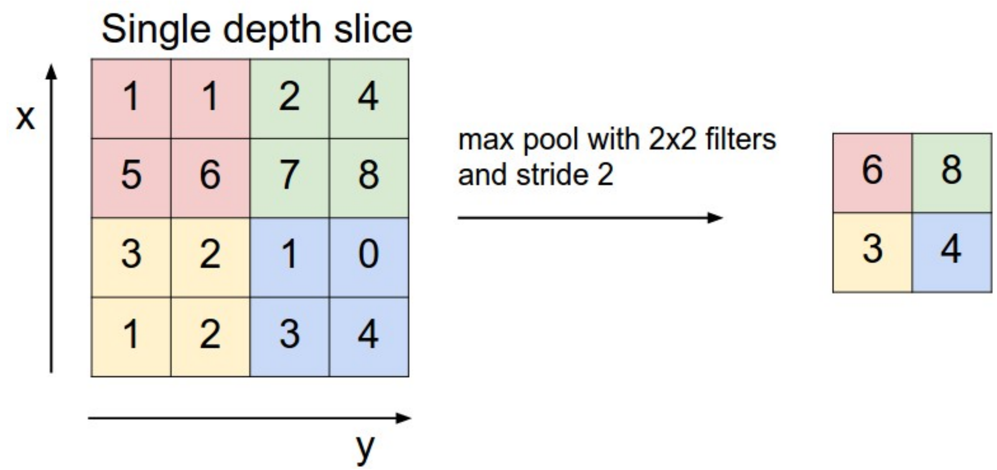
Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)

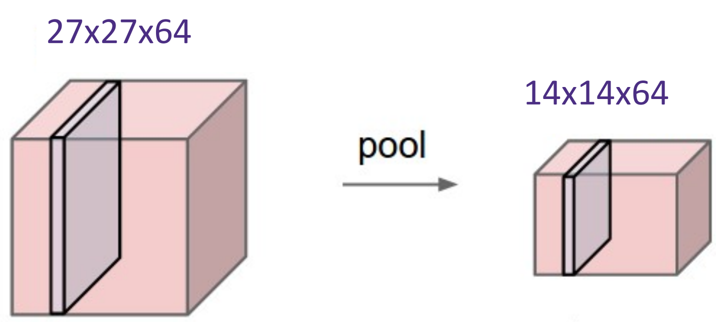


Other choices: sigmoid, arctan

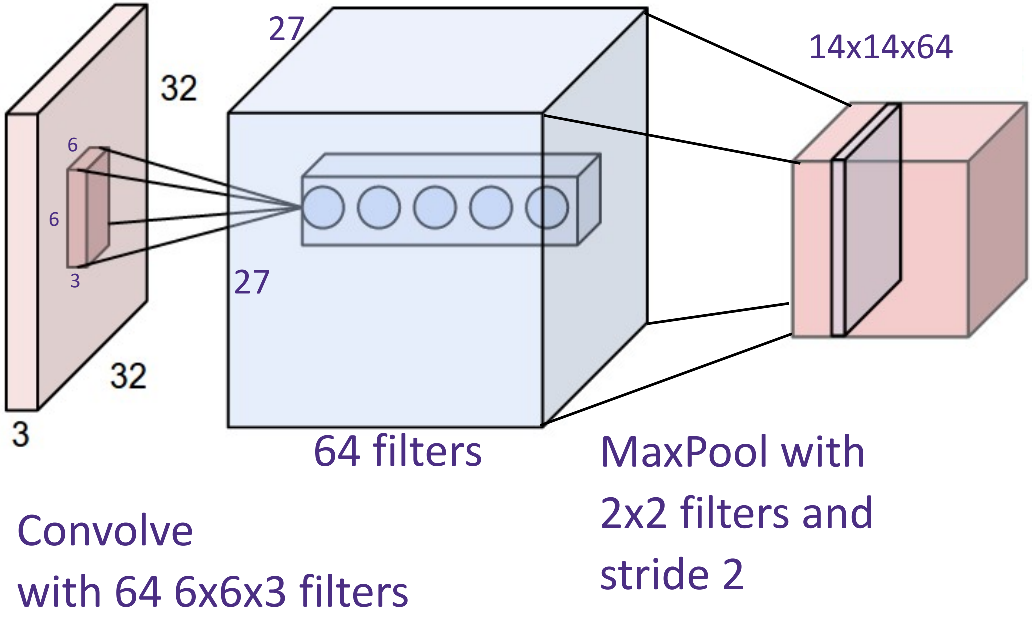
Pooling

Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”

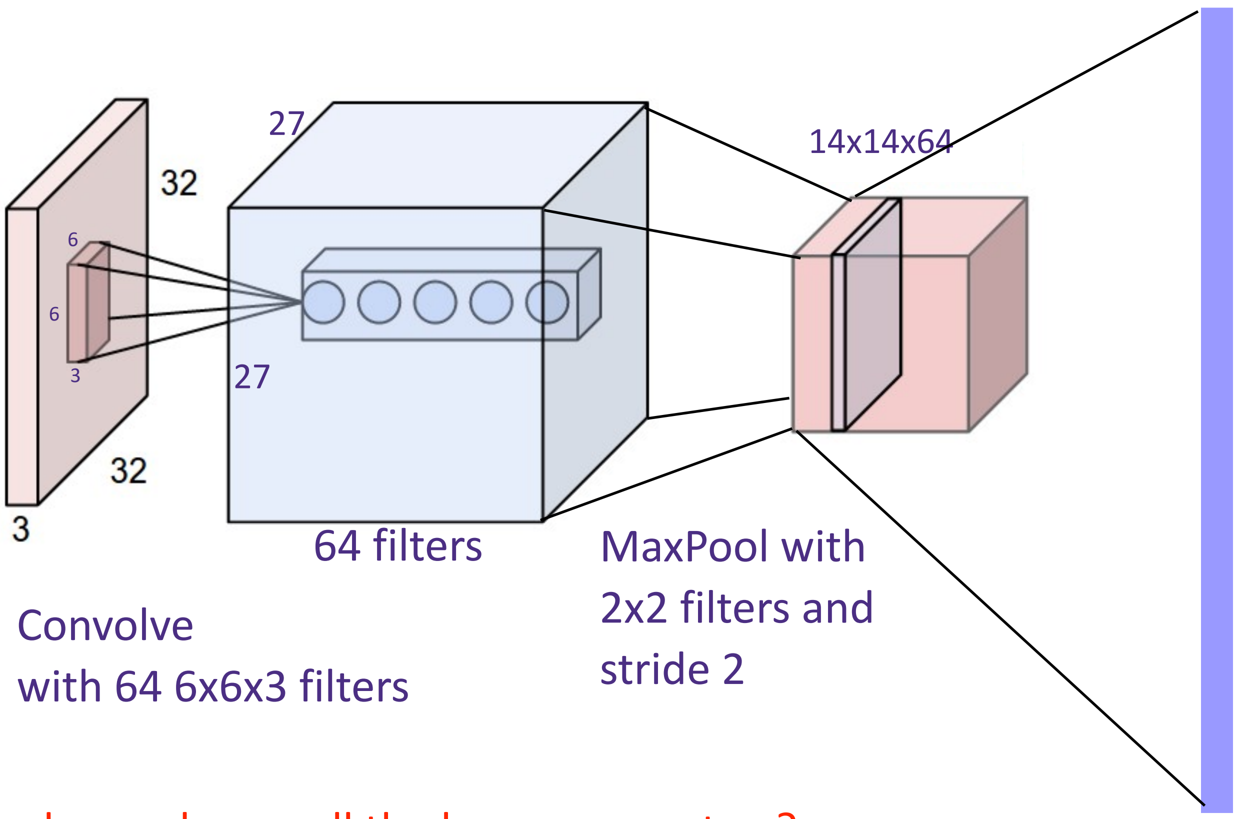




Pooling Convolution layer



Simplest feature pipeline



Convolve
with 64 6x6x3 filters

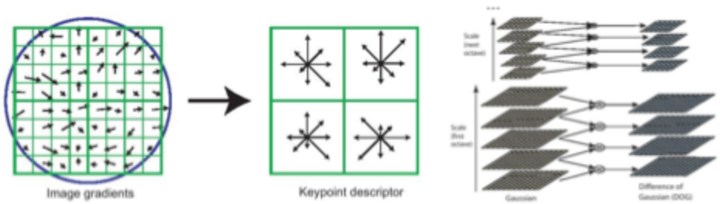
MaxPool with
2x2 filters and
stride 2

Flatten into a single
vector of size
14*14*64=12544

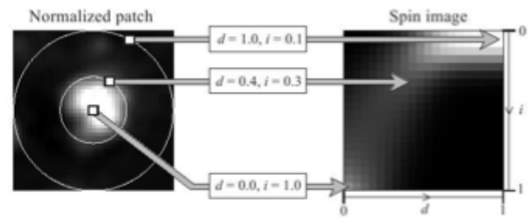
How do we choose all the hyperparameters?

How do we choose the filters?
- Hand crafted (digital signal processing, c.f. wavelets)
- Learn them (deep learning)

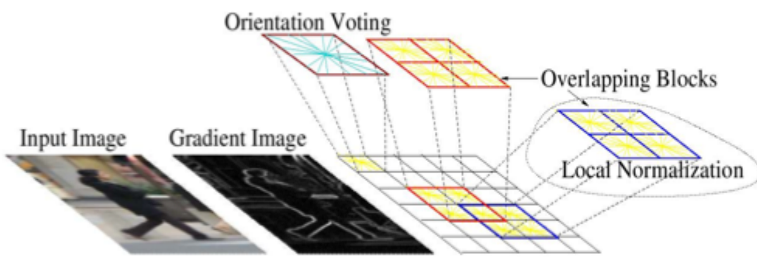
Some hand-created image features



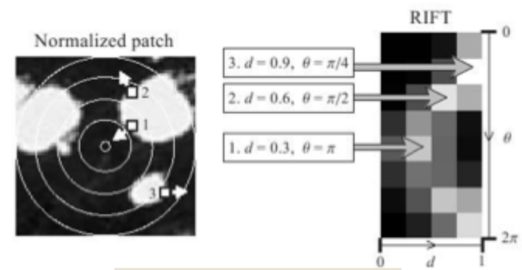
SIFT



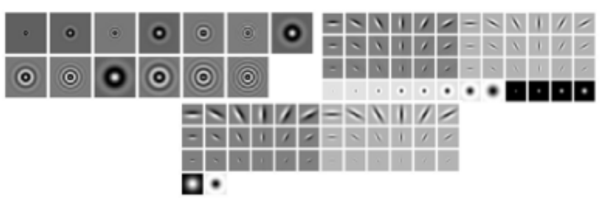
Spin Image



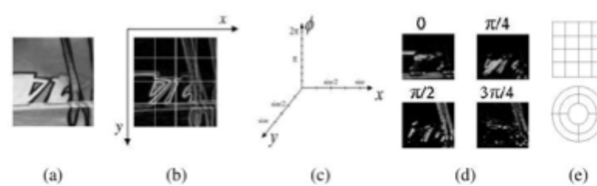
HoG



RIFT



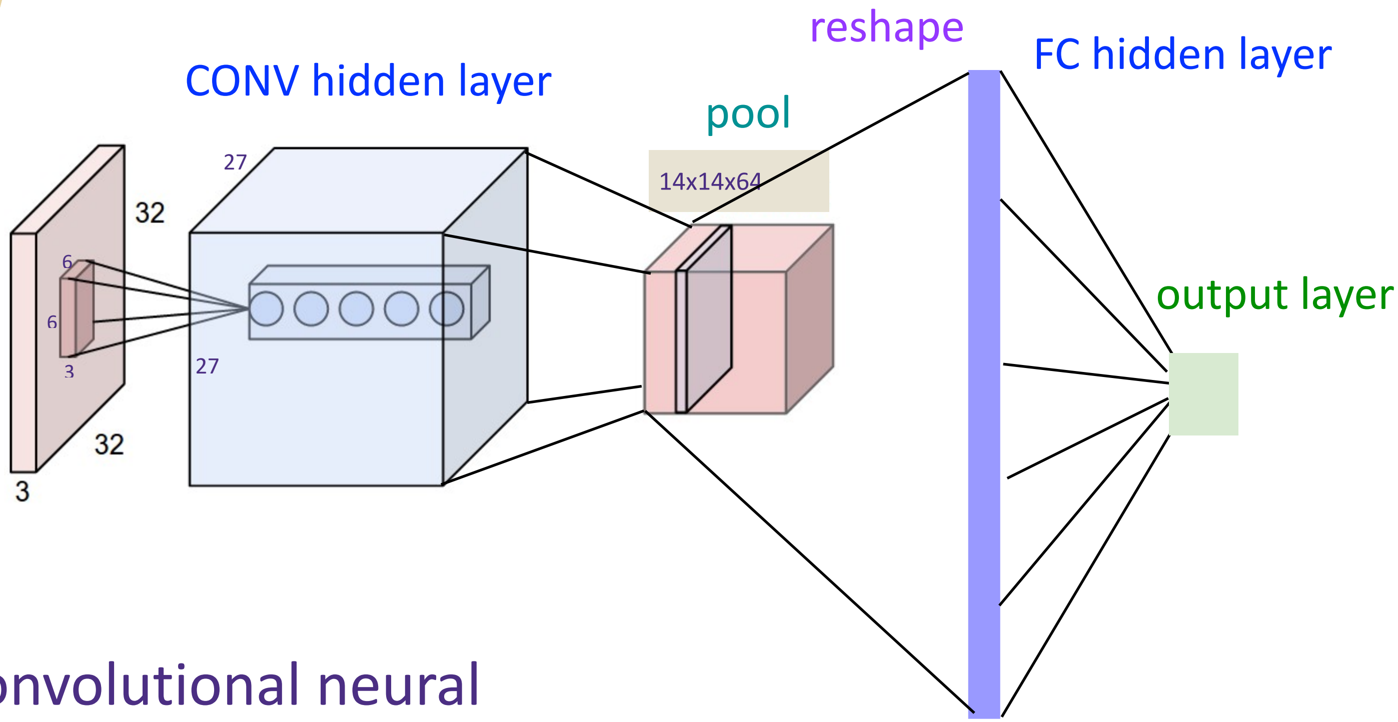
Texton



GLOH

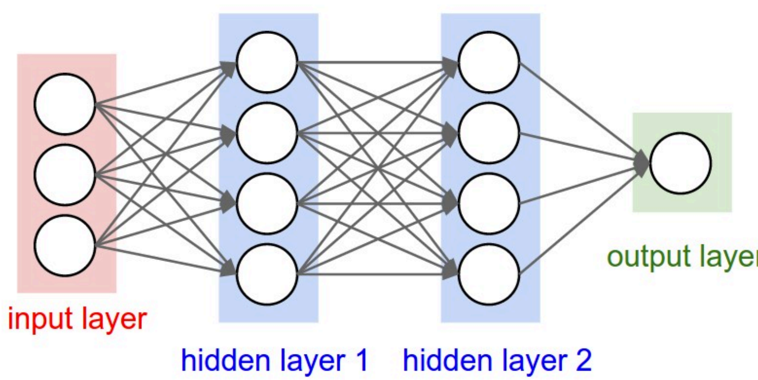
Slide from Honglak Lee

Learning Features with Convolutional Networks

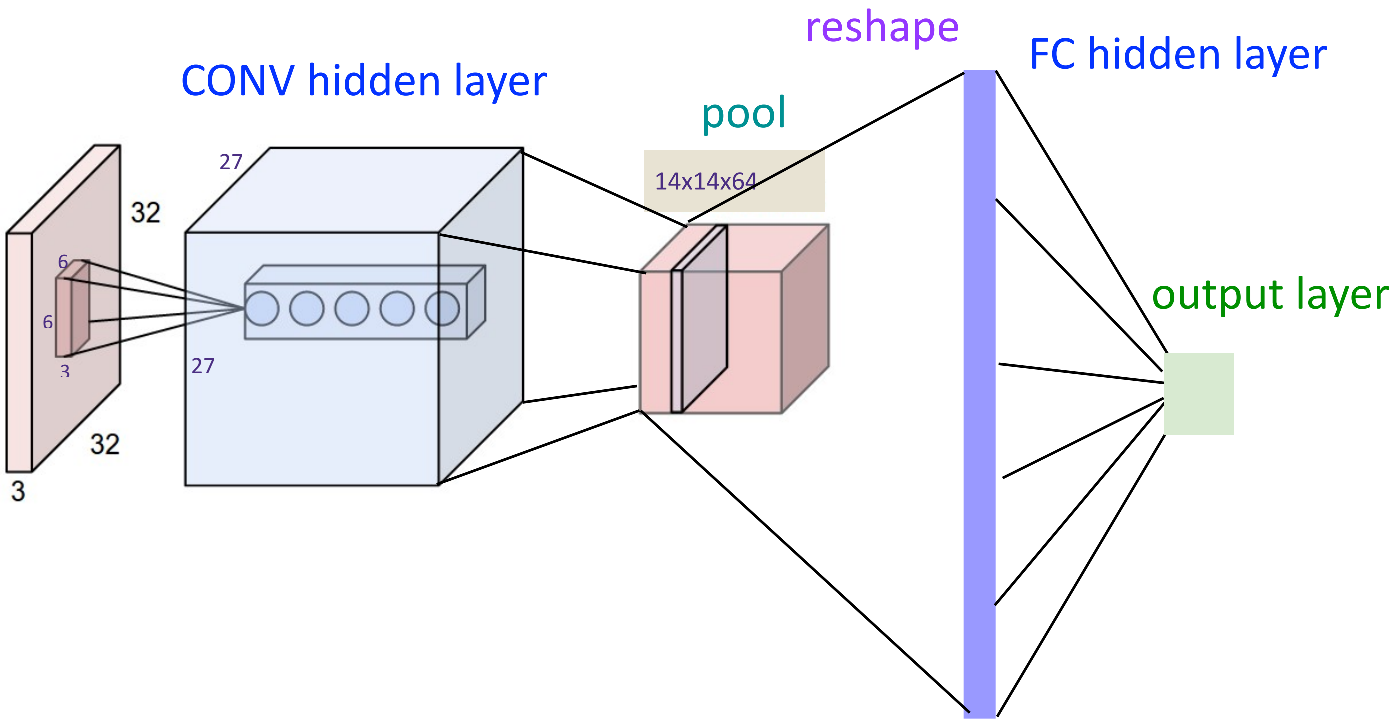


Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.

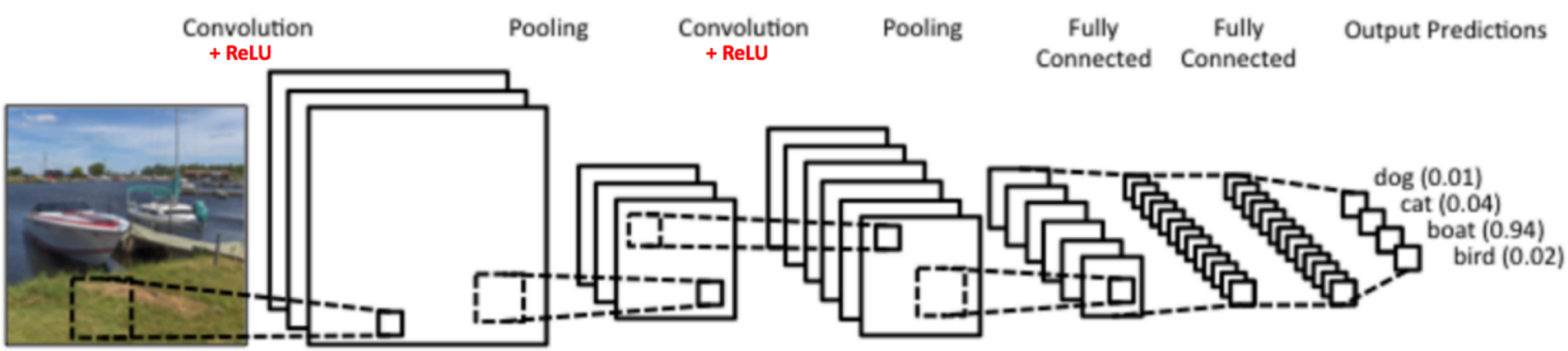
Train with SGD!

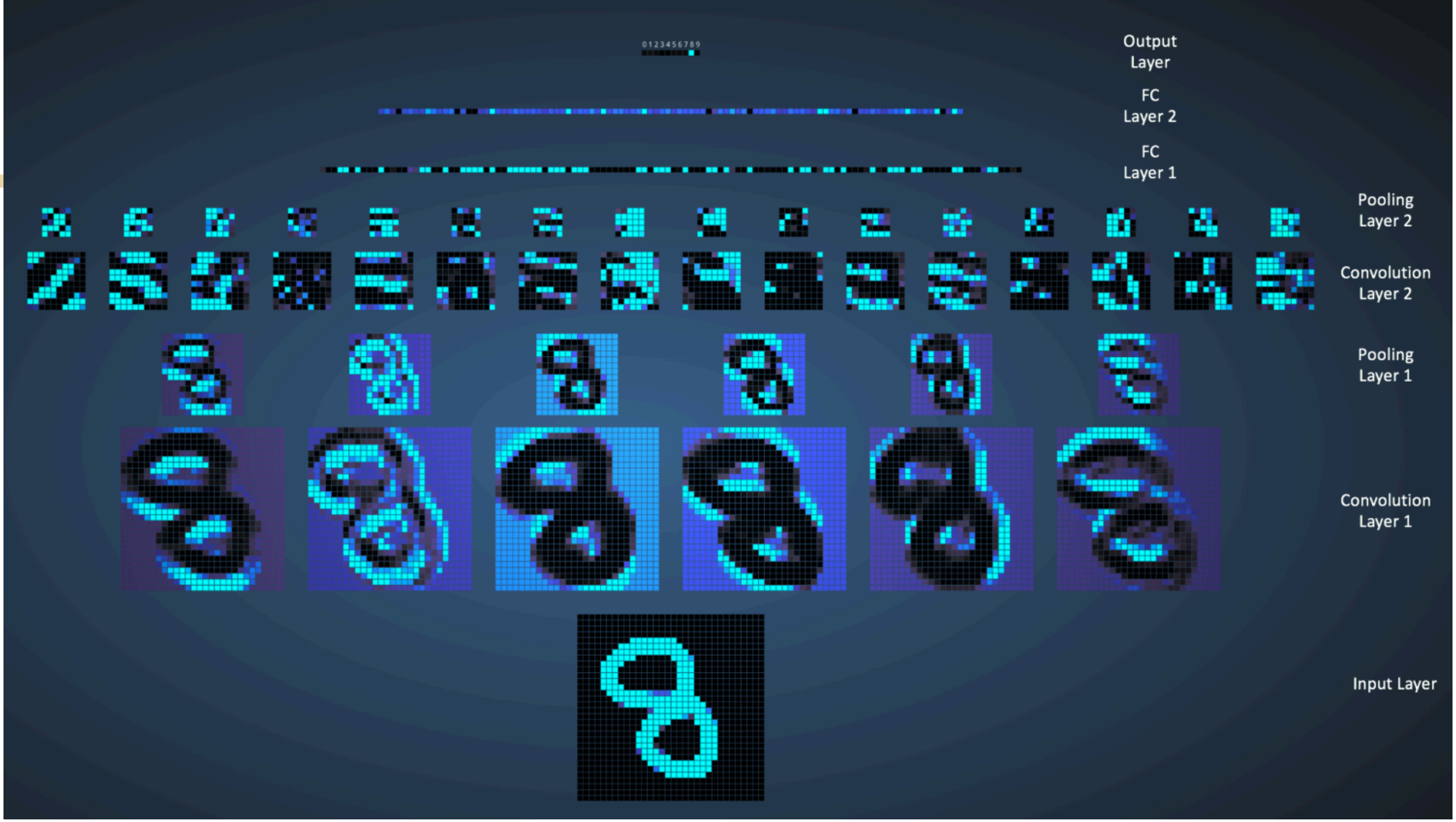


Training Convolutional Networks

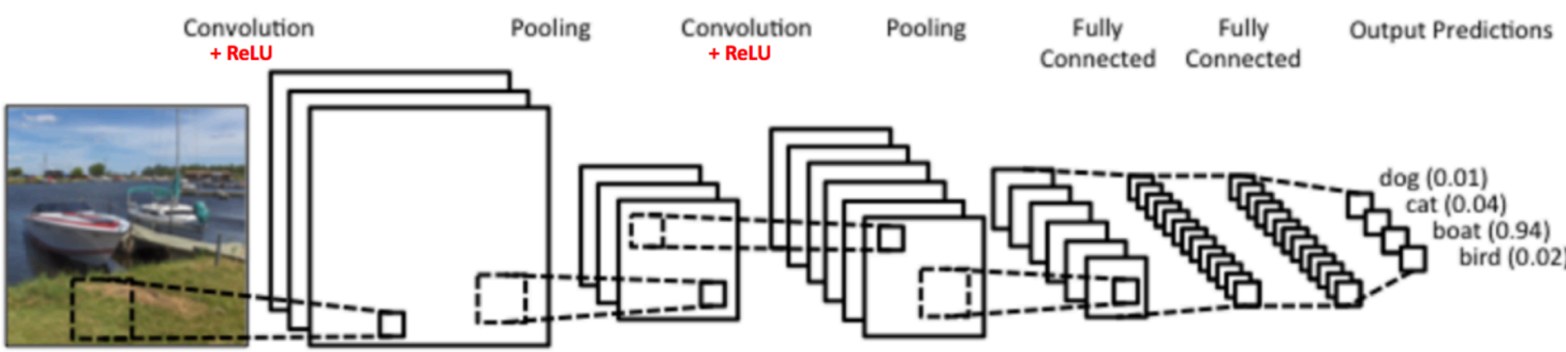


Real example network: LeNet





Real example network: LeNet



Remarks

- Convolution is a fundamental operation in signal processing. Instead of hand-engineering the filters (e.g., Fourier, Wavelets, etc.) **Deep Learning *learns* the filters and CONV layers with back-propagation**, replacing fully connected (FC) layers with convolutional (CONV) layers
- **Pooling** is a dimensionality reduction operation that summarizes the output of convolving the input with a filter

- Typically the last few layers are **Fully Connected (FC)**, with the interpretation that the CONV layers are feature extractors, preparing input for the final FC layers. Can replace last layers and retrain on different dataset+task.

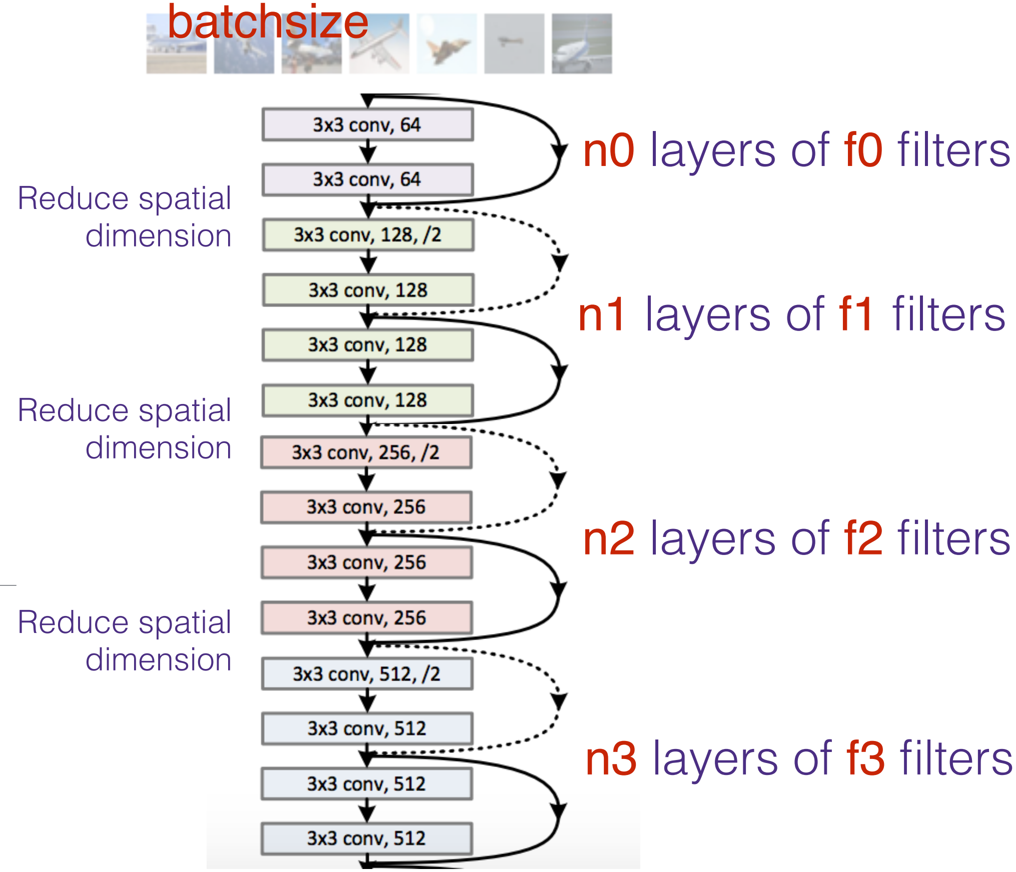
- Just as hard to train as regular neural networks.
- More exotic network architectures for specific tasks

Real networks

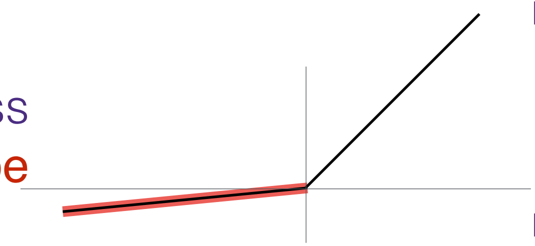
Residual Network of
[HeZhangRenSun’15]

Modern networks have
dozens of parameters to tune.

Data augmentation?
Batch norm?



RELU leakiness
slope



Learning rate schedule