Neural Networks

Machine Learning – CSE 446

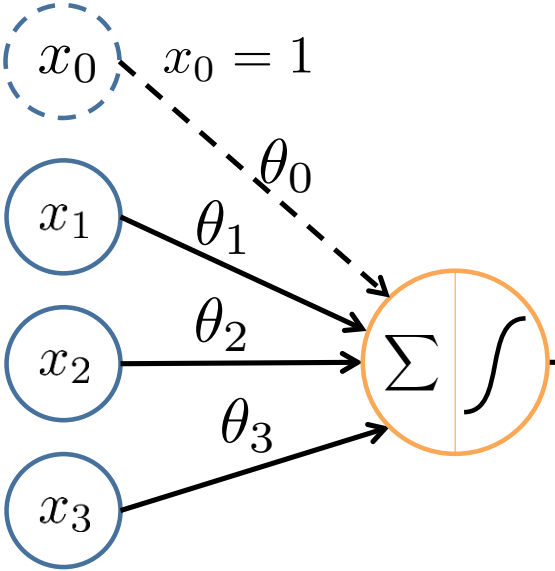
Kevin Jamieson
University of Washington

May 29, 2019

Contains slides from…

- LeCun & Ranzato
- Russ Salakhutdinov
- Honglak Lee
- Andrew Ng
- Google images
- https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
- http://cs231n.github.io/convolutional-networks/
-

Single Node



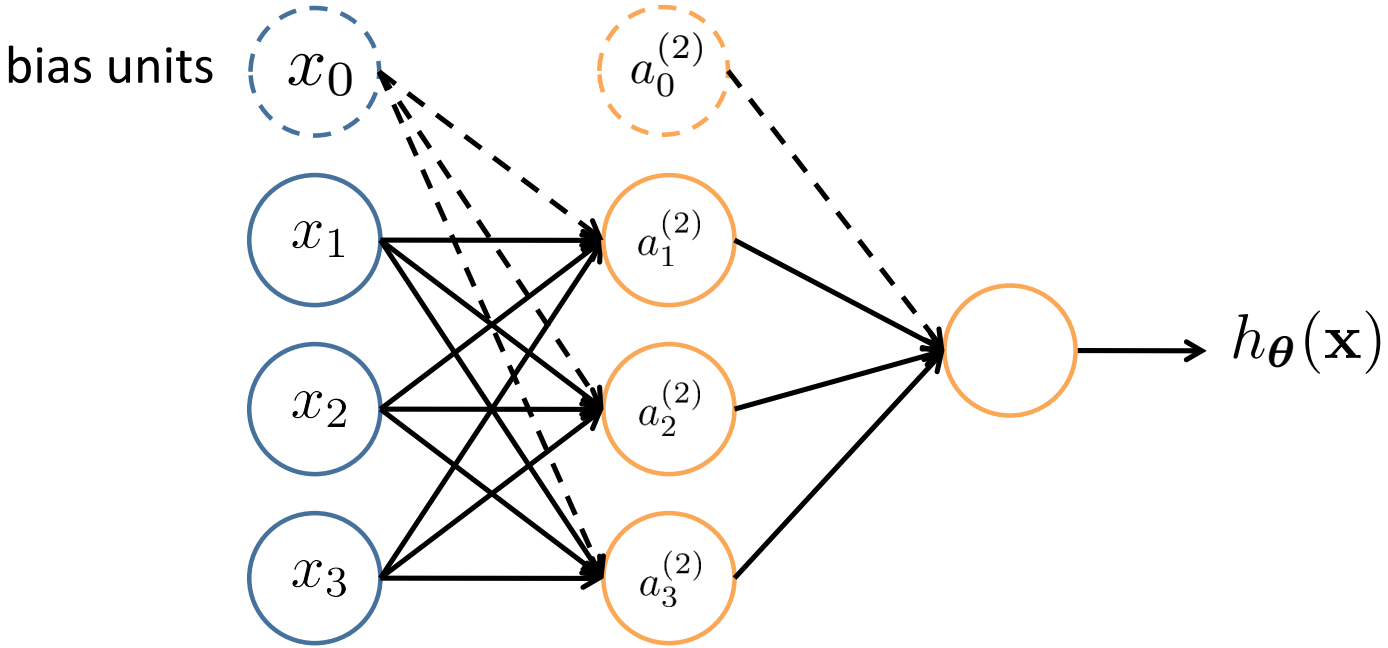
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

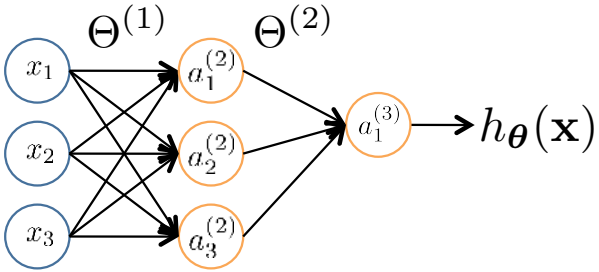
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

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

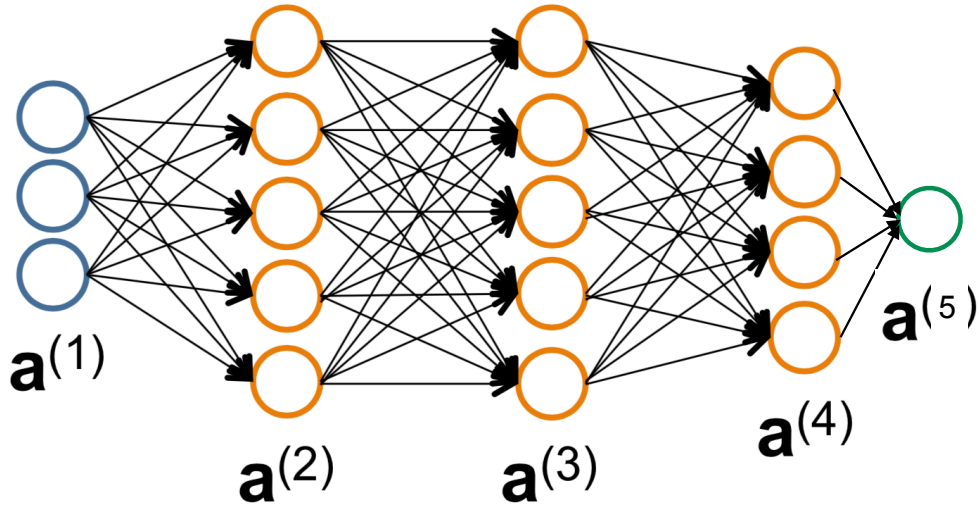
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)}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

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

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$



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

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

Binary Logistic Regression

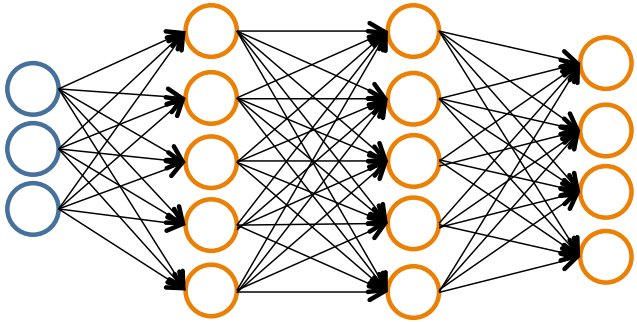
Multiple Output Units: One-vs-Rest

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

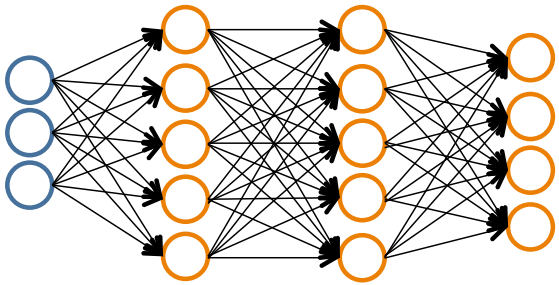
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

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

when truck

Multiple Output Units: One-vs-Rest



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

We want:

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

---

- 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 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, $\left|F(\boldsymbol{x}) - \hat{F}(\boldsymbol{x})\right| < \epsilon$.*

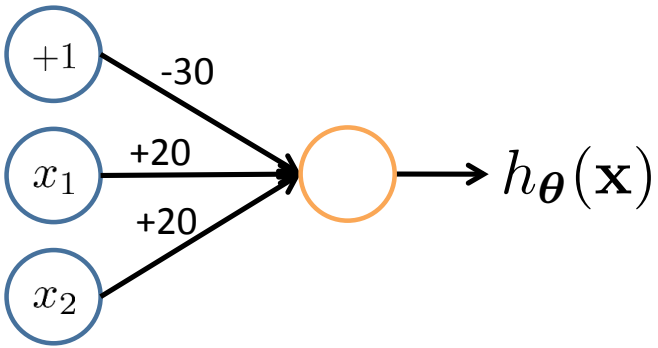
Cybenko, Hornik (theorem reproduced from CIML, Ch. 10)

Representing Boolean Functions

Simple example: AND

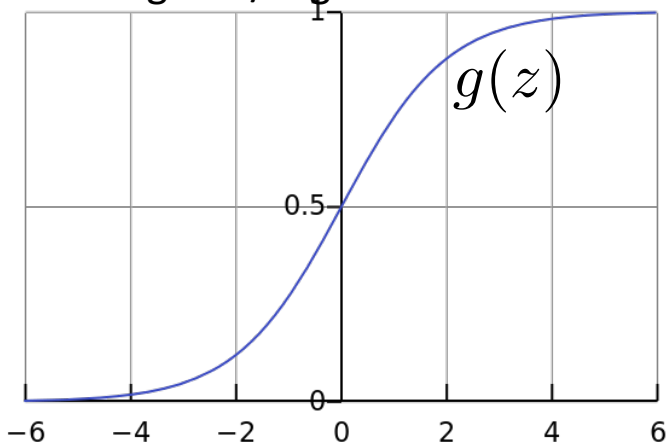
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



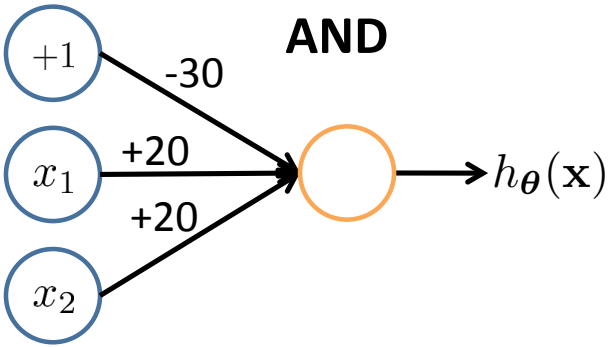
$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$

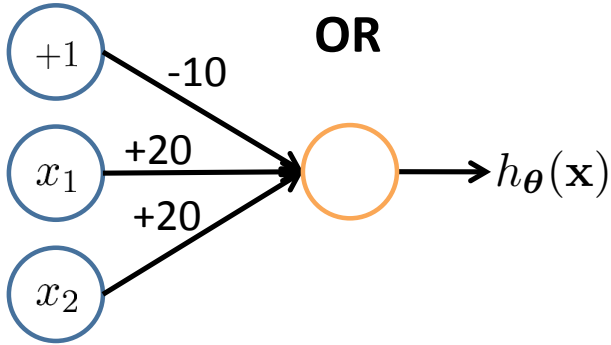
Logistic / Sigmoid Function

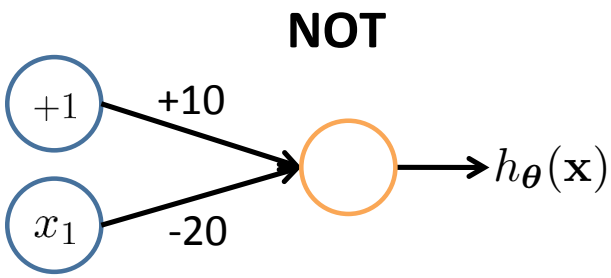


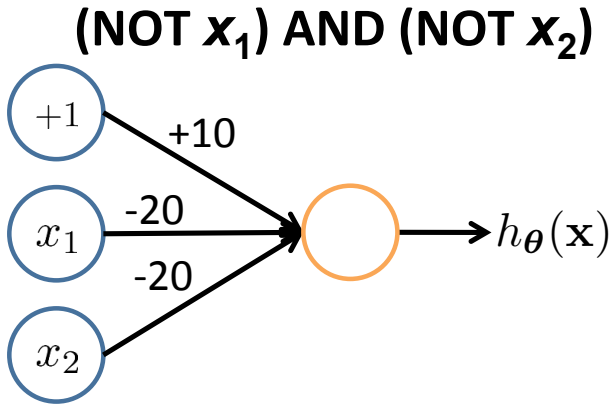
x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Representing Boolean Functions

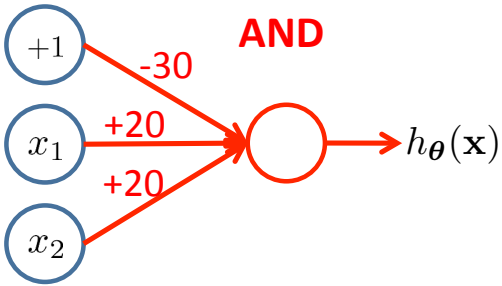


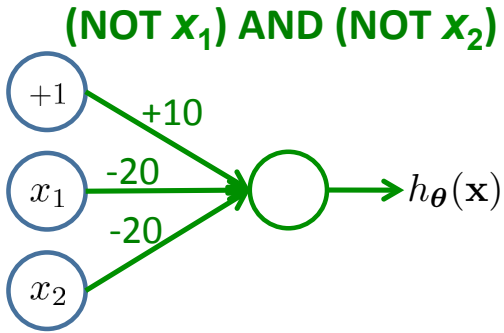


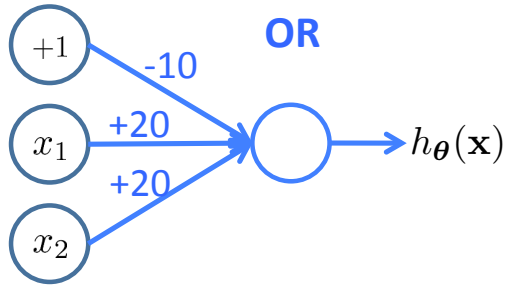


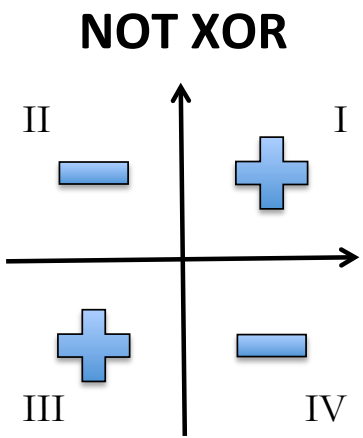


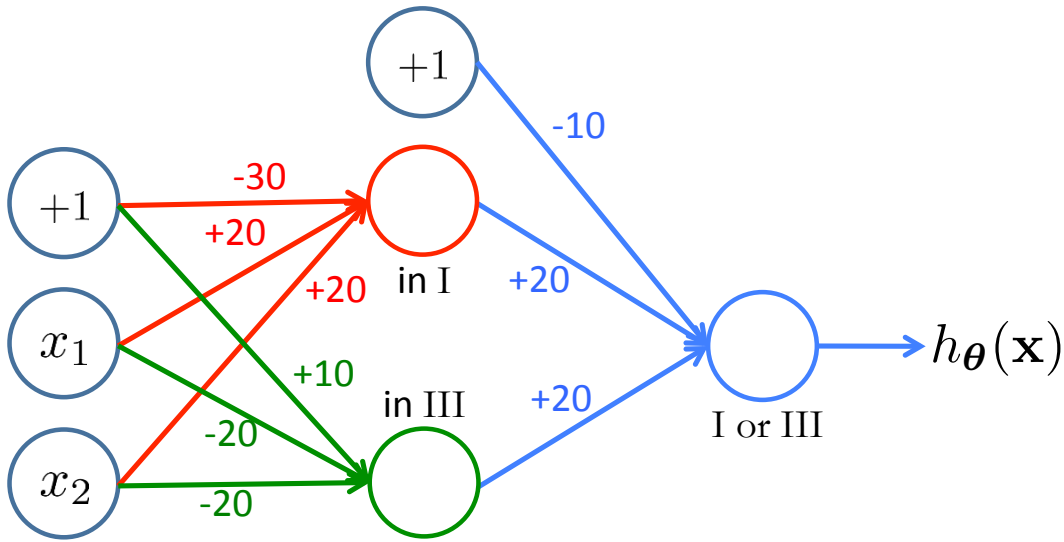
Combining Representations to Create Non-Linear Functions







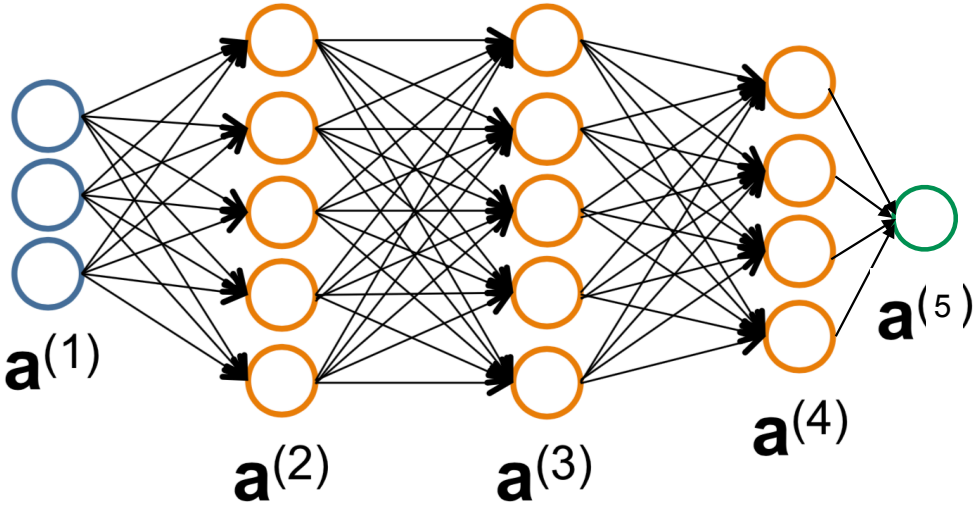




Based on example by Andrew Ng

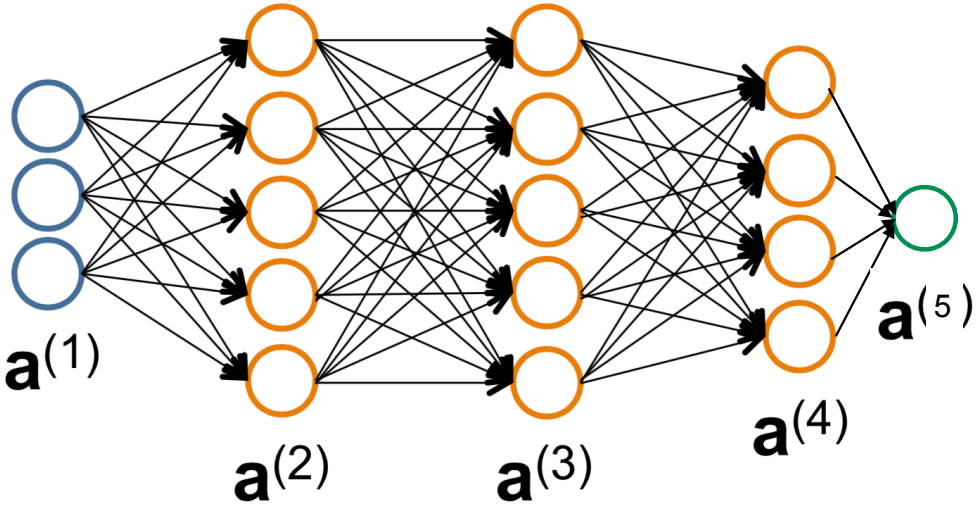
Neural Network Architecture

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**.



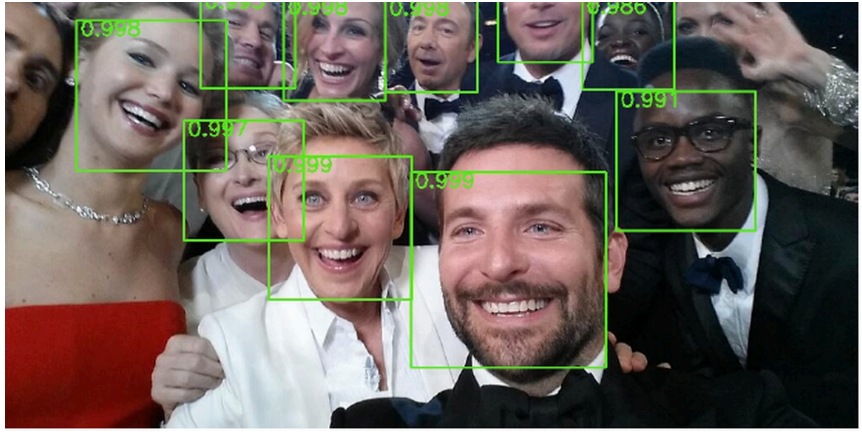
We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

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

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

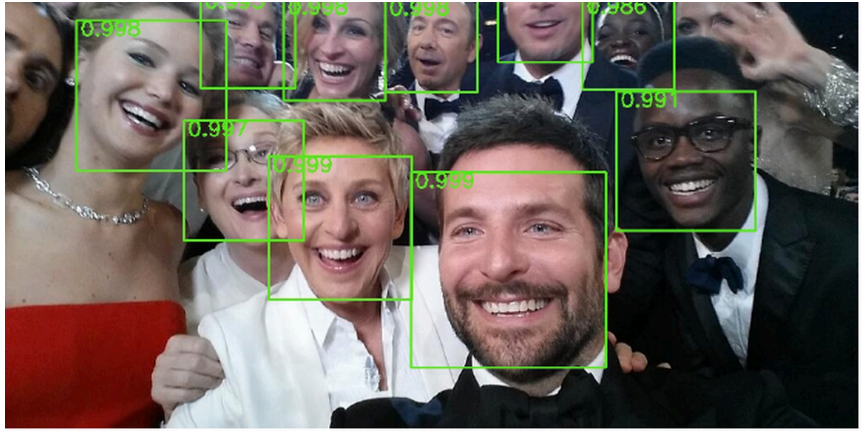
Neural Network Architecture

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.

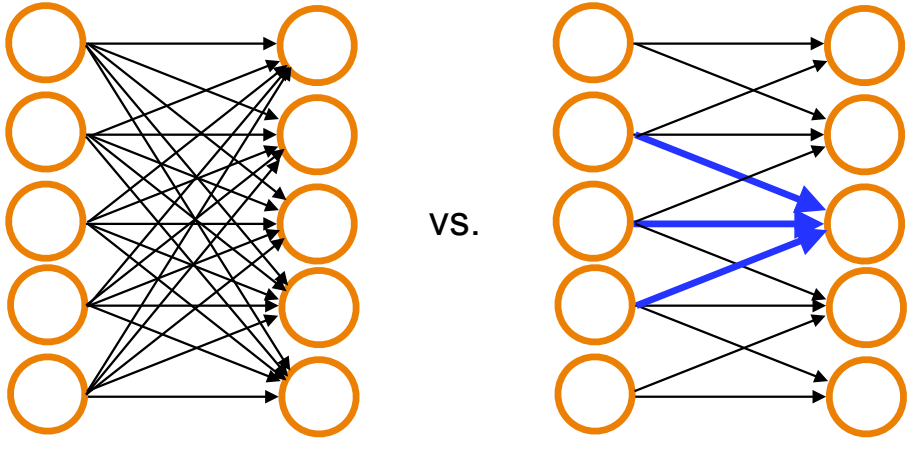


Neural Network Architecture

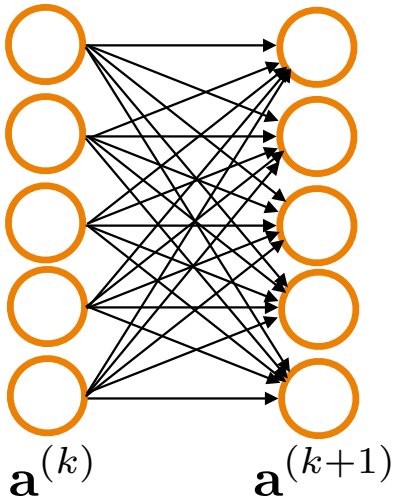
Objects are often **localized in space** so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.



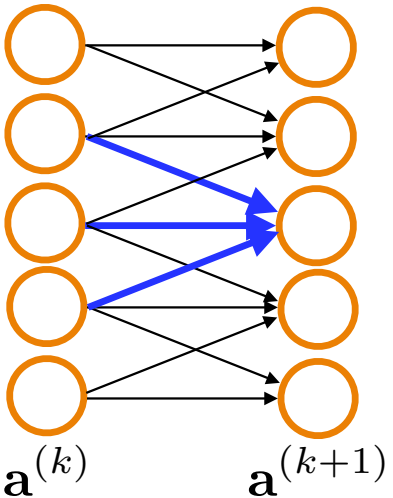
Similarly, to identify edges or other local structure, it makes sense to only look at **local information**



Neural Network Architecture



vs.

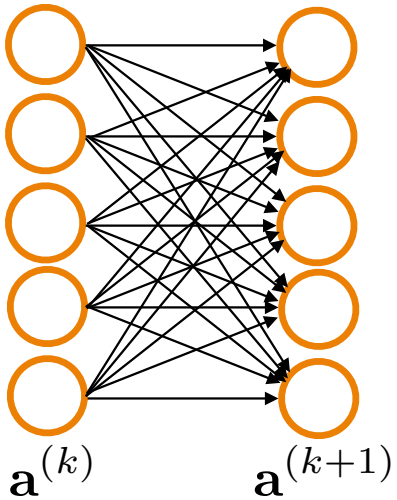


$$\begin{bmatrix} \Theta_{0,0} & \Theta_{0,1} & \Theta_{0,2} & \Theta_{0,3} & \Theta_{0,4} \\ \Theta_{1,0} & \Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} \\ \Theta_{2,0} & \Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} \\ \Theta_{3,0} & \Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3}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

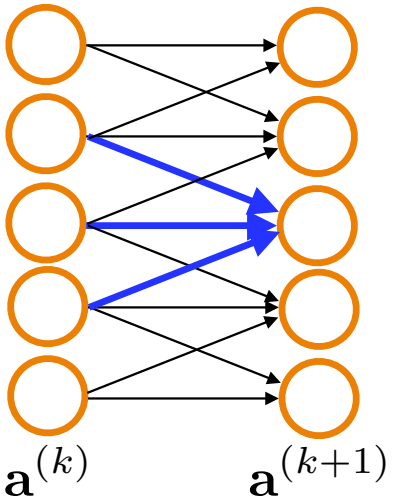
Parameters: n^2 $3n-2$

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

Neural Network Architecture



vs.



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

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

Parameters: n^2 $3n-2$ 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"

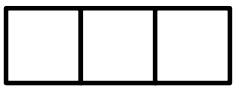
* Actually defined as the closely related quantity of "cross-correlation" but the deep learning literature just calls this "convolution"

Example (1d convolution)

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

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

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



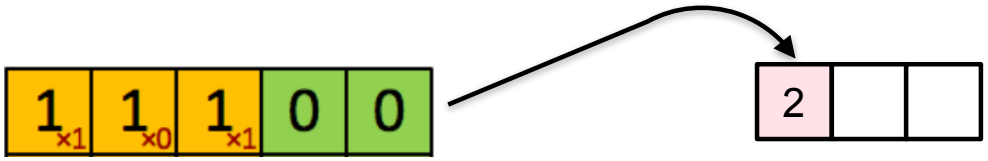
Output $\theta * x$

Example (1d convolution)

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

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

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



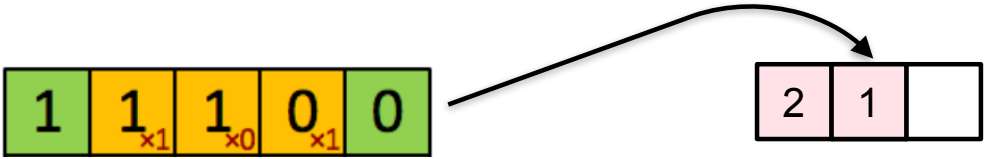
Output $\theta * x$

Example (1d convolution)

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

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

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



Output $\theta * x$

Example (1d convolution)

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

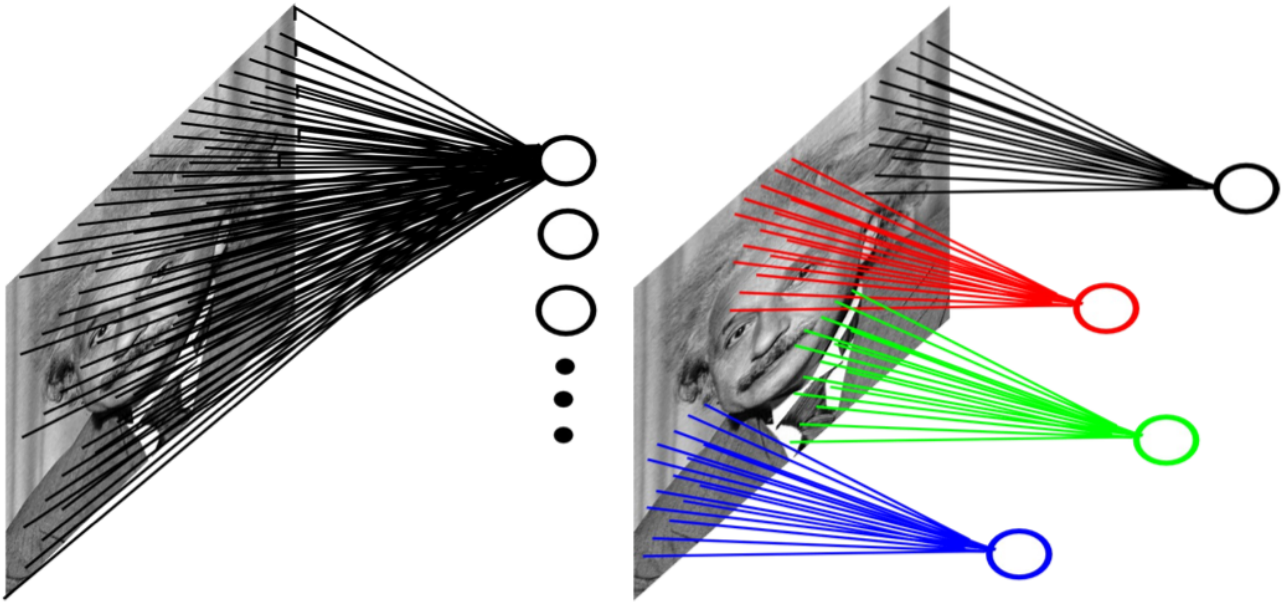
Input $x \in \mathbb{R}^n$

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

Output $\theta * x$

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)

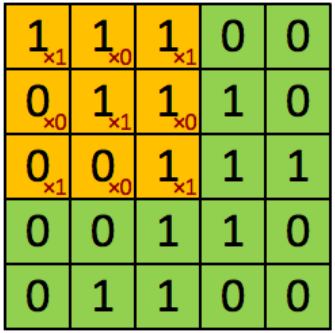
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

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

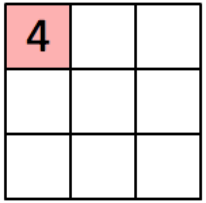
Image I

1	0	1
0	1	0
1	0	1

Filter K



Image



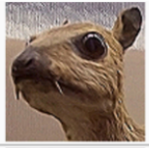
Convolved Feature

$I * K$

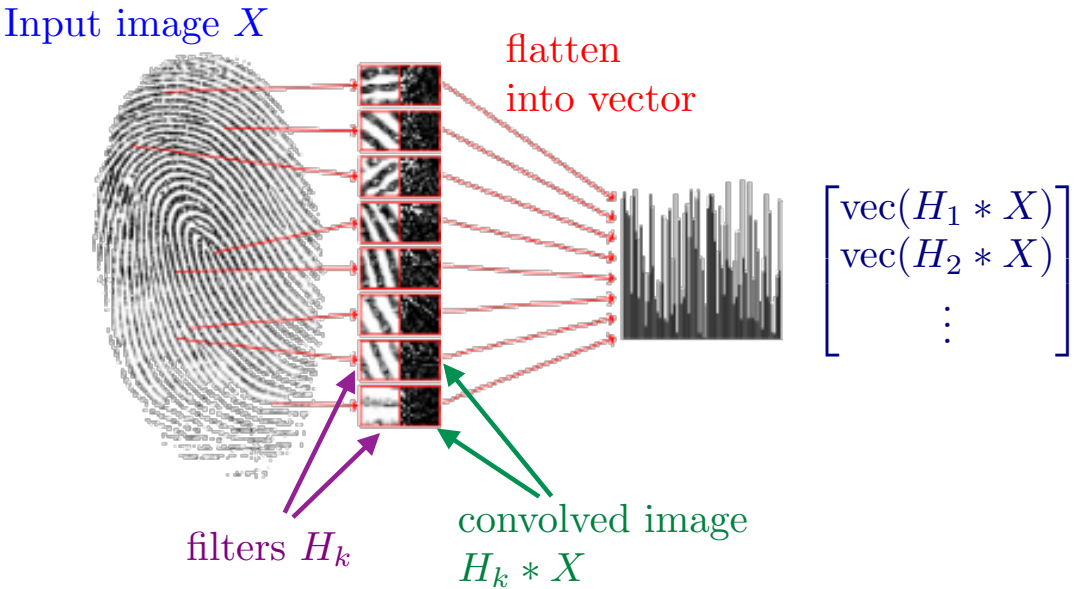
Convolution of images

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

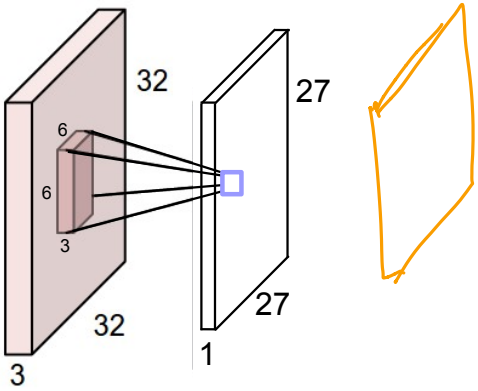
Image I

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



3d Convolution

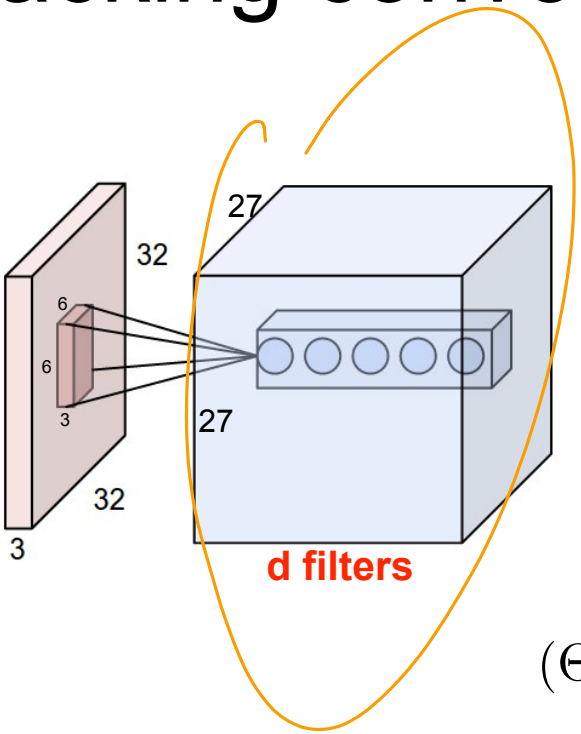


$$\Theta \in \mathbb{R}^{m \times m \times r}$$

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

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

Stacking convolved images



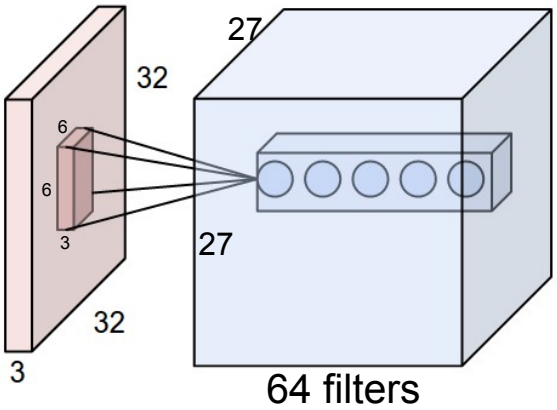
Repeat with d filters!

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

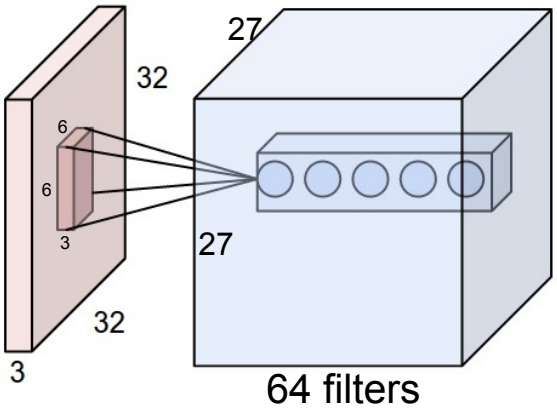
$$x \in \mathbb{R}^{n \times n \times r}$$

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

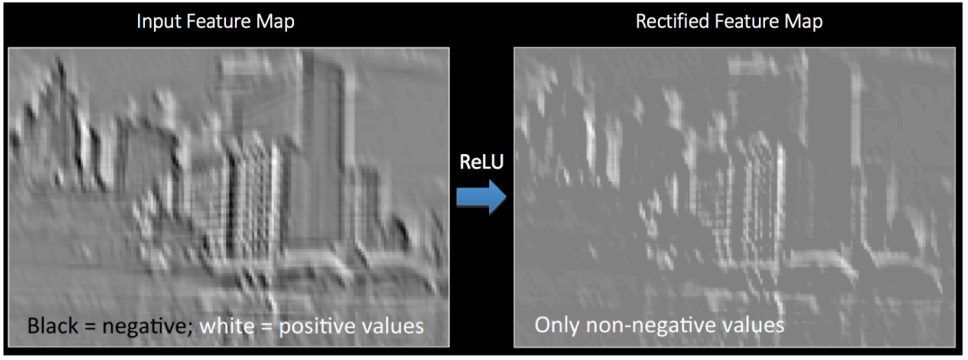
Stacking convolved images



Stacking convolved images



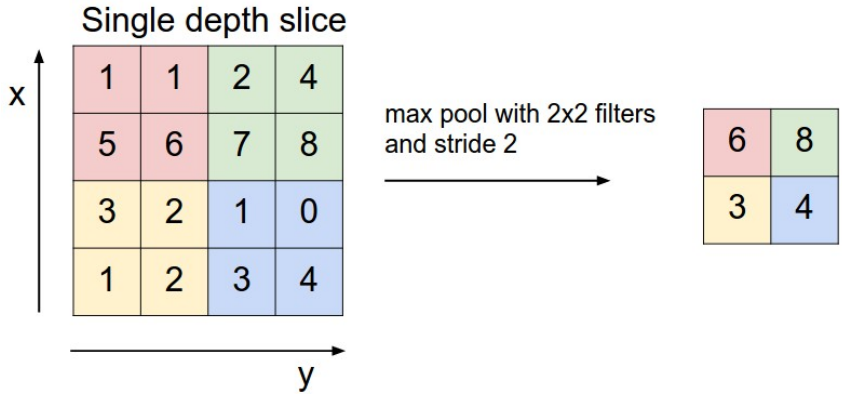
Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)

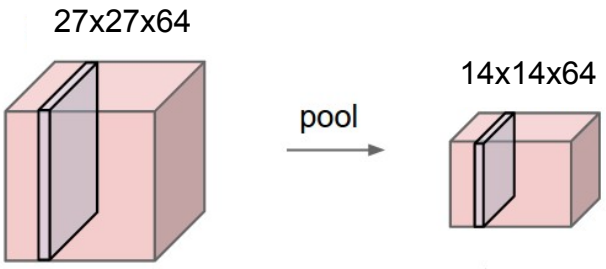


Other choices: sigmoid, arctan

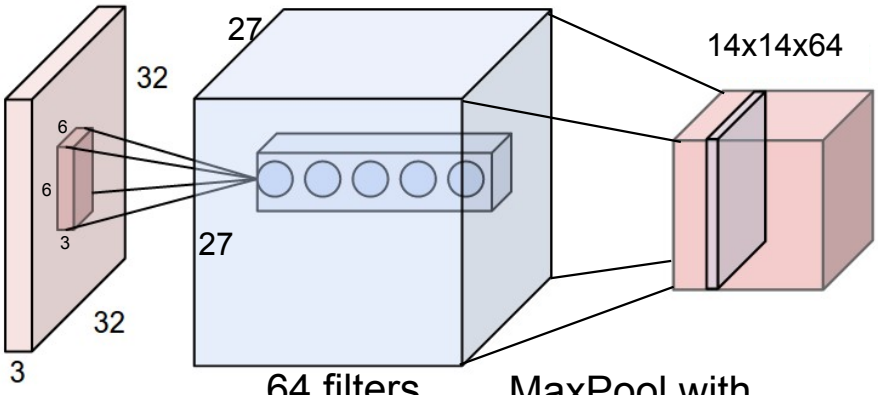
Pooling

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





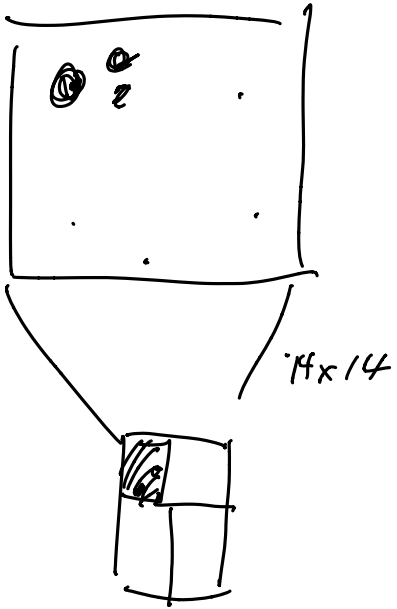
Pooling Convolution layer



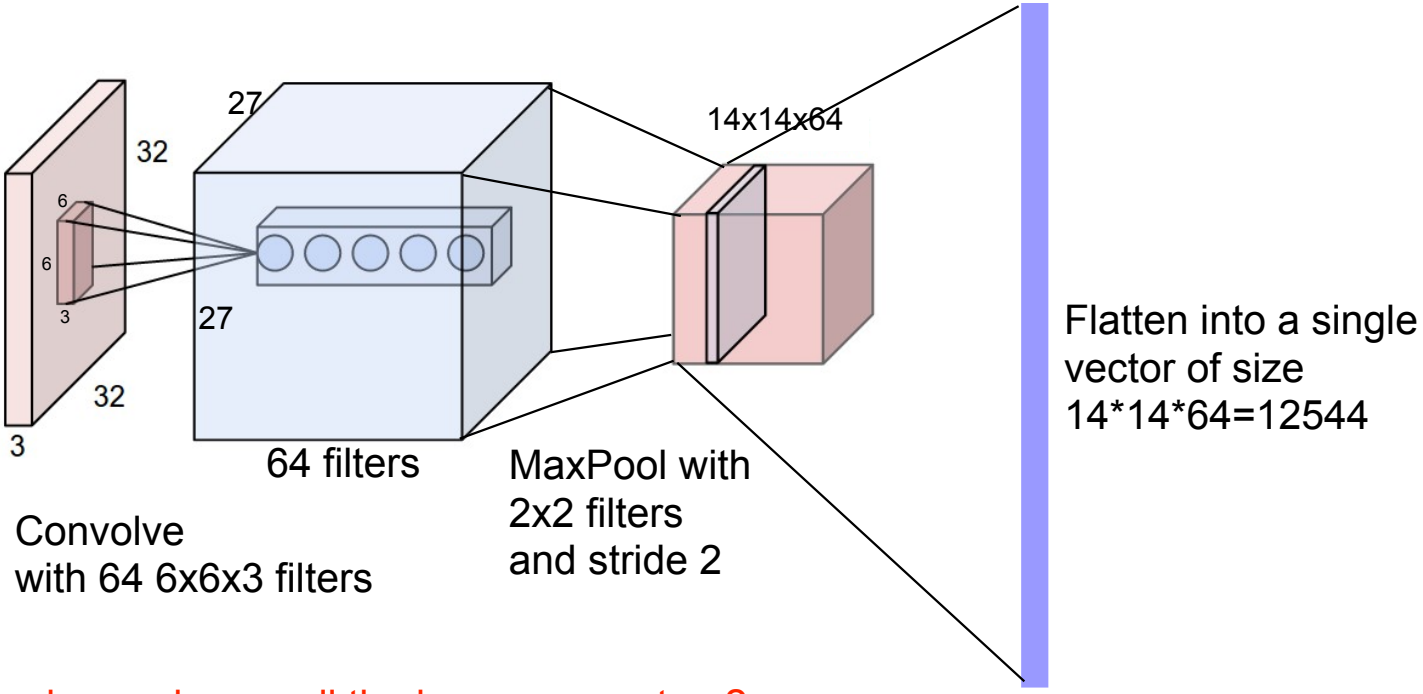
64 filters

MaxPool with
2x2 filters
and stride 2

Convolve
with 64 6x6x3 filters



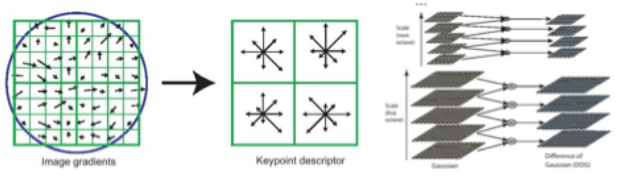
Simplest feature pipeline



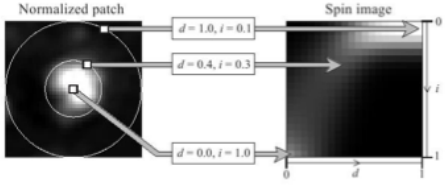
How do we choose all the hyperparameters?

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

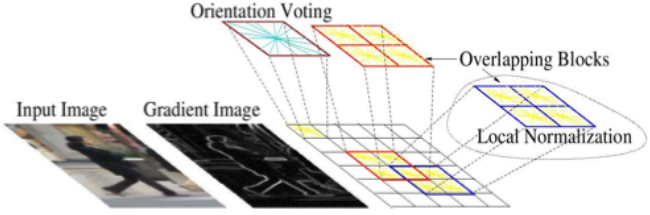
Some hand-created image features



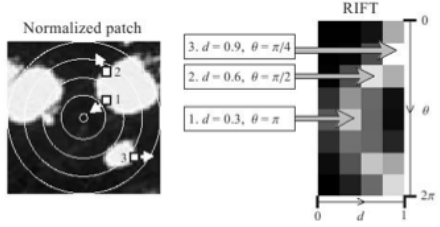
SIFT



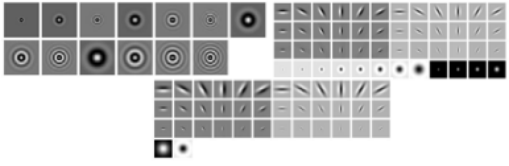
Spin Image



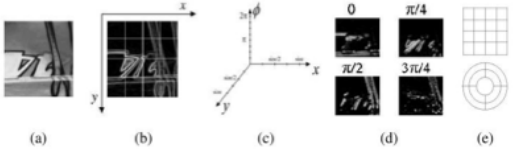
HoG



RIFT



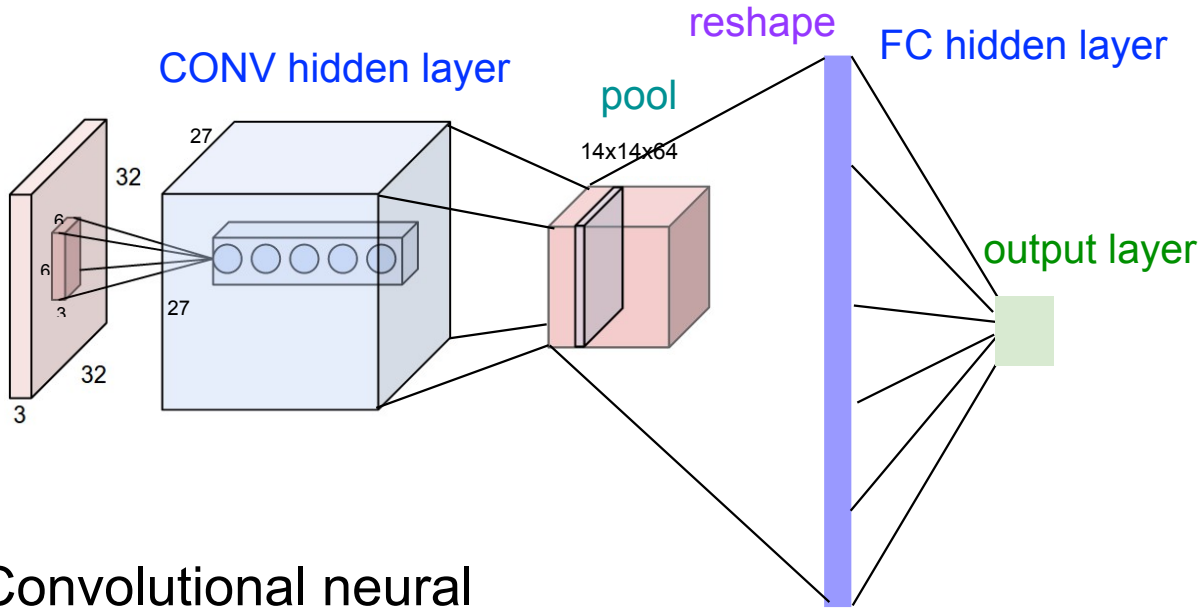
Texton



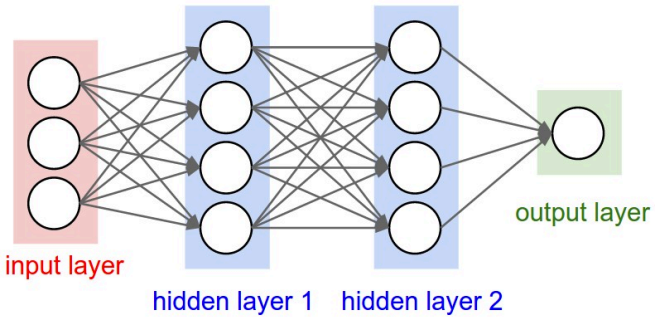
GLOH

Slide from Honglak Lee

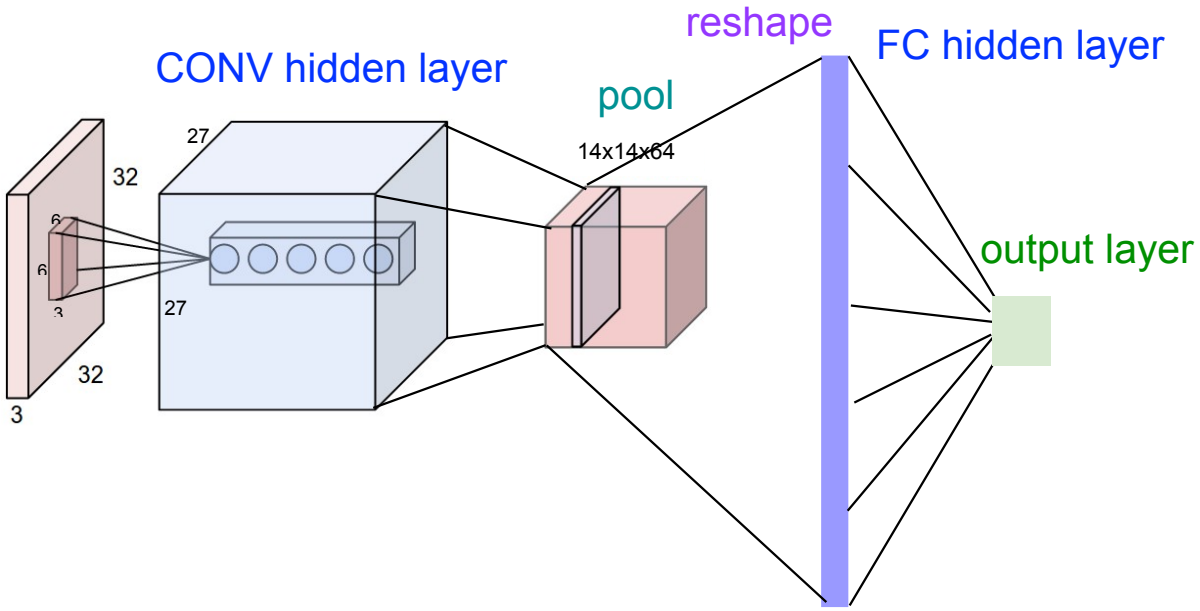
Learning Features with Convolutional Networks



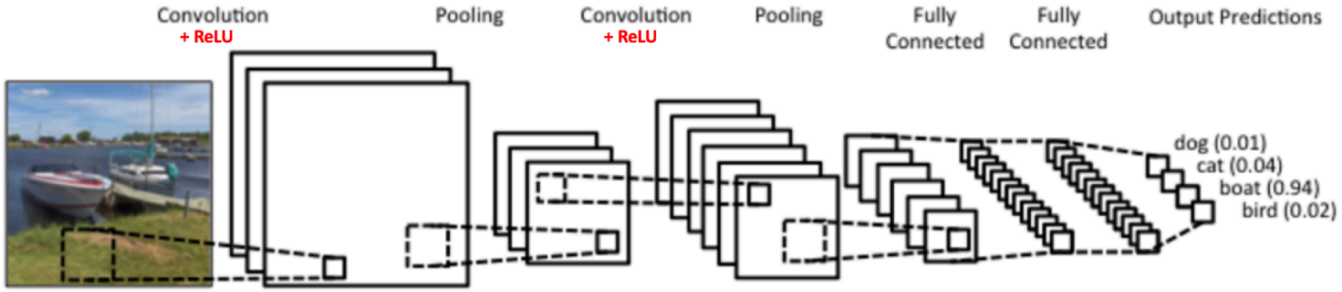
Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed.
Train with SGD!

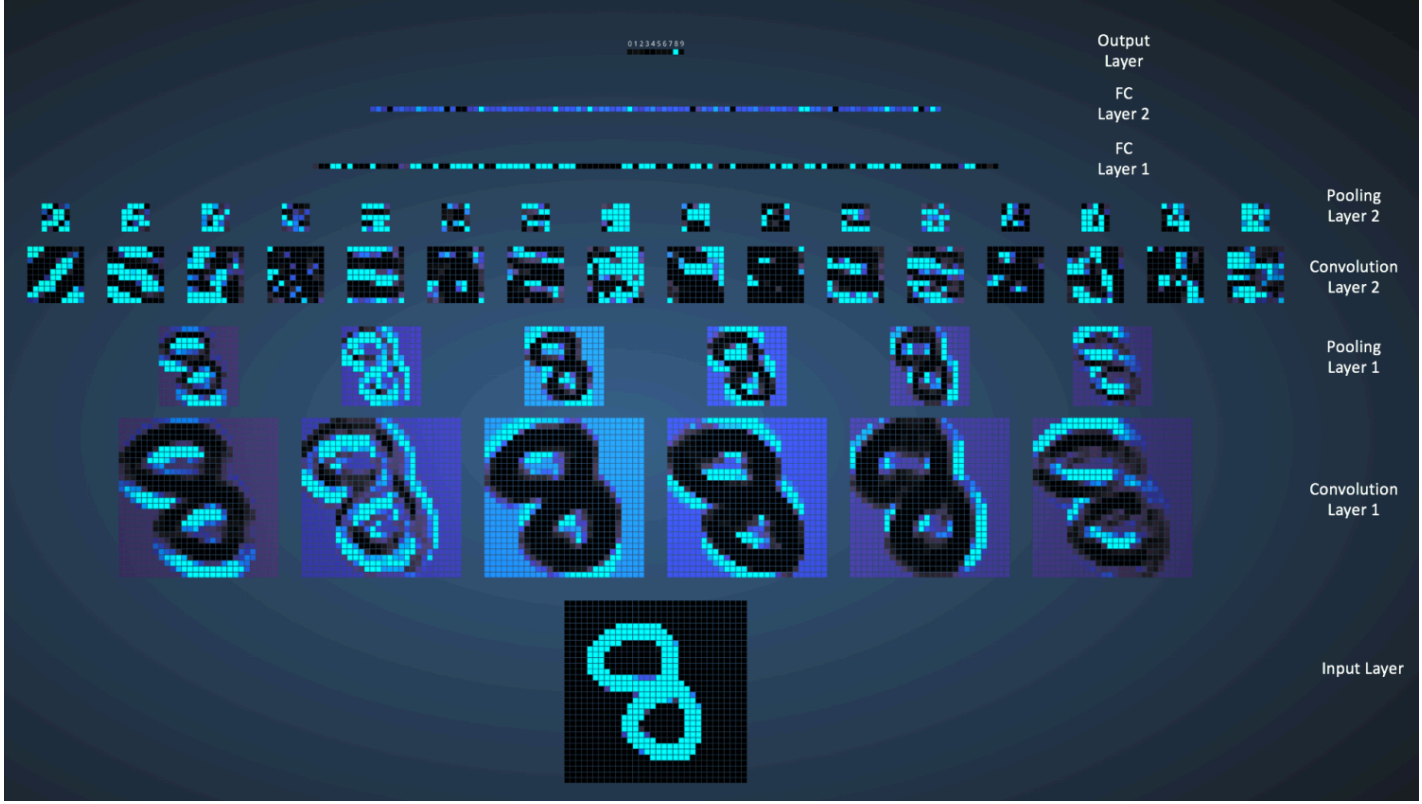


Training Convolutional Networks

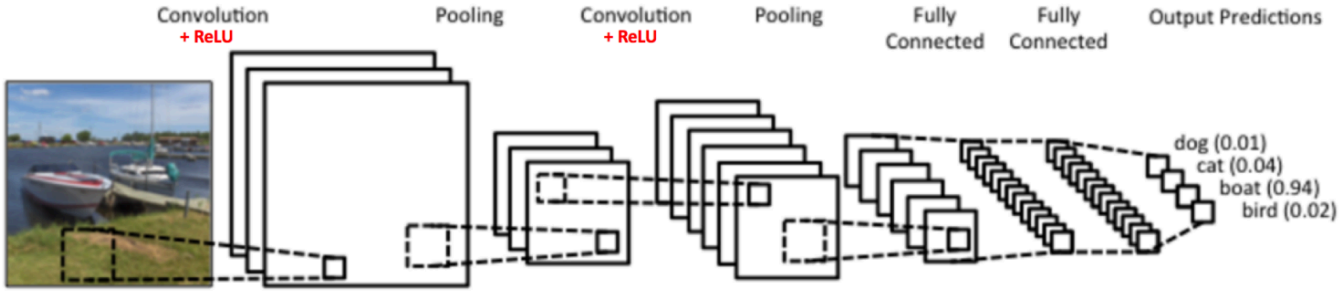


Real example network: LeNet





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

Training Neural Networks

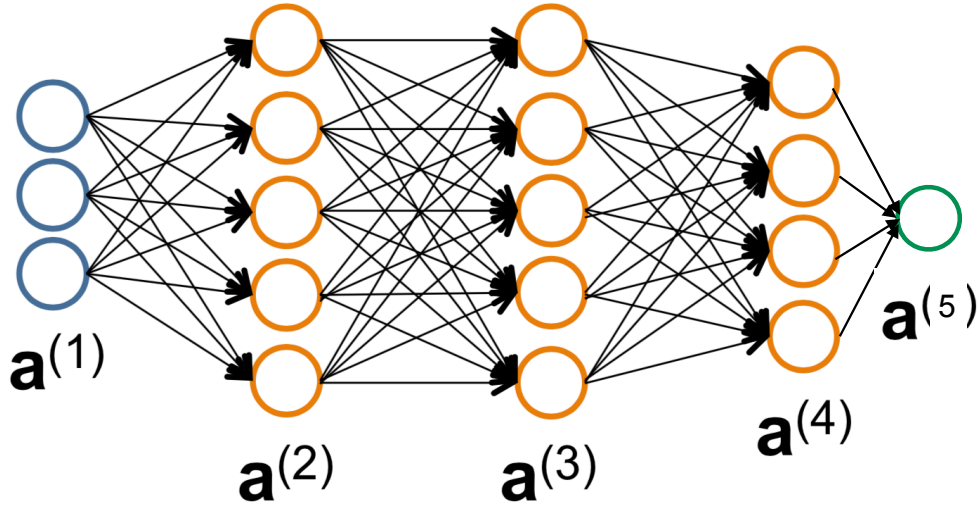
Machine Learning – CSE446

Kevin Jamieson
University of Washington

May 29, 2019

©Kevin Jamieson 2018

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$



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

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

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

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 enou
Tensorflow, Theano,
learning?

1. Automatic diffe

2. Convenient lib

```python
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
```

```python
# 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

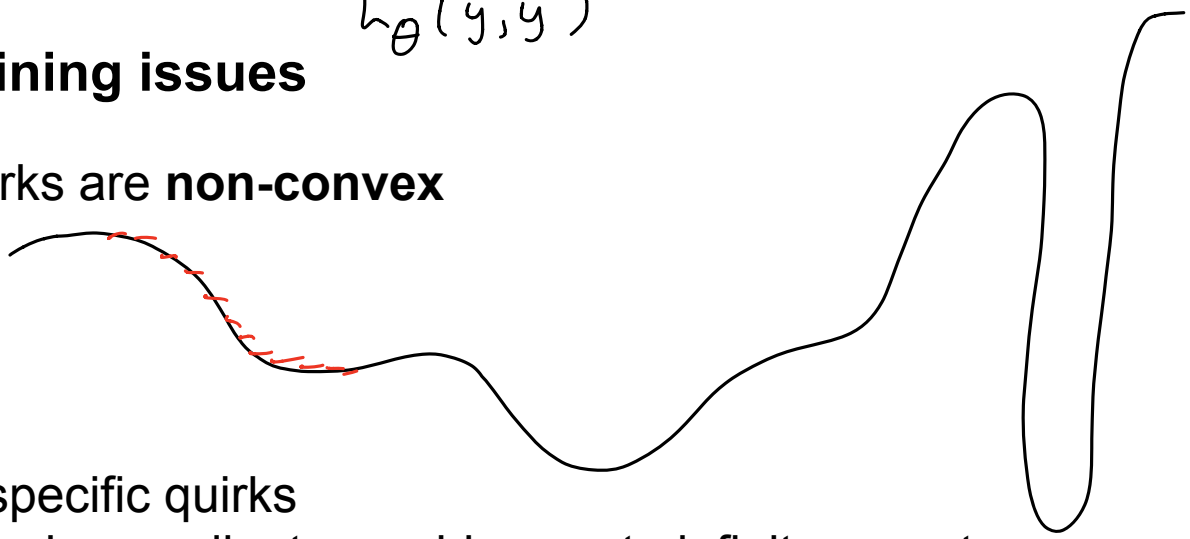
2. Convenient libraries

3. GPU support

Common training issues

$L_\theta(y, \hat{y})$

Neural networks are **non-convex**



Deep-learning specific quirks

- For large networks, gradients can blow up to infinity or go to zero. This can be helped by **batchnorm** or Residual Networks
- Stepsize, batchsize, momentum all have enormous impact on optimizing the training error *and* generalization performance
- LOTS of attention on fancy optimizers (Adagrad, Adam, etc.) that minimize training error faster. Some evidence you are speeding toward a bad local minima. Vanilla SGD generalizes, but can take a very long time.
- Making the network *bigger* may make training *faster*!

Common training issues

Training is too slow:
- Use larger step sizes, develop step size reduction schedule
- 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.)
- Use GPU

Test accuracy is bad
- Try modifying all of the above, plus changing other hyperparameters

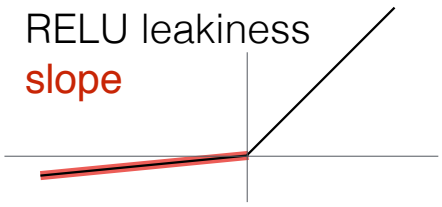
Overfitting: it is not atypical to make your network large enough to achieve 100% training accuracy (which is a good stopping condition) even as test accuracy is 20%

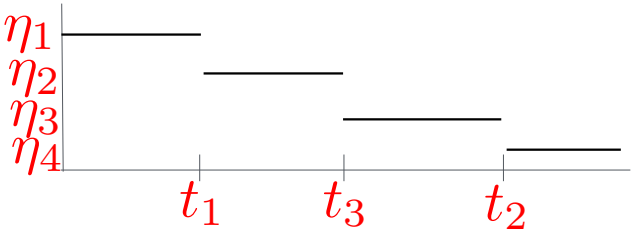
Real networks

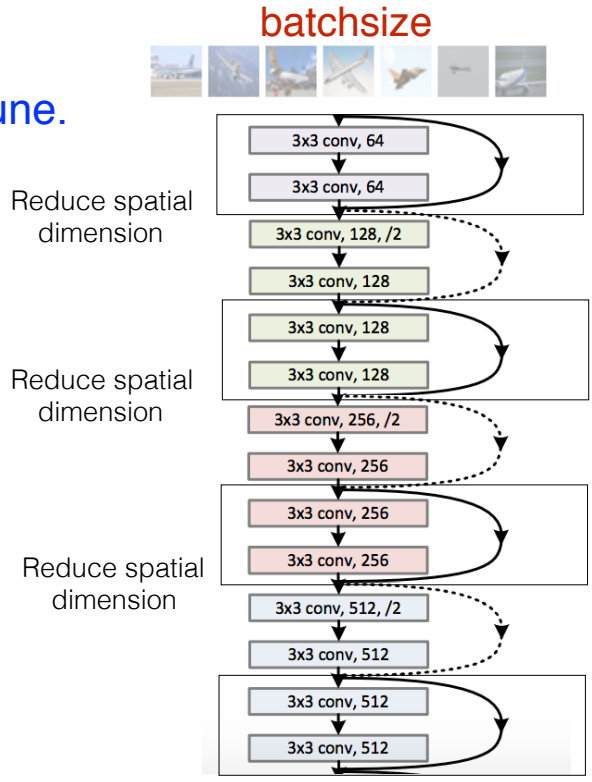
Residual Network of [HeZhangRenSun'15]

Modern networks have dozens of parameters to tune.

Data augmentation?
Batch norm?







n0 layers of f0 filters

n1 layers of f1 filters

n2 layers of f2 filters

n3 layers of f3 filters

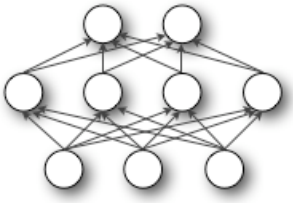
Hyperparameter Optimization

Machine Learning – CSE446

Kevin Jamieson
University of Washington

June 3, 2019

©Kevin Jamieson 2018



$N_{out} = 10$

N_{hid}

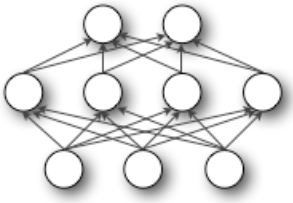
$N_{in} = 784$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$

$\widehat{f}$

Hyperparameters

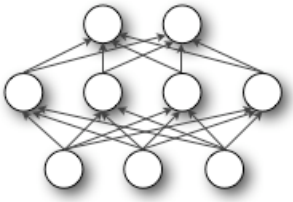
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

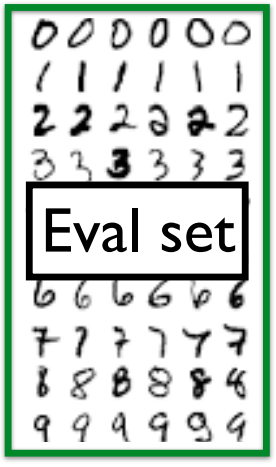
hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

Hyperparameters

$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

Eval-loss

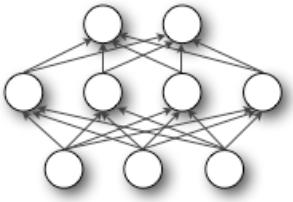
0.0577

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$

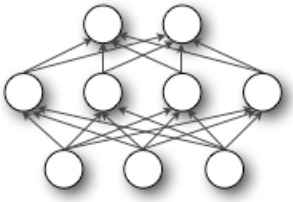
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

hyperparameters

learning rate $\eta \in [10^{-3}, 10^{-1}]$

ℓ_2-penalty $\lambda \in [10^{-6}, 10^{-1}]$

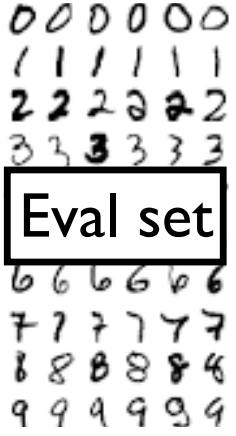
hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$

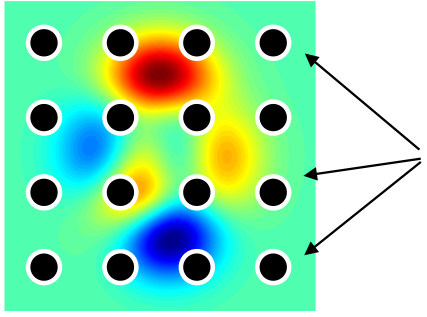


Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029

How do we choose hyperparameters to train and evaluate?

How do we choose hyperparameters to train and evaluate?

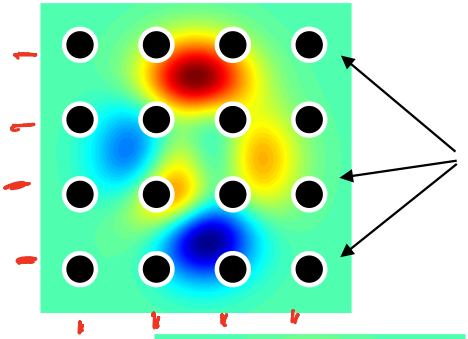
Grid search:



Hyperparameters on 2d uniform grid

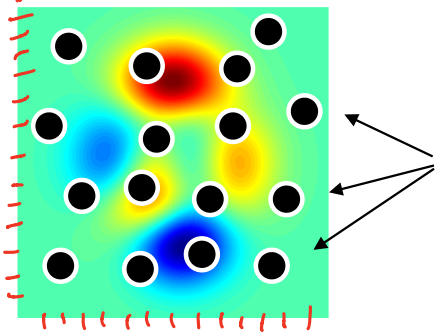
How do we choose hyperparameters to train and evaluate?

Grid search:



Hyperparameters on 2d uniform grid

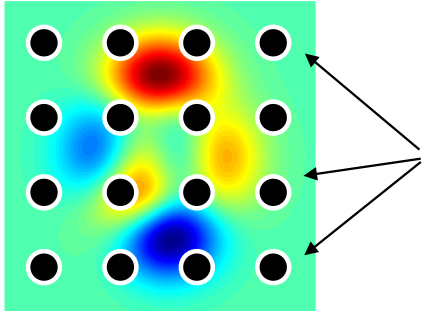
Random search:



Hyperparameters randomly chosen

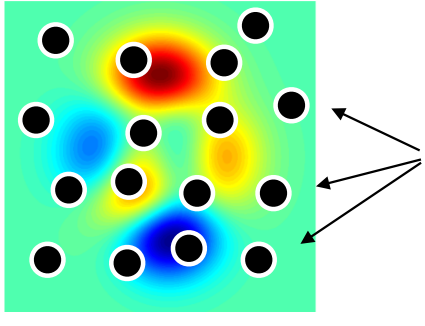
How do we choose hyperparameters to train and evaluate?

Grid search:



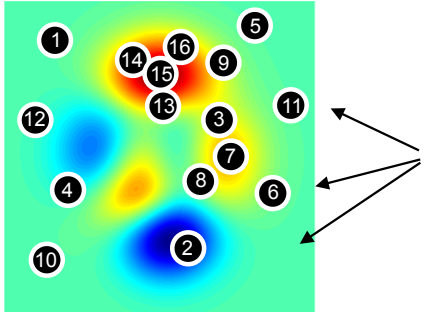
Hyperparameters on 2d uniform grid

Random search:



Hyperparameters randomly chosen

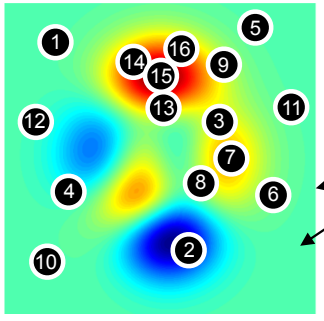
Bayesian Optimization:



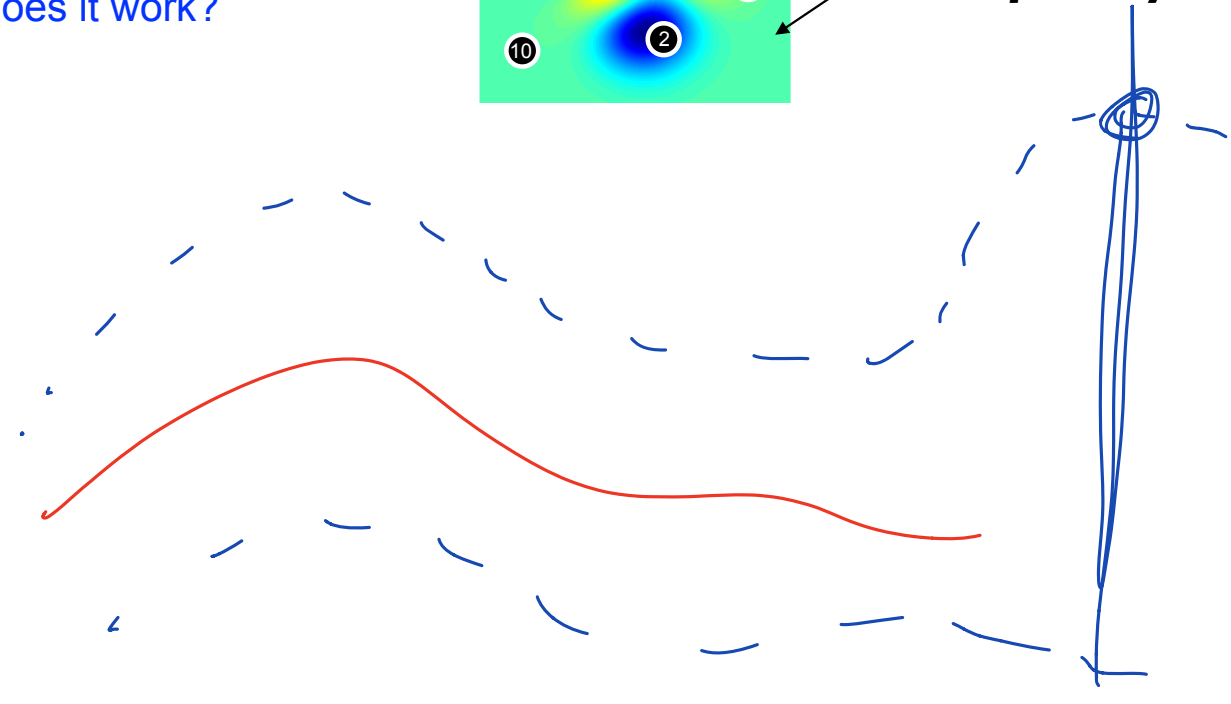
Hyperparameters ***adaptively*** chosen

Bayesian Optimization:

How does it work?



Hyperparameters ***adaptively*** chosen



Recent work attempts to speed up hyperparameter evaluation by stopping poor performing settings before they are fully trained.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. arXiv:1406.3896, 2014.

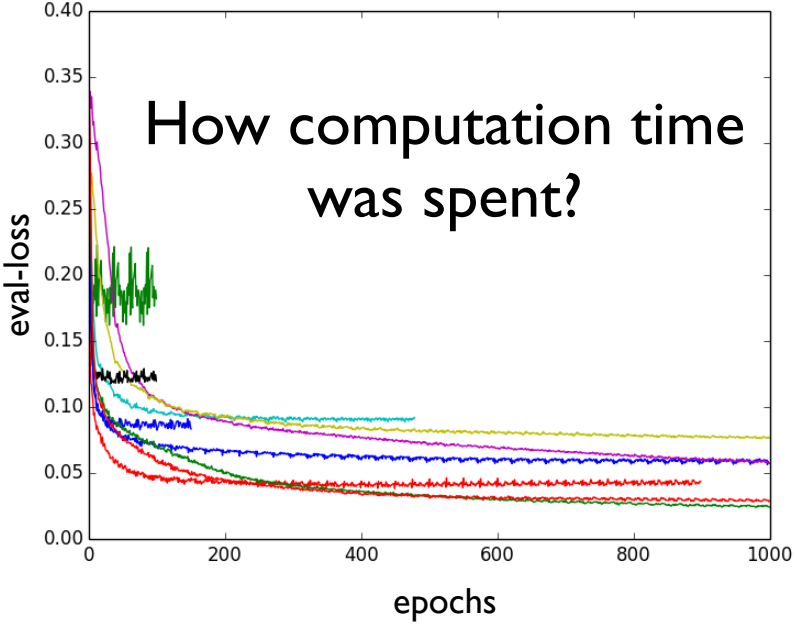
Alekh Agarwal, Peter Bartlett, and John Duchi. Oracle inequalities for computationally adaptive model selection. COLT, 2012.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

András György and Levente Kocsis. Efficient multi-start strategies for local search algorithms. JAIR, 41, 2011.

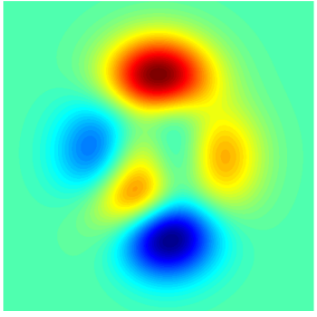
Li, Jamieson, DeSalvo, Rostamizadeh, Talwalkar. Hyperband: Hyperband: A Novel Bandit-Based Approach to Hyperparameter. JMLR 2018.

Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4}, 10^{2.6})$	0.1196
$(10^{-1.4}, 10^{-2.1}, 10^{1.5})$	0.0834
$(10^{-1.9}, 10^{-5.8}, 10^{2.1})$	0.0242
$(10^{-1.8}, 10^{-5.6}, 10^{1.7})$	0.029



Hyperparameter Optimization

In general, hyperparameter optimization is non-convex optimization and little is known about the underlying function (only observe validation loss)



Your time is valuable, computers are cheap:
Do not employ "grad student descent" for hyper parameter search.
Write modular code that takes parameters as input and automate this embarrassingly parallel search.

Tools for different purposes:

- Very few evaluations: use random search (and pray) or be *clever*
- Few evaluations and long-running computations: see refs on last slide
- Moderate number of evaluations (but still exp(#params)) and high accuracy needed: use Bayesian Optimization
- Many evaluations possible: use random search. Why overthink it?

Backprop

Machine Learning – CSE446

Kevin Jamieson
University of Washington

May 29, 2019

©Kevin Jamieson 2018

Backprop

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

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

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

$$\vdots$$

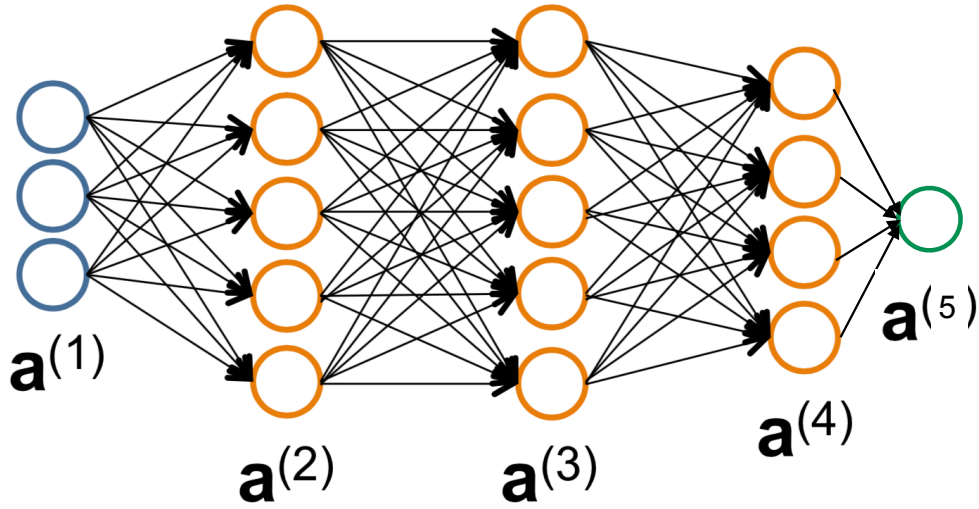
$$a^{(l)} = g\left(z^{(l)}\right)$$

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

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

$$\vdots$$

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



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

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

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:

for i, j, l

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

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

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

Backprop

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$a^{(l)} = g\left(z^{(l)}\right)$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

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

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

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

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

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

$$\vdots$$

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

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

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

$$\vdots$$

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

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

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

$\Theta^{(l)} g'(z^{(l)}) \cdot 1$

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

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

Backprop

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$a^{(l)} = g\left(z^{(l)}\right)$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

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

$$\delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}$$
$$= \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)} \, g'(z_i^{(l)})$$
$$= a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i}^{(l)}$$

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

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

Backprop

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$a^{(l)} = g\left(z^{(l)}\right)$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

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

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

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

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

Backprop

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

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

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

$$\vdots$$

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

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

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

$$\vdots$$

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

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

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

$$\delta_i^{(L+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)})) \right]$$

$$= \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)})$$

$$= y - g(z^{(L+1)}) = y - a^{(L+1)}$$

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

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

Backprop

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g\left(z^{(2)}\right)$$
$$\vdots$$
$$a^{(l)} = g\left(z^{(l)}\right)$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g\left(z^{(l+1)}\right)$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

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

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

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

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

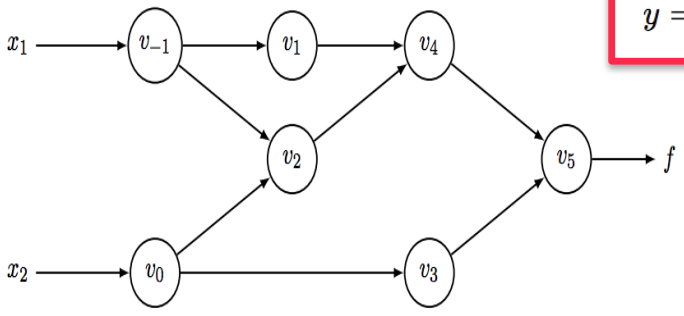
Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

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

$\boldsymbol{D}^{(l)}$ 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

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

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

Unsupervised Learning

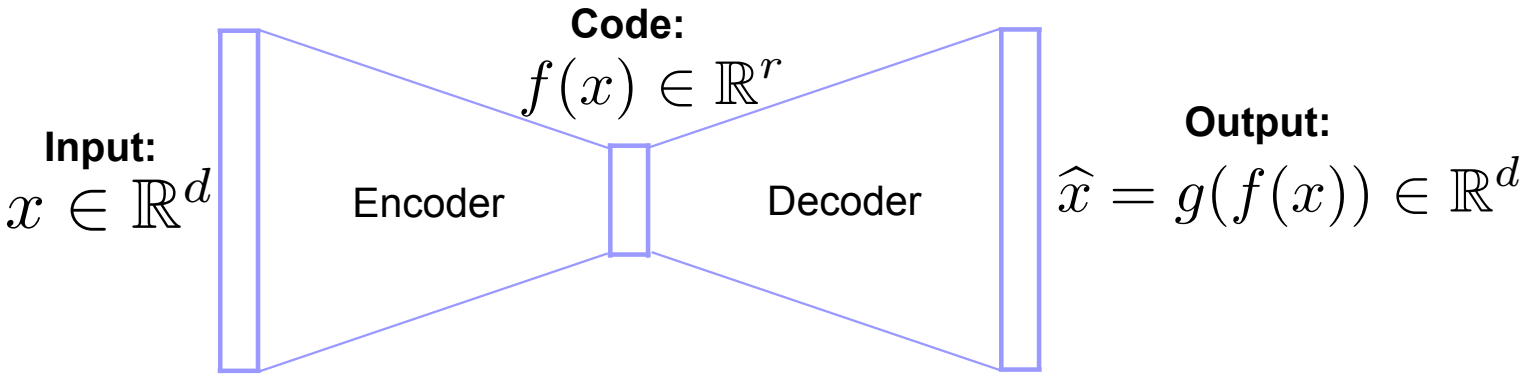
Machine Learning – CSE446

Kevin Jamieson
University of Washington

June 3, 2019

Autoencoders

Find a low dimensional representation for your data by predicting your data

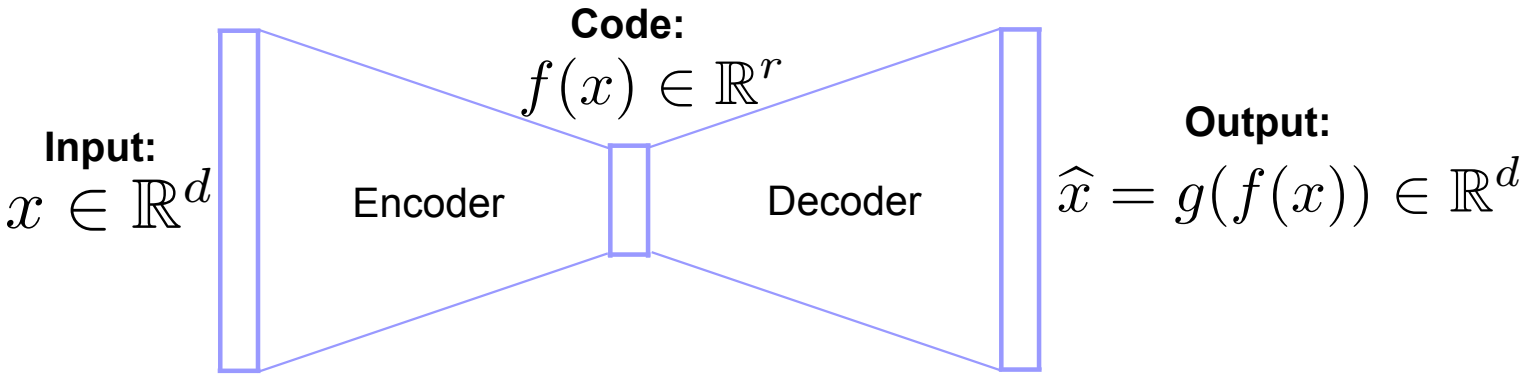


$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

Three major applications
1. De-noising
2. Feature extraction
3. Manifold learning

Autoencoders

Find a low dimensional representation for your data by predicting your data

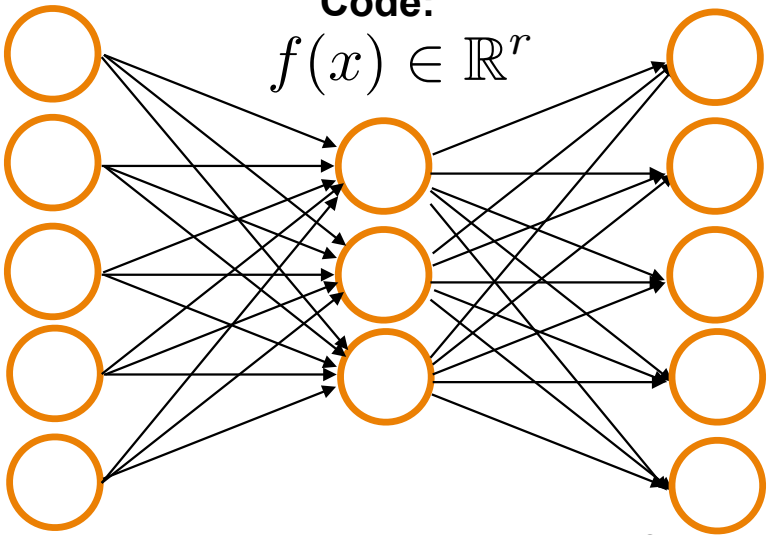


$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

Sparse Autoencoder

$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2 + \lambda \|f(x_i)\|_1$$

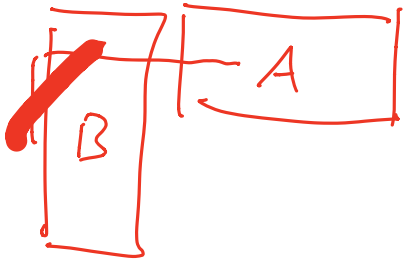
Autoencoders



Code:
$f(x) \in \mathbb{R}^r$

Input:
$x \in \mathbb{R}^d$

Output:
$\hat{x} = g(f(x)) \in \mathbb{R}^d$



$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

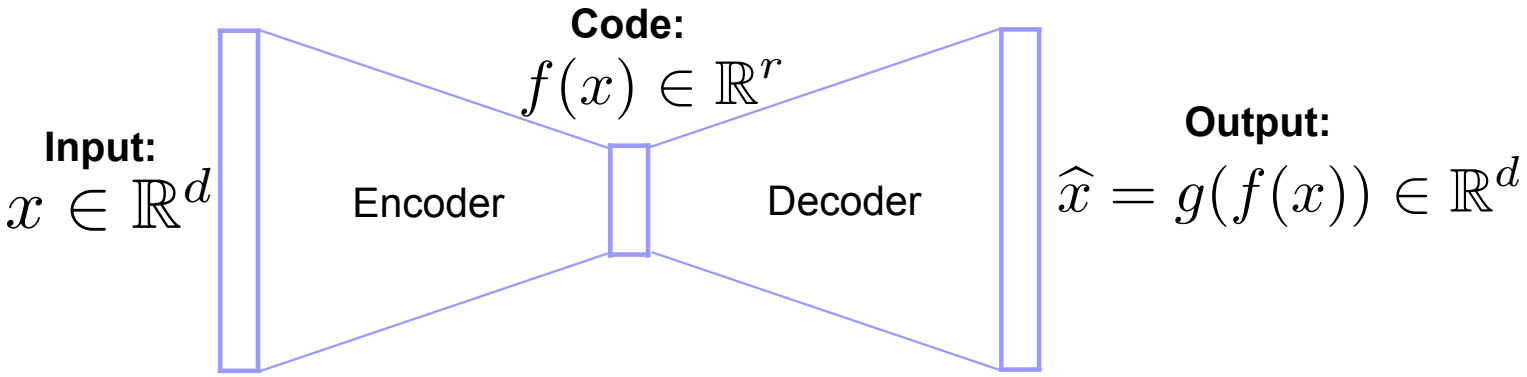
$\|x_i - BAx_i\|_2^2$

What if $f(X) = Ax$ and $g(y) = By$?

$\|x_i - VV^T x_i\|_2^2$

Autoencoders

Find a low dimensional representation for your data by predicting your data



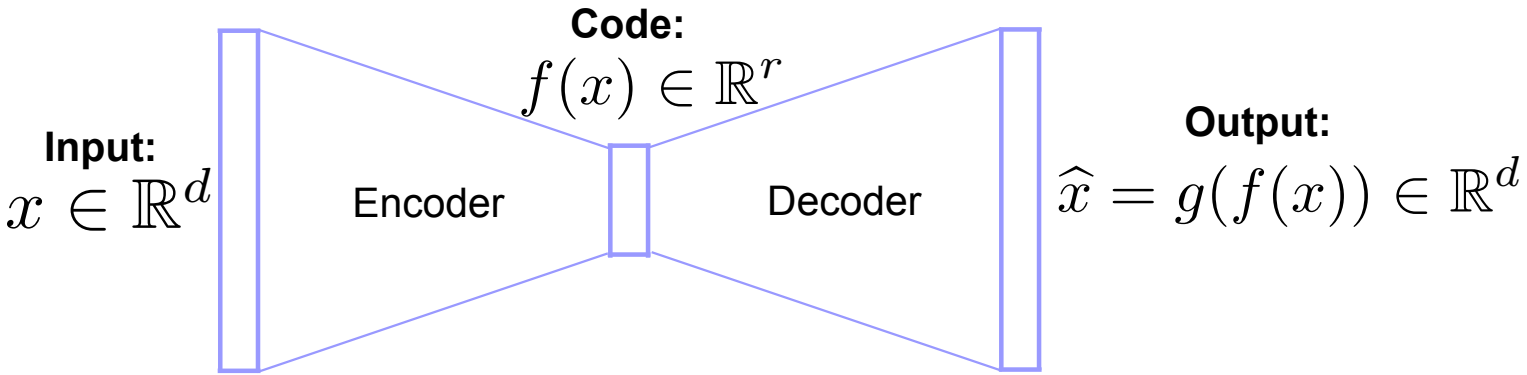
$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

Three major applications
1. De-noising
2. Feature extraction
3. Manifold learning

Related application: *Generating new samples (see autoencoder)*

GAN (Generative Adversarial Networks)

Find a low dimensional representation for your data by predicting your data



$$\underset{f,g}{\text{minimize}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|_2^2$$

Three major applications
1. De-noising
2. Feature extraction
3. Manifold learning

Related application: *Generating new samples* (see variational autoencoders (VAE) or generative adversarial networks (GAN)

Sequences and Recurrent Neural Networks

Machine Learning – CSE446

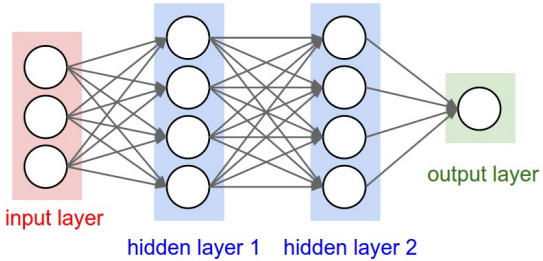
Kevin Jamieson
University of Washington

June 3, 2019

Variable length sequences

Images are usually standardized to be the same size (e.g., 256x256x3)

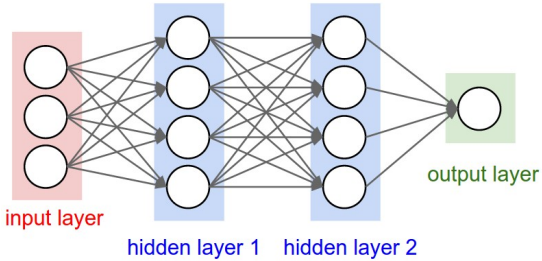
Neural Network



Variable length sequences

Images are usually standardized to be the same size (e.g., 256x256x3)

Neural Network



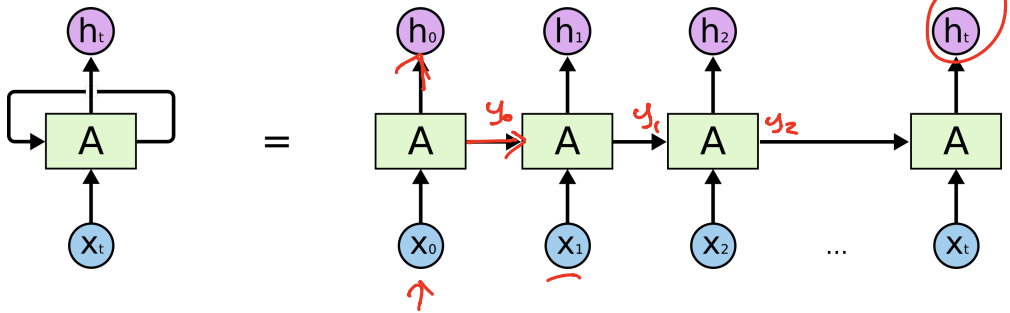
$$y_t = \sigma(W_1 x_t + W_2 y_{t-1})$$
$$h_t = \sigma(W_3 x_t + W_4 y_{t-1})$$

But what if we wanted to do classification on country-of-origin for names?

Hinton →

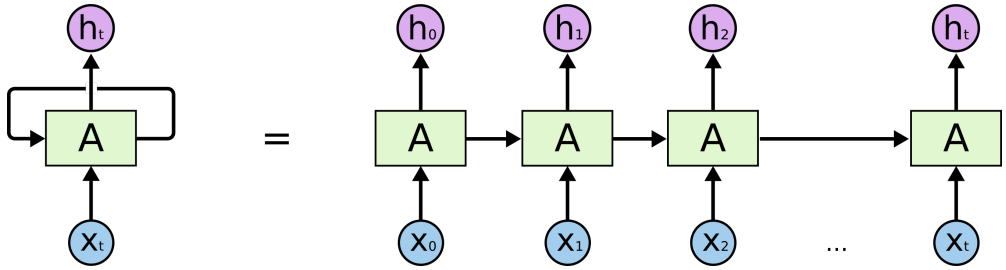
Scottish
English
Irish

Recurrent Neural Network

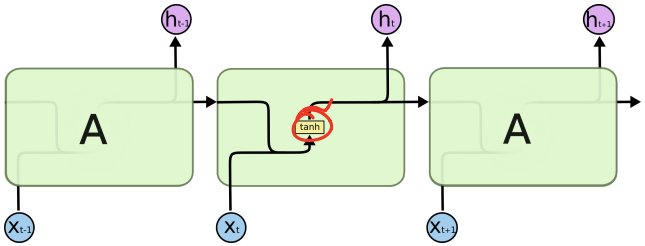


Variable length sequences

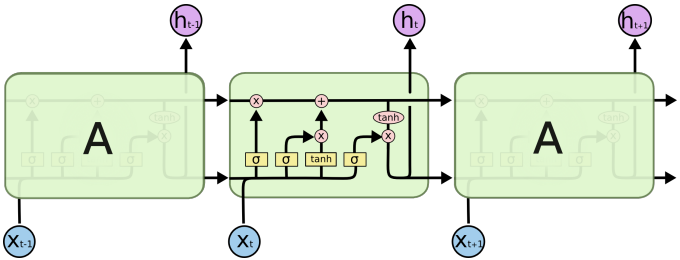
Recurrent Neural Network

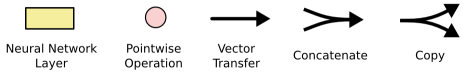


Standard RNN



Gated RNN
LSTM





Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/