Poster Session

T 12/4, 4-7pm

W 12/12, 4:30-7:30

Feature generation for images

Machine Learning – CSE4546
Kevin Jamieson
University of Washington

November 27, 2018

©Kevin Jamieson

Contains slides from…

- LeCun & Ranzato
- Russ Salakhutdinov
- Honglak Lee
- Google images…

Convolution of images

(Note to EEs: deep learning uses the word “convolution” to mean what is usually known as “cross-correlation”, e.g., neither signal is flipped)

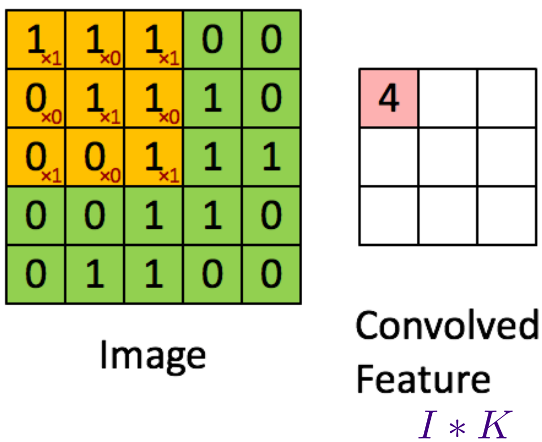
$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

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



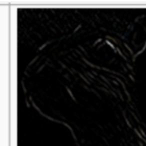
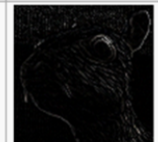
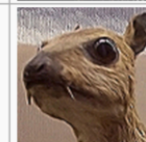
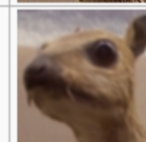
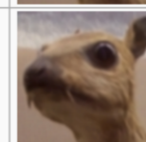
Slide credit: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Convolution of images

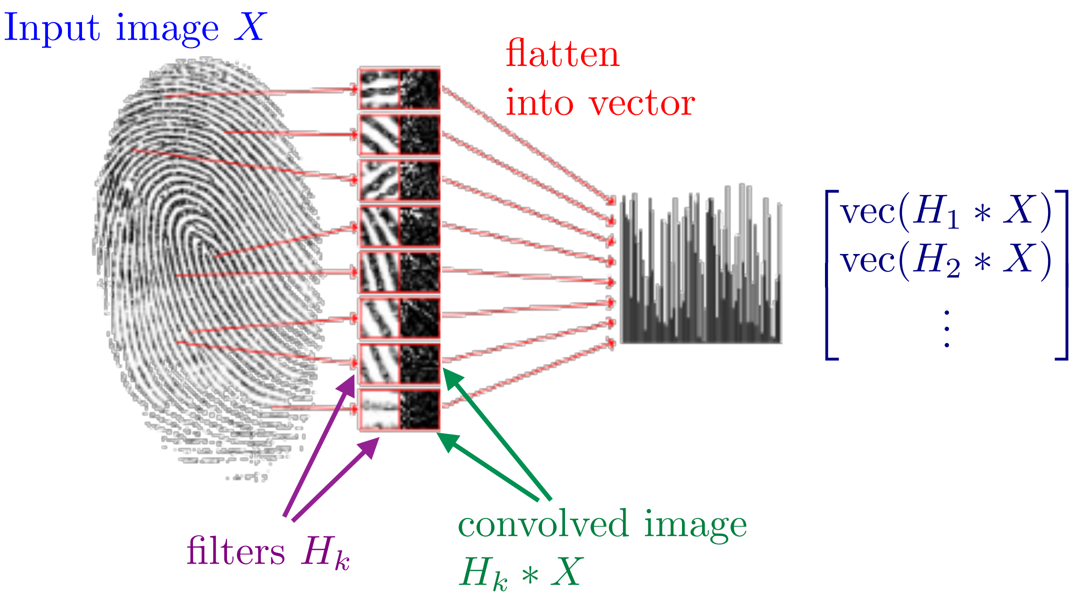
(Note to EEs: deep learning uses the word "convolution" to mean what is usually known as "cross-correlation", e.g., neither signal is flipped)

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

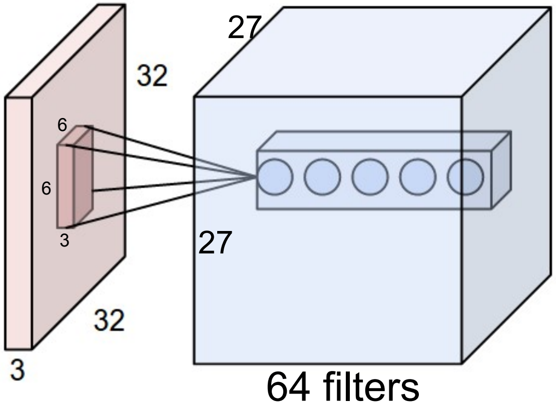
Image I

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)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

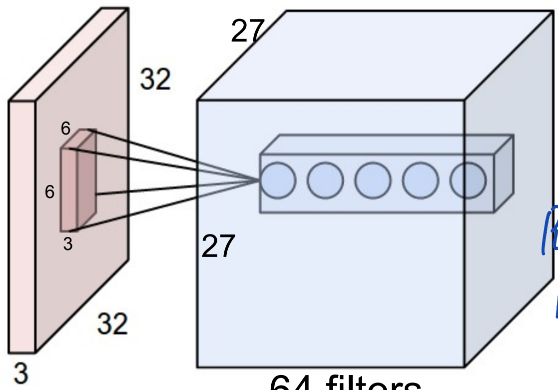
Convolution of images

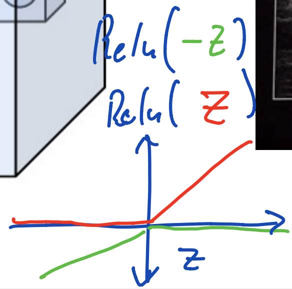


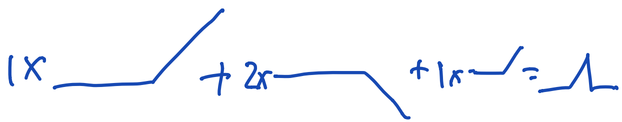
Stacking convolved images



Stacking convolved images

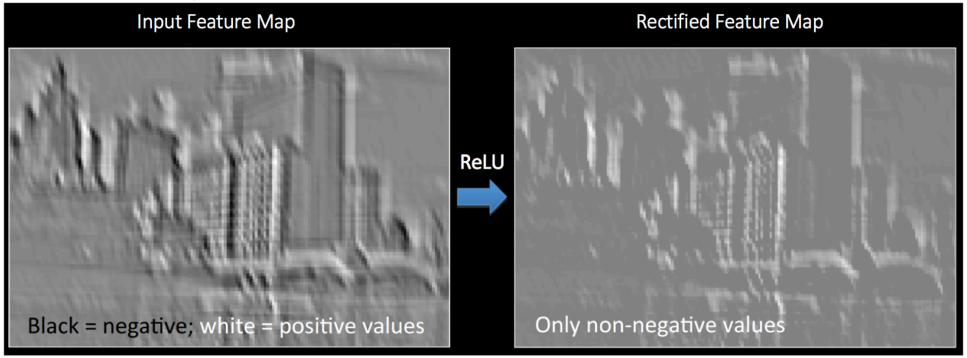






Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)

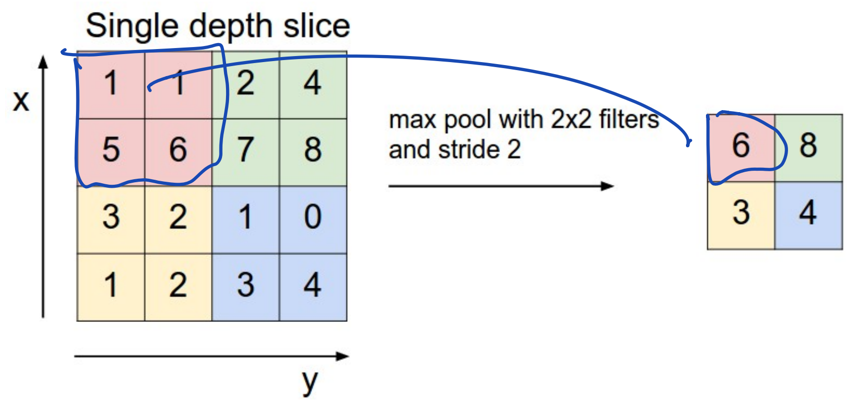
$\mathrm{Relu}(x) = \max\{0, x\}$

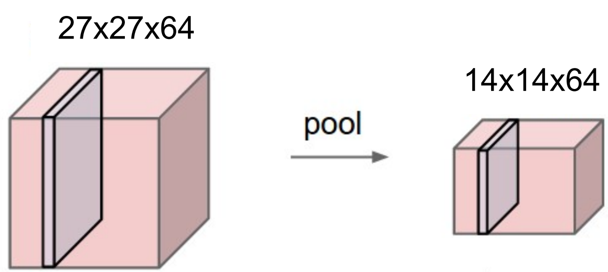


Other choices: sigmoid, arctan

Pooling

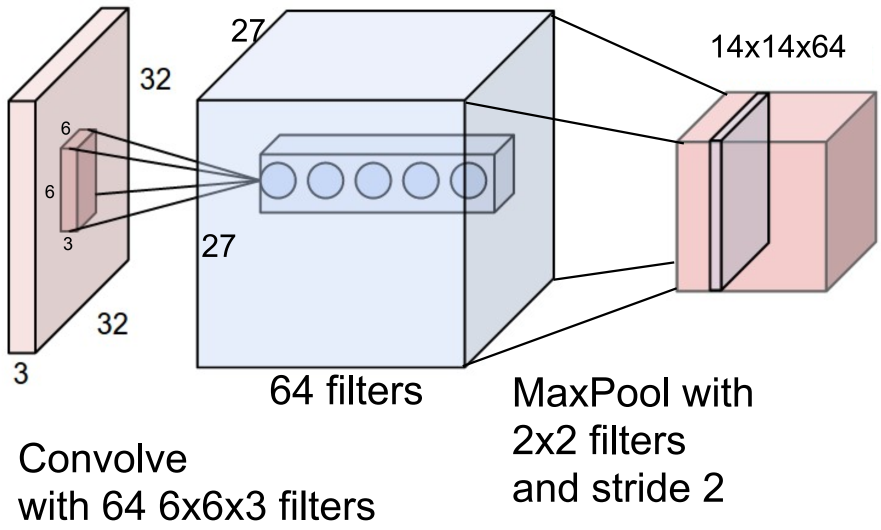
Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”



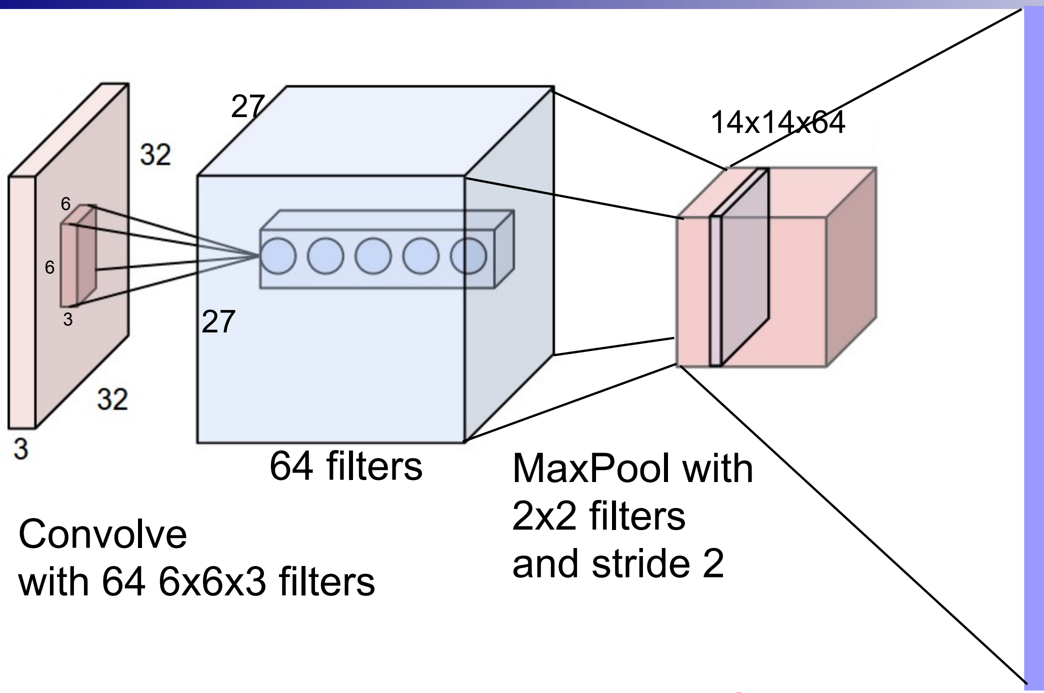


©Kevin Jamieson

Pooling Convolution layer



Full feature pipeline



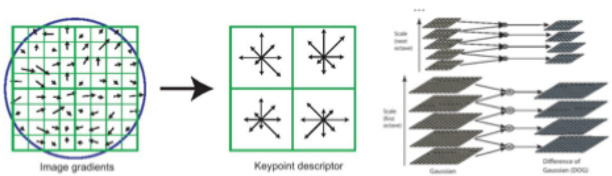
Flatten into a single vector of size 14*14*64=12544

How do we choose all the hyperparameters?

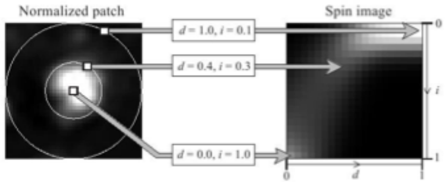
How do we choose the filters?
- Hand design them (digital signal processing, c.f. wavelets)
- Learn them (deep learning)

©Kevin Jamieson

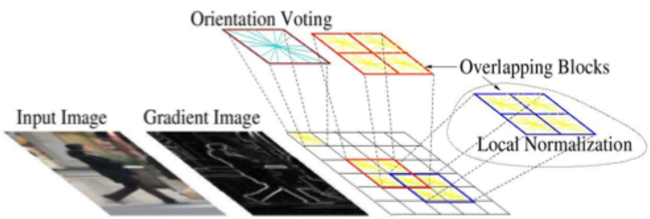
Some hand-created image features



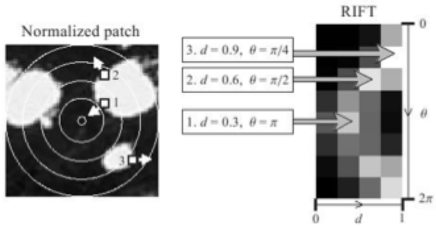
SIFT



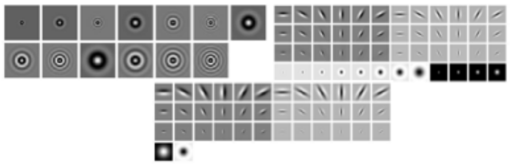
Spin Image



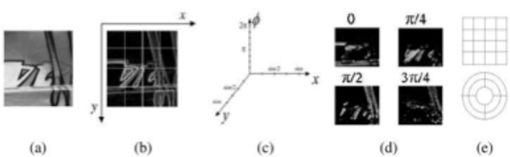
HoG



RIFT



Texton



GLOH

Slide from Honglak Lee

ML Street Fight

Machine Learning – CSE546
Kevin Jamieson
University of Washington

November 27, 2018

©Kevin Jamieson 2018

Mini case study

Inspired by Coates and Ng (2012)

Input is CIFAR-10 dataset: 50000 examples of 32x32x3 images

1. Construct set of patches by random selection from images
2. Standardize patch set (de-mean, norm 1, whiten, etc.)
3. Run k-means on random patches
4. Convolve each image with all patches (plus an offset)
5. Push through ReLu
6. Solve least squares for multiclass classification
7. Classify with argmax

©Kevin Jamieson

Mini case study

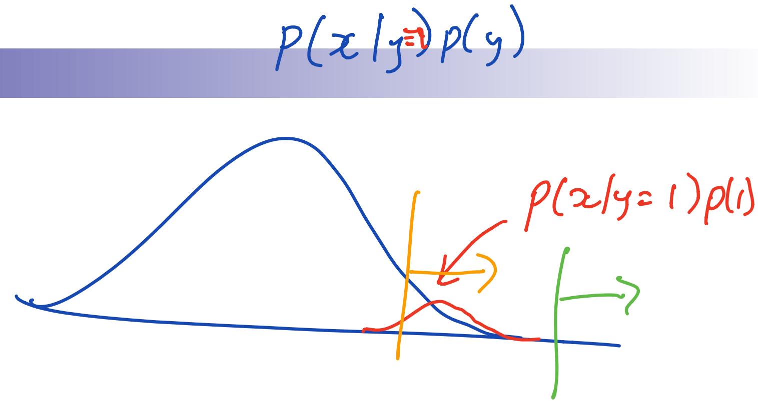
Methods of standardization:

$$\frac{x_{i,j} - \mu_j}{\sigma_j} \quad \forall i \qquad \mu_j = \frac{1}{n} \sum_i x_{i,j}$$

$$\|x_i\|_2 = 1$$

PCA, non-linear dim reduction.

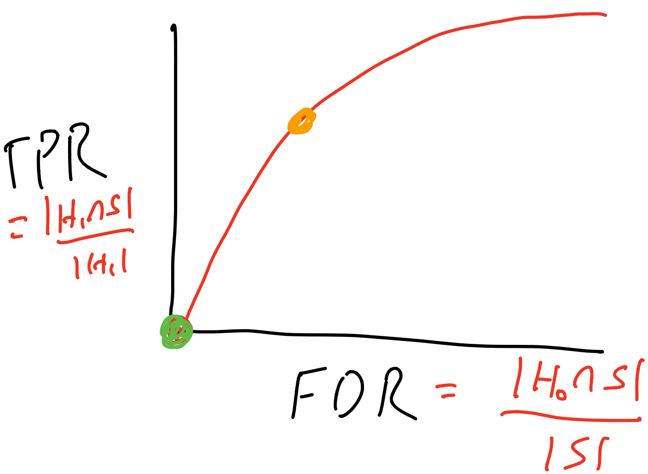
Mini case study



Dealing with class imbalance:

$$\widehat{w} = \arg\min_w \sum_{i=1}^n \max\{0, 1 - y_i \underbrace{x_i^T w}_{\hat{y}_i}\}$$

Calibrate with ROC curve:



©Kevin Jamieson

Mini case study

Dealing with class imbalance:

$$\widehat{w} = \arg\min_w \sum_{i=1}^n \max\{0, 1 - y_i x_i^T w\}$$

Change to weighted objective:

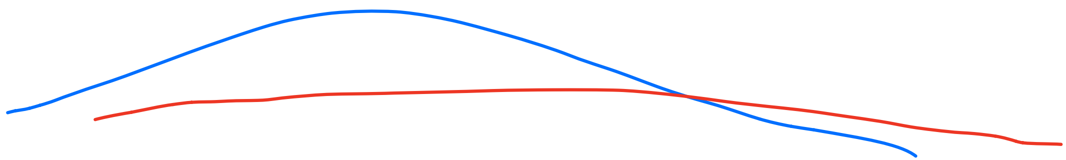
$$\widehat{w}_\alpha = \arg\min_w \sum_{i=1}^n \tfrac{1}{\alpha}\mathbf{1}\{y_i = 1\}\max\{0, 1 - x_i^T w\} + \tfrac{1}{1-\alpha}\mathbf{1}\{y_i = -1\}\max\{0, 1 + x_i^T w\}$$

$$\text{Natural choice: } \alpha = \frac{\sum_{i=1}^n \mathbf{1}\{y_i = 1\}}{n}$$

Mini case study

Dealing with class imbalance:

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} \max\{0, 1 - y_i x_i^T w\}$$



Change to optimization procedure:

$$w_{t+1} = w_t - \gamma \nabla \ell((x_{I_t}, y_{I_t}), w_t)$$

$$\mathbb{P}(I_t = i | y_i = 1) \propto \frac{1}{\alpha}$$

$$\mathbb{P}(I_t = i | y_i = -1) \propto \frac{1}{1-\alpha}$$

$$\text{Natural choice: } \alpha = \frac{\sum_{i=1}^n \mathbf{1}\{y_i = 1\}}{n}$$

©Kevin Jamieson

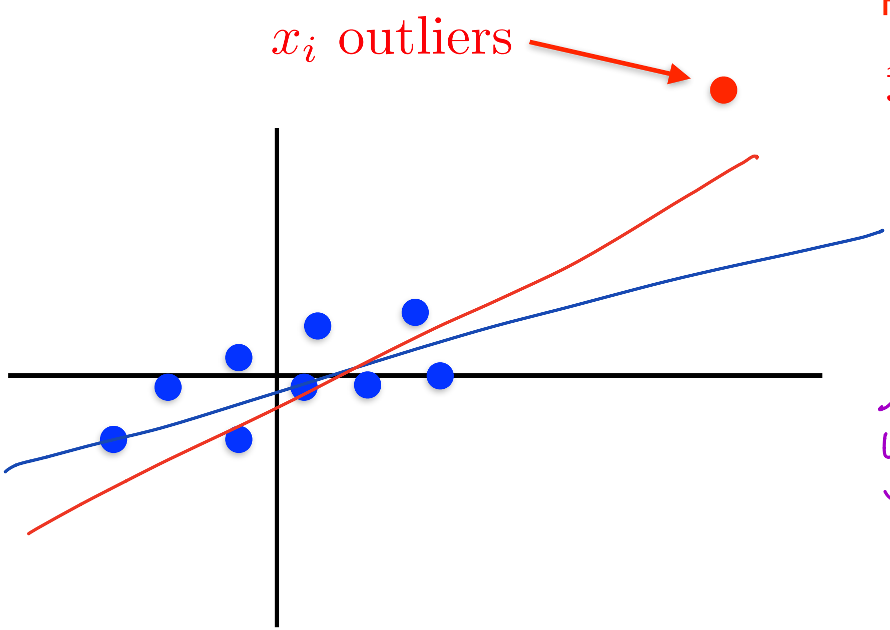
Mini case study

$X = USV^T$

Dealing with outliers:

$w_{t+1} = w_t + \gamma x_i (y_i - x_i^T w_t)$

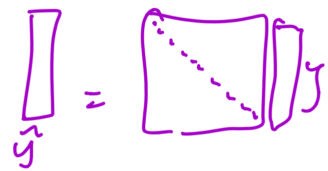
Observe $\{(x_i, y_i)\}_{i=1}^n$. $\widehat{w} = \arg\min_w \|\mathbf{X}w - \mathbf{y}\|^2 = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$



Have undue influence on w

$\widehat{\mathbf{y}} = \mathbf{X}\widehat{w} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$

$= \mathbf{U}\mathbf{U}^T\mathbf{y}$

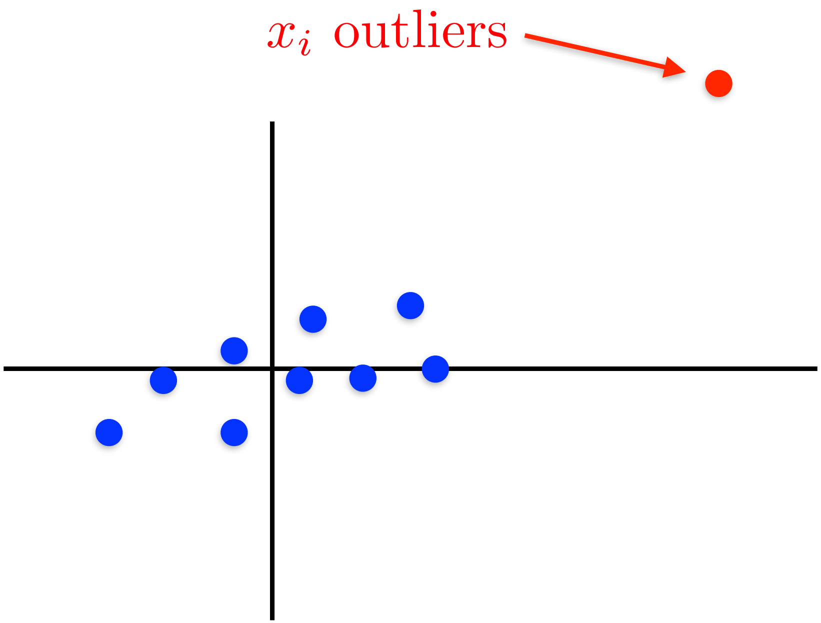


$\hat{y}_i = \sum_j (UU^T)_{i,j} \, y_j$

Mini case study

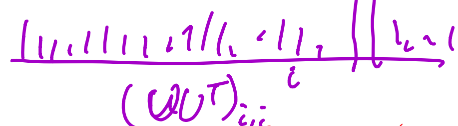
Dealing with outliers:

Observe $\{(x_i, y_i)\}_{i=1}^n$. $\widehat{w} = \arg\min_w \|\mathbf{X}w - \mathbf{y}\|^2 = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$



Have undue influence on w

$$\widehat{\mathbf{y}} = \mathbf{X}\widehat{w} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$
$$= \mathbf{U}\mathbf{U}^T\mathbf{y}$$



Leverage score for x_i: $(\mathbf{U}\mathbf{U}^T)_{i,i}$

Remove points with large leverage scores from dataset

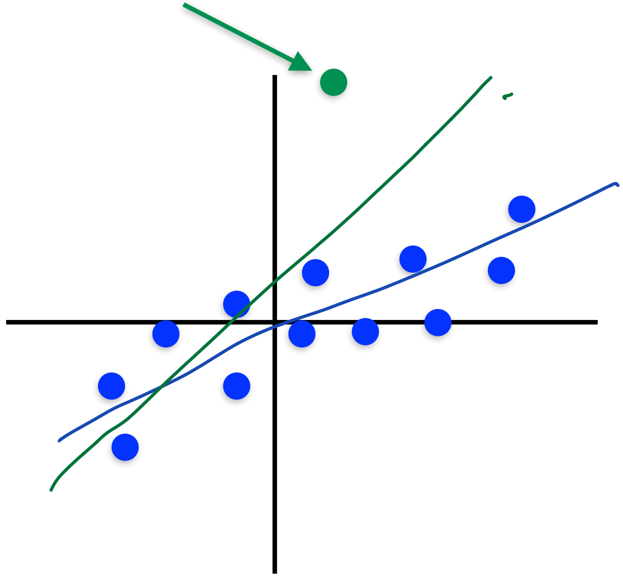
©Kevin Jamieson

Mini case study

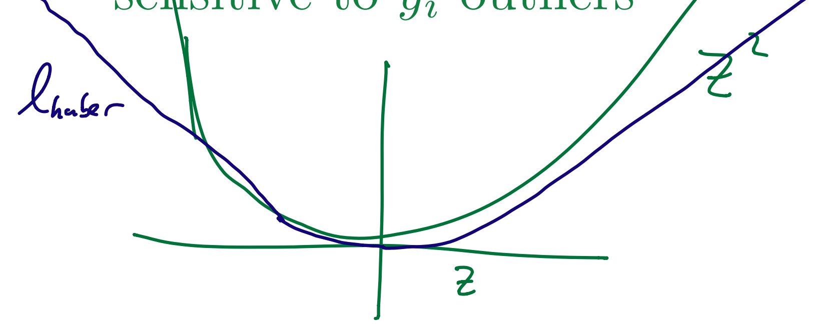
Dealing with outliers:

Observe $\{(x_i, y_i)\}_{i=1}^n$. $\widehat{w} = \arg\min_w \|\mathbf{X}w - \mathbf{y}\|^2 = \sum_{i=1}^n (x_i^T w - y_i)^2$

y_i outliers



Loss $\ell_{LS}(z) = z^2$ is very sensitive to y_i outliers



$$\ell_{huber}(z) = \begin{cases} \frac{1}{2}z^2 & \text{if } |z| \leq 1 \\ |z| - \frac{1}{2} & \text{otherwise} \end{cases}$$

©Kevin Jamieson

Mini case study

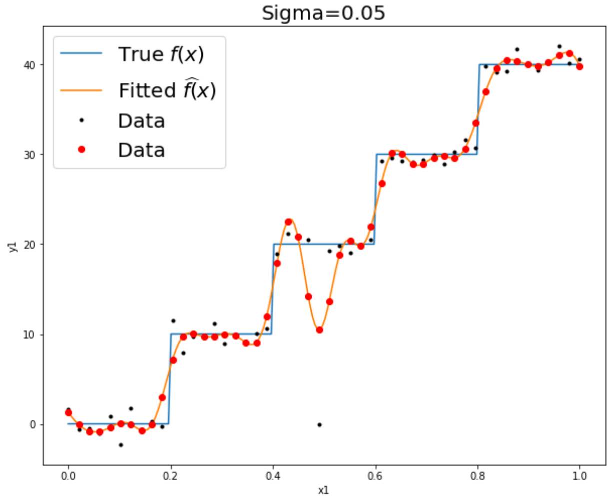
Dealing with outliers:

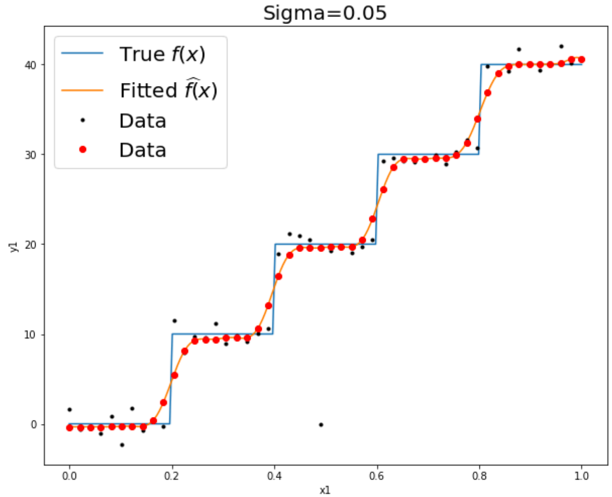
$$\ell_{huber}(z) = \begin{cases} \frac{1}{2}z^2 & \text{if } |z| \leq 1 \\ |z| - \frac{1}{2} & \text{otherwise} \end{cases}$$

y_i outliers

$$\arg\min_{\alpha} \sum_{i=1}^{n} \left(\sum_j k(x_i, x_j)\alpha_j - y_i \right)^2 + \lambda \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j)$$

$$\arg\min_{\alpha} \sum_{i=1}^{n} \ell_{huber} \left(\sum_j k(x_i, x_j)\alpha_j - y_i \right) + \lambda \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j)$$





©Kevin Jamieson

Mini case study

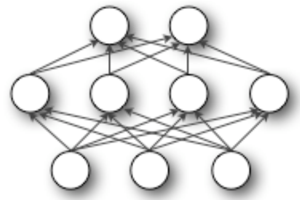
Dealing with hyperparameters:

Hyperparameter Optimization

Machine Learning – CSE546
Kevin Jamieson
University of Washington

November 27, 2018

©Kevin Jamieson 2018
©Kevin Jamieson 2018



$N_{out} = 10$

N_{hid}

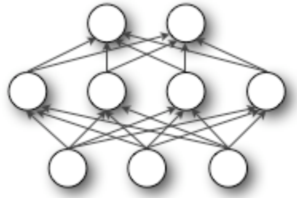
$N_{in} = 784$

hyperparameters

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

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

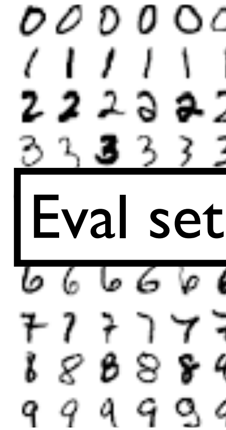
hidden nodes $N_{hid} \in [10^1, 10^3]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

Hyperparameters

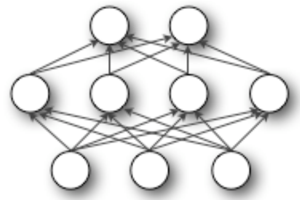
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$

hyperparameters

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

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

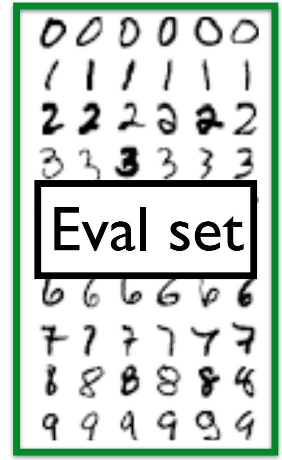
$\#$ hidden nodes $N_{hid} \in [10^{1}, 10^{3}]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



$\widehat{f}$

Hyperparameters

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

Eval-loss

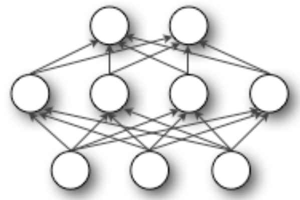
`0.0577`

hyperparameters

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

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

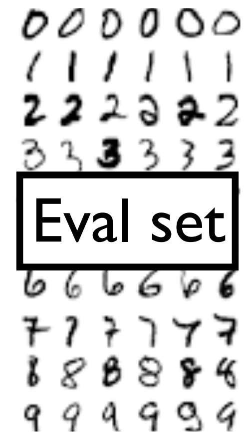
$\#$ hidden nodes $N_{hid} \in [10^{1}, 10^{3}]$



$N_{out} = 10$

N_{hid}

$N_{in} = 784$



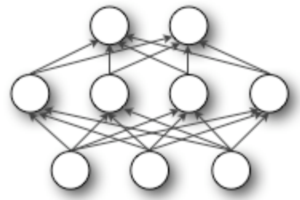
Hyperparameters	Eval-loss
$(10^{-1.6}, 10^{-2.4}, 10^{1.7})$	0.0577
$(10^{-1.0}, 10^{-1.2}, 10^{2.6})$	0.182
$(10^{-1.2}, 10^{-5.7}, 10^{1.4})$	0.0436
$(10^{-2.4}, 10^{-2.0}, 10^{2.9})$	0.0919
$(10^{-2.6}, 10^{-2.9}, 10^{1.9})$	0.0575
$(10^{-2.7}, 10^{-2.5}, 10^{2.4})$	0.0765
$(10^{-1.8}, 10^{-1.4},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

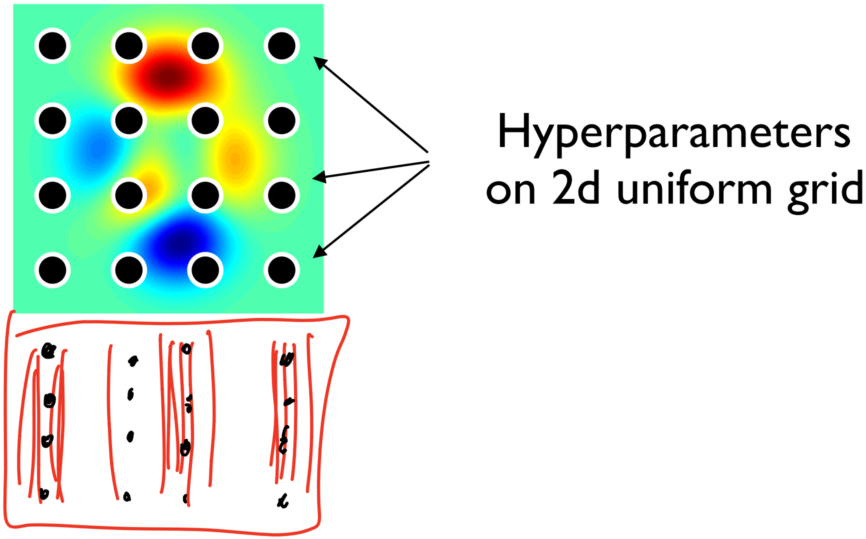
$N_{in} = 784$

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?

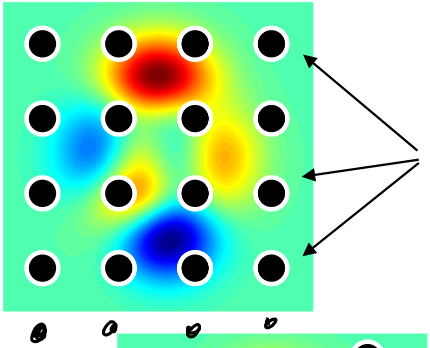
How do we choose hyperparameters to train and evaluate?

Grid search:



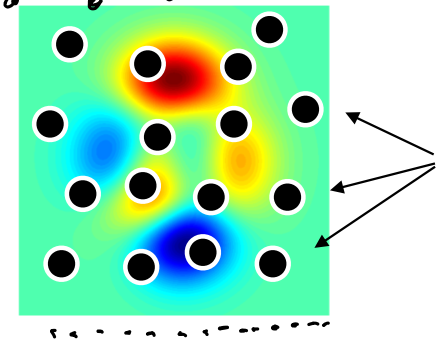
How do we choose hyperparameters to train and evaluate?

Grid search:



Hyperparameters on 2d uniform grid

Random search:

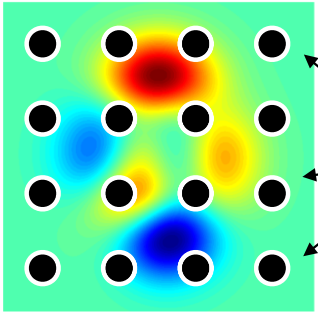


Hyperparameters randomly chosen

Low discrepancy sequence (eg Sobol)

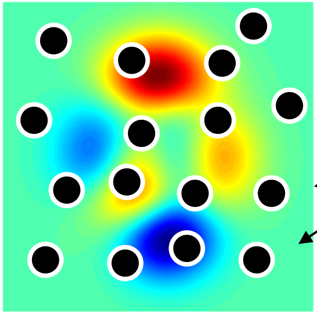
How do we choose hyperparameters to train and evaluate?

Grid search:



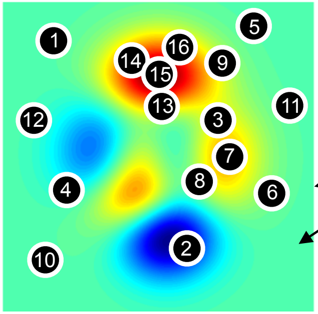
Hyperparameters on 2d uniform grid

Random search:



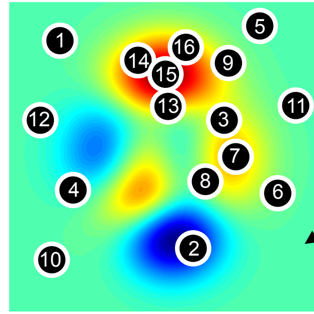
Hyperparameters randomly chosen

Bayesian Optimization:



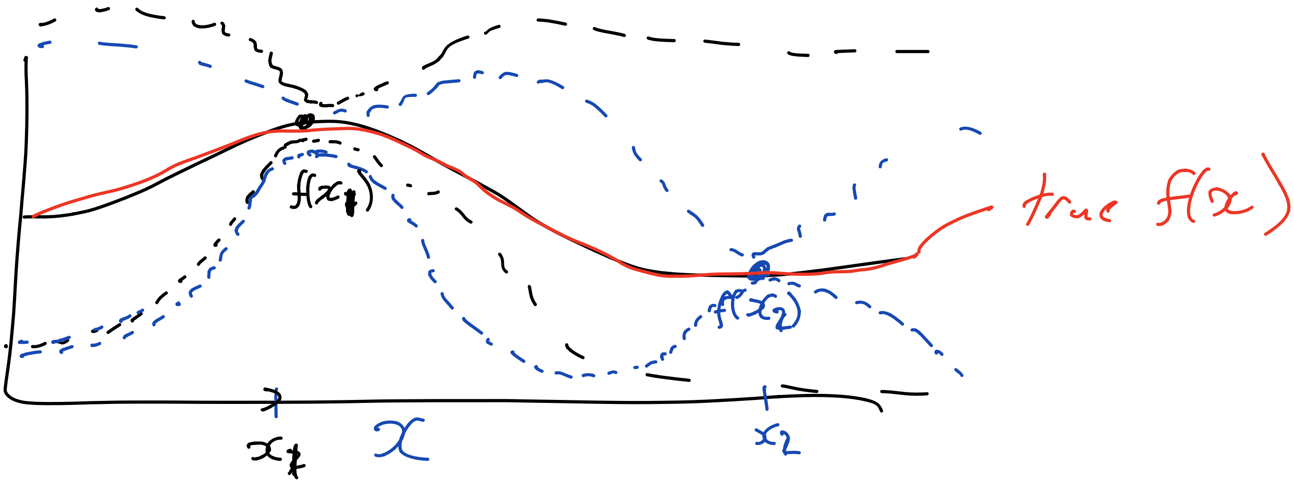
Hyperparameters ***adaptively*** chosen

Bayesian Optimization:



Hyperparameters
adaptively chosen

How does it work?



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.

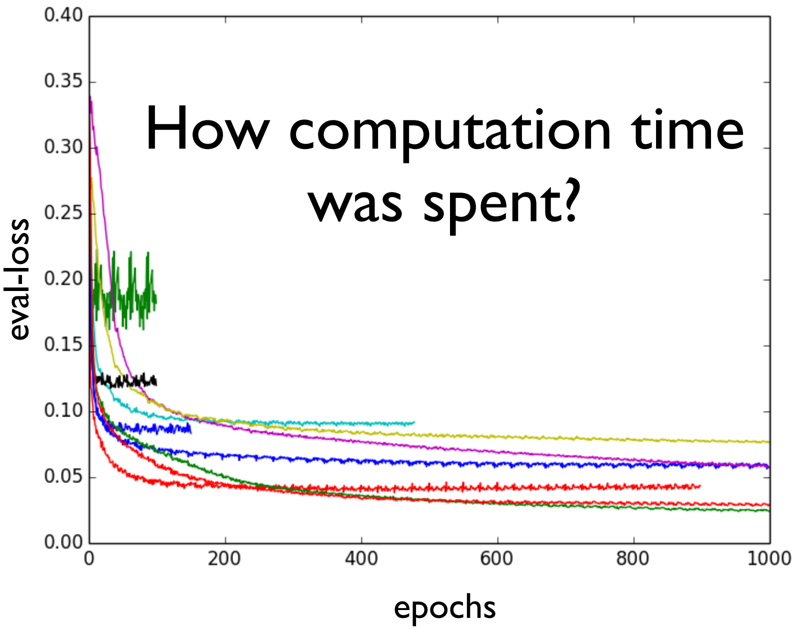
Alekh Agarwal, Peter Bartlett, and John Duchi. Oracle inequalities for computationally adaptive model selection. COLT, 2012.

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.

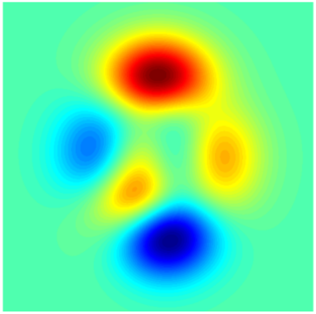
Li, Jamieson, DeSalvo, Rostamizadeh, Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. ICLR 2016.

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. Use crowd resources (see `pywren`)

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?

Neural Networks

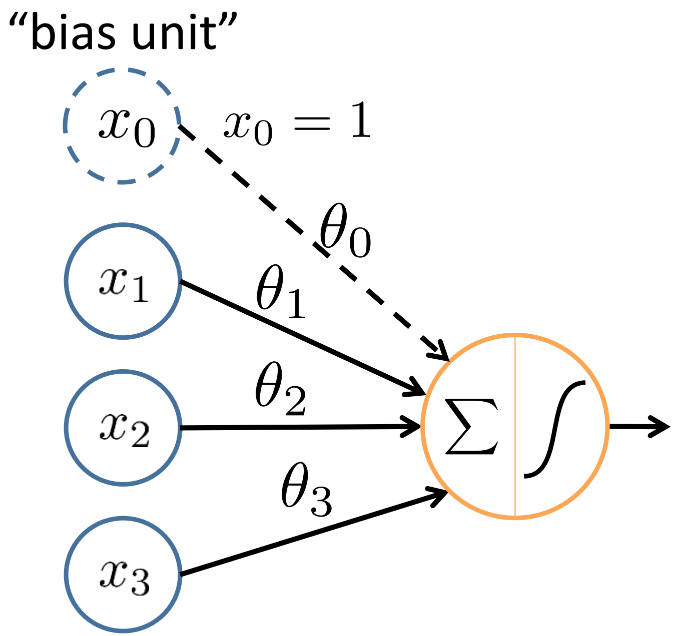
These slides were assembled by Eric Eaton, with grateful acknowledgement of the many others who made their course materials freely available online. Feel free to reuse or adapt these slides for your own academic purposes, provided that you include proper attribution. Please send comments and corrections to Eric.

Robot Image Credit: Viktoriya Sukhanova © 123RF.com

Neural Networks

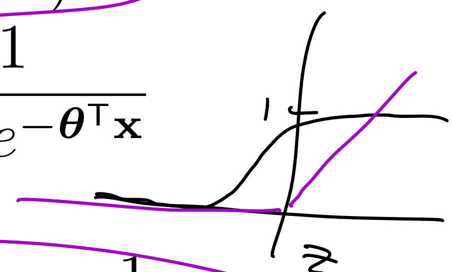
- Origins: Algorithms that try to mimic the brain
- 40s and 50s: Hebbian learning and Perceptron
- Perceptrons book in 1969 and the XOR problem
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

Single Node



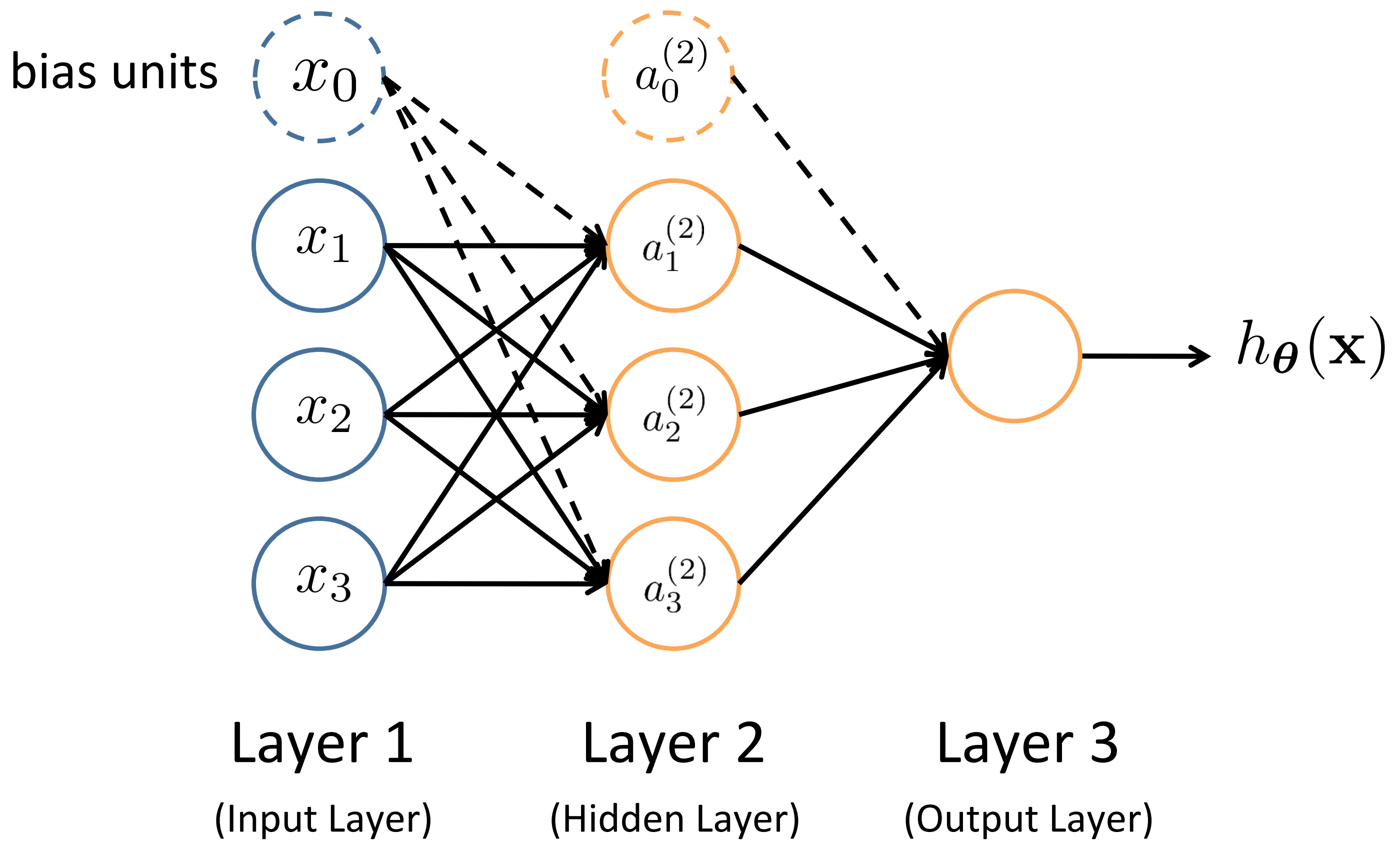
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

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

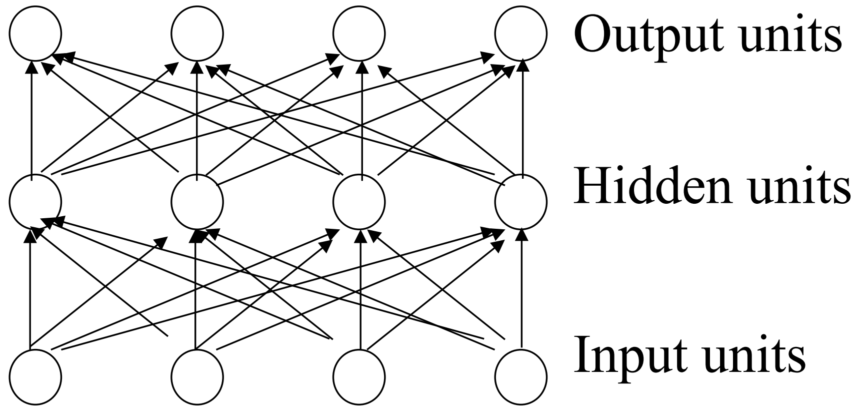


Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Neural Network



Neural networks Terminology



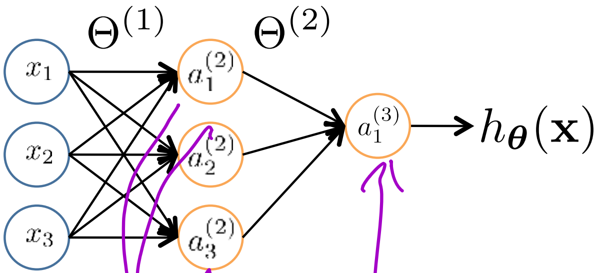
Layered feed-forward network

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

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Feed-Forward Process

- Input layer units are set by external data, which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node



$a_i^{(j)}$ = "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 $s_{j+1} \times (s_j+1)$.

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

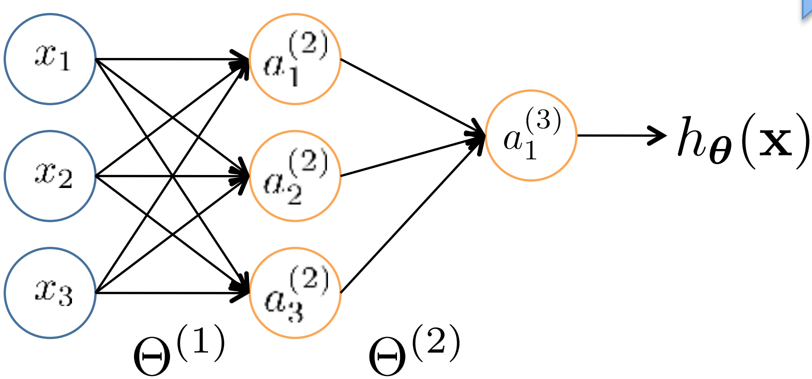
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

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

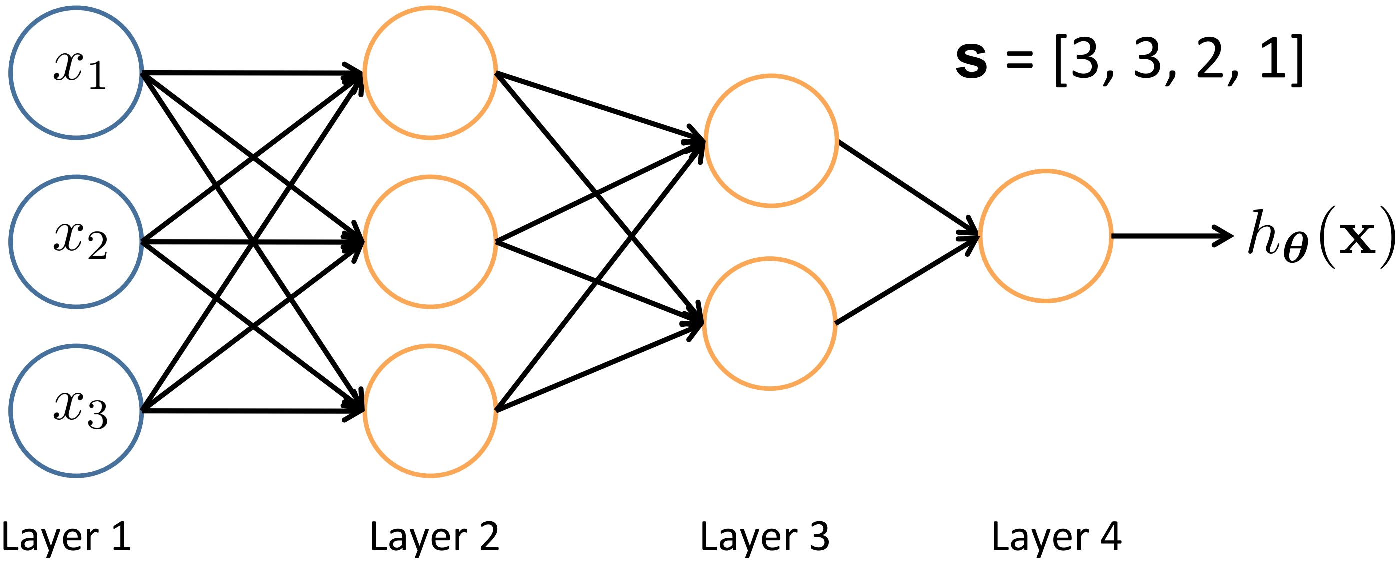
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Other Network Architectures



L denotes the number of layers

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

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1}=K$ (# classes)

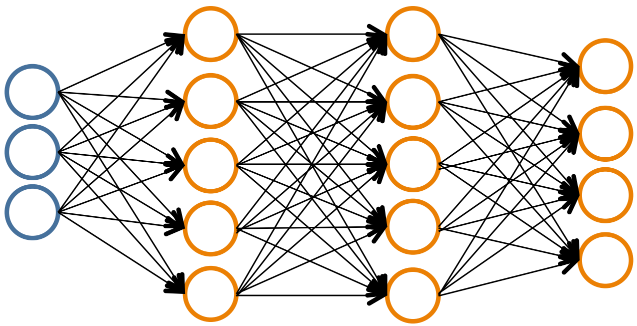
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



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

We want:

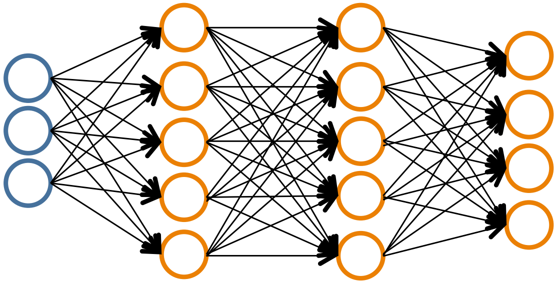
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
when pedestrian

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

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

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

Multiple Output Units: One-vs-Rest



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

We want:

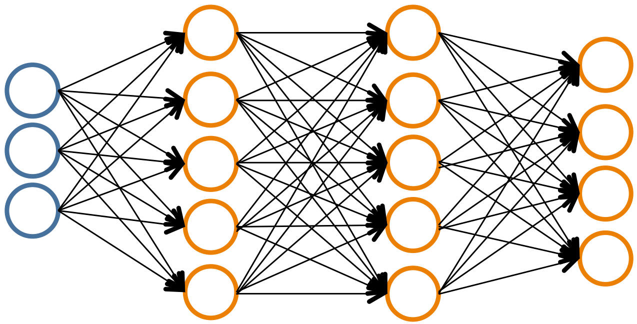
$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ when pedestrian $h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car $h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle $h_\Theta(\mathbf{x}) \approx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

---

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

Based on slide by Andrew Ng

Neural Network Classification



Given:

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

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

– $s_0 = d$ (# features)

<u>Binary classification</u>

y = 0 or 1

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

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

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

pedestrian car motorcycle truck

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

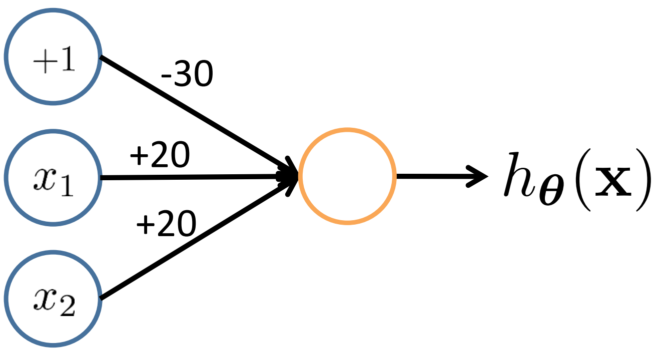
Understanding Representations

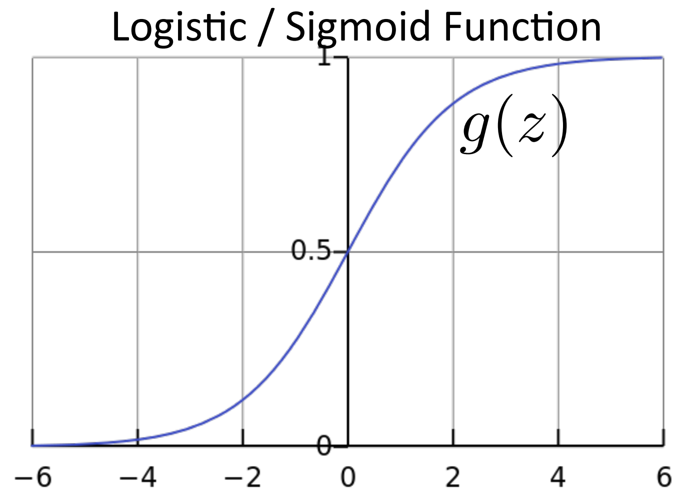
Representing Boolean Functions

Simple example: AND

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

$y = x_1 \text{ AND } x_2$

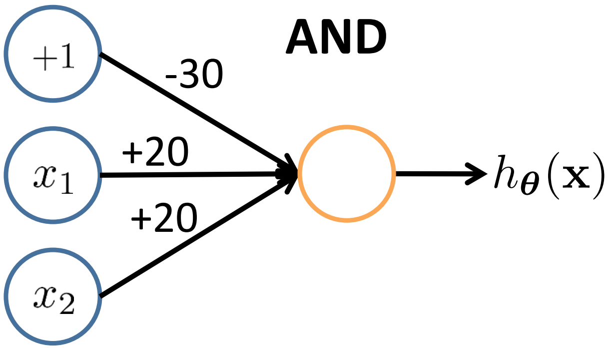


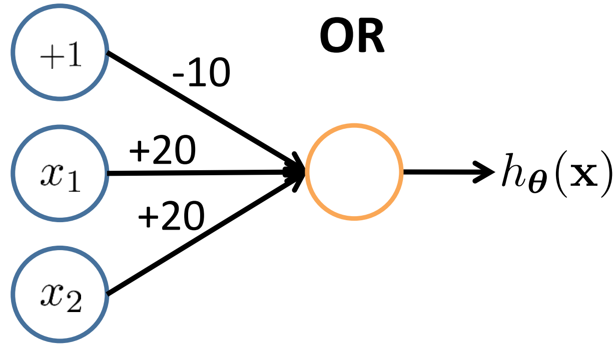


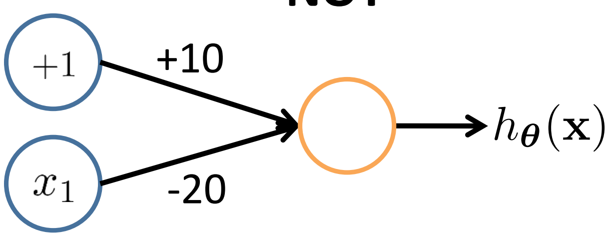
$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$

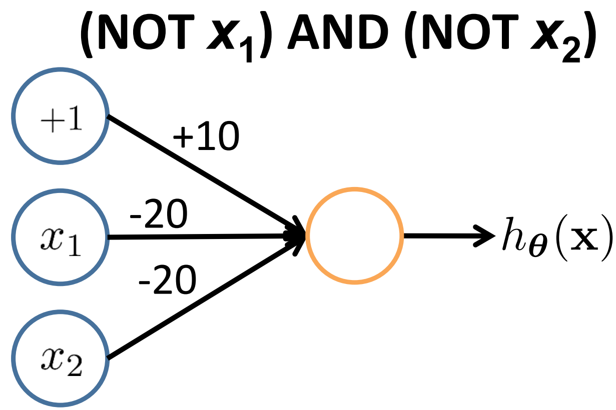
x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Representing Boolean Functions

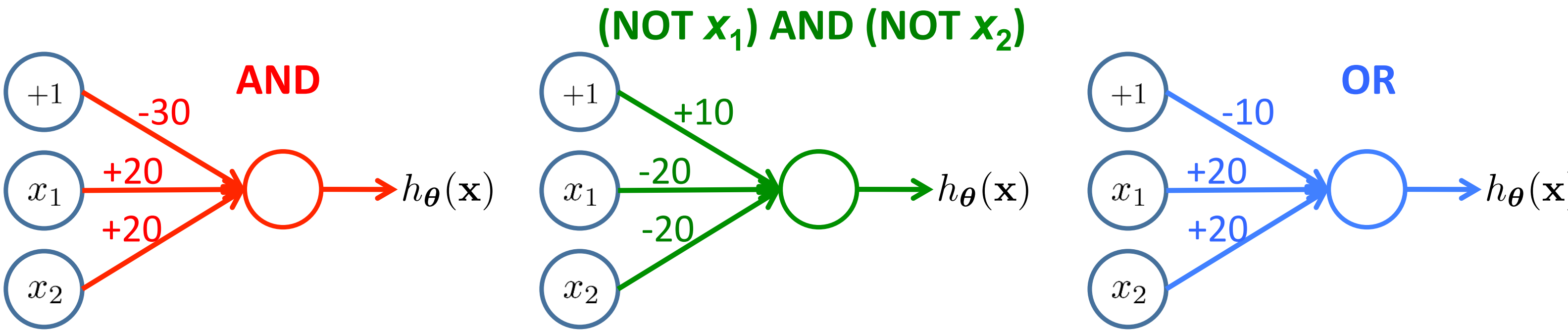


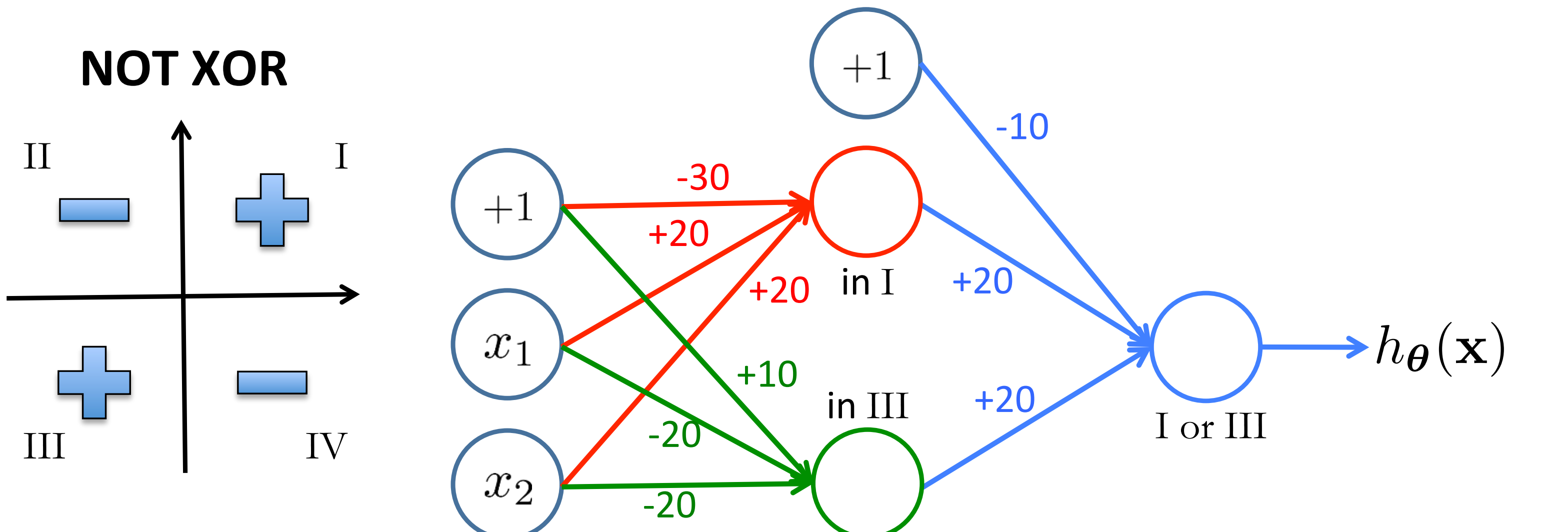






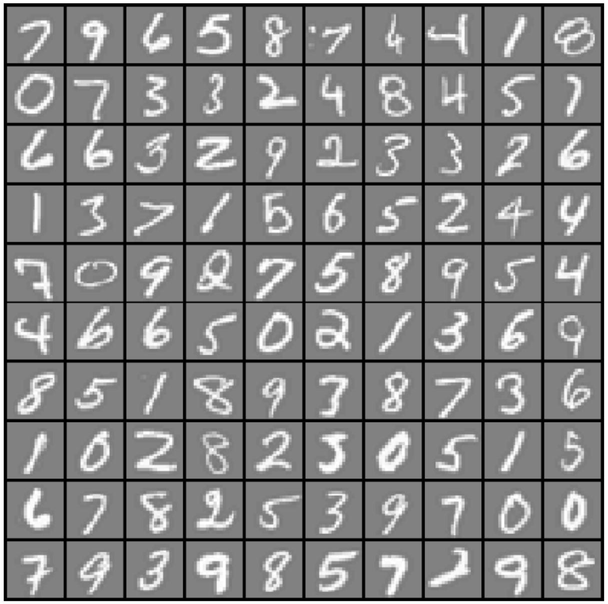
Combining Representations to Create Non-Linear Functions





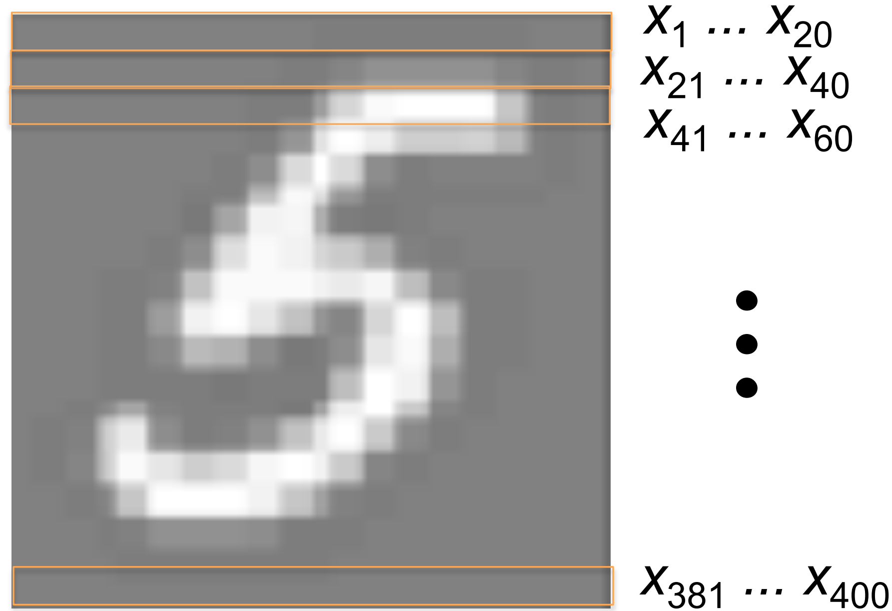
Based on example by Andrew Ng

Layering Representations



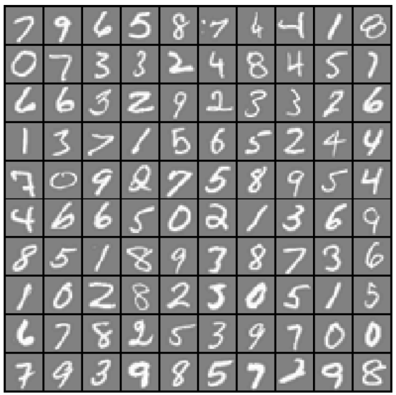
20 × 20 pixel images

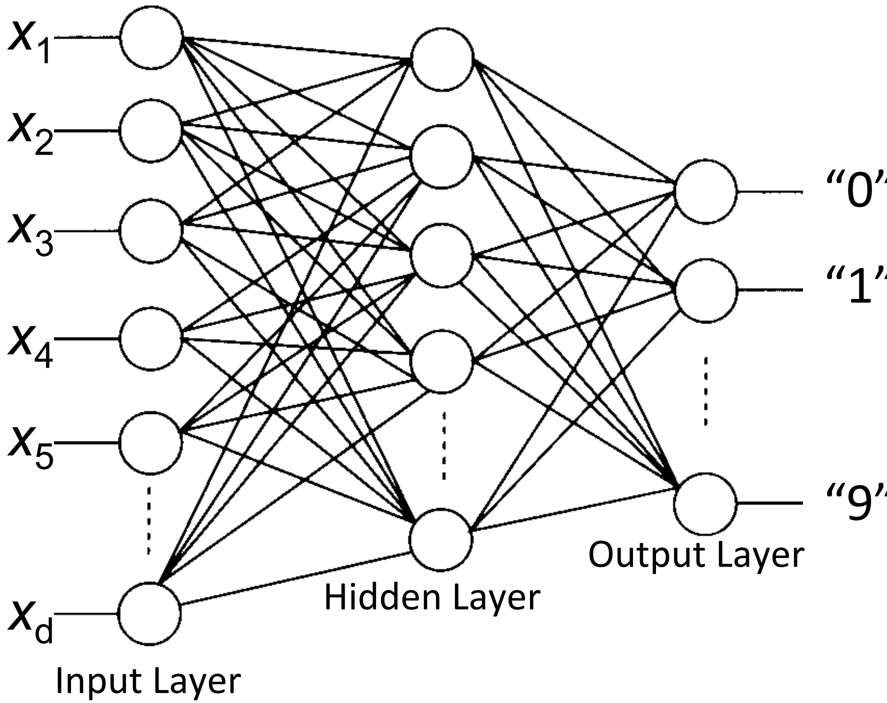
d = 400   10 classes

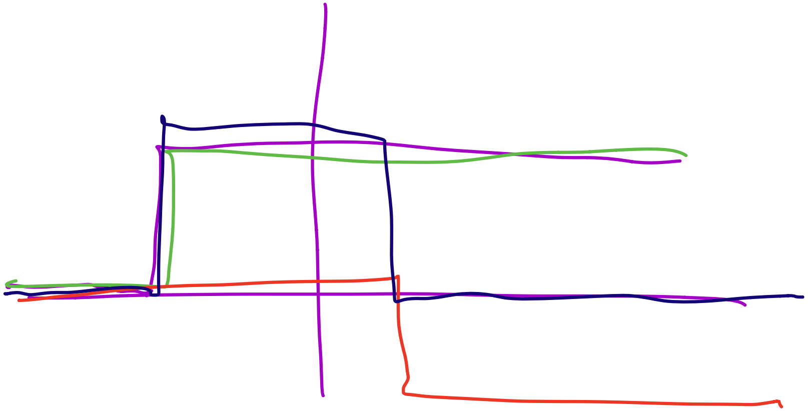


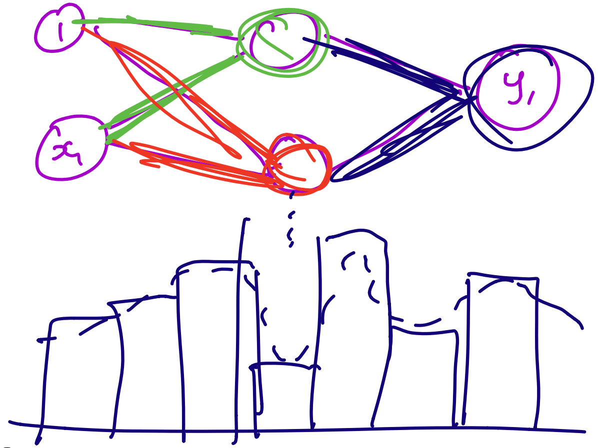
Each image is "unrolled" into a vector **x** of pixel intensities

Layering Representations









Neural Network Learning

Perceptron Learning Rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(y - h(\mathbf{x}))\mathbf{x}$$

Intuitive rule:

- If output is correct, don't change the weights
- If output is low ($h(\mathbf{x}) = 0$, $y = 1$), increment weights for all the inputs which are 1
- If output is high ($h(\mathbf{x}) = 1$, $y = 0$), decrement weights for all inputs which are 1

Perceptron Convergence Theorem:

- If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge [Minksy & Papert, 1969]

Batch Perceptron

Given training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
    Let $\boldsymbol{\Delta} \leftarrow [0, 0, \ldots, 0]$
    for $i = 1 \ldots n$, do
        if $y^{(i)} \boldsymbol{x}^{(i)} \boldsymbol{\theta} \leq 0$ // prediction for ith instance is incorrect
            $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} + y^{(i)} \boldsymbol{x}^{(i)}$
    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} / n$ // compute average update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{\Delta}$
Until $\|\boldsymbol{\Delta}\|_2 < \epsilon$

- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Each increment of outer loop is called an **epoch**

Learning in NN: Backpropagation

- Similar to the perceptron learning algorithm, we cycle through our examples
  - If the output of the network is correct, no changes are made
  - If there is an error, weights are adjusted to reduce the error

- We are just performing (stochastic) gradient descent!

Cost Function

Logistic Regression:

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i)\log\left(1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i)\right)\right] + \frac{\lambda}{2n}\sum_{j=1}^{d}\theta_j^2$$

Neural Network:

$$h_\Theta \in \mathbb{R}^K \qquad (h_\Theta(\mathbf{x}))_i = i^{th}\text{output}$$

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log\left(h_\Theta(\mathbf{x}_i)\right)_k + (1 - y_{ik})\log\left(1 - (h_\Theta(\mathbf{x}_i))_k\right)\right]$$

$$+ \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

k^{th} class: true, predicted
not k^{th} class: true, predicted

Based on slide by Andrew Ng

Optimizing the Neural Network

$$J(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\Theta(\mathbf{x}_i))_k + (1-y_{ik})\log\Big(1-(h_\Theta(\mathbf{x}_i))_k\Big)\right]$$
$$+\frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Solve via: $\min_{\Theta} J(\Theta)$

Unlike before, $J(\Theta)$ is not convex, so GD on a neural net yields a local optimum

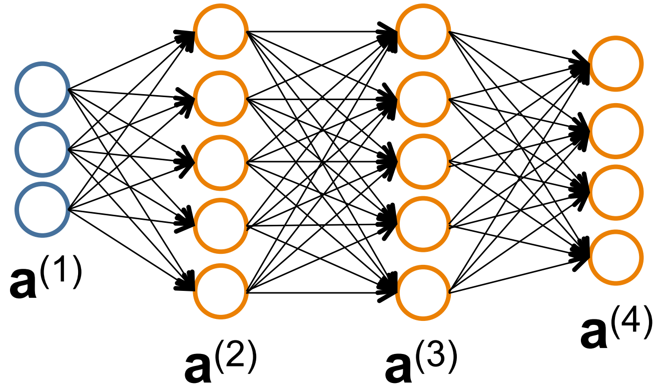
$$\frac{\partial}{\partial\Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)}\,\delta_i^{(l+1)}$$ (ignoring λ; if λ = 0)

Forward Propagation

Backpropagation

Forward Propagation

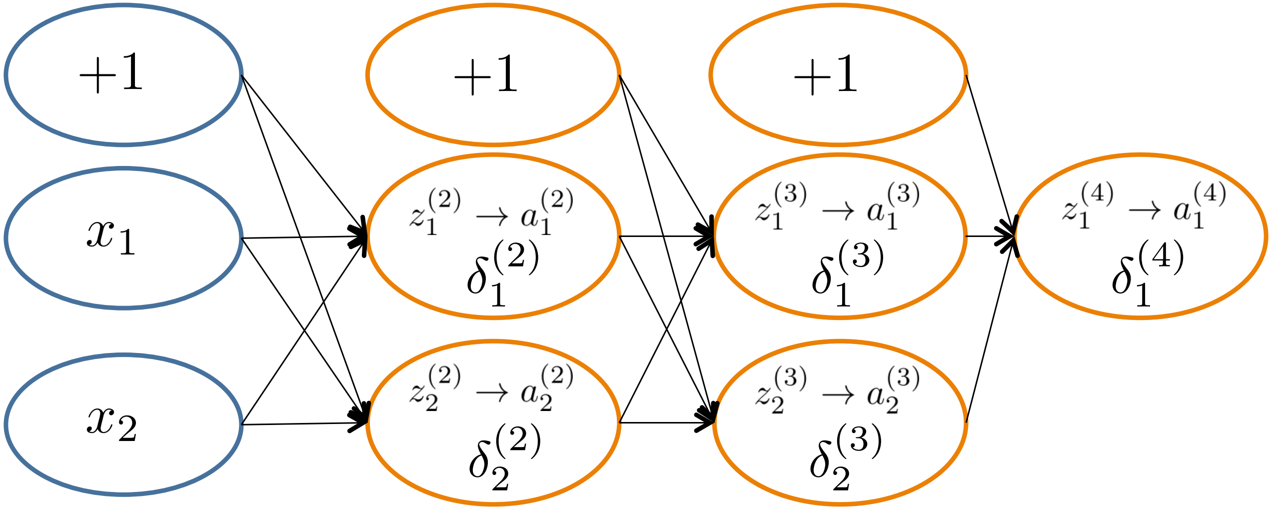
- Given one labeled training instance $(\mathbf{x}, y)$:



<u>Forward Propagation</u>

- $\mathbf{a}^{(1)} = \mathbf{x}$
- $\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{a}^{(1)}$
- $\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$ [add $a_0^{(2)}$]
- $\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$
- $\mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$ [add $a_0^{(3)}$]
- $\mathbf{z}^{(4)} = \Theta^{(3)}\mathbf{a}^{(3)}$
- $\mathbf{a}^{(4)} = h_\Theta(\mathbf{x}) = g(\mathbf{z}^{(4)})$

Backpropagation Intuition

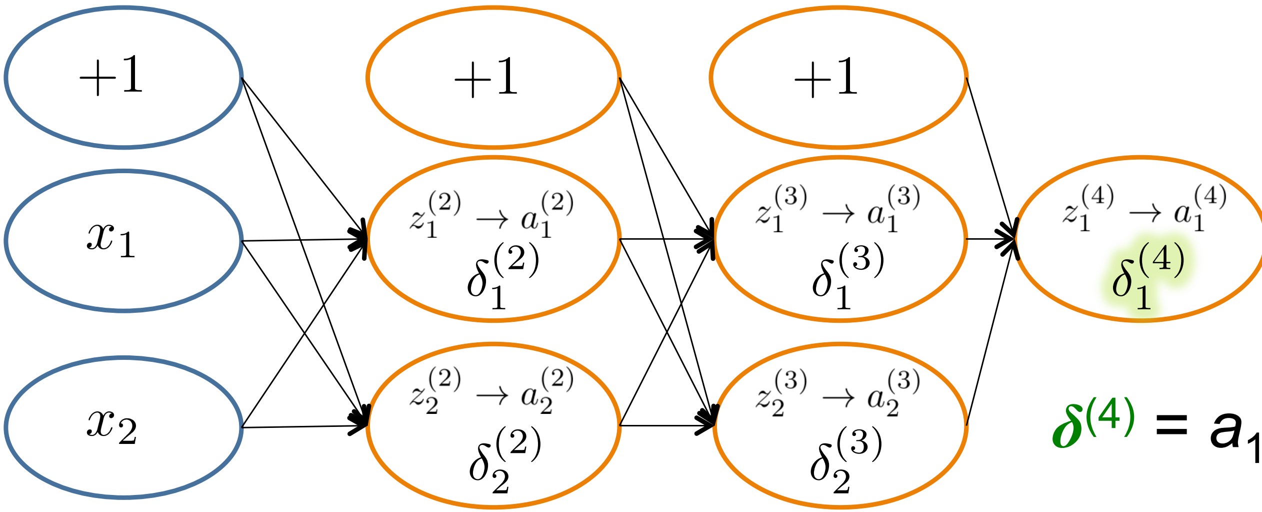


$\delta_j^{(l)}$ = "error" of node *j* in layer *l*

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

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

Backpropagation Intuition



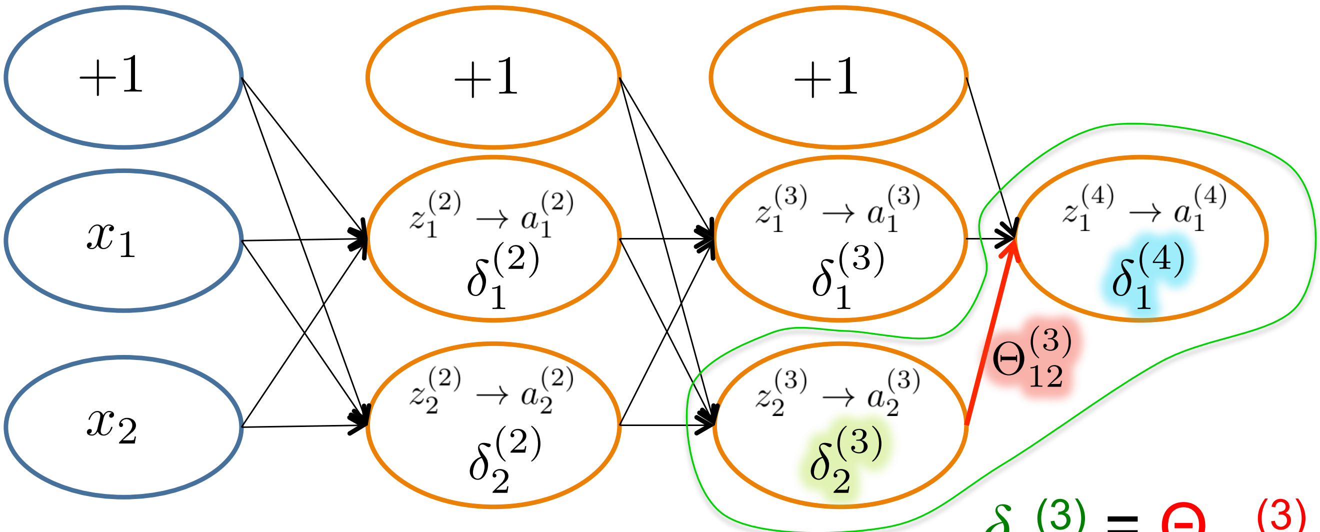
$\boldsymbol{\delta}^{(4)} = a_1^{(4)} - y$

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

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

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

Backpropagation Intuition



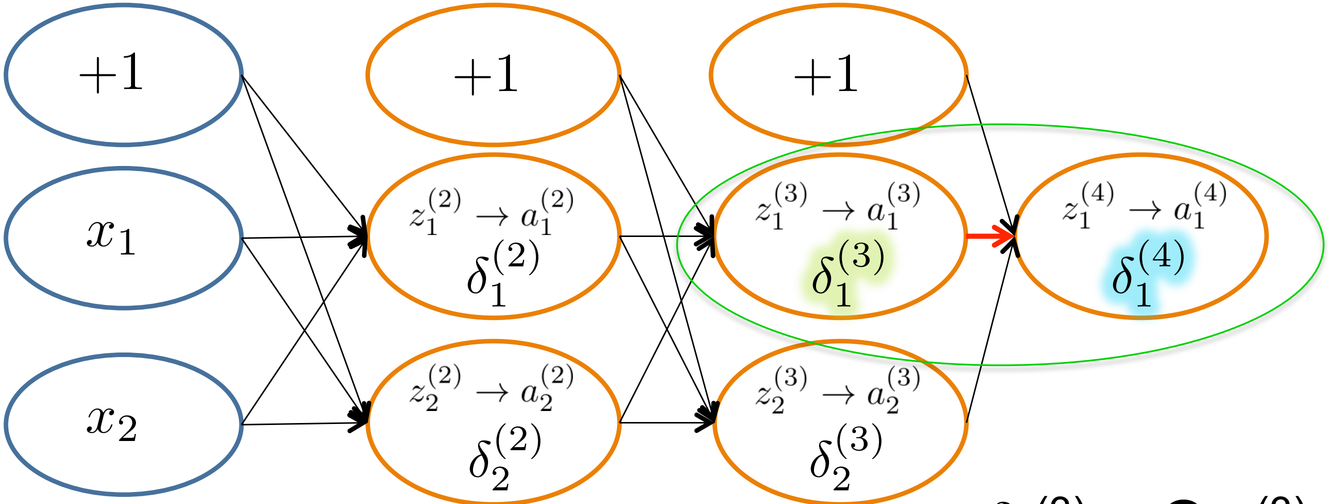
$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$

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

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

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

Backpropagation Intuition



$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$

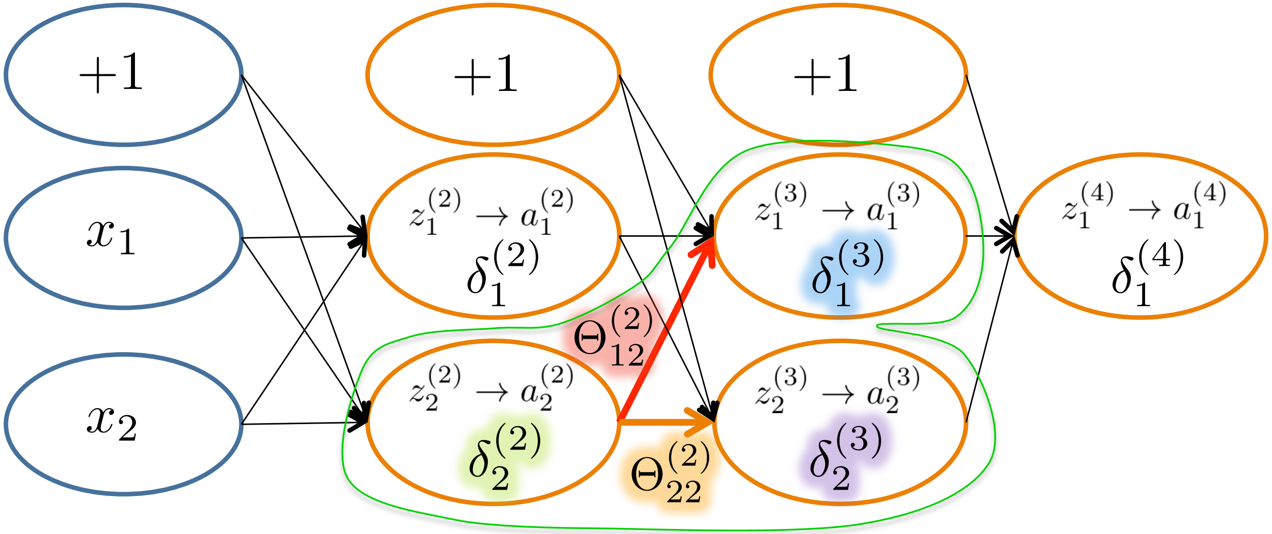
$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$

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

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

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

Backpropagation Intuition



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)}$$

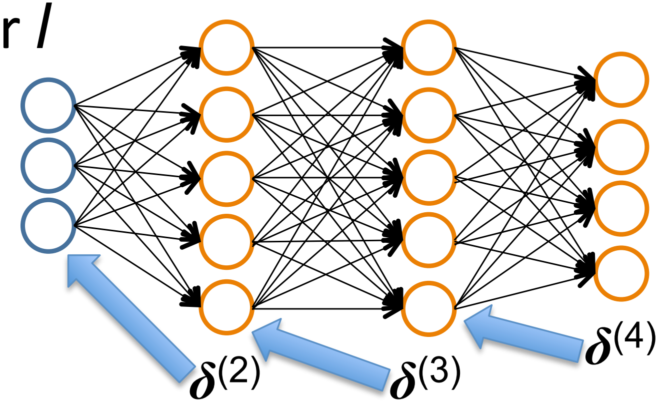
$\delta_j^{(l)}$ = "error" of node *j* in layer *l*

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

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

Backpropagation: Gradient Computation

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



(#layers $L = 4$)

Element-wise product .*

<u>Backpropagation</u>

- $\boldsymbol{\delta}^{(4)} = \mathbf{a}^{(4)} - \mathbf{y}$
- $\boldsymbol{\delta}^{(3)} = (\Theta^{(3)})^{\mathsf{T}}\boldsymbol{\delta}^{(4)} \;.\!*\; g'(\mathbf{z}^{(3)})$
- $\boldsymbol{\delta}^{(2)} = (\Theta^{(2)})^{\mathsf{T}}\boldsymbol{\delta}^{(3)} \;.\!*\; g'(\mathbf{z}^{(2)})$
- (No $\boldsymbol{\delta}^{(1)}$)

$g'(\mathbf{z}^{(3)}) = \mathbf{a}^{(3)} \;.\!*\; (1-\mathbf{a}^{(3)})$

$g'(\mathbf{z}^{(2)}) = \mathbf{a}^{(2)} \;.\!*\; (1-\mathbf{a}^{(2)})$

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 $J(\Theta)$

Training a Neural Network via Gradient Descent with Backprop

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch

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

Backpropagation

Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$
Until weights converge or max #epochs is reached

Based on slide by Andrew Ng

Several Practical Tricks

Initialization

- Problem is highly non-convex, and heuristics exist to start training (at the least, randomize initial weights)

Optimization tricks

- Momentum-based methods
- Decaying step size
- Dropout to avoid co-adaptation / overfitting

Minibatch

- Use more than a single point to estimate gradient

…

Neural Networks vs Deep Learning?

DNN are big neural networks

- Depth: often ~5 layers (but some have 20+)
  - Typically not fully connected!
- Width: hidden nodes per layer in the thousands
- Parameters: millions to billions

Algorithms / Computing

- New algorithms (pre-training, layer-wise training, dropout, etc.)
- Heavy computing requirements (GPUs are essential)