



# COMP [56]630– Machine Learning

Lecture 1 – Course Structure and ML Basics



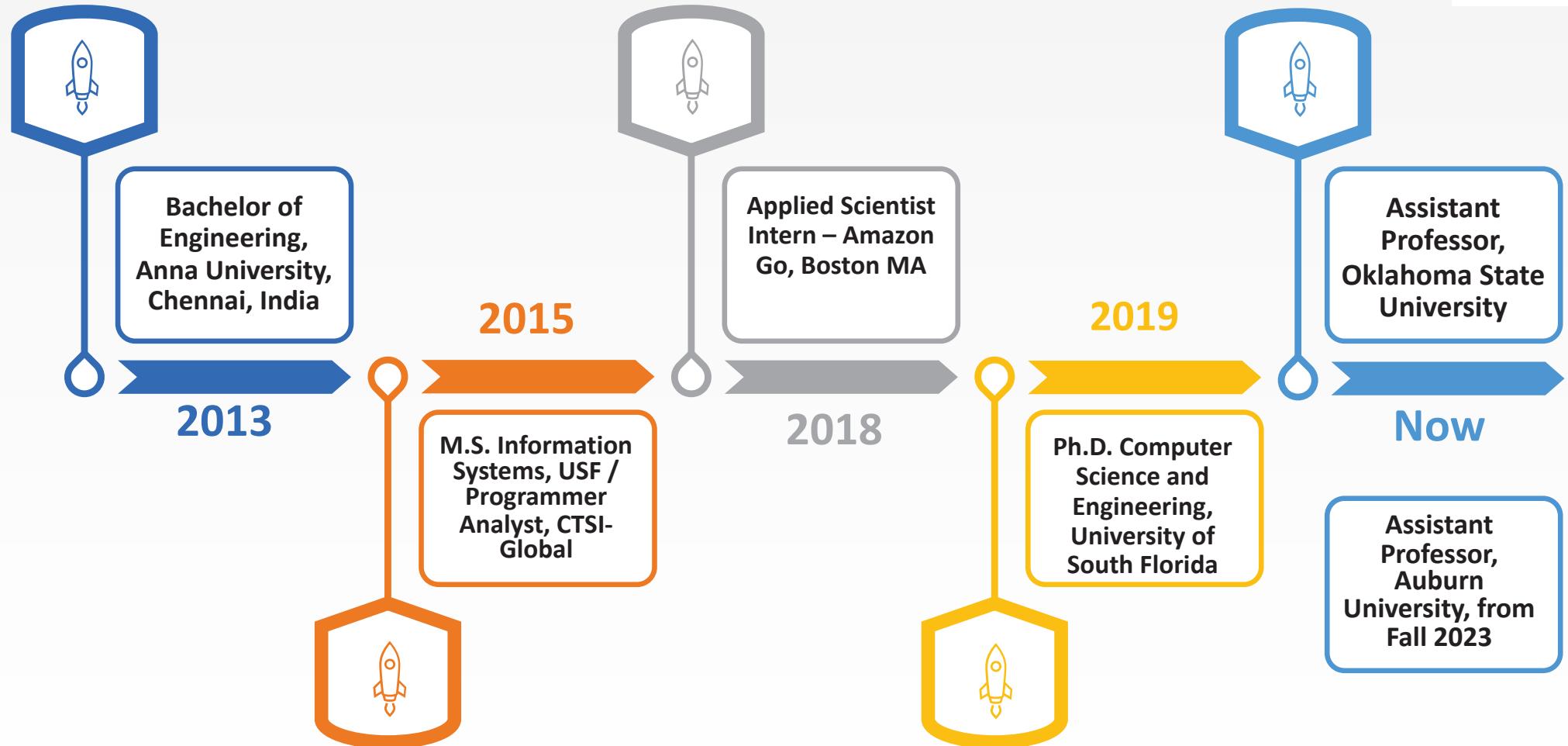
# The Teaching Team

- Instructor:
  - Dr. Sathyanarayanan Aakur
  - Email: san0028@auburn.edu
  - Office hours: Monday/Wednesday 9:00 AM to 10:30 AM, 3101P Shelby Center, or by appointment.
- Teaching Assistant(s):
  - Shubham Trehan
  - Email: szt0113@auburn.edu
  - Office hours: TBD



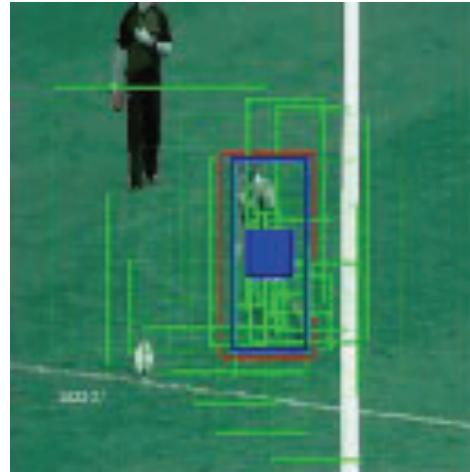
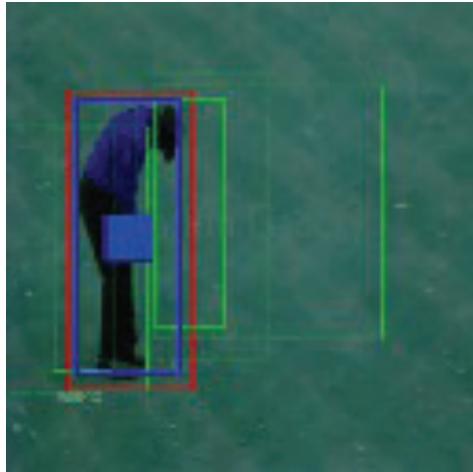


# About Me





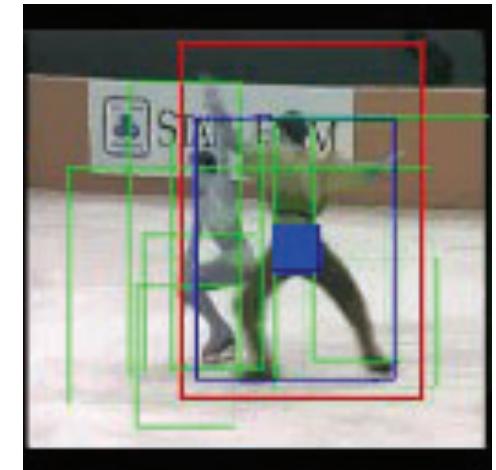
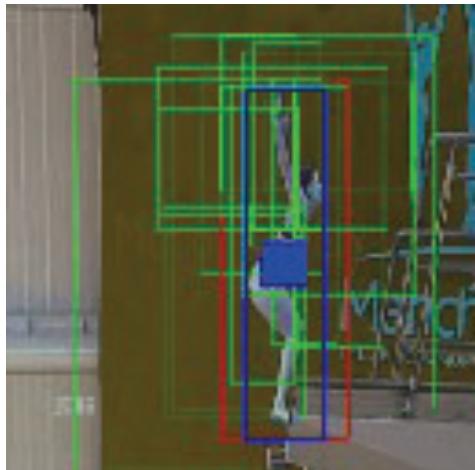
# My Research – Computer Vision (Action Localization)



*Blue BB: Prediction, Red BB: GT, Green BB: Other possible ROIs, Shaded Blue: Attention Location*

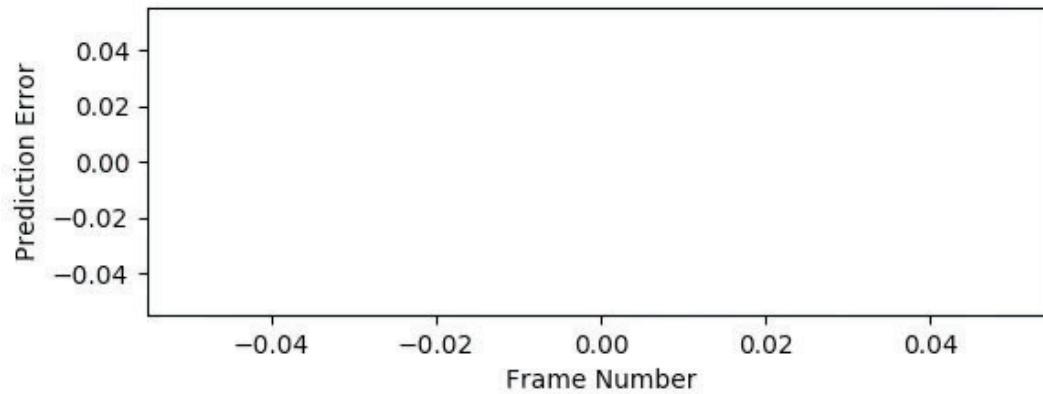


# My Research – Computer Vision (Action Localization)



*Blue BB: Prediction, Red BB: GT, Green BB: Other possible ROIs, Shaded Blue: Attention Location*

# My Research – Computer Vision (Event Segmentation)

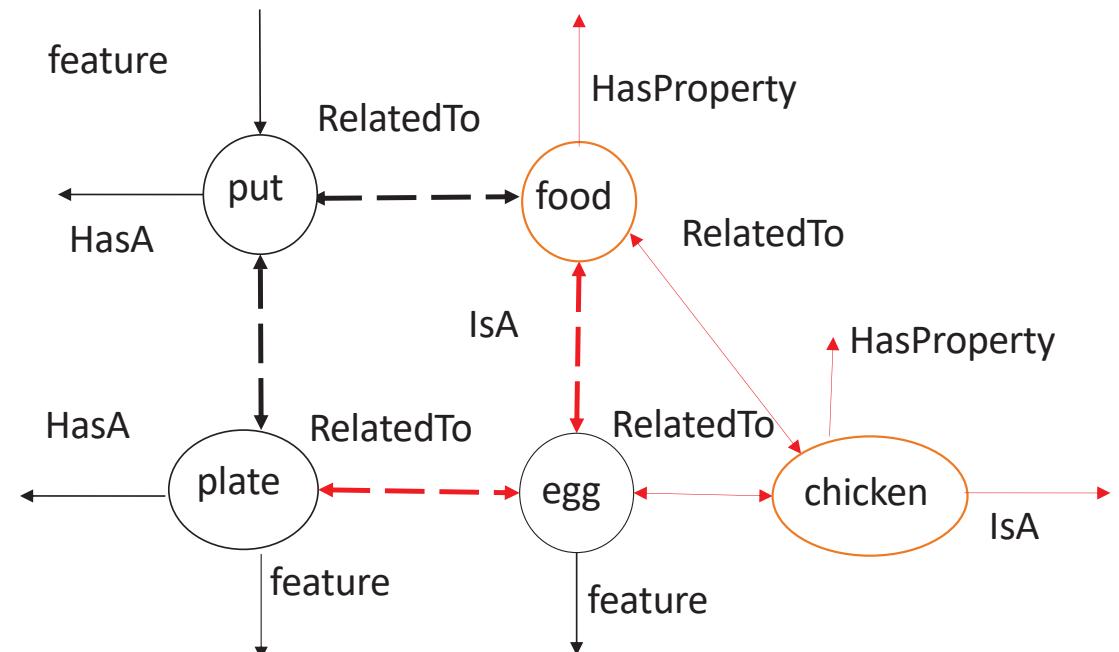
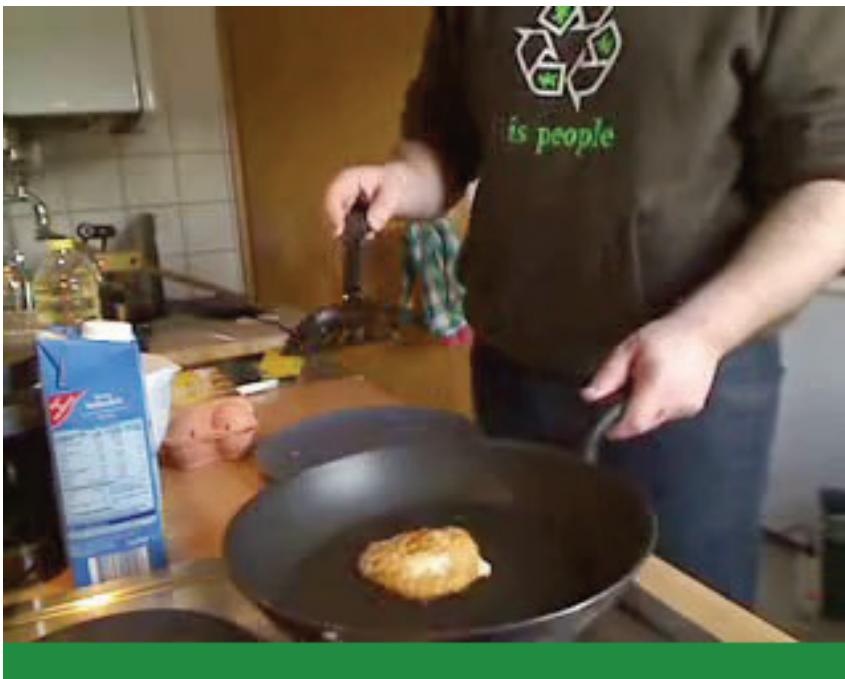


S. Aakur, S. Sarkar. A Perceptual Prediction Framework for Self- Supervised Event Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.*

# My Research – Computer Vision (Active Vision)

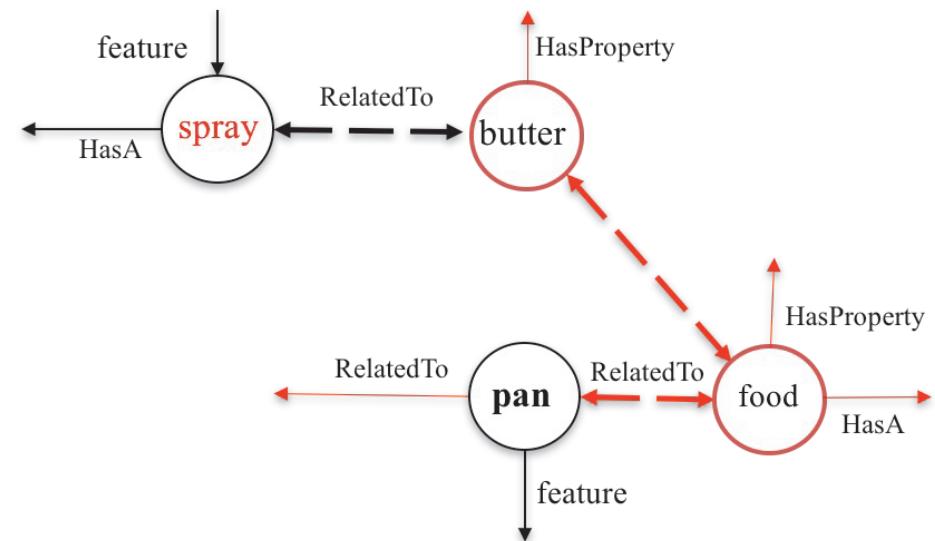


# My Research – Computer Vision (Learning from Experience)



Aakur, S., de Souza, F., & Sarkar, S. (2019). Generating open world descriptions of video using common sense knowledge in a pattern theory framework. *Quarterly of Applied Mathematics*, 77(2), 323-356.

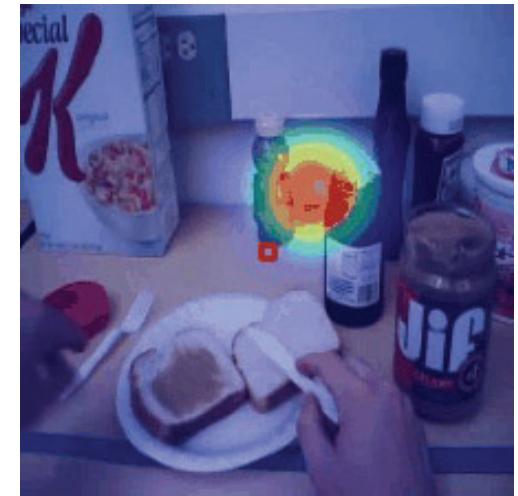
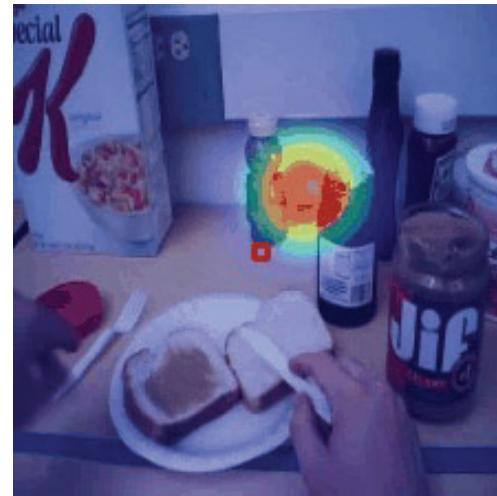
# My Research – Computer Vision (Learning from Experience)



Aakur, S., de Souza, F., & Sarkar, S. (2019). Generating open world descriptions of video using common sense knowledge in a pattern theory framework. *Quarterly of Applied Mathematics*, 77(2), 323-356.



# My Research – Computer Vision (Gaze Prediction)



Unsupervised Gaze Prediction in Egocentric Videos by Energy-based Surprise Modeling. Sathyanarayanan N. Aakur, Arunkumar Bagavathi.  
International Conference on Computer Vision Theory and Applications (VISAPP), 2021



# Course Administration

- Textbook:
  - No particular book is required for this course. However, I would recommend the following books for reference.
    - Textbook 1: Christopher M Bishop, Pattern Recognition and Machine Learning
    - Textbook 2: Introduction to Machine Learning, Third Edition by Ethem Alpaydin.
- Supplementary material will be posted on Canvas as needed.



# Grading Information (5600)

## Exams (20%)

- One Midterm Exam

## Quizzes (25%)

- One quiz at the end of every week on Canvas.
- Open book and open notes

## Assignments (35%)

- Five (5) assignments – One every two weeks
- Typically contains two (2) to three (3) problems and one (1) bonus problem

## Final Exam (20%)

- A Comprehensive final exam at the end of the semester
- Talk to us about any concerns about projects/exams



# Grading Information (6600)

## Midterm (20%)

- One Midterm Exam

## Quizzes (20%)

- One quiz at the end of every week on Canvas.
- Open book and open notes

## Assignments (25%)

- Four (4) assignments – One every two weeks
- Typically contains two (2) to three (3) problems and one (1) bonus problem

## Final Project (15%)

- Team effort (2-3 students per team)
- Proposal due at the end of 6<sup>th</sup> week
- Project report and presentation due at the end of the course
  - Only for graduate section (6600)

## Final Exam (20%)

- A Comprehensive final exam at the end of the semester
- Talk to us about any concerns about projects/exams



# More details

## Assignments

- Do them to truly understand the material, not to get the grade
- All homework write-ups **must** be your own work, written up individually and independently
  - Peer discussions are encouraged. Please don't share solutions!
- No late submissions will be accepted

## Quizzes and Exams

- Will cover material from each week lectures, reading assignments and supplementary material.
- Only one attempt per quiz



# Grading Policies

Percentage	Grade	GPA Quality Points
<b>90 - 100</b>	A	4.0
<b>80 - 89</b>	B	3.0
<b>70 - 79</b>	C	2.0
<b>60 - 69</b>	D	1.0



# Tentative Schedule

<b>Week 1</b>	Syllabus, Course policies, What is ML?, ML Basics
<b>Week 2</b>	Linear Regression
<b>Week 3</b>	Logistic Regression, Model Selection, Evaluation Metrics
<b>Week 4</b>	Neural Networks
<b>Week 5</b>	Deep Learning - Convolutional Neural Networks
<b>Week 6</b>	Deep Learning - Sequence Learning
<b>Week 7</b>	Deep Learning - Recent Advances, Bayesian Learning
<b>Week 8</b>	Midterm 1, Probabilistic Graphical Models
<b>Week 9</b>	Spring Break

<b>Week 10</b>	Support Vector Machines
<b>Week 11</b>	Decision Trees
<b>Week 12</b>	Unsupervised Learning
<b>Week 13</b>	Expectation Maximization and Gaussian Mixture Models
<b>Week 14</b>	Hidden Markov Models and Dimensionality Reduction
<b>Week 15</b>	Reinforcement Learning and ML Applications
<b>Week 16</b>	Bias, Fairness and Ethics in AI
<b>Week 17</b>	Final Exam Week



# Software Requirements

- We will use Python as the major language in this course
  - Intermediate level is a pre-requisite
- Different packages/libraries will be used to create ML applications.
  - NumPy
  - Tensorflow
  - Sci-kit Learn
- A short intro will be given in class
  - Some self-study is expected to learn the intricacies of using these libraries.



# Expectations from You

- **Work Hard!**

- This is a heavy course that covers ML fundamentals.
- A little heavy on mathematics to provide the foundations of each algorithm
- Exciting and stimulating topics
- Attend all lectures (I try to keep it engaging and interactive!)
- Ask questions, participate in discussions
- Exams are comprehensive. Study all materials posted.
- Complete all assignments
  - Try to understand the algorithm, usage and implementation



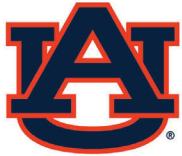
# Academic Dishonesty

- **Absolutely no form of cheating will be tolerated**
- See syllabus, AU Policy, and CSSE Academic Integrity Policy
- Cheating → Failing grade (no exceptions)



# Tips to succeed in the course

- Read any supplementary material posted in addition to the lectures
- Do the assignments and quizzes on your own to understand the concepts
- Don't be discouraged by errors!
  - Learning is a repetitive process!
  - Mistakes are your keys to success! ☺
- Ask thorough questions to understand the concepts
- Balance your time!
  - I understand that you have several things to do. Allot at least 2-3 hours per week.
  - Remember. Hard work always pays off!



Have you ever used machine learning?  
How? Where?



**Laptop:** Biometrics auto-login (face recognition, 3D), OCR

**Smartphones:** QR codes, computational photography (Android Lens Blur, iPhone Portrait Mode), panorama construction (Google Photo Spheres), face detection, expression detection (smile), Snapchat filters (face tracking), FaceID (iPhone), Night Sight (Pixel), iPhone 12 Pro (LiDAR)

**Web:** Image search, Google photos (face recognition, object recognition, scene recognition, geolocalization from vision), Facebook (image captioning), Google maps aerial imaging (image stitching), YouTube (content categorization)

**VR/AR:** Outside-in tracking (HTC VIVE), inside out tracking (simultaneous localization and mapping, HoloLens), object occlusion (dense depth estimation)

**Motion:** Kinect, full body tracking of skeleton, gesture recognition, virtual try-on

**Medical imaging:** CAT / MRI reconstruction, assisted diagnosis, automatic pathology, connectomics, endoscopic surgery

**Industry:** Vision-based robotics (marker-based), machine-assisted router (jig), automated post, ANPR (number plates), surveillance, drones, shopping

**Transportation:** Assisted driving (everything), face tracking/iris dilation for drunkenness, drowsiness, automated distribution (all modes)

**Media:** Visual effects for film, TV (reconstruction), virtual sports replay (reconstruction), semantics-based auto edits (reconstruction, recognition)



# What is Machine Learning anyway?

No, Really. What is it?



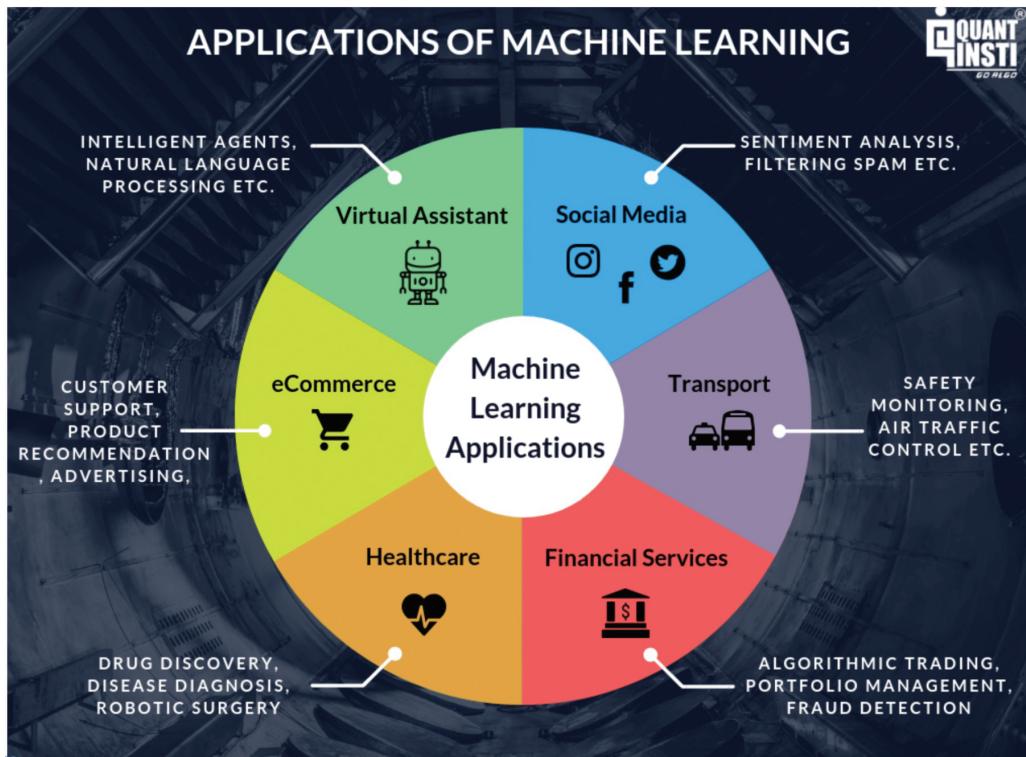
What you think it is

© Sathyanarayanan Aakur

24

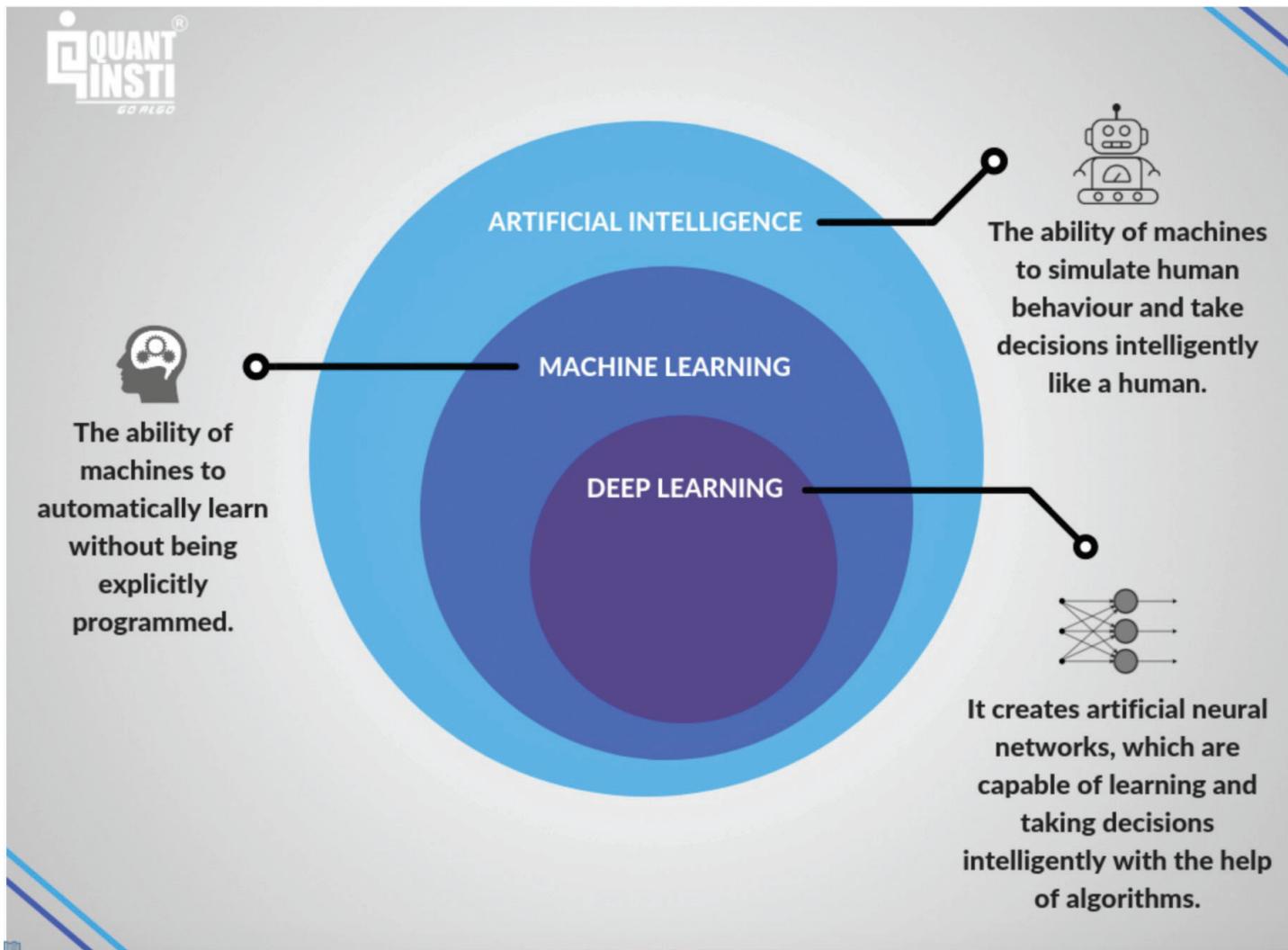


# What it actually is





Aritificial Intelligence (AI)  
≠ Machine Learning (ML)  
≠ Deep Learning (DL)





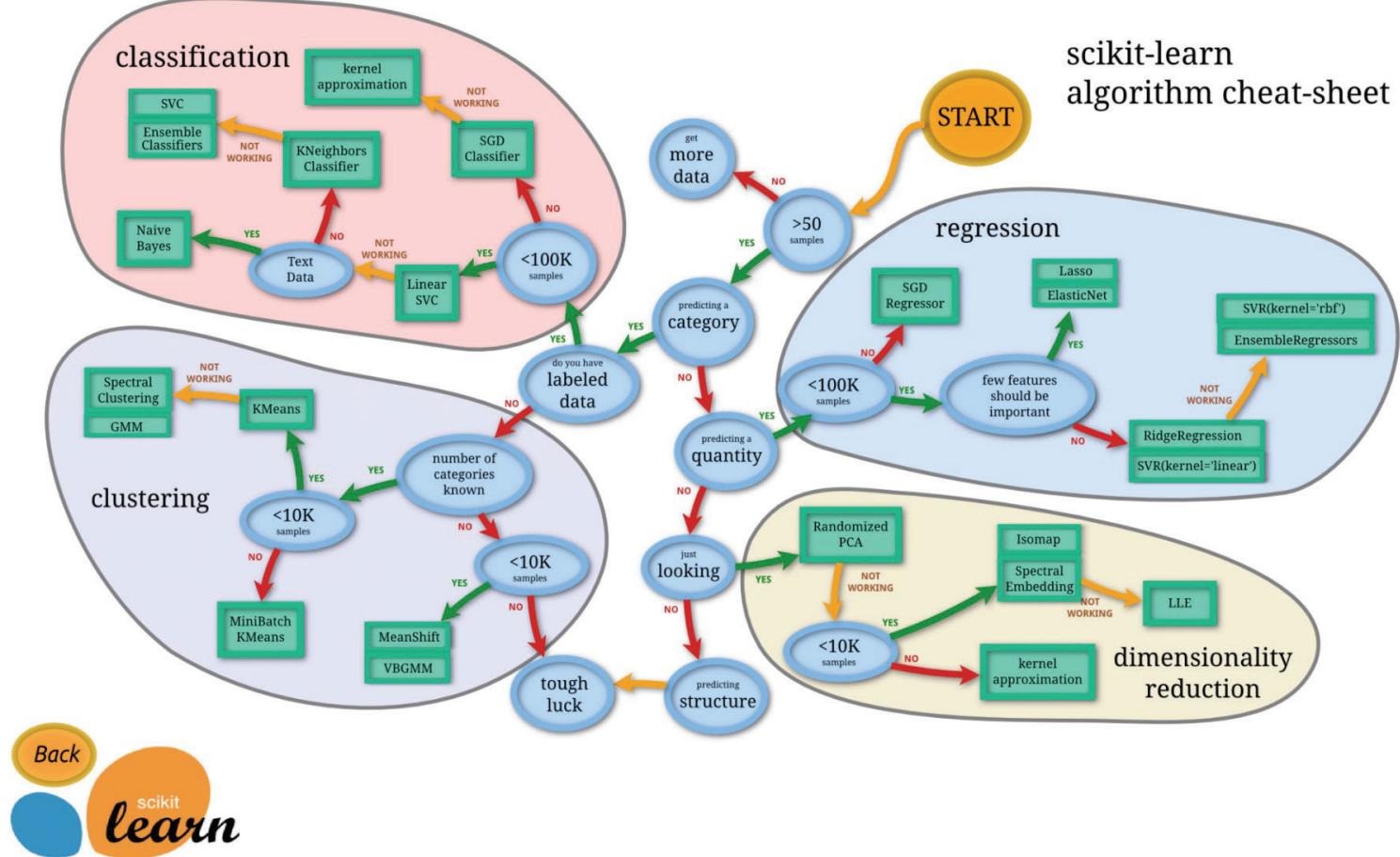
# What is Machine Learning?

- **Definition:** study of computer algorithms that improve automatically through experience i.e. developing computational methods that use experience to improve performance, e.g., making accurate predictions.
- **What is Experience?**
  - Data! ML is a data-driven task,
- **Interdisciplinary!**
  - **Computer science:** need to design efficient and accurate algorithms, analysis of complexity, theoretical guarantees.
  - **Mathematics:** ML is based on the ideas from linear algebra, statistics and probability
- **Example application:** use words from an email, sender information, etc. to determine if it is a spam email



# ML Objectives

- **Algorithms:**
  - Efficient, accurate, and general
  - Scalability
- **Theoretical questions:**
  - What can be learned?
  - How to model learning computationally?





# COMP [56]630– Machine Learning

Lecture 2 –ML Basics



# Machine Learning Vocabulary

- **Example:** an object or instance in data used.
- **Features:** the set of attributes, often represented as a vector, associated to an example
- **Labels:**
  - in *classification*, category associated to an object
  - in *regression*, real-valued numbers.



# Machine Learning Vocab (contd.)

- **Training data:** data used for training the ML algorithm.
- **Test data:** data exclusively used for testing the ML algorithm.
- Some standard learning scenarios:
  - **supervised learning:** labeled training data.
  - **unsupervised learning:** no *labeled* data.
  - **semi-supervised learning:** both labeled and unlabeled training data



# Supervised Learning

- Inputs:
  - Series of data examples:  $x_1, x_2, x_3, \dots, x_n$  and corresponding labels  $y_1, y_2, y_3, \dots, y_n$
- Goal:
  - Learn to associate patterns from the examples ( $x_1, x_2, x_3, \dots, x_n$ ) with their labels ( $y_1, y_2, y_3, \dots, y_n$ )
  - Learn to produce the **desired output** ( $y$ ) given a **new input** that may or may not be from the training examples
- Labels can be categorical (classification) or continuous (regression)
- Example: SPAM vs NON-SPAM classification, Predicting house prices from details like # of bedrooms, # of bathrooms, size, location, etc.



# Unsupervised Learning

- Inputs:
  - Series of data examples:  $x_1, x_2, x_3, \dots, x_n$
- Goal:
  - Build a model of  $x$  that can be used for reasoning, decision making, predicting things, communicating
  - Note that we do not use any labels ( $y_i$ ) that may or may not be present
- Example: grouping data examples  $x_i$  based on features (clustering), learn good, common representations of  $x_i$  for other purposes (representation learning)

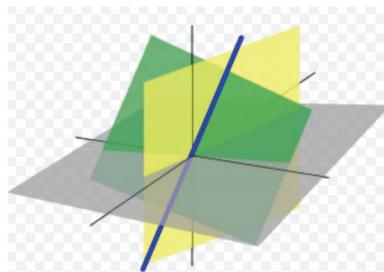


# ML Basics – Linear Algebra



# What is linear algebra?

- Branch of mathematics that deal with linear equations such as
$$a_1x_1 + \dots + a_nx_n = b$$
- Vector notation:  $\mathbf{a}^T \mathbf{x} = b$   
→ Called a *linear transformation of the variable x*
- Linear algebra is fundamental to geometry, for defining objects such as lines, planes, rotations



Linear equation  $a_1x_1 + \dots + a_nx_n = b$   
defines a plane in  $(x_1, \dots, x_n)$  space  
Straight lines define common solutions  
to equations



# Why linear algebra?

- It is based on continuous values.
  - Used throughout engineering for various applications
- Essential for understanding ML algorithms
  - E.g., We convert input vectors ( $x_1, \dots, x_n$ ) into outputs by a series of linear transformations
- In this lecture, we cover enough of the basics to understand ML algorithms.



# Linear Algebra Topics

- Scalars, Vectors, Matrices and Tensors
- Multiplying Matrices and Vectors
- Identity and Inverse Matrices
- Linear Dependence and Span
- Norms
- Special kinds of matrices and vectors
- Eigen decomposition
- Singular value decomposition
- The Moore Penrose pseudoinverse
- Trace and determinants of a matrix



# Scalars, Vectors and Matrix

- Scalar
  - Single number
  - Represented in lower-case italic  $x$
  - They can be real-valued or be integers
    - i.e. slope of a line (real-valued)
- Vector
  - An array of numbers arranged in order
  - Each number can be identified by an index
  - Written in lower-case bold such as  $\mathbf{x}$
  - We can think of vectors as points in space
    - Each element gives coordinate along an axis



# Scalars, Vectors and Matrix

- A vector's elements are in italics lower case, subscripted

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- $\rightarrow n$  element vector

- Matrix: 2-D array of numbers

- So each element identified by two indices
- Denoted by bold typeface **A**
- If A has shape of height m and width n with real-values then  $\mathbf{A} \in \mathbb{R}^{m \times n}$

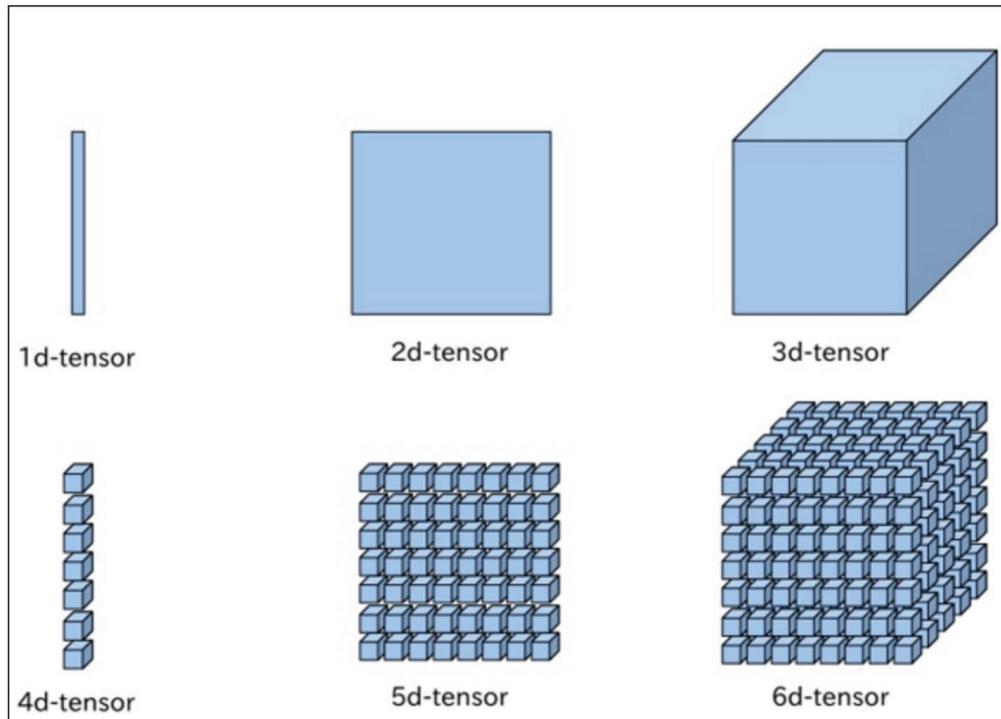


# Matrix, Tensors

- Matrix:
  - Elements indicated by name in italic but not bold
  - $A_{ij}$  represents the element in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column
- Example:  $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$
- Tensors:
  - Arrays with **more than 2 dimensions**
    - Why?
  - A tensor is an array of numbers arranged on a regular grid with variable number of axes
  - Again, denoted by bold typeface **A**. Elements given by  $A_{ijk}$  for 3-d tensor.



# Shapes of tensors





# Matrix/Tensor operations



# Transpose

- Denoted as  $\mathbf{A}^T$
- Defined as

$$(\mathbf{A}^T)_{i,j} = A_{j,i}$$

- Mirror image across the (main) diagonal of a matrix
  - Main diagonal -> running down from upper left to the bottom right

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \\ A_{1,3} & A_{2,3} & A_{3,3} \end{bmatrix}$$

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$



# Vectors/Scalars as matrices

- Vectors → Matrices with one column
- Written as

$$x = [x_1 \quad \dots \quad x_n]^T$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \Rightarrow x^T = [x_1 \quad \dots \quad x_n]^T$$

- Scalar → Matrix with one element

$$a = a^T$$



# Matrix Addition

- We can add matrices to each other if they have the same shape, by adding corresponding elements
  - If A and B have same shape (height m, width n)
$$\mathbf{C} = \mathbf{A} + \mathbf{B}$$
$$C_{i,j} = A_{i,j} + B_{i,j}$$
- You can add or multiply a matrix by a scalar
  - $\mathbf{D} = a\mathbf{B} + c$ 
$$D_{i,j} = aB_{i,j} + c$$
- Addition vector to matrix → i.e. **broadcasting** since vector added to each row of  $\mathbf{A}$

$$\mathbf{C} = \mathbf{A} + b$$

$$C_{i,j} = A_{i,j} + b_j$$



# Matrix Multiplication

- For product  $\mathbf{C} = \mathbf{AB}$  to be defined,  $\mathbf{A}$  has to have the same no. of columns as the no. of rows of  $\mathbf{B}$ 
  - If  $\mathbf{A}$  is of shape  $m \times n$  and  $\mathbf{B}$  is of shape  $n \times p$  then matrix product  $\mathbf{C}$  is of shape  $m \times p$
- **Product of two matrices is not the product of their individual elements!**
  - It is called element-wise product or the **Hadamard product**  $\mathbf{A} \odot \mathbf{B}$
  - We can think of matrix product  $\mathbf{C} = \mathbf{AB}$  as computing  $C_{i,j}$  the dot product of row  $i$  of  $\mathbf{A}$  and column  $j$  of  $\mathbf{B}$



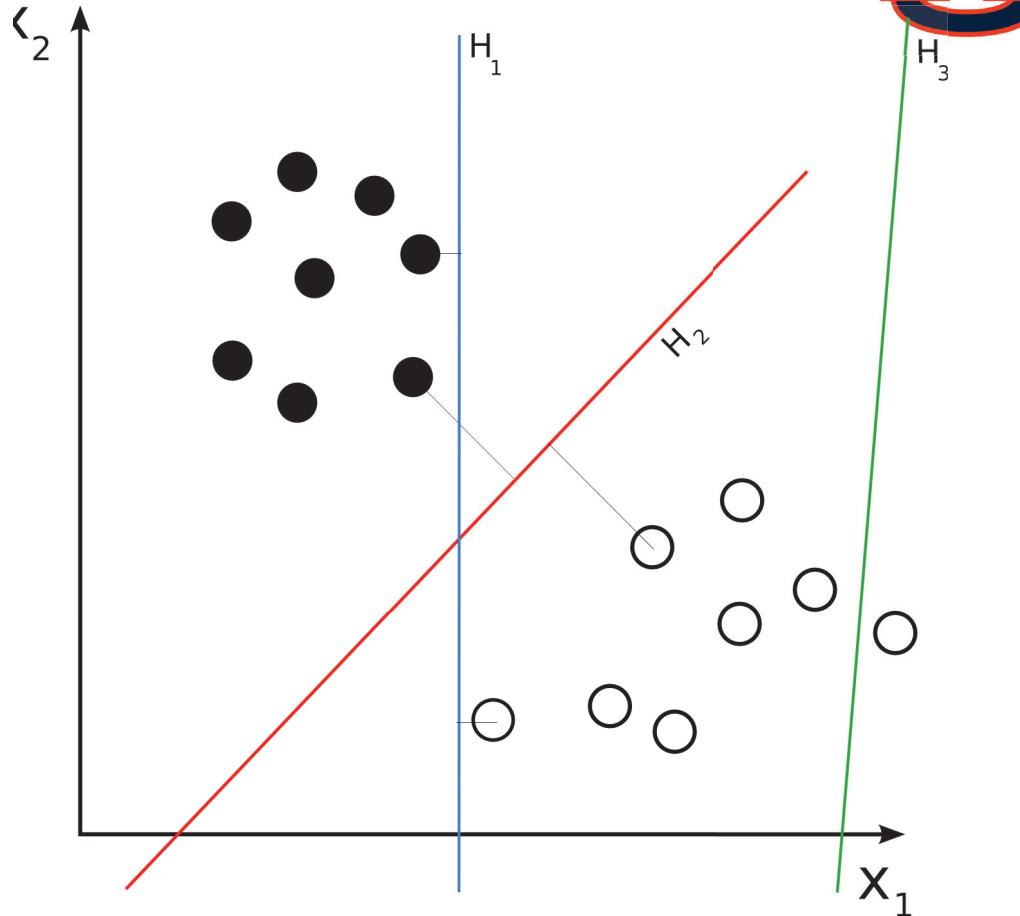
# Matrix Product Properties

- *Distributivity over addition:  $A(B+C)=AB+AC$*
- *Associativity:  $A(BC)=(AB)C$*
- *NOT commutative:  $AB=BA$  is not always true*
- *Dot product between vectors is commutative:  $x^T y = y^T x$*
- *Transpose of a matrix product has a simple form:  $(AB)^T=B^T A^T$*



# Linear Classifier

- The simplest ML model
- Makes a *classification* decision based on the value of a linear combination of the characteristics (features).
- Black and white circles are different labels.  $H_1, H_2, \dots$  represent different *decision boundaries* i.e. linear functions that best map the classification process.
  - **Goal:** find the best linear function that has highest accuracy





## Linear Classifier (cntd.)

- Mathematically represented as

$$y = \mathbf{W}\mathbf{x}^T + b$$

where  $y \rightarrow$  labels (vector)

$\mathbf{W} \rightarrow$  model parameter matrix

$\mathbf{x} \rightarrow$  feature vector

$b \rightarrow$  bias term (scalar)

- Very similar in the mathematical representation of a line

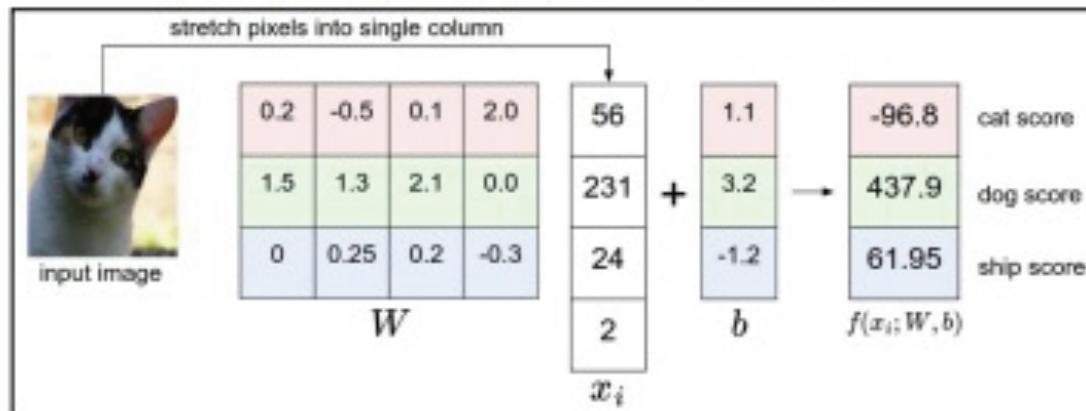
$$y = mx + c$$

→ Hence the term ***linear classifier***



## Linear Classifier (cntd.)

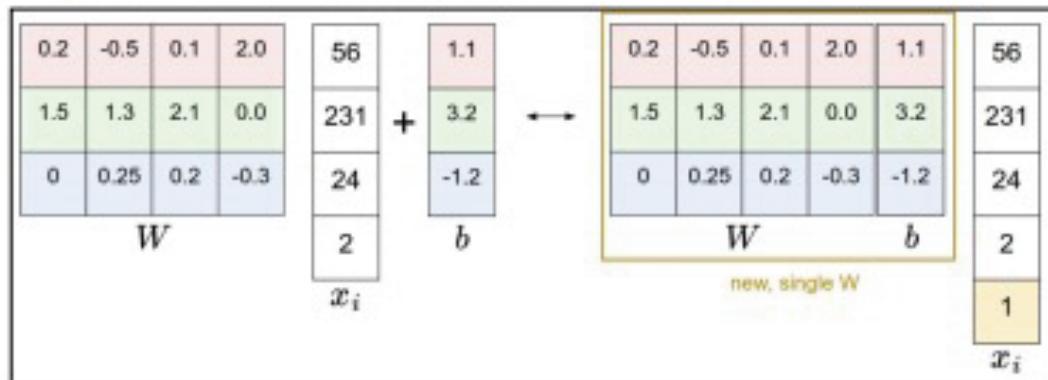
$$\text{A linear classifier } y = \mathbf{W}\mathbf{x}^T + b$$





# Linear Classifier (cntd.)

A linear classifier with bias eliminated  $y = \mathbf{W}\mathbf{x}^T$





# Linear Transformation

$$Ax = b$$

- where  $A \in \mathbb{R}^{n \times n}$  and  $b \in \mathbb{R}^n$
- More explicitly

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n &= b_2 \\ A_{n1}x_1 + A_{n2}x_2 + \dots + A_{nn}x_n &= b_n \end{aligned}$$

$n$  equations in  
 $n$  unknowns



# Linear Transformation

$$Ax = b$$

- where  $A \in \mathbb{R}^{n \times n}$  and  $b \in \mathbb{R}^n$
- More explicitly

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n &= b_2 \\ \vdots & \\ A_{n1}x_1 + A_{n2}x_2 + \dots + A_{nn}x_n &= b_n \end{aligned}$$

*System of equations*

*n equations in  
n unknowns*



# Linear Transformation

$$A\mathbf{x} = \mathbf{b}$$

- where  $A \in \mathbb{R}^{n \times n}$  and  $\mathbf{b} \in \mathbb{R}^n$
- More explicitly

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n &= b_2 \\ A_{n1}x_1 + A_{n2}x_2 + \dots + A_{nn}x_n &= b_n \end{aligned}$$

*System of equations*

*n equations in  
n unknowns*

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{n,1} & \cdots & A_{nn} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$n \times n$        $n \times 1$        $n \times 1$

Can view  $A$  as a *linear transformation* of vector  $\mathbf{x}$  to vector  $\mathbf{b}$



# Linear Transformation

$$A\mathbf{x} = \mathbf{b}$$

- where  $A \in \mathbb{R}^{n \times n}$  and  $\mathbf{b} \in \mathbb{R}^n$
- More explicitly

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \dots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + \dots + A_{2n}x_n &= b_2 \\ A_{n1}x_1 + A_{n2}x_2 + \dots + A_{nn}x_n &= b_n \end{aligned}$$

*System of equations*

*n equations in  
n unknowns*

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{n,1} & \cdots & A_{nn} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$n \times n$        $n \times 1$        $n \times 1$

Can view  $A$  as a *linear transformation* of vector  $\mathbf{x}$  to vector  $\mathbf{b}$

**How to solve this?**



# Linear Transformation (cntd.)

- Matrix Inverse to the rescue!
  - Inverse of a matrix is defined as  $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$ 
    - $\mathbf{I}_n$  is an identity matrix of dimension  $n \times n$
    - $\mathbf{A}$  is a square matrix
- Solving  $\mathbf{Ax}=\mathbf{b}$ :

$$\begin{aligned}\mathbf{Ax} &= \mathbf{b} \\ \mathbf{A}^{-1}\mathbf{Ax} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{I}_n\mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{b}\end{aligned}$$



# Linear Transformation (cntd.)

- Matrix Inverse to the rescue!
  - Inverse of a matrix is defined as  $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$ 
    - $\mathbf{I}_n$  is an identity matrix of dimension  $n \times n$
    - $\mathbf{A}$  is a square matrix
- Solving  $\mathbf{Ax}=\mathbf{b}$ :

$$\begin{aligned}\mathbf{Ax} &= \mathbf{b} \\ \mathbf{A}^{-1}\mathbf{Ax} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{I}_n\mathbf{x} &= \mathbf{A}^{-1}\mathbf{b} \\ \mathbf{x} &= \mathbf{A}^{-1}\mathbf{b}\end{aligned}$$

*Will this work for all cases?*



## Linear Transformation (cntd.)

- This depends on being able to find  $A^{-1}$
- If  $A^{-1}$  exists there are several methods for finding it
- **$A^{-1}$  does not exist for ALL matrices and is possible to find only for square matrices and non-singular matrices**
- **Alternative:** Use *Gaussian elimination* and back-substitution.
  - Transform the matrix  $A$  into an upper triangular matrix using a series of row-wise operations such as
    - Swapping two rows,
    - Multiplying a row by a nonzero number,
    - Adding a multiple of one row to another row.



# Linear Transformation (cntd.)

- Gaussian elimination example:
- Given a system of equations:

$$\begin{aligned} 2x + y - z &= 8 \\ -3x - y + 2z &= -11 \\ -2x + y + 2z &= -3 \end{aligned}$$

- Construct a matrix:

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array} \right]$$

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 2 & 1 & 5 \end{array} \right]$$

- Perform operations until you transform into upper triangular matrix

$$L_2 + \frac{3}{2}L_1 \rightarrow L_2$$

$$L_3 + L_1 \rightarrow L_3$$

$$L_3 + -4L_2 \rightarrow L_3$$

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array} \right]$$



## Linear Transformation (cntd.)

- Using the final matrix from the previous steps, we can see that the value of  $z = -1$  (last row)
- Using back substitution, we get  $y = 3$  (second row)  
and  $x = 2$  (first row)

$$\left[ \begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array} \right]$$



# Disadvantages of Gaussian Elimination and Matrix Inverse

- Matrix Inverse:
  - Can only be used if  $\mathbf{A}^{-1}$  exists
  - If  $\mathbf{A}^{-1}$  exists, the same  $\mathbf{A}^{-1}$  can be used for any given  $\mathbf{b}$
  - But  $\mathbf{A}^{-1}$  cannot be represented with sufficient precision
    - It is not used in practice
- Gaussian Elimination:
  - numerical instability (i.e. division by small no.)
  - Complexity if  $O(n^3)$  for  $n \times n$  matrix
- Software solutions use value of  $\mathbf{b}$  in finding  $\mathbf{x}$ :
  - difference (derivative) between  $\mathbf{b}$  and prediction is used iteratively
    - *Least squares solvers*



# Solving $Ax = b$

- Remember  $Ax = b$  is a system of equations
- Solution is  $x = A^{-1}b$ 
  - =>  $Ax = b$  can be solved if  $A^{-1}$  exists
  - =>  $A^{-1}$  exists only if exactly one solution exists for each value of  $b$ 
    - Not always true!
- A system of equations can have no or infinite solutions for some values of  $b$ 
  - Note: It is not possible to have more than one but fewer than infinitely many solutions
- **$Ax = b$**  is a linear transformation i.e. it is a linear combination of those factors and hence
  - A column of  $A$ , i.e.,  $A_{:,i}$  specifies travel in direction  $i$
  - How much we need to travel is given by  $x_i$
  - Thus determining whether  $Ax=b$  has a solution is equivalent to determining whether  $b$  is in the span of columns of  $A$ 
    - Span of a set of vectors: set of points obtained by a linear combination of those vectors



# Norms

- Measures size of a vector  $\mathbf{x}$ 
  - i.e. distance from origin to  $\mathbf{x}$
- Helps maps vectors to non-negative scalar values
- Norm of a vector  $\mathbf{x} = [x_1 \dots x_n]^T$  is any function that satisfies the triangle inequality

$$\boxed{\begin{aligned} f(\mathbf{x})=0 &\Rightarrow \mathbf{x}=0 \\ f(\mathbf{x}+\mathbf{y}) &\leq f(\mathbf{x})+f(\mathbf{y}) \quad \text{Triangle Inequality} \\ \forall \alpha \in R \quad f(\alpha \mathbf{x}) &= |\alpha| f(\mathbf{x}) \end{aligned}}$$



## Norms (contd.)

- Different kinds of norms exists and can generally be defined as the  $L^P$  norm and is given by

$$\|x\|_p = (\sum_i |x_i|^p)^{\frac{1}{p}}$$

- If  $p=2$ , then it is called the L-2 norm or Euclidean norm
  - Euclidean distance between origin and  $x$  and is represented as  $\|x\| = \sqrt{x^T x}$
- If  $p=1$ , then it is called the L-1 norm.
  - Used when you need to distinguish zero and non-zero vectors
- If  $p = \infty$ , it is given by  $L^\infty = \|x\|_\infty = \max |x_i|$ 
  - Called the max norm

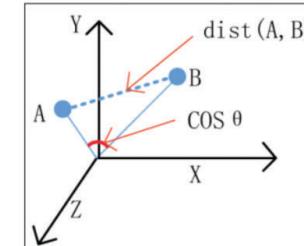


# “Special” Vectors

- Unit vector:
  - A vector with unit norm:  $L^2(x) = 1$
- Orthogonal vectors:
  - Vectors  $x$  and  $y$  are orthogonal if  $x^T y = 0$ 
    - i.e. if the vectors have non-zero norm, they are at 90 degrees to each other
- Orthonormal vector:
  - Vectors are orthogonal and have unit norm
- Dot product of two vectors: 
$$x^T y \Rightarrow \|x\|_2 \|y\|_2 \cos \theta$$

## Distance between two vectors ( $v, w$ )

$$\begin{aligned} - \text{dist}(v, w) &= \|v - w\| \\ &= \sqrt{(v_1 - w_1)^2 + \dots + (v_n - w_n)^2} \end{aligned}$$





# “Special” Matrices

- **Diagonal Matrix:** mostly zeros, with nonzero entries only in diagonal
  - Eg. Identity matrix  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
  - $\text{diag}(v)$  denotes a square diagonal matrix with diagonal elements given by entries of vector  $v$
- **Symmetric matrix:** any matrix  $A$  which satisfies  $A=A^T$
- **Singular matrix:** A square matrix that does not have a matrix inverse.  
i.e. a matrix is singular iff its determinant is 0.
  - Determinant of a matrix: a scalar value that can be computed from the elements of a square matrix and encodes certain properties of the linear transformation described by the matrix

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$



# Matrix Decomposition

- Matrices can be decomposed into factors to learn universal properties
- Many popular algorithms leverage matrix decomposition for solving tasks ranging from data cleaning to label prediction
  - Common applications include
    - Dimensionality reduction
    - Preventing overfitting
    - Finding better features (ignoring clutter, background noise, etc.) by focusing on important aspects of the input
- Many possible ways to matrix decomposition
  - Eigen decomposition
  - QR decomposition
  - Single Value Decomposition



# Eigen Decomposition

- We can decompose a matrix  $\mathbf{A}$  as  $\mathbf{A} = \mathbf{V} \text{diag}(\lambda) \mathbf{V}^{-1}$ 
  - Where  $\mathbf{V} \rightarrow$  eigenvectors,  $\lambda \rightarrow$  eigenvalues

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

has eigenvalues  $\lambda=1$  and  $\lambda=3$  and eigenvectors  $\mathbf{V}$ :

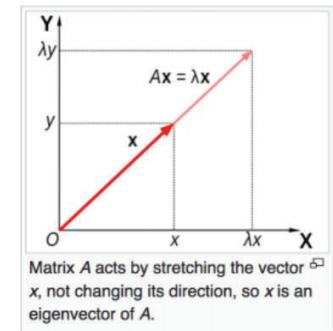
$$v_{\lambda=1} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, v_{\lambda=3} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- An eigenvector of a square matrix  $\mathbf{A}$  is a non-zero vector  $\mathbf{v}$  such that multiplication by  $\mathbf{A}$  only changes the scale of  $\mathbf{v}$

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

Where  $\lambda$  is called the eigenvalue

- If  $\mathbf{v}$  is an eigenvector of  $\mathbf{A}$ , so is any rescaled vector  $s\mathbf{v}$ .
  - Note:  $s\mathbf{v}$  still has the same eigen value



[Wikipedia](#)



# What does Eigen Decomp. tell us?

- Provides insights about the matrix:
  - Singular matrix: A matrix is said to be singular **if & only if** any eigenvalue is zero
  - Useful to optimize quadratic expressions of form
$$f(x) = x^T Ax \text{ such that } \|x\|_2 = 1$$
- Whenever  $x$  is equal to an eigenvector,  $f$  is equal to the corresponding eigenvalue
- Maximum value of  $f$  is max eigen value, minimum value is min eigen value
- This property is very useful in solving several algorithm formulations such as modeling a multivariate Gaussian

$$N(\mathbf{x} | \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}$$



# Things to remember

- A matrix whose eigenvalues are all positive is called *positive definite*
  - Positive or zero is called *positive semidefinite*
  - Why is this important?
    - Positive definite matrices guarantee that  $x^T Ax \geq 0$
- If eigen values are all negative it is negative definite



# Single Value Decomposition

- Eigen decomposition of A is of the form
$$A = V \text{diag}(\lambda) V^{-1}$$
  - If A is not square, you cannot do Eigen decomposition
- SVD can help solve this issue.
  - It is of the form  $A = UDV^T$
- It is more general than Eigen decomposition
  - Can be used for any matrix
    - Eigen is restricted to symmetric, square matrices
  - All real matrices can be factorized using SVD



## SVD (contd.)

- It is of the form  $A = UDV^T$
- U and V are orthogonal matrices
- D is a diagonal matrix
  - Not necessarily square
    - Elements of Diagonal of D are called singular values of A
    - Columns of U are called left singular vectors
    - Columns of V are called right singular vectors
- SVD can be represented in terms of Eigen decomposition:
  - Left singular vectors of  $\mathbf{A}$  are eigenvectors of  $\mathbf{AA}^T$
  - Right singular vectors of  $\mathbf{A}$  are eigenvectors of  $\mathbf{A}^T\mathbf{A}$
  - Nonzero singular values of  $\mathbf{A}$  are square roots of eigen values of  $\mathbf{A}^T\mathbf{A}$ . Same is true of  $\mathbf{AA}^T$



# Moore-Penrose pseudoinverse

- Most useful feature of SVD is that it can be used to generalize matrix inversion to non-square matrices
- **pseudoinverse** of a matrix generalizes the notion of an inverse
  - Not every matrix has an inverse, but every matrix has a pseudoinverse, even non-square matrices.
- Practical algorithms for computing the pseudoinverse of A are based on SVD

$$A^+ = V D^+ U^T$$

- where U,D,V are the SVD of A, '+' refers to the pseudoinverse
  - Pseudoinverse ( $D^+$ ) of D is obtained by taking the reciprocal of its nonzero elements when taking transpose of resulting matrix



**If you did not understand it in detail**



If you did not understand it in detail



# How to do all this in Python?



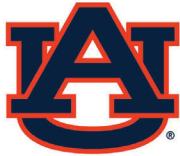
# NumPy

- NumPy is a Python library
- Supports large, multi-dimensional arrays and matrices
- Provides a large collection of high-level mathematical functions to operate on these arrays.
- Runs on CPU
- Highly optimized by computer scientists and mathematicians



# COMP [56]630– Machine Learning

Lecture 3 – Linear Regression Pt. 1



# Logistics

- Assignment 0 due Friday 01/26 11:59 pm
- No Quiz this week



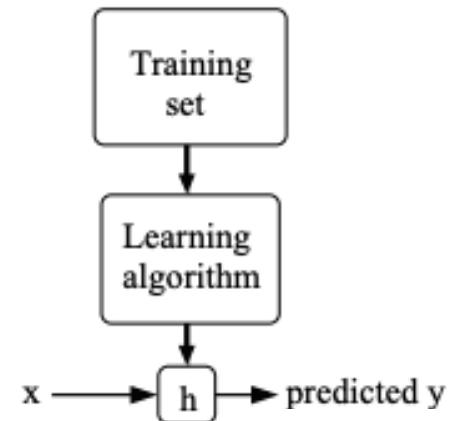
# Supervised Learning – Linear Models

Finally!



# The basics

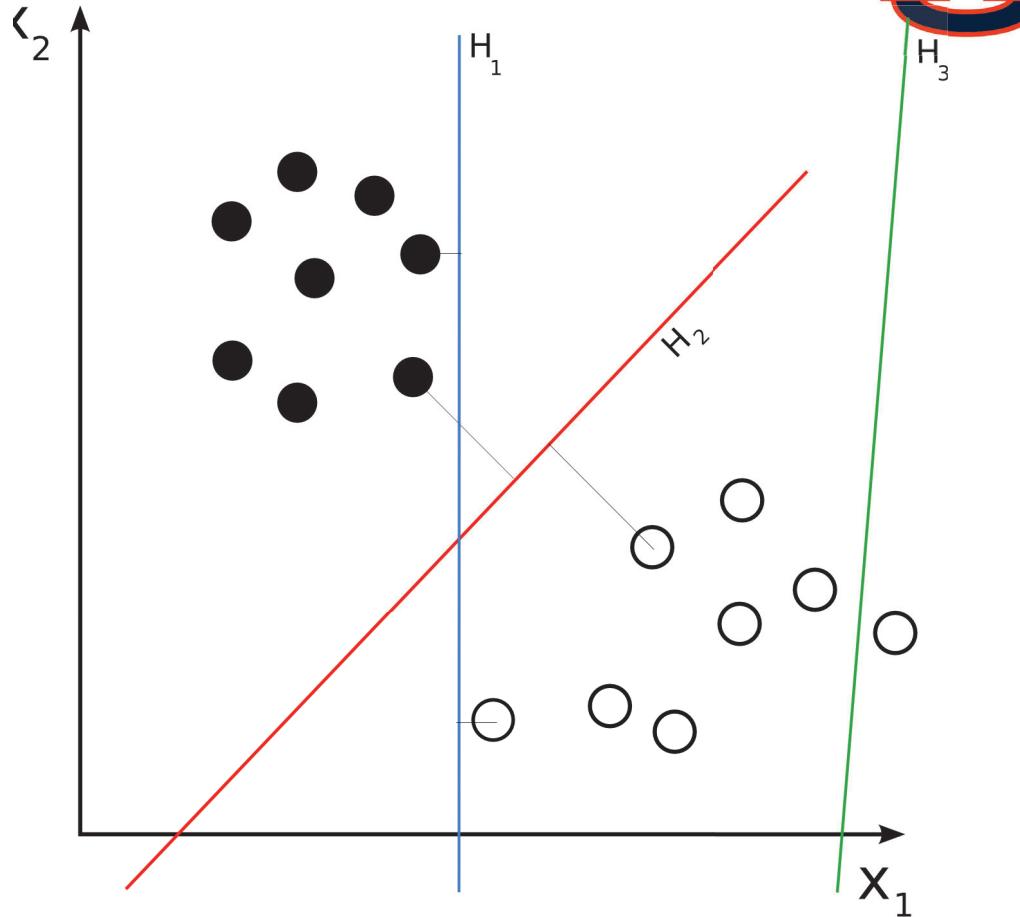
- Input: a set of inputs  $X = \{x_1, x_2, \dots x_n\}$ , also called **features**
- Output: a set of expected outputs or **targets**  $Y = \{y_1, y_2, \dots y_n\}$
- Goal: to learn a function  $h : X \rightarrow Y$  such that the function  $h(x_i)$  is a good predictor of the corresponding value  $y_i$ 
  - $h(x)$  is called the **hypothesis**
- If the target is continuous the problem setting is called **regression**.
- If the target is discrete or categorical, the problem is called **classification**.





# Linear Classifier

- The simplest ML model
- Makes a *classification* decision based on the value of a linear combination of the characteristics (features).
- Black and white circles are different labels.  $H_1, H_2, \dots$  represent different *decision boundaries* i.e. linear functions that best map the classification process.
  - **Goal:** find the best linear function that has highest accuracy





## Linear Classifier (cntd.)

- Mathematically represented as

$$y = \mathbf{W}\mathbf{x}^T + b$$

where  $y \rightarrow$  labels (vector)

$\mathbf{W} \rightarrow$  model parameter matrix

$\mathbf{x} \rightarrow$  feature vector

$b \rightarrow$  bias term (scalar)

- Very similar in the mathematical representation of a line

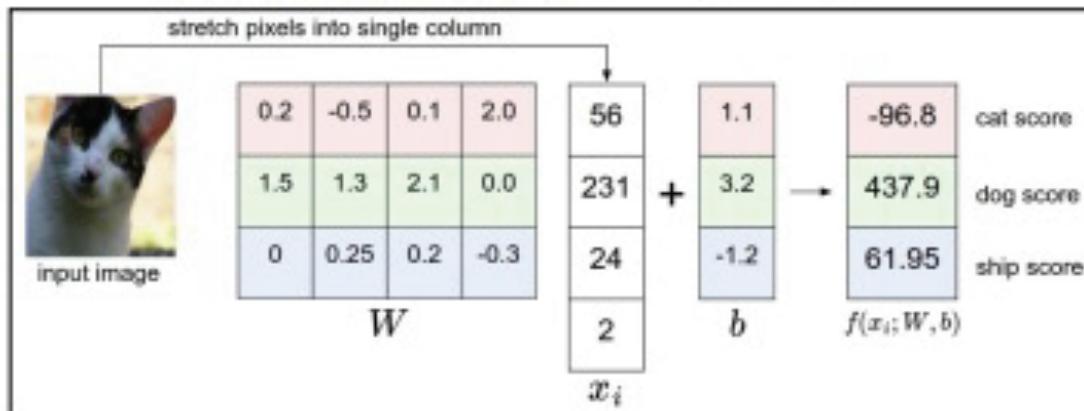
$$y = mx + c$$

→ Hence the term ***linear classifier***



## Linear Classifier (cntd.)

$$\text{A linear classifier } y = \mathbf{W}\mathbf{x}^T + b$$





# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)
1643	4	256
1356	3	202
1678	3	287
...	...	...
3000	4	400



# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)
1643	4	256
1356	3	202
1678	3	287
...	...	...
3000	4	400

**Features (X)** A red bracket is positioned below the first two columns of the table, spanning from the header to the last data row. A red arrow points from the text 'Features (X)' to the right end of this bracket.

**Targets (Y)** A red arrow points from the text 'Targets (Y)' to the third column of the table, specifically pointing at the value '400'.



# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)
1643	4	256
1356	3	202
1678	3	287
...	...	...
3000	4	400

Features (X)      Targets (Y)

Targets are continuous valued! => Task is regression



# Linear Regression

- Goal: formulate a hypothesis function  $h(x)$  which will model the 2-d input feature (size, # bedrooms) and produce the expected target value (the house price in 1000\$s).
- We can say that the hypothesis function could be a linear function of  $x$ :

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- Here,  $\theta_i$  represent the **parameters** or **weights** of the linear model characterizing  $X \rightarrow Y$
- A more simpler model then will be

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$



## Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of  $\theta$ .
- We will need to use the *training* data to learn these parameters. This process is called *learning*
- **What do we need to achieve this?**

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$



## Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of  $\theta$ .
- We will need to use the *training* data to learn these parameters. This process is called *learning*
- **What do we need to achieve this?**
- We will define a function that measures the quality of predictions for each value of  $\theta$ .
- This is called the **cost function or objective function**

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$



# Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of  $\theta$ .
- We will need to use the *training* data to learn these parameters. This process is called *learning*
- **What do we need to achieve this?**
- We will define a function that measures the quality of predictions for each value of  $\theta$ .
- This is called the **cost function or objective function**

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2.$$



## Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of  $\theta$ .
- We will need to use the *training* data to learn these parameters. This process is called *learning*
- **What do we need to achieve this?**
- We will define a function that measures the quality of predictions for each value of  $\theta$ .
- This is called the **cost function or objective function**

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2.$$



*Ordinary least squares  
regression model*



# Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of  $\theta$ .
- We will need to use the *training* data to learn these parameters. This process is called *learning*
- **What do we need to achieve this?**
- We will define a function that measures the quality of predictions for each value of  $\theta$ .
- This is called the **cost function or objective function**

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$

*How to obtain  
parameter matrix  
using this objective  
function?*

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2.$$

*Ordinary least squares  
regression model*



# Learning

- Goal: To find a set of parameters  $\theta$  that will minimize the cost function  $J(\theta)$ .
- Common approach: *gradient descent*



# Learning

- Goal: To find a set of parameters  $\theta$  that will minimize the cost function  $J(\theta)$ .
- Common approach: *gradient descent*
- What does it do?



# Learning

- Goal: To find a set of parameters  $\theta$  that will minimize the cost function  $J(\theta)$ .
- Common approach: *gradient descent*
- What does it do?
  - Start with an initial “guess” for  $\theta$
  - Update values of  $\theta$  that will gradually move towards the “optimal solution”
  - What is the optimal solution?
  - The value of  $\theta$  that minimizes the cost function
- How do we do it computationally?



# Gradient Descent

- How do we do it computationally?

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- $\alpha$  is the learning rate
  - Modulates how much of the change that we need to propagate at each instant
- Each update of  $\theta$  will be a step in the *steepest decrease of the cost function  $J(\theta)$*
- How to Compute the derivative of the cost function  $J(\theta)$ ?



$$h_{\theta}(x) = \theta^T x$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2$$

$$\frac{d J(\theta)}{d \theta} = \frac{d}{d \theta} \left[ \frac{1}{2} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2 \right]$$



For convenience,  $x^{(i)} \rightarrow x$ ,  $y^{(i)} \rightarrow y$   
and compute the summation over the  
derivatives.

$$\frac{d J(\theta)}{d\theta} = \frac{d}{d\theta} \left[ \frac{1}{2} (\theta^T x - y)^2 \right]$$

$$= \frac{1}{2} \cdot \frac{d}{d\theta} [( \theta^T x ) - y]^2$$

We know that  $\frac{d}{dz} (z^n) = n \cdot z^{n-1}$



$$\Rightarrow \frac{d J(\theta)}{d \theta} = \frac{1}{2} \cdot \left[ 2 \cdot \frac{d}{d \theta} (\theta^T x) - \frac{d}{d \theta} (y) \right] (\theta^T x - y)$$

We know that  $y$  is the true label

$\Rightarrow y$  is a constant

$$\therefore \frac{d J(\theta)}{d \theta} = \frac{2}{2} \left[ x - \theta \right] (\theta^T x - y)$$

$$\therefore \frac{d (J(\theta))}{d \theta} = (\theta^T x - y) x$$



# Gradient Descent

- Hence each update is given by

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

- This is called the **LMS** update rule or the ***Least Mean Squares*** update rule.
  - Also known as the ***Widrow-Hoff*** learning rule.



# Gradient Descent

- Has several properties:
  - Magnitude of update is proportional to the error ( $y - h(x)$ )
    - What does this mean?



# Gradient Descent

- Has several properties:
  - Magnitude of update is proportional to the error ( $y - h(x)$ )
    - What does this mean?
    - If we have a very good prediction i.e.  $h(x) \approx y$ , then the update is very small.
    - Conversely, if the prediction is very far off i.e.  $h(x) \gg y$  or  $h(x) \ll y$  then the update will be large.
- For learning over the complete training set, we iteratively update the parameters as below

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}



# Gradient Descent

- If we use all the examples in the training set *at once*:
  - Batch gradient descent
- What if we update at every single data point?
  - Stochastic or incremental gradient descent

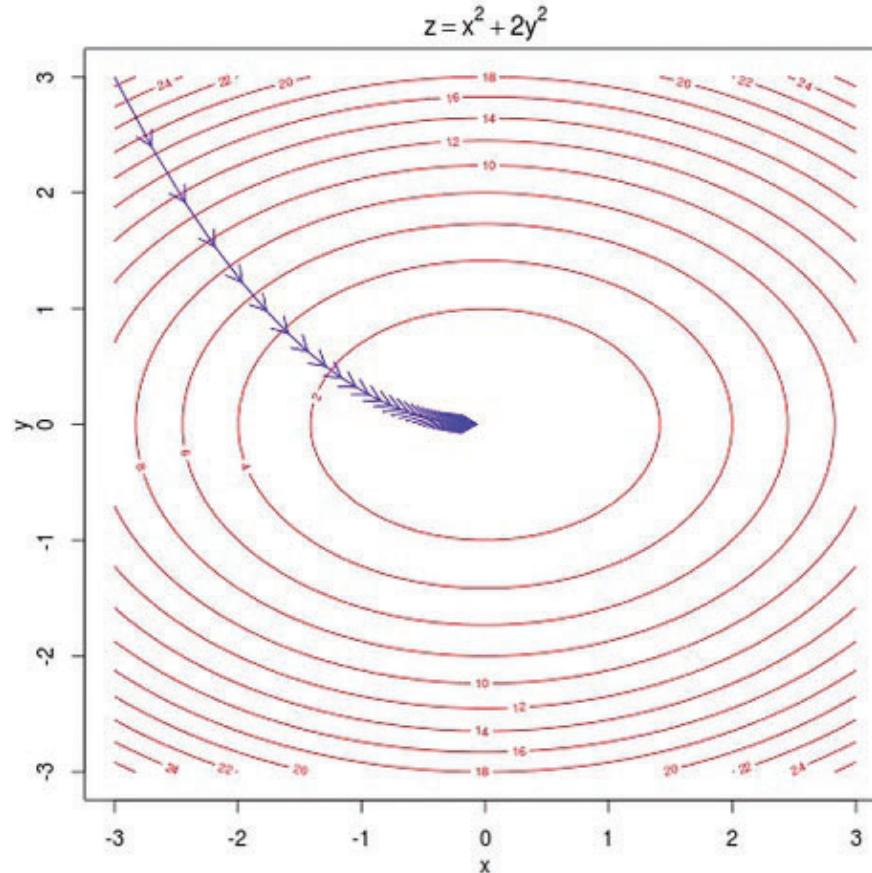
Loop {

    for i=1 to m, {

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j)$$

    }

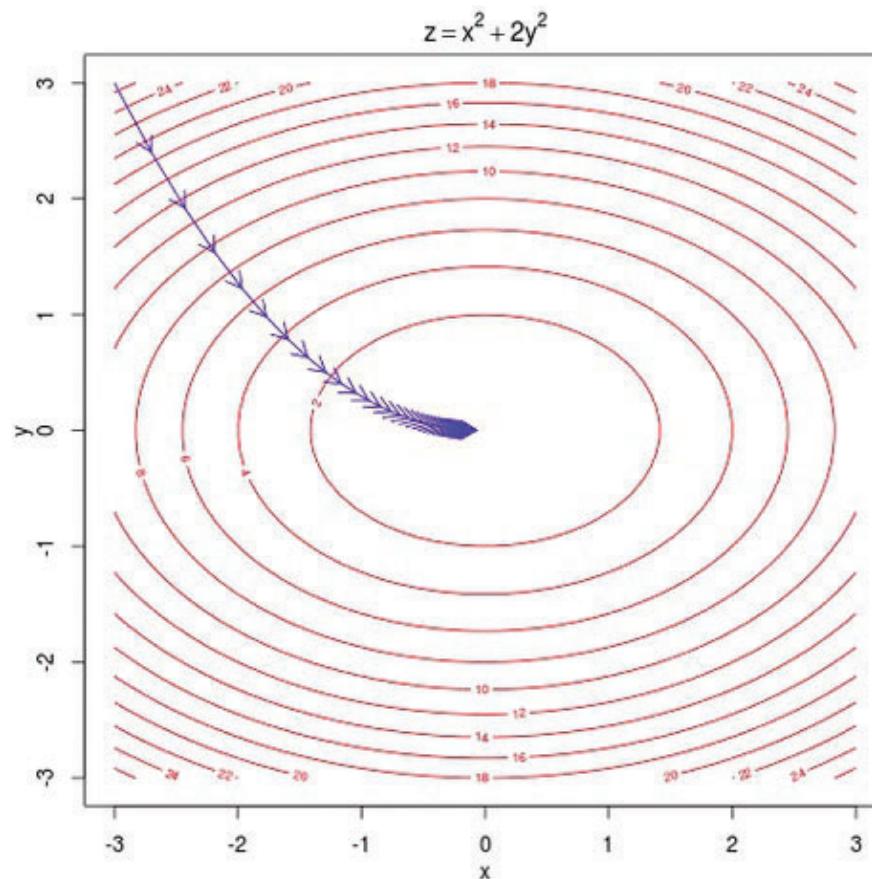
}



- Gradient Descent in 2D. Images Credit: <http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r/>



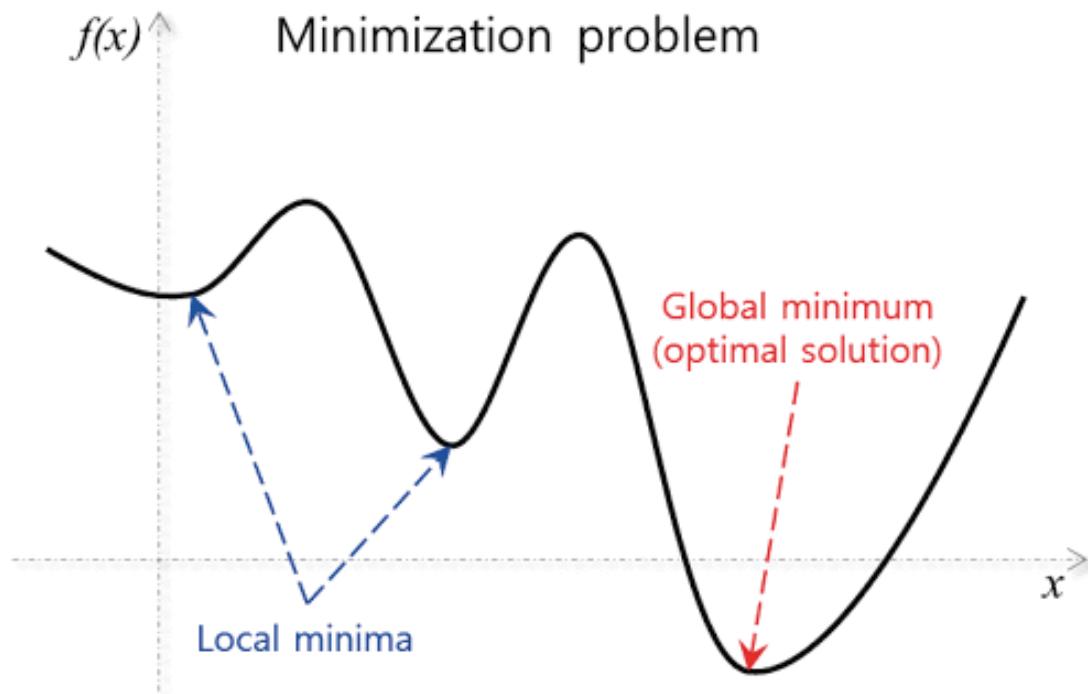
*Will we always  
get an optimal  
solution?*



- Gradient Descent in 2D. Images Credit: <http://vis.supstat.com/2013/03/gradient-descent-algorithm-with-r/>

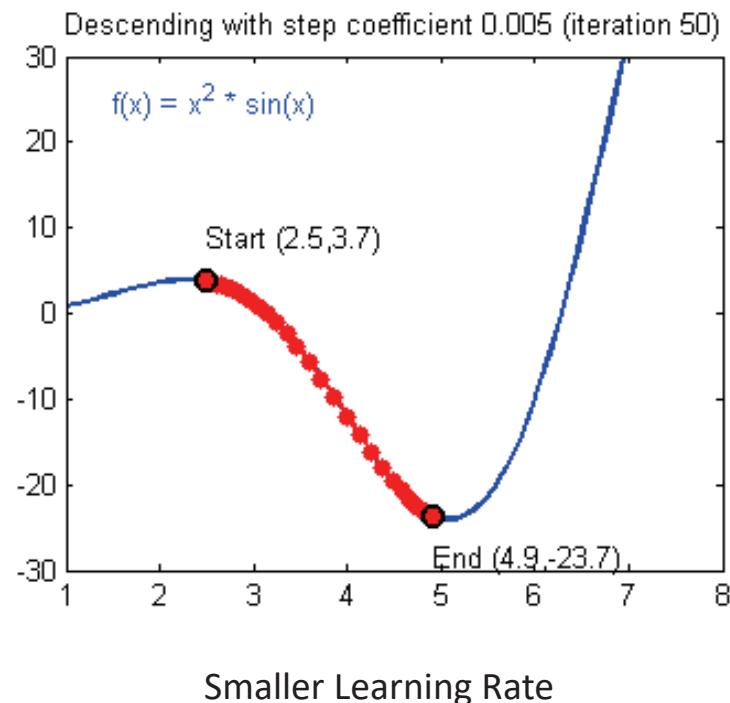


# Not really...



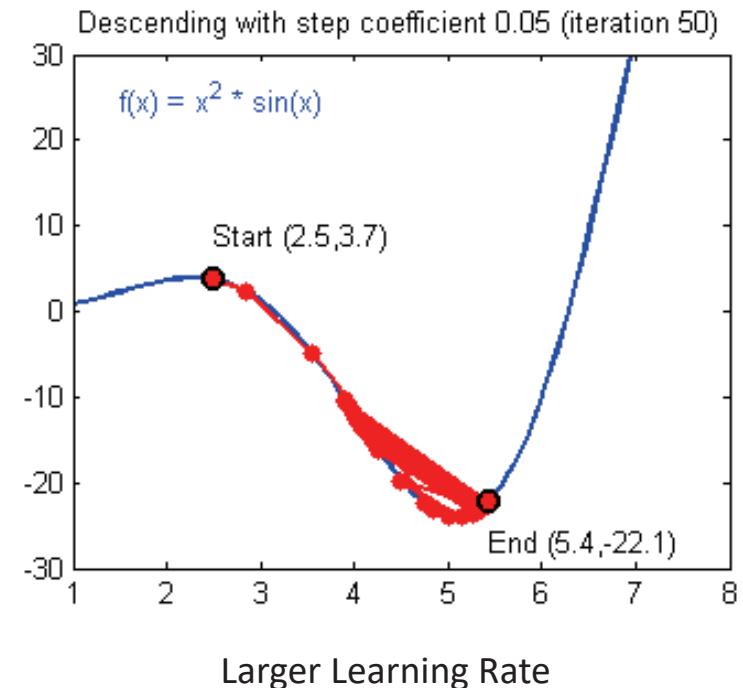
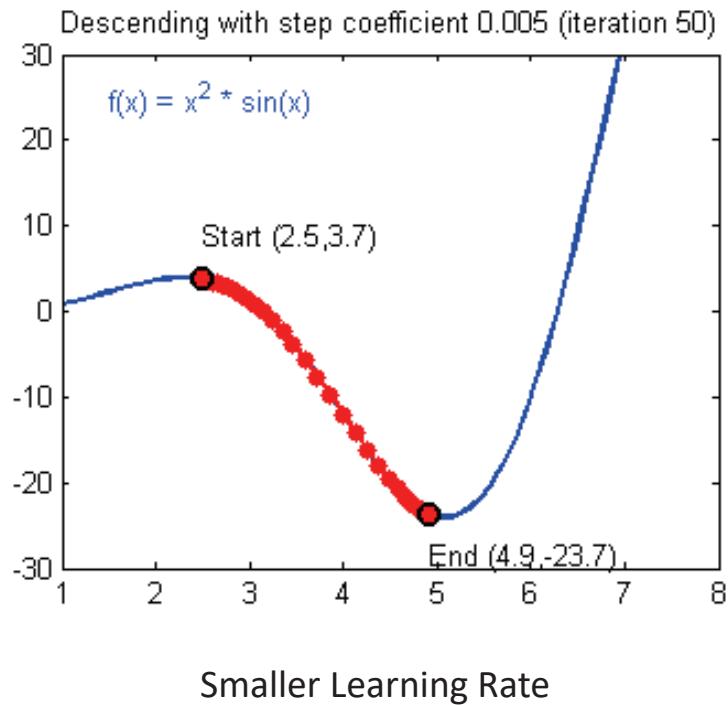


# Is the learning rate that important?



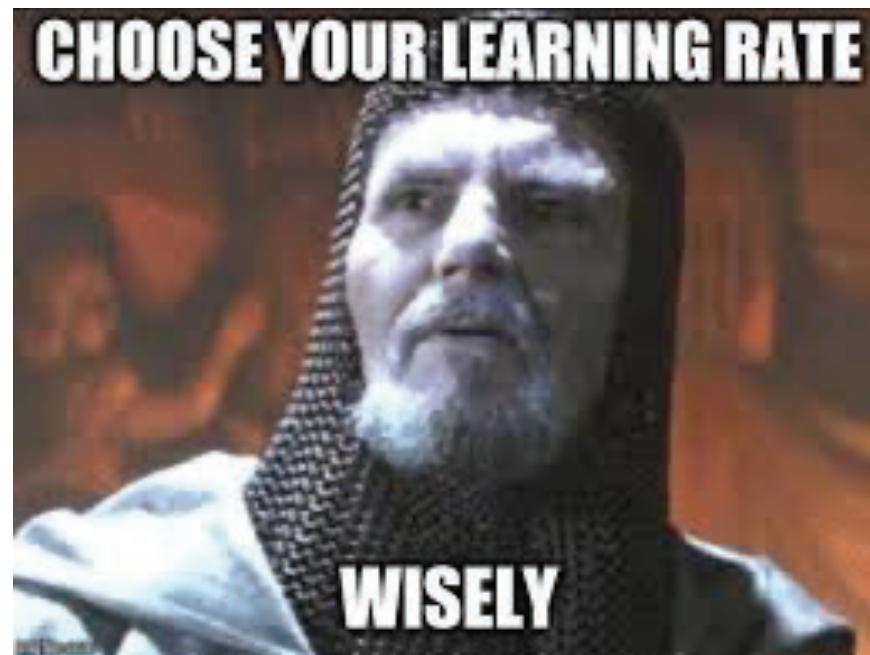


# Is the learning rate that important?





# Is the learning rate that important?





# Is there a non-iterative solution?



# Is there a non-iterative solution?

- Yes! We can directly find the solution using “Normal Equations”.
- It is an analytical approach used for optimization
- Can be done as follows:
  - Set the partial derivatives of the cost function  $J(\theta)$  to zero
    - Remember, vectorized version of  $J(\theta) = \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y})$
    - Hence, taking the Partial Derivative gives us

$$\nabla_{\theta} J(\theta) = X^T X \theta - X^T \vec{y}$$



# Is there a non-iterative solution?

- Yes! We can directly find the solution using “Normal Equations”.
- It is an analytical approach used for optimization
- Can be done as follows:
  - Set the partial derivatives of the cost function  $J(\theta)$  to zero
  - Then you can estimate the parameters as follows:

$$\nabla_{\theta} J(\theta) = X^T X \theta = X^T \vec{y}$$

$$\Theta = (X^T X)^{-1} X^T y$$

- Where  $X$  is the input feature vector
- $y$  is the expected target value



# Linear Regression with Basis Functions



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$
- **But in general, it is rather unlikely that a true function is linear**



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$
- **But in general, it is rather unlikely that a true function is linear**

*How do we handle non-linear relationships?*



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$
- **But in general, it is rather unlikely that a true function is linear**

*How do we handle non-linear relationships?*

- Simple! In addition to the original features, add more features that are deterministic functions of the original features



# Basis Functions

- If the original variables comprise the vector  $x$ , then the features can be expressed in terms of basis functions  $\{\phi_j(x)\}$
- By using nonlinear basis functions, we allow the hypothesis function  $h(x,w)$  to be a nonlinear function of the input vector  $x$ 
  - They are linear functions of parameters, yet are nonlinear with respect to the input variables
- You can have multiple basis functions to model different nonlinearities

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x)$$



# Basis Functions

- If the original variables comprise the vector  $x$ , then the features can be expressed in terms of basis functions  $\{\phi_j(x)\}$
- By using nonlinear basis functions, we allow the hypothesis function  $h(x,w)$  to be a nonlinear function of the input vector  $x$ 
  - They are linear functions of parameters, yet are nonlinear with respect to the input variables
- You can have multiple basis functions to model different nonlinearities

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x)$$

Basis functions



# Basis Functions

- More generally,

$$y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x)$$

where  $w=(w_0, w_1, \dots, w_{M-1})$  and  $\Phi=(\phi_0, \phi_1, \dots, \phi_{M-1})^T$

- We now need  $M$  weights for basis functions instead of  $D$  weights for features



# Types of Basis Functions

- Linear Basis function:

$$y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x)$$

- Polynomial Basis Function:

- Model with M-1 degree polynomial  $\phi_j(x) = x^j$
- Works really well for 1-d feature vector i.e. only modelling one variable  $x$



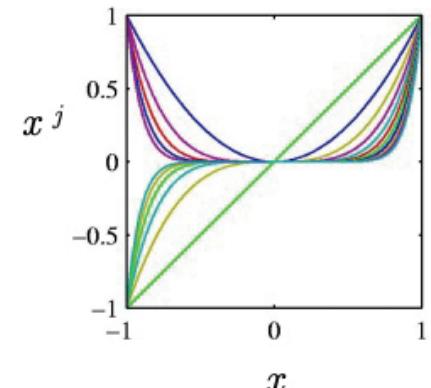
# Types of Basis Functions

- Linear Basis function:

$$y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(x) = \mathbf{w}^T \phi(x)$$

- Polynomial Basis Function:

- Model with M-1 degree polynomial  $\boxed{\phi_j(x) = x^j}$
- Works really well for 1-d feature vector i.e. only modelling one variable  $x$





# Types of Basis Functions

- Linear Basis function:

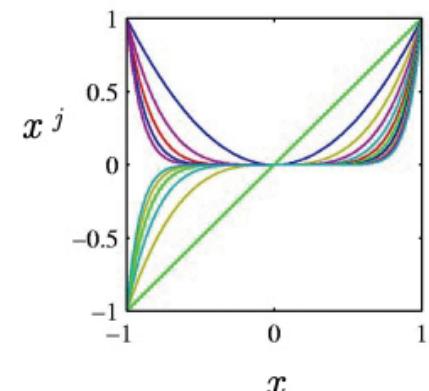
$$y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x)$$

- Polynomial Basis Function:

- Model with M-1 degree polynomial  $\phi_j(x) = x^j$
- Works really well for 1-d feature vector i.e. only modelling one variable  $x$

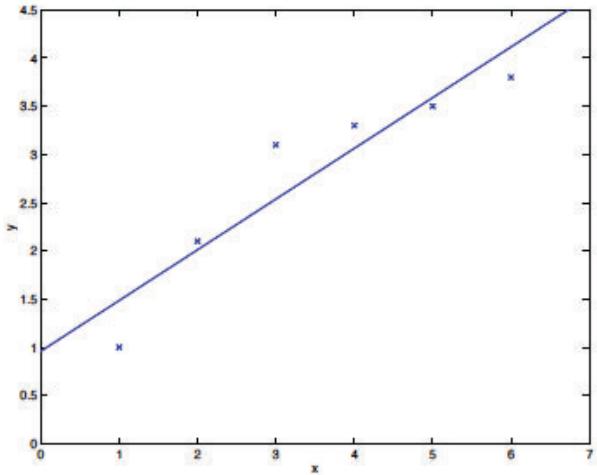
- Disadvantage

- Global:
  - changes in one region of input space affects others
- Difficult to formulate
  - Number of polynomials increases exponentially with M
- Can divide input space into regions
  - use different polynomials in each region
  - equivalent to spline functions





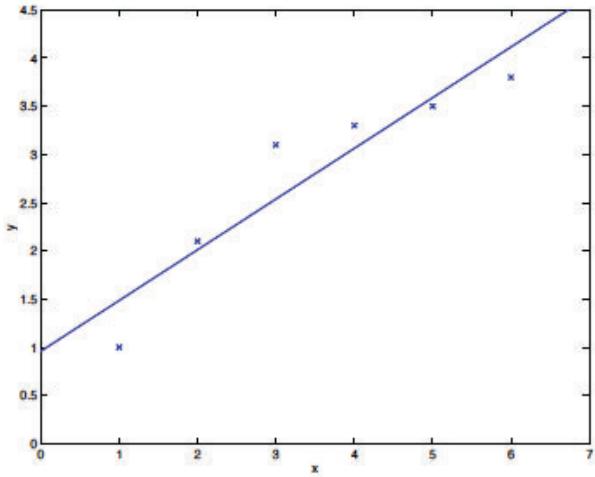
# Example



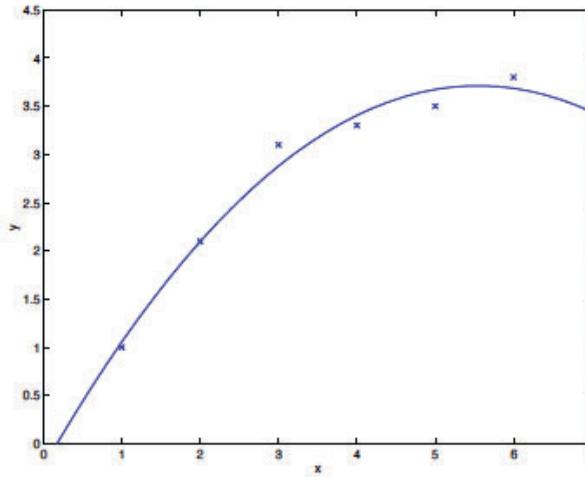
Linear Basis Function



# Example



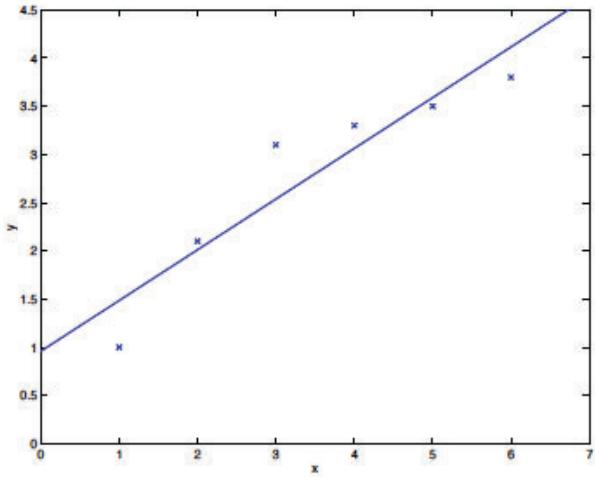
Linear Basis Function



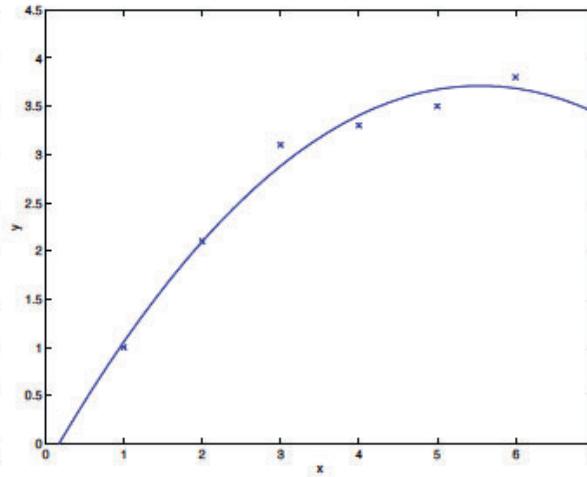
2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



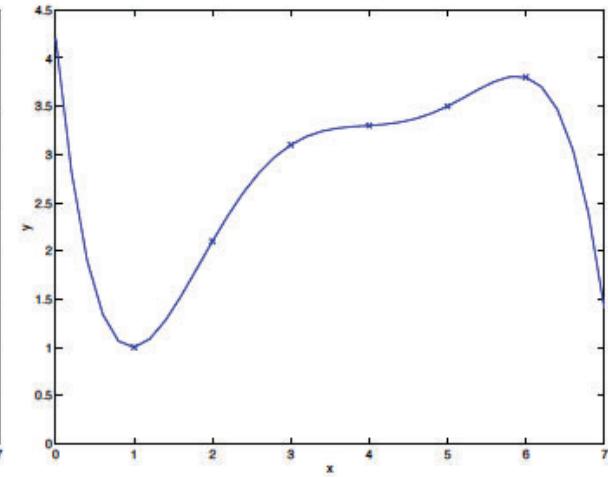
# Example



Linear Basis Function



2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

$$y = \sum_{j=0}^5 \theta_j x^j$$



# Other Basis Functions

- Gaussian Radial Basis function:
  - $\mu_j$  govern the locations of the basis functions
    - Can be an arbitrary set of points within the range of the data
      - Can choose some representative data points
    - $\sigma$  governs the spatial scale
      - Could be chosen from the data set e.g., average variance

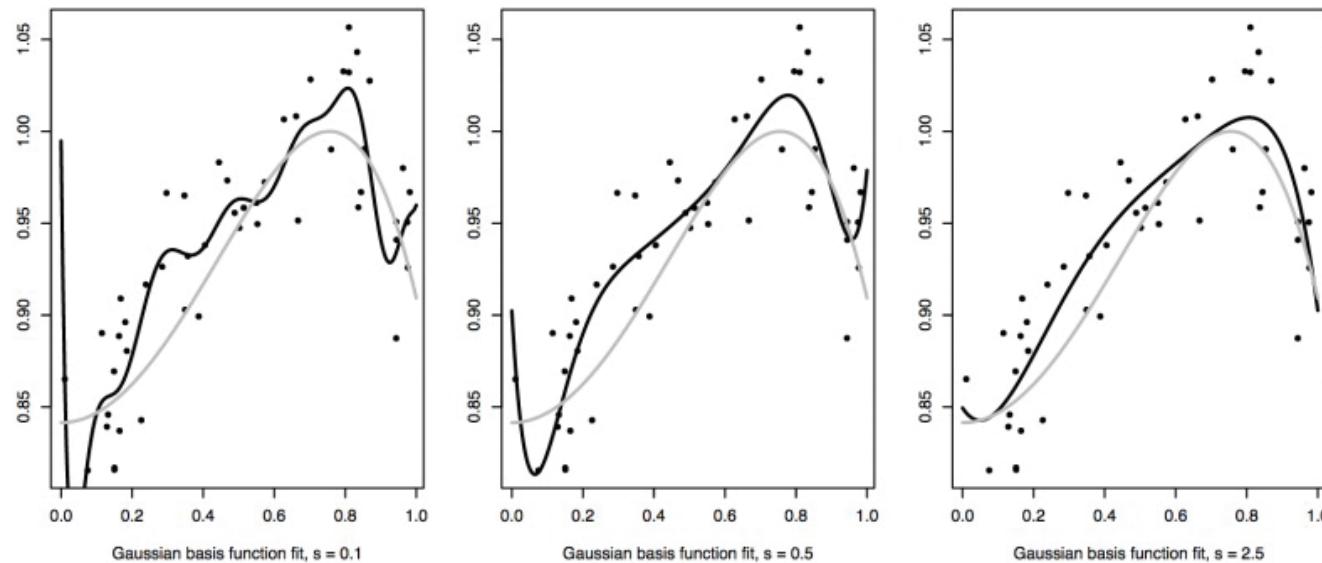
$$\phi_j(x) = \exp\left(\frac{(x - \mu_j)^2}{2\sigma^2}\right)$$



# Other Basis Functions

- Gaussian Radial Basis function:
  - $\mu_j$  govern the locations of the basis functions
  - $\sigma$  governs the spatial scale
    - Could be chosen from the data set e.g., average variance

$$\phi_j(x) = \exp\left(\frac{(x - \mu_j)^2}{2\sigma^2}\right)$$





# Other Basis Functions

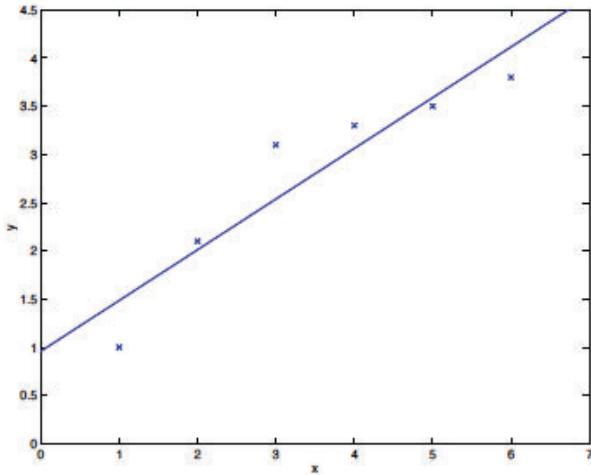
- Sigmoid:  $\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$
- Fourier
  - Expansion in sinusoidal functions
- Signal Processing
  - Also called *wavelets*
  - Functions grounded in time and frequency
- Linear
  - $\phi(x) = x$



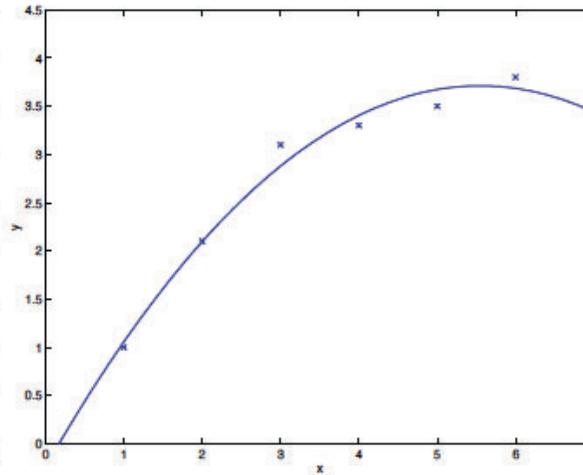
# Does more features always mean better?



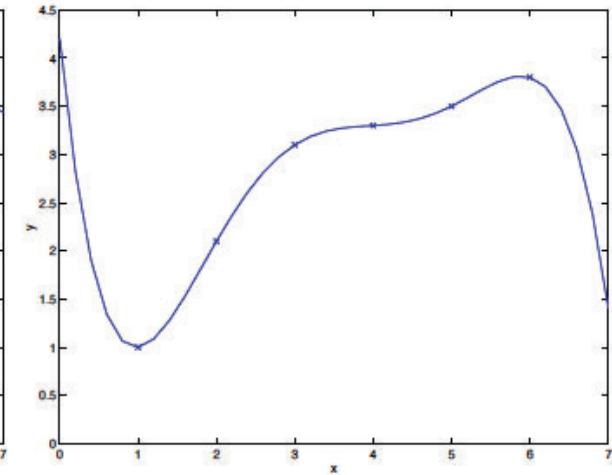
# Does more features always mean better?



Linear Basis Function



2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature

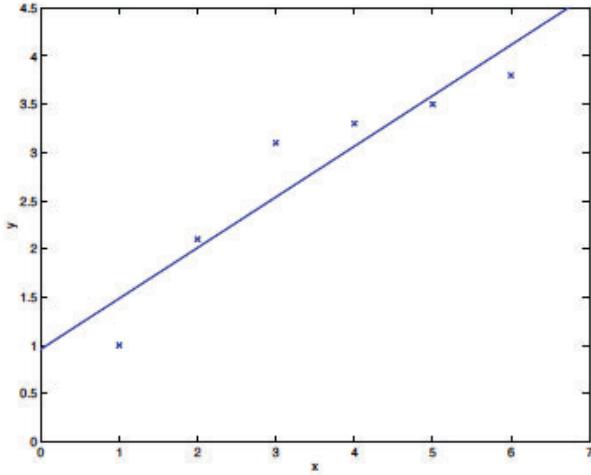


5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

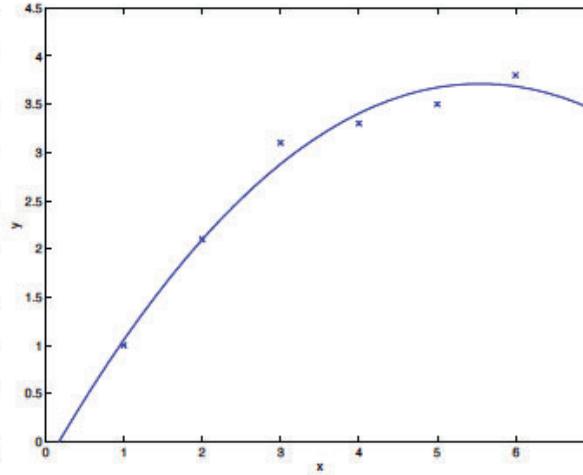
$$y = \sum_{j=0}^5 \theta_j x^j$$



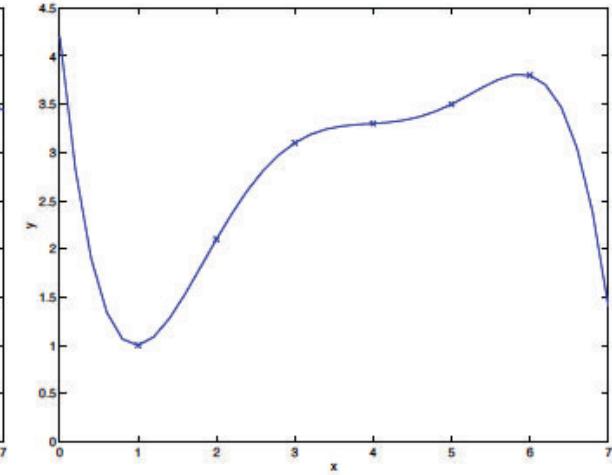
# Does more features always mean better?



Linear Basis Function



2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



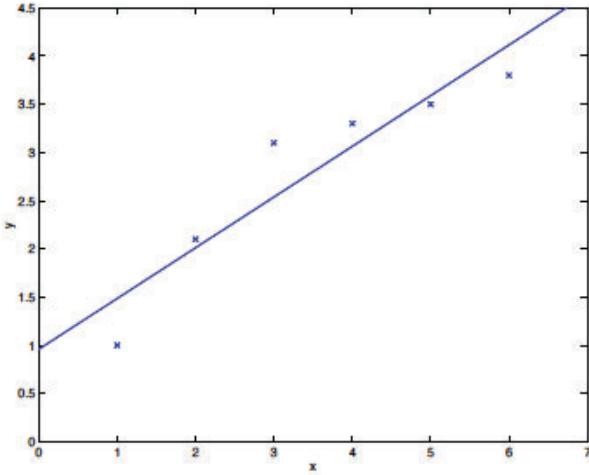
5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

$$y = \sum_{j=0}^5 \theta_j x^j$$

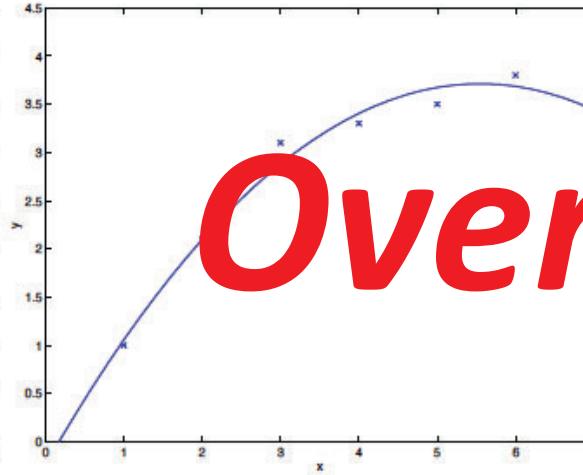
# NO!



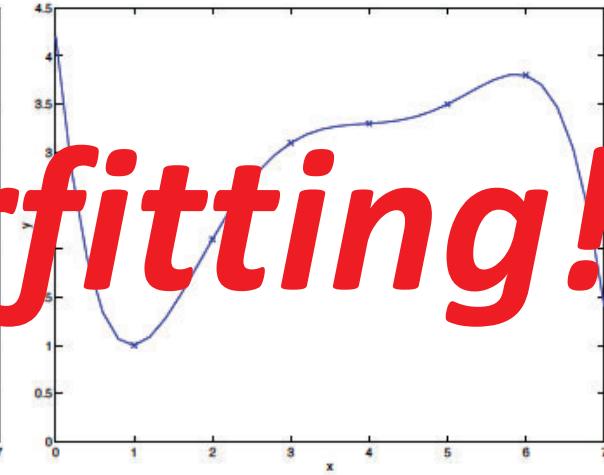
# Does more features always mean better?



Linear Basis Function



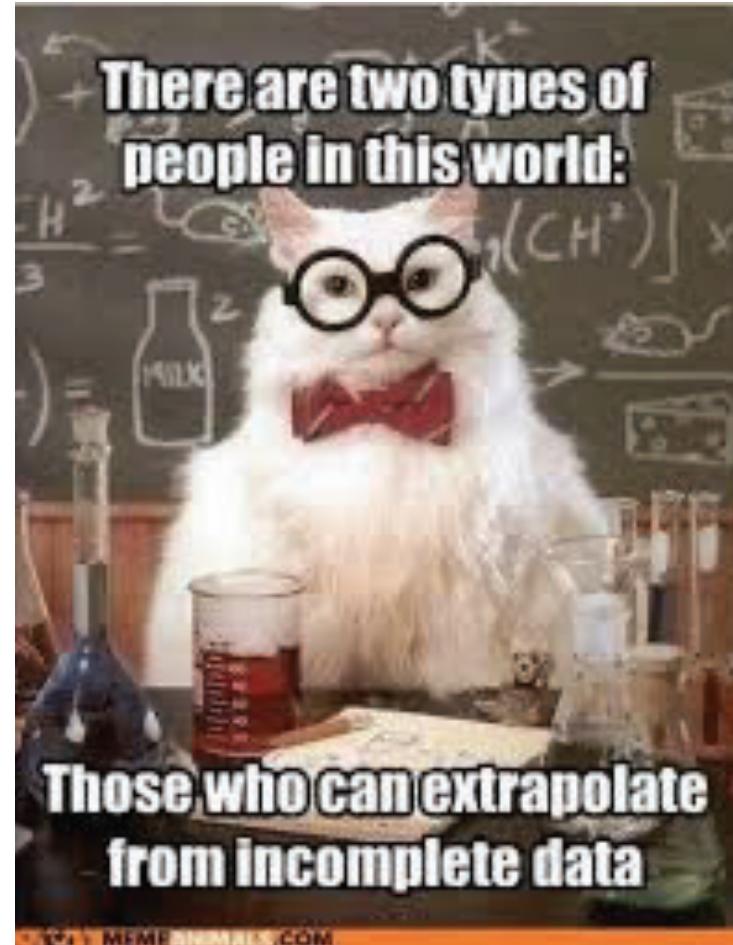
2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



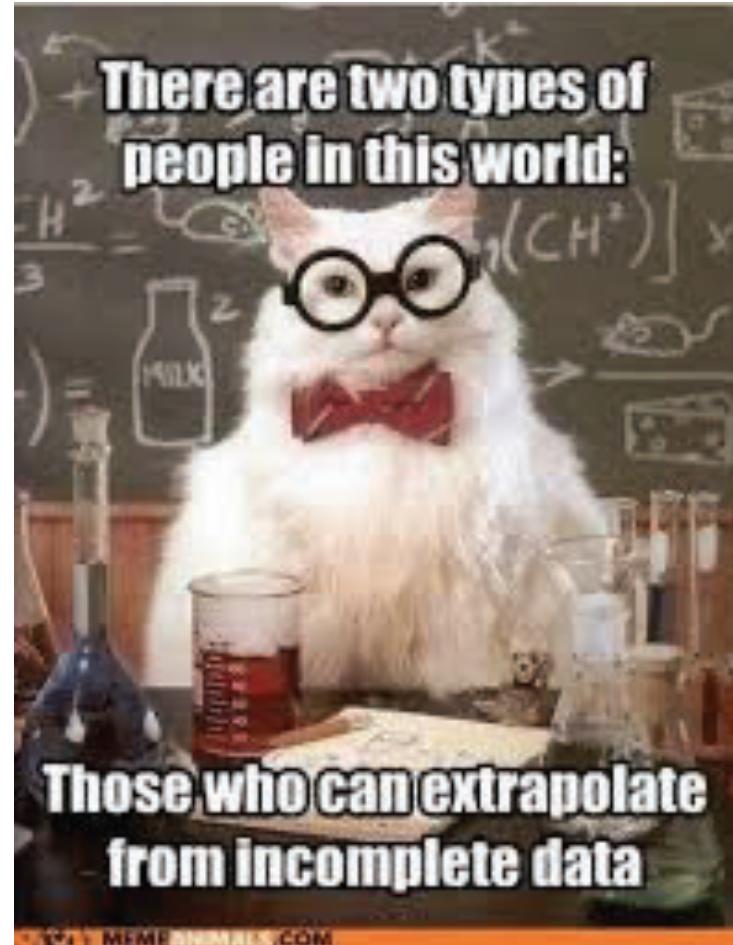
5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

$$y = \sum_{j=0}^5 \theta_j x^j$$

# NO!



© Sathyanarayanan Aakur



*You don't want  
your model to  
be other kind!*





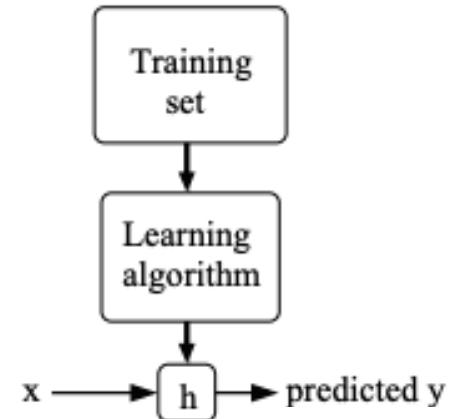
# COMP [56]630– Machine Learning

Lecture 4 – Linear Regression Pt. 2



# The basics

- Input: a set of inputs  $X = \{x_1, x_2, \dots x_n\}$ , also called **features**
- Output: a set of expected outputs or **targets**  $Y = \{y_1, y_2, \dots y_n\}$
- Goal: to learn a function  $h : X \rightarrow Y$  such that the function  $h(x_i)$  is a good predictor of the corresponding value  $y_i$ 
  - $h(x)$  is called the **hypothesis**
- If the target is continuous the problem setting is called **regression**.
- If the target is discrete or categorical, the problem is called **classification**.





# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)
1643	4	256
1356	3	202
1678	3	287
...	...	...
3000	4	400



# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)
1643	4	256
1356	3	202
1678	3	287
...	...	...
3000	4	400

Features (X)      Targets (Y)

Targets are continuous valued! => Task is regression



# Linear Regression

- Goal: formulate a hypothesis function  $h(x)$  which will model the 2-d input feature (size, # bedrooms) and produce the expected target value (the house price in 1000\$s).
- We can say that the hypothesis function could be a linear function of  $x$ :

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- Here,  $\theta_i$  represent the **parameters** or **weights** of the linear model characterizing  $X \rightarrow Y$
- A more simpler model then will be

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$



# Linear Regression (cntd.)

- **Learning:** Given this formulation, we will need to identify a way to find the values of  $\theta$ .
- We will need to use the *training* data to learn these parameters. This process is called *learning*
- **What do we need to achieve this?**
- We will define a function that measures the quality of predictions for each value of  $\theta$ .
- This is called the **cost function or objective function**

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$

*How to obtain  
parameter matrix  
using this objective  
function?*

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2.$$

*Ordinary least squares  
regression model*



# Learning

- Goal: To find a set of parameters  $\theta$  that will minimize the cost function  $J(\theta)$ .
- Common approach: *gradient descent*
- What does it do?
  - Start with an initial “guess” for  $\theta$
  - Update values of  $\theta$  that will gradually move towards the “optimal solution”
  - What is the optimal solution?
  - The value of  $\theta$  that minimizes the cost function
- How do we do it computationally?



# Gradient Descent

- How do we do it computationally?

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- $\alpha$  is the learning rate
  - Modulates how much of the change that we need to propagate at each instant
- Each update of  $\theta$  will be a step in the *steepest decrease of the cost function  $J(\theta)$*
- How to Compute the derivative of the cost function  $J(\theta)$ ?



# Gradient Descent

- Hence each update is given by

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

- This is called the **LMS** update rule or the ***Least Mean Squares*** update rule.
  - Also known as the ***Widrow-Hoff*** learning rule.



# Gradient Descent

- Has several properties:
  - Magnitude of update is proportional to the error ( $y - h(x)$ )
    - What does this mean?
    - If we have a very good prediction i.e.  $h(x) \approx y$ , then the update is very small.
    - Conversely, if the prediction is very far off i.e.  $h(x) \gg y$  or  $h(x) \ll y$  then the update will be large.
- For learning over the complete training set, we iteratively update the parameters as below

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

}



# Is there a non-iterative solution?

- Yes! We can directly find the solution using “Normal Equations”.
- It is an analytical approach used for optimization
- Can be done as follows:
  - Set the partial derivatives of the cost function  $J(\theta)$  to zero
  - Then you can estimate the parameters as follows:

$$\nabla_{\theta} J(\theta) = 0$$

$$X^T X \theta = X^T \vec{y}$$

$$\Theta = (X^T X)^{-1} X^T y$$

- Where  $X$  is the input feature vector
- $y$  is the expected target value



# Linear Regression with Basis Functions



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$
- **But in general, it is rather unlikely that a true function is linear**



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$
- **But in general, it is rather unlikely that a true function is linear**

*How do we handle non-linear relationships?*



# Basis Functions

- Sometimes, you will need to have some fixed pre-processing of the input data.
  - Also called feature extraction
- Linearity is often a good assumption when many inputs influence the output
  - Natural examples include  $F = ma$
- **But in general, it is rather unlikely that a true function is linear**

*How do we handle non-linear relationships?*

- Simple! In addition to the original features, add more features that are deterministic functions of the original features



# Basis Functions

- If the original variables comprise the vector  $x$ , then the features can be expressed in terms of basis functions  $\{\phi_j(x)\}$
- By using nonlinear basis functions, we allow the hypothesis function  $h(x,w)$  to be a nonlinear function of the input vector  $x$ 
  - They are linear functions of parameters, yet are nonlinear with respect to the input variables
- You can have multiple basis functions to model different nonlinearities

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x)$$



# Basis Functions

- If the original variables comprise the vector  $x$ , then the features can be expressed in terms of basis functions  $\{\phi_j(x)\}$
- By using nonlinear basis functions, we allow the hypothesis function  $h(x,w)$  to be a nonlinear function of the input vector  $x$ 
  - They are linear functions of parameters, yet are nonlinear with respect to the input variables
- You can have multiple basis functions to model different nonlinearities

$$y(x,w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x)$$

Basis functions



# Basis Functions

- More generally,

$$y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x)$$

where  $w=(w_0, w_1, \dots, w_{M-1})$  and  $\Phi=(\phi_0, \phi_1, \dots, \phi_{M-1})^T$

- We now need  $M$  weights for basis functions instead of  $D$  weights for features



# Types of Basis Functions

- Linear Basis function:

$$y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x)$$

- Polynomial Basis Function:

- Model with M-1 degree polynomial  $\phi_j(x) = x^j$
- Works really well for 1-d feature vector i.e. only modelling one variable  $x$



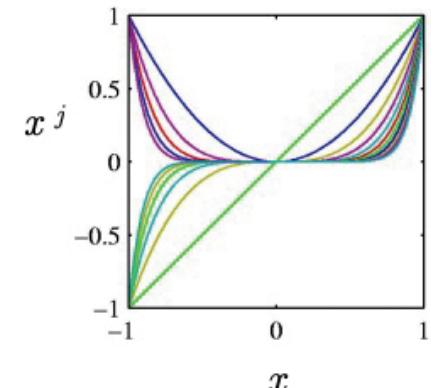
# Types of Basis Functions

- Linear Basis function:

$$y(x, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(x) = \mathbf{w}^T \phi(x)$$

- Polynomial Basis Function:

- Model with M-1 degree polynomial  $\boxed{\phi_j(x) = x^j}$
- Works really well for 1-d feature vector i.e. only modelling one variable  $x$





# Types of Basis Functions

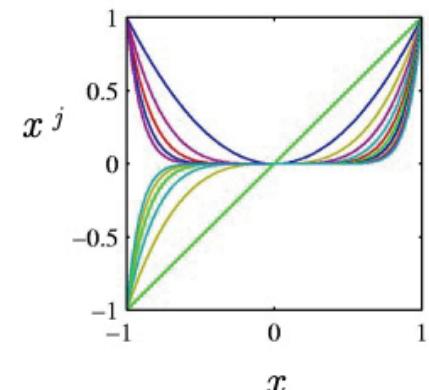
- Linear Basis function:

$$y(x, w) = \sum_{j=0}^{M-1} w_j \phi_j(x) = w^T \phi(x)$$

- Polynomial Basis Function:
  - Model with M-1 degree polynomial  $\phi_j(x) = x^j$
  - Works really well for 1-d feature vector i.e. only modelling one variable  $x$

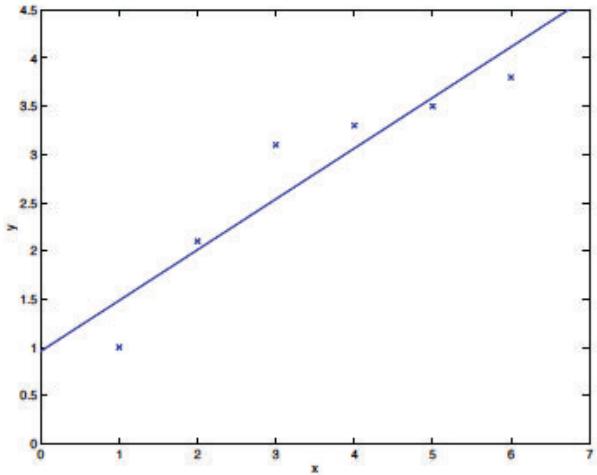
- Disadvantage

- Global:
  - changes in one region of input space affects others
- Difficult to formulate
  - Number of polynomials increases exponentially with M
- Can divide input space into regions
  - use different polynomials in each region
  - equivalent to spline functions

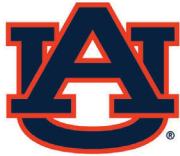




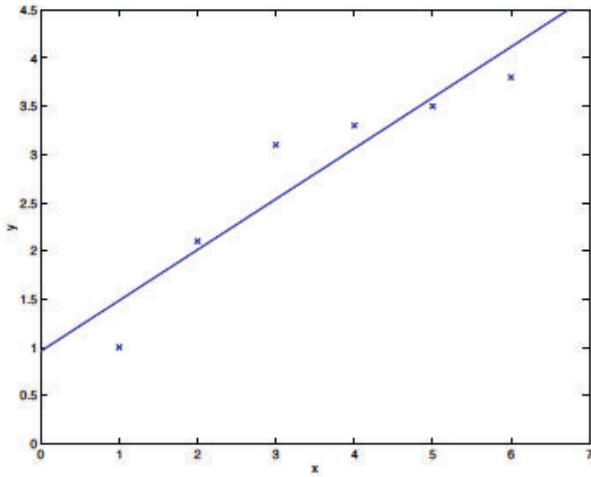
# Example



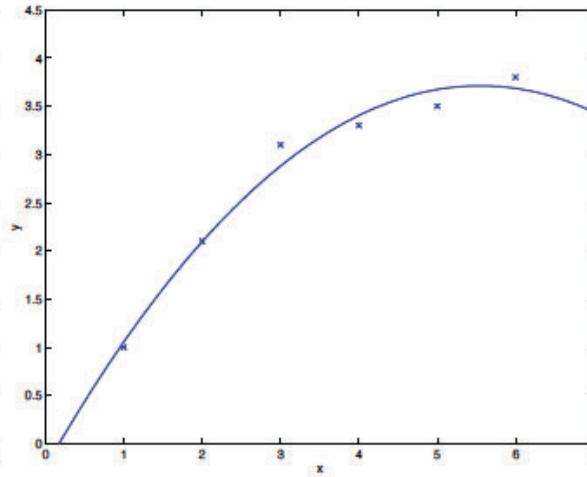
Linear Basis Function



# Example



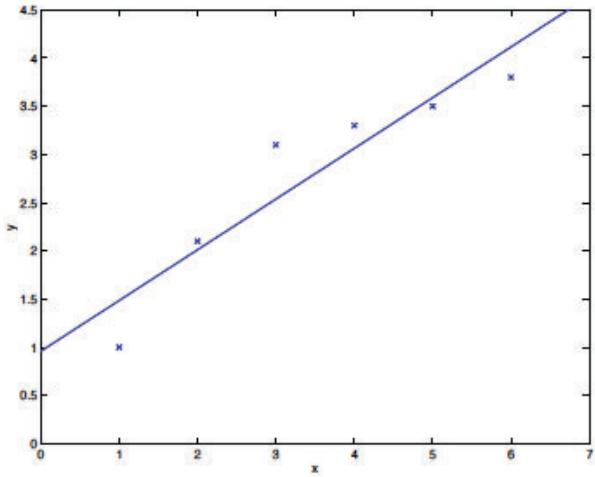
Linear Basis Function



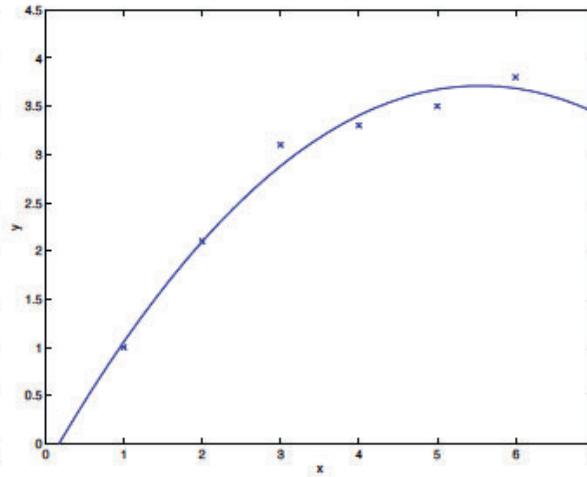
2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



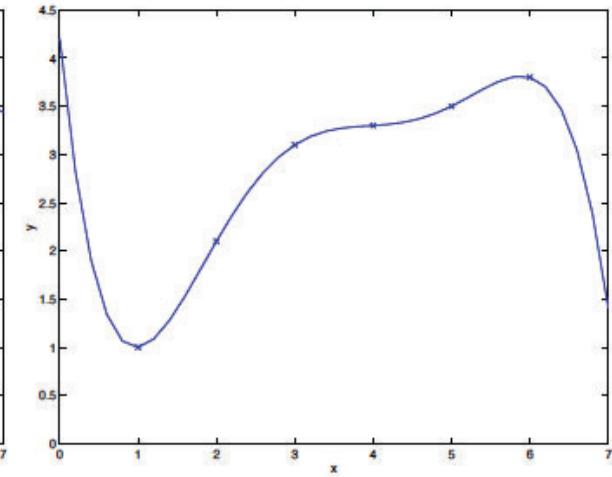
# Example



Linear Basis Function



2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

$$y = \sum_{j=0}^5 \theta_j x^j$$



# Other Basis Functions

- Gaussian Radial Basis function:
  - $\mu_j$  govern the locations of the basis functions
    - Can be an arbitrary set of points within the range of the data
      - Can choose some representative data points
    - $\sigma$  governs the spatial scale
      - Could be chosen from the data set e.g., average variance

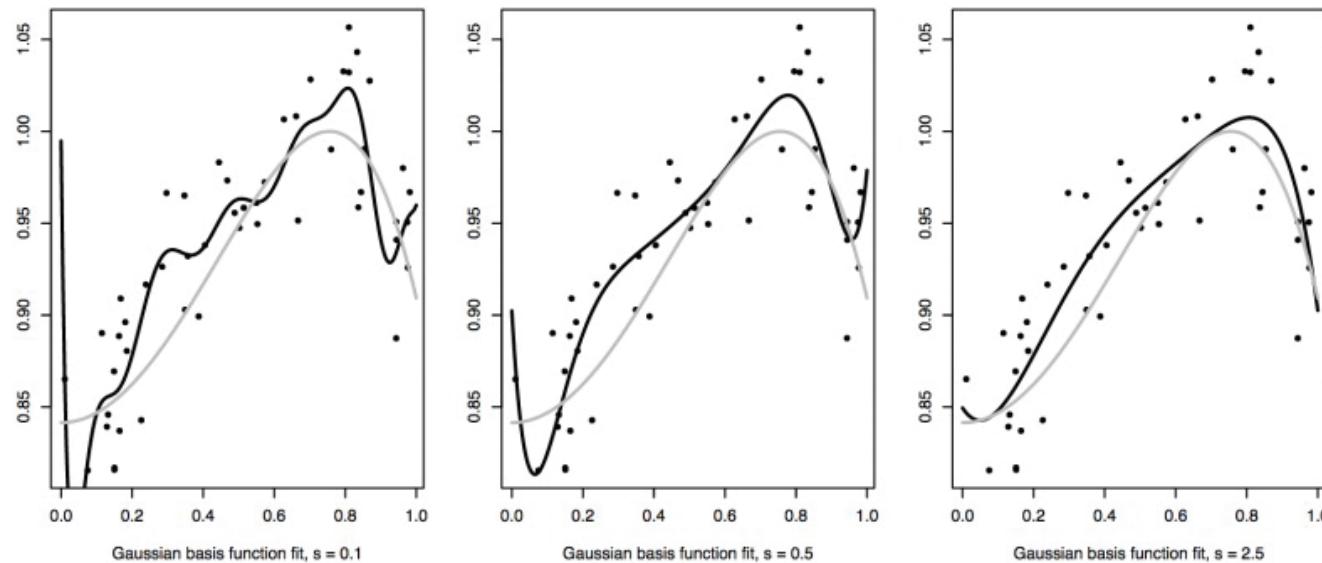
$$\phi_j(x) = \exp\left(\frac{(x - \mu_j)^2}{2\sigma^2}\right)$$



# Other Basis Functions

- Gaussian Radial Basis function:
  - $\mu_j$  govern the locations of the basis functions
  - $\sigma$  governs the spatial scale
    - Could be chosen from the data set e.g., average variance

$$\phi_j(x) = \exp\left(\frac{(x - \mu_j)^2}{2\sigma^2}\right)$$





# Other Basis Functions

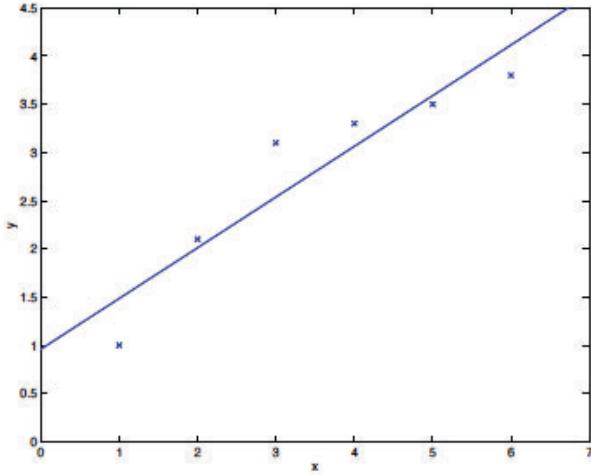
- Sigmoid:  $\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$
- Fourier
  - Expansion in sinusoidal functions
- Signal Processing
  - Also called *wavelets*
  - Functions grounded in time and frequency
- Linear
  - $\phi(x) = x$



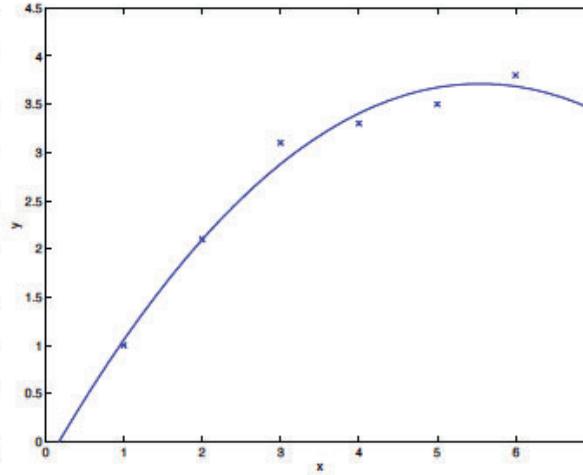
# Does more features always mean better?



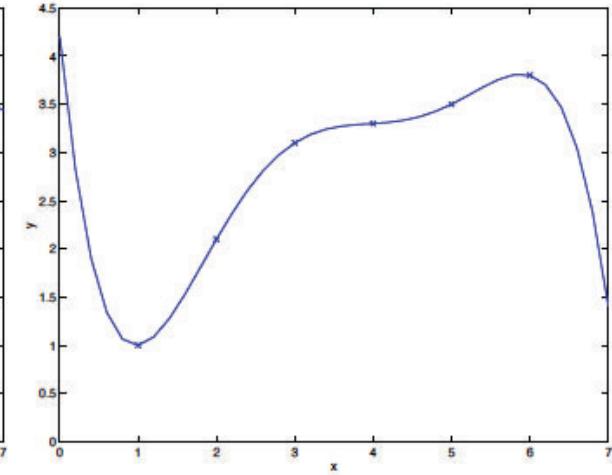
# Does more features always mean better?



Linear Basis Function



2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature

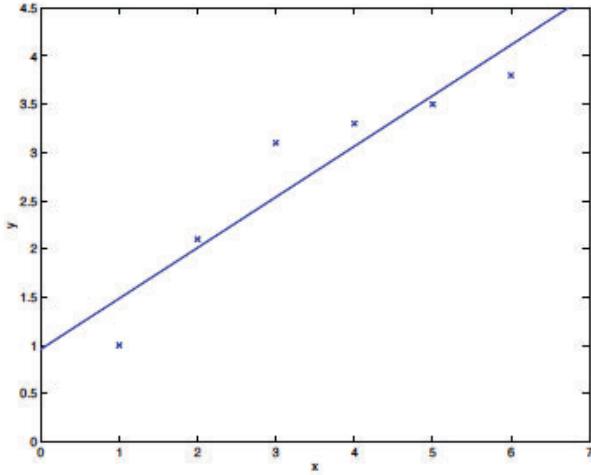


5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

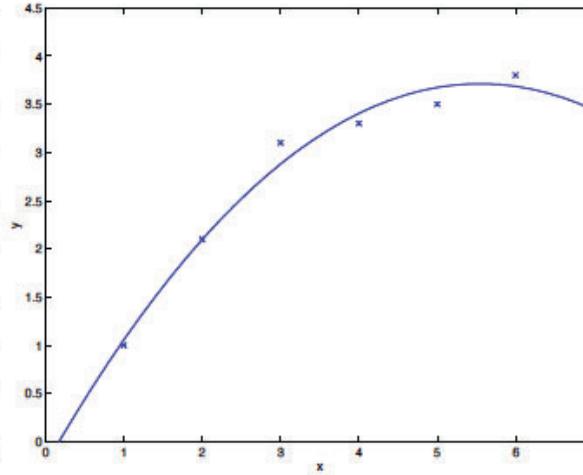
$$y = \sum_{j=0}^5 \theta_j x^j$$



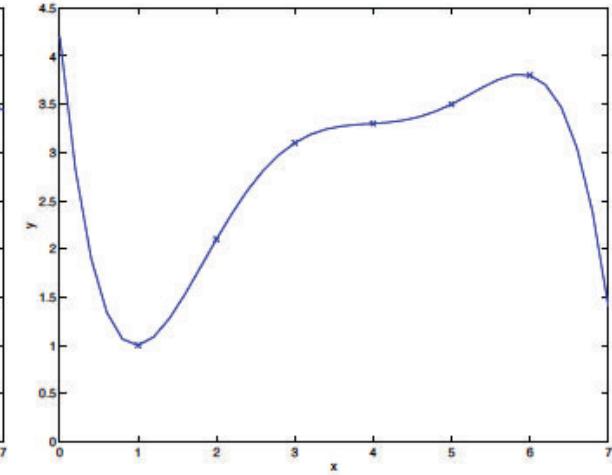
# Does more features always mean better?



Linear Basis Function



2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



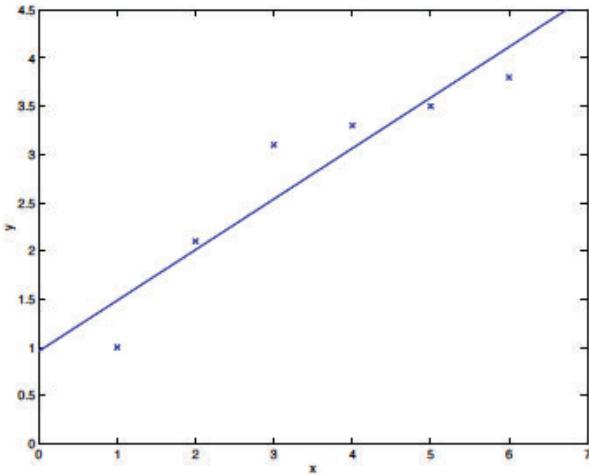
5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

$$y = \sum_{j=0}^5 \theta_j x^j$$

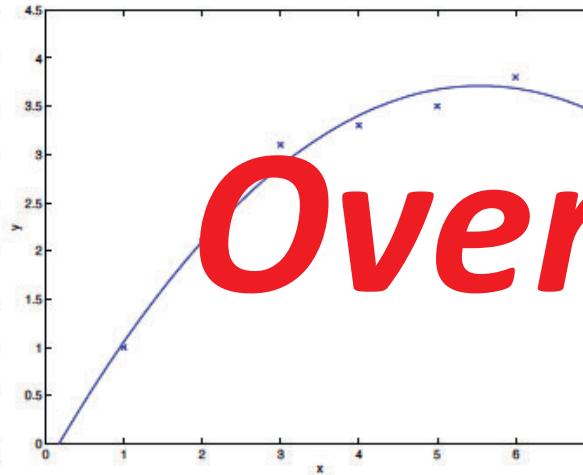
# NO!



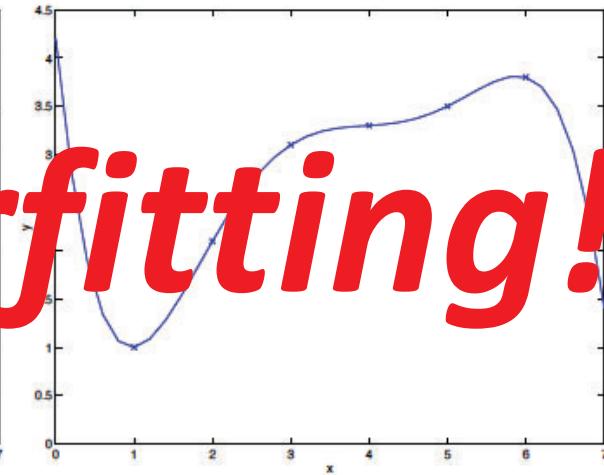
# Does more features always mean better?



Linear Basis Function



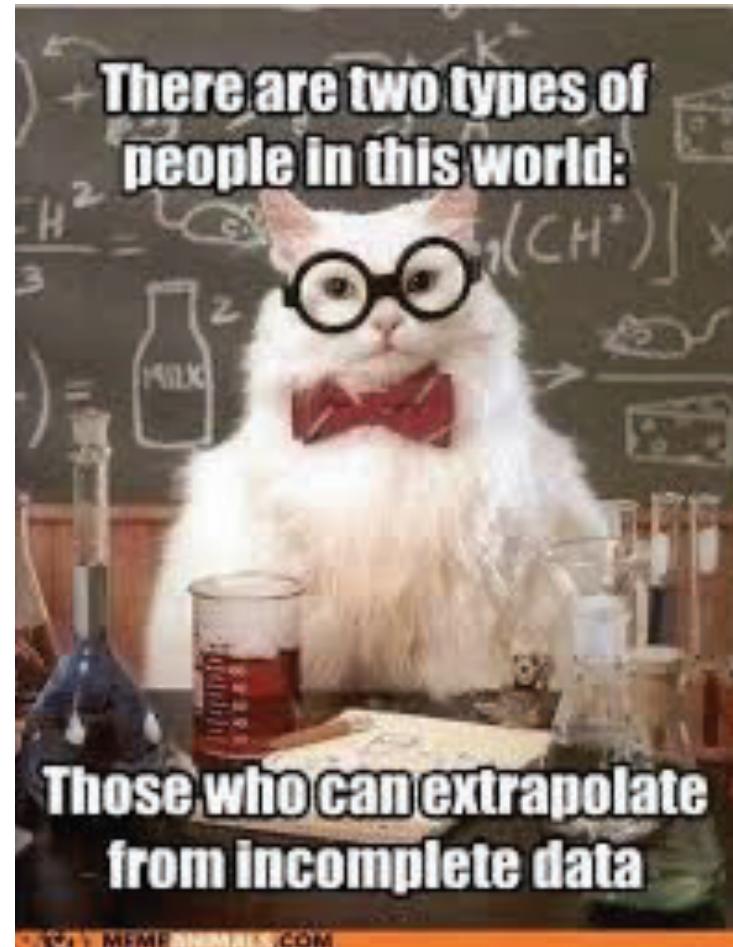
2<sup>nd</sup> order basis Function  
i.e. add  $x^2$  feature



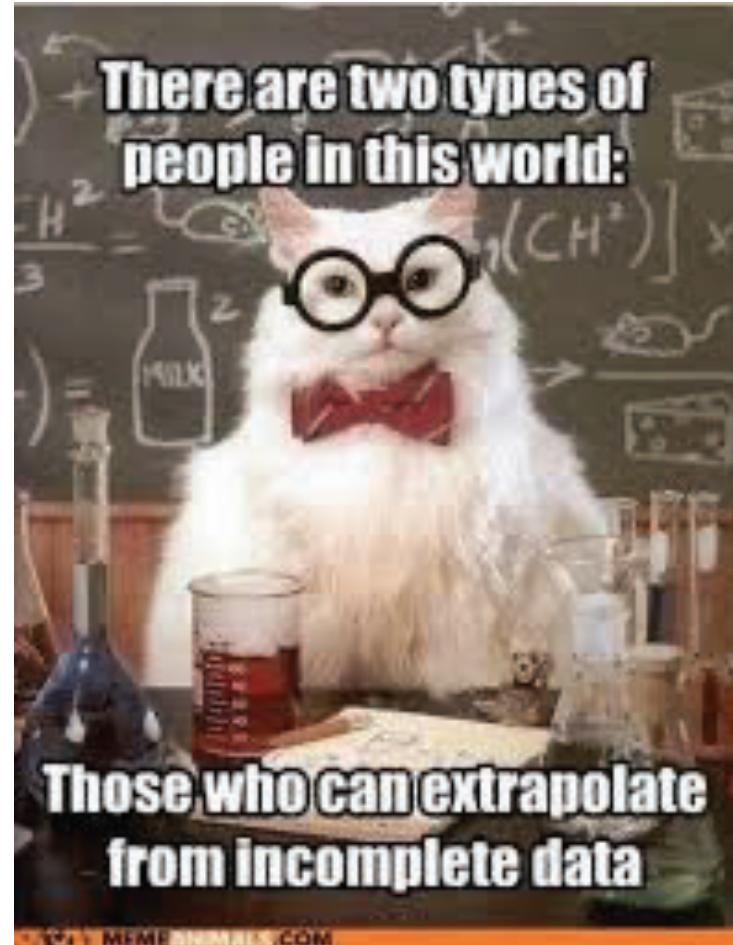
5<sup>th</sup> order basis Function  
i.e. add up to  $x^5$  feature

$$y = \sum_{j=0}^5 \theta_j x^j$$

# NO!



© Sathyanarayanan Aakur



*You don't want  
your model to  
be other kind!*



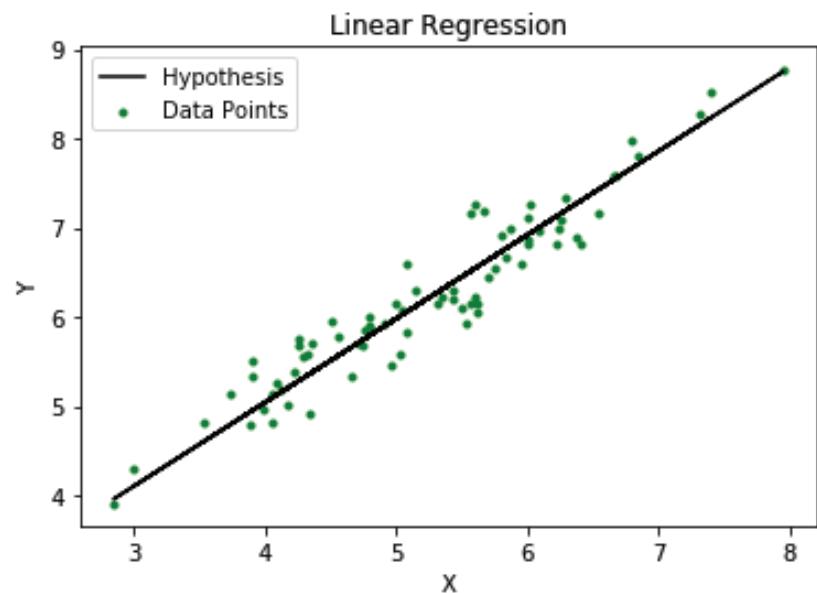


# Locally Weighted Linear Regression

Who needs parameters or learning, anyway?

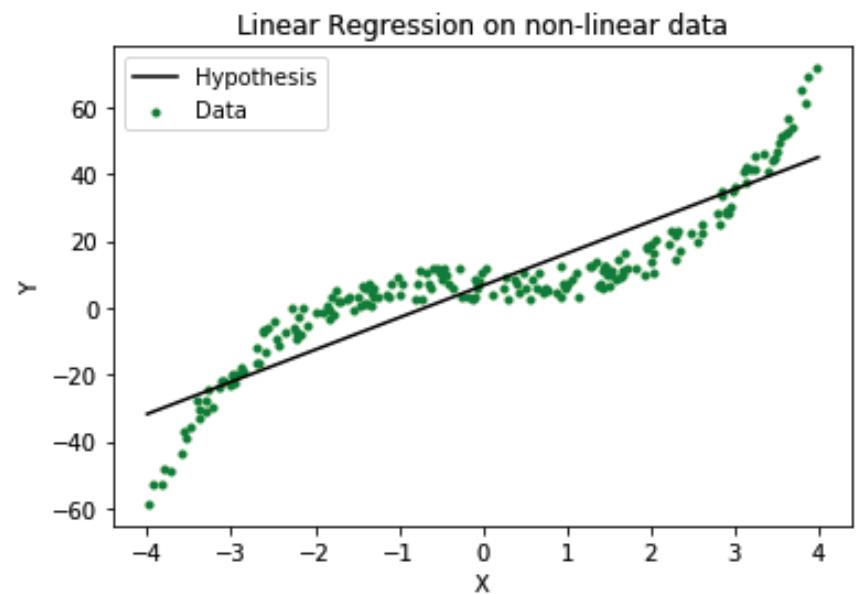
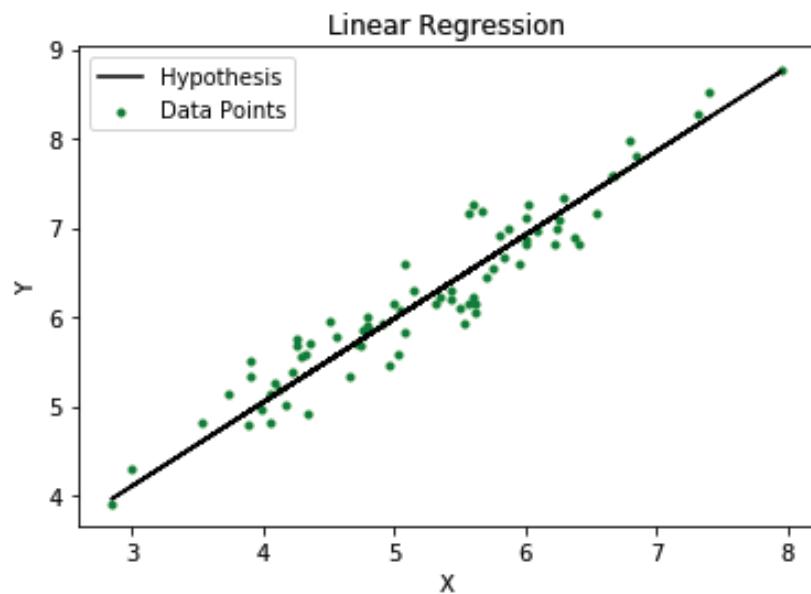


# Non-linear Data





# Non-linear Data





# Locally weighted LR

- Non-parametric algorithm
  - Linear regression is parametric i.e. once  $\theta$  is learned from training data, you can use it without re-using training data.



# Locally weighted LR

- Non-parametric algorithm
  - Linear regression is parametric i.e. once  $\theta$  is learned from training data, you can use it without re-using training data.
  - Non-parametric methods need the training data for making predictions at *each instance*.
    - The term “non-parametric” (roughly) refers to the fact that the amount of training data that is needed represent the hypothesis function  $h(x)$  grows linearly with the size of the training set.



# Locally weighted LR

- Non-parametric algorithm
  - Linear regression is parametric i.e. once  $\theta$  is learned from training data, you can use it without re-using training data.
  - Non-parametric methods need the training data for making predictions at *each instance*.
    - The term “non-parametric” (roughly) refers to the fact that the amount of training data that is needed represent the hypothesis function  $h(x)$  grows linearly with the size of the training set.
- So, how do we do it?



# Locally weighted LR

- Non-parametric algorithm
  - Linear regression is parametric i.e. once  $\theta$  is learned from training data, you can use it without re-using training data.
  - Non-parametric methods need the training data for making predictions at *each instance*.
    - The term “non-parametric” (roughly) refers to the fact that the amount of training data that is needed represent the hypothesis function  $h(x)$  grows linearly with the size of the training set.
- So, how do we do it?
  - Learn  $\theta$  that minimizes the objective function:

$$\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$$



# Locally weighted LR

- Non-parametric algorithm
  - Linear regression is parametric i.e. once  $\theta$  is learned from training data, you can use it without re-using training data.
  - Non-parametric methods need the training data for making predictions at *each instance*.
    - The term “non-parametric” (roughly) refers to the fact that the amount of training data that is needed represent the hypothesis function  $h(x)$  grows linearly with the size of the training set.
- So, how do we do it?
  - Learn  $\theta$  that minimizes the objective function:

Weights for  
each example  
in training set

$$\sum_i w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2$$

© Sathyanarayanan Aakur



# Locally weighted LR

- How do we set the value of  $w$ ?



# Locally weighted LR

- How do we set the value of w?

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$



# Locally weighted LR

- How do we set the value of w?

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

$\tau$  is called the **bandwidth** parameter.

- If  $x^{(i)}$  is closer to the query value  $x$ , then w is almost 1. If it is farther away, then w will be small.



# Example

- Input value:  $x = 5.0$
- Two values in training set:  $x_1=4.9, x_2=3.0$
- What would the values of  $w_1$  and  $w_2$  be, if  $\tau = 0.5$ ?



# Example

- Input value:  $x = 5.0$
- Two values in training set:  $x_1=4.9, x_2=3.0$
- What would the values of  $w_1$  and  $w_2$  be, if  $\tau = 0.5$ ?

$$w^{(1)} = \exp\left(\frac{-(4.9-5.0)^2}{2(0.5)^2}\right) = 0.9802$$



# Example

- Input value:  $x = 5.0$
- Two values in training set:  $x_1=4.9, x_2=3.0$
- What would the values of  $w_1$  and  $w_2$  be, if  $\tau = 0.5$ ?

$$w^{(1)} = \exp\left(\frac{-(4.9-5.0)^2}{2(0.5)^2}\right) = 0.9802$$

$$w^{(2)} = \exp\left(\frac{-(3.0-5.0)^2}{2(0.5)^2}\right) = 0.000335$$



# Example

- Input value:  $x = 5.0$
- Two values in training set:  $x_1=4.9, x_2=3.5$
- What would the values of  $w_1$  and  $w_2$  be, if  $\tau = 0.5$ ?

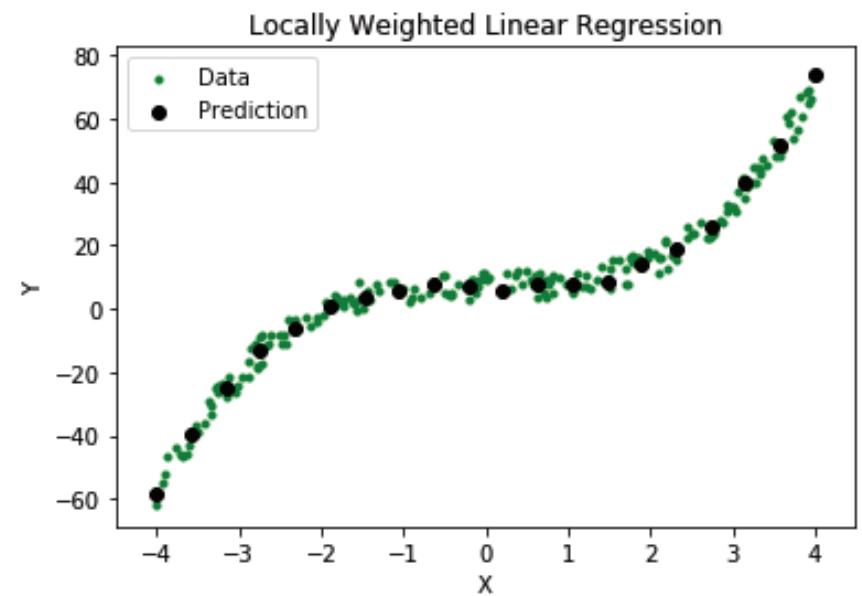
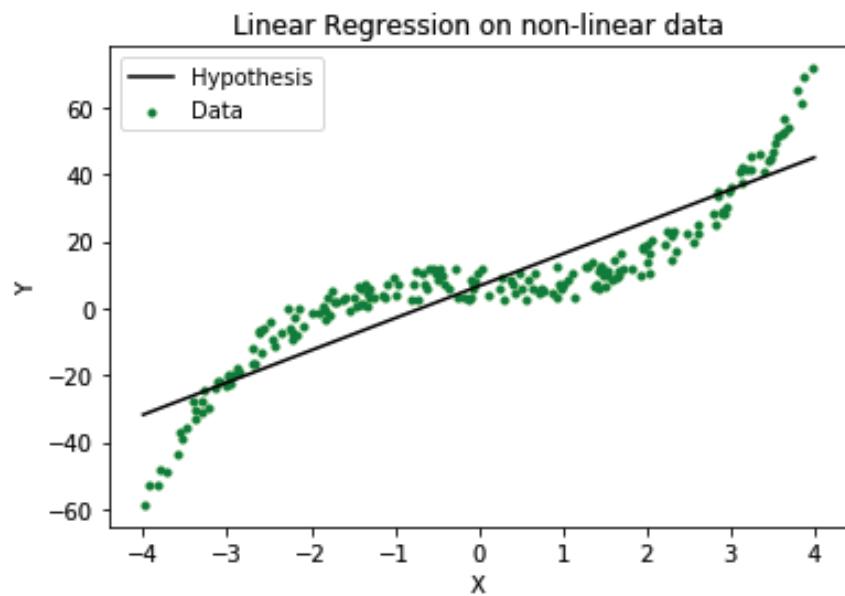
$$w^{(1)} = \exp\left(\frac{-(4.9-5.0)^2}{2(0.5)^2}\right) = 0.9802$$

$$w^{(2)} = \exp\left(\frac{-(3.0-5.0)^2}{2(0.5)^2}\right) = 0.000335$$

$$J(\theta) = 0.9802 * (\theta^T x^{(1)} - y^{(1)}) + 0.000335 * (\theta^T x^{(2)} - y^{(2)})$$



# Non-linear Data





# How to learn $\theta$ ?

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^m w^i (\theta^T x^i - y^i)^2 \\ &= \frac{1}{2} \sum_{i=1}^m (\theta^T x^i - y^i) w^i (\theta^T x^i - y^i) \\ &= \frac{1}{2} (X\theta - y)^T W (X\theta - y) \end{aligned}$$



# How to learn $\theta$ ?

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^m w^i (\theta^T x^i - y^i)^2 \\ &= \frac{1}{2} \sum_{i=1}^m (\theta^T x^i - y^i) w^i (\theta^T x^i - y^i) \\ &= \frac{1}{2} (X\theta - y)^T W (X\theta - y) \end{aligned}$$

Hence the gradient is given by  $\nabla_{\theta} J = X^T W (X\theta - Y)$



# How to learn $\theta$ ?

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^m w^i (\theta^T x^i - y^i)^2 \\ &= \frac{1}{2} \sum_{i=1}^m (\theta^T x^i - y^i) w^i (\theta^T x^i - y^i) \\ &= \frac{1}{2} (X\theta - y)^T W (X\theta - y) \end{aligned}$$

Hence the gradient is given by  $\nabla_{\theta} J = X^T W (X\theta - Y)$

Set the gradient to 0 to get the normal equations

$$\begin{aligned} \nabla_{\theta} J &= 0 \\ X^T W (X\theta - Y) &= 0 \\ X^T W X \theta &= X^T W Y \\ \theta &= (X^T W X)^{-1} X^T W Y \end{aligned}$$



# Practical Implementation

- Use Normal Equations approach
- Compute Weight Matrix
  - Diagonal matrix
  - Each element in the diagonal is the weight for that point given by

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

$$\theta = (X^T W X)^{-1} X^T W Y$$

- Have to compute Theta for every point you want to predict.
  - So to predict 100 points → Compute theta for that point, and then predict using

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$



# Do Non-Parametric models overfit?





# COMP [56]630– Machine Learning

Lecture 4 – Regularized LR, Bias-Variance Tradeoff



# Regularized Least Squares



# Regularized Least Squares

- How do you prevent overfitting?



# Regularized Least Squares

- How do you prevent overfitting?
- ***Regularization!***
- As model complexity increases, e.g., degree of polynomial or no. of basis functions, then it is likely that we overfit



# Regularized Least Squares

- How do you prevent overfitting?
- ***Regularization!***
- As model complexity increases, e.g., degree of polynomial or no. of basis functions, then it is likely that we overfit
- One way to control overfitting is not to limit complexity but to add a regularization term to the error function  $E_D$

$$E(w) = E_D(w) + \lambda E_W(w)$$

- where  $\lambda$  is the regularization coefficient that controls relative importance of data-dependent error  $E_D(w)$  and regularization term  $E_W(w)$



# Regularized Least Squares

- Regularized least squares is  $E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$
- Simple form of regularization term is  $E_W(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$
- Hence total error is given by 
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(x_n) \right\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$
- This regularization function is called *weight decay*
  - Weight values decay towards zero unless supported by training data examples



# Regularized Least Squares

- Error function with weight decay (quadratic) regularizer is

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(x_n) \right\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$



# Regularized Least Squares

- Error function with weight decay (quadratic) regularizer is

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \Phi(x_n) \right\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

- Its exact minimizer can be found in closed form and is given by

$$\mathbf{w} = (\lambda \mathbf{I} + \Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$



# Regularized Least Squares

- Error function with weight decay (quadratic) regularizer is

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \Phi(x_n) \right\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

- Its exact minimizer can be found in closed form and is given by

$$\mathbf{w} = (\lambda \mathbf{I} + \Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$

*Simple extension of ordinary least squares solution*

$$\mathbf{w}_{ML} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{t}$$



# A General Regularizer

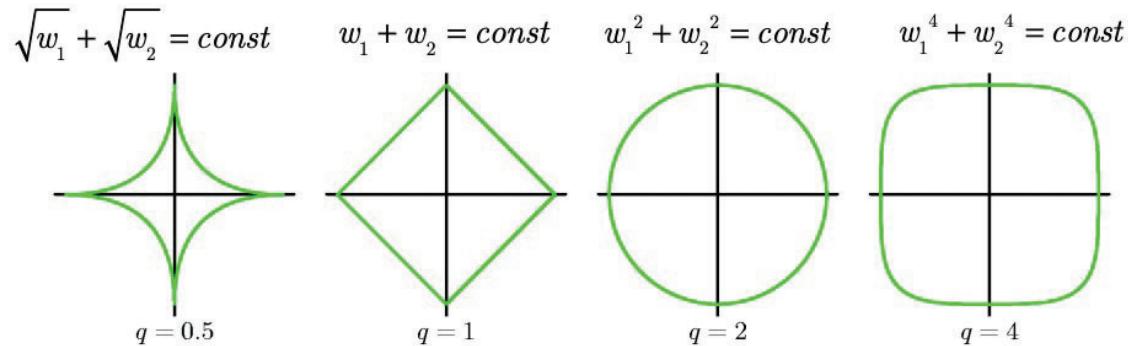
$$\frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$

- q=2 corresponds to the quadratic regularizer
- q=1 is known as lasso
  - Lasso has the property that if  $\lambda$  is sufficiently large some of the coefficients  $w_j$  are driven to zero
    - Leads to a sparse model in which the corresponding basis functions play no role



# A General Regularizer

$$\frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$





# Summary

- Regularization allows
  - complex models to be trained on small data sets
  - without severe over-fitting
- It limits model complexity
  - i.e., how many basis functions to use?
- Problem of limiting complexity is shifted to
  - one of determining suitable value of regularization coefficient



# Bias-Variance Tradeoff

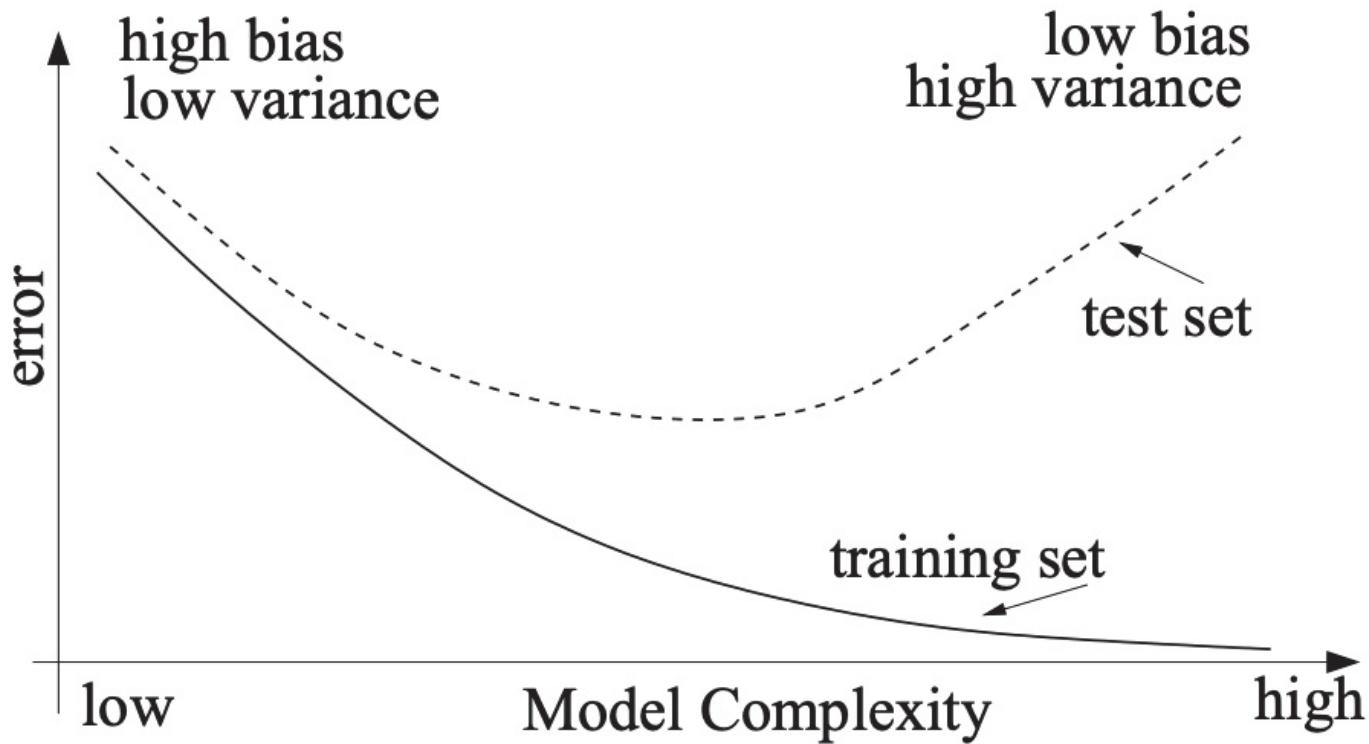


# Model Complexity and Overfitting

- We looked at linear regression where form of basis functions  $\varphi$  and number of functions  $M$  are fixed
- Using maximum likelihood (equivalently least squares) leads to severe overfitting if complex models are trained with limited data
  - However limiting  $M$  to avoid overfitting has side effect of not capturing important trends in the data
- Regularization can control overfitting for models with many parameters
  - But seeking to minimize wrt both  $w$  and  $\lambda$  leads to unregularized solution with  $\lambda=0$



### TYPICAL BEHAVIOUR





# Some Definitions

- The *generalization* of a machine learning is the performance (classification, regression, density estimation) on test data, not used for training, but drawn from the same (joint) distribution as the training data. Often, our real goal is to get good generalization.
- When our model is too complex for the amount of training data we have, it memorizes parts of the noise as well as learning the true problem structure. This is called *overfitting or model variance*.
- When our model is not complex enough, it cannot capture the structure in our data, no matter how much data we give it. This is called *underfitting or model bias*.
- In statistics, there is an incomprehensible obsession with the minimum variance unbiased estimator. But if all we are about is generalization then we should be happy to introduce a little bit of bias if it reduces the variance a lot.



# What is Bias and Variance?

- The ***bias error*** is an error from ***erroneous assumptions*** in the learning algorithm. ***High bias*** can cause an algorithm to miss the relevant relations between features and target outputs (***underfitting***).
- The ***variance*** is an error from ***sensitivity to small fluctuations*** in the training set. ***High variance*** can cause an algorithm to ***model*** the ***random noise*** in the training data, rather than the intended outputs (***overfitting***).



# What are we trying to do?

- What basic problems are we trying to avoid with model selection?
- *Overfitting*: if we chose a model that is too complex, it will overfit to the noise in our training set. Another way of saying this is that the machine we end up with is very sensitive to the particular training sample we use. The model has a lot of variance across training samples of a fixed size.
- *Underfitting*: if we chose a model that is not complex enough, it cannot fit the true structure, and so no matter what training sample we use there is some error between the true function and our model approximation. The model has a lot of bias.
- Intuitively, we need the right balance. How can we formalize this?



# Example

- Let us consider a supervised learning setup (scalar for now), with random noise (uncorrelated to inputs/outputs) and squared error:

$$y = g(x) + \text{noise}$$

$$y' = f(x)$$

$$\text{error} = (y - y')^2$$

- Consider the expected error at a single test point  $x_0$ , averaged over all possible training sets of size  $N$ , drawn from the joint distribution over inputs and outputs  $p(x, y) = p(x)p(y|x)$ .

$$\begin{aligned} e(x_0) &= \langle (y_0 - \hat{y}_0)^2 \rangle \\ &= \langle (g(x_0) + \epsilon_0 - \hat{y}_0)^2 \rangle \\ &= \langle \epsilon_0^2 \rangle + (\langle f(x_0) \rangle - g(x_0))^2 + \langle f(x_0) - \langle f(x_0) \rangle \rangle^2 \\ &= \sigma^2 + (\text{mean}[f(x_0)] - g(x_0))^2 + \text{var}[f(x_0)] \\ &= \text{Unavoidable Error} + \text{Bias}^2 + \text{Variance} \end{aligned}$$



## MODEL SELECTION & PERFORMANCE ESTIMATION

---

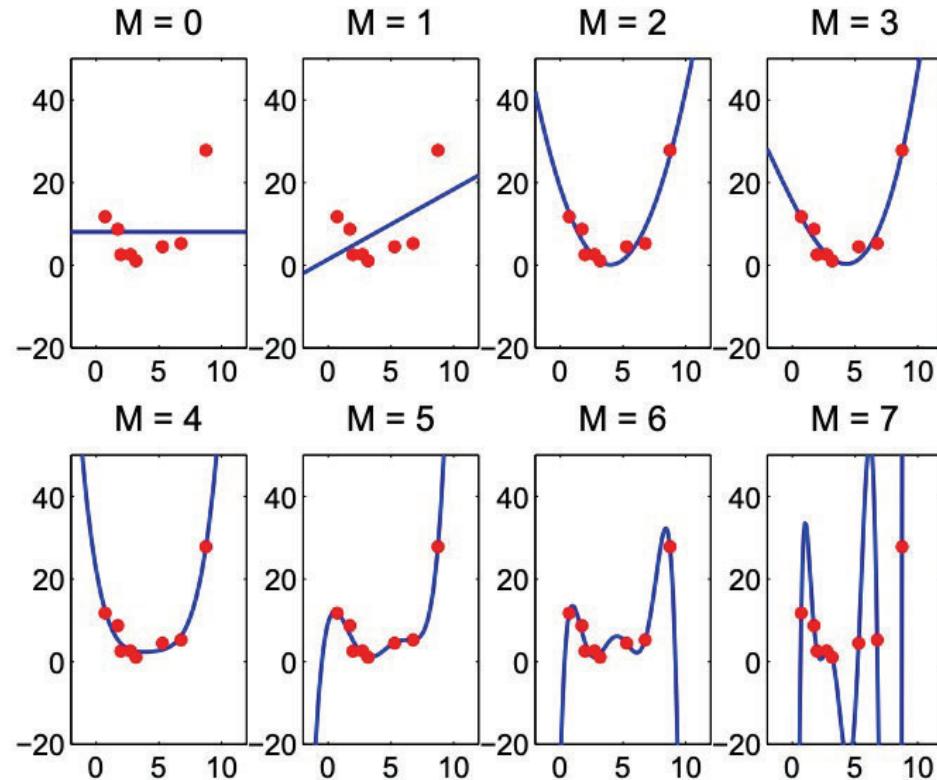
- Model Selection: out of a set of models (or continuum of model complexity), choose the model which will perform the best on future test data.
- Model Assessment: for the selected model, estimate its generalization error on new data.
- If we have lots of data, these two problems can be solved by dividing our data into 3 parts:
  - Training Data – used to train each model
  - Validation Data – used to measure performance of each trained model in order to select the best model
  - Assessment Data – used only once, on the final selected model, to estimate performance on future test data
- Typical split is 60% training, 20% validation, 20% assessment.  
So that's it, are we done?



---

### EXAMPLE: POLYNOMIAL FITTING

---





## APPROXIMATIONS WHEN DATA IS LIMITED

---

- Often, we don't have enough data to make 3 separate and reasonably sized training, validation and assessment sets.
- If we don't have very much data, we can try to *approximate* the results of validation and assessment.
- Two basic approaches for finite datasets:
  - Analytic methods: derive algebraic expressions which try to approximate the test error, e.g. using a complexity penalty which scales as the ratio between the number of parameters in the model and the number of training cases.  
Examples: BIC, AIC, MDL, VC-dimension.
  - Sample-recycling methods: try to estimate the test error computationally, using the same data that we trained on.  
Examples: jackknife, cross-validation, bootstrap.



## REGULARIZATION AND CAPACITY CONTROL

---

- How can we improve generalization? Reduce either bias or variance!
- One obvious way: use more training data, and commensurately more complex models. If we scale up model complexity slowly enough, using more data reduces *both* bias and variance.
- But what if we can't get more data?  
Our goal should be to reduce variance (by using simpler models) while not increasing our bias too much (by not using *too* simple a model). We should not force ourselves to use unbiased ( $\text{Bias}=0$ ) models, because we only really care about the sum  $\text{Bias}^2 + \text{Variance}$ .
- We need a knob to control this tradeoff (e.g. by discretely constraining model *structure* or by continuously *regularizing* model complexity or smoothness) and a way to set the knob (i.e. decide on the right tradeoff balance).



## ANOTHER REGULARIZER: PARAMETER SHARING

---

- Another way to control model complexity is to *tie together* or *share* various parameters. This allows us to have a complete model structure but not have to estimate a huge number of free parameters.
- This is used in mixtures of factor analyzers, to jointly estimate the sensor noises, in mixtures of Gaussians to jointly estimate cluster covariances (e.g. Fisher's discriminant is a class-conditional Gaussian model with shared covariances), in vision neural networks to learn translation-invariant receptive fields, etc.



## MORE REGULARIZATION: METHOD WITH LOCAL SUPPORT

---

- Yet another way to control model complexity is to restrict the amount of training data that can be used to predict the output on any new test case.
- Each test case prediction is only allowed to use a small fraction of the training data, typically the training points whose inputs are close to the input of the test case.
- This is known as a *locally weighted* method, e.g. nearest neighbour classification, Parzen density estimation, locally weighted regression.
- Local methods are related to “semi-parametric” models, which try to use the reservoir of training data to store most of the bits of their capacity, and only have a few “metaparameters” which control how that reservoir is used at test time.  
Examples: K-NN classifier, locally-weighted regression, parzen window density estimators



## REGULARIZATION BY ADDING A PENALTY TERM

---

- Instead of discrete complexity controls, it is often useful to have a continuous range of complexity, set by one or more real valued “fudge-factors” or “hyperparameters”.
- The most common way to achieve this is to add a “penalty term” to the cost function (error, log likelihood, etc) which measures in a quantitative and continuous way how complex/simple our model is:

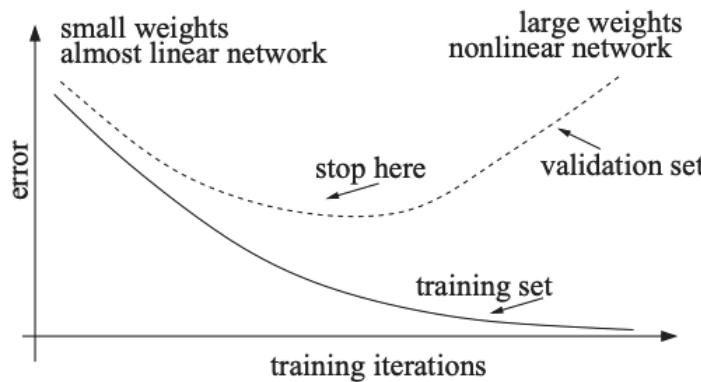
$$\text{cost}(\theta) = \text{error}(\text{data}, \theta) + \lambda \text{penalty}(\theta)$$

- We can then weight this penalty term relative to the original error (or likelihood) and minimize the resulting penalized cost.
- The larger the penalty weight  $\lambda$ , the simpler our model will be.
- How can we set  $\lambda$ ? If we have lots of data, we can use the performance on a held out set of validation examples to determine the correct value of the penalty weight.



## EXAMPLE PENALTY: EARLY STOPPING

- Another approach to regularization in models whose complexity grows with training time is to stop training early.
- This works quite well in neural networks, since small weights mean that the network is mostly linear (low complexity) and it takes a while for the weights to get bigger, giving nonlinear networks (high complexity). Essentially a penalty equal to # training iterations.
- A validation set can be used to detect stopping point.





## EXAMPLE PENALTY: WEIGHT DECAY

---

- The most common regularization is the ridge regression penalty (weight decay) which discourages large parameter values in generalized linear models:

$$\text{cost}(\theta) = \text{error}(\text{data}, \theta) + \lambda \sum_k \theta_k^2$$

- This says: “don’t use big weights unless they really help to reduce your error a lot”. Otherwise, there is nothing to stop the model from using enormous positive and negative weights to gain a tiny benefit in error.
- Remember: on a finite training set, there will always be some tiny, accidental correlation between the noise in the inputs and the target values.



## PRACTICAL SOLUTIONS

---

- Several simple ways to good generalization in practice.
- Use model classes with flexible control over their complexity.  
(e.g. ridge regression, mixture models)
- Employ regularization (capacity control) and (cross) validation,to match model complexity with the amount of data available.
- Build in as much reliable prior knowledge as possible, so algorithms don't have to waste data learning things we already know.
- Use cross-validation/bootstrap to make efficient use of limited data.
- Use subsampling or sparse methods to speed up algorithms on huge training sets, and keep them fast and small at test time.



## POTENTIAL PITFALLS

---

- Several things can cause us trouble when we are trying to get good generalization from a learning algorithm:
  - we might not have enough training data to learn target concept
  - our testing might not *really* be from the same distribution as our training data
  - our model might not be complex enough, so it underfits
  - our model might be too complex, so it overfits
  - we have too much training data to run the algorithm in a reasonable amount of time or memory
- Sounds hopeless!  
What can we do?



# Bias-Variance vs Bayesian

- Bias-Variance decomposition provides insight into model complexity issue
- Limited practical value since it is based on ensembles of data sets
  - In practice there is only a single observed data set
  - If there are many training samples then combine them
    - which would reduce over-fitting for a given model complexity
- Bayesian approach gives useful insights into over-fitting and is also practical



# Bayesian Model Comparison



# Remember: Curve Fitting

- Regression using basis functions and MSE:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right\}^2$$

- Need an M that gives best generalization
  - M = No. of free parameters in model or model complexity
- With regularized least squares
  - $\lambda$  also controls model complexity (and hence degree of over-fitting)

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$



# Which model to use?

- There can be many models that you can use!
- Depending on basis functions, number of functions, etc...





# Which model to use?

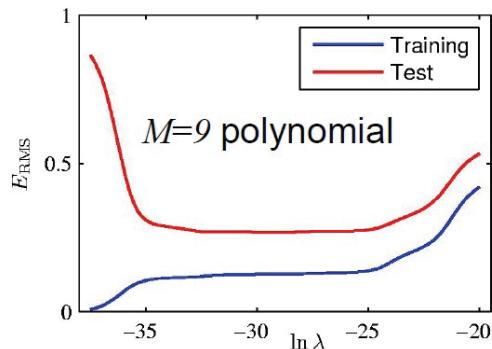
- There can be many models that you can use!
- Depending on basis functions, number of functions, etc...





# Choosing a model using data

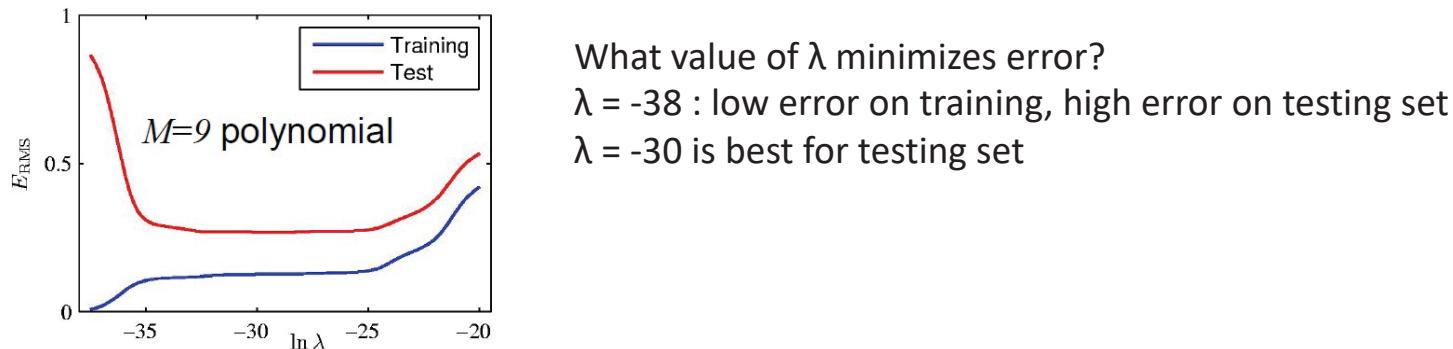
- $\lambda$  controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or  $\lambda$ )
  - Validation set (holdout)
    - to optimize model complexity (M or  $\lambda$ )





# Choosing a model using data

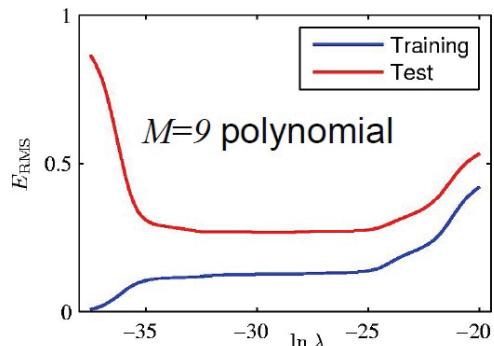
- $\lambda$  controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or  $\lambda$ )
  - Validation set (holdout)
    - to optimize model complexity (M or  $\lambda$ )





# Choosing a model using data

- $\lambda$  controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or  $\lambda$ )
  - Validation set (holdout)
    - to optimize model complexity (M or  $\lambda$ )



What value of  $\lambda$  minimizes error?

$\lambda = -38$  : low error on training, high error on testing set

$\lambda = -30$  is best for testing set

$$E_{RMS} = \sqrt{2E(w^*) / N}$$

Division by N allows different data sizes to be compared since E is a sum over N Sqrt (of squared error) measures on same scale as t



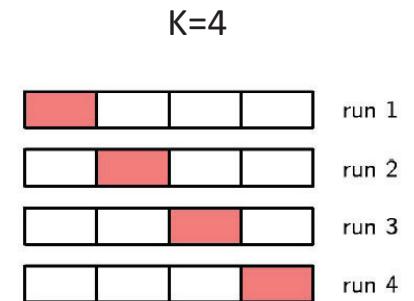
# Use the Validation Set!

- Performance on training set is not a good indicator of predictive performance
- If there is plenty of data,
  - use some of the data to train a range of models Or a given model with a range of values for its parameters
  - Compare them on an independent set, called validation set
  - Select one having best predictive performance
- If data set is small then some over-fitting can occur and it is necessary to keep aside a test set



# K-fold Cross Validation

- Supply of data is limited
- All available data is partitioned into K groups
- $K-1$  groups are used to train and evaluated on remaining group
- Repeat for all K choices of held-out group
- Performance scores from K runs are averaged



If  $K=N$ , then it is called the  
leave-one-out cross  
validation



# Disadvantage of Cross-Validation

- No. of training runs is increased by factor of K
- Problematic if training itself is expensive
- Different data sets can yield different complexity parameters for a single model
  - E.g., for given M several values of  $\lambda$
- Combinations of parameters is exponential
- Need a better approach
  - Ideally one that depends only on a single run with training data and should allow multiple hyperparameters and models to be compared in a single run



## POTENTIAL PROBLEMS WITH CROSS-VALIDATION

---

- CV is awesome and it can be used on clustering, density estimation, classification, regression, etc.
- But intensive use of cross-validation can overfit, if you explore too many models, by finding a model that accidentally predicts the whole training set well (and thus every leave-one-out sample well).
- CV can also be very time consuming if done naively.
- Often there are efficient tricks for computing all possible leave-one-out cross validation folds, which can save you a lot of work over brute-force retraining on all  $N$  possible LOO datasets.
- For example, in linear regression, the term  $(\sum_{n \neq \ell} \mathbf{x}_n \mathbf{x}_n^\top)^{-1}$  which leaves out datapoint  $\ell$  can be computed using the matrix inversion lemma:  $(\sum_n \mathbf{x}_n \mathbf{x}_n^\top - \mathbf{x}_\ell \mathbf{x}_\ell^\top)^{-1}$ .
- This is also true of the Generalized Cross Validation (GCV) estimate of Golub and Wahaba. (see extra readings)



## MODEL AVERAGING

---

- One last way to reduce variance, while not affecting bias too severely, is to average together the predictions of a bunch of different models.
- These models must be different in some way, either because they were trained on different subsets of the data, or with different regularization parameters, different local optima, or something.
- When we average them together, we would like to weight more strongly the models we believe are fitting the data better.
- Such systems are often called *committee machines*.
- Really, this is just a weak form of Bayesian learning.  
MAP = estimate of mode of posterior over models  
Bagging (next class) = estimate of mean of posterior over models  
BIC/AIC = estimates of correct predictive distribution



# What do we need?



# What do we need?

- Need to find a performance measure that depends only on the training data and which does not suffer from bias due to over-fitting
- Historically various information criteria have been proposed that attempt to correct the for the bias of maximum likelihood by the addition of a penalty term to compensate for the overfitting of more complex models



# Akaike Information Criterion (AIC)

- AIC chooses model that maximizes  $\ln p(D|\mathbf{w}_{\text{ML}}) - M$ 
  - First term is best-fit log-likelihood for given model
  - M is no of adjustable parameters in model
- Penalty term M is added for over-fitting using many parameters
- Bayesian Information Criterion (BIC) is a variant of this quantity
- Disadvantages:
  - need runs with each model, prefers overly simple models

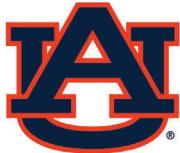


# Bayesian Perspective

- Avoids over-fitting
  - By marginalizing over model parameters
    - sum over model parameters instead of point estimates
- Models compared directly over training data
  - No need for validation set
  - Allows use of all available data in training set
  - Avoids multiple training runs for each model associated with cross-validation
  - Allows multiple complexity parameters to be simultaneously determined during training
    - Relevance vector m/c is Bayesian model with one complexity parameter for every data point



# How do you choose a model?



# How do you choose a model?

- Based on its evidence!
- A model is a probability distribution over data D – E.g., a polynomial model is a distribution over target values  $t$  when input  $x$  is known, i.e.,  $p(t|x,D)$
- Uncertainty in model itself can be represented by probabilities
- Compare a set of models  $M_i$ ,  $i=1,..L$
- Given training set, wish to evaluate posterior

$$p(M_i|D) \propto p(M_i) p(D|M_i)$$

Prior expresses preference for different models  
We assume equal priors

Model Evidence (or Marginal Likelihood)



# Model Evidence

- $p(D | M_i)$  is preference shown by data for model
- Called Model evidence or *marginal likelihood*
  - because parameters have been *marginalized*



# Bayes Factor

- From model evidence  $p(D|M_i)$  for each model:
  - The ratio  $\frac{p(D|M_i)}{p(D|M_j)}$  is called the Bayes Factor
  - Shows the preference for Model  $i$  over Model  $j$



# Predictive Distribution

- Given  $p(M_i | D)$  the predictive distribution is given by

$$p(t | \mathbf{x}, D) = \sum_{i=1}^L \underbrace{p(t | \mathbf{x}, M_i, D)}_{\text{Prediction under model}} p(M_i | D)$$

- This is called a *mixture*:
  - predictions under different models are weighted by posterior probabilities of models
- Instead of all models  $M_i$ , approximate with single most probable model
  - Known as *Model Selection*



# Bayes factor always favors correct model

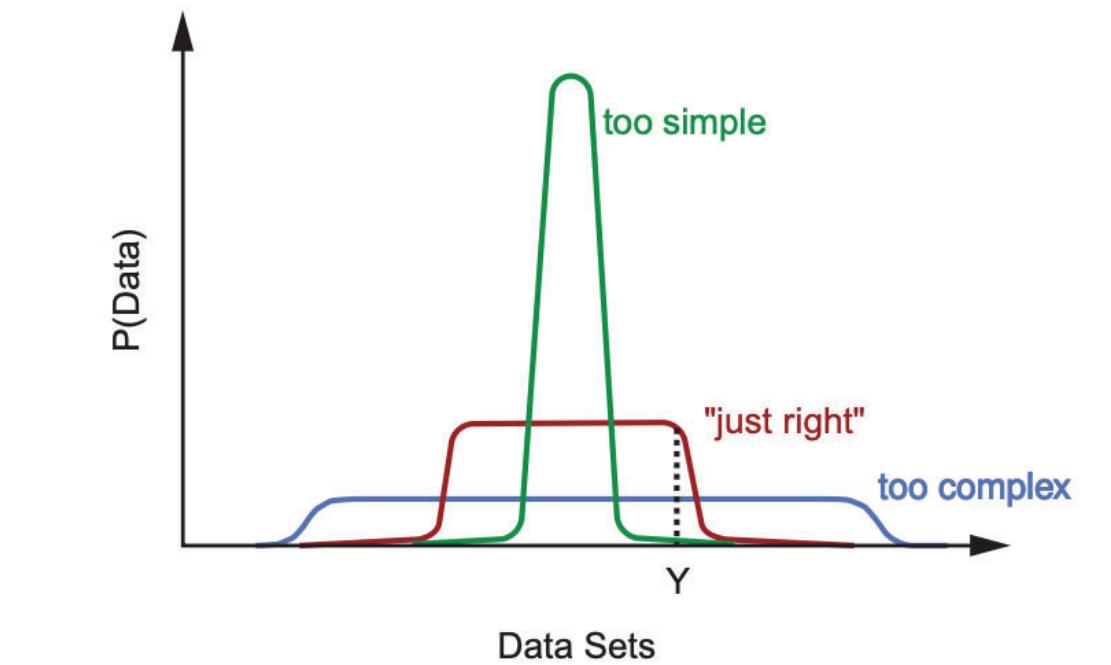
- We are assuming the true model is contained among the  $M_i$
- If this is so, Bayes model comparison will favor the correct model
- If  $M_1$  is the true model, and we average over the distribution of data sets, Bayes factor has the form

$$\int p(D | M_1) \ln \frac{p(D | M_1)}{p(D | M_2)} dD$$

- This is K-L Divergence which is always positive
  - Thus Bayes factor favors the correct model



## OCKHAM'S RAZOR



We want to use the simplest model which explains the data well.  
[A now famous figure, first introduced by Mackay.]



## NO FREE LUNCH

---

- David Wolpert and others have proven a series of theorems, known as the “no free lunch” theorems which, roughly speaking, say that *unless you make some assumptions* about the nature of the functions or densities you are modeling, no one learning algorithm can *a priori* be expected to do better than any other algorithm.
- In particular, this lack of clear advantage includes any algorithm and any meta-learning procedure applied to that algorithm. In fact, “anti-cross-validation” (i.e. picking the regularization parameters that give the *worst* performance on the CV samples) is *a priori* just as likely to do well as cross-validation. Without assumptions, random guessing is no worse than any other algorithm.
- So capacity control, regularization, validation tricks and meta-learning (next class) cannot *always* be successful.



## GENERALIZATION ERROR VS. LEARNING ERROR

---

- A key issue here is the difference between test error on a test set drawn from the same distribution as the training data (may contain duplicates) and *out of sample* test error.
- Remember back to the first class: learning binary functions.  
No assumptions == no generalization on out of sample cases.
- The only way to learn is to wait until you have seen the whole world and memorize it.
- Luckily, we *can* make some progress in real life.
- Why? Because the assumptions we make about function classes are often partly true.



# Summary

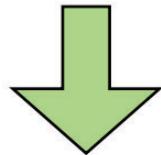
- Avoids problem of over-fitting
- Allows models to be compared using training data alone
- However needs to make assumptions about form of model
  - Model evidence can be sensitive to many aspects of prior such as behavior of tails
- In practice necessary to keep aside independent test data to evaluate performance



# 3 levels of inference

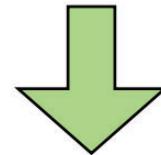
## LEVEL 1

I have selected a model  $M$  and prior  $P(\theta|M)$



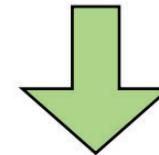
## LEVEL 2

Actually, there are several possible models:  $M_0, M_1, \dots$



## LEVEL 3

None of the models is clearly the best



### Parameter inference

What are the favourite values of the parameters?  
(assumes  $M$  is true)

### Model comparison

What is the relative plausibility of  $M_0, M_1, \dots$  in light of the data?

### Model averaging

What is the inference on the parameters accounting for model uncertainty?



# What is SOTA Research?

## META-LEARNING

---

- The idea of meta-learning is to come up with some procedure for taking a learning algorithm and a fixed training set, and somehow repeatedly applying the algorithm to *different* subsets (weightings) of the training set or using *different* random choices within the algorithm in order to get a large ensemble of machines.
- The machines in the ensemble are then *combined* in some way to define the final output of the learning algorithm (e.g. classifier)
- The hope of meta-learning is that it can “supercharge” a mediocre learning algorithm into an excellent learning algorithm, without the need for any new ideas!
- There is, as always, good news and bad news....
  - The Bad News: there is (quite technically) No Free Lunch.
  - The Good News: for many real world datasets, meta learning works well because its implicit assumptions are often reasonable.

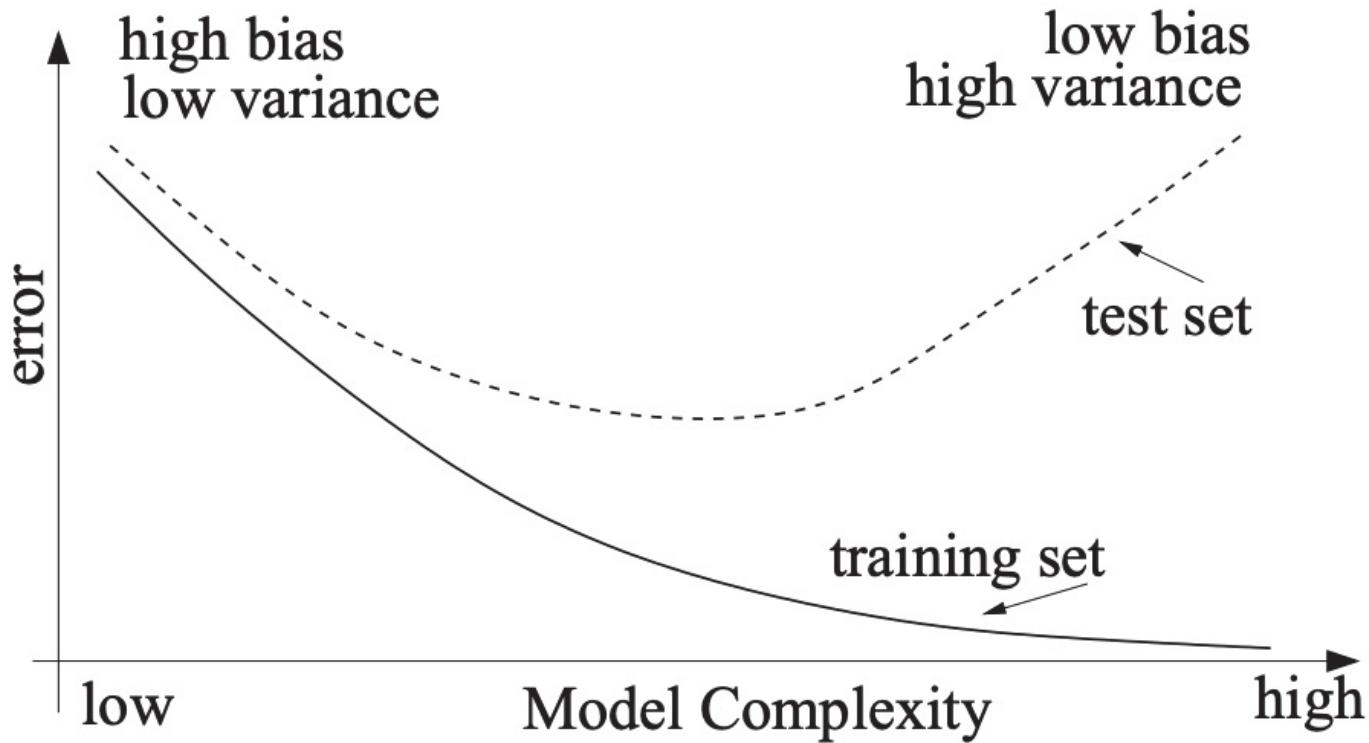


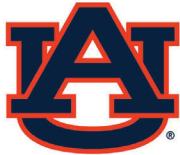
# COMP [56]630– Machine Learning

Lecture 6 – Bias-Variance Tradeoff, Model Selection, Logistic Regression  
Pt. 1



### TYPICAL BEHAVIOUR





# What is Bias and Variance?

- The ***bias error*** is an error from ***erroneous assumptions*** in the learning algorithm. ***High bias*** can cause an algorithm to miss the relevant relations between features and target outputs (***underfitting***).
- The ***variance*** is an error from ***sensitivity to small fluctuations*** in the training set. ***High variance*** can cause an algorithm to ***model*** the ***random noise*** in the training data, rather than the intended outputs (***overfitting***).



# Mitigation Techniques

- Model selection
  - Use model assessment to pick the best model
  - Use validation data if available. Otherwise, K-Fold Cross validation
- Add more training data
- Regularization
- Parameter sharing
- Locally weighted or non-parametric models to use less data
- Early stopping



## PRACTICAL SOLUTIONS

---

- Several simple ways to good generalization in practice.
- Use model classes with flexible control over their complexity.  
(e.g. ridge regression, mixture models)
- Employ regularization (capacity control) and (cross) validation,to match model complexity with the amount of data available.
- Build in as much reliable prior knowledge as possible, so algorithms don't have to waste data learning things we already know.
- Use cross-validation/bootstrap to make efficient use of limited data.
- Use subsampling or sparse methods to speed up algorithms on huge training sets, and keep them fast and small at test time.



## POTENTIAL PITFALLS

---

- Several things can cause us trouble when we are trying to get good generalization from a learning algorithm:
  - we might not have enough training data to learn target concept
  - our testing might not *really* be from the same distribution as our training data
  - our model might not be complex enough, so it underfits
  - our model might be too complex, so it overfits
  - we have too much training data to run the algorithm in a reasonable amount of time or memory
- Sounds hopeless!  
What can we do?



# Bias-Variance vs Bayesian

- Bias-Variance decomposition provides insight into model complexity issue
- Limited practical value since it is based on ensembles of data sets
  - In practice there is only a single observed data set
  - If there are many training samples then combine them
    - which would reduce over-fitting for a given model complexity
- Bayesian approach gives useful insights into over-fitting and is also practical



# Bayesian Model Comparison



# Remember: Curve Fitting

- Regression using basis functions and MSE:

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right\}^2$$

- Need an M that gives best generalization
  - M = No. of free parameters in model or model complexity
- With regularized least squares
  - $\lambda$  also controls model complexity (and hence degree of over-fitting)

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \left\{ t_n - \mathbf{w}^T \phi(\mathbf{x}_n) \right\}^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$



# Which model to use?

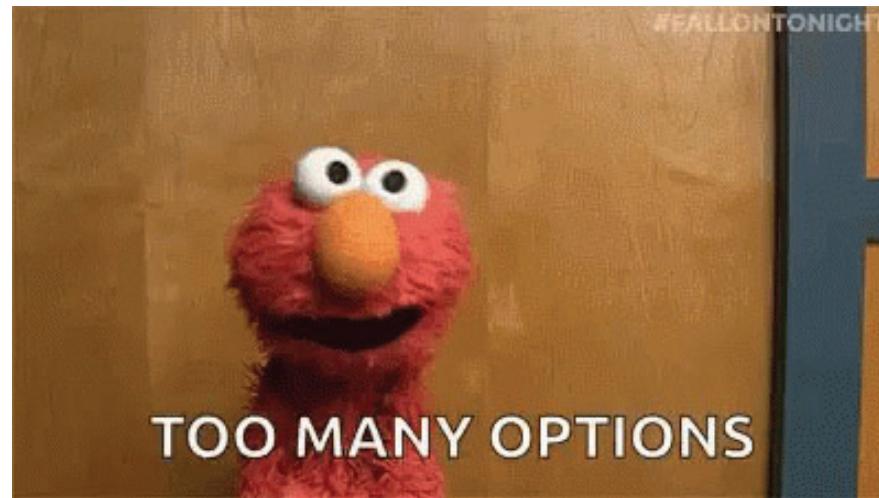
- There can be many models that you can use!
- Depending on basis functions, number of functions, etc...





# Which model to use?

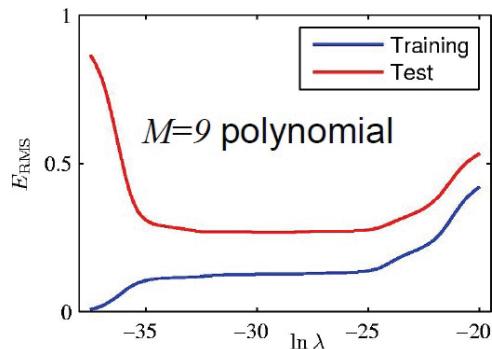
- There can be many models that you can use!
- Depending on basis functions, number of functions, etc...





# Choosing a model using data

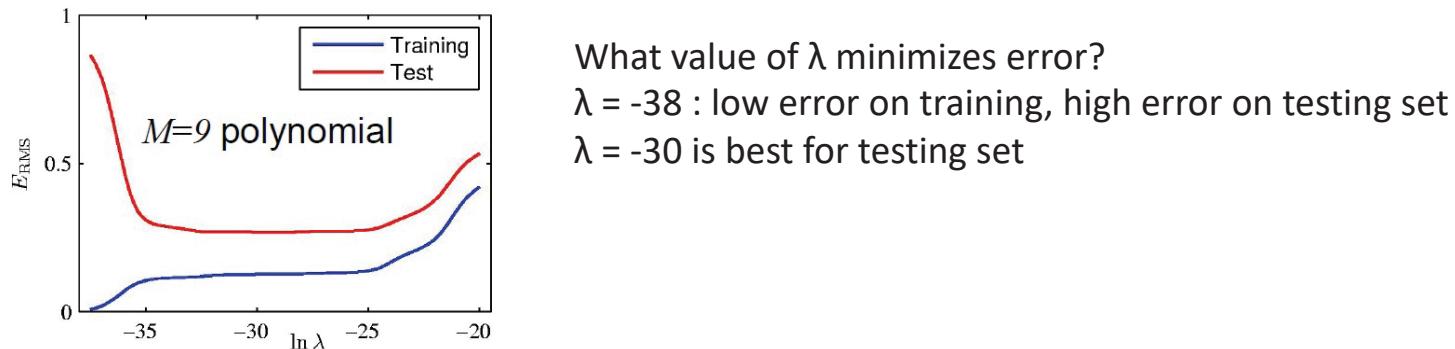
- $\lambda$  controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or  $\lambda$ )
  - Validation set (holdout)
    - to optimize model complexity (M or  $\lambda$ )





# Choosing a model using data

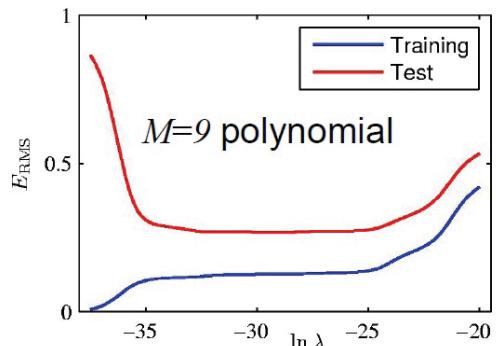
- $\lambda$  controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or  $\lambda$ )
  - Validation set (holdout)
    - to optimize model complexity (M or  $\lambda$ )





# Choosing a model using data

- $\lambda$  controls model complexity (similar to choice of M)
- Frequentist Approach:
  - Training set
    - To determine coefficients w for different values of (M or  $\lambda$ )
  - Validation set (holdout)
    - to optimize model complexity (M or  $\lambda$ )



What value of  $\lambda$  minimizes error?

$\lambda = -38$  : low error on training, high error on testing set

$\lambda = -30$  is best for testing set

$$E_{RMS} = \sqrt{2E(w^*) / N}$$

Division by N allows different data sizes to be compared since E is a sum over N Sqrt (of squared error) measures on same scale as t



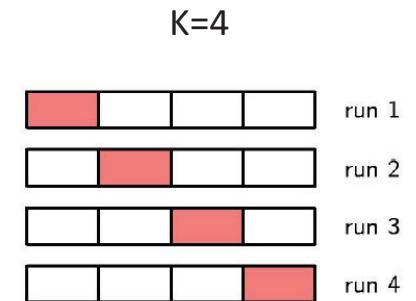
# Use the Validation Set!

- Performance on training set is not a good indicator of predictive performance
- If there is plenty of data,
  - use some of the data to train a range of models Or a given model with a range of values for its parameters
  - Compare them on an independent set, called validation set
  - Select one having best predictive performance
- If data set is small then some over-fitting can occur and it is necessary to keep aside a test set



# K-fold Cross Validation

- Supply of data is limited
- All available data is partitioned into K groups
- $K-1$  groups are used to train and evaluated on remaining group
- Repeat for all K choices of held-out group
- Performance scores from K runs are averaged



If  $K=N$ , then it is called the  
leave-one-out cross  
validation



# Disadvantage of Cross-Validation

- No. of training runs is increased by factor of K
- Problematic if training itself is expensive
- Different data sets can yield different complexity parameters for a single model
  - E.g., for given M several values of  $\lambda$
- Combinations of parameters is exponential
- Need a better approach
  - Ideally one that depends only on a single run with training data and should allow multiple hyperparameters and models to be compared in a single run



## POTENTIAL PROBLEMS WITH CROSS-VALIDATION

---

- CV is awesome and it can be used on clustering, density estimation, classification, regression, etc.
- But intensive use of cross-validation can overfit, if you explore too many models, by finding a model that accidentally predicts the whole training set well (and thus every leave-one-out sample well).
- CV can also be very time consuming if done naively.
- Often there are efficient tricks for computing all possible leave-one-out cross validation folds, which can save you a lot of work over brute-force retraining on all  $N$  possible LOO datasets.
- For example, in linear regression, the term  $(\sum_{n \neq \ell} \mathbf{x}_n \mathbf{x}_n^\top)^{-1}$  which leaves out datapoint  $\ell$  can be computed using the matrix inversion lemma:  $(\sum_n \mathbf{x}_n \mathbf{x}_n^\top - \mathbf{x}_\ell \mathbf{x}_\ell^\top)^{-1}$ .
- This is also true of the Generalized Cross Validation (GCV) estimate of Golub and Wahaba. (see extra readings)



## MODEL AVERAGING

---

- One last way to reduce variance, while not affecting bias too severely, is to average together the predictions of a bunch of different models.
- These models must be different in some way, either because they were trained on different subsets of the data, or with different regularization parameters, different local optima, or something.
- When we average them together, we would like to weight more strongly the models we believe are fitting the data better.
- Such systems are often called *committee machines*.
- Really, this is just a weak form of Bayesian learning.  
MAP = estimate of mode of posterior over models  
Bagging (next class) = estimate of mean of posterior over models  
BIC/AIC = estimates of correct predictive distribution



# What do we need?



# What do we need?

- Need to find a performance measure that depends only on the training data and which does not suffer from bias due to over-fitting
- Historically various information criteria have been proposed that attempt to correct the for the bias of maximum likelihood by the addition of a penalty term to compensate for the overfitting of more complex models



# Akaike Information Criterion (AIC)

- AIC chooses model that maximizes  $\ln p(D|\mathbf{w}_{\text{ML}}) - M$ 
  - First term is best-fit log-likelihood for given model
  - M is no of adjustable parameters in model
- Penalty term M is added for over-fitting using many parameters
- Bayesian Information Criterion (BIC) is a variant of this quantity
- Disadvantages:
  - need runs with each model, prefers overly simple models



# Bayesian Perspective

- Avoids over-fitting
  - By marginalizing over model parameters
    - sum over model parameters instead of point estimates
- Models compared directly over training data
  - No need for validation set
  - Allows use of all available data in training set
  - Avoids multiple training runs for each model associated with cross-validation
  - Allows multiple complexity parameters to be simultaneously determined during training
    - Relevance vector m/c is Bayesian model with one complexity parameter for every data point



# How do you choose a model?



# How do you choose a model?

- Based on its evidence!
- A model is a probability distribution over data D – E.g., a polynomial model is a distribution over target values  $t$  when input  $x$  is known, i.e.,  $p(t|x,D)$
- Uncertainty in model itself can be represented by probabilities
- Compare a set of models  $M_i$ ,  $i=1,..L$
- Given training set, wish to evaluate posterior

$$p(M_i|D) \propto p(M_i) p(D|M_i)$$

Prior expresses preference for different models  
We assume equal priors

Model Evidence (or Marginal Likelihood)



# Model Evidence

- $p(D | M_i)$  is preference shown by data for model
- Called Model evidence or *marginal likelihood*
  - because parameters have been *marginalized*



# Bayes Factor

- From model evidence  $p(D|M_i)$  for each model:
  - The ratio  $\frac{p(D|M_i)}{p(D|M_j)}$  is called the Bayes Factor
  - Shows the preference for Model  $i$  over Model  $j$



# Predictive Distribution

- Given  $p(M_i | D)$  the predictive distribution is given by

$$p(t | \mathbf{x}, D) = \sum_{i=1}^L \underbrace{p(t | \mathbf{x}, M_i, D)}_{\text{Prediction under model}} p(M_i | D)$$

- This is called a *mixture*:
  - predictions under different models are weighted by posterior probabilities of models
- Instead of all models  $M_i$ , approximate with single most probable model
  - Known as *Model Selection*



# Bayes factor always favors correct model

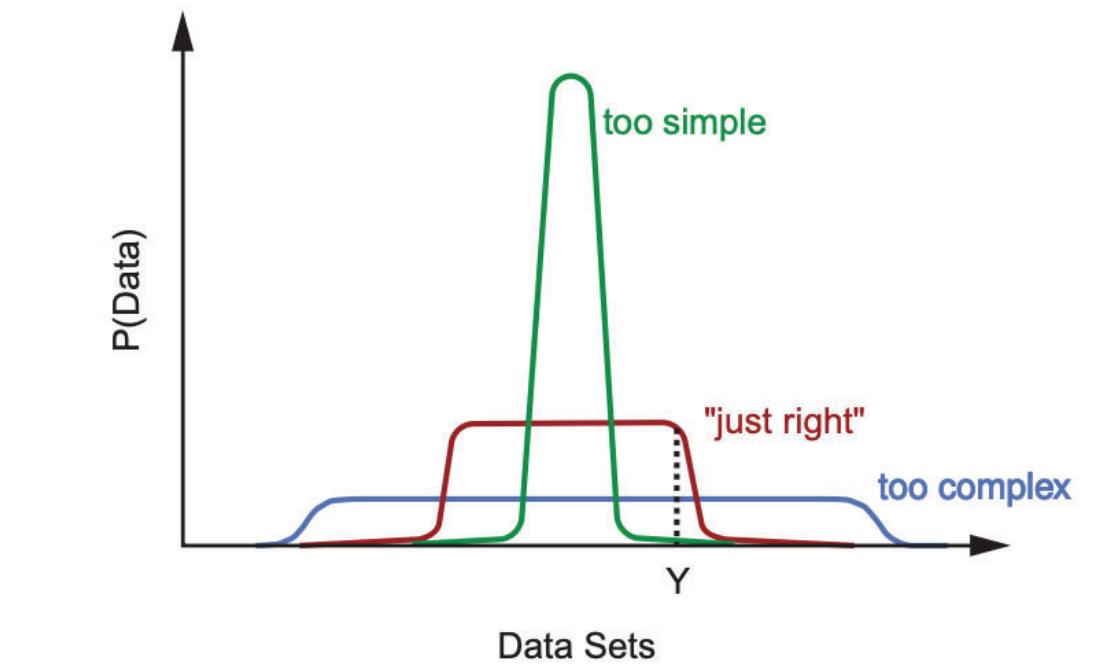
- We are assuming the true model is contained among the  $M_i$
- If this is so, Bayes model comparison will favor the correct model
- If  $M_1$  is the true model, and we average over the distribution of data sets, Bayes factor has the form

$$\int p(D | M_1) \ln \frac{p(D | M_1)}{p(D | M_2)} dD$$

- This is K-L Divergence which is always positive
  - Thus Bayes factor favors the correct model



## OCKHAM'S RAZOR



We want to use the simplest model which explains the data well.  
[A now famous figure, first introduced by Mackay.]



## NO FREE LUNCH

---

- David Wolpert and others have proven a series of theorems, known as the “no free lunch” theorems which, roughly speaking, say that *unless you make some assumptions* about the nature of the functions or densities you are modeling, no one learning algorithm can *a priori* be expected to do better than any other algorithm.
- In particular, this lack of clear advantage includes any algorithm and any meta-learning procedure applied to that algorithm. In fact, “anti-cross-validation” (i.e. picking the regularization parameters that give the *worst* performance on the CV samples) is *a priori* just as likely to do well as cross-validation. Without assumptions, random guessing is no worse than any other algorithm.
- So capacity control, regularization, validation tricks and meta-learning (next class) cannot *always* be successful.



## GENERALIZATION ERROR VS. LEARNING ERROR

---

- A key issue here is the difference between test error on a test set drawn from the same distribution as the training data (may contain duplicates) and *out of sample* test error.
- Remember back to the first class: learning binary functions.  
No assumptions == no generalization on out of sample cases.
- The only way to learn is to wait until you have seen the whole world and memorize it.
- Luckily, we *can* make some progress in real life.
- Why? Because the assumptions we make about function classes are often partly true.



# Summary

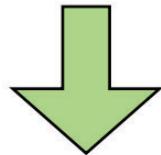
- Avoids problem of over-fitting
- Allows models to be compared using training data alone
- However needs to make assumptions about form of model
  - Model evidence can be sensitive to many aspects of prior such as behavior of tails
- In practice necessary to keep aside independent test data to evaluate performance



# 3 levels of inference

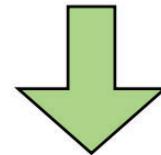
## LEVEL 1

I have selected a model  $M$  and prior  $P(\theta|M)$



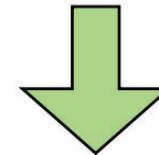
## LEVEL 2

Actually, there are several possible models:  $M_0, M_1, \dots$



## LEVEL 3

None of the models is clearly the best



### Parameter inference

What are the favourite values of the parameters?  
(assumes  $M$  is true)

### Model comparison

What is the relative plausibility of  $M_0, M_1, \dots$  in light of the data?

### Model averaging

What is the inference on the parameters accounting for model uncertainty?



# What is SOTA Research?

## META-LEARNING

---

- The idea of meta-learning is to come up with some procedure for taking a learning algorithm and a fixed training set, and somehow repeatedly applying the algorithm to *different* subsets (weightings) of the training set or using *different* random choices within the algorithm in order to get a large ensemble of machines.
- The machines in the ensemble are then *combined* in some way to define the final output of the learning algorithm (e.g. classifier)
- The hope of meta-learning is that it can “supercharge” a mediocre learning algorithm into an excellent learning algorithm, without the need for any new ideas!
- There is, as always, good news and bad news....
  - The Bad News: there is (quite technically) No Free Lunch.
  - The Good News: for many real world datasets, meta learning works well because its implicit assumptions are often reasonable.



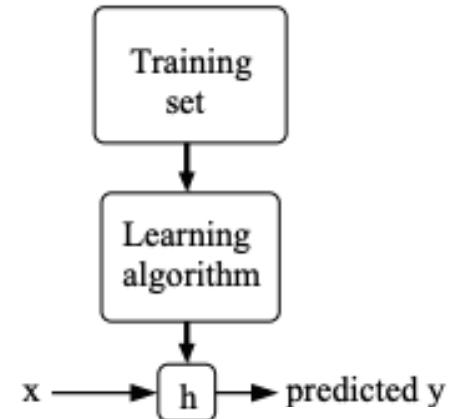
# Logistic Regression

On to Classification!



# The basics

- Input: a set of inputs  $X = \{x_1, x_2, \dots x_n\}$ , also called **features**
- Output: a set of expected outputs or **targets**  $Y = \{y_1, y_2, \dots y_n\}$
- Goal: to learn a function  $h : X \rightarrow Y$  such that the function  $h(x_i)$  is a good predictor of the corresponding value  $y_i$ 
  - $h(x)$  is called the **hypothesis**
- If the target is continuous the problem setting is called **regression**.
- If the target is discrete or categorical, the problem is called **classification**.





# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)	House Type
1643	4	256	Condo
1356	3	202	Apartment
1678	3	287	House
...	...	...	
3000	4	400	House

**Features (X)** A red bracket is positioned below the first three columns of the table, spanning from the header to the last data row. An arrow points from the text 'Features (X)' to the right end of this bracket.

**Targets (Y)** A red arrow points from the text 'Targets (Y)' to the fourth column of the table, specifically pointing at the 'House Type' header.



# Logistic Regression

- Goal: formulate a hypothesis function  $h(x)$  which will model the 3-d input feature (size, # bedrooms, price) and produce the expected target value (type of home i.e. condo, apartment, house, etc.).
- Let us consider a 2-class problem i.e. a binary classifier that says whether the given home is a house or not a house.
  - Represent as 0 and 1
  - 0 → negative class
  - 1 → positive class
- Given  $x^{(i)}$ , the corresponding  $y^{(i)}$  is also called the label for the training example.



# Logistic Regression

- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$



# Logistic Regression

- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$ 
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less than 0.
    - Since the categories are 1 and 0 and  $y \in \{0, 1\}$ .



# Logistic Regression

- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$ 
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less than 0.
    - Since the categories are 1 and 0 and  $y \in \{0, 1\}$ .
- New hypothesis function:

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$



# Logistic Regression

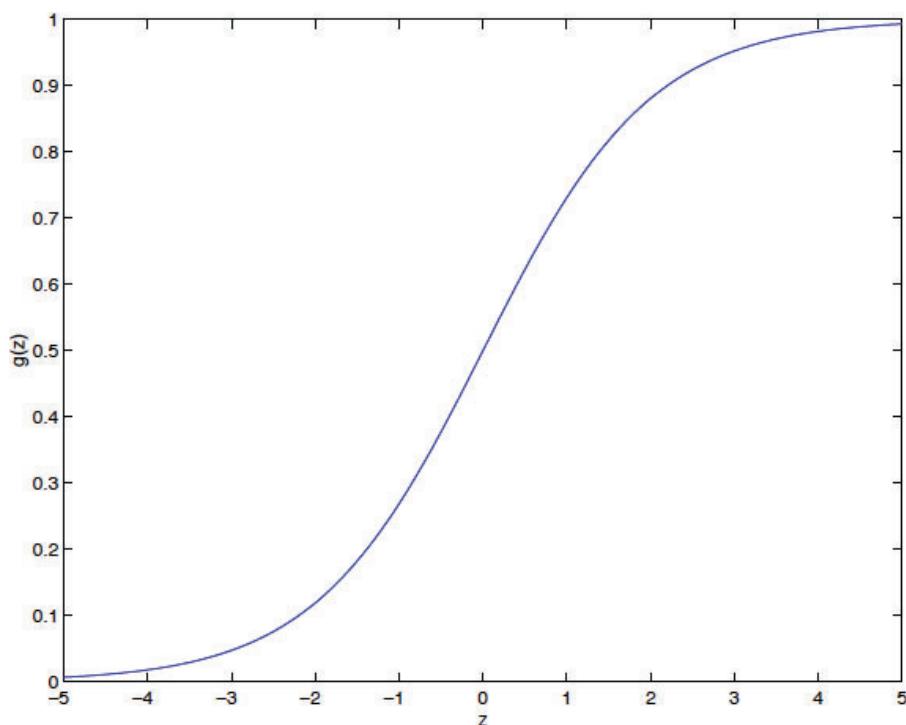
- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$ 
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less than 0.
    - Since the categories are 1 and 0 and  $y \in \{0, 1\}$ .
- New hypothesis function:

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

• where  $g(z) = \frac{1}{1 + e^{-z}}$  **Logistic Function**  $\theta^T x = \theta_0 + \sum_{j=1}^n \theta_j x_j$



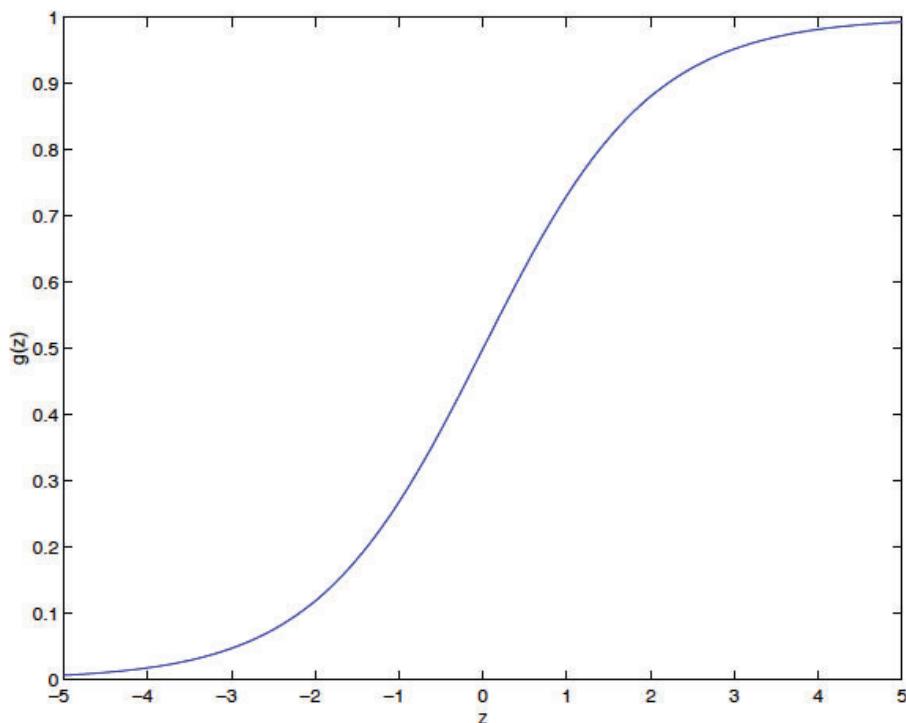
# Logistic Function



- $g(z)$  tends towards 1 as  $z \rightarrow \infty$
- $g(z)$  tends towards 0 as  $z \rightarrow -\infty$ .
- What does this tell us?



# Logistic Function



- $g(z)$  tends towards 1 as  $z \rightarrow \infty$
- $g(z)$  tends towards 0 as  $z \rightarrow -\infty$ .
  
- What does this tell us?
- **$g(z)$ , and hence also  $h(x)$ , is always bounded between 0 and 1.**



# How do we get $\theta$ ?

- Gradient Descent!
- What do we need for gradient descent?
  - An objective function  $J(\theta)$
  - A Learning rate  $\alpha$
  - An initial “guess” for  $\theta$  called  $\theta_j$
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$



# Defining the Objective Function $J(\theta)$

- Let us say that

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

- Or, more concisely:

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$



# Defining the Objective Function $J(\theta)$

- Given:  $p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$
- We want to estimate  $\theta$  that will capture the dependency between  $y$  and  $x$ . When we wish to explicitly view this as a function of  $\theta$ , we will instead call it the ***likelihood function*** that maximizes  $p(y|X; \theta)$  and is given by

$$L(\theta) = \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta)$$



# Defining the Objective Function $J(\theta)$

- Given:  $p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$
- We want to estimate  $\theta$  that will capture the dependency between  $y$  and  $x$ . When we wish to explicitly view this as a function of  $\theta$ , we will instead call it the ***likelihood function*** that maximizes  $p(y|X; \theta)$  and is given by

$$\begin{aligned} L(\theta) &= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?
- The principle of ***maximum likelihood*** says that we should choose  $\theta$  that makes the data as high probability as possible. i.e., we should choose  $\theta$  to maximize  $L(\theta)$ .



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?
- The principle of ***maximum likelihood*** says that we should choose  $\theta$  that makes the data as high probability as possible. i.e., we should choose  $\theta$  to maximize  $L(\theta)$ .
- Instead of maximizing  $L(\theta)$ , we can also maximize any strictly increasing function of  $L(\theta)$ . So we will maximize the log likelihood  $\ell(\theta)$ :

$$\ell(\theta) = \log L(\theta)$$



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?
- The principle of ***maximum likelihood*** says that we should choose  $\theta$  that makes the data as high probability as possible. i.e., we should choose  $\theta$  to maximize  $L(\theta)$ .
- Instead of maximizing  $L(\theta)$ , we can also maximize any strictly increasing function of  $L(\theta)$ . So we will maximize the log likelihood  $\ell(\theta)$ :

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- So what is  $\frac{\partial}{\partial \theta_j} J(\theta)$ ?



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

$$\theta_j := \theta_j + \alpha \left( y^{(i)} - h_{\theta}(x^{(i)}) \right) x_j^{(i)}$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

Looks familiar?

$$\theta_j := \theta_j + \alpha \left( y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

Looks familiar?

$$\theta_j := \theta_j + \alpha \left( y^{(i)} - h_{\theta}(x^{(i)}) \right) x_j^{(i)}$$

Identical to Linear Regression Update rule!



© Sathyanarayanan Aakur

59



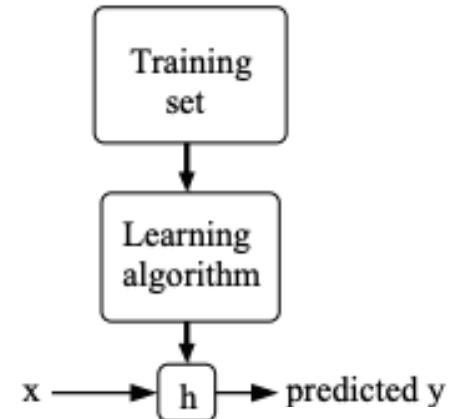
# COMP [56]630– Machine Learning

Lecture 7 – Logistic Regression



# The basics

- Input: a set of inputs  $X = \{x_1, x_2, \dots x_n\}$ , also called **features**
- Output: a set of expected outputs or **targets**  $Y = \{y_1, y_2, \dots y_n\}$
- Goal: to learn a function  $h : X \rightarrow Y$  such that the function  $h(x_i)$  is a good predictor of the corresponding value  $y_i$ 
  - $h(x)$  is called the **hypothesis**
- If the target is continuous the problem setting is called **regression**.
- If the target is discrete or categorical, the problem is called **classification**.





# Example

- Suppose we have a dataset giving the living areas and prices of 47 houses from Stillwater, OK

Living area (ft <sup>2</sup> )	# bedrooms	Price (1000\$s)	House Type
1643	4	256	Condo
1356	3	202	Apartment
1678	3	287	House
...	...	...	
3000	4	400	House

**Features (X)** A red bracket spans across the first three columns of the table, with a red arrow pointing to the right end of the bracket labeled 'Features (X)'.

**Targets (Y)** A red arrow points from the 'House Type' column to the text 'Targets (Y)' located below the table.



# Logistic Regression

- Goal: formulate a hypothesis function  $h(x)$  which will model the 3-d input feature (size, # bedrooms, price) and produce the expected target value (type of home i.e. condo, apartment, house, etc.).
- Let us consider a 2-class problem i.e. a binary classifier that says whether the given home is a house or not a house.
  - Represent as 0 and 1
  - $0 \rightarrow$  negative class
  - $1 \rightarrow$  positive class
- Given  $x^{(i)}$ , the corresponding  $y^{(i)}$  is also called the label for the training example.



# Logistic Regression

- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$



# Logistic Regression

- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$ 
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less than 0.
    - Since the categories are 1 and 0 and  $y \in \{0, 1\}$ .



# Logistic Regression

- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$ 
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less than 0.
    - Since the categories are 1 and 0 and  $y \in \{0, 1\}$ .
- New hypothesis function:

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$



# Logistic Regression

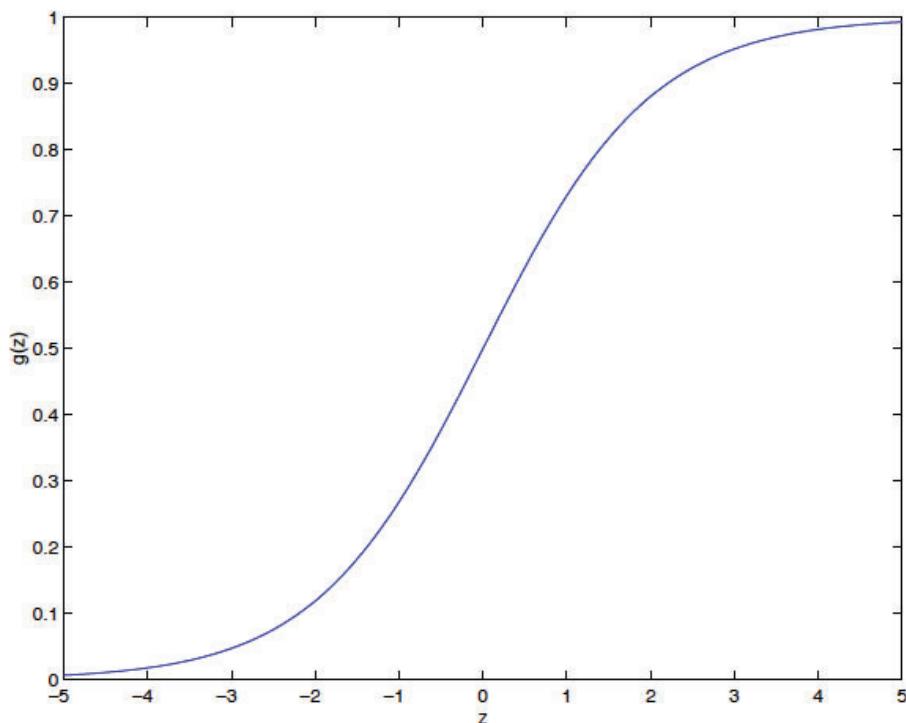
- We could approach the classification problem ignoring the fact that  $y$  is discrete-valued, and use our old linear regression algorithm to try to predict  $y$  given some features  $x$ 
  - Does not work all the time.
  - Does not make sense to have predictions greater than 1 or less than 0.
    - Since the categories are 1 and 0 and  $y \in \{0, 1\}$ .
- New hypothesis function:

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

• where  $g(z) = \frac{1}{1 + e^{-z}}$  **Logistic Function**  $\theta^T x = \theta_0 + \sum_{j=1}^n \theta_j x_j$



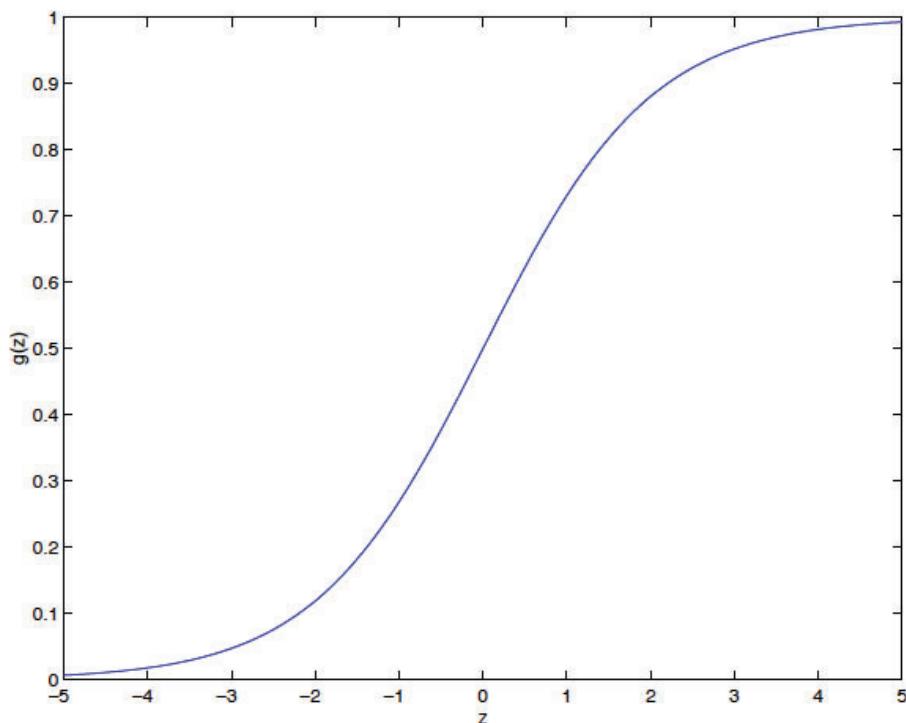
# Logistic Function



- $g(z)$  tends towards 1 as  $z \rightarrow \infty$
- $g(z)$  tends towards 0 as  $z \rightarrow -\infty$ .
- What does this tell us?



# Logistic Function



- $g(z)$  tends towards 1 as  $z \rightarrow \infty$
- $g(z)$  tends towards 0 as  $z \rightarrow -\infty$ .
  
- What does this tell us?
- **$g(z)$ , and hence also  $h(x)$ , is always bounded between 0 and 1.**



# How do we get $\theta$ ?

- Gradient Descent!
- What do we need for gradient descent?
  - An objective function  $J(\theta)$
  - A Learning rate  $\alpha$
  - An initial “guess” for  $\theta$  called  $\theta_j$
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$



# Defining the Objective Function $J(\theta)$

- Let us say that

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

- Or, more concisely:

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$



# Defining the Objective Function $J(\theta)$

- Given:  $p(y | x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$
- We want to estimate  $\theta$  that will capture the dependency between  $y$  and  $x$ . When we wish to explicitly view this as a function of  $\theta$ , we will instead call it the ***likelihood function*** that maximizes  $p(y|X; \theta)$  and is given by

$$L(\theta) = \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta)$$



# Defining the Objective Function $J(\theta)$

- Given:  $p(y \mid x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$
- We want to estimate  $\theta$  that will capture the dependency between  $y$  and  $x$ . When we wish to explicitly view this as a function of  $\theta$ , we will instead call it the ***likelihood function*** that maximizes  $p(y|X; \theta)$  and is given by

$$\begin{aligned} L(\theta) &= \prod_{i=1}^m p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?
- The principle of ***maximum likelihood*** says that we should choose  $\theta$  that makes the data as high probability as possible. i.e., we should choose  $\theta$  to maximize  $L(\theta)$ .



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?
- The principle of ***maximum likelihood*** says that we should choose  $\theta$  that makes the data as high probability as possible. i.e., we should choose  $\theta$  to maximize  $L(\theta)$ .
- Instead of maximizing  $L(\theta)$ , we can also maximize any strictly increasing function of  $L(\theta)$ . So we will maximize the log likelihood  $\ell(\theta)$ :

$$\ell(\theta) = \log L(\theta)$$



# Defining the Objective Function $J(\theta)$

- Now, given this probabilistic model how can we get  $\theta$ ?
- The principle of ***maximum likelihood*** says that we should choose  $\theta$  that makes the data as high probability as possible. i.e., we should choose  $\theta$  to maximize  $L(\theta)$ .
- Instead of maximizing  $L(\theta)$ , we can also maximize any strictly increasing function of  $L(\theta)$ . So we will maximize the log likelihood  $\ell(\theta)$ :

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- So what is  $\frac{\partial}{\partial \theta_j} J(\theta)$ ?



$$l(\theta) = \log(L(\theta)) \quad h(x) = g(\theta^T x)$$

$$= \sum [y \cdot \log(h(x)) + (1-y) \cdot \log(1-h(x))]$$

$$= \sum [y \cdot \log(g(\theta^T x)) + (1-y) \cdot \log(1-g(\theta^T x))]$$

$$\frac{d l(\theta)}{d \theta} = \sum \frac{d}{d \theta} [y \cdot \log(g(\theta^T x)) + (1-y) \cdot \log(1-g(\theta^T x))]$$

For simplicity, ignore the summation.

$$\therefore \frac{d l(\theta)}{d \theta} = y \cdot \frac{d}{d \theta} (\log(g(\theta^T x))) + (1-y) \cdot \frac{d}{d \theta} (\log(1-g(\theta^T x)))$$



we know that  $\frac{d}{dz}(\log(z)) = \frac{1}{z}$

$$\begin{aligned}\therefore \frac{d l(\theta)}{d\theta} &= \frac{y}{g(\theta^T x)} \cdot \frac{d}{d\theta}(g(\theta^T x)) + \left( \frac{(1-y)}{1-g(\theta^T x)} \right) \cdot \frac{d}{d\theta}(1-g(\theta^T x)) \\ &= \left[ \frac{y}{g(\theta^T x)} + \frac{(-1) \cdot (1-y)}{1-g(\theta^T x)} \right] \cdot \frac{d}{d\theta}(g(\theta^T x)) \\ &= \frac{y(1-g(\theta^T x)) - (1-y) \cdot g(\theta^T x)}{g(\theta^T x) \cdot (1-g(\theta^T x))} \cdot \frac{d}{d\theta}(g(\theta^T x))\end{aligned}$$



$$= \frac{y - y \cdot g(\theta^T x) - g(\theta^T x) + y \cdot g(\theta^T x)}{g(\theta^T x)(1 - g(\theta^T x))} \cdot \frac{d}{d\theta} (g(\theta^T x))$$

$$= \frac{y - g(\theta^T x)}{g(\theta^T x) \cdot (1 - g(\theta^T x))} \cdot \frac{d}{d\theta} (g(\theta^T x))$$



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

$$\frac{d g(z)}{dz} = \frac{d}{dz} \left( \frac{1}{1+e^{-z}} \right) = \frac{d}{dz} (1+e^{-z})^{-1}$$

we know that  $\frac{d}{dz}(z^n) = n(z^{n-1})$

$$\therefore \frac{d g(z)}{dz} = \frac{-1}{(1+e^{-z})^2} \cdot \frac{d}{dz} (1+e^{-z})$$

we know that  $\frac{d}{dz} e^z = e^z$



$$\therefore \frac{d}{dz} (g(z)) = \frac{-1}{(1+e^{-z})^2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1+e^{-z}-1}{1+e^{-z}} \cdot \left( \frac{1}{1+e^{-z}} \right)$$

$$= \left( 1 - \frac{1}{1+e^{-z}} \right) \cdot \left( \frac{1}{1+e^{-z}} \right)$$

$$\therefore \frac{d}{dz} (g(z)) = g(z) \cdot (1-g(z))$$



$$\therefore \frac{d \ell(\theta)}{d\theta} = \frac{y - g(\theta^T x)}{g(\theta^T x) \cdot (1 - g(\theta^T x))} \left[ g(\theta^T x) \cdot (1 - g(\theta^T x)) \right] \cdot \frac{d}{d\theta} (g(\theta^T x))$$

$$\therefore \frac{d \ell(\theta)}{d\theta} = [y - g(\theta^T x)] \cdot x$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

$$\theta_j := \theta_j + \alpha \left( y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

Looks familiar?

$$\theta_j := \theta_j + \alpha \left( y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$



# How do we get $\theta$ ?

- Gradient Descent!
- Then, update  $\theta_j$  until convergence as follows

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Hence our new update rule for *logistic* regression is

Looks familiar?

$$\theta_j := \theta_j + \alpha \left( y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Identical to Linear Regression Update rule!



© Sathyanarayanan Aakur

29



# How do we get $\theta$ ?

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

- Although the update rule is similar it is not the same algorithm!
- We have a non-linear hypothesis function!
- Is this coincidence, or is there a deeper reason behind this?



# How do we get $\theta$ ?

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

- Although the update rule is similar it is not the same algorithm!
- We have a non-linear hypothesis function!
- Is this coincidence, or is there a deeper reason behind this?
- **Both are part of a family of models called Generalized Linear Models!**



# Generalized Linear Models

- Not to be confused with the term *general linear model* (GLM).
  - It usually refers to conventional linear regression models for a continuous response variable given continuous predictors
- The form is  $y_i \sim N(x_i^T \beta, \sigma^2)$ , where  $x_i$  contains known covariates and  $\beta$  contains the coefficients to be estimated. These models are fit by least squares and weighted least squares
- The term *generalized linear model* (GLIM or GLM) refers to a larger class of models popularized by McCullagh and Nelder (1982, 2nd edition 1989).
  - In these models, the response variable  $y_i$  is assumed to follow an exponential family distribution with mean  $\mu_i$ , which is assumed to be some (often nonlinear) function of  $x_i^T \beta$ .



# Generalized Linear Models

- There are three components to any GLM:
- **Random Component** – refers to the probability distribution of the response variable ( $Y$ ); e.g. normal distribution for  $Y$  in the linear regression, or binomial distribution for  $Y$  in the binary logistic regression. Also called a noise model or error model. How is random error added to the prediction that comes out of the link function?
- **Systematic Component** - specifies the explanatory variables ( $X_1, X_2, \dots, X_k$ ) in the model, more specifically their linear combination in creating the so called *linear predictor*; e.g.,  $\beta_0 + \beta_1 x_1 + \beta_2 x_2$  as we have seen in a linear regression, or as we will see in a logistic regression in this lesson.
- **Link Function,  $\eta$  or  $g(\mu)$**  - specifies the link between random and systematic components. It says how the expected value of the response relates to the linear predictor of explanatory variables; e.g.,  $\eta = g(E(Y_i)) = E(Y_i)$  for linear regression, or  $\eta = \text{logit}(\pi)$  for logistic regression.



# Common Assumptions

- The data  $Y_1, Y_2, \dots, Y_n$  are independently distributed, i.e., cases are independent.
- The dependent variable  $Y_i$  does NOT need to be normally distributed, but it typically assumes a distribution from an exponential family (e.g. binomial, Poisson, multinomial, normal,...)
- GLM does NOT assume a linear relationship between the dependent variable and the independent variables, but it does assume linear relationship between the transformed response in terms of the link function and the explanatory variables; e.g., for binary logistic regression  $\text{logit}(\pi) = \beta_0 + \beta X$ .
- Independent (explanatory) variables can be even the power terms or some other nonlinear transformations of the original independent variables.
- The homogeneity of variance does NOT need to be satisfied. In fact, it is not even possible in many cases given the model structure, and *overdispersion* (when the observed variance is larger than what the model assumes) maybe present.
- Errors need to be independent but NOT normally distributed.



# Newton's Method for Estimating Theta



# Newton's Method

- an iterative equation solver
  - Typically used to find the roots of a polynomial function
  - Performs iterative updates as follows

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

- Where  $f'(\theta)$  is the derivative.



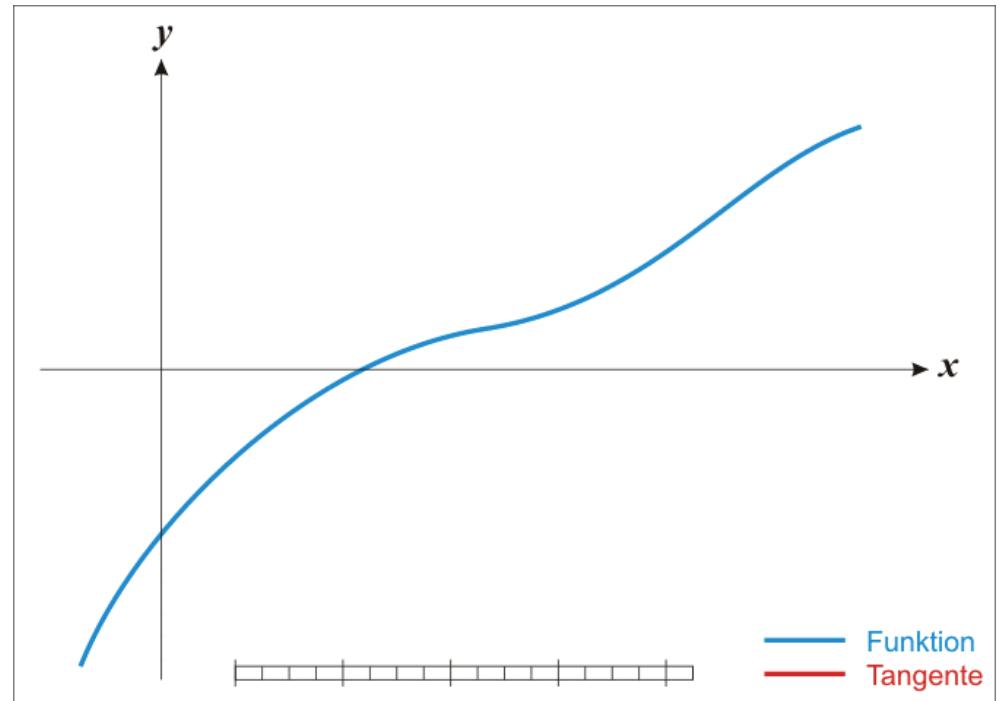
# Newton's Method

- Intuitively, we can think of it as approximating the function  $f$  via a linear function that is tangent to  $f$  at the current guess  $\theta$ , solving for where that linear function equals to zero, and letting the next guess for  $\theta$  be where that linear function is zero.



# Newton's Method

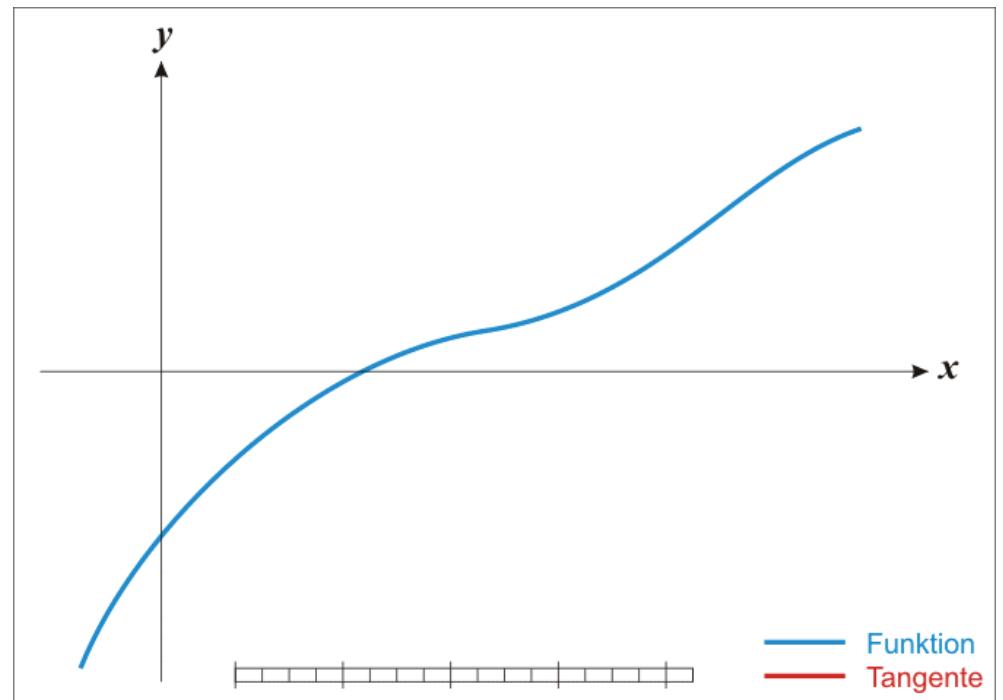
- Intuitively, we can think of it as approximating the function  $f$  via a linear function that is tangent to  $f$  at the current guess  $\theta$ , solving for where that linear function equals to zero, and letting the next guess for  $\theta$  be where that linear function is zero.





# Newton's Method

- Intuitively, we can think of it as approximating the function  $f$  via a linear function that is tangent to  $f$  at the current guess  $\theta$ , solving for where that linear function equals to zero, and letting the next guess for  $\theta$  be where that linear function is zero.  
. Until  $x_n - x_{n+1} \approx 0$ :  
◦  $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$





# Newton's Method for LR

- In our case, we consider the log likelihood as the function for which we want to find the ideal  $\theta$
- But we want to maximize the log likelihood, whereas Newton's method gives us the point at which it is 0.
- **How do we handle this?**



# Newton's Method for LR

- In our case, we consider the log likelihood as the function for which we want to find the ideal  $\theta$
- But we want to maximize the log likelihood, whereas Newton's method gives us the point at which it is 0.
- **How do we handle this?**
- Remember the maxima of  $\ell$  correspond to points where its first derivative  $\ell'(\theta)$  is zero. So, by letting  $f(\theta) = \ell'(\theta)$ , we can use the same algorithm to maximize  $\ell$ , and we obtain update rule:

$$\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$



# Newton's Method for LR

- $\theta$  is vector-valued (multi-dimensional), so we need to generalize Newton's method to this setting.
- The generalization of Newton's method to this multidimensional setting is called the Newton-Raphson method is given by

$$\theta := \theta - H^{-1} \nabla_{\theta} \ell(\theta)$$

- Where  $\nabla_{\theta} \ell(\theta)$  is the vector of partial derivatives of  $\ell(\theta)$  w.r.t.  $\theta$  and  $H$  is a  $n \times n$  matrix called the **Hessian**

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$$



# Newton's Method for LR

- $\theta$  is vector-valued (multi-dimensional), so we need to generalize Newton's method to this setting.
- The generalization of Newton's method to this multidimensional setting is called the Newton-Raphson method is given by

$$\theta := \theta - H^{-1} \nabla_{\theta} \ell(\theta)$$

- Where  $\nabla_{\theta} \ell(\theta)$  is the vector of partial derivatives of  $\ell(\theta)$  w.r.t.  $\theta$  and  $H$  is a  $n \times n$  matrix called the **Hessian**

$$H_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}$$

The Hessian is a square matrix of second-order partial derivatives of order  $n \times n$



# Newton's Method vs Gradient Descent

- Advantages:
  - Newton's method typically enjoys faster convergence than (batch) gradient descent
  - It requires many fewer iterations to get very close to the minimum.
- Disadvantages:
  - One iteration of Newton's can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an n-by-n Hessian; but so long as n is not too large, it is usually much faster overall.
  - When Newton's method is applied to maximize the logistic regression log likelihood function  $\ell(\theta)$ , the resulting method is also called **Fisher scoring**.



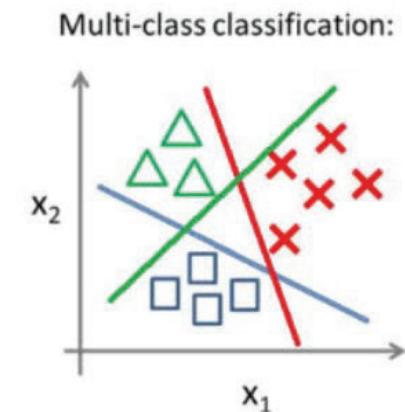
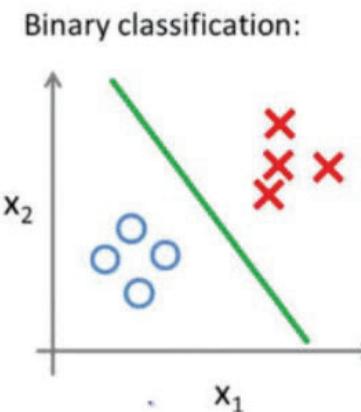
# Multiclass predictions

Extending binary classification with Logistic Regression to multiple classes



# Binary vs Multi-class Classification

- More than two classes (binary) in our problem setting.
- Logistic regression returns a confidence score that scores the probability of whether an example is positive class or negative class.
- Binary classification requires only one classifier model.

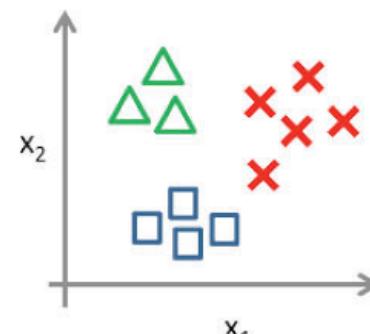




# One-vs-all

- Create the N-binary classifier models, one for each class in the data.
- The number of class labels present in the dataset and the number of generated binary classifiers must be the same.
- Also called one-vs-rest classification

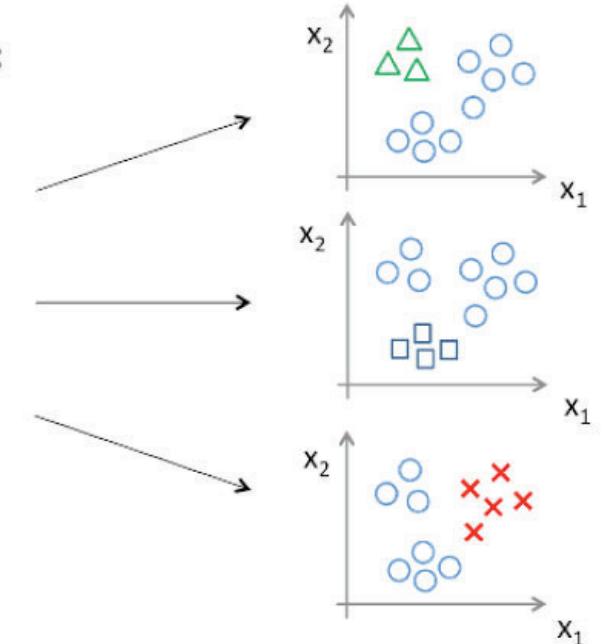
One-vs-all (one-vs-rest):



Class 1: Green

Class 2: Blue

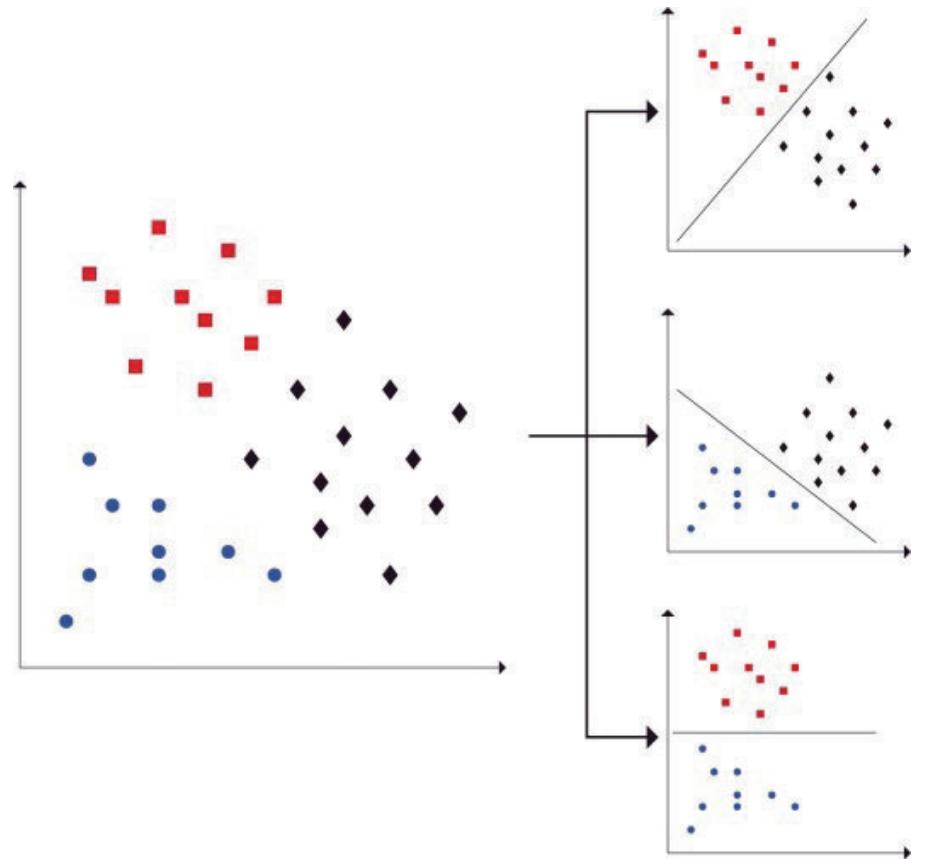
Class 3: Red





# One-vs-one

- Create the  $N^* (N-1)/2$  binary classifier models, one for each combination of classes in the data.
- Let us say there are three classes, A, B and C.
  - Create classifier for A-vs-B, A-vs-C, B-vs-C.
  - Each binary classifier predicts one class label.
  - Take the class with majority votes as final prediction





# Perceptron Algorithm



# The Perceptron

- Consider modifying the logistic regression method to “force” it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of  $g$  to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- If we then let  $h(x) = g(\theta^T x)$  as before but using this modified definition of  $g$ , and if we use the update rule

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$



# Perceptron

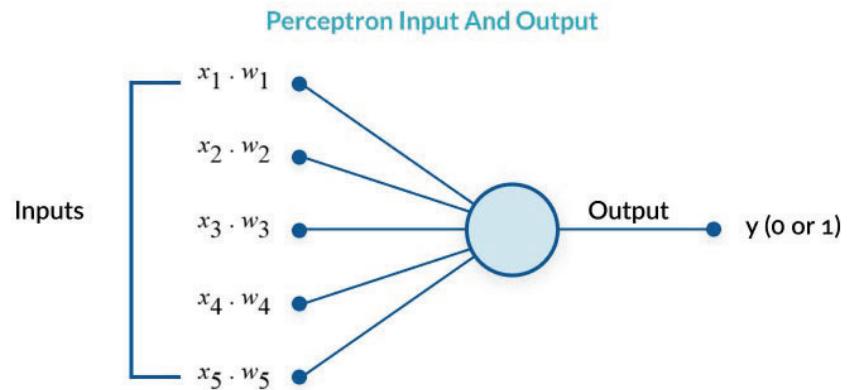
- In the 1960s, this “perceptron” was argued to be a rough model for how individual neurons in the brain work.
- Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression.
  - The hypothesis in logistic regression provides a measure of uncertainty in the occurrence of a binary outcome based on a linear model.
  - The output from a step function can of course not be interpreted as any kind of probability.
  - Since a step function is not differentiable, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.



# Multi-layer perceptron



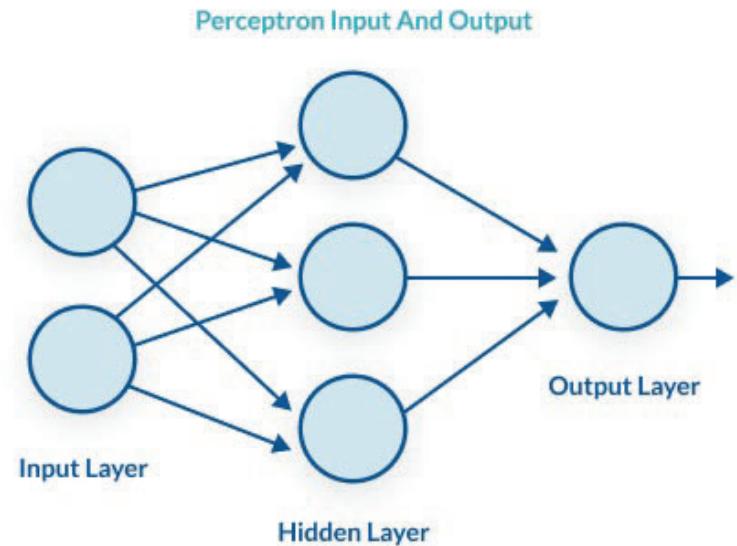
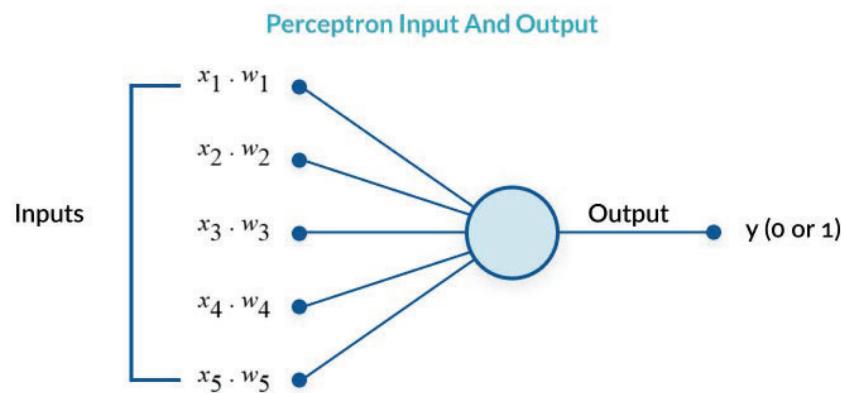
# Perceptron vs Multilayer Perceptron (MLP)





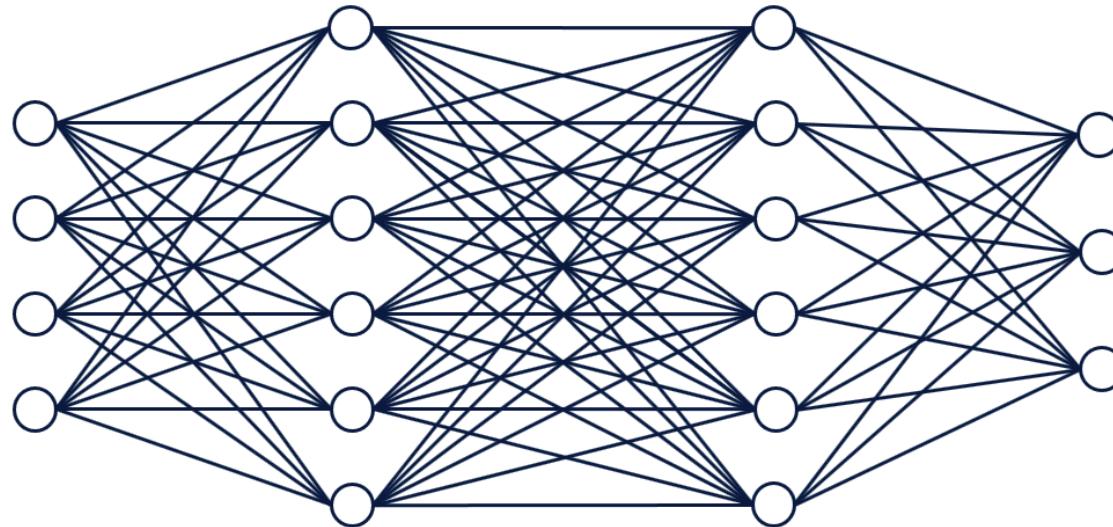


# Perceptron vs Multilayer Perceptron (MLP)





# Why do we need more layers?





# COMP [56]630– Machine Learning

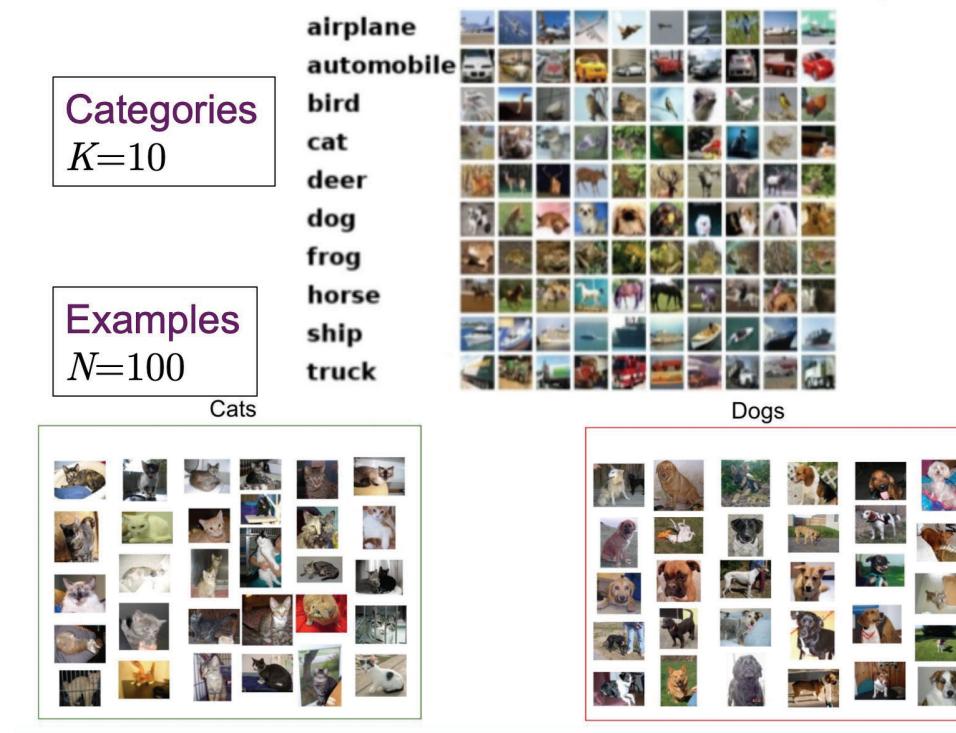
Lecture 8 – Multiclass LR, Evaluation Metrics



# Multi-class problems



# Multi-class classification

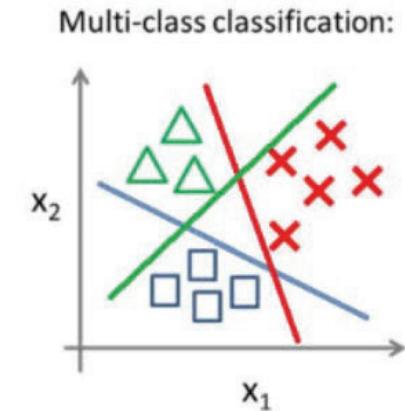
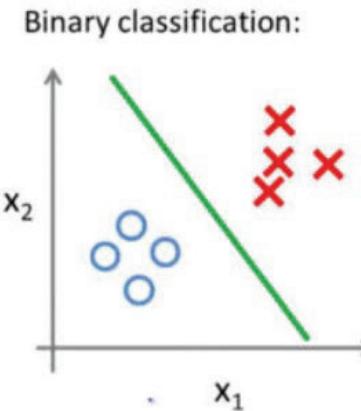


Credits to Dr. Srihari @ Buffalo



# Binary vs Multi-class Classification

- More than two classes (binary) in our problem setting.
- Logistic regression returns a confidence score that scores the probability of whether an example is positive class or negative class.
- Binary classification requires only one classifier model.

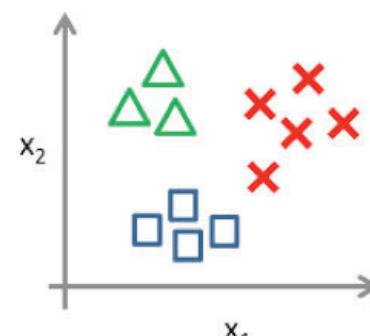




# One-vs-all

- Create the N-binary classifier models, one for each class in the data.
- The number of class labels present in the dataset and the number of generated binary classifiers must be the same.
- Also called one-vs-rest classification

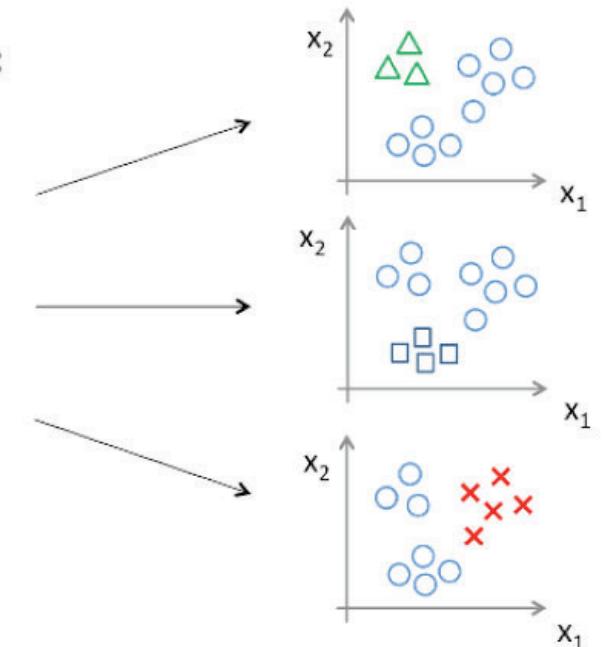
One-vs-all (one-vs-rest):



Class 1: Green

Class 2: Blue

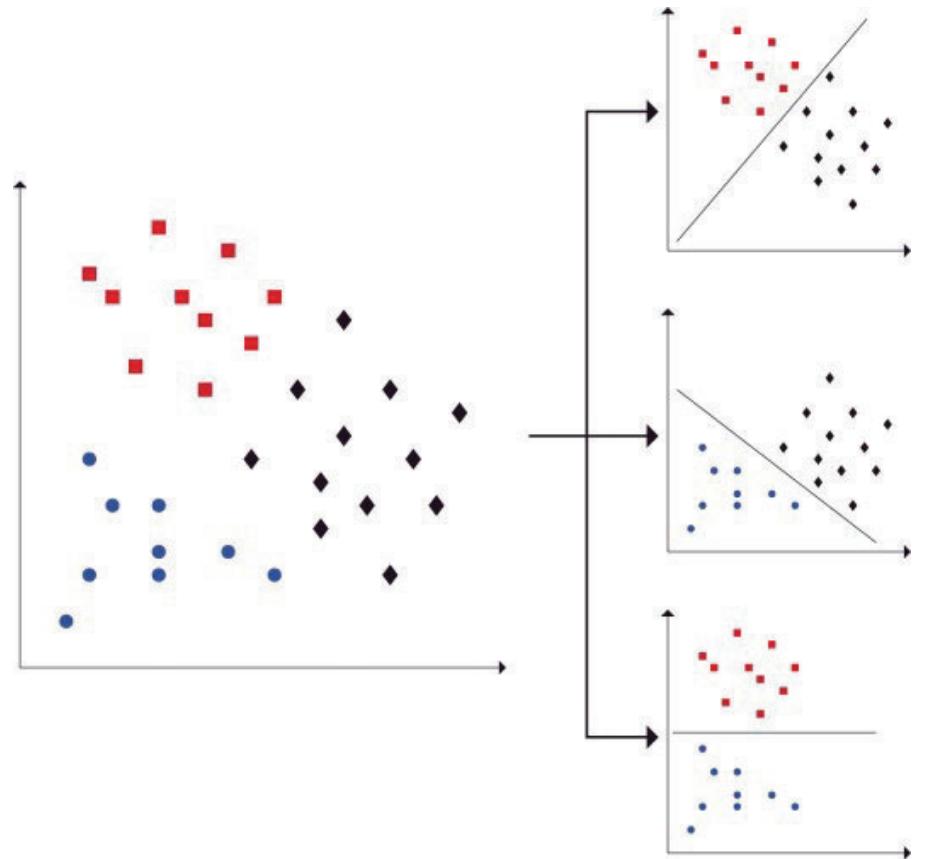
Class 3: Red





# One-vs-one

- Create the  $N^* (N-1)/2$  binary classifier models, one for each combination of classes in the data.
- Let us say there are three classes, A, B and C.
  - Create classifier for A-vs-B, A-vs-C, B-vs-C.
  - Each binary classifier predicts one class label.
  - Take the class with majority votes as final prediction





In the two-class case  $p(C_1|\phi) = y(\phi) = \sigma(\mathbf{w}^T\phi + b)$

where  $\phi = [\phi_1, \dots, \phi_M]^T$ ,  $\mathbf{w} = [w_1, \dots, w_M]^T$  and  $a = \mathbf{w}^T\phi + b$  is the “activation”

For  $K$  classes, we work with soft-max function instead of logistic sigmoid (*Softmax regression*)

$$p(C_k | \phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

where  $a_k = \mathbf{w}_k^T \phi + b_k$ ,  $k = 1, \dots, K$   
 $\mathbf{w}_k = [w_{k1}, \dots, w_{kM}]^T$  and  $\mathbf{a} = \{a_1, \dots, a_K\}$

– We learn a set of  $K$  weight vectors  $\{\mathbf{w}_1, \dots, \mathbf{w}_K\}$  and biases  $b$

Arranging weight vectors as a matrix  $W$

$$\mathbf{a} = W^T \phi + \mathbf{b}$$

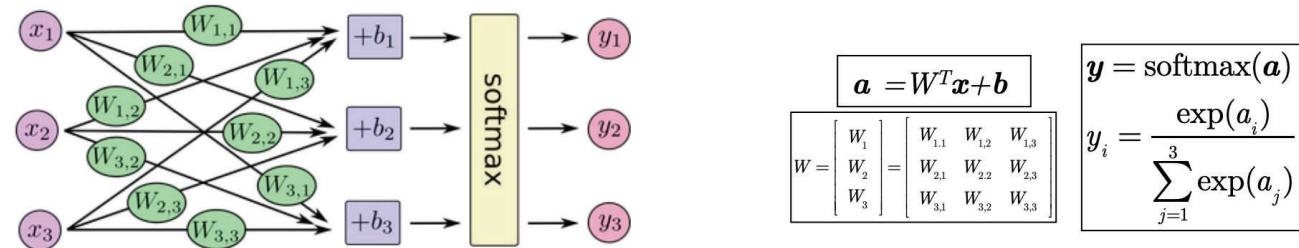
$$W = \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_K \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1M} \\ \vdots & \ddots & \vdots \\ w_{K1} & \cdots & w_{KM} \end{bmatrix}$$

$$\mathbf{y} = \text{softmax}(\mathbf{a})$$

$$y_i = \frac{\exp(a_i)}{\sum_{j=1}^3 \exp(a_j)}$$



# Multi-class Logistic Regression



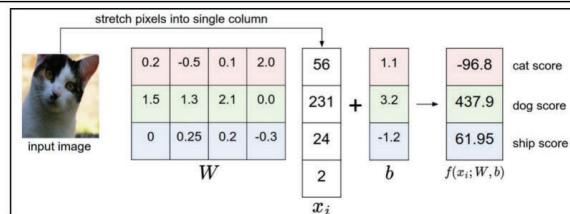
Network  
Computes

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{array}{l} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{array} \right)$$

In matrix  
multiplication  
notation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left( \begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

An example





We use maximum likelihood to determine the parameters  $\{w_k\}, k=1,..K$

The exp within softmax 
$$\text{softmax}(\mathbf{a})_i = \frac{\exp(a_i)}{\sum_j \exp(a_j)}$$
 works very well when training using log-likelihood

- Log-likelihood can undo the exp of softmax

$$\log \text{softmax}(\mathbf{a})_i = a_i - \log \sum_j \exp(a_j)$$

- Input  $a_i$  always has a direct contribution to cost
  - Because this term cannot saturate, learning can proceed even if second term becomes very small
- First term encourages  $a_i$  to be pushed up
- Second term encourages all  $a$  to be pushed down



# Derivatives

The multiclass logistic regression model is

$$p(C_k | \phi) = y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

For maximum likelihood we will need the derivatives of  $y_k$  wrt all of the activations  $a_j$

These are given by

$$\frac{\partial y_k}{\partial a_j} = y_k(I_{kj} - y_j)$$

– where  $I_{kj}$  are the elements of the identity matrix



# One-hot targets

Classes  $C_1, \dots, C_K$  represented by 1-of- $K$  scheme

– One-hot vector:

- class  $C_k$  is a  $K$ -dim vector or  $[t_1, \dots, t_K]^T$ ,  $t_i \in \{0,1\}$ 
  - With  $K=6$ , class  $C_3$  is  $(0,0,1,0,0,0)^T$  with  $t_1=t_2=t_4=t_5=t_6=0$ , &  $t_3=1$

– The class probabilities obey

$$\sum_{k=1}^K p(C_k) = \sum_{k=1}^K t_k = 1$$

– If  $p(t_k=1) = \mu_k$  then

$$p(C_k) = \prod_{k=1}^K \mu_k^{t_k} \text{ where } \boldsymbol{\mu} = (\mu_1, \dots, \mu_K)^T$$

e.g., probability of  $C_3$  is

$$p([0,0,1,0,0,0]) = \mu_3$$

Why use one-hot representation?

If we used numerical categories 1,2,3... we would impute ordinality.  
We can now use simpler Bernoulli instead of multinoulli



# Target Matrix

Classes have values  $1, \dots, K$

Each represented as a  $K$ -dimensional binary vector

We have  $N$  labeled samples

- So instead of target vector  $t$  we have a target matrix  $T$

$$T = \begin{bmatrix} t_{11} & \cdot & \cdot & t_{1K} \\ \cdot & & & \\ \cdot & & & \\ t_{N1} & & & t_{NK} \end{bmatrix}$$

Classes →

Samples ↓

Note that  $t_{nk}$  corresponds to sample  $n$  and class  $k$



# Objective Function & Gradient

Likelihood of observations

$$p(T | \mathbf{w}_1, \dots, \mathbf{w}_K) = \prod_{n=1}^N \prod_{k=1}^K p(C_k | \phi_n)^{t_{nk}} = \prod_{n=1}^N \prod_{k=1}^K y_k^{t_{nk}}$$

$$T = \begin{bmatrix} t_{11} & \dots & t_{1K} \\ \vdots & & \vdots \\ t_{N1} & & t_{NK} \end{bmatrix}$$

- Where, for feature vector  $\phi_n$   $y_{nk} = y_k(\phi_n) = \frac{\exp(\mathbf{w}_k^T \phi_n)}{\sum_j \exp(\mathbf{w}_j^T \phi_n)}$

Objective Function: negative log-likelihood

$$E(\mathbf{w}_1, \dots, \mathbf{w}_K) = -\ln p(T | \mathbf{w}_1, \dots, \mathbf{w}_K) = -\sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln y_{nk}$$

– Known as cross-entropy error for multi-class

Gradient of error function wrt parameter  $\mathbf{w}_j$

$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1, \dots, \mathbf{w}_K) = -\sum_{n=1}^N (y_{nj} - t_{nj}) \phi_n$$

using  $\sum_k t_{nk} = 1$

Error x Feature Vector

$$y_k(\phi) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \quad a_k = \mathbf{w}_k^T \phi$$

$\frac{\partial y_k}{\partial a_j} = y_k (I_{kj} - y_j)$  where  $I_{kj}$  are elements of the identity matrix



# Performance of Classifiers



# Confusion Matrix

- Used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.



# Confusion Matrix

- Used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN



# Confusion Matrix

- Used to **describe the performance of a classification model** (or "classifier") on a set of test data for which the true values are known.
- Models Four major characteristics
- **true positives (TP)**
- **true negatives (TN)**
- **false positives (FP)**
- **false negatives (FN)**

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN



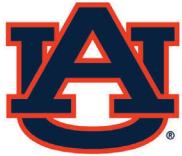
# Perf. Measures from Confusion Matrix

- **Accuracy:** Overall, how often is the classifier correct?
  - $(TP+TN)/\text{total}$
- **Misclassification Rate:** Overall, how often is it wrong?
  - $(FP+FN)/\text{total}$
- **Recall:** When it's actually yes, how often does it predict yes?
  - also known as "Sensitivity"

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **Precision:** When it predicts yes, how often is

$$\text{Precision} = \frac{TP}{TP + FP}$$



# Perceptron Algorithm



# The Perceptron

- Consider modifying the logistic regression method to “force” it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of  $g$  to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- If we then let  $h(x) = g(\theta^T x)$  as before but using this modified definition of  $g$ , and if we use the update rule

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$



# Perceptron

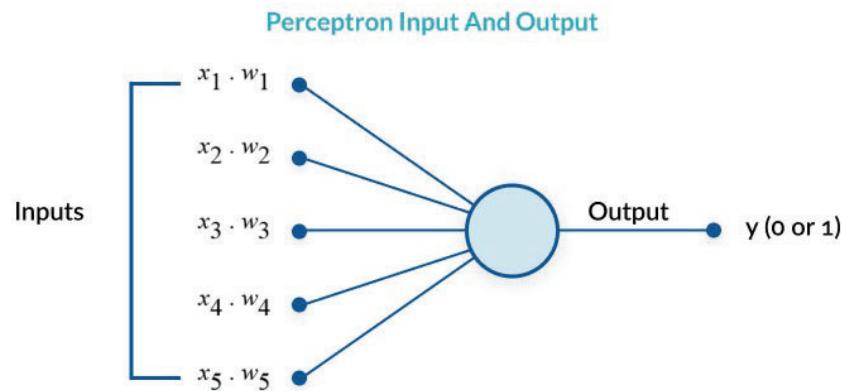
- In the 1960s, this “perceptron” was argued to be a rough model for how individual neurons in the brain work.
- Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression.
  - The hypothesis in logistic regression provides a measure of uncertainty in the occurrence of a binary outcome based on a linear model.
  - The output from a step function can of course not be interpreted as any kind of probability.
  - Since a step function is not differentiable, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.



# Multi-layer perceptron



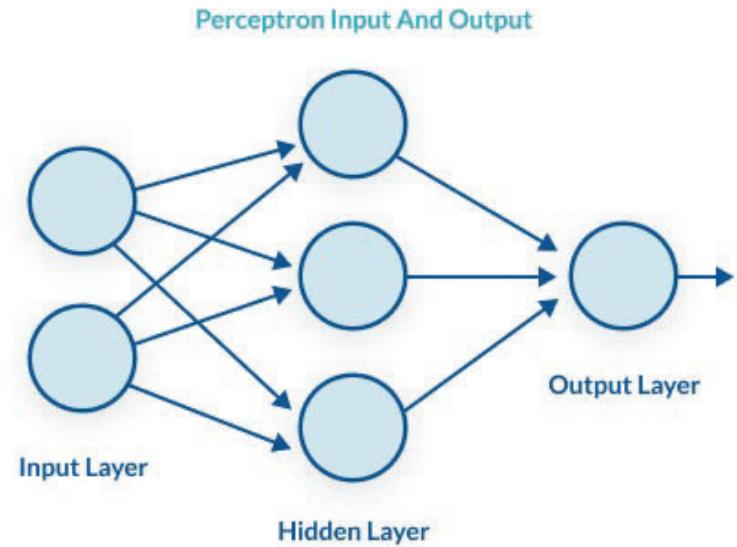
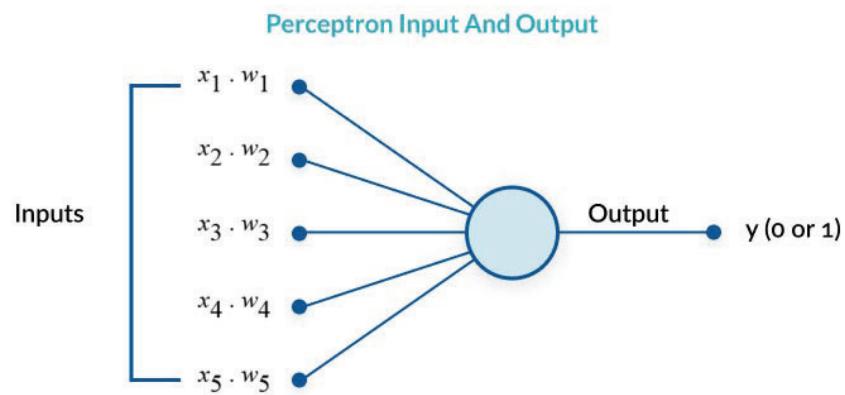
# Perceptron vs Multilayer Perceptron (MLP)





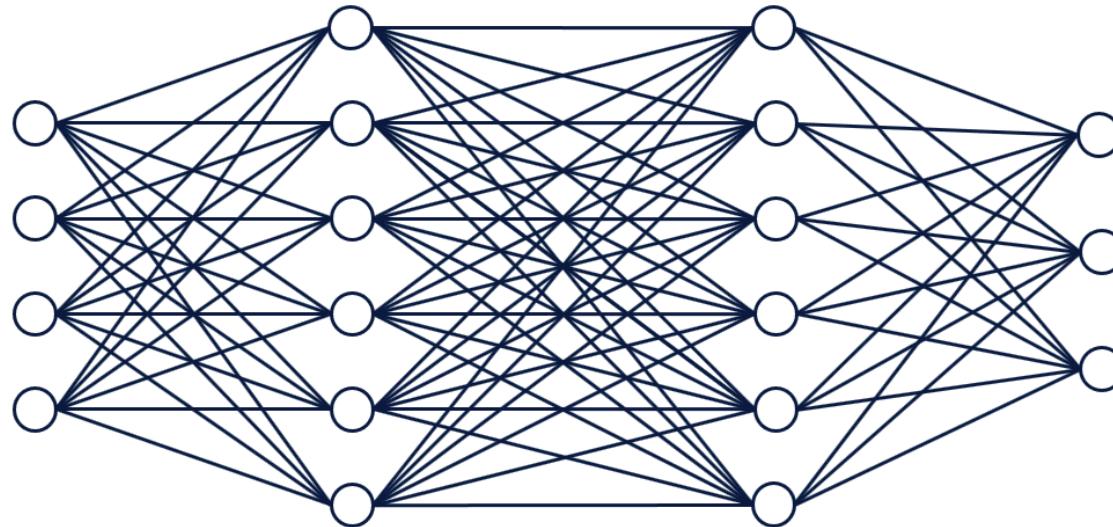


# Perceptron vs Multilayer Perceptron (MLP)

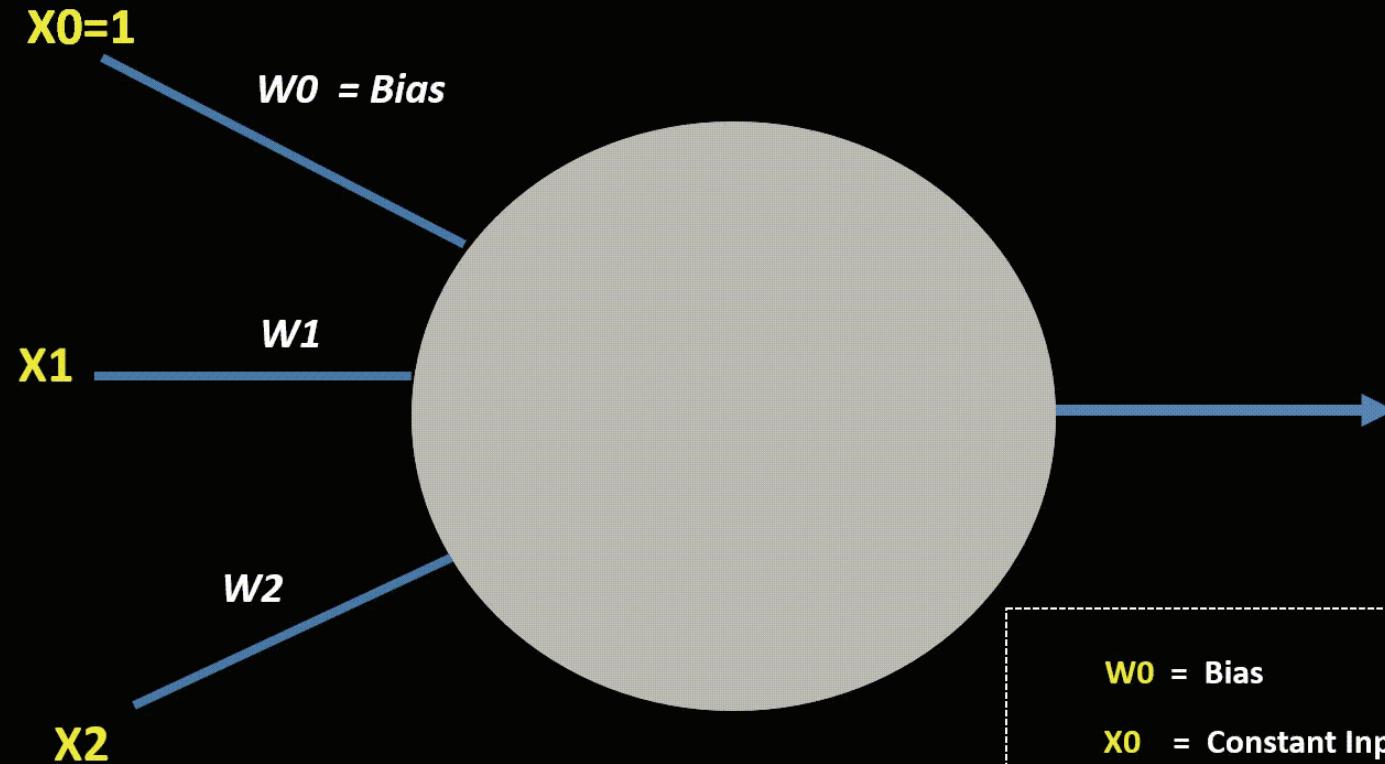




# Why do we need more layers?



# Artificial Neuron



**W0 = Bias**

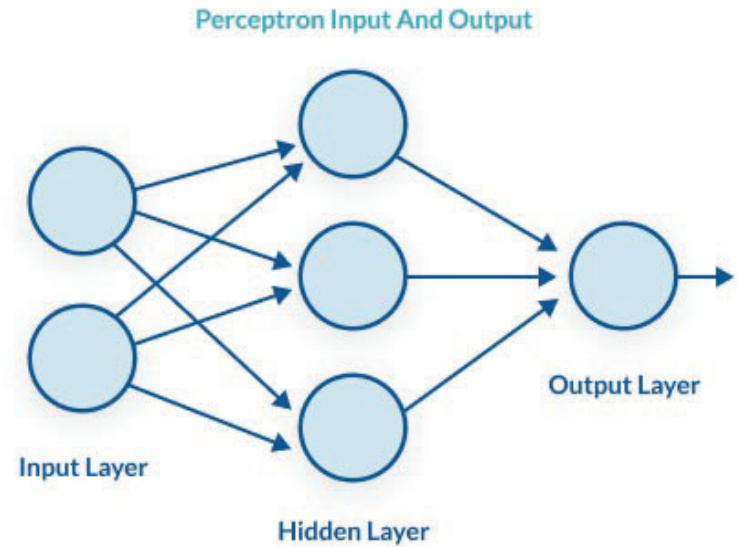
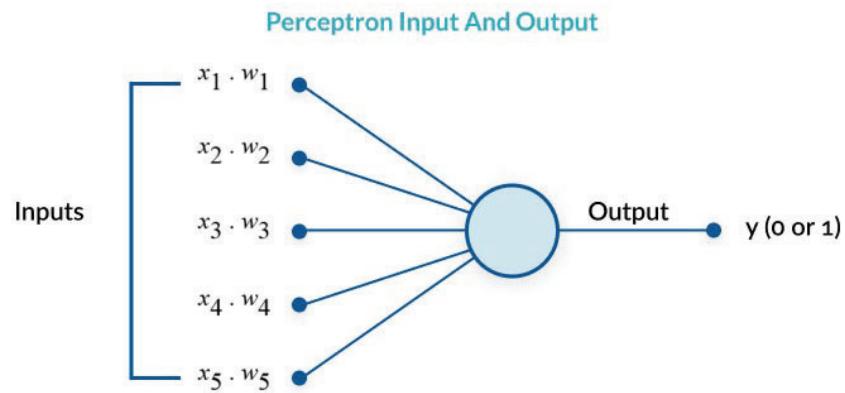
**X0 = Constant Input 1  
for Bias**

**X1,X2 = Attribute Inputs**

**W1,W2 = Weights of Inputs  
X1,X2**



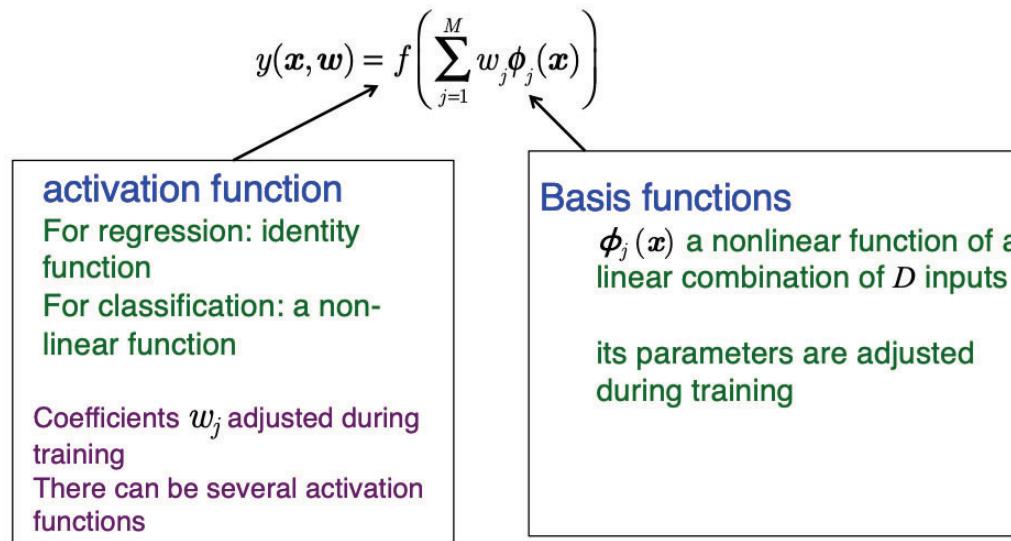
# Perceptron vs Multilayer Perceptron (MLP)





# Feed forward Neural Networks

A neural network can also be represented similar to linear models but basis functions are generalized





# Activation Functions

- Each activation  $a_j$  is transformed using differentiable nonlinear activation functions  $z_j = h(a_j)$
- The  $z_j$  correspond to outputs of basis functions  $\varphi_j(x)$ 
  - or first layer of network or hidden units
- Nonlinear functions  $h$
- Three examples of activation functions:

1. Logistic sigmoid

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

2. Hyperbolic tangent

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

3. Rectified Linear unit

$$f(x) = \max(0, x)$$



# Activation Function

- Determined by the nature of the data and the assumed distribution of the target variables
- For standard regression problems the activation function is the identity function so that  $y_k = a_k$
- For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that  $y_k = \sigma(a_k)$
- For multiclass problems, a softmax activation function of the form:

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$



# Visualization of activation functions

Identity	Sigmoid	TanH	ArcTan
ReLU	Leaky ReLU	Randomized ReLU	Parameteric ReLU
Binary	Exponential Linear Unit	Soft Sign	Inverse Square Root Unit (ISRU)
Inverse Square Root Linear	Square Non-Linearity	Bipolar ReLU	Soft Plus



# Forward Pass

- Output of a layer  $l$  is given by

$$z^{[\ell]} = W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}$$

- Where  $W^l$  is the weights of the layer,  $b^l$  is the bias and  $a^l$  is the activation

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

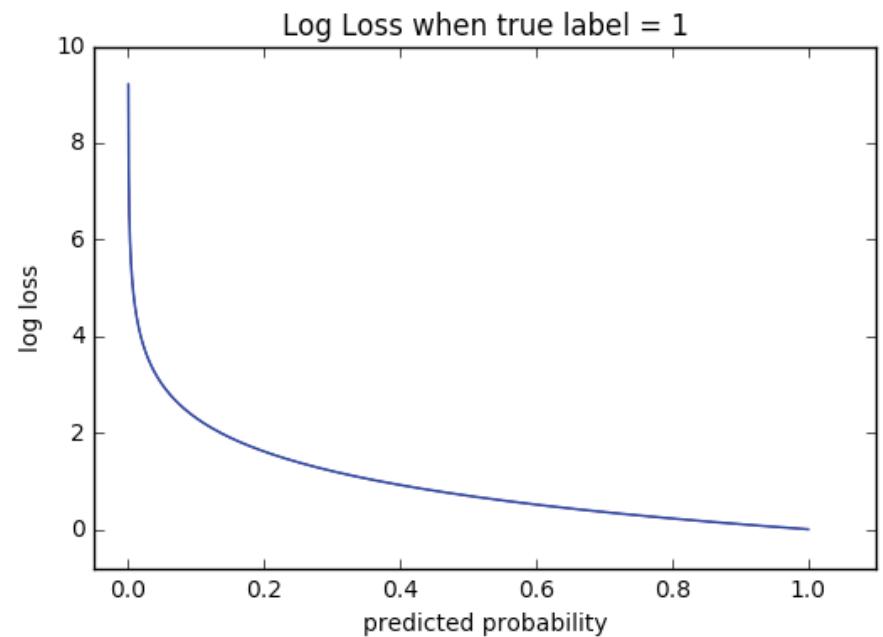


# Loss Function: Log loss

- Measures the performance of a classification model whose output is a probability value between 0 and 1.

$$-(y \log(p) + (1 - y) \log(1 - p))$$

- Cross-entropy loss increases as the predicted probability diverges from the actual label.





# How to train this network?

- Gradient Descent + Backprop!
- What is Backprop?
- Backprop or Backpropagation is a way to train multilayer neural networks using gradients.



# Backpropagation

- Before training the neural network, select an initial value for parameters.
  - Common practice: randomly initialize the parameters to small values (e.g., normally distributed around zero;  $N(0; 0:1)$ )
- Next step: update the parameters.
- After a single forward pass through the neural network, the output will be a predicted value
- We can then compute the loss  $L$ , in our case the log loss

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



# Defining Variables

Let  $X$  be the input features.  $[n \times f]$   
 $y$  be the output(target) labels  $[n \times 1]$

Define the weights [parameters] of the neural network

First layer  $\rightarrow W_1$ , bias  $\rightarrow b_1$

Second layer  $\rightarrow W_2$ , bias  $\rightarrow b_2$

Third layer  $\rightarrow W_3$ , bias  $\rightarrow b_3$



# The forward pass

The output of layer 1 is

$$z_1 = w_1 \cdot x + b_1$$

$a_1 = g(z_1)$  where "g" is the activation function  
The output of layer 2 is

$$z_2 = w_2 \cdot a_1 + b_2$$

$$a_2 = g(z_2)$$

The output of layer 3 is

$$z_3 = w_3 \cdot a_2 + b_3$$

$$a_3 = g(z_3)$$

The o/p of the neural network is

$$\hat{y} = a_3$$



## Defining the loss

Task : Binary classification

⇒ loss is log loss or binary cross entropy

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



# Learning with gradient descent

To learn the parameters

① Provide random values for weights  
 $w_1, w_2, w_3$  & biases  $b_1, b_2, b_3$

② Update the weights using gradient descent by

$$w_i := w_i - \alpha \cdot \frac{\partial L}{\partial w_i}$$

$$b_i := b_i - \alpha \cdot \frac{\partial L}{\partial b_i}$$

where  $i$  refers to the  $i$ th layer.

③ repeat until convergence



## Finding $dL/dW_3$

To find  $\frac{\partial L}{\partial w_3}$  to update the third layer.

$$\frac{\partial L}{\partial w_3} = \frac{\partial}{\partial w_3} \left[ -[(1-y) \cdot \log(1-\hat{y}) + y \cdot \log(\hat{y})] \right]$$

$$= -[(1-y) \cdot \frac{\partial}{\partial w_3} (\log(1-\hat{y})) + y \cdot \frac{\partial}{\partial w_3} (\log(\hat{y}))]$$



# Finding $dL/dW_3$

Remember:  $\frac{\partial}{\partial z} (\log(z)) = \frac{1}{z}$



## Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = - \left[ \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial}{\partial w_3} (1-\hat{y}) + \frac{y}{\hat{y}} \cdot \frac{\partial}{\partial w_3} (\hat{y}) \right]$$

$$= - \left[ \frac{-(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} + \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3} \right]$$

$$= \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} - \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \left[ \frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}} \right] \cdot \frac{\partial \hat{y}}{\partial w_3}$$



## Finding $dL/dW_3$

$$\Rightarrow \frac{\partial L}{\partial w_3} = \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$



# Finding $dL/dW_3$

we know that  $\hat{y} = a_3 = g(z_3)$



## Finding $dL/dW_3$

if  $g(z)$  is a sigmoid function,

$$g'(z) = g(z) \cdot (1-g(z))$$

$$\begin{aligned}\therefore \frac{\partial \hat{y}}{\partial w_3} &= \frac{\partial a_3}{\partial w_3} = \frac{\partial \cdot g(z_3)}{\partial w_3} \\ &= g(z_3) \cdot (1-g(z_3)) \cdot \frac{\partial z_3}{\partial w_3} \\ &= a_3 (1-a_3) \cdot \frac{\partial [w_3 a_2 + b_3]}{\partial w_3} \\ &= a_3 (1-a_3) \cdot a_2 \\ &= \hat{y} (1-\hat{y}) \cdot a_2\end{aligned}$$



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Similarly,

$$\boxed{\frac{\partial L}{\partial b_3} = a_3 - y}$$



# Finding $dL/dW_2$

- We need to use the chain rule from calculus.
- Why?
- There is no direct relationship between the weights  $W_2$  and the loss  $L$ .
- You cannot differentiate a variable by another if there is no direct relationship
  - Else it would be 0
  - Not true if the variable/function is composite



## Finding $dL/dW_2$

We know that Loss is dependent on  $\hat{y} = a_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial ?} \cdot \frac{?}{\partial w_2}$$

We know that  $a_3 = g(z_3)$

$\Rightarrow a_3$  is dependent on  $z_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial ?} \cdot \frac{?}{\partial w_2}$$



## Finding $dL/dW_2$

We know that  $z_3 = w_3 a_2 + b_3$

$\Rightarrow z_3$  is dependent on  $a_2$

$$\Rightarrow \frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta a_3} \cdot \frac{\delta a_3}{\delta z_3} \cdot \frac{\delta z_3}{\delta a_2} \cdot \frac{\delta a_2}{?} \cdot \frac{?}{\delta w_2}$$



## Finding $dL/dW_2$

But  $a_2 = g(z_2)$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} \cdot ?$$

But  $z_2 = w_2 \cdot a_1 + b_2$

$$\therefore \boxed{\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}}$$



## Finding $dL/dW_2$

If we rewrite  $\frac{\partial L}{\partial w_3}$  using chain rule,

$$\frac{\partial L}{\partial w_3} = \underbrace{\frac{\partial L}{\partial a_3}}_{\cdot a_3 - y} \cdot \underbrace{\frac{\partial a_3}{\partial z_3}}_{a_2^T} \cdot \frac{\partial z_3}{\partial w_3}$$

Since  
 $z_3 = w_3 a_2 + b_3$



# Finding $dL/dW_2$

$$\Rightarrow \frac{\partial L}{\partial w_2} = (a_3 - y) \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

if  $a_2 = g(z_2)$

then  $\frac{\partial a_2}{\partial (z_2)} = g'(z_2)$

If  $z_3 = w_3 \cdot a_2 + b_3$

Then  $\frac{\partial z_3}{\partial a_2} = w_3$

If  $z_2 = w_2 \cdot a_1 + b_2$

then  $\frac{\partial z_2}{\partial w_2} = a_1$



# Finding $dL/dW_2$

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot a_1}$$

Re-arranging for vectorized,

$$\boxed{\frac{\partial L}{\partial w_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y) \cdot a_1^T}$$

$$\boxed{\frac{\partial L}{\partial b_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y)}$$



# Finding $dL/dW_1$

Use the chain rule!

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

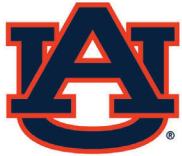
$$\boxed{\frac{\partial L}{\partial w_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1) \cdot x}$$

$$\boxed{\text{III}^{ly}} \quad \frac{\partial L}{\partial b_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1)$$



# COMP [56]630– Machine Learning

Lecture 9 – Neural Networks



# Perceptron Algorithm



# The Perceptron

- Consider modifying the logistic regression method to “force” it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of  $g$  to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

- If we then let  $h(x) = g(\theta^T x)$  as before but using this modified definition of  $g$ , and if we use the update rule

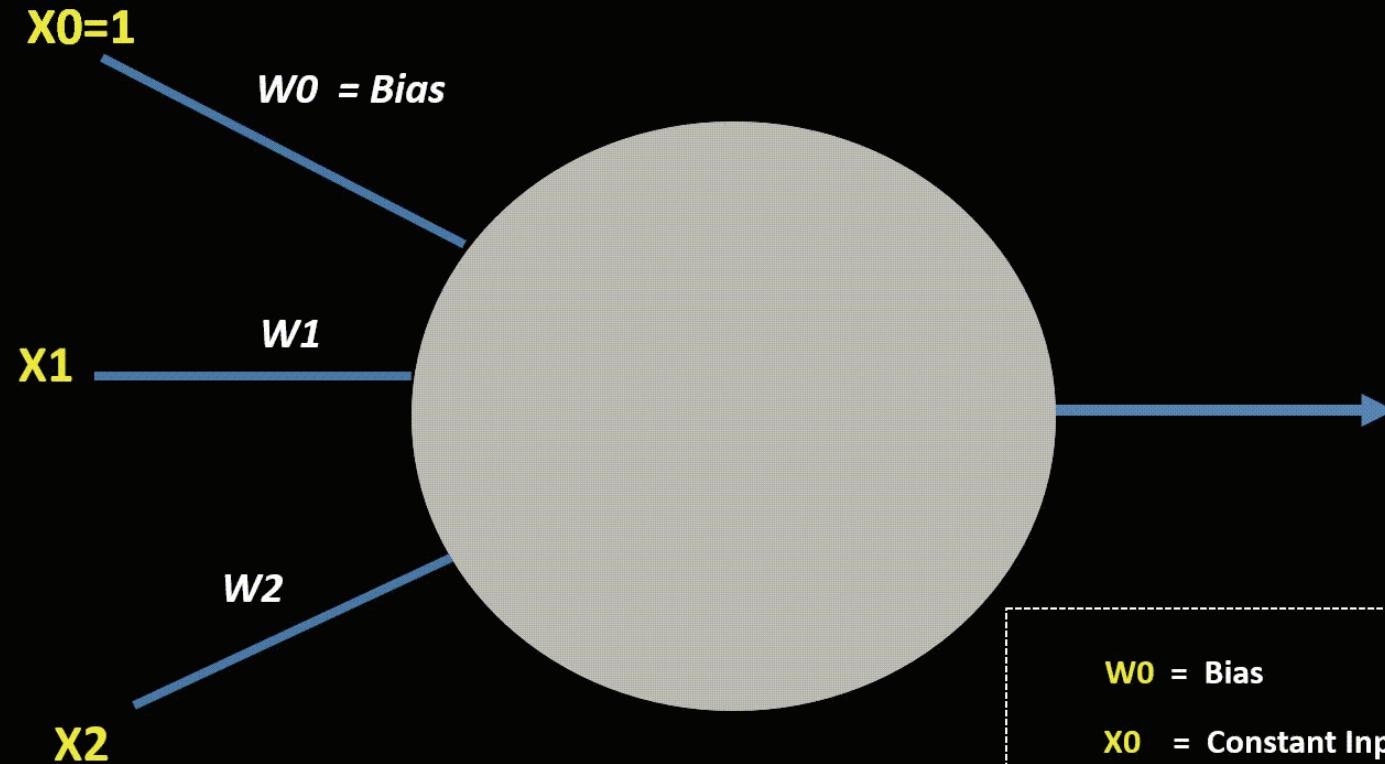
$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$



# Perceptron

- In the 1960s, this “perceptron” was argued to be a rough model for how individual neurons in the brain work.
- Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression.
  - The hypothesis in logistic regression provides a measure of uncertainty in the occurrence of a binary outcome based on a linear model.
  - The output from a step function can of course not be interpreted as any kind of probability.
  - Since a step function is not differentiable, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.

# Artificial Neuron



**W0 = Bias**

**X0 = Constant Input 1  
for Bias**

**X1,X2 = Attribute Inputs**

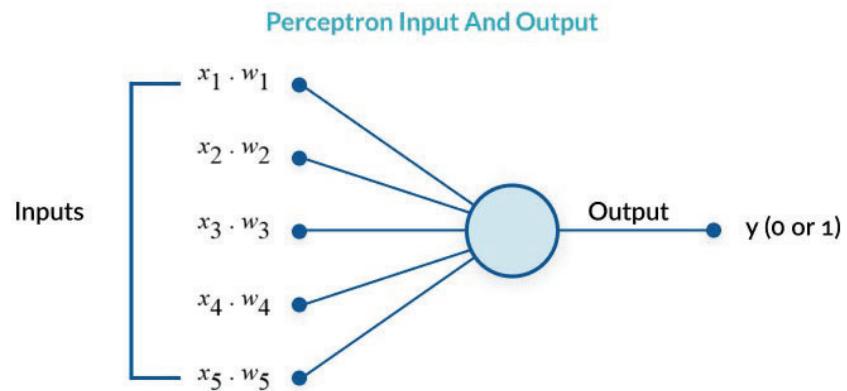
**W1,W2 = Weights of Inputs  
X1,X2**



# Multi-layer perceptron



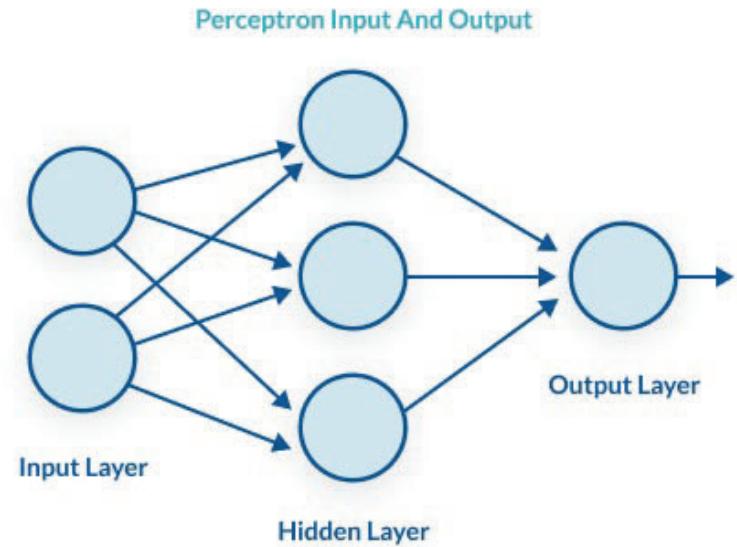
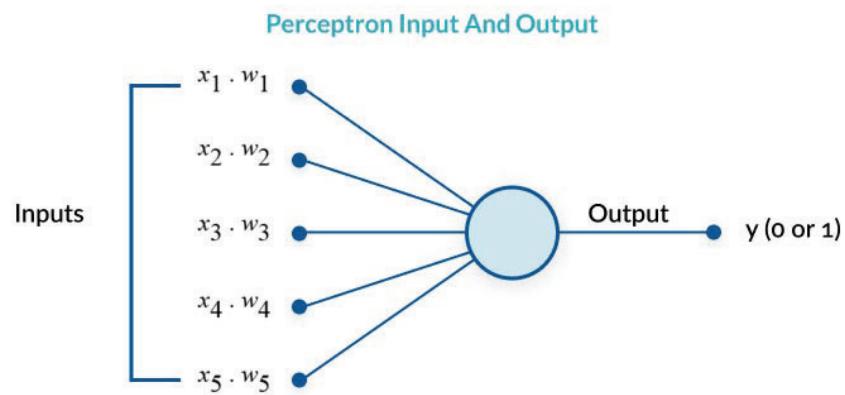
# Perceptron vs Multilayer Perceptron (MLP)





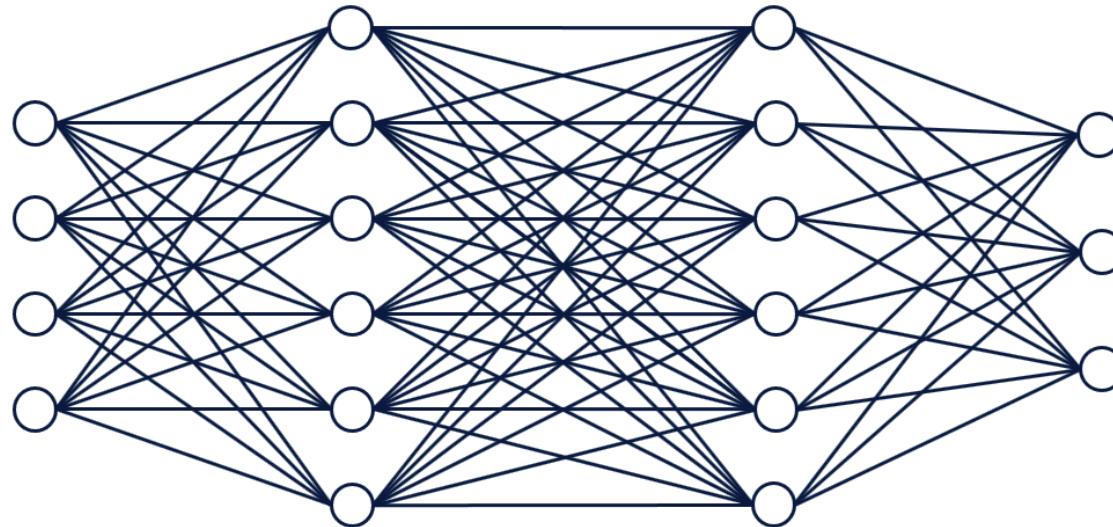


# Perceptron vs Multilayer Perceptron (MLP)





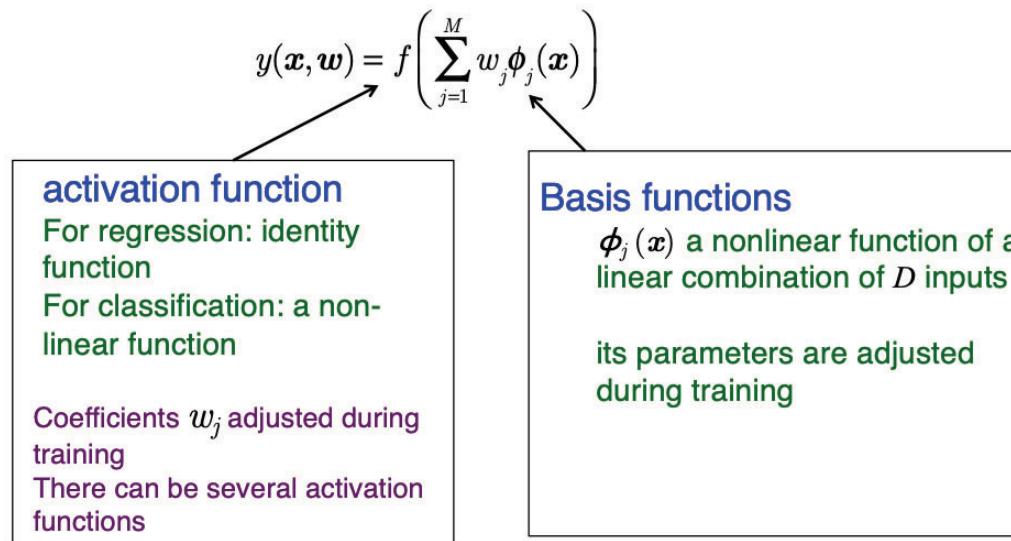
# Why do we need more layers?





# Feed forward Neural Networks

A neural network can also be represented similar to linear models but basis functions are generalized





# Activation Functions

- Each activation  $a_j$  is transformed using differentiable nonlinear activation functions  $z_j = h(a_j)$
- The  $z_j$  correspond to outputs of basis functions  $\varphi_j(x)$ 
  - or first layer of network or hidden units
- Nonlinear functions  $h$
- Three examples of activation functions:

1. Logistic sigmoid

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

2. Hyperbolic tangent

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

3. Rectified Linear unit

$$f(x) = \max(0, x)$$



# Activation Function

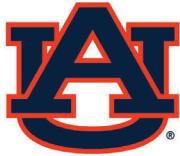
- Determined by the nature of the data and the assumed distribution of the target variables
- For standard regression problems the activation function is the identity function so that  $y_k = a_k$
- For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that  $y_k = \sigma(a_k)$
- For multiclass problems, a softmax activation function of the form:

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$



# Visualization of activation functions

Identity	Sigmoid	TanH	ArcTan
ReLU	Leaky ReLU	Randomized ReLU	Parameteric ReLU
Binary	Exponential Linear Unit	Soft Sign	Inverse Square Root Unit (ISRU)
Inverse Square Root Linear	Square Non-Linearity	Bipolar ReLU	Soft Plus



# Forward Pass

- Output of a layer  $l$  is given by

$$z^{[\ell]} = W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}$$

- Where  $W^l$  is the weights of the layer,  $b^l$  is the bias and  $a^l$  is the activation

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

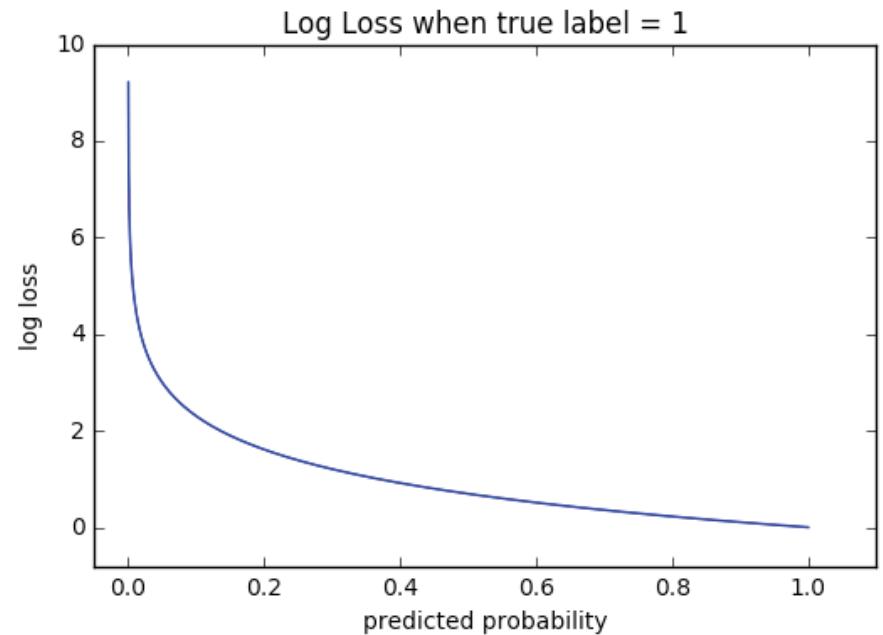


# Loss Function: Log loss

- Measures the performance of a classification model whose output is a probability value between 0 and 1.

$$-(y \log(p) + (1 - y) \log(1 - p))$$

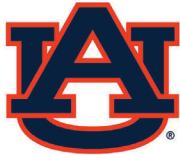
- Cross-entropy loss increases as the predicted probability diverges from the actual label.





# How to train this network?

- Gradient Descent + Backprop!
- What is Backprop?
- Backprop or Backpropagation is a way to train multilayer neural networks using gradients.



# Backpropagation

- Before training the neural network, select an initial value for parameters.
  - Common practice: randomly initialize the parameters to small values (e.g., normally distributed around zero;  $N(0; 0:1)$ )
- Next step: update the parameters.
- After a single forward pass through the neural network, the output will be a predicted value
- We can then compute the loss  $L$ , in our case the log loss

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



# Defining Variables

Let  $X$  be the input features.  $[n \times f]$   
 $y$  be the output(target) labels  $[n \times 1]$

Define the weights [parameters] of the neural network

First layer  $\rightarrow W_1$ , bias  $\rightarrow b_1$

Second layer  $\rightarrow W_2$ , bias  $\rightarrow b_2$

Third layer  $\rightarrow W_3$ , bias  $\rightarrow b_3$



# The forward pass

The output of layer 1 is

$$z_1 = w_1 \cdot x + b_1$$

$a_1 = g(z_1)$  where "g" is the activation function  
The output of layer 2 is

$$z_2 = w_2 \cdot a_1 + b_2$$
$$a_2 = g(z_2)$$

The output of layer 3 is

$$z_3 = w_3 \cdot a_2 + b_3$$
$$a_3 = g(z_3)$$

The o/p of the neural network is

$$\hat{y} = a_3$$



## Defining the loss

Task : Binary classification

⇒ loss is log loss or binary cross entropy

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



# Learning with gradient descent

To learn the parameters

① Provide random values for weights  
 $w_1, w_2, w_3$  & biases  $b_1, b_2, b_3$

② Update the weights using gradient descent by

$$w_i := w_i - \alpha \cdot \frac{\partial L}{\partial w_i}$$

$$b_i := b_i - \alpha \cdot \frac{\partial L}{\partial b_i}$$

where  $i$  refers to the  $i$ th layer.

③ repeat until convergence



## Finding $dL/dW_3$

To find  $\frac{\partial L}{\partial w_3}$  to update the third layer.

$$\frac{\partial L}{\partial w_3} = \frac{\partial}{\partial w_3} \left[ -[(1-y) \cdot \log(1-\hat{y}) + y \cdot \log(\hat{y})] \right]$$

$$= -[(1-y) \cdot \frac{\partial}{\partial w_3} (\log(1-\hat{y})) + y \cdot \frac{\partial}{\partial w_3} (\log(\hat{y}))]$$



# Finding $dL/dW_3$

Remember:  $\frac{\partial}{\partial z} (\log(z)) = \frac{1}{z}$



## Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = - \left[ \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial}{\partial w_3} (1-\hat{y}) + \frac{y}{\hat{y}} \cdot \frac{\partial}{\partial w_3} (\hat{y}) \right]$$

$$= - \left[ \frac{-(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} + \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3} \right]$$

$$= \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} - \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \left[ \frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}} \right] \cdot \frac{\partial \hat{y}}{\partial w_3}$$



## Finding $dL/dW_3$

$$\Rightarrow \frac{\partial L}{\partial w_3} = \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$



# Finding $dL/dW_3$

we know that  $\hat{y} = a_3 = g(z_3)$



# Finding $dL/dW_3$

if  $g(z)$  is a sigmoid function,

$$g'(z) = g(z) \cdot (1-g(z))$$

$$\begin{aligned}\therefore \frac{\partial \hat{y}}{\partial w_3} &= \frac{\partial a_3}{\partial w_3} = \frac{\partial \cdot g(z_3)}{\partial w_3} \\ &= g(z_3) \cdot (1-g(z_3)) \cdot \frac{\partial z_3}{\partial w_3} \\ &= a_3 (1-a_3) \cdot \frac{\partial [w_3 a_2 + b_3]}{\partial w_3} \\ &= a_3 (1-a_3) \cdot a_2 \\ &= \hat{y} (1-\hat{y}) \cdot a_2\end{aligned}$$



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Similarly,

$$\boxed{\frac{\partial L}{\partial b_3} = a_3 - y}$$



# Finding $dL/dW_2$

- We need to use the chain rule from calculus.
- Why?
- There is no direct relationship between the weights  $W_2$  and the loss  $L$ .
- You cannot differentiate a variable by another if there is no direct relationship
  - Else it would be 0
  - Not true if the variable/function is composite



## Finding $dL/dW_2$

We know that Loss is dependent on  $\hat{y} = a_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial ?} \cdot \frac{?}{\partial w_2}$$

We know that  $a_3 = g(z_3)$

$\Rightarrow a_3$  is dependent on  $z_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial ?} \cdot \frac{?}{\partial w_2}$$



## Finding $dL/dW_2$

We know that  $z_3 = w_3 a_2 + b_3$

$\Rightarrow z_3$  is dependent on  $a_2$

$$\Rightarrow \frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta a_3} \cdot \frac{\delta a_3}{\delta z_3} \cdot \frac{\delta z_3}{\delta a_2} \cdot \frac{\delta a_2}{?} \cdot \frac{?}{\delta w_2}$$



## Finding $dL/dW_2$

But  $a_2 = g(z_2)$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} \cdot ?$$

But  $z_2 = w_2 \cdot a_1 + b_2$

$$\therefore \boxed{\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}}$$



## Finding $dL/dW_2$

If we rewrite  $\frac{\partial L}{\partial w_3}$  using chain rule,

$$\frac{\partial L}{\partial w_3} = \underbrace{\frac{\partial L}{\partial a_3}}_{\cdot a_3 - y} \cdot \underbrace{\frac{\partial a_3}{\partial z_3}}_{a_2^T} \cdot \frac{\partial z_3}{\partial w_3}$$

Since  
 $z_3 = w_3 a_2 + b_3$



# Finding $dL/dW_2$

$$\Rightarrow \frac{\partial L}{\partial w_2} = (a_3 - y) \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

if  $a_2 = g(z_2)$

then  $\frac{\partial a_2}{\partial (z_2)} = g'(z_2)$

If  $z_3 = w_3 \cdot a_2 + b_3$

Then  $\frac{\partial z_3}{\partial a_2} = w_3$

If  $z_2 = w_2 \cdot a_1 + b_2$

then  $\frac{\partial z_2}{\partial w_2} = a_1$



# Finding $dL/dW_2$

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot a_1}$$

Re-arranging for vectorized,

$$\boxed{\frac{\partial L}{\partial w_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y) \cdot a_1^T}$$

$$\frac{\partial L}{\partial b_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y)$$



# Finding $dL/dW_1$

Use the chain rule!

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial L}{\partial w_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1) \cdot x}$$

$$\boxed{\text{III}^{ly}} \quad \frac{\partial L}{\partial b_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1)$$



# NumPy Demo



# Regularization



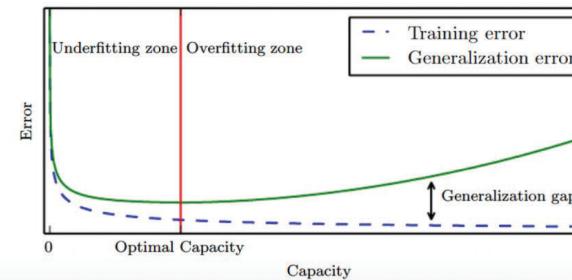
# What is regularization?

## Generalization error

- Performance on inputs not previously seen
  - Also called as *Test error*

## Regularization is:

- “any modification to a learning algorithm to reduce its *generalization error* but not its *training error*”
- Reduce generalization error even at the expense of increasing training error
  - E.g., Limiting model capacity is a regularization method





# Goals of Regularization

Some goals of regularization

1. Encode prior knowledge
2. Express preference for simpler model
3. Need to make underdetermined problem determined



# Why regularization?

Generalization

- Prevent over-fitting

Occam's razor

Bayesian point of view

- Regularization corresponds to prior distributions on model parameters



# Limiting Number of Neurons

No. of input/output units determined by dimensions

Number of hidden units  $M$  is a free parameter

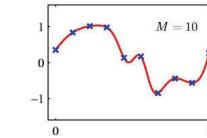
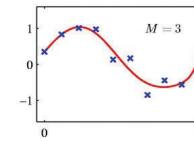
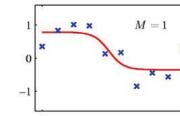
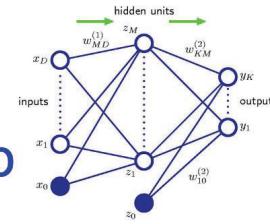
- Adjusted to get best predictive performance

Possible approach is to get maximum likelihood estimate of  $M$  for balance between under-fitting and over-fitting



# Limiting Number of Neurons

Sinusoidal Regression Prob



Two layer network trained on 10 data points

$M = 1, 3$  and  $10$  hidden units

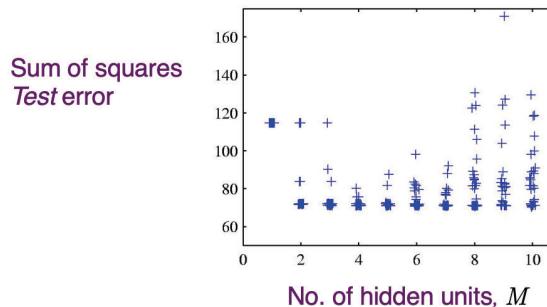
Minimizing sum-of-squared error function  
Using conjugate gradient descent

Generalization error is not a simple function of  $M$   
due to presence of local minima in error function



# Using Validation Set to fix no. of hidden units

Plot a graph choosing random starts and different numbers of hidden units  $M$  and choose the specific solution having smallest generalization error



- 30 random starts for each  $M$   
30 points in each column of graph
- Overall best validation set performance happened at  $M=8$

There are other ways to control the complexity of a neural network in order to avoid over-fitting

Alternative approach is to choose a relatively large value of  $M$  and then control complexity by adding a regularization term



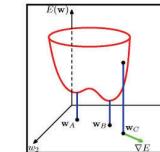
# Parameter Norm Penalty

Generalization error not a simple function of  $M$

- Due to presence of local minima
- Need to control capacity to avoid over-fitting
  - Alternatively choose large  $M$  and control complexity by addition of regularization term

Simplest regularizer is *weight decay*

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$



- Effective model complexity determined by choice of  $\lambda$
- Regularizer is equivalent to a Gaussian prior over  $\mathbf{w}$

In a 2-layer neural network

$$J_{regularized} = \underbrace{-\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)}))}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum_l \sum_k \sum_j W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$



# Weight Decay

The name weight decay is due to the following

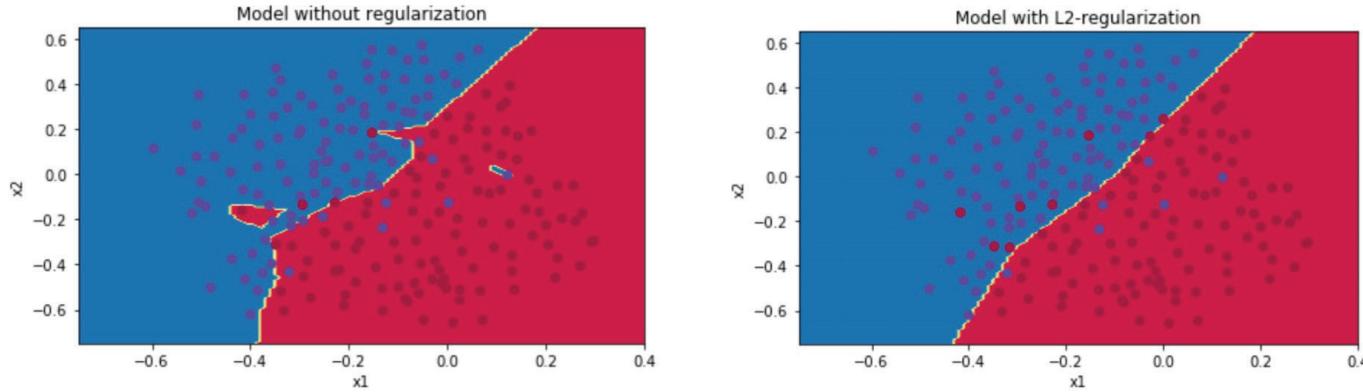
$$w_{t+1} = w_t - \alpha \nabla_w J - \lambda w_t$$

To prevent overfitting, every time we update a weight  $w$  with the gradient  $\nabla J$  in respect to  $w$ , we also subtract from it  $\lambda w$ .

This gives the weights a tendency to decay towards zero, hence the name.



# Effect of Regularization



$$J_{regularized} = \underbrace{-\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)}))}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum_l \sum_k \sum_j W_{kj}^{[l]2}}_{\text{L2 regularization cost}}$$

[https://medium.com/@shay.palachy/understanding-the-scaling-of-l2-regularization  
in-the-context-of-neural-networks-e3d25f8b50db?fbclid=IwAR17rKfnM2JnkWBOZFvQg6ftl0k52diyEra\\_66UCwI18Gtf7nFd6l3l4VMY](https://medium.com/@shay.palachy/understanding-the-scaling-of-l2-regularization-in-the-context-of-neural-networks-e3d25f8b50db?fbclid=IwAR17rKfnM2JnkWBOZFvQg6ftl0k52diyEra_66UCwI18Gtf7nFd6l3l4VMY)



# COMP [56]630– Machine Learning

Lecture 10 – Backprop (revisited), Demo, Regularization, Intro to DL



# Backpropagation

- Before training the neural network, select an initial value for parameters.
  - Common practice: randomly initialize the parameters to small values (e.g., normally distributed around zero;  $N(0; 0:1)$ )
- Next step: update the parameters.
- After a single forward pass through the neural network, the output will be a predicted value
- We can then compute the loss  $L$ , in our case the log loss

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



# Defining Variables

Let  $X$  be the input features.  $[n \times f]$   
 $y$  be the output(target) labels  $[n \times 1]$

Define the weights [parameters] of the neural network

First layer  $\rightarrow W_1$ , bias  $\rightarrow b_1$

Second layer  $\rightarrow W_2$ , bias  $\rightarrow b_2$

Third layer  $\rightarrow W_3$ , bias  $\rightarrow b_3$



# The forward pass

The output of layer 1 is

$$z_1 = w_1 \cdot x + b_1$$

$a_1 = g(z_1)$  where "g" is the activation function  
The output of layer 2 is

$$z_2 = w_2 \cdot a_1 + b_2$$

$$a_2 = g(z_2)$$

The output of layer 3 is

$$z_3 = w_3 \cdot a_2 + b_3$$

$$a_3 = g(z_3)$$

The o/p of the neural network is

$$\hat{y} = a_3$$



## Defining the loss

Task : Binary classification

⇒ loss is log loss or binary cross entropy

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



# Learning with gradient descent

To learn the parameters

① Provide random values for weights  
 $w_1, w_2, w_3$  & biases  $b_1, b_2, b_3$

② Update the weights using gradient descent by

$$w_i := w_i - \alpha \cdot \frac{\partial L}{\partial w_i}$$

$$b_i := b_i - \alpha \cdot \frac{\partial L}{\partial b_i}$$

where  $i$  refers to the  $i$ th layer.

③ repeat until convergence



## Finding $dL/dW_3$

To find  $\frac{\partial L}{\partial w_3}$  to update the third layer.

$$\frac{\partial L}{\partial w_3} = \frac{\partial}{\partial w_3} \left[ -[(1-y) \cdot \log(1-\hat{y}) + y \cdot \log(\hat{y})] \right]$$

$$= -[(1-y) \cdot \frac{\partial}{\partial w_3} (\log(1-\hat{y})) + y \cdot \frac{\partial}{\partial w_3} (\log(\hat{y}))]$$



# Finding $dL/dW_3$

Remember:  $\frac{\partial}{\partial z} (\log(z)) = \frac{1}{z}$



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = - \left[ \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial}{\partial w_3} (1-\hat{y}) + \frac{y}{\hat{y}} \cdot \frac{\partial}{\partial w_3} (\hat{y}) \right]$$

$$= - \left[ \frac{-(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} + \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3} \right]$$

$$= \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} - \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \left[ \frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}} \right] \cdot \frac{\partial \hat{y}}{\partial w_3}$$



## Finding $dL/dW_3$

$$\Rightarrow \frac{\partial L}{\partial w_3} = \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$



# Finding $dL/dW_3$

we know that  $\hat{y} = a_3 = g(z_3)$



# Finding $dL/dW_3$

if  $g(z)$  is a sigmoid function,

$$g'(z) = g(z) \cdot (1-g(z))$$

$$\begin{aligned}\therefore \frac{\partial \hat{y}}{\partial w_3} &= \frac{\partial a_3}{\partial w_3} = \frac{\partial \cdot g(z_3)}{\partial w_3} \\ &= g(z_3) \cdot (1-g(z_3)) \cdot \frac{\partial z_3}{\partial w_3} \\ &= a_3 (1-a_3) \cdot \frac{\partial [w_3 a_2 + b_3]}{\partial w_3} \\ &= a_3 (1-a_3) \cdot a_2 \\ &= \hat{y} (1-\hat{y}) \cdot a_2\end{aligned}$$



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.



# Finding $dL/dW_3$

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Similarly,

$$\boxed{\frac{\partial L}{\partial b_3} = a_3 - y}$$



# Finding $dL/dW_2$

- We need to use the chain rule from calculus.
- Why?
- There is no direct relationship between the weights  $W_2$  and the loss  $L$ .
- You cannot differentiate a variable by another if there is no direct relationship
  - Else it would be 0
  - Not true if the variable/function is composite



## Finding $dL/dW_2$

We know that Loss is dependent on  $\hat{y} = a_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{?} \cdot \frac{?}{\partial w_2}$$

We know that  $a_3 = g(z_3)$

$\Rightarrow a_3$  is dependent on  $z_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{?} \cdot \frac{?}{\partial w_2}$$



## Finding $dL/dW_2$

We know that  $Z_3 = w_3 a_2 + b_3$

$\Rightarrow Z_3$  is dependent on  $a_2$

$$\Rightarrow \frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta a_3} \cdot \frac{\delta a_3}{\delta Z_3} \cdot \frac{\delta Z_3}{\delta a_2} \cdot \frac{\delta a_2}{?} \cdot \frac{?}{\delta w_2}$$



## Finding $dL/dW_2$

But  $a_2 = g(z_2)$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2} \cdot ?$$

But  $z_2 = w_2 \cdot a_1 + b_2$

$$\therefore \boxed{\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}}$$



## Finding $dL/dW_2$

If we rewrite  $\frac{\partial L}{\partial w_3}$  using chain rule,

$$\frac{\partial L}{\partial w_3} = \underbrace{\frac{\partial L}{\partial a_3}}_{\cdot a_3 - y} \cdot \underbrace{\frac{\partial a_3}{\partial z_3}}_{a_2^T} \cdot \frac{\partial z_3}{\partial w_3}$$

Since  
 $z_3 = w_3 a_2 + b_3$



# Finding $dL/dW_2$

$$\Rightarrow \frac{\partial L}{\partial w_2} = (a_3 - y) \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

if  $a_2 = g(z_2)$

then  $\frac{\partial a_2}{\partial (z_2)} = g'(z_2)$

If  $z_3 = w_3 \cdot a_2 + b_3$

Then  $\frac{\partial z_3}{\partial a_2} = w_3$

If  $z_2 = w_2 \cdot a_1 + b_2$

then  $\frac{\partial z_2}{\partial w_2} = a_1$



# Finding $dL/dW_2$

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot a_1}$$

Re-arranging for vectorized,

$$\boxed{\frac{\partial L}{\partial w_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y) \cdot a_1^T}$$

$$\boxed{\frac{\partial L}{\partial b_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y)}$$



# Finding $dL/dW_1$

Use the chain rule!

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$\boxed{\frac{\partial L}{\partial w_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1) \cdot x}$$

$$\boxed{\text{III}^{ly}} \quad \frac{\partial L}{\partial b_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1)$$



# NumPy Demo



# Regularization



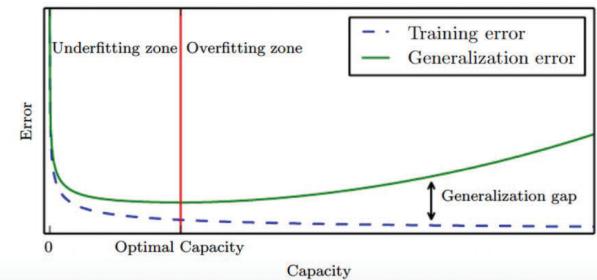
# What is regularization?

## Generalization error

- Performance on inputs not previously seen
  - Also called as *Test error*

## Regularization is:

- “any modification to a learning algorithm to reduce its *generalization error* but not its *training error*”
- Reduce generalization error even at the expense of increasing training error
  - E.g., Limiting model capacity is a regularization method





# Goals of Regularization

Some goals of regularization

1. Encode prior knowledge
2. Express preference for simpler model
3. Need to make underdetermined problem determined



# Why regularization?

Generalization

- Prevent over-fitting

Occam's razor

Bayesian point of view

- Regularization corresponds to prior distributions on model parameters



# Limiting Number of Neurons

No. of input/output units determined by dimensions

Number of hidden units  $M$  is a free parameter

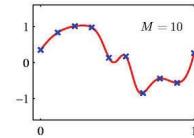
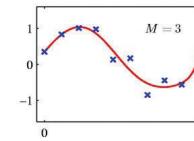
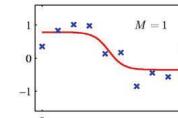
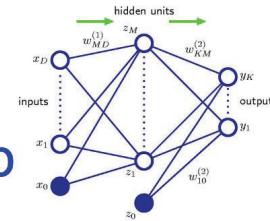
- Adjusted to get best predictive performance

Possible approach is to get maximum likelihood estimate of  $M$  for balance between under-fitting and over-fitting



# Limiting Number of Neurons

Sinusoidal Regression Prob



Two layer network trained on 10 data points

$M = 1, 3$  and  $10$  hidden units

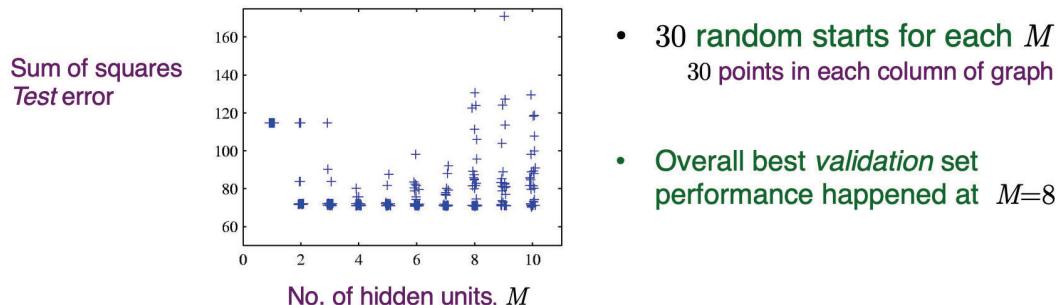
Minimizing sum-of-squared error function  
Using conjugate gradient descent

Generalization error is not a simple function of  $M$   
due to presence of local minima in error function



# Using Validation Set to fix no. of hidden units

Plot a graph choosing random starts and different numbers of hidden units  $M$  and choose the specific solution having smallest generalization error



- 30 random starts for each  $M$   
30 points in each column of graph
- Overall best validation set performance happened at  $M=8$

There are other ways to control the complexity of a neural network in order to avoid over-fitting

Alternative approach is to choose a relatively large value of  $M$  and then control complexity by adding a regularization term



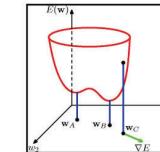
# Parameter Norm Penalty

Generalization error not a simple function of  $M$

- Due to presence of local minima
- Need to control capacity to avoid over-fitting
  - Alternatively choose large  $M$  and control complexity by addition of regularization term

Simplest regularizer is *weight decay*

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$



- Effective model complexity determined by choice of  $\lambda$
- Regularizer is equivalent to a Gaussian prior over  $\mathbf{w}$

In a 2-layer neural network

$$J_{regularized} = \underbrace{-\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)}))}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum_l \sum_k \sum_j W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$



# Weight Decay

The name weight decay is due to the following

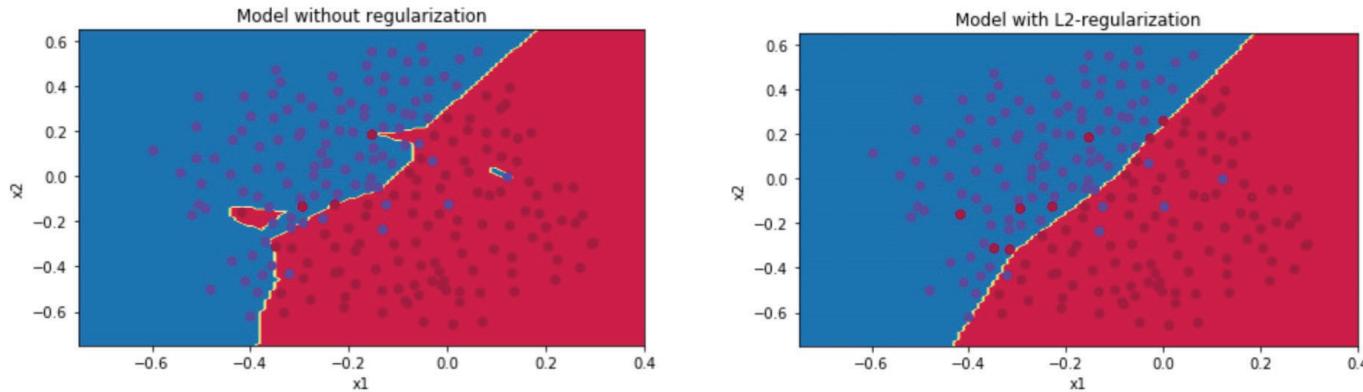
$$w_{t+1} = w_t - \alpha \nabla_w J - \lambda w_t$$

To prevent overfitting, every time we update a weight  $w$  with the gradient  $\nabla J$  in respect to  $w$ , we also subtract from it  $\lambda w$ .

This gives the weights a tendency to decay towards zero, hence the name.



# Effect of Regularization



$$J_{regularized} = \underbrace{-\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)}))}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m} \frac{\lambda}{2} \sum_l \sum_k \sum_j W_{kj}^{[l]2}}_{\text{L2 regularization cost}}$$

[https://medium.com/@shay.palachy/understanding-the-scaling-of-l2-regularization  
in-the-context-of-neural-networks-e3d25f8b50db?fbclid=IwAR17rKfnM2JnkWBOZFvQg6ftl0k52diyEra\\_66UCwI18Gtf7nFd6l3l4VMY](https://medium.com/@shay.palachy/understanding-the-scaling-of-l2-regularization-in-the-context-of-neural-networks-e3d25f8b50db?fbclid=IwAR17rKfnM2JnkWBOZFvQg6ftl0k52diyEra_66UCwI18Gtf7nFd6l3l4VMY)



# Deep Neural Networks



# What is Deep Learning?

1. Computational models composed of multiple processing layers
  - To learn representations of data with multiple levels of abstraction
2. Dramatically improved state-of-the-art in:
  - Speech recognition, Visual object recognition, Object detection
  - Other domains: Drug discovery, Genomics
3. Discovers intricate structure in large data sets
  - Using backpropagation to change parameters
  - Compute representation in each layer from previous layer
4. Deep convolutional nets: image proc, video, speech
5. Recurrent nets: sequential data, e,g., text, speech



# Limitations of Conventional ML

- Limited in ability to process natural data in raw form
- Pattern Recognition and Machine Learning systems require careful engineering and domain expertise to transform raw data, e.g., pixel values, into a feature vector for a classifier



# Automatic Representation Learning

- Methods that allow a machine to be fed with raw data to automatically discover representations needed for detection or classification
- Deep Learning methods are Representation Learning Methods
- Use multiple levels of representation
  - Composing simple but non-linear modules that transform representation at one level (starting with raw input) into a representation at a higher slightly more abstract level
  - Complex functions can be learned
  - Higher layers of representation amplify aspects of input important for discrimination and suppress irrelevant variations



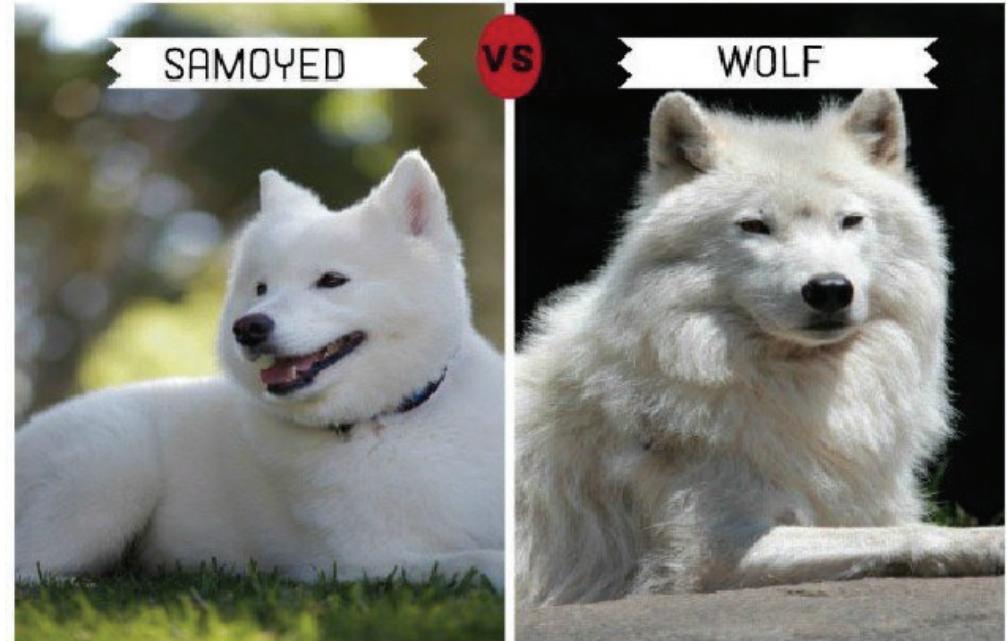
# Example: Images

- Input is an array of pixel values
  - First stage is presence or absence of edges at particular locations and orientations of image
  - Second layer detects motifs by spotting particular arrangements of edges, regardless of small variations in edge positions
  - Third layer assembles motifs into larger combinations that corresponds to parts of familiar objects
  - Subsequent layers would detect objects as combinations of these parts
- Key aspect of deep learning:
  - These layers of features are not designed by human engineers
  - They are learned from data using a general purpose learning procedure



# Deep versus Shallow Classifiers

- Linear classifiers can only carve the input space into very simple regions
- Image and speech recognition require input-output function to be insensitive to irrelevant variations of the input,
  - e.g., position, orientation and illumination of an object
  - Variations in pitch or accent of speech
  - While being sensitive to minute variations, e.g., white wolf and breed of wolf-like white dog called Samoyed
  - At pixel level two Samoyeds in different positions may be very different, whereas a Samoyed and a wolf in the same position and background may be very similar





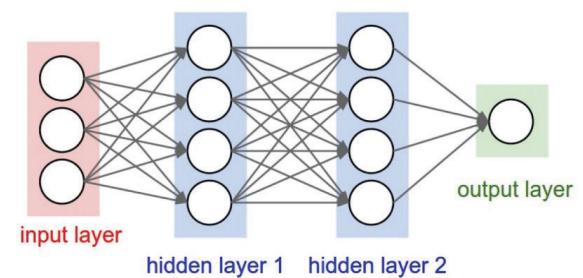
# Selectivity-Invariance dilemma

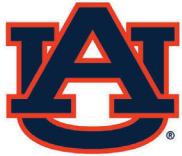
- Shallow classifiers need a good feature extractor
- One that produces representations that are:
  - selective to aspects of image important for discrimination
  - but invariant to irrelevant aspects such as pose of the animal
- Generic features (e.g., Gaussian kernel) do not generalize well far from training examples
- Hand-designing good feature extractors requires engineering skill and domain expertise
- Deep learning learns features automatically



# DL Architectures

- Multilayer stack of simple modules
- All modules (or most) subject to:
  - Learning
  - Non-linear input-output mappings
- Each module transforms input to improve both selectivity and invariance of the representation
- With depth of 5 to 20 layers can implement extremely intricate functions of input
  - Sensitive to minute details
    - Distinguish Samoyeds from white wolves
- Insensitive to irrelevant variations
  - Background, pose, lighting, surrounding objects





# Convolutional Neural Networks



# Key Ideas

- Take advantage of properties of natural signals
  - Local connections
  - Shared weights
  - Pooling
  - Use of many layers



# Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**



# Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**
- **Exploitation of Local Properties!**
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image



# Basic Mechanisms in CNNs

- Three Mechanisms of Convolutional Neural Networks:
  - Convolution Operation
  - Local Receptive Fields
  - Subsampling
  - Weight (Parameter) Sharing



# COMP [56]630– Machine Learning

Lecture 11 – Deep Learning (CNNs)



# Deep Neural Networks



# What is Deep Learning?

1. Computational models composed of multiple processing layers
  - To learn representations of data with multiple levels of abstraction
2. Dramatically improved state-of-the-art in:
  - Speech recognition, Visual object recognition, Object detection
  - Other domains: Drug discovery, Genomics
3. Discovers intricate structure in large data sets
  - Using backpropagation to change parameters
  - Compute representation in each layer from previous layer
4. Deep convolutional nets: image proc, video, speech
5. Recurrent nets: sequential data, e,g., text, speech



# Limitations of Conventional ML

- Limited in ability to process natural data in raw form
- Pattern Recognition and Machine Learning systems require careful engineering and domain expertise to transform raw data, e.g., pixel values, into a feature vector for a classifier



# Automatic Representation Learning

- Methods that allow a machine to be fed with raw data to automatically discover representations needed for detection or classification
- Deep Learning methods are Representation Learning Methods
- Use multiple levels of representation
  - Composing simple but non-linear modules that transform representation at one level (starting with raw input) into a representation at a higher slightly more abstract level
  - Complex functions can be learned
  - Higher layers of representation amplify aspects of input important for discrimination and suppress irrelevant variations



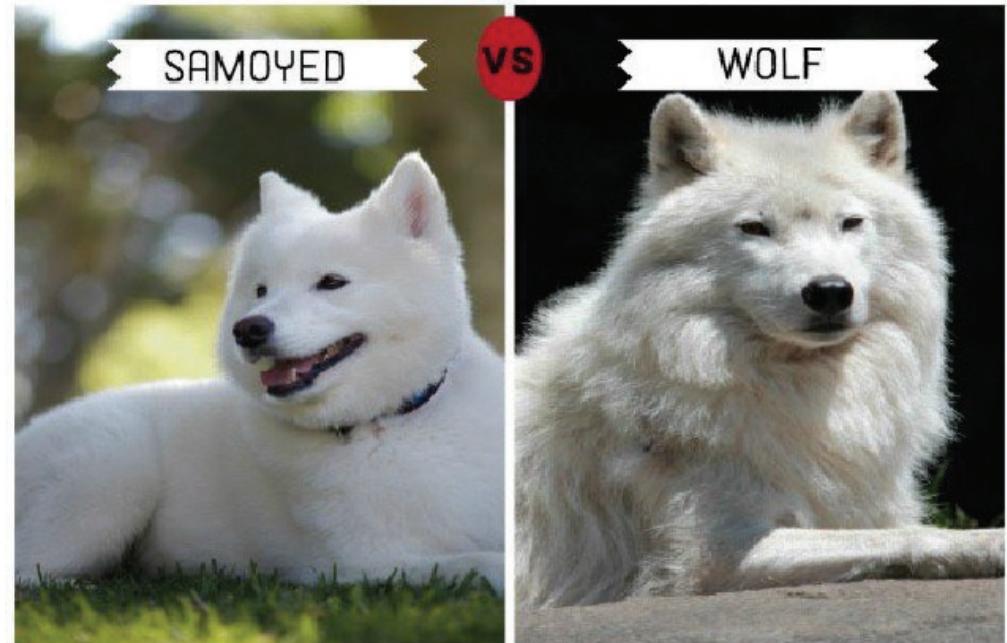
# Example: Images

- Input is an array of pixel values
  - First stage is presence or absence of edges at particular locations and orientations of image
  - Second layer detects motifs by spotting particular arrangements of edges, regardless of small variations in edge positions
  - Third layer assembles motifs into larger combinations that corresponds to parts of familiar objects
  - Subsequent layers would detect objects as combinations of these parts
- Key aspect of deep learning:
  - These layers of features are not designed by human engineers
  - They are learned from data using a general purpose learning procedure



# Deep versus Shallow Classifiers

- Linear classifiers can only carve the input space into very simple regions
- Image and speech recognition require input-output function to be insensitive to irrelevant variations of the input,
  - e.g., position, orientation and illumination of an object
  - Variations in pitch or accent of speech
  - While being sensitive to minute variations, e.g., white wolf and breed of wolf-like white dog called Samoyed
  - At pixel level two Samoyeds in different positions may be very different, whereas a Samoyed and a wolf in the same position and background may be very similar





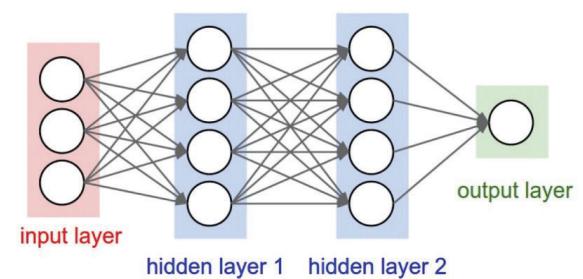
# Selectivity-Invariance dilemma

- Shallow classifiers need a good feature extractor
- One that produces representations that are:
  - selective to aspects of image important for discrimination
  - but invariant to irrelevant aspects such as pose of the animal
- Generic features (e.g., Gaussian kernel) do not generalize well far from training examples
- Hand-designing good feature extractors requires engineering skill and domain expertise
- Deep learning learns features automatically



# DL Architectures

- Multilayer stack of simple modules
- All modules (or most) subject to:
  - Learning
  - Non-linear input-output mappings
- Each module transforms input to improve both selectivity and invariance of the representation
- With depth of 5 to 20 layers can implement extremely intricate functions of input
  - Sensitive to minute details
    - Distinguish Samoyeds from white wolves
- Insensitive to irrelevant variations
  - Background, pose, lighting, surrounding objects





# Convolutional Neural Networks



# Key Ideas

- Take advantage of properties of natural signals
  - Local connections
  - Shared weights
  - Pooling
  - Use of many layers



# Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**



# Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**
- **Exploitation of Local Properties!**
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image

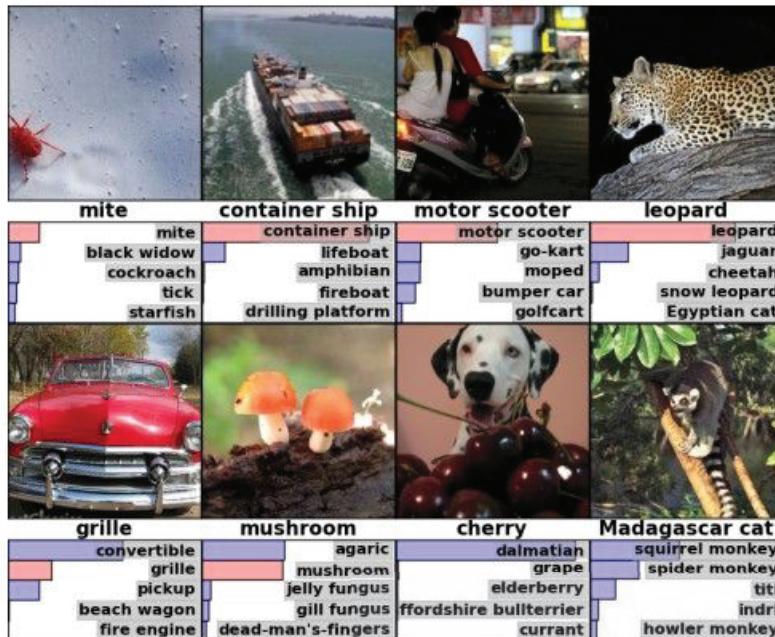


# Basic Mechanisms in CNNs

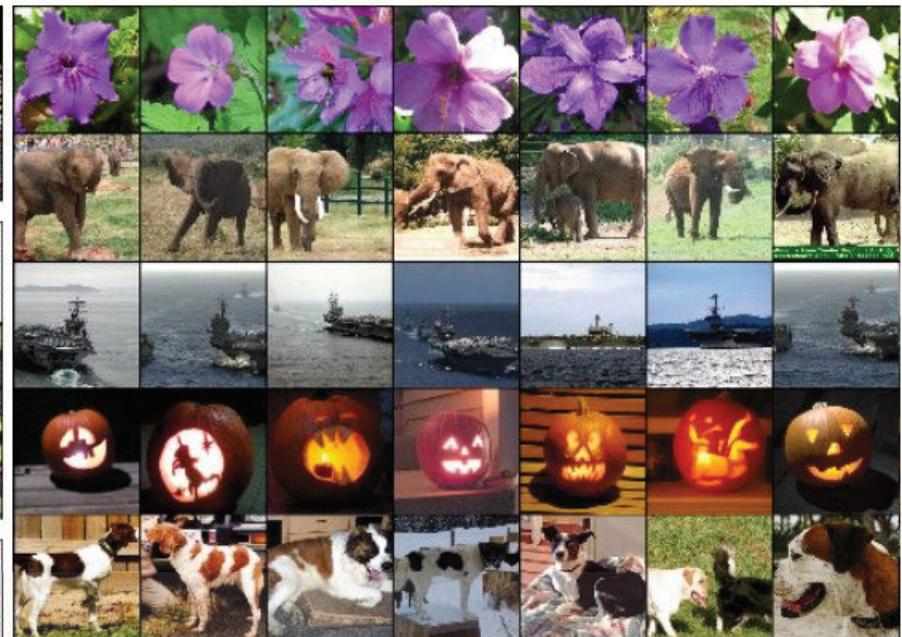
- Three Mechanisms of Convolutional Neural Networks:
  - Convolution Operation
  - Local Receptive Fields
  - Subsampling
  - Weight (Parameter) Sharing

# Fast-forward to today: ConvNets are everywhere

Classification



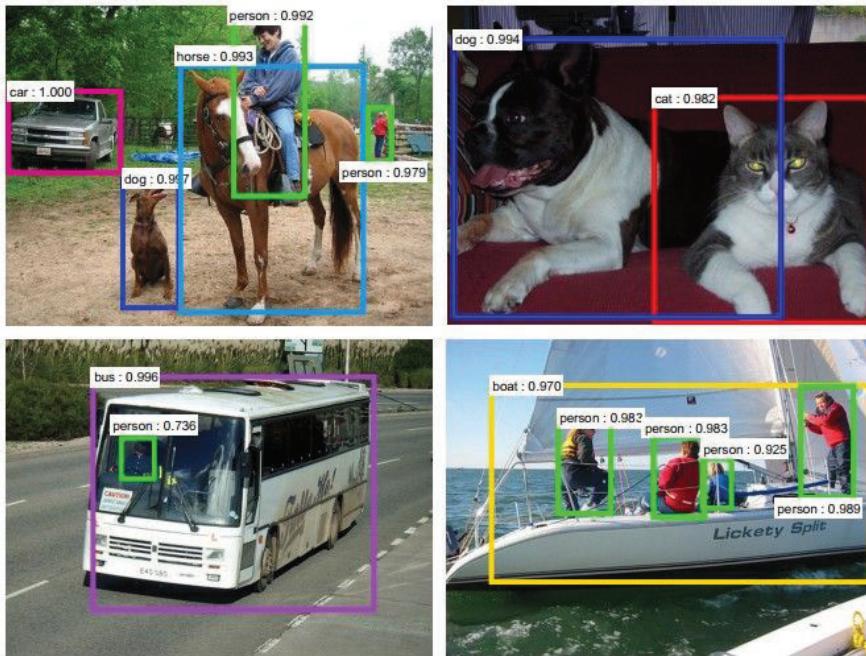
Retrieval



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

# Fast-forward to today: ConvNets are everywhere

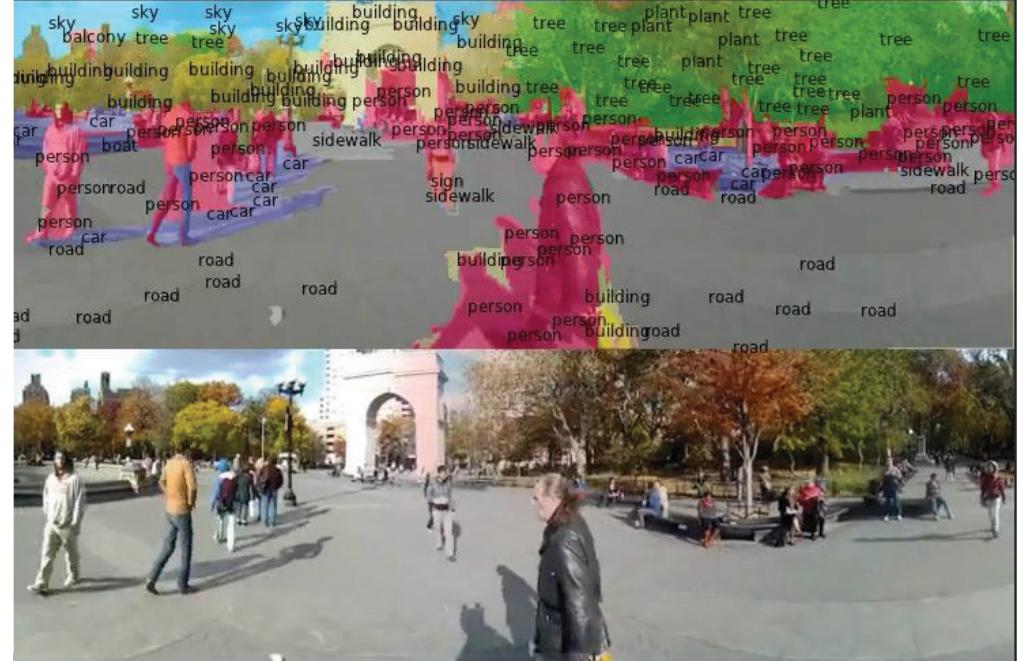
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

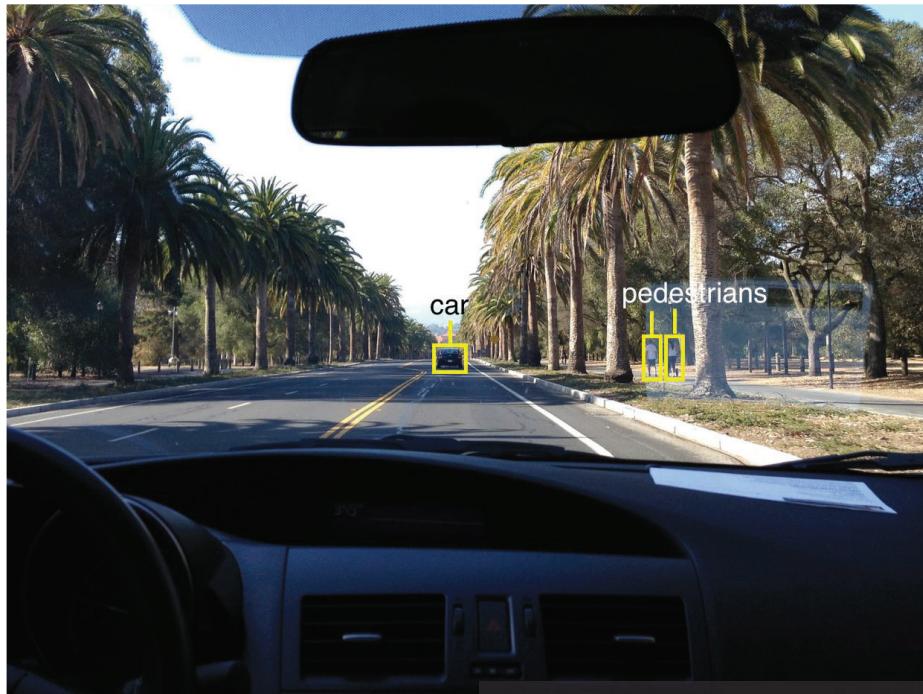


Figures copyright Clement Farabet, 2012.

Reproduced with permission.

[Farabet et al., 2012]

# Fast-forward to today: ConvNets are everywhere



self-driving cars

Photo by Lane McIntosh. Copyright CS231n 2017.



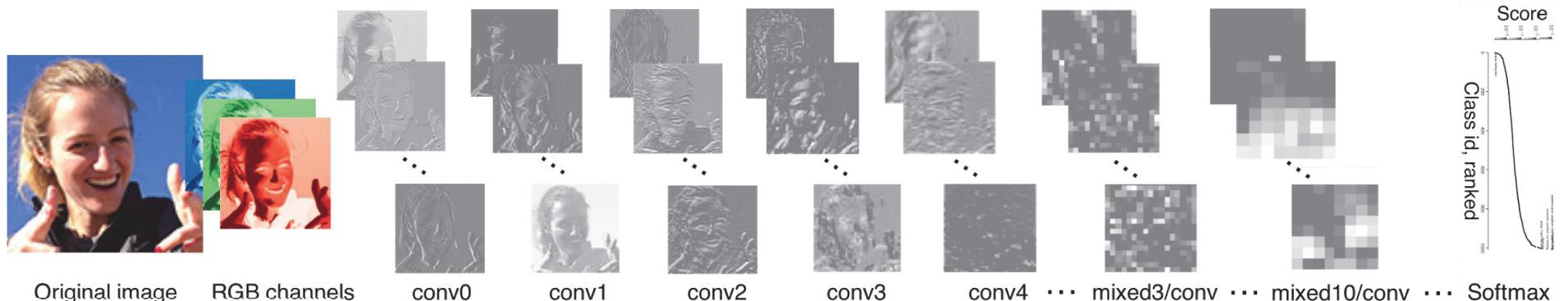
[This image](#) by GBPublic\_PR is  
licensed under [CC-BY 2.0](#)

## NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

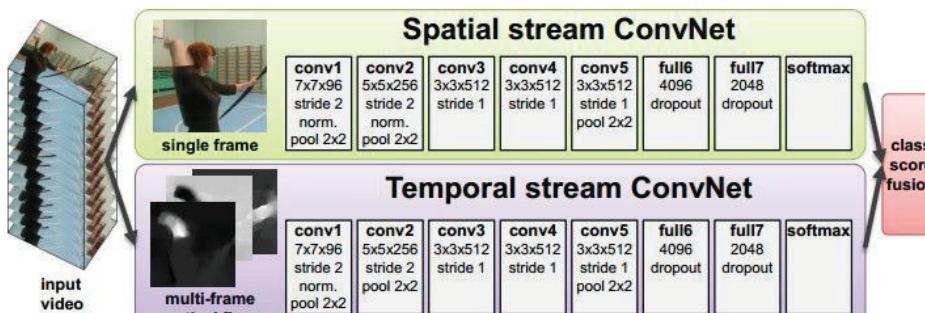
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

# Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]

Activations of [Inception-v3 architecture](#) [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014.  
Reproduced with permission.

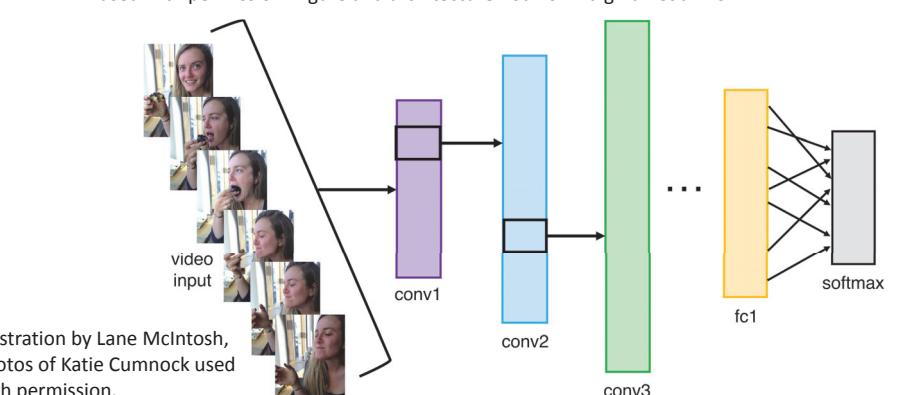


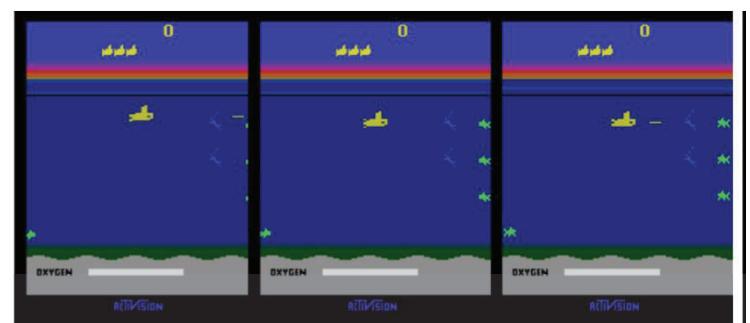
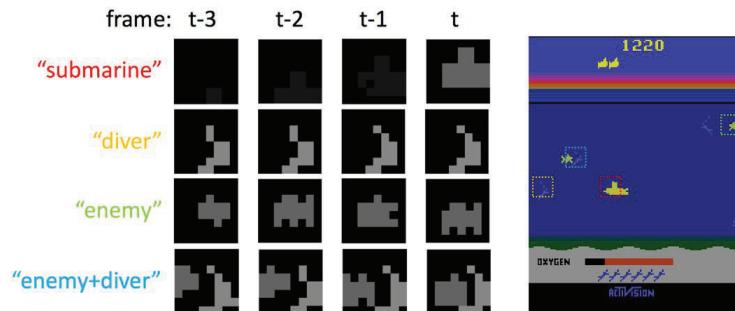
Illustration by Lane McIntosh,  
photos of Katie Cumnock used  
with permission.

# Fast-forward to today: ConvNets are everywhere



Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

# Fast-forward to today: ConvNets are everywhere

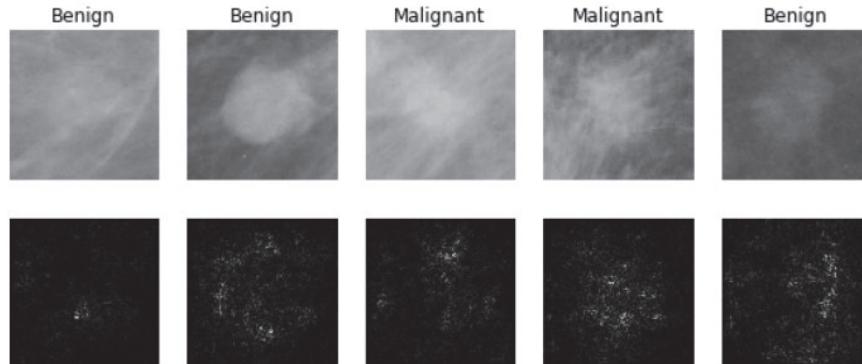


Figure copyright Levy et al. 2016.  
Reproduced with permission.



From left to right: [public domain by NASA](#), usage [permitted](#) by  
ESA/Hubble, [public domain by NASA](#), and [public domain](#).



Photos by Lane McIntosh.  
Copyright CS231n 2017.

[Sermanet et al. 2011]

[Ciresan et al.]

[This image](#) by Christin Khan is in the public domain and originally came from the U.S. NOAA.



*Whale recognition, Kaggle Challenge*

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.



*Mnih and Hinton, 2010*

# Image Captioning

[Vinyals et al., 2015]  
[Karpathy and Fei-Fei, 2015]

No errors



*A white teddy bear sitting in the grass*

Minor errors



*A man in a baseball uniform throwing a ball*

Somewhat related



*A woman is holding a cat in her hand*



*A man riding a wave on top of a surfboard*



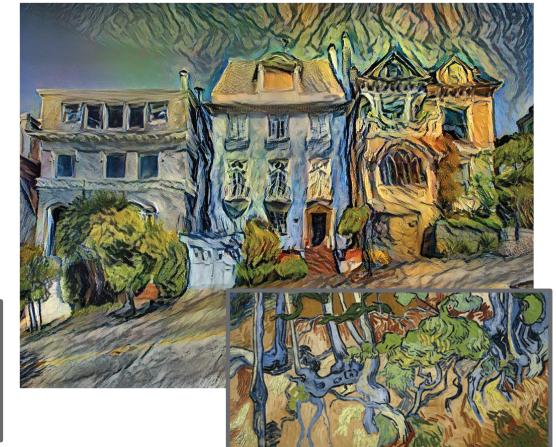
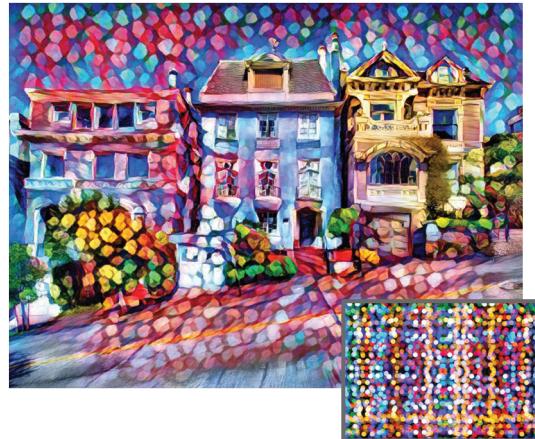
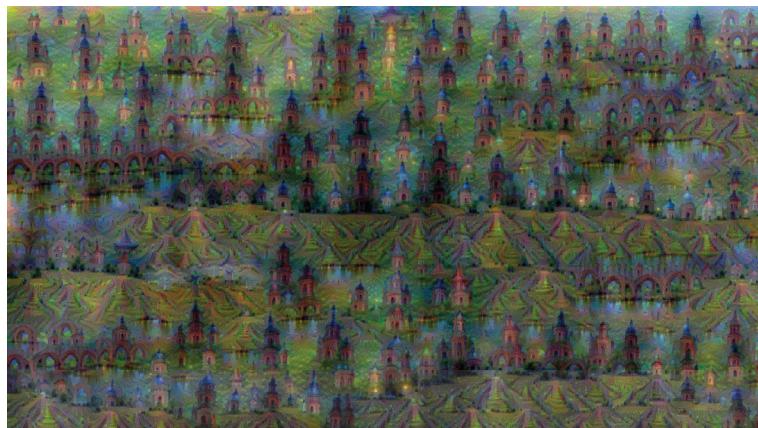
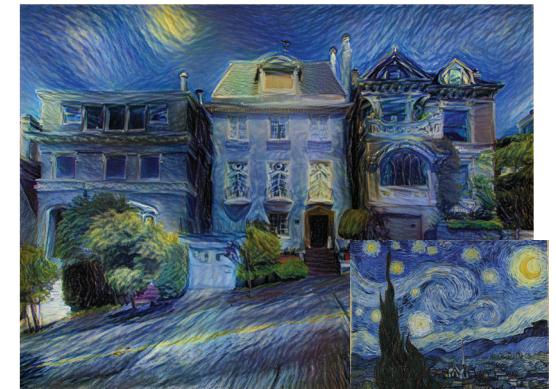
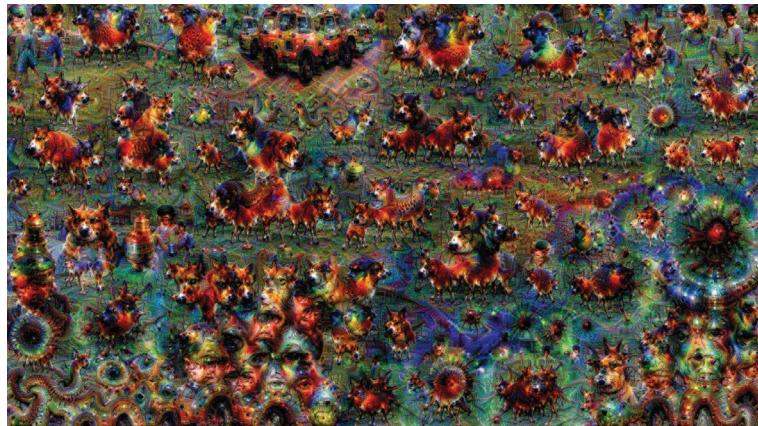
*A cat sitting on a suitcase on the floor*



*A woman standing on a beach holding a surfboard*

All images are CC0 Public domain:  
<https://pixabay.com/en/luggage-antique-cat-1643010/>  
<https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/>  
<https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/>  
<https://pixabay.com/en/woman-female-model-portrait-adult-983967/>  
<https://pixabay.com/en/handstand-lake-meditation-496008/>  
<https://pixabay.com/en/baseball-player-shortstop-infield-1045263/>

Captions generated by Justin Johnson using [Neuraltalk2](#)



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a [blog post](#) by Google Research.

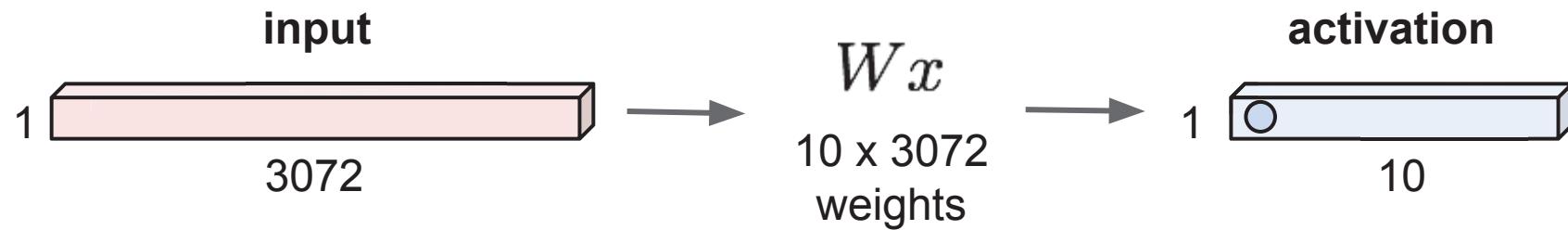
Original image is CC0 public domain  
[Starry Night](#) and [Tree Roots](#) by Van Gogh are in the public domain  
[Bokeh image](#) is in the public domain  
Stylized images copyright Justin Johnson, 2017;  
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016  
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

# Convolutional Neural Networks

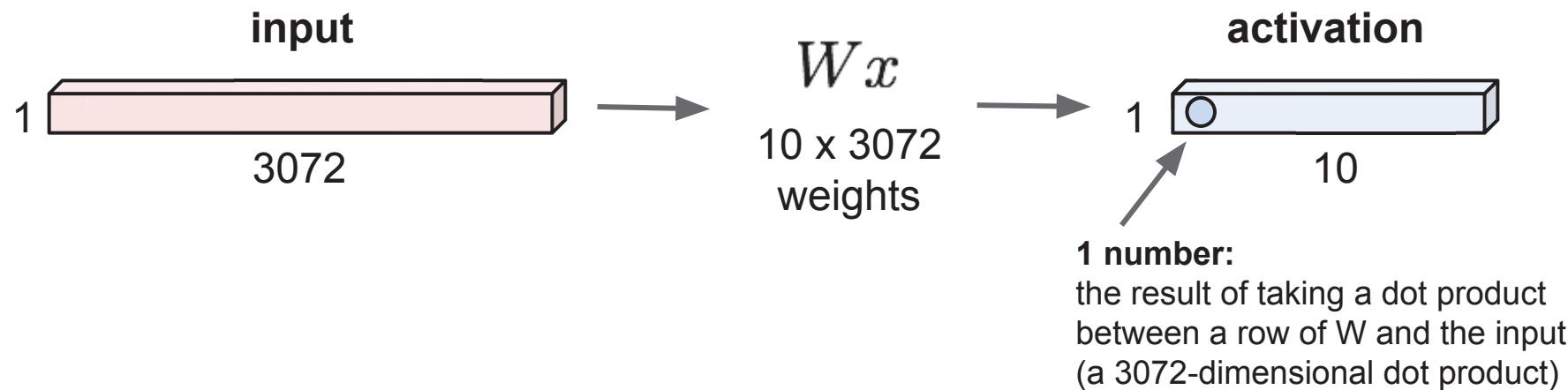
# Recap: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



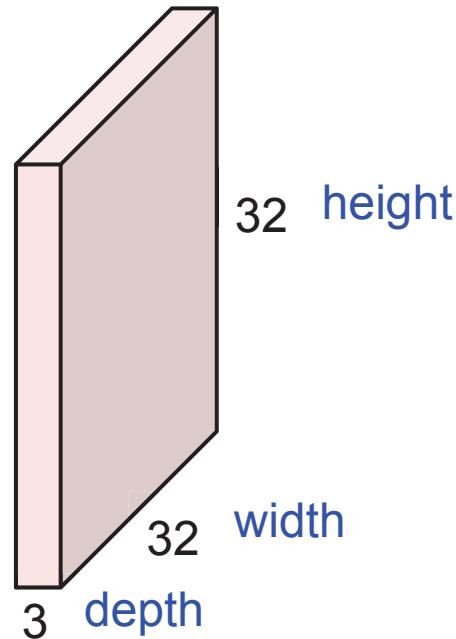
# Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



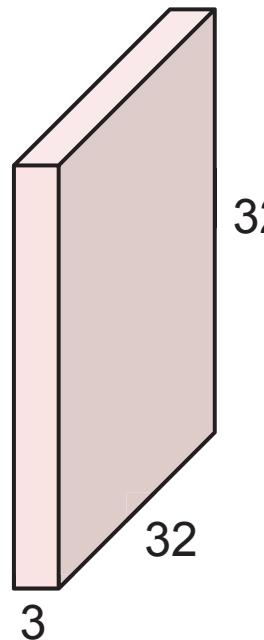
# Convolution Layer

32x32x3 image -> preserve spatial structure



# Convolution Layer

32x32x3 image



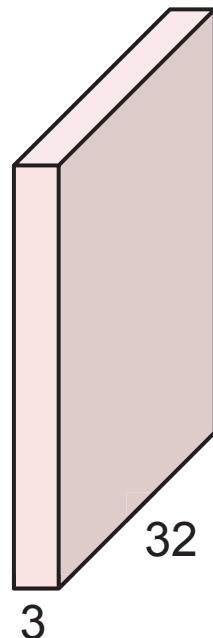
5x5x3 filter



**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

32x32x3 image



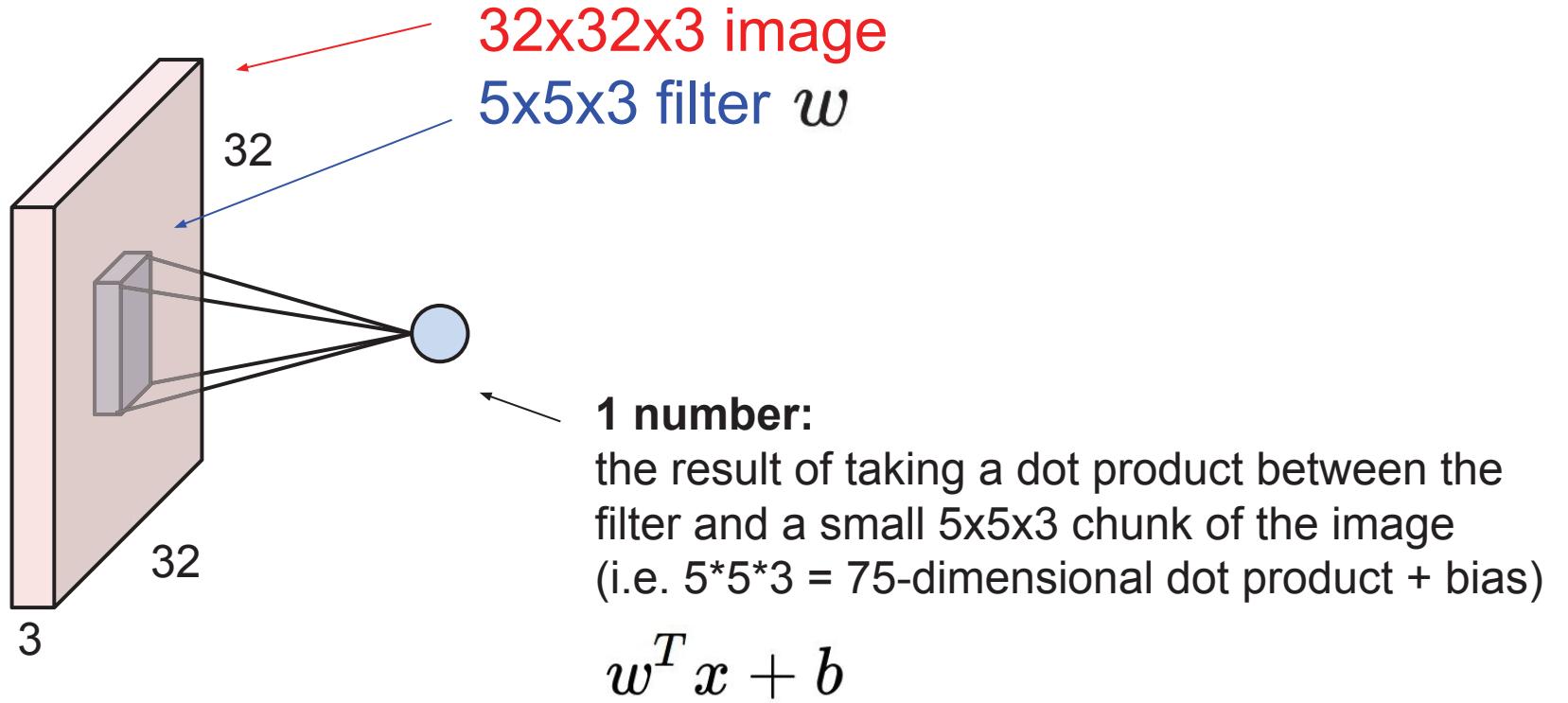
5x5x3 filter



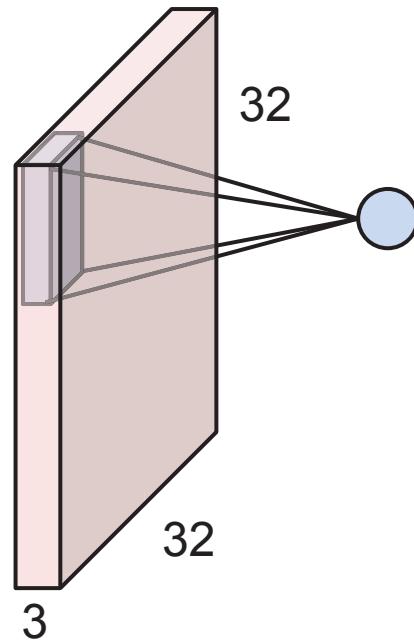
Filters always extend the full depth of the input volume

**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

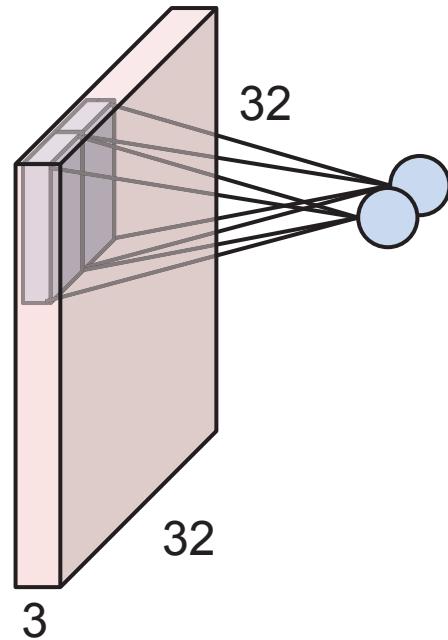
# Convolution Layer



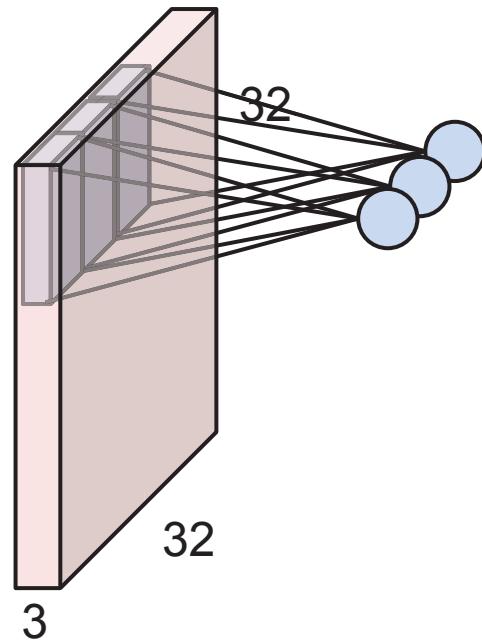
# Convolution Layer



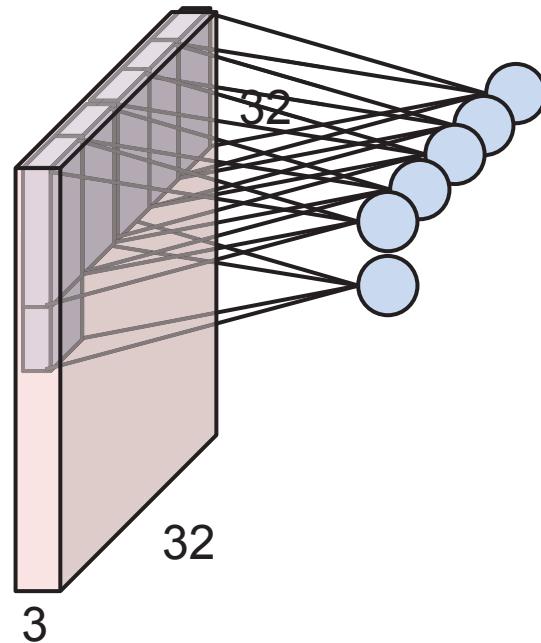
# Convolution Layer



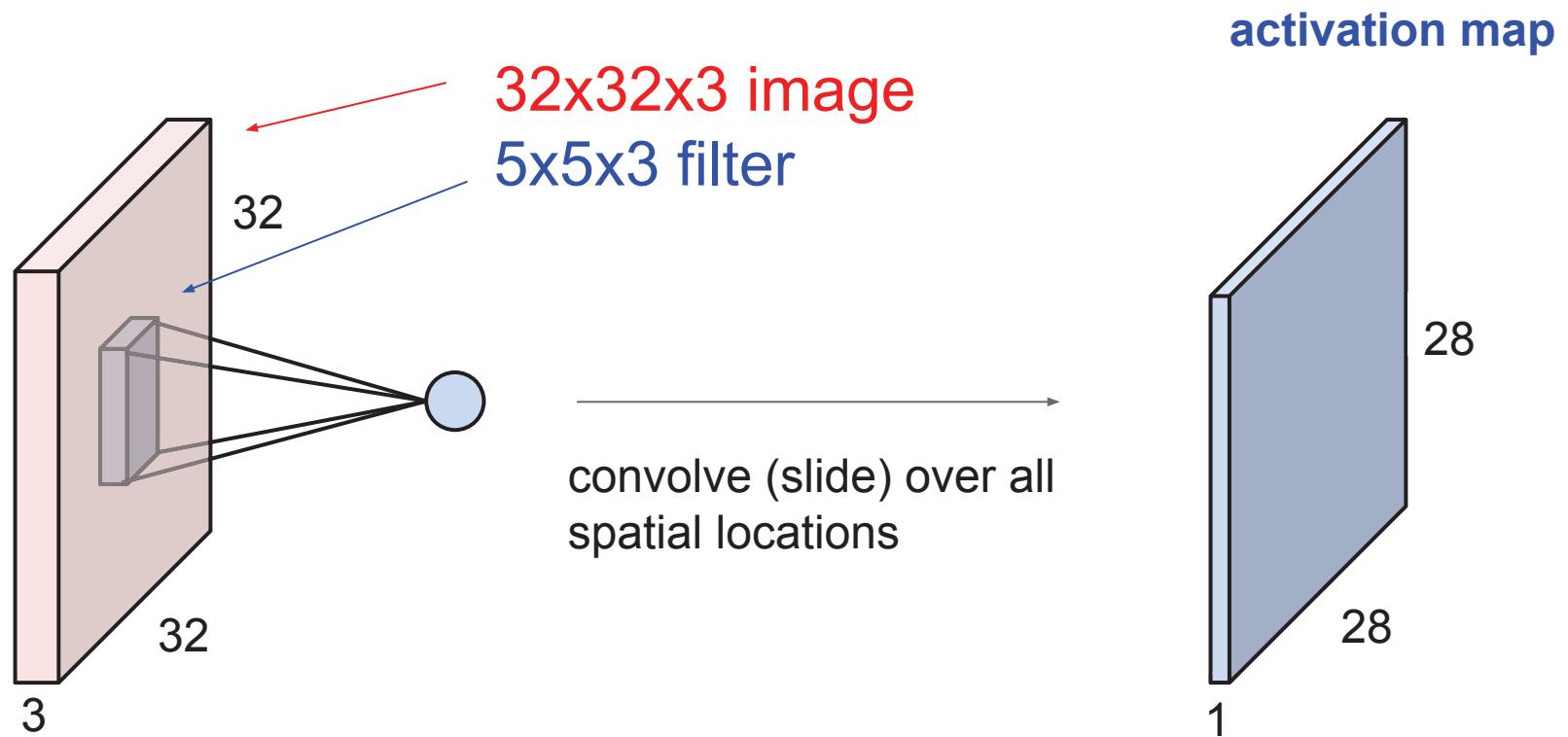
# Convolution Layer



# Convolution Layer

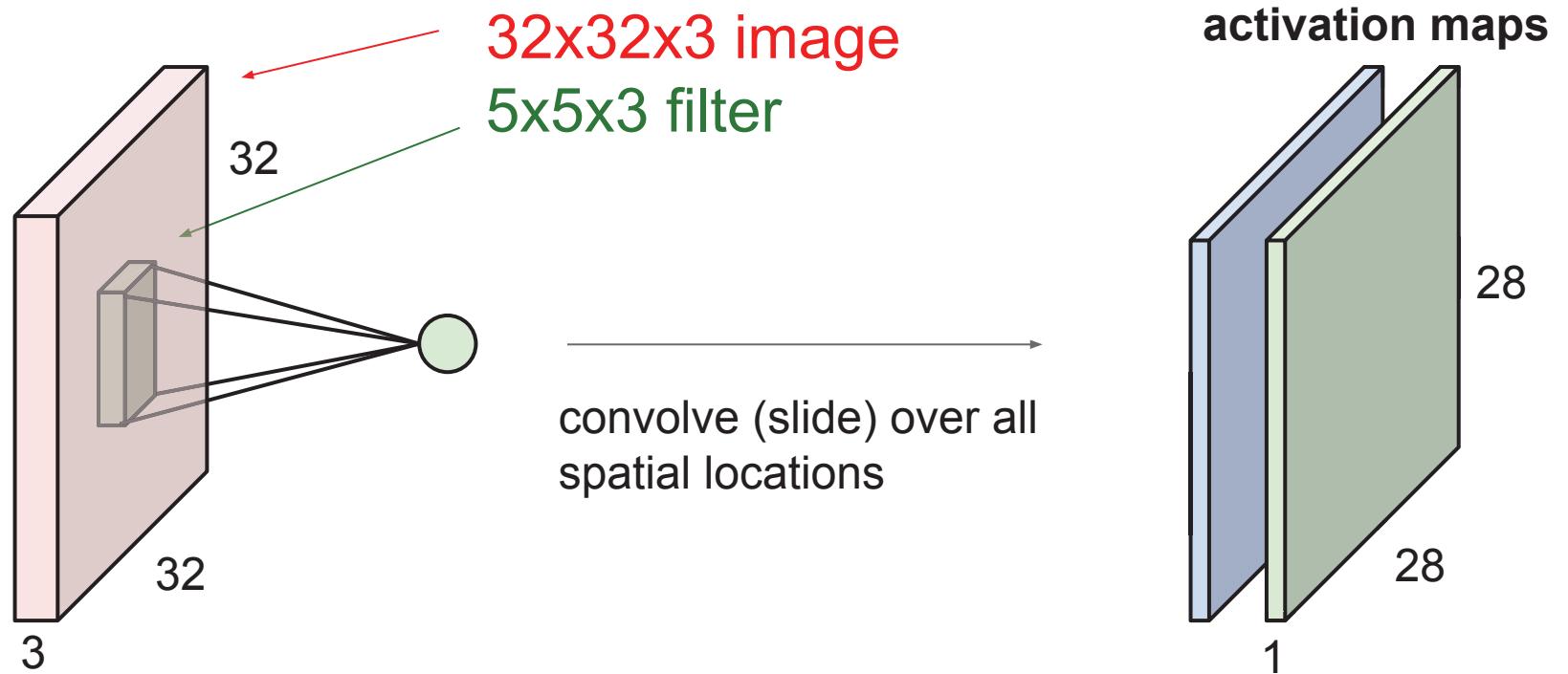


# Convolution Layer



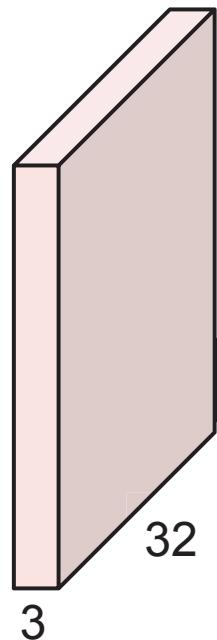
# Convolution Layer

consider a second, green filter



# Convolution Layer

3x32x32 image

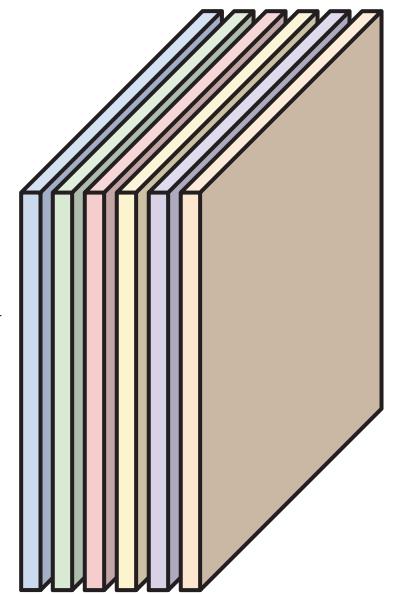


Consider 6 filters,  
each  $3 \times 5 \times 5$

6x3x5x5  
filters



6 activation maps,  
each  $1 \times 28 \times 28$

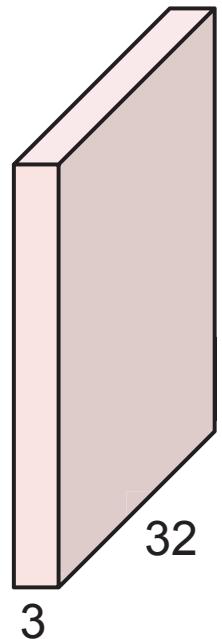


Stack activations to get a  
 $6 \times 28 \times 28$  output image!

Slide inspiration: Justin Johnson

# Convolution Layer

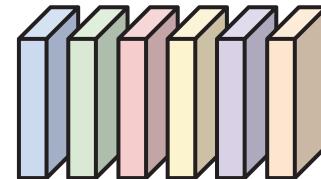
3x32x32 image



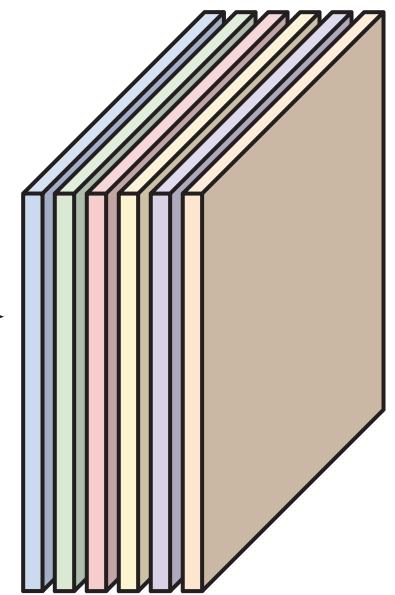
Also 6-dim bias vector:



6x3x5x5  
filters



6 activation maps,  
each 1x28x28



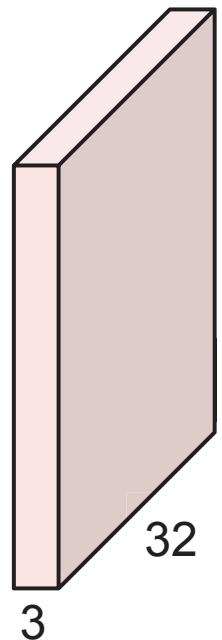
Stack activations to get a  
6x28x28 output image!

Slide inspiration: Justin Johnson

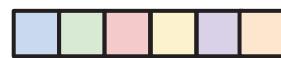
# Convolution Layer

28x28 grid, at each point a 6-dim vector

3x32x32 image

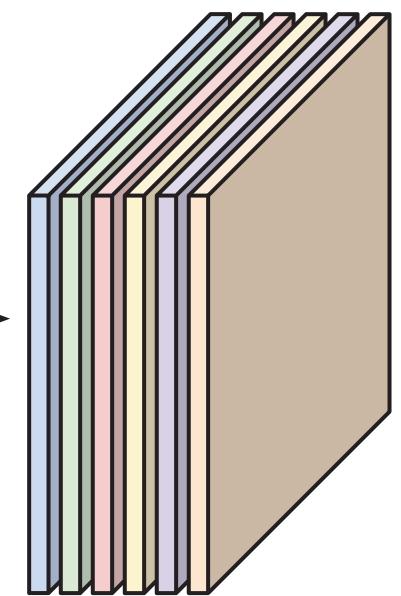
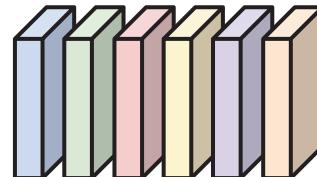


Also 6-dim bias vector:



Convolution  
Layer

6x3x5x5  
filters

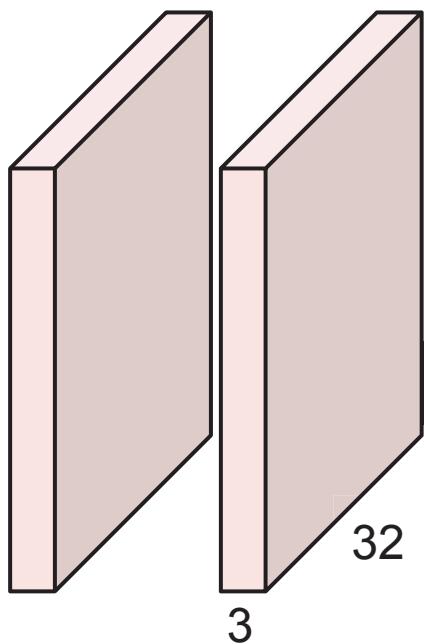


Stack activations to get a  
6x28x28 output image!

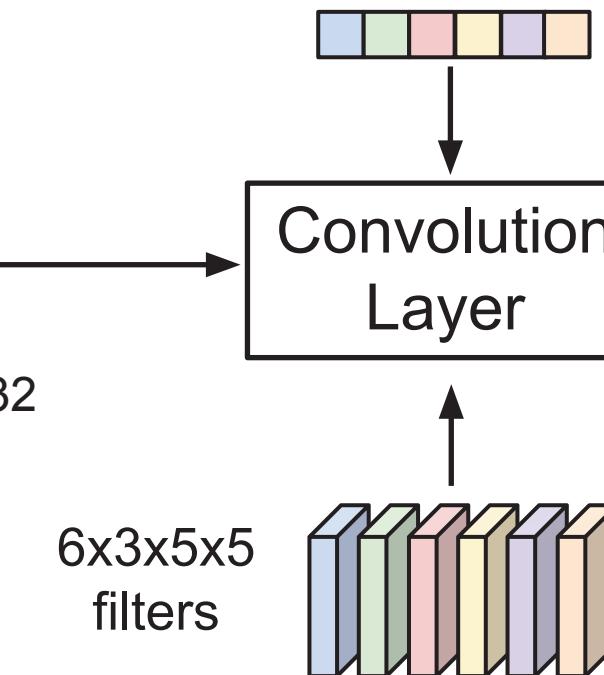
Slide inspiration: Justin Johnson

# Convolution Layer

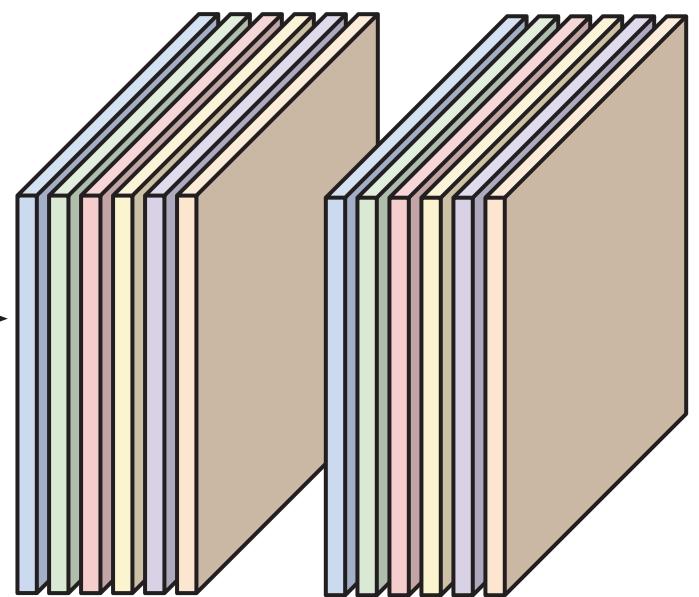
$2 \times 3 \times 32 \times 32$   
Batch of images



Also 6-dim bias vector:



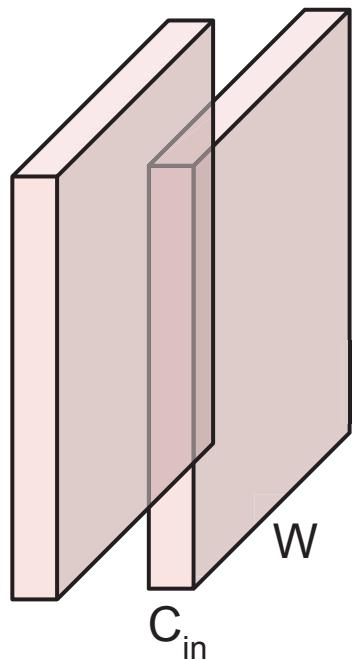
$2 \times 6 \times 28 \times 28$   
Batch of outputs



Slide inspiration: Justin Johnson

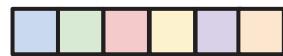
# Convolution Layer

$N \times C_{in} \times H \times W$   
Batch of images



$C_{out} \times C_{in} \times K_w \times K_h$   
filters

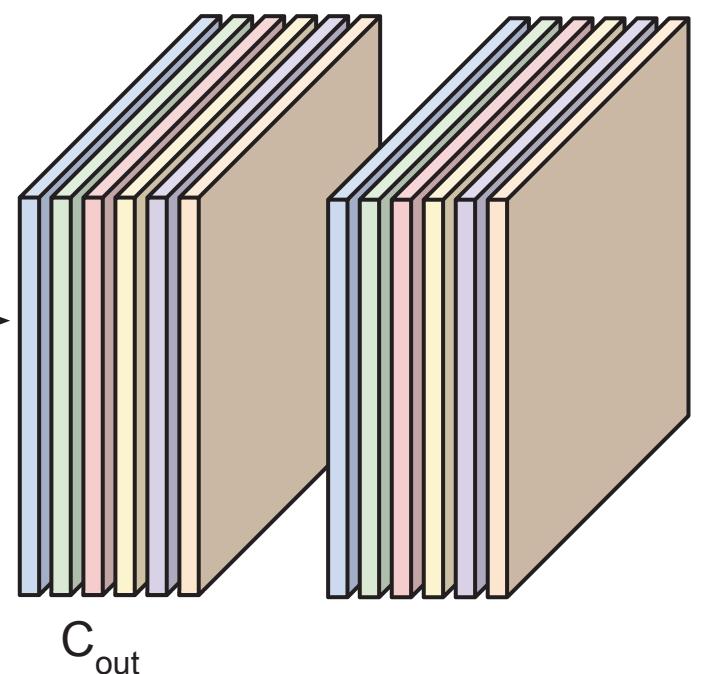
Also  $C_{out}$ -dim bias vector:



Convolution  
Layer

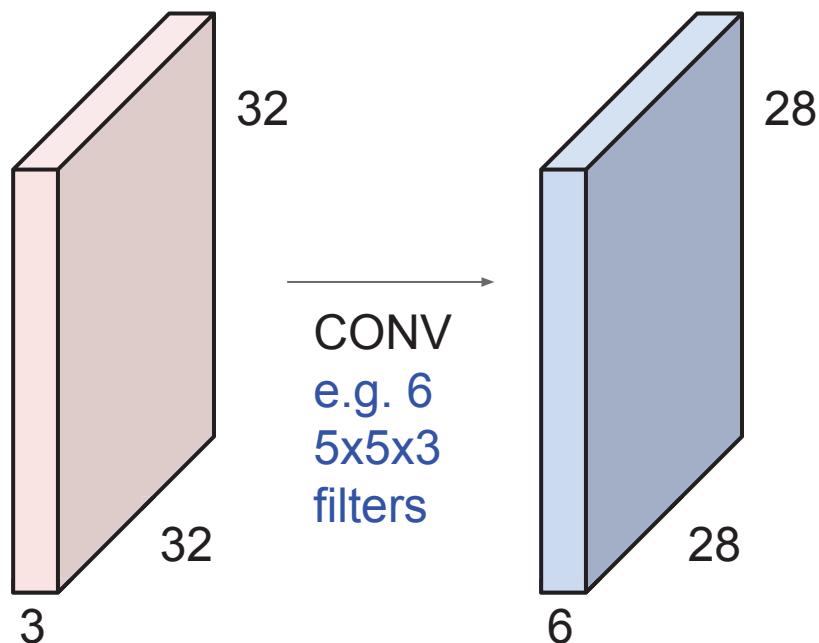


$N \times C_{out} \times H' \times W'$   
Batch of outputs

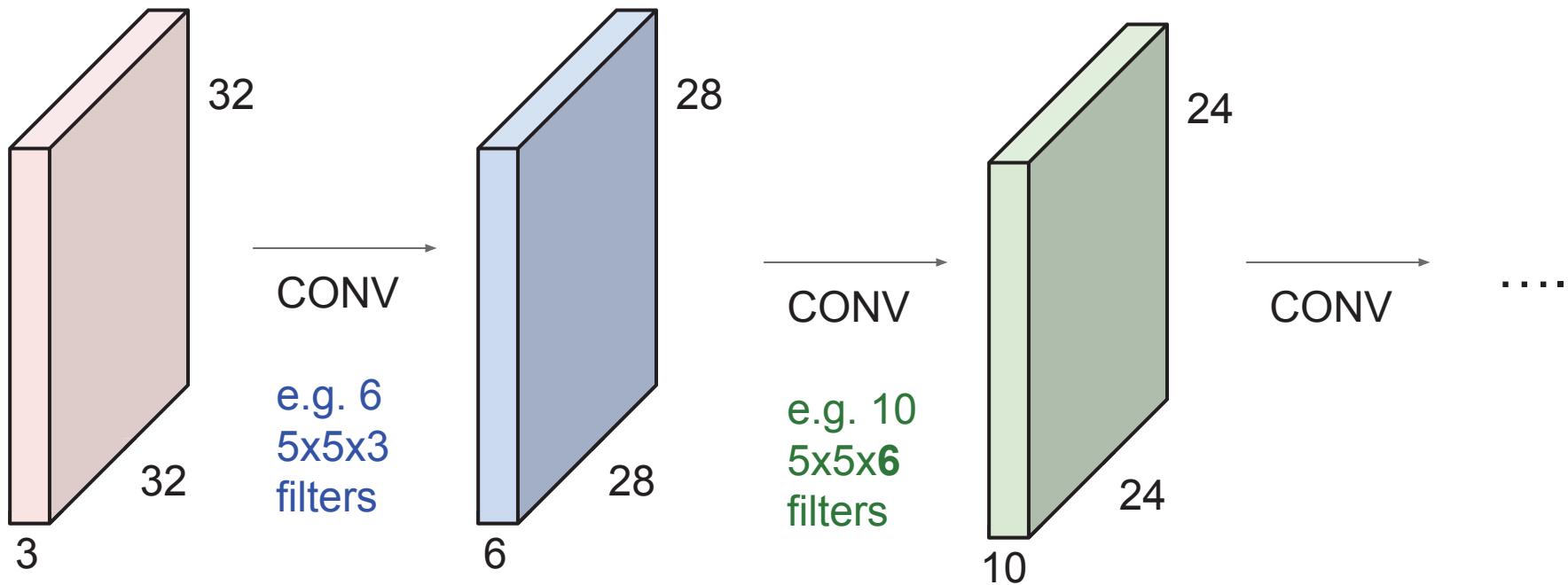


Slide inspiration: Justin Johnson

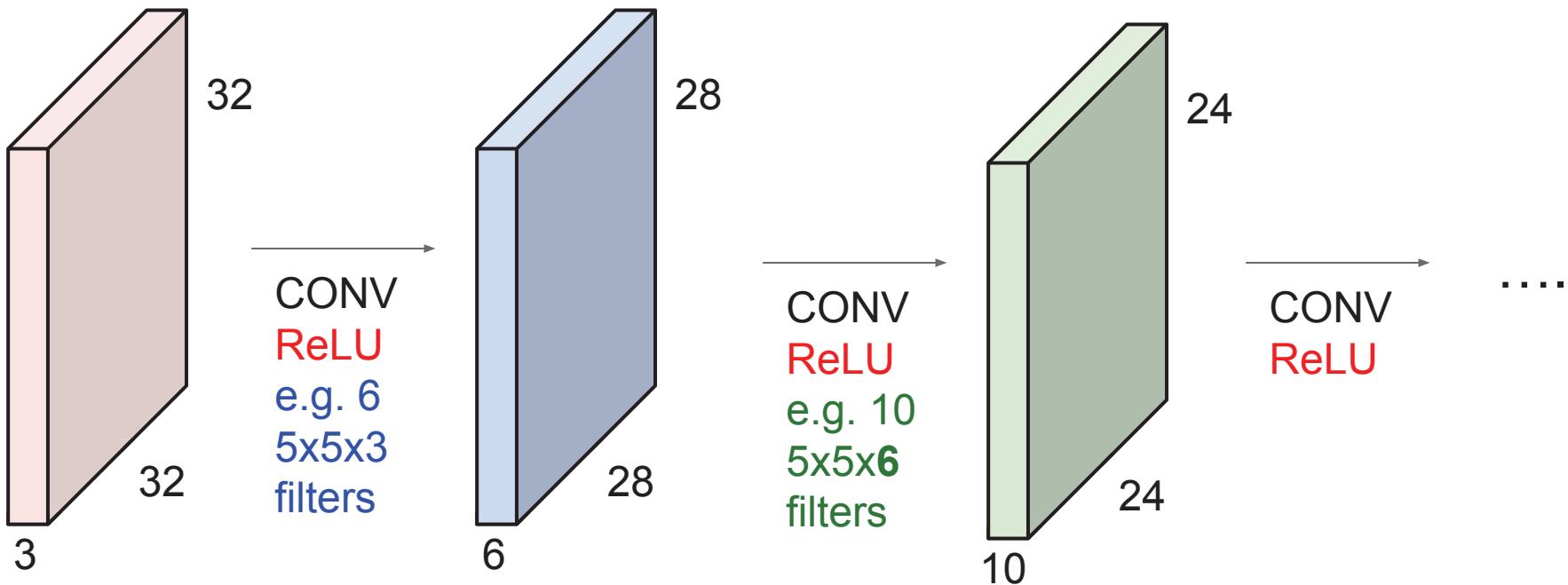
**Preview:** ConvNet is a sequence of Convolution Layers



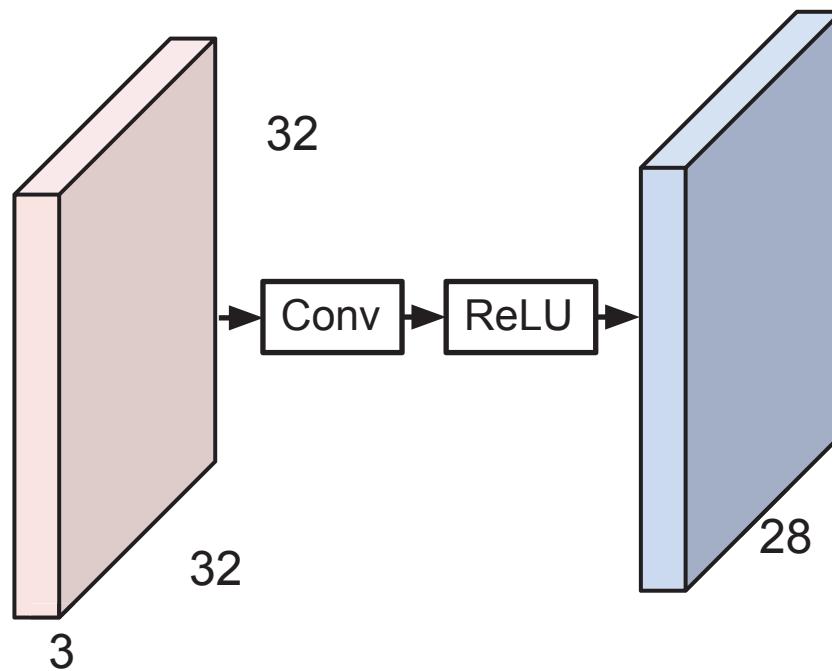
## Preview: ConvNet is a sequence of Convolution Layers



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



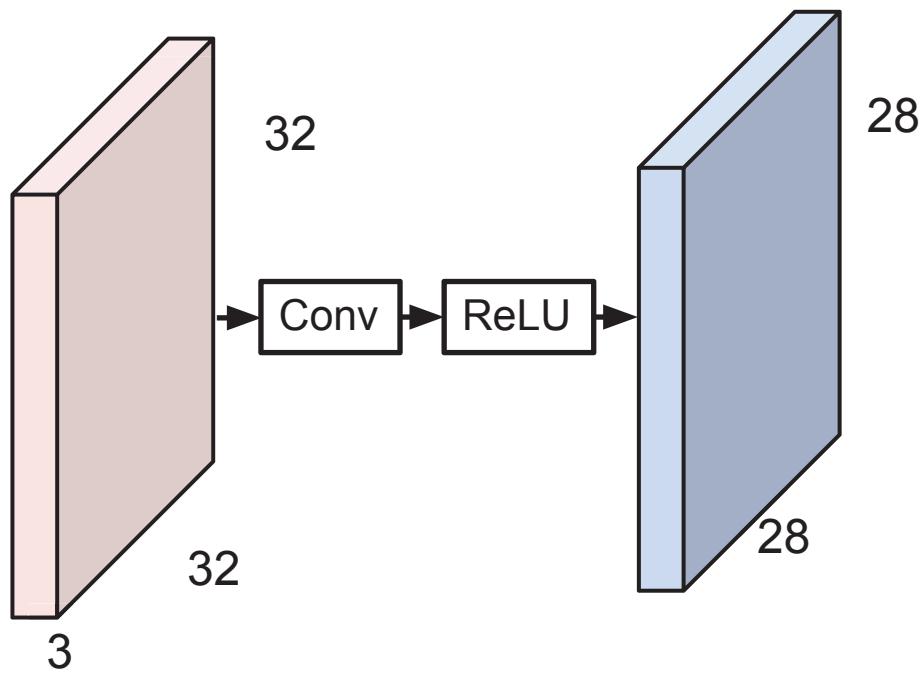
## Preview: What do convolutional filters learn?



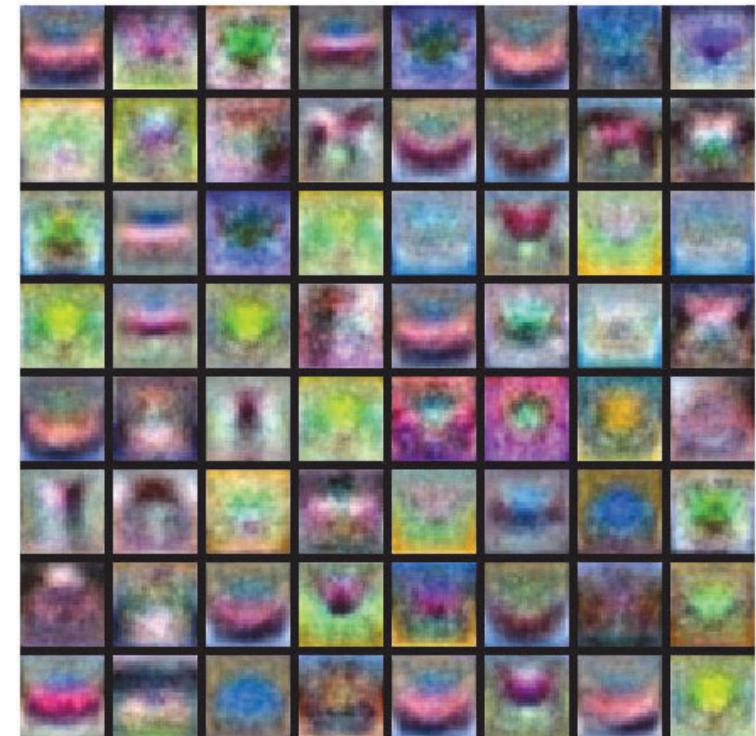
Linear classifier: One template per class



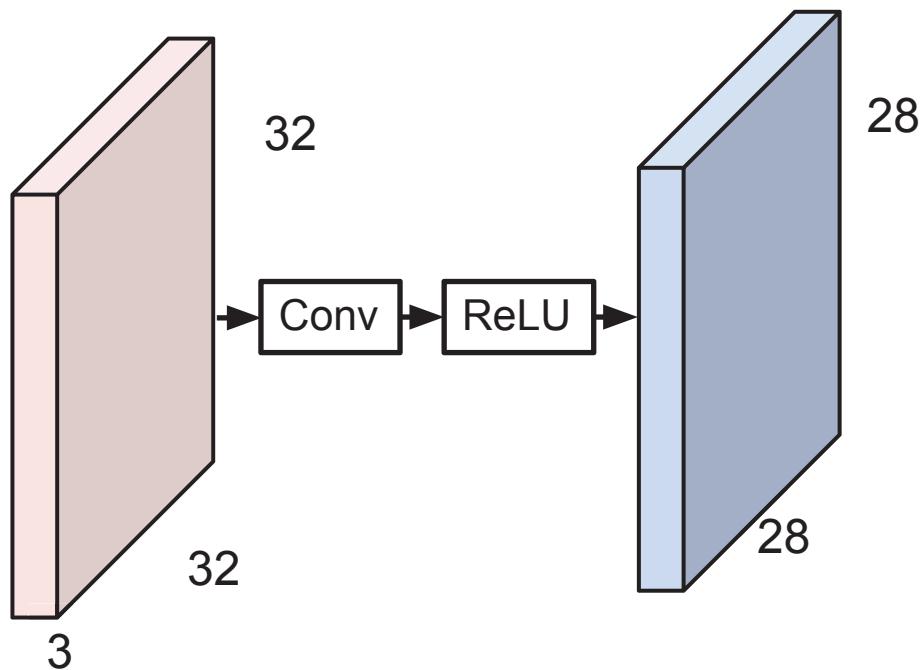
**Preview:** What do convolutional filters learn?



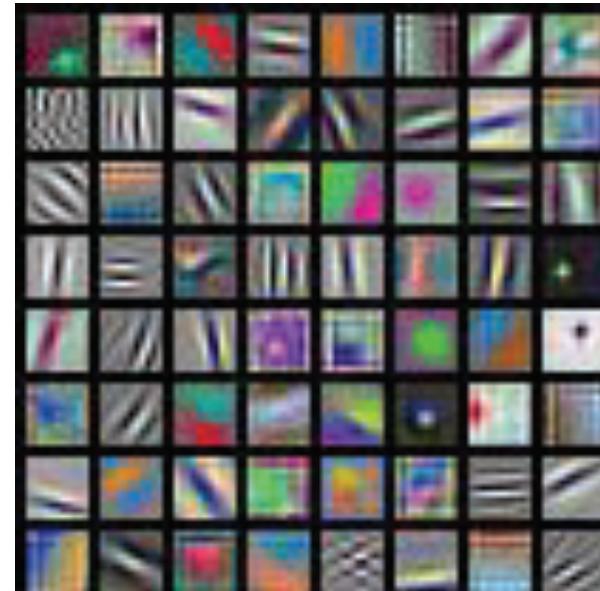
MLP: Bank of whole-image templates



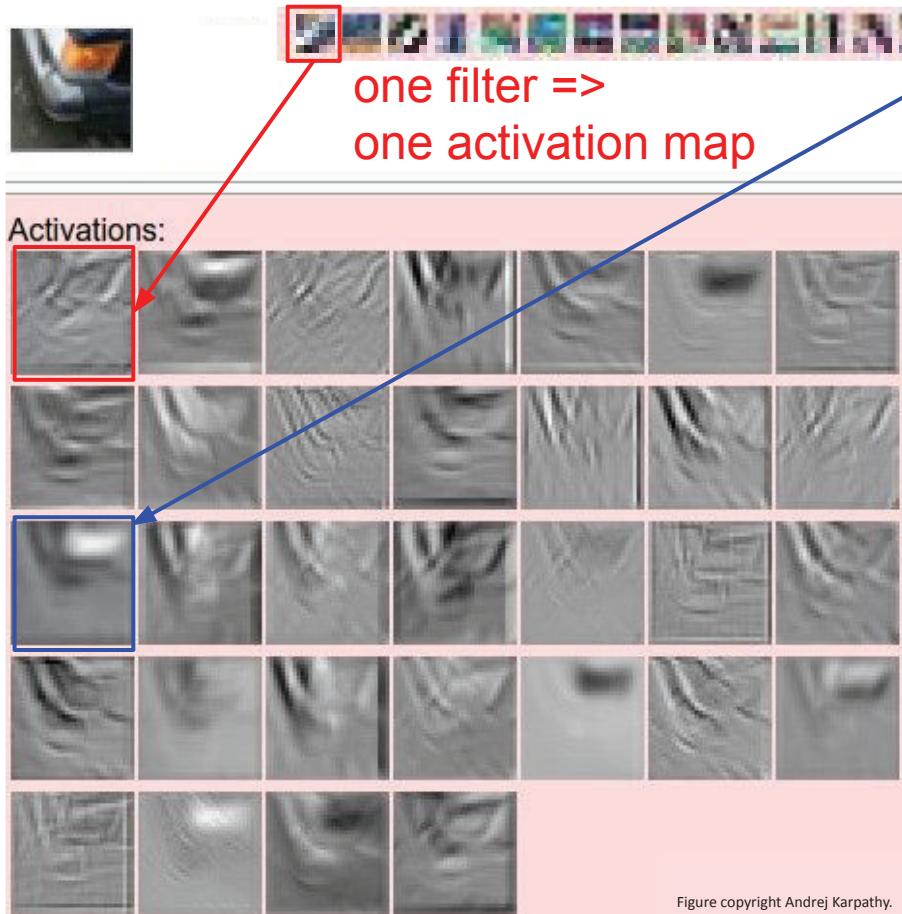
## Preview: What do convolutional filters learn?



First-layer conv filters: local image templates  
(Often learns oriented edges, opposing colors)



AlexNet: 64 filters, each 3x11x11



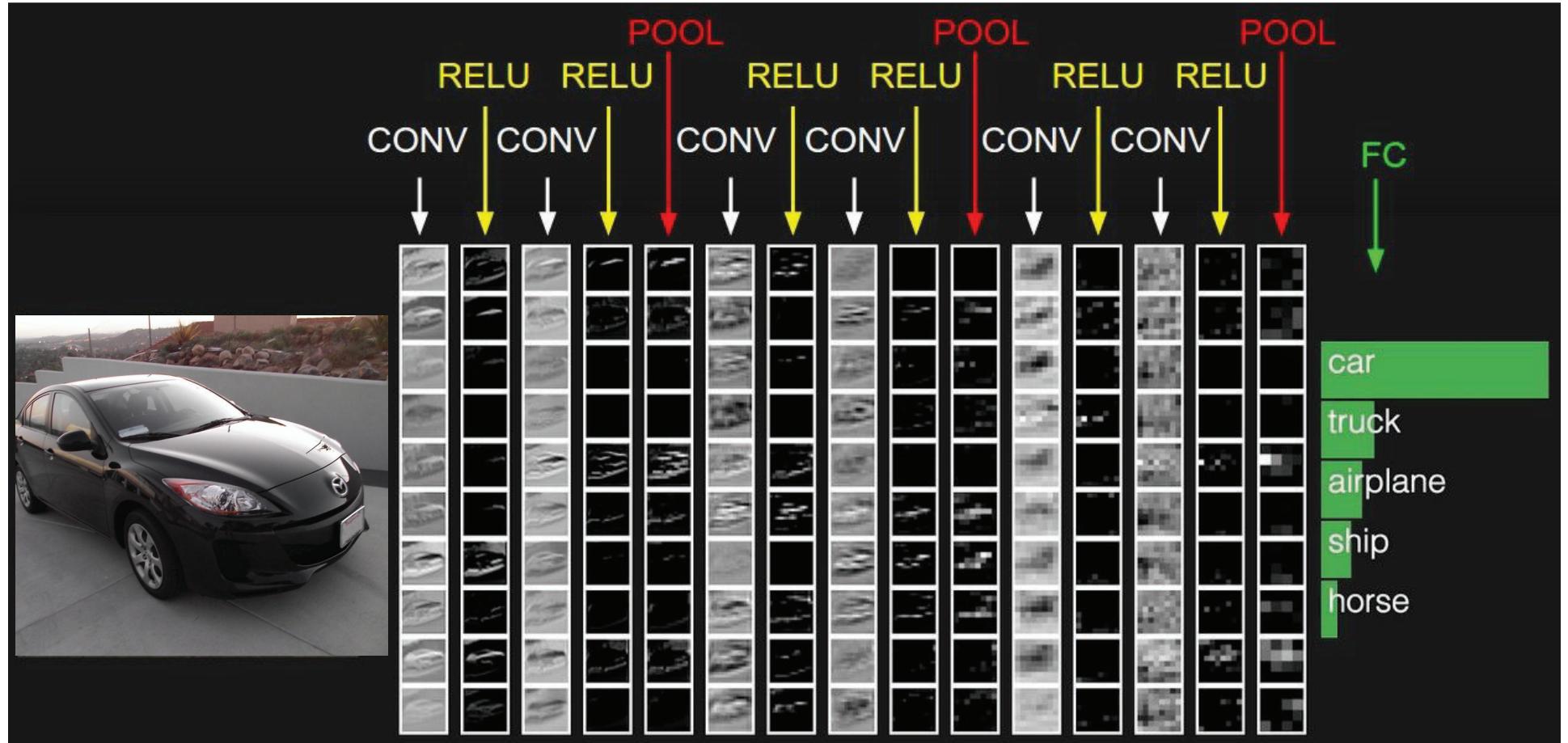
example 5x5 filters  
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

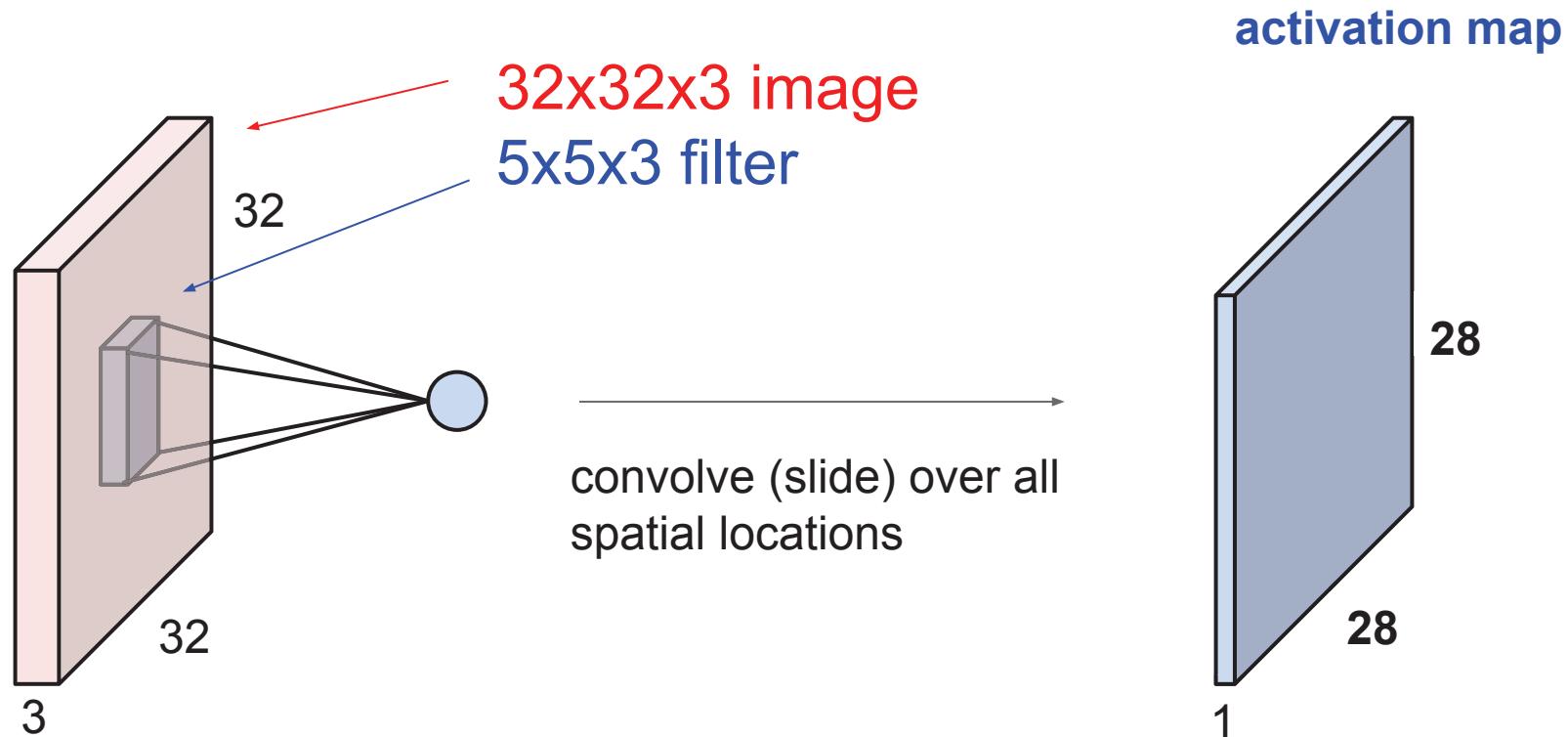
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

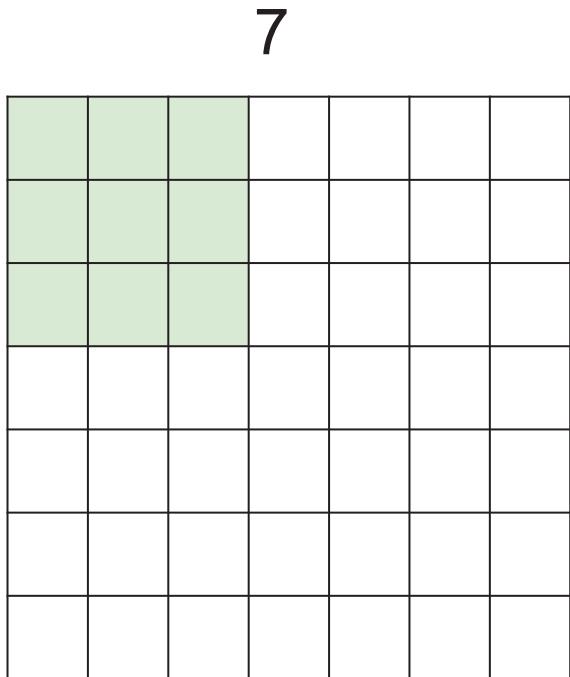
preview:



A closer look at spatial dimensions:

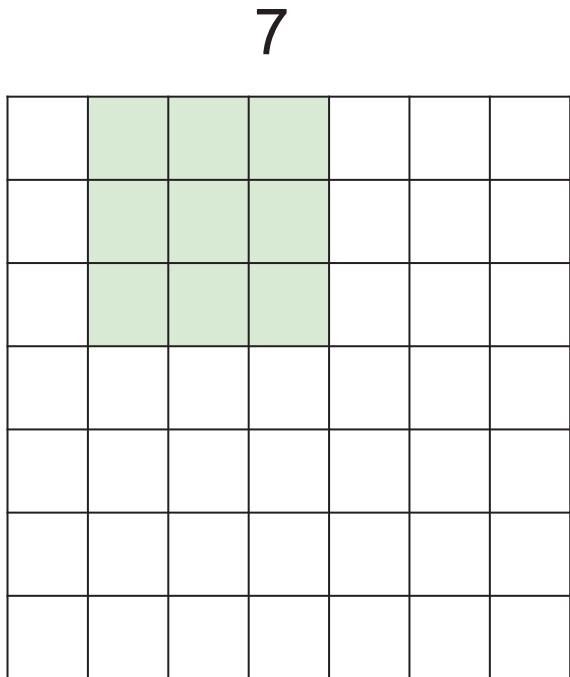


A closer look at spatial dimensions:



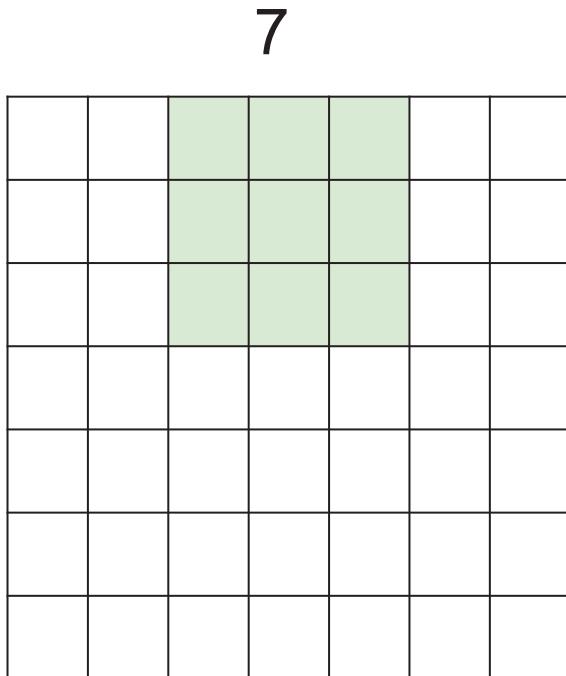
7x7 input (spatially)  
assume 3x3 filter

A closer look at spatial dimensions:



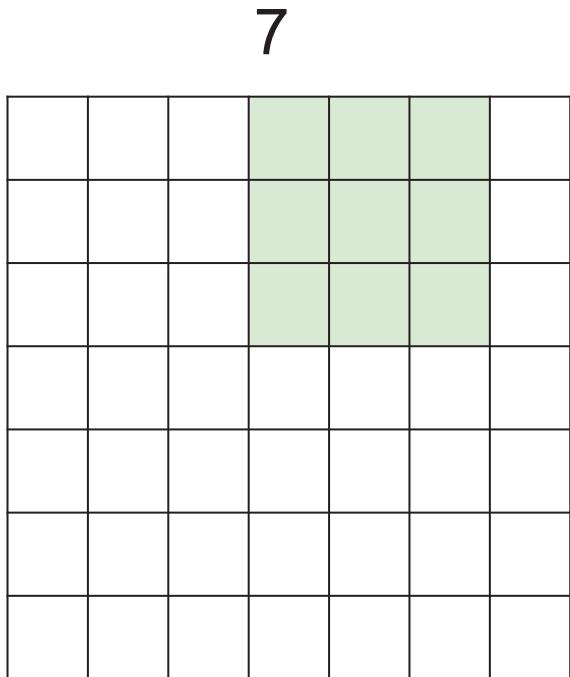
7x7 input (spatially)  
assume 3x3 filter

A closer look at spatial dimensions:



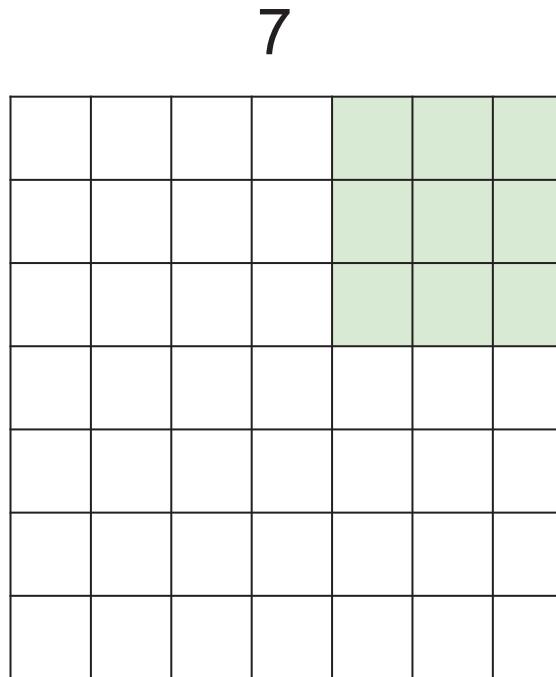
7x7 input (spatially)  
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter

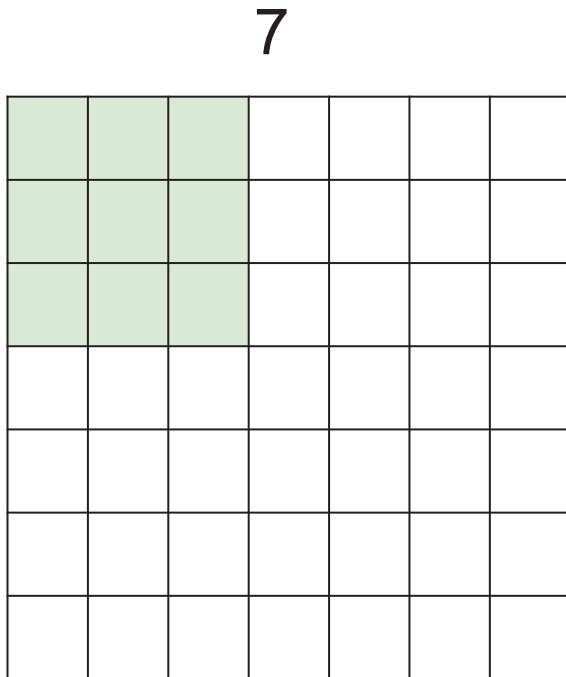
A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter

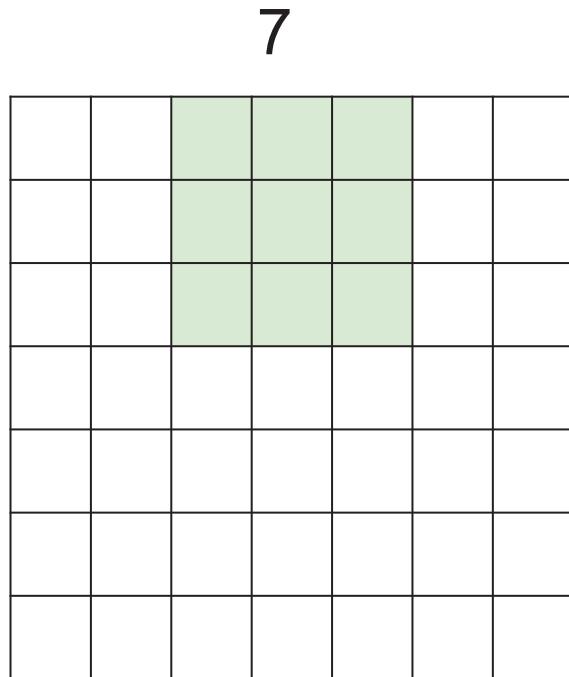
**=> 5x5 output**

A closer look at spatial dimensions:



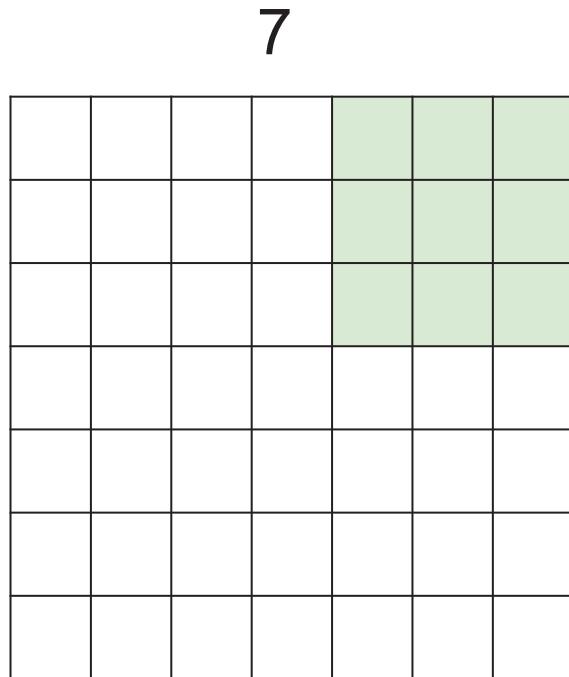
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

A closer look at spatial dimensions:



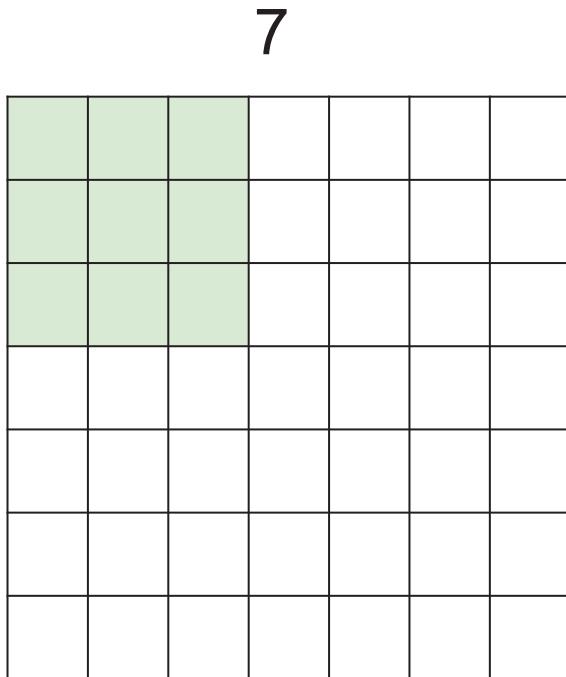
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

A closer look at spatial dimensions:



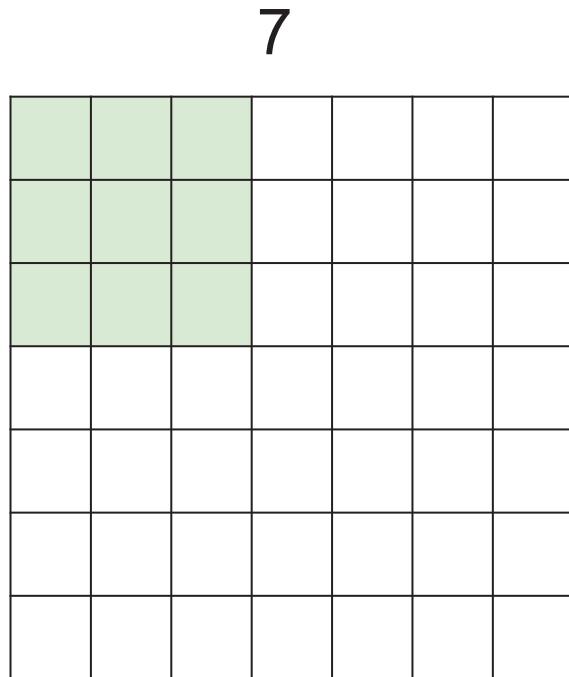
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**  
**=> 3x3 output!**

A closer look at spatial dimensions:



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

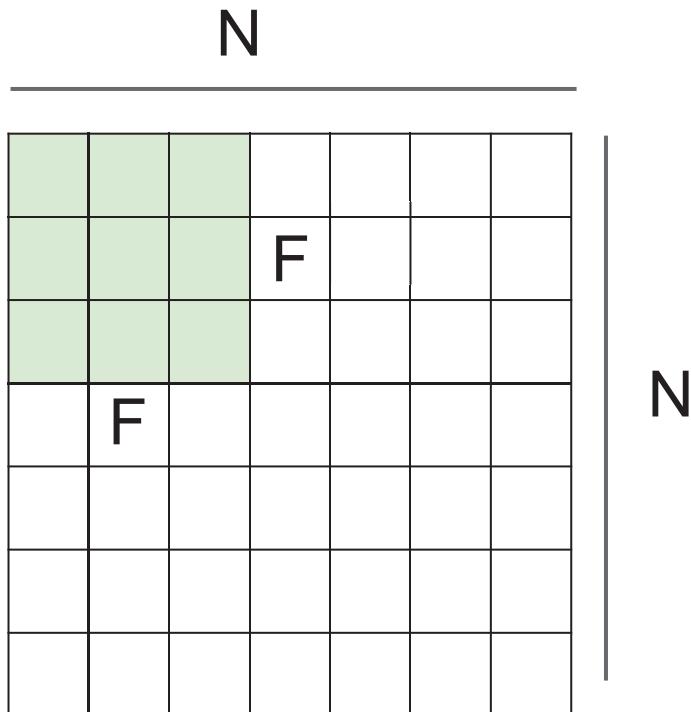
A closer look at spatial dimensions:



7

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

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



Output size:  
 $(N - F) / \text{stride} + 1$

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

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

(recall:)  
$$(N - F) / \text{stride} + 1$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

(recall:)

$$(N + 2P - F) / \text{stride} + 1$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

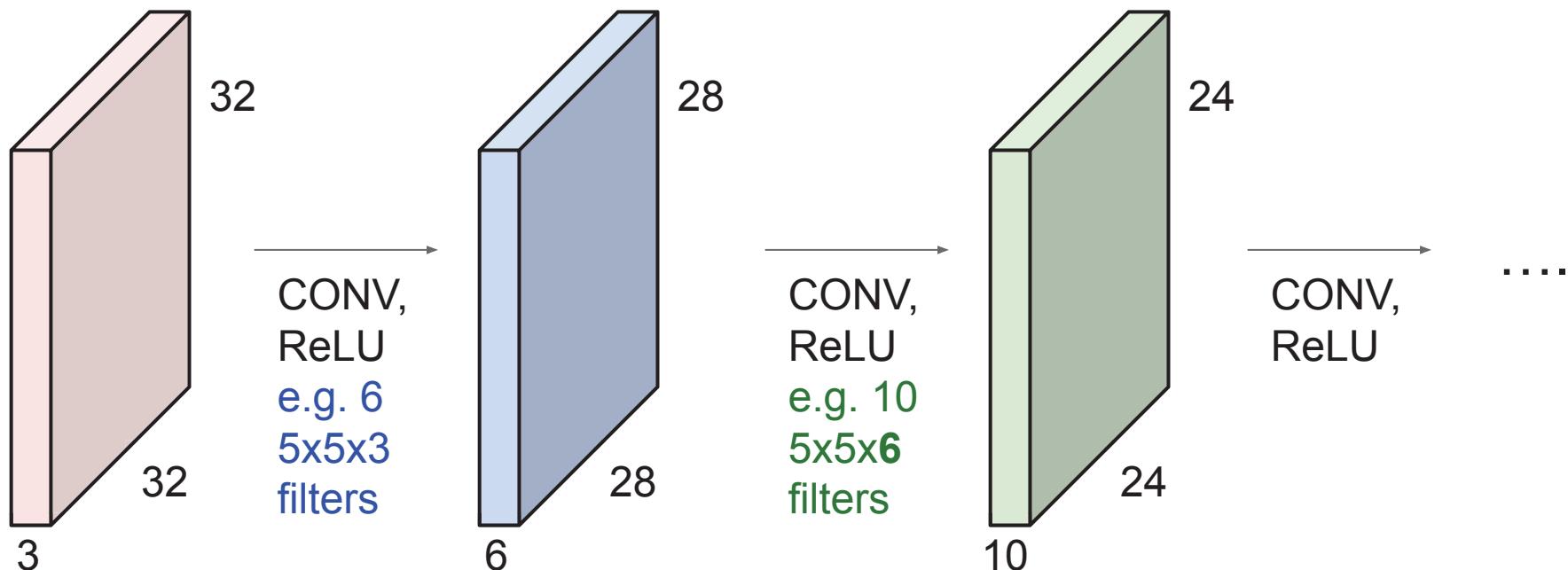
e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

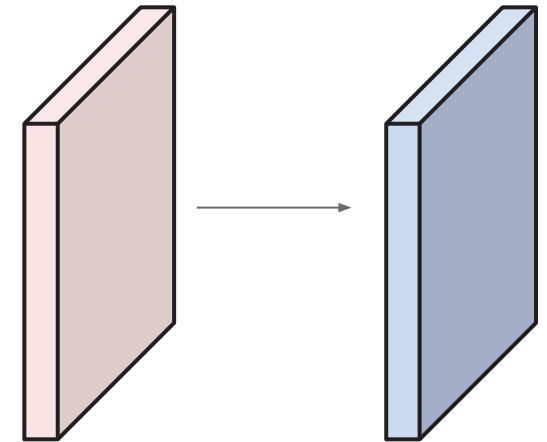
## Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!  
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time:

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

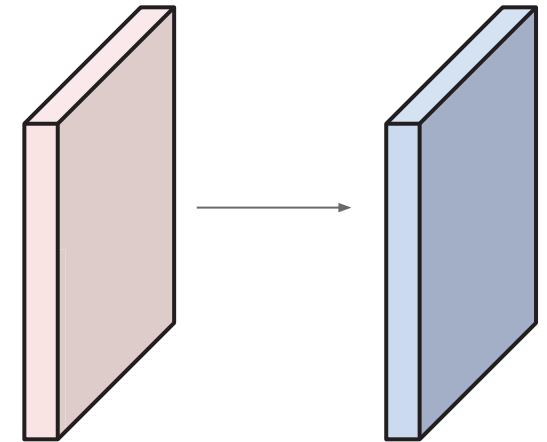


Output volume size: ?

Examples time:

Input volume: **32x32x3**

**10 5x5 filters with stride 1, pad 2**



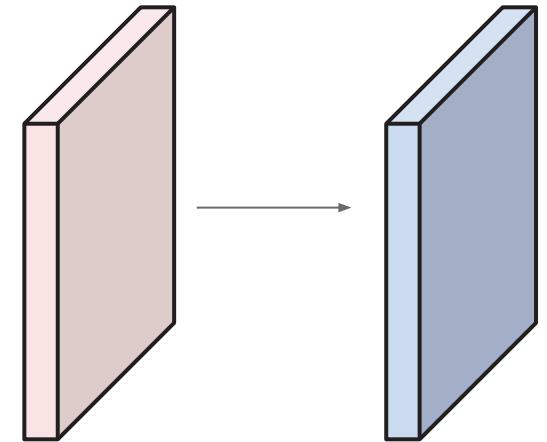
Output volume size:

$(32+2*2-5)/1+1 = 32$  spatially, so

**32x32x10**

Examples time:

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

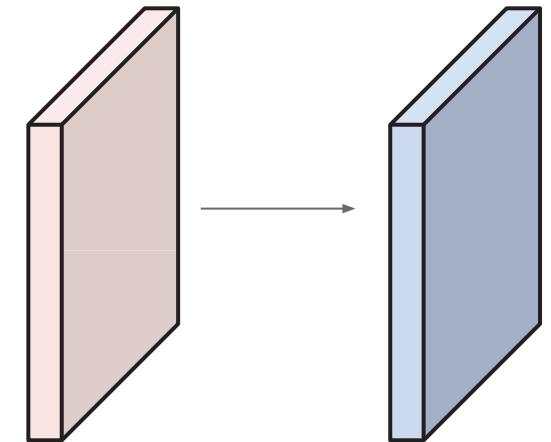


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

**10 5x5 filters with stride 1, pad 2**



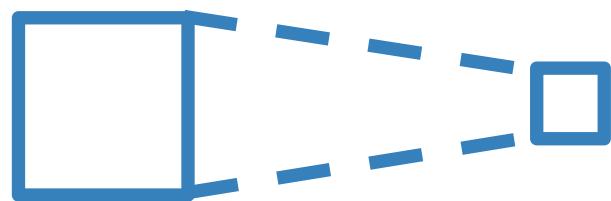
Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params      (+1 for bias)

$$\Rightarrow 76 * 10 = 760$$

## Receptive Fields

For convolution with kernel size K, each element in the output depends on a  $K \times K$  **receptive field** in the input



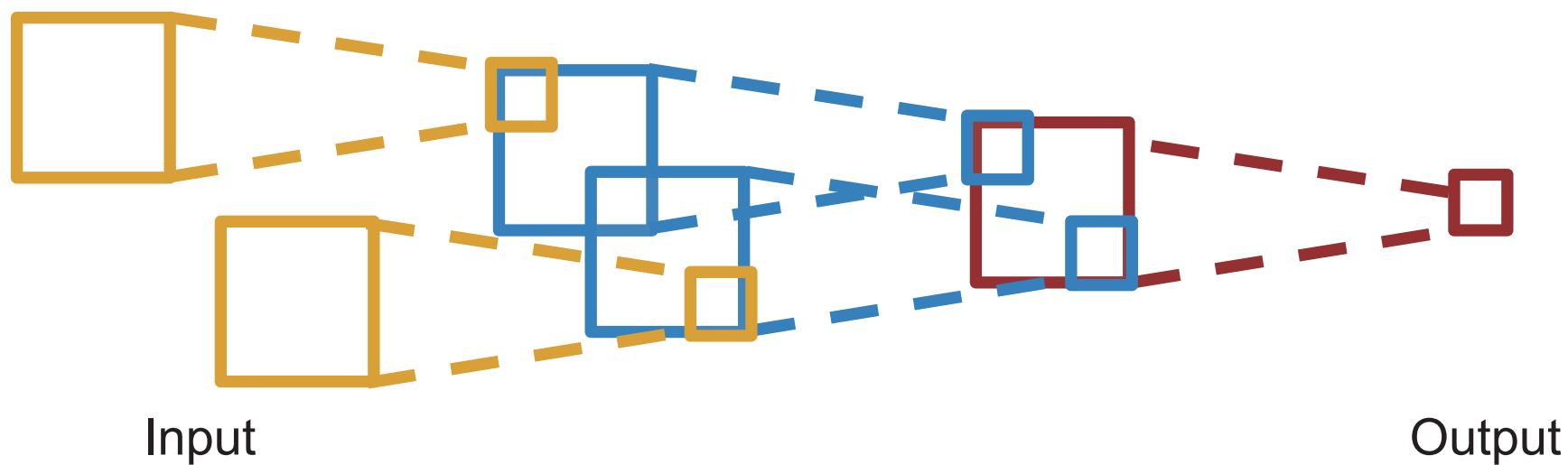
Input

Output

Slide inspiration: Justin Johnson

# Receptive Fields

Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$

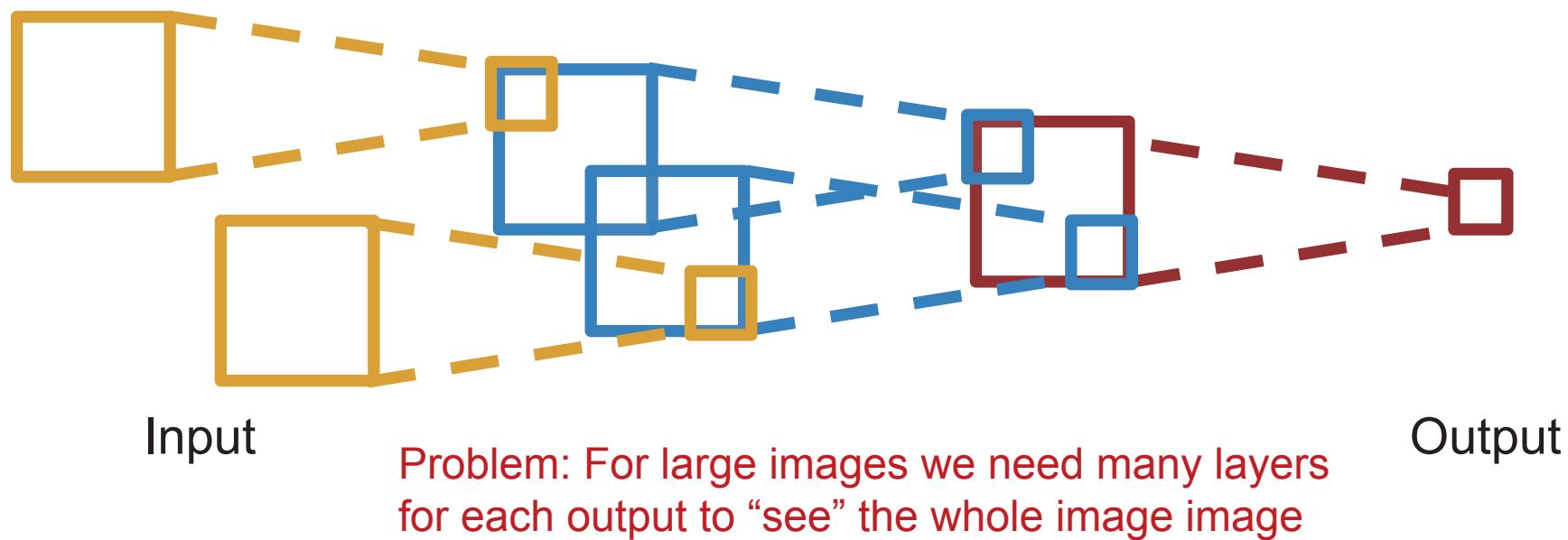


Be careful – “receptive field in the input” vs. “receptive field in the previous layer”

Slide inspiration: Justin Johnson

## Receptive Fields

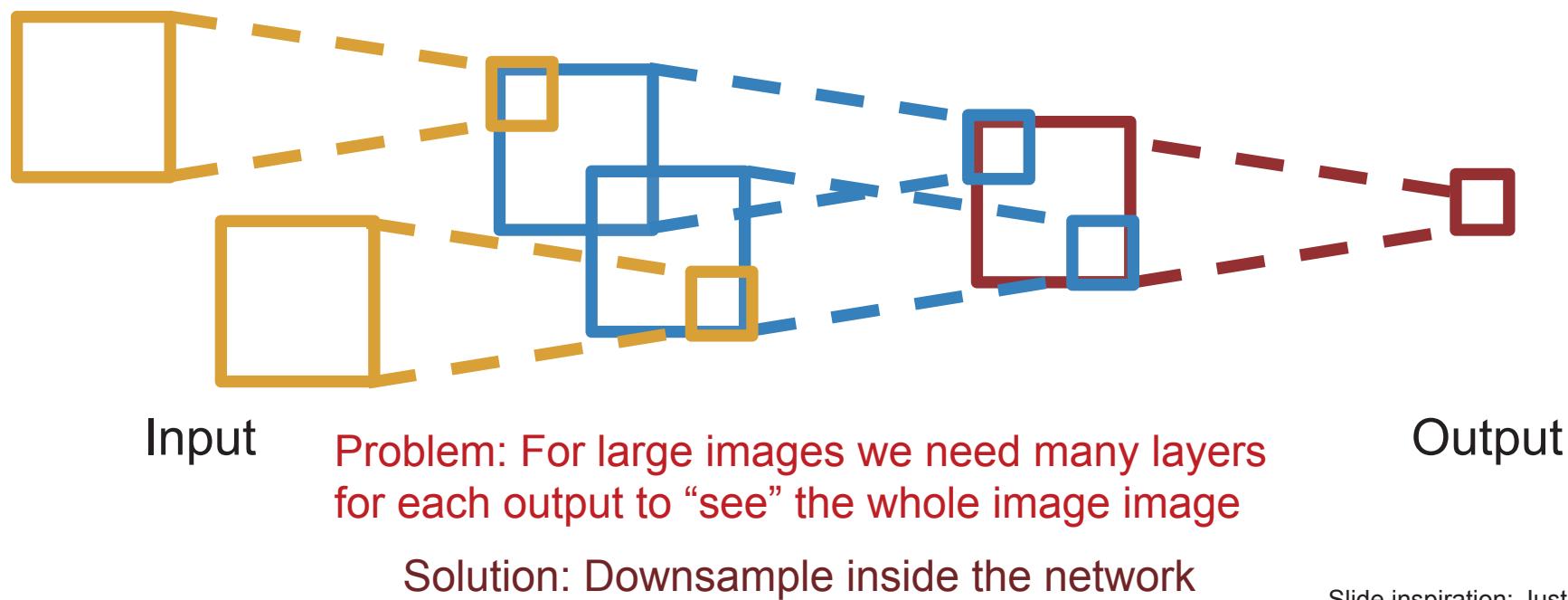
Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



Slide inspiration: Justin Johnson

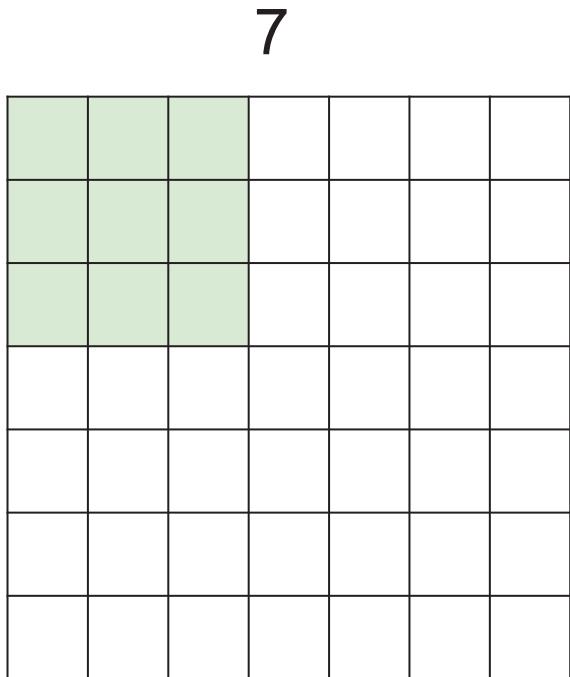
## Receptive Fields

Each successive convolution adds  $K - 1$  to the receptive field size  
With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



Slide inspiration: Justin Johnson

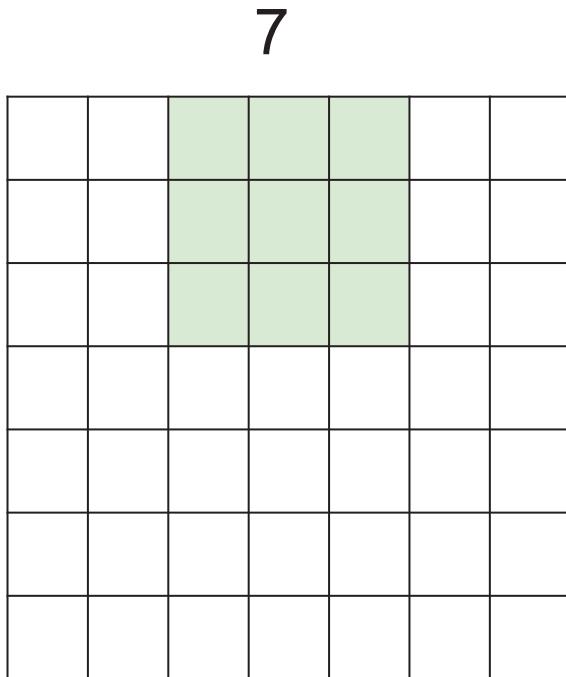
## Solution: Strided Convolution



7

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

## Solution: Strided Convolution



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

=> 3x3 output!

# Convolution layer: summary

Let's assume input is  $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of  $W_2 \times H_2 \times K$

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters:  $F^2CK$  and  $K$  biases

## Convolution layer: summary

Common settings:

Let's assume input is  $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

**K** = (powers of 2, e.g. 32, 64, 128, 512)

- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$  (whatever fits)
- $F = 1, S = 1, P = 0$

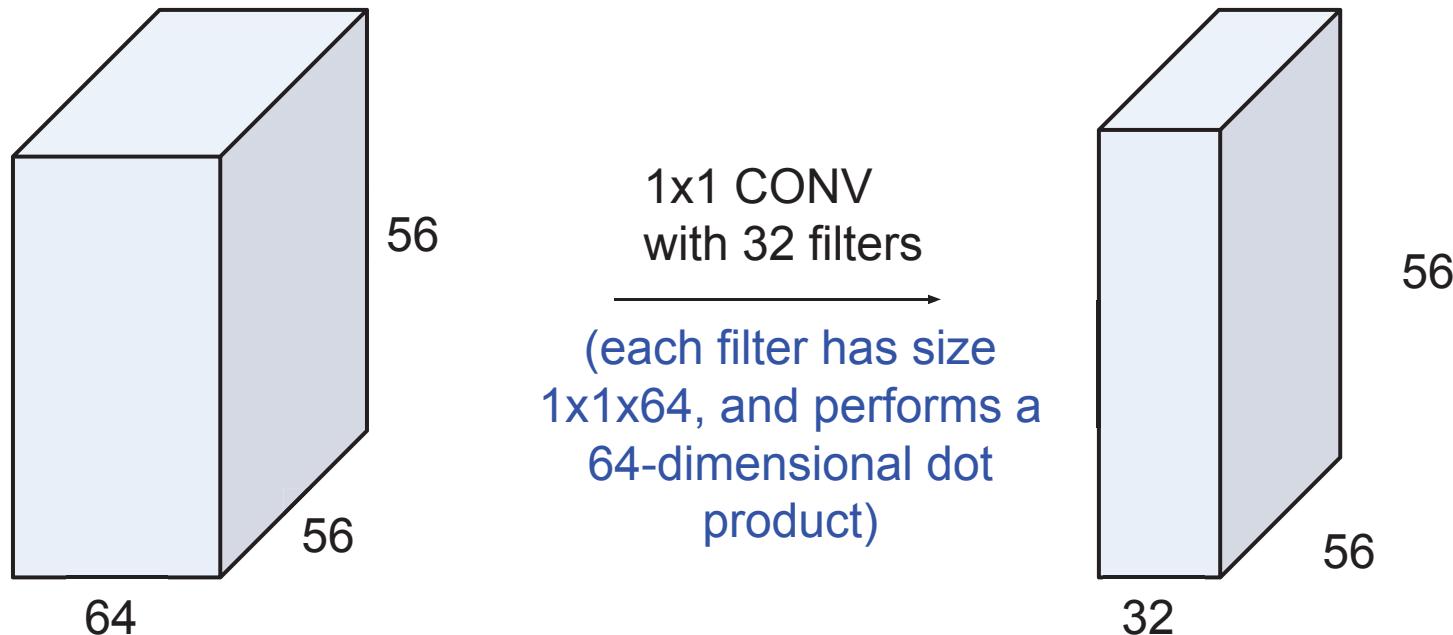
This will produce an output of  $W_2 \times H_2 \times K$

where:

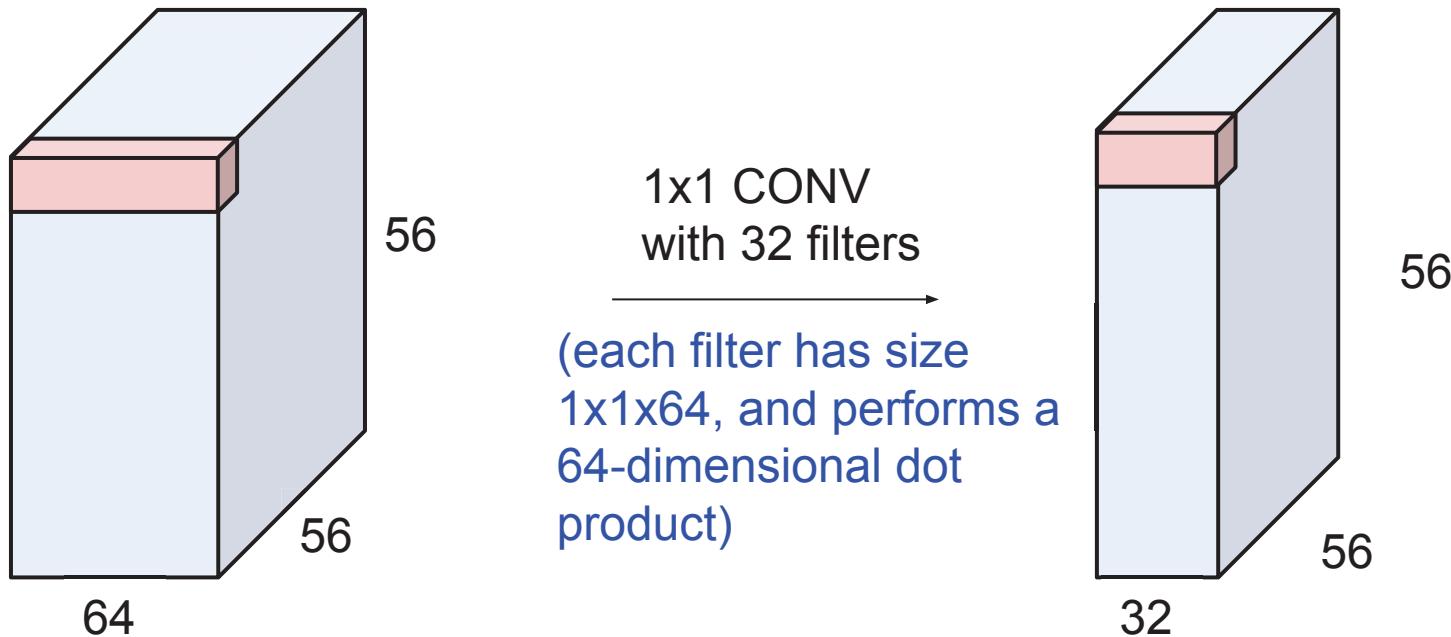
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters:  $F^2CK$  and  $K$  biases

(btw, 1x1 convolution layers make perfect sense)



(btw, 1x1 convolution layers make perfect sense)



# Example: CONV layer in PyTorch

## Conv2d

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,  
dilation=1, groups=1, bias=True)
```

[SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size  $(N, C_{\text{in}}, H, W)$  and output  $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$  can be precisely described as:

$$\text{out}(N_i, C_{\text{out},j}) = \text{bias}(C_{\text{out},j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out},j}, k) * \text{input}(N_i, k)$$

where  $*$  is the valid 2D cross-correlation operator,  $N$  is a batch size,  $C$  denotes a number of channels,  $H$  is a height of input planes in pixels, and  $W$  is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At `groups=1`, all inputs are convolved to all outputs.
  - At `groups=2`, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At `groups= in_channels`, each input channel is convolved with its own set of filters, of size:  $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$ .

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

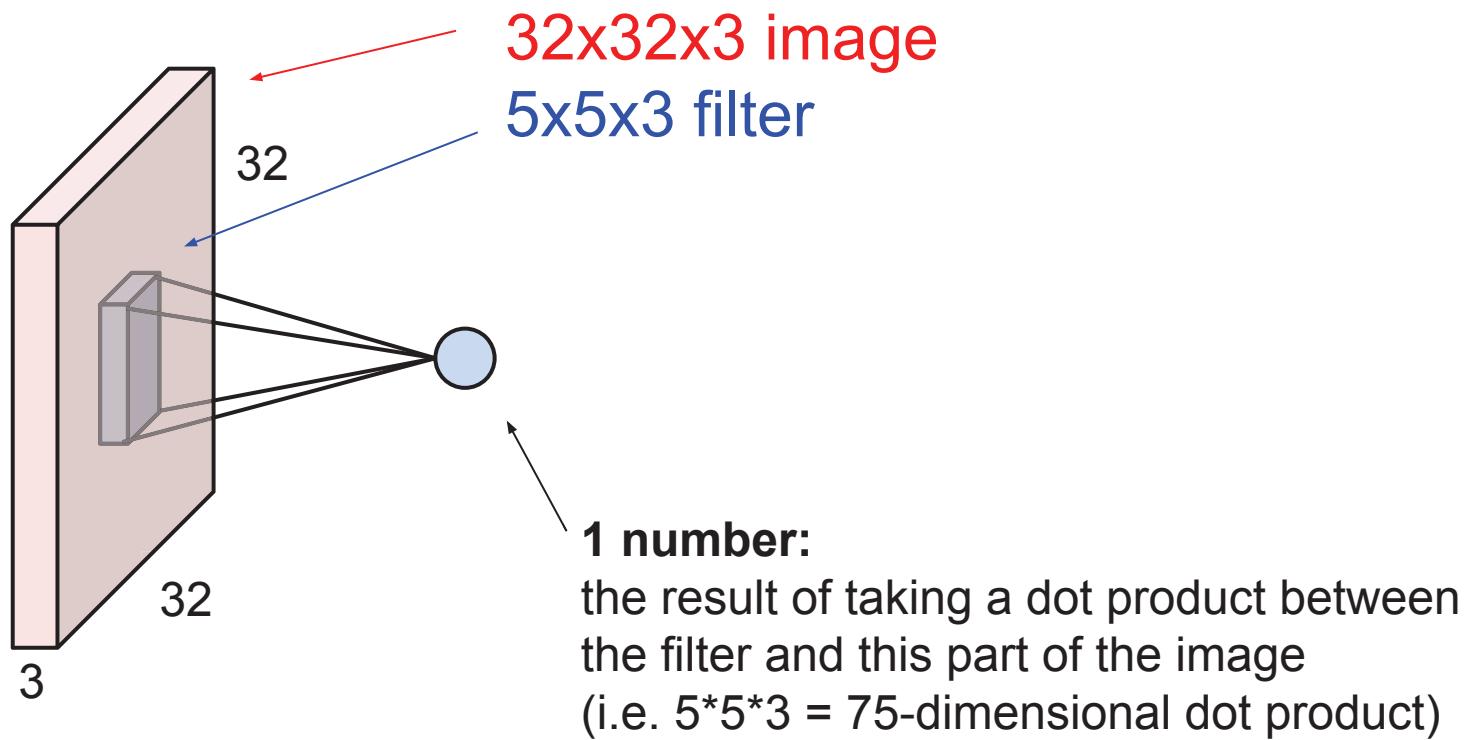
- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two `ints` – in which case, the first `int` is used for the height dimension, and the second `int` for the width dimension

[PyTorch](#) is licensed under [BSD 3-clause](#).

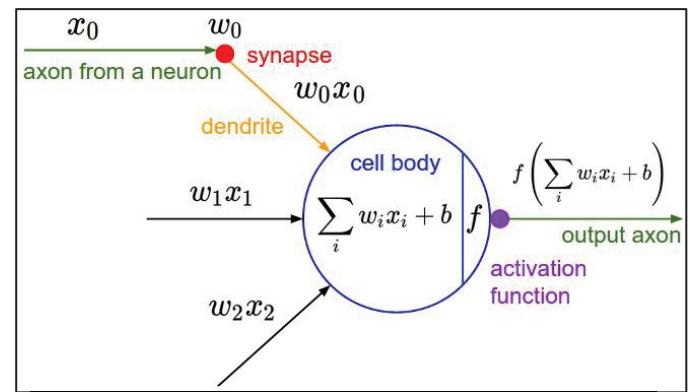
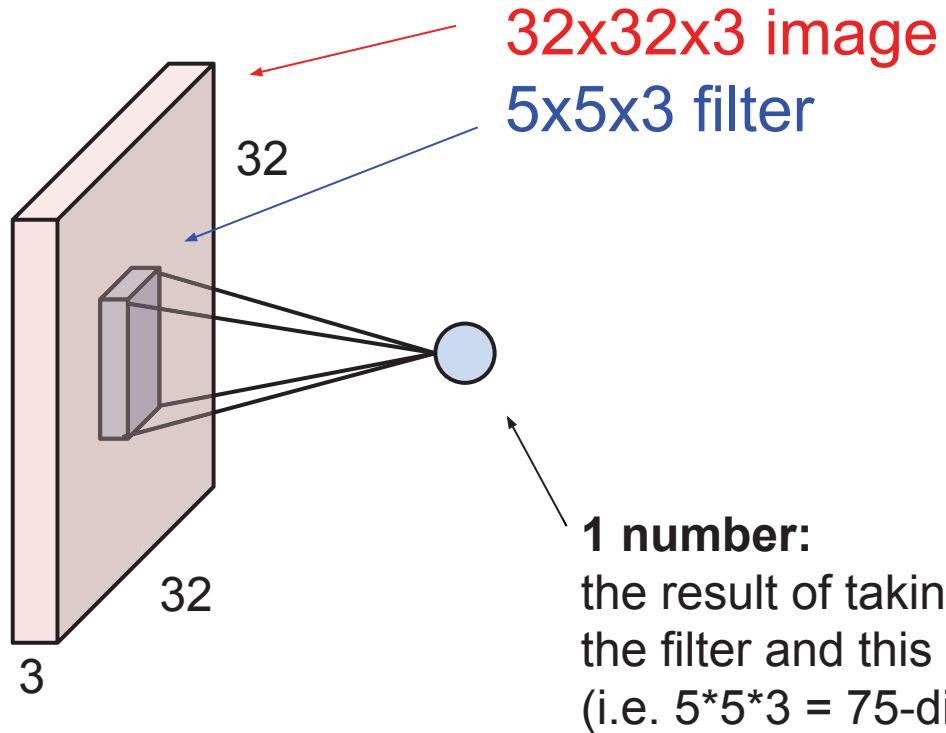
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

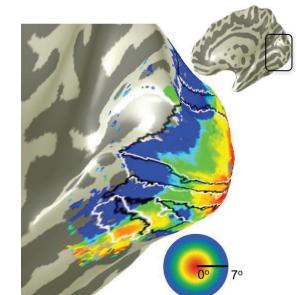
## The brain/neuron view of CONV Layer



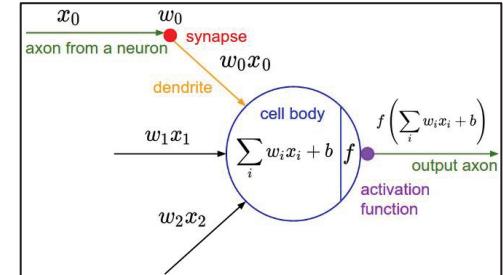
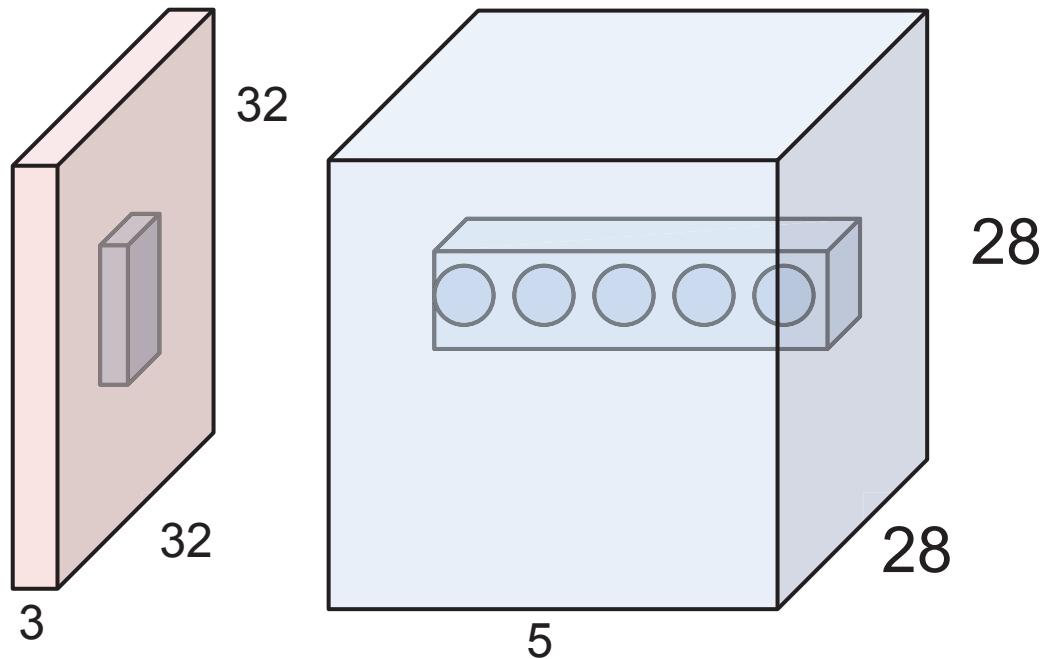
## The brain/neuron view of CONV Layer



It's just a neuron with local connectivity...



## The brain/neuron view of CONV Layer



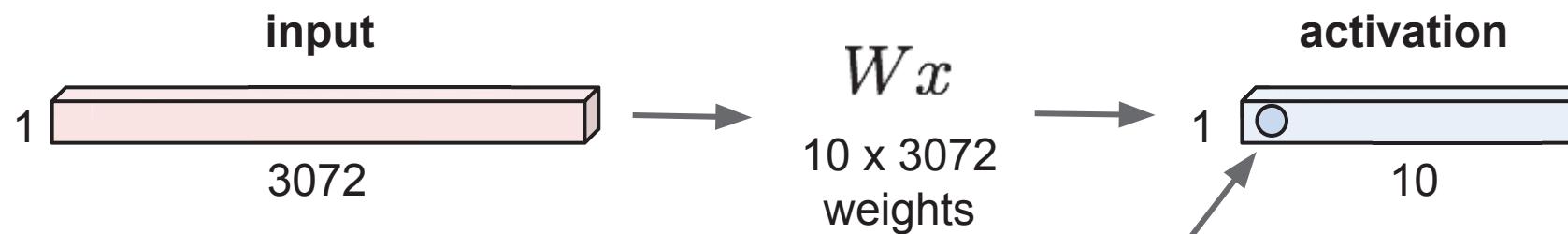
E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
( $28 \times 28 \times 5$ )

There will be 5 different  
neurons all looking at the same  
region in the input volume

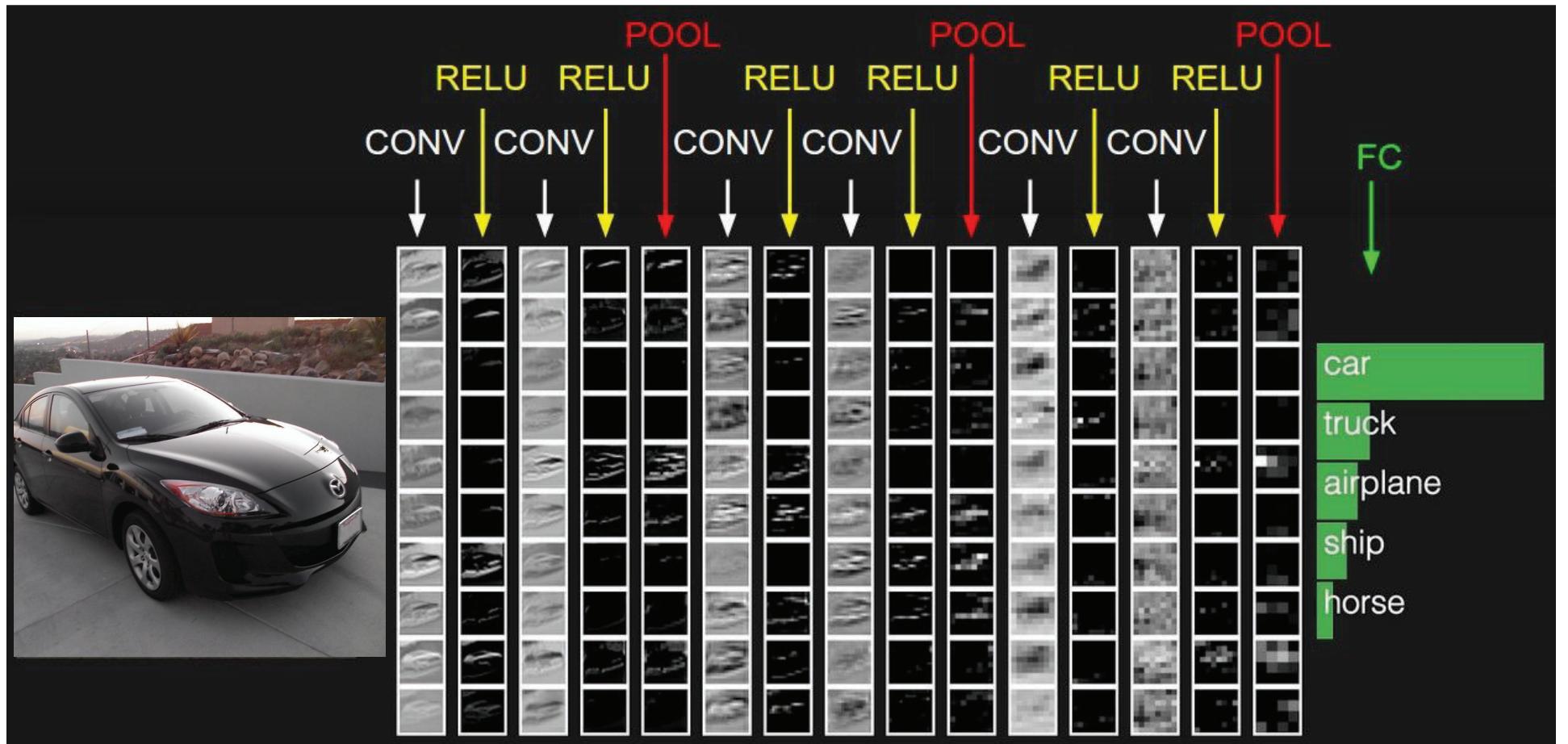
# Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron  
looks at the full  
input volume

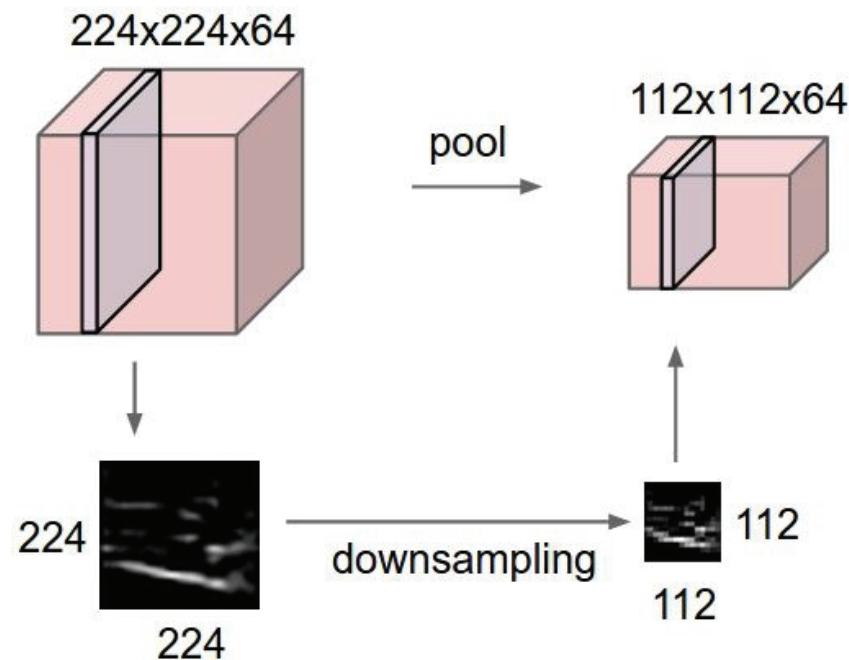


**1 number:**  
the result of taking a dot product  
between a row of  $W$  and the input  
(a 3072-dimensional dot product)

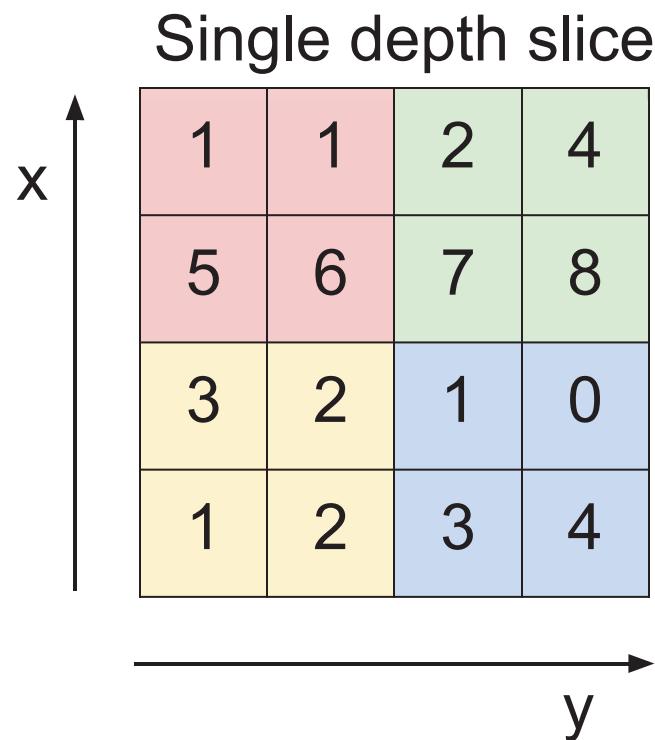


# Pooling layer

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



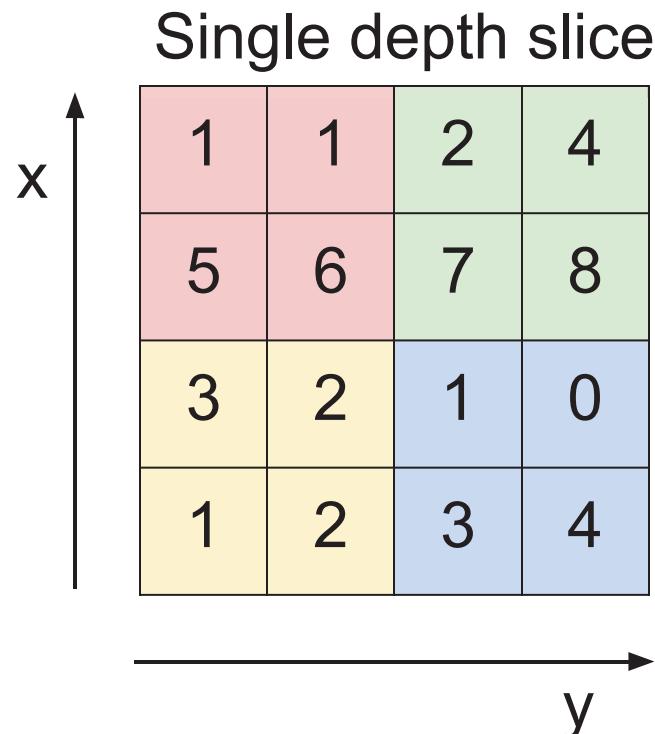
# MAX POOLING



max pool with 2x2 filters  
and stride 2

6	8
3	4

# MAX POOLING



max pool with 2x2 filters  
and stride 2



6	8
3	4

- No learnable parameters
- Introduces spatial invariance

## Pooling layer: summary

Let's assume input is  $W_1 \times H_1 \times C$

Conv layer needs 2 hyperparameters:

- The spatial extent **F**
- The stride **S**

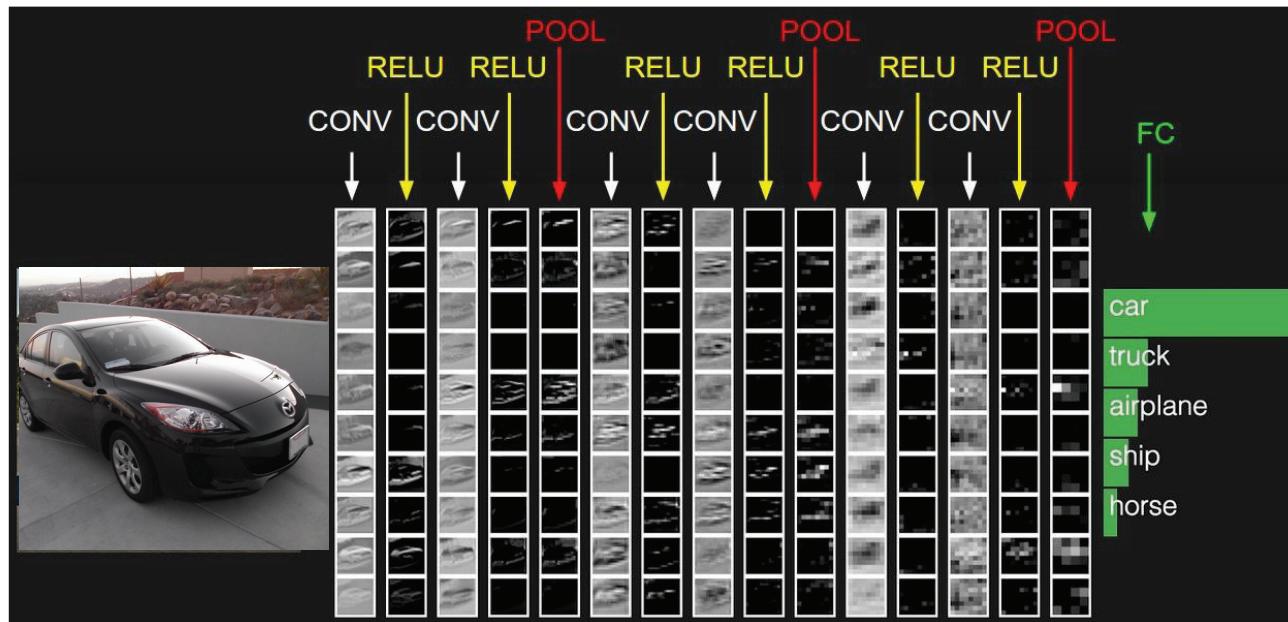
This will produce an output of  $W_2 \times H_2 \times C$  where:

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

# Fully Connected Layer (FC layer)

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



# [ConvNetJS demo: training on CIFAR-10]

## [ConvNetJS CIFAR-10 demo](#)

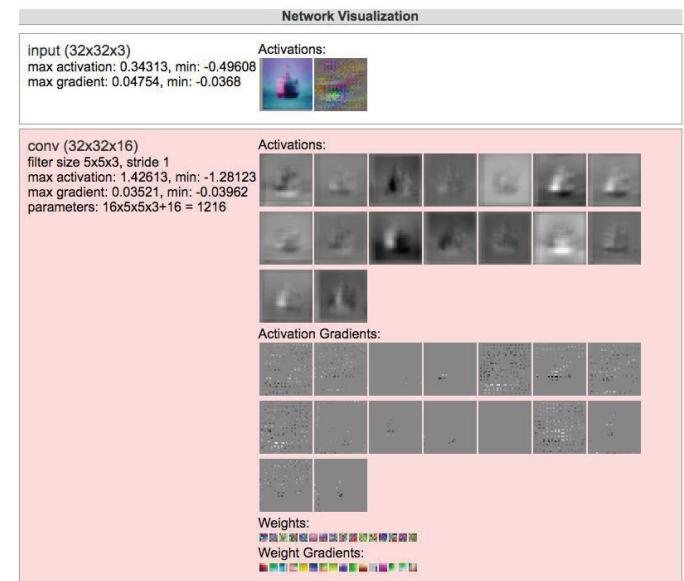
### Description

This demo trains a Convolutional Neural Network on the [CIFAR-10 dataset](#) in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used [this python script](#) to parse the [original files](#) (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to [@karpathy](#).



<http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html>

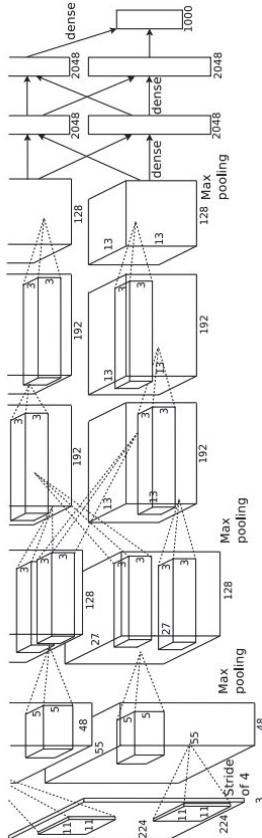
# Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like  
 **$[(CONV-RELU)*N-POOL?] * M - (FC-RELU)*K, SOFTMAX$**   
where N is usually up to ~5, M is large,  $0 \leq K \leq 2$ .
- But recent advances such as ResNet/GoogLeNet have challenged this paradigm

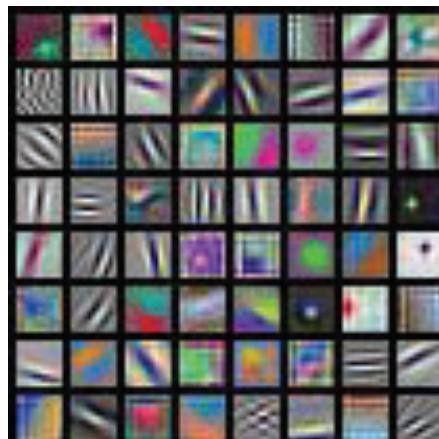
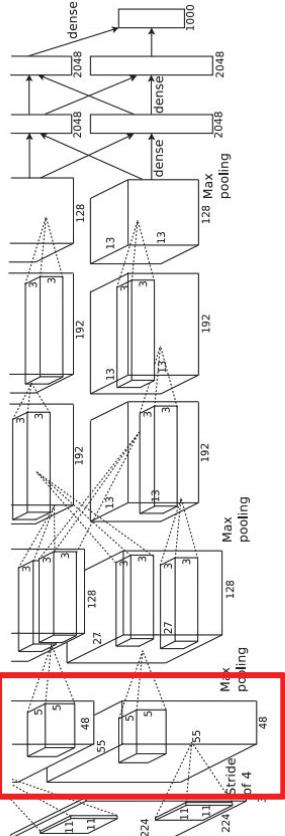
# Transfer learning

You need a lot of a data if you want to  
train/use CNNs?

# Transfer Learning with CNNs



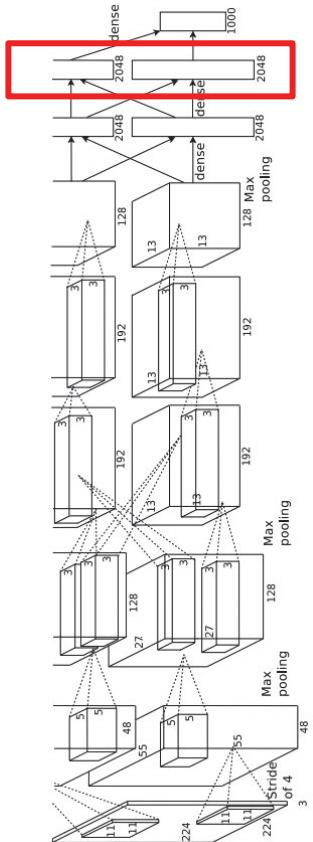
# Transfer Learning with CNNs



AlexNet:  
64 x 3 x 11 x 11

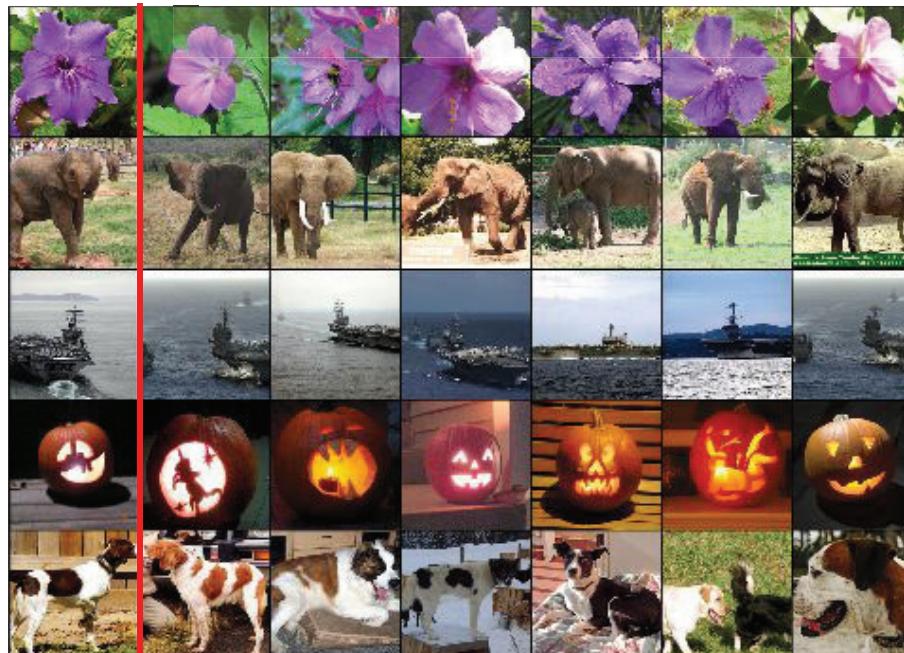
(More on this in Lecture 13)

# Transfer Learning with CNNs



Test image

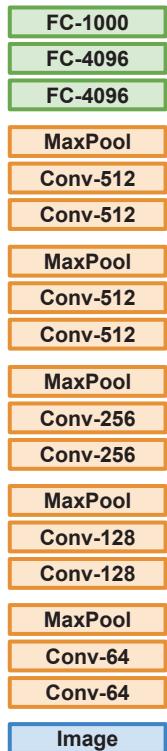
L2 Nearest neighbors in feature space



(More on this in Lecture 13)

# Transfer Learning with CNNs

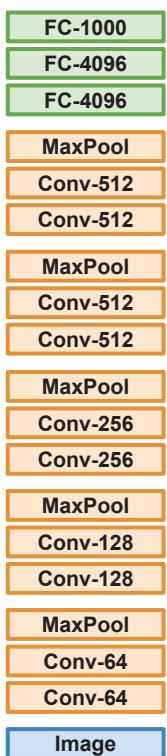
## 1. Train on Imagenet



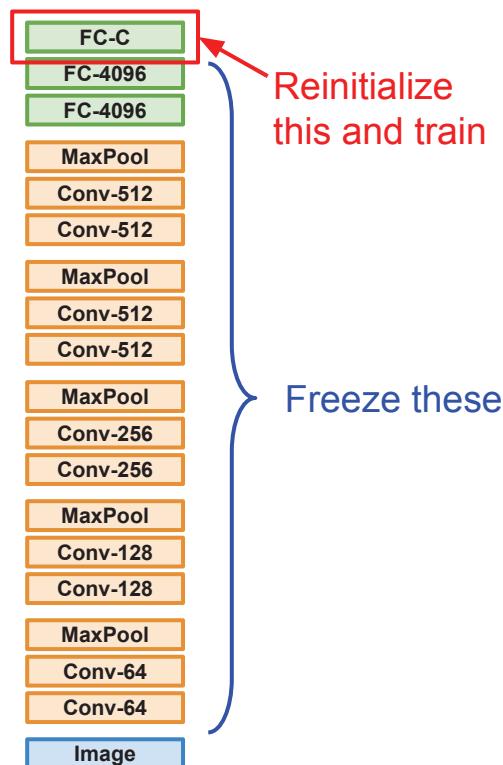
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

# Transfer Learning with CNNs

## 1. Train on Imagenet



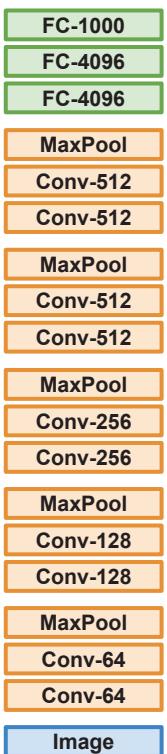
## 2. Small Dataset (C classes)



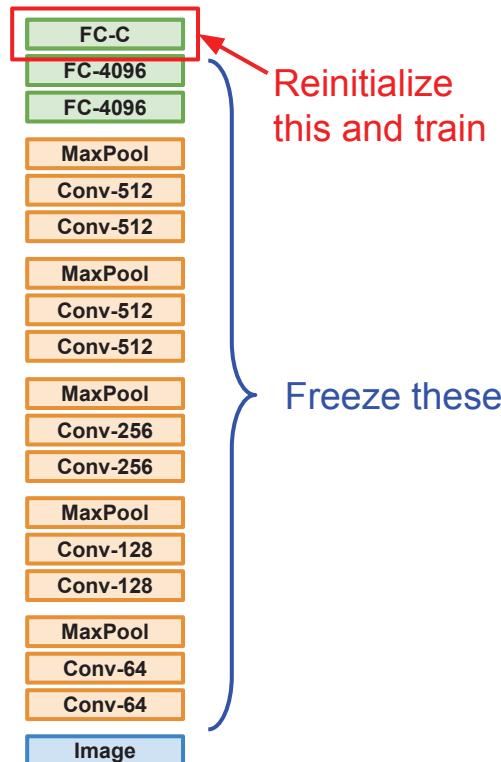
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

# Transfer Learning with CNNs

## 1. Train on Imagenet

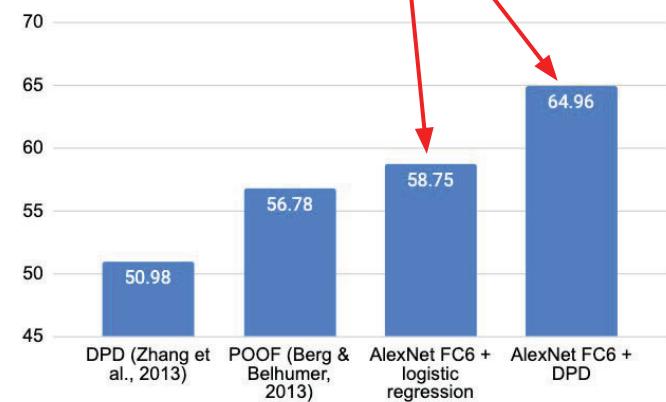


## 2. Small Dataset (C classes)



Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

Finetuned from AlexNet

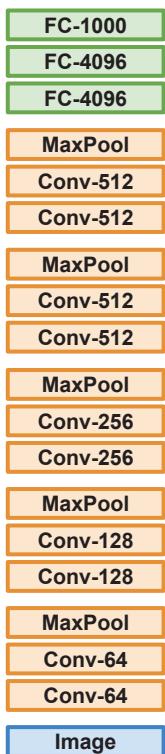


Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

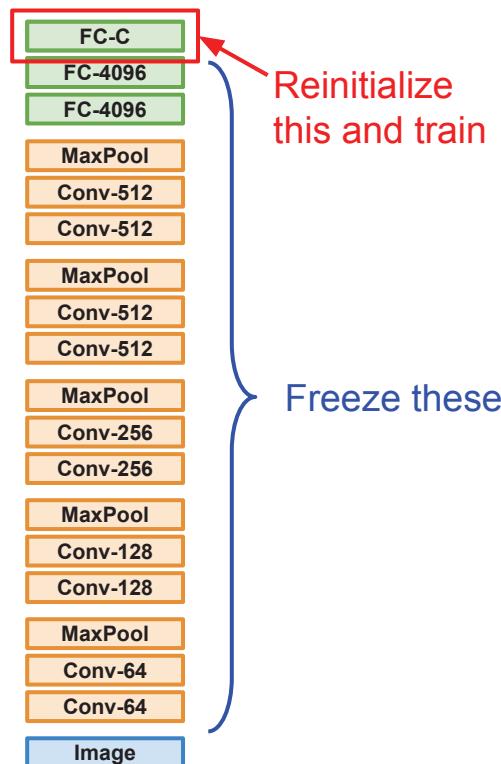
# Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014  
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

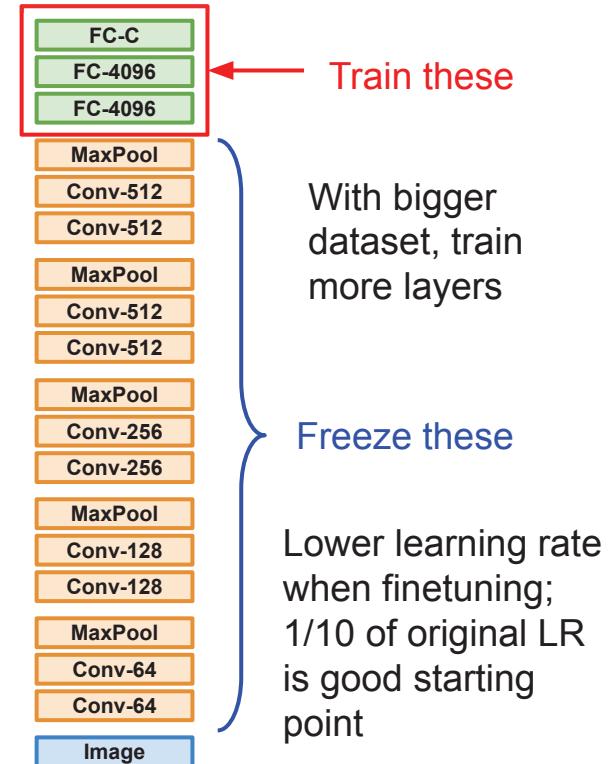
## 1. Train on Imagenet

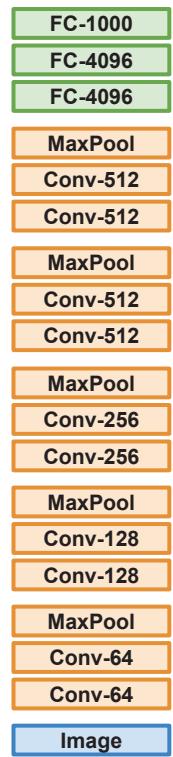


## 2. Small Dataset (C classes)

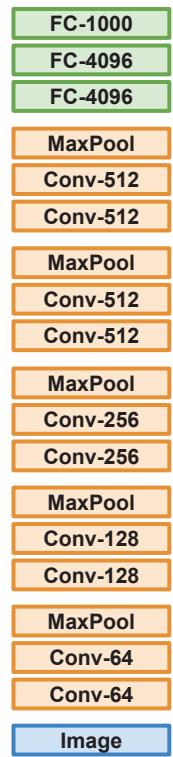


## 3. Bigger dataset



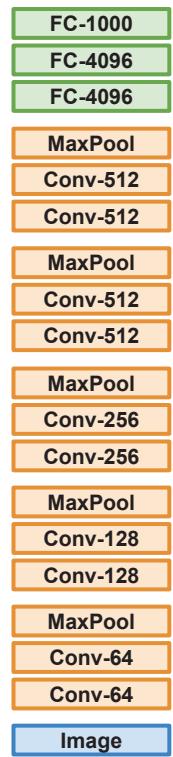


	<b>very similar dataset</b>	<b>very different dataset</b>
<b>very little data</b>	?	?
<b>quite a lot of data</b>	?	?



More specific  
More generic

	<b>very similar dataset</b>	<b>very different dataset</b>
<b>very little data</b>	Use Linear Classifier on top layer	?
<b>quite a lot of data</b>	Finetune a few layers	?

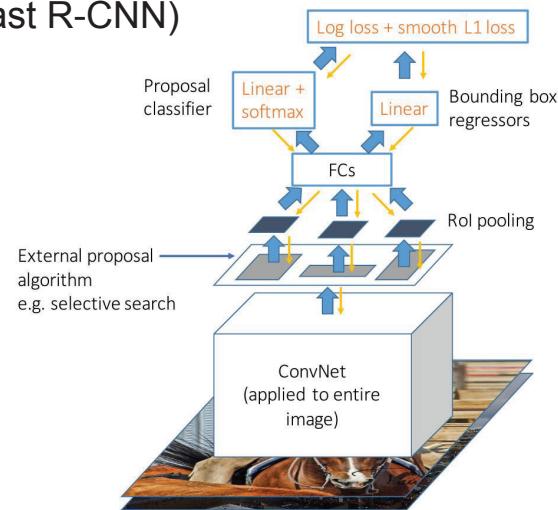


More specific  
More generic

	<b>very similar dataset</b>	<b>very different dataset</b>
<b>very little data</b>	Use Linear Classifier on top layer	You're in trouble... Try linear classifier from different stages
<b>quite a lot of data</b>	Finetune a few layers	Finetune a larger number of layers

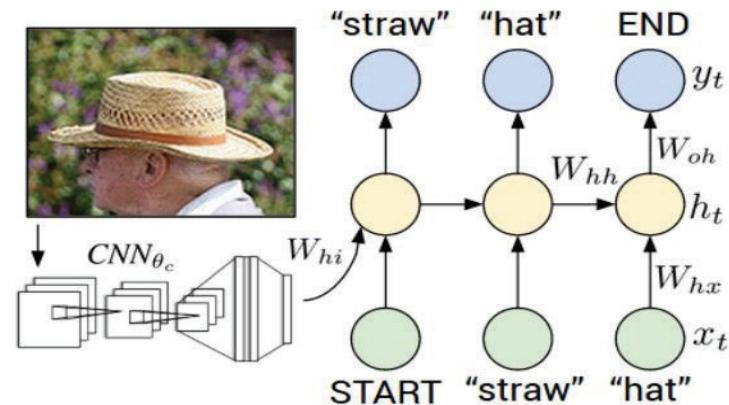
# Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

Object Detection  
(Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015  
Figure copyright Ross Girshick, 2015. Reproduced with permission.

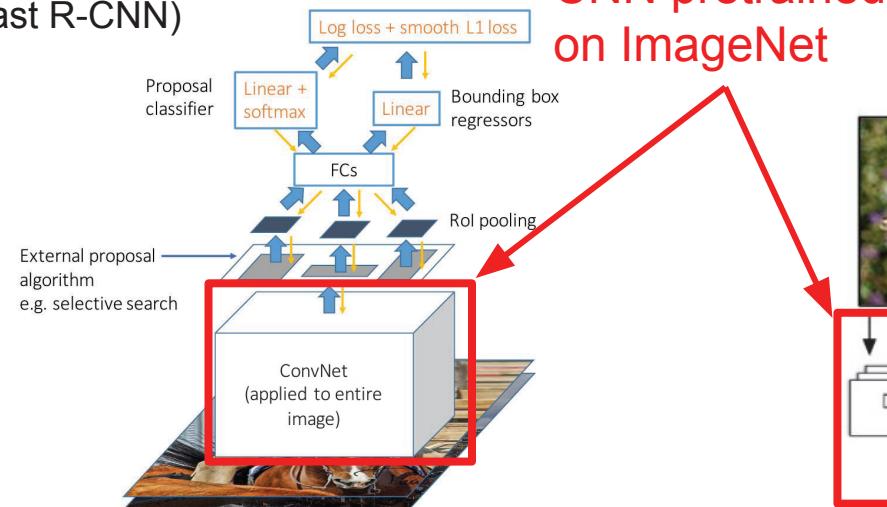
Image Captioning: CNN + RNN



Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015  
Figure copyright IEEE, 2015. Reproduced for educational purposes.

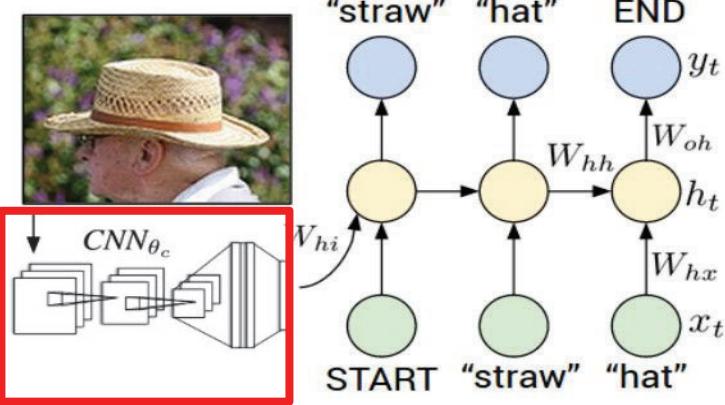
# Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

Object Detection  
(Fast R-CNN)



CNN pretrained  
on ImageNet

Image Captioning: CNN + RNN

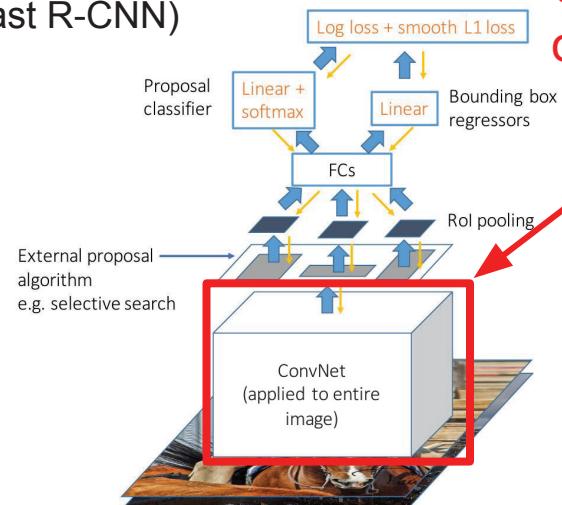


Girshick, "Fast R-CNN", ICCV 2015  
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015  
Figure copyright IEEE, 2015. Reproduced for educational purposes.

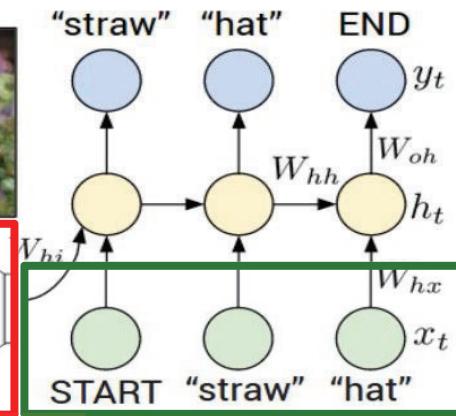
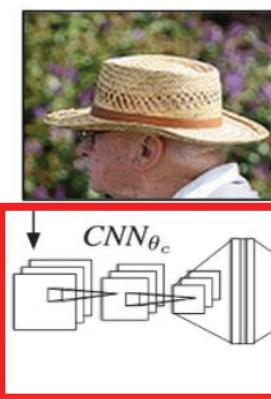
# Transfer learning with CNNs is pervasive... (it's the norm, not an exception)

Object Detection  
(Fast R-CNN)



CNN pretrained  
on ImageNet

Image Captioning: CNN + RNN

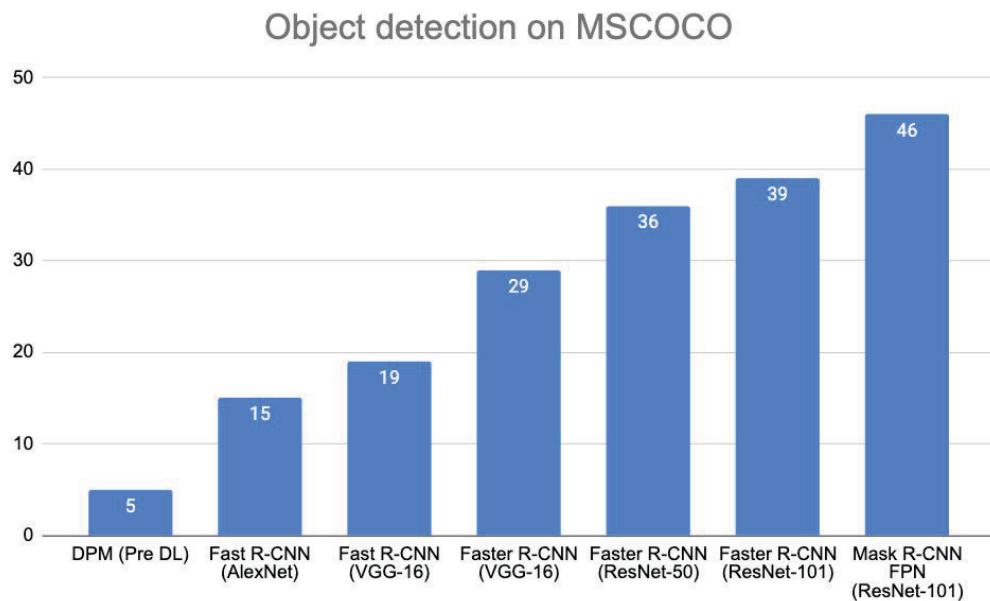


Word vectors pretrained  
with word2vec

Girshick, "Fast R-CNN", ICCV 2015  
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015  
Figure copyright IEEE, 2015. Reproduced for educational purposes.

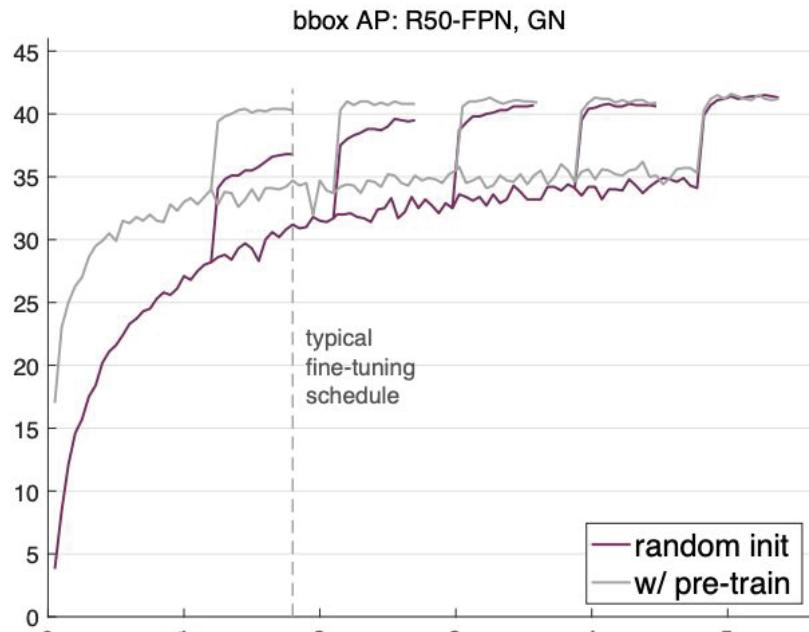
# Transfer learning with CNNs - Architecture matters



Girshick, "The Generalized R-CNN Framework for Object Detection", ICCV 2017 Tutorial on Instance-Level Visual Recognition

# Transfer learning with CNNs is pervasive...

## But recent results show it might not always be necessary!



He et al, "Rethinking ImageNet Pre-training", ICCV 2019  
Figure copyright Kaiming He, 2019. Reproduced with permission.

Training from scratch can work just as well as training from a pretrained ImageNet model for object detection

But it takes 2-3x as long to train.

They also find that collecting more data is better than finetuning on a related task

## **Takeaway for your projects and beyond:**

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

TensorFlow: <https://github.com/tensorflow/models>

PyTorch: <https://github.com/pytorch/vision>