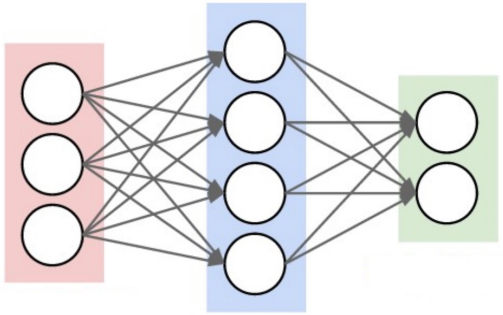
Algorithms for Modern Artificial Intelligence (NeoScholar Spring Program)

Lec 2A: Machine Learning and Neural Networks



Haifeng Xu

Professor of Computer Science and Data Science

University of Chicago

Overview

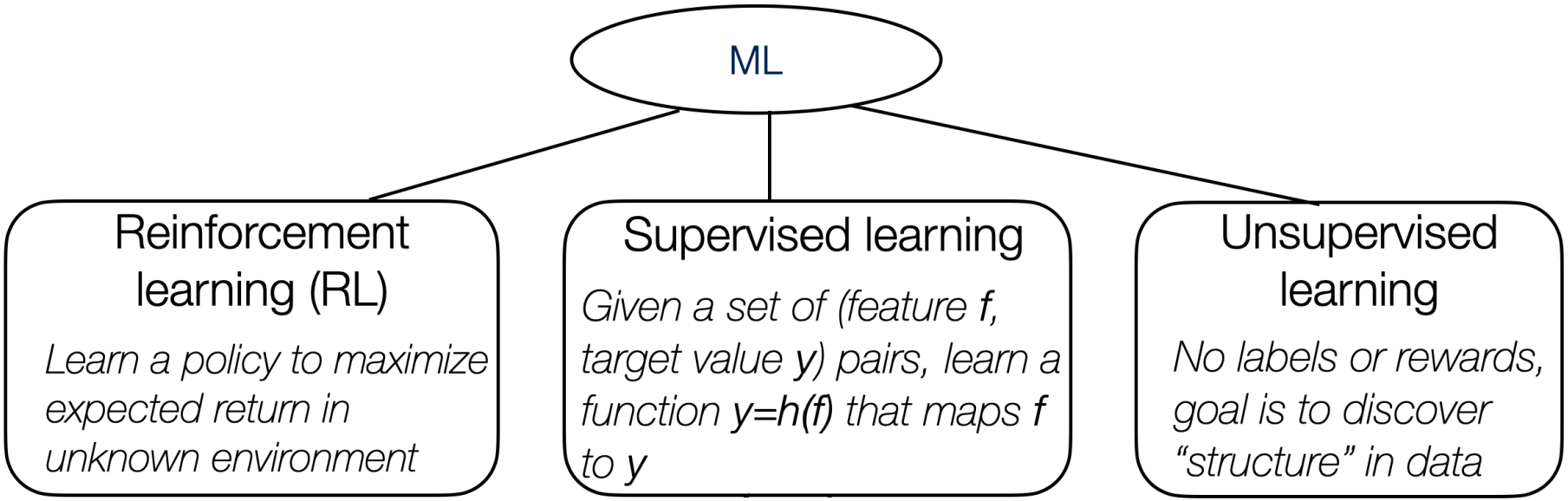
Last Lecture: Overview of classic AI research

This Lecture: beginning of modern AI, centered around machine learning

For those who are thinking of research projects

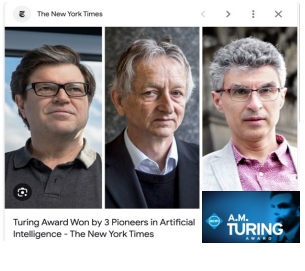
- Each of covered topics from here have active research
- More towards later, more active the research area is (particularly from Lec 4)
- Materials in this course will cover basics that help you to understand what problems each research area studies

Three Major ML Paradigms

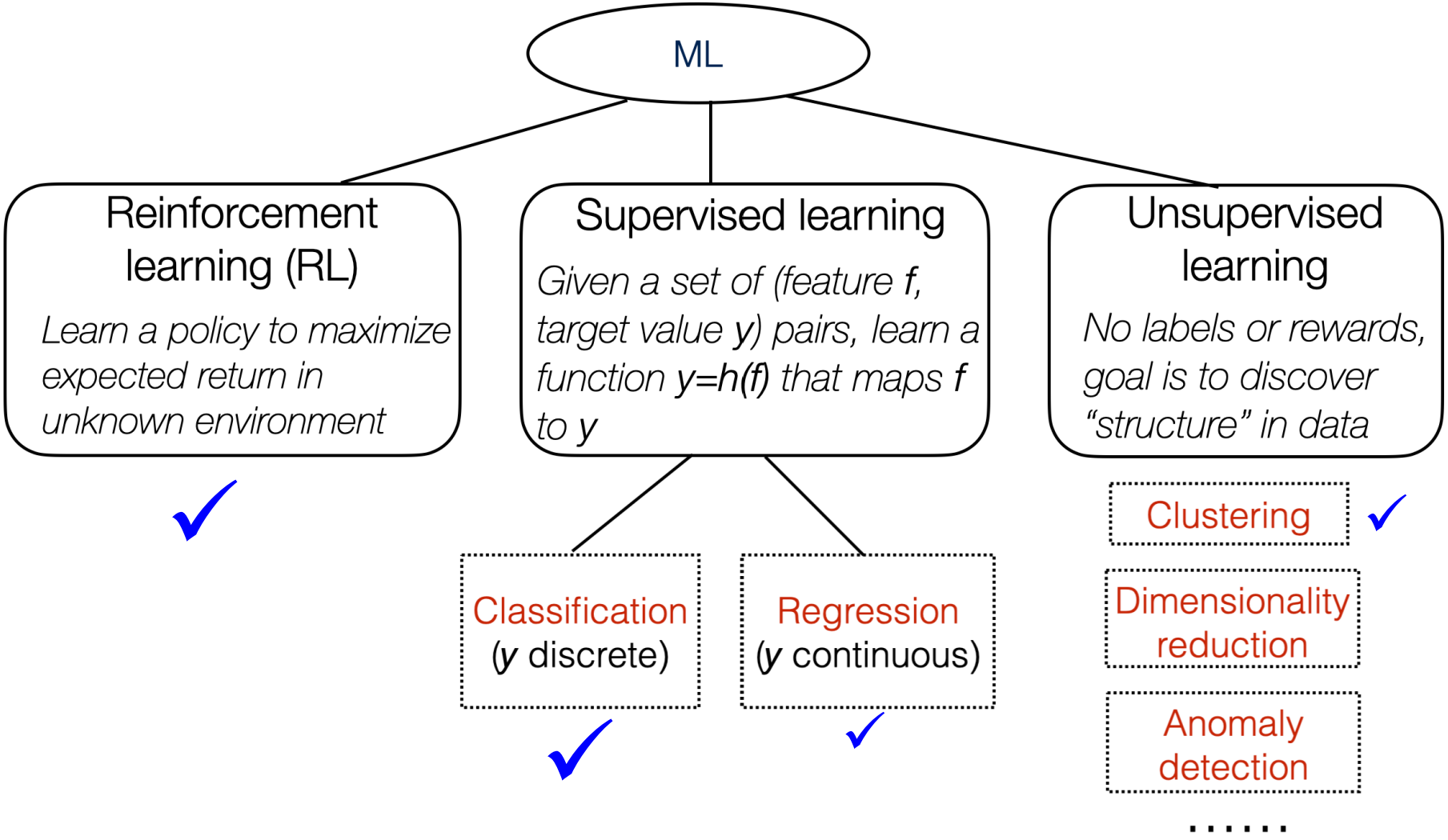


- Iteratively integrate **reward** feedback to improve
- Reinforce good actions and discard bad ones
- Like human the most

- Fundamentally is about **function fitting** from examples (supervisions)
- Give rises to deep learning – fitting complex functions via networks, and learn parameters via massive examples and **optimization**



Three Major ML Paradigms

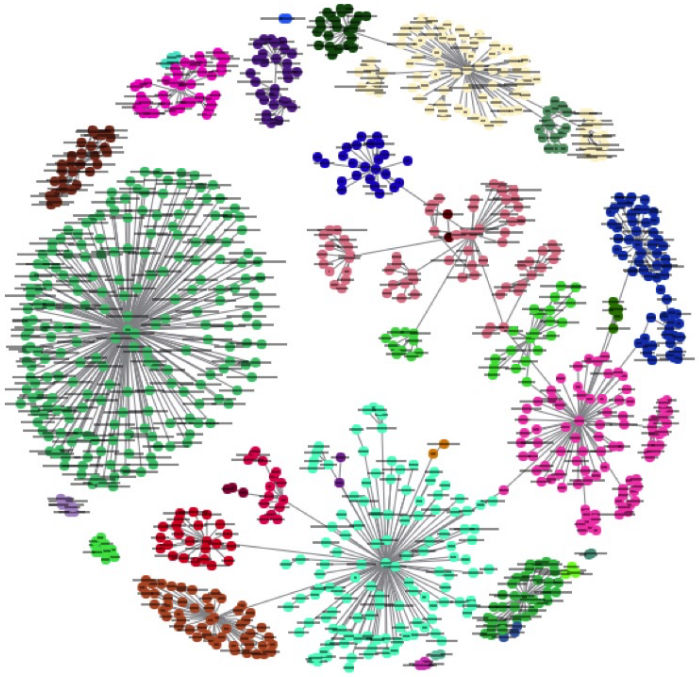


Outline

- Introduction to Clustering
- Supervised Learning and Linear Models
- Neural Networks

Clustering

- Clustering —detect community patterns in unlabeled data
  - E.g. detect communities on FB
  - E.g. group news or movies
  - "Unsupervised" learning

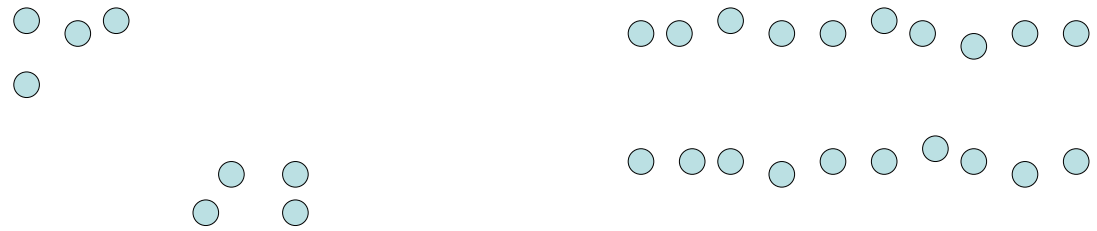


Clustering Examples

- Group customers based on their purchase histories
- Group news into categories
- Identify crime localities
- Identify communities on social networks
- Group movies/tv shows based on viewing histories
- Identify product groups based on the sets of customers who purchased them
- . . . . . .

Clustering

- Basic idea: group similar instances together
- Example: 2D point patterns

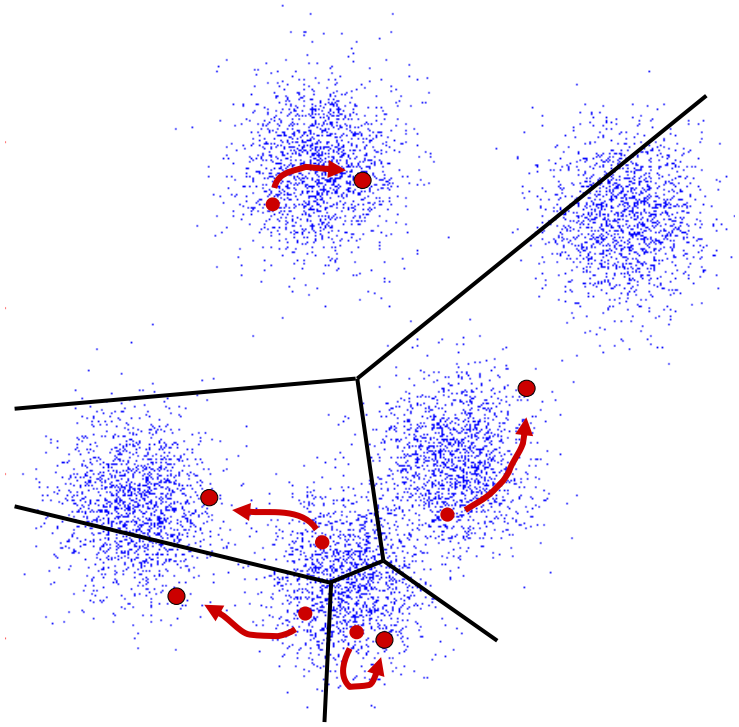


- What could “similar” mean?
  - The canonic option: small (squared) Euclidean distance

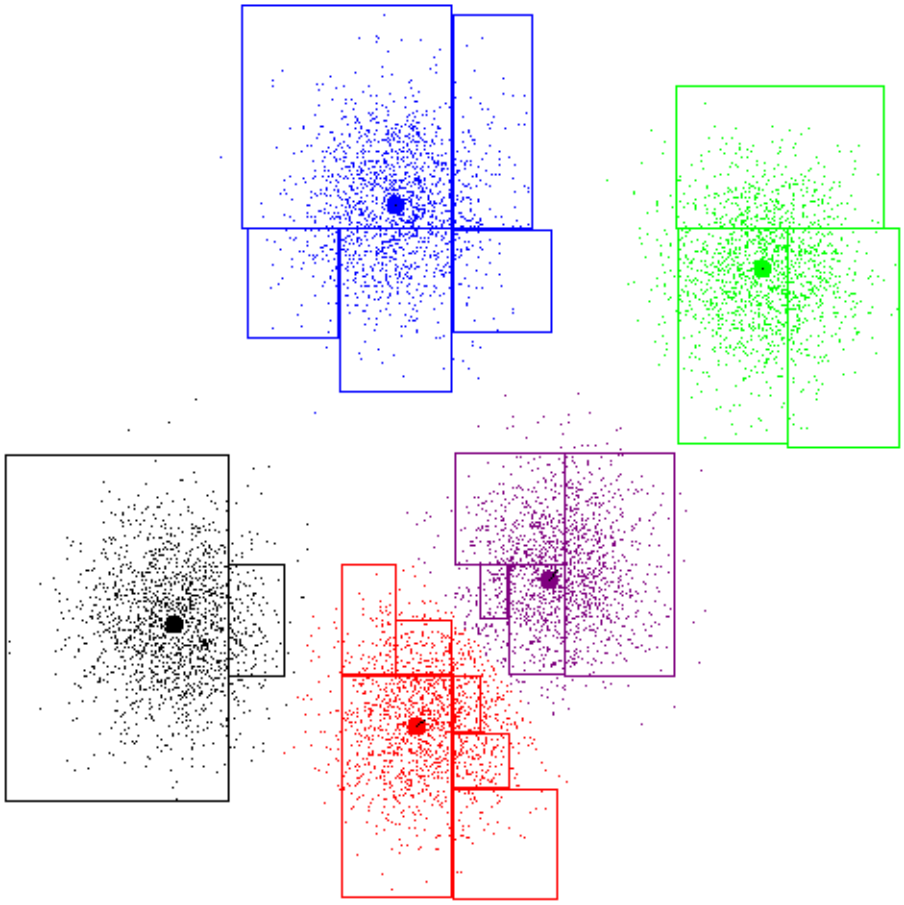
$$\text{dist}(x, y) = (x - y)^{\mathsf{T}}(x - y) = \sum_i (x_i - y_i)^2$$

Question 1: How to Find an Effective Algorithm to cluster these points into k subsets

- One solution is K-means Algorithm – an iterative algorithm to cluster points into k sets
  - Initialization: pick *K* random points as cluster centers (means)
  - Each iteration:
    - Assign data instances to closest mean
    - Update each mean to the average of its assigned points
  - Repeat until no points' assignments change



K-Means Example



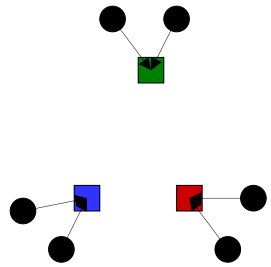
Question 2: How do we know K-means Algorithm is good?

⇒ K-means analysis from optimization perspective:

- Total distance as a function of the means and assignments:

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

points — assignments — means



- Clustering seeks to find centers $c_1, \ldots, c_k$ to minimize ϕ

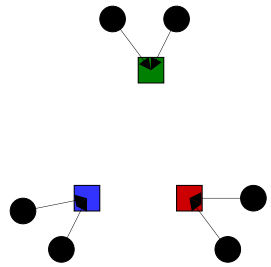
Question 2: How do we know K-means Algorithm is good?

⇒ K-means analysis from optimization perspective:

- Total distance as a function of the means and assignments:

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

points, assignments, means



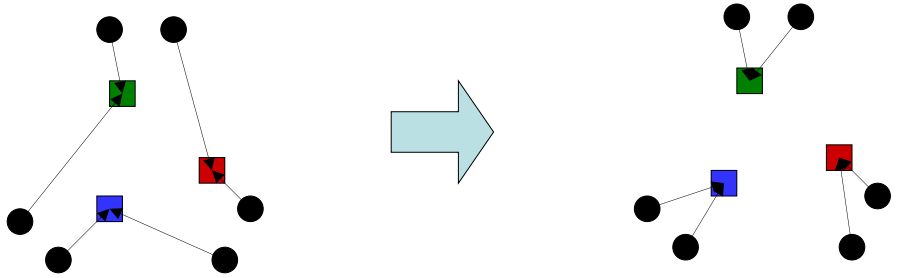
Claim: each iteration of K-means reduces ϕ

- Two phases at each iteration:
  - Update assignments: fix means *c*, change assignments *a*
  - Update means: fix assignments *a*, update means *c*
  - **Will prove each phase reduces**

Phase I: Update Assignments

- For each point, re-assign to closest mean:

$$a_i = \underset{k}{\text{argmin}}\, \text{dist}(x_i, c_k)$$



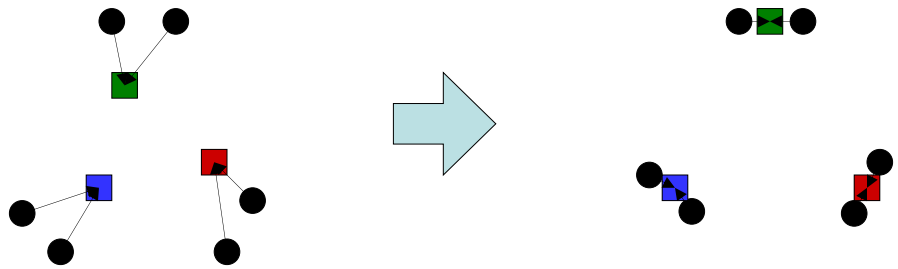
- Can only decrease total distance ϕ!

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

Phase II: Update Means

- Move each mean to the average of its assigned points:

$$c_k = \frac{1}{|\{i : a_i = k\}|} \sum_{i:a_i=k} x_i$$



- Also can only decrease total distance… (Why?)

Claim. The point y with minimum total squared Euclidean distance to a set of points $\{x_i\}$ is their mean

- We prove the 1-dimensional case: for any $x_1, x_2, \cdots, x_n$ which y minimizes $(y - x_1)^2 + \cdots + (y - x_n)^2 = d(y, \{x_i\})$?
- Take derivative of d on y, and set it to 0

$$2(y - x_1) + \cdots + 2(y - x_n) = 0 \Rightarrow y = \frac{x_1 + \cdots + x_n}{n}$$

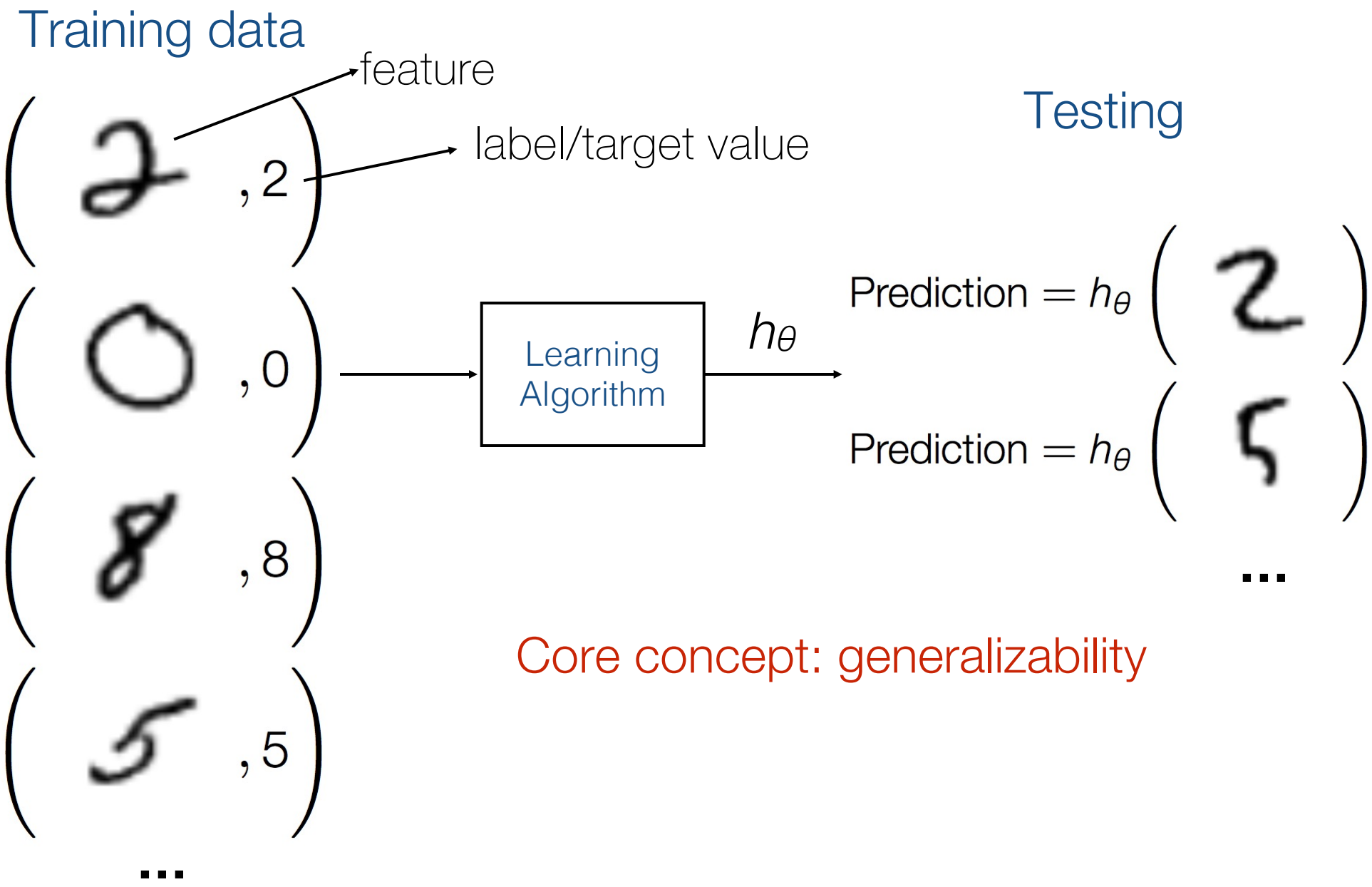
Remarks

- Such optimization-driven algorithm design/analysis is a dominating paradigm in today's ML (will see many examples later)
- Such (a) design of an algorithm, and (b) analysis of its performance – via theoretical proofs or empirical evaluations – is a very common procedure in ML research
  - ✓ Many materials of this course will be organized in this way to illustrate the "research pipeline" for you
- Difference between high or low-quality research lies at
  1. The importance or timeliness of the problem they address
  2. The quality of the algorithms they developed
  3. The thoroughness of the analysis/evaluation about the algorithm

Outline

- Introduction to Clustering
- Supervised Learning and Linear Models
- Neural Networks

Supervised Learning



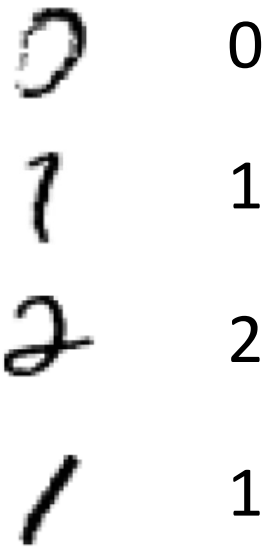
Core concept: generalizability

Classification Problems

- Classification Problem
  - *Input*: A sampled feature vectors *f*
    - *f* can be continuous or discrete
  - *Output*: A label *y*, corresponding to input feature
    - *y* is discrete (otherwise it is called a “regression” problem)
- Goal: given a set of labeled training data $(f_1,y_1),\ldots,(f_n,y_n)$, build a model that predicts label *on unseen data* with high accuracy

Example: Digit Recognition

- Input: an image (pixel grids)
- Output: a digit prediction 0-9
- Setup:
  - Get a large collection of example images, each labeled with a digit
    - Someone has to manually label all this data!
    - These data "supervise" your learning



Example: Digit Recognition

- Input: an image (pixel grids)
- Output: a digit prediction 0-9
- Setup:
  - Get a large collection of example images, each labeled with a digit
    - Someone has to manually label all this data!
    - These data "supervise" your learning
  - Want to learn to predict labels of ***new***, ***future digit images***
- Features — description of the input
  - Pixels' RGB: position (13,17) = (63, 11, 210)
  - Shape Patterns: NumComponents, NumLoops
  - …


0


1

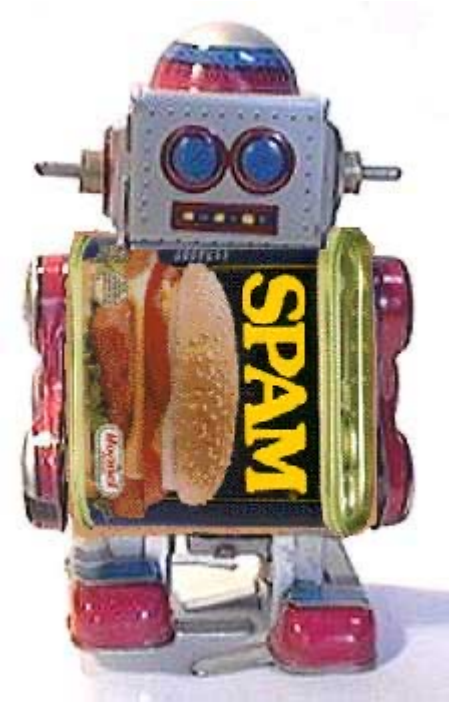

2


1


??

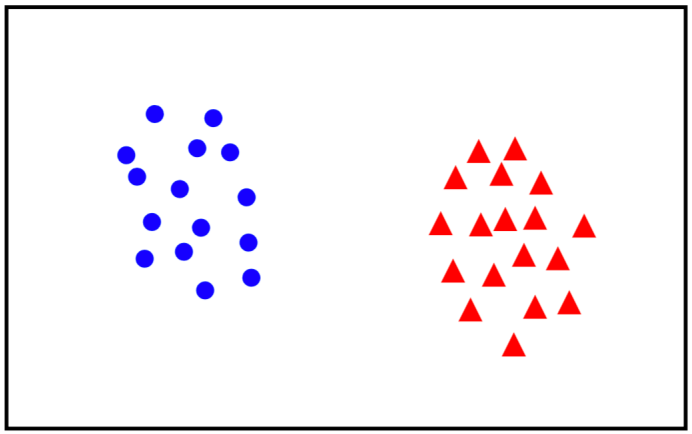
Example: Spam Filter

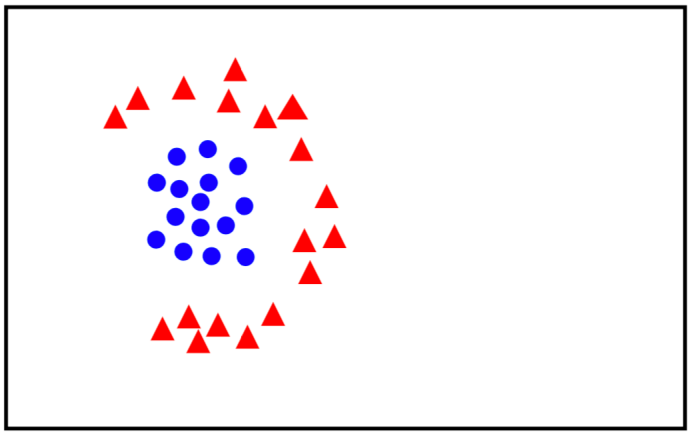
- Input: an email (array of strings)
- Output: spam or not
- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham (not spam)”
  - Want to learn to predict labels of ***new, future emails***
- Features:
  - Words: “FREE!”,"Billion dollars!"
  - Text Patterns: $dd, link
  - Non-text metadata: SenderInContacts
  - …



A Fundamental Task: Binary Classification

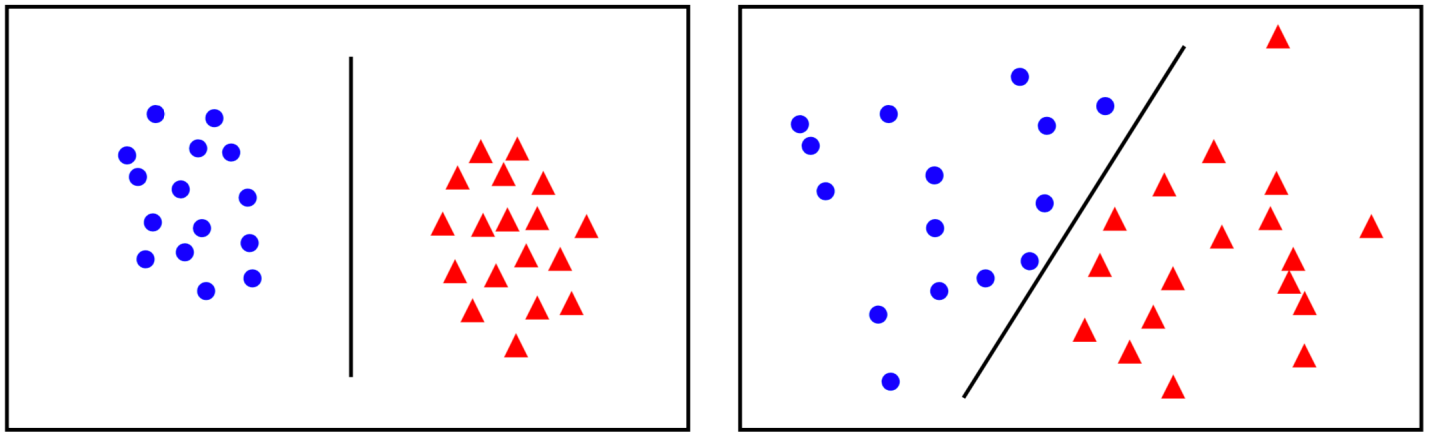
- Only two possible labels, w.l.o.g., y ∈ { -1, 1}
  - e.g., spam or not spam



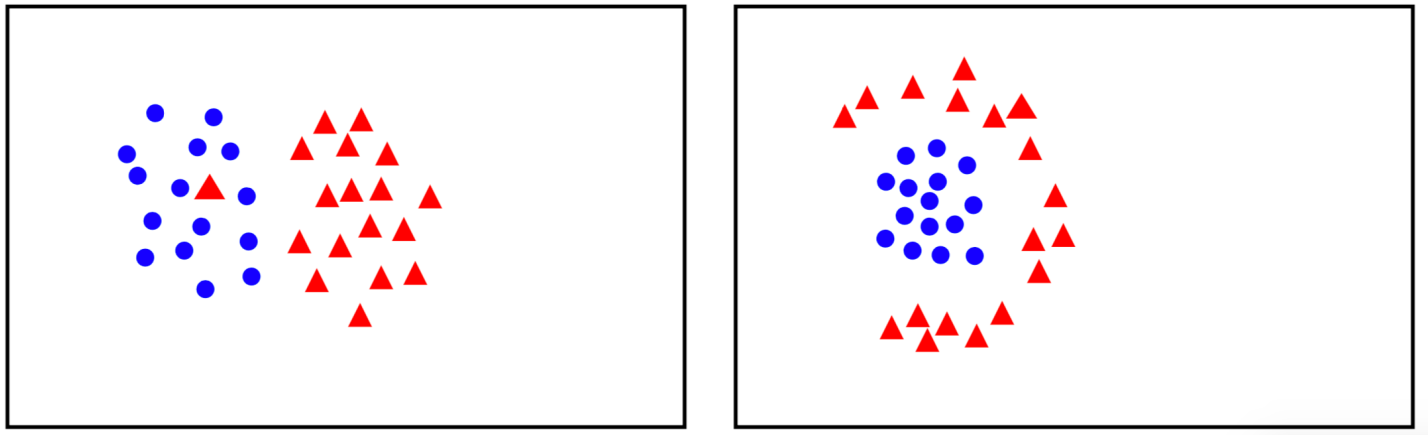


Linear Separability

Linearly separable



Not Linearly separable



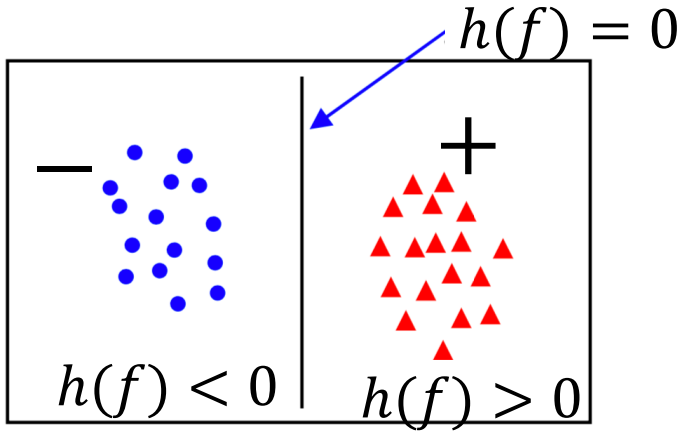
Linear Classifiers

- Given training data $(f_1, y_1), \ldots, (f_n, y_n)$, want to find a linear function that (ideally) separates positively-labeled points from the negative

The form of a linear classifier:

$$h(f) = \theta^T \cdot f + b,$$

(θ, b) is model parameter



- Notably, $h(f_i) \times y_i > 0$ if data linearly separable by h

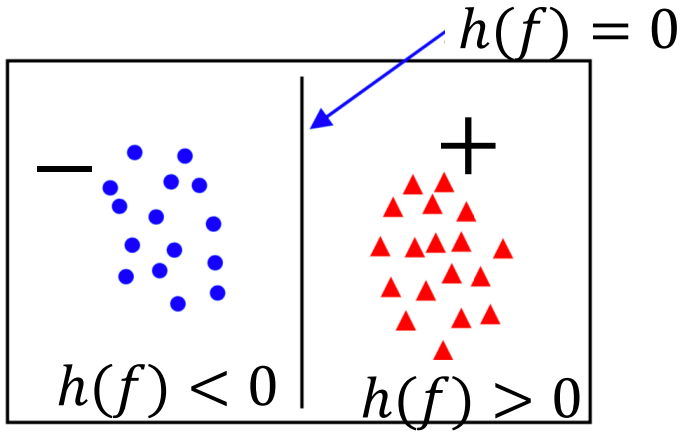
Linear Classifiers

- Given training data $(f_1, y_1), \ldots, (f_n, y_n)$, want to find a linear function that (ideally) separates positively-labeled points from the negative

The form of a linear classifier:

$$h(f) = \theta^T \cdot f + b,$$

(θ, b) is model parameter



- In 2-D, the classifier is a line
- $\boldsymbol{\theta}$ is direction of the line, $\boldsymbol{b}$ is the bias
- $\boldsymbol{\theta}$ is also called the weight vector

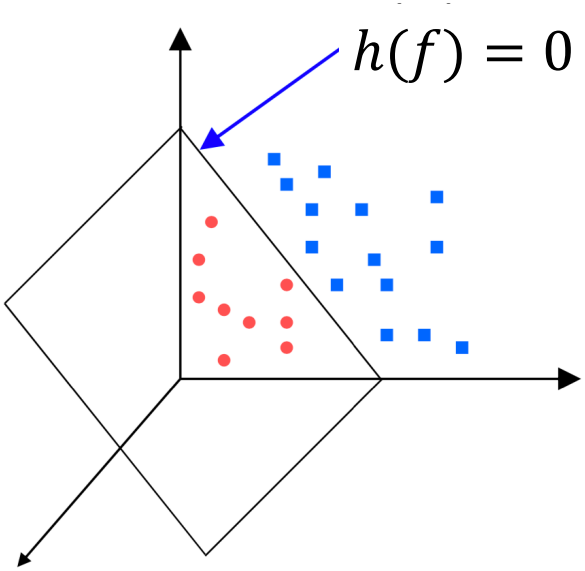
Linear Classifiers

- Given training data $(f_1, y_1), \ldots, (f_n, y_n)$, want to find a linear function that (ideally) separates positively-labeled points from the negative

The form of a linear classifier:

$$h(f) = \theta^T \cdot f + b,$$

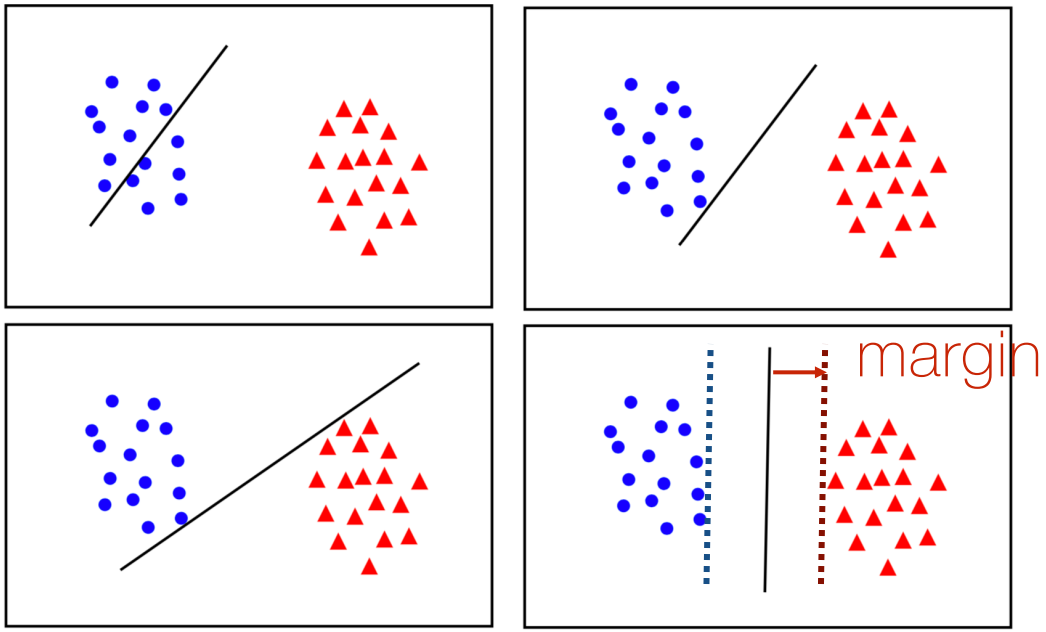
(θ, b) is model parameter



- In 3-D, the classifier is a hyperplane
- Training data is used to learn the best $(\boldsymbol{\theta}, b)$
- Learned classifier only depends on $(\boldsymbol{\theta}, b)$: y = 1 if and only if $h(f) > 0$

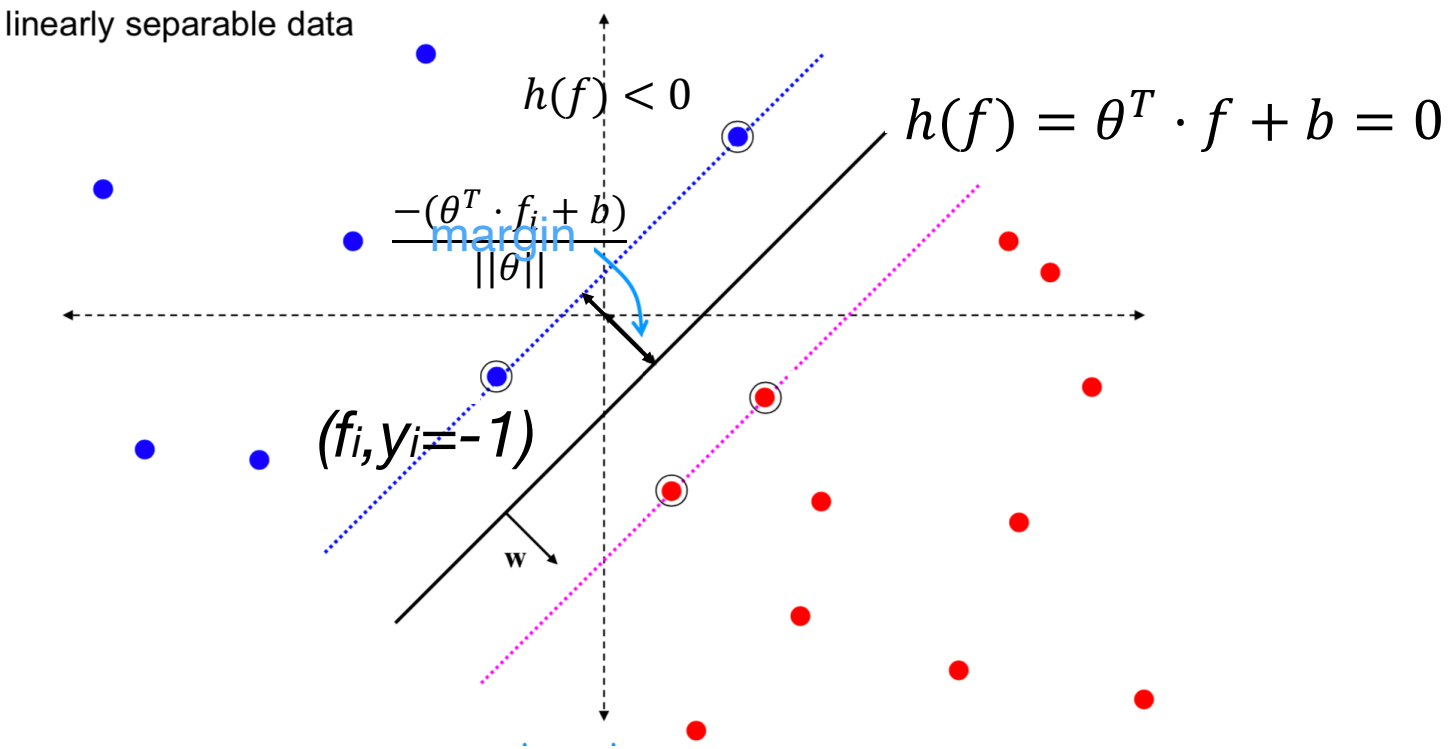
Question: Which Hyperplane is the Best?

- Separable cases:



- This research question leads to development of **support vector machine (SVM)**: the classifier with **maximum margin**
  - Intuition: most stable under perturbations of the inputs
- SVM (a) enjoys elegant theoretical analysis; (b) also work well in practice with small amount of training data

Approach: Formulation SVM as Optimization

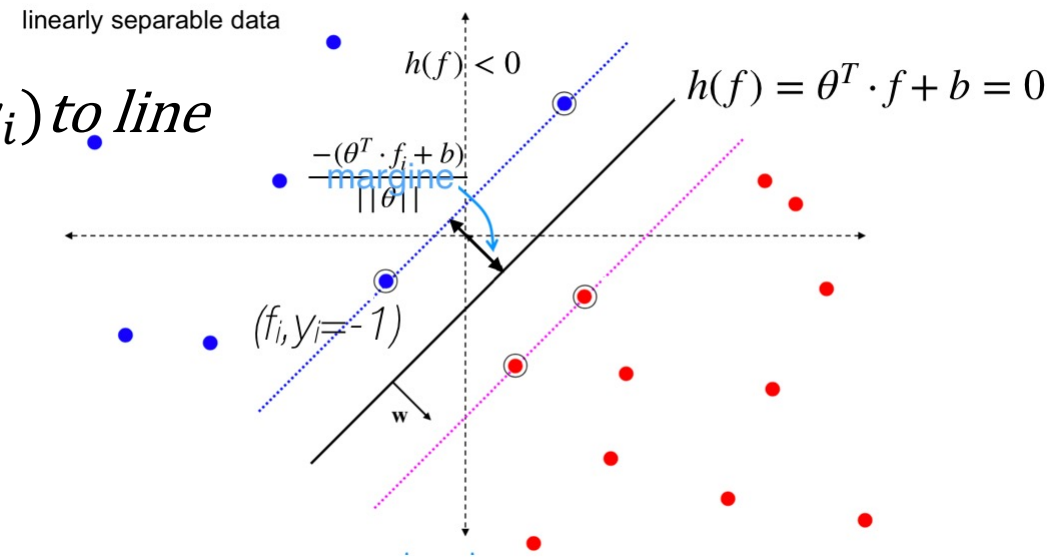


- Margin = distance of a support vector to the classifier
  - Recall the distance formula from linear algebra
  - Note: $y_i(\theta^T \cdot f_i + b) = y_i \cdot h(f_i) > 0$ in separable case

Approach: Formulation SVM as Optimization

Distance of any data point (f_i, y_i) *to line*

is $\frac{y_i \cdot h(f_i)}{||\theta||} = \frac{y_i \cdot (\theta^T \cdot f_i + b)}{||\theta||}$

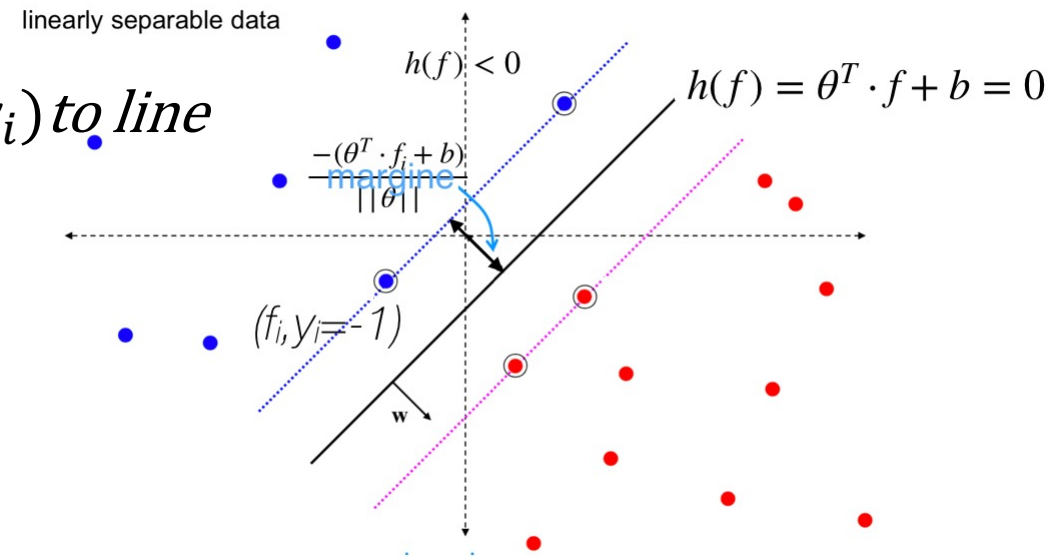


Margin=: $\min_i \frac{y_i \cdot (\theta^T \cdot f_i + b)}{||\theta||}$

Approach: Formulation SVM as Optimization

Distance of any data point (f_i, y_i) *to line*

is $\frac{y_i \cdot h(f_i)}{||\theta||} = \frac{y_i \cdot (\theta^T \cdot f_i + b)}{||\theta||}$



$$\max_{\theta,b} \min_i \frac{y_i \cdot (\theta^T \cdot f_i + b)}{||\theta||}$$

SVM computes the (θ, b) *that maximizes the margin*

Approach: Formulation SVM as Optimization

$$\max_{\theta,b} \min_{i} \frac{y_i \cdot (\theta^T \cdot f_i + b)}{||\theta||}$$

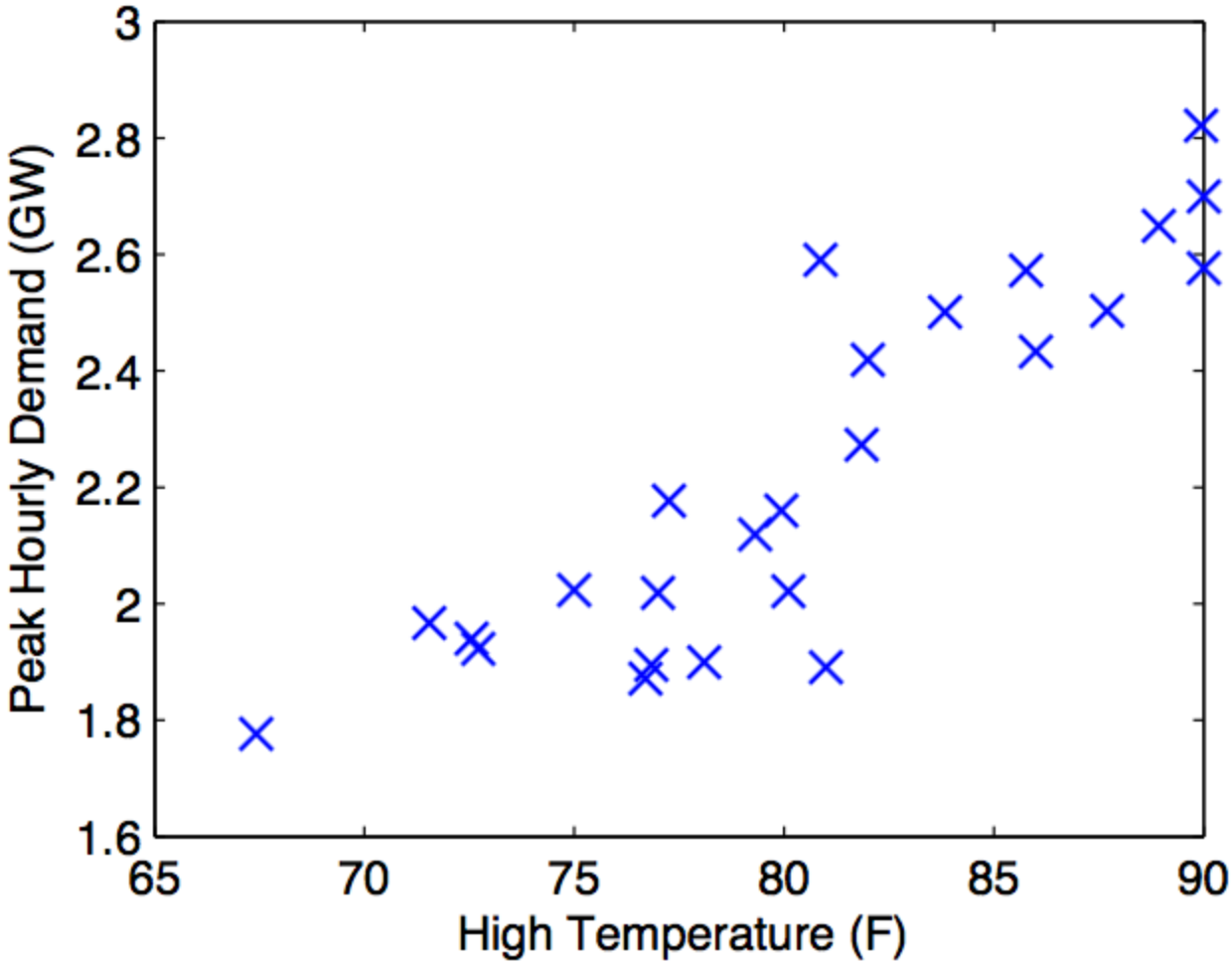
SVM computes the (θ, b) *that maximizes the margin*

- We have seen this idea before, e.g., k-means computes k centers to minimize the total distances
- Many ML problem can be viewed as an optimization problem — compute the model parameter that minimizes some cost function

Regression Problems

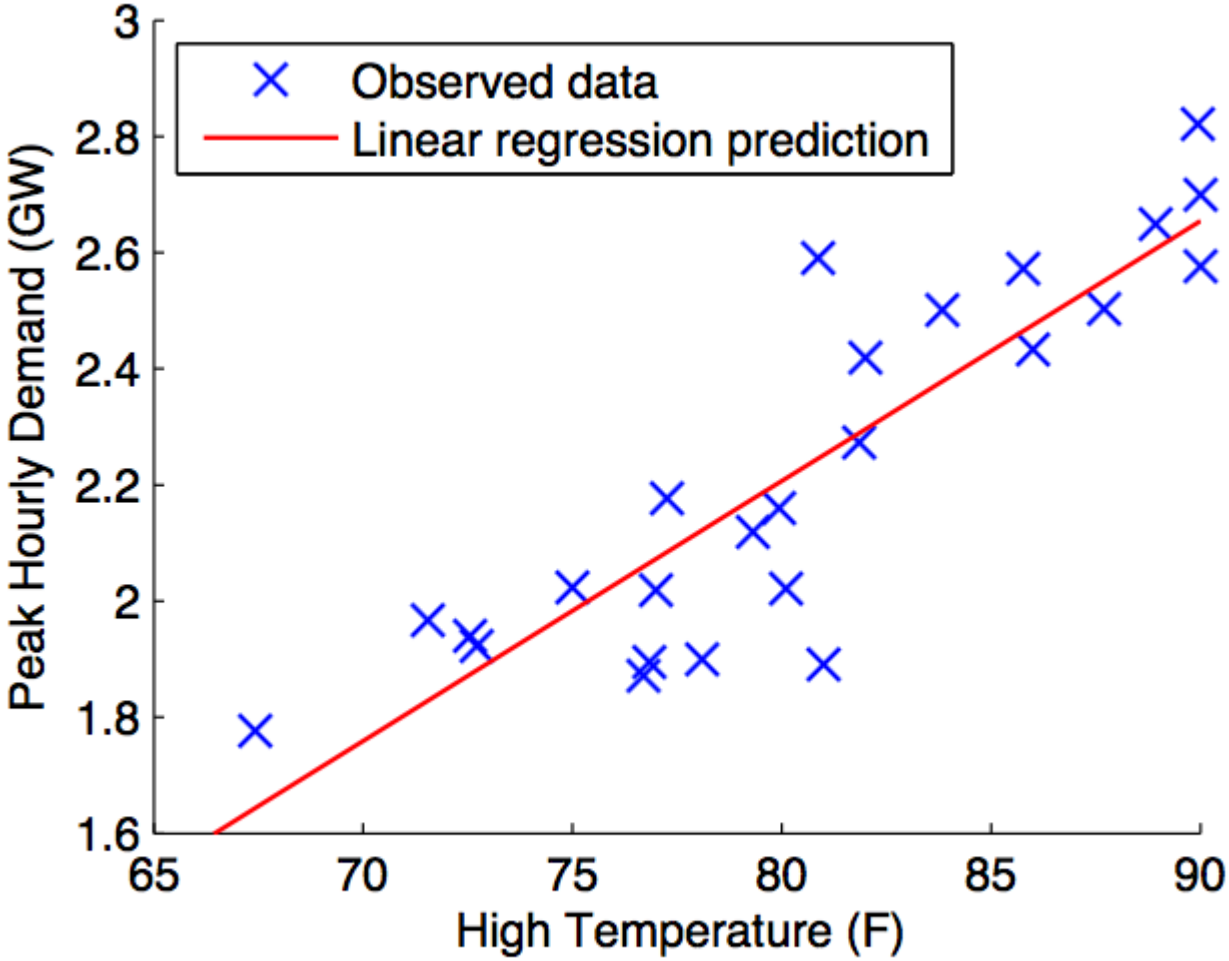
- Very similar idea to classification, except label *y* is continuous
- Example: what will the peak power consumption in Charlottesville be on a given day?
  - Possible features: high temperature of the day
  - ***Target values***: peak demand

Regression



$$\text{Peak demand} \approx \theta_1(\text{high temp}) + \theta_2$$

Regression



$$\text{Peak demand} \approx \theta_1(\text{high temp}) + \theta_2$$

Question: How to Do Regression?

- Assume a model h_θ to capture relation between features and target values
  - E.g., linear relation between high temperate and peak demand
  - Linear regression = h_θ is a linear function
- Learn parameters θ
  - In regression, we learn θ via optimization
  - Define an objective called *loss function* (designed to measure how "good" a predictor is against the data)
  - For regression, we usually use a squared loss:

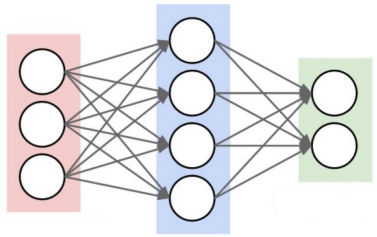
$$L(h_\theta(f_i), y_i) = (h_\theta(f_i) - y_i)^2$$

linear regression

Half Time Break

Algorithms for Modern Artificial Intelligence (NeoScholar Spring Program)

Lec 2B: Machine Learning and Neural Networks



Haifeng Xu

Professor of Science and Data Science

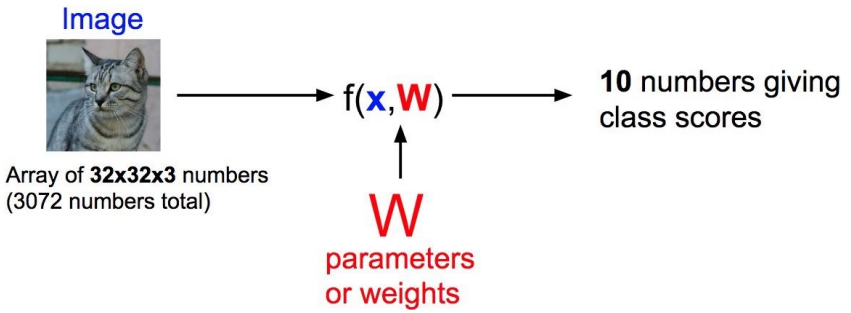
University of Chicago

Outline

- Introduction to Clustering
- Supervised Learning and Linear Models
- Neural Networks

Recap: linear classifiers from first half

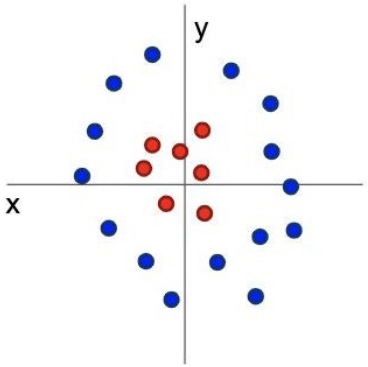
$$f(x, W) = Wx + b$$



Note, generally f can linearly map x to a vector

Problem: Linear Classifiers are not very powerful

Geometric Viewpoint



Linear classifiers
can only draw linear
decision boundaries

Visual Viewpoint

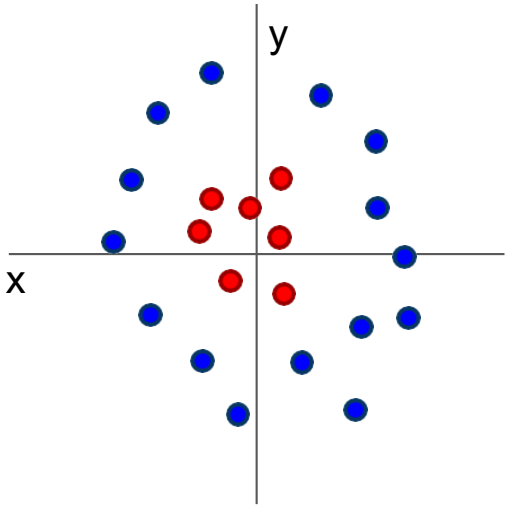
Linear classifiers learn
one template per class
(stored in weights W)

Pixel Features

$$f(x) = Wx$$

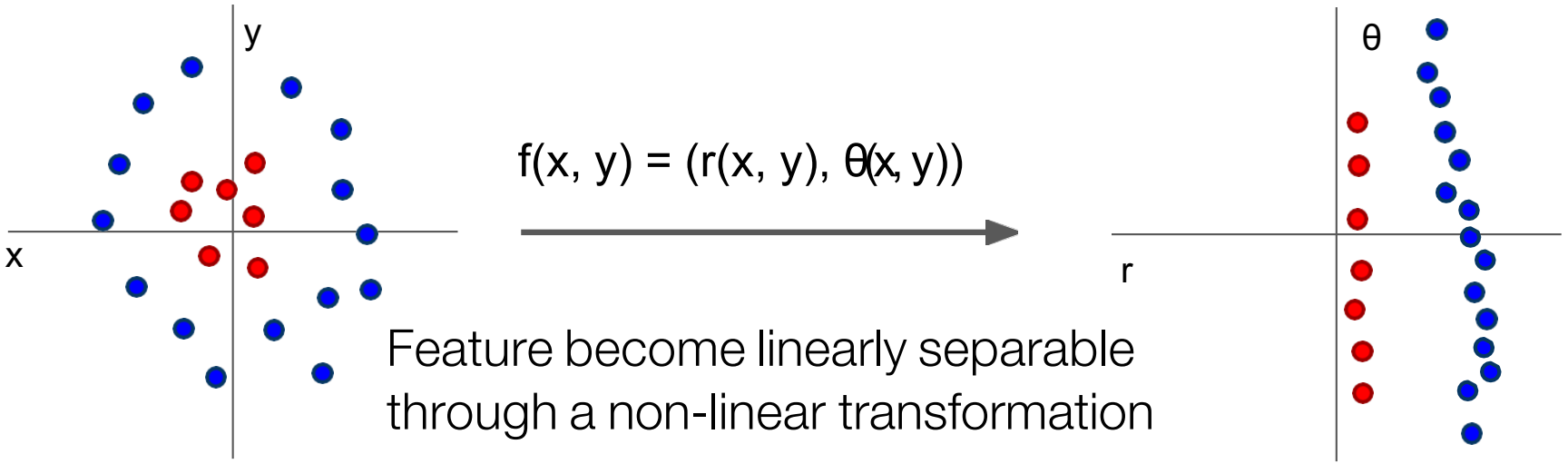
Class scores

How to do Better? Introduce Non-linearity



Cannot separate red and blue points with linear classifier

How to do Better? Introduce Non-linearity

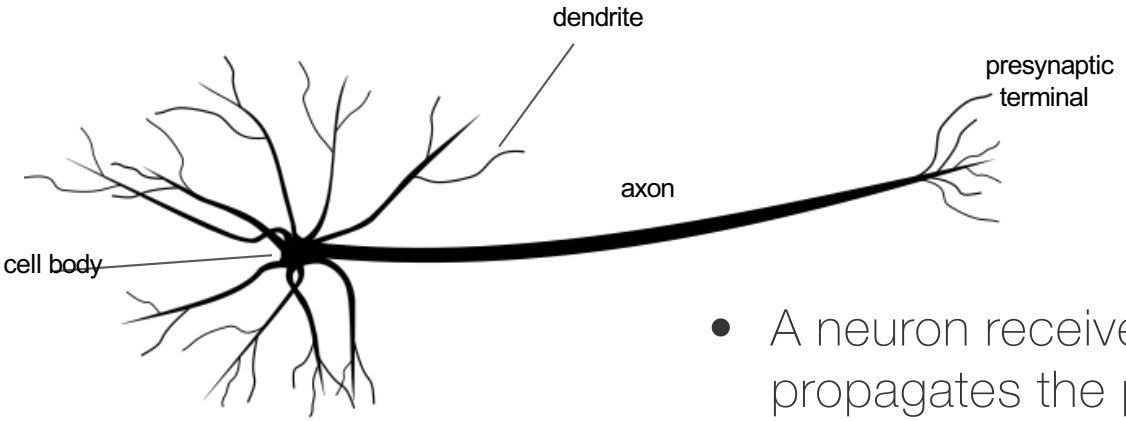


Feature become linearly separable through a non-linear transformation

Cannot separate red and blue points with linear classifier

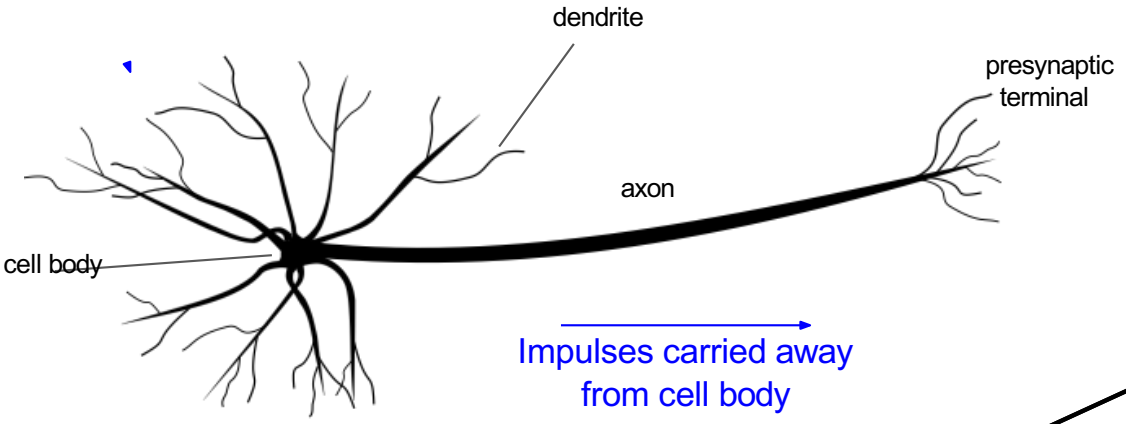
After applying feature transform, points can be separated by linear classifier

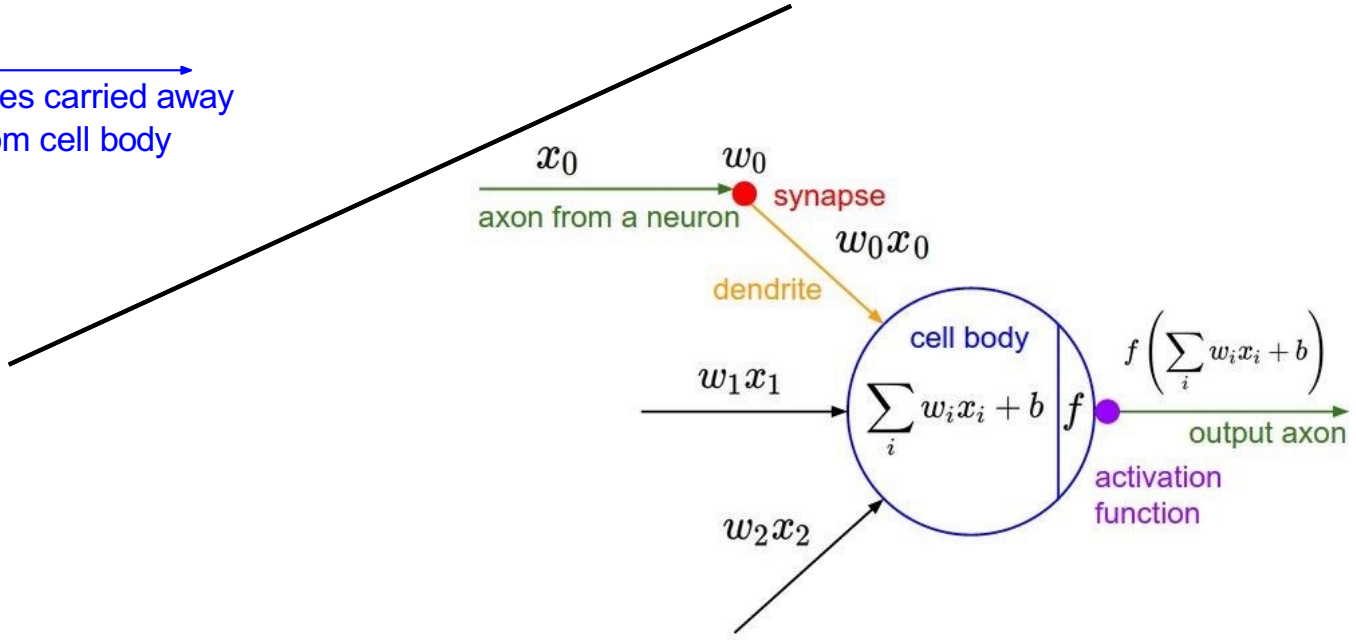
Neural networks are inspired by “neurons” in our brain



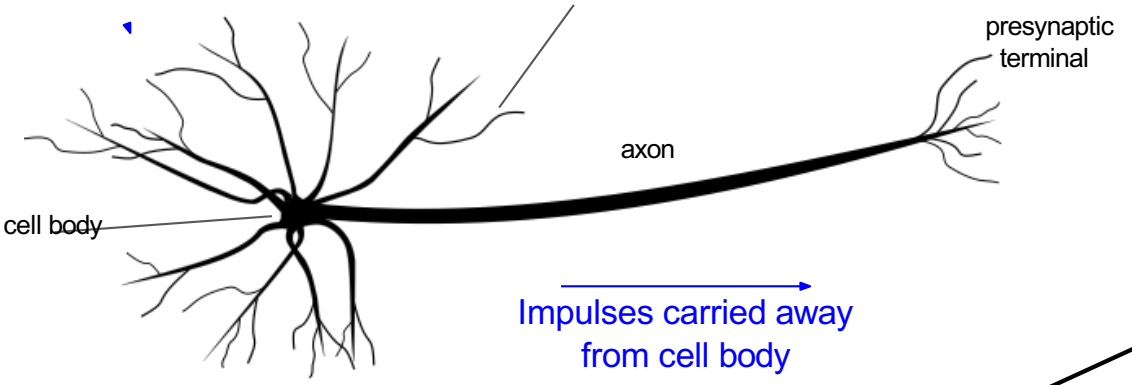
- A neuron receives a signal, processes it, and propagates the processed information
- The brain is comprised of around 100 billion neurons, each connected to ~10k other neurons: 10^{15} connections
- NNs are a simplistic imitation of a brain comprised of dense net of simple structures

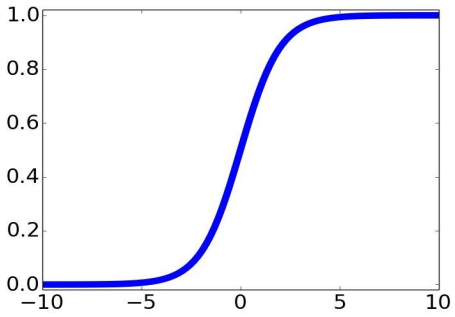
(Simplified) mathematical model of a neuron





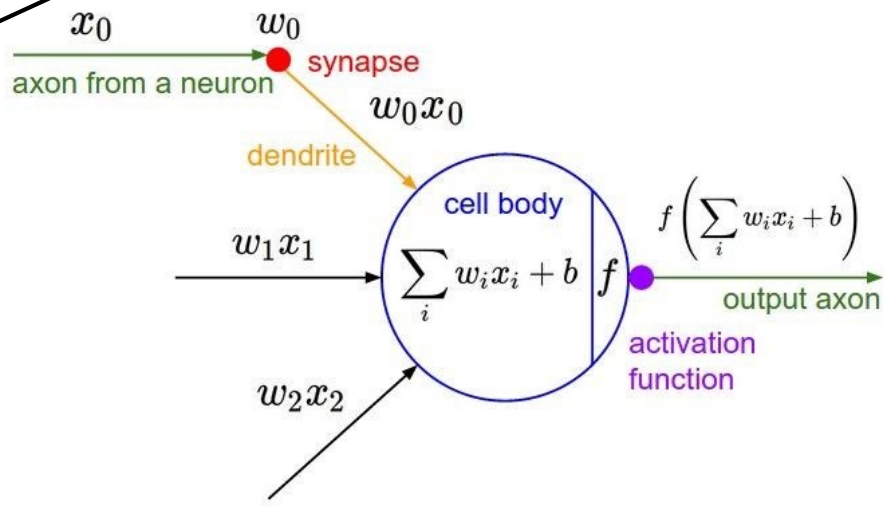
(Simplified) mathematical model of a neuron



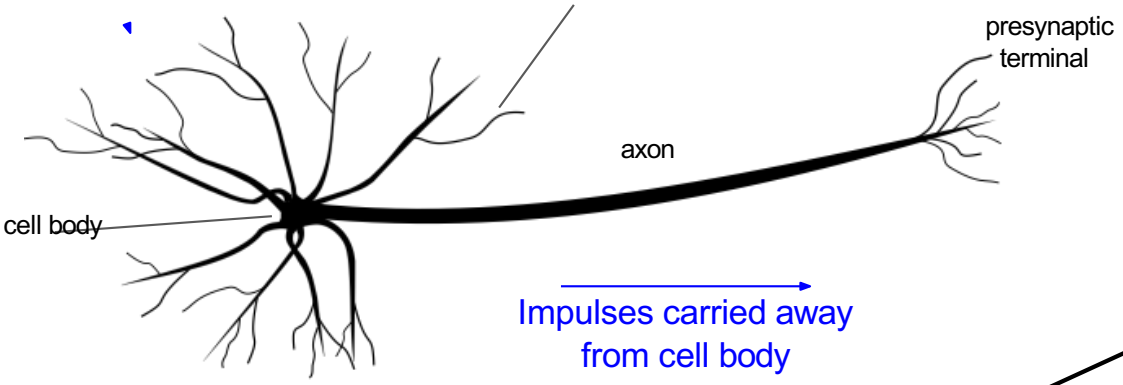


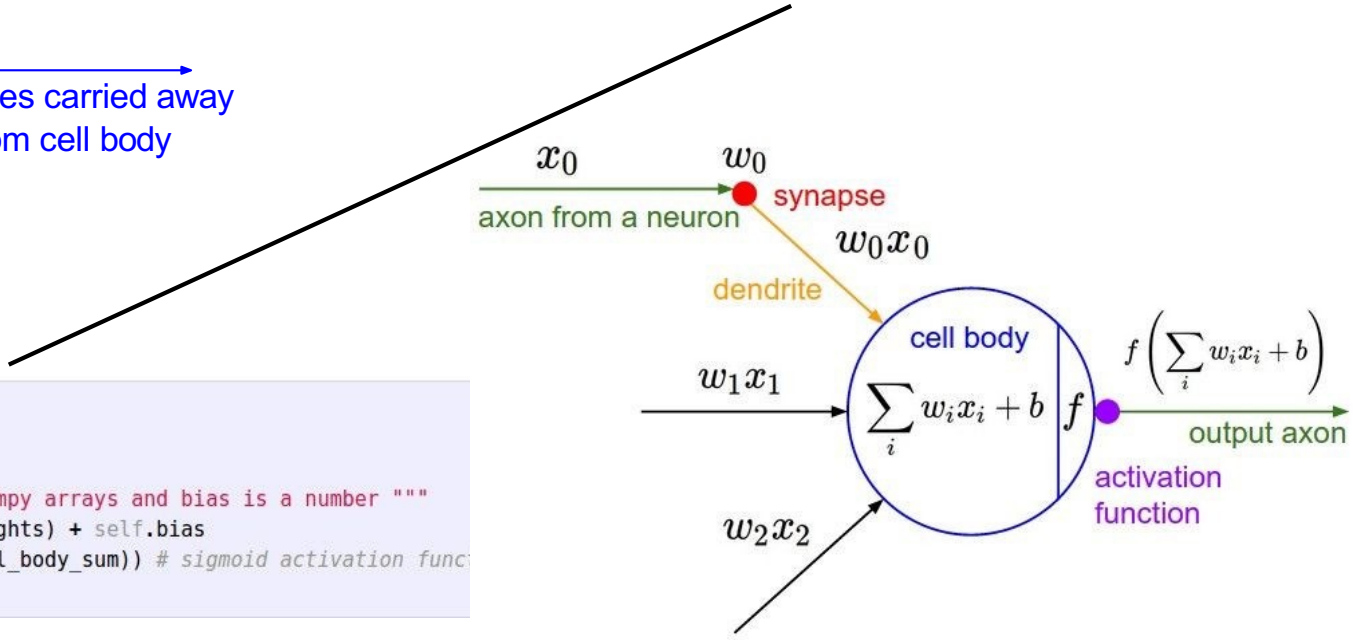
sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



(Simplified) mathematical model of a neuron



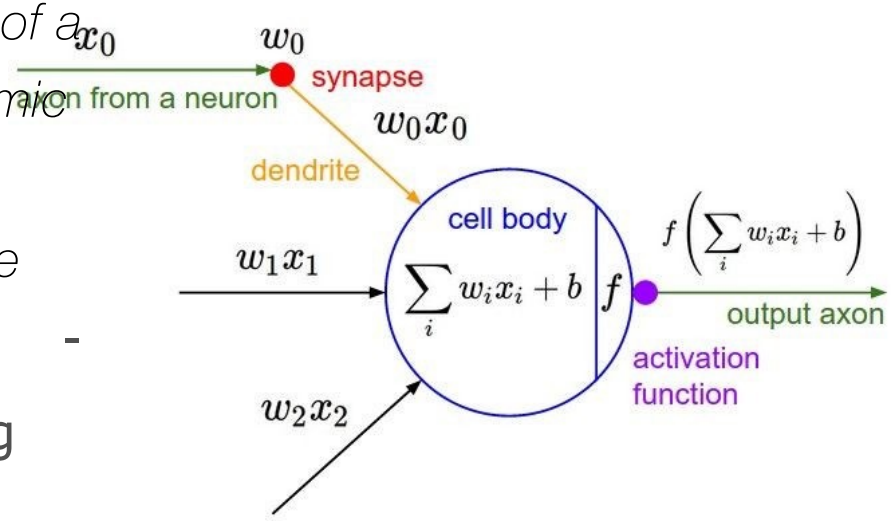



```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation func
    return firing_rate
```

(Simplified) mathematical model of a neuron

"A single neuron in the brain is an incredibly complex machine that even today we don't understand. A single "neuron" in a neural network is an incredibly simple mathematical function that captures a minuscule fraction of the complexity of a biological neuron. So to say neural networks mimic the brain, that is true at the level of loose inspiration, but really artificial neural networks are nothing like what the biological brain does."

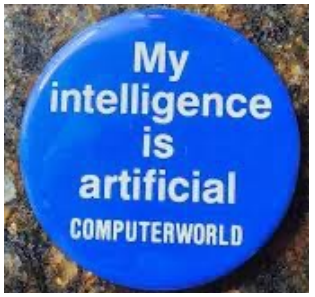
- Andrew Ng

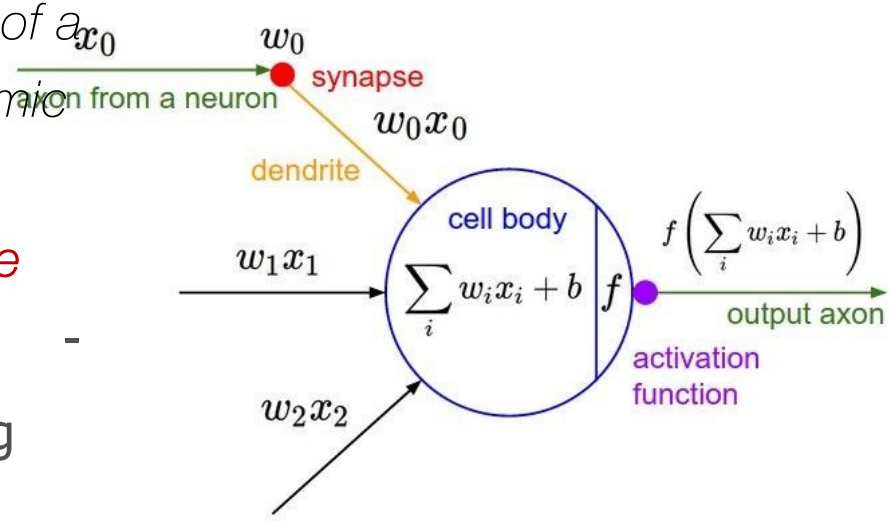


(Simplified) mathematical model of a neuron

"A single neuron in the brain is an incredibly complex machine that even today we don't understand. A single "neuron" in a neural network is an incredibly simple mathematical function that captures a minuscule fraction of the complexity of a biological neuron. So to say neural networks mimic the brain, that is true at the level of loose inspiration, but really artificial neural networks are nothing like what the biological brain does."

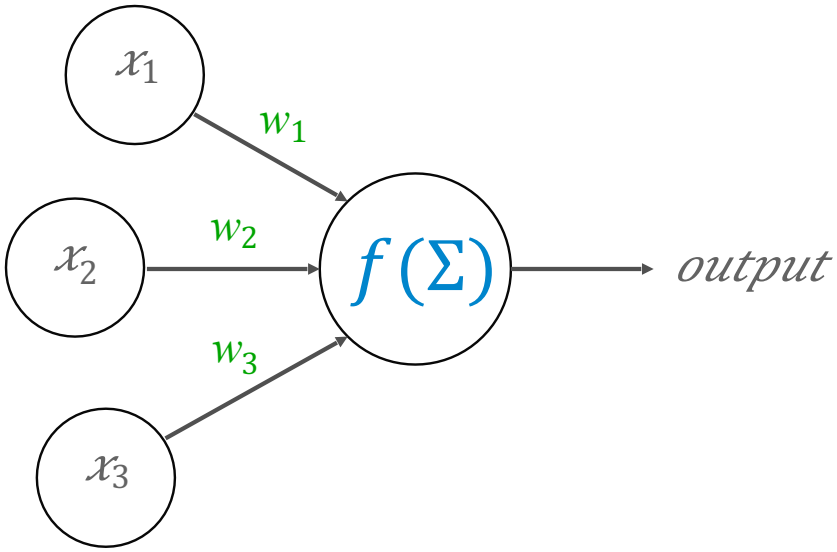
- Andrew Ng





A Simple Example Neuron

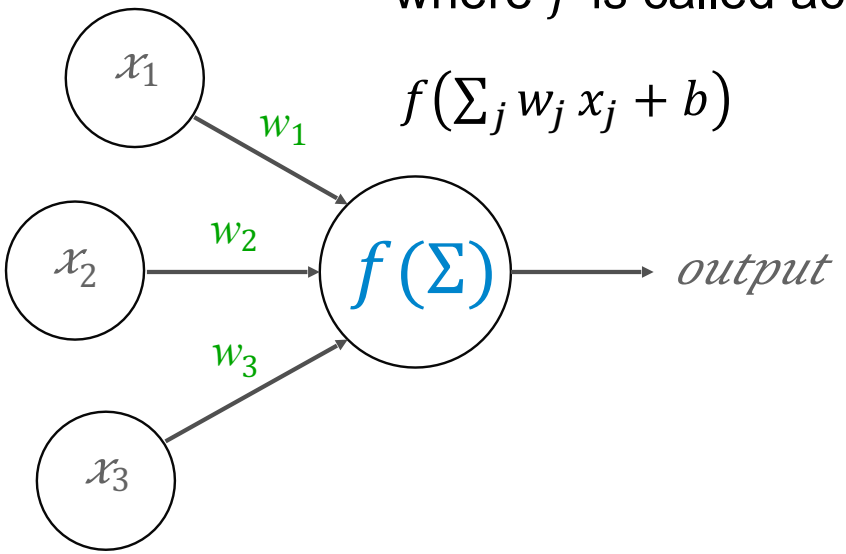
- This is precisely binary linear classifier
- Such a neuron is also called a “perceptron”



When $f(\Sigma) = \mathbb{I}(\Sigma > 0)$ is the indicator fnc

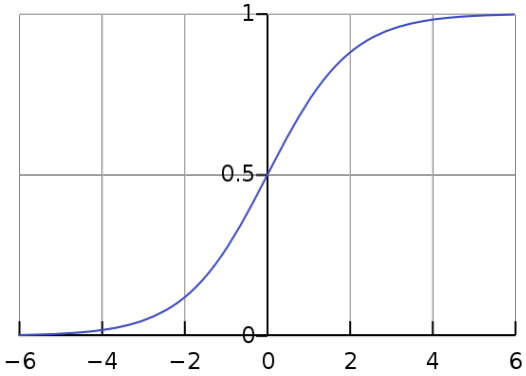
$$output = \begin{cases} 0, & \sum_{j=0}^{n} w_j x_j \leq threshold \\ 1, & \sum_{j=0}^{n} w_j x_j > threshold \end{cases}$$

Artificial Neuron in General Form



where f is called activation function

$$f\left(\sum_j w_j\, x_j + b\right)$$



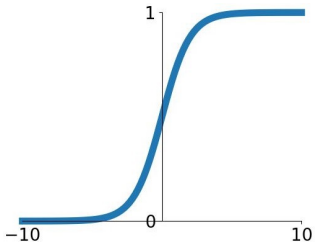
A typical shape of f

Introduce non-linearity which is precisely what we wanted

Activation functions

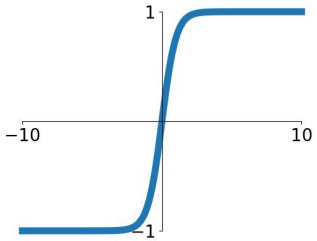
Sigmoid

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



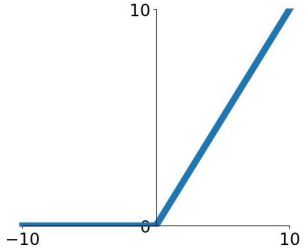
tanh

$$\tanh(x)$$



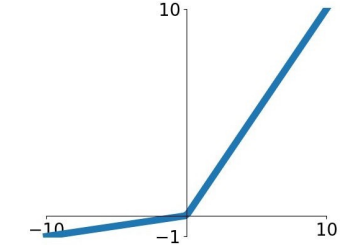
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

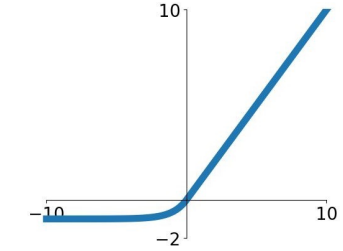


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

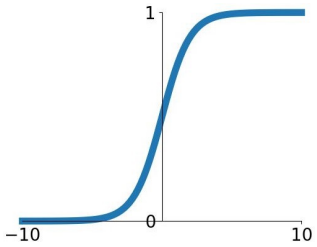


Activation functions

ReLU is a good default choice for most problems

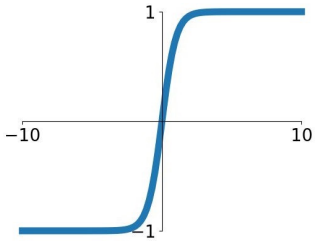
Sigmoid

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



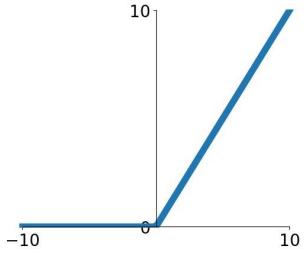
tanh

$$\tanh(x)$$



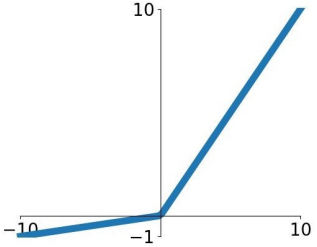
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

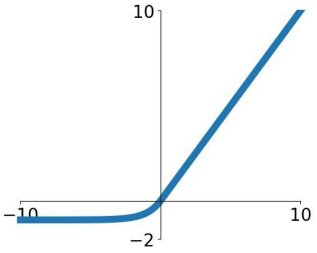


Maxout

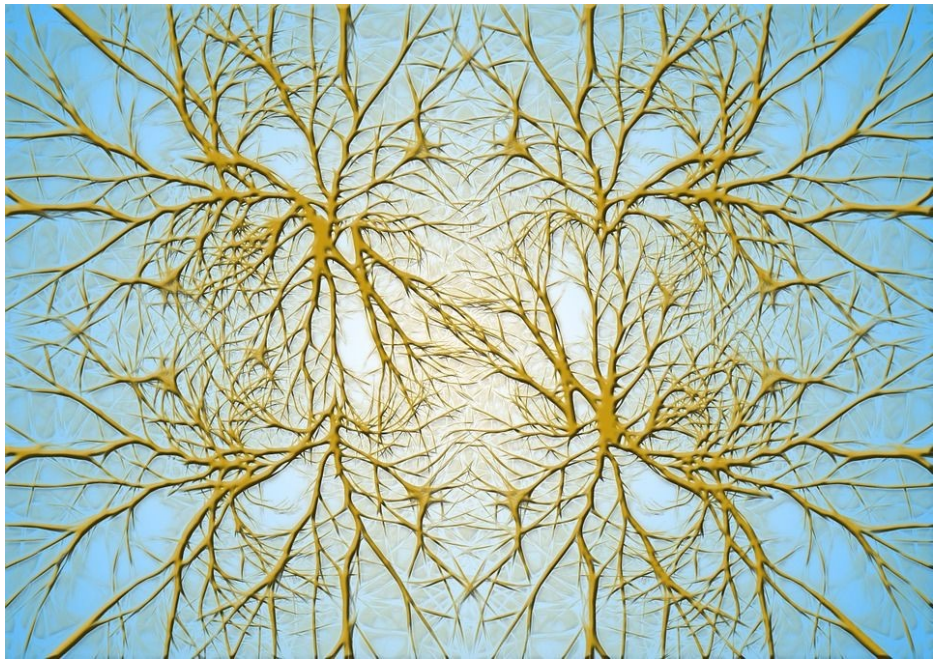
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

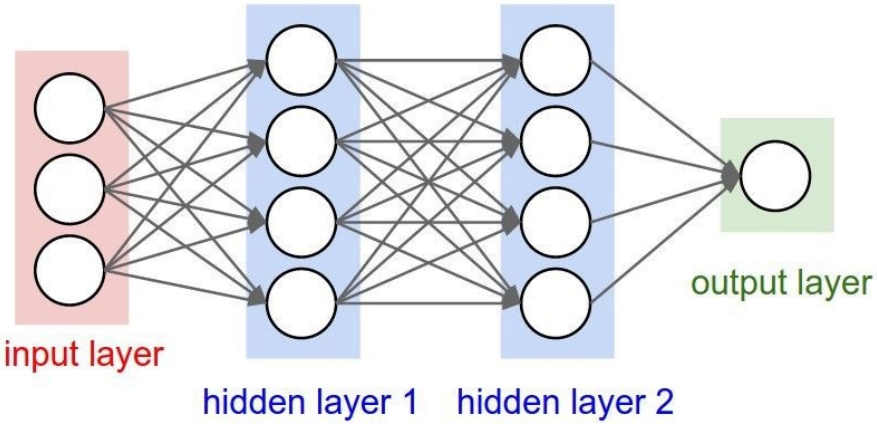
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



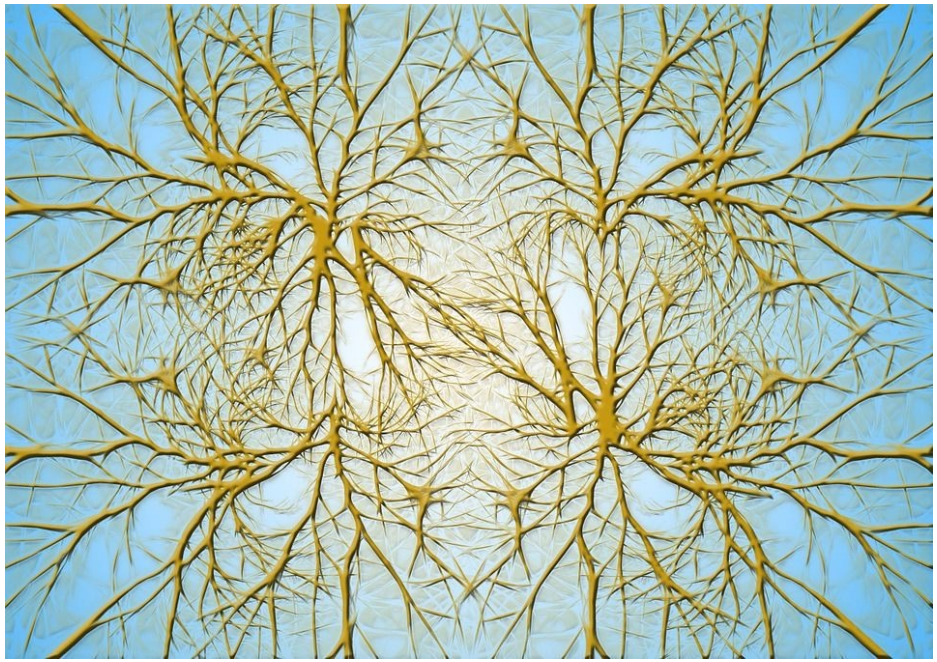
Biological Neurons:
Complex connectivity patterns



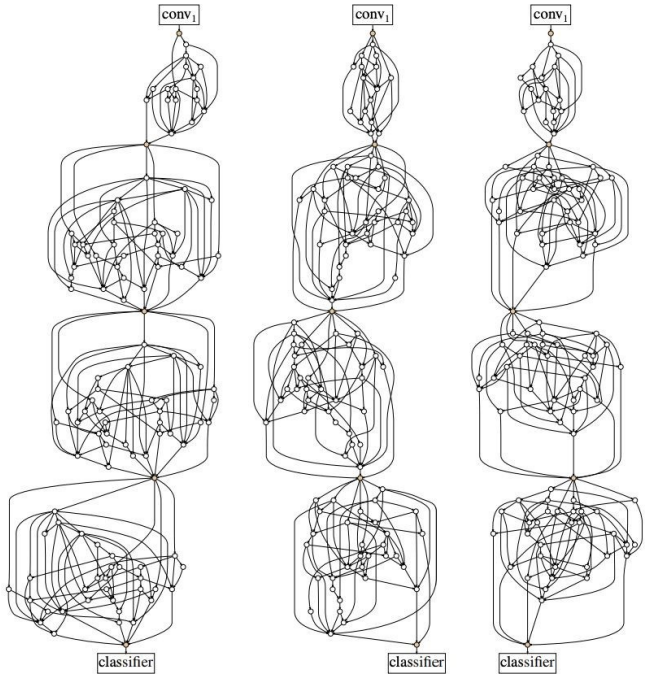
(Artificial) Neural Networks (NN) consists of neurons organized into regular layers for computational efficiency



Biological Neurons:
Complex connectivity patterns

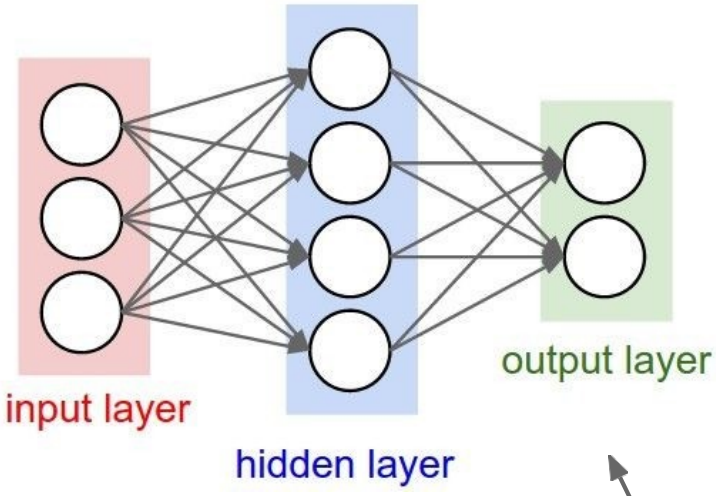


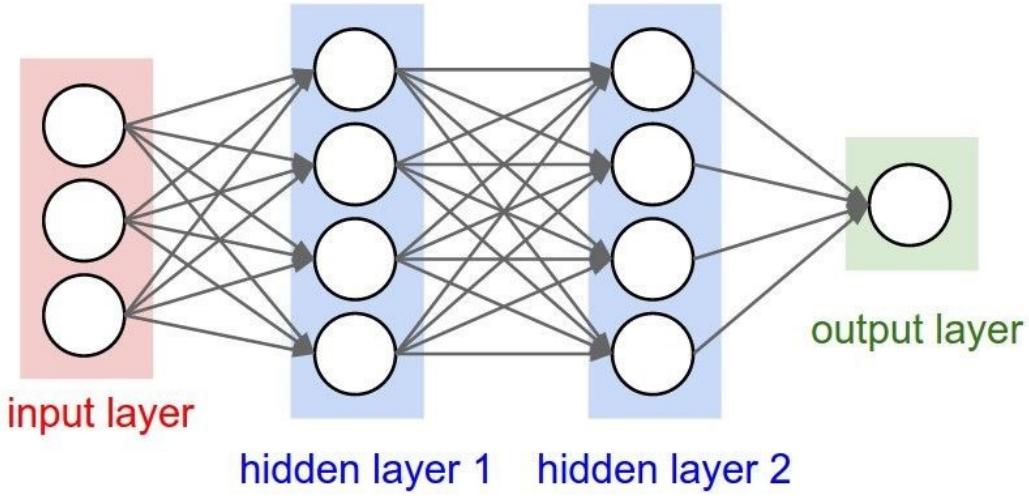
But neural networks with random connections can work too!



Xie et al, “Exploring Randomly Wired Neural Networks for Image Recognition”, arXiv 2019

Neural networks: Architectures



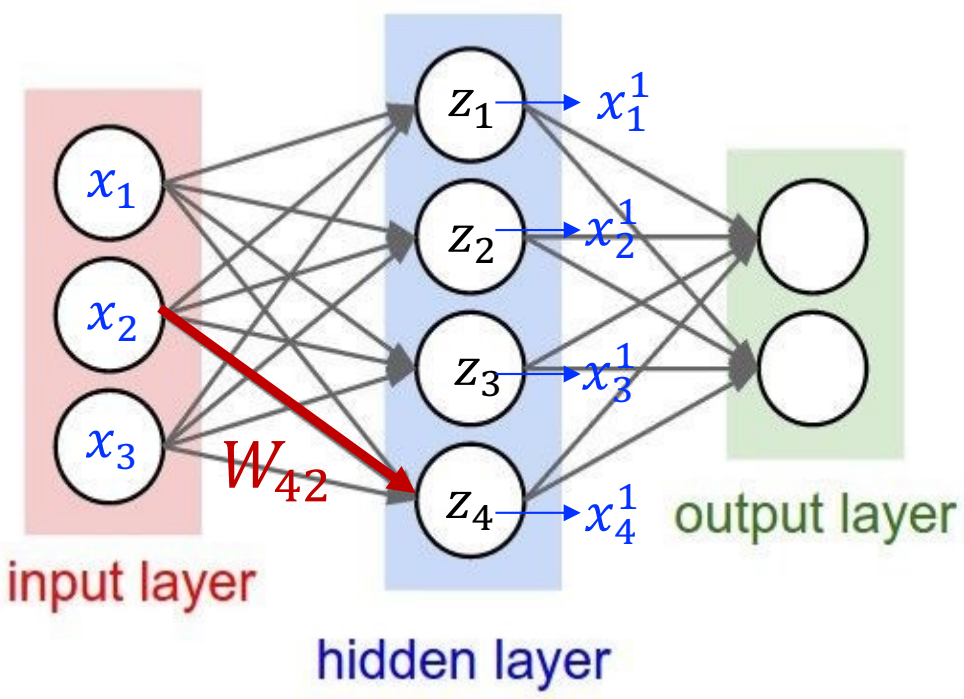


"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

How Does the Math Work?



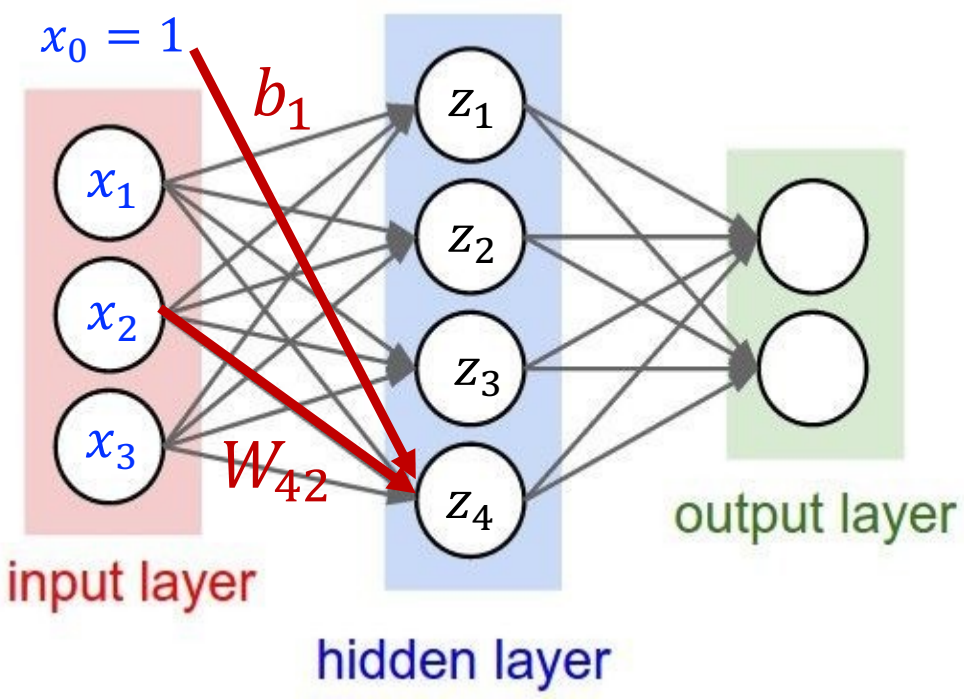
- ✓ Input vector $x \in R^n$
- ✓ Each edge between two layers has a weight parameter $W_{ij} \in R^{m \times n}$
- ✓ Raw values of next layer is $z = Wx \in R^m$
- ✓ Apply activation function to each entry of z

$$x^1 = f(z) = (f(z_1), f(z_2), \cdots, f(z_m))$$

Repeat for $x^1, x^2, \ldots$

W_{42} = contribution of x_2 to neuron 4

How Does the Math Work?



Remark 1

- ✓ In practice, we also add bias term b to get $z = Wx + b \in R^m$
- ✓ But mathematically equivalent

Remark 2

- ✓ This is called “fully connected NN” – the simplest NN architecture

Neurons turn out to have real meanings!

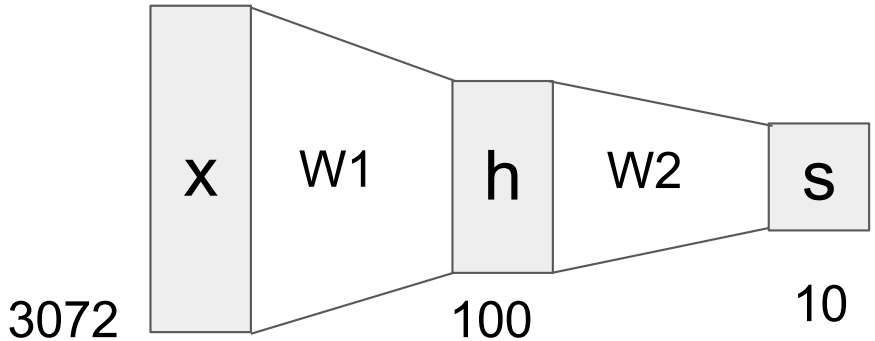
(**Before**) Linear score function: $f = Wx$

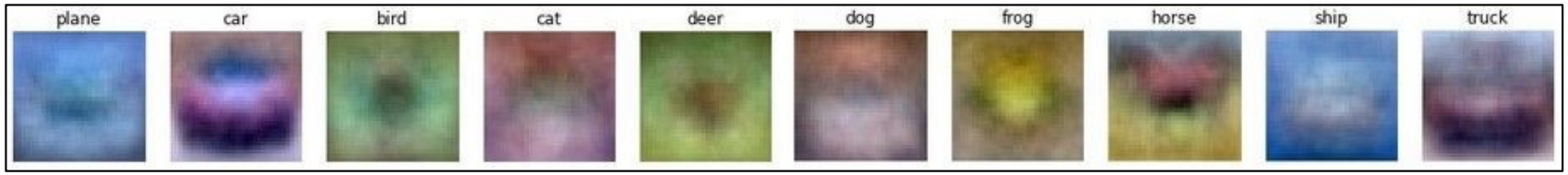
Pixel features

Neurons turn out to have real meanings!

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

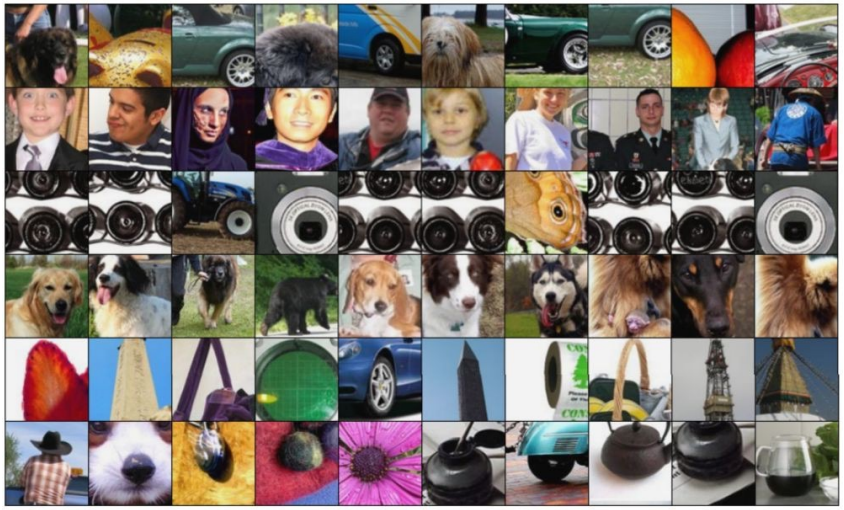




Learn 100 templates instead of 10.

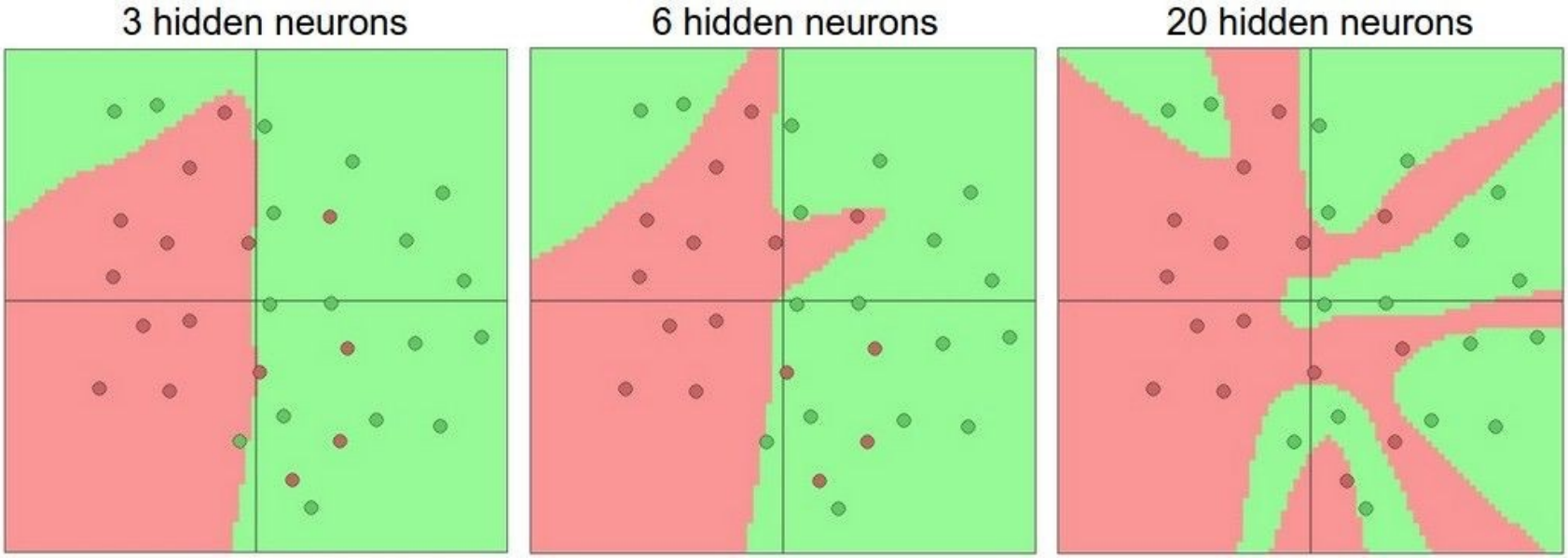
Share templates between classes

Examples of templates from real neural networks



Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

Setting the number of layers and their sizes



- Basically, NN is a more complex function
- "deep" NN = large layers

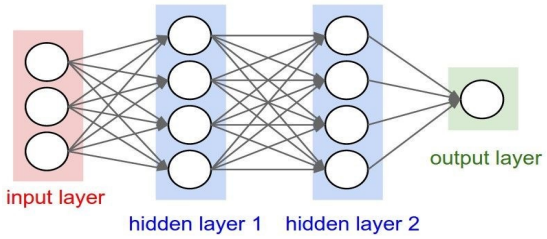
more neurons = more expressiveness of the classifier

Okay… so that's it for neural network research?

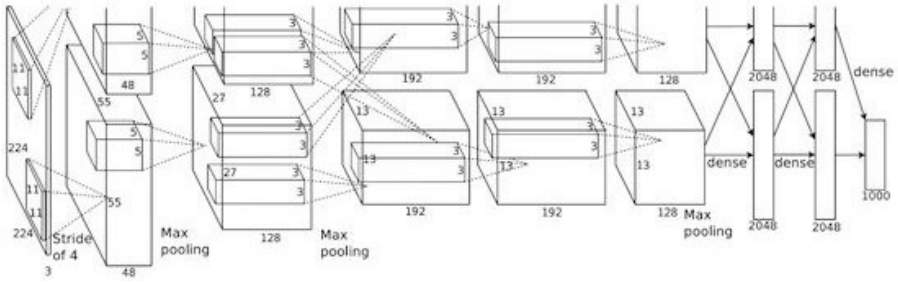
- Just the introduction…
- The real science/art (or dark art?) lies at the following

  1. How to design the architecture of the NN

     Key ideas: should leverage problem's intrinsic structure to more cleverly connect neurons, so that learning is more effective



Or

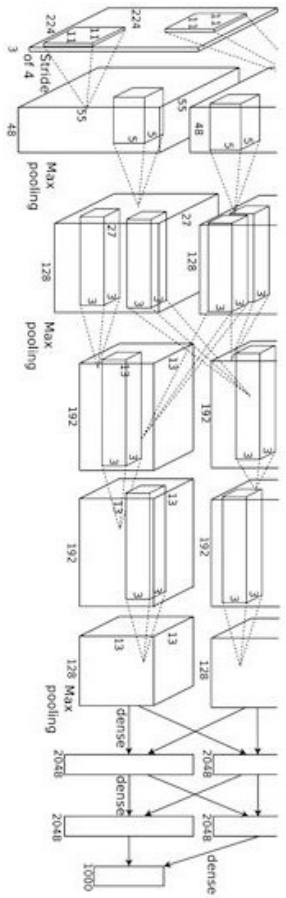


Neural Network Architecture Design

input image

weights

loss

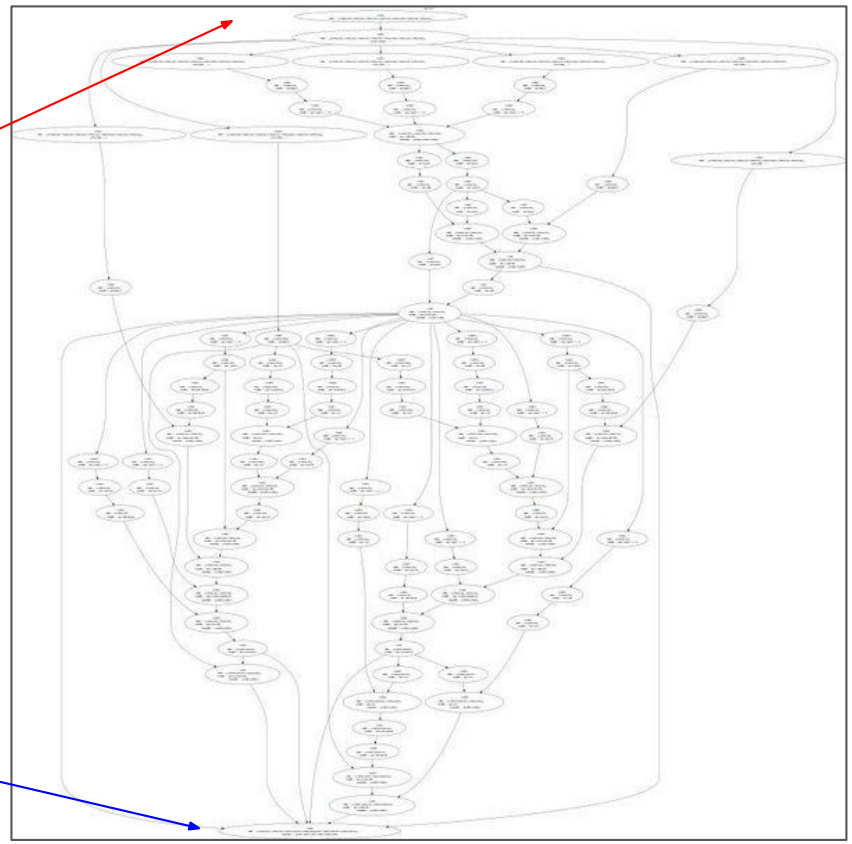


AlexNet

Figure from the seminal paper of Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012.

input image

loss



Really complex neural networks!!

Convolutional neural network (CNN)

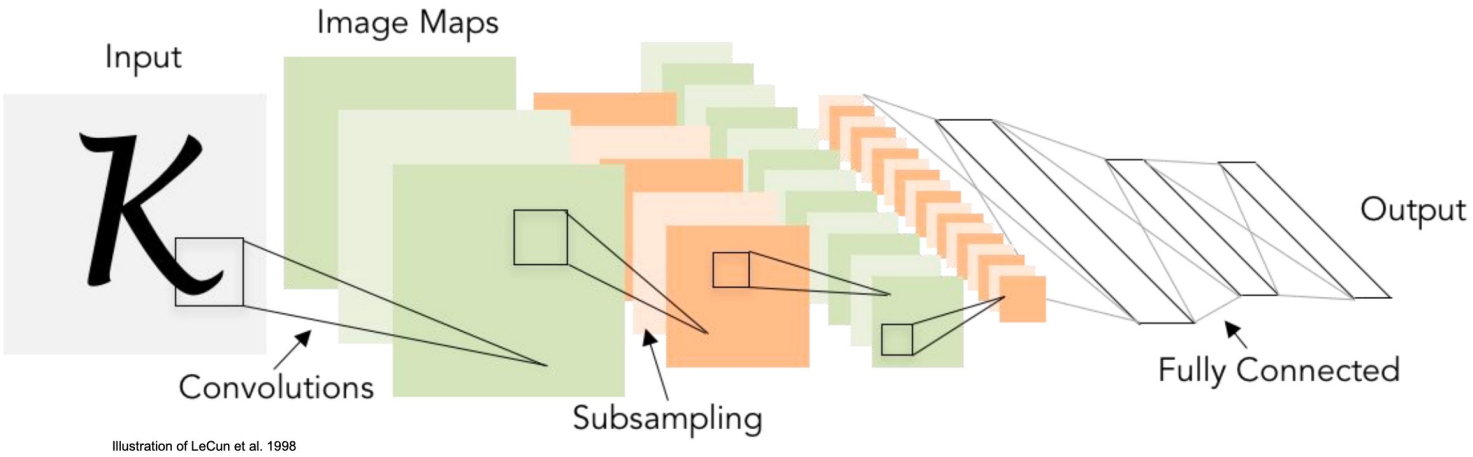
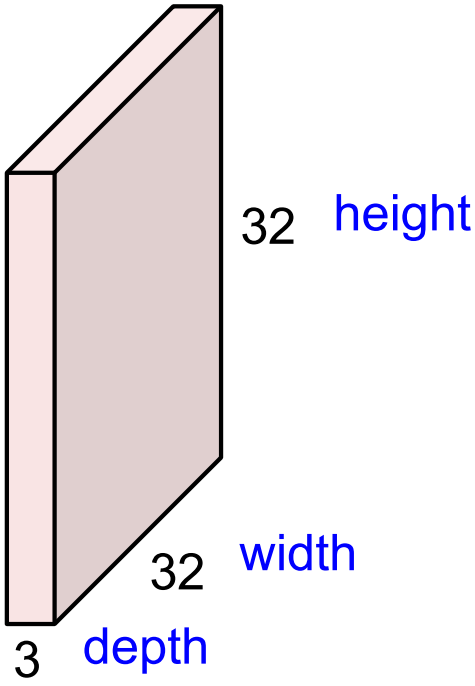


Illustration of LeCun et al. 1998

Convolution Layer

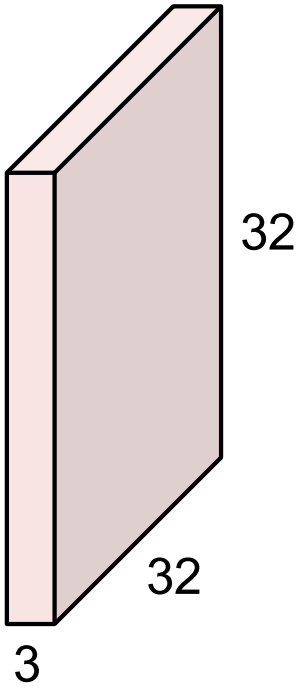
32x32x3 image -> preserve spatial structure



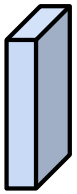
Main idea: only look at small patches of an image

Convolution Layer

32x32x3 image



5x5x3 filter

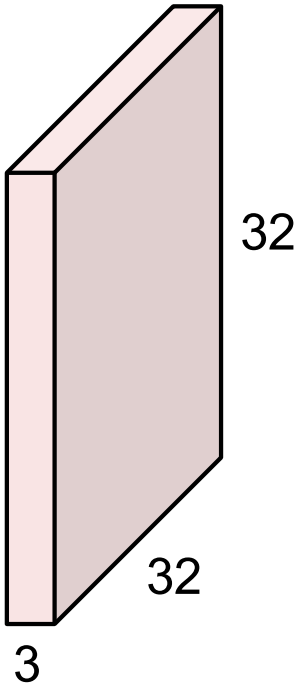


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

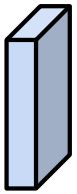
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

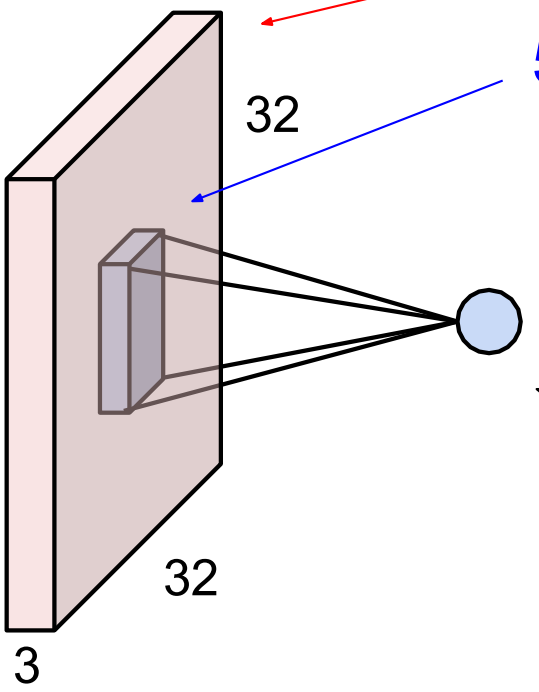


5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer



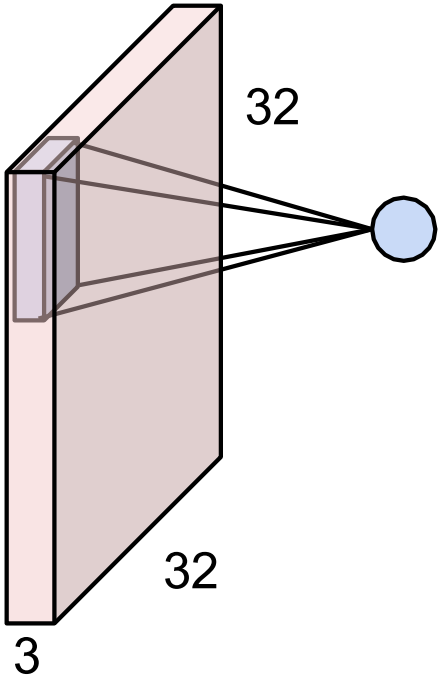
32x32x3 image

5x5x3 filter w

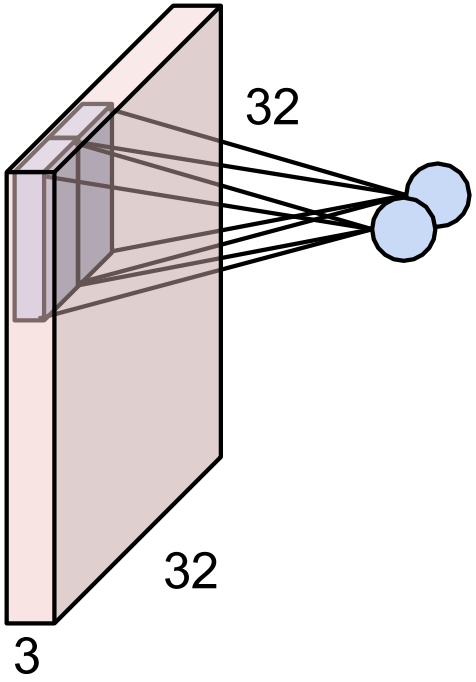
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

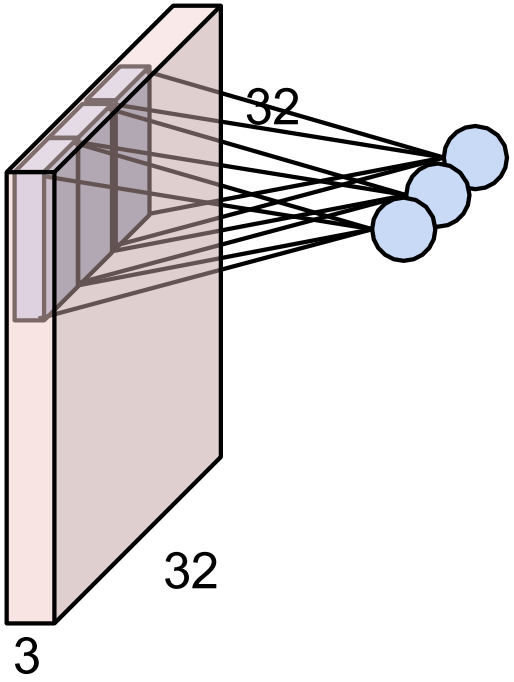
Convolution Layer



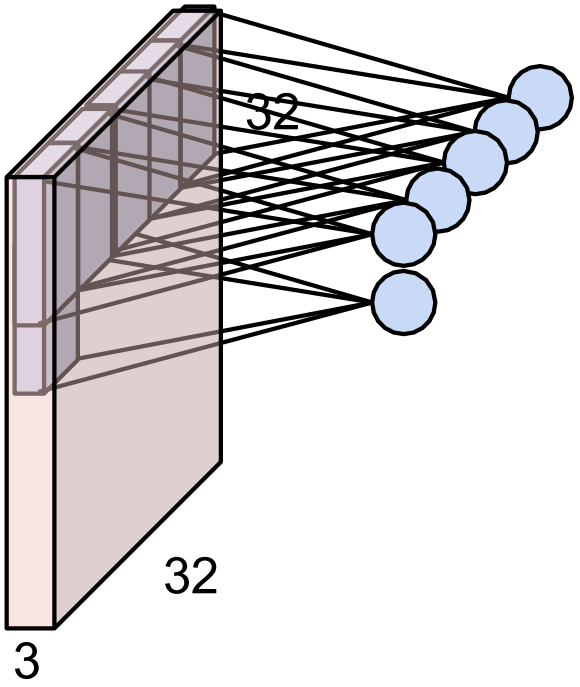
Convolution Layer



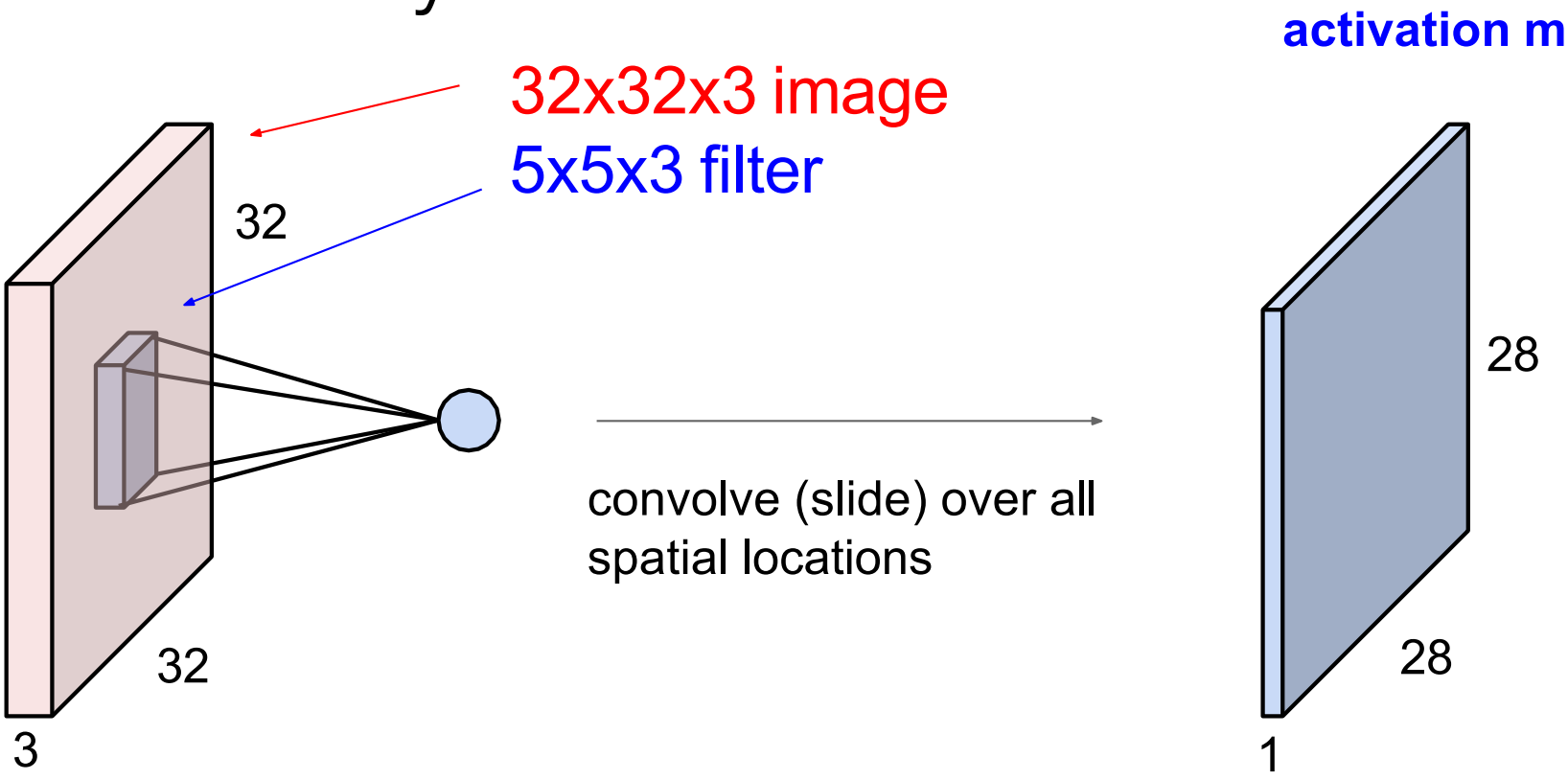
Convolution Layer



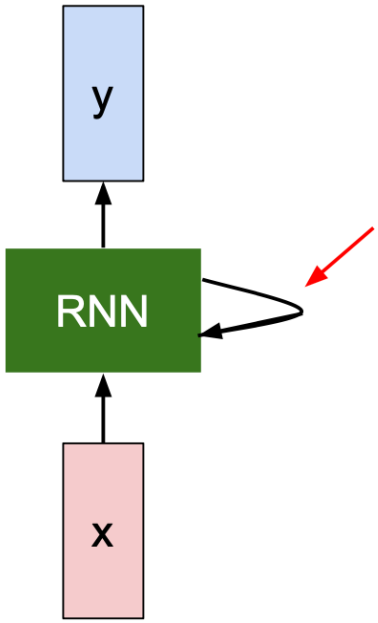
Convolution Layer



Convolution Layer

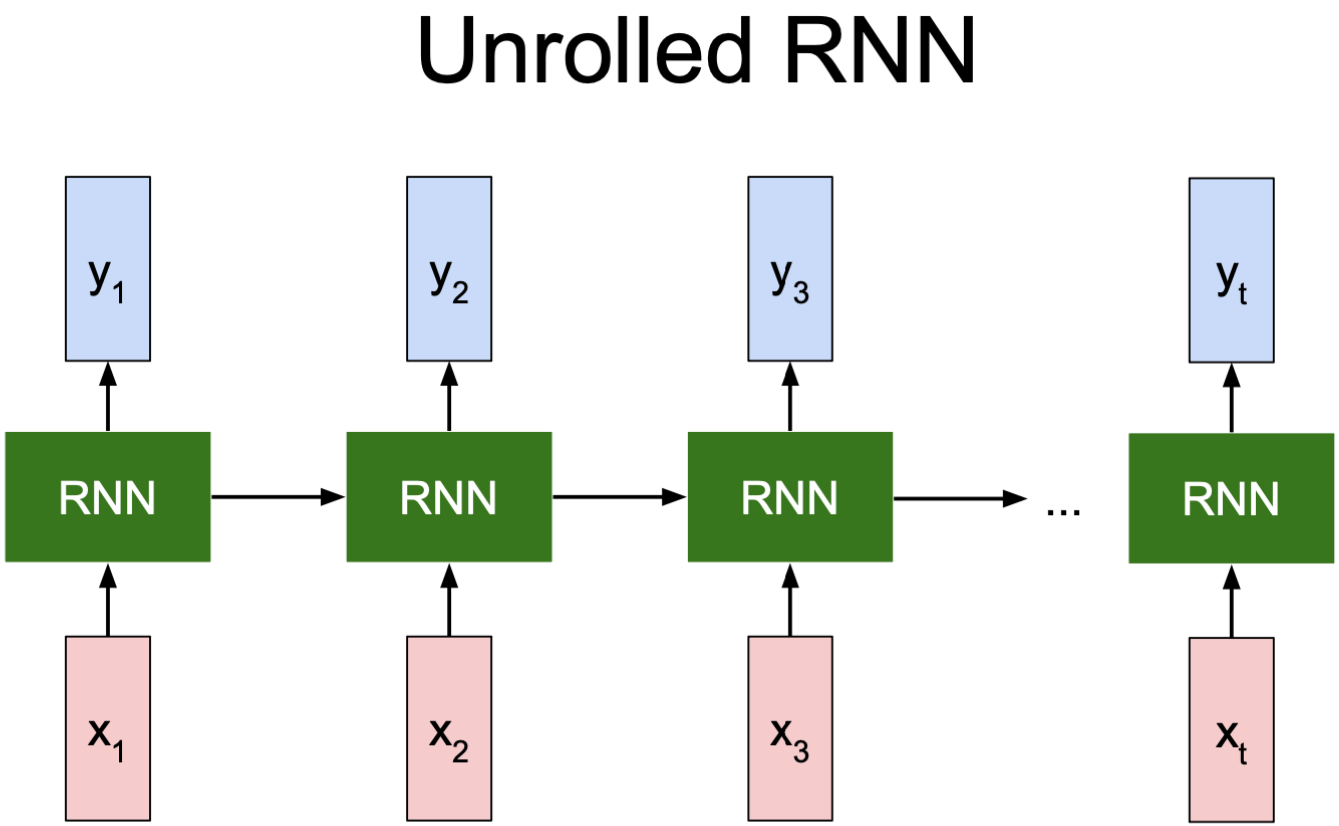


Recurrent neural network (RNN)



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Recurrent neural network (RNN)



- RNN turns out to be very helpful for processing sequential data
- Transformer (the architecture most LLMs rely on these days) is a variant of RNN that uses something called “attention mechanism”

Okay… so that's it for neural network research?

- Just the introduction…
- The real science/art (or dark art?) lies at the following

1. How to design the architecture of the NN
2. How to find the weight parameters of the designed NN?

Answer:

- ✓ Optimizing loss function over data – a procedure called "training" neural networks
- ✓ Another aspect of deep learning that is full of mysteries – this is what we will cover in next lecture

Great Resources

- Andrej Karpathy's CS231N Neural Networks
- Steve Miller's How to build a neural network
- Michael Nielsen's Neural Networks and Deep Learning, Chapter 1

End of Lecture for Today
Thank You