CS 540 Introduction to Artificial Intelligence
Midterm Review II

University of Wisconsin–Madison
Fall 2025, Section 3
October 22, 2025

Announcements

- HW6 released Friday
- Classroom change this Friday, 10/24
  Psychology Building, Room 113
  (regular time)

Midterm Information

- **Time:** Thursday October 23rd 7:30-9 PM
- **Location:**
  - Section 001 : 6210 Social Sciences Building
  - Section 002 : B10 Ingrahm Hall
  - **Section 003: 3650 Humanities Building**

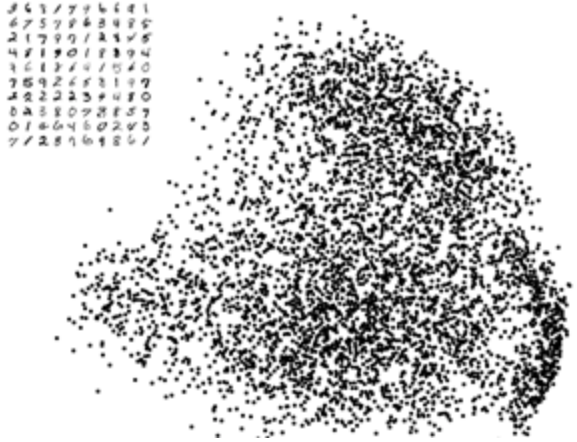
- McBurney students and students requesting alternate: reach out to your instructor if you have not received any email!
- Format: multiple choice
- Cheat sheet: single handwritten piece of paper, front and back
- Calculator: fine if it doesn't have an Internet connection
- Detailed topic list + practice on Piazza and Canvas

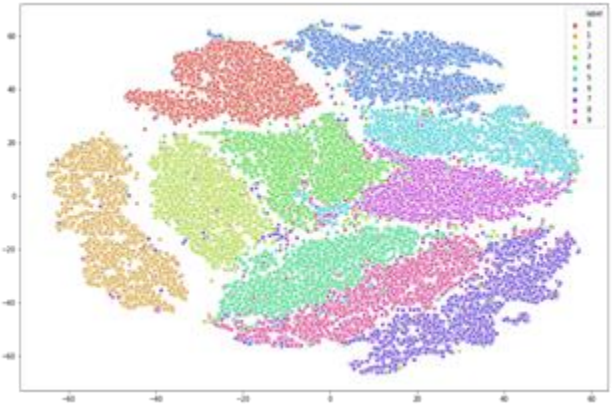
Requested Topics

t-SNE

Common tool for visualization

- Project/embed data into low dimensions
- Not a linear transformation
- Tries to maintain structure





MNIST visualizations
Above: using PCA
Below: using t-SNE

n-grams & (Laplace) smoothing

- Standard: train a bigram model by counting

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

- Laplace smoothing adds one to every count, bigrams we haven't seen in the data

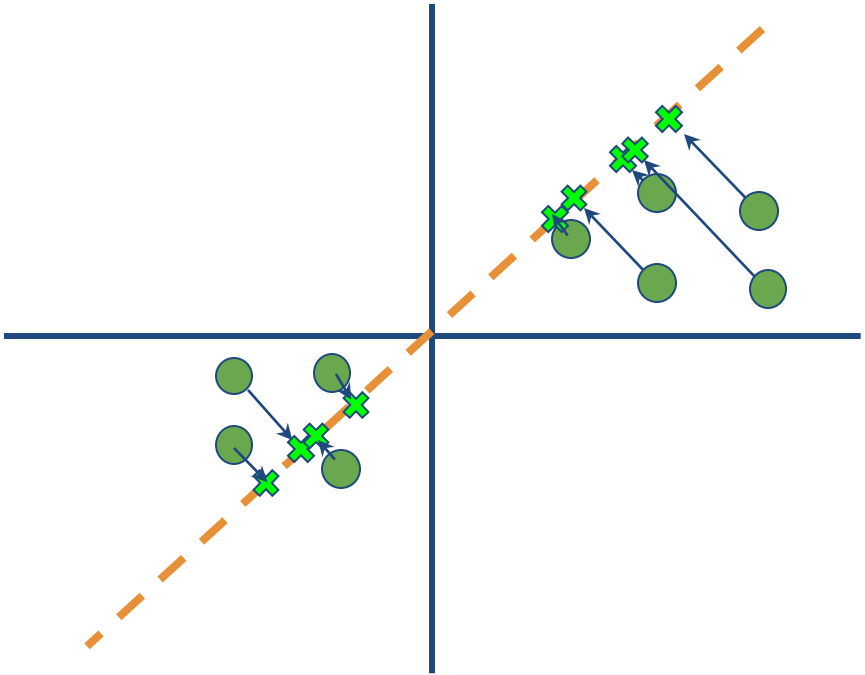
$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + V}$$

Vocabulary size, number of words or characters

- This smooths out/regularizes the language model

PCA Details (1/5)

$x_1, x_2, \ldots, x_n \in \mathbb{R}^2$



PCA says: a good projection makes the data “spread out”

PCA Details (2/5)

- Direction: vector v with length $\|v\|_2 = 1$

$$\underset{v\,:\,\|v\|_2=1}{\operatorname{argmax}} \sum_{i=1}^{n} \langle x_i, v \rangle^2$$

- Inner product $\Leftrightarrow$ projection

How can we solve this?
Turn to linear algebra!

PCA Details (3/5)

- Write $x_1, \ldots, x_n$ as a matrix X

$$X = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ — & \vdots & — \\ — & x_n & — \end{bmatrix}$$

$X \in \mathbb{R}^{n \times d}$
n rows and
d columns

PCA Details (4/5)

PCA objective:

$$\underset{v\,:\,\|v\|_2=1}{\operatorname{argmax}} \sum_{i=1}^{n} \langle x_i, v \rangle^2$$

Equivalent:

$$\underset{v\,:\,\|v\|_2=1}{\operatorname{argmax}}\, v^T X^T X v$$

Inner product!

$$Xv = \begin{bmatrix} — & x_1 & — \\ — & x_2 & — \\ — & \vdots & — \\ — & x_n & — \end{bmatrix} \begin{bmatrix} | \\ v \\ | \end{bmatrix} = \begin{bmatrix} \langle x_n, v \rangle \\ \langle x_n, v \rangle \\ \vdots \\ \langle x_n, v \rangle \end{bmatrix}$$

PCA Details (5/5)

Equivalent to PCA objective:

$$\underset{v\,:\,\|v\|_2=1}{\operatorname{argmax}}\, v^T X^T X v$$

Recall: eigenvector & eigenvalue of A satisfy

$$Av = \lambda v$$

- Suppose v is an eigenvector of $X^T X$
- Then
$$\begin{aligned} v^T X^T X v &= v^T(\lambda v) \\ &= \lambda v^T v \\ &= \lambda \|v\|_2^2 = \lambda \end{aligned}$$
- PCA objective ⇔ top eigenvector of $X^T X$
- $\frac{1}{n} X^T X$ is **sample covariance**

Bayesian classification

What if **x** has multiple attributes $\mathbf{x} = \{X_1, \ldots, X_k\}$

$$\hat{y} = \arg\max_y p(y \mid X_1, \ldots, X_k)$$ (Posterior)

(Prediction)

$$= \arg\max_y \frac{p(X_1, \ldots, X_k \mid y) \cdot p(y)}{p(X_1, \ldots, X_k)}$$ (by Bayes' rule)

$$= \arg\max_y \; p(X_1, \ldots, X_k \mid y)\; p(y)$$

Class conditional likelihood

Class prior

Naïve Bayes Assumption

Conditional independence of feature attributes

$$p(X_1, \ldots, X_k \mid y)p(y) = \Pi_{i=1}^{k} p(X_i \mid y)p(y)$$

Easier to estimate

(using MLE!)

Example: Classify emails as spam

- Features = words in vocabulary
- One parameter θ_w for each word w
- Classify new emails as spam or not spam

Dear Valued Winner,

Congratulations! Your email address has been randomly selected as the GRAND PRIZE WINNER of **$5,000,000 USD** in our International Lottery Promotion.

Reply immediately with your credit card information to claim your prize…

$$p(\text{spam} \mid \text{email}) \propto p(\text{email} \mid \text{spam})p(\text{spam})$$

use naïve Bayes assumption to simplify this term

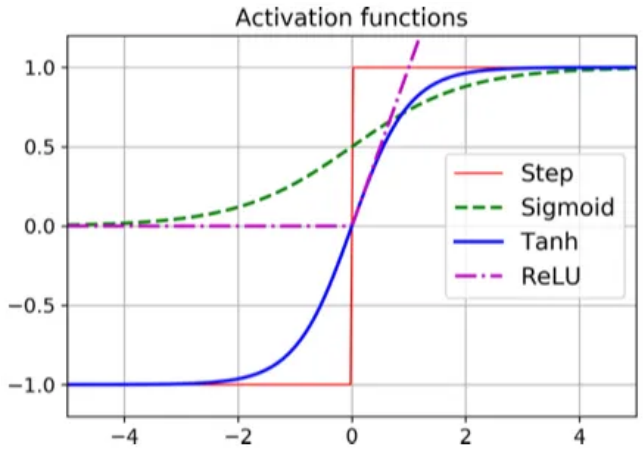
$$p(\text{“dear”} \mid \text{spam})p(\text{“valued”} \mid \text{spam})p(\text{“winner”} \mid \text{spam}) \cdots p(\text{“prize”} \mid \text{spam})$$

estimate each term independently with MLE

Activation Functions

Used for neural network hidden nodes

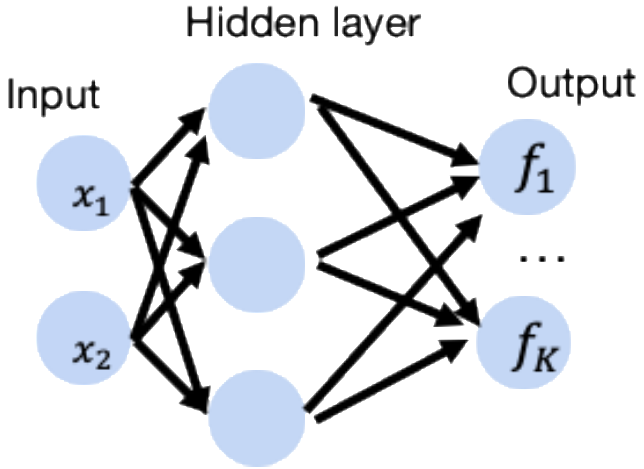
- Step/Hard Threshold: $\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & o.w. \end{cases}$
- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
- tanh: $\sigma(z) = \tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$
- ReLU: $\sigma(z) = \max\{0, z\}$



https://kinder-chen.medium.com/hidden-layer-activation-functions-6fd65489ed25

Softmax

- Used for multiclass classification
- Not an activation function
  - Not applied element-wise



- Turns k values $f_1, f_2, \ldots, f_k$ into probability distribution over k classes

$$\text{softmax}(f) = \frac{\exp(f_y(x))}{\sum_{k=1}^{K} \exp(f_k(x))}$$

Loss Functions: Regression

- Squared Error: $\ell(y, \hat{y}) = (\hat{y} - y)^2$

- If our model predicts $\hat{y} = f_\theta(x)$, we write
$$\ell(\theta; x, y) = (f_\theta(x) - y)^2$$

- Over a dataset of n examples: MSE
$$L(\theta; X, y) = \frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2$$

Loss Functions: Classification

- Misclassification Error
  - Used for perceptron

$$\ell(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{if } y \neq \hat{y} \end{cases}$$

- Binary Cross-Entropy
  - $y \in \{0,1\}, \hat{y} \in [0,1]$

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

- Cross Entropy
  - For $k \geq 2$ classes
  - True label y is one-hot vector
  - Prediction $\hat{y}$ is a probability distribution over k classes (like the output of softmax)

$$\ell(y, \hat{y}) = -\sum_{i=1}^{k} y_i \log \hat{y}_i$$

Foundations

Bayesian **Inference**

- Fancy name for what we just did. Terminology:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- *H* is the hypothesis
- *E* is the evidence

Bayesian **Inference**

- Terminology:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \longleftarrow \textbf{Prior}$$

- Prior: estimate of the probability **without** evidence

Bayesian Inference

- Terminology:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

- Likelihood: probability of evidence **given a hypothesis**

Bayesian Inference

- Terminology:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

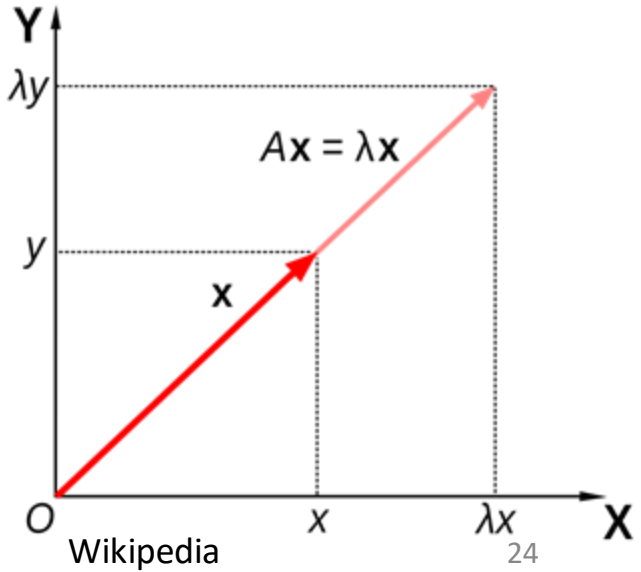
↑

Posterior

- Posterior: probability of hypothesis **given evidence**.

Eigenvalues & Eigenvectors

- For a square matrix *A*, solutions to $Av = \lambda v$
  - *v* (nonzero) is a vector: **eigenvector**
  - λ is a scalar: **eigenvalue**
  - Intuition: A is a linear transformation;
  - Can stretch/rotate vectors;
  - E-vectors: only stretched (by e-vals)



Wikipedia

Dimensionality Reduction

Reduce dimensions

- Why?
  - Lots of features redundant
  - Storage & computation costs

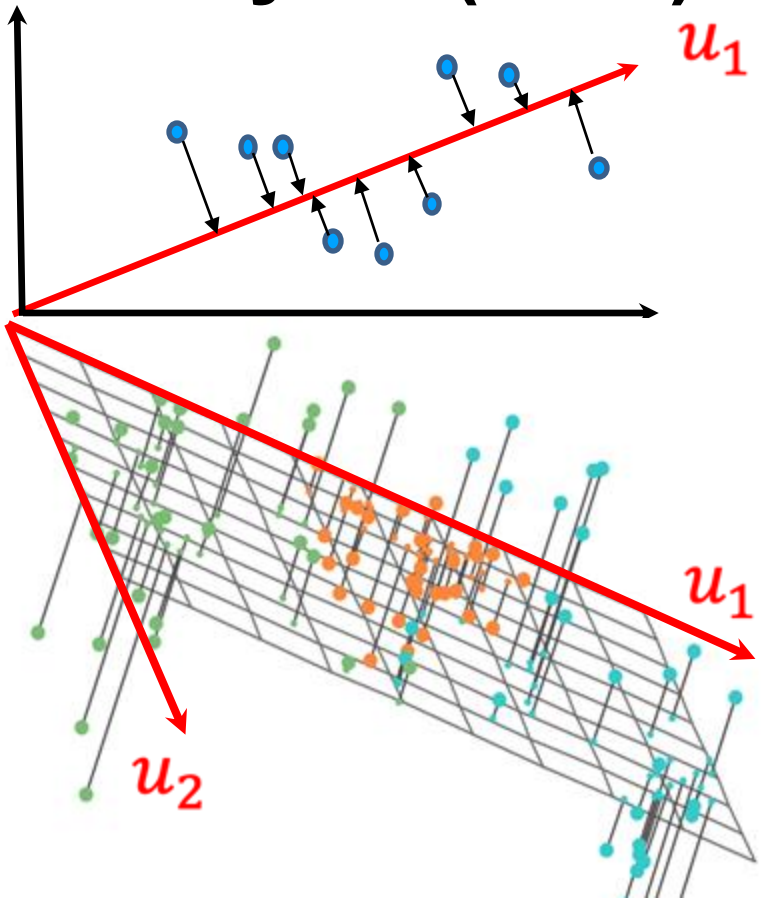
CreativeBloq

- Goal: take $x \in \mathbb{R}^d \rightarrow x \in \mathbb{R}^r$ for $r << d$
  - But, minimize information loss

Principal Components Analysis (PCA)

- Find **axes** $u_1, u_2, \ldots, u_m \in \mathbb{R}^d$ of a subspace
  - Will project to this subspace
- Want to preserve data
  - minimize projection error

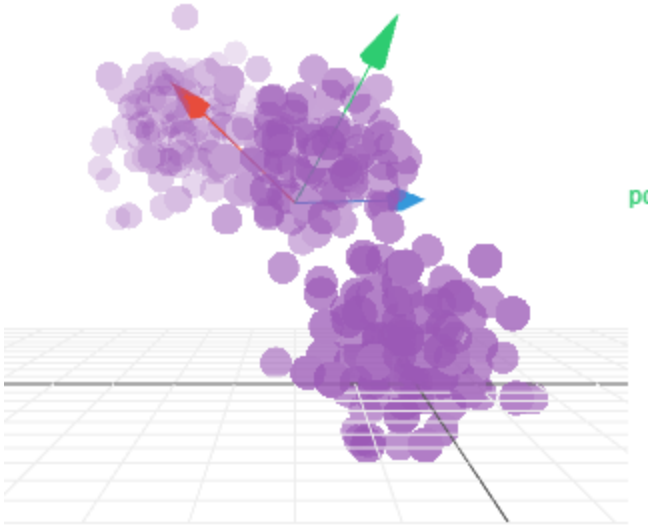
- These vectors are the **principal components**



PCA Procedure

Inputs: data $x_1, x_2, \ldots, x_n \in \mathbb{R}^d$

– **Center data so that** $\frac{1}{n}\sum_{i=1}^{n} x_i = 0$



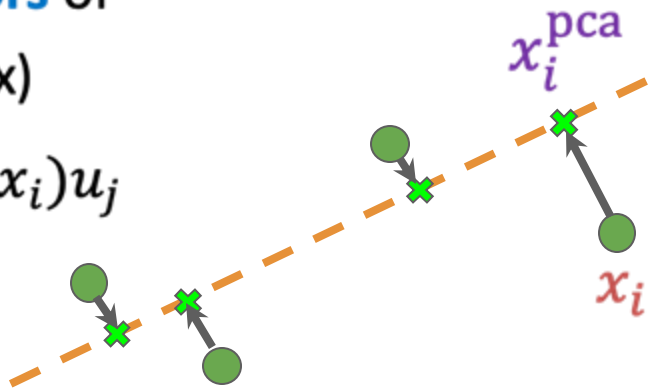
Victor Powell

PCA Procedure

Output:

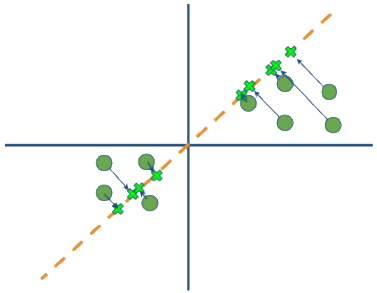
principal components $u_1, \ldots, u_m \in \mathbb{R}^d$

- Orthogonal
- Can show: they are top-m **eigenvectors** of $S = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^\top$ (covariance matrix)
- Each x_i projected to $x_i^{\text{pca}} = \sum_{j=1}^{m} (u_j^\top x_i) u_j$



PCA and Eigenvectors

- Want to project down to one dimension



- Goal: maximize variance after projection

$$\underset{v\,:\,\|v\|_2=1}{\operatorname{argmax}} \sum_{i=1}^{n} \langle x_i, v \rangle^2$$

- Principal component = top eigenvector of sample covariance

$$(X^T X)v = \lambda v$$

Basic Logic

Arguments, premises, conclusions

- Argument: a set of sentences (premises) + a sentence (a conclusion)
- **Validity:** argument is valid iff it's necessary that if all premises are true, the conclusion is true
- **Soundness:** argument is sound iff valid & premises true
- **Entailment:** when valid arg., premises entail conclusion

Evaluating a Sentence

Example:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
$false$	$false$	$true$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$false$	$true$	$true$	$false$
$true$	$false$	$false$	$false$	$true$	$false$	$false$
$true$	$true$	$false$	$true$	$true$	$true$	$true$

Note:

- If P is false, P $\Rightarrow$ Q is true regardless of Q (“5 is even implies 6 is odd” is True!)
- Causality not needed: “5 is odd implies the Sun is a star” is True!)

First-Order Logic: Quantifiers

- Universal quantifier: ∀
- Sentence is true **for all** values of x in the domain of variable x.

- Main connective typically is ⇒
  - Forms if-then rules
  - “all humans are mammals”

$$\forall x \ \mathtt{human}(x) \Rightarrow \mathtt{mammal}(x)$$

  - Means if x is a human, then x is a mammal

First-Order Logic: Quantifiers

- Existential quantifier: ∃
- Sentence is true **for some** value of x in the domain of variable x.

- Main connective typically is ∧
  - "some humans are male"

$$\exists x \ \text{human}(x) \wedge \text{male}(x)$$

  - Means there is an x who is a human and is a male

Language Models

Basic idea: use probabilistic models to **assign a probability to a sentence W**

$$P(W) = P(w_1, w_2, \ldots, w_n) \text{ or } P(w_{\text{next}} | w_1, w_2 \ldots)$$

Goes back to Shannon

- Information theory: letters

Zero-order approximation	XFOML RXKHRJFFJUJ ALPWXFWJXYJ FFJEYVJCQSGHYD QPAAMKBZAACIBZLKJQD
First-order approximation	OCRO HLO RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL
Second-order approximation	ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE
Third-order approximation	IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE
First-order word approximation	REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE

Training: Make Assumptions

- Markov assumption with shorter history:

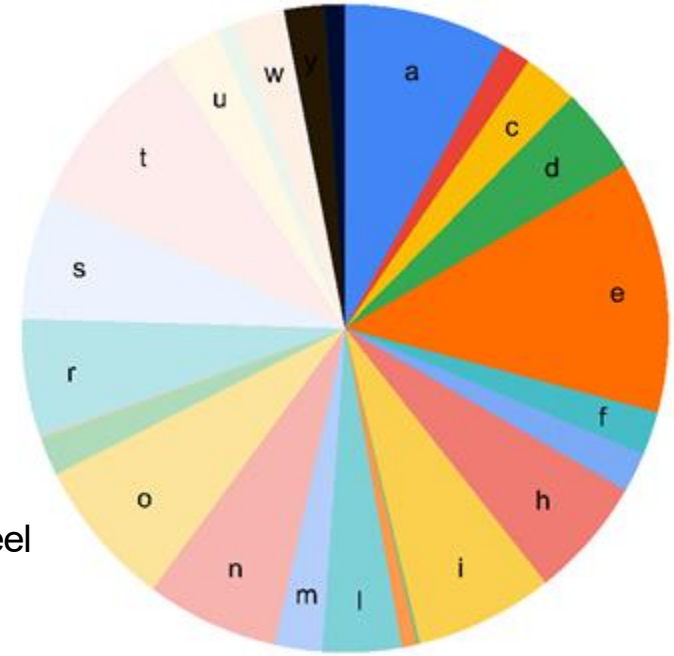
$$P(w_i|w_{i-1}w_{i-2}\dots w_1) = P(w_i|w_{i-1}w_{i-2}\dots w_{i-k})$$

- Present doesn't depend on whole past
  - Just recent past, i.e., *context*.
  - What's ***k=0?***

k=0: Unigram Model

- Full independence assumption:
  - (Present doesn't depend on the past)

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2)\ldots P(w_n)$$



The English letter frequency wheel

k=1: Bigram Model

- Markov Assumption:
  - (Present depends on immediate past)

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2)\ldots P(w_n|w_{n-1})$$

$p(.|q)$: the "after q" wheel

$p(.|j)$: the "after j" wheel

texaco, rose, one, in, this, issue, is, pursuing, growth, in, a, boiler, house, said, mr., gurria, mexico, 's, motion, control, proposal, without, permission, from, five, hundred, fifty, five, yen outside, new, car, parking, lot, of, the, agreement, reached this, would, be, a, record, november

k=n-1: n-gram Model

Can do trigrams, 4-grams, and so on

- More expressive as *n* goes up
- Harder to estimate

Training: just count? I.e, for bigram:

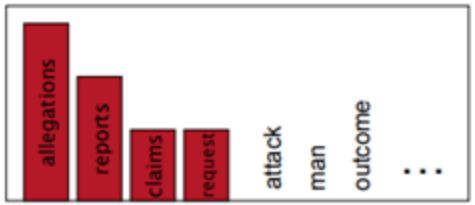
$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

n-gram Training

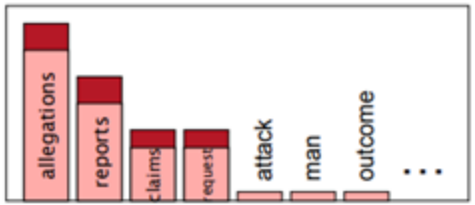
Issues:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

- **1**. Multiply tiny numbers?
  - **Solution**: use logs; add instead of multiply
- **2.** n-grams with zero probability?
  - **Solution**: smoothing



$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + V}$$



Dan Klein

Representing Words

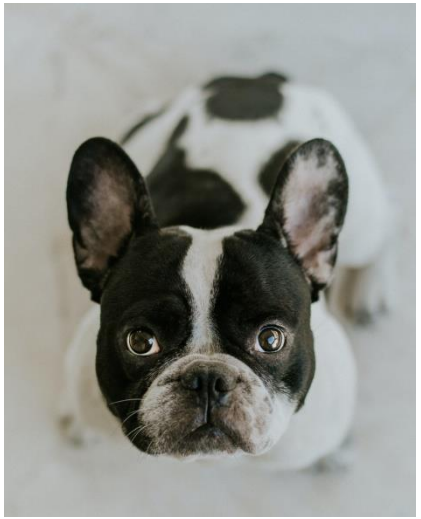
Remember value of random variables **(RVs)**

Easier to work with than objects like ‘dog’

Traditional representation: **one-hot vectors**

$$\text{dog} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Dimension: # of words in vocabulary
- Relationships between words?



Smarter Representations

Distributional semantics: account for relationships

Reps should be close/similar to other words that appear in a similar context

Dense vectors:

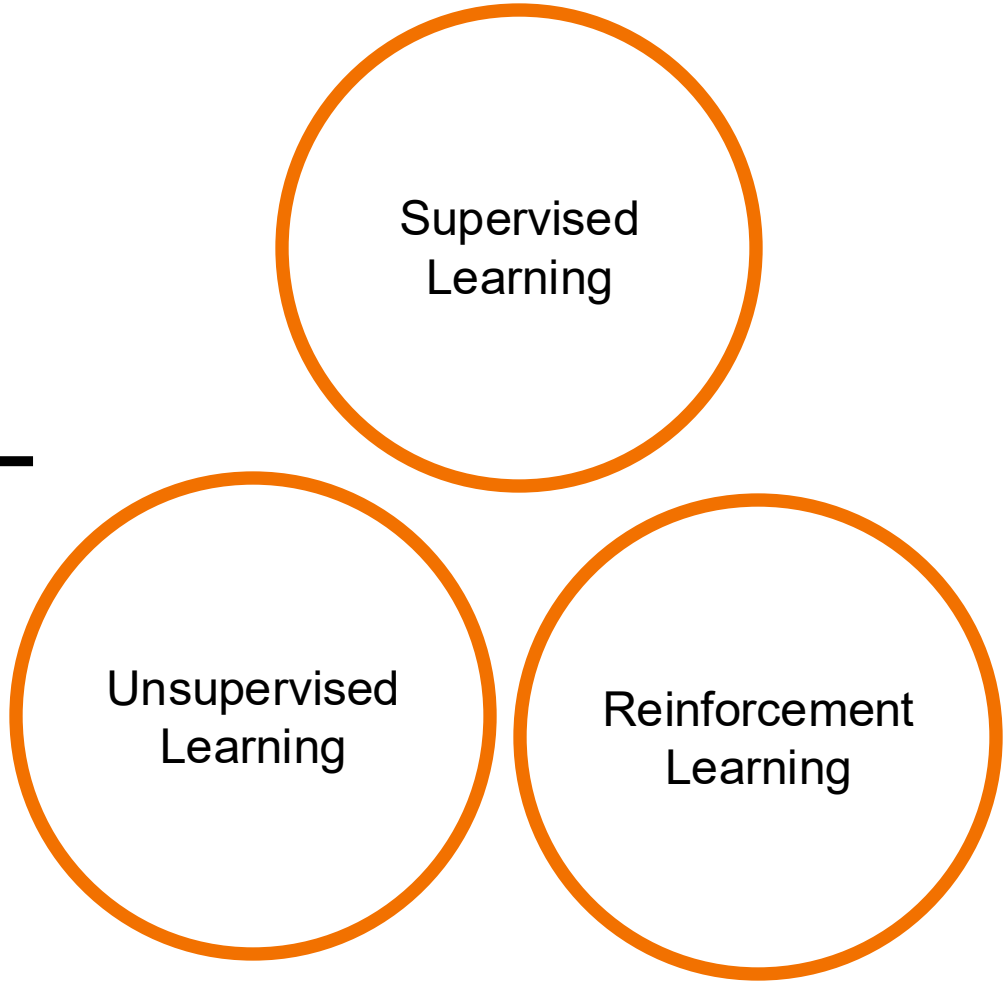
$$\text{dog} = \begin{bmatrix} 0.13 & 0.87 & -0.23 & 0.46 & 0.87 & -0.31 \end{bmatrix}^T$$

$$\text{cat} = \begin{bmatrix} 0.07 & 1.03 & -0.43 & -0.21 & 1.11 & -0.34 \end{bmatrix}^T$$

AKA **word embeddings**

Machine Learning Basics

Taxonomy of ML



Supervised/Unsupervised Learning

Supervised learning:

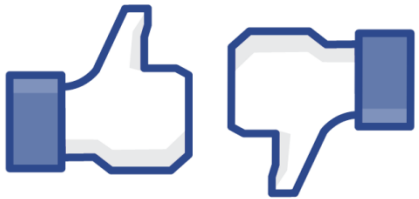
Make predictions, classify data, perform regression

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

Features / Covariates / Input

Labels / Outputs

Goal: find function $f : X \to Y$ to predict label on **new** data



indoor

outdoor

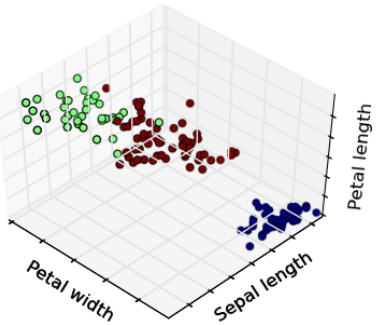
Supervised/Unsupervised Learning

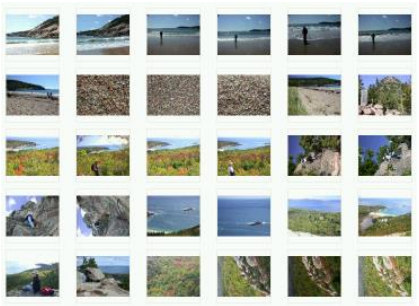
Unsupervised learning:

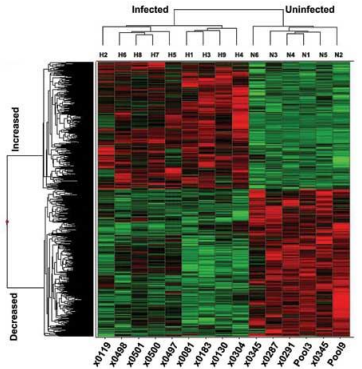
No labels; generally, won't be making predictions

Dataset: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$

Goal: find patterns & structures that help better understand data.



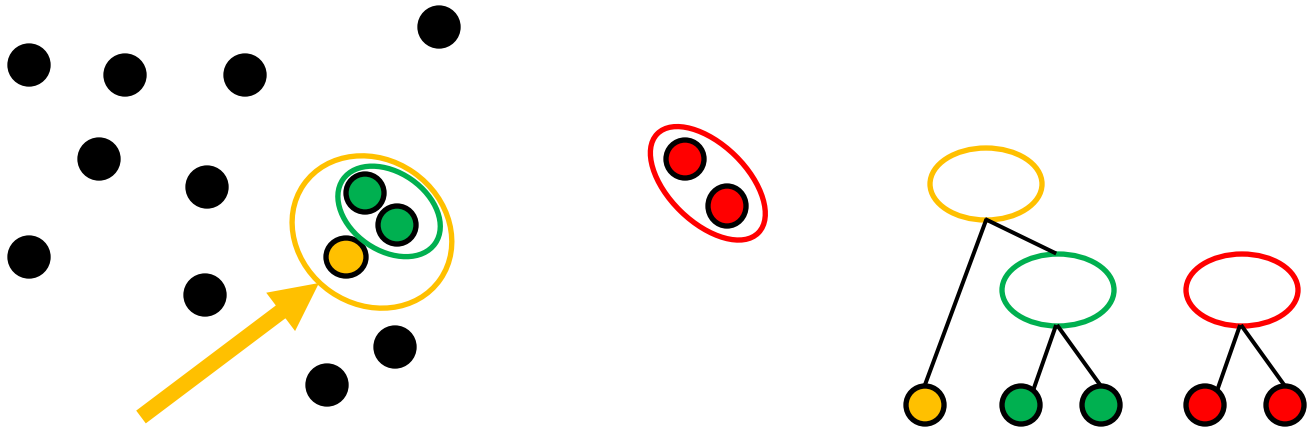




Mulvey and Gingold

Agglomerative Clustering Example

Repeat: Get pair of clusters that are closest and merge



Merging Criteria

Merge: use closest clusters. Define closest?

Single-linkage

$$d(A,B) = \min_{x_1 \in A, x_2 \in B} d(x_1, x_2)$$

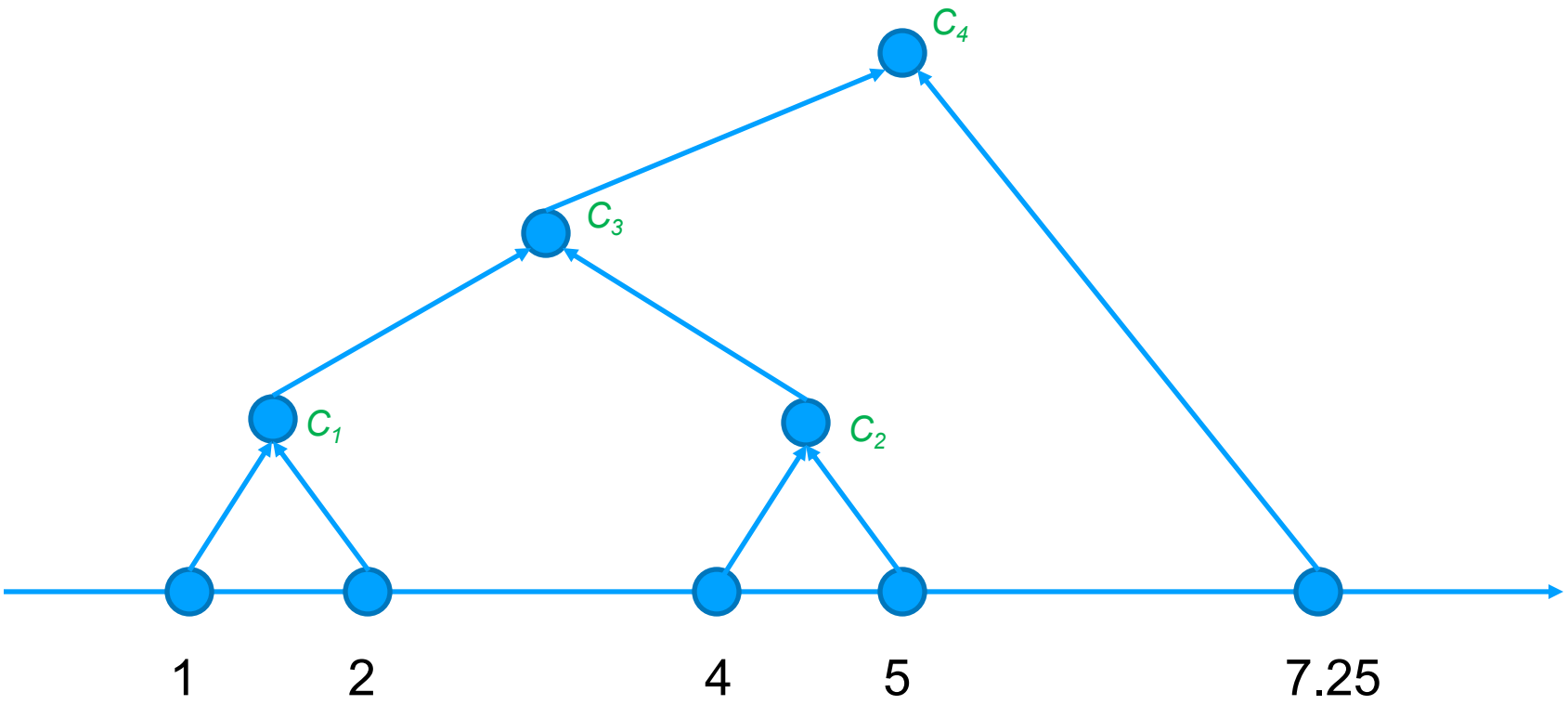
Complete-linkage

$$d(A,B) = \max_{x_1 \in A, x_2 \in B} d(x_1, x_2)$$

Average-linkage

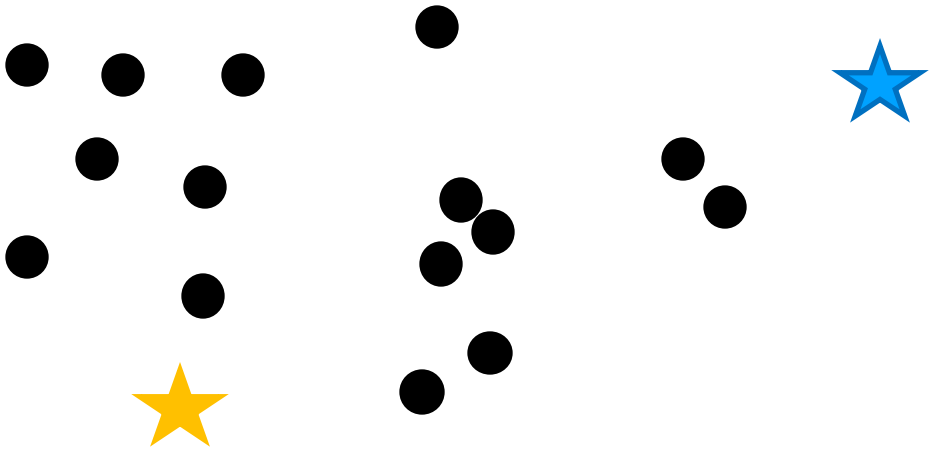
$$d(A,B) = \frac{1}{|A||B|} \sum_{x_1 \in A, x_2 \in B} d(x_1, x_2)$$

Single-linkage Example



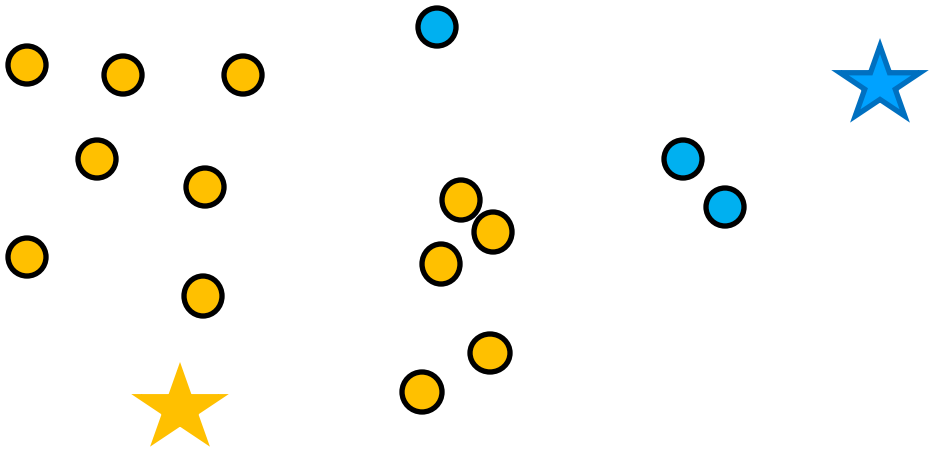
K-Means Clustering

Steps: **1**. Randomly pick k cluster centers



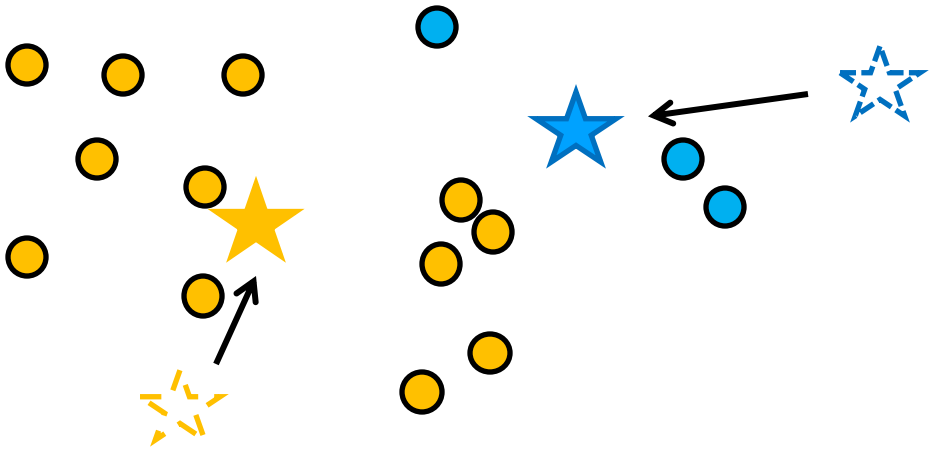
K-Means Clustering

2. Find closest center for each point



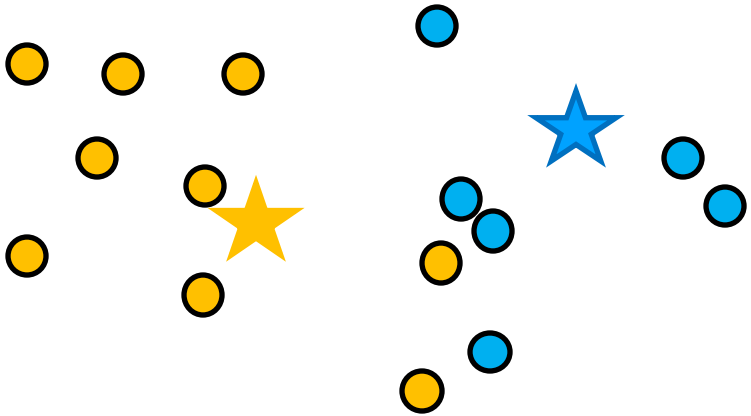
K-Means Clustering

3. Update cluster centers by computing centroids



K-Means Clustering

Repeat Steps 2 & 3 until convergence



K-means algorithm

Input: $x_1, x_2, \ldots, x_n, k$

Step 1: select k cluster centers $c_1, c_2, \ldots, c_k$

Step 2: for each point x_i, assign it to the closest center in Euclidean distance:

$$y(x_i) = \text{argmin}_j \, ||x_i - c_j||$$

Step 3: update all cluster centers as the centroids:

$$c_j = \frac{\sum_{x:y(x)=j} x}{\sum_{x:y(x)=j} 1}$$

Repeat Step 2 and 3 until cluster centers no longer change

Two Types of Supervised Learning Algorithms

Classification

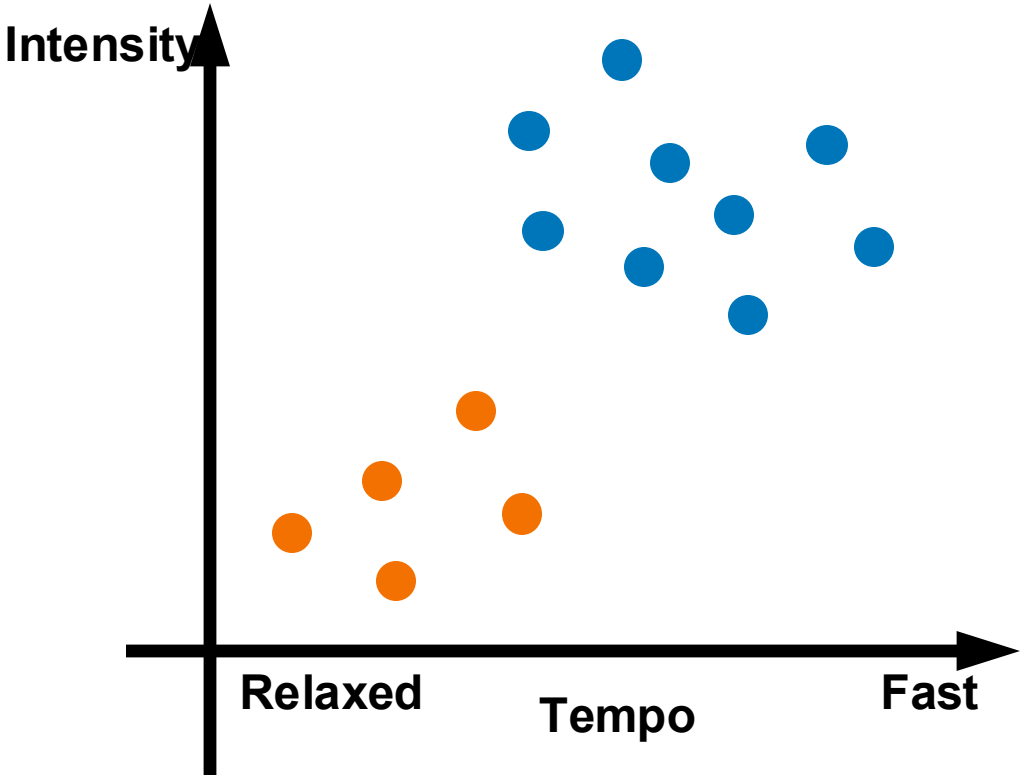
Regression

Example of Classification: Predict Song Preferences

User Sharon

Dislike

Like

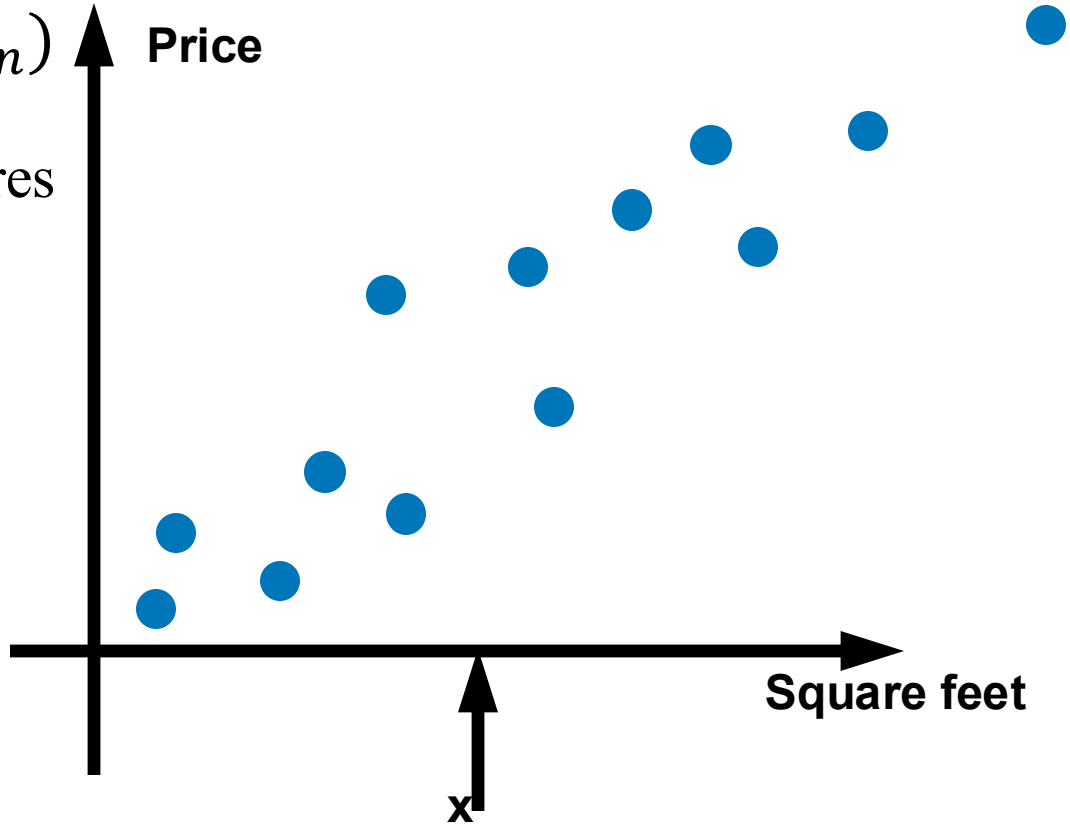


Example of regression: housing price prediction

Given: a dataset that contains n samples

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$$

Task: if a residence has **x** squares feet, predict the price

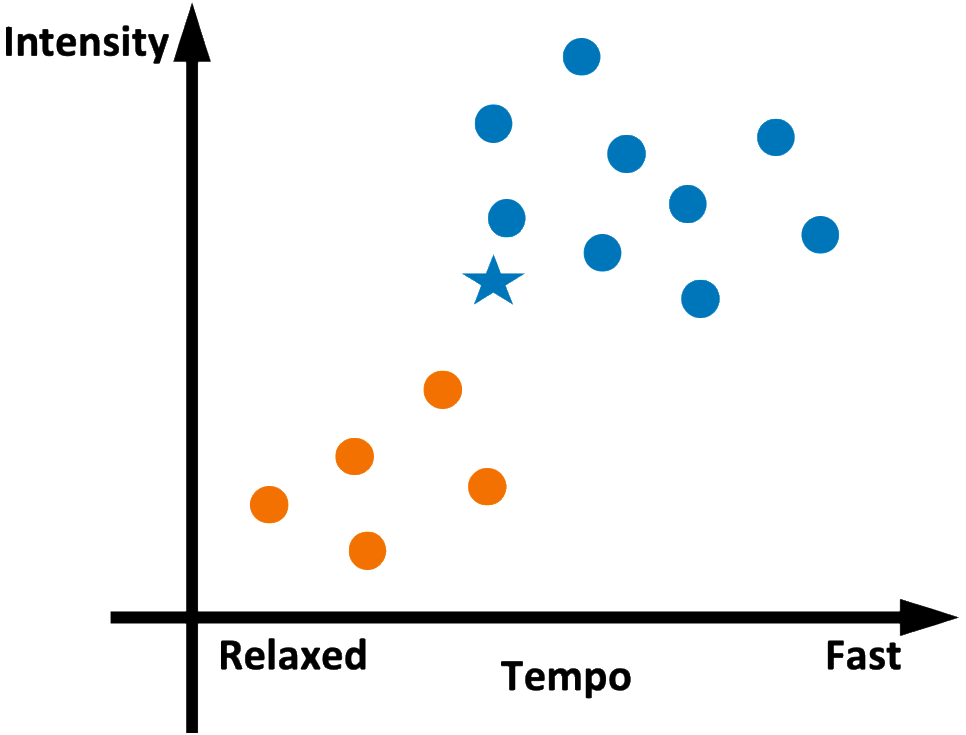


k-Nearest Neighbor Classifier

User Sharon

Dislike

Like



k-Nearest Neighbor Classifier

- **Input**: Training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$

  Distance function $d(\mathbf{x}_i, \mathbf{x}_j)$; number of neighbors k; test data $\mathbf{x}^*$

1. Find the k training instances $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$ closest to $\mathbf{x}^*$ under $d(\mathbf{x}_i, \mathbf{x}_j)$
2. Output y^*, the majority class of $y_{i_1}, \ldots, y_{i_k}$. Break ties randomly.

Basic Recipe for Supervised Learning

1. Select model class
2. Select loss
3. Optimize parameters
4. Evaluate output

Usually: apply a variant of **gradient descent**

Usually: compute loss on **test set**

Maximum Likelihood Estimation

Find θ which is the best "explanation" for the data

- MLE solves

$$\underset{\theta}{\operatorname{argmax}}\, p(x_1, \ldots, x_n \mid \theta) = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} p(x_i \mid \theta)$$

- Rewrite the problem in an equivalent form

$$\underset{\theta}{\operatorname{argmax}}\, p(x_1, \ldots, x_n \mid \theta) = \underset{\theta}{\operatorname{argmin}}(-\log p(x_1, \ldots, x_n \mid \theta))$$

$$= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} -\log p(x_i \mid \theta)$$

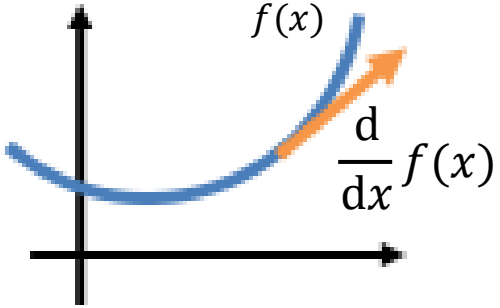
Connecting MLE and Loss Minimization

- We call "$-\log p(x_i \mid \theta)$" the **negative log likelihood**
- May define $\ell(\theta; x_i) \coloneqq -\log p(x_i \mid \theta)$
- Maximum likelihood estimation is **loss minimization.** Different notation, same computation.

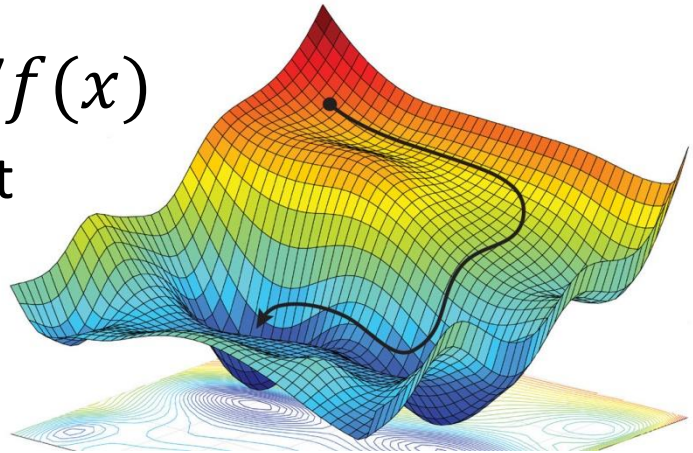
$$\underset{\theta}{\operatorname{argmax}}\, p(x_1, \ldots, x_n \mid \theta) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(\theta; x_i)$$

Gradients & Gradient Descent

- One dimension: derivative $\frac{\mathrm{d}}{\mathrm{d}x} f(x)$
  - How to shift x to make $f(x)$ larger



- Higher dimensions: gradient $\nabla f(x)$
  - Direction where f grows fastest

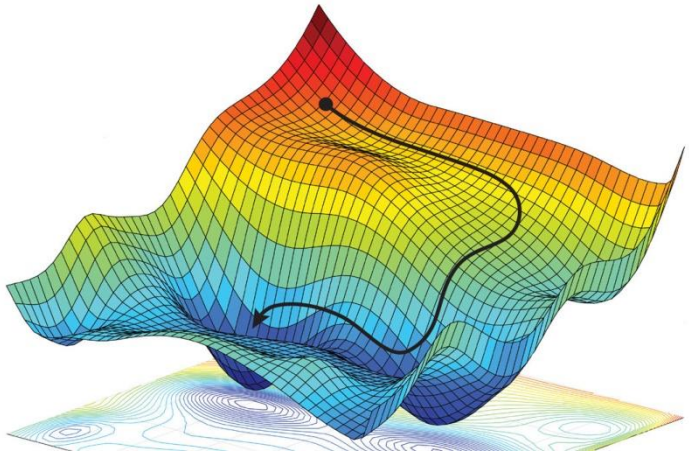


M. Hutson

Gradients & Gradient Descent

- Gradient descent takes iterative steps to make loss function smaller

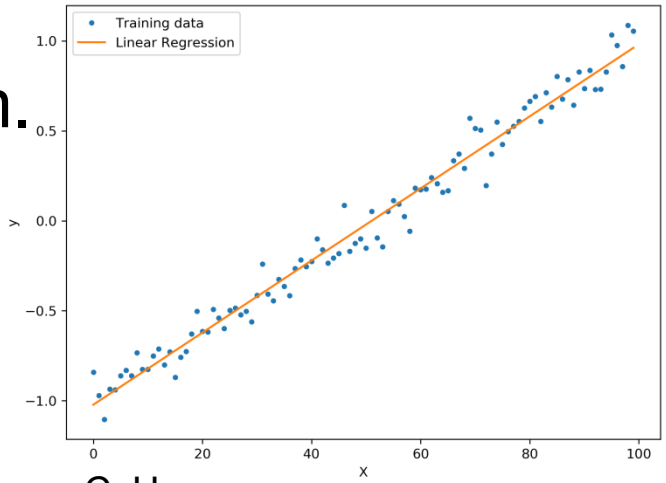
Gradient Descent
Input: dataset (X, y), loss function L, number of steps T, step size η
1. Initialize θ_0 2. For $t = 1,2, \ldots, T$ 3. Calculate $g_t = \nabla L(\theta_{t-1}; X, y)$ 4. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$ 5. Return θ_T



M. Hutson

Linear Regression

Simplest type of regression problem.



C. Hansen

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

- *x*'s are vectors, *y*'s are scalars.
- "**Linear**": predict a linear combination of x components + intercept

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d = \theta_0 + x^T \theta$$

Want: parameters θ

Linear Regression Setup

Problem Setup

Goal: figure out how to minimize square loss

Let's organize it. Train set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$

- Since $f(x) = \theta_0 + x^T\theta$, use a notational trick by augmenting feature vector with a constant dimension of 1: $x = \begin{bmatrix} 1 \\ x \end{bmatrix}$

- Then, with this one more dimension we can write (θ contains θ_0 now) $f(x) = x^T\theta$

Linear Regression Setup

Problem Setup

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

Take train features and make it a n*(d+1) matrix, and y a vector:

$$X = \begin{bmatrix} x_1^T \\ \ldots \\ x_n^T \end{bmatrix} \qquad y = \begin{bmatrix} y_1 \\ \ldots \\ y_n \end{bmatrix}$$

Then, the empirical risk is $\frac{1}{n}\|X\theta - y\|^2$

Linear Regression → Classification?

What if we want the same idea, but *y* is 0 or 1?

Need to convert the $\theta^T x$ to a probability in [0,1]

$$p(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)}$$

Logistic function

Why does this work?

If $\theta^T x$ is really big, $\exp(-\theta^T x)$ is really small → *p* close to 1

If really negative exp is huge → *p* close to 0

"Logistic Regression"

Neural Networks: Multilayer Perceptron

Review: Perceptron

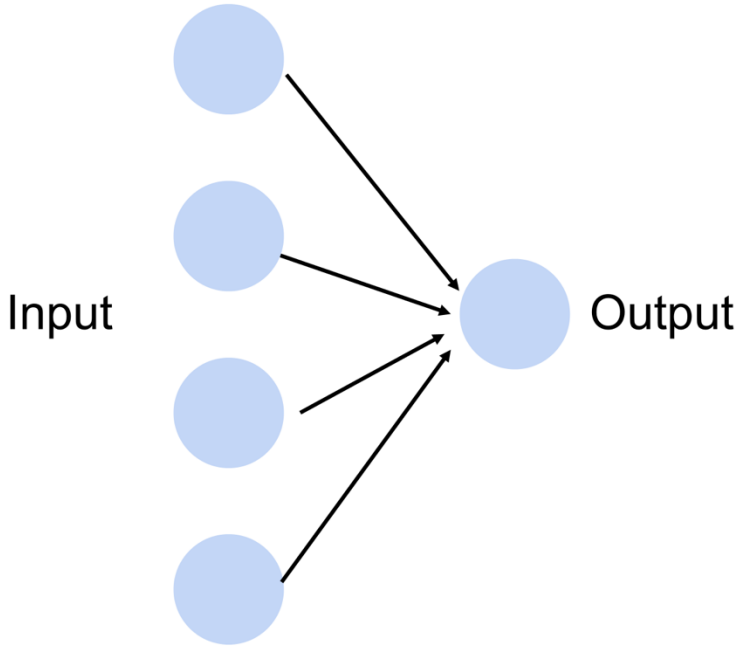
- Given input $\mathbf{x}$, weight $\mathbf{w}$ and bias b, perceptron outputs:

$$o = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle + b)$$

$$\sigma(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$$

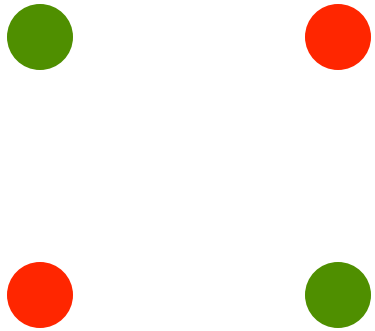
Activation function

Cats vs. dogs?



XOR Problem (Minsky & Papert, 1969)

The perceptron cannot learn an XOR function
(it can only generate linear separators)



This contributed to the first AI winter

More complicated neural networks: multiple hidden layers

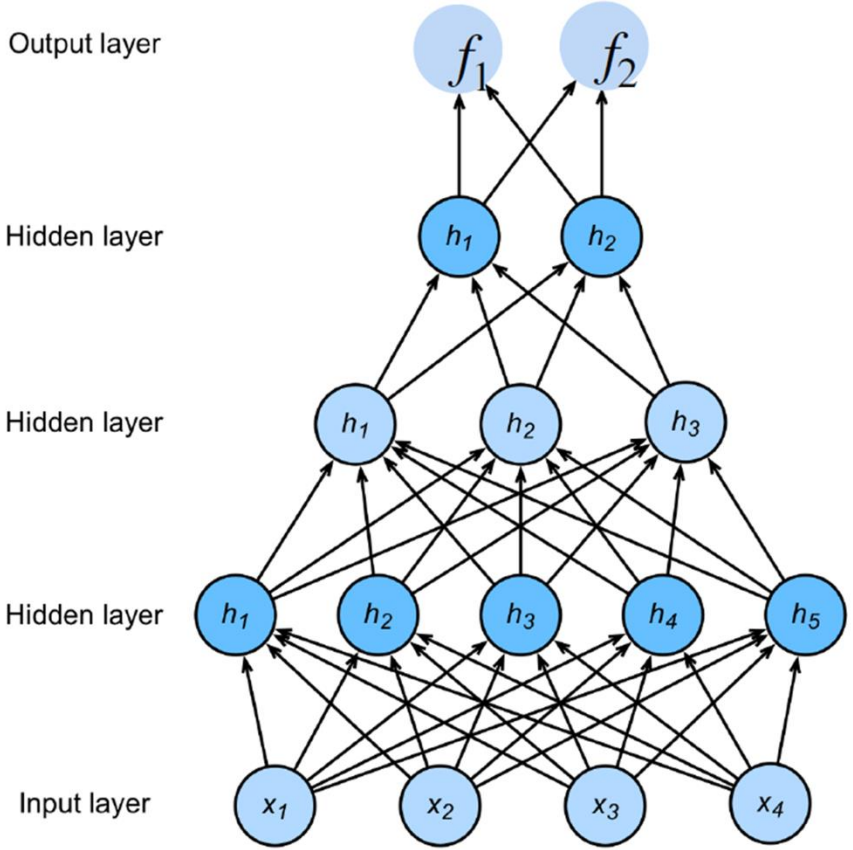
$$\mathbf{h}_1 = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

$$\mathbf{h}_2 = \sigma(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)})$$

$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

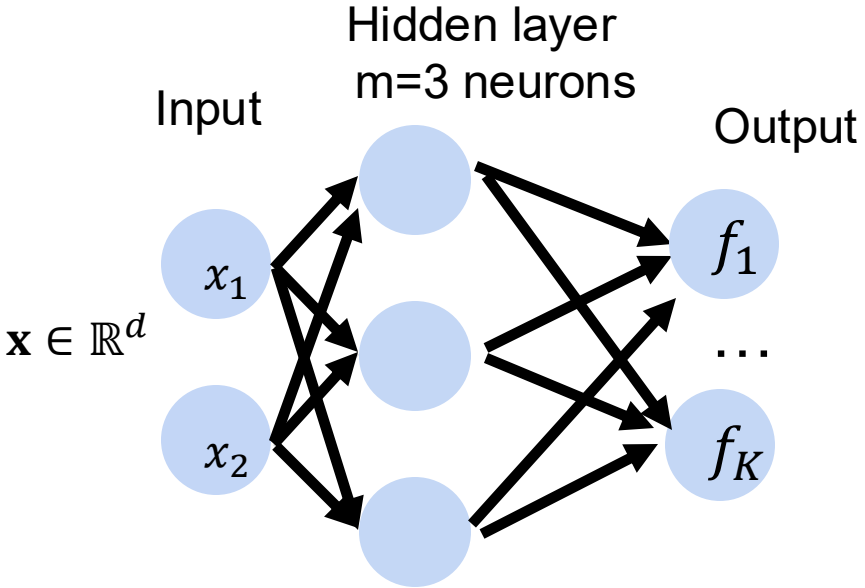
$$\mathbf{f} = \mathbf{W}^{(4)}\mathbf{h}_3 + \mathbf{b}^{(4)}$$

$$\mathbf{p} = \text{softmax}(\mathbf{f})$$



Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)

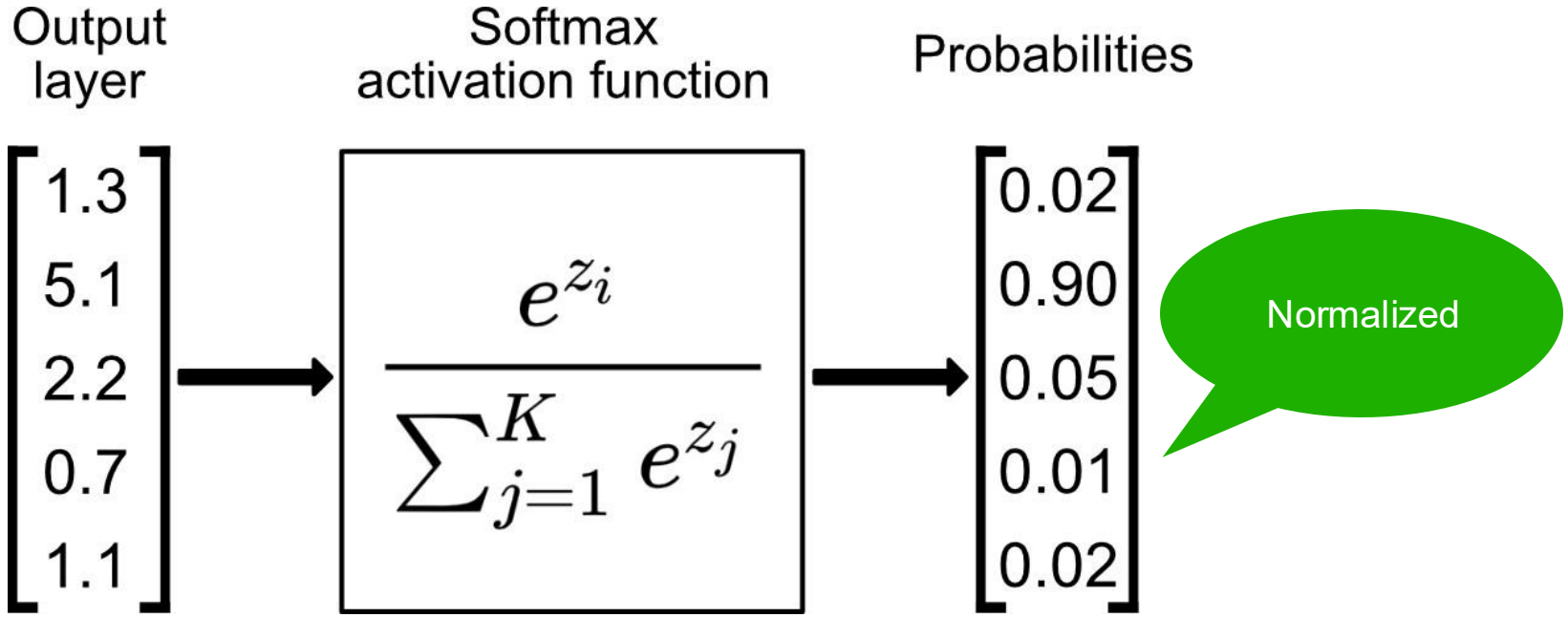


$$p(y|\mathbf{x}) = softmax(f)$$

$$= \frac{\exp(f_y(x))}{\sum_{k=1}^{K} \exp(f_k(x))}$$

Softmax

Turns outputs f into probabilities (sum up to 1 across K classes)



Neural Networks: Training

Basic Recipe: Multilayer Perceptron

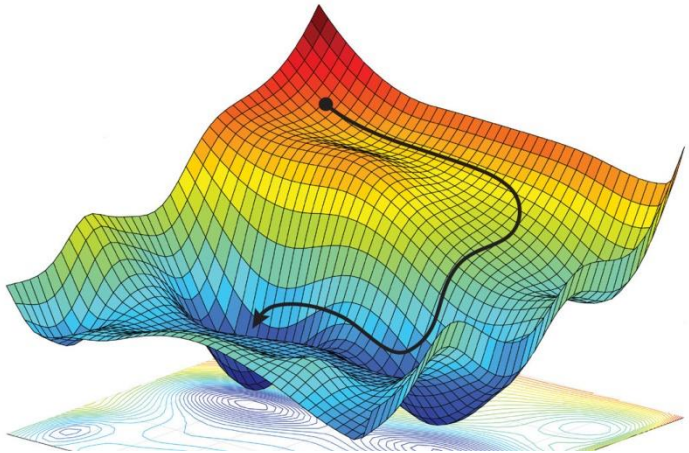
1. Select model class
2. Select loss
3. Optimize parameters
4. Evaluate output

1. Stacked Perceptrons
   (various activation functions)
2. Cross-entropy loss
   (when performing classification)
3. Gradient Descent
   (usually, a variant of gradient decent)
4. Test/train split

Gradients & Gradient Descent

Gradient descent takes iterative steps to make loss function smaller

Gradient Descent
Input: dataset (X, y), loss function L, number of steps T, step size η
1. Initialize θ_0 2. For $t = 1,2, \ldots, T$ 3. Calculate $g_t = \nabla L(\theta_{t-1}; X, y)$ 4. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$ 5. Return θ_T



M. Hutson

(Minibatch) Stochastic Gradient Descent

- Gradient descent uses loss on entire dataset every step:

$$\nabla L(\theta; X, y) = \sum_{i=1}^{n} \nabla \ell(\theta; x_i, y_i)$$

- This is extremely inefficient!

- On big datasets, better to update based on a few examples

Stochastic Gradient Descent
Input: dataset (X, y), loss function L, number of steps T, step size η, batch size m
1. Initialize θ_0 2. For $t = 1,2, \dots, T$ 3. Select random $(x_1, y_1), \dots, (x_m, y_m)$ 4. Calculate $g_t = \sum_{i=1}^{m} \nabla_\theta \ell(\theta_{t-1}; x_i, y_i)$ 5. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$ 6. Return θ_T

Gradient Descent, Alternative View

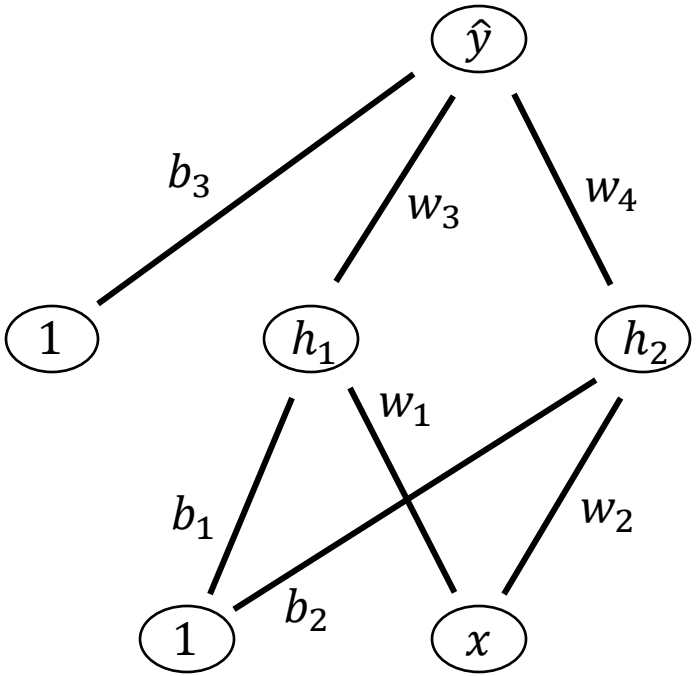
- Parameter vector θ contains all weights and biases:

$$\theta = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

- Gradient update step:

$$\theta_t \leftarrow \theta_{t-1} - \eta \nabla_\theta L(\theta; X, y)$$

- Update all the parameters at once



Gradient Descent, Alternative View

Gradient Descent

Input: dataset (X, y), loss function L, number of steps T, step size η

1. Initialize θ_0
2. For $t = 1,2, \ldots, T$
3. Calculate $g_t = \nabla L(\theta_{t-1}; X, y)$
4. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$
5. Return θ_T

Look at one update operation

Compute partial derivative of loss for each weight and bias parameter:

$$\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \frac{\partial L}{\partial w_3}, \frac{\partial L}{\partial w_4}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial b_3}$$

Update all parameter values:

$$w_1^{\text{new}} \leftarrow w_1^{\text{old}} - \eta \cdot \frac{\partial L}{\partial w_1}$$

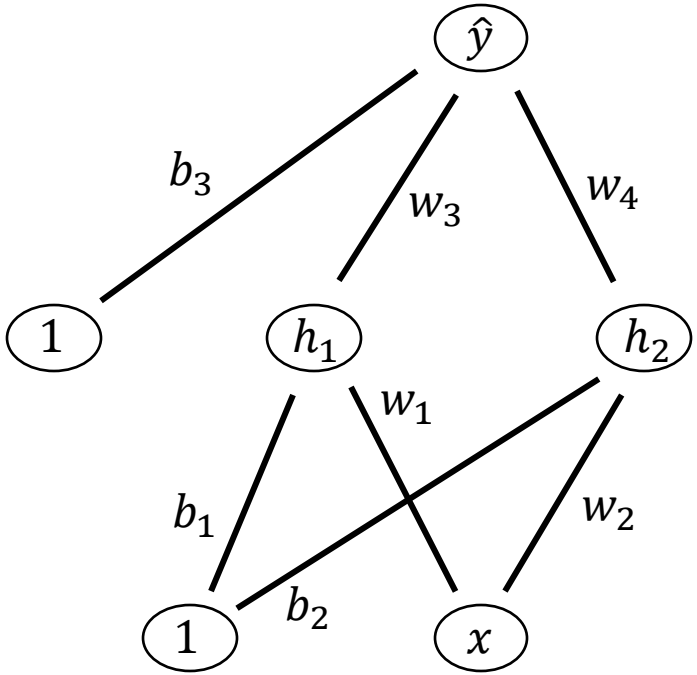
$$w_2^{\text{new}} \leftarrow w_2^{\text{old}} - \eta \cdot \frac{\partial L}{\partial w_2}$$

$$\vdots$$

$$b_3^{\text{new}} \leftarrow b_3^{\text{old}} - \eta \cdot \frac{\partial L}{\partial b_3}$$

How do we compute all these partial derivatives? Backprop!

A Simple Multilayer Perceptron



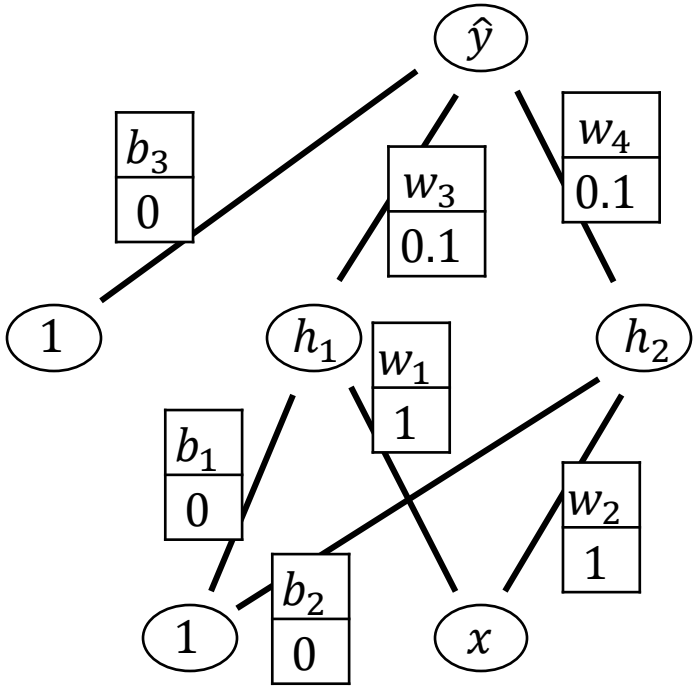
Hidden node: ReLU activations

$$h_1 = \max\{0, w_1 x + b_1\}$$

Output node: no activation

$$\hat{y} = w_3 h_1 + w_4 h_2 + b_3$$

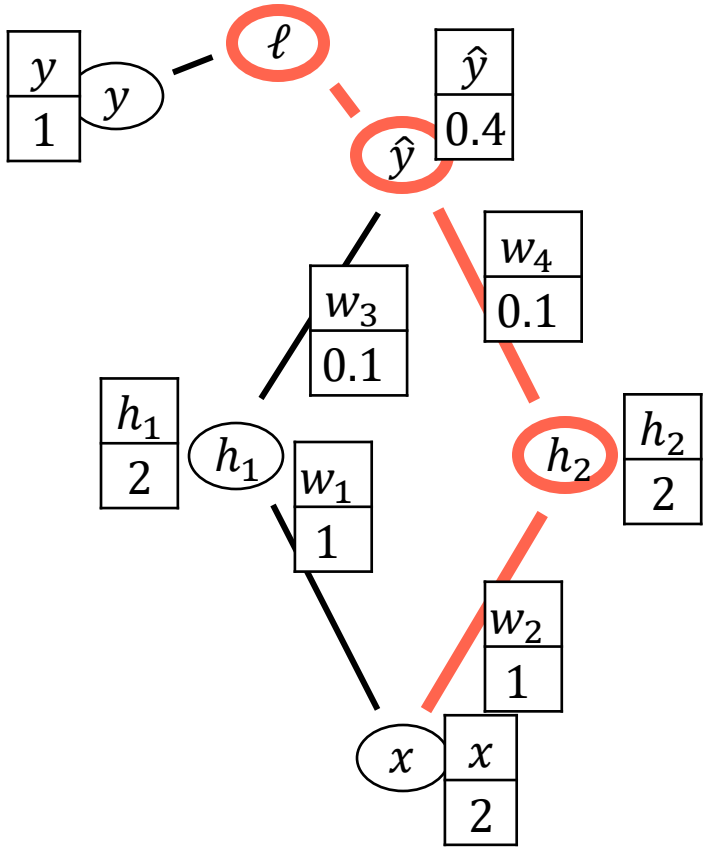
The Same Network, With Numbers



Compute: $\hat{y}$ on input $x = 2$

This is a “forward pass” through the network

Back to Our Simple Example



Use the squared loss: $\ell(y, \hat{y}) = (\hat{y} - y)^2$

Let's compute $\frac{\partial \ell}{\partial w_2}$

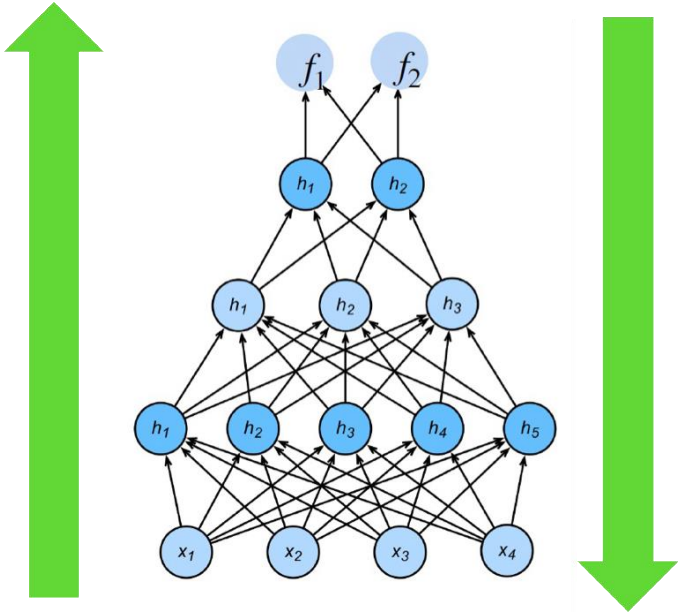
Orange lines: single path from w_2 to loss

Chain rule: $\frac{\partial \ell}{\partial w_2} = \frac{\partial \ell}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial h_2} \times \frac{\partial h_2}{\partial w_2}$

Backpropagation: An Efficient Algorithm for Gradients in Neural Networks

Forward pass:

Start with input layer, compute all hidden nodes and outputs layer-by-layer



Backward pass:

Start with output layer, compute all partial derivatives layer-by-layer

The Backpropagation Algorithm: Layer-by-Layer, Backwards

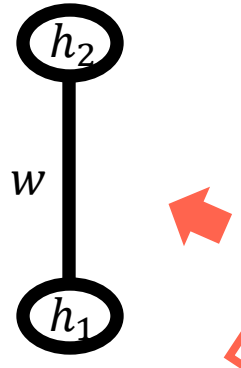
Want to find partial derivative for weight w in middle of network

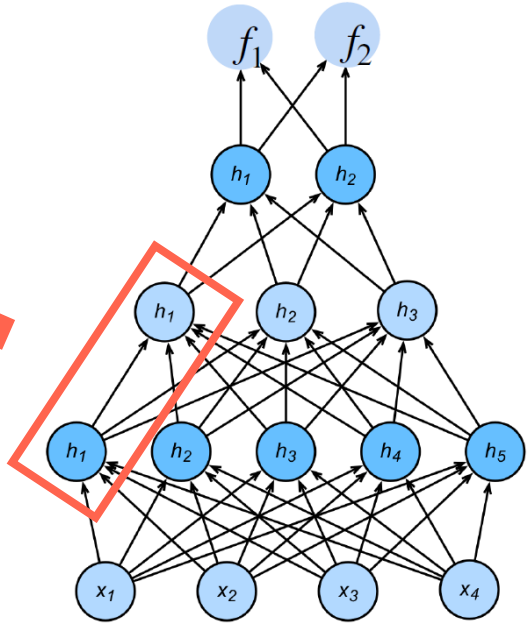
Connects from h_1 to h_2

Chain rule: $\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial h_2} \times \frac{\partial h_2}{\partial w}$

We already computed this

This is simple to compute





Gradients for Neural Networks

Compute the gradient of the loss ℓ w.r.t. $\mathbf{W}_t$

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \cdots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Multiplication of *many* matrices

Wikipedia

Two Issues for Deep Neural Networks

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}}$$

Gradient Exploding

$$1.5^{100} \approx 4 \times 10^{17}$$

Gradient Vanishing

$$0.8^{100} \approx 2 \times 10^{-10}$$

How good are the models?

Training Error and Generalization Error

Training error: model error on the training data

Generalization error: model error on new data

Example: practice a future exam with past exams

- Doing well on past exams (training error) doesn’t guarantee a good score on the future exam (generalization error)

Underfitting
Overfitting

Image credit: hackernoon.com

Model Capacity

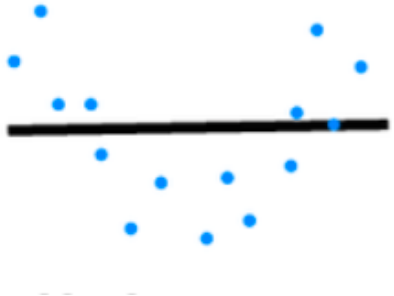
The ability to fit variety of functions

Low capacity models struggles to fit training set

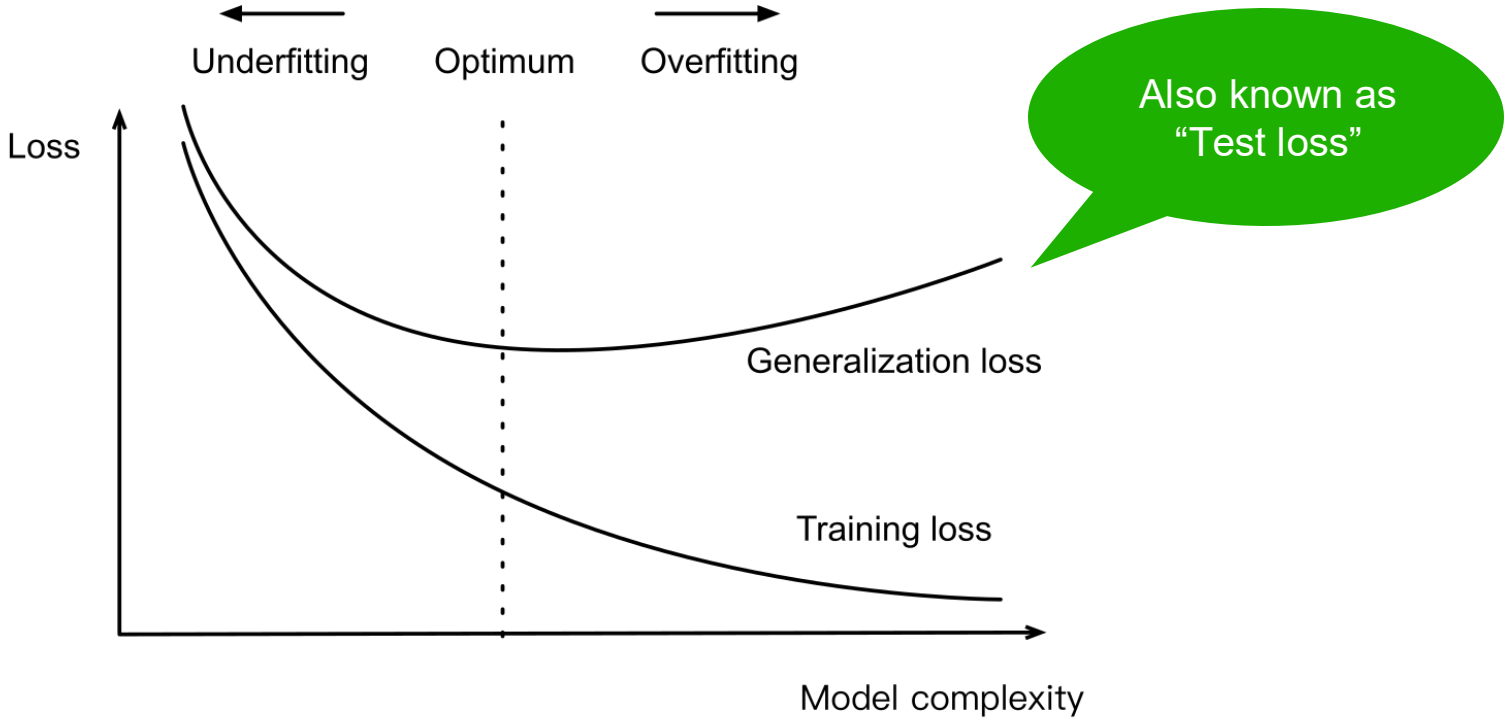
- Underfitting

High capacity models can memorize the training set

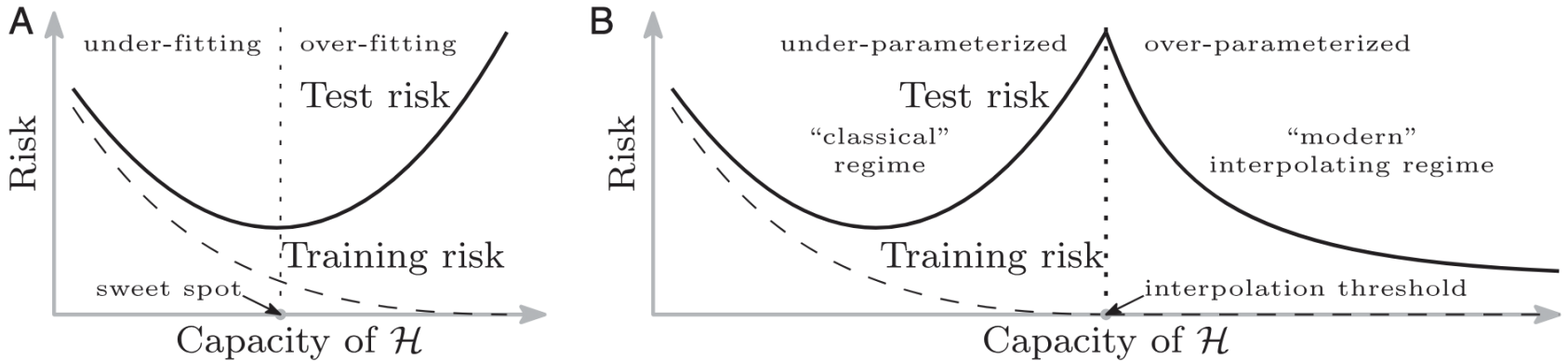
- Overfitting



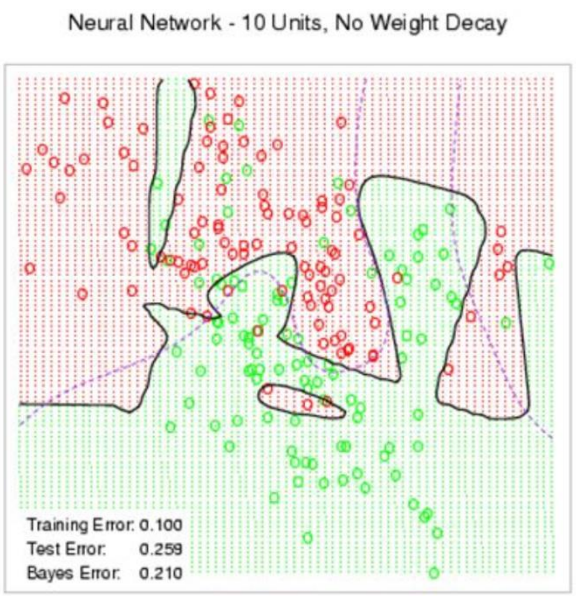
Classical View of Model Complexity

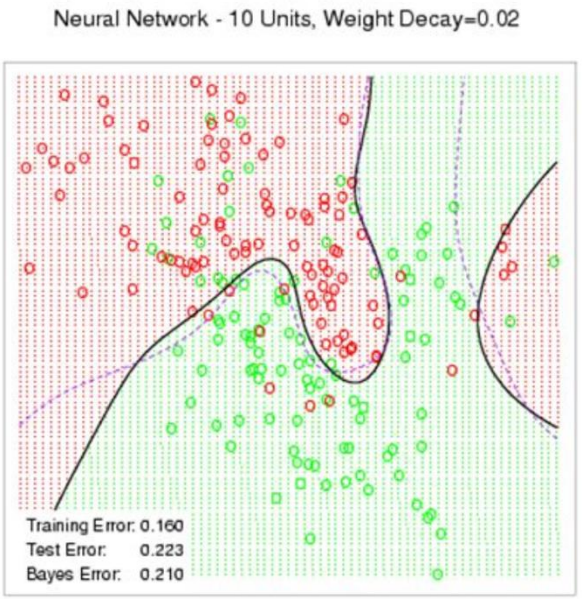


A Modern View: Double Descent



Weight Decay





Squared Norm Regularization as Soft Constraint

We can rewrite the hard constraint version as

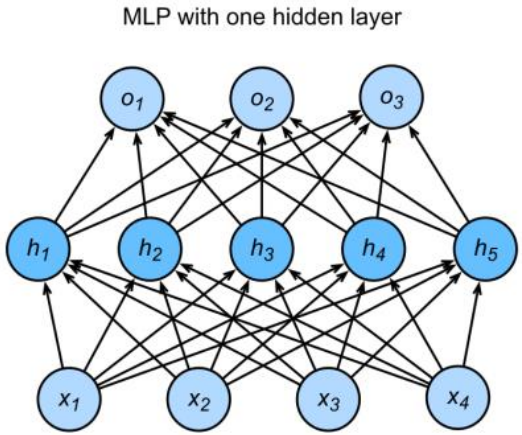
$$min L(\mathbf{w}, b) + \frac{\lambda}{2} \| \mathbf{w} \|^2$$

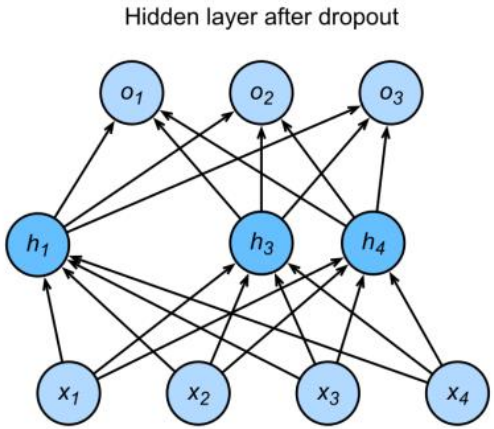
- Hyper-parameter λ controls regularization importance
- $\lambda = 0$: no effect
  $\lambda \to \infty, \mathbf{w}^* \to \mathbf{0}$

Apply Dropout

- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}' + \mathbf{b}^{(2)}$$
$$\mathbf{p} = \text{softmax}(\mathbf{o})$$





Thanks!