

CS 540 Introduction to Artificial Intelligence Midterm Review I

University of Wisconsin–Madison Fall 2025, Section 3 October 20, 2025

Announcements

- HW6 released Friday
- Classroom change this Friday, 10/22
 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



Foundations

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



Terminology:

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

Prior: estimate of the probability without evidence

• Terminology:

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

Likelihood: probability of evidence given a hypothesis

Terminology:

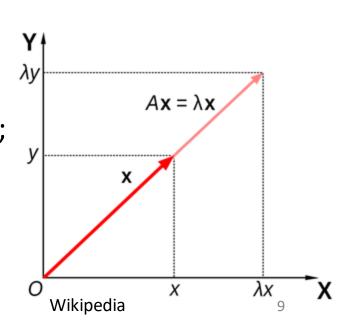
$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$
Posterior

• Posterior: probability of hypothesis given evidence.

Eigenvalues & Eigenvectors

- For a square matrix A, solutions to $Av=\lambda v$
 - $-\nu$ (nonzero) is a vector: **eigenvector**
 - $-\lambda$ is a scalar: **eigenvalue**

- Intuition: A is a linear transformation;
- Can stretch/rotate vectors;
- E-vectors: only stretched (by e-vals)



Dimensionality Reduction

Reduce dimensions

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



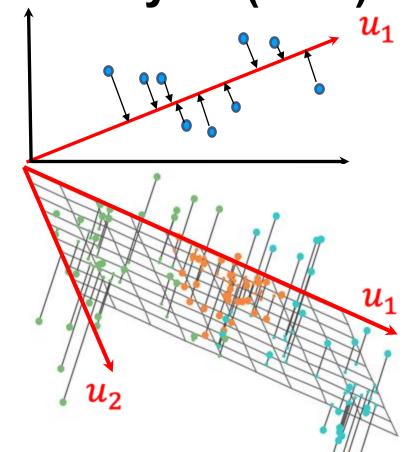
CreativeBloq

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

Principal Components Analysis (PCA)

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

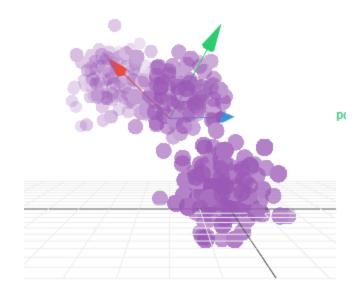
 These vectors are the principal components



PCA Procedure

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

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



Victor Powell

PCA Procedure

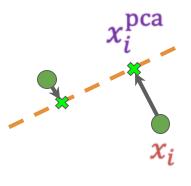
Output:

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

- Orthogonal
- Can show: they are top-m eigenvectors of

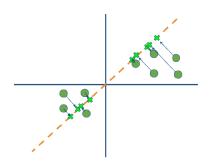
$$S = \frac{1}{n-1} \sum_{i=1}^{n} x_i x_i^{\mathsf{T}} \text{ (covariance matrix)}$$

- Each x_i projected to $x_i^{\text{pca}} = \sum_{j=1}^m (u_j^{\mathsf{T}} x_i) u_j$



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

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

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	$\neg P$	$P \wedge Q$	$P \lor Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
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 ⇒ 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 \text{ human}(x) \Rightarrow \text{mammal}(x)
```

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

First-Order Logic: Quantifiers

- Existential quantifier: 3
- Sentence is true for some value of x in the domain of variable x.
- Main connective typically is A
 - "some humans are male"

```
\exists x \text{ human}(x) \land \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

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

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:

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

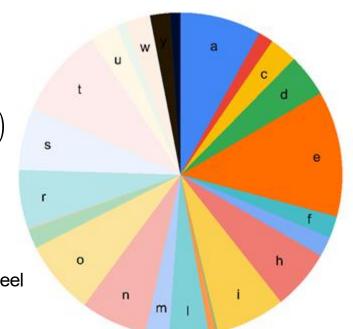
- Present doesn't depend on whole past
 - Just recent past, i.e., context.
 - What's **k=0?**

k=0: Unigram Model

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

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

The English letter frequency wheel

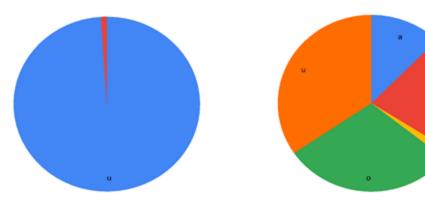


k=1: Bigram Model

Markov Assumption:

(Present depends on immediate past)

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



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:

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

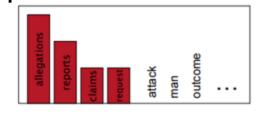
n-gram Training

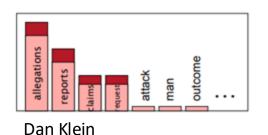
Issues:

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

- 1. Multiply tiny numbers?
 - **Solution**: use logs; add instead of multiply P(w|denied the)
- 2. n-grams with zero probability?
 - Solution: smoothing

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





Representing Words

Remember value of random variables (RVs) Easier to work with than objects like 'dog'

Traditional representation: one-hot vectors

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

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

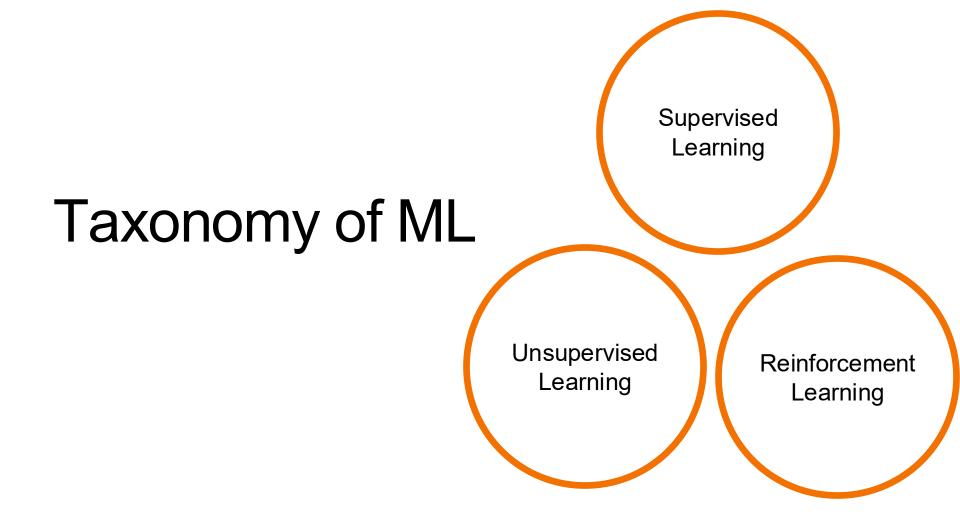
$$dog = \begin{bmatrix} 0.13 & 0.87 & -0.23 & 0.46 & 0.87 & -0.31 \end{bmatrix}^{T}$$

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

AKA word embeddings



Machine Learning Basics



Supervised/Unsupervised Learning

Supervised learning:

Make predictions, classify data, perform regression

Dataset: $(\mathbf{x}_{1}, y_{1}), (\mathbf{x}_{2}, y_{2}), \dots, (\mathbf{x}_{n}, y_{n})$

Features / Covariates / Input

Labels / Outputs

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







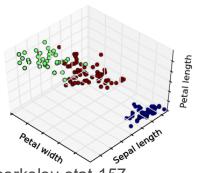
Supervised/Unsupervised Learning

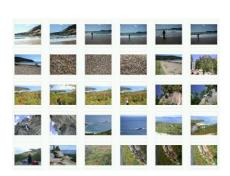
Unsupervised learning:

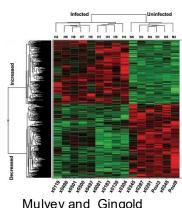
No labels; generally, won't be making predictions

Dataset: x_1, x_2, \dots, x_n

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



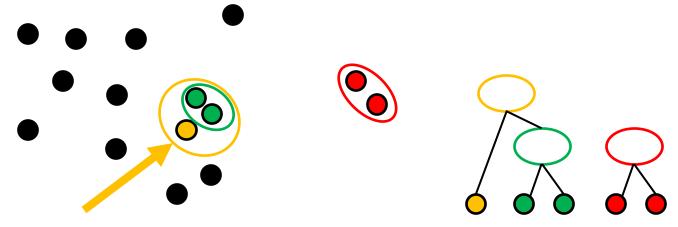




Mulvey and Gingold

Agglomerative Clustering Example

Repeat: Get pair of clusters that are closest and merge



Merging Criteria

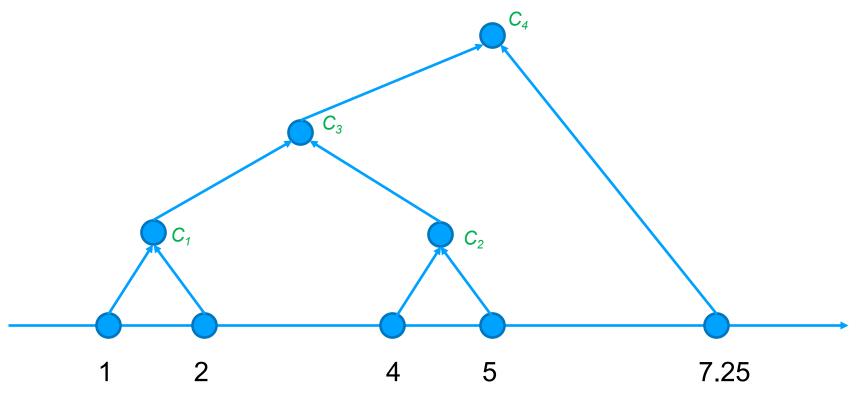
Merge: use closest clusters. Define closest? Single-linkage

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

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

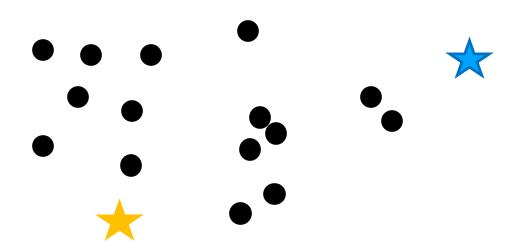
$$d(A,B) = \frac{1}{|A||B|} \sum_{x_1 \in A, x_2 \in B} d(x_1, x_2)$$

Single-linkage Example



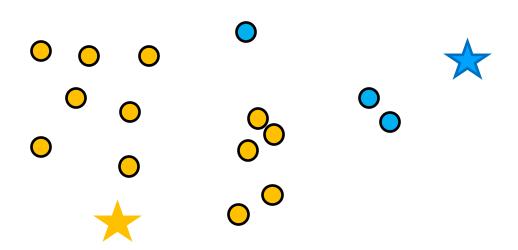
K-Means Clustering

Steps: 1. Randomly pick k cluster centers



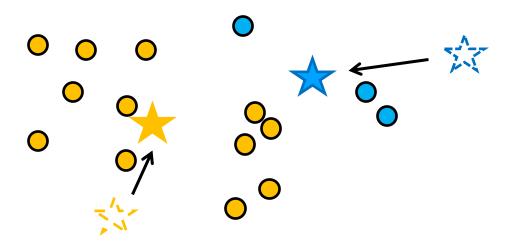
K-Means Clustering

2. Find closest center for each point



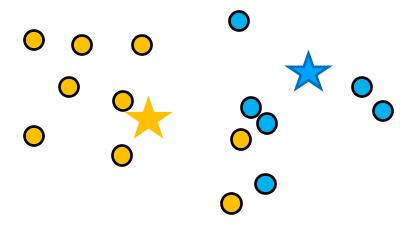
K-Means Clustering

3. Update cluster centers by computing centroids



K-Means Clustering

Repeat Steps 2 & 3 until convergence



K-means algorithm

Input: $x_1, x_2, ..., x_n, k$

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

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

$$y(x_i) = \operatorname{argmin}_i ||x_i - c_i||$$

Step 3: update all cluster centers as the centroids:

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

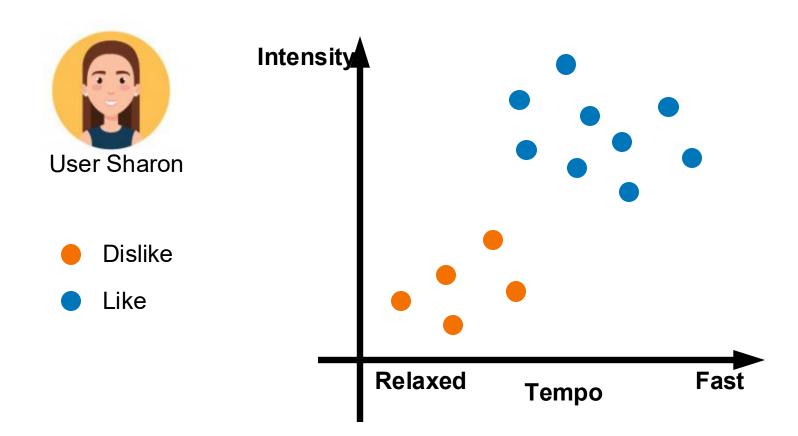
Repeat Step 2 and 3 until cluster centers no longer change

Two Types of Supervised Learning Algorithms

Classification

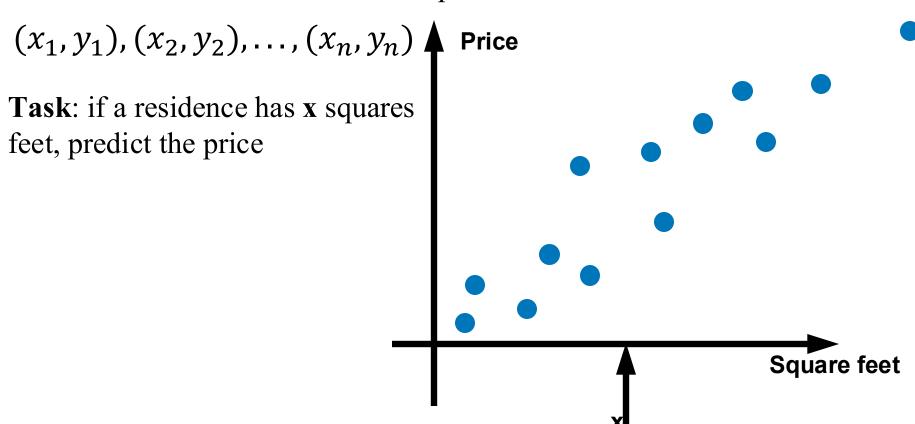
Regression

Example of Classification: Predict Song Preferences



Example of regression: housing price prediction

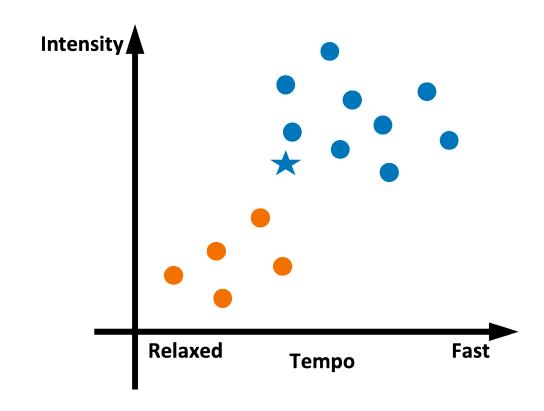
Given: a dataset that contains n samples



k-Nearest Neighbor Classifier



- Dislike
- Like



k-Nearest Neighbor Classifier

- Input: Training data $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \dots, (\mathbf{X}_n, y_n)$ Distance function $d(\mathbf{X}_i, \mathbf{X}_i)$; number of neighbors k; test data \mathbf{X}^*
- 1. Find the k training instances $\mathbf{X}_{i_1}, \dots, \mathbf{X}_{i_k}$ closest to \mathbf{X}^* under $d(\mathbf{X}_i, \mathbf{X}_i)$
- 2. Output y^* , the majority class of y_{i_1}, \ldots, y_{i_k} . Break ties randomly.

Basic Recipe for Supervised Learning

- Select model class
- 2. Select loss
- 3. Optimize parameters

Usually: apply a variant of gradient descent

4. Evaluate output

Usually: compute loss on test set

Maximum Likelihood Estimation

MLE solves

Find
$$\theta$$
 which is the best "explanation" for the data

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

Rewrite the problem in an equivalent form

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

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

Connecting MLE and Loss Minimization

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

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

Bayesian classification

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

$$\hat{y} = \underset{y}{\operatorname{arg\,max}} p(y | X_1, \dots, X_k)$$
 (Posterior)

(Prediction)

$$= \arg\max_{y} \frac{p(X_1, \dots, X_k | y) \cdot p(y)}{p(X_1, \dots, X_k)}$$
 (by Bayes' rule)

 $= \underset{y}{\operatorname{arg \, max}} \ p(X_1, \dots, X_k | y) \ p(y)$



Class prior

Class conditional likelihood

Naïve Bayes Assumption

Conditional independence of feature attributes

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.

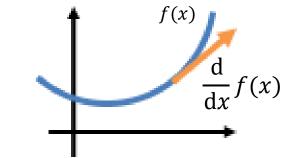
Congratulations! Your email address has been randomly selected as the GRAND PRIZE WINNER of **\$5,000,000 USD** in our International Lottery Promotion.

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
```

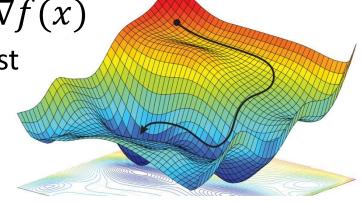
Gradients & Gradient Descent

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





- Direction where f grows fastest



M. Hutson

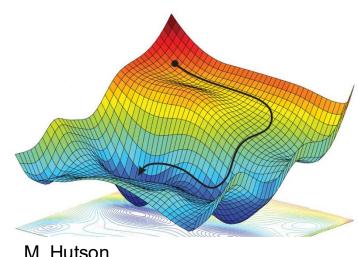
Gradients & Gradient Descent

 Gradient descent takes iterative steps to make loss function smaller

Gradient Descent

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

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

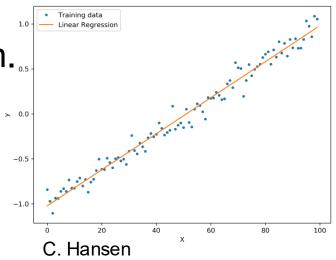


Linear Regression

Simplest type of regression problem...

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

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

Take train features and make it a n*(d+1) matrix, and y a vector: $\begin{bmatrix} x_1^T \end{bmatrix}$

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

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]



If $\theta^T x$ is really big, $\exp(-\theta^T x)$ is really small $\to p$ close to 1 If really negative exp is huge $\rightarrow p$ close to 0

> "Logistic Regression"



Neural Networks

Review: Perceptron

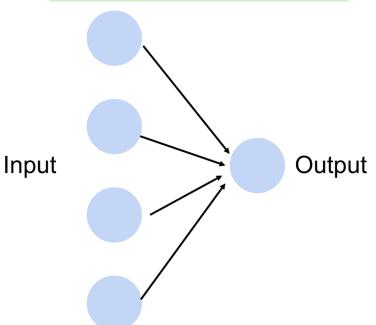
Given input x , weight w and bias b , perceptron outputs:

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

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

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

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

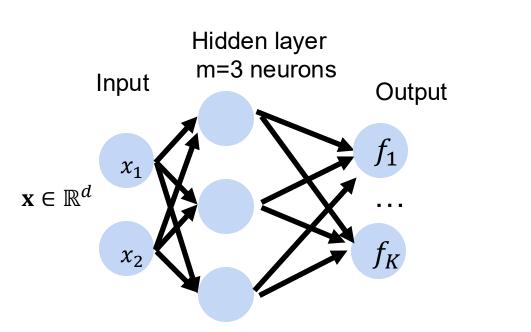
$$\mathbf{h}_3 = \sigma(\mathbf{W}^{(3)}\mathbf{h}_2 + \mathbf{b}^{(3)})$$

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

$$\mathbf{p} = \text{softmax}(\mathbf{f})$$
Hidden layer
$$\mathbf{h}_1 = \mathbf{f}_2$$
Hidden layer
$$\mathbf{h}_1 = \mathbf{f}_2$$
Hidden layer
$$\mathbf{f}_1 = \mathbf{f}_2$$
Hidden layer
$$\mathbf{f}_2 = \mathbf{f}_3$$
Hidden layer
$$\mathbf{f}_1 = \mathbf{f}_2$$
Hidden layer
$$\mathbf{f}_2 = \mathbf{f}_3$$
Hidden layer
$$\mathbf{f}_3 = \mathbf{f}_3$$
Hidden layer

Softmax

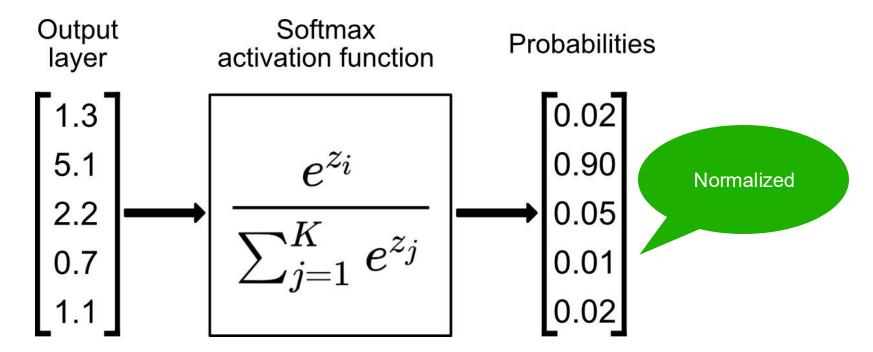
Turns outputs *f* into probabilities (sum up to 1 across K classes)



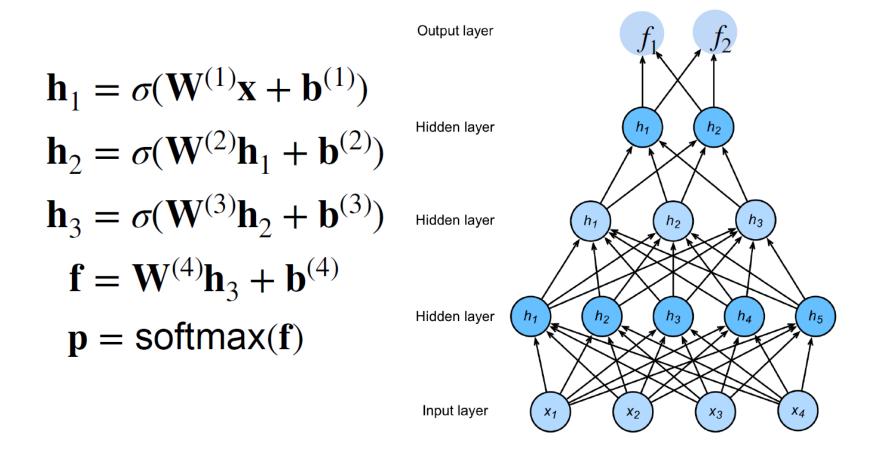
$$p(y|\mathbf{x}) = softmax(f)$$
$$= \frac{\exp(f_y(x))}{\sum_{k=1}^{K} \exp(f_k(x))}$$

Softmax

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



More complicated neural networks: multiple hidden layers



How to train a neural network? Binary classification

 $\mathbf{x} \in \mathbb{R}^d$ One training data point in the training set D

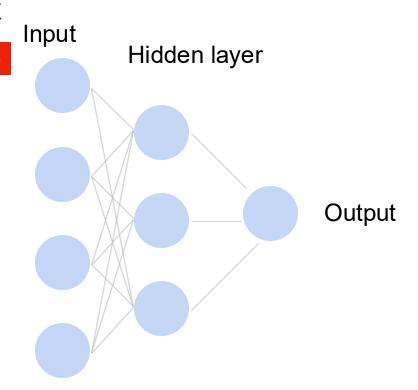
 $\hat{y} \in [0,1]$ Model output for example **x**

(This is a function of all weights $W: \hat{y} = g(W)$)

y Ground truth label for example x

Learning by matching the output to the label

We want $\hat{y} \to 1$ when y = 1, and $\hat{y} \to 0$ when y = 0



How to train a neural network? Binary classification

Loss function:
$$\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$$

Per-sample loss:

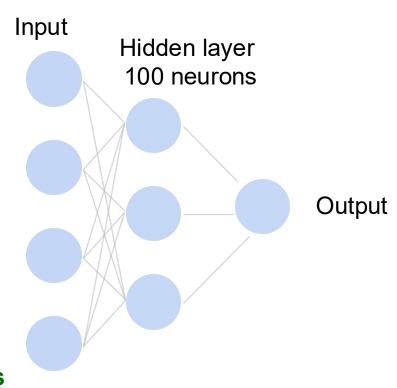
$$\ell(\mathbf{x}, y) = -y\log(\hat{y}) - (1 - y)\log(1 - \hat{y})$$



Negative log likelihood

Minimizing NLL is equivalent to Max Likelihood Learning (MLE)

Also known as binary cross-entropy loss



How to train a neural network? Multiclass

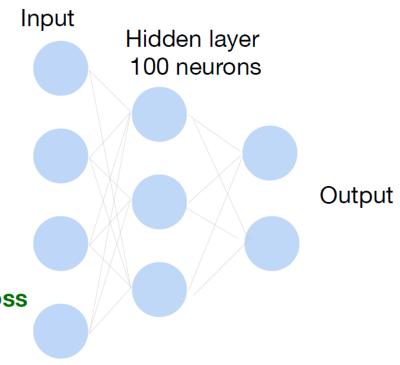
Loss function:
$$\frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} \ell(\mathbf{x}, y)$$

Per-sample loss:

$$\mathcal{E}(\mathbf{x}, y) = \sum_{k=1}^{K} -Y_k \log p_k = -\log p_y$$

where Y is one-hot encoding of y

Also known as cross-entropy loss or softmax loss



(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!

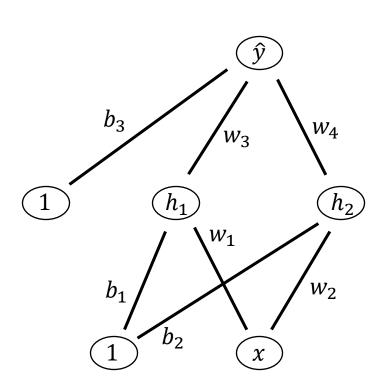
 On big datasets, better to update based on a few examples

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, ..., 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

A Simple Multilayer Perceptron



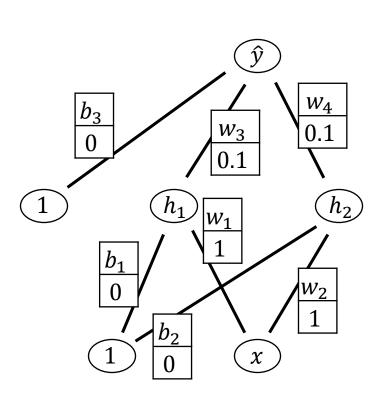
Hidden node: ReLU activations

$$h_1 = \max\{0, w_1x + b_1\}$$

Output node: no activation

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

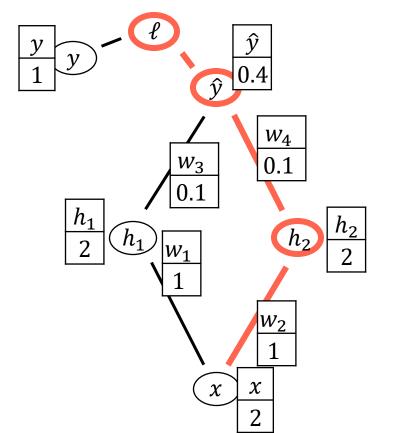
The Same Network, With Numbers



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 $\frac{\partial \ell}{\partial w_2}$

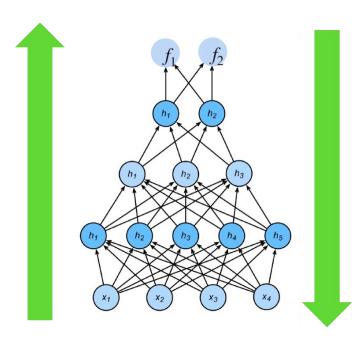
Orange lines: single path from w_2 to loss

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

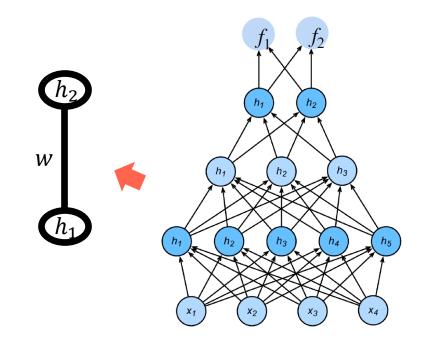
The Backpropagation Algorithm: Layer-by-Layer, Backwards

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

Connects from h_1 to h_2

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





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

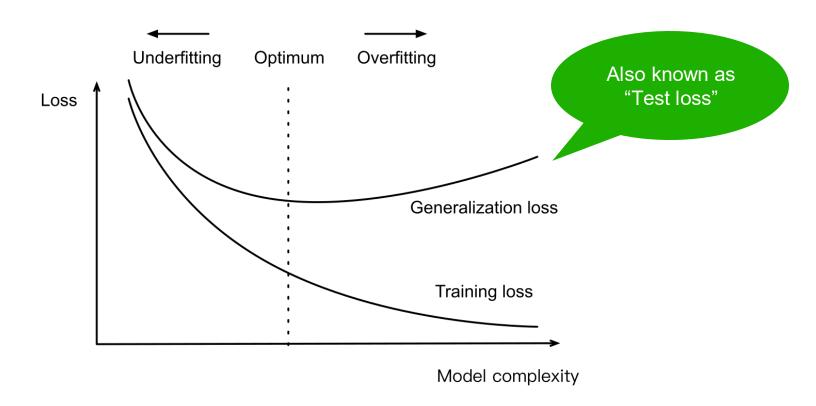
Underfitting

High capacity models can memorize the training set

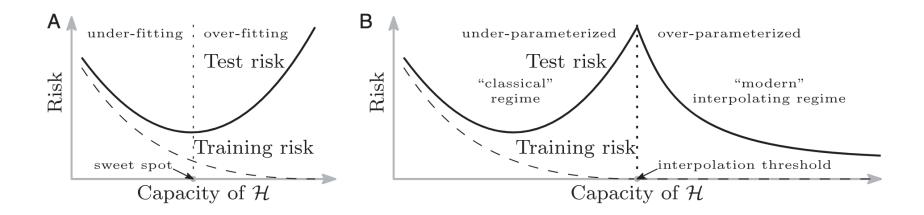
Overfitting



Classical View of Model Complexity

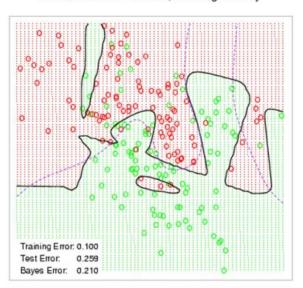


A Modern View: Double Descent

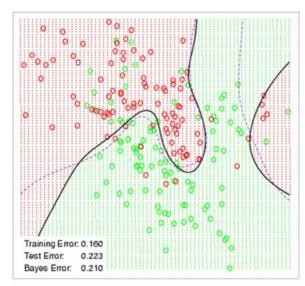


Neural Network - 10 Units, No Weight Decay

Weight Decay







Squared Norm Regularization as Soft Constraint

We can rewrite the hard constraint version as

$$minL(\mathbf{w},b) + \frac{\lambda}{2} \parallel \mathbf{w} \parallel^2$$

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

Apply Dropout

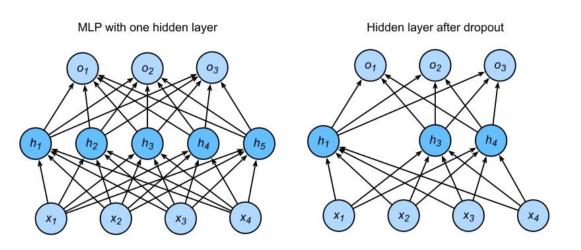
Often apply dropout on the output of hidden fully-connected layers

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

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}' + \mathbf{b}^{(2)}$$

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





Thanks!