# Information Retrieval

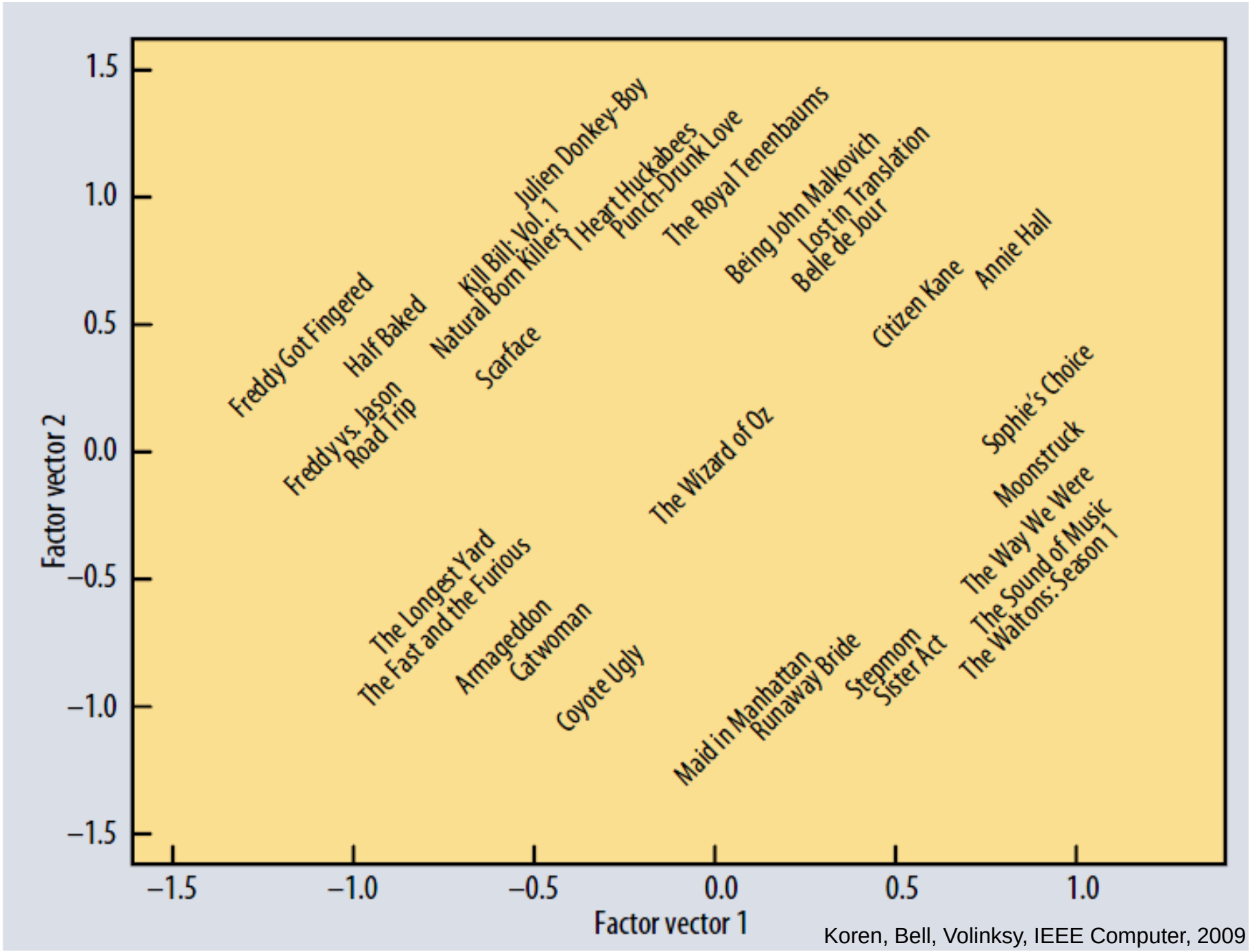
CS 547/DS 547

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

# Project Team

1. Bo Yu, Jin Yang, Kangjian Wu
2. Vignesh Sundaram, Amey More, Akanksha Pawar, Padmesh Naik
3. Xiaofan Zhou, Yiming Liu, Yuyuan Liu, Akhil Daphara
4. Sidhant jain, Parth dhruv, Darshan Swami, Aditya Ramesh
5. Chandra Rachabathuni, Ayush Shinde, Chao Wang, Anthony Chen, Adhiraj Budukh
6. Joe Scheufele, Alex Alvarez, Matt Suyer, Jason Dykstra
7. Adityavikram Gurao, Snehith Varma Datla, Sree Likhith Dasari, Supreeth Bandi
8. Samar, Bao, Michael, Megan

# Recommenders




Koren, Bell, Volinksy, IEEE Computer, 2009

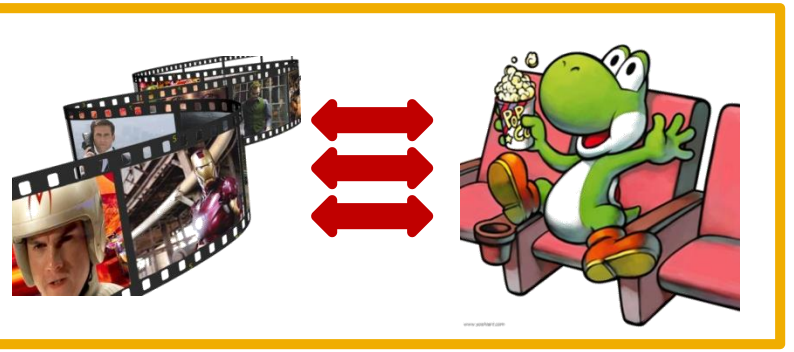
# Extending Latent Factor Model to Include Biases

# Modeling Biases and Interactions

user bias

movie bias

user-movie interaction



**Baseline predictor**

- Separates users and movies
- Benefits from insights into user's behavior
- Among the main practical contributions of the competition

**User-Movie interaction**

- Characterizes the matching between users and movies
- Attracts most research in the field
- Benefits from algorithmic and mathematical innovations

- $\mu$ = overall mean rating
- $b_x$ = bias of user $x$
- $b_i$ = bias of movie $i$

# Putting It All Together

$$r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$$

- $\mu$: Overall mean rating
- $b_x$: Bias for user ***x***
- $b_i$: Bias for movie ***i***
- $q_i \cdot p_x$: User-Movie interaction

- **Example:**
  - Mean rating: $\mu = 3.7$
  - You are a critical reviewer: your ratings are 1 star lower than the mean: $b_x = -1$
  - Star Wars gets a mean rating of *0.5* higher than average movie: $b_i = +0.5$
  - Predicted rating for you on Star Wars:
    $= 3.7 - 1 + 0.5 = 3.2$

# Fitting the New Model

- **Solve:**

$$\min_{Q,P} \sum_{(x,i)\in R} \left(r_{xi} - (\mu + b_x + b_i + q_i\, p_x)\right)^2$$

goodness of fit

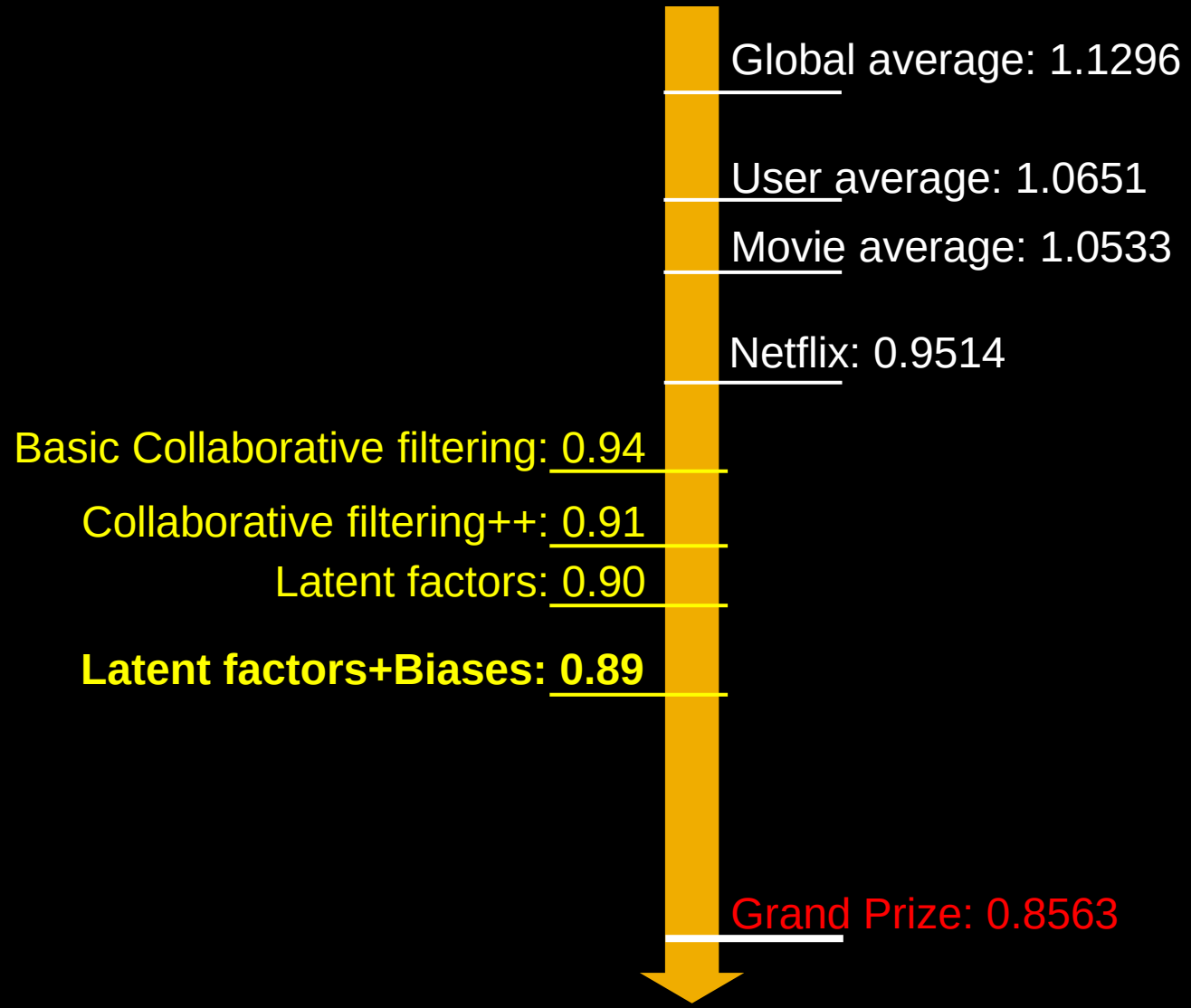
$$+\left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2\right)$$

regularization

λ is selected via grid-search on a validation set

- **Stochastic gradient decent to find parameters**
  - **Note:** Both biases $b_x$, $b_i$ as well as interactions $q_i$, $p_x$ are treated as parameters (and we learn them)

# Performance of Various Methods

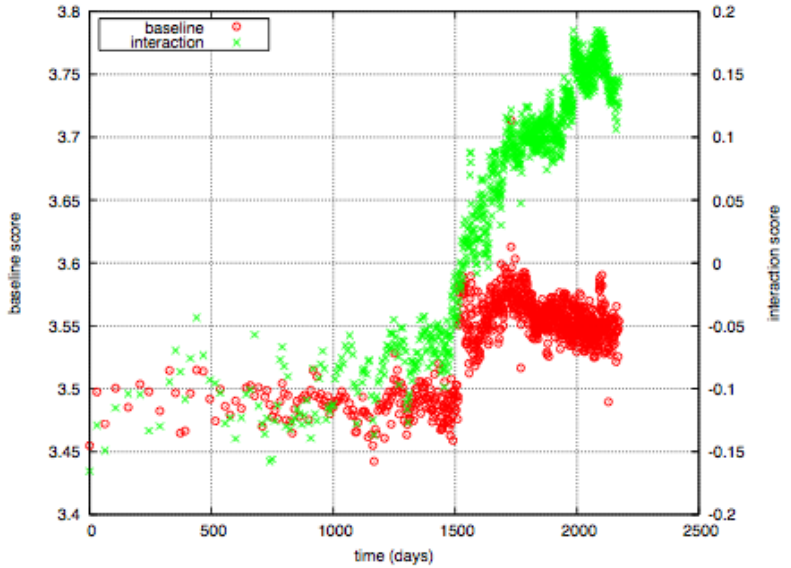


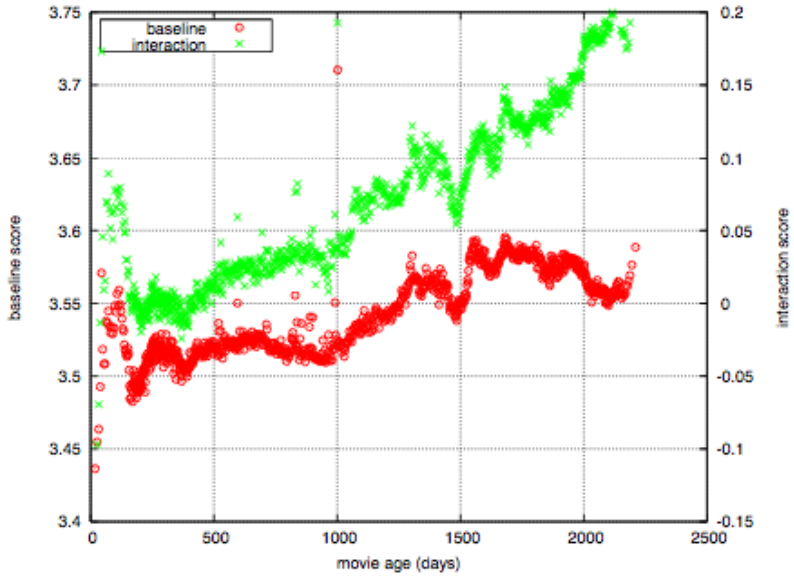
# The Netflix Challenge: 2006-09

# Temporal Biases Of Users

- **Sudden rise in the average movie rating** (early 2004)
  - Improvements in Netflix
  - GUI improvements
  - Meaning of rating changed
- **Movie age**
  - Users prefer new movies without any reasons
  - Older movies are just inherently better than newer ones

Y. Koren, Collaborative filtering with temporal dynamics, KDD '09



# Temporal Biases & Factors

- **Original model:**

  $r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$

- **Add time dependence to biases:**

  $r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x$

  - Make parameters $b_x$ and $b_i$ to depend on time
  - **(1)** Parameterize time-dependence by linear trends
    **(2)** Each bin corresponds to 10 consecutive weeks

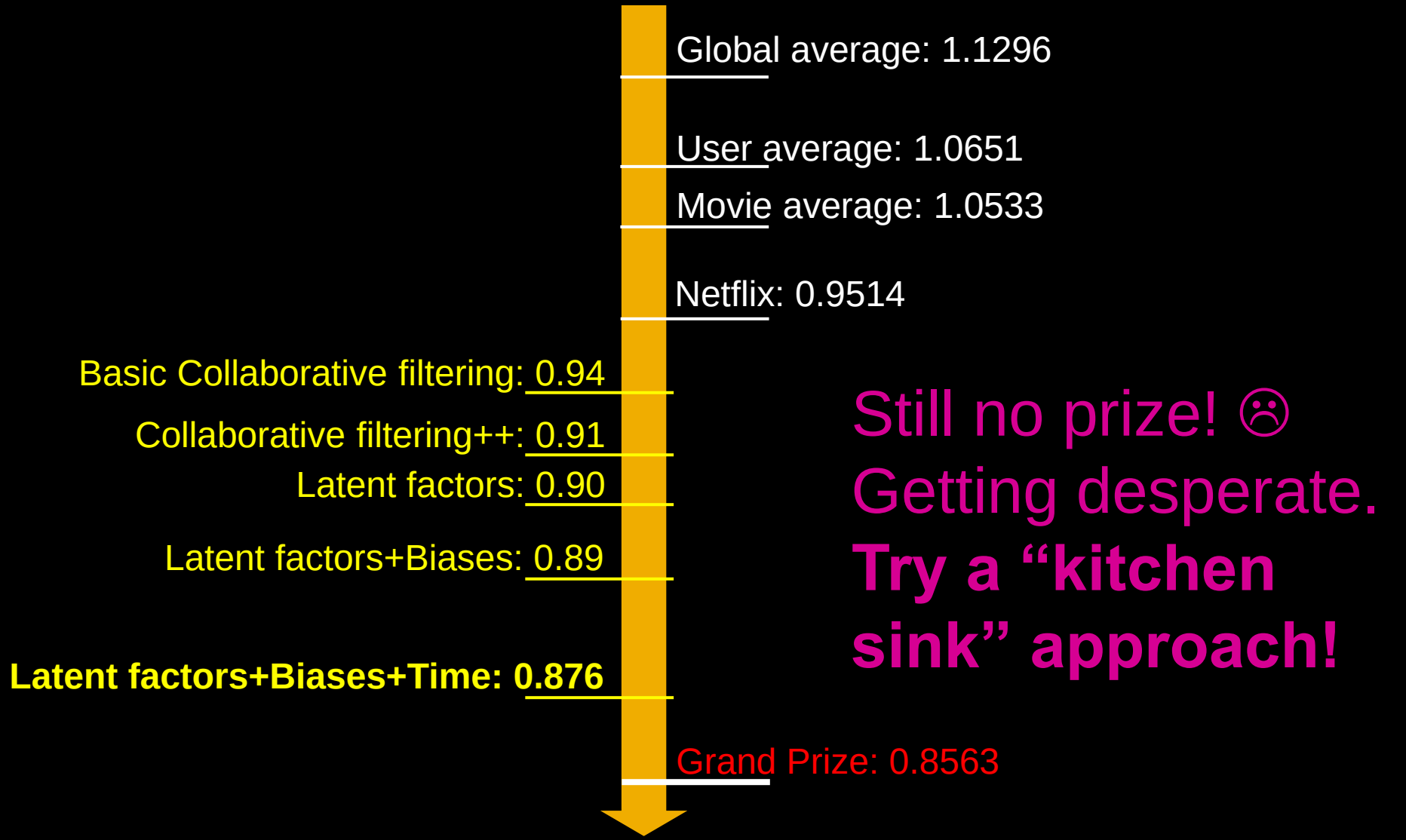
    $$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}$$

- **Add temporal dependence to factors**

  - $p_x(t)$… user preference vector on day $t$

Y. Koren, Collaborative filtering with temporal dynamics, KDD '09

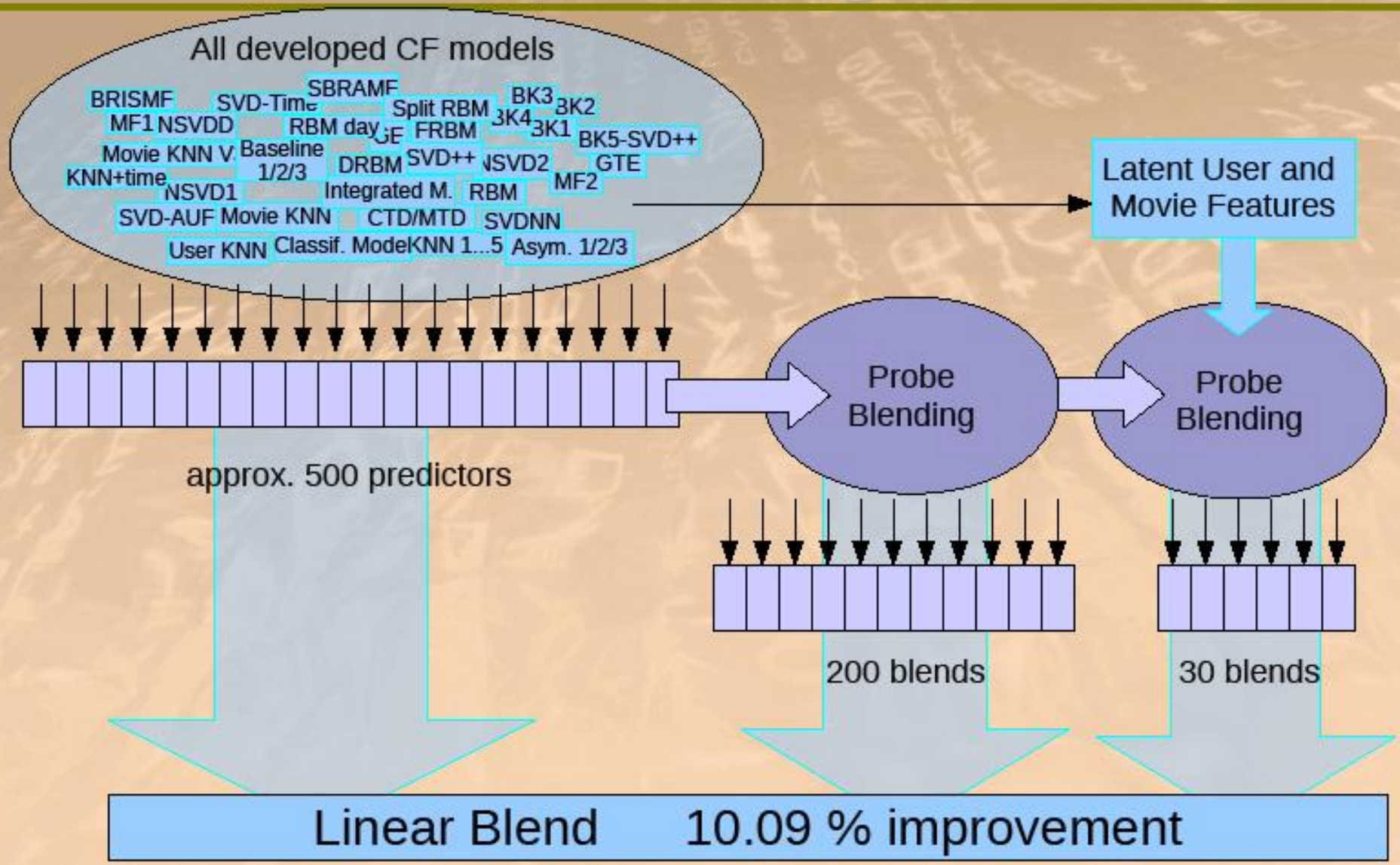
# Performance of Various Methods




Still no prize! ☹
Getting desperate.
**Try a "kitchen sink" approach!**

# The big picture

# Solution of BellKor's Pragmatic Chaos



# Standing on June 26th 2009

NETFLIX

## Netflix Prize

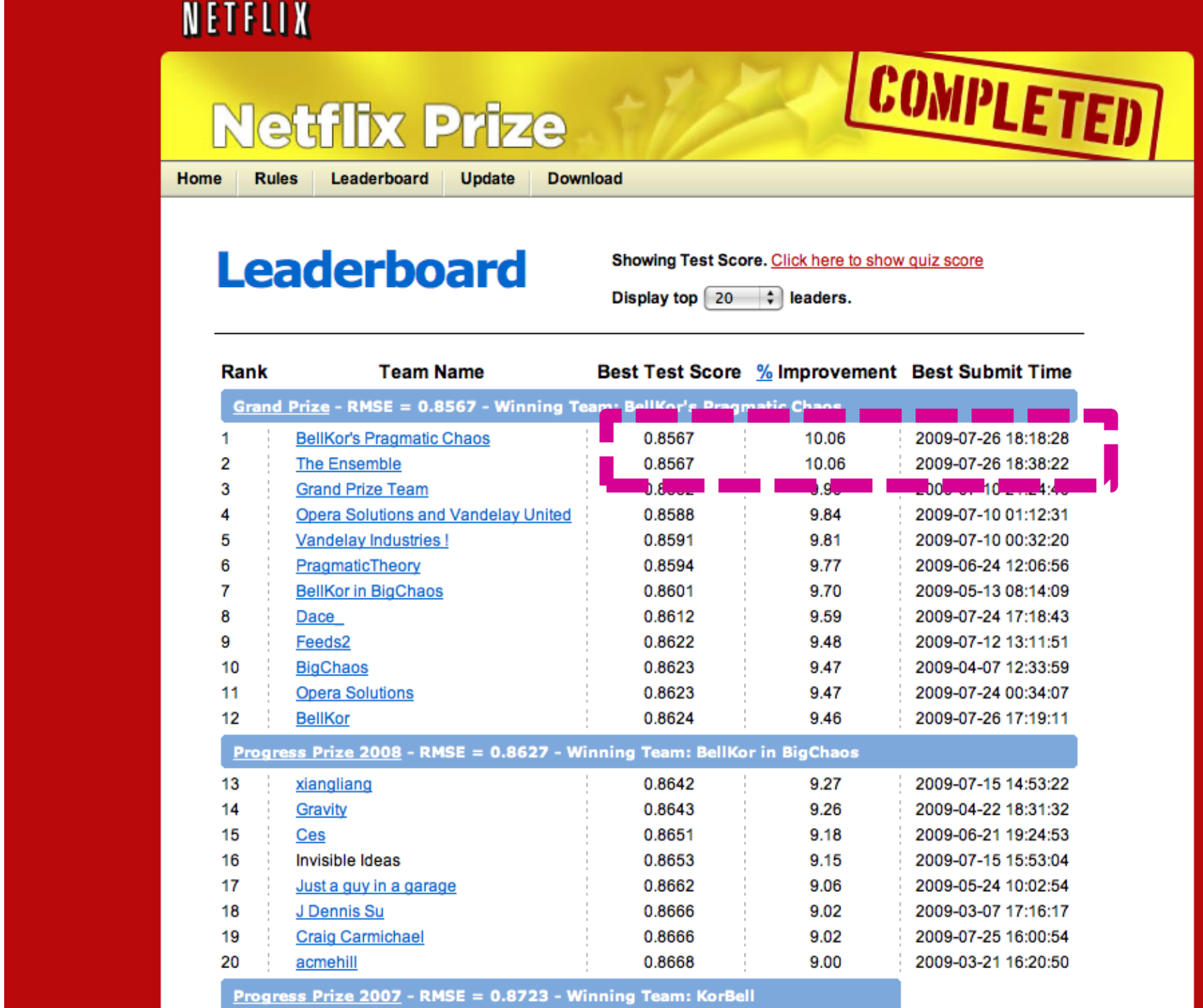
Home | Rules | Leaderboard | Register | Update | Submit | Download

## Leaderboard

Display top 20 leaders.

| Rank | Team Name | Best Score | % Improvement | Last Submit Time |
|---|---|---|---|---|
| 1 | BellKor's Pragmatic Chaos | 0.8558 | 10.05 | 2009-06-26 18:42:37 |
| Grand Prize - RMSE <= 0.8563 | | | | |
| 2 | PragmaticTheory | 0.8582 | 9.80 | 2009-06-25 22:15:51 |
| 3 | BellKor in BigChaos | 0.8590 | 9.71 | 2009-05-13 08:14:09 |
| 4 | Grand Prize Team | 0.8593 | 9.68 | 2009-06-12 08:20:24 |
| 5 | Dace | 0.8604 | 9.56 | 2009-04-22 05:57:03 |
| 6 | BigChaos | 0.8613 | 9.47 | 2009-06-23 23:06:52 |
| Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos | | | | |
| 7 | BellKor | 0.8620 | 9.40 | 2009-06-24 07:16:02 |
| 8 | Gravity | 0.8634 | 9.25 | 2009-04-22 18:31:32 |
| 9 | Opera Solutions | 0.8638 | 9.21 | 2009-06-26 23:18:13 |
| 10 | BruceDengDaoCiYiYou | 0.8638 | 9.21 | 2009-06-27 00:55:55 |
| 11 | pengpengzhou | 0.8638 | 9.21 | 2009-06-27 01:06:43 |
| 12 | xlvector | 0.8639 | 9.20 | 2009-06-26 13:49:04 |
| 13 | xiangliang | 0.8639 | 9.20 | 2009-06-26 07:47:34 |

**June 26th submission triggers 30-day "last call"**


NETFLIX
Netflix Prize
COMPLETED
Home
Rules
Leaderboard
Update
Download
Leaderboard
Showing Test Score. Click here to show quiz score
Display top 20 leaders.
Rank
Team Name
Best Test Score
% Improvement
Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos
1 BellKor's Pragmatic Chaos 0.8567 10.06 2009-07-26 18:18:28
2 The Ensemble 0.8567 10.06 2009-07-26 18:38:22
3 Grand Prize Team
4 Opera Solutions and Vandelay United 0.8588 9.84 2009-07-10 01:12:31
5 Vandelay Industries ! 0.8591 9.81 2009-07-10 00:32:20
6 PragmaticTheory 0.8594 9.77 2009-06-24 12:06:56
7 BellKor in BigChaos 0.8601 9.70 2009-05-13 08:14:09
8 Dace 0.8612 9.59 2009-07-24 17:18:43
9 Feeds2 0.8622 9.48 2009-07-12 13:11:51
10 BigChaos 0.8623 9.47 2009-04-07 12:33:59
11 Opera Solutions 0.8623 9.47 2009-07-24 00:34:07
12 BellKor 0.8624 9.46 2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos
13 xiangliang 0.8642 9.27 2009-07-15 14:53:22
14 Gravity 0.8643 9.26 2009-04-22 18:31:32
15 Ces 0.8651 9.18 2009-06-21 19:24:53
16 Invisible Ideas 0.8653 9.15 2009-07-15 15:53:04
17 Just a guy in a garage 0.8662 9.06 2009-05-24 10:02:54
18 J Dennis Su 0.8666 9.02 2009-03-07 17:16:17
19 Craig Carmichael 0.8666 9.02 2009-07-25 16:00:54
20 acmehill 0.8668 9.00 2009-03-21 16:20:50
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell

# Million $ Awarded Sept 21st 2009

# Acknowledgments


- Some slides and plots borrowed from
  Yehuda Koren, Robert Bell and Padhraic Smyth, Jure Leskovec


- **Further reading:**
  - Y. Koren, Collaborative filtering with temporal dynamics, KDD ’09
  - Matrix Factorization Techniques for Recommender Systems
  - How the Netflix Prize was won

# Today

- Text Classification: Definition and Overview
- Vector Space Classification
  - Rocchio
  - kNN
  - Naïve Bayes

90° Earthquuuuuuuuaaakesss!!!

the ONION
America's Finest News Source



# Grandmother Classifies 79% Of Everything A Shame

NEWS · Family · Local · ISSUE 47•47 ISSUE 45•29 · Jul 18, 2009

3.0K 382 20

SANDUSKY, OH—According to those close to Gertrude Wharton, the grandmother of nine declares 79 percent of everything she witnesses, experiences, or hears about from friends to be "a shame."

Wharton, 83, says it's a shame her husband isn't alive to see what a shame their grandchildren have become.

"No matter what happens, her response is always, 'That's a shame,'" said Wharton's son Kevin, 46. "From the recent passing of her friend Lillian to the fact that her coupon for chicken bouillon cubes expired last week, I can't have a conversation with her without being told something is a shame. Is this really how she sees the world now?"

Though Wharton, 83, has proclaimed things to be a shame in the past, her current usage of the word has been a growing cause for concern. Witnesses report that in the past 24 hours alone she has

# Standing queries

- The path from IR to text classification:
  - You have an information need to monitor, say:
    - presidential polls
  - You want to rerun an appropriate query periodically to find new news items on this topic
  - You will be sent new documents that are found
    - I.e., it's not ranking but classification (relevant vs. not relevant)
- Such queries are called **standing queries**
  - Long used by "information professionals"
  - A modern mass instantiation is **Google Alerts**
- Standing queries are (hand-written) text classifiers

http://www.google.com/alerts

# A text classification task: Email spam filtering

From: "" <takworlld@hotmail.com>

Subject: real estate is the only way... gem  oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the

methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

==========

Click Below to order:

http://www.wholesaledaily.com/sales/nmd.htm

==========

**How would you write a program that would automatically detect and delete this type of message?**

# Formal definition of Text Classification: Training

Given:

- A document space X
  - Documents are represented in this space – typically some type of high-dimensional space.
- A fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
  - The classes are human-defined for the needs of an application (e.g., relevant vs. nonrelevant).
- A training set D of labeled documents with each labeled document $\langle d, c \rangle \in X \times C$

Using a learning method or learning algorithm, we then wish to learn a classifier $\Upsilon$ that maps documents to classes:

$$\Upsilon : X \rightarrow C$$

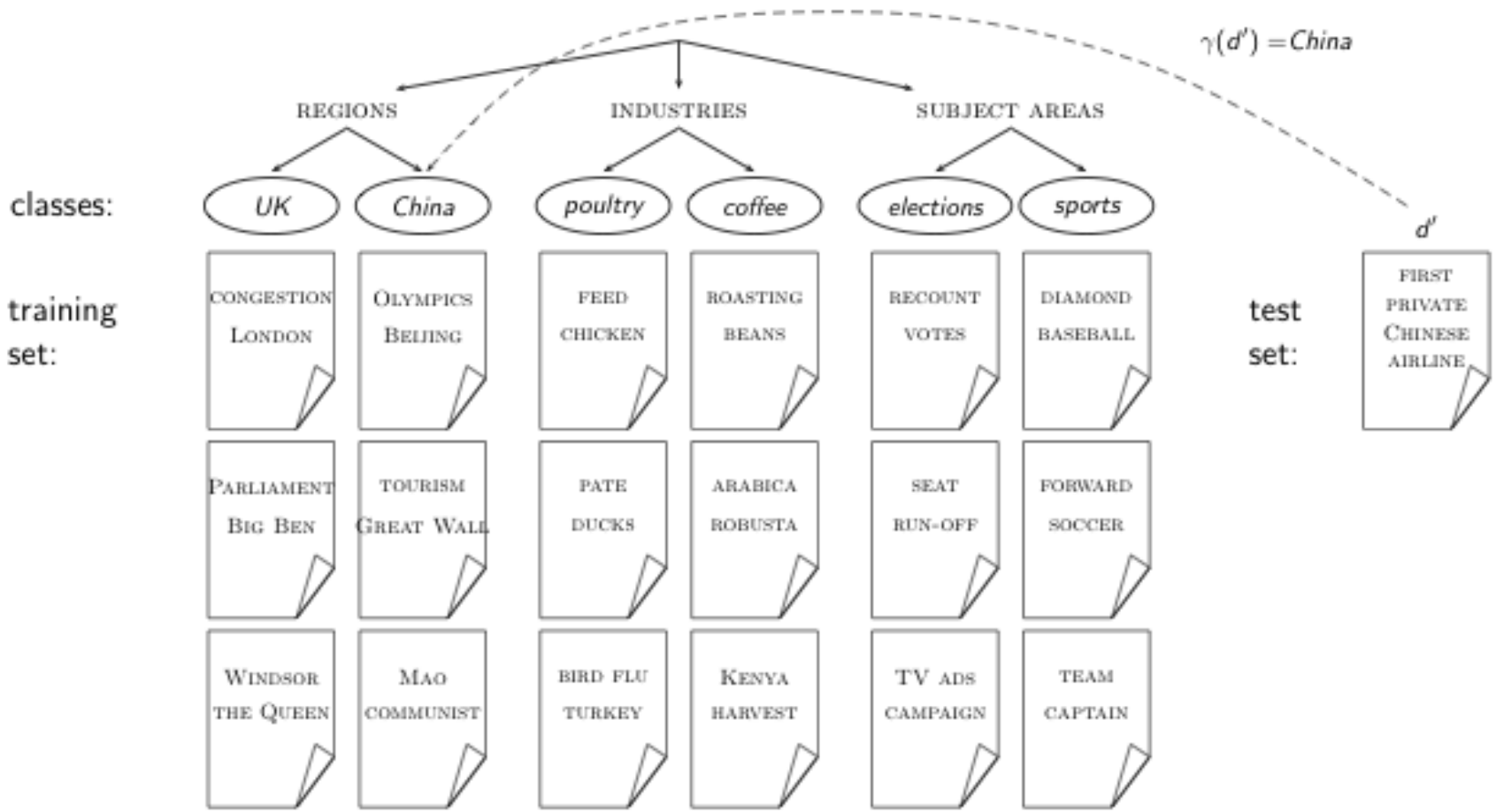
# Formal definition of Text Classification : Application/Testing

Given: a description $d \in X$ of a document

Determine: $\Upsilon(d) \in C$,

that is, the class that is most appropriate for $d$

# Topic classification



# Exercise

- Find examples of uses of text classification in information retrieval

# Examples of how search engines use classification

- Language identification (classes: English vs. French etc.)
- The automatic detection of spam pages (spam vs. nonspam)
- The automatic detection of sexually explicit content (sexually explicit vs. not)
- Topic-specific or *vertical* search – restrict search to a "vertical" like "related to health" (relevant to vertical vs. not)
- Standing queries (e.g., Google Alerts)
- Sentiment detection: is a movie or product review positive or negative (positive vs. negative)

How can we classify?

Any classification method?

# Classification in Machine Learning

- Text classification as a learning problem
- (i) Supervised learning of a the classification function $\Upsilon$ and

  (ii) its application to classifying new documents
- We will look at a couple of methods for doing this:Rocchio, kNN, Naive Bayes
- No free lunch: requires hand-classified training data
- But this manual classification can be done by non-experts.

# Vector Space Classification

# Recall: Vector Space Representation

- Each document is a vector, one component for each term.
- Terms are axes.
- High-dimensional vector space: 100,000s dimensions
- Normalize vectors (documents) to unit length.
- How can we do classification in this space?

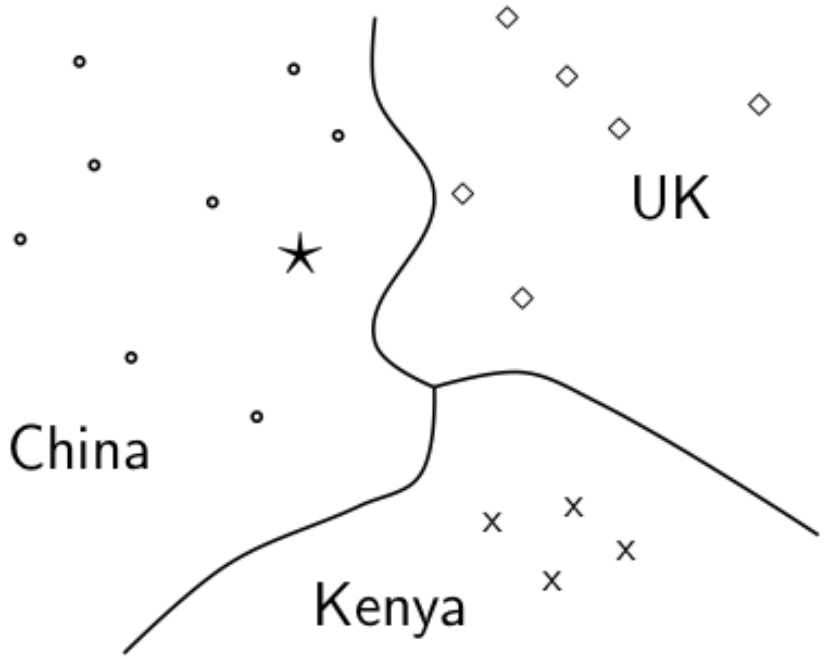
# Vector Space Classification

- As before, the training set is a set of documents, each labeled with its class (e.g., topic)
- In vector space classification, this set corresponds to a labeled set of points (or, equivalently, vectors) in the vector space

- Premise 1: Documents in the same class form a contiguous region
- Premise 2: Documents from different classes don't overlap.

- We define lines, surfaces, hypersurfaces to divide regions.

# Classes in the vector space




Should the document ⋆ be assigned to China, UK or Kenya?
Find separators between the classes
Based on these separators:
⋆ should be assigned to China
How do we find separators that do a good job at classifying new documents like ⋆? – Main topic of today

# Rocchio

# Rocchio classification: Basic idea

- Compute a centroid for each class
  - The centroid is the average of all documents in the class.
- Assign each test document to the class of its closest centroid.

# Recall definition of centroid

$$\vec{\mu}(c) = \frac{1}{|D_c|} \sum_{d \in D_c} \vec{v}(d)$$

where $D_c$ is the set of all documents that belong to class c and $\vec{v}(d)$ is the vector space representation of *d*.

# Rocchio algorithm

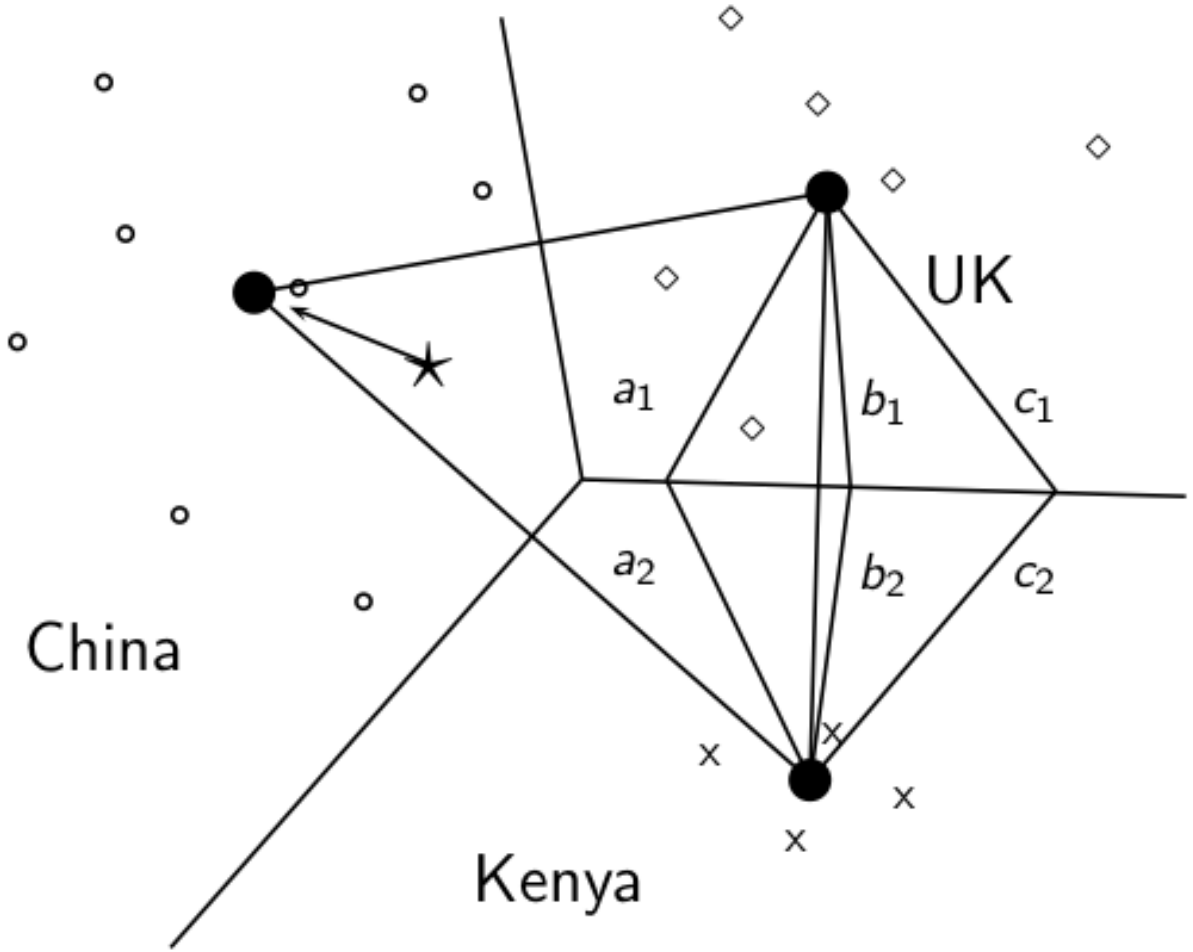
TRAINROCCHIO($\mathbb{C}, \mathbb{D}$)

1 **for each** $c_j \in \mathbb{C}$
2 **do** $D_j \leftarrow \{d : \langle d, c_j \rangle \in \mathbb{D}\}$
3 $\quad \vec{\mu}_j \leftarrow \frac{1}{|D_j|} \sum_{d \in D_j} \vec{v}(d)$
4 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}$

APPLYROCCHIO($\{\vec{\mu}_1, \ldots, \vec{\mu}_J\}, d$)

1 **return** $\arg\min_j |\vec{\mu}_j - \vec{v}(d)|$

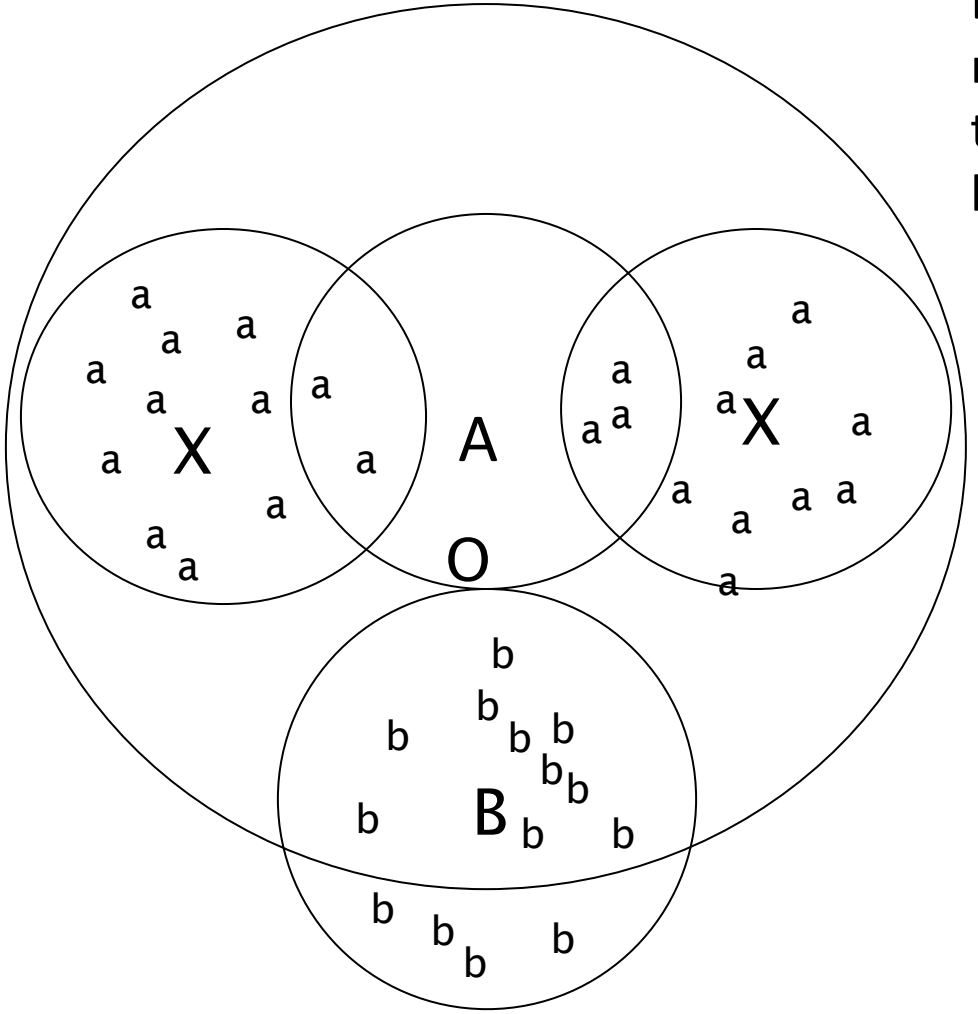
# Rocchio illustrated : a1 = a2, b1 = b2, c1 = c2



# Rocchio properties

- Rocchio forms a simple representation for each class: the centroid
  - We can interpret the centroid as the prototype of the class.
- Classification is based on similarity to / distance from centroid/prototype.
- However, does not guarantee that classifications are consistent with the training data!

# Rocchio cannot handle nonconvex, multimodal classes




Exercise: Why is Rocchio not expected to do well for the classification task a vs. b here?

- A is centroid of the a's, B is centroid of the b's.
- The point o is closer to A than to B.
- But o is a better fit for the b class.
- A is a multimodal class with two prototypes.
- But in Rocchio we only have one prototype.

# kNN (k Nearest Neighbors)

# kNN classification

- kNN classification is another vector space classification method.
- It also is very simple and easy to implement.
- kNN is more accurate (in most cases) than Naive Bayes and Rocchio.
- If you need to get a pretty accurate classifier up and running in a short time . . .
- . . . and you don’t care about efficiency that much . . .
- . . . use kNN.

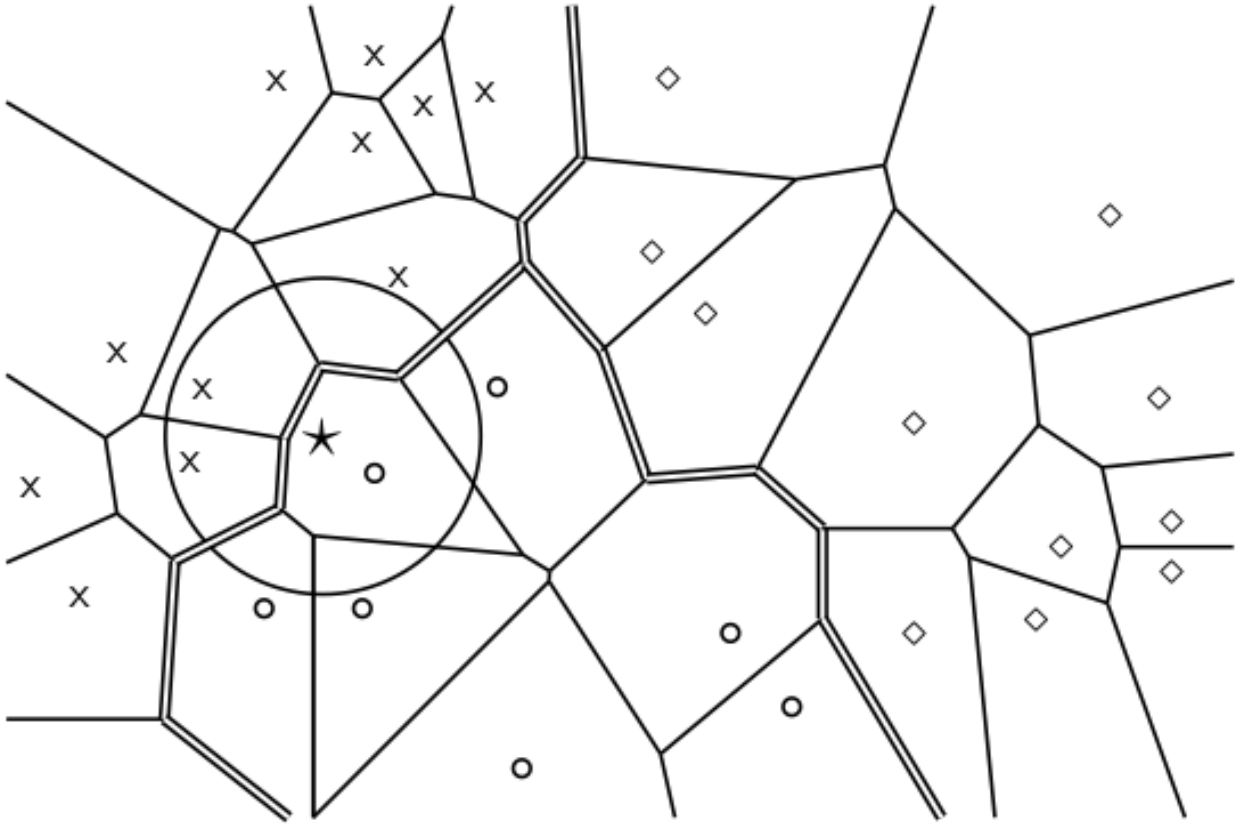
# kNN classification

- kNN = *k* nearest neighbors
- kNN classification rule for *k* = 1 (1NN): Assign each test document to the class of its nearest neighbor in the training set.
- 1NN is not very robust – one document can be mislabeled or atypical.
- kNN classification rule for *k* > 1 (kNN): Assign each test document to the majority class of its *k* nearest neighbors in the training set.
- Rationale of kNN: contiguity hypothesis
  - We expect a test document d to have the same label as the training documents located in the local region surrounding *d*.

# Probabilistic kNN

- Probabilistic version of kNN: $P(c|d)$ = fraction of $k$ neighbors of $d$ that are in $c$
- kNN classification rule for probabilistic kNN: Assign d to class $c$ with highest $P(c|d)$

# Probabilistic kNN



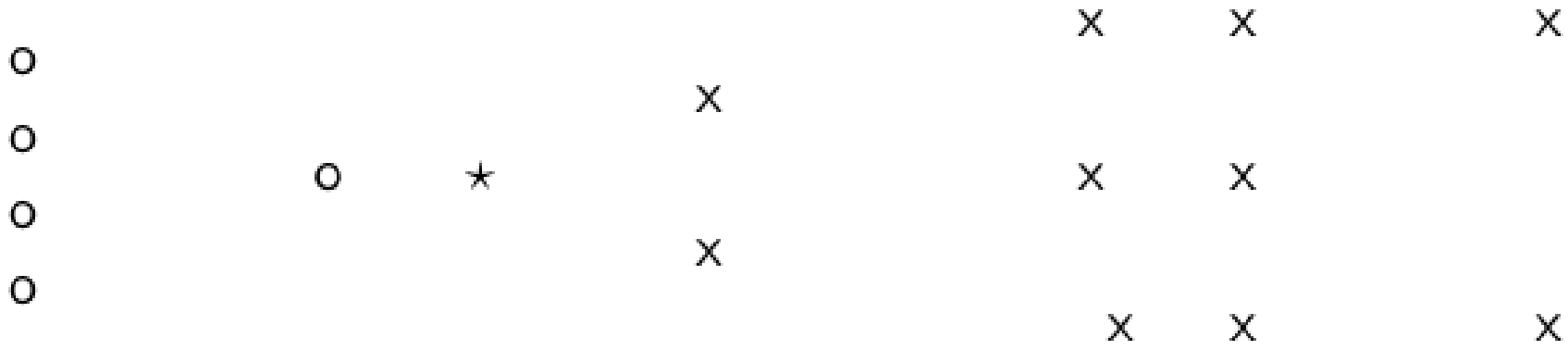
1NN, 3NN
classification
decision
for star?

# kNN algorithm

TRAIN-KNN($\mathbb{C}, \mathbb{D}$)
1 $\mathbb{D}' \leftarrow$ PREPROCESS($\mathbb{D}$)
2 $k \leftarrow$ SELECT-K($\mathbb{C}, \mathbb{D}'$)
3 **return** $\mathbb{D}', k$

APPLY-KNN($\mathbb{D}', k, d$)
1 $S_k \leftarrow$ COMPUTENEARESTNEIGHBORS($\mathbb{D}', k, d$)
2 **for each** $c_j \in \mathbb{C}(\mathbb{D}')$
3 **do** $p_j \leftarrow |S_k \cap c_j|/k$
4 **return** $\arg\max_j p_j$

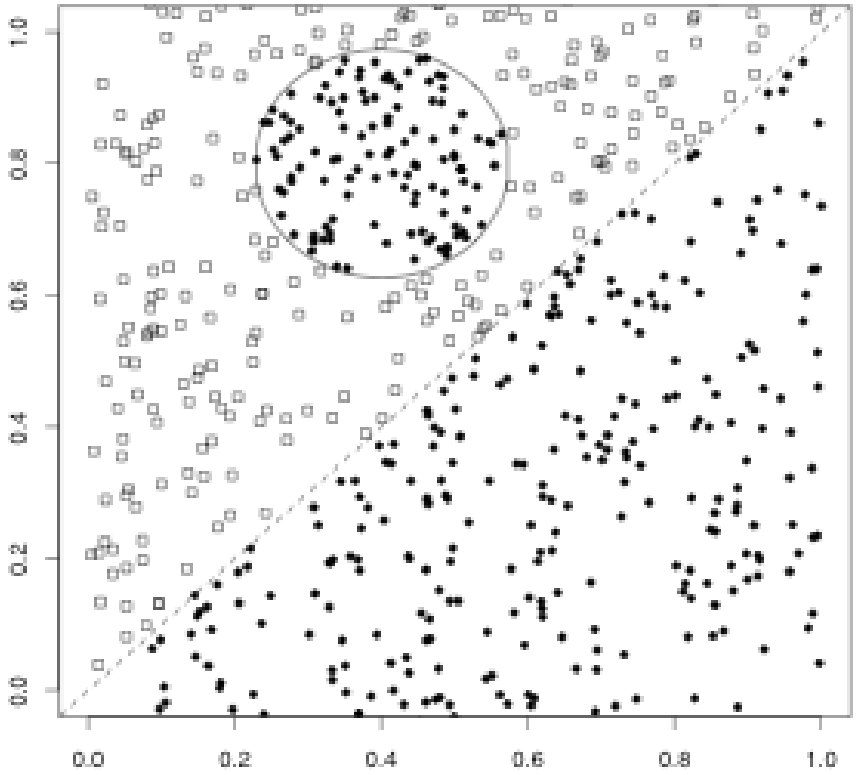
# Exercise



How is star classified by:

(i) 1-NN (ii) 3-NN (iii) 9-NN (iv) 15-NN (v) Rocchio?

# A nonlinear problem (Rocchio vs kNN)




- Linear classifier like Rocchio does badly on this task.
- kNN will do well (assuming enough training data)

# kNN: Discussion

- No training necessary
  - But linear preprocessing of documents is as expensive as training Naive Bayes.
  - We always preprocess the training set, so in reality training time of kNN is linear.
- kNN is very accurate if training set is large.

- But kNN can be very inaccurate if training set is small.

# Naive Bayes Classifier

# The Naive Bayes Classifier

- The Naive Bayes classifier is a probabilistic classifier
- We compute the probability of a document *d* being in a class *c* as follows:

Prior

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

Posterior

- $P(c)$ is the prior probability of $c$.
- $n_d$ is the length of the document. (number of tokens)
- $P(t_k|c)$ is the conditional probability of term $t_k$ occurring in a document of class $c$
- $P(t_k|c)$ as a measure of how much evidence $t_k$ contributes that $c$ is the correct class.
- If a document's terms do not provide clear evidence for one class vs. another, we choose the $c$ with highest $P(c)$ probability.

# Maximum a posteriori class

- Our goal in Naive Bayes classification is to find the "best" class.
- The best class is the most likely or maximum a posteriori (MAP) class $c_{map}$:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$

# Taking the log

- Multiplying lots of small probabilities can result in floating point underflow.
- Since log(xy) = log(x) + log(y), we can sum log probabilities instead of multiplying probabilities.
- Since log is a monotonic function, the class with the highest score does not change.
- So what we usually compute in practice is:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k|c)]$$

# Naive Bayes classifier

- Classification rule:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)]$$

- Simple interpretation:
  - Each conditional parameter log $\hat{P}(t_k|c)$ is a weight that indicates how good an indicator $t_k$ is for c.
  - The prior log $\hat{P}(c)$ is a weight that indicates the relative frequency of c.
  - The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
  - We select the class with the most evidence.

# Parameter estimation take 1: Maximum likelihood

- Estimate parameters $\hat{P}(c)$ and $\hat{P}(t_k|c)$ from train data: How?
- Prior:

$$\hat{P}(c) = \frac{N_c}{N}$$

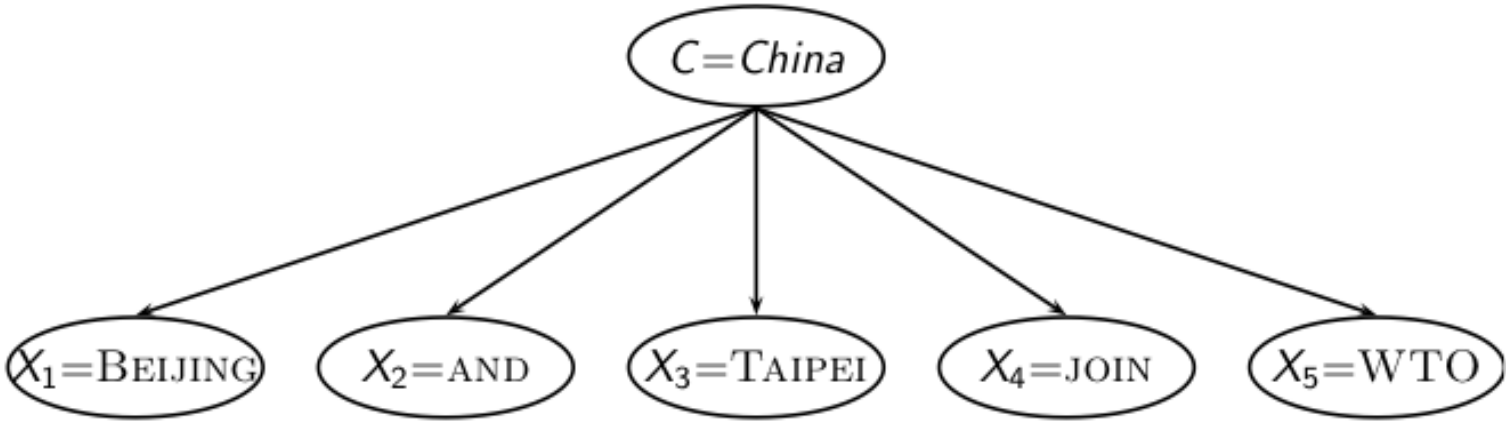
- $N_c$ : number of docs in class *c*; *N*: total number of docs
- Conditional probabilities:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- $T_{ct}$ is the number of tokens of *t* in training documents from class *c* (includes multiple occurrences)
- We've made a Naive Bayes independence assumption here:

$$\hat{P}(t_{k_1}|c) = \hat{P}(t_{k_2}|c)$$

# The problem with maximum likelihood estimates: Zeros




$P(China|d) \propto P(China) \cdot P(\texttt{BEIJING}|China) \cdot P(\texttt{AND}|China) \cdot P(\texttt{TAIPEI}|China) \cdot P(\texttt{JOIN}|China) \cdot P(\text{WTO}|China)$

$$\hat{P}(\text{WTO}|China) = \frac{T_{China,\text{WTO}}}{\sum_{t' \in V} T_{China,t'}} = \frac{0}{\sum_{t' \in V} T_{China,t'}} = 0$$

# The problem with maximum likelihood estimates: Zeros (cont)

- If there were no occurrences of WTO in documents in class China, we'd get a zero estimate:

$$\hat{P}(\text{WTO}|\textit{China}) = \frac{T_{\textit{China},\text{WTO}}}{\sum_{t' \in V} T_{\textit{China},t'}} = 0$$

- → We will get P(China|d) = 0 for any document that contains WTO!
- Zero probabilities cannot be conditioned away.

# To avoid zeros: Add-one smoothing

- Before:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- Now: Add one to each count to avoid zeros:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

- B is the number of different words (in this case the size of the vocabulary: $|V| = M$)

# To avoid zeros: Add-one smoothing

- Estimate parameters from the training corpus using add-one smoothing
- For a new document, for each class, compute sum of (i) log of prior and (ii) logs of conditional probabilities of the terms
- Assign the document to the class with the largest score

# Naive Bayes: Training

TrainMultinomialNB($\mathbb{C}, \mathbb{D}$)

1 $V \leftarrow$ ExtractVocabulary($\mathbb{D}$)
2 $N \leftarrow$ CountDocs($\mathbb{D}$)
3 **for each** $c \in \mathbb{C}$
4 **do** $N_c \leftarrow$ CountDocsInClass($\mathbb{D}, c$)
5 $prior[c] \leftarrow N_c/N$
6 $text_c \leftarrow$ ConcatenateTextOfAllDocsInClass($\mathbb{D}, c$)
7 **for each** $t \in V$
8 **do** $T_{ct} \leftarrow$ CountTokensOfTerm($text_c, t$)
9 **for each** $t \in V$
10 **do** $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
11 **return** $V, prior, condprob$

# Naive Bayes: Testing

```
APPLYMULTINOMIALNB(ℂ, V, prior, condprob, d)
1  W ← EXTRACTTOKENSFROMDOC(V, d)
2  for each c ∈ ℂ
3  do score[c] ← log prior[c]
4     for each t ∈ W
5     do score[c] += log condprob[t][c]
6  return arg max_{c∈ℂ} score[c]
```

# Example

| | docID | words in document | in *c* = *China*? |
|---|---|---|---|
| training set | 1 | Chinese Beijing Chinese | yes |
| | 2 | Chinese Chinese Shanghai | yes |
| | 3 | Chinese Macao | yes |
| | 4 | Tokyo Japan Chinese | no |
| test set | 5 | Japan Chinese Chinese Chinese Tokyo | ? |

- What do we need?
  - Class priors: P(c), P(not c)
  - Conditional probabilities: P(t|c), P(t|not c)

# Example

| | docID | words in document | in $c$ = *China*? |
|---|---|---|---|
| training set | 1 | Chinese Beijing Chinese | yes |
| | 2 | Chinese Chinese Shanghai | yes |
| | 3 | Chinese Macao | yes |
| | 4 | Tokyo Japan Chinese | no |
| test set | 5 | Japan Chinese Chinese Chinese Tokyo | ? |

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)]$$

$$\hat{P}(c) = \boxed{\phantom{00}} \text{ and } \hat{P}(\overline{c}) = \boxed{\phantom{00}}$$

$$\begin{aligned} \hat{P}(\text{Chinese}|c) &= \\ \hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= \\ \hat{P}(\text{Chinese}|\overline{c}) &= \\ \hat{P}(\text{Tokyo}|\overline{c}) = \hat{P}(\text{Japan}|\overline{c}) &= \end{aligned} \quad \boxed{\phantom{00}}$$

# Example

| | docID | words in document | in $c$ = *China*? |
|---|---|---|---|
| training set | 1 | Chinese Beijing Chinese | yes |
| | 2 | Chinese Chinese Shanghai | yes |
| | 3 | Chinese Macao | yes |
| | 4 | Tokyo Japan Chinese | no |
| test set | 5 | Japan Chinese Chinese Chinese Tokyo | ? |

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)]$$

$$\hat{P}(c) = 3/4 \text{ and } \hat{P}(\overline{c}) = 1/4$$

$$\begin{aligned}
\hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\text{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\text{Tokyo}|\overline{c}) = \hat{P}(\text{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

# Example

| | docID | words in document | in $c$ = *China*? |
|---|---|---|---|
| training set | 1 | Chinese Beijing Chinese | yes |
| | 2 | Chinese Chinese Shanghai | yes |
| | 3 | Chinese Macao | yes |
| | 4 | Tokyo Japan Chinese | no |
| test set | 5 | Japan Chinese Chinese Chinese Tokyo | ? |

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)]$$

$$\hat{P}(c|d) \propto \log 3/4 + \log 1/14 + 3\log 3/7 + \log 1/14 \approx -3.52$$

$$\hat{P}(\overline{c}|d) \propto \log 1/4 + \log 2/9 + 3\log 2/9 + \log 2/9 \approx -3.86$$

Thus, the classifier assigns the test document to $c$ = *China*.

# Example for Rocchio Classification

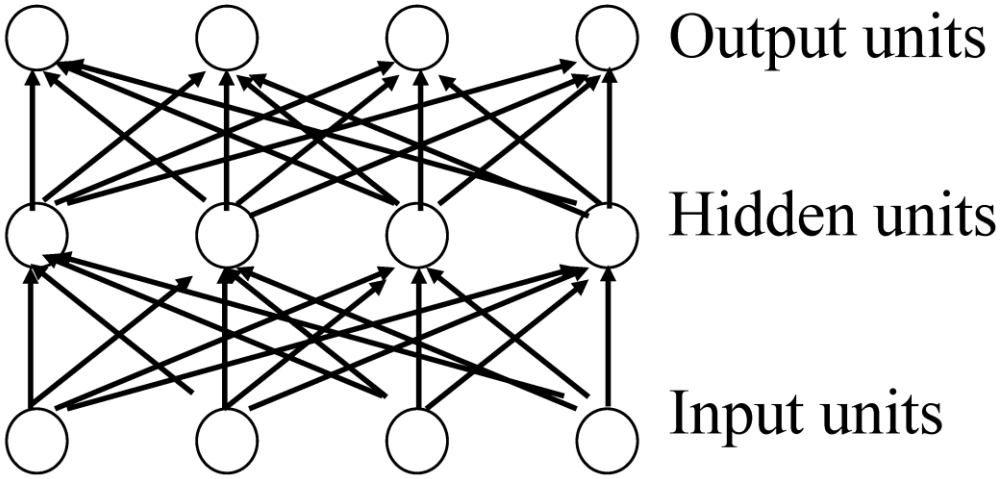
| | docID | words in document | in $c$ = *China*? |
|---|---|---|---|
| training set | 1 | Chinese Beijing Chinese | yes |
| | 2 | Chinese Chinese Shanghai | yes |
| | 3 | Chinese Macao | yes |
| | 4 | Tokyo Japan Chinese | no |
| test set | 5 | Japan Chinese Chinese Chinese Tokyo | ? |

# Other Classification Methods

- Neural Network
- Deep Learning (e.g., recurrent neural network, transformer)

# Neural Network

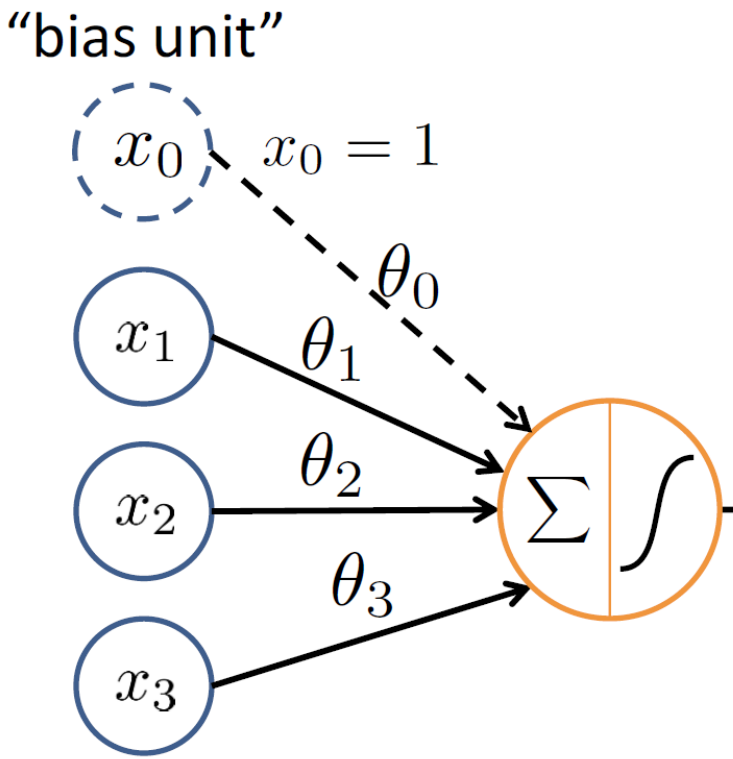
# Neural Networks



Layered feed-forward network

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

# Neuron Model: Logistic Unit

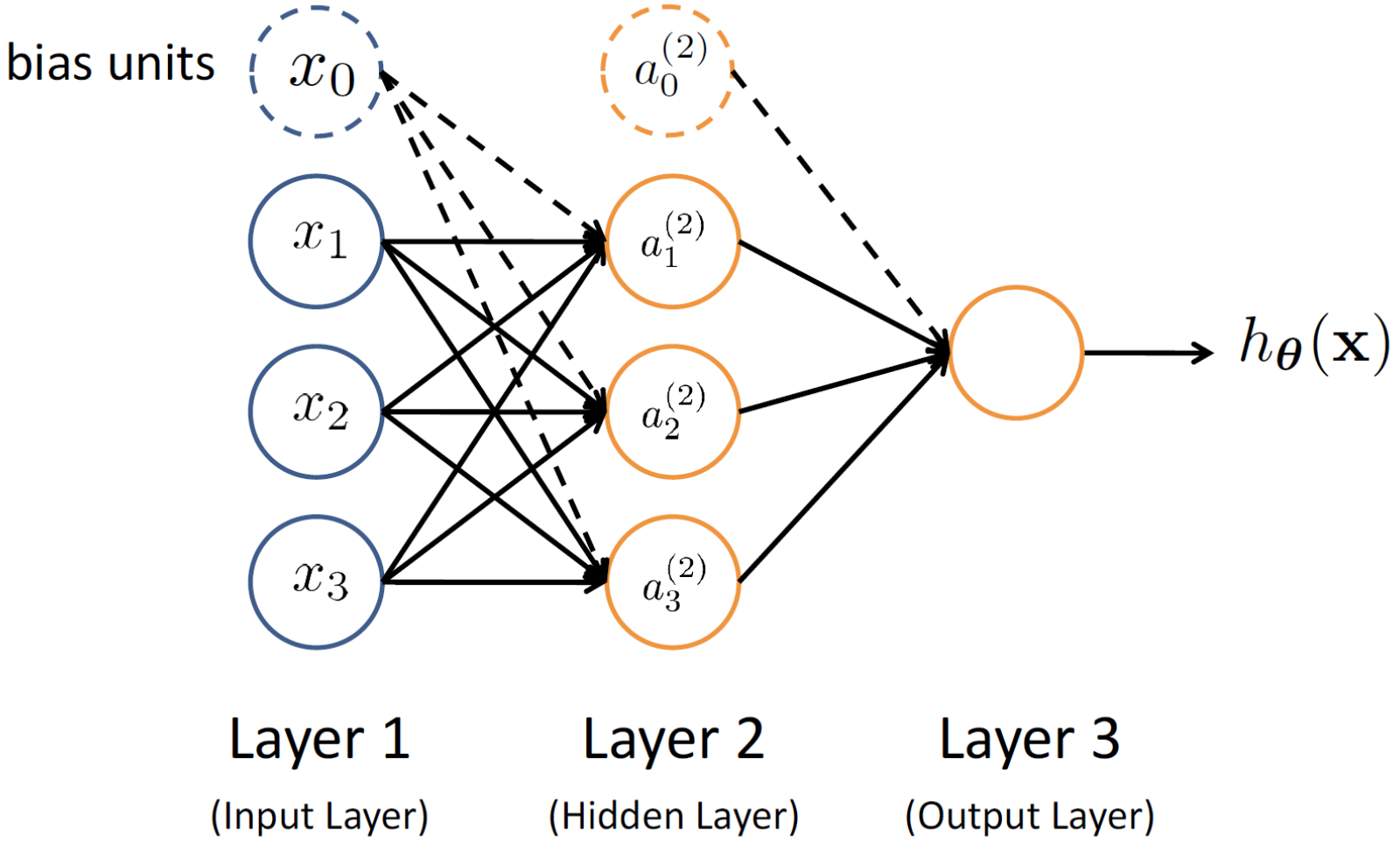



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right)$$

$$= \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

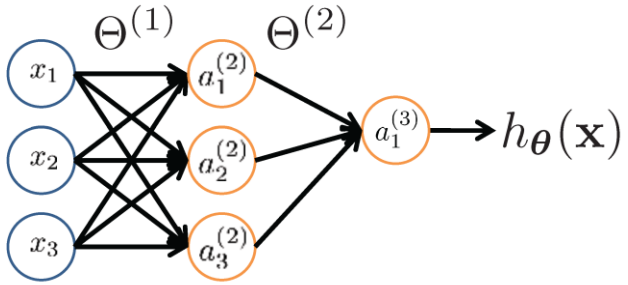
# Neural Network



# Feed-Forward Process

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

# Neural Network




$a_i^{(j)} =$ "activation" of unit $i$ in layer $j$

$\Theta^{(j)} =$ weight matrix controlling function mapping from layer $j$ to layer $j$ + 1

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

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

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

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

If network has $s_j$ units in layer $j$ *and* $s_{j+1}$ units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$

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

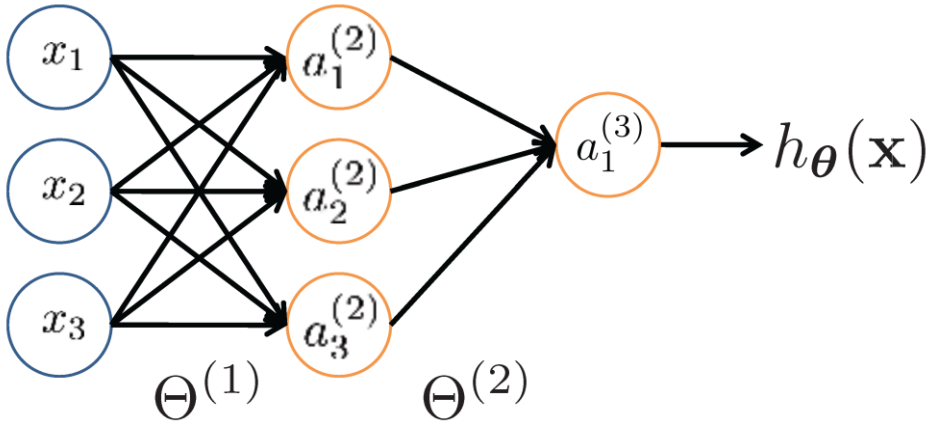
# Vectorization

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

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

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

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




Feed-Forward Steps:

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

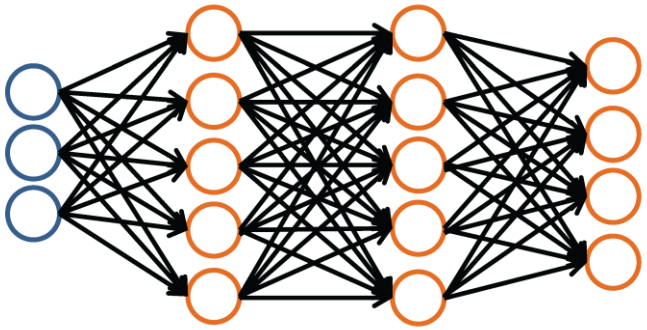
$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

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

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

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

# Neural Network Classification



**Given:**

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

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

- $s_0 = d$ (# features)

Binary classification

$y$ = 0 or 1

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

Multi-class classification ($K$ classes)

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

positive neural negative

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

# Neural Network Learning

# Learning in NN: Backpropagation

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

- The trick is to assess the blame for the error and divide it among the contributing weights

# General NN Cost Function

$$J(\Theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\big(h_\Theta(\mathbf{x}_i), y_i\big) \quad + \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

Given $h_\Theta(\mathbf{x}_i) = \hat{y}$

Loss of Binary classification (1 or 0) ← log loss

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

Loss of Regression ← squared error loss

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

Loss of Multi-class classification with softmax regression ← cross-entropy loss

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} \mathbf{1}\{y = j\} \log \hat{y}_j = -\sum_{j=1}^{k} y_j \log \hat{y}_j$$

# Optimizing the Neural Network

$$J(\Theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\big(h_\Theta(\mathbf{x}_i), y_i\big) \quad + \frac{\lambda}{2n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(\Theta_{ji}^{(l)}\right)^2$$

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

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

- But, tends to work well in practice

Need code to compute:

- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

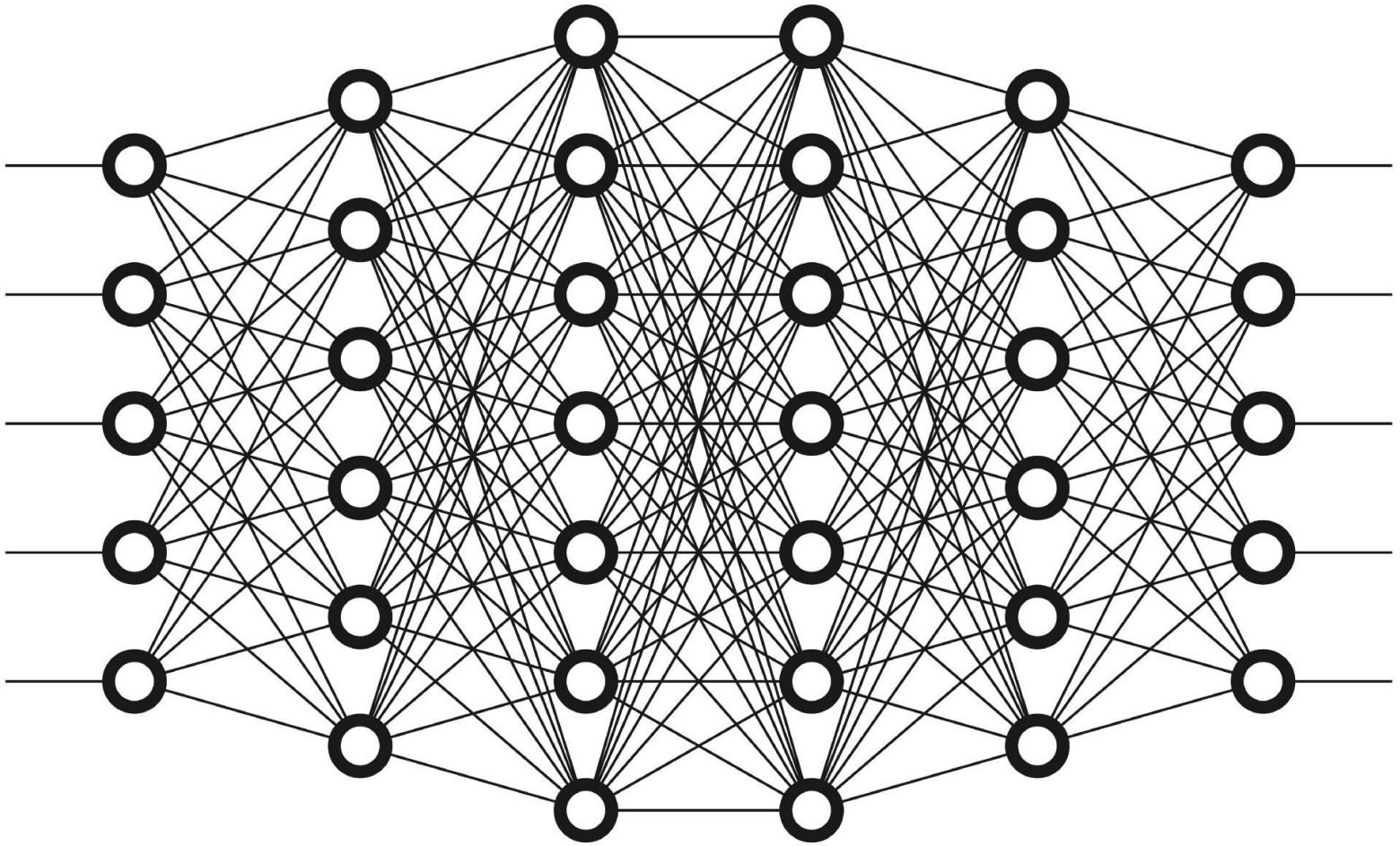
# Training a Neural Network via Gradient Descent with Backprop

Given: training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
Initialize all $\Theta^{(l)}$ randomly (NOT to 0!)
Loop // each iteration is called an epoch
    Set $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$ (Used to accumulate gradient)
    For each training instance $(\mathbf{x}_i, y_i)$:
        Set $\mathbf{a}^{(1)} = \mathbf{x}_i$
        Compute $\{\mathbf{a}^{(2)}, \ldots, \mathbf{a}^{(L)}\}$ via forward propagation
        Compute $\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - y_i$
        Compute errors $\{\boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}\}$
        Compute gradients $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
    Compute avg regularized gradient $D_{ij}^{(l)} = \begin{cases} \frac{1}{n}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n}\Delta_{ij}^{(l)} & \text{otherwise} \end{cases}$
    Update weights via gradient step $\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$
Until weights converge or max #epochs is reached

Backpropagation

# Deep Learning

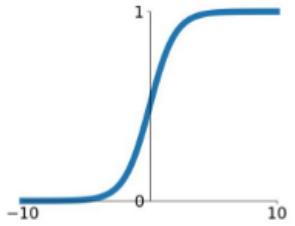
# Deep Neural Network



# Deep Net Activation Functions

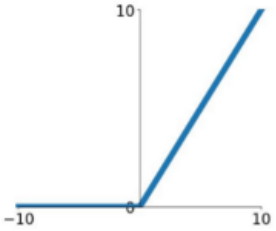
**Sigmoid**

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



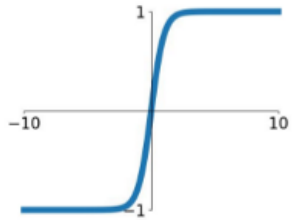

**ReLU**

$$\max(0, x)$$



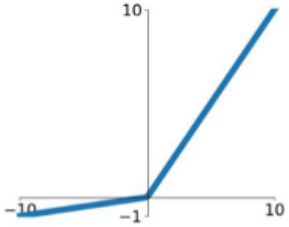

**tanh**

$$\tanh(x)$$




**Leaky ReLU**

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




## Benefits of ReLU

- Cheap to compute as there is no complicated math and hence easier to optimize
- It converges faster.

## Best Practices

- Use **ReLU** in hidden layer activation, but be careful with the learning rate and monitor the fraction of dead units. If **ReLU** is giving problems. Try Leaky ReLU.

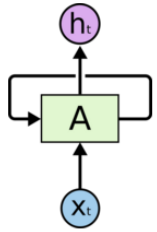
https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044

# Beyond the Bag-of-words

- Some linguistic phenomena require going beyond the bag-of-words:
  - That’s not bad for the first day
  - This is not the worst thing that can happen
  - It would be nice if you acted like you understood
  - This film should be brilliant. The actors are first grade. Stallone plays a happy, wonderful man. His sweet wife is beautiful and adores him. He has a fascinating gift for living life fully. It sounds like a great plot, however, the film is a failure.

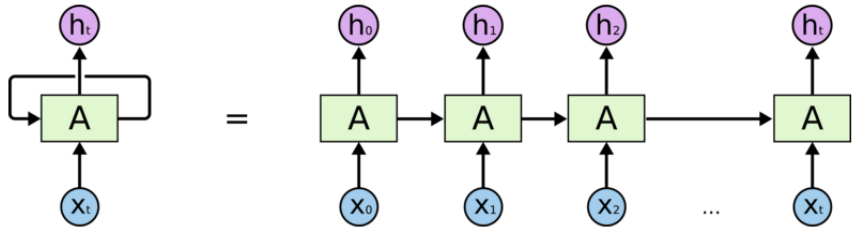
# Recurrent Neural Networks

- Recurrent Neural Networks are networks with loops in them, allowing information to persist.




Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, ***A***, looks at some input $x_t$ and outputs a value $h_t$.
A loop allows information to be passed from one step of the network to the next.




An unrolled recurrent neural network.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
The diagram above shows what happens if we unroll the loop.

# Recurrent Neural Networks

- Intuition of Recurrent Neural Networks
  - Human thoughts have persistence; humans don't start their thinking from scratch every second.
    - As you read this sentence, you understand each word based on your understanding of previous words.
  - One of the appeals of RNNs is the idea that they are able to connect previous information to the present task
    - E.g., using previous video frames to inform the understanding of the present frame.
    - E.g., a language model tries to predict the next word based on the previous ones.

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential()
model.add(layers.Embedding(10000, 32, input_length=20))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(1, activation="sigmoid"))
model.summary()
```

Size of the vocabulary

Dimension of the dense embedding

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 20, 32)            320000
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 32)                2080
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

An embedding is a relatively low-dimensional space into which you can translate high-dimensional vectors.

In Keras, turns positive integers (indexes) into dense vectors of fixed size.

e.g. [[4], [20]] -> [[0.25, 0.1], [0.6, -0.2]]

https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

https://keras.io/api/layers/core_layers/embedding/

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential()
model.add(layers.Embedding(10000, 32, input_length=20))
model.add(layers.LSTM(32))
model.add(layers.Dense(1, activation="sigmoid"))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 20, 32)            320000
```

```
from tensorflow import keras
from tensorflow.keras import layers
model = keras.Sequential()
model.add(layers.Embedding(10000, 32, input_length=20))
model.add(layers.GRU(32))
model.add(layers.Dense(1, activation="sigmoid"))
model.summary()
```

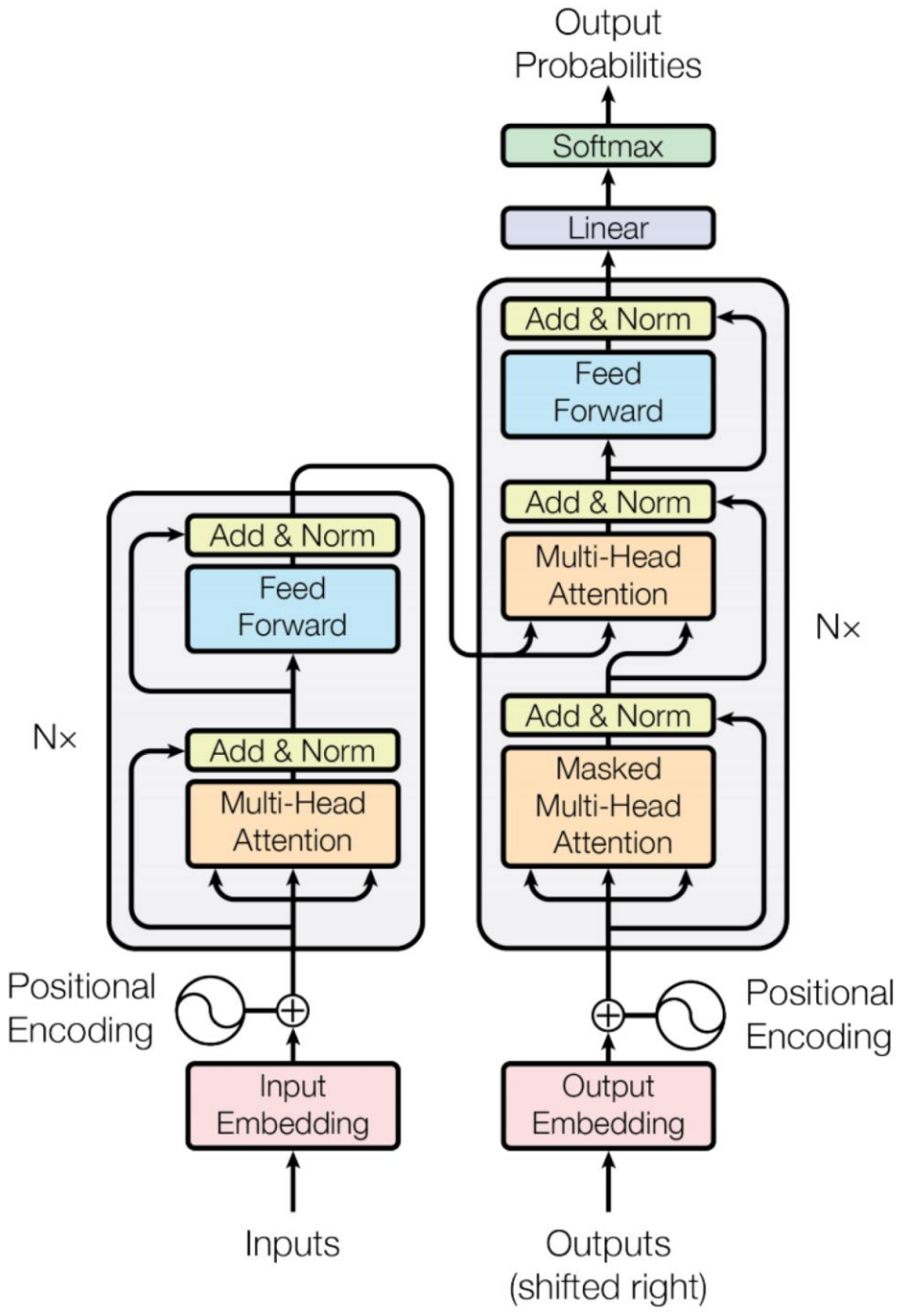
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 20, 32)            320000
_________________________________________________________________
gru (GRU)                    (None, 32)                6336
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 326,369
Trainable params: 326,369
Non-trainable params: 0
_________________________________________________________________
```

# Transformer Overview

Attention is all you need. 2017. Aswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin https://arxiv.org/pdf/1706.03762.pdf

- Non-recurrent sequence-to-sequence encoder-decoder model
- Task: machine translation with parallel corpus
- Predict each translated word
- Final cost/error function is standard cross-entropy error on top of a softmax classifier

This and related figures from paper ⇑




Some of the following slides were adapted from Christopher Manning and John Hewitt

# References for Transformer

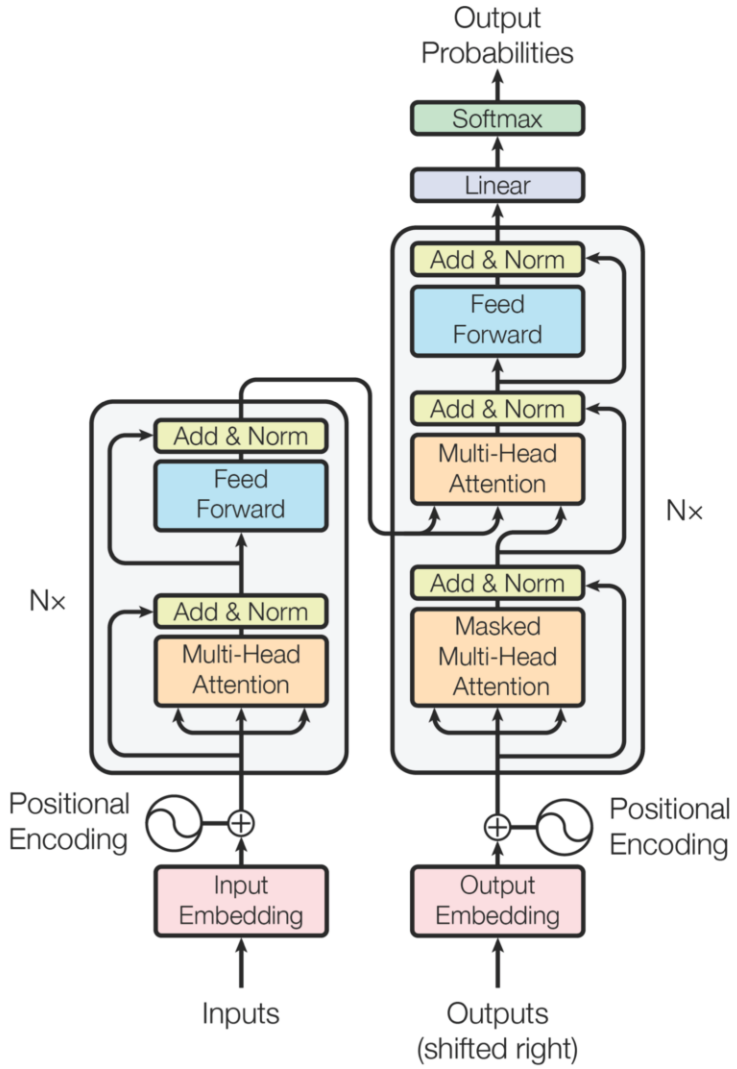


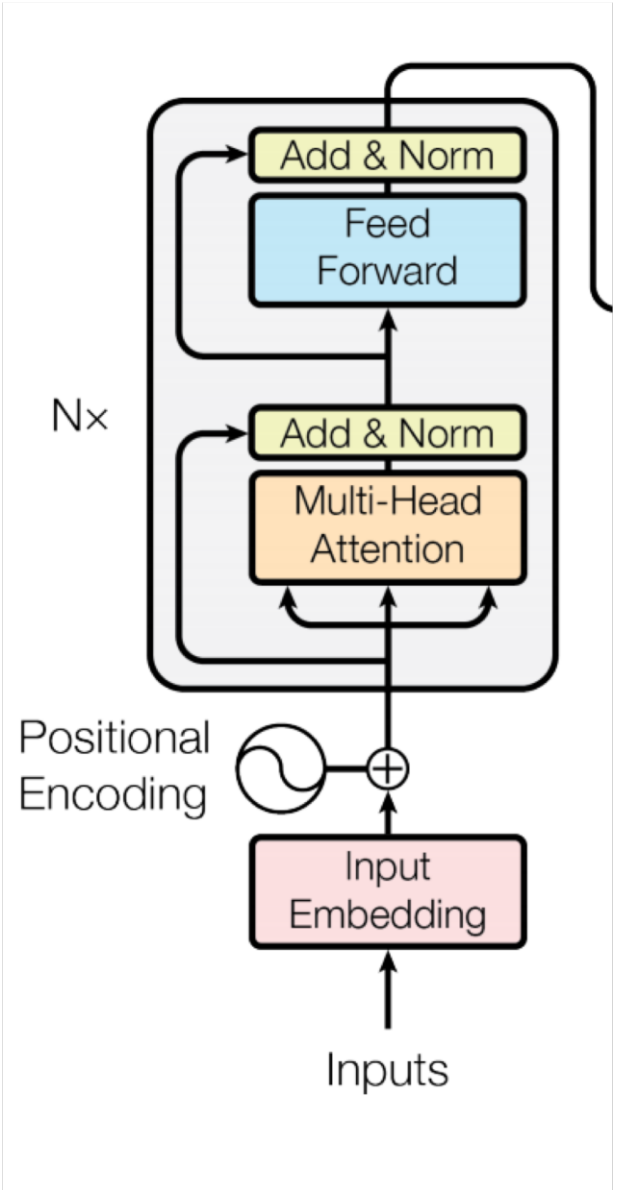

Figure 1: The Transformer - model architecture.

http://jalammar.github.io/illustrated-transformer/
https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0
Code: https://www.tensorflow.org/text/tutorials/transformer

# Transformer Encoder

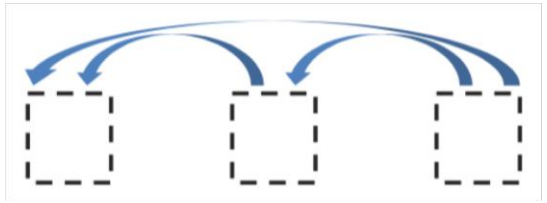
- For encoder, at each block, we use the same Q, K and V
  from the previous layer

- Blocks are repeated 6 times (in vertical stack)

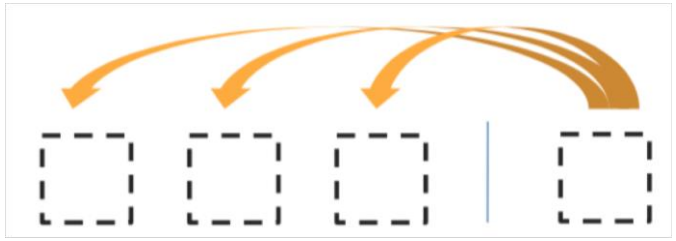


# Transformer Decoder

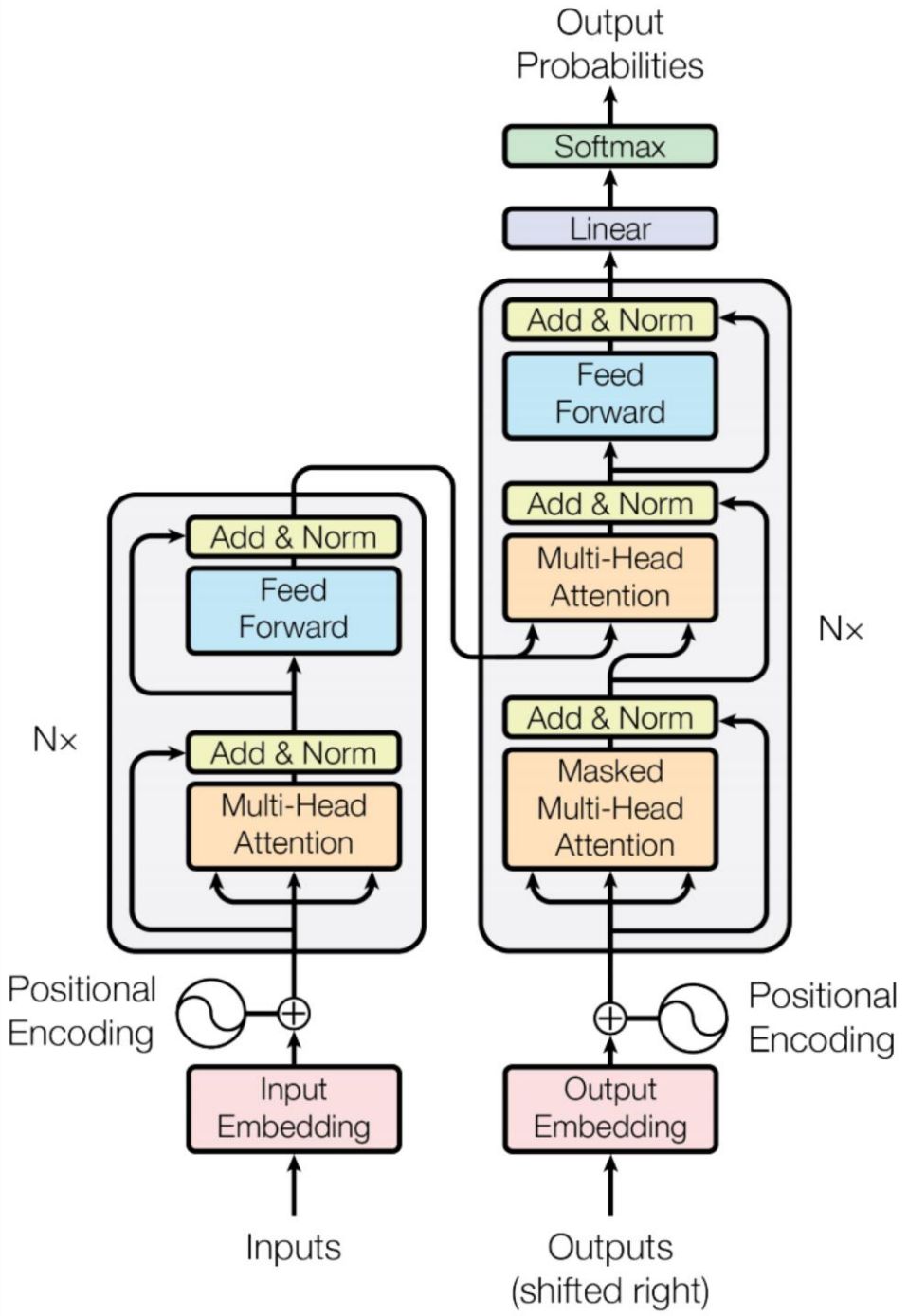
- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder

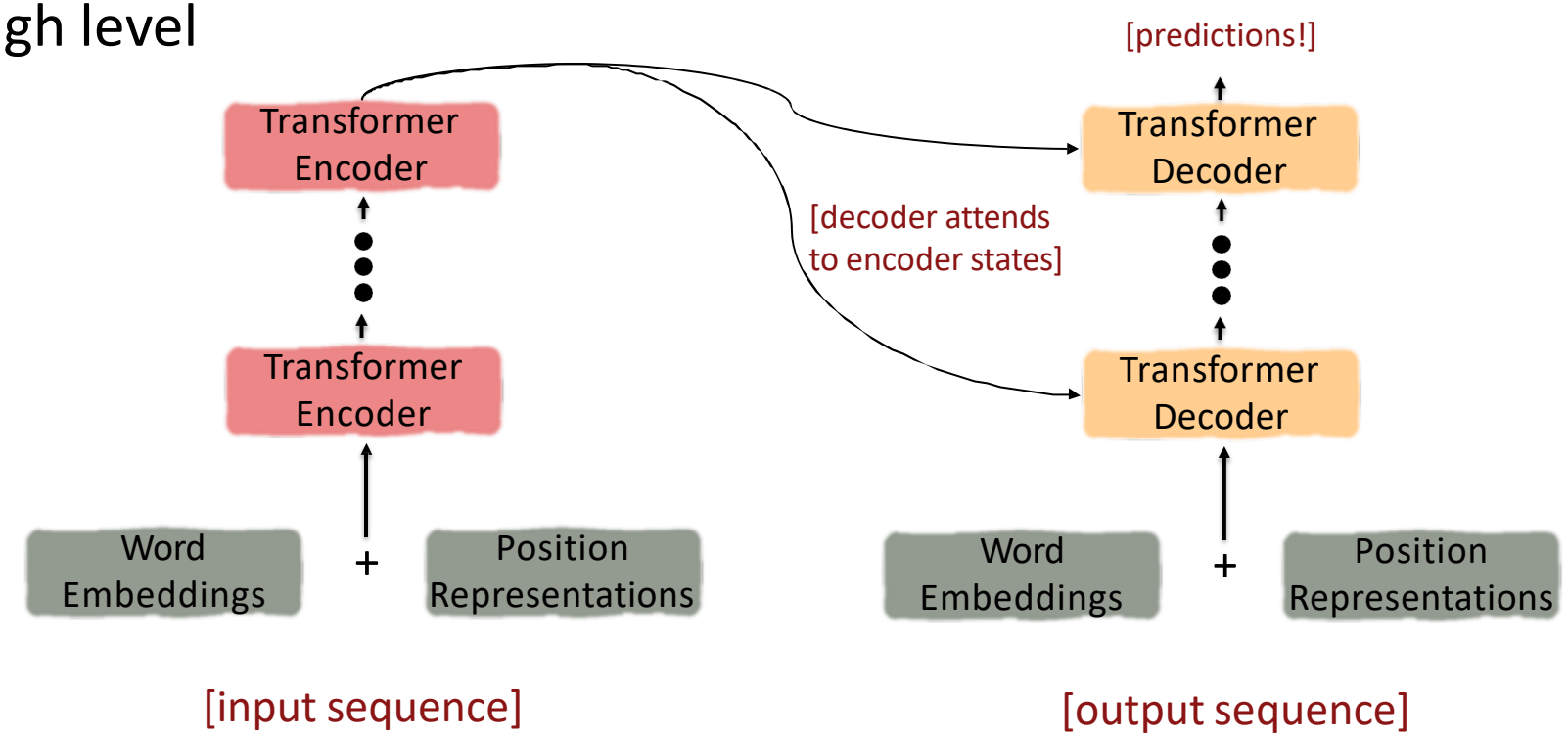


Blocks repeated 6 times also



# The Transformer Encoder-Decoder [Vaswani et al., 2017]

Another look at the Transformer Encoder and Decoder Blocks at a high level



# The Transformer Encoder-Decoder [Vaswani et al., 2017]

Next, let's look at the Transformer Encoder and Decoder Blocks

1. **Key-query-value attention:** How do we get the $k, q, v$ vectors from a single word embedding?
2. **Multi-headed attention**: Attend to multiple places in a single layer!
3. **Tricks to help with training!**
    1. Scaling the dot product
    2. Residual connections
    3. Layer normalization
    4. These tricks **don't improve** what the model is able to do; they help improve the training process

# The Transformer Encoder:

## Dot-Product Attention

- Inputs: a query q and a set of key-value (k-v) pairs to an output
- Query, keys, values, and output are all vectors

- Output is weighted sum of values, where
- Weight of each value is computed by an inner product of query and corresponding key
- Queries and keys have same dimensionality $d_k$ , value have $d_v$

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

# The Transformer Encoder:
## Dot-Product Attention – Matrix notation

- When we have multiple queries q, we stack them in a matrix Q:

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

- Becomes: $A(Q, K, V) = softmax(QK^T)V$

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

softmax row-wise

$= [|Q| \times d_v]$

# The Transformer Encoder:
# **Key-Query-Value Attention**

- Self-attention is when keys, queries, and values come from the same source. The Transformer does this in a particular way:
  - Let $x_1, \ldots, x_T$ be input vectors to the Transformer encoder; $x_i \in \mathbb{R}^d$

- Then keys, queries, values are:
  - $k_i = Kx_i$, where $K \in \mathbb{R}^{d \times d}$ is the key matrix.
  - $q_i = Qx_i$, where $\mathrm{Q} \in \mathbb{R}^{d \times d}$ is the query matrix.
  - $v_i = Vx_i$, where $\mathrm{V} \in \mathbb{R}^{d \times d}$ is the value matrix.

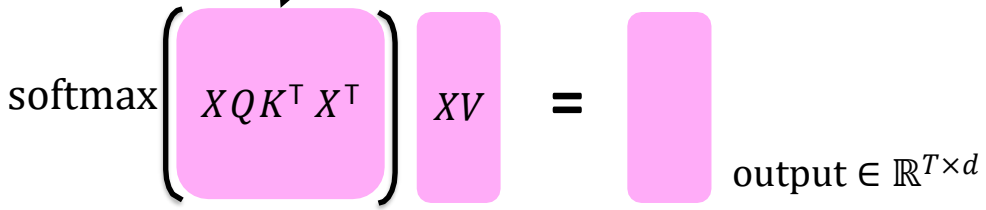
- These matrices allow *different aspects* of the $x$ vectors to be used/emphasized in each of the three roles.

# The Transformer Encoder:
# Key-Query-Value Attention

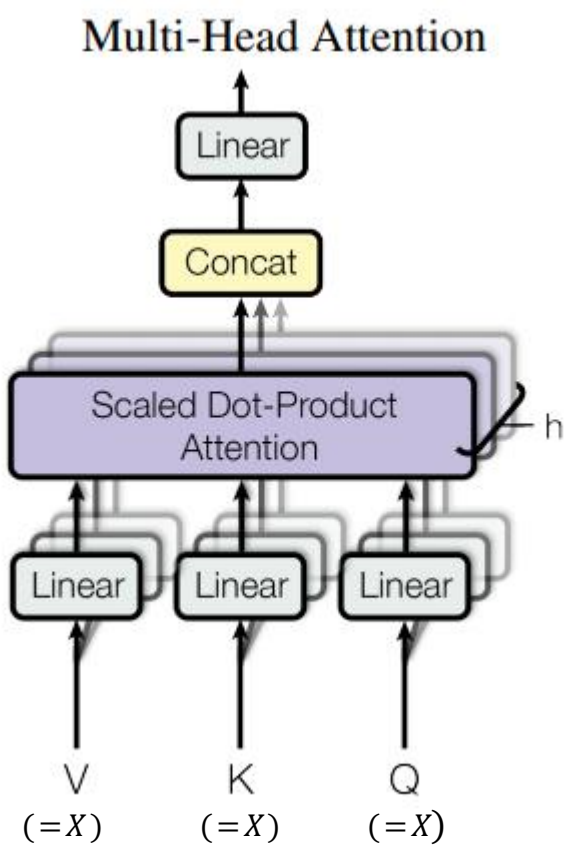
- Let's look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \ldots; x_T] \in \mathbb{R}^{T \times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{T \times d}$, $XQ \in \mathbb{R}^{T \times d}$, $XV \in \mathbb{R}^{T \times d}$.
  - The output is defined as output $= \text{softmax}(XQ(XK)^\top) \times XV$.

First, take the query-key dot products in one matrix multiplication: $XQ\,(XK)$


Next, softmax, and compute the weighted average with another matrix multiplication.



# The Transformer Encoder:
# **Multi-headed attention**

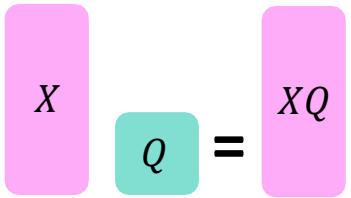



- What if we want to look in multiple places in the sentence at once?
  - For word $i$, self-attention "looks" where $x^\top Q^\top K x_j$ is high, but maybe we want to focus on different $j$ for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_P, K_P, V_P \in \mathbb{R}^{d \times \frac{d}{h}}$, where $h$ is the number of attention heads, and $P$ ranges from 1 to $h$.
- Each attention head performs attention independently:
  - $\text{output}_P = \text{softmax}\left(XQ_P K_P^\top X^\top\right) * XV_P$, where $\text{output}_P \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = Y[\text{output}_1; \ldots; \text{output}_h]$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differently.

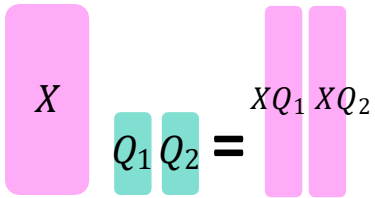
# The Transformer Encoder:
# Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word $i$, self-attention "looks" where $x^\top Q^\top K x_j$ is high, but maybe we want to focus on different $j$ for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_P, K_P, V_P \in \mathbb{R}^{d \times \frac{d}{h}}$, where $h$ is the number of attention heads, and $P$ ranges from 1 to $h$.

**Single-head attention**
(just the query matrix)



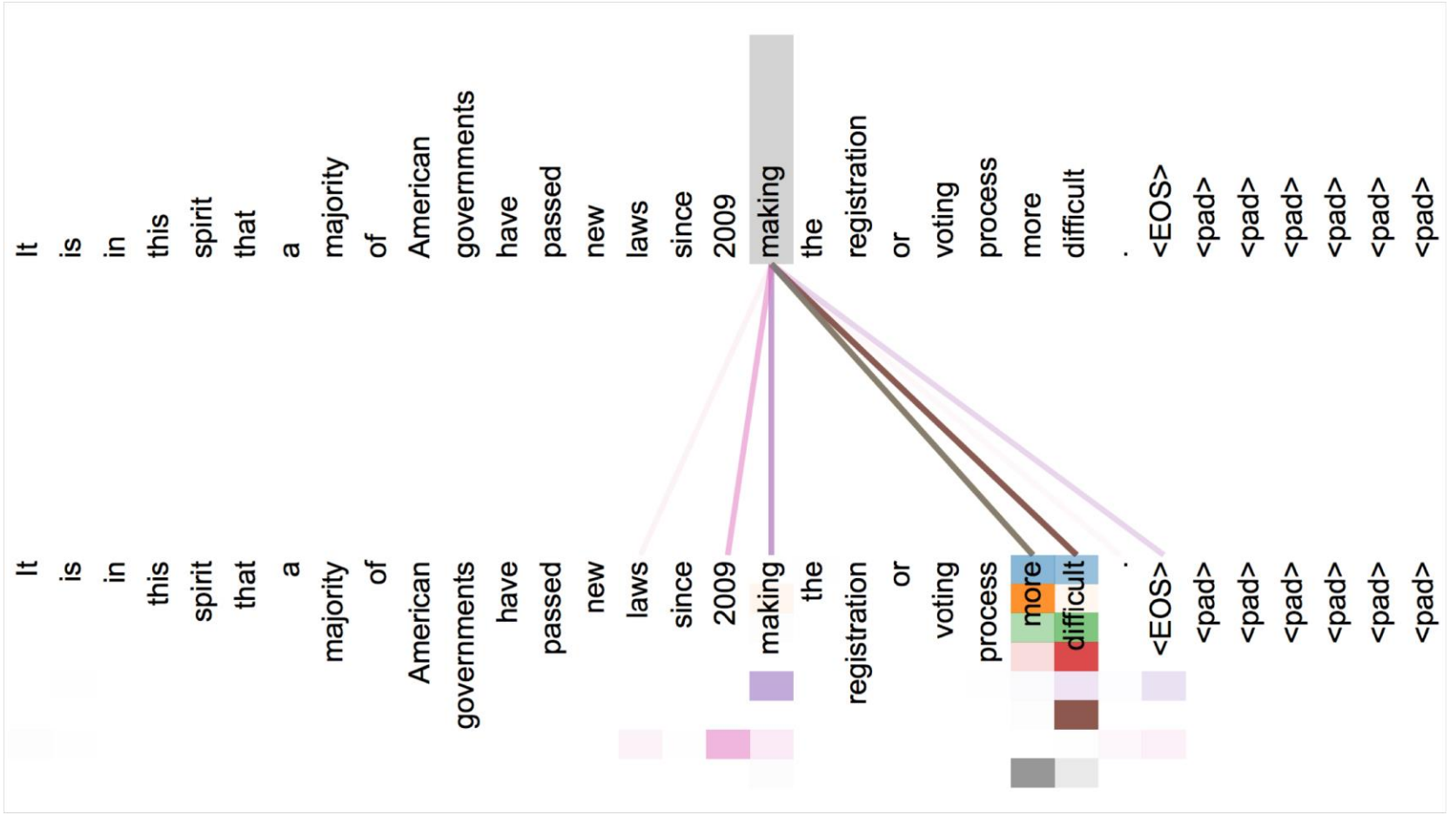

**Multi-head attention**
(just two heads here)



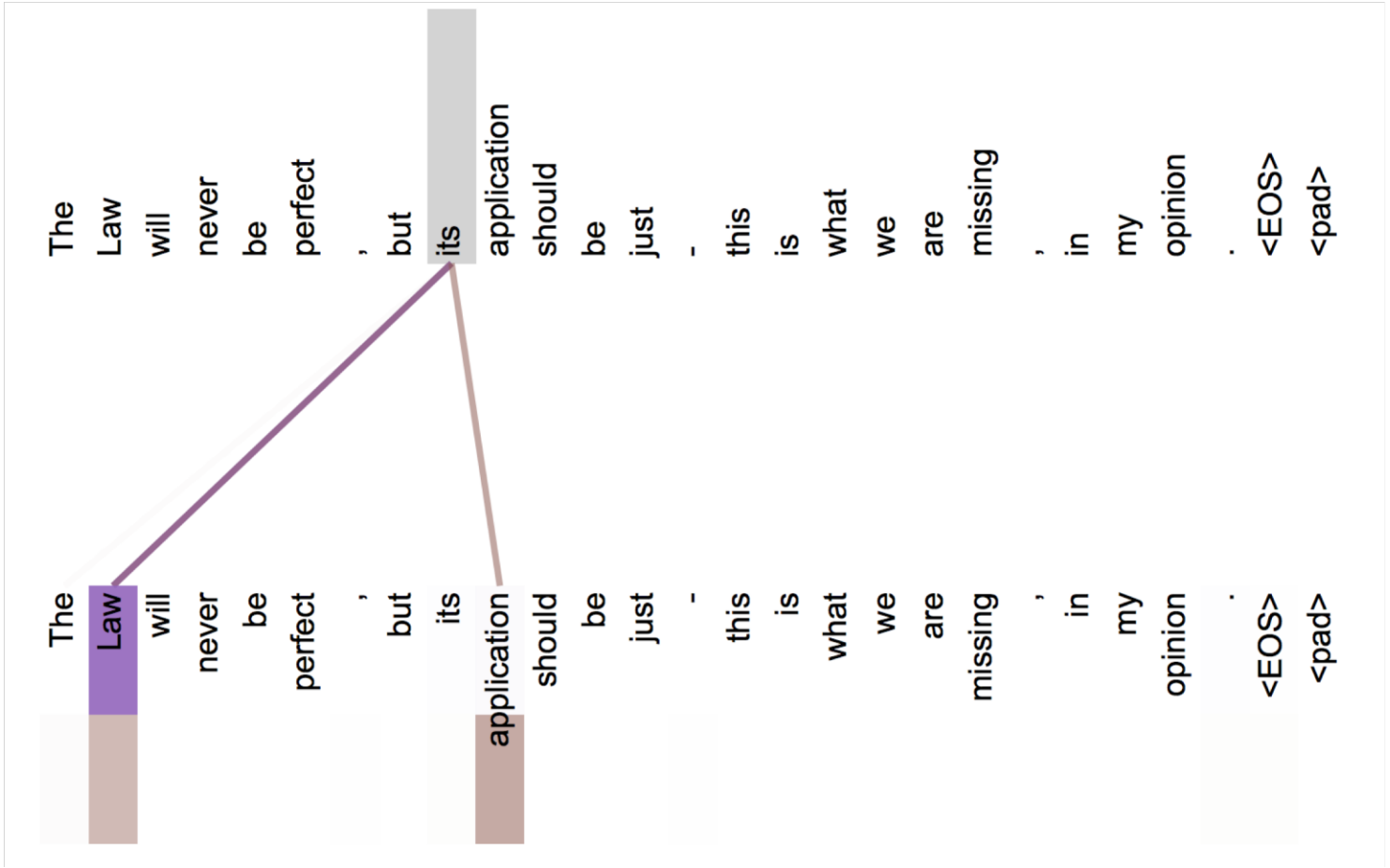

Same amount of computation as single-head self-attention!

# Attention visualization in layer 5

- Words start to pay attention to other words in sensible ways

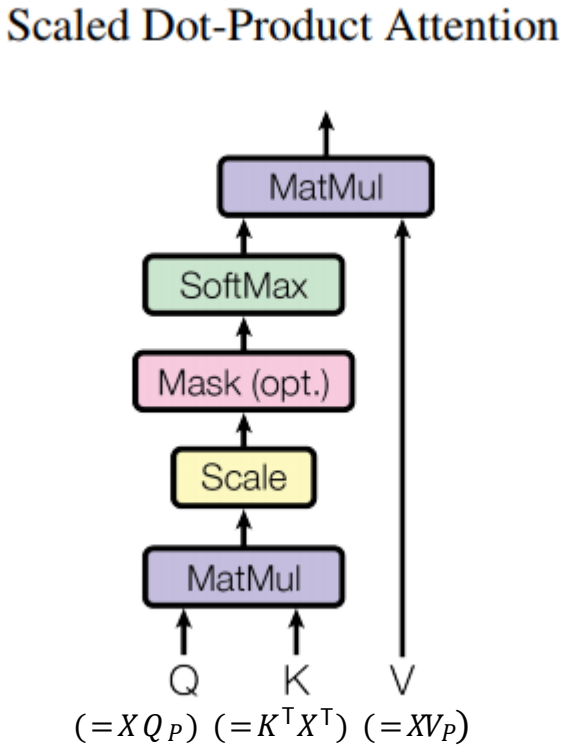


# Attention visualization: Implicit anaphora resolution




In 5th layer. Isolated attentions from just the word ‘its’ for attention heads 5 and 6. Note that the attentions are very sharp for this word.

# The Transformer Encoder:
# **Scaled Dot Product** [Vaswani et al., 2017]




- **"Scaled Dot Product"** attention is a final variation to aid in Transformer training.
- When dimensionality $d$ becomes large, dot products between vectors become large, inputs to the softmax function can be large, making gradients small.

- Instead of the self-attention function we've seen:

$$\text{output}_P = \text{softmax}\left(XQ_P K_P^\top X^\top\right) * XV_P$$

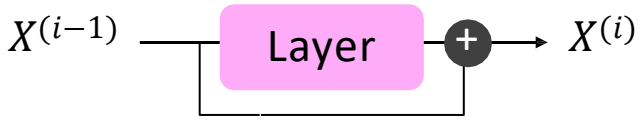
- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of $d/h$ (The dimensionality divided by the number of heads.)

$$\text{output}_P = \text{softmax}\left(\frac{XQ_P K_P^\top X^\top}{\sqrt{d/h}}\right) * XV_P$$

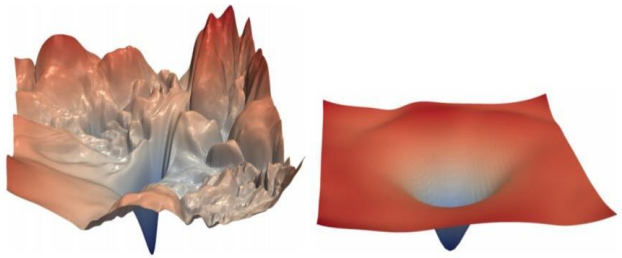
# The Transformer Encoder:
# **Residual connections** [He et al., 2016]

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where $i$ represents the layer)


  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)




  - Residual connections are thought to make the loss landscape considerably smoother (thus easier training!)



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

# The Transformer Encoder:
# **Layer normalization** [Ba et al., 2016]

- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \Sigma_{j=1}^{d} x_j$ ; this is the mean; $\mu \in \mathbb{R}$.
  - Let $\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x - \mu)^2}$ ; this is the standard deviation; $\sigma \in \mathbb{R}$.
  - Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
  - Then layer normalization computes:

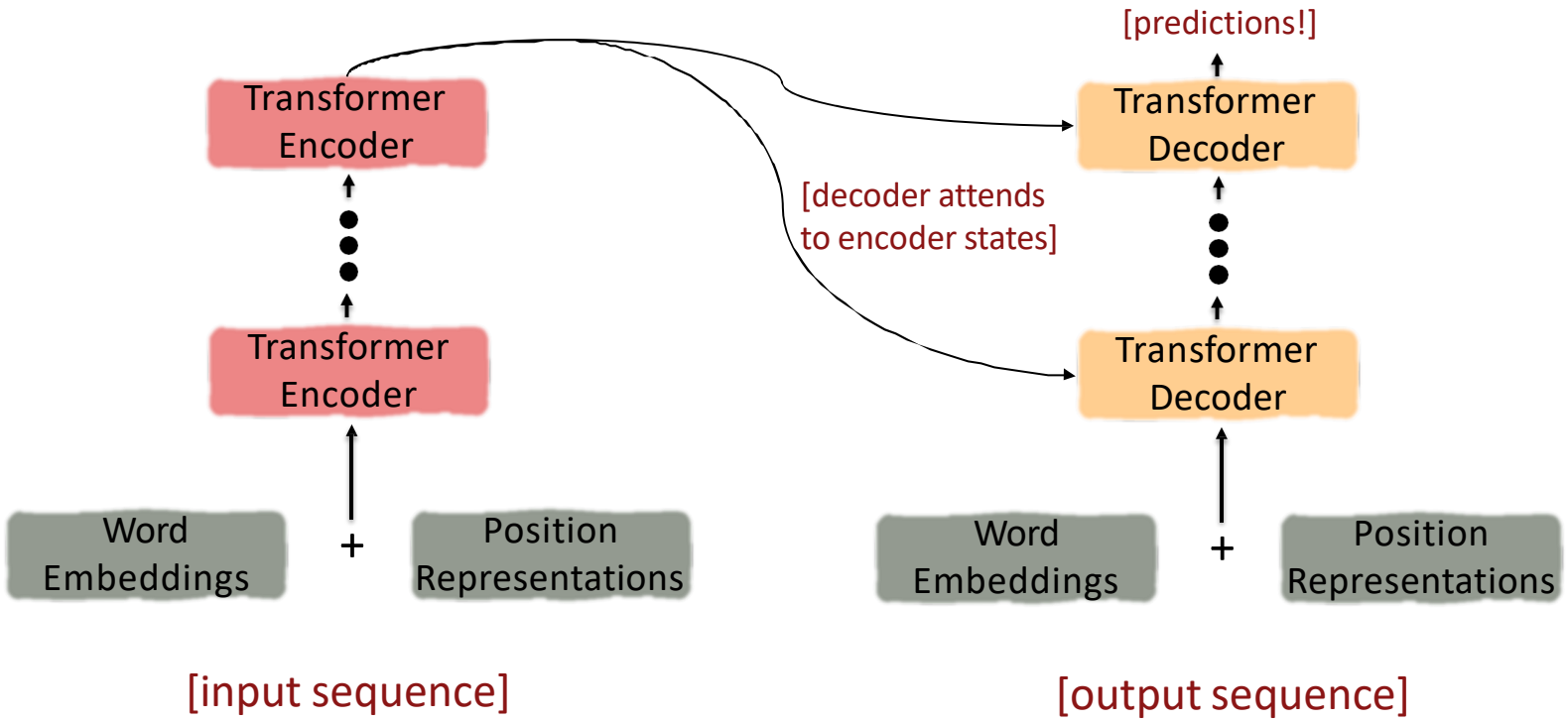
$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

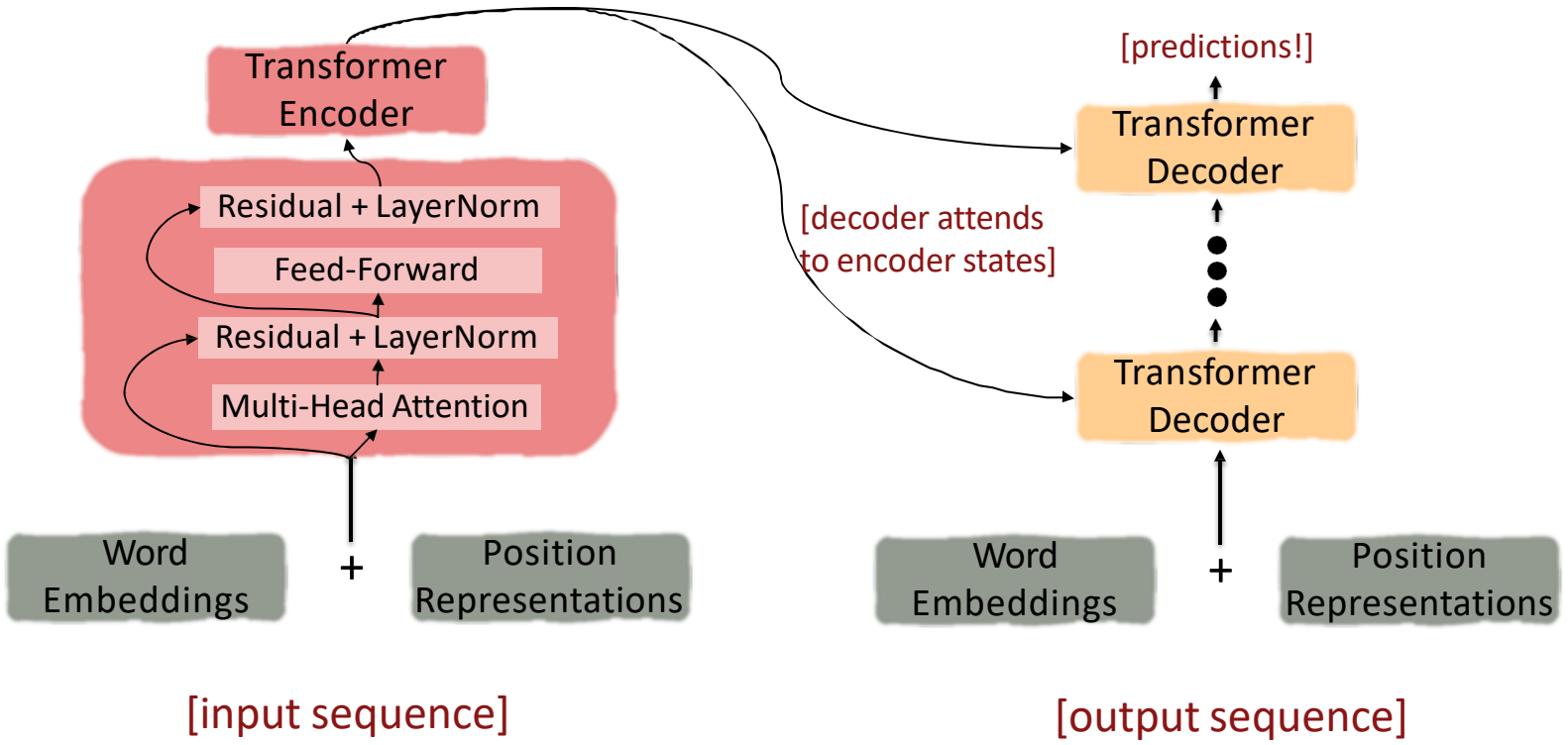
# The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



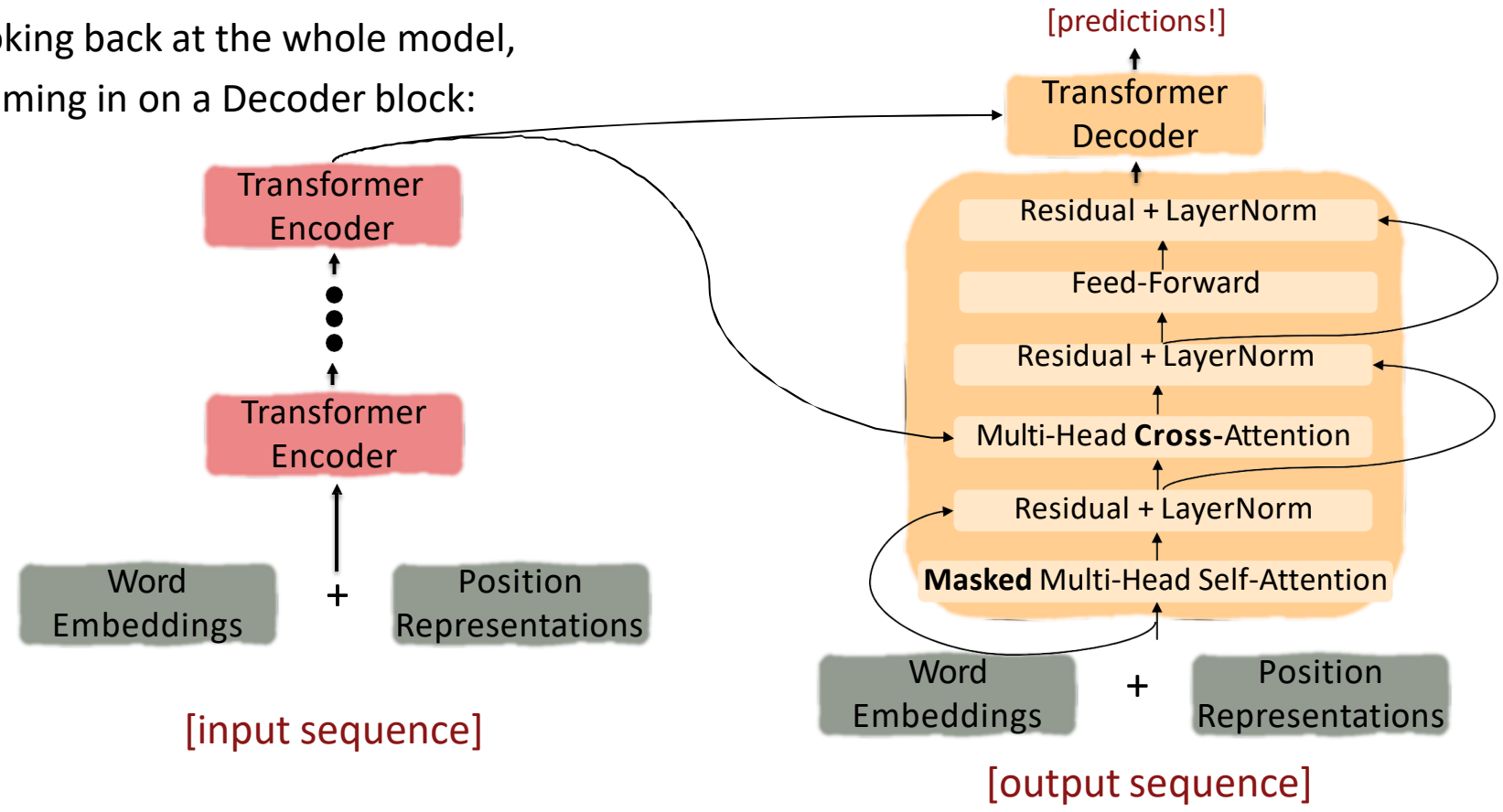
# The Transformer Encoder-Decoder [Vaswani et al., 2017]

Looking back at the whole model, zooming in on an Encoder block:



# The Transformer Encoder-Decoder [Vaswani et al., 2017]

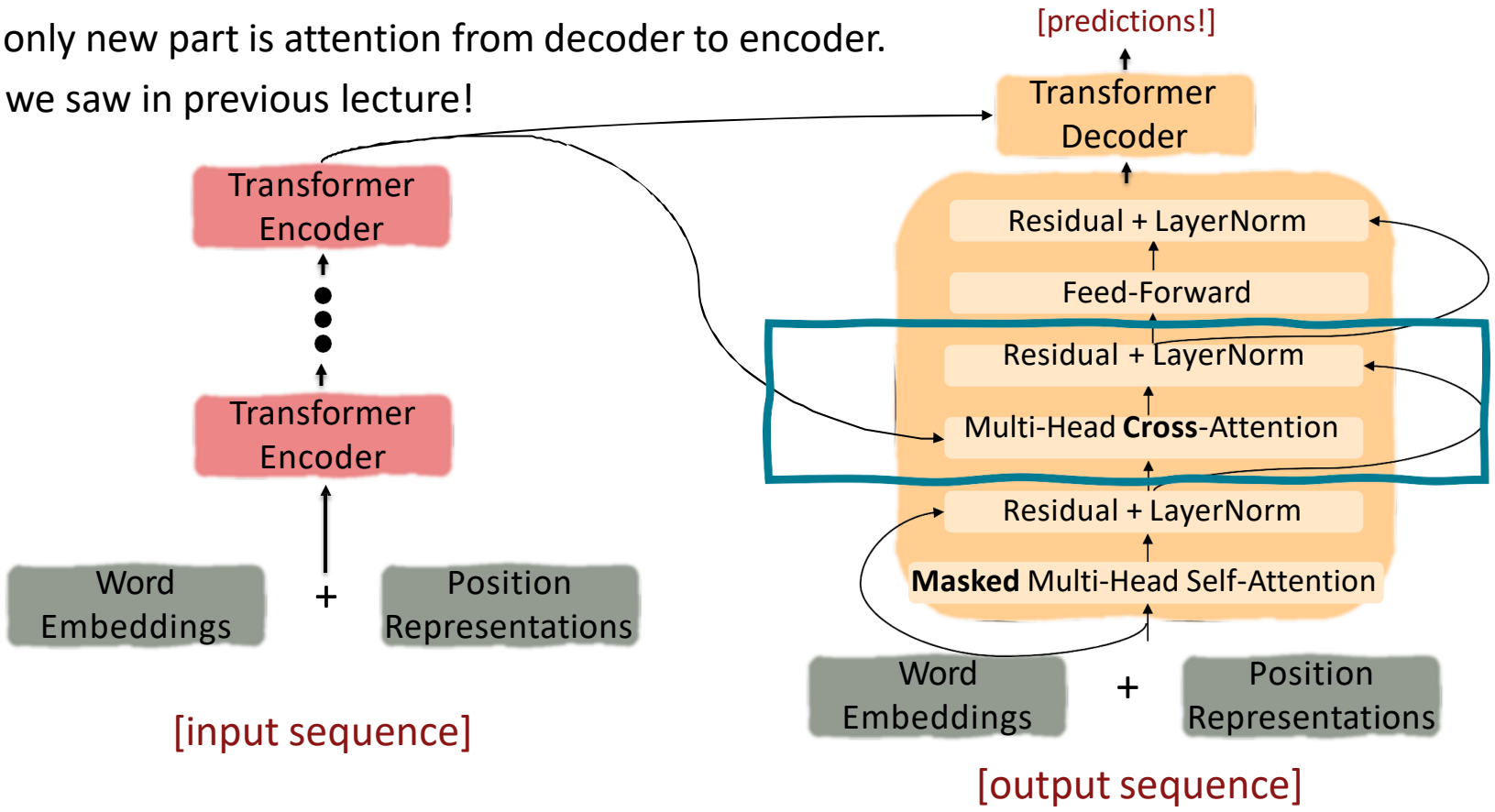
Looking back at the whole model, zooming in on a Decoder block:



# The Transformer Encoder-Decoder [Vaswani et al., 2017]

The only new part is attention from decoder to encoder.

Like we saw in previous lecture!



# The Transformer Decoder:
# Cross-attention (details)

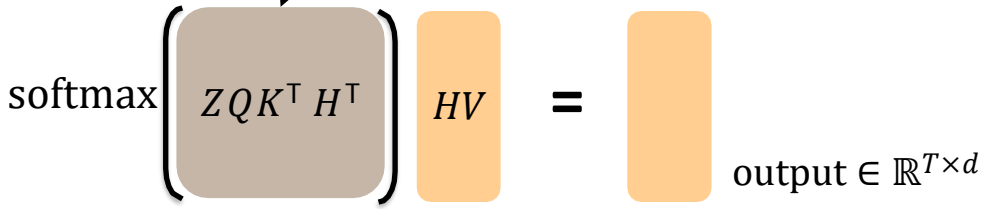
- We saw self-attention is when keys, queries, and values come from the same source.
- Let $h_1, \ldots, h_T$ be **output** vectors from the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i$, $v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.

# The Transformer Encoder:
# Cross-attention (details)

- Let's look at how cross-attention is computed, in matrices.
  - Let $\mathrm{H} = [h_1; \ldots; h_T] \in \mathbb{R}^{T \times d}$ be the concatenation of encoder vectors.
  - Let $\mathrm{Z} = [z_1; \ldots; z_T] \in \mathbb{R}^{T \times d}$ be the concatenation of decoder vectors.
  - The output is defined as $\text{output} = \text{softmax}(ZQ(HK)^\intercal)HV$.

First, take the query-key dot products in one matrix multiplication: $ZQ(HK)^\intercal$


Next, softmax, and compute the weighted average with another matrix multiplication.



# References for Transformer

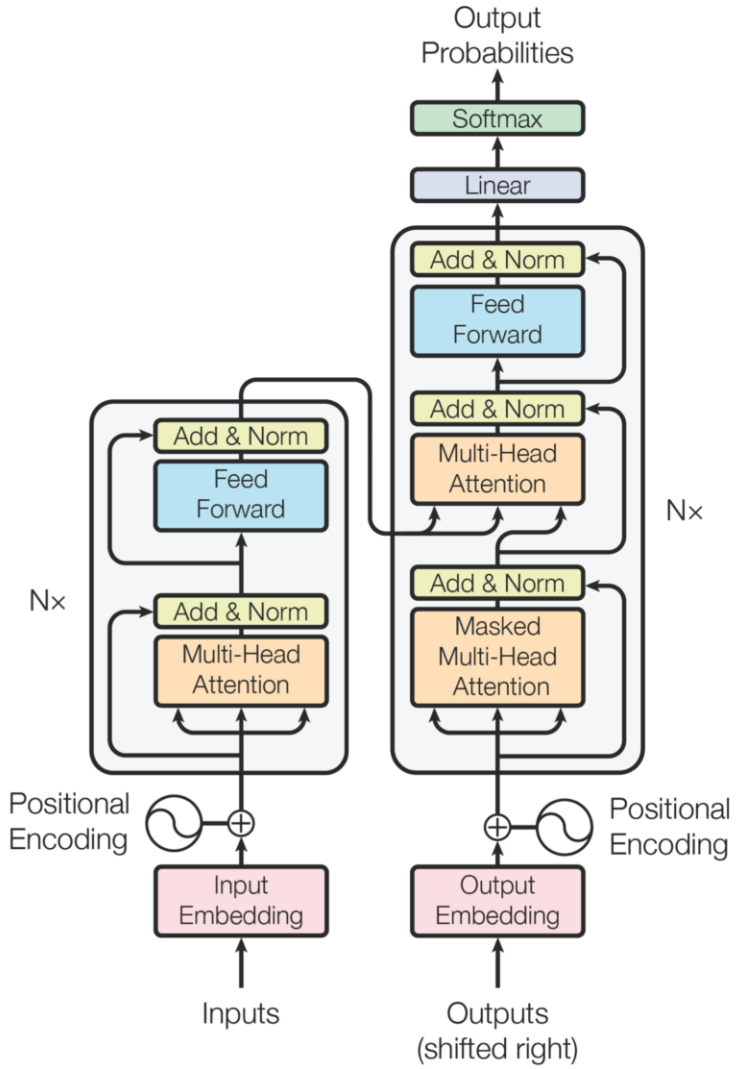



Figure 1: The Transformer - model architecture.

http://jalammar.github.io/illustrated-transformer/
https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0
Code: https://www.tensorflow.org/text/tutorials/transformer

.GlobalAveragePooling

# An example of Transformer-based Classification

- On top of Transformer Encoder Block
- Add GlobalAveragePooling
- Add Feedforward Neural Net
- Then, do softmax

- https://keras.io/examples/nlp/text_classification_with_transformer/

# Evaluating a Classifier

# Evaluating classification

- Evaluation must be done on test data that are independent of the training data (usually a disjoint set of instances).
- It's easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set).
- Measures: Precision, recall, $F_1$, classification accuracy

# Precision *P* and recall *R*

| | in the class | not in the class |
|---|---|---|
| predicted to be in the class | true positives (TP) | false positives (FP) |
| predicted to not be in the class | false negatives (FN) | true negatives (TN) |

$P = TP / ( TP + FP)$

$R = TP / ( TP + FN)$

# A combined measure: *F*

- $F_1$ allows us to trade off precision against recall.
- $$F_1 = \frac{1}{\frac{1}{2}\frac{1}{P} + \frac{1}{2}\frac{1}{R}} = \frac{2PR}{P+R}$$
- This is the harmonic mean of *P* and *R*: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$

# Averaging: Micro vs. Macro

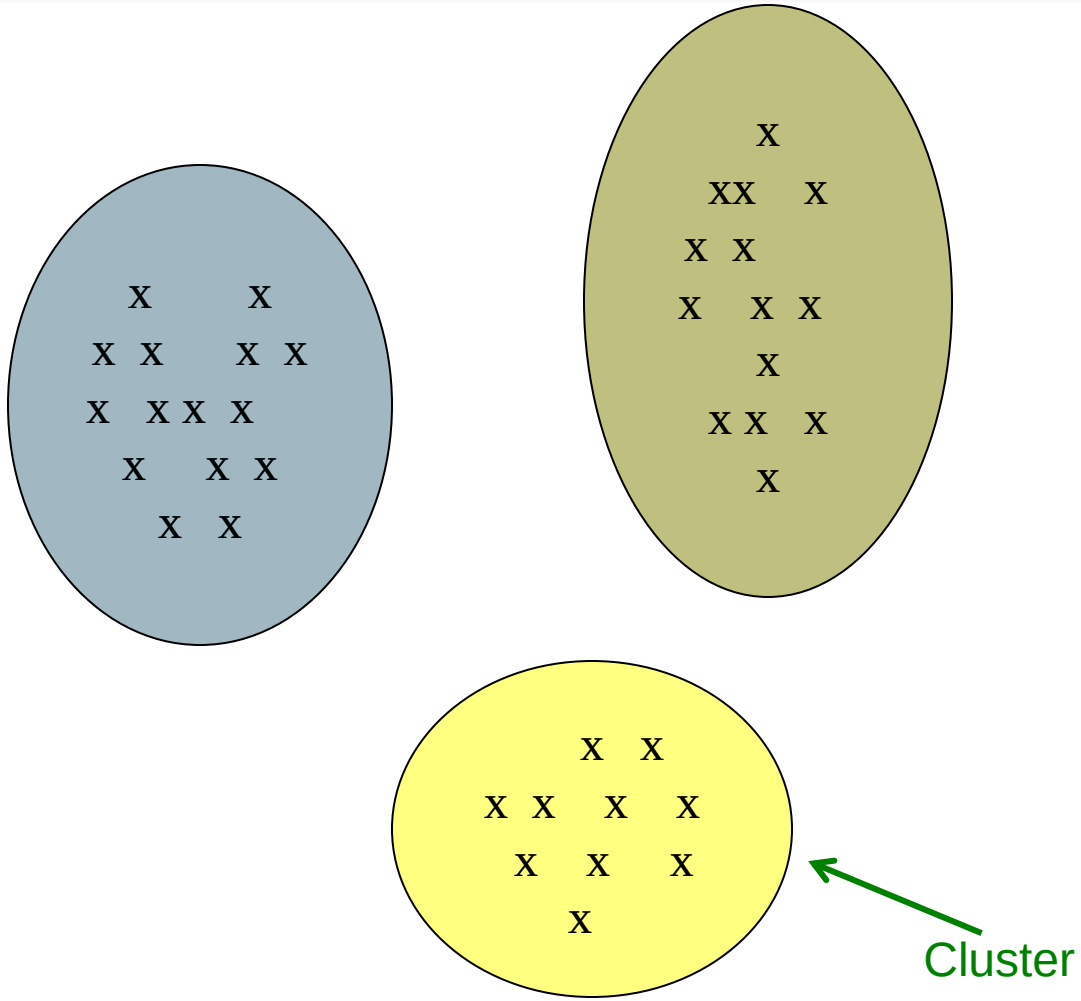
- We now have an evaluation measure ($F_1$) for one class.
- But we also want a single number that measures the aggregate performance over all classes in the collection.
- Macroaveraging
  - Compute $F_1$ for each of the $C$ classes
  - Average these $C$ numbers
- Microaveraging
  - Compute TP, FP, FN for each of the $C$ classes
  - Sum these $C$ numbers (e.g., all TP to get aggregate TP)
  - Compute $F_1$ for aggregate TP, FP, FN

# Clustering

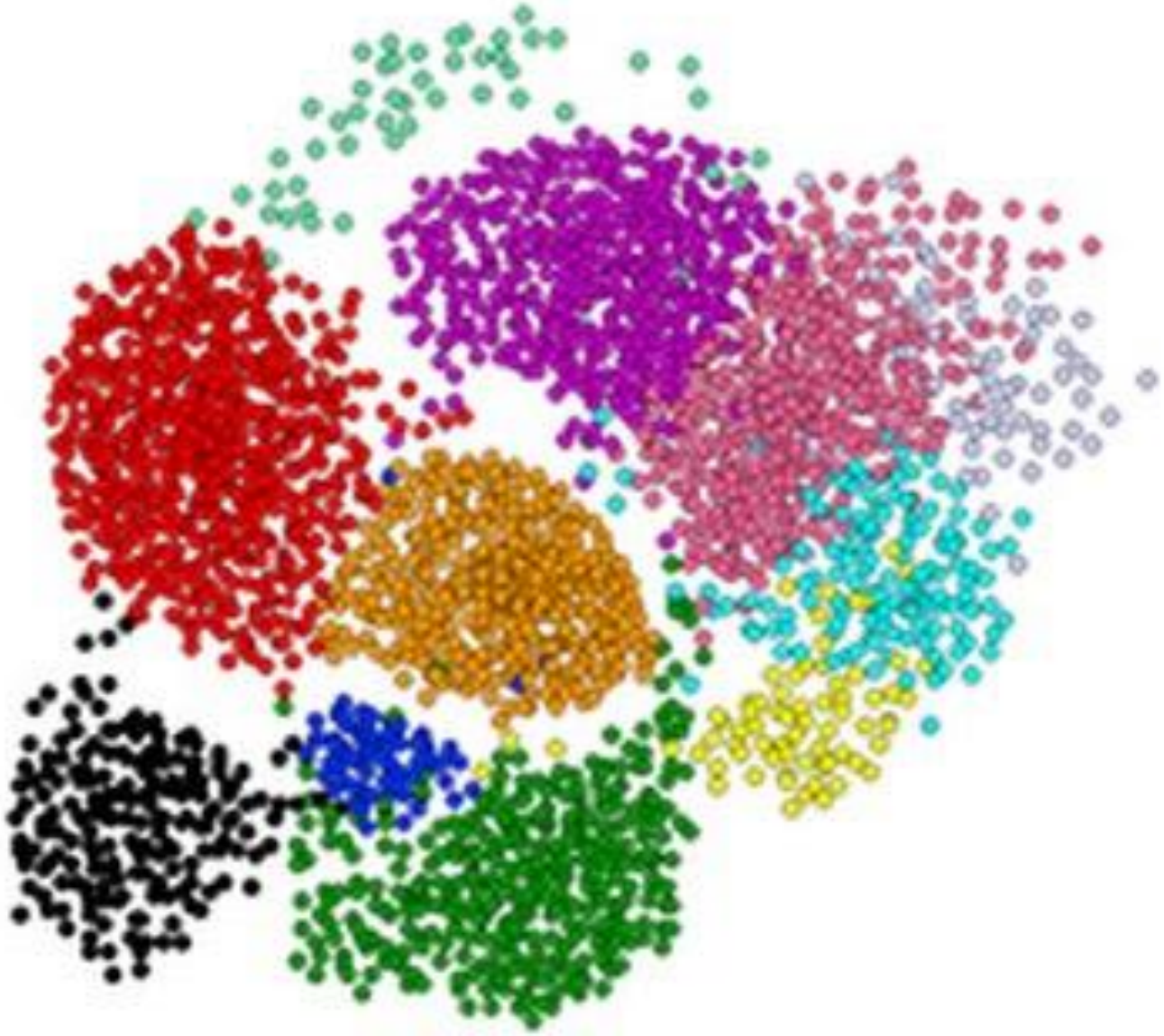
# What is Clustering?

- **Clustering** is the process of grouping a set of documents into clusters of similar documents.
  - Documents within a cluster should be similar.
  - Documents from different clusters should be dissimilar.
- Clustering is the most common form of *unsupervised learning.*
  - Unsupervised learning = learning from raw data, as opposed to supervised data where a classification of examples is given
- A common and important task that finds many applications in IR and other places

# Example Clusters



# Clustering is Hard!



# Why is it hard?

- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving

- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance

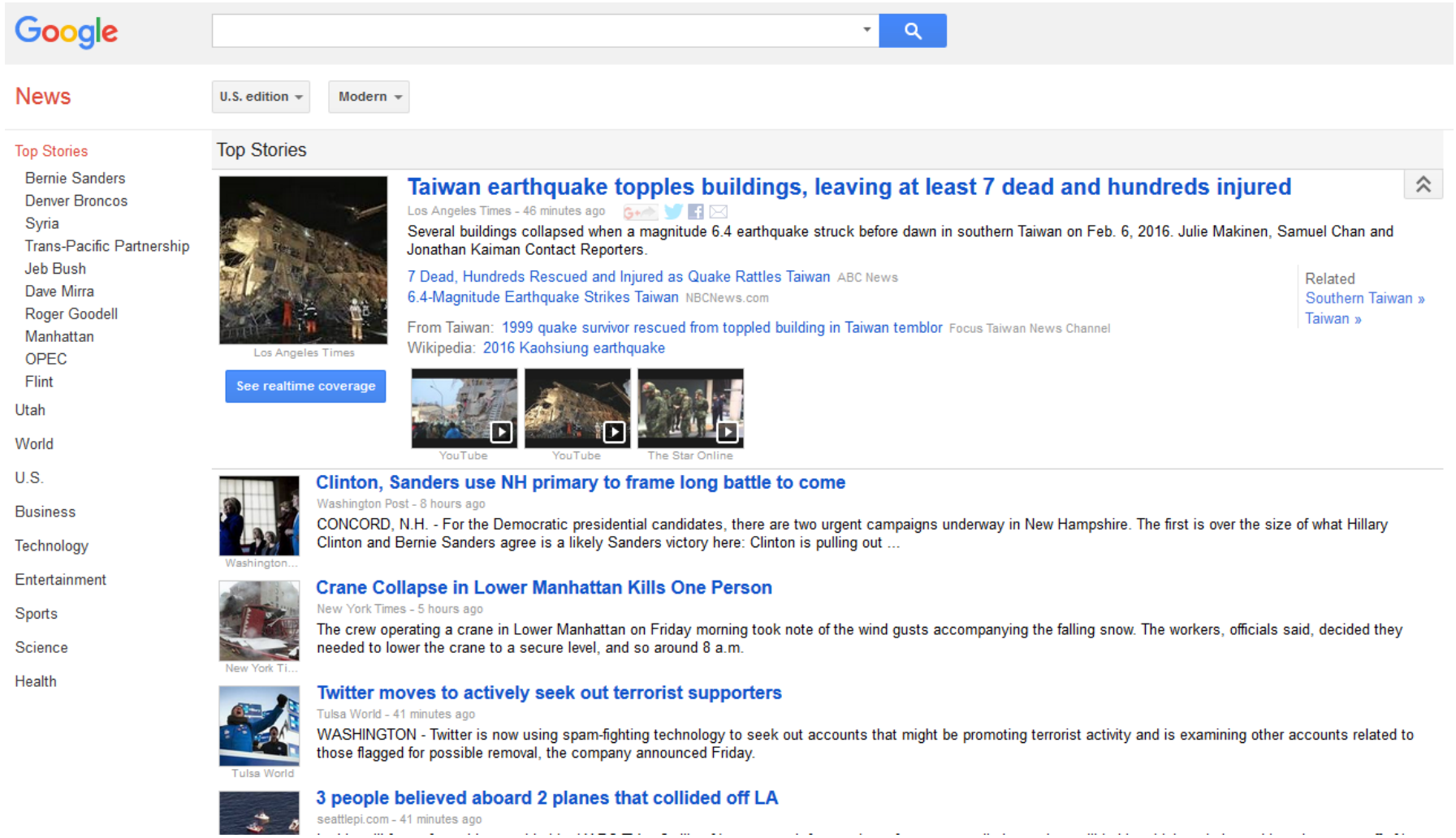
- Typical applications
  - As a stand-alone tool to get insight into data distribution
  - As a preprocessing step for other algorithms

# Clustering in IR

# Applications of clustering in IR

- **Whole corpus analysis/navigation**
  - **Better user interface: search without typing**
- For improving recall in search applications
  - Better search results
- For better navigation of search results
  - Effective “user recall” will be higher

# Google News: automatic clustering gives an effective news presentation metaphor



# Applications of clustering in IR

- Whole corpus analysis/navigation
  - Better user interface: search without typing
- **For improving recall in search applications**
  - **Better search results**
- For better navigation of search results
  - Effective “user recall” will be higher

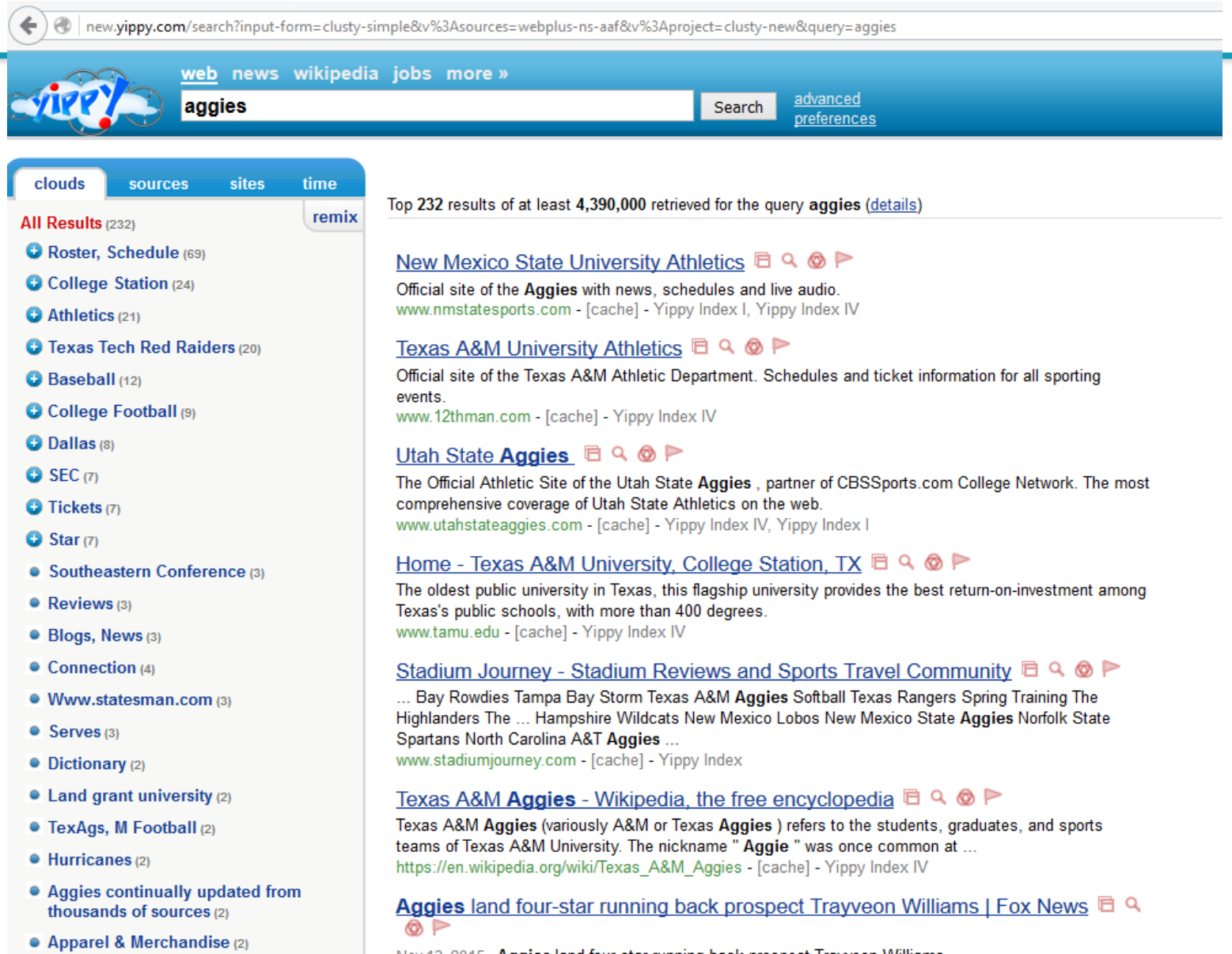
# For improving search recall

- *Cluster hypothesis* - Documents in the same cluster behave similarly with respect to relevance to information needs
- Therefore, to improve search recall:
  - Cluster docs in corpus a priori
  - When a query matches a doc *D*, also return other docs in the cluster containing *D*
- Hope if we do this: The query "car" will also return docs containing *automobile*
  - Because clustering grouped together docs containing *car* with those containing *automobile.*

# Applications of clustering in IR

- Whole corpus analysis/navigation
  - Better user interface: search without typing
- For improving recall in search applications
  - Better search results
- **For better navigation of search results**
  - **Effective "user recall" will be higher**

# Yippy.com (no longer active)



# Issues for clustering

- Representation for clustering
  - Document representation
    - Vector space? Normalization?
  - Need a notion of similarity/distance
- How many clusters?
  - Fixed a priori?
  - Completely data driven?
    - Avoid "trivial" clusters - too large or small
      - If a cluster's too large, then for navigation purposes you've wasted an extra user click without whittling down the set of documents much.

# Flat vs. Hierarchical Clustering

- Flat algorithms
  - Usually start with a random (partial) partitioning
  - Refine it iteratively
    - Main algorithm: K-means
- Hierarchical algorithms
  - Create a hierarchy
  - Bottom-up, agglomerative
    - Start with all documents belong to the same cluster.
    - Eventually each node forms a cluster on its own.
  - Top-down, divisive
    - Start with each document being a single cluster.
    - Eventually all documents belong to the same cluster.

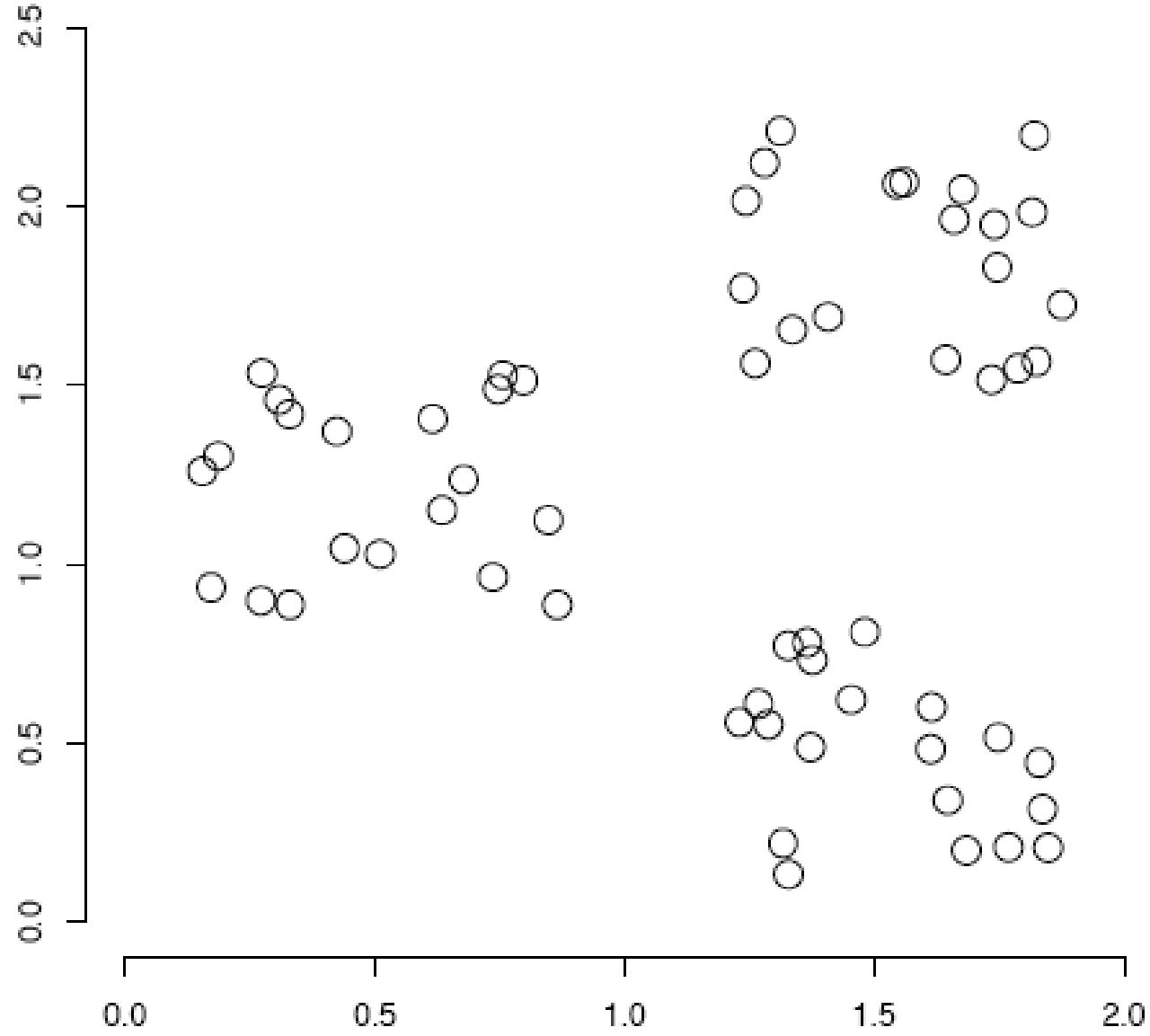
# Hard vs. soft clustering

- Hard clustering: Each document belongs to exactly one cluster
  - More common and easier to do
- Soft clustering: A document can belong to more than one cluster.
  - Makes more sense for applications like creating browsable hierarchies

# Flat algorithms

- Flat algorithms compute a partition of *N* documents into a set of *K* clusters.
- Given: a set of documents and the number *K*
- Find: a partition into *K* clusters that optimizes the chosen partitioning criterion
- Global optimization: exhaustively enumerate partitions, pick optimal one
  - Not tractable
- Effective heuristic method: *K*-means algorithm

# K-means


2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0

# K-means (in one slide!)

Input is **k** (the number of clusters), **data**

**points** in Euclidean space

0. Initialize clusters by picking one point per

cluster

Loop:

1. Place each point in the cluster whose current centroid is nearest
2. Find the new centroid for each cluster

# K-means

- Objective/partitioning criterion: minimize the average squared difference from the centroid
- Assumes documents are real-valued vectors
- Clusters based on *centroids* (aka the *center of gravity* or mean) of points in a cluster, $\omega$:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

- We try to find the minimum average iterating two steps:
  - reassignment: assign each vector to its closest centroid
  - recomputation: recompute each centroid as the average of the vectors that were assigned to it in reassignment

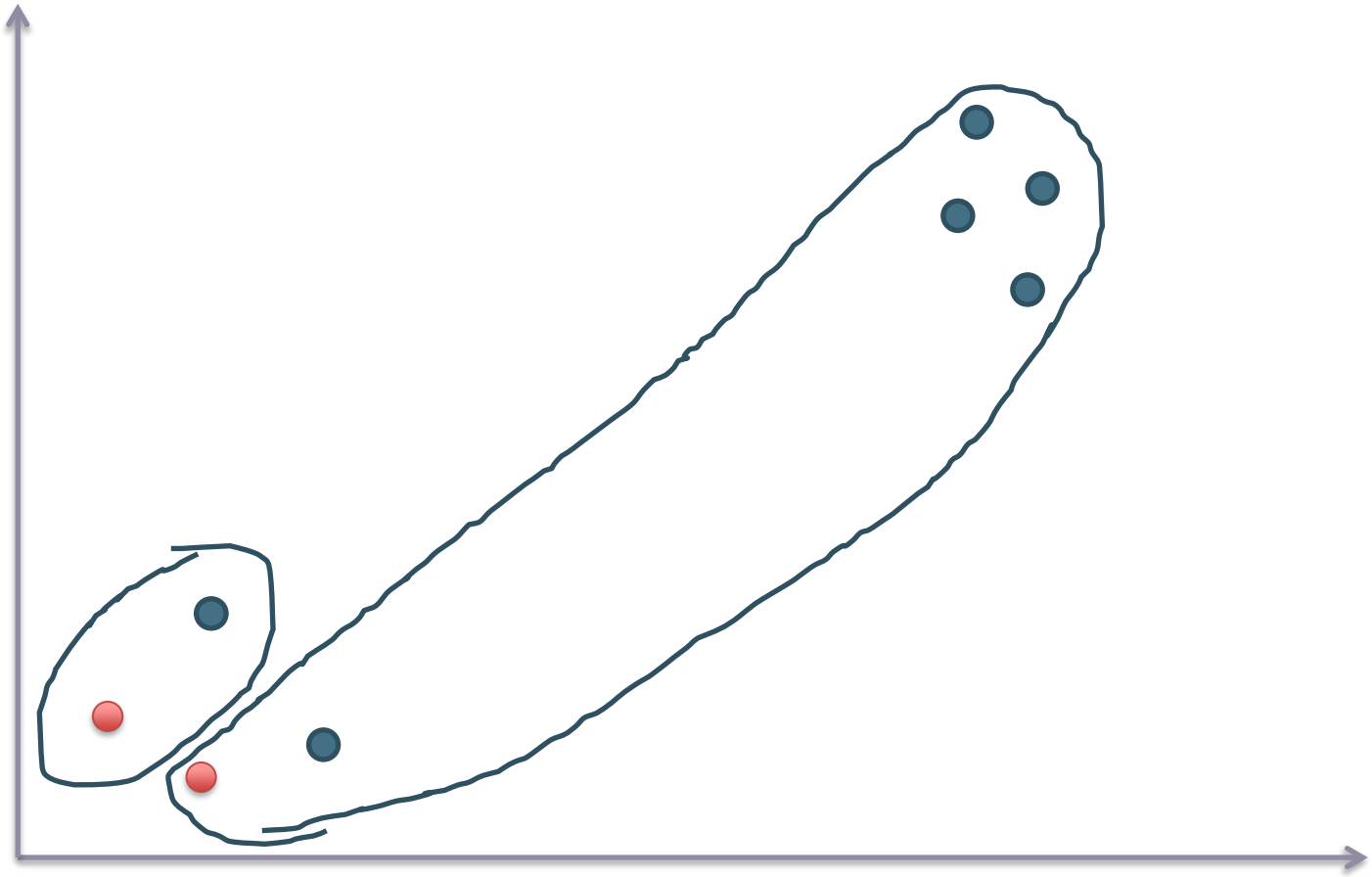
$K$-MEANS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$

1 $(\vec{s}_1, \vec{s}_2, \ldots, \vec{s}_K) \leftarrow$ SELECTRANDOMSEEDS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$
2 **for** $k \leftarrow 1$ **to** $K$
3 **do** $\vec{\mu}_k \leftarrow \vec{s}_k$
4 **while** stopping criterion has not been met
5 **do for** $k \leftarrow 1$ **to** $K$
6 **do** $\omega_k \leftarrow \{\}$
7 **for** $n \leftarrow 1$ **to** $N$
8 **do** $j \leftarrow \arg\min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$
9 $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$ *(reassignment of vectors)*
10 **for** $k \leftarrow 1$ **to** $K$
11 **do** $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$ *(recomputation of centroids)*
12 **return** $\{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$

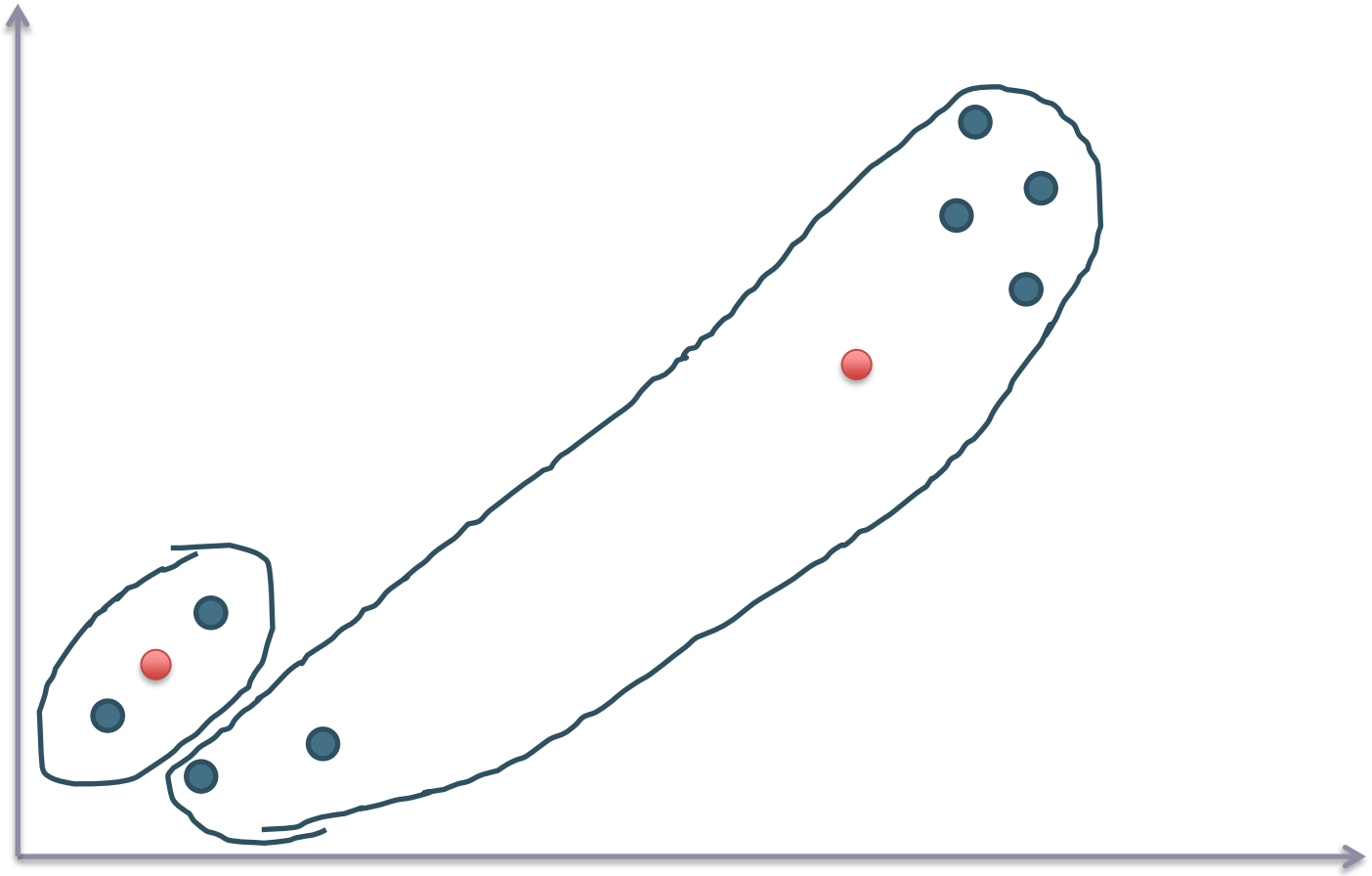
# K-Means Clustering Example

# K-Means Clustering Example

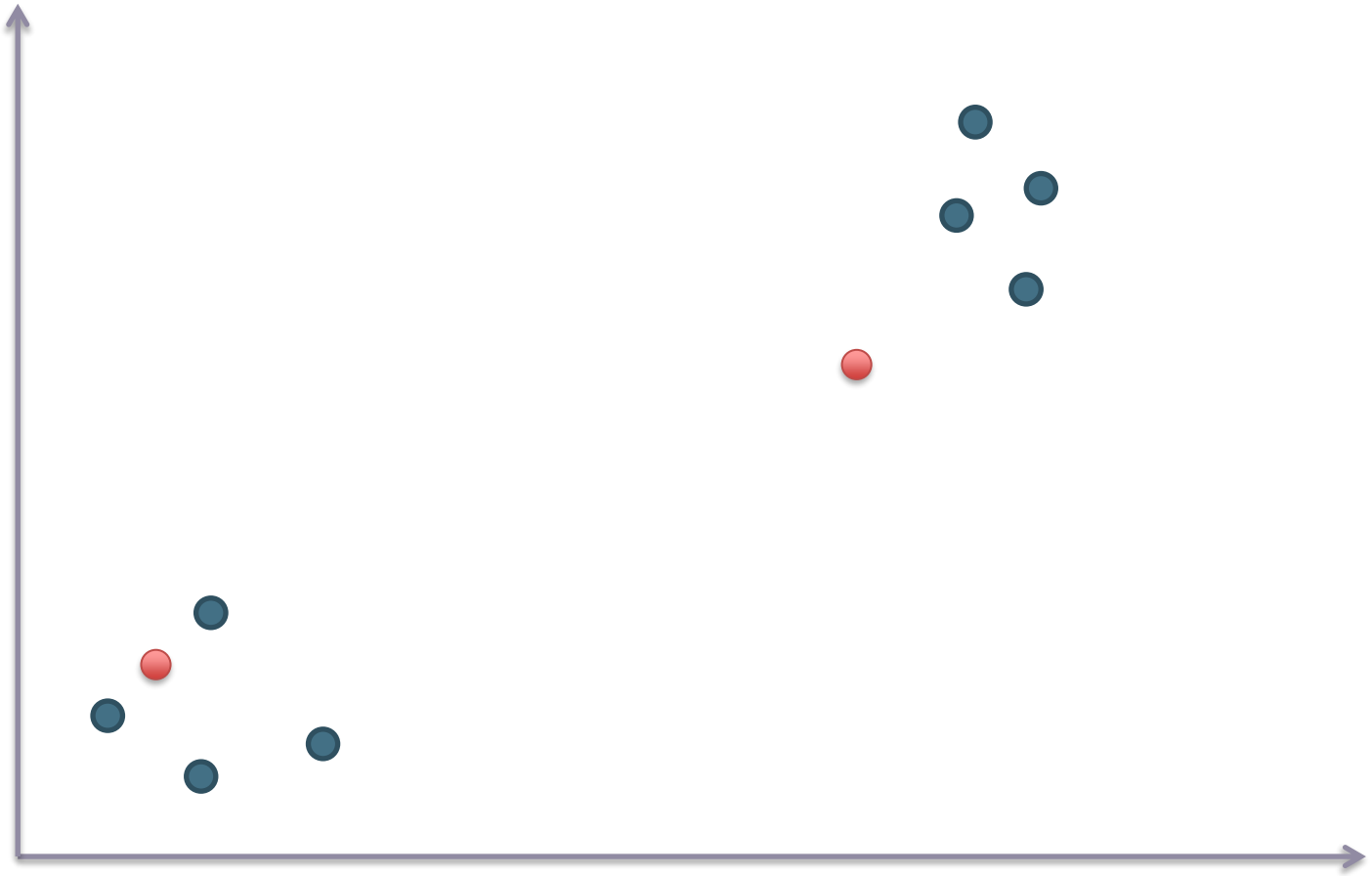
# K-Means Clustering Example



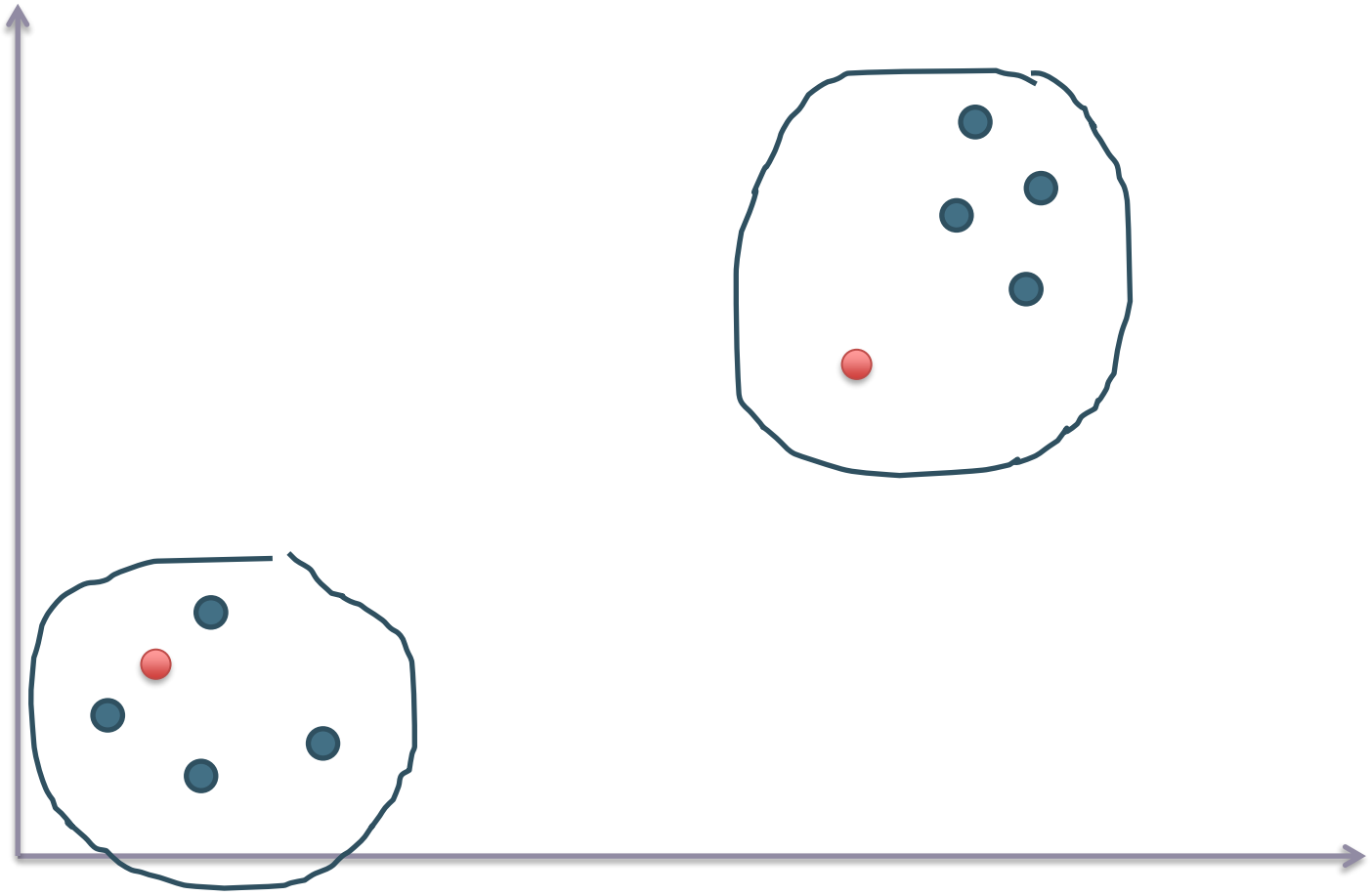
# K-Means Clustering Example



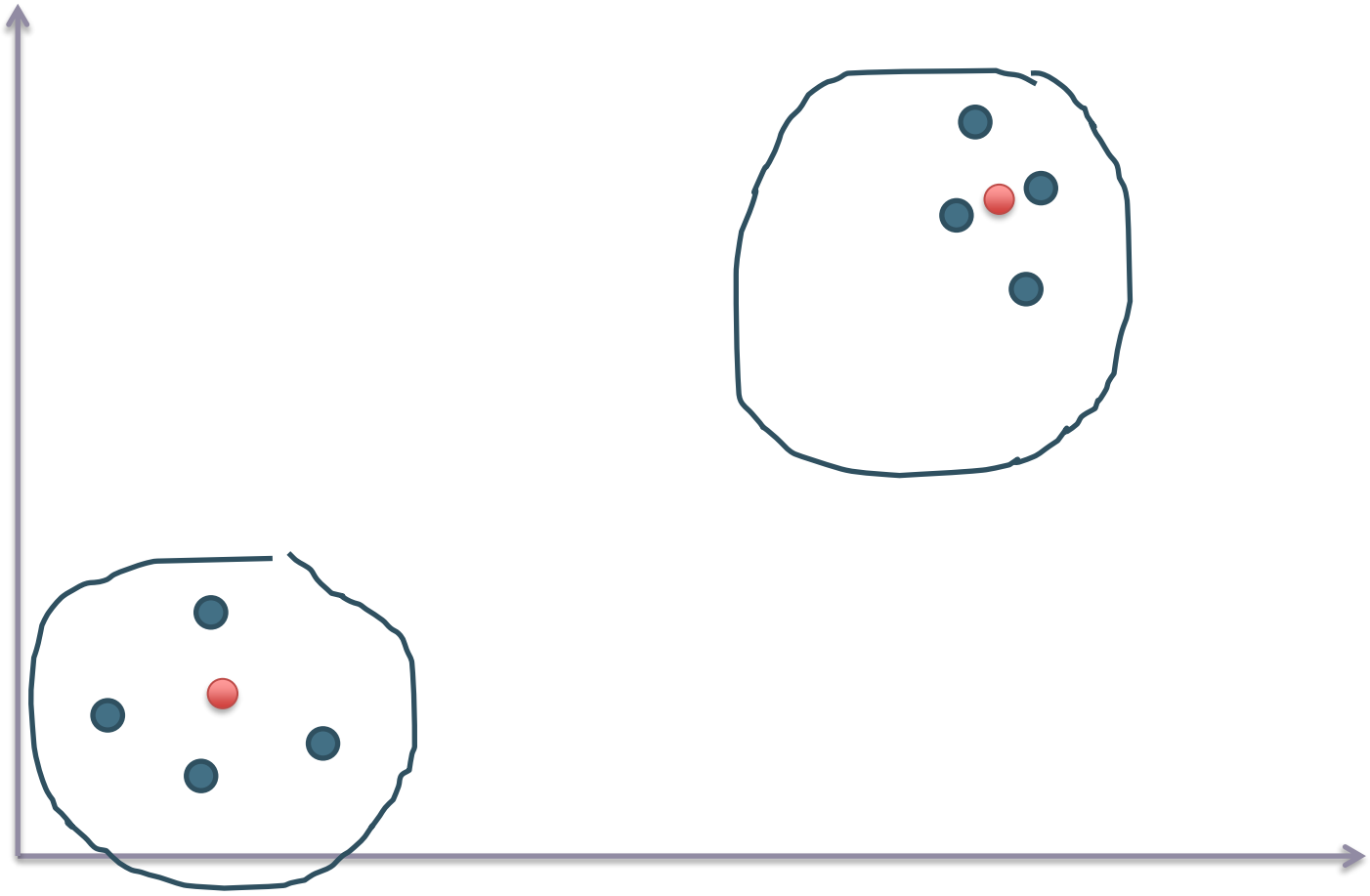
# K-Means Clustering Example



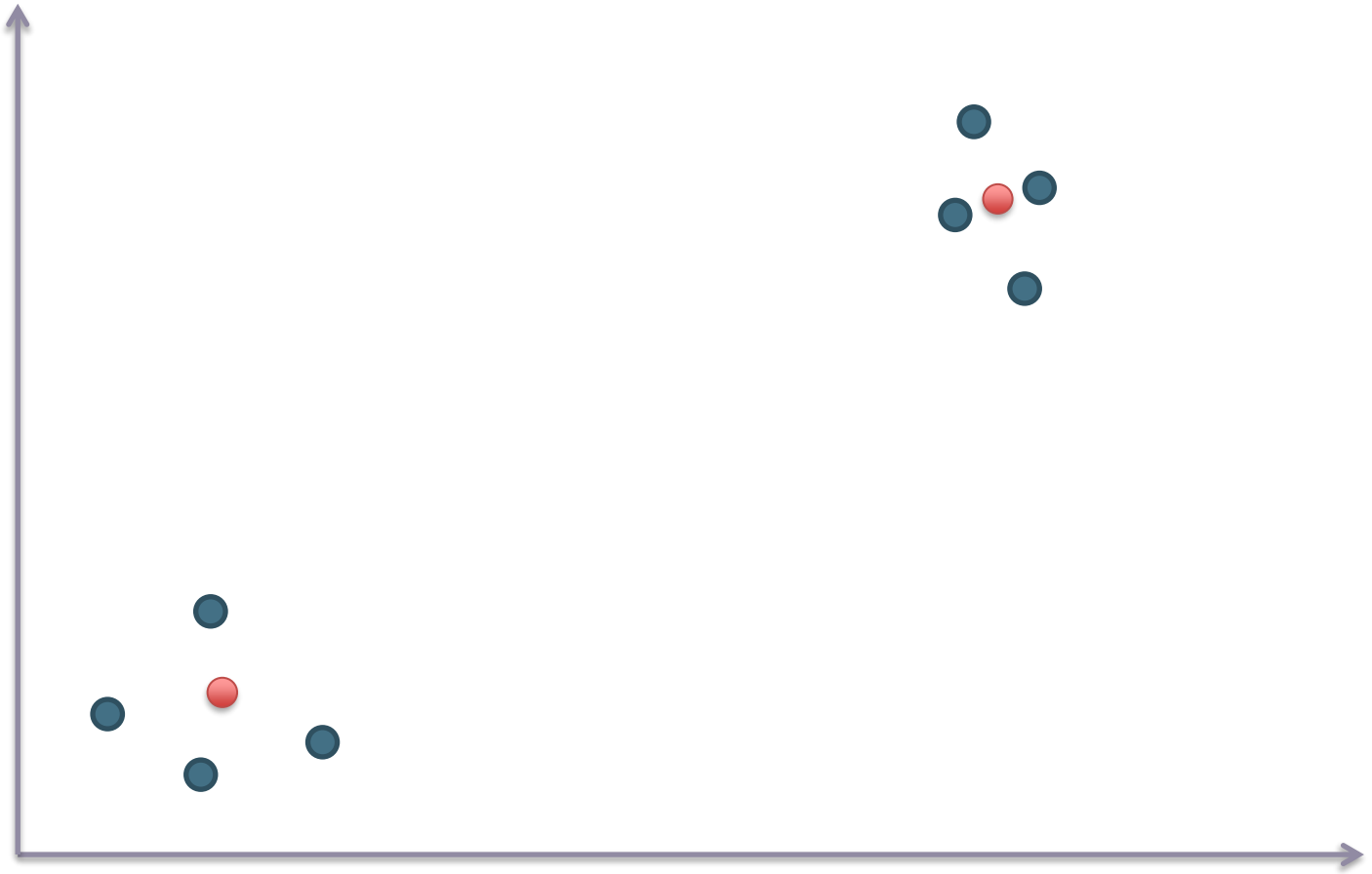
# K-Means Clustering Example



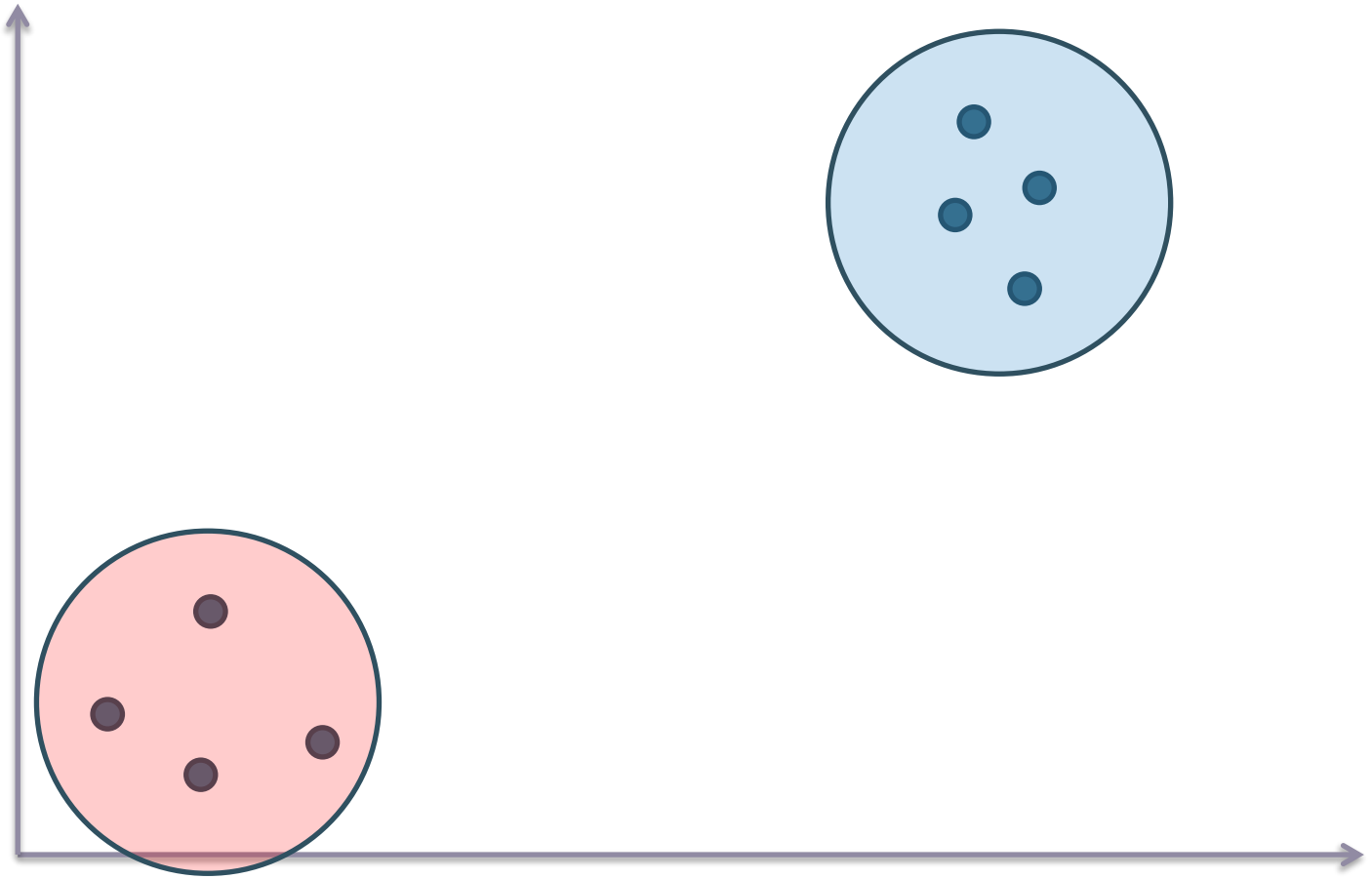
# K-Means Clustering Example



# K-Means Clustering Example

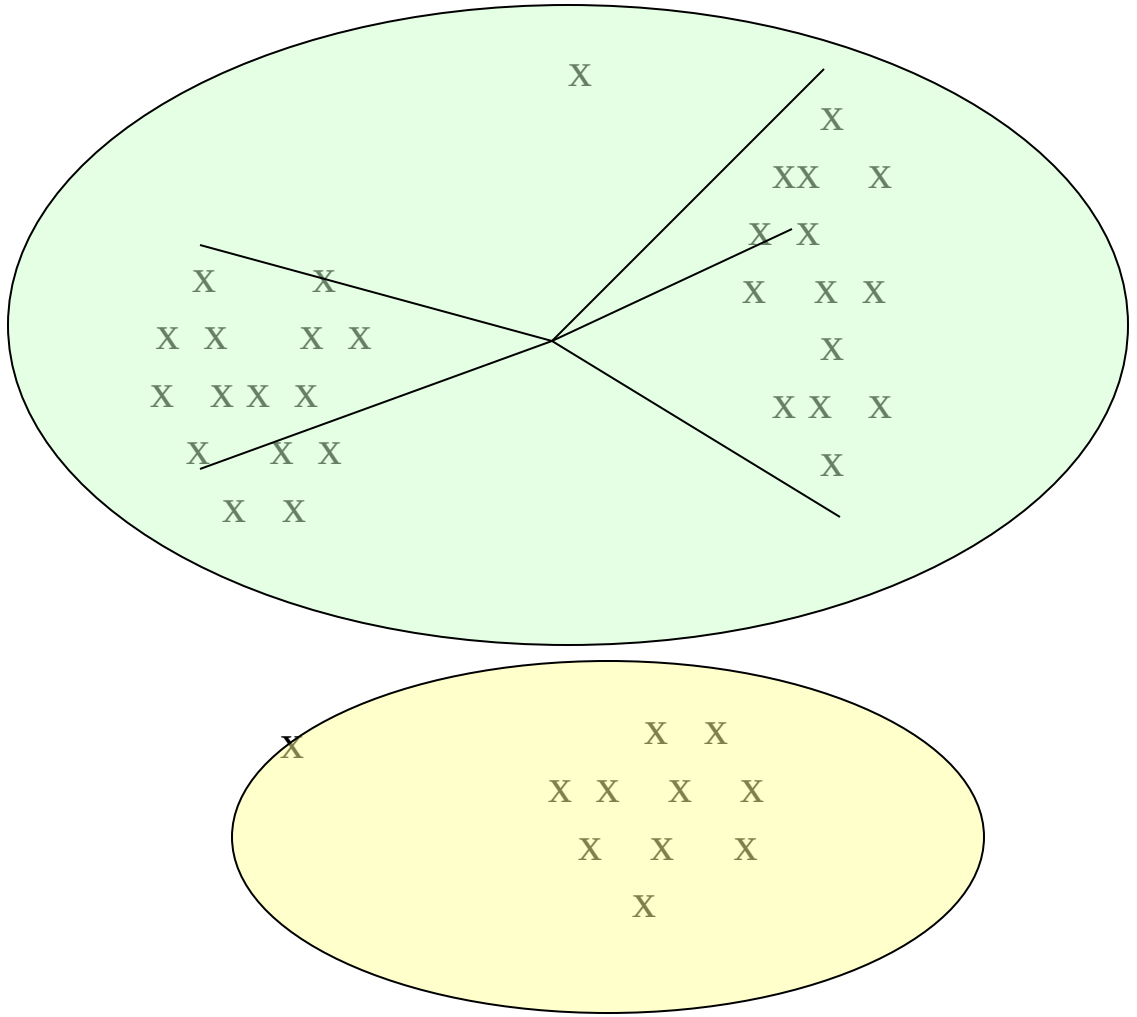


# K-Means Clustering Example



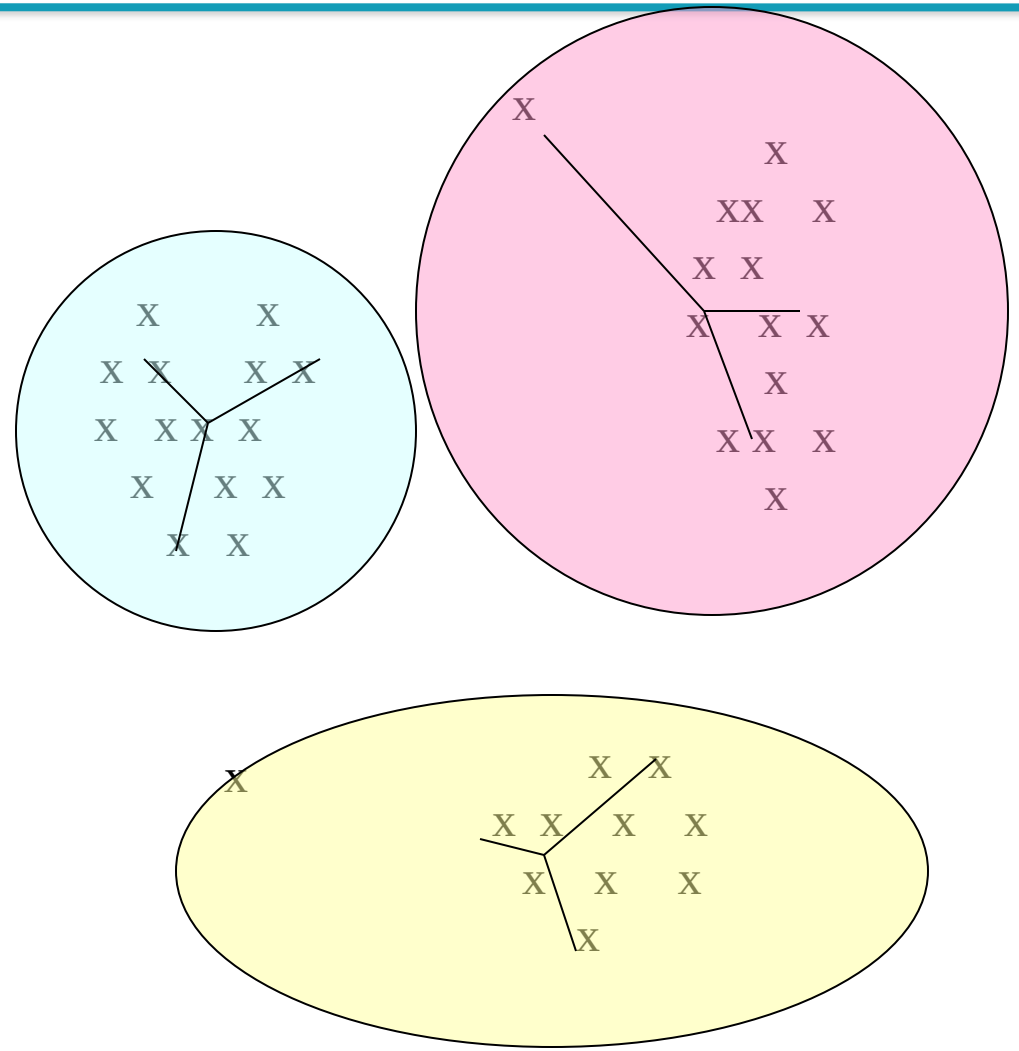
# Example: Picking *k*

**Too few;**
many long
distances
to centroid.



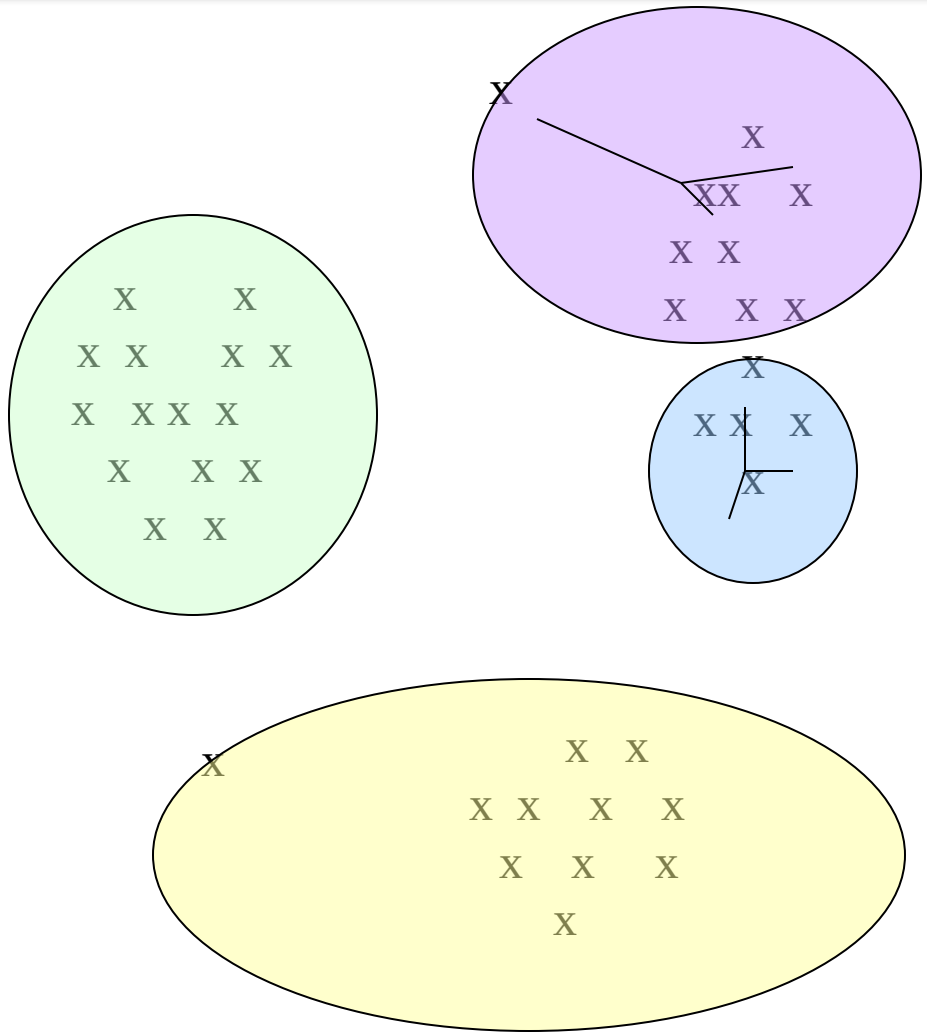
# Example: Picking *k*

**Just right;**
distances
rather short.



# Example: Picking *k*

**Too many;**
little improvement
in average
distance.



# Convergence of K Means

- K-means converges to a fixed point in a finite number of iterations.
- Proof:
  - The sum of squared distances (RSS) decreases during reassignment.
  - (because each vector is moved to a closer centroid)
  - RSS decreases during recomputation.
  - There is only a finite number of clusterings
  - Thus: We must reach a fixed point.
- But we don’t know how long convergence will take!
- If we don’t care about a few docs switching back and forth, then convergence is usually fast (< 10-20 iterations).

# Recomputation decreases average distance

- RSS = residual sum of squares (the "goodness" measure G)

$$\mathrm{RSS}_k(\vec{v}) = \sum_{\vec{x}\in\omega_k} \|\vec{v}-\vec{x}\|^2 = \sum_{\vec{x}\in\omega_k}\sum_{m=1}^{M}(v_m - x_m)^2$$

$$\frac{\partial \mathrm{RSS}_k(\vec{v})}{\partial v_m} = \sum_{\vec{x}\in\omega_k} 2(v_m - x_m) = 0$$

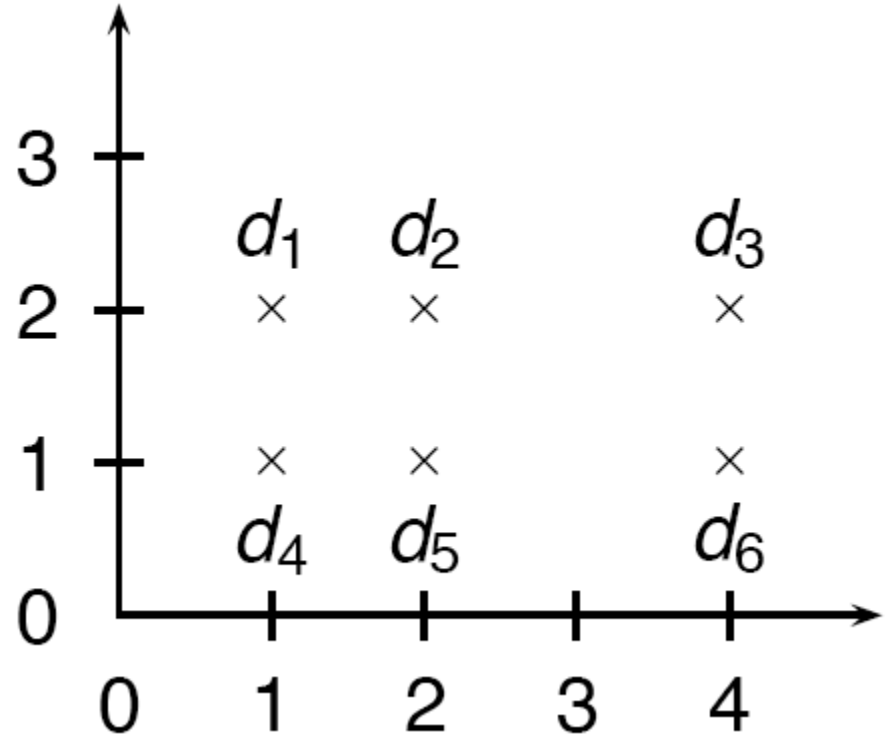
$$v_m = \frac{1}{|\omega_k|}\sum_{\vec{x}\in\omega_k} x_m$$

- The last line is the componentwise definition of the centroid! We minimize RSSk when the old centroid is replaced with the new centroid. RSS, the sum of the RSSk , must then also decrease during recomputation.

# Optimality of *K*-means

- Convergence does not mean that we converge to the optimal clustering!
- This is the great weakness of K-means.
- If we start with a bad set of seeds, the resulting clustering can be horrible.

# Example of suboptimal clustering!!!!




- What is the optimal clustering for K=2?
- What happens when our seeds are: d2, d5?

# Initialization of *K*-means

- Results can vary based on random seed selection.
- Some seeds can result in poor convergence rate, or convergence to sub-optimal clusterings.
  - Select good seeds using a heuristic (e.g., doc least similar to any existing mean)
  - Try out multiple starting points
  - Initialize with the results of another method.

# How many clusters?

# Hmm...

- **Either: Number of clusters K is given.**
  - Then partition into K clusters
  - K might be given because there is some external constraint. Example: You cannot show more than 10–20 clusters on a screen.

- **Or: Finding the "right" number of clusters is part of the problem.**
  - Given docs, find K for which an optimum is reached.
  - How to define "optimum"?
  - Why can't we use RSS or average distance from centroid?

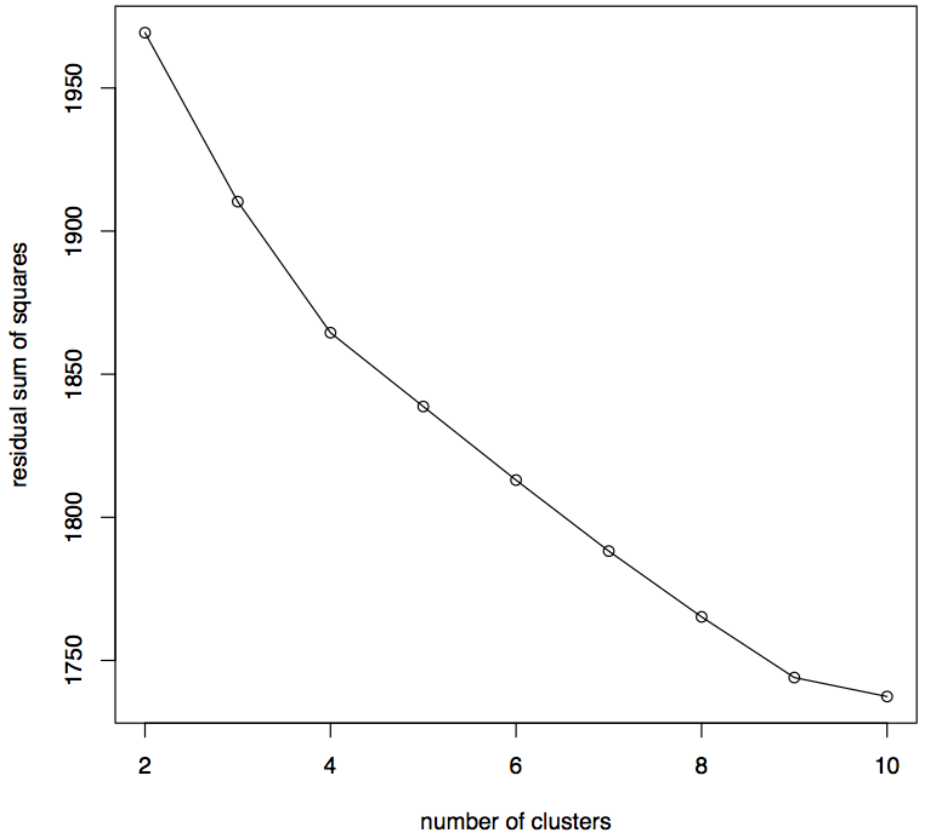
# Simple objective function for *K*

- Basic idea:
  - Start with 1 cluster (*K* = 1)
  - Keep adding clusters (= keep increasing *K*)
  - Add a penalty for each new cluster
- Trade off cluster penalties against average squared distance from centroid
- Choose the value of *K* with the best tradeoff

# Simple objective function for *K*

- Given a clustering, define the cost for a document as (squared) distance to centroid
- Define total distortion RSS(K) as sum of all individual document costs (corresponds to average distance)
- Then: penalize each cluster with a cost λ
- Thus for a clustering with *K* clusters, total cluster penalty is *K*λ
- Define the total cost of a clustering as distortion plus total cluster penalty: RSS(K) + *K*λ
- Select K that minimizes (RSS(K) + *K*λ)
- Still need to determine good value for λ . . .

# Finding the “knee” in the curve




Pick the number of clusters where curve “flattens”. Here: 4 or 9.

# Labeling

# Major issue - labeling

- After clustering algorithm finds clusters - how can they be useful to the end user?
- Need simple label for each cluster
  - In search results, say “Animal” or “Car” in the ***jaguar*** example.
  - In topic trees (Yahoo), need navigational cues.
    - Often done by hand, a posteriori.

# Ideas?

- Use metadata like Titles
- Use the medoid (document) itself – Title
- Top-terms (most frequent)
  - Stop-words, duplicates
- Most distinguishing terms (in my cluster, not in your cluster)
  - Measure Mutual Information

# How to Label Clusters

- Show titles of typical documents
  - Titles are easy to scan
  - Authors create them for quick scanning!
  - But you can only show a few titles which may not fully represent cluster
- Show words/phrases prominent in cluster
  - More likely to fully represent cluster
  - Use distinguishing words/phrases
    - Differential labeling
  - But harder to scan

# Labeling

- Common heuristics - list 5-10 most frequent terms in the centroid vector.
  - Drop stop-words; stem.
- Differential labeling by frequent terms
  - Within a collection “Computers”, clusters all have the word ***computer*** as frequent term.
  - Discriminant analysis of centroids.
- Perhaps better: distinctive noun phrase

# Cluster labeling: example

| | # docs | labeling method | | |
|---|---|---|---|---|
| | | centroid | mutual information | title |
| 4 | 622 | oil plant mexico production crude **power 000 refinery gas** bpd | plant oil production **barrels** crude bpd mexico **dolly capacity petroleum** | MEXICO: Hurricane Dolly heads for Mexico coast |
| 9 | 1017 | police security **russian** people military peace killed told **grozny court** | police killed military security peace told **troops forces rebels** people | RUSSIA: Russia's Lebed meets rebel chief in Chechnya |
| 10 | 1259 | 00 000 tonnes traders futures wheat prices **cents september** tonne | **delivery** traders futures tonne tonnes **desk** wheat prices 000 00 | USA: Export Business - Grain/oilseeds complex |

- Three methods: most prominent terms in centroid, differential labeling using MI, title of doc closest to centroid
- Any feature selection method can also be used for labeling

# Final word

- In clustering, clusters are inferred from the data without human input (unsupervised learning)

- However, in practice, it's a bit less clear: there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, . . .

# Evaluation

# What is a good clustering?

- **Internal criteria**
  - Example of an internal criterion: RSS in K-means
- But an internal criterion often does not evaluate the actual utility of a clustering in the application
- **Alternative: External criteria**
  - Evaluate with respect to human-defined classification
  - Require the ground truth

# External criteria for clustering quality

- Based on a gold standard data set, e.g., the Reuters collection
- Goal: Clustering should reproduce the classes in the gold standard
- (But we only want to reproduce how documents are divided into groups, not the class labels.)
- First measure for how well we were able to reproduce the classes: purity

# External criterion: Purity

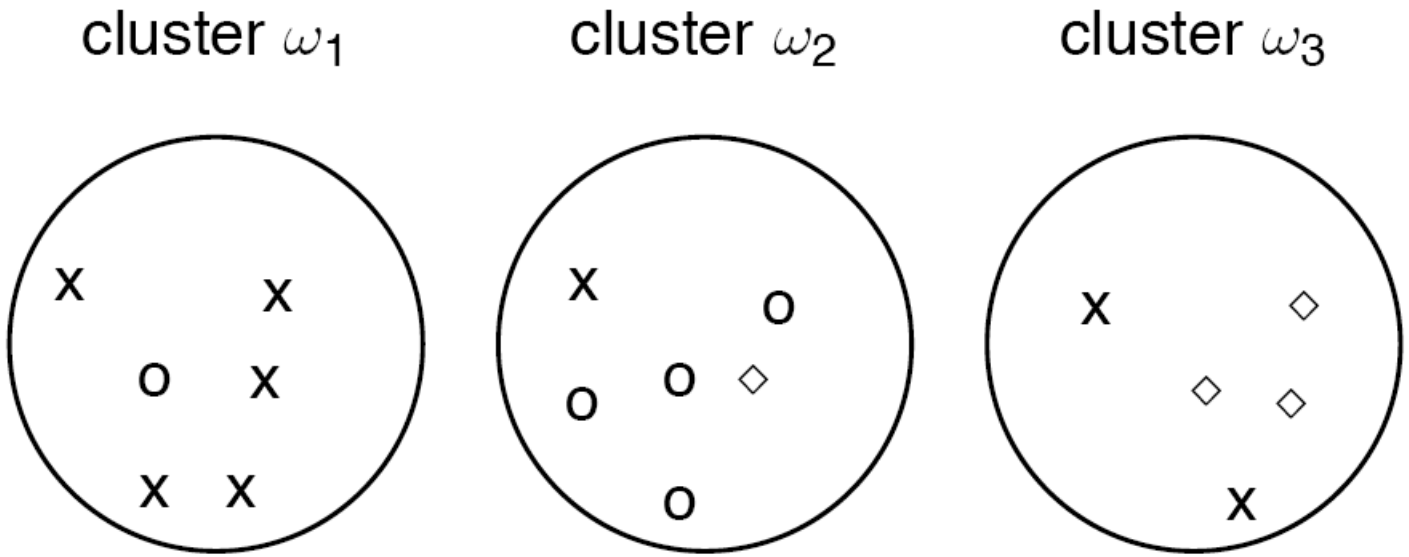
$$\text{purity}(\Omega, \Gamma) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

$\Omega = \{\omega_1, \omega_2, \ldots, \omega_K\}$ is the set of clusters and
$\Gamma = \{c_1, c_2, \ldots, c_J\}$ is the set of classes.

For each cluster $\omega_k$ : find class $c_j$ with most members $n_{kj}$ in cluster

Sum all $n_{kj}$ and divide by total number of points

# Example

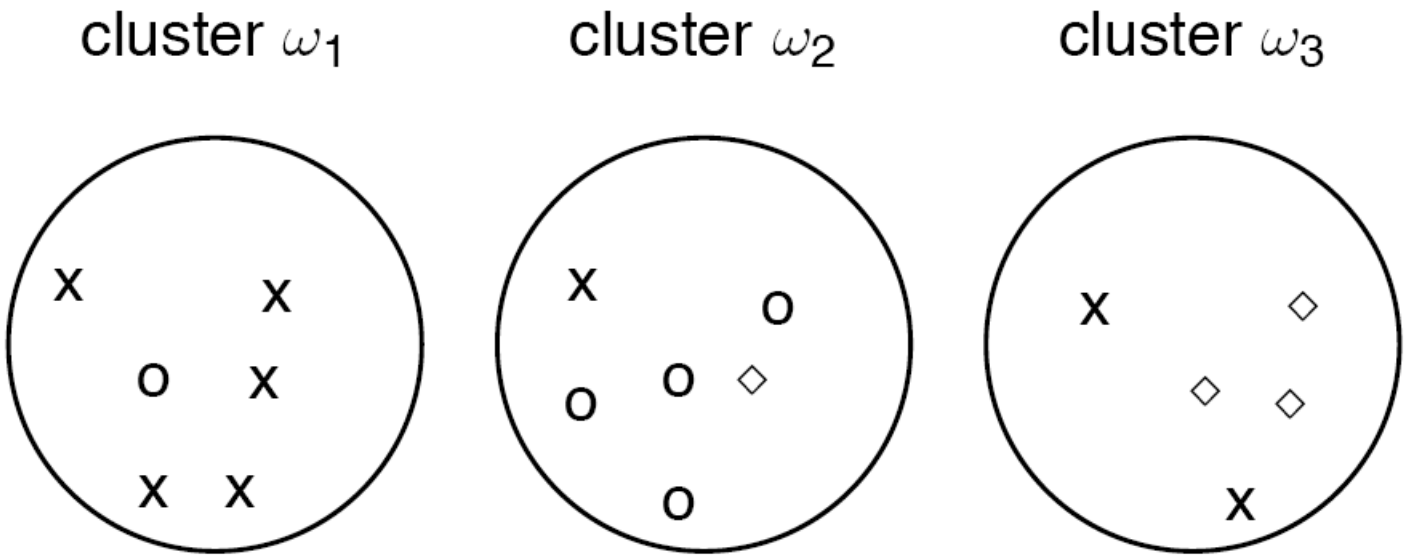



$$\text{good\_docs}(\omega_1) = \max(5,1,0) = 5$$
$$\text{good\_docs}(\omega_2) = \max(1,4,1) = 4$$
$$\text{good\_docs}(\omega_3) = \max(2,0,3) = 3$$
$$\text{purity}(\Omega) = 1/17 \cdot (5+4+3) = 12/17$$

# Three other external evaluation measures

- Rand Index
- Normalized mutual information (NMI)
  - How much information does the clustering contain about the classification?
  - Singleton clusters (number of clusters = number of docs) have maximum MI
  - Therefore: normalize by entropy of clusters and classes
- F measure
  - Like Rand, but “precision” and “recall” can be weighted

# Evaluation results




| purity | NMI | RI | $F_5$ |
|---|---|---|---|
| 0.71 | 0.34 | 0.68 | 0.46 |

- All four measures range from 0 (really bad clustering) to 1 (perfect clustering).