Information Retrieval

CS 547/DS 547

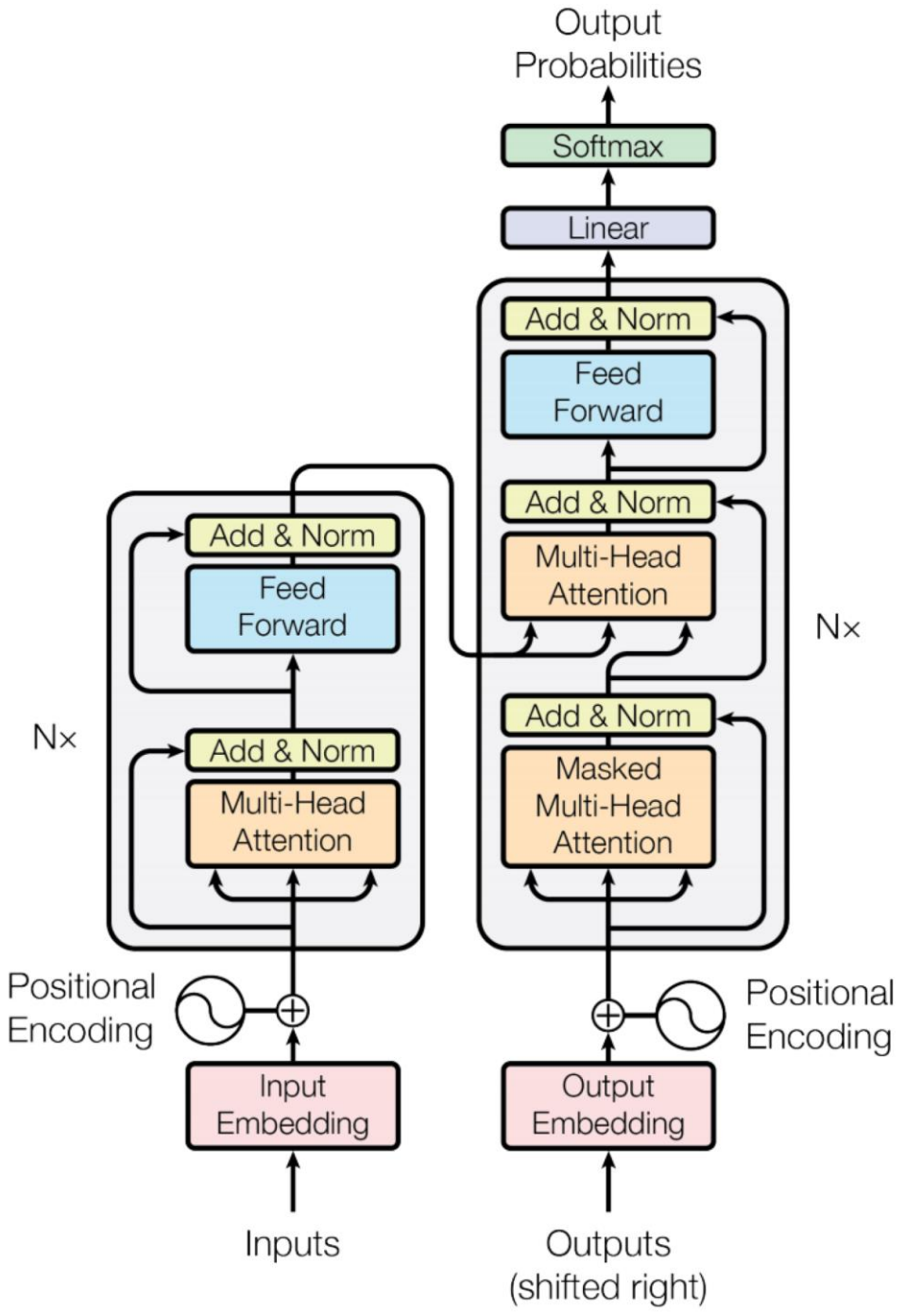
Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

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

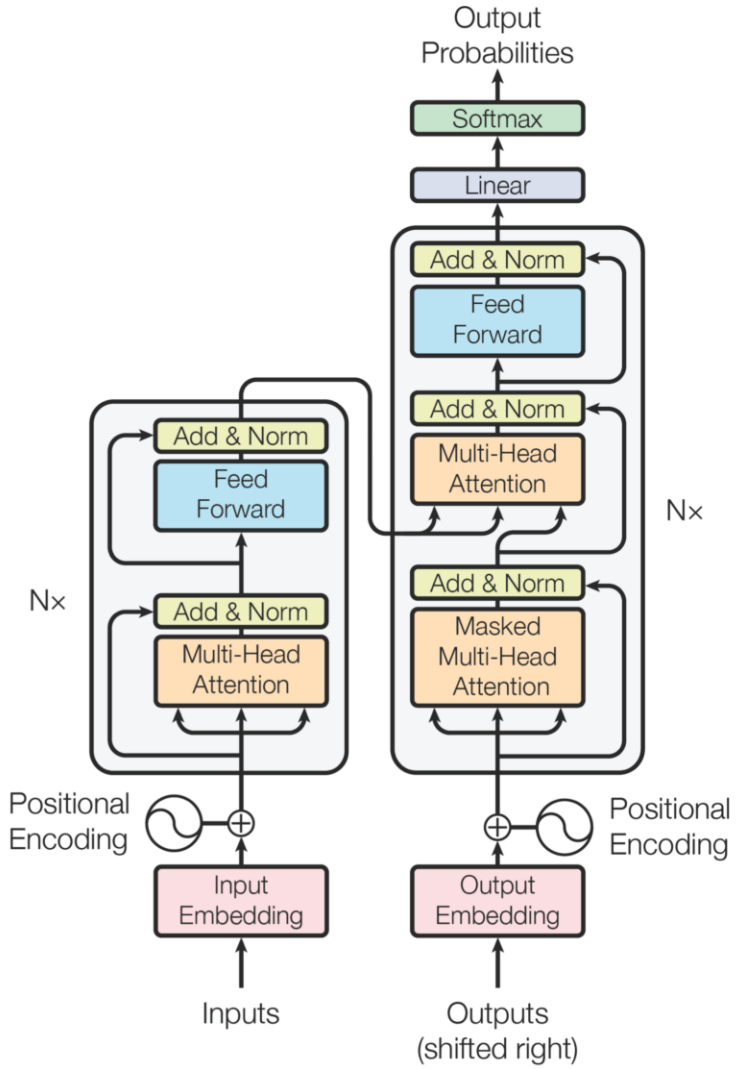


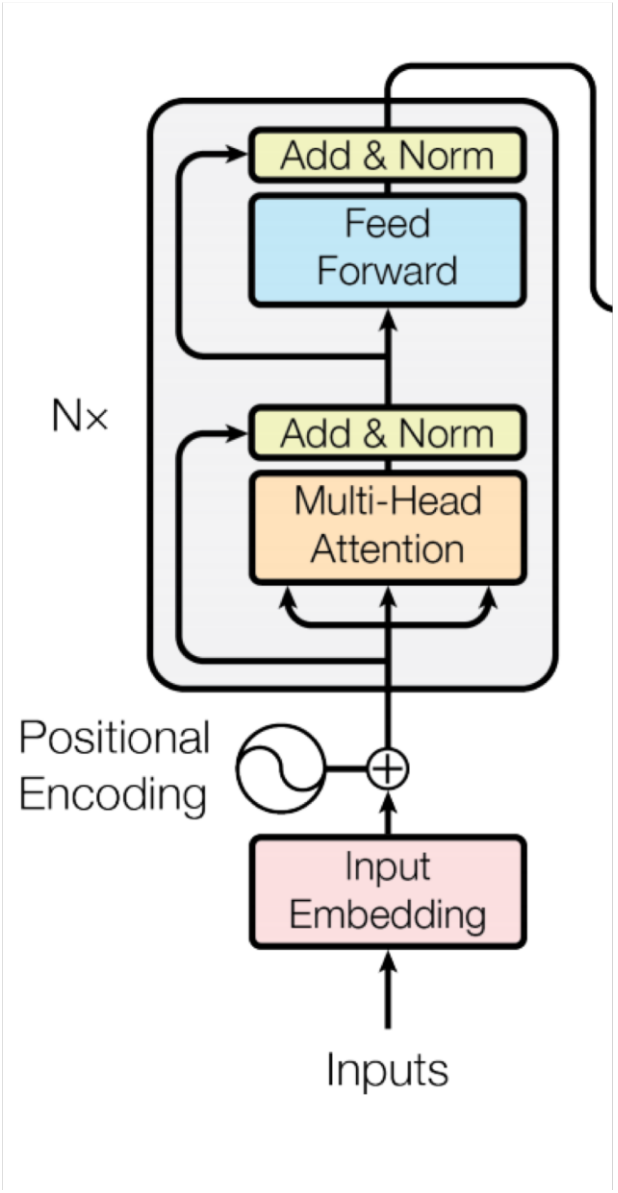
Figure 1: The Transformer - model architecture.

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

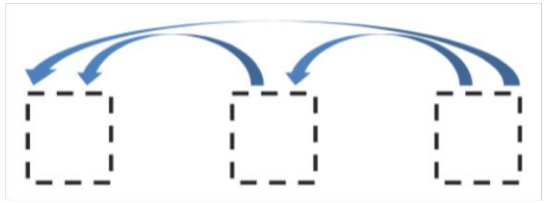
- For encoder, at each block, we use the same Q, K and V
  from the previous layer

- Blocks are repeated 6 times (in vertical stack)

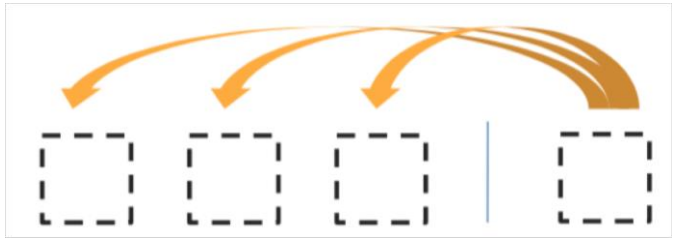


Transformer Decoder

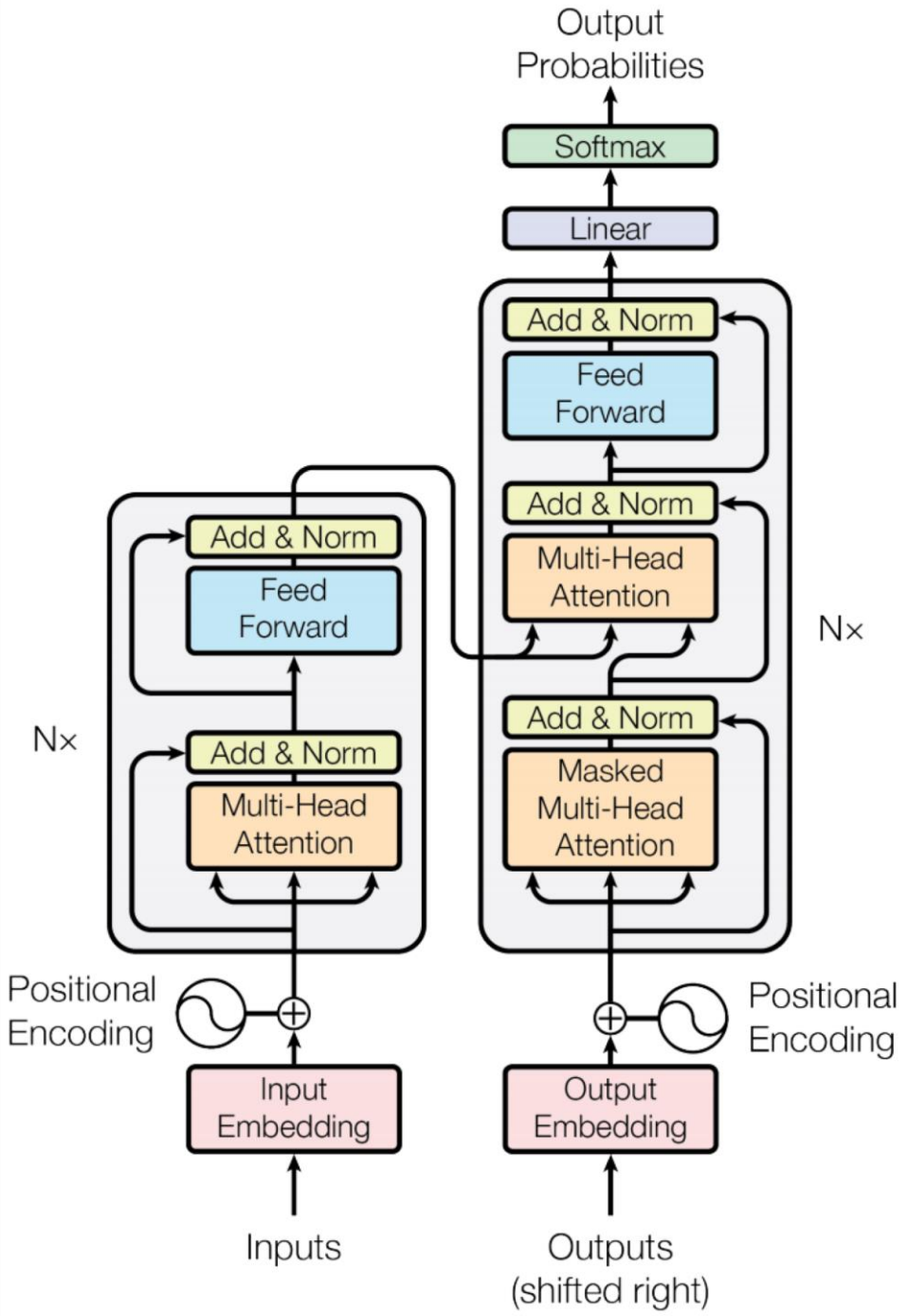
- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



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

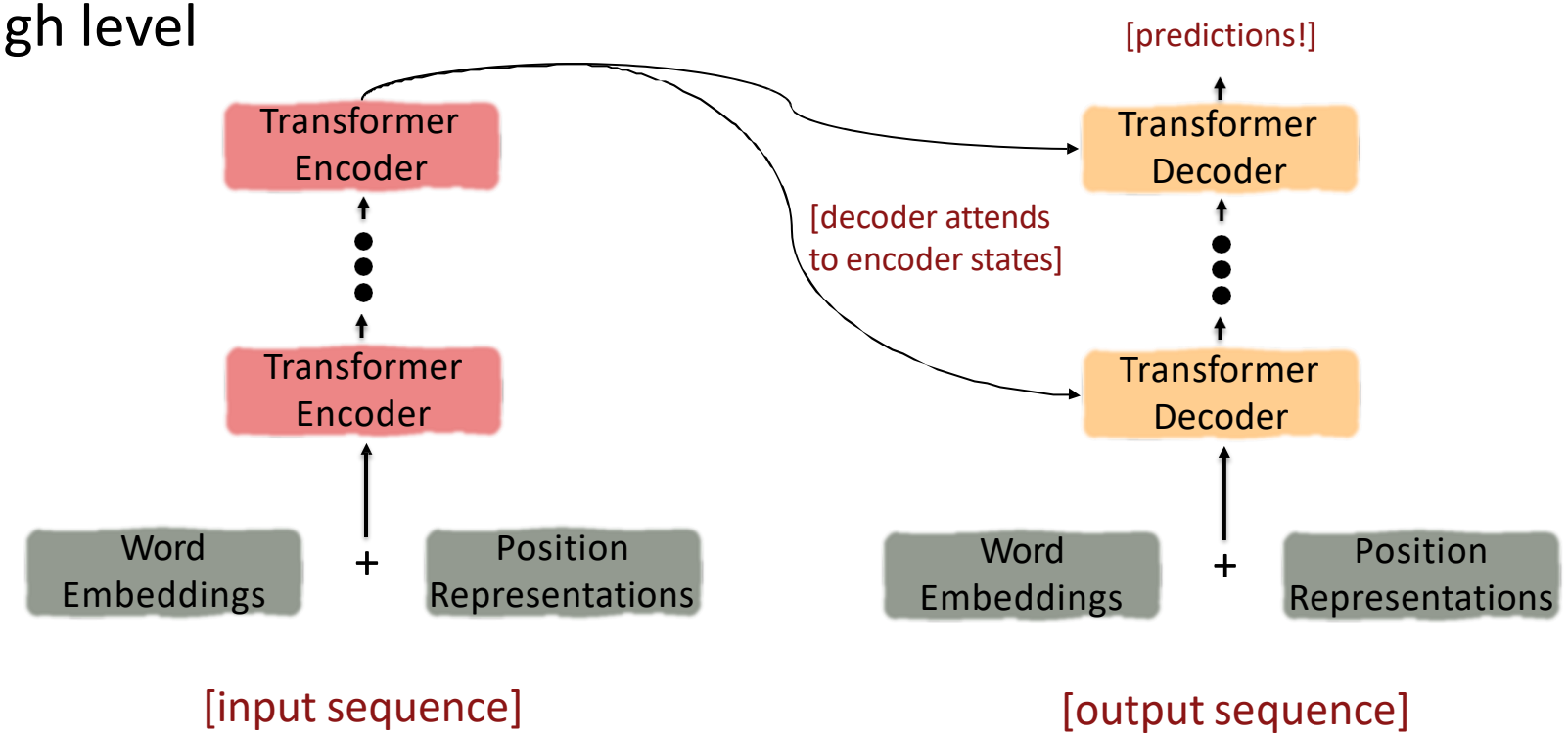


Blocks repeated 6 times also



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

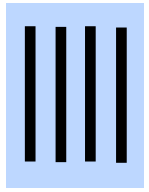
- When we have multiple queries q, we stack them in a matrix Q:

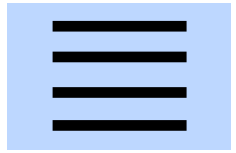
$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

- 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.

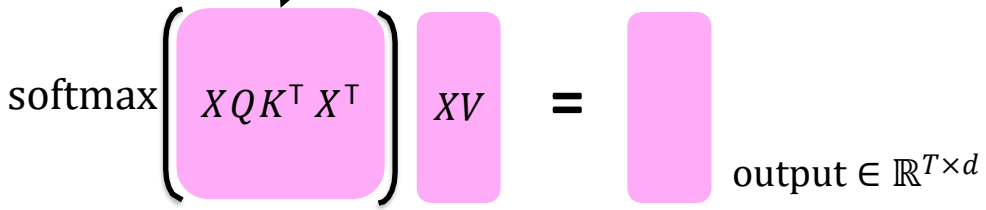
- These matrices allow *different aspects* of the x vectors to be used/emphasized in each of the three roles.

The Transformer Encoder:
Key-Query-Value Attention

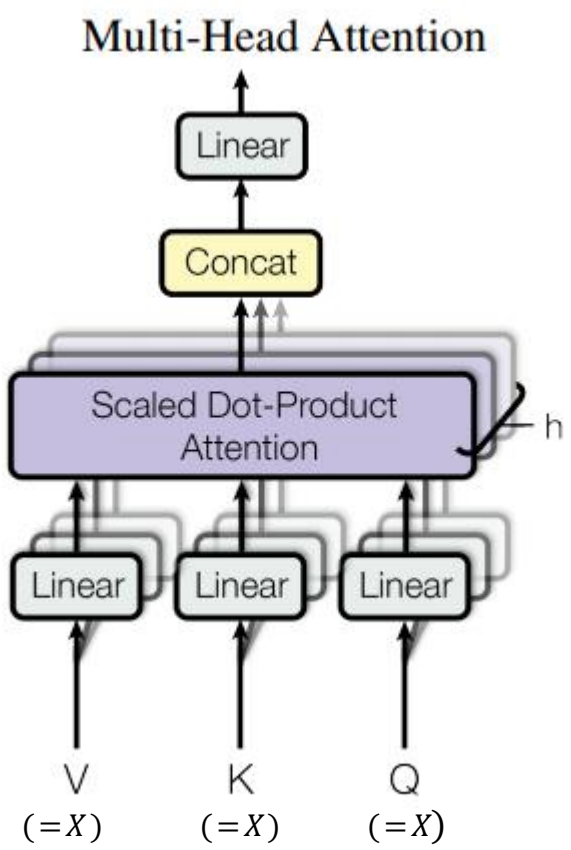
- Let's look at how key-query-value attention is computed, in matrices.
  - Let $X = [x_1; \ldots; x_T] \in \mathbb{R}^{T \times d}$ be the concatenation of input vectors.
  - First, note that $XK \in \mathbb{R}^{T \times d}$, $XQ \in \mathbb{R}^{T \times d}$, $XV \in \mathbb{R}^{T \times d}$.
  - The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

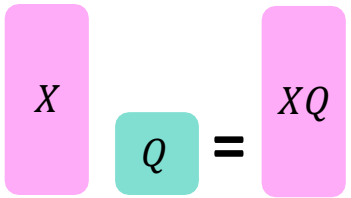


- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_P,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

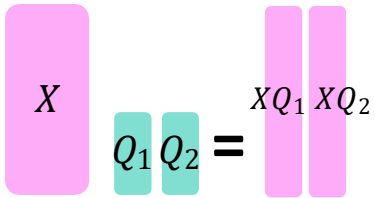
The Transformer Encoder:
Multi-headed attention

- 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)



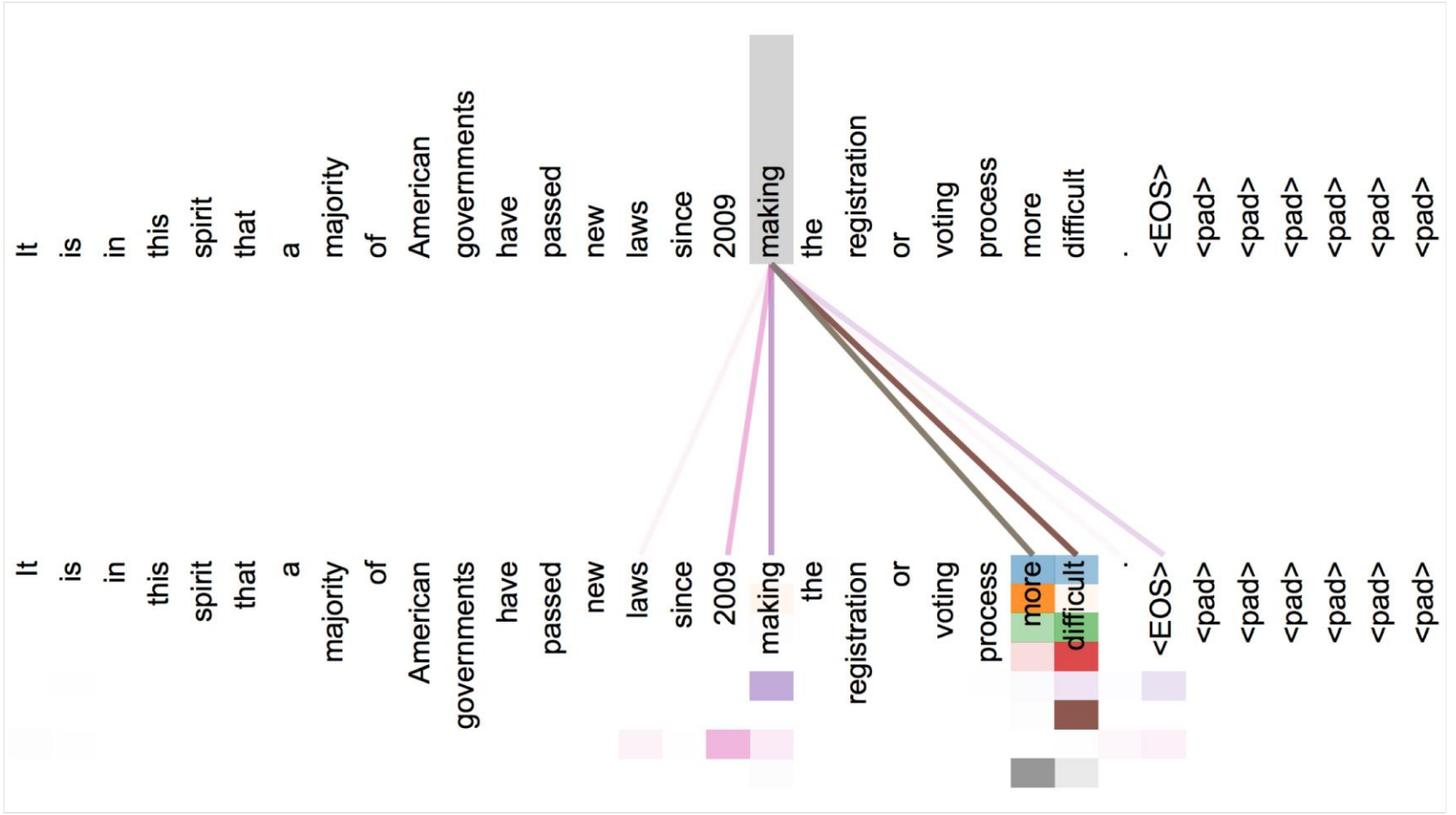
Multi-head attention
(just two heads here)



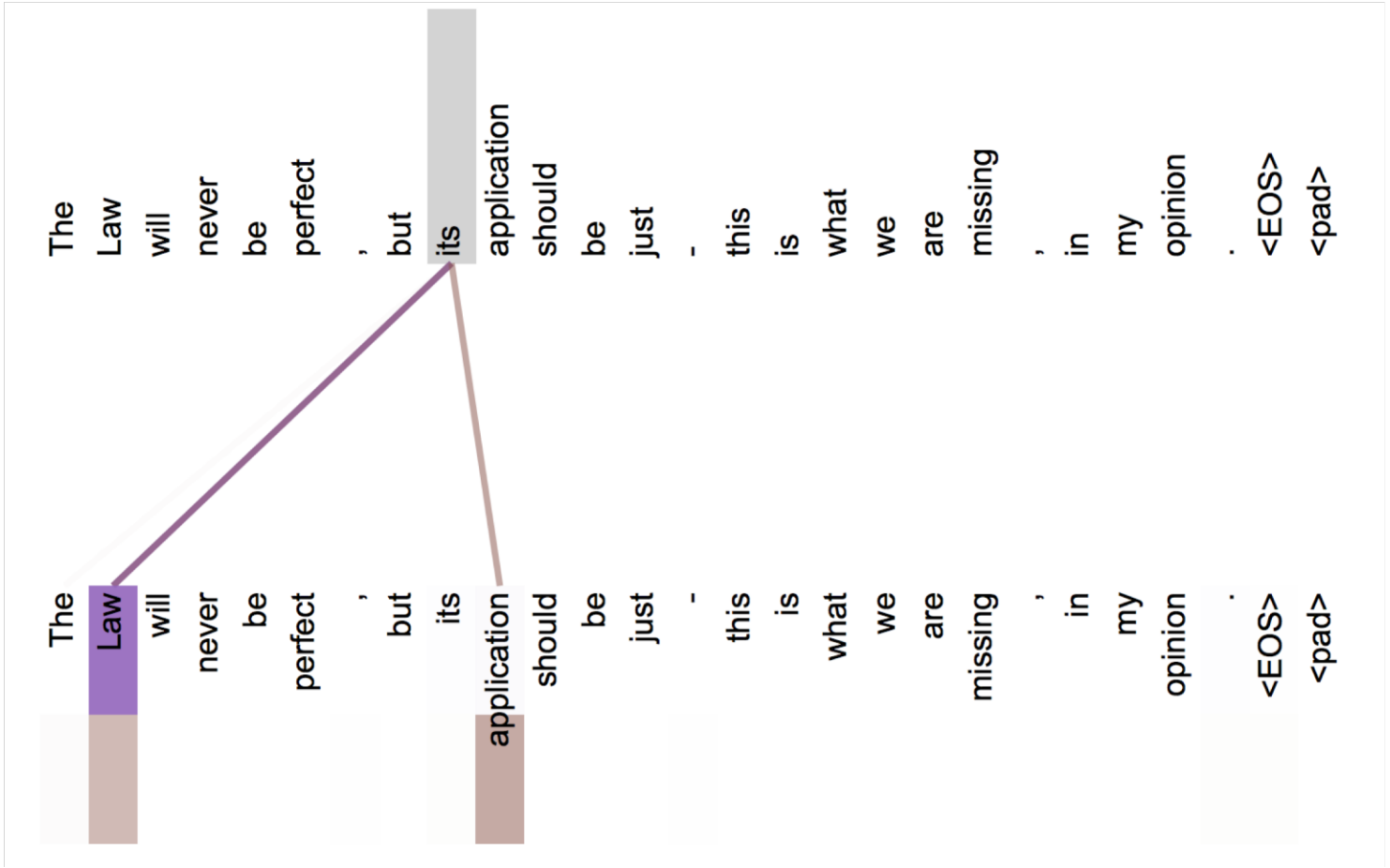
Same amount of computation as single-head self-attention!

Attention visualization in layer 5

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

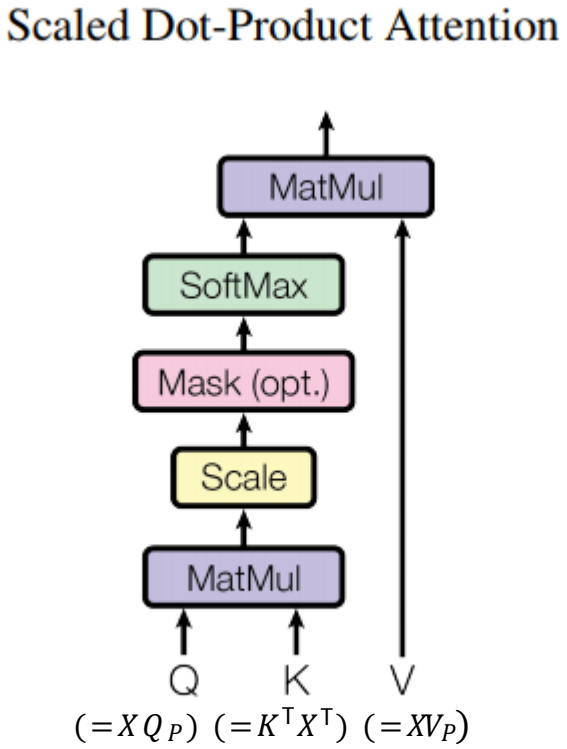


Attention visualization: Implicit anaphora resolution



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:

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

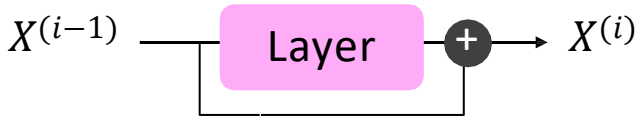
- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

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

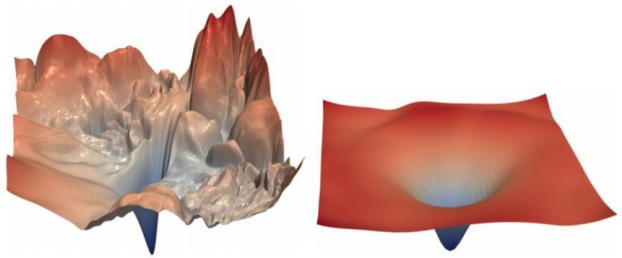
The Transformer Encoder:
Residual connections [He et al., 2016]

- **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:

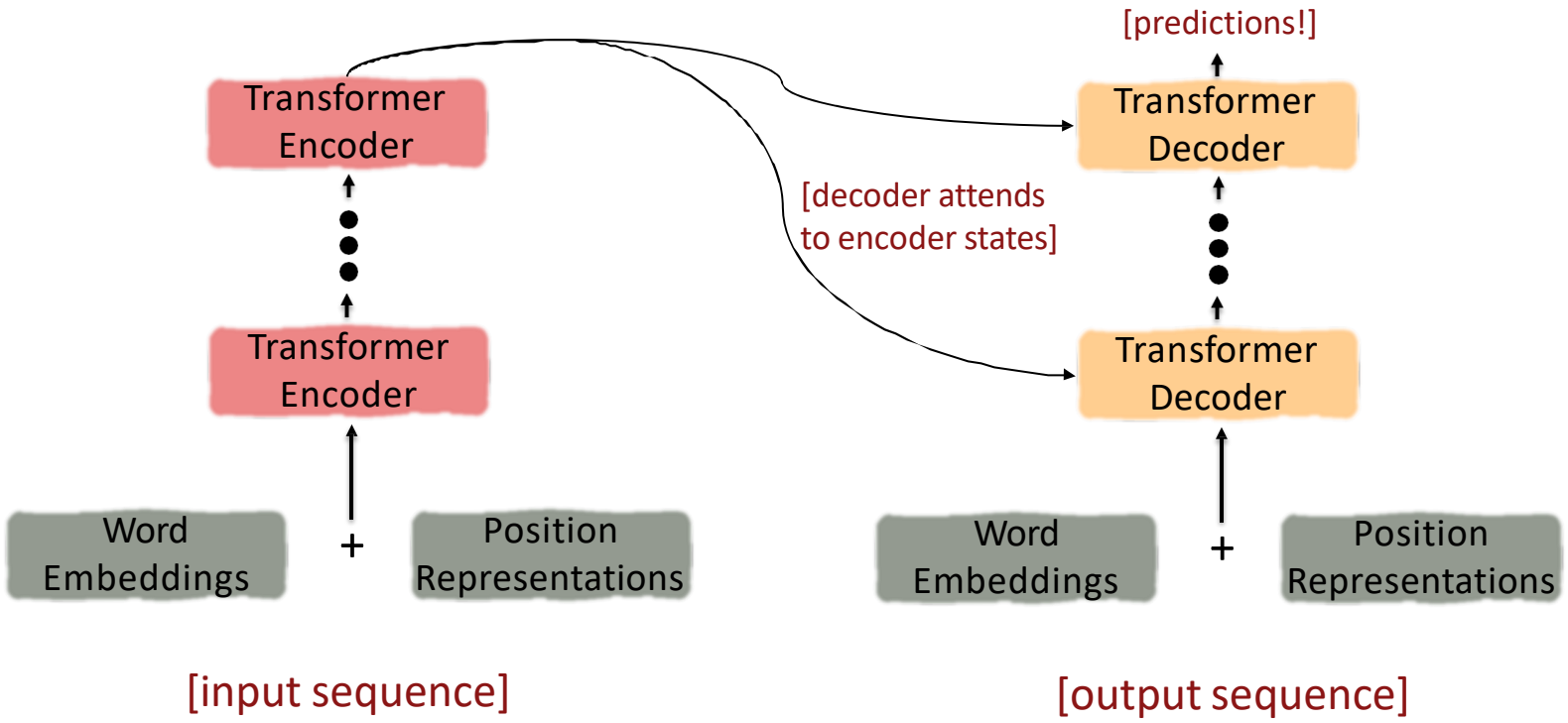
$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

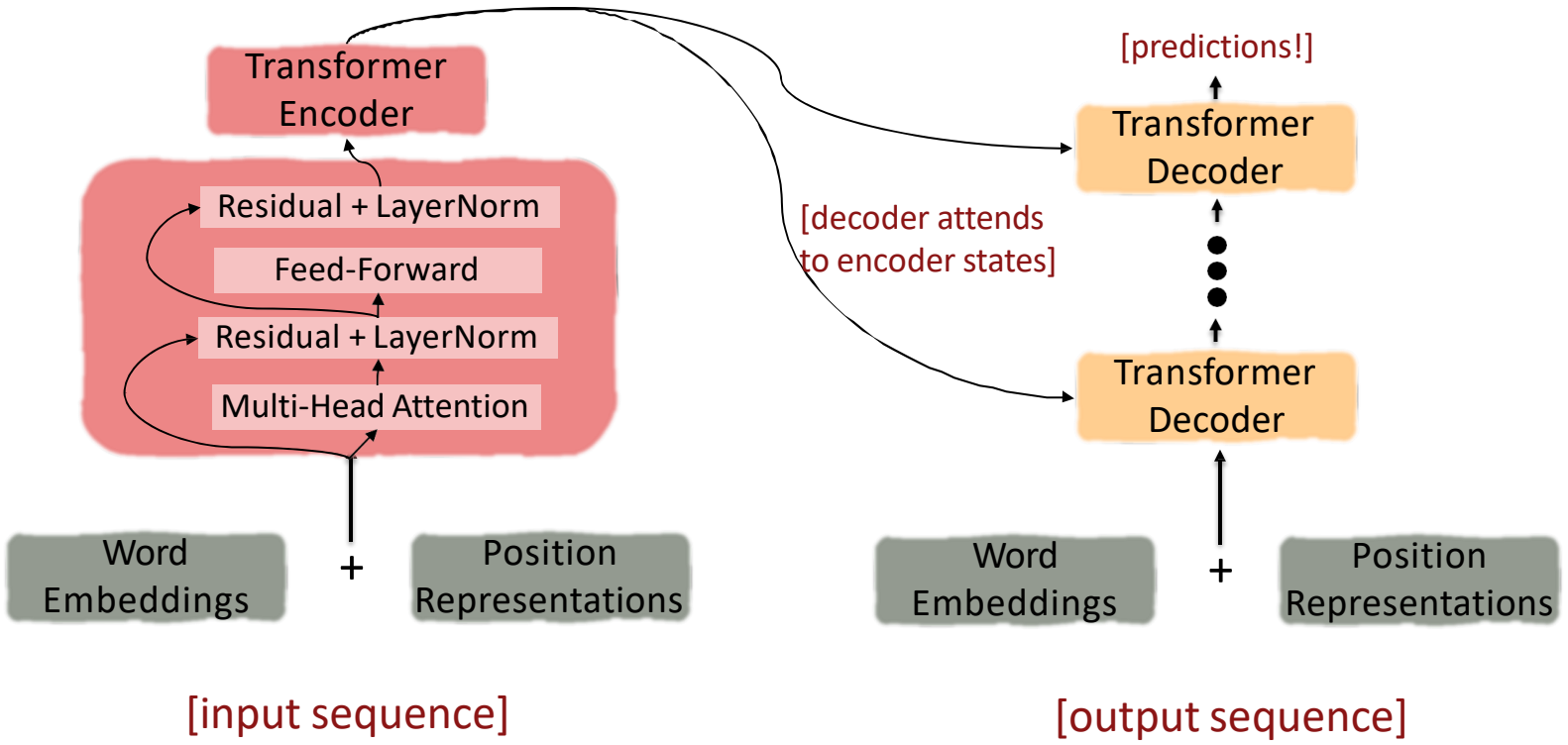
The Transformer Encoder-Decoder [Vaswani et al., 2017]

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



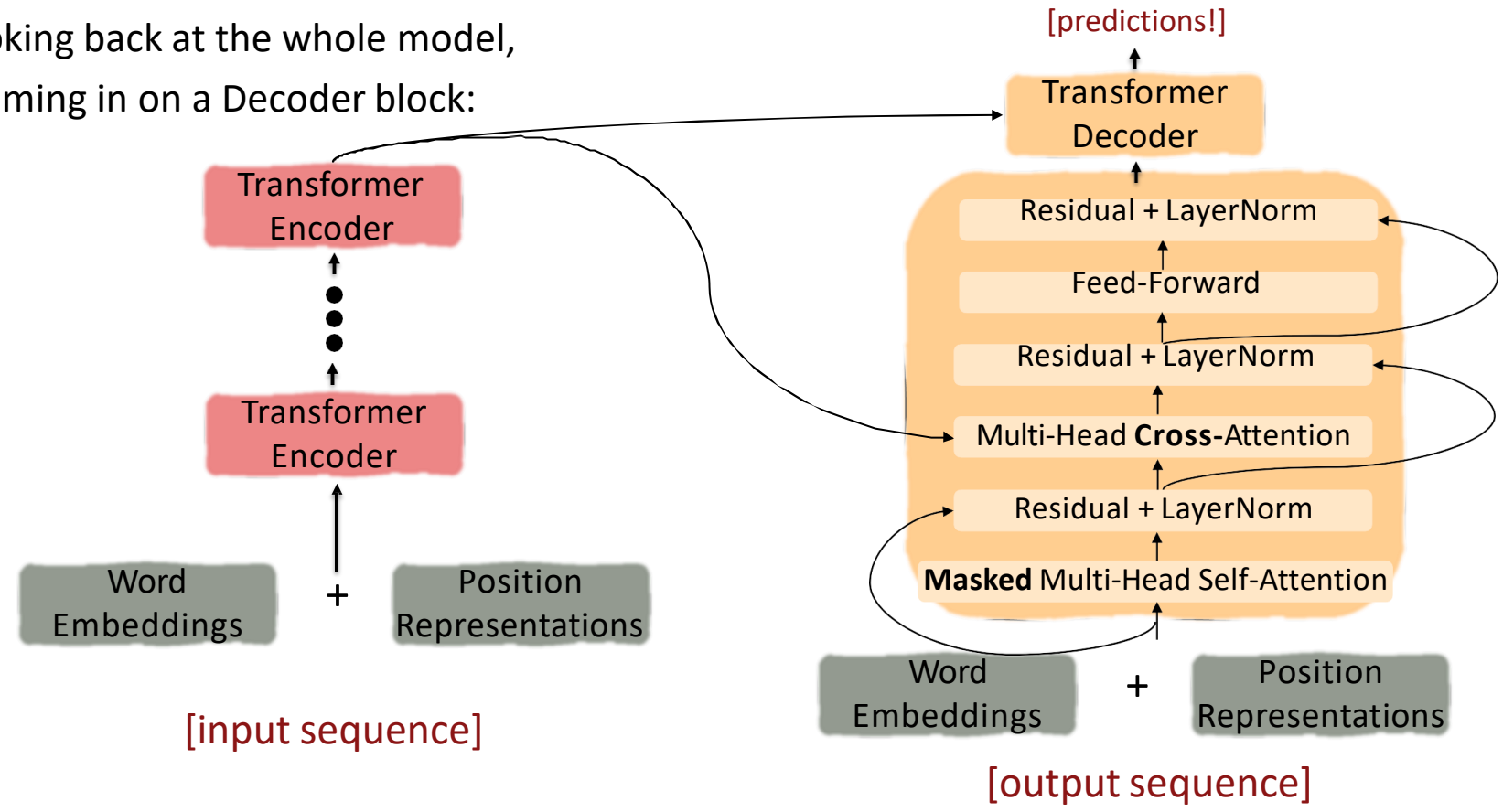
The Transformer Encoder-Decoder [Vaswani et al., 2017]

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



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

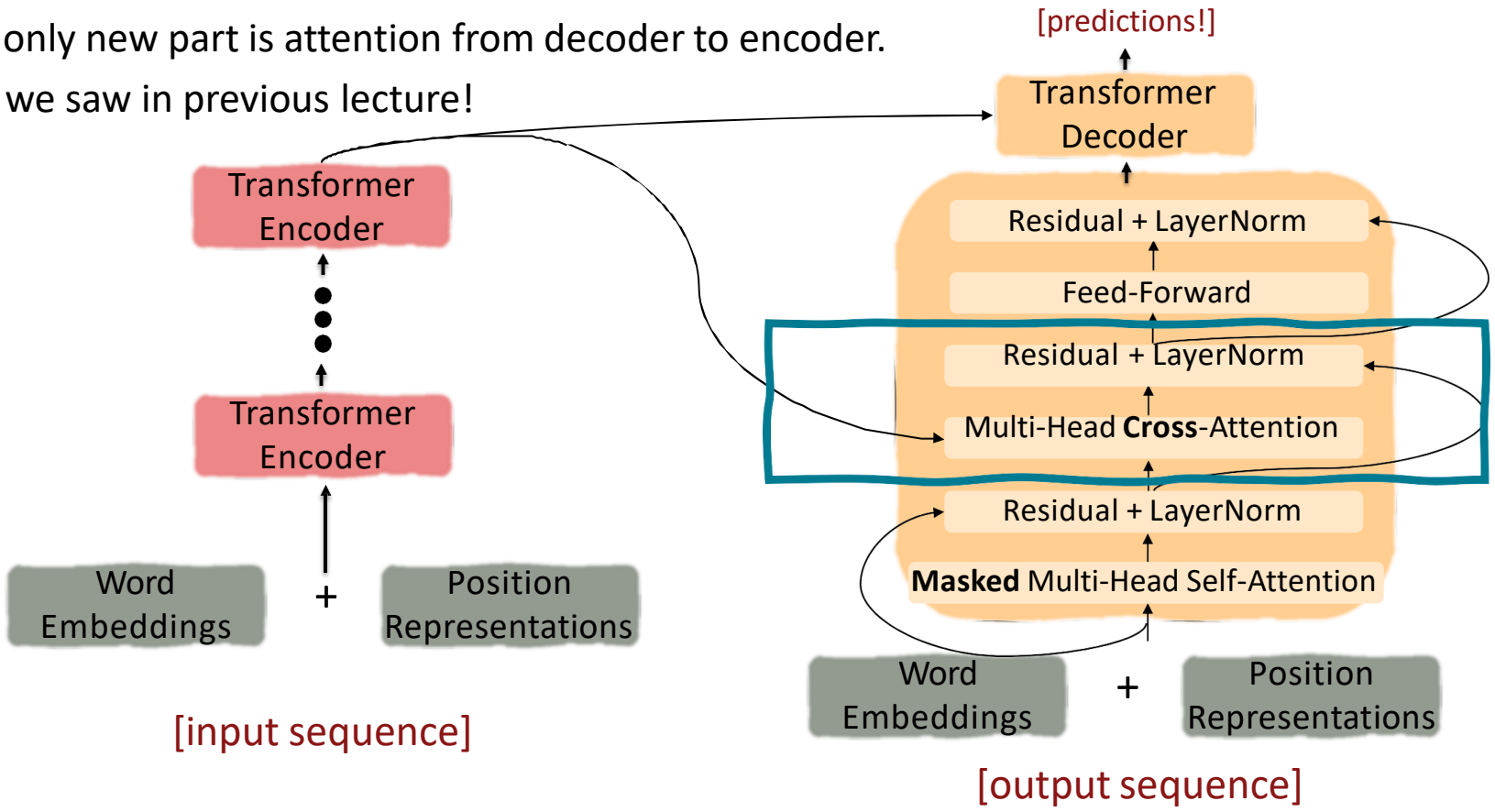
Looking back at the whole model, zooming in on a Decoder block:



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)

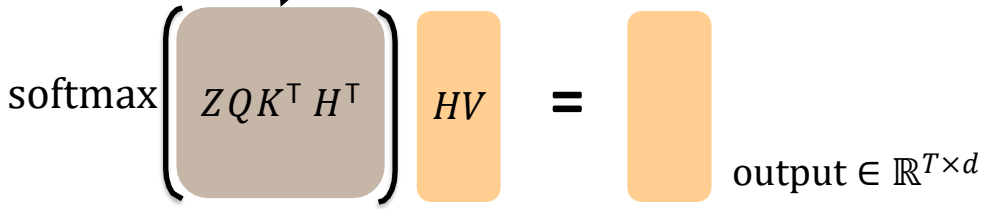
- We saw self-attention is when keys, queries, and values come from the same source.
- Let $h_1, \ldots, h_T$ be **output** vectors from the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and 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

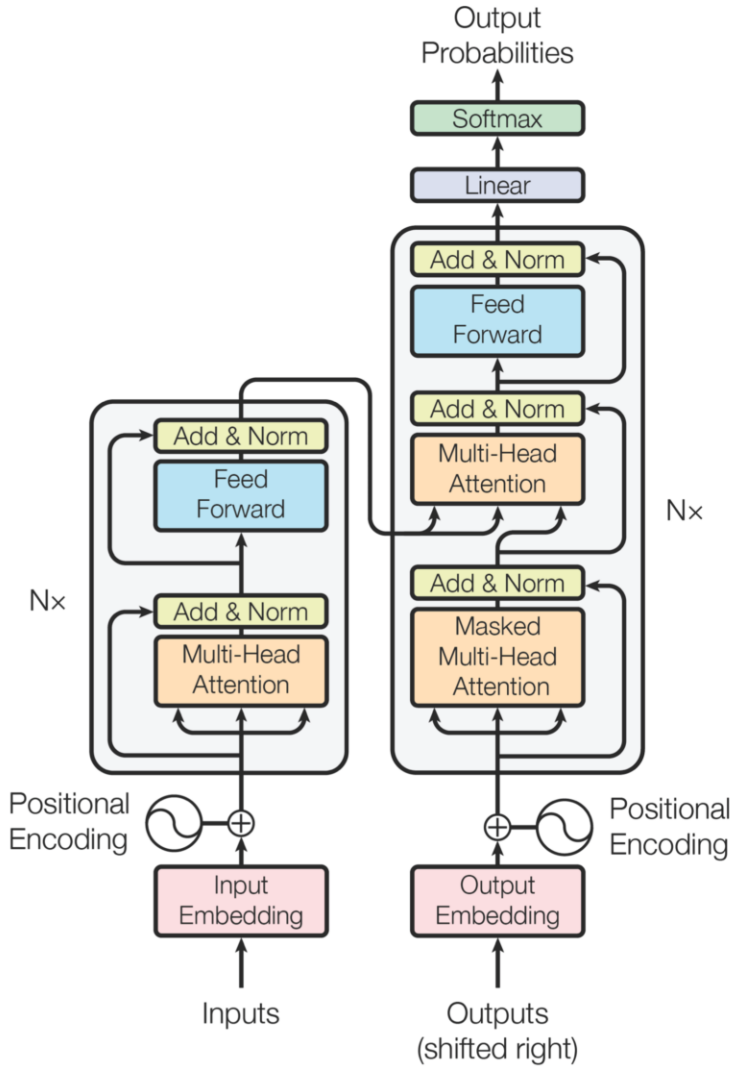


Figure 1: The Transformer - model architecture.

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

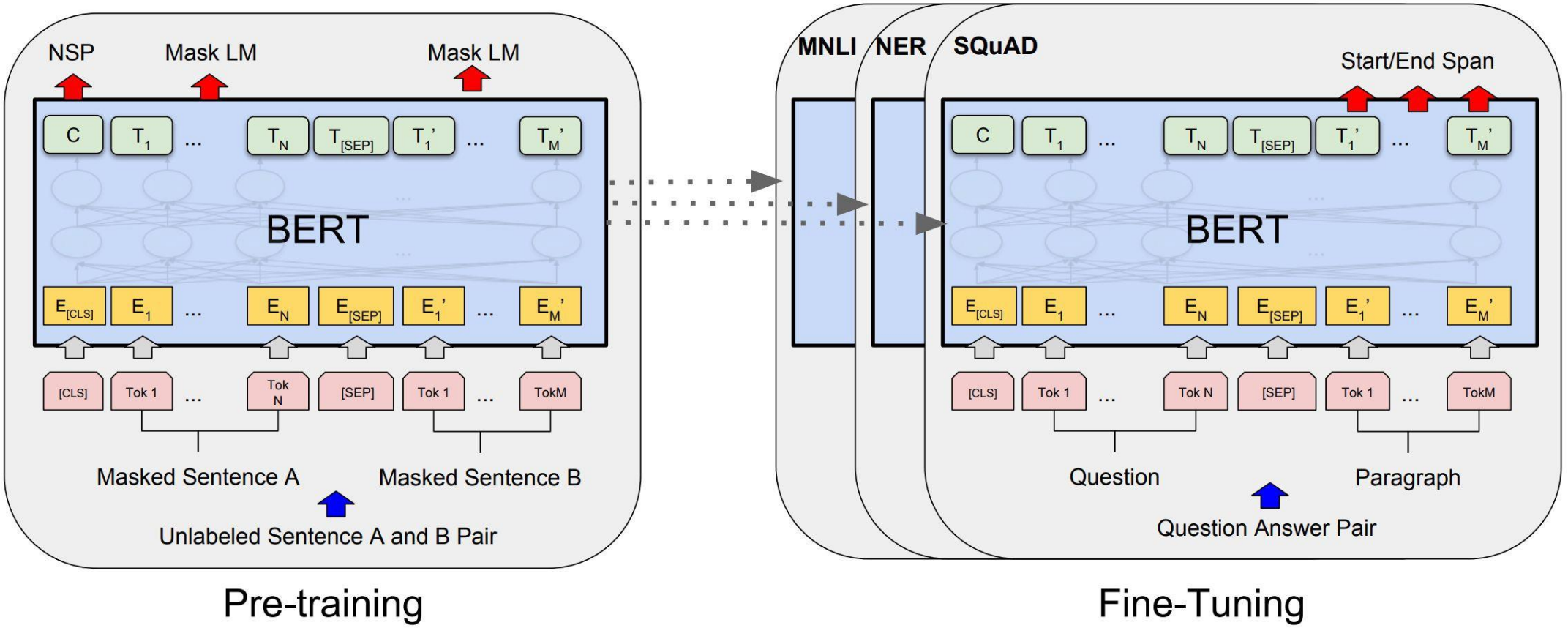
- On top of Transformer Encoder Block
- Add GlobalAveragePooling
- Add Feedforward Neural Net
- Then, do softmax

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

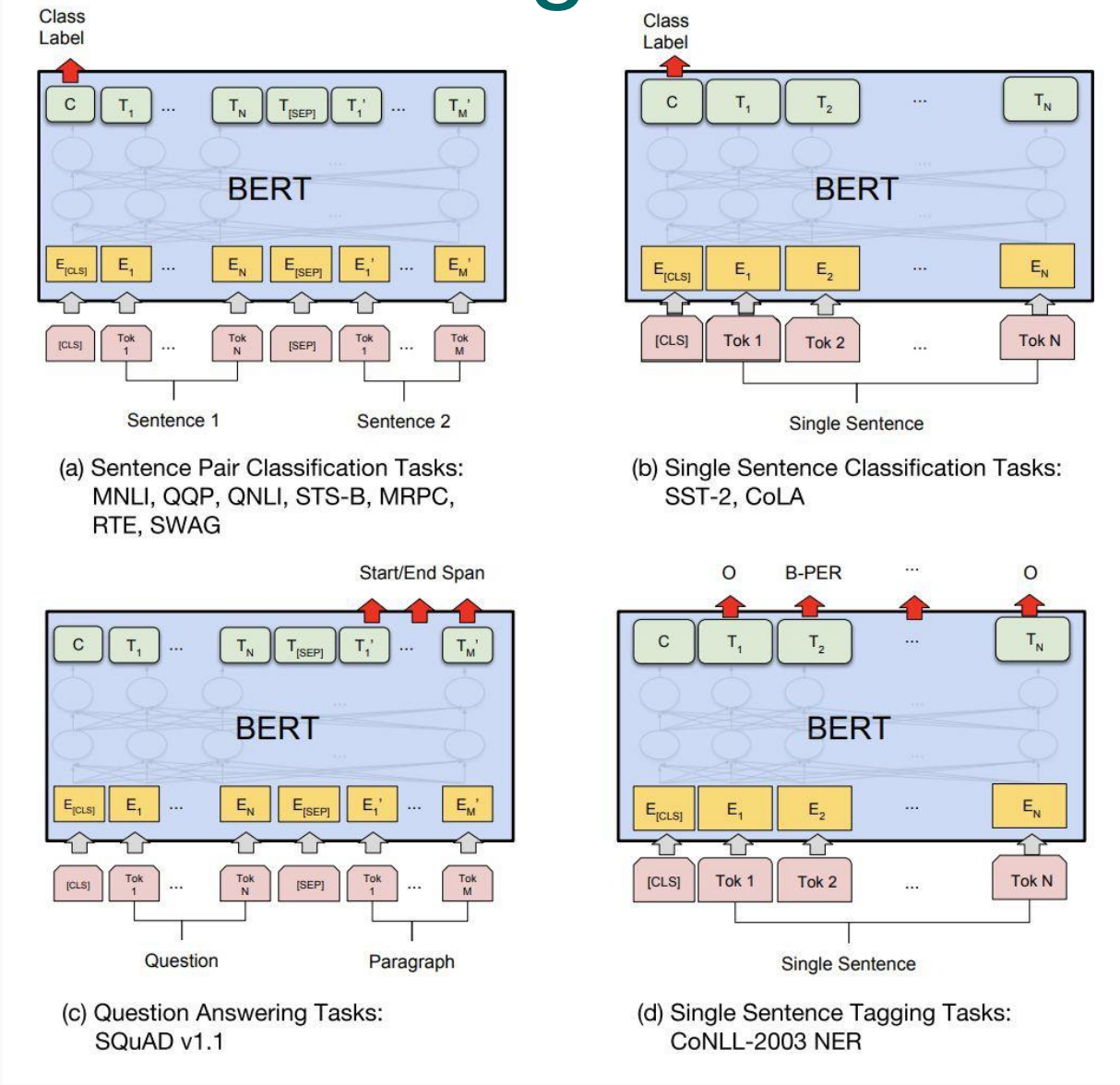
BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Google, 2018

Fine-Tuning Procedure



Fine-Tuning Procedure



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

BERT was released in 2018

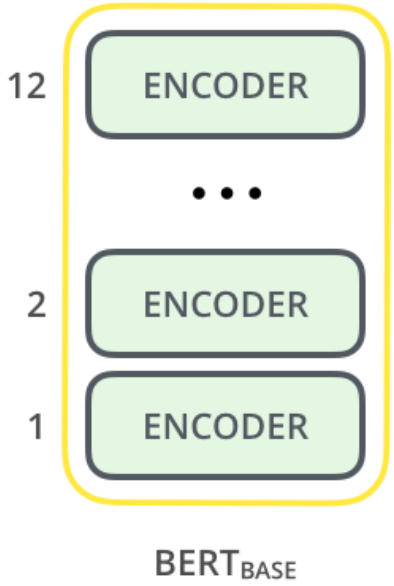
$BERT_{BASE}$ (L=12, H=768, A=12, Total Parameters=110M)

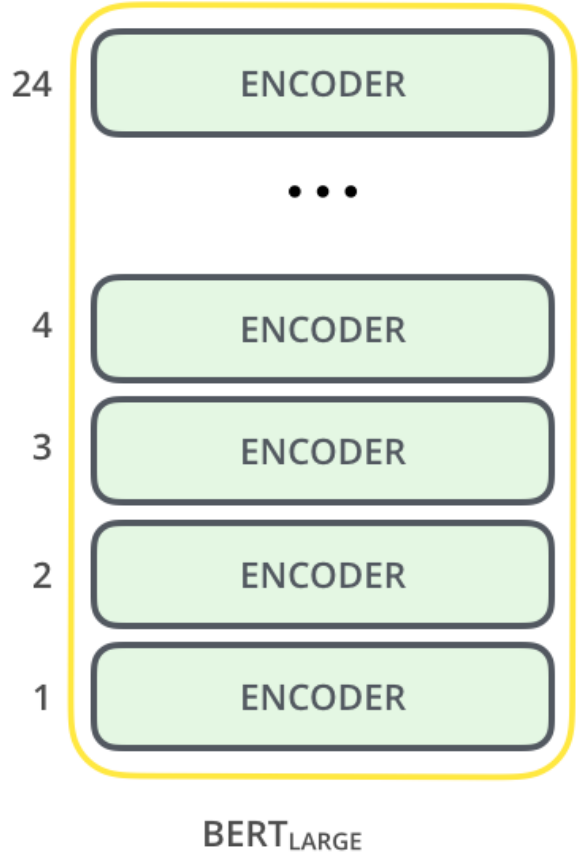
$BERT_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M)

L = the number of layers;

H = hidden size

A = number of attention head





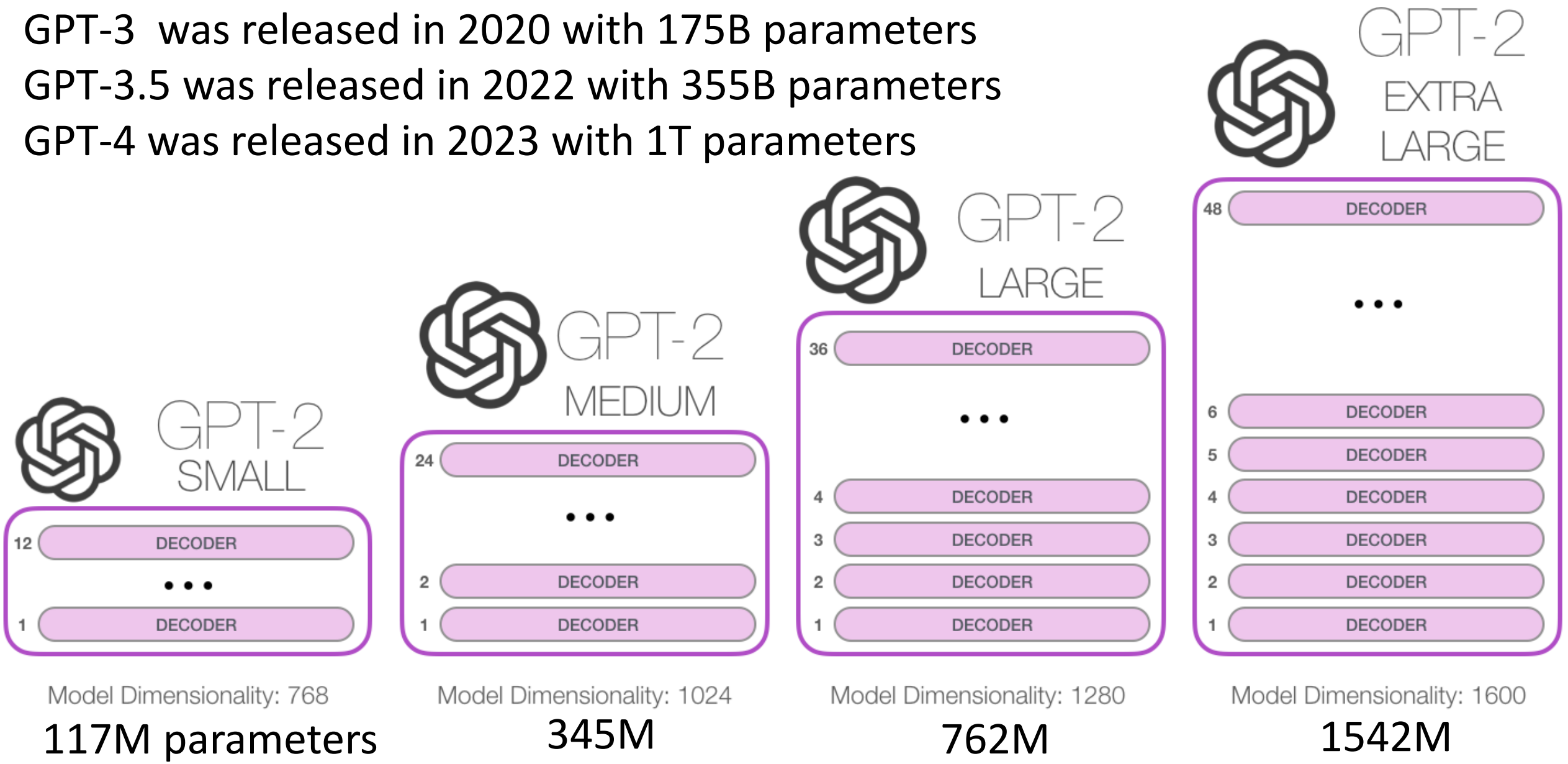
GPT was released in 2018

GPT-2 was released in 2019 with 1.5B parameters

GPT-3 was released in 2020 with 175B parameters

GPT-3.5 was released in 2022 with 355B parameters

GPT-4 was released in 2023 with 1T parameters



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

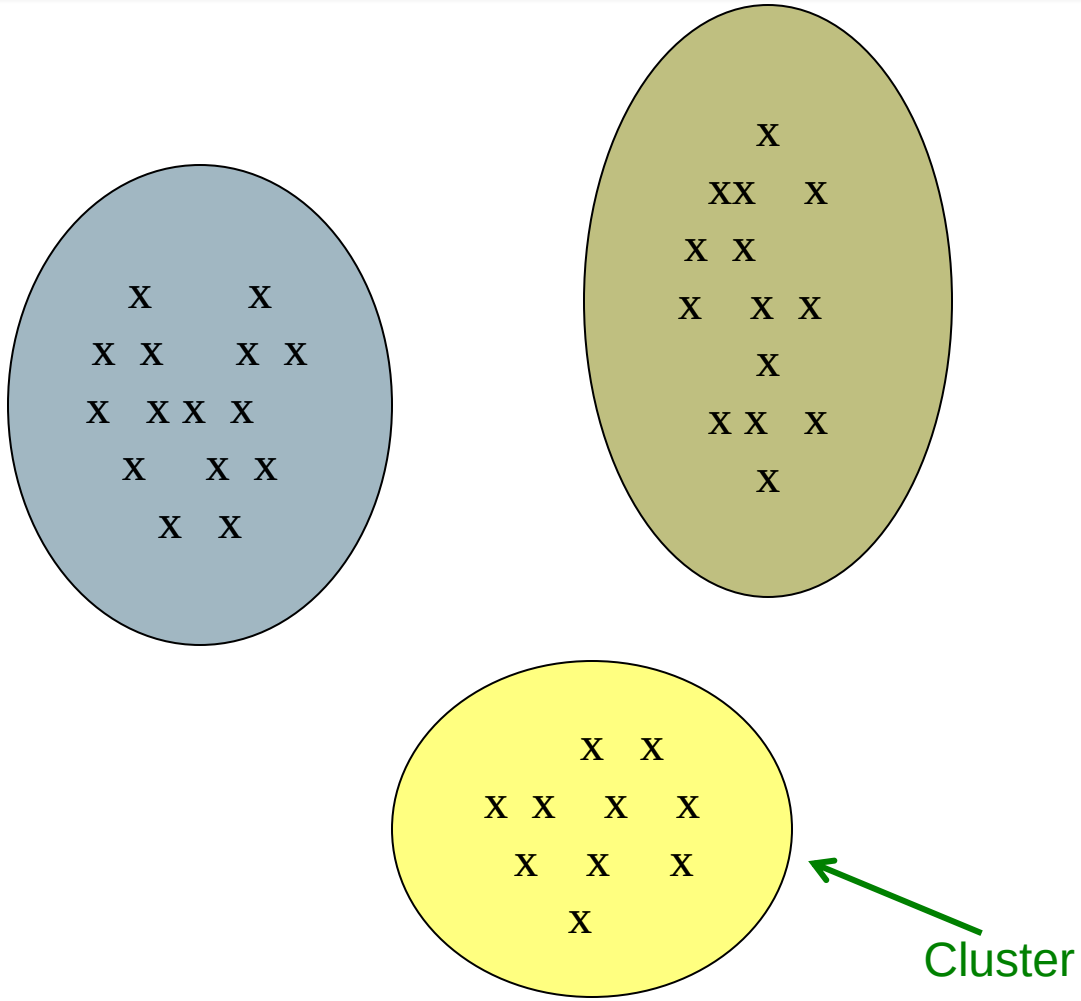
- We now have an evaluation measure (F_1) for one class.
- But we also want a single number that measures the aggregate performance over all classes in the collection.
- Macroaveraging
  - Compute F_1 for each of the C classes
  - Average these C numbers
- Microaveraging
  - Compute TP, FP, FN for each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

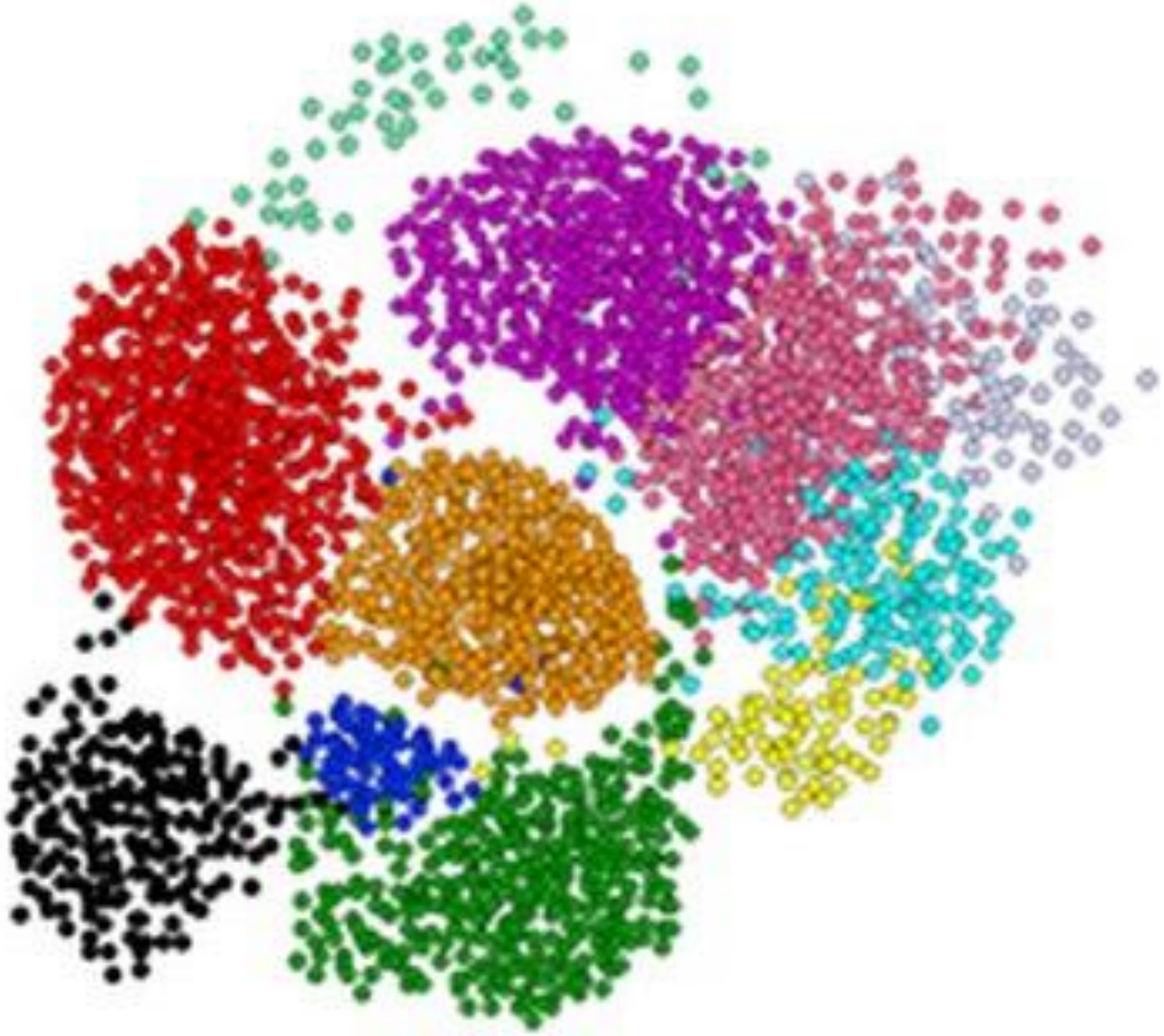
What is Clustering?

- **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

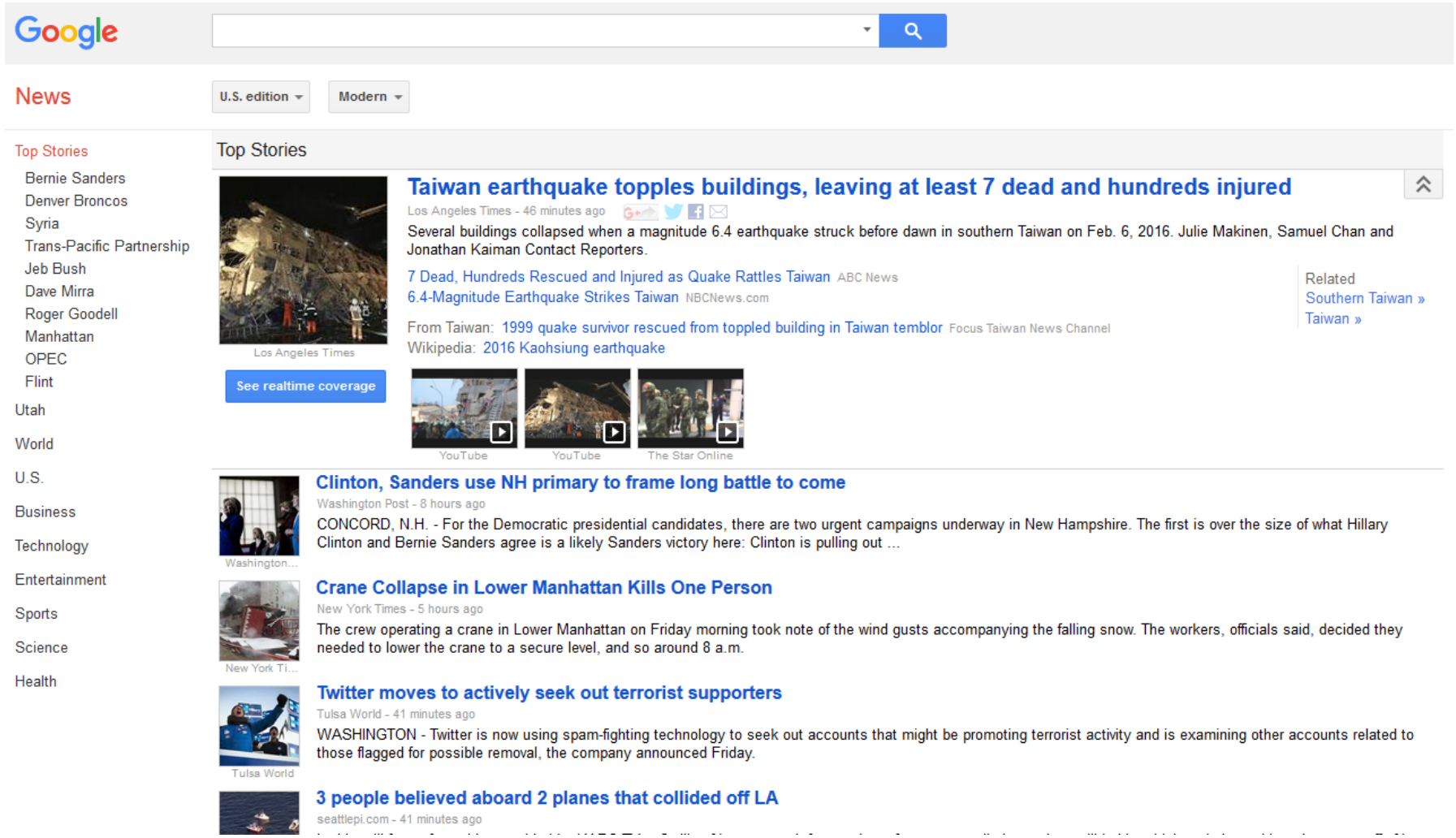
- Typical applications
  - As a stand-alone tool to get insight into data distribution
  - As a preprocessing step for other algorithms

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

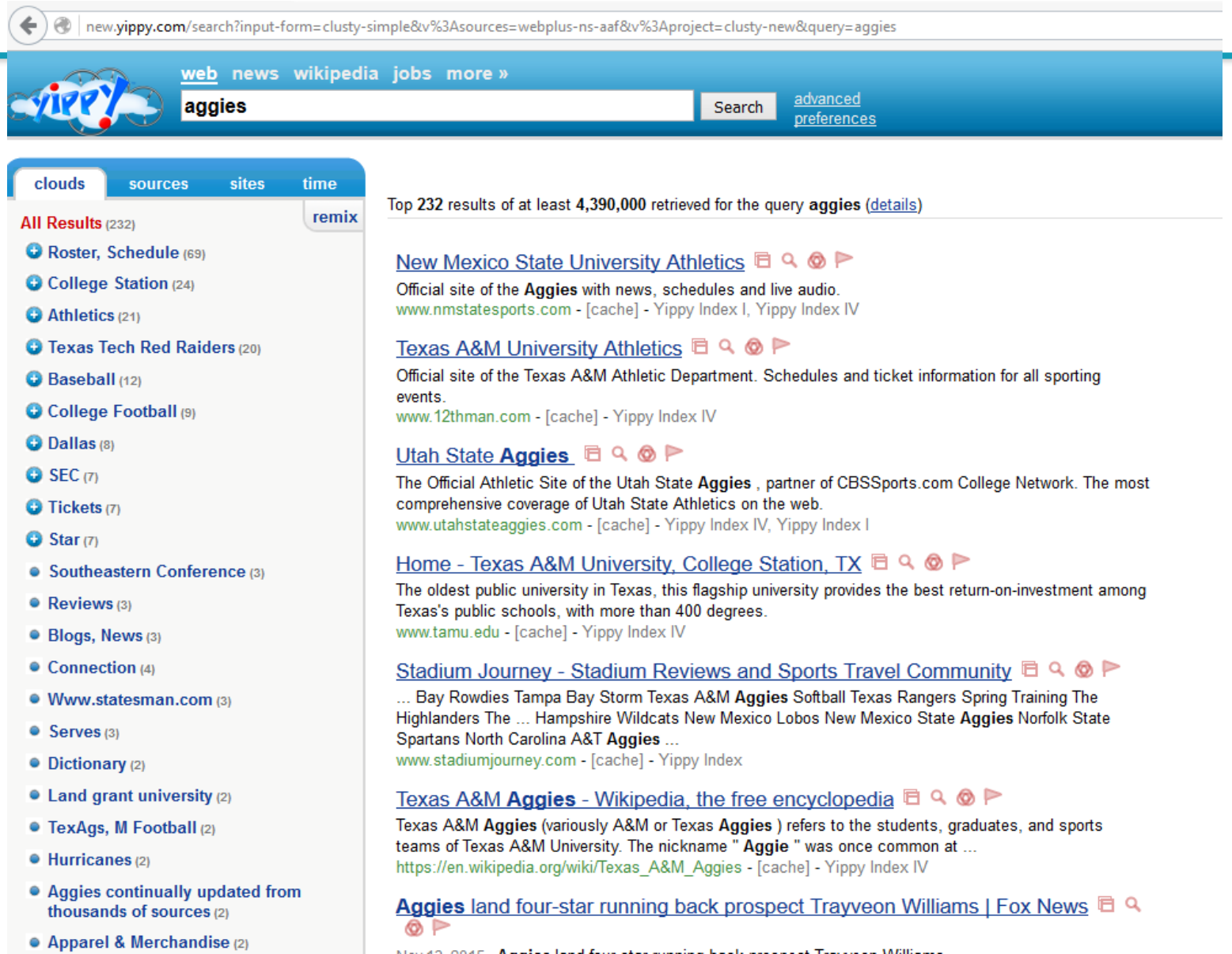
For improving search recall

- *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.

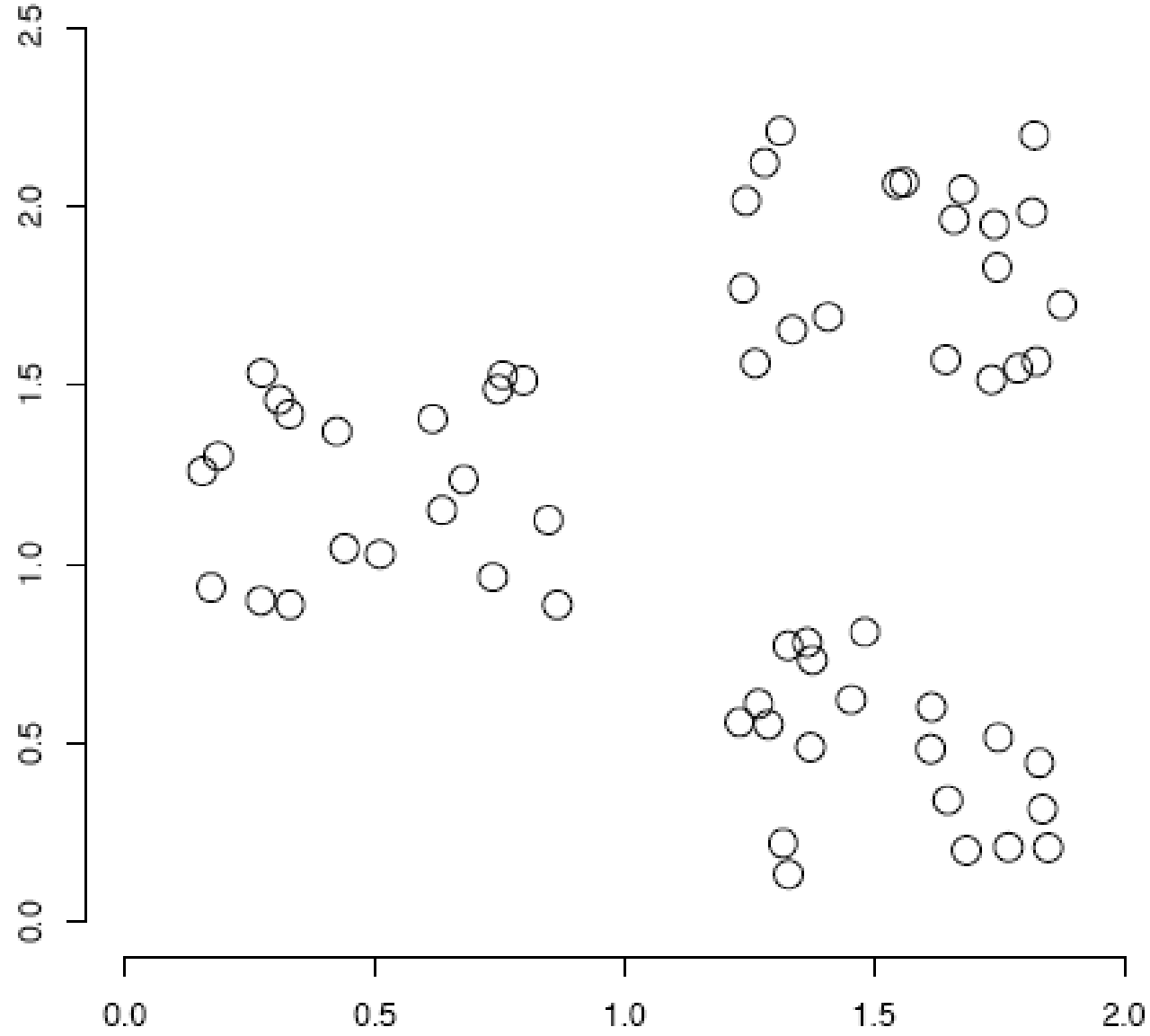
Hard vs. soft clustering

- 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, ω:

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

- 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

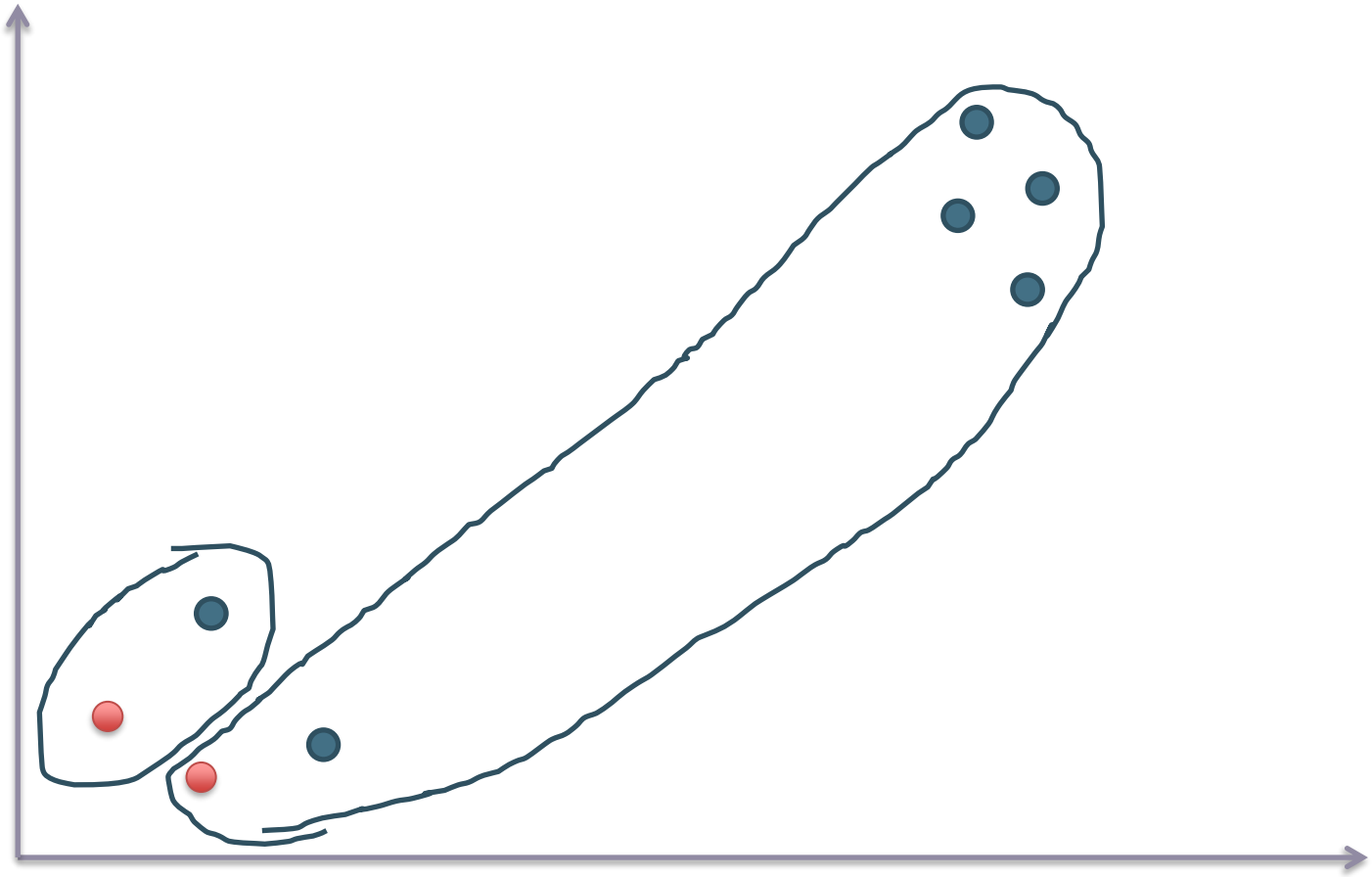
K-MEANS$(\{\vec{x}_1, \ldots, \vec{x}_N\}, K)$

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** $\{\vec{\mu}_1, \ldots, \vec{\mu}_K\}$

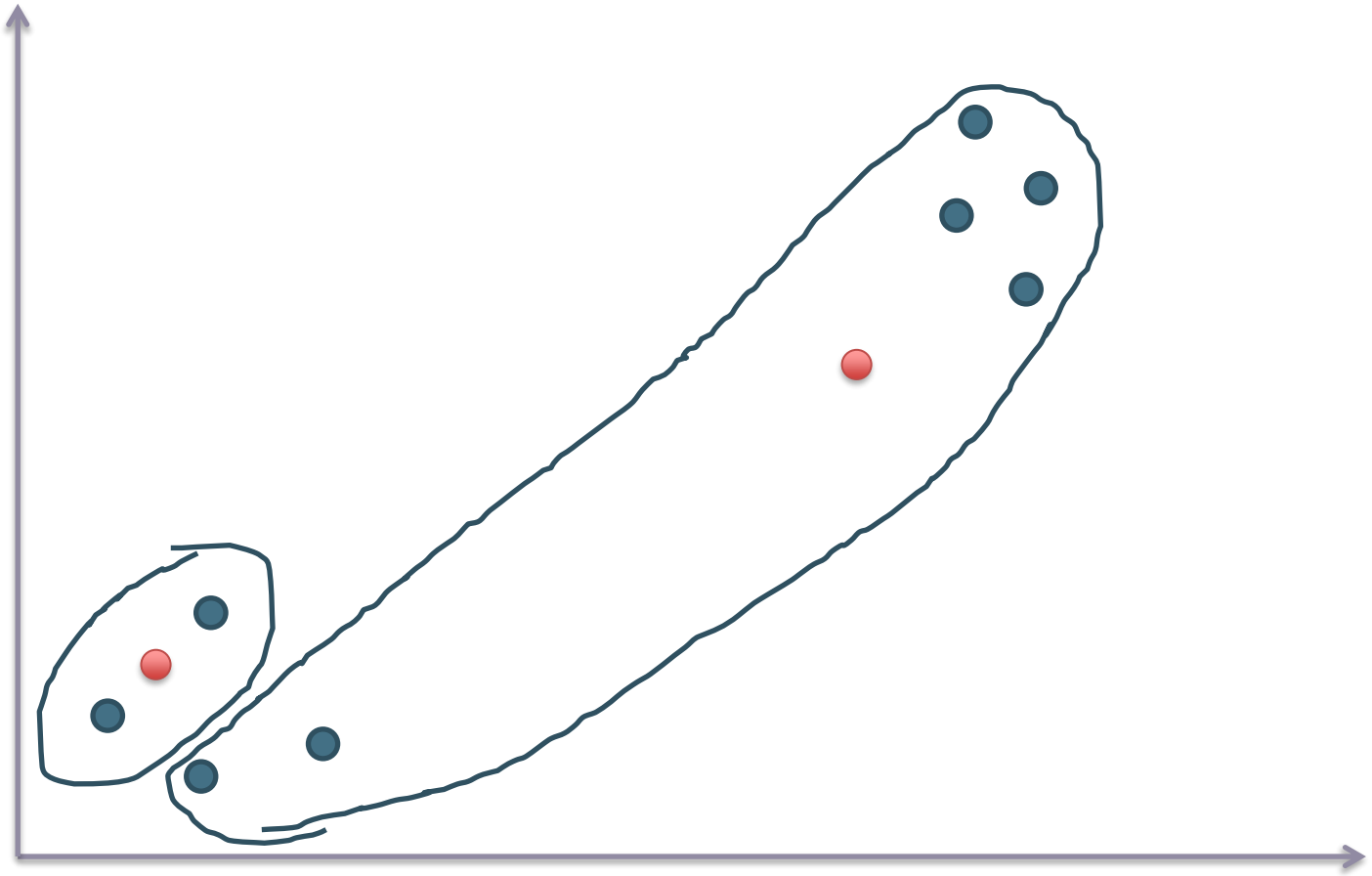
K-Means Clustering Example

K-Means Clustering Example

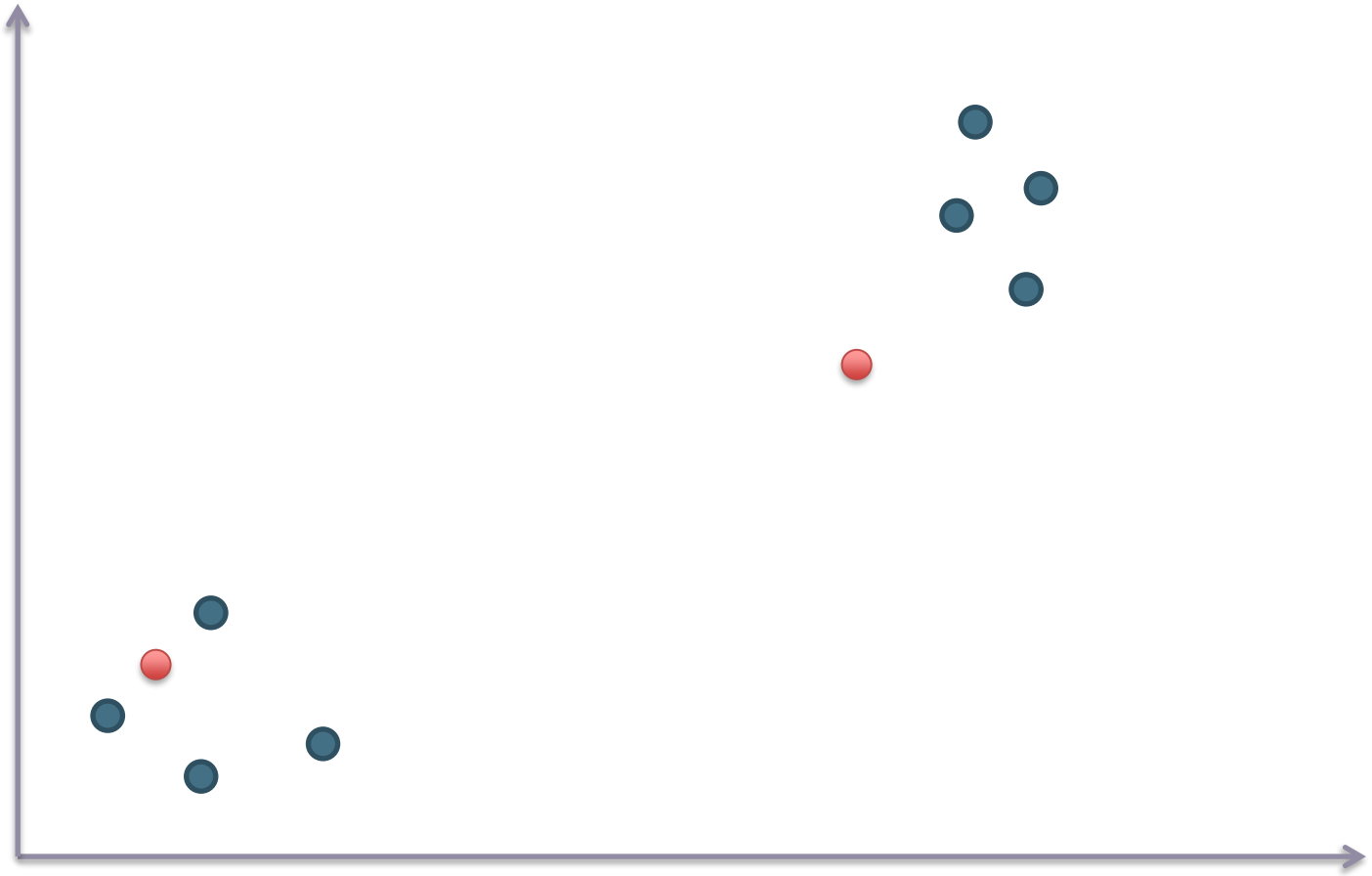
K-Means Clustering Example



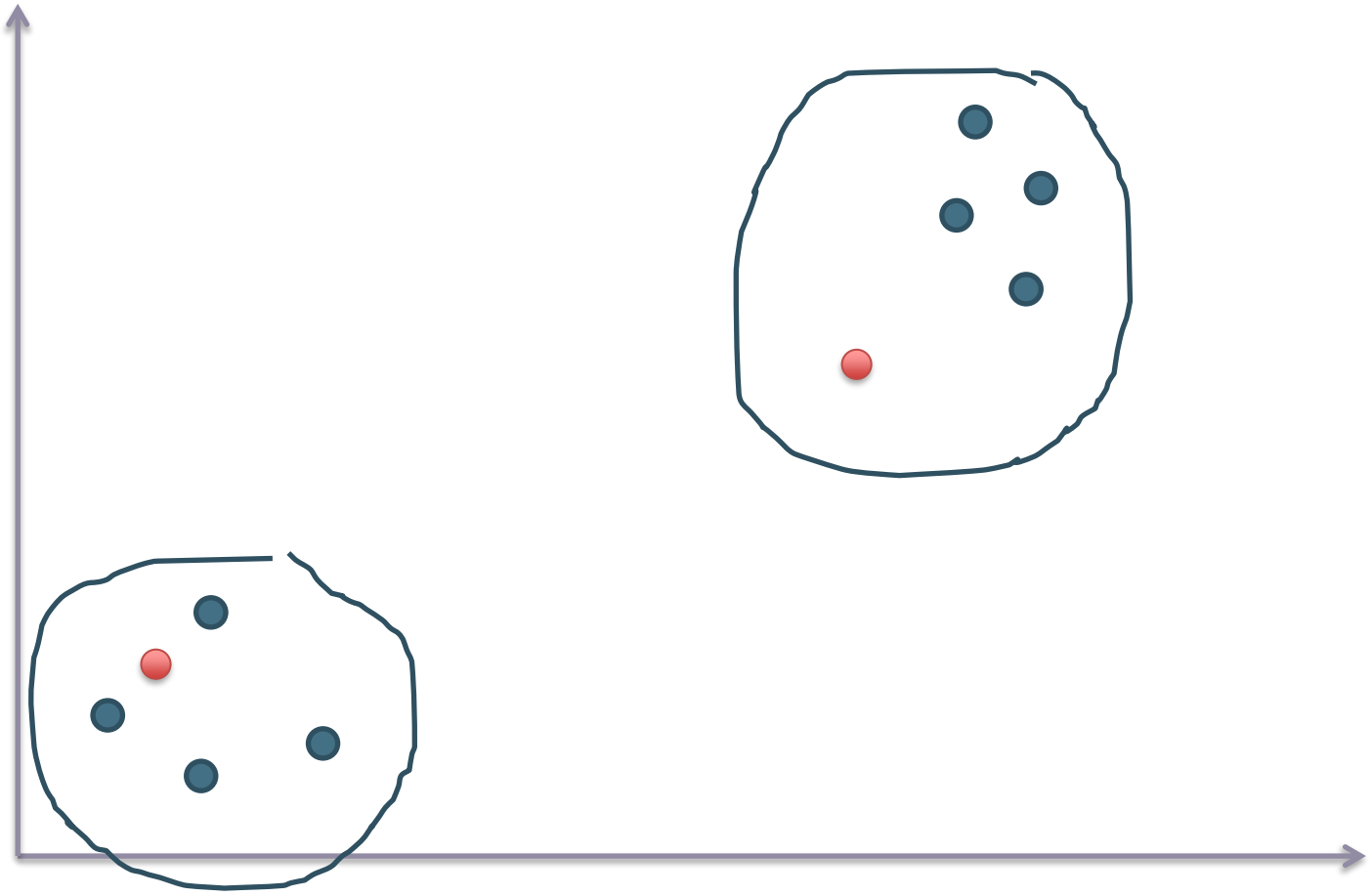
K-Means Clustering Example



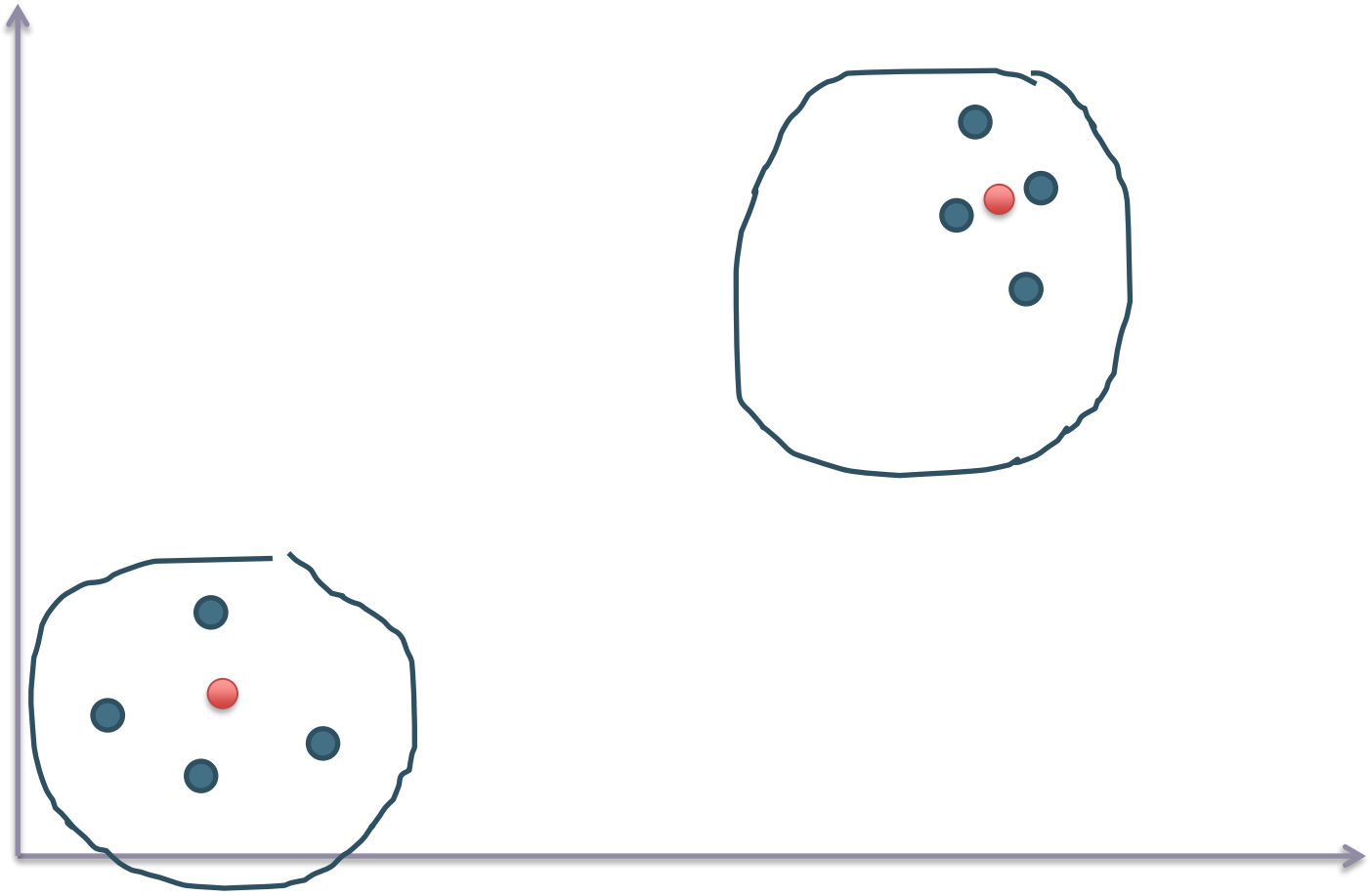
K-Means Clustering Example



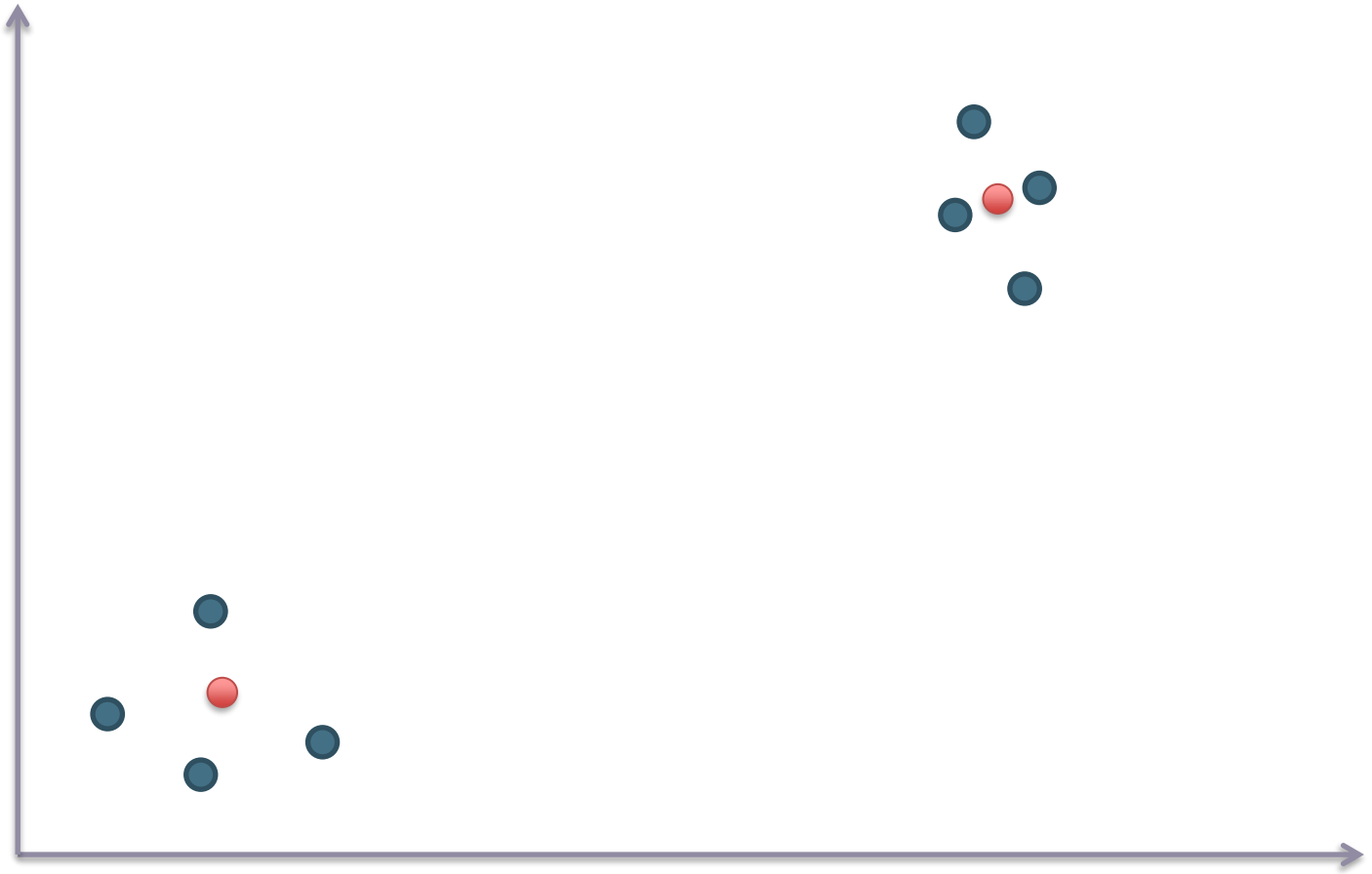
K-Means Clustering Example



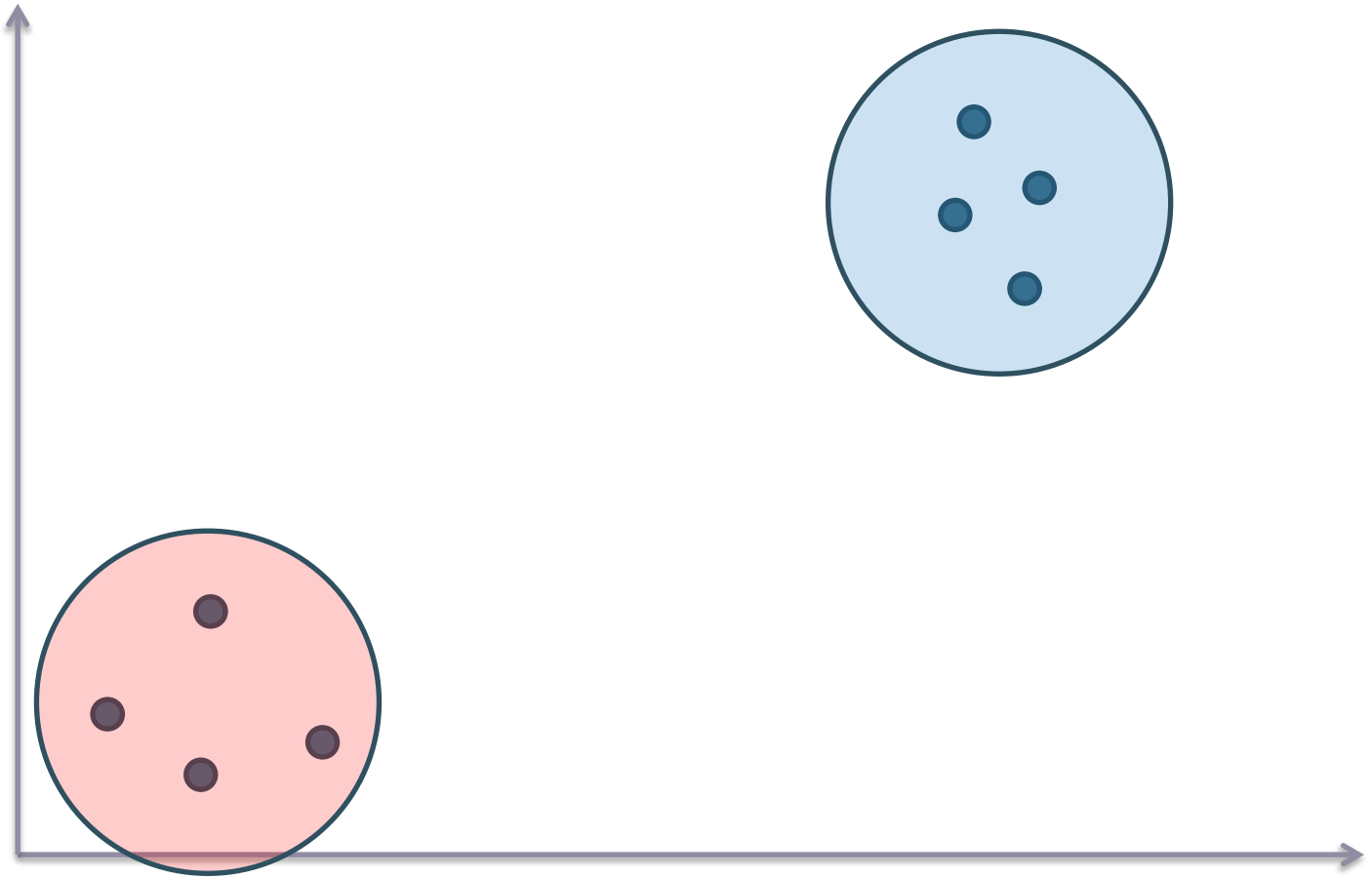
K-Means Clustering Example



K-Means Clustering Example

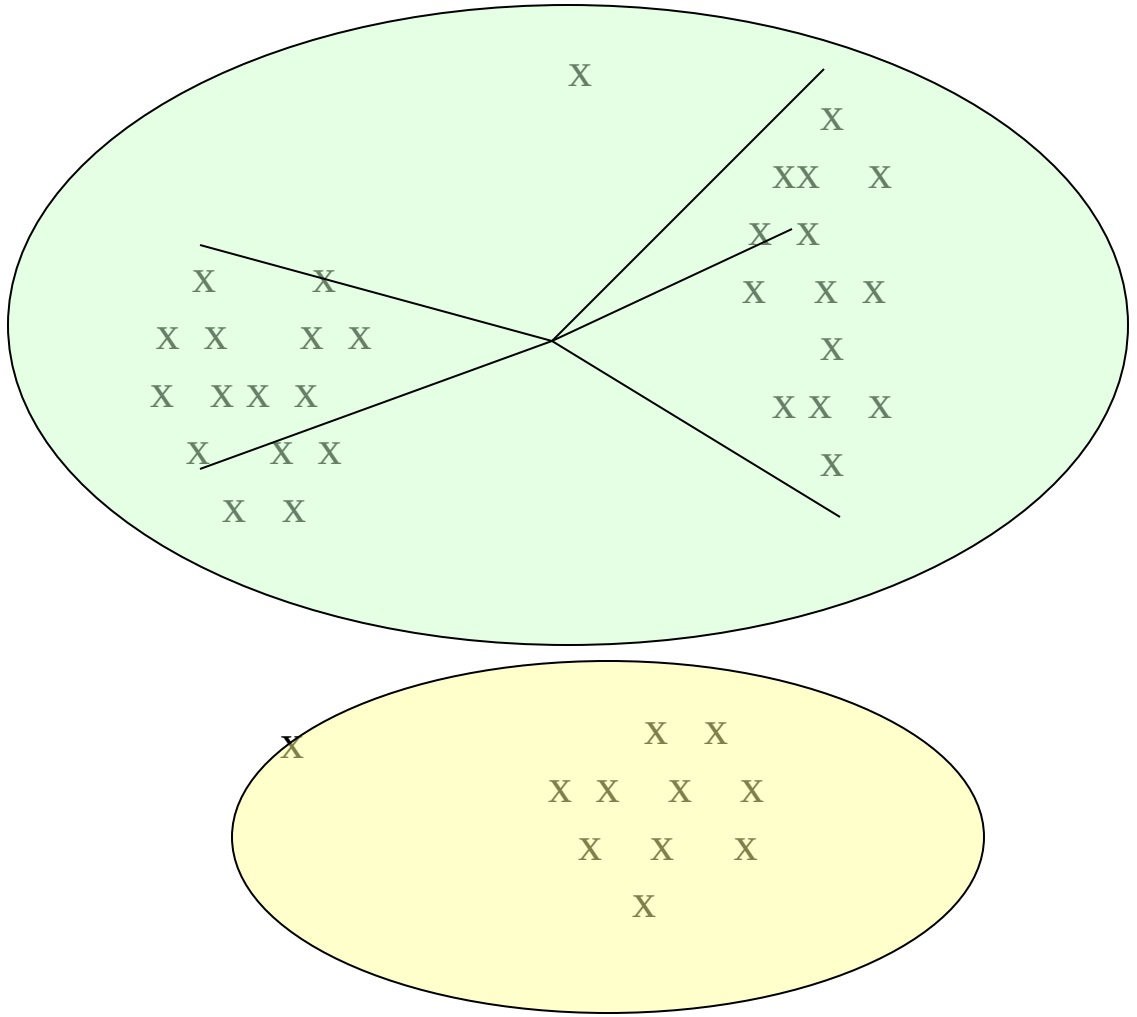


K-Means Clustering Example



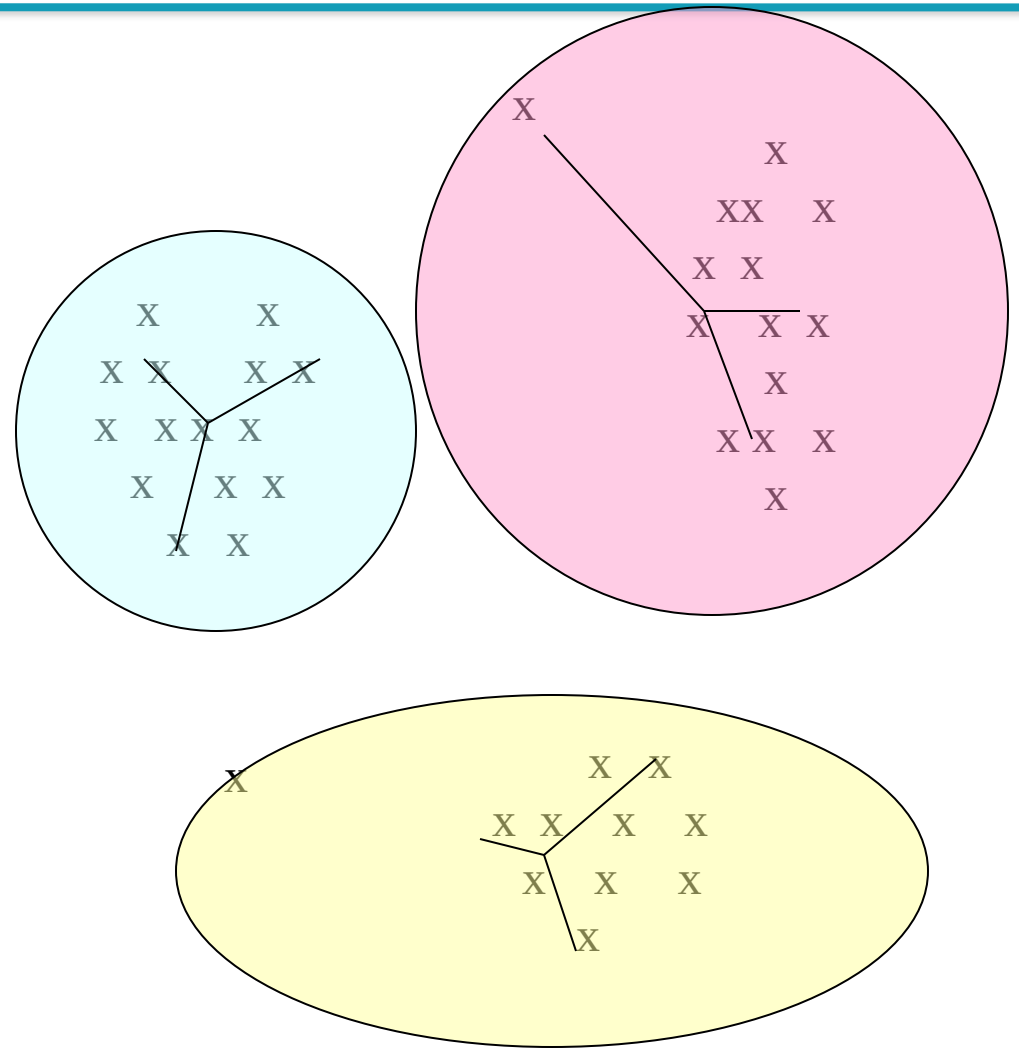
Example: Picking *k*

Too few;
many long
distances
to centroid.



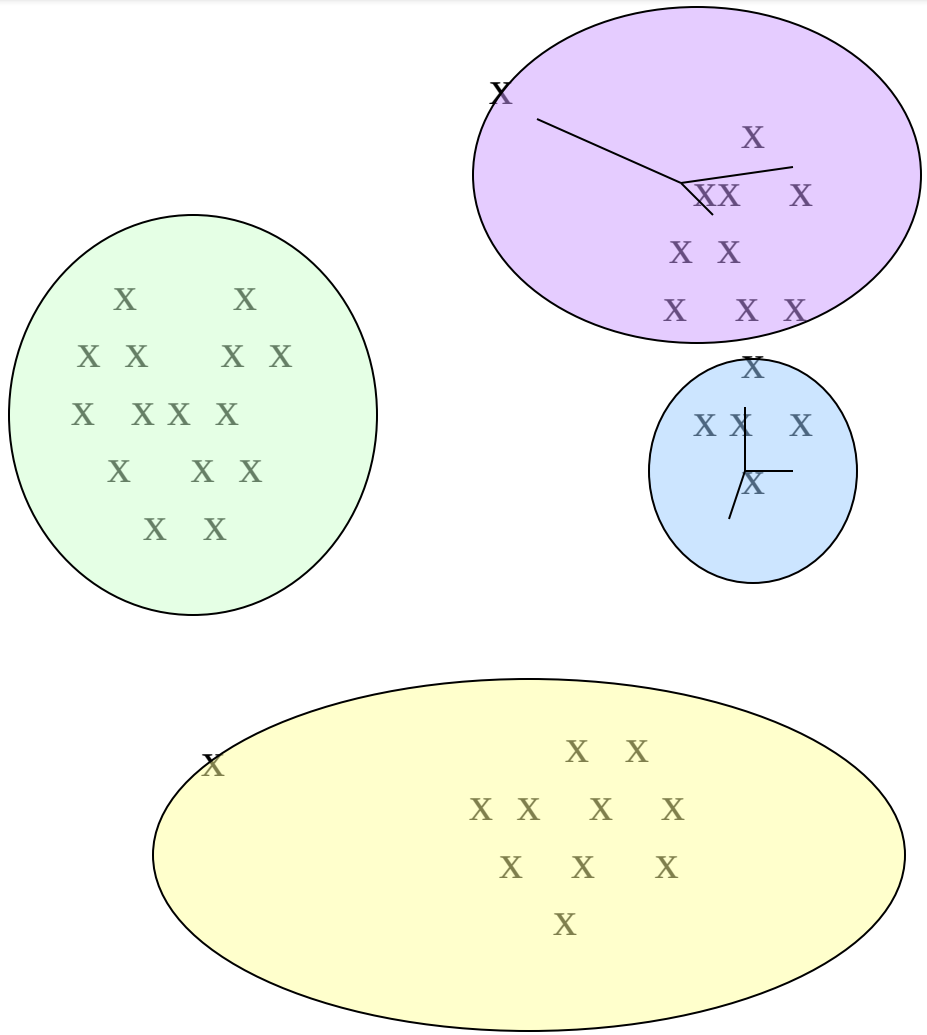
Example: Picking *k*

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)

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

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

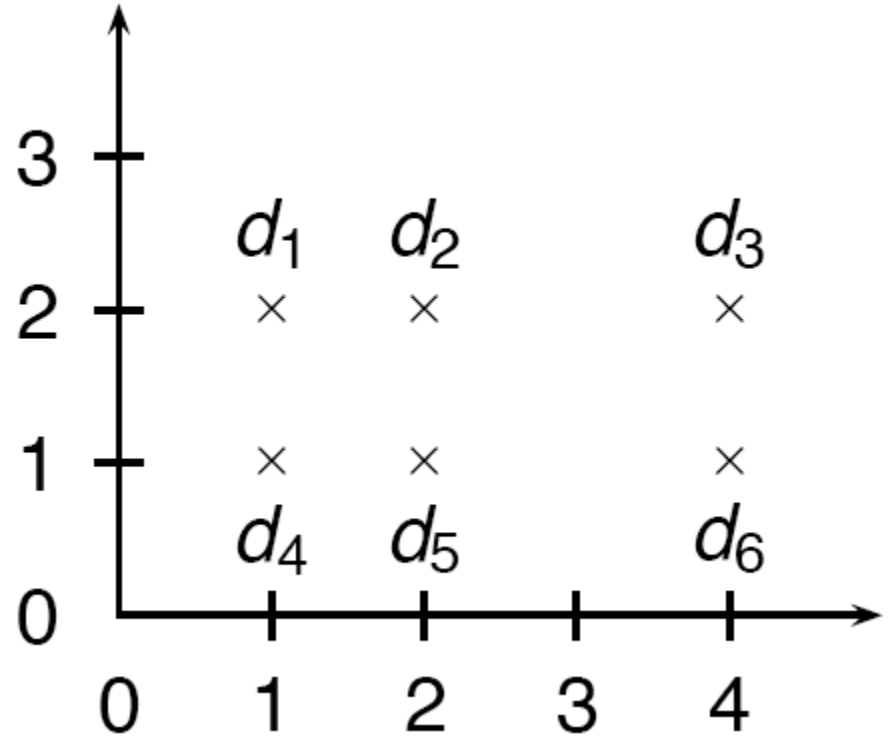
$$v_m = \frac{1}{|\omega_k|}\sum_{\vec{x}\in\omega_k} x_m$$

- 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?

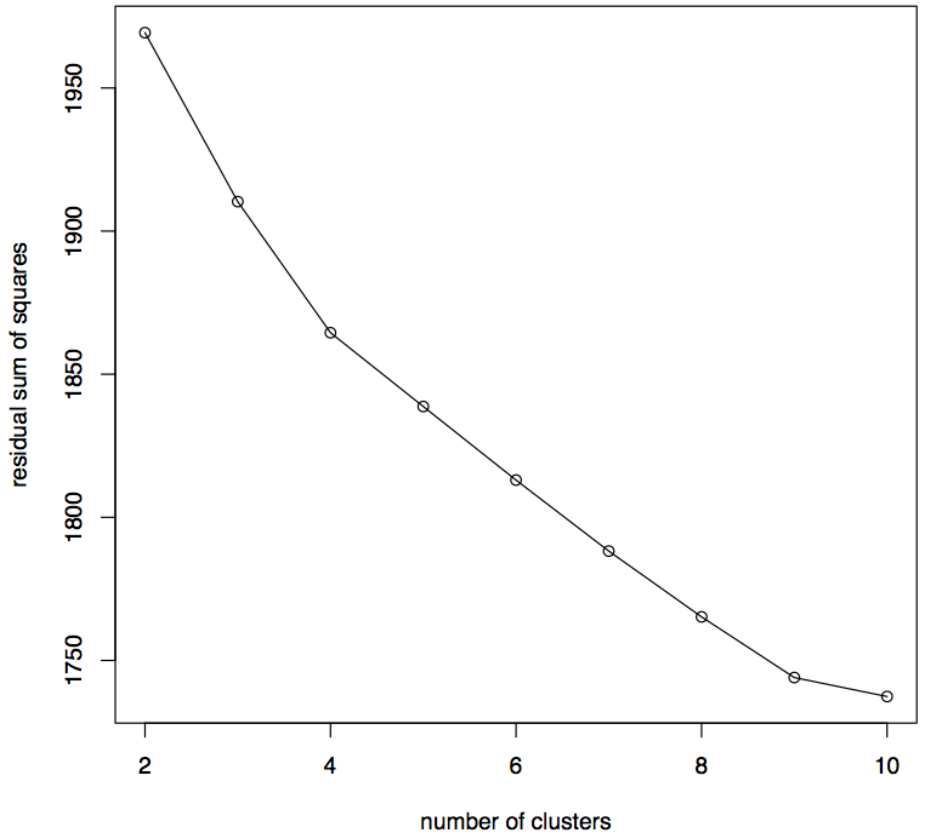
Simple objective function for *K*

- 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

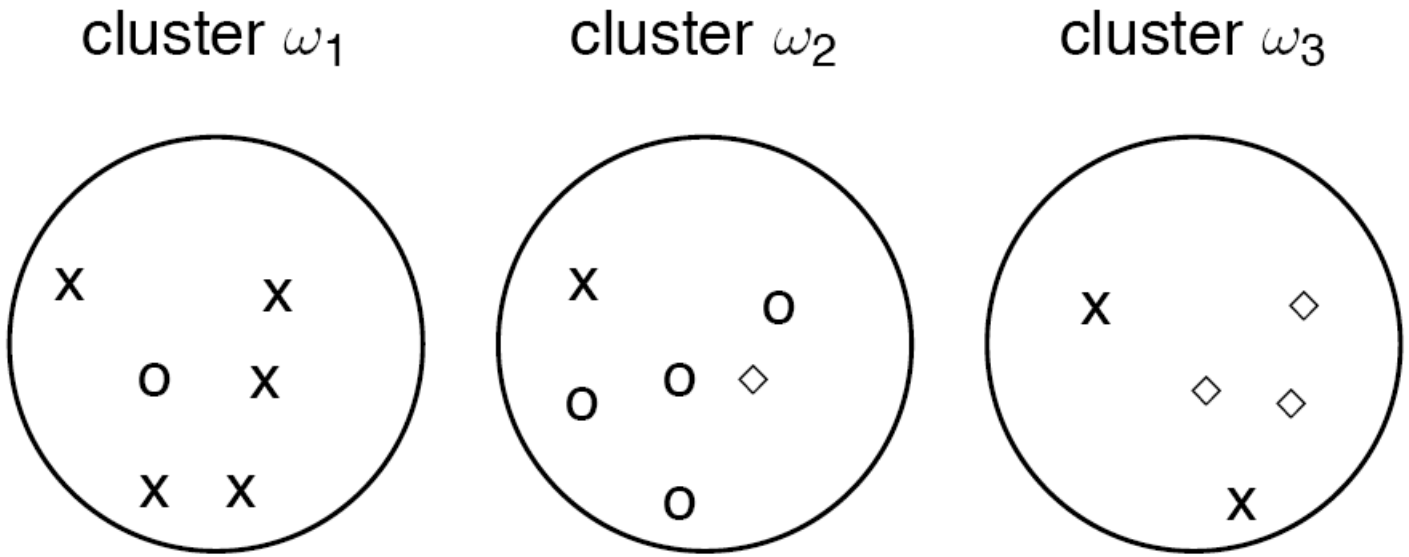
$$\text{purity}(\Omega, \Gamma) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$$

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

For each cluster ω_k : find class c_j with most members n_{kj} in cluster

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

Example

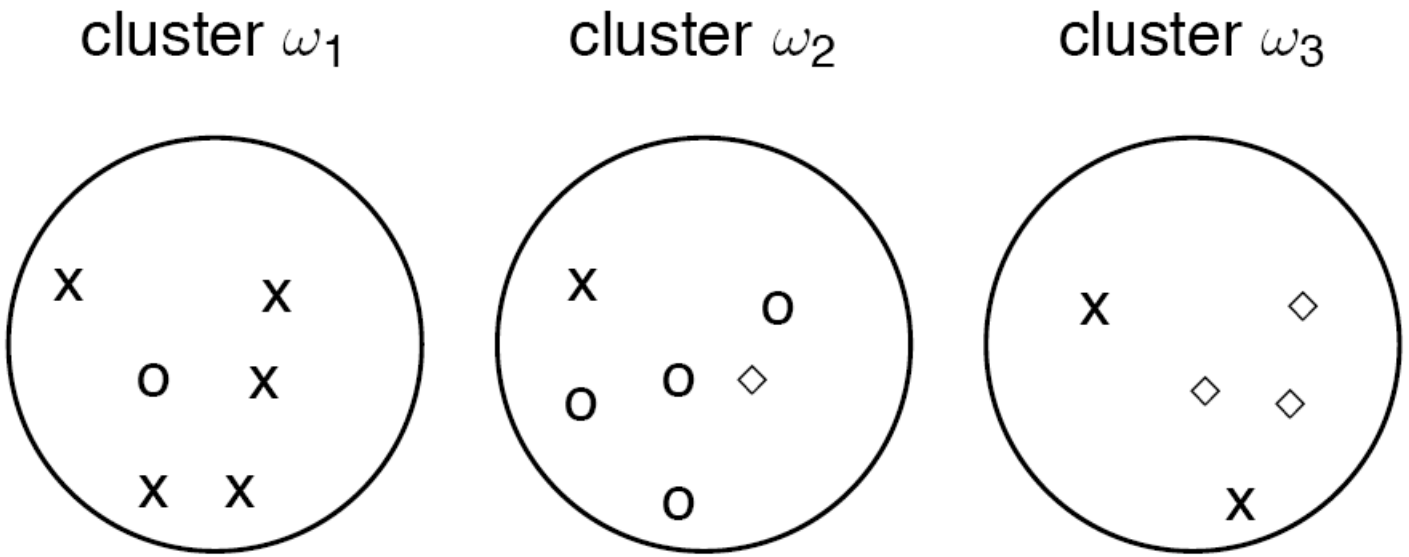


$$\text{good\_docs}(\omega_1) = \max(5,1,0) = 5$$
$$\text{good\_docs}(\omega_2) = \max(1,4,1) = 4$$
$$\text{good\_docs}(\omega_3) = \max(2,0,3) = 3$$
$$\text{purity}(\Omega) = 1/17 \cdot (5+4+3) = 12/17$$

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).

Learning to Rank
(Machine Learned-Ranking)

Conventional Ranking Models

- Content relevance
  - Boolean, vector space, language model, ...
- Page importance
  - Link analysis: PageRank, HITS, ...
  - Query log mining, clickthroughs, …

Machine learning for IR ranking?

- There's a large body of work in machine learning
- Surely we can also use machine learning to rank the documents displayed in search results?
  - Sounds like a good idea
  - => "machine-learned relevance" or "learning to rank"

What is Machine Learning?

- Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence.
- Machine learning is a "Field of study that gives computers the ability to learn without being explicitly programmed".

Company Info

Follow

Get job updates from eBay Inc.

eBay Inc.

982 reviews

Founded in 1995 in San Jose, Calif., eBay (NASDAQ:EBAY) is where the world goes to shop, sell and give. Whether you're buying new or used,...

Data Scientist/Applied Researcher

eBay Inc. ★★★★☆ 982 reviews - San Jose, CA

Looking for a company that inspires passion, courage and imagination, where you can be part of the team shaping the future of global commerce? Want to shape how millions of people buy, sell, connect, and share around the world? If you're interested in joining a purpose driven community that is dedicated to creating an ambitious and inclusive workplace, join eBay – a company you can be proud to be a part of.

Looking for a company that inspires passion, courage and imagination, where you can be part of the team shaping the future of global commerce? Want to shape how millions of people buy, sell, connect, and share around the world? If you're interested in joining a purpose driven community that is dedicated to creating an ambitious and inclusive workplace, join eBay – a company you can be proud to be a part of.

Join the Search Science team at eBay!

Do you have what it takes to improve a world-class real-time search engine that serves millions of queries a day? Do you thrive on developing data mining techniques to pull insight
eCommerce re
transaction da

We are passio
online marketp
scientists

You will

- Be directly responsible for improving search recall and ranking via query understanding and machine learned ranking models that power the end-end search experience at eBay.
- Showcase the spectrum of value in the world's most diverse ecommerce inventory, by improving whole page relevance, query recommendations, spell corrections, synonym/acronym expansions, and query rewrites.
- Conceptualize, code, deploy, and iterate on designs from prototypes all the way through to production systems.
- Analyze petabytes of real-world behavioral data to understand patterns and trends.
- Transform data insights into actionable reports, targeting algorithms, and model features.
- Provide technical leadership in statistical analysis, data mining, machine learning, NLP, and information retrieval.

Company Info

Follow

Get job updates from Apple

Apple

4.4 5,066 reviews

This is where you can do the best work of your life. Where you'll join some of the world's smartest, most innovative people to create...

Siri - Data Scientist

Apple 4.4 5,066 reviews - San Francisco, CA 94114

The Siri Search team is building groundbreaking technology for algorithmic search, machine learning, natural language processing, and artificial intelligence. The features we build are redefining how hundreds of millions of people use their computers and mobile devices to search and find what they are looking for. Siri's universal search engine powers search features across a variety of Apple products, including Siri, Spotlight, Safari, Messages and Lookup. As part of this group, you will work with one of the most exciting high performance computing environments, with petabytes of data, millions of queries per second, and have an opportunity to imagine and build products that delight our customers every single day.

Key Qualifications

- 3+ years of experiences in a Machine Learning or a Data Science role
- You have excellent knowledge and good practical skills in major machine learning algorithms
- You have experience with machine learning tools and libraries such as Scikit-learn, TensorFlow, Spark
- You have excellent data analytical and problem solving skills
- Mastery of one of the following languages: Python, Go, Java, C++
- Your experience with large scale search and machine learning systems is helpful
- Good
- Good

Description

As a member of our fast-paced group, you'll have the unique and rewarding opportunity to shape upcoming products from Apple. We are looking for people with excellent applied machine learning experience and solid engineering skills in creating outstanding search service. This role will have the following responsibilities: Analyzing search ranking requirements, issues and opportunities Understanding product requirements, translate them into modeling tasks and engineering tasks Building machine learned models for search relevance, ranking and content understanding problems Integrating search functions into Apple products, such as Siri, Spotlight, Safari, Messages, Lookup, etc. Utilizing Spark, Hadoop MapReduce, HBase, Hive, Impala, Kafka, and RabbitMQ to perform distributed data processing Work alone or as part of small team to deliver complete systems Work project management and other engineering teams

Learning to rank algorithms

Least Square Retrieval Function (TOIS 1989)

Query refinement (WWW 2008)

SVM-MAP (SIGIR 2007)

ListNet (ICML 2007)

Pranking (NIPS 200

LambdaRank (NIPS 2006)

Frank (SIGIR

MHR (SIGIR 2007)

RankBoost (JMLR 2003)

LDM (SIGI

Large margin ranker (NIPS 2002)

RankNet (ICML 2005)

Ranking SVM (ICANN 199

Discriminative model for IR (SIGIR 2004)

OAP-BPM (ICML 2003)

GPRank (LR4IR 2007)

QBRank (NIPS 2007)

GBRar

Constraint Ordinal Regression (ICML 2005)

McRank (NIPS 2

AdaRank (SIGIR 2007)

CCA (SIGIR 2007)

RankCosine (IP&M 2007)

Supervised Rank

Year	Name
2008	ListMLE
2008	PermuRank
2008	SoftRank
2008	Ranking Refinement [24]
2008	SSRankBoost [25]
2008	SortNet [26]
2009	MPBoost
2009	BoltzRank
2009	BayesRank
2010	NDCG Boost [27]
2010	GBlend
2010	IntervalRank
2010	CRR
2014	LCR
2015	FaceNet
2016	XGBoost
2017	ES-Rank
2018	DLCM [28]
2018	PolyRank [29]
2018	FATE-Net/FETA-Net [30]
2019	FastAP [31]
2019	Mulberry
2019	DirectRanker
2019	GSF [32]
2020	PRM [33]
2020	SetRank [34]
2021	PiRank [35]

https://en.wikipedia.org/wiki/Learning_to_rank

Simple example: Using classification for ad hoc IR

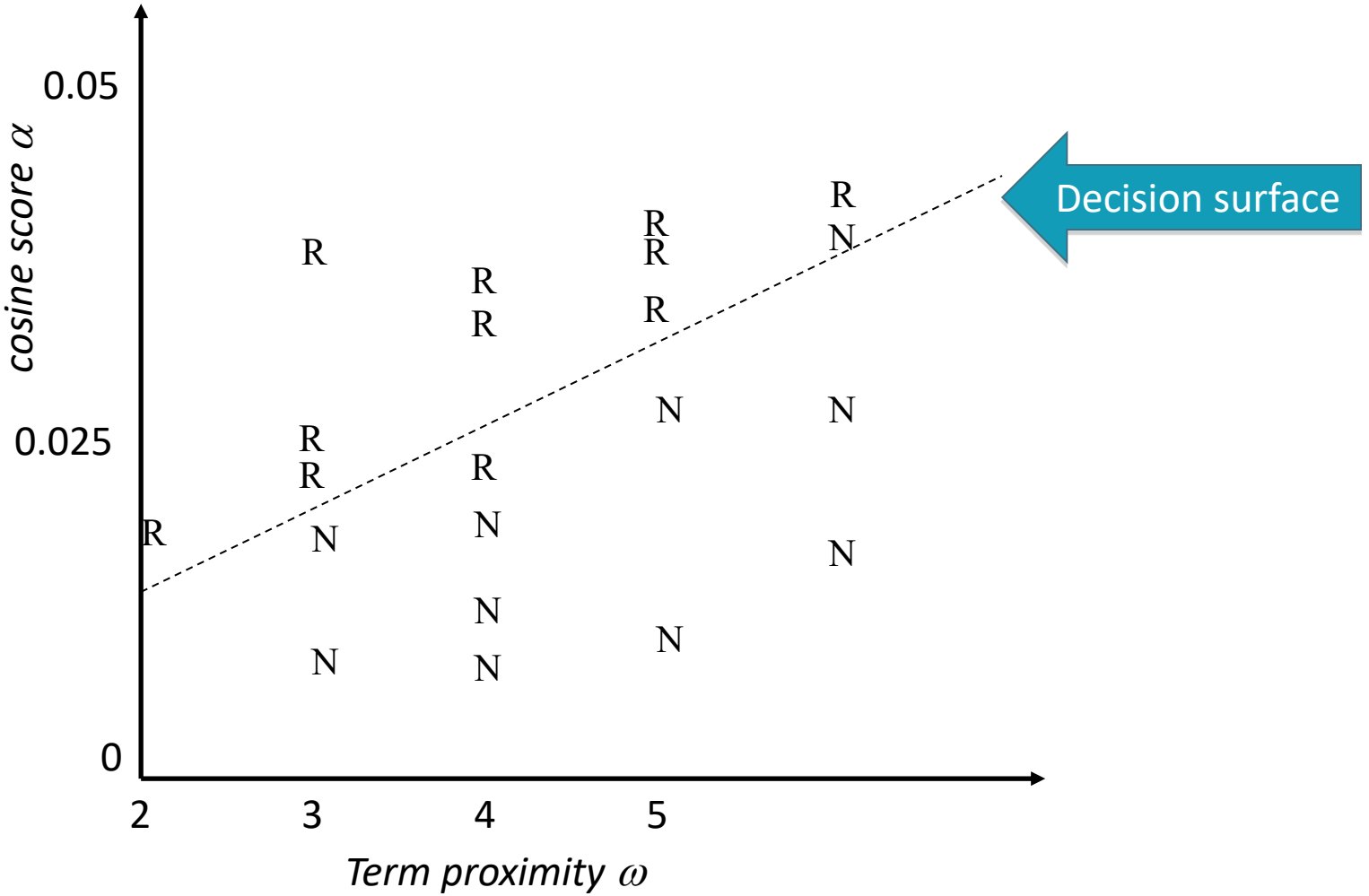
- Collect a training corpus of (*q, d, r*) triples
  - Relevance *r* is here binary
  - Document is represented by a feature vector
  - **x** = (α, ω): α is cosine similarity, ω is minimum query window size
    - Query term proximity is a **very important** weighting factor
- Train a machine learning model to predict the class *r* of a document-query pair

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	*relevant*
Φ_2	37	penguin logo	0.02	4	*nonrelevant*
Φ_3	238	operating system	0.043	2	*relevant*
Φ_4	238	runtime environment	0.004	2	*nonrelevant*
Φ_5	1741	kernel layer	0.022	3	*relevant*
Φ_6	2094	device driver	0.03	2	*relevant*
Φ_7	3191	device driver	0.027	5	*nonrelevant*

Simple example: Using classification for ad hoc IR

- A linear score function is then
  - *Score(d, q) = Score(α, ω) = aα + bω + c*
- And the linear classifier is
  - Decide relevant if *Score(d, q) > threshold*
- … this is exactly like text classification

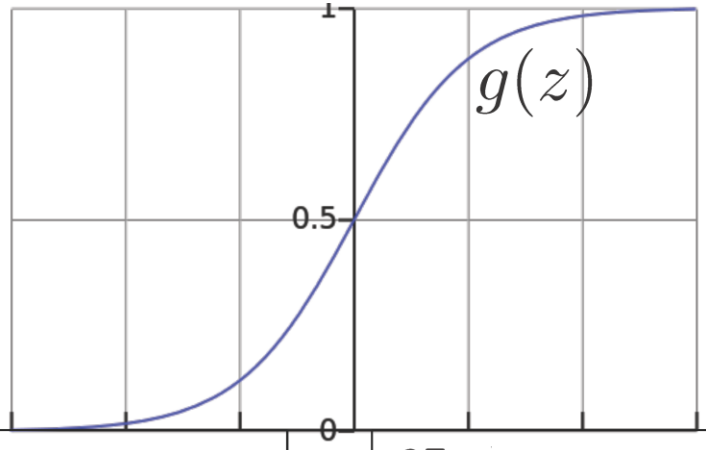
Simple example: Using classification for ad hoc IR



e.g., Logistic Regression

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

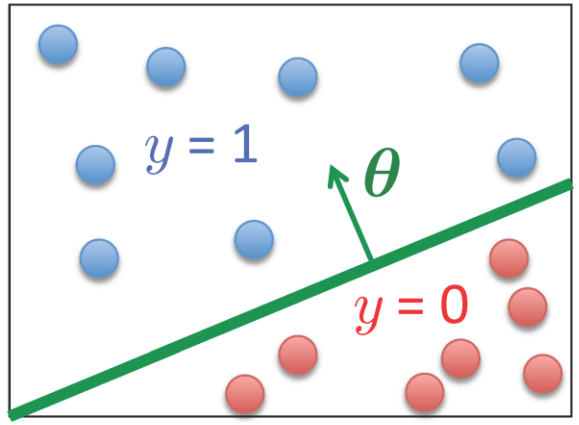
$$g(z) = \frac{1}{1+e^{-z}}$$



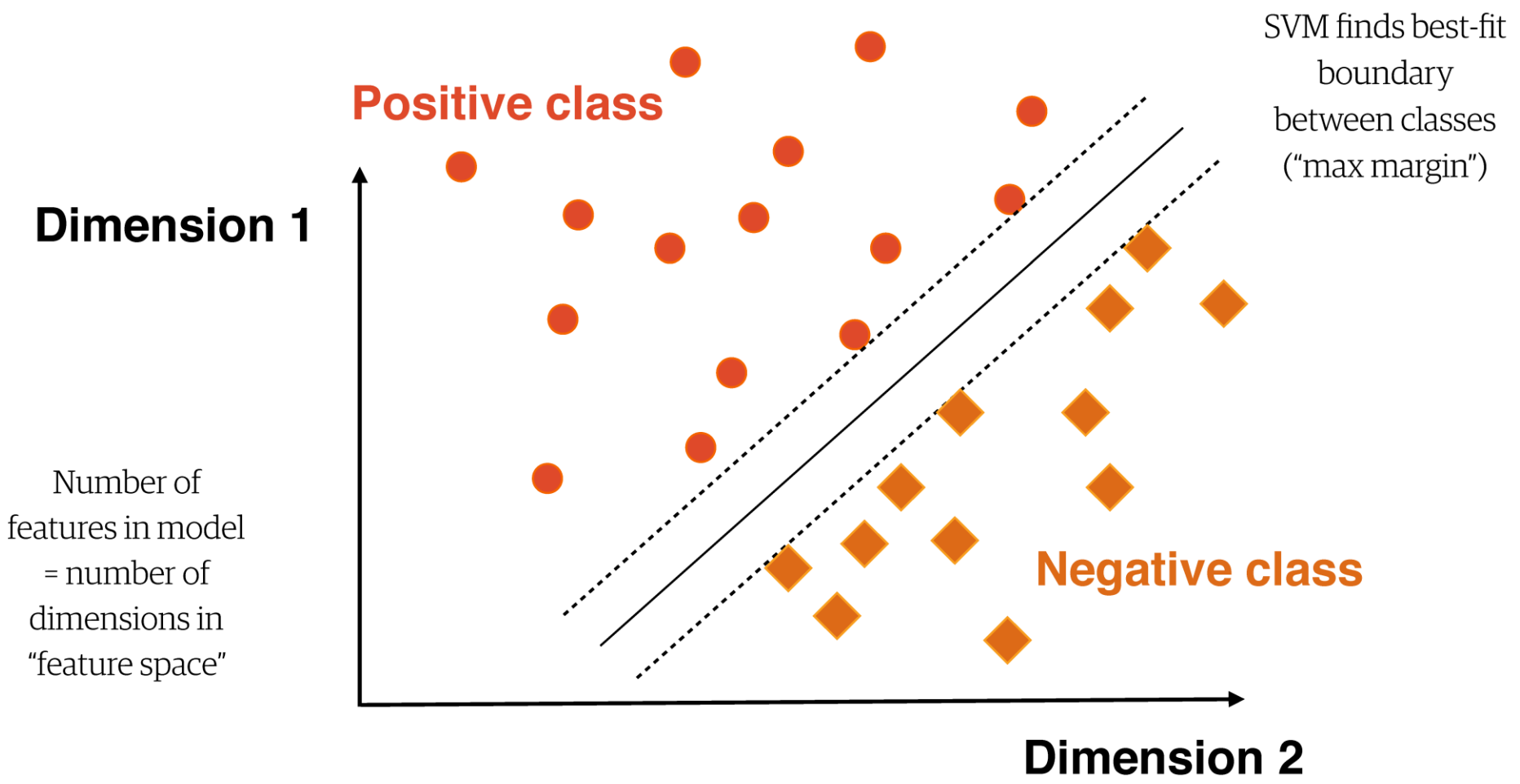
$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large negative values for negative instances

$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large positive values for positive instances

- Assume a threshold and...
  - Predict $y = 1$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0.5$
  - Predict $y = 0$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$



e.g., Support Vector Machine (SVM)



Extending the model

- We can generalize this to classifier functions over more features

- Machine learning for IR ranking has been actively researched – and actively deployed by major web search engines

Why is ML needed now?

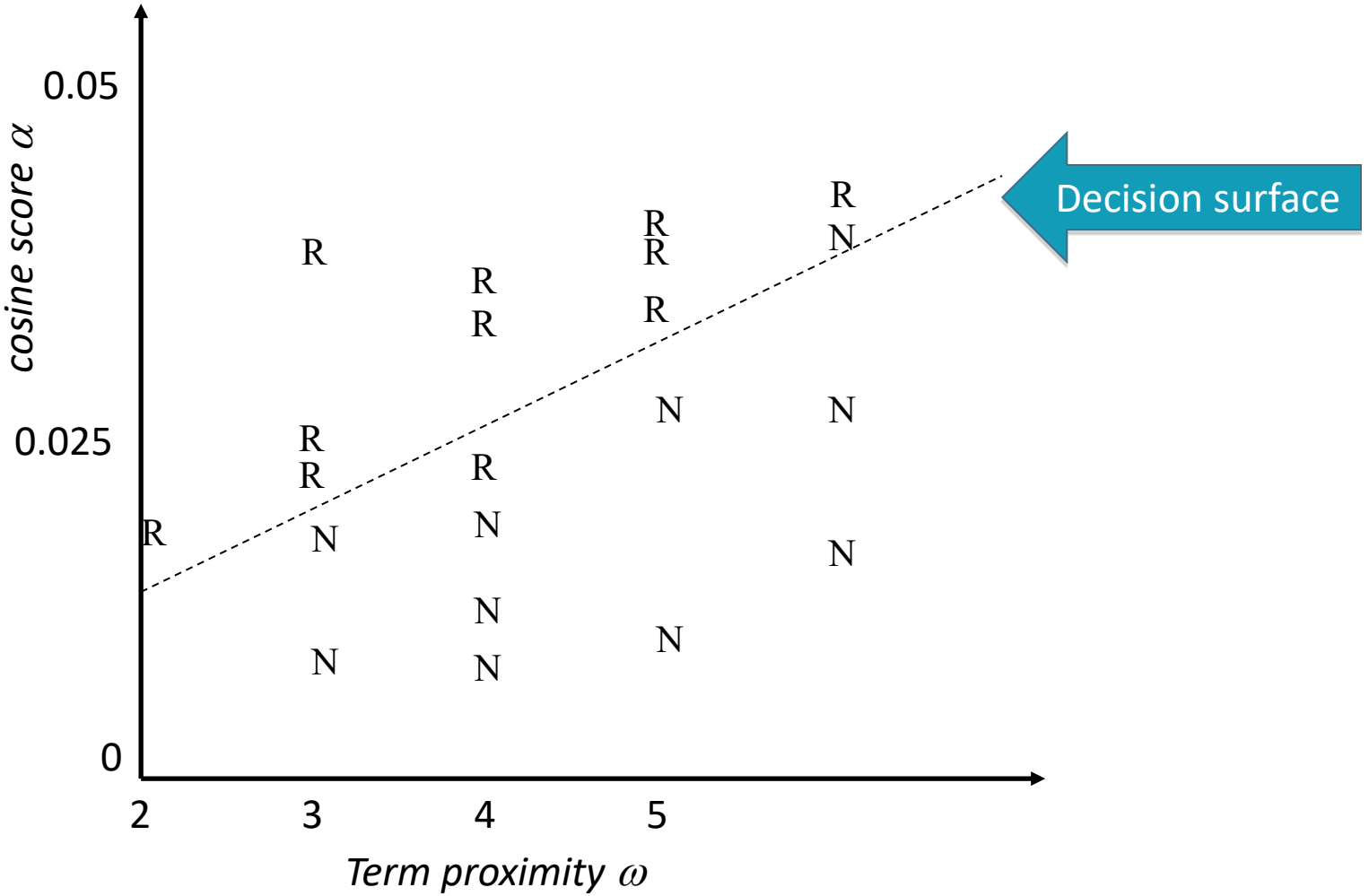
- Modern systems – especially on the Web – use a great number of features:
  - Arbitrary useful features – not a single unified model
  - Log frequency of query word in anchor text?
  - Query word in color on page?
  - # of images on page?
  - # of (out) links on page?
  - PageRank of page?
  - URL length?
  - URL contains "~"?
  - Page edit recency?
  - Page length?
- The *New York Times* (2008-06-03) quoted Amit Singhal as saying Google was using **over 200 such features**.

136 Features released from Microsoft Research in 2010

http://research.microsoft.com/en-us/projects/mslr/feature.aspx

Zones: body, anchor, title, url, whole document
Features: query term number, query term ratio, stream length, idf, sum of term frequency, min of term frequency, max of term frequency, mean of term frequency, variance of term frequency, sum of stream length normalized term frequency, min of stream length normalized term frequency, max of stream length normalized term frequency, mean of stream length normalized term frequency, variance of stream length normalized term frequency, sum of tf*idf, min of tf*idf, max of tf*idf, mean of tf*idf, variance of tf*idf, boolean model, vector space model, BM25, LMIR.ABS, LMIR.DIR, LMIR.JM, number of slash in url, length of url, inlink number, outlink number, PageRank, SiteRank, QualityScore, QualityScore2, query-url click count, url click count, url dwell time.

Simple example: Using classification for ad hoc IR



Result ranking by machine learning

- The above ideas can be readily generalized to functions of **many more than two** variables
- In addition to cosine similarity and query term window, there are lots of other indicators of relevance, e.g. PageRank-style measures, document age, zone contributions, document length, etc.
- If these measures can be calculated for a training document collection with relevance judgments, any number of such measures can be used to train a machine learning classifier

Learning to rank

- Purpose
  - Learn a function automatically to rank results (items) effectively
- Point-wise learning
  - The function is based on features of a single object
  - e.g., regress the relevance score, classify docs into R and NR
  - Classic retrieval models are also point-wise: score(q, D)
- Pair-wise learning
  - The function is based on a pair of items
  - e.g., given two documents, predict partial ranking
- List-wise learning
  - The function is based on a ranked list of items
  - e.g., given two ranked list of the same items, which is better? Or can we learn cross-document interactions and learn the best order of the documents.

https://en.wikipedia.org/wiki/Learning_to_rank

Learning to rank

- Classification probably isn't the right way to think about approaching ad hoc IR:
  - Classification problems: Map to a unordered set of classes
  - Regression problems: Map to a real value
  - Ordinal regression problems: Map to an *ordered* set of classes
    - A fairly obscure sub-branch of statistics, but what we want here
- This formulation gives extra power:
  - Relations between relevance levels are modeled
  - Documents are good versus other documents for query given collection; not an absolute scale of goodness

Learning to rank

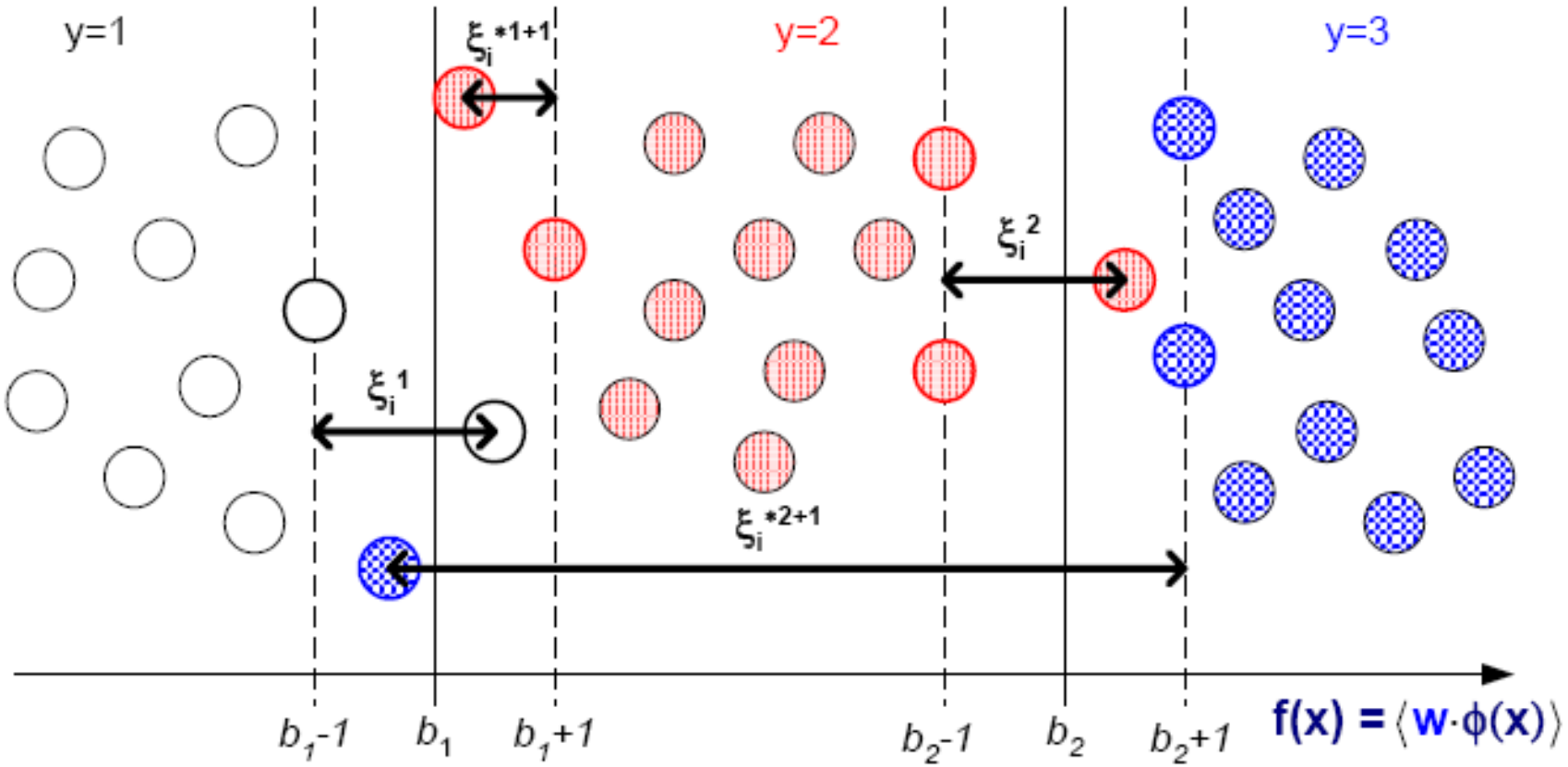
- Assume a number of categories ***C*** of relevance exist
  - These are totally ordered: $c_1 < c_2 < \ldots < c_J$
  - This is the ordinal regression setup

- Assume training data is available consisting of document-query pairs represented as feature vectors ψ_i and relevance ranking c_i

- We could do ***point-wise*** **learning**, where we try to map items of a certain relevance rank to a subinterval (e.g, Crammer et al. 2002 Prank)

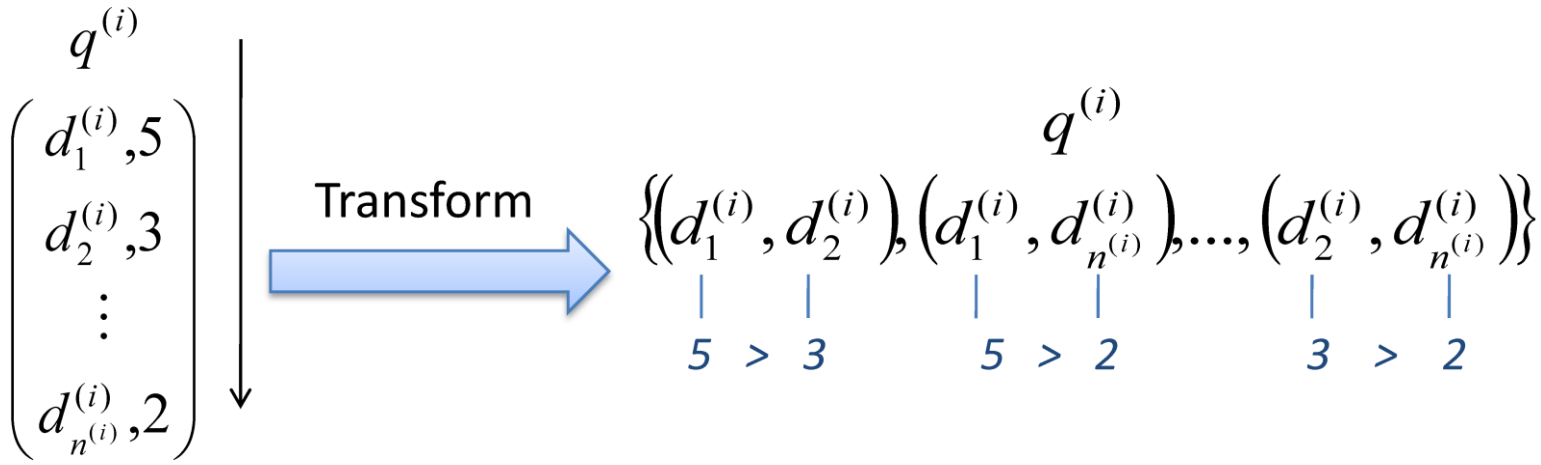
Point-wise learning

- Goal is to learn a threshold to separate each rank



Learning to rank

- But most work does ***pair-wise* learning**, where the input is a pair of results for a query, and the class is the relevance ordering relationship between them



An Example of the Pairwise approach: The Ranking SVM

- We begin with a set of judged queries
- For each training query q, we have a set of documents returned in response to the query, which have been totally ordered by a person for relevance to the query
- We construct a vector of features $\psi_j = \psi(d_j , q)$ for each document/query pair, using features such as those discussed, and many more
- For two documents d_i and d_j , we then form the vector of feature differences:

$$\Phi(d_i , d_j , q) = \psi(d_i , q) - \psi(d_j , q)$$

The construction of a ranking SVM

- By hypothesis, one of d_i and d_j has been judged more relevant
- If d_i is judged more relevant than dj , denoted $d_i \prec d_j$ (d_i should precede d_j in the results ordering), then we will assign the vector $\Phi(d_i , d_j , q)$ the class $y_{ijq} = +1$; otherwise -1
- The goal then is to build a classifier which will return

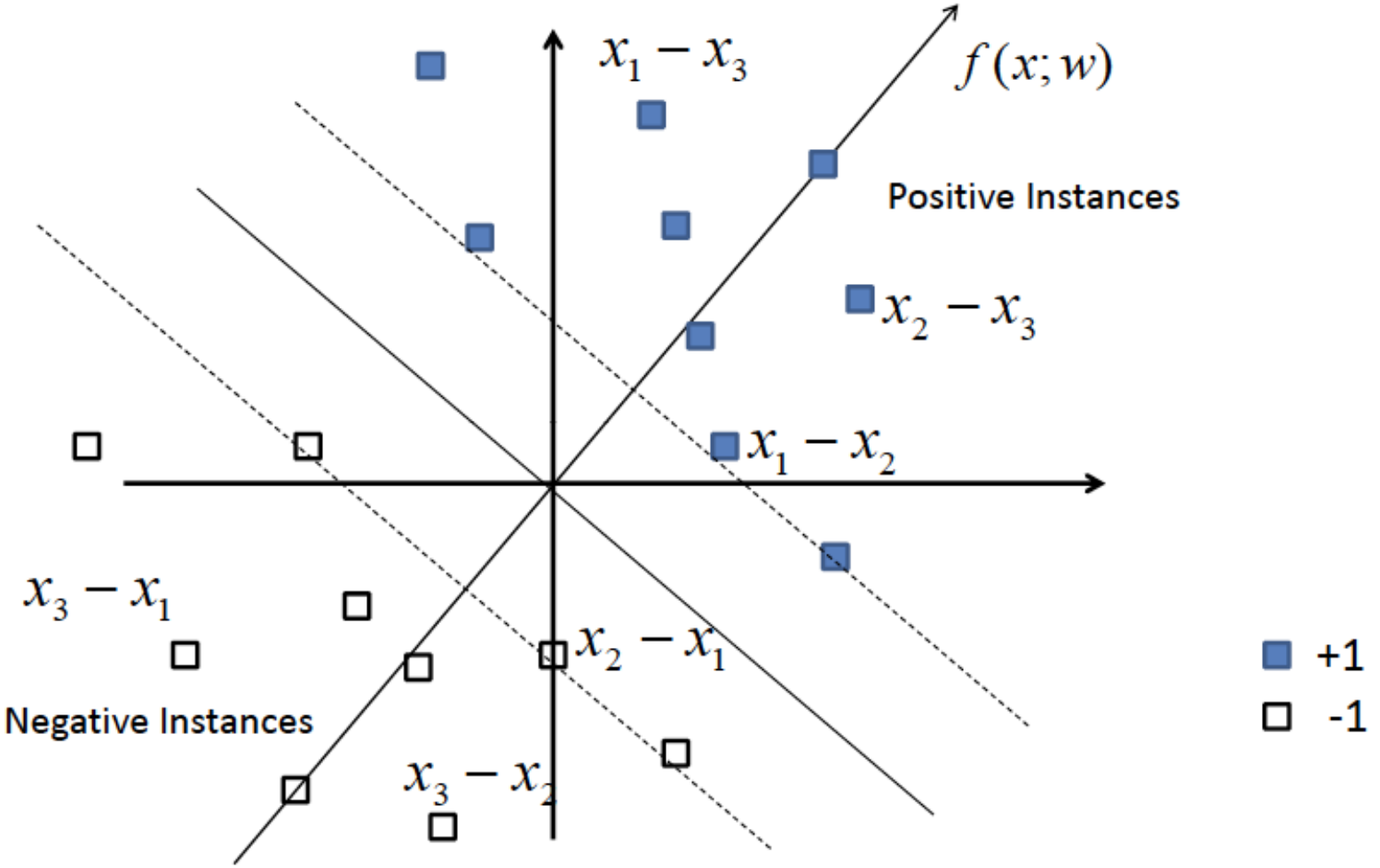
$$\mathbf{w} \bullet \Phi(d_i , d_j , q) > 0 \text{ iff } d_i \prec d_j$$

Testing the Ranking SVM

- Input: a pair of documents Di and Dj
  - Characterized by feature vectors ψ_i *and* ψ_j

- Classify:
  - *rank i before j* iff $\mathbf{w} \bullet (\psi_i - \psi_j) > 0$

An Example of the Pairwise approach: The Ranking SVM



Ranking SVM

- This approach has been used to build ranking functions which outperform standard hand-built ranking functions in IR evaluations on standard data sets

The ranking classifier fails to model the IR problem well …

- Correctly ordering the most relevant documents is crucial to the success of an IR system, while misordering less relevant results matters little
  - The ranking SVM considers all ordering violations as the same
- Some queries have many (somewhat) relevant documents, and other queries few. If we treat all pairs of results for a query equally, queries with many results will dominate the learning
  - But actually queries with few relevant results are at least as important to do well on

https://en.wikipedia.org/wiki/Learning_to_rank

SetRank: Learning a Permutation-Invariant Ranking Model for Information Retrieval -- An example of list-wise learning

- Given a candidate documents per query by a traditional ranking method (e.g., BM25 and LambdaMART), rerank the candidate documents. LambdaMART
- Consider two requirements:
  - (1) Model cross-document interactions so as to capture local context information in a query
  - (2) Be permutation invariant, which means that any permutation of the inputted documents would not change the output ranking.

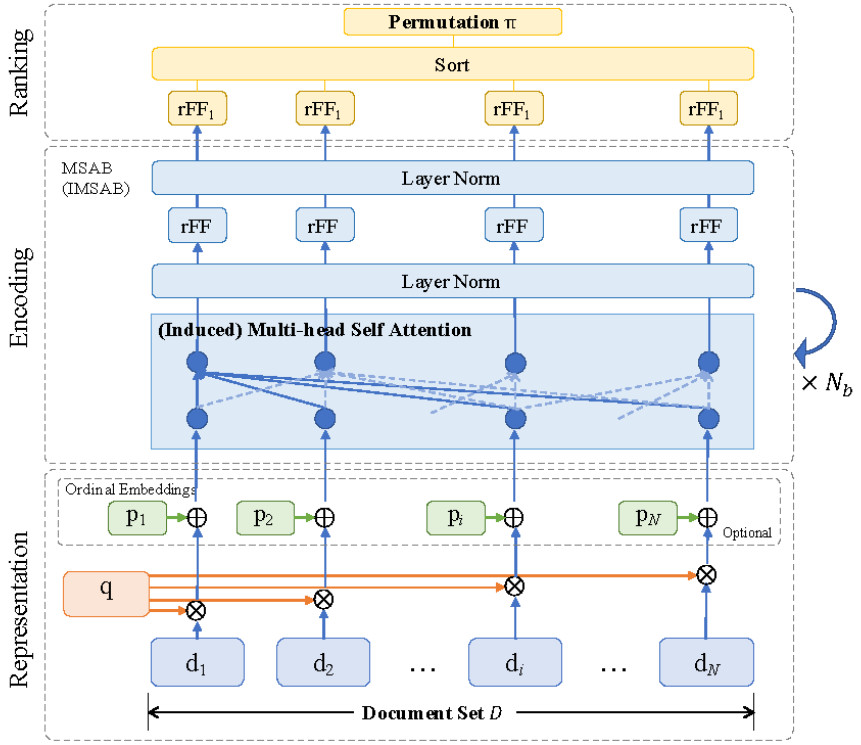


Figure 1: Architecture of SetRank. The representation layer generates representation of query-document pairs separately; the encoding layer jointly process the documents with multiple sub-layers of MSAB (or IMSAB), and output internal document representations; the ranking layer calculates the scores and sorts the documents.

https://arxiv.org/abs/1912.05891

1. Document Representation

Given a query q and its associated document set $D = [d_1, d_2, \cdots, d_N]$, each of the document in D can be represented as a feature vector

$$\mathbf{d}_i = \phi(q, d_i), \text{ where } \mathbf{d}_i \in \mathbb{R}^E,$$

where ϕ is the function for feature extraction and E is the dimension of the vector. The features extracted in traditional learning-to-rank are used here, including document only feature of PageRank, query-document matching features of TF-IDF, BM25 etc. In the experiments of this paper, we used the features provided by the benchmark datasets.

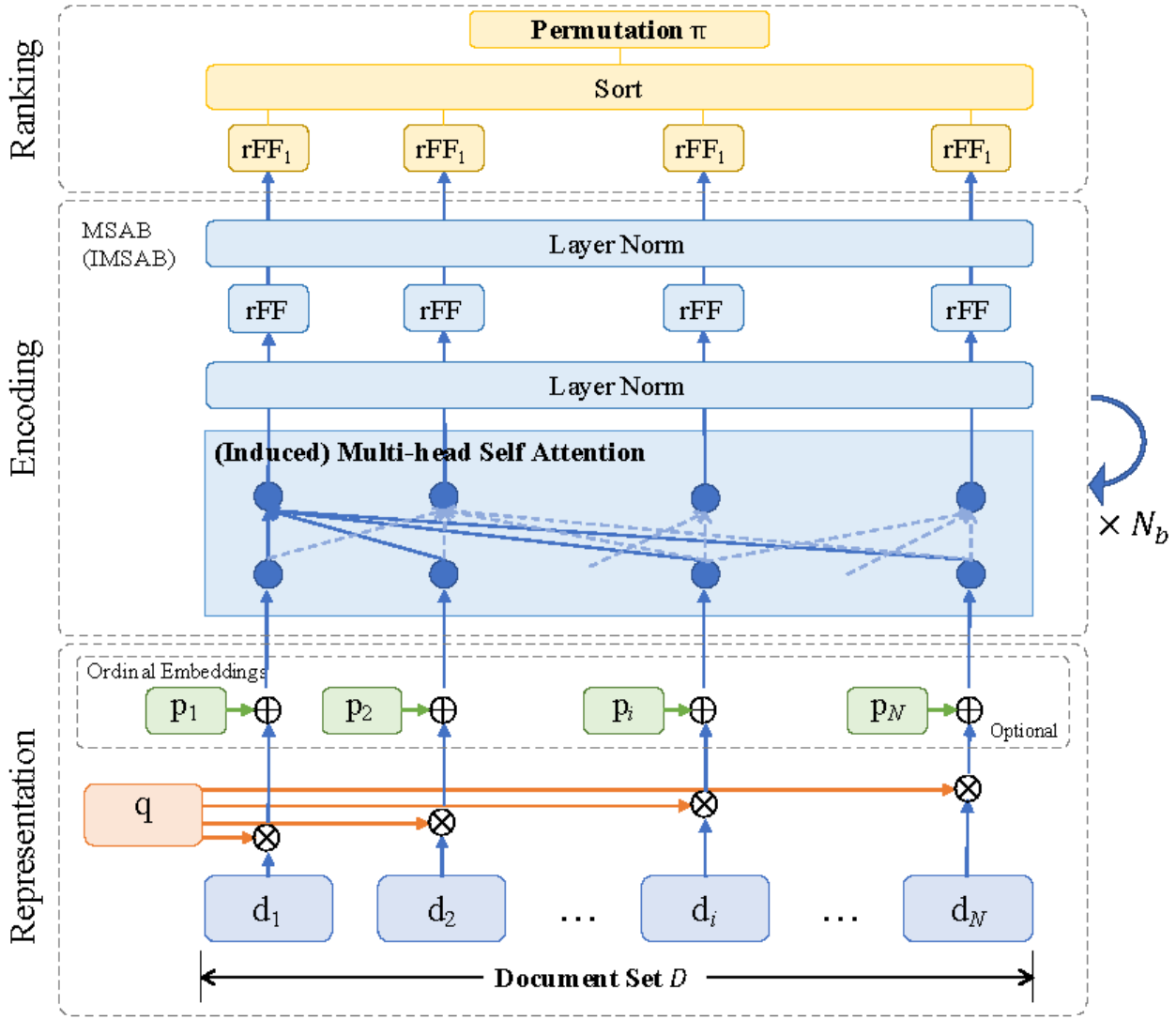
$$\mathbf{p}_i = P(\text{rank}(d_i)), \text{ where } \mathbf{p}_i \in \mathbb{R}^E,$$

where $\text{rank}(d_i)$ denotes the absolute rank position of d_i in the initial ranking generated by models such as BM25, LambdaMART etc.

Output of document representation

$$\mathbf{X} = [\mathbf{d}_1 + \mathbf{p}_1, \mathbf{d}_2 + \mathbf{p}_2, \cdots, \mathbf{d}_N + \mathbf{p}_N]^T.$$

$$\mathbf{X} \in \mathbb{R}^{N \times E}$$



2. Document Encoding with (Induced) Multi-head Self Attention Block

Multi-head attention block (MAB)

$$\begin{aligned} \mathrm{MAB}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= \mathrm{LayerNorm}(\mathbf{B} + \mathrm{rFF}(\mathbf{B})), \\ \text{where } \mathbf{B} &= \mathrm{LayerNorm}(\mathbf{Q} + \mathrm{Multihead}(\mathbf{Q},\mathbf{K},\mathbf{V})) \end{aligned} \tag{5}$$

where $\mathrm{rFF}(\cdot)$ is a row-wise feedforward layer, and $\mathrm{LayerNorm}(\cdot)$ is layer normalization [3].

Two approaches:

a) Multi-head self attention block (MSAB)

$$\mathrm{MSAB}(\mathbf{X}) = \mathrm{MAB}(\mathbf{X},\mathbf{X},\mathbf{X}).$$

b) Induced multi-head self attention block (IMSAB)

First, construct M fake attention query vectors, denoted as, $\mathbf{I} \in \mathbb{R}^{M \times E}$, to extract information from original keys/values. M is less than N (# of candidate docs)

$$\mathrm{IMSAB}_M(\mathbf{X}) = \mathrm{MAB}(\mathbf{X},\mathbf{H},\mathbf{H}), \text{ where } \mathbf{H} = \mathrm{MAB}(\mathbf{I},\mathbf{X},\mathbf{X})$$

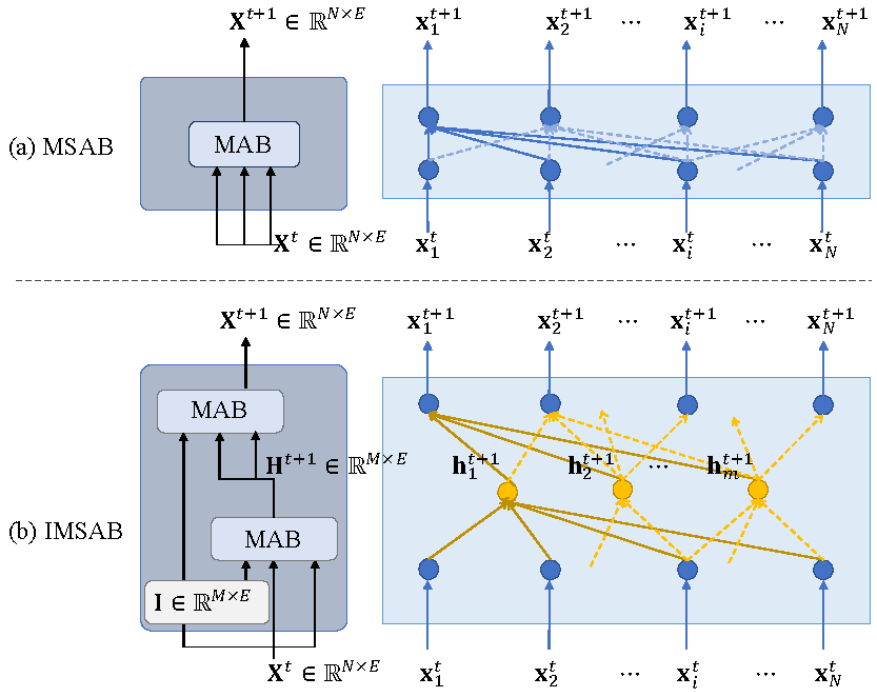


Figure 2: (a) The structure of multi-head self attention block (MSAB). MSAB encodes an arbitrary set $\mathbf{X}^t$ of size N and output an set $\mathbf{X}^{t+1}$ of size N. (b) The structure of induced multi-head self attention block (IMSAB) with induced size of M. IMSAB first encodes an arbitrary set $\mathbf{X}^t$ of size N to a fixed set $\mathbf{H}^{t+1}$ of size M, and then encodes $\mathbf{H}^{t+1}$ to an N-size output $\mathbf{X}^{t+1}$. IMSAB can be interpreted as first creating M cluster centers and then encoding the inputted N documents using these M cluster centers.

3. Document Ranking

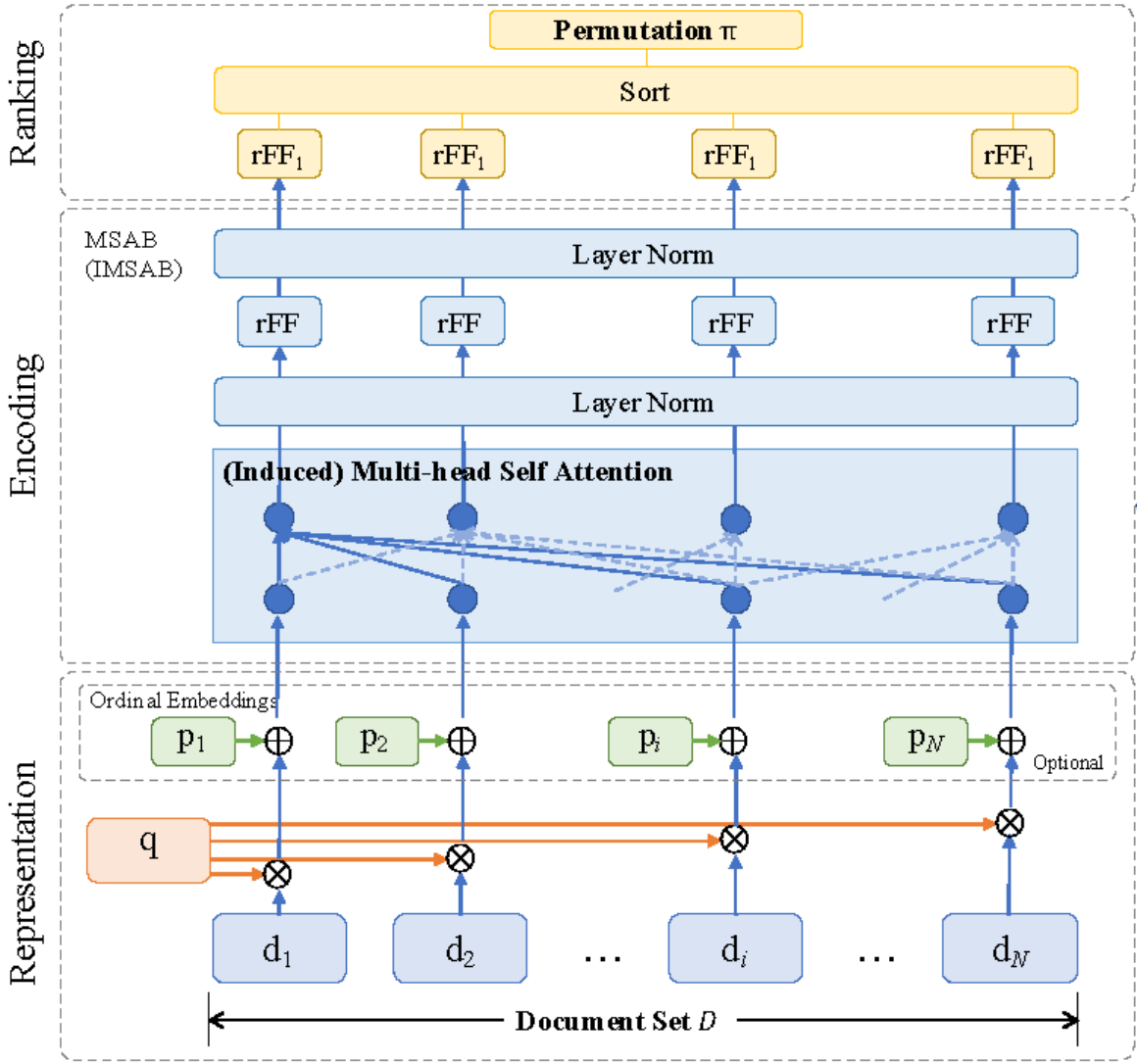
Stacking aforementioned MSAB or IMSABM blocks and passing the results to row-wise feedforward neural networks, we achieve two versions of SetRank scoring functions, respectively named as SetRankMSAB and SetRankIMSAB:

$$\mathbf{X}_{\mathrm{MSAB}}^{N_b} = \underbrace{\mathrm{MSAB}(\mathrm{MSAB}\ldots(\mathrm{MSAB}}_{N_b}(\mathbf{X}^0))),$$

$$\mathrm{SetRank}_{\mathrm{MSAB}}(D) = \mathrm{rFF}_1\left(\mathbf{X}_{\mathrm{MSAB}}^{N_b}\right),$$

Finally, sort each doc's score

$$\hat{\pi}_{\mathrm{MSAB}} = sort \circ \mathrm{SetRank}_{\mathrm{MSAB}}(D).$$



Training and Datasets

- Model Training

Given a labeled query $\psi_q = \{D = \{d_i\}, \mathbf{y} = \{y_i\} | 1 \le i \le N\}$, where y_i is the relevance label and denotes the information gain of document d_i for query q. The optimal attention allocation strategy in terms of relevance labels for document $d_i (i = 1, \cdots, N)$ is:

$$a_i^y = \frac{\tau(y_i)}{\sum_{d_k \in D} \tau(y_k)}, \quad \tau(x) = \begin{cases} \exp(x) & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Similarly, given the predicted ranking scores $\{s_1, \cdots, s_N\} = \text{SetRank}(D)$, the attention w.r.t. the predicted scores for document $d_i (i = 1, \cdots, N)$ is:

$$a_i^s = \exp(s_i) / \sum_{d_k \in D} \exp(s_k).$$

Therefore, the list-wise cross entropy loss function can be constructed on the basis of these two attention distributions:

$$\mathcal{L} = \sum_{d_i \in D} a_i^y \log a_i^s + (1 - a_i^y) \log(1 - a_i^s).$$

Table 1: Dataset statistics.

Property	Istella	MSLR 30K	Yahoo!
#features	220	136	700
#queries in training	20,317	18,919	19,944
#queries in validation	2,902	6,306	2,994
Total #docs in train	7,325,625	2,270,296	473,134
#queries in test	9,799	6,306	6,983
Total #docs in test	3,129,004	753,611	165,660
Avg. #docs per query in test	319.31	119.51	23.72

Results

Table 2: Performance comparison of different models on Istella, MSLR30K and Yahoo datasets. Significant performance improvements (paired t-test with p-value ≤ 0.05) over LambdaMart and DLCM are denoted as '+' and '†', respectively. Boldface indicates the best performed results.

(a) Ranking accuracies on Istella LETOR dataset

	NDCG			
Model	@1	@3	@5	@10
RankSVM	0.5269	0.4867	0.5041	0.5529
RankBoost	0.4457	0.3977	0.4097	0.4511
Mart	0.6185	0.5633	0.5801	0.6285
LambdaMart	0.6571	0.5982	0.6118	0.6591
Without initial rankings				
DLCM$^{w/o\ init}$	0.6272	0.5717	0.5848	0.6310
GSF	0.6224	0.5796	0.5968	0.6508
SetRank$_{\text{MSAB}}$	0.6702$^{+\dagger}$	0.6150$^{+\dagger}$	0.6282$^{+\dagger}$	0.6766$^{+\dagger}$
SetRank$_{\text{IMSAB}}$	0.6733$^{+\dagger}$	0.6136$^{+\dagger}$	0.6278$^{+\dagger}$	0.6737$^{+\dagger}$
With initial rankings generated by LambdaMart				
DLCM	0.6558	0.6030^{+}	0.6194^{+}	0.6680^{+}
SetRank$^{init}_{\text{MSAB}}$	0.6745$^{+\dagger}$	0.6201$^{+\dagger}$	**0.6350**$^{+\dagger}$	0.6819$^{+\dagger}$
SetRank$^{init}_{\text{IMSAB}}$	**0.6760**$^{+\dagger}$	**0.6202**$^{+\dagger}$	0.6345$^{+\dagger}$	**0.6834**$^{+\dagger}$

(b) Ranking accuracies on Microsoft LETOR 30K dataset

	NDCG			
Model	@1	@3	@5	@10
RankSVM	0.3010	0.3180	0.3350	0.3650
RankBoost	0.2788	0.2897	0.3043	0.3339
Mart	0.4436	0.4344	0.4414	0.4633
LambdaMart	0.4570	0.4420	0.4450	0.4640
Without initial rankings				
DLCM$^{w/o\ init}$	0.3985	0.3919	0.4001	0.4245
GSF	0.4129	0.4073	0.4151	0.4374
SetRank$_{\text{MSAB}}$	0.4243	0.4116	0.4177	0.4403
SetRank$_{\text{IMSAB}}$	0.4290	0.4166	0.4220	0.4428
With initial rankings generated by LambdaMart				
DLCM	**0.4630**$^{+}$	0.4450^{+}	0.4500^{+}	0.4690^{+}
SetRank$^{init}_{\text{MSAB}}$	0.4572	0.4452^{+}	0.4499^{+}	0.4692^{+}
SetRank$^{init}_{\text{IMSAB}}$	0.4591^{+}	**0.4469**$^{+\dagger}$	**0.4515**$^{+}$	**0.4696**$^{+}$

(c) Ranking accuracies on Yahoo! LETOR challenge set1 dataset

	NDCG			
Model	@1	@3	@5	@10
RankSVM	0.6370	0.6500	0.6740	0.7260
RankBoost	0.6293	0.6409	0.6661	0.7159
Mart	0.6830	0.6827	**0.7034**	**0.7469**
LambdaMart	0.6770	0.6760	0.6960	0.7380
Without initial rankings				
DLCM$^{w/o\ init}$	0.6693	0.6751	0.6958	0.7391
GSF	0.6429	0.6604	0.6838	0.7316
SetRank$_{\text{MSAB}}$	0.6623	0.6698	0.6911	0.7369
SetRank$_{\text{IMSAB}}$	0.6711	0.6760	0.6960	0.7398
With initial rankings generated by LambdaMart				
DLCM	0.6760	0.6810^{+}	0.6990^{+}	0.7430^{+}
SetRank$^{init}_{\text{MSAB}}$	**0.6837**$^{+\dagger}$	0.6820^{+}	0.7009^{+}	0.7443^{+}
SetRank$^{init}_{\text{IMSAB}}$	0.6822$^{+\dagger}$	**0.6835**$^{+\dagger}$	0.7029^{+}	0.7453$^{+\dagger}$

Without initial rankings mean without incorporating ordinal embedding. Simple $\mathbf{X} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \ldots, \mathbf{d}_N]$