Day-98, (Falgun 24, 2081) Mar 8, 2025

1. Collaborative filtering vs Content-Based filtering
2. Deep Learning for Content base filtering
3. Recommending from a large Catalogue part-I
4. finding the Related Items:
5. K-means Intuition

Source: Unsupervised Learning, Recommenders, Reinforcement Learning
Coursera Course.

Finding the Related Items:

Finding related items

The features $x^{(i)}$ of item i are quite hard to interpret.

romance

action

To find other items related to it,

find item k with $x^{(k)}$ similar to $x^{(i)}$

x_1, x_2, x_3

n

i.e. smallest distance.

$$\sum_{l=1}^{n} \left(x_l^{(k)} - x_l^{(i)}\right)^2$$

$x^{(k)}$

$$\|x^{(k)} - x^{(i)}\|^2$$

find this to show more recommendations

$x^{(k)}$

$x^{(i)}$

collectively x_1, x_2, x_3,

Summary

Recommends Similar items by learning vectors $x^{(i)}$ for each item based on user interactions.

$$\sum_{i=1}^{n} \left(x_l^{(k)} - x_l^{(i)}\right)^2$$

Where $x^{(i)}$ is the feature vector of item i and $x^{(k)}$ are another item —

Smaller Distance → More Similar the items.

Finding related items

The features $x^{(i)}$ of item i are quite hard to interpret.

romance
action

x_1, x_2, x_3
n

To find other items related to it,
find item k with $x^{(k)}$ similar to $x^{(i)}$

i.e. with smallest distance

$$\sum_{l=1}^{n} \left(x_l^{(k)} - x_l^{(i)}\right)^2$$

$x^{(k)}$ $x^{(i)}$

$$\|x^{(k)} - x^{(i)}\|^2$$

Limitations of Collaborative Filtering

Cold start problem. How to

(Challenges of Recommender)

- rank new items that few users have rated?
- show something reasonable to new users who have rated few items?

Use side information about items or users:

(web browsers, IP Address.)

- Item: Genre, movie stars, studio, ….
- User: Demographics (age, gender, location), expressed preferences, …

a few limitations of collaborative filtering.

Content-Based filtering : Collaborative filtering vs Content-Based filtering

Content-Based filtering

→ users features

[ ] → vector.

Collaborative filtering vs Content-based filtering

→ **Collaborative filtering:**
Recommend items to you based on ratings of users who gave similar ratings as you

→ **Content-based filtering:**
Recommend items to you based on features of user and item to find good match

$r(i,j)=1$ if user j has rated item i

$y^{(i,j)}$ rating given by user j on item i (if defined)

potentially a pure collaborative

DeepLearning.AI Stanford ONLINE Andrew Ng

Examples of user and item features

User features:

- Age
- Gender (1 hot)
- Country (1 hot, 200)
- Movies watched (1000)
- Average rating per genre
- ...

$x_u^{(j)}$ for user j

Vector size could be different.

Movie features:

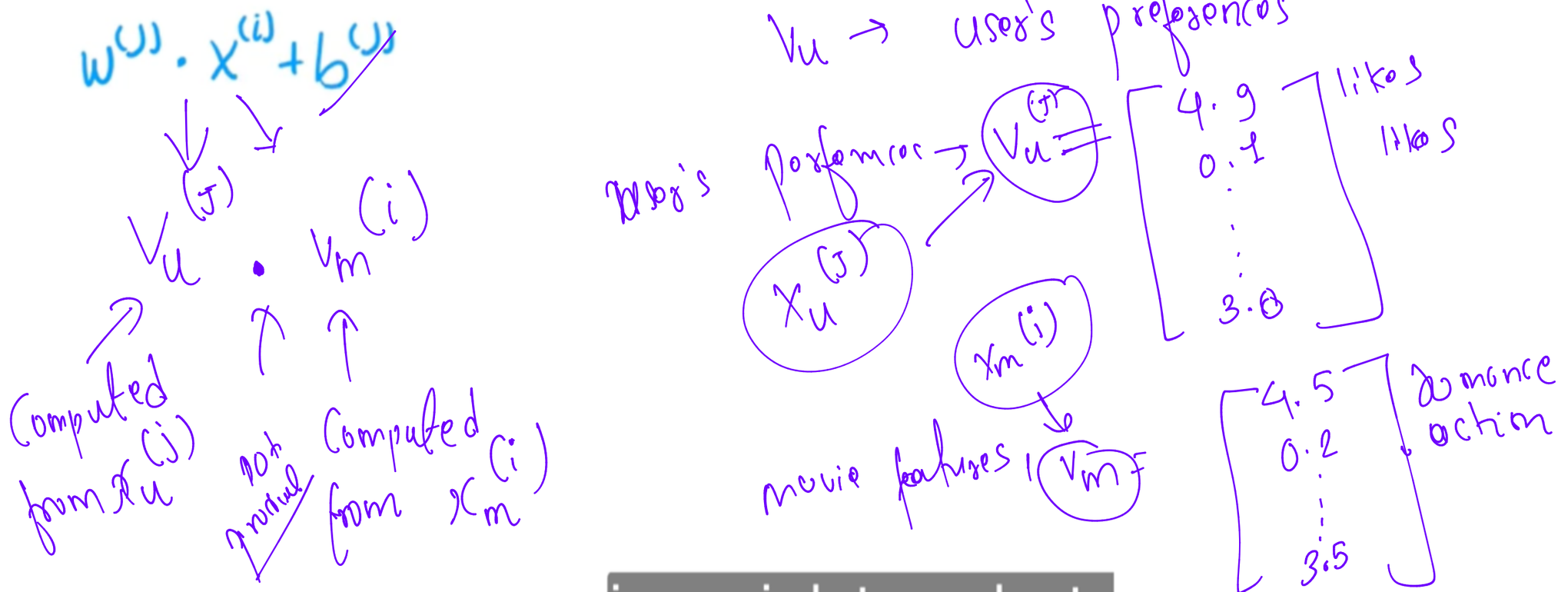
- Year
- Genre/Genres
- Reviews
- Average rating
- ...

$x_m^{(i)}$ for movie i

It turns out that if you have a set of movies

Content-based filtering: Learning to match

Predict rating of user *j* on movie *i* as



i as wj dot products of xi plus bj.

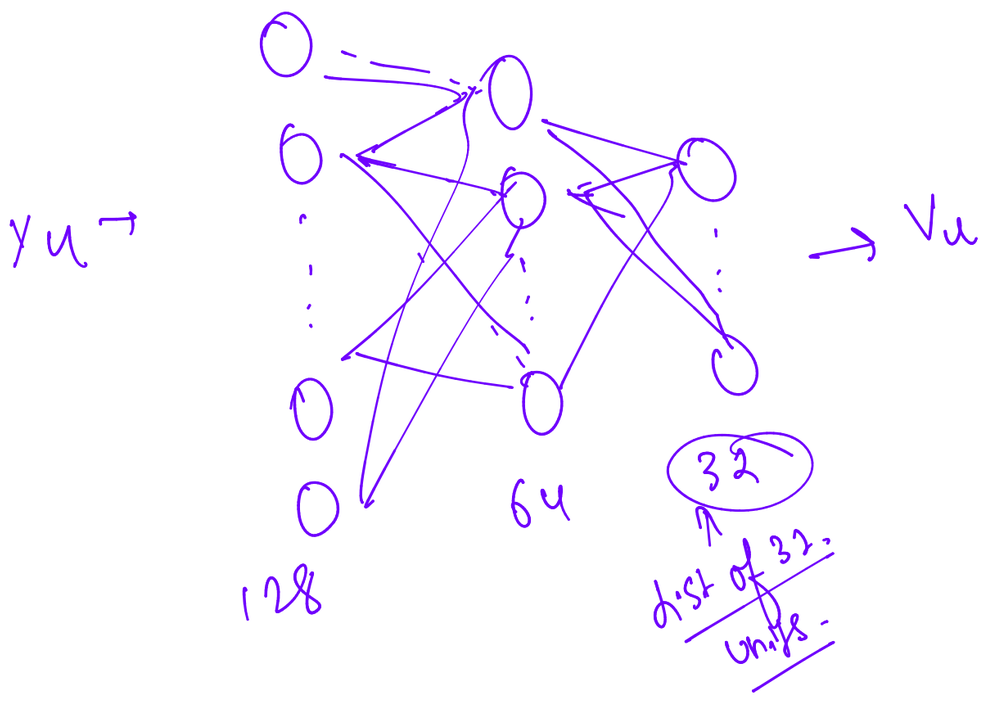
Deep Learning for Content-Based filtering:

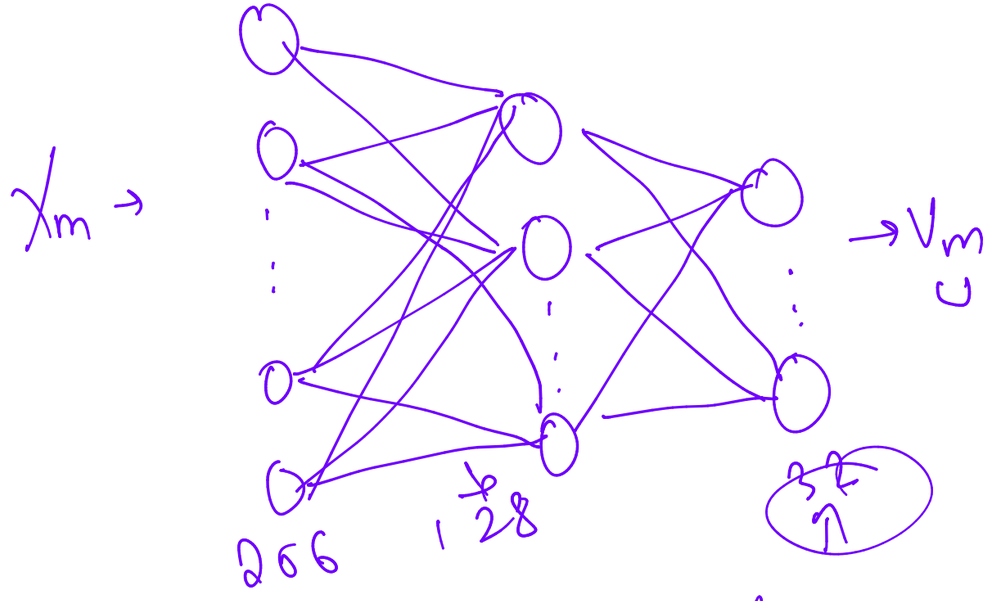
Neural network architecture

$x_u \rightarrow v_u$ uses network

$x_m \rightarrow v_m$

[Both $x_u \rightarrow v_u$ and $x_m \rightarrow v_m \rightarrow$ 32 units o/p]





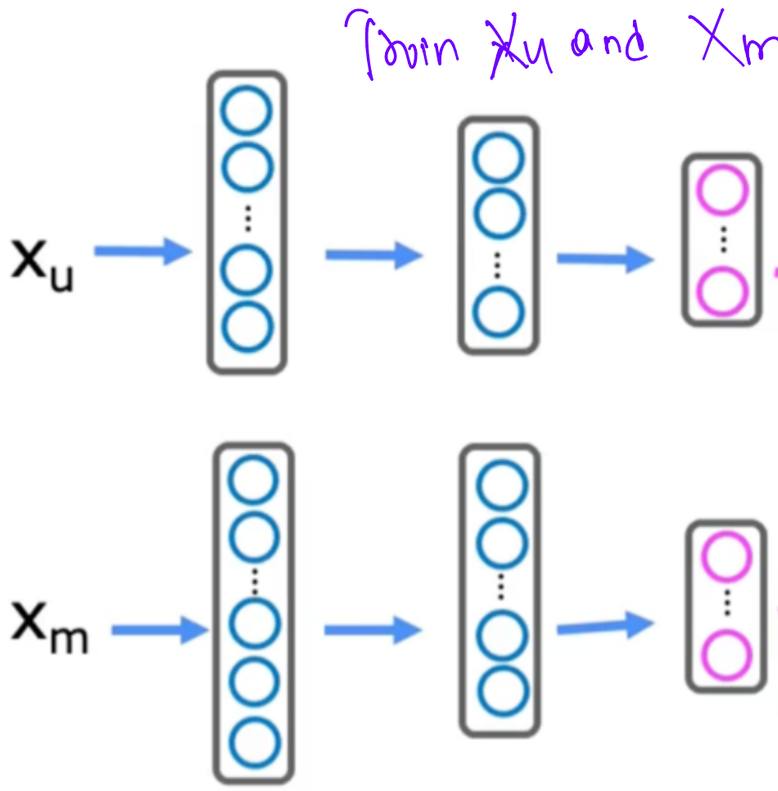
Prediction = $v_u \cdot v_m$

$g(v_u^{(j)} \cdot v_m^{(i)})$ to predict the β that $y^{(i,j)}$ is 2.

and so on, we have to compute

DeepLearning.AI Stanford ONLINE Andrew Ng

Neural network architecture



Train x_u and x_m

→ Dot product

↷ Different vectors.

↓ ↓

$(v_u^{(j)} \cdot v_m^{(i)} - y^{(i,j)})^2$ → Squared Error.

Cost function $J = \sum_{(i,j):r(i,j)=1}$ + NN Regularization term

train on the parameters

On the upper portion of this diagram,

DeepLearning.AI Stanford ONLINE Andrew Ng

Learned user and item vectors:

$v_u^{(j)}$ is a vector of length 32 that describes user j with features $x_u^{(j)}$

$v_m^{(i)}$ is a vector of length 32 that describes movie i with features $x_m^{(i)}$

To find movies similar to movie i: $\|v_m^{(k)} - v_m^{(i)}\|^2$ small

$\|x^{(k)} - x^{(i)}\|^2$

Note:- (Can be pre-computed ahead of time)

V_u^j is a vector of length 32 that describes

Deep Learning for Content-Based Filtering: Equations & Explanation

Deep learning enhances **content-based filtering** by learning user and item representations through **neural networks**.

1. User & Movie Representation with Neural Networks

We define two neural networks:

1. **User Network**: Maps user features x_u to a latent vector v_u.
2. **Movie Network**: Maps movie features x_m to a latent vector v_m.

Each network consists of multiple layers, with the final layer producing a **fixed-dimensional** embedding vector (e.g., 32-dimensional).

For a given user j, the user representation is computed as:

$$v_u^j = f_u(x_u^j, W_u)$$

Similarly, for a given movie i, the movie representation is computed as:

$$v_m^i = f_m(x_m^i, W_m)$$

where:

- f_u and f_m are neural network functions.
- x_u^j and x_m^i are the feature vectors.
- W_u and W_m are the weights of the respective networks.

2. Predicting Ratings Using Dot Product

The predicted rating for user j and movie i is computed using the **dot product** of their vectors:

$$\hat{y}^{ij} = v_u^j \cdot v_m^i = \sum_{k=1}^{d} v_u^j[k] \cdot v_m^i[k]$$

where:

- d is the embedding dimension (e.g., 32).
- $v_u^j[k]$ and $v_m^i[k]$ are the kth elements of the vectors.

For binary classification (e.g., predicting **likes/dislikes** instead of ratings), apply a **sigmoid activatio**

$$\hat{y}^{ij} = \sigma(v_u^j \cdot v_m^i) = \frac{1}{1 + e^{-(v_u^j \cdot v_m^i)}}$$

3. Cost Function for Training

The neural network parameters (W_u, W_m) are trained by minimizing a **Mean Squared Error (MSE)** loss:

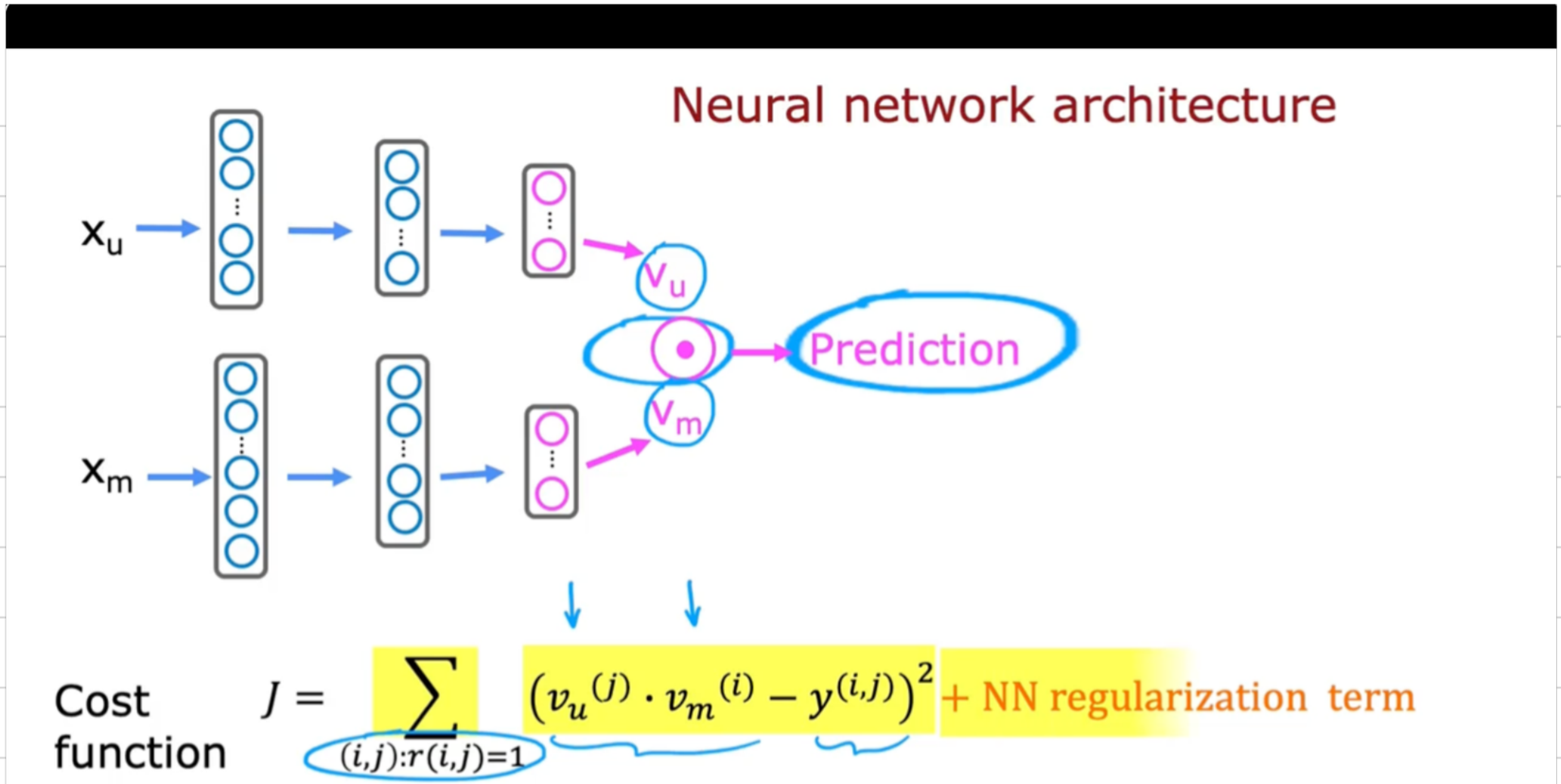
$$J = \sum_{(i,j)\in\mathcal{D}} (v_u^j \cdot v_m^i - y^{ij})^2$$

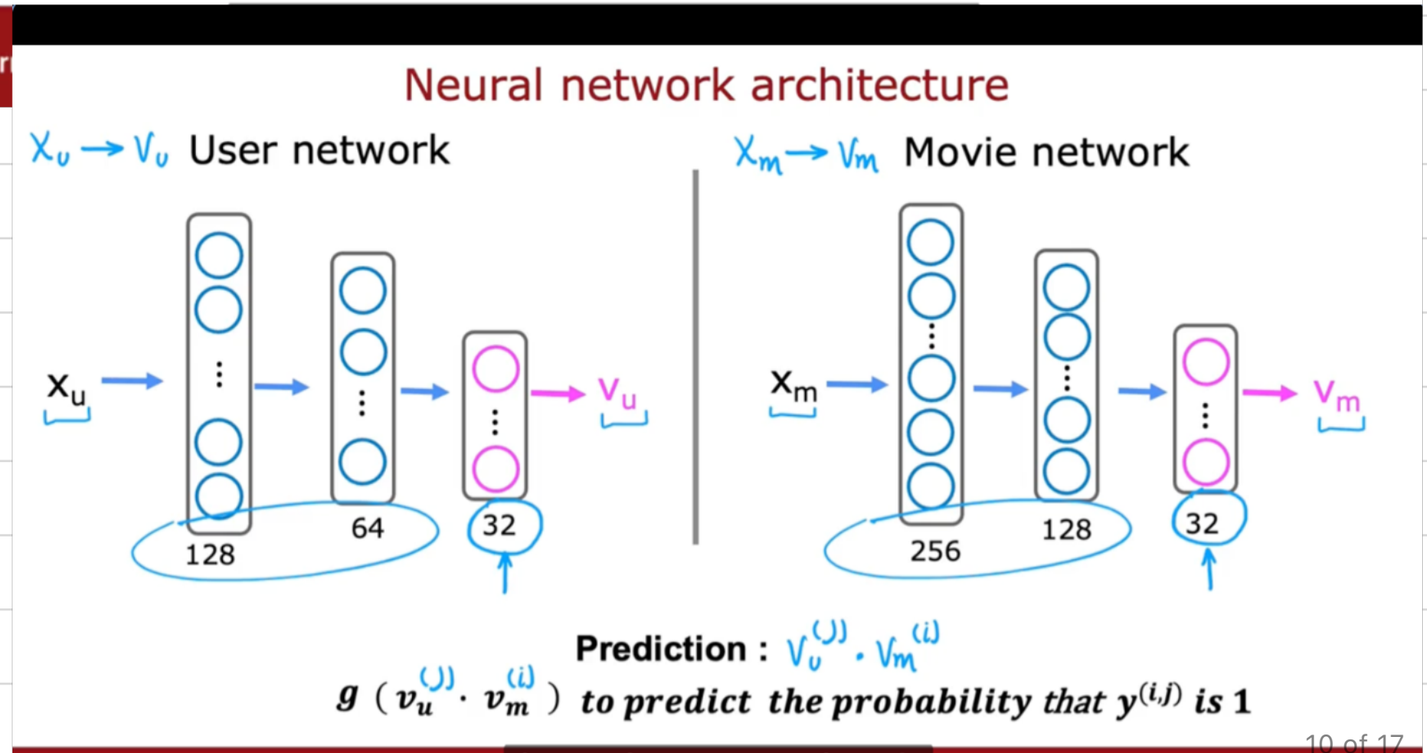
where:

- $\mathcal{D}$ is the set of observed user-movie interactions.
- y^{ij} is the actual rating given by user j to movie i.

To prevent overfitting, **regularization** can be applied:

$$J = \sum (v_u^j \cdot v_m^i - y^{ij}) + \lambda (\|W_u\|^2 + \|W_m\|^2)$$





User network & Movie Network generates embeddings

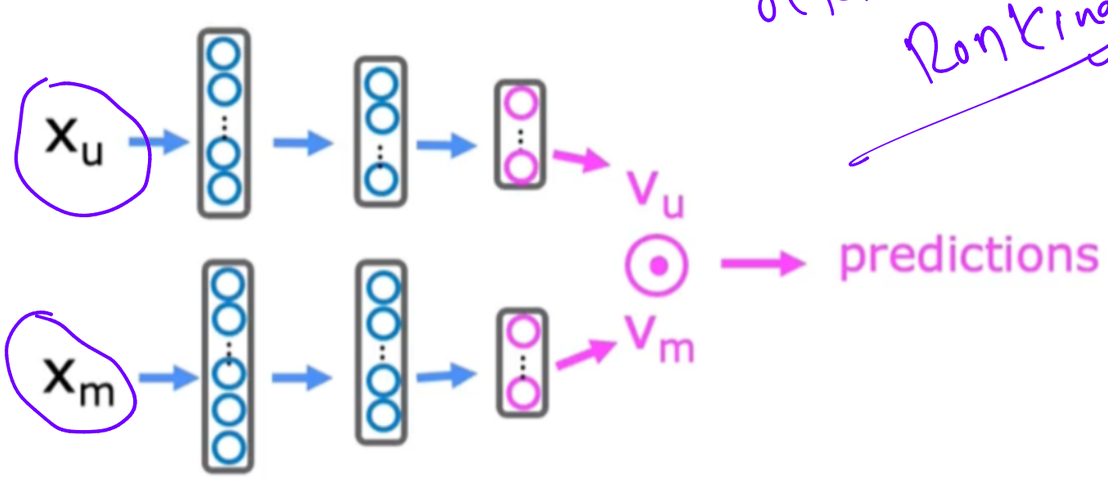
Recommending from a large Catalogue:

How to efficiently find recommendation from a large set of items?

Expensive! Infeasible to Compute

So we use retrieval & Ranking

- Movies 1000+
- Ads 1m+
- Songs 10m+
- Products 10m+



10m+ Videos, Products..

a system that is trying to decide what ad to show.

DeepLearning.AI Stanford ONLINE Andrew Ng

Two steps: Retrieval & Ranking

Retrieval:

- Generate large list of plausible item candidates

e.g.

1) For each of the last 10 movies watched by the user, find 10 most similar movies

2) For most viewed 3 genres, find the top 10 movies

3) Top 20 movies in the country

- Combine retrieved items into list, removing duplicates and items already watched/purchased

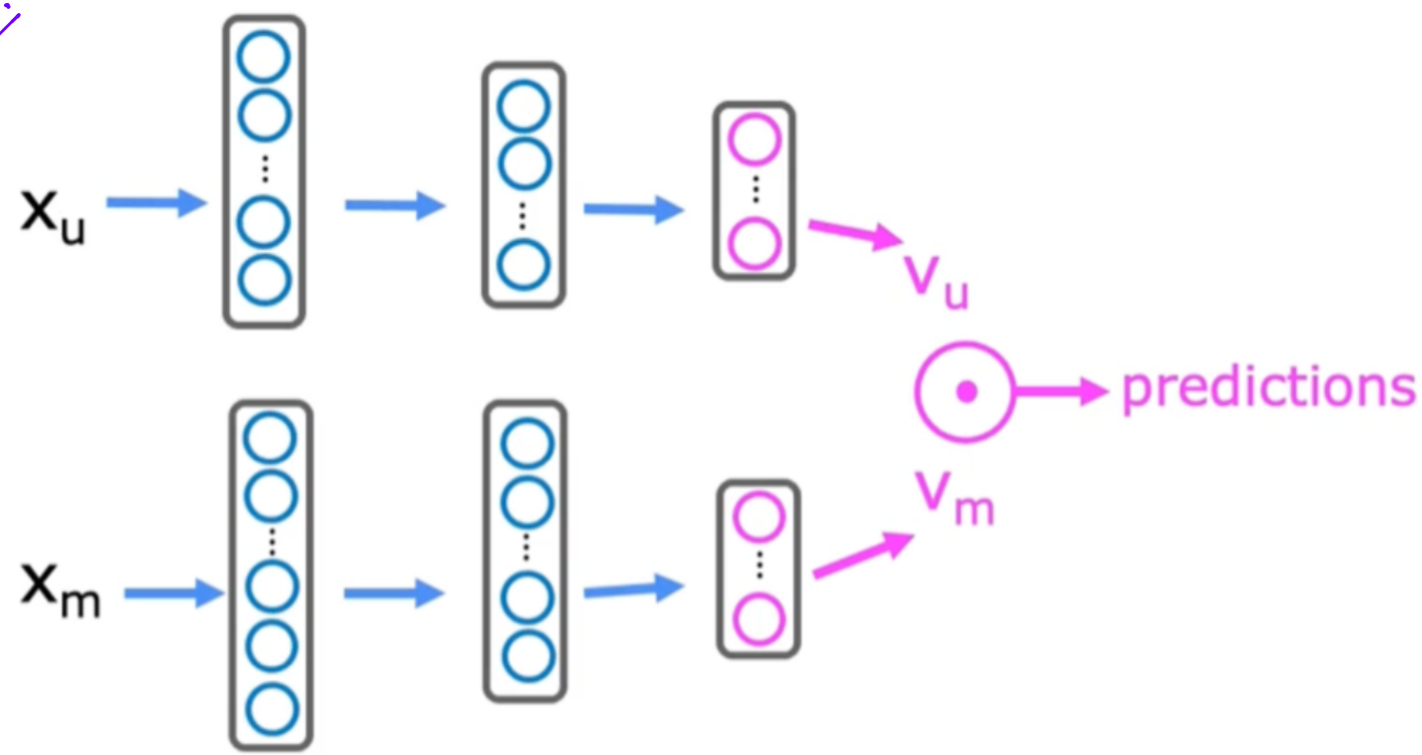
a large list of plausible item candidates.

DeepLearning.AI Stanford ONLINE Andrew Ng

Two steps: Retrieval & ranking

Ranking:

- Take list retrieved and rank using learned model



- Display ranked items to user

During the ranking step you will take the list retrieved during the retrieval step.

DeepLearning.AI Stanford ONLINE Andrew Ng

Retrieval step

- Retrieving more items results in better performance, but slower recommendations.
- To analyse/optimize the trade-off, carry out offline experiments to see if retrieving additional items results in more relevant recommendations (i.e., $p(y^{(i,j)}) = 1$ of items displayed to user are higher).

Separate Retrieval step | Ranking Step.

But the algorithm will end up being slower to analyze or

Efficient Recommender Systems: Retrieval & Ranking Approach

Recommender systems must efficiently select relevant items from massive catalogs (millions+). Direct inference for all items is computationally infeasible, so a **two-step approach** is used:

1. **Retrieval Step**: Quickly generate a **candidate set** (~100-500 items) from a large catalog.
2. **Ranking Step**: Use a neural network to score and rank these candidates.

---

1. Retrieval Step: Generating Candidate Items

The goal is to **narrow down the item pool** efficiently.

(a) Content-Based Similarity

Precompute **similar items** using the **dot product similarity**:

$$\text{Similarity}(v_m^i, v_m^k) = v_m^i \cdot v_m^k$$

where:

- v_m^i = embedding of watched movie i
- v_m^k = embedding of candidate movie k

Precompute and store similar items in a **lookup table** for fast retrieval.

(b) Genre-Based Retrieval

Retrieve top N movies from the **user's top genres**:

$$\mathcal{M}_{\text{genre}} = \bigcup_{\text{top } G} \text{Top-}N(G)$$

(b) Genre-Based Retrieval

Retrieve top N movies from the **user's top genres**:

$$\mathcal{M}_{\text{genre}} = \bigcup_{\text{top } G} \text{Top-}N(G)$$

(c) Location-Based Retrieval

Retrieve **top items in user's country**:

$$\mathcal{M}_{\text{country}} = \text{Top-}N(\text{Country})$$

After these steps, we obtain a candidate set $\mathcal{M}$ of size ~100-500.

---

2. Ranking Step: Scoring & Selecting Top Items

Now, we **score and rank** candidates using a trained model.

(a) Compute User & Item Representations

Generate user embedding v_u from user features x_u:

$$v_u = f_u(x_u, W_u)$$

where W_u is the user model weights.

(b) Compute Score Using Inner Product

The preference score $\hat{y}^{ij}$ between user u and item i is:

$$\hat{y}^{ij} = v_u^j \cdot v_m^i$$

For **classification (e.g., click prediction)**, apply a **sigmoid activation**:

$$\hat{y}^{ij} = \sigma(v_u^j \cdot v_m^i) \downarrow \frac{1}{1 + e^{-(v_u^j \cdot v_m^i)}}$$

Items are **ranked** based on $\hat{y}^{ij}$, and the top K items are recommended.

---

3. Optimization: Balancing Speed vs. Performance

- **More retrieved items → better recommendations** but **slower inference**.
- **Offline experiments**: Test different retrieval sizes (e.g., 100 vs. 500).
- If retrieving more items significantly **improves predicted scores**, increase the retrieval count.

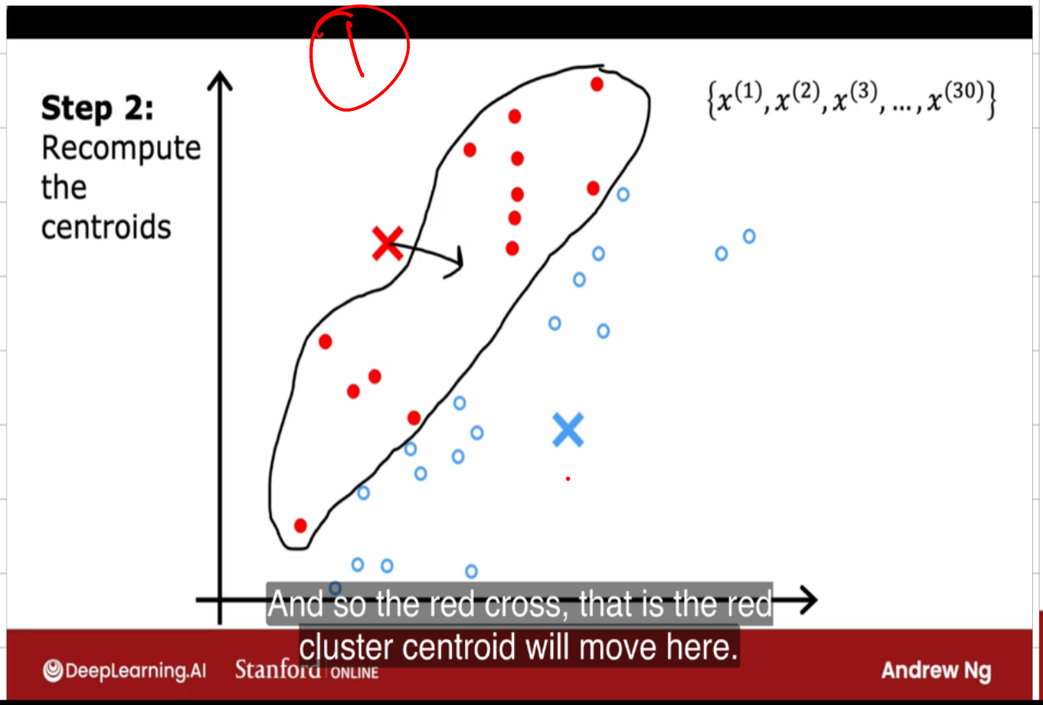
4. Ethical Considerations

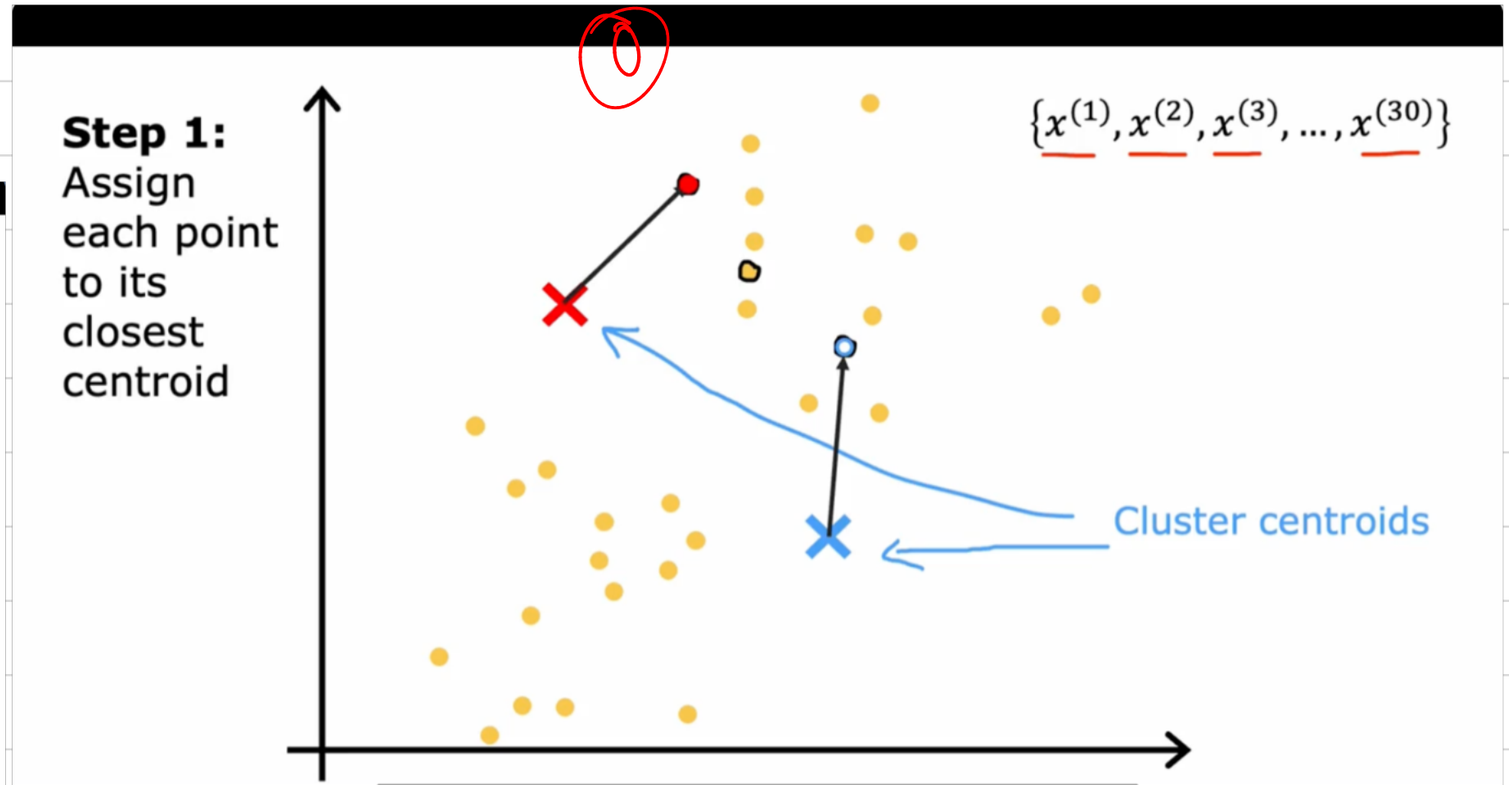
- **Bias & Filter Bubbles**: Diversify recommendations.
- **Privacy Risks**: Handle user data responsibly.
- **Fairness & Transparency**: Ensure unbiased recommendations.

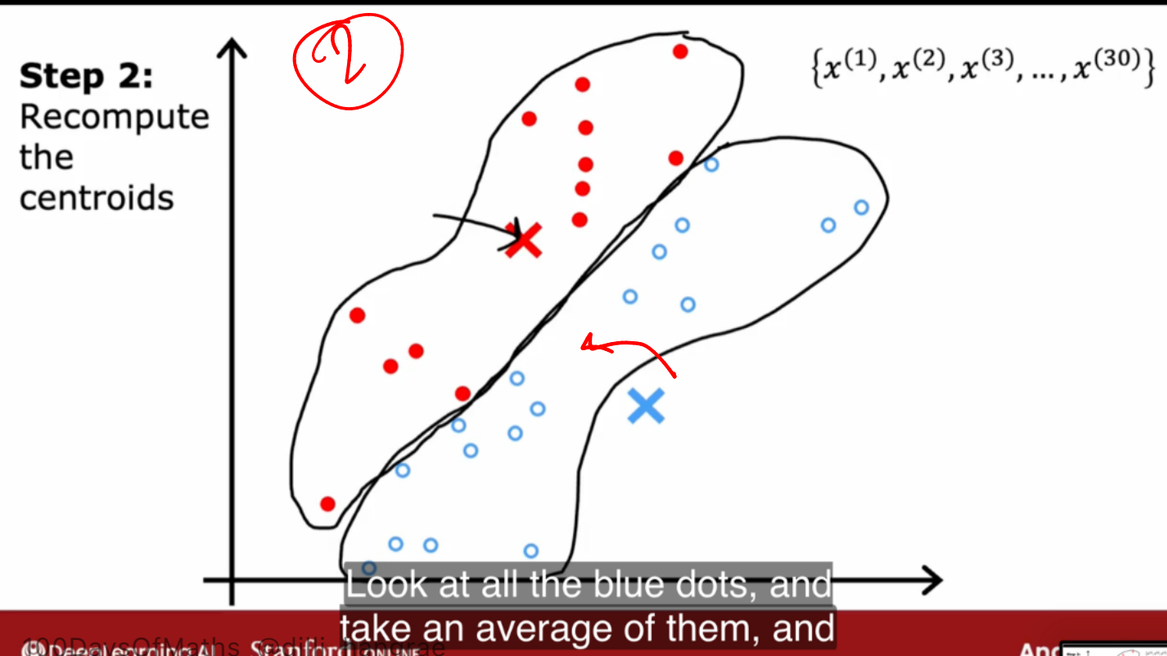
Conclusion

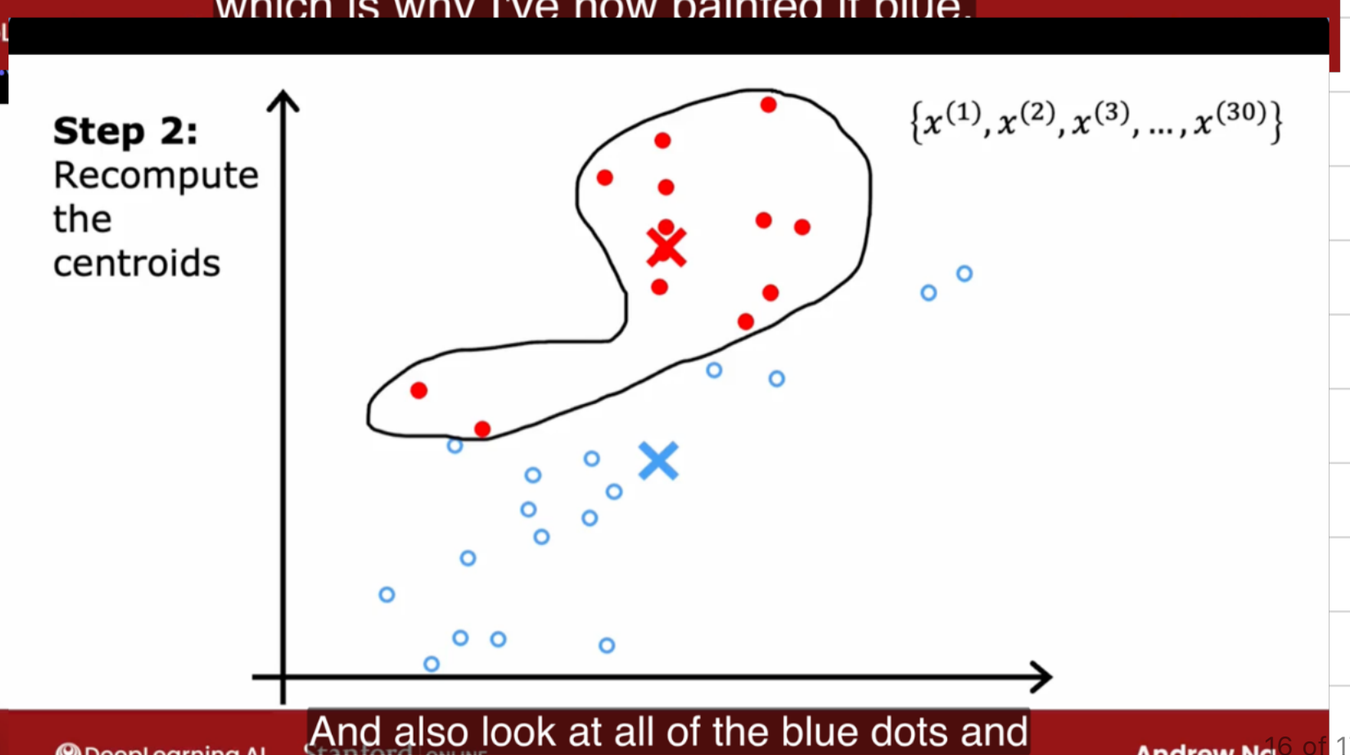
By **separating retrieval & ranking**, recommender systems efficiently handle **millions of items** while ensuring **high-quality recommendations**. 🚀

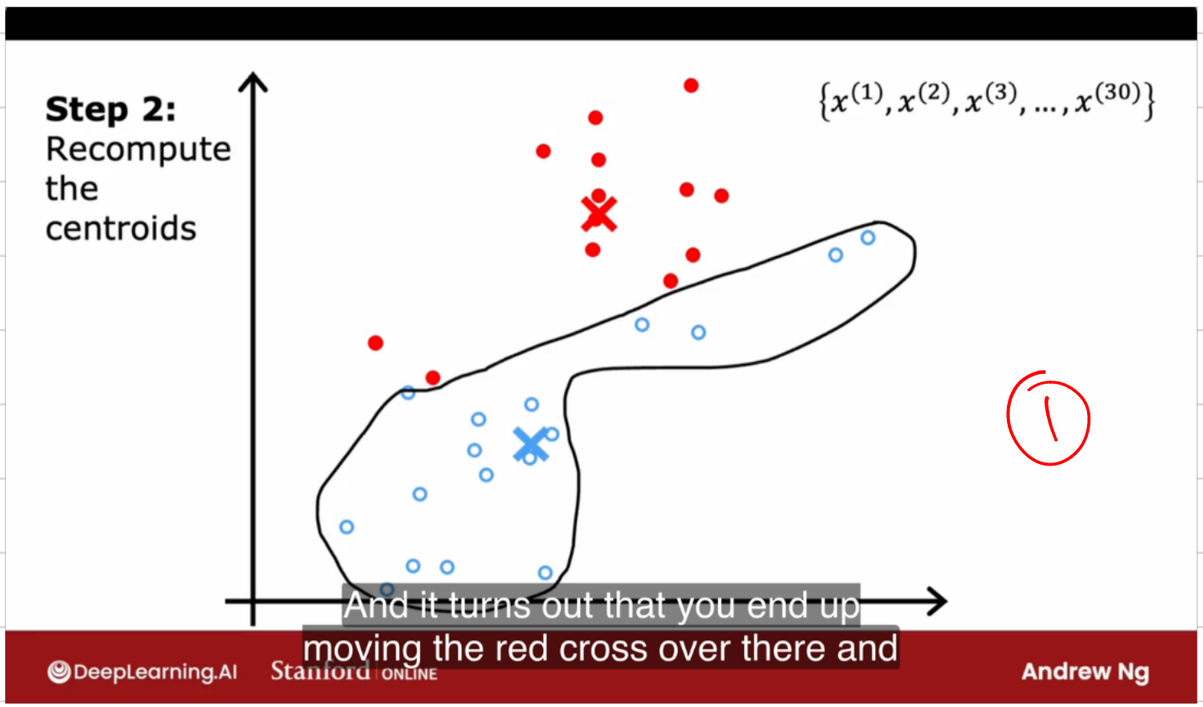
K-means Clustering

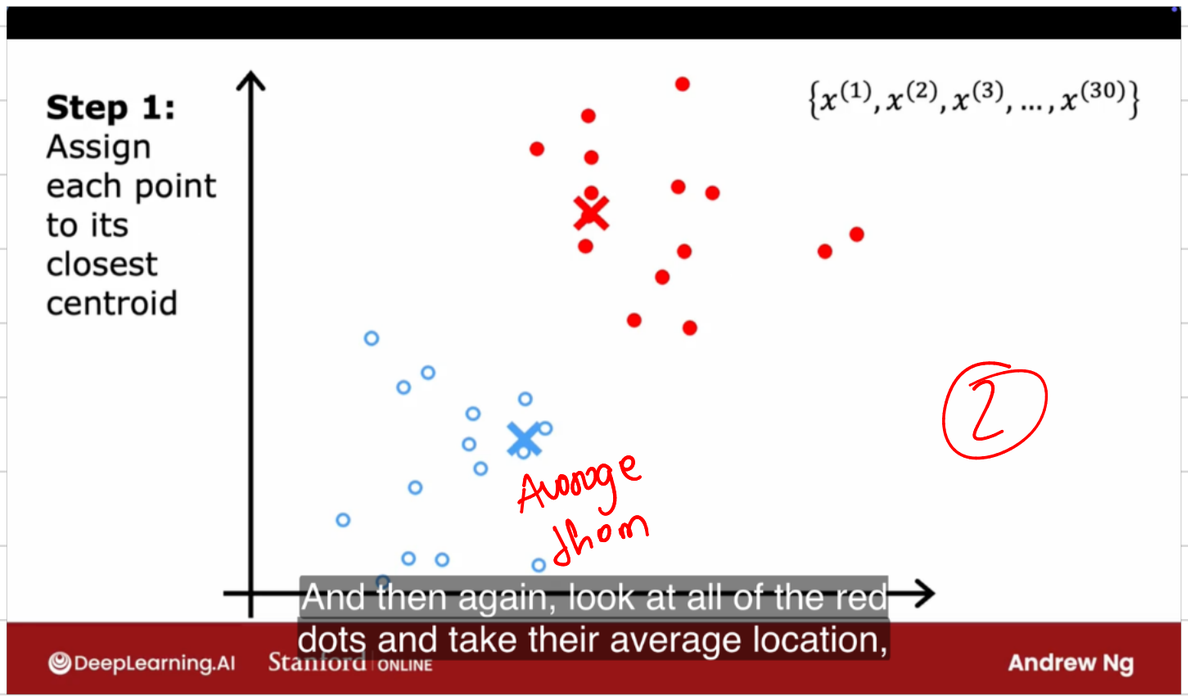


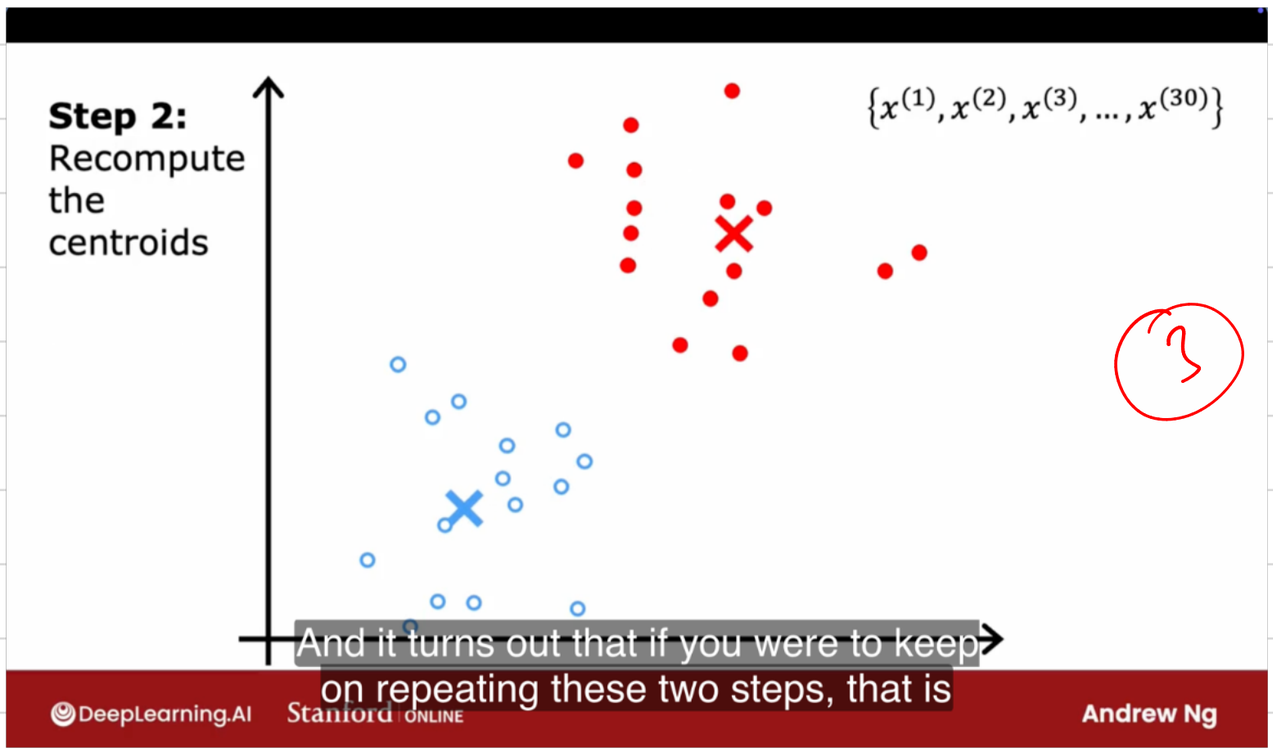


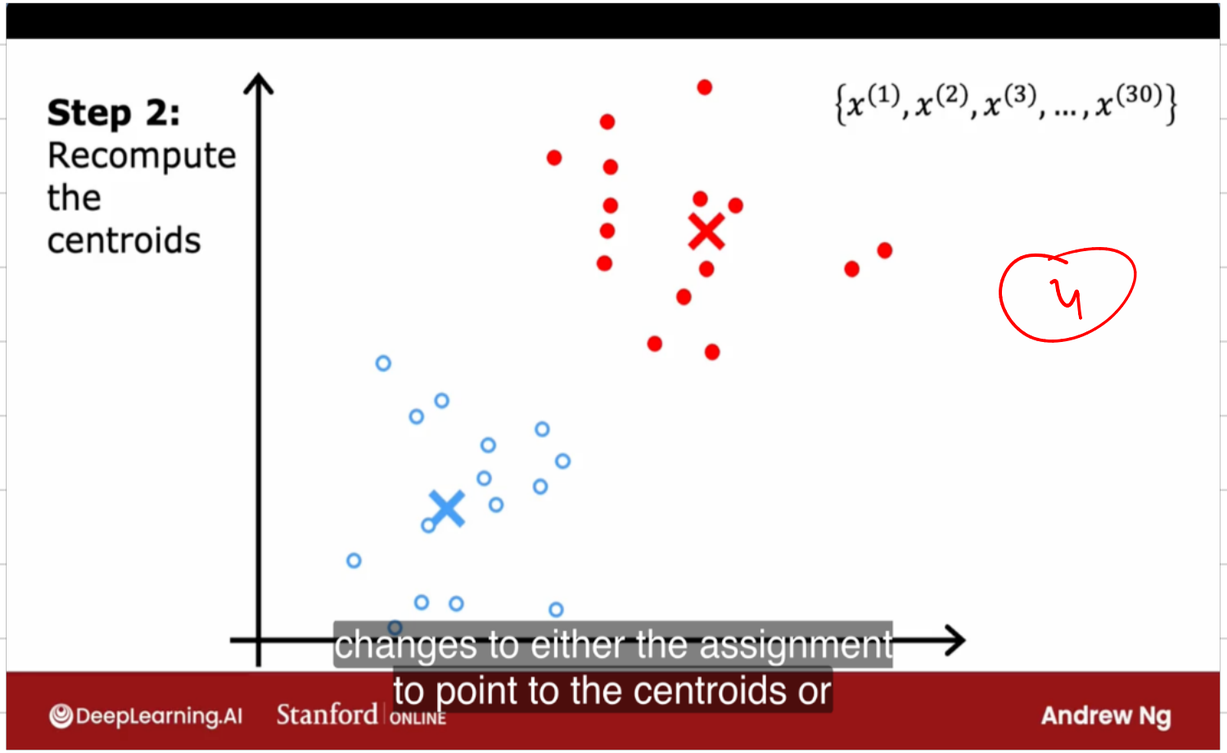





Step 2:
Recompute
the
centroids
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$
1
And it turns out that you end up
moving the red cross over there and
DeepLearning.AI
Stanford ONLINE
Andrew Ng


Step 1:
Assign
each point
to its
closest
centroid
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$
2
Average
them
And then again, look at all of the red
dots and take their average location,
DeepLearning.AI
Stanford ONLINE
Andrew Ng


Step 2:
Recompute
the
centroids
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$
3
And it turns out that if you were to keep
on repeating these two steps, that is
DeepLearning.AI
Stanford ONLINE
Andrew Ng


Step 2:
Recompute
the
centroids
$\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(30)}\}$
4
changes to either the assignment
to point to the centroids or
DeepLearning.AI
Stanford ONLINE
Andrew Ng