W4995 Applied Machine Learning
Fall 2021

Lecture 11

Dr. Vijay Pappu

Slides by Shoya Yoshida (SEAS '21)

Announcements

- HW4 posted and due on 12/20 11:59PM EST
- Project final deliverable also due on 12/15 11:59 EST
- Please fill in the final course evaluation
- Course offered again in Spring 2022

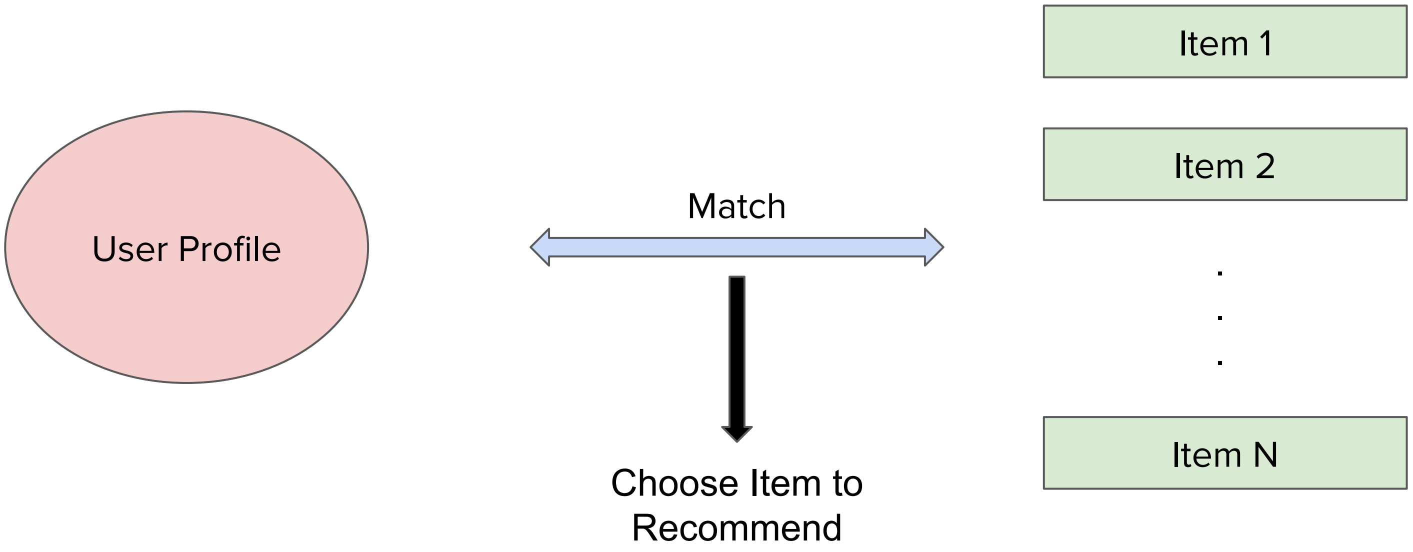
In today's lecture, we will cover...

- Recommender Systems
    - Motivation
    - Classical Approaches
        - Content-based Filtering
        - Collaborative Filtering
    - Evaluating Recommender Systems
    - Modern Recommender Systems using Deep Learning
    - Challenges

Motivation

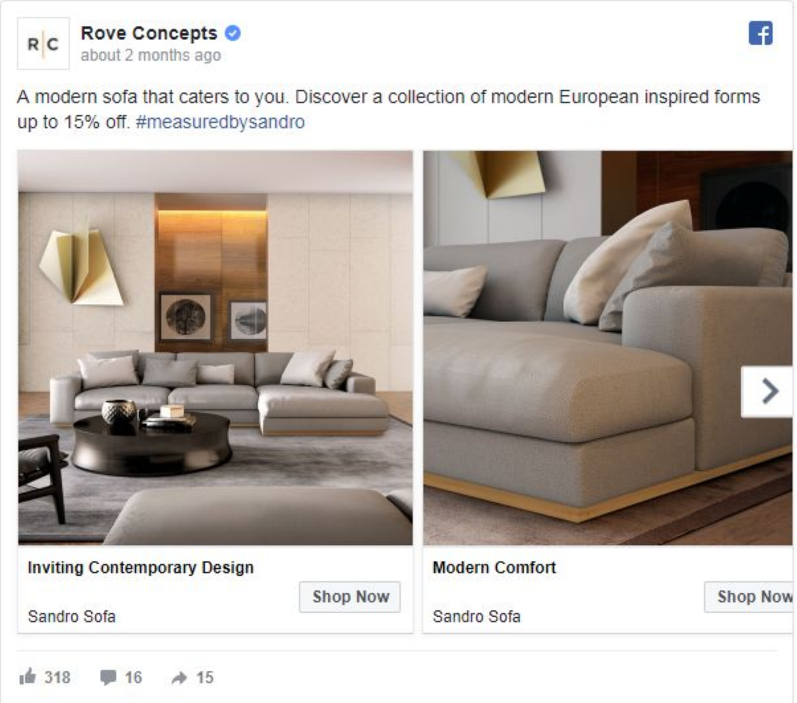
What is a Recommender System?

A recommender system aims to **recommend some item/product** that would **likely of be interest** to a user based on information about the item and/or the user



Recommender Systems are Ubiquitous

Ex. 1: Personalized Ads



Ex. 2: Product Recommendations

Ex. 3: Video Recommendations

Alice's Homescreen

Completely different!

Bob's Homescreen

- **70%** of watched content is from recommendations on YouTube (2018)
- **80%** of watched content is from recommendations on Netflix

Why do we even need recommendations?

- **Time and attention span of user is limited**
  - Most websites have too many items for the user to browse through all of them
    - Help users by **filtering down** the list of available items
  - Ex. Netflix: Need to quickly help a user find a video or the user may drift away to another platform

Approaches

Problem Setup

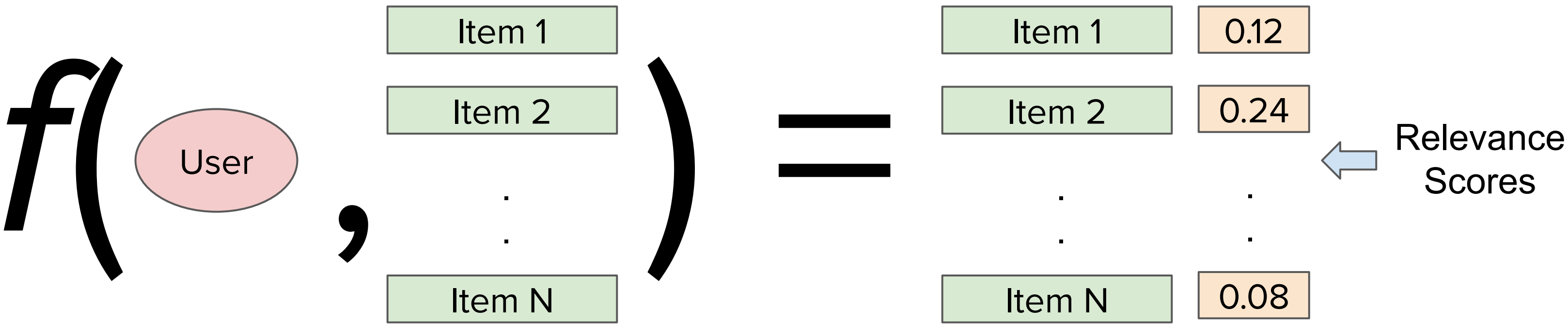
Given

- U = set of Users
- X = set of Items
- R = set of Ratings,

Use/Learn some function ***f*** such that ***f(U, X) → R***

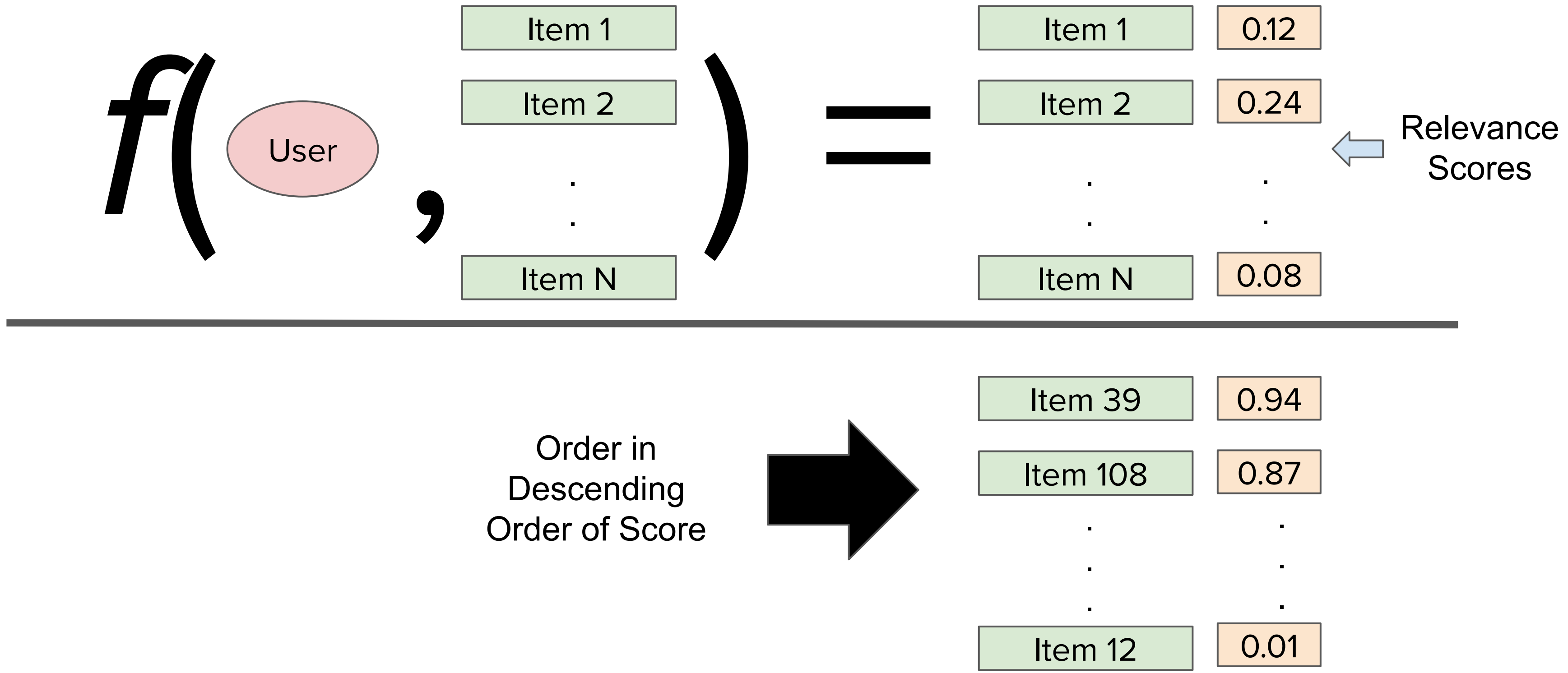
- Use this function ***f*** to **score** the relevance of each item for each user, then **recommend the items with highest scores** for each user

Typical Flow - Step 1: **Score** the set of items for each user

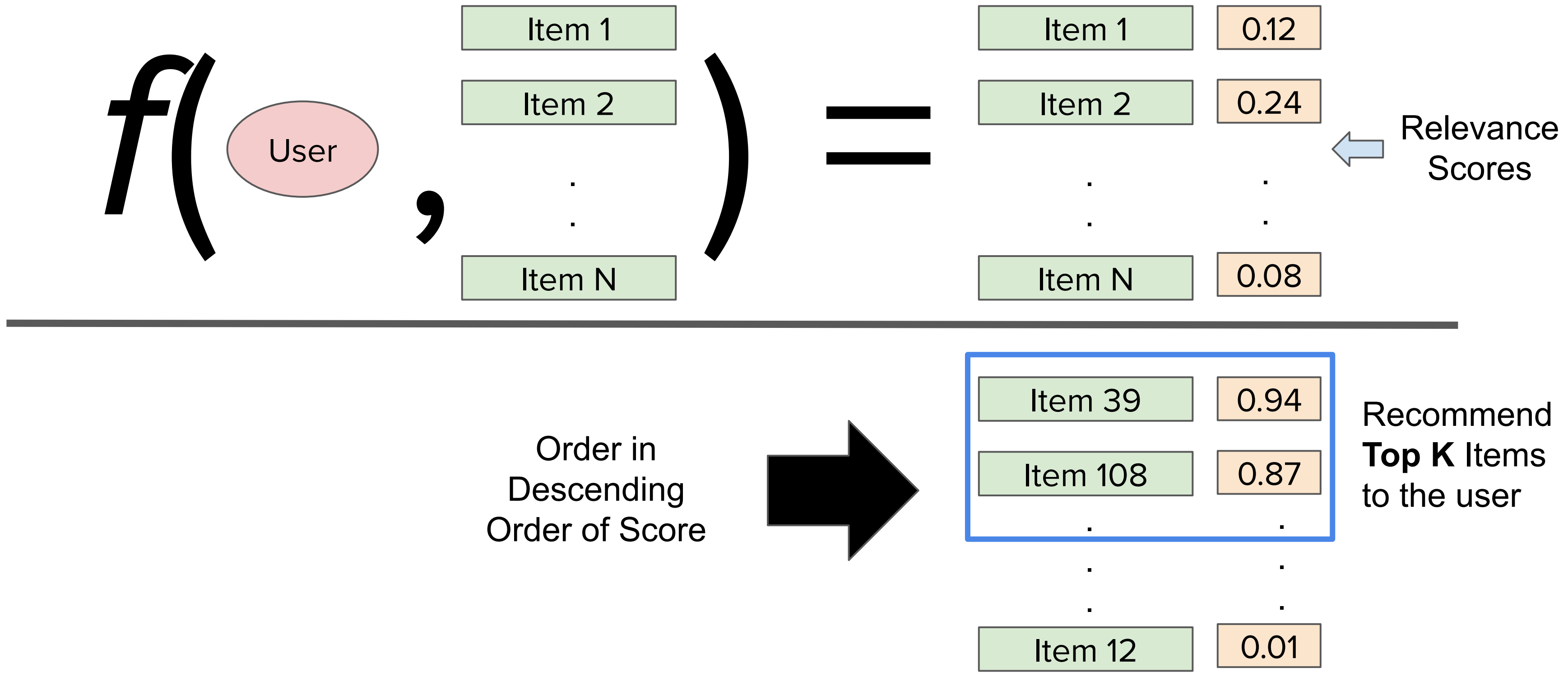


Note: In practice, we also **filter out certain items before scoring** them if the user has already dismissed them, interacted with them, or for other business reasons.

Typical Flow - Step 2: **Rank** the items using the scores



Typical Flow - Step 3: **Recommend top K** Items



Classical Approaches

- Content-based filtering
- Collaborative filtering

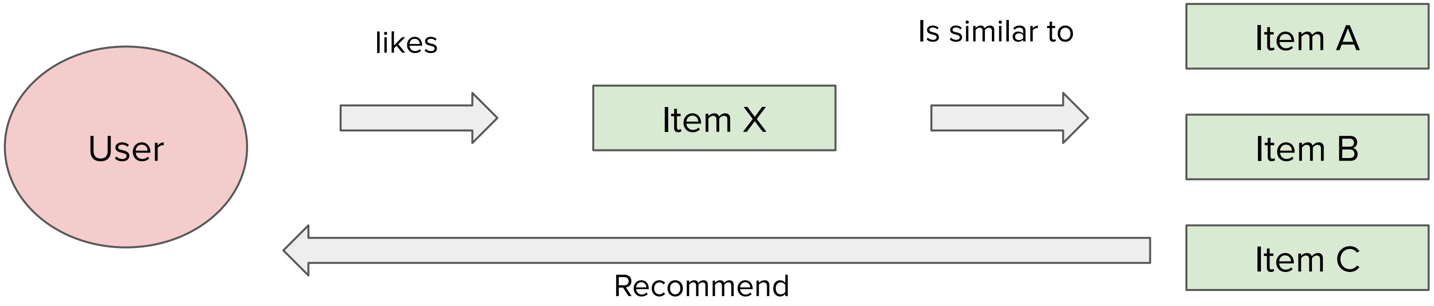
Content-Based Filtering

Content-Based Filtering

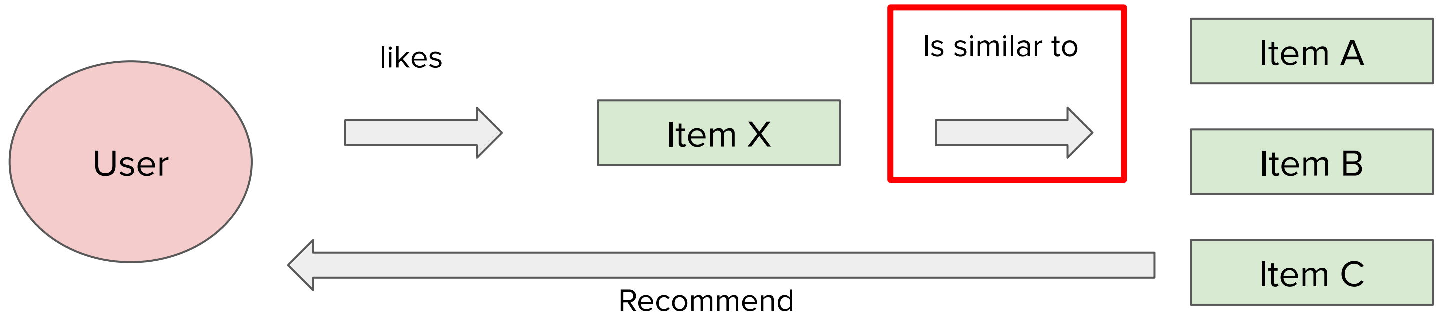
- Main Idea: Recommend items that are **similar to the ones that the user has already liked**
- Examples
    - User liked “Squid Game” on Netflix
        - Perhaps recommend to this user
            - Other K-Dramas
            - Other “Death Game” series
            - Other series starring similar actors
            - Other series made by the same director
    - Amazon: “Similar item to consider”

Image from [here]()

Content-Based Filtering Flow



How should we calculate similarities between items?



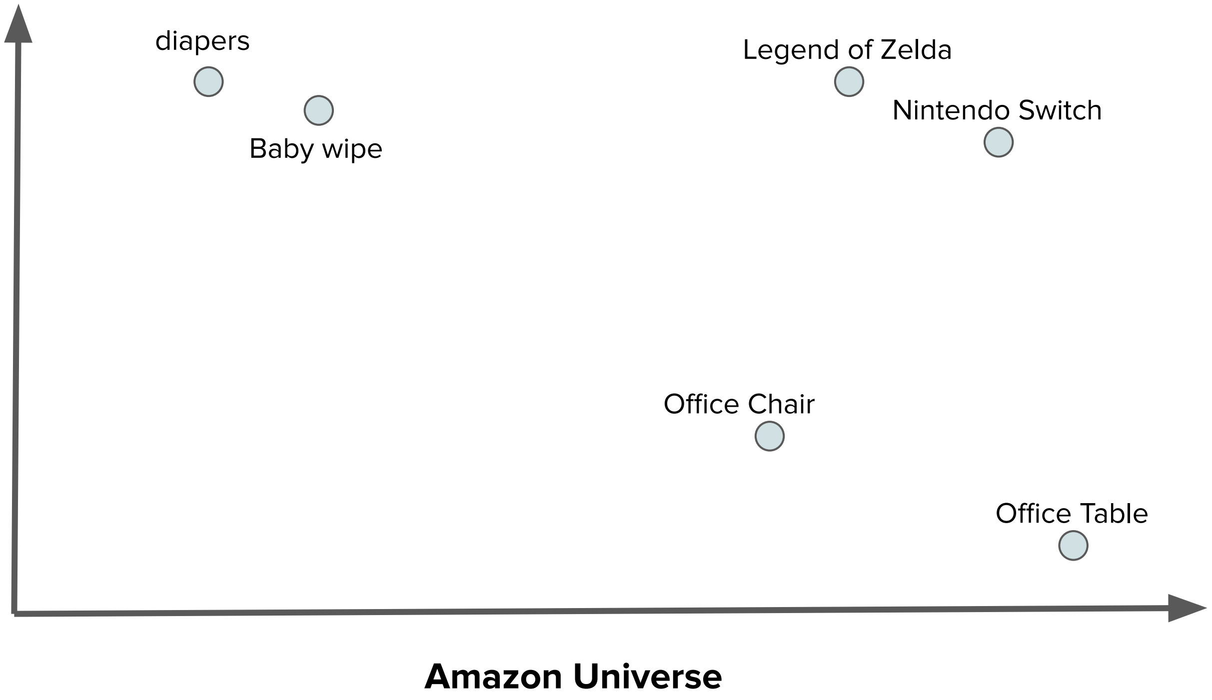
Let's first represent each item as *some* vector!

- One Approach: Build an **item profile** for each item by representing an item as a **set of its explicit features**
  - Netflix Series:
    - Genre, original language, actors, directors, studios, country of origin, budget, year released, etc.
  - Amazon Products:
    - Department, Item category, price, popularity, seller,

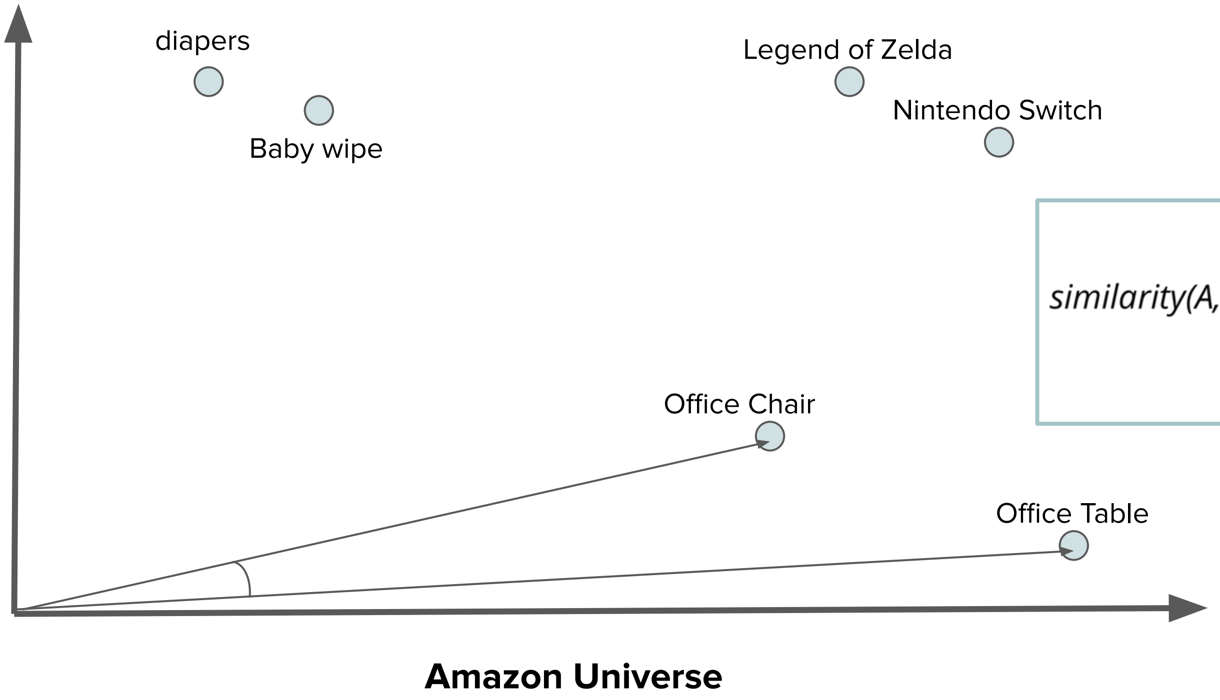
Example:

Item	Is_in_home_department	Price_between_100_150	is_prime	...
Office Chair X	1	1	1	...
Office Table Y	1	0	1	...

By doing this, we are essentially plotting each item as **a point in some vector-space**

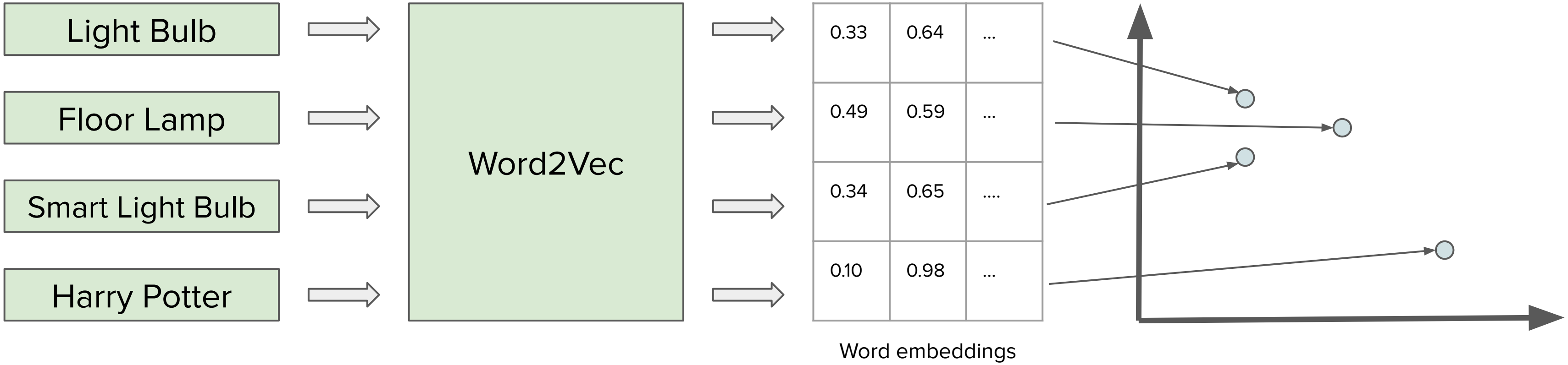


Similarity between items



$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

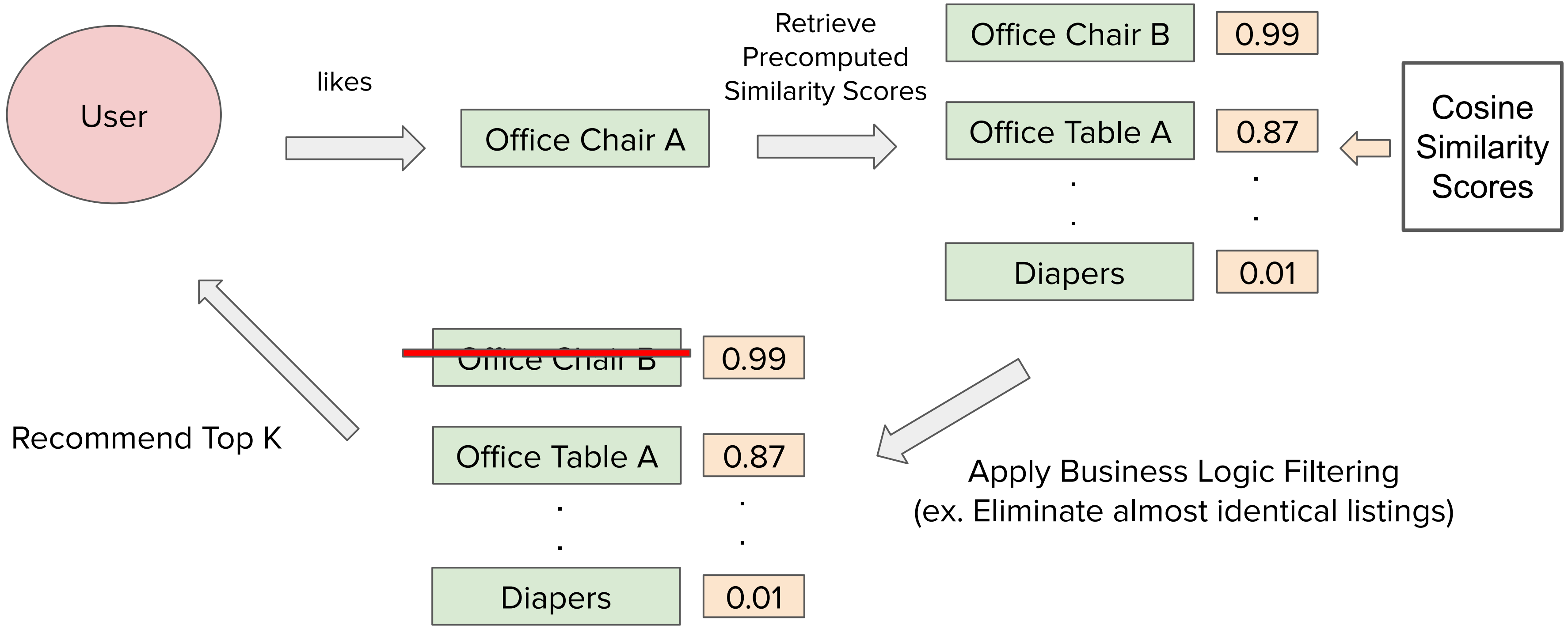
Side Note: In fact, in some cases when the item name is descriptive (i.e. Amazon), we could **use word embeddings to represent each item** as well!



Here, we set the semantic meaning of the item's name as the item profile

Takeaway: There are multiple ways to represent an item

Content-Based Filtering Full Example



Content-Based Filtering Pros and Cons

Pros

- \+ This framework needs **no data about what other users are doing**, so it is easy to scale
- \+ The framework can cover corner cases when a user has **very niche interests**
- **+** **No Cold-start Problem** (will be covered in later slide)
- \+ Model is **interpretable**

Cons

- Choosing features to represent an item is up to the engineer and will thus **require careful feature engineering and domain knowledge**
- **Predictive power can be limited**
  - Model will only make recommendations based on the user’s existing interests

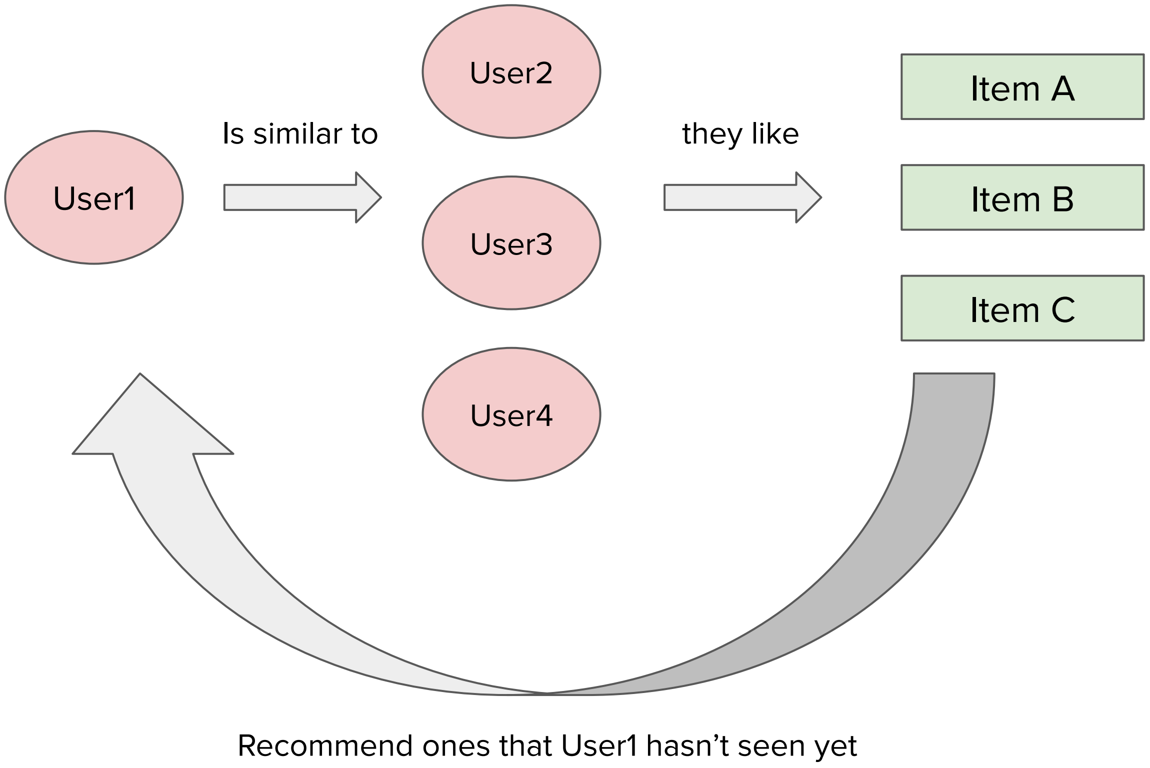
Collaborative Filtering

Collaborative Filtering Overview

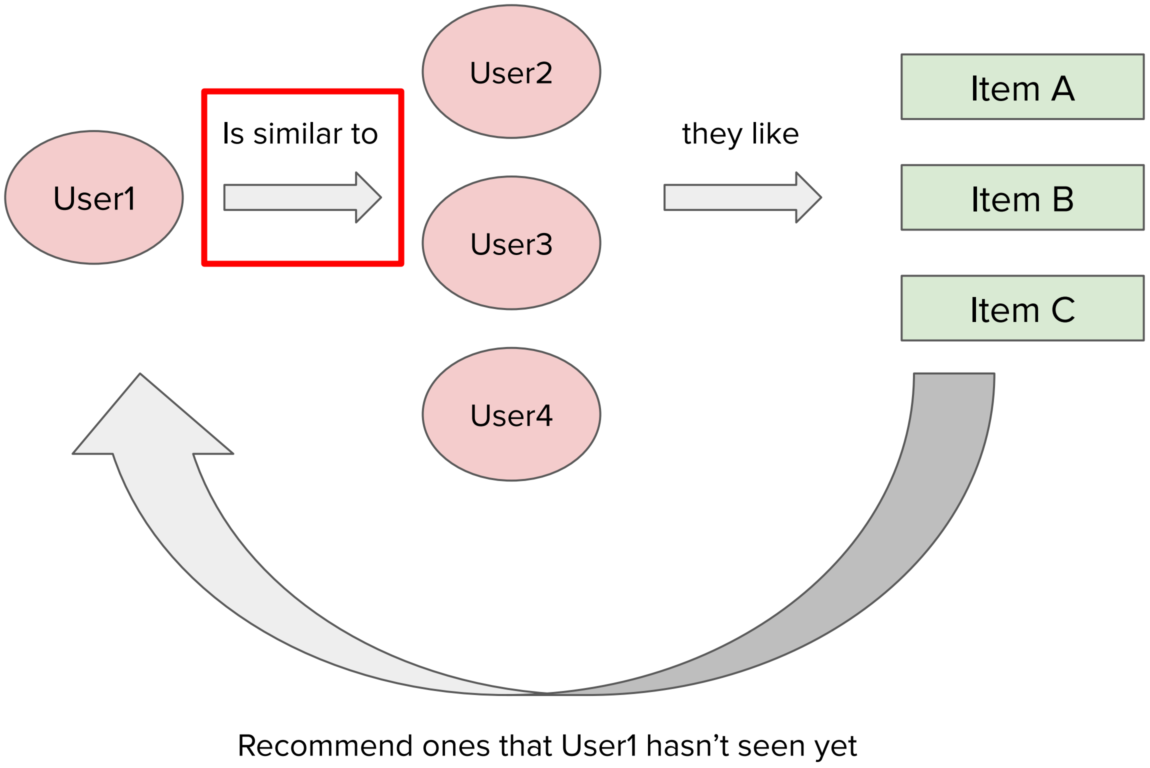
- Main Idea: Recommend items that **similar users to you** have already interacted with
    - Estimate user's ratings based on ratings of other similar users
- Examples
    - Netflix: "Other users who liked this show also watched…"
    - Amazon: "Other users who bought this product also bought…"

Image from here

Collaborative Filtering Flow



How to Retrieve Similar Users?



Similar to Content-Based Filtering Approach,
Let’s Represent a User with *some* vector too!

Let’s use a **rating vector** to represent each user!

Idea: You are defined by the things you like and dislike

Rating Matrix

- A **Rating Matrix** R is a M x N matrix, where M is the number of users, and N is the number of items
    - R[*i*, *j*] represents how the *i*-th user rated item *j*
    - A slot can be **left blank**, which is **normal in recommender systems** since each user can't interact with all items available on the platform

Example: A Netflix example with 3 users and 4 items (M=3, N=4).

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2		5
Bob		3		4
Chris	2		4	2
Derek	4	3		

Goal: **Estimate the missing values** to decide which item that user hasn't interacted with should be recommended

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3	?	?

Estimate what rating Bob *would* give to Squid Game and Breaking Bad, and recommend whichever one has a higher score

Idea: We want an algorithm that can **find users who are similar to you** and estimate your ratings on unseen items

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	~5	3	?	4
Chris	2	?	4	2
Derek	4	3		

In this example, Alice and Bob have similar opinions "The Crown" and "The Office," so they have relatively similar tastes

- Alice watched Squid Game and loved it, so Bob must like it too!

How should we estimate these values?

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3	?	?

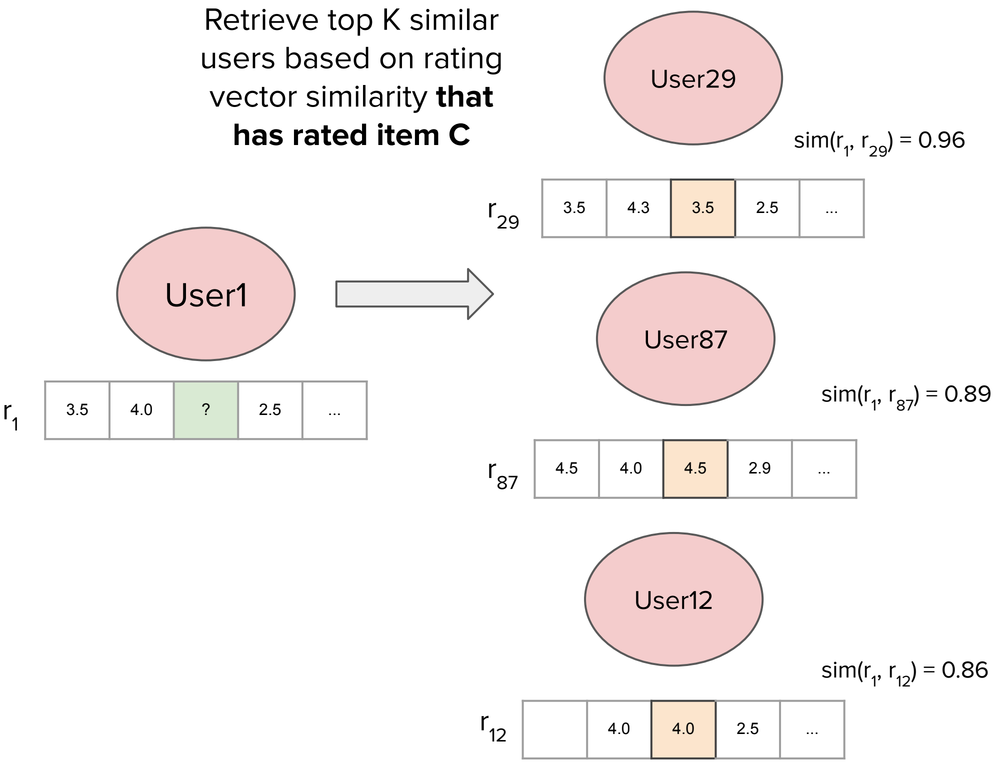
Naive Approach Walkthrough

rating(User1 , Item C) = ?

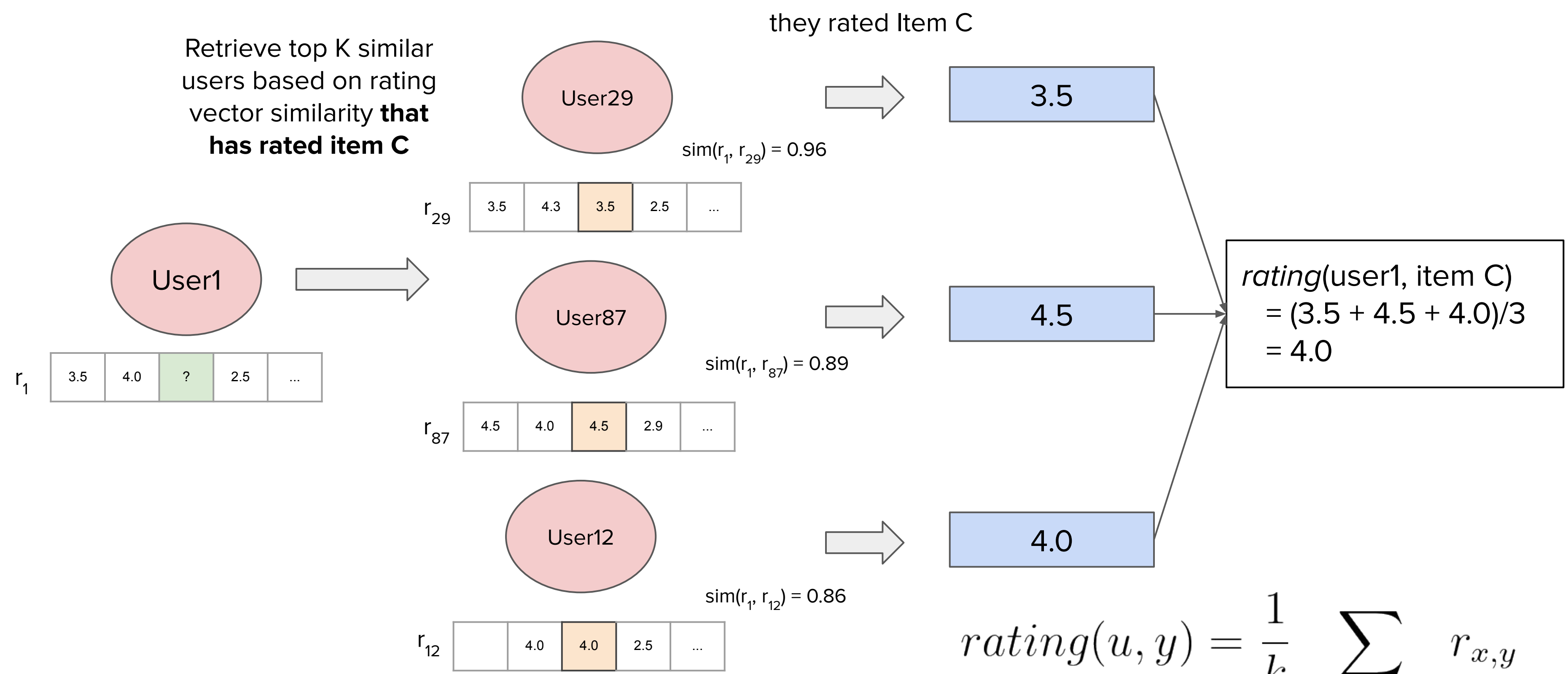
r_1

3.5	4.0	?	2.5	...

Represent each user with its rating vector, then calculate similarities to retrieve top K similar users



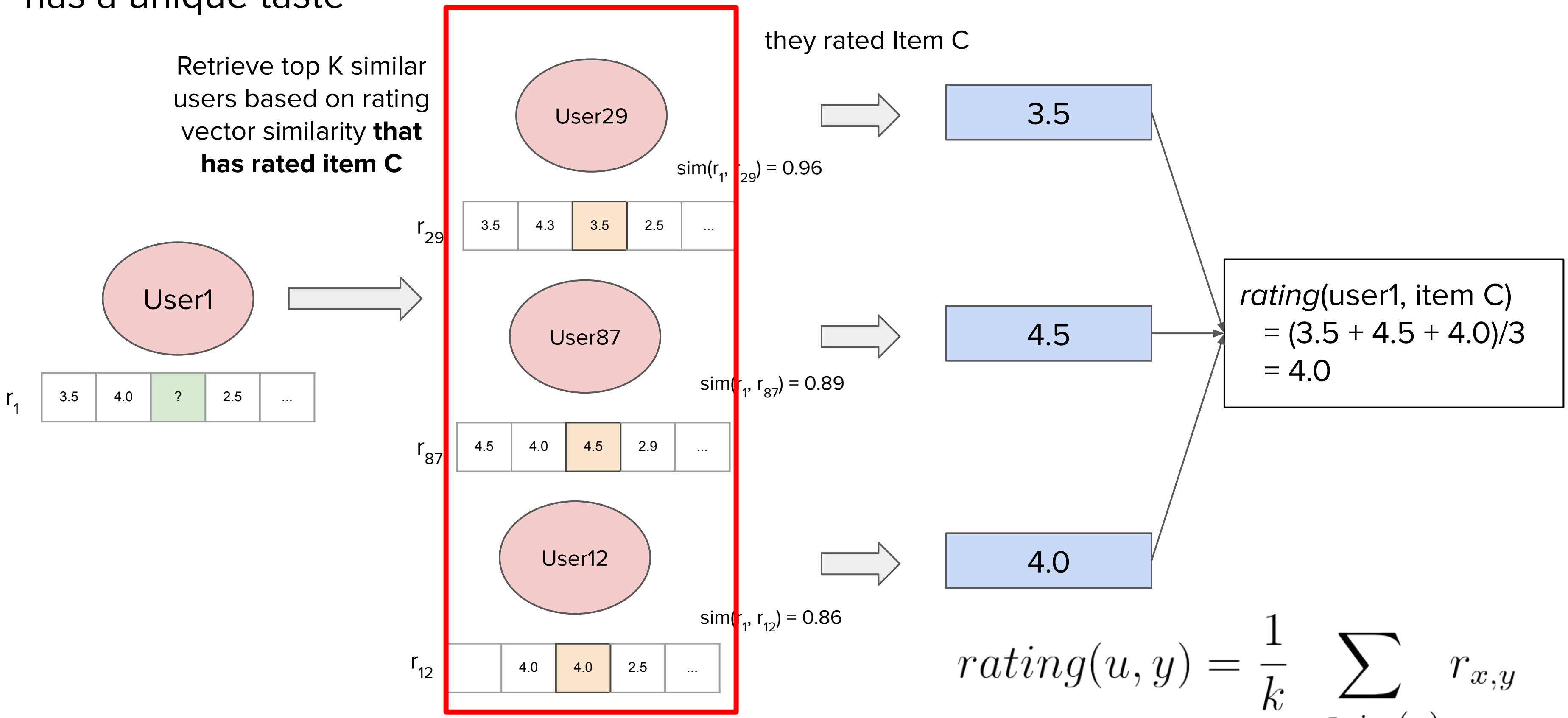
Then just average the similar users' ratings together



$$rating(u, y) = \frac{1}{k} \sum_{x \in sim(u)} r_{x,y}$$

Where *u* is a user and *y* is an item

Problem: It can be hard to find users that rated the same item, especially if a user has a unique taste



$$rating(u, y) = \frac{1}{k} \sum_{x \in sim(u)} r_{x,y}$$

Where *u* is a user and *y* is an item

More Involved Approach: Matrix Factorization: Decompose R into 2 Matrices

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	3.1	1.4
Bob	1.7	1.8
Chris	0.23	0.53
Derek	0.42	0.98

X

Item Matrix (V)

Squid Game	The Crown	Breaking Bad	The Office
1.5	0.19	0.97	0.12
0.8	0.63	0.82	0.25

$$\boxed{R \approx U \cdot V^T}$$

- Let's assume we can find 2 matrices U and V such that **when they are multiplied, it reconstructs the rating matrix R**
  - U is a matrix size of MxE, where M is the number of users and E is a **hyperparameter** for the embedding size (just like in word embeddings)
  - V is a matrix size of NxE, where N is the number of items

Estimate scores by dot-producting respective rows

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	$\boxed{?}$	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	3.1	1.4
Bob	1.7	1.8
Chris	0.23	0.53
Derek	0.42	0.98

X

Item Matrix (V)

Squid Game	The Crown	Breaking Bad	The Office
1.5	0.19	0.97	0.12
0.8	0.63	0.82	0.25

These are embeddings for these entries

$$\boxed{R_{\text{Bob,Squid Game}} = U_{\text{Bob}} \cdot V_{\text{SG}} = 1.7 * 1.5 + 1.8 * 0.8 = 4.0}$$

How do we learn the appropriate values for U and V?

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	?	?
Bob	?	?
Chris	?	?
Derek	?	?

X

Item Matrix (V)

Squid Game	The Crown	Breaking Bad	The Office
?	?	?	?
?	?	?	?

We know what the dot product SHOULD be if the user rated an item

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	$\boxed{5}$	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	$\boxed{X1}$	$\boxed{X2}$
Bob	?	?
Chris	?	?
Derek	?	?

X

Item Matrix (V)

Squid Game	The Crown	Breaking Bad	The Office
$\boxed{Y1}$	?	?	?
$\boxed{Y2}$	?	?	?

$$R_{Alice,Squid\ Game} = U_{Alice} \cdot V_{SG} = (X1 * Y1) + (X2 * Y2) = 5$$

So let's initialize the matrices U and V randomly, then iteratively improve it with some objective function

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	0.22	0.81
Bob	0.82	0.14
Chris	0.23	0.53
Derek	0.42	0.98

X

Item Matrix (V)

Squid Game	The Crown	Breaking Bad	The Office
0.42	0.19	0.97	0.12
0.31	0.63	0.82	0.25

If there is an error between the dot product and the actual rating, let’s **tweak the values** in U and V to reduce that error!

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	0.22	0.81
Bob	0.82	0.14
Chris	0.23	0.53
Derek	0.42	0.98

X

Item Matrix (V)

Squid Game	The Crown	Breaking Bad	The Office
0.42	0.19	0.97	0.12
0.31	0.63	0.82	0.25

$$R_{\text{Alice,Squid Game}} = U_{\text{Alice}} \cdot V_{SG} = (0.22 * 0.42 + 0.81 * 0.31) = 0.34$$

$\text{Error} = (5 - 0.34)^2 = 21.7$ ⇨ Let’s change U_{Alice} and V_{SG} to reduce this error

Example objective function to learn values for U and V

Rating Matrix (R)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2	?	5
Bob	?	3	?	4
Chris	2	?	4	2
Derek	4	3		

≈

User Matrix (U)

Alice	0.22	0.81
Bob	0.82	0.14
Chris	0.23	0.53
Derek	0.42	0.98

X

Item Matrix (V)

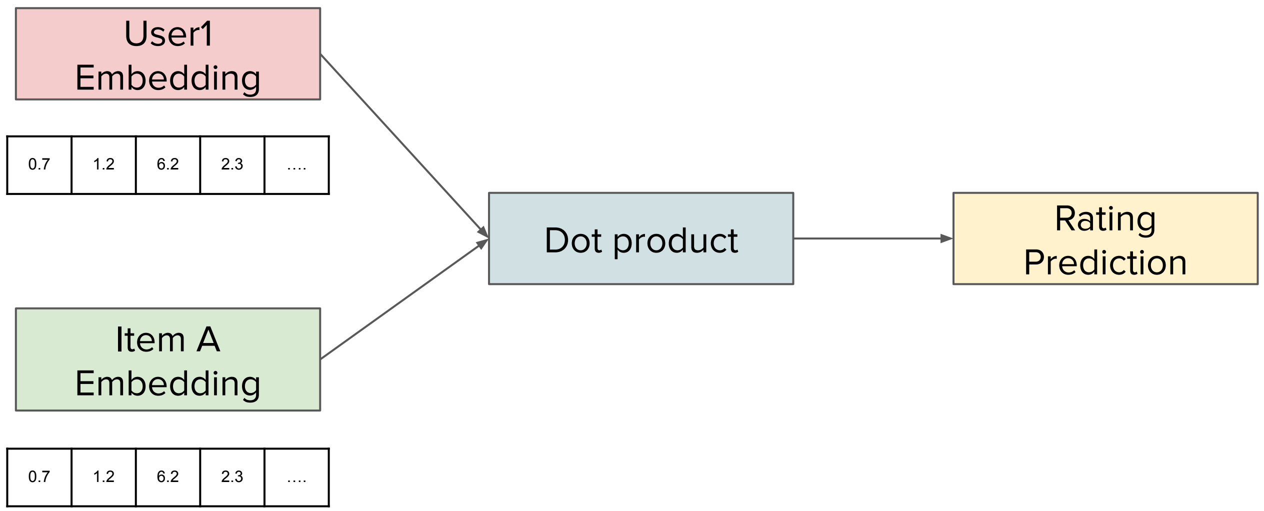
Squid Game	The Crown	Breaking Bad	The Office
0.42	0.19	0.97	0.12
0.31	0.63	0.82	0.25

$$\min_{U \in \mathbb{R}^{mxd}, V \in \mathbb{R}^{nxd}} \sum_{(i,j) \in obs} (R_{i,j} - \langle U_i, V_j \rangle)^2$$

Note: Only learn from observed interactions

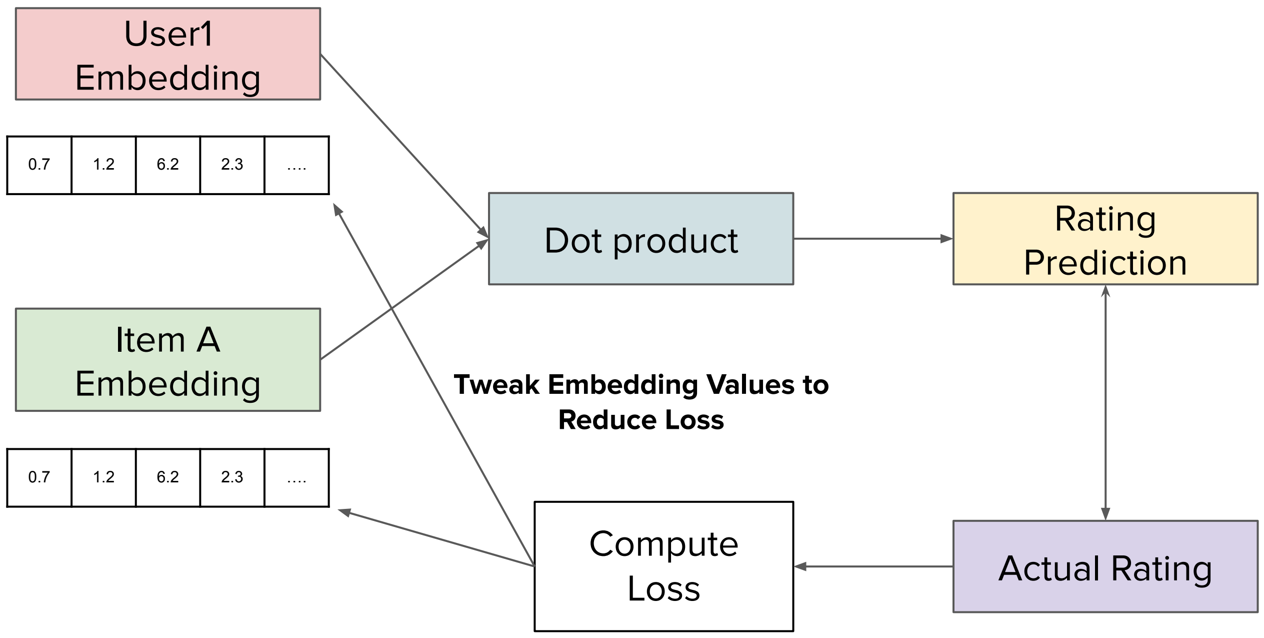
And minimize objective function with gradient descent, etc.

To Visually Summarize



To make a prediction, dot product the user and item embedding

Then compute loss with the actual rating, and tweak embedding according to gradient of the loss



Collaborative Filtering Pros and Cons

Pros

+ Unlike Content-based Filtering, there is **no domain knowledge** needed because embeddings are learned automatically based on user activity
+ **Serendipity**
  + Unlike Content-based filtering, model can still recommend items unrelated to current set of items the user interacts with, but still may be interested in

Cons

- **Cold Start Problem**
- **No Contextual Information Included**
  - Only learn from implicit data
- **Popularity Bias**
  - Tends to recommend popular items more

Quick Aside:
What if users don’t give us explicit feedback (i.e. no ratings)?

	Squid Game	The Crown	Breaking Bad	The Office
Alice	5	2		5
Bob		3		4
Chris	2		4	2
Derek	4	3		

Then we would use implicit data (whether user interacted with the item, which is binary)

	Squid Game	The Crown	Breaking Bad	The Office
Alice	1	1	0	1
Bob	0	1	0	1
Chris	1	0	1	1
Derek	1	1	0	0

Note: How we solve the problem with matrix factorization is still the exact same

Things to watch out for when using Implicit Data

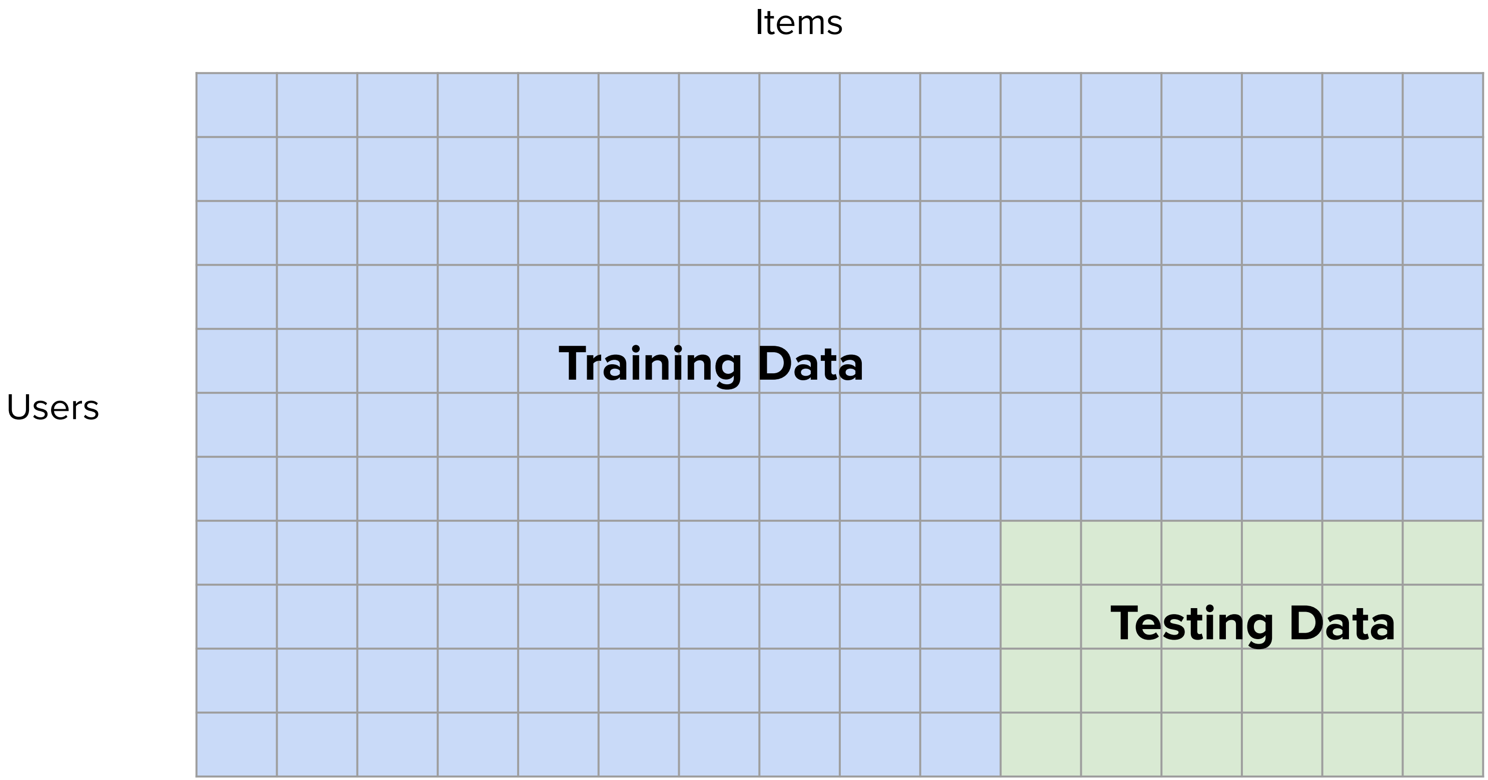
- We only know if user interacted with it or not, and **we don't know for sure if the user actually liked the item or not.**
    - Industry practices to mitigate this problem
        - Netflix
            - **Look at how much of the video they completed**, at what cadence
                - If they are the type of user to typically binge watch everything and they only watched the first episode, then this is perhaps a bad item to recommend
                - On the other hand, if the user typically only watches one episode a day anyways, we don't know immediately if this is an item the user likes
        - Facebook/Instagram
            - **Measure how long they were paused on the particular post** (implying they were actively observing the content)
    - Essentially, any information that hints at the **quality of interaction** the user had with an item can be used to modify the binary dataset

Questions?

Let’s take a 10 min break!

Evaluating Recommender Systems

How Do We Evaluate Recommender Systems?



We hold out *X* percentage of ratings from *Y* percentage of users to use as testing data

Evaluating Predictions

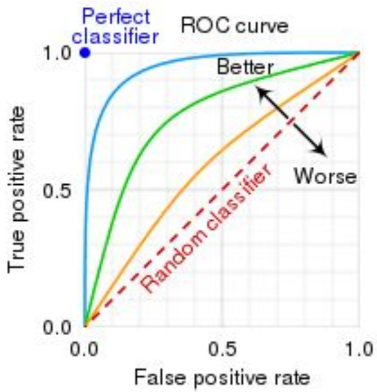


Image from [here]

- When given Explicit Ratings - “Regression” Metrics
  - RMSE
  - Rank correlation
    - Calculate correlation metrics between predictions and user’s ratings
- When given Implicit Ratings (0/1) - Binary Classification Metrics
  - Precision
  - Area Under ROC Curve (AUC)
    - Trade-off curve between false positives and false negatives
- **Ranking Metrics**
  - Predict some score with the model regardless of explicit or implicit
    - Then sort them in descending order of the score, thereby ranking the recommendation
  - MAP@K

Mean Average Precision@K (MAP@K)

- First, remember from earlier in the course:

$$Precision = \frac{\text{Number of Correct Positives}}{\text{Number of Times Model Predicted Positive}} = \frac{\text{Number of times our recommendation is relevant}}{\text{Number of total items we recommended}}$$

$$Recall = \frac{\text{Number of Correct Positives}}{\text{Number of Positives}} = \frac{\text{Number of times our recommendation is relevant}}{\text{Number of all possible recommendable items}}$$

But Precision and Recall **doesn't care about the order** of things, but we are trying to measure the quality of a ranked list

Excellent blog on this topic where the images and examples are taken from here

Let’s incorporate information about order with Precision@K

Algorithm Outputs...

P(k=3) = 1/3

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
~~4~~	~~Auto loan~~	~~False positive~~
~~5~~	~~HELOC~~	~~Correct Positive~~
~~6~~	~~College Fund~~	~~Correct positive~~
~~7~~	~~Personal loan~~	~~False positive~~

Algorithm Outputs...

Rank	Product Recommended	Result
1	Credit card	Correct positive
2	Christmas Fund	False positive
3	Debit Card	False positive
4	Auto loan	False positive
5	HELOC	Correct Positive
6	College Fund	Correct positive
~~7~~	~~Personal loan~~	~~False positive~~

P(k=6) = 3/6

- Precision @ 3, or P(k=3), is precision if we were only able to give our top 3 best guesses. P(k=6) allows up to 6 best predictions
- @K essentially means “Up until cutoff k”

Excellent blog on this topic where the images and examples are taken from here

Average Precision @ N

It could be the case that our recommendations are really good for the first 3 best predictions, and it gets really bad after that.

To get a holistic view of our quality of ranked list, let's average the Precision @ K's together for varying K!

$$AP@N = \frac{1}{N}\sum_{k=1}^{N}(P(k) \text{ if k-th item recommendation was correct})$$

Excellent blog on this topic where the images and examples are taken from [here]()

Average Precision @ N Example

$$AP@N = \frac{1}{N}\sum_{k=1}^{N}(P(k) \text{ if k-th item recommendation was correct})$$

We are recommending N=3 products

Recommendations	Precision @k's	AP@3
[0, 0, 1]	[0, 0, 1/3]	(1/3)**(1/3)** = 0.11

$P(k=1) = P(k=2) = 0, P(k=3) = 1/3$

The 3rd recommendation was a hit

There are 3 P(k)'s,
so divide by 3

Sum of P(K)'s is 0+0+1/3.
You can think of this as
"collecting awards" for
each correct prediction

Excellent blog on this topic where the images and examples are taken from here

Average Precision @ N Example

$$AP@N = \frac{1}{N}\sum_{k=1}^{N}(P(k) \text{ if k-th item recommendation was correct})$$

Recommendations	Precision @k's	AP@3
[0, 0, 1]	[0, 0, 1/3]	(1/3)**(1/3)** = 0.11
[0, 1, 1]	[0, 1/2, 2/3]	(1/3)[(1/2) + **(2/3)**] = 0.38
[1, 1, 1]	[1/1, 2/2, 3/3]	(1/3)[(1) + (2/2) + **(3/3)**] = 1

Excellent blog on this topic where the images and examples are taken from [here]

Average Precision @ N Another Example

We are recommending N=3 products

__Recommendations___	___Precision @k's__	__AP@3__
[1, 0, 0]	[1/1, 1/2, 1/3]	(1/3)(1) = 0.33
[0, 1, 0]	[0, 1/2, 1/3]	(1/3)(1/2) = 0.15
[0, 0, 1]	[0, 0, 1/3]	(1/3)(1/3) = 0.11

Notice here, each user **only had one correct prediction** in their recommendation, but AP@3 is **highest for the first user** because **AP@K rewards front-loading correct predictions**

Excellent blog on this topic where the images and examples are taken from [here]()

Mean Average Precision @ K (MAP@K)

Precision @ K only measures quality of ranking for **ONE USER**
So let's **average** Precision @ K for **ALL USERS** to get MAP@K

Recommendations	Precision @k's	AP@3
[1, 0, 0]	[1/1, 1/2, 1/3]	(1/3)(1) = 0.33
[0, 1, 0]	[0, 1/2, 1/3]	(1/3)(1/2) = 0.15
[0, 0, 1]	[0, 0, 1/3]	(1/3)(1/3) = 0.11

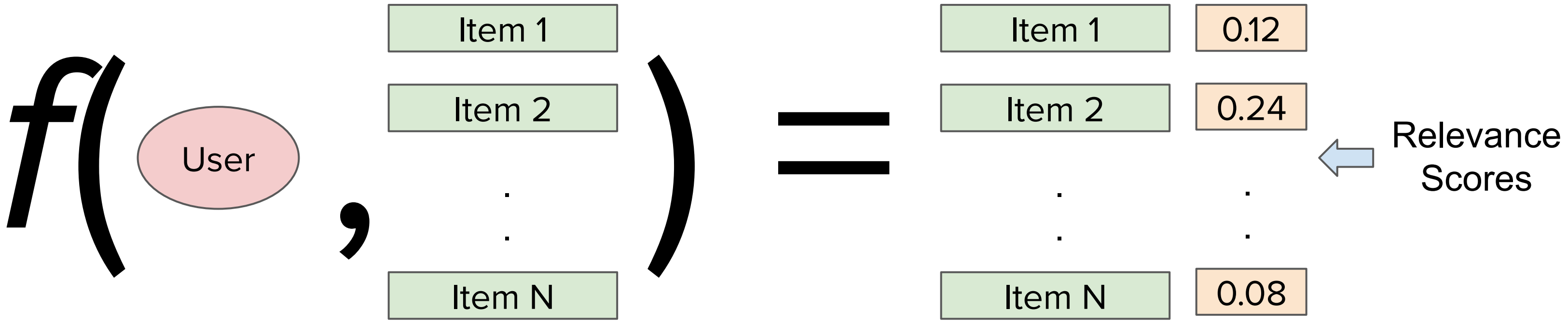
$$MAP@N = \frac{1}{|U|}\sum_{u=1}^{|U|}(AP@N)_u$$

MAP@K = (0.33 + 0.15 + 0.11) / 3 = 0.197

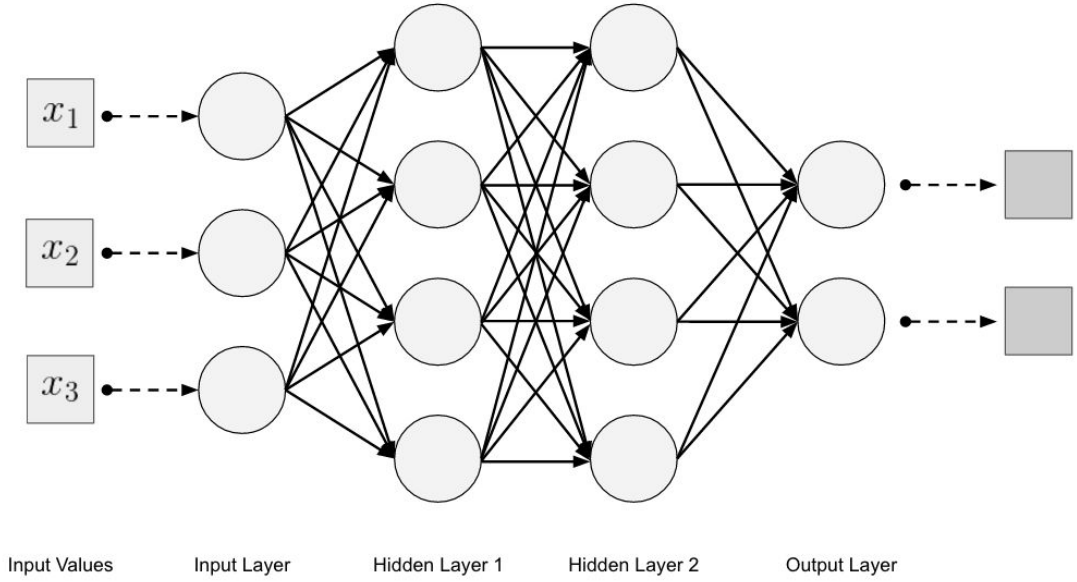
Excellent blog on this topic where the images and examples are taken from here

Deep Learning-based Approaches

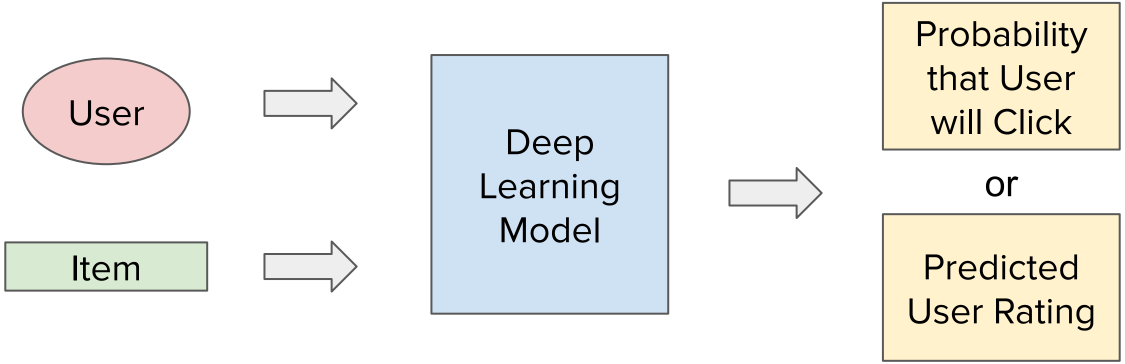
We discussed earlier that a recommender system needs to be able to **score a user-item pair**



With the **Universal Approximation Theorem**, deep learning should be able to learn to score user-item pairs as well!



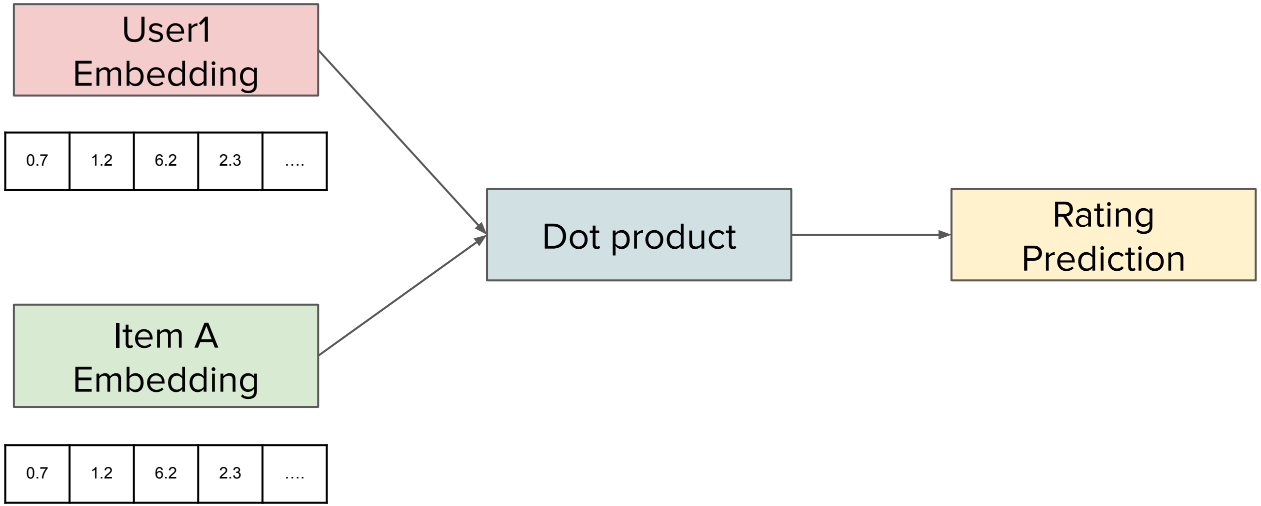
Click Through Rate (CTR) Models



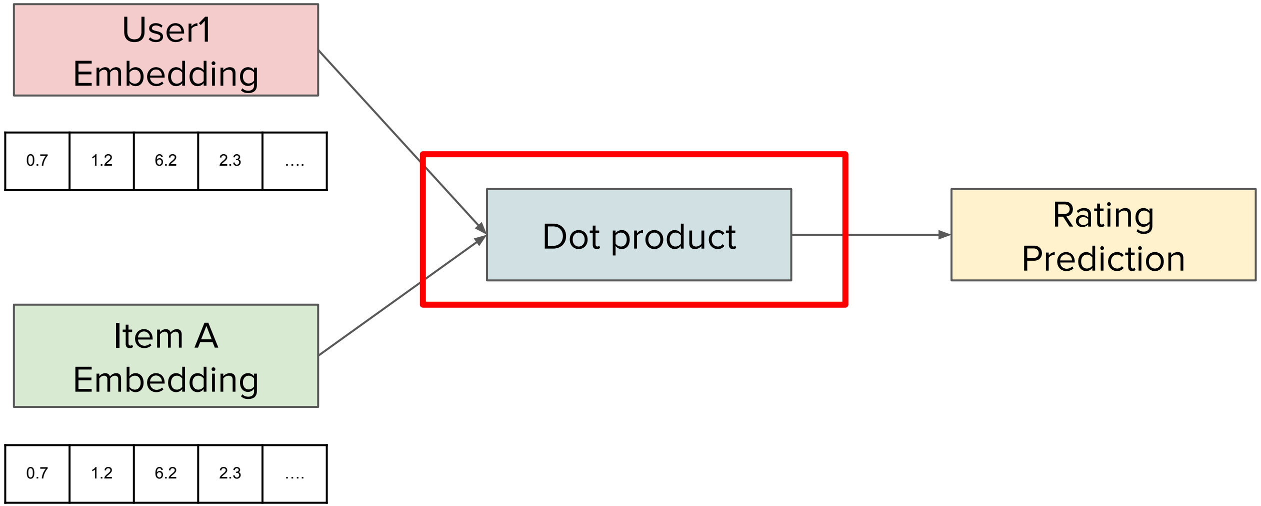
Model takes in ***some* representation** of the user and an item, and it **predicts the probability the user will click**, or the user rating, depending on if we are training it with implicit or explicit data

After model is trained, model can be used to score the user-item pair

Let's think of ways to improve on the Matrix Factorization from earlier

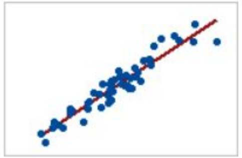
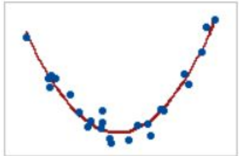
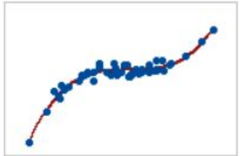


A good recommender system needs to be able to **model various interactions** between a user and an item to predict well



Matrix Factorization uses a **simple dot product** to model that interaction, and a dot product is a **degree 2 interaction**, which means it can’t capture complex interactions

Let’s take a step back - What does it mean to model only “Degree 2” Interaction?

Model order	Example
Linear $Y = b_o + b_1X$ (first order)	
Quadratic $Y = b_o + b_1X + b_{11}X^2$ (second order)	
Cubic $Y = b_o + b_1X + b_{11}X^2 + b_{111}\, X^3$ (third order)	

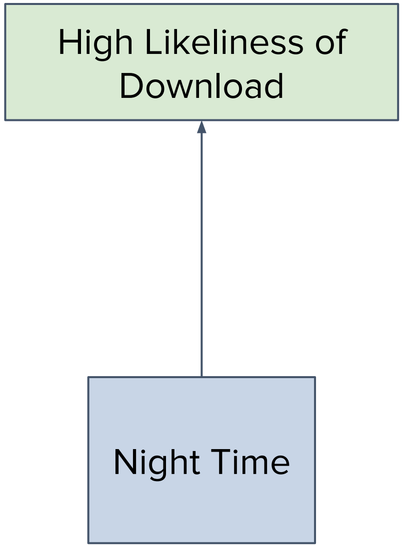
Degree/Order = Complexity of Model

- Higher the degree, the more complex pattern the model can fit

Google Play Example

- Let's say we are building a recommender system to recommend apps to install on the Google Play Store

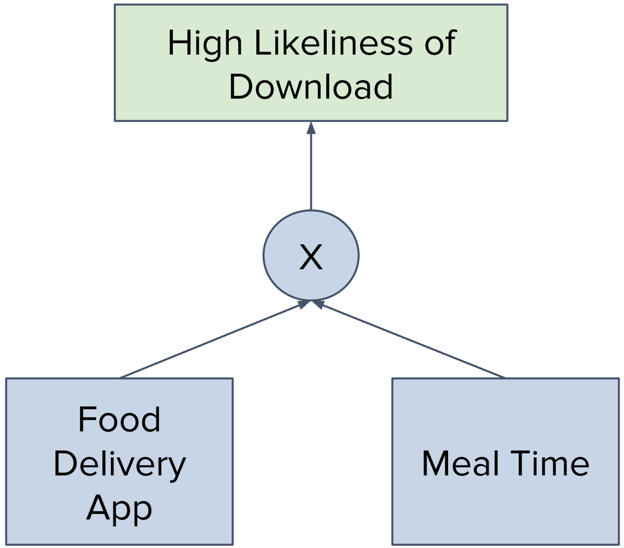
    - Degree 1 Interaction: If model captures the **linear relationship** that night time generally converts to higher likeliness of downloads



Degree 1 Interaction

Google Play Example

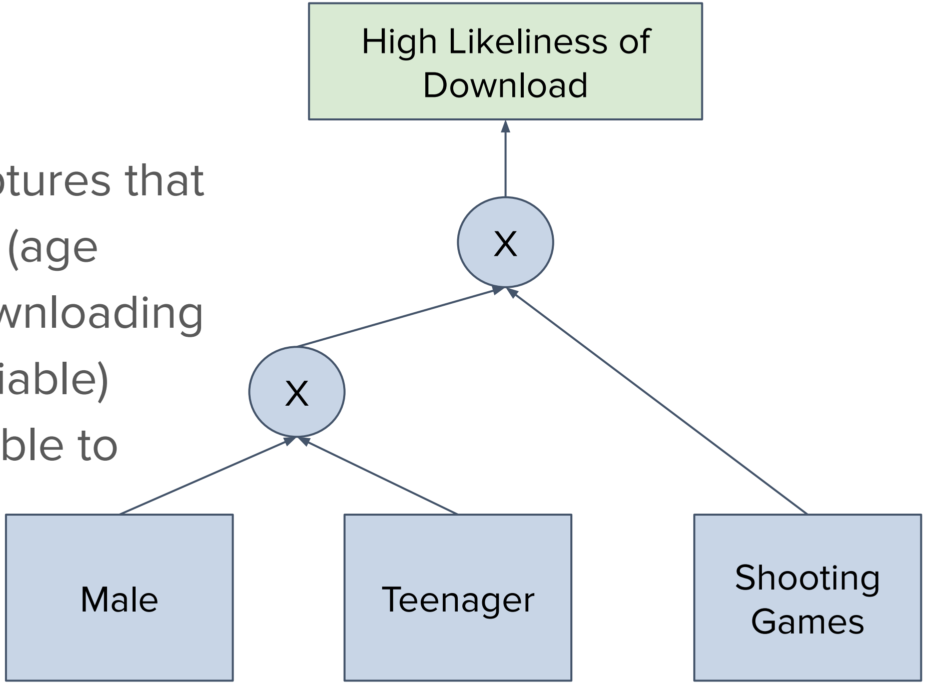
- Degree 2 Interaction: If model captures that **during meal time** (time variable), **food delivery app** (app category variable) download likeliness goes up



Degree-2 Interaction

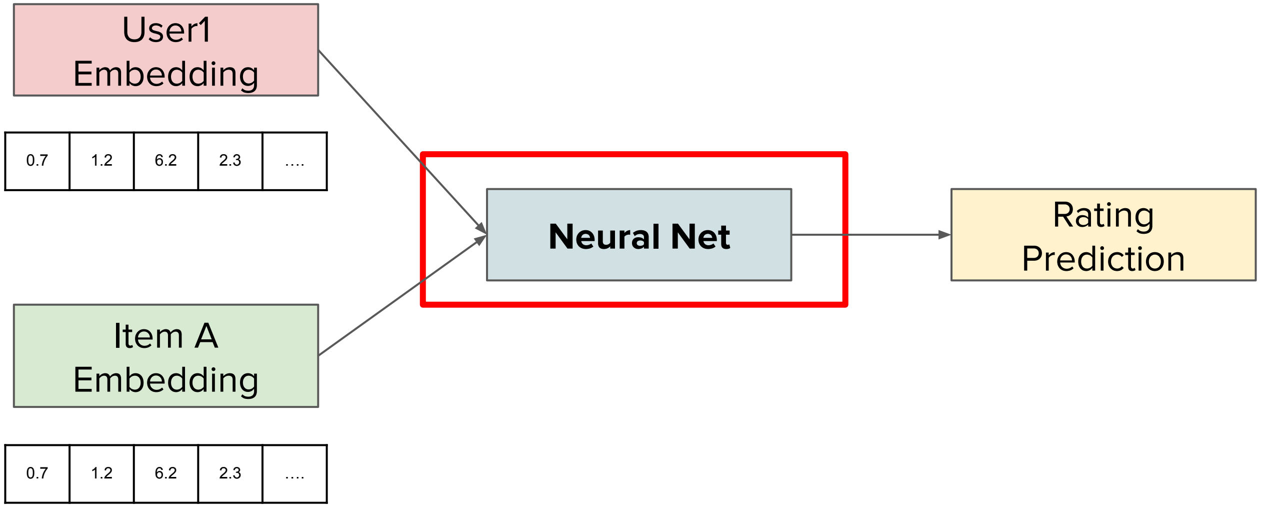
Google Play Example

- Degree 3 Interaction: If model captures that **male** (gender variable) **teenagers** (age variable) has high likeliness of downloading **shooting games** (game genre variable)
- Matrix Factorization wouldn't be able to model this relationship



Degree-3 Interaction

Why don’t we replace the dot product with some other function that can model higher expressive power?



Let’s **replace** the dot product with a **neural network** (Multi-Layer Perceptrons) and have it learn **higher degree** interactions between the user and item embeddings!

Neural Collaborative Filtering (2017)

Neural Collaborative Filtering*

Xiangnan He
National University of Singapore, Singapore
xiangnanhe@gmail.com

Lizi Liao
National University of Singapore, Singapore
liaolizi.llz@gmail.com

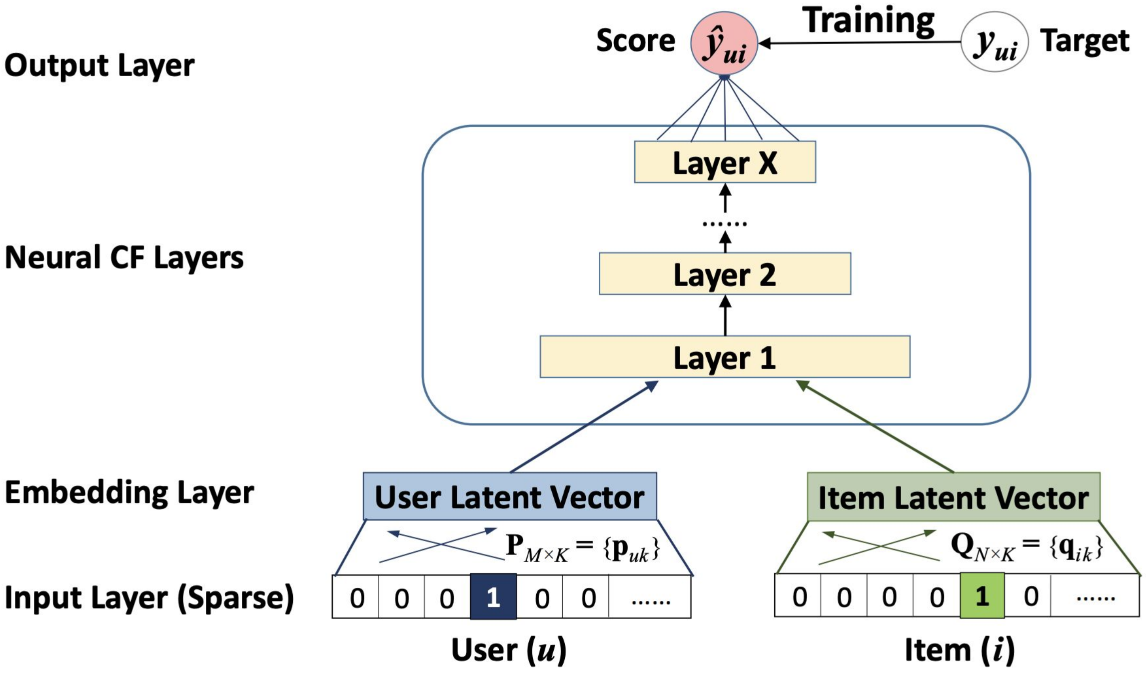
Hanwang Zhang
Columbia University
USA
hanwangzhang@gmail.com

Liqiang Nie
Shandong University
China
nieliqiang@gmail.com

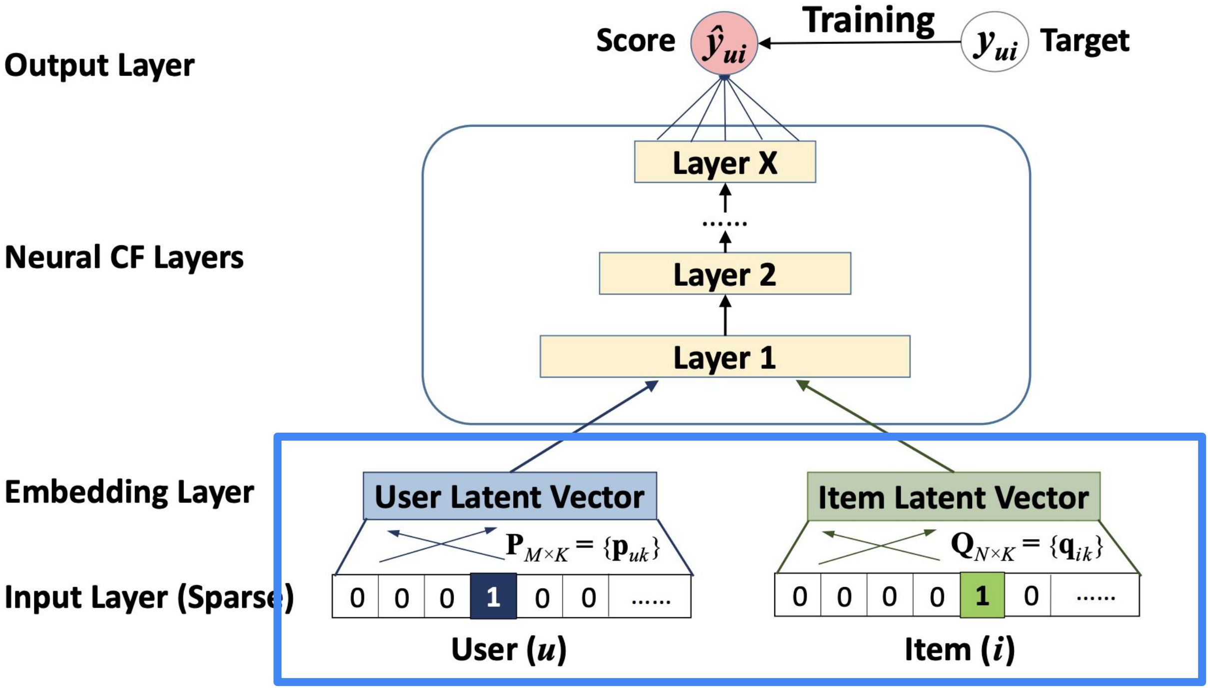
Xia Hu
Texas A&M University
USA
hu@cse.tamu.edu

Tat-Seng Chua
National University of Singapore, Singapore
dcscts@nus.edu.sg

Neural Collaborative Filtering Model Architecture

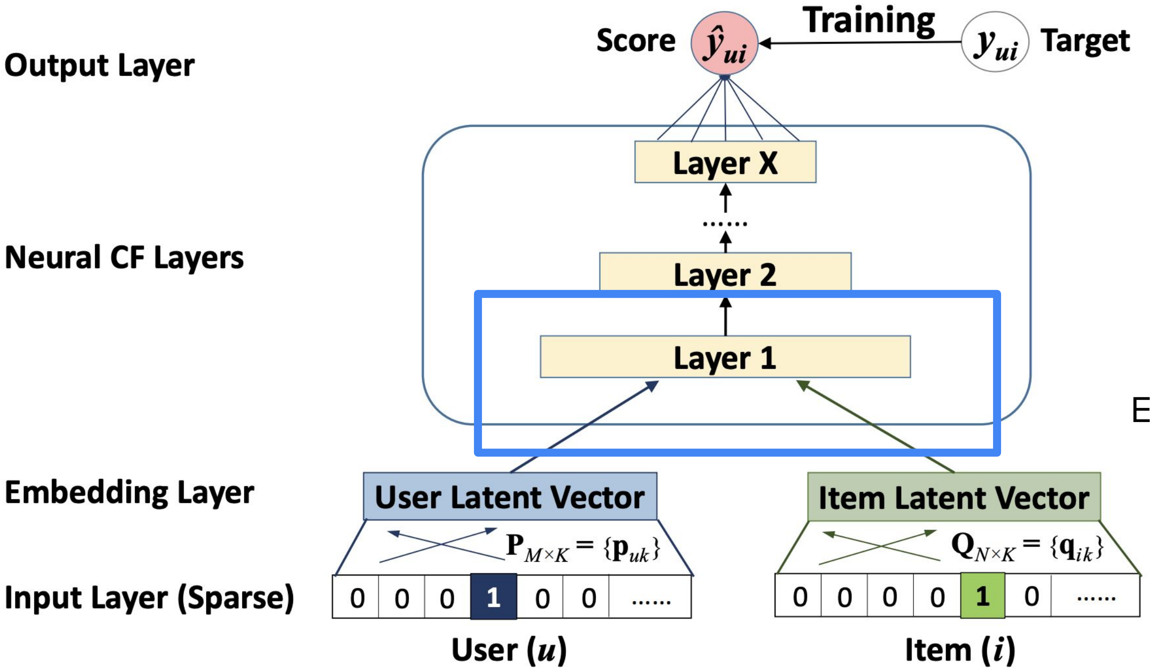


Neural Collaborative Filtering Model Architecture



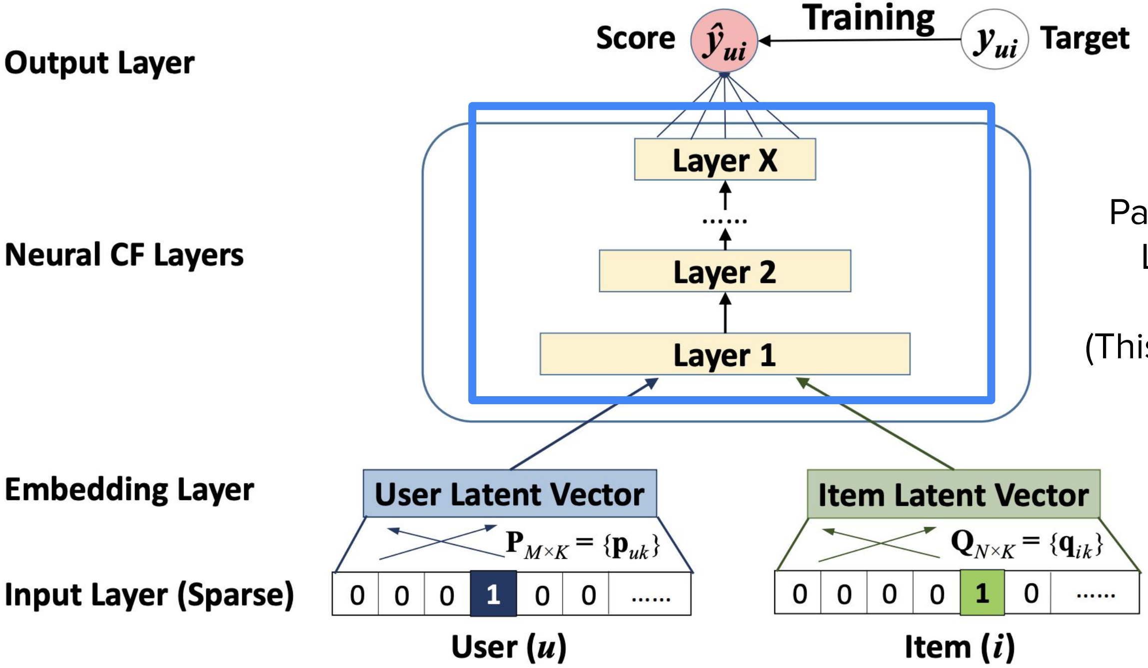
Embed User and Item into Dense Vectors

Neural Collaborative Filtering Model Architecture



Concatenate
Embedding Vectors

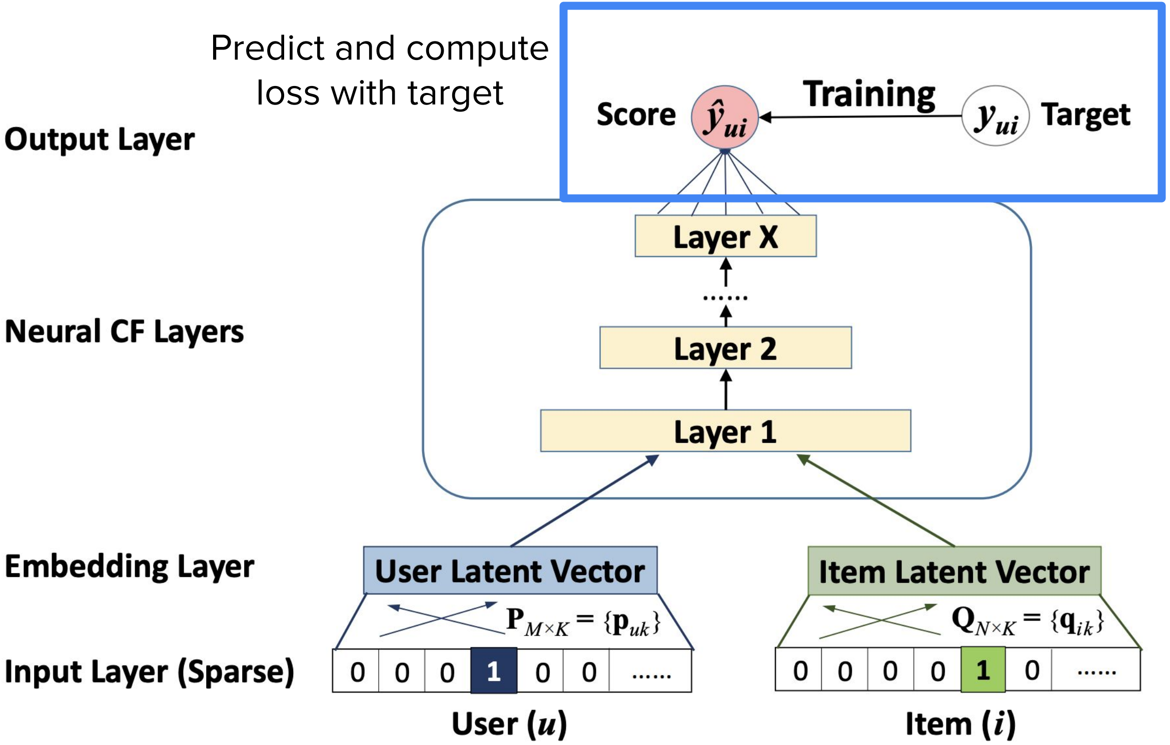
Neural Collaborative Filtering Model Architecture



Pass Through MLP to Learn Non-Linear Interactions (This acts in place of the dot product)

Neural Collaborative Filtering Model Architecture

Predict and compute loss with target



But **just** learning higher order interaction is **not always good**...

Deep neural networks with embeddings can **over-generalize** and recommend less relevant items when the user-item interactions are sparse

-> Let's have the neural network take into account **low order interactions** too!

Wide & Deep Learning for Recommender Systems

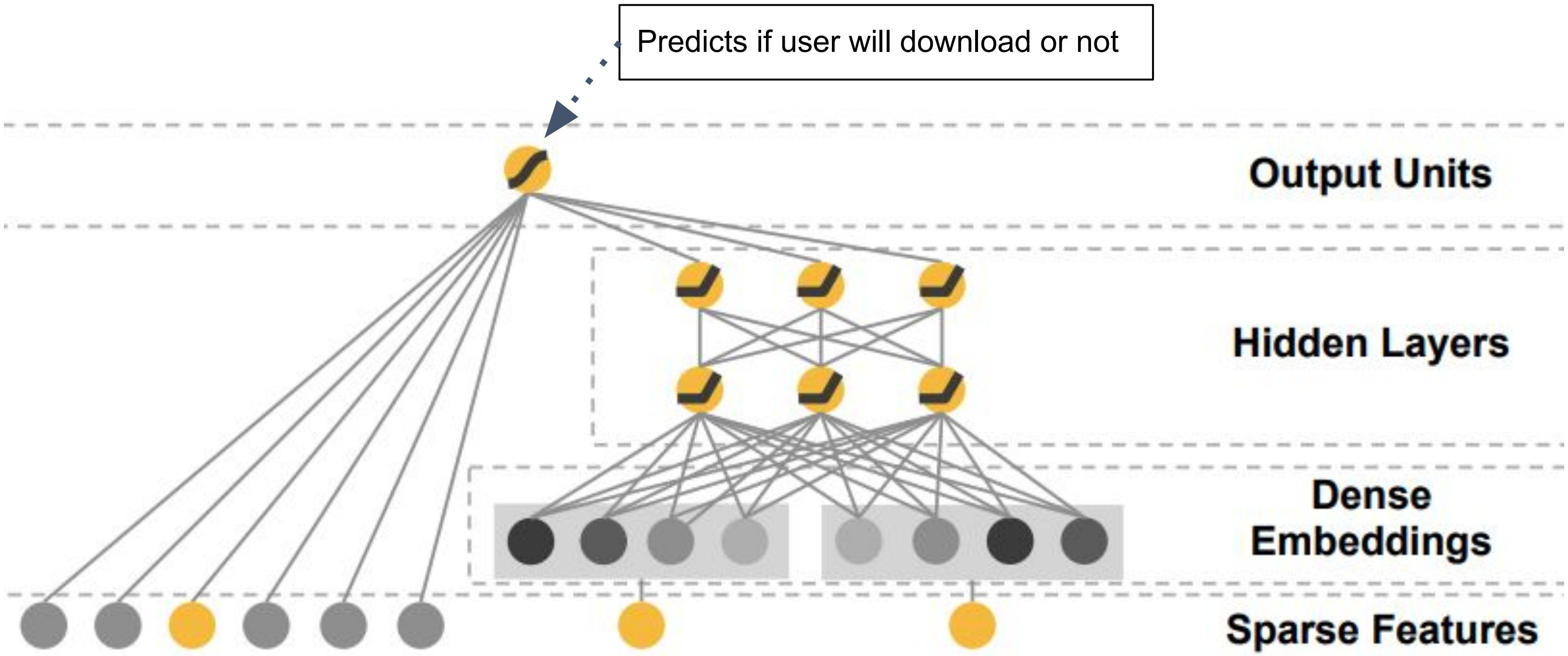
- Published by Google in 2016
- Used for recommending apps in the Play Store
  - Served to billions of users

Wide & Deep Learning for Recommender Systems

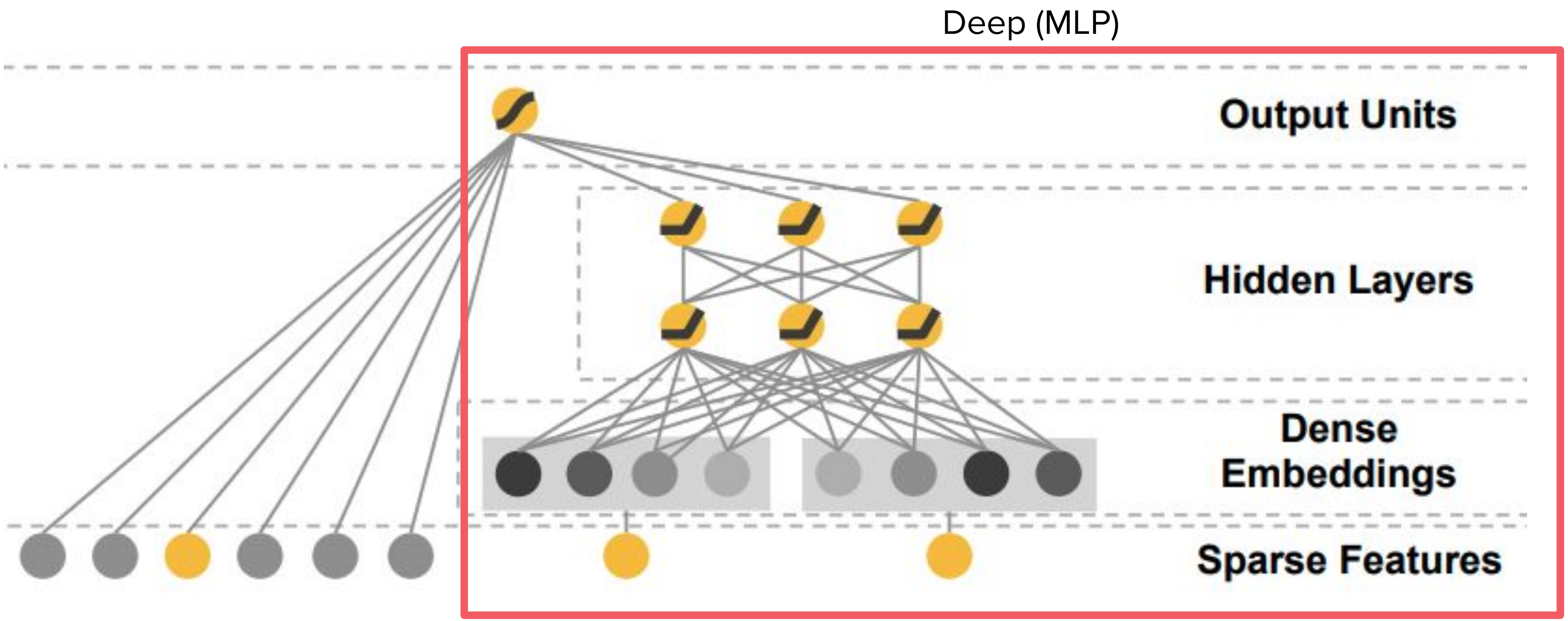
Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra,
Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil,
Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah

Google Inc.*

Wide & Deep Model Architecture

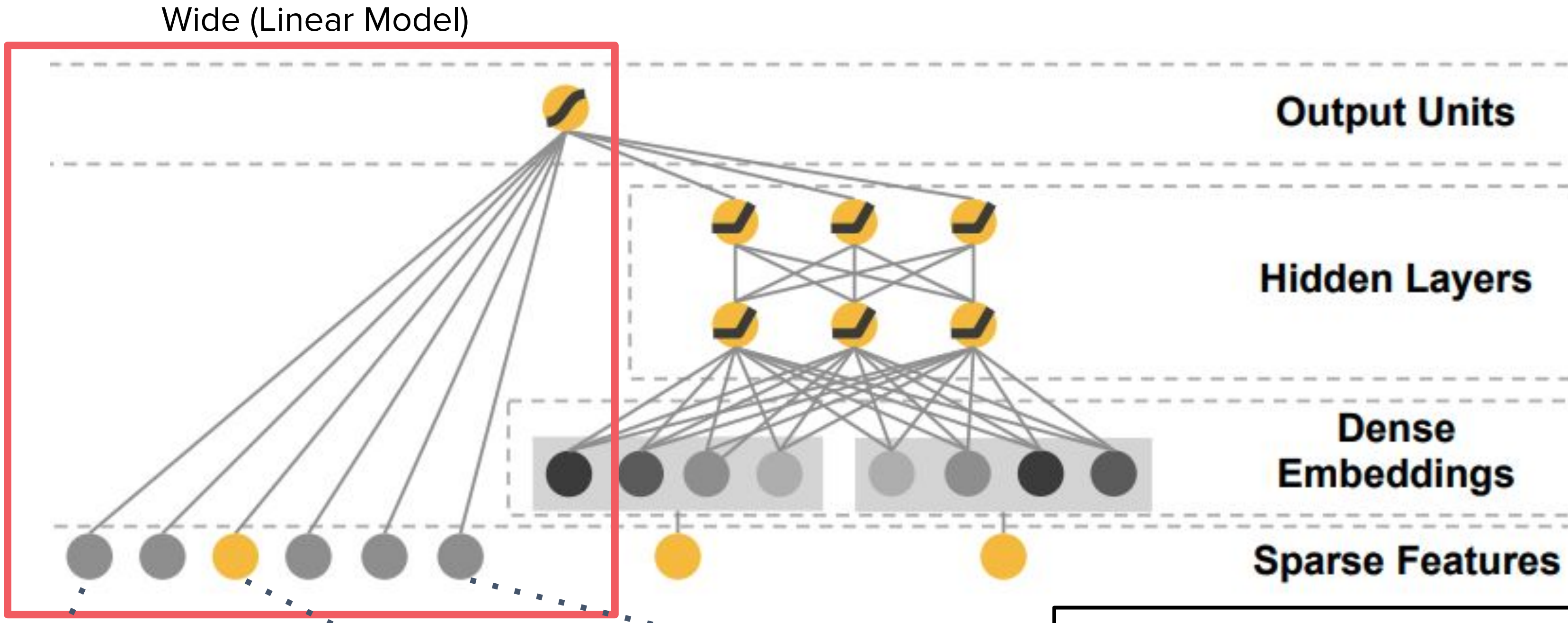


Deep Component Models **Complex Interactions** Between Categorical Values



Same Architecture as Neural Collaborative Filtering, but now we are learning embeddings for different categorical features

Wide Component Models **Simple Interactions**



Downloaded Spotify

Downloaded Shazam

AND(Downloaded Spotify, Downloaded Shazam)

Non-Linearity and Low Order Interactions are Manually Engineered via Cross Product Features

Wide and Deep Components Serve **Complementary** Roles

Wide Component

- Models the the **simple**, frequent co-occurrences and correlations that it can **exploit** in recommendations
- Uses **order 1-2** interactions

Deep Component

- Models more **complex** relationships between different features to generalize well, even with unseen queries
- Learns **order 3+** (complex) interactions

Wide and Deep Components Serve **Complementary** Roles

Wide Component

- Models the the **simple**, frequent co-occurrences and correlations that it can **exploit** in recommendations
- Uses **order 1-2** interactions
- Handles **niche** cases that we know about but **don't have enough data** to have the deep part take into account
- Ex. Can memorize that Japanese Males in Age 18-24 category living in Ann Arbor all downloaded a specific Karaoke App

Deep Component

- Models more **complex** relationships between different features to generalize well, even with unseen queries
- Learns **order 3+** (complex) interactions
- Performs **badly for niche cases** with small training data
- Learns relationships between features that we can't even think of (ex. beer + diapers)

Wide and Deep Components Serve **Complementary** Roles

Wide Component

- Models the the **simple**, frequent co-occurrences and correlations that it can **exploit** in recommendations
- Uses **order 1-2** interactions
- Handles **niche** cases that we know about but **don't have enough data** to have the deep part take into account
- Ex. Can memorize that Japanese Males in Age 18-24 category living in Ann Arbor all downloaded a specific Karaoke App

Deep Component

- Models more **complex** relationships between different features to generalize well, even with unseen queries
- Learns **order 3+** (complex) interactions
- Performs **badly for niche cases** with small training data
- Learns relationships between features that we can't even think of (ex. beer + diapers)

Paper shows that Wide **AND** Deep Model Outperformed **JUST** Wide **OR** Deep Models in A/B Testing

But why **manually engineer interactions** when you can **automate** it

Let’s utilize a **Factorization Machine** to get feature interactions for every single combination of input features!

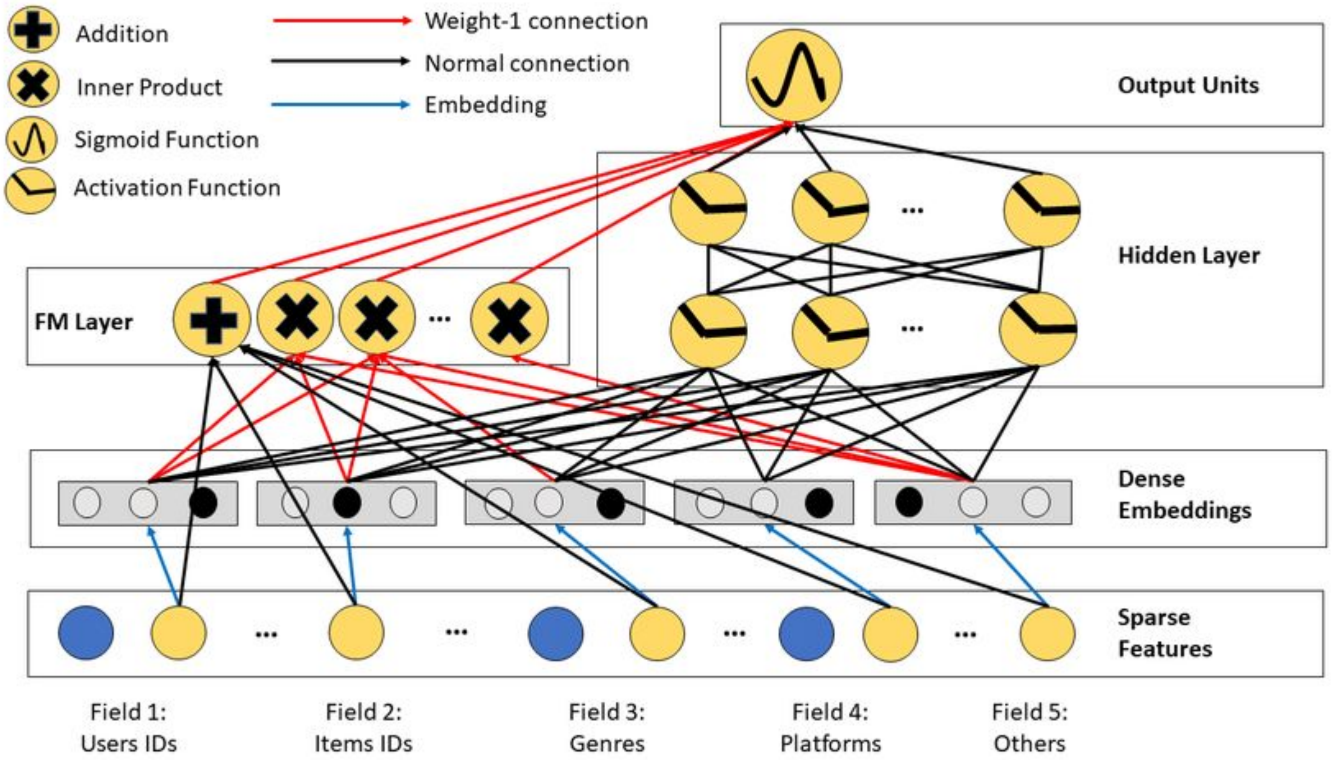
DeepFM (2017)

- Work by Huawei Research team
- Directly builds on Wide & Deep
- Adds a Factorization Machine (2010) to automatically calculate low-order interactions
  - Pairwise feature interactions between all features
  - No feature engineering needed apart from choosing raw inputs
- Just like Wide & Deep, FM portion (Wide) models low-order interactions and Deep portion models high-order interactions

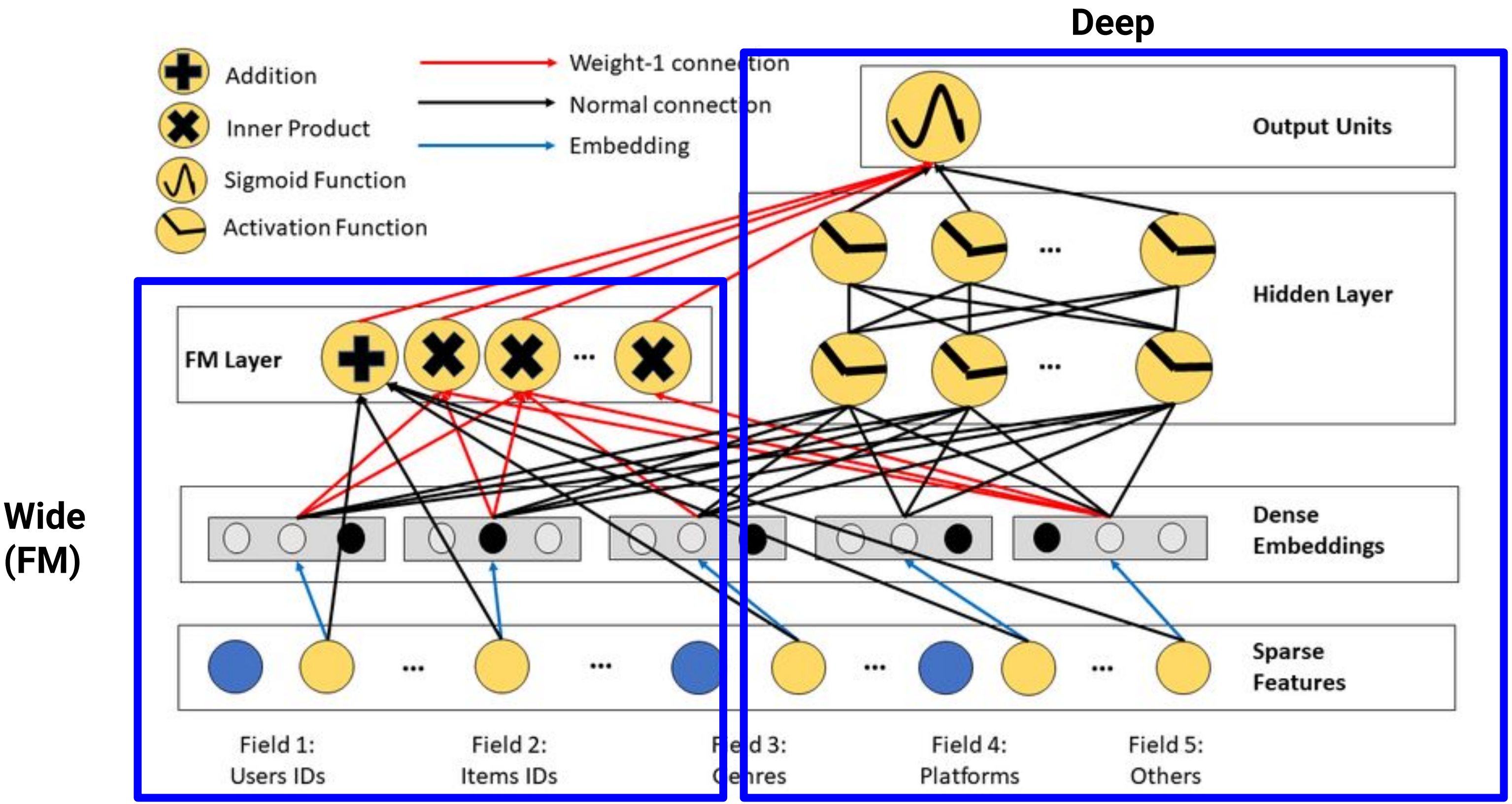
DeepFM: A Factorization-Machine based Neural Network for CTR Prediction

Huifeng Guo[*1] , Ruiming Tang[2], Yunming Ye[†1], Zhenguo Li[2], Xiuqiang He[2]
[1]Shenzhen Graduate School, Harbin Institute of Technology, China
[2]Noah's Ark Research Lab, Huawei, China
[1]huifengguo@yeah.net, yeyunming@hit.edu.cn
[2]{tangruiming, li.zhenguo, hexiuqiang}@huawei.com

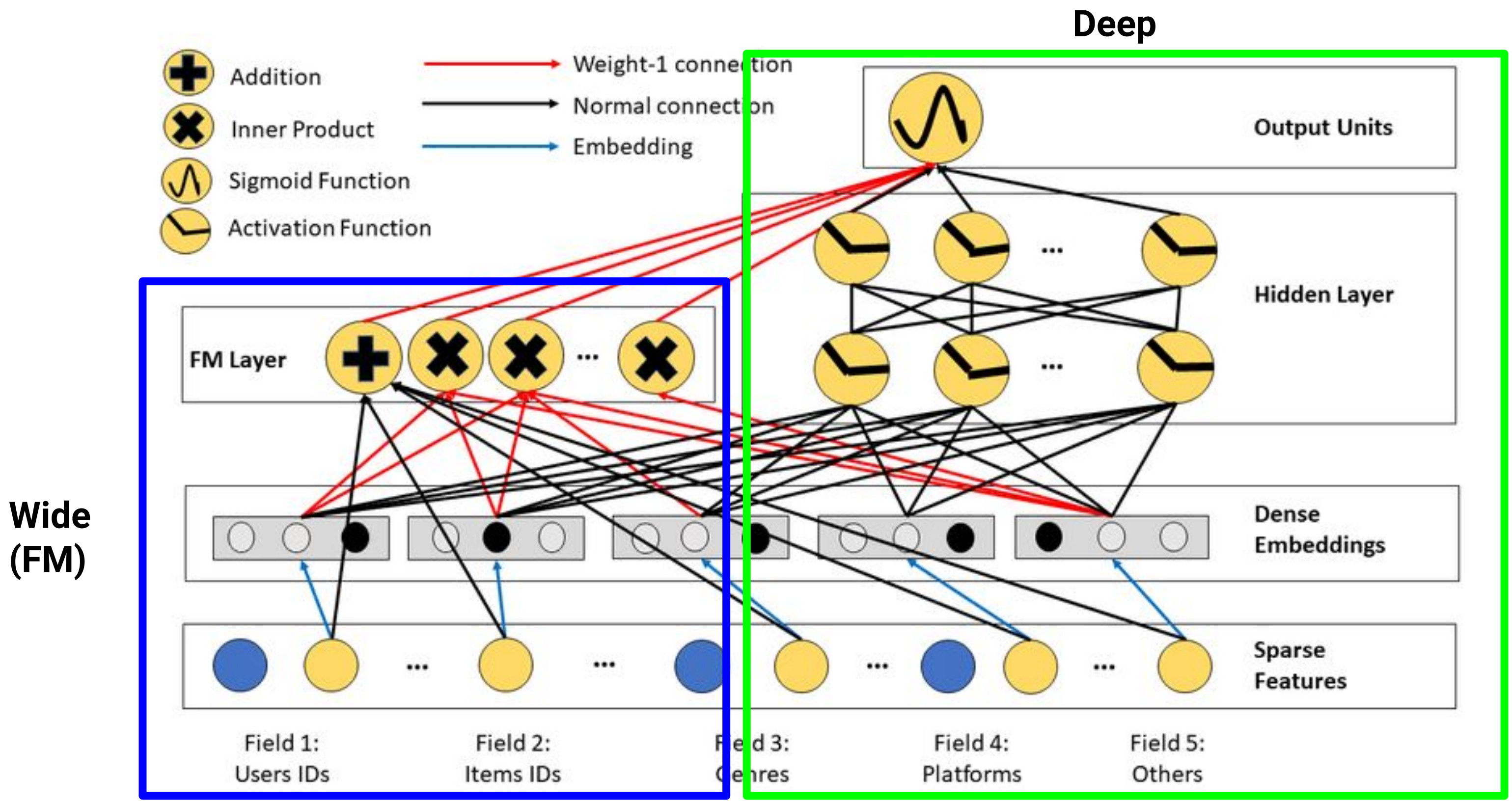
DeepFM Model Architecture



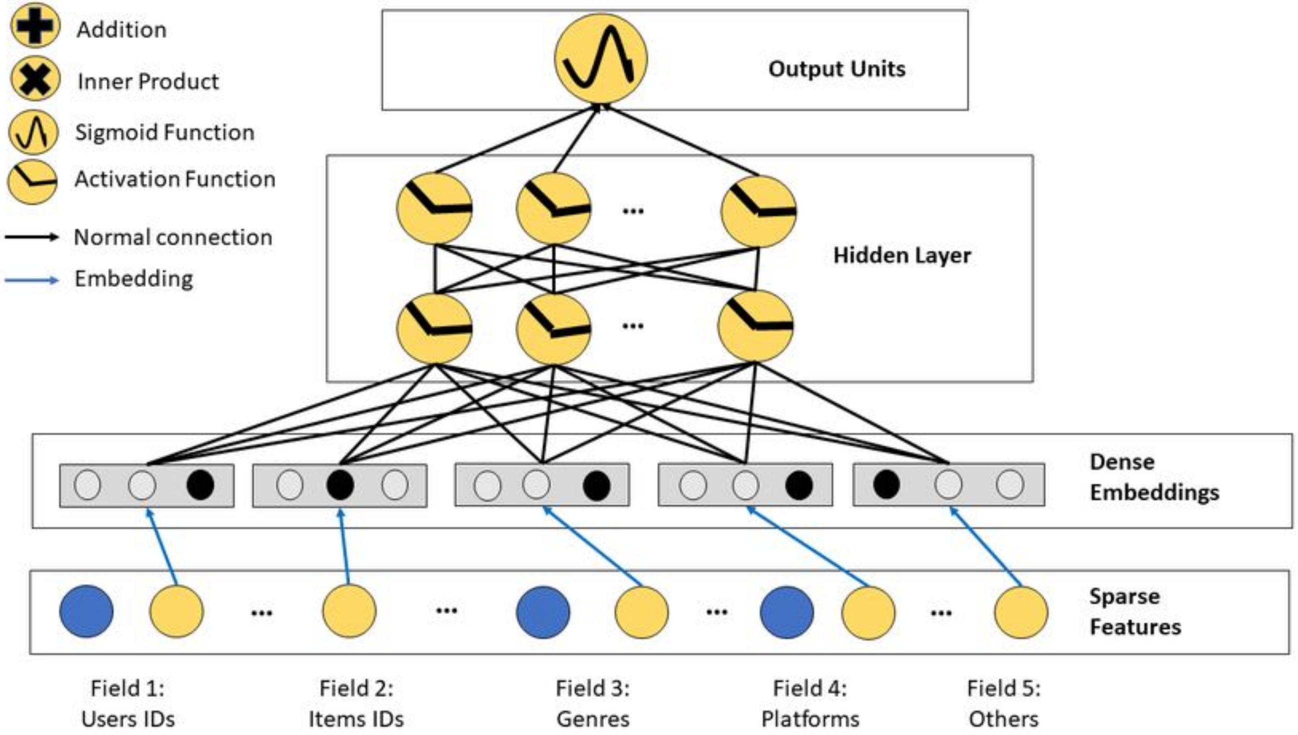
DeepFM Model Architecture



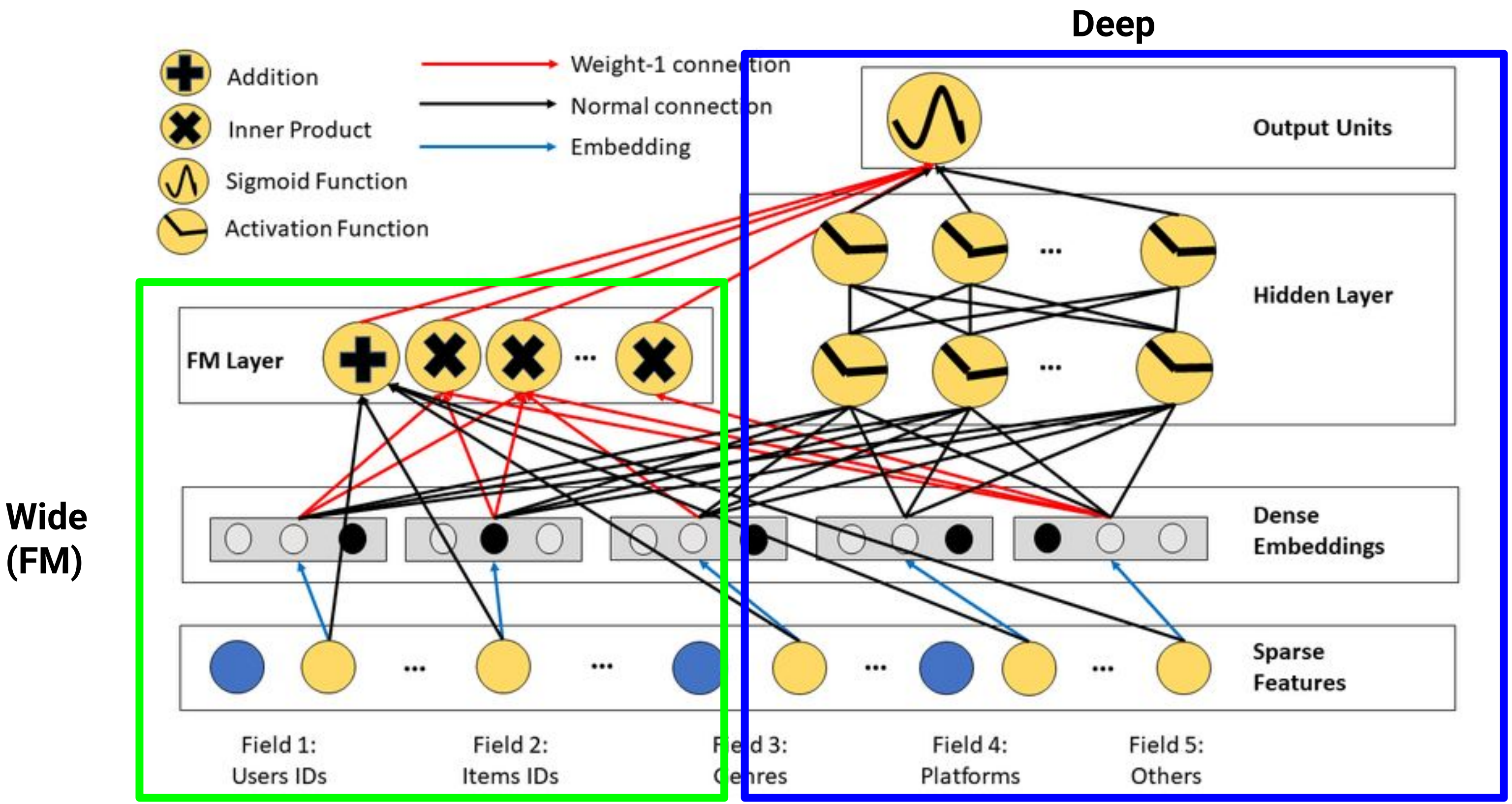
DeepFM Model Architecture



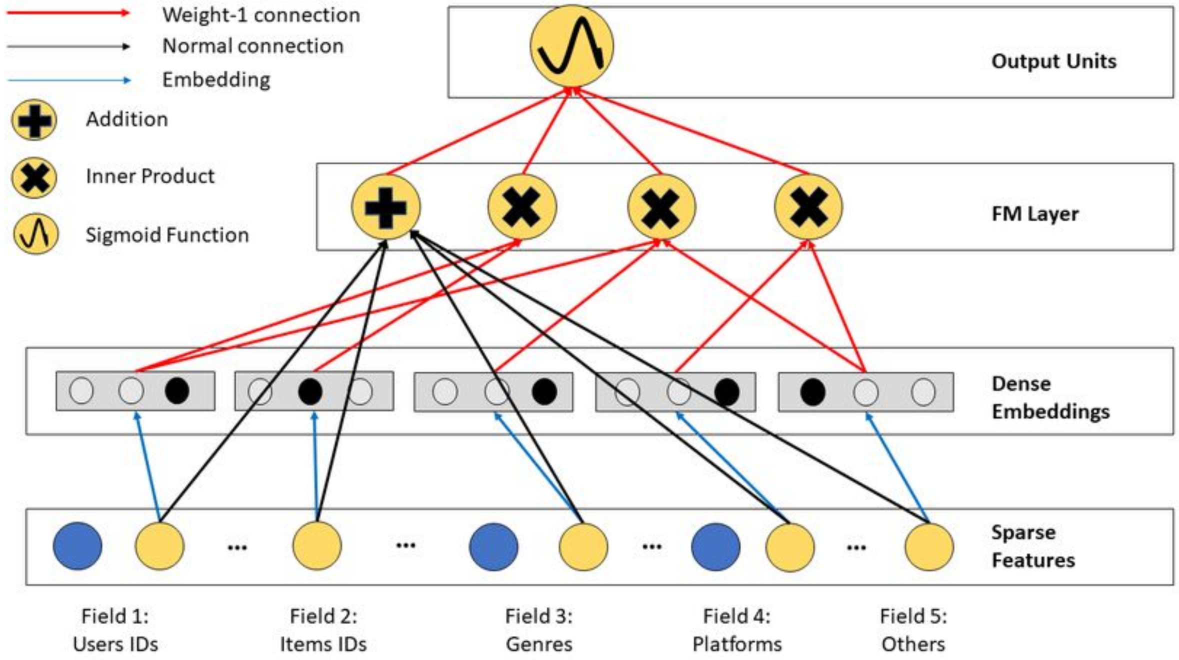
Deep Component is the **Same** as Wide & Deep Model



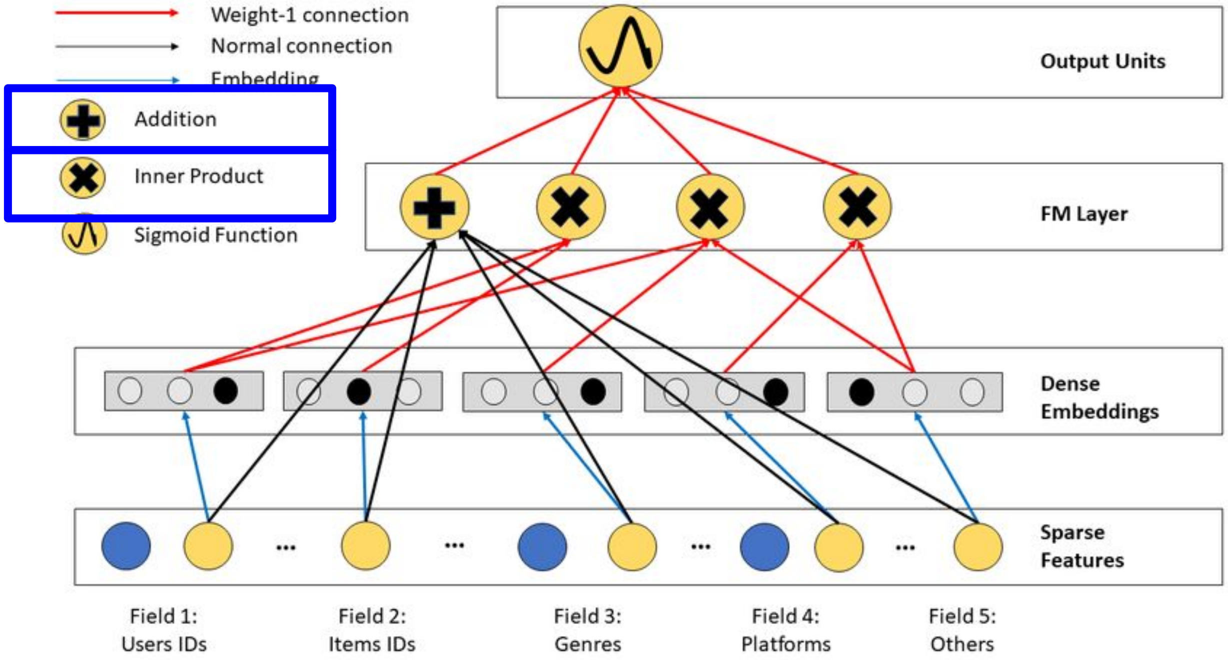
DeepFM Model Architecture



Factorization Machine Component of DeepFM

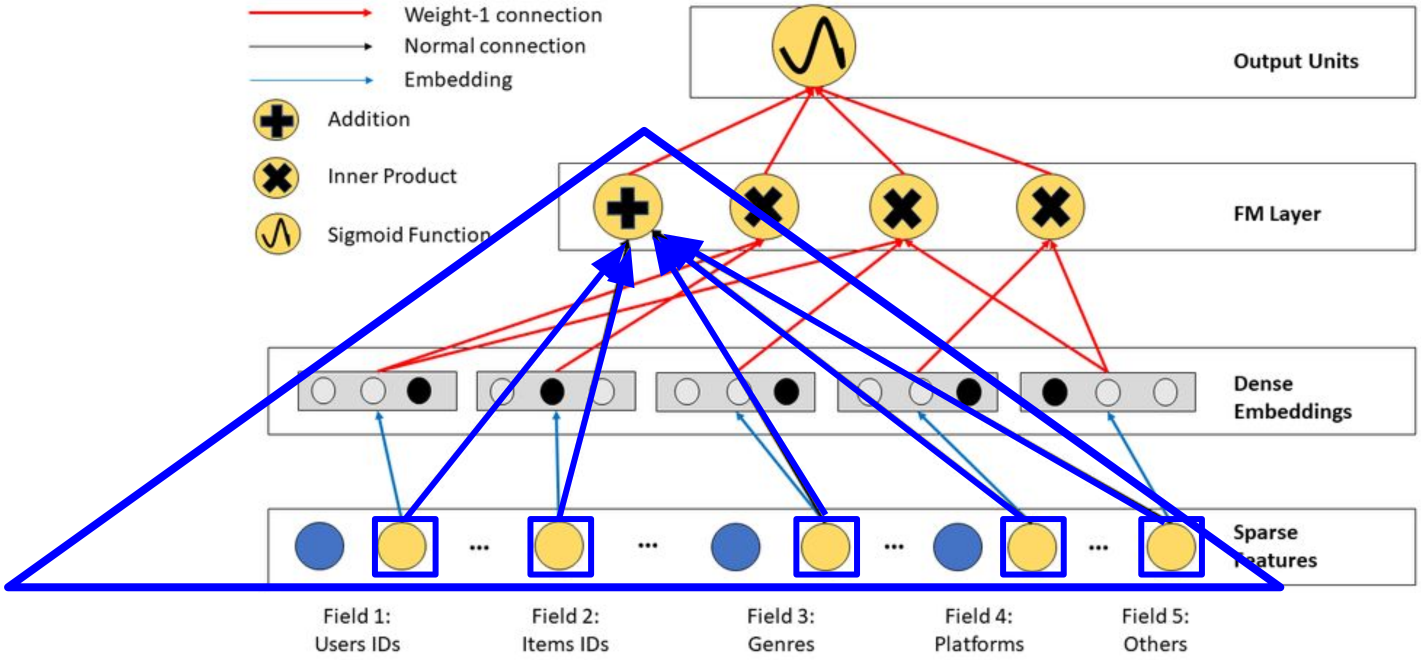


Factorization Machine Component of DeepFM



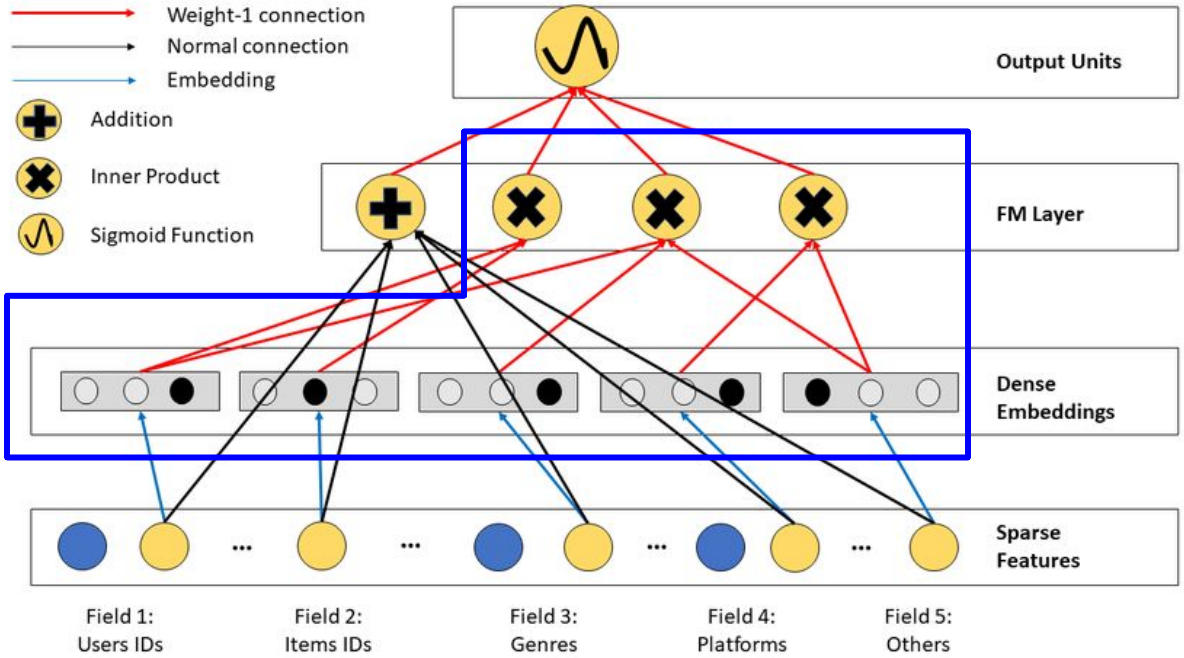
FM models order 1 interaction (addition) and order 2 interaction (multiplication)

Factorization Machine Component of DeepFM



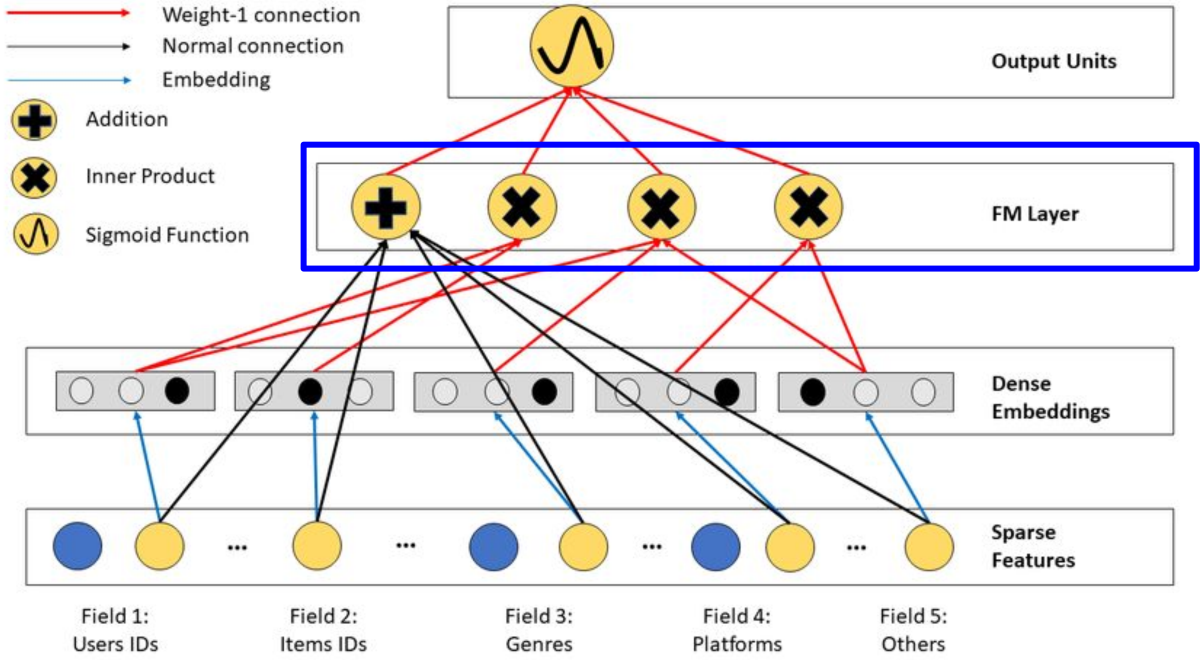
Order 1 Interaction: Simply Add the Raw Sparse Inputs Together

Factorization Machine Component of DeepFM



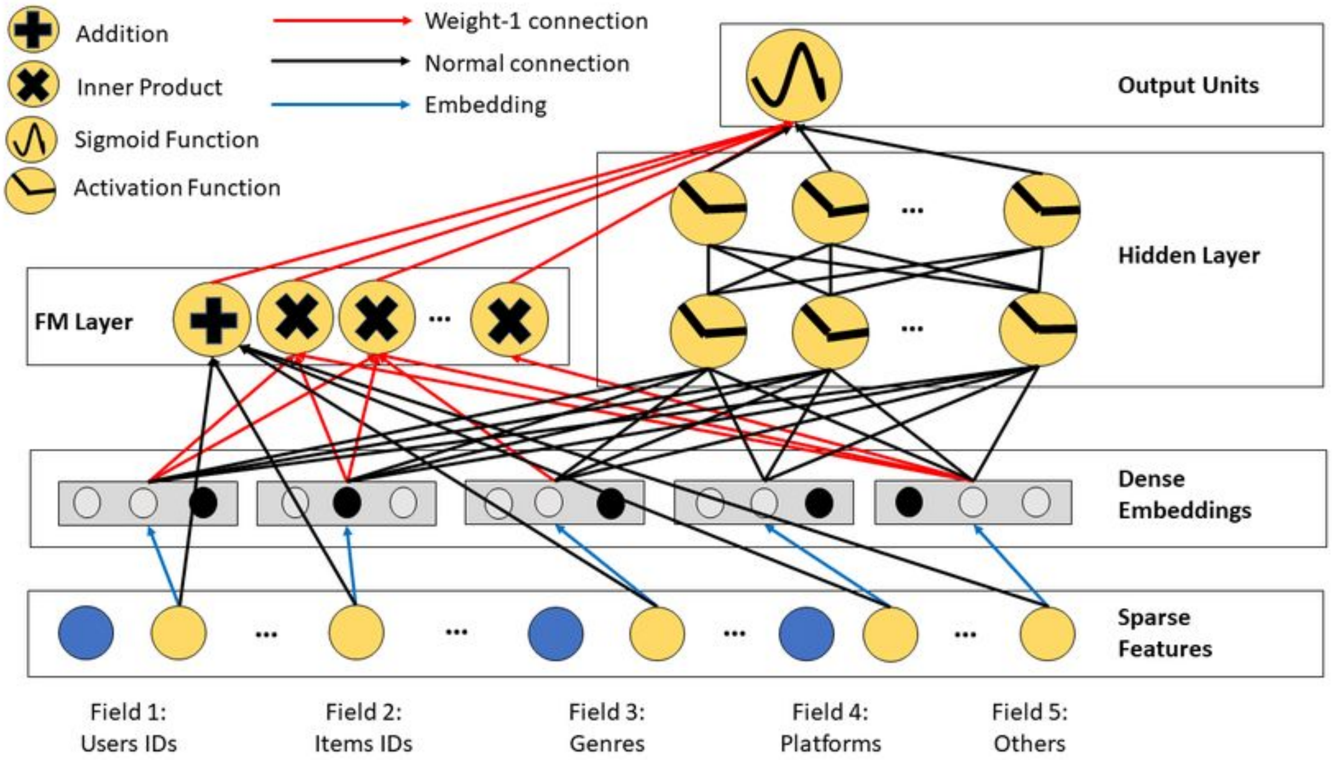
Order 2 Interaction: Do a Inner Product between all combinations of the input embedding vector

Factorization Machine Component of DeepFM



Concatenate Together Output of Order 1 and Order 2 Interactions, then Pass to Output Unit

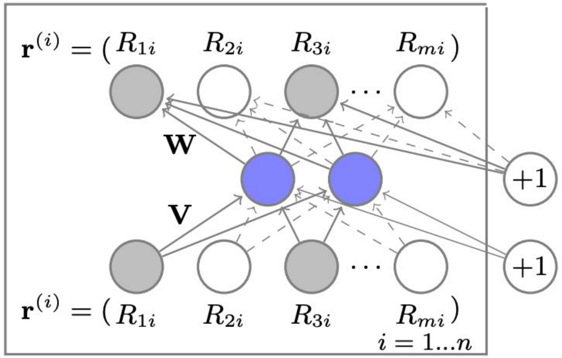
Both Outputs from Deep and Wide Components are Fed to Output Units



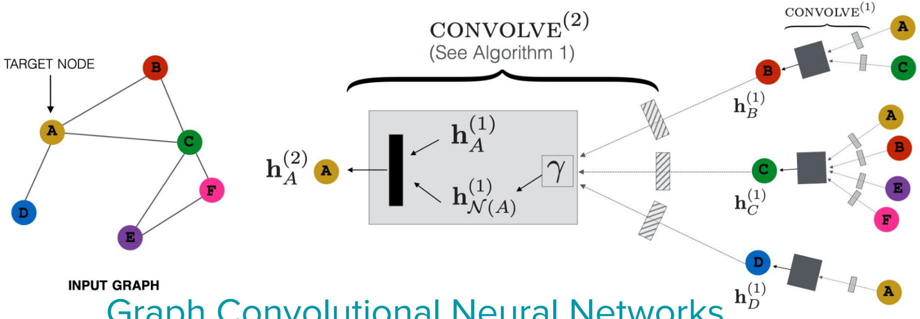
Summary of Modern, Deep Learning-Based Recommenders

- Deep Learning Methods can Incorporate a **whole variety of inputs flexibly**
- DL methods tend to learn **high-order** interactions, but having some sort of a **“wide”** component to learn **low-order** interactions is **just as important**
  - Wide & Deep: Manually engineered “cross-product” features
  - DeepFM: Use Factorization Machine to model interactions

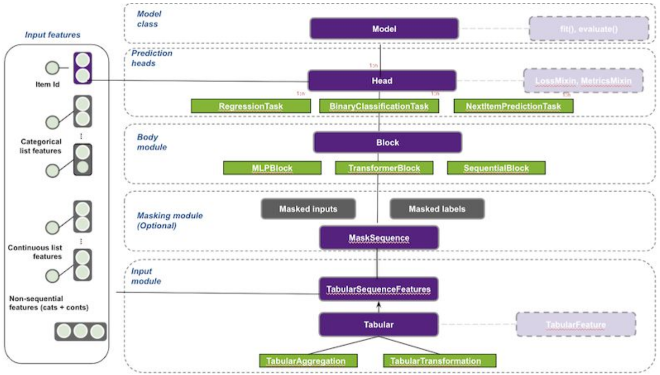
And of course, there are a whole variety of deep learning approaches!



[AutoRec: Autoencoders Meet Collaborative Filtering]



Graph Convolutional Neural Networks for Web-Scale Recommender Systems



[Transformers4Rec: Bridging the Gap between NLP and Sequential / Session-Based Recommendation]

Practical Aspects of Building Recommender Systems

Type of models we are dealing with is different

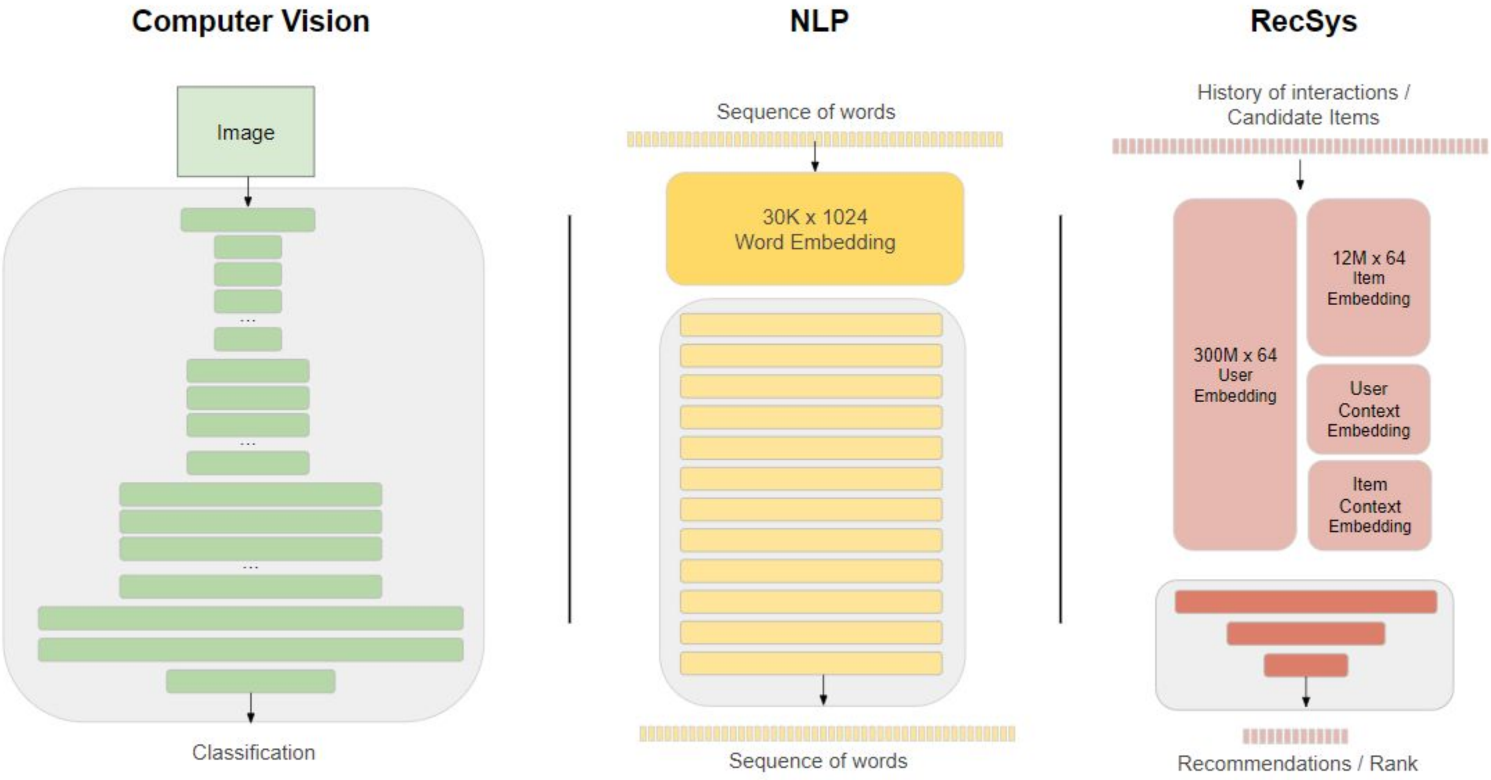


Image from Even Oldridge's Amazing Blog Post "Why isn't your recommender system training faster on GPU?" [here](#)

Problems Specific to Recommender Systems

- **Model doesn't fit on a single GPU** because embedding tables are so huge
- Unlike CV/NLP that have simple input data, recommender system **inputs must be fetched from databases**
  - Fetching a wide variety of features in real time is non-trivial
- Training data must be collected carefully
  - Implicit data is hard to work with
  - **"Self-feedback loop"** of training data
    - Items recommended and clicked on will then become training data for the model again, which can be dangerous

Questions?

Acknowledgement

Some Slides Inspired by Stanford’s CS246 Class Slides here