Introduction to Recommender Systems

Dr. Ahmad El Sallab
AI Senior Expert

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
- Build simple recommender system
- Build content-based recommender system
- Collaborative filtering
  - Matrix Factorization
  - Entity Embedding
  - Vector Similarity measures
  - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
  - Scaling
  - Movie and User Embedding
  - Meta data integration
  - Deeper model
  - T-SNE visualization

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
  - Build simple recommender system
  - Build content-based recommender system
- Collaborative filtering
  - Matrix Factorization
  - Entity Embedding
  - Vector Similarity measures
  - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
  - Scaling
  - Movie and User Embedding
  - Meta data integration
  - Deeper model
  - T-SNE visualization

What are recommendations?

- Related item recommendations → **Similar items**
  - Depends on similar items of the current searched items
  - No track of the user history

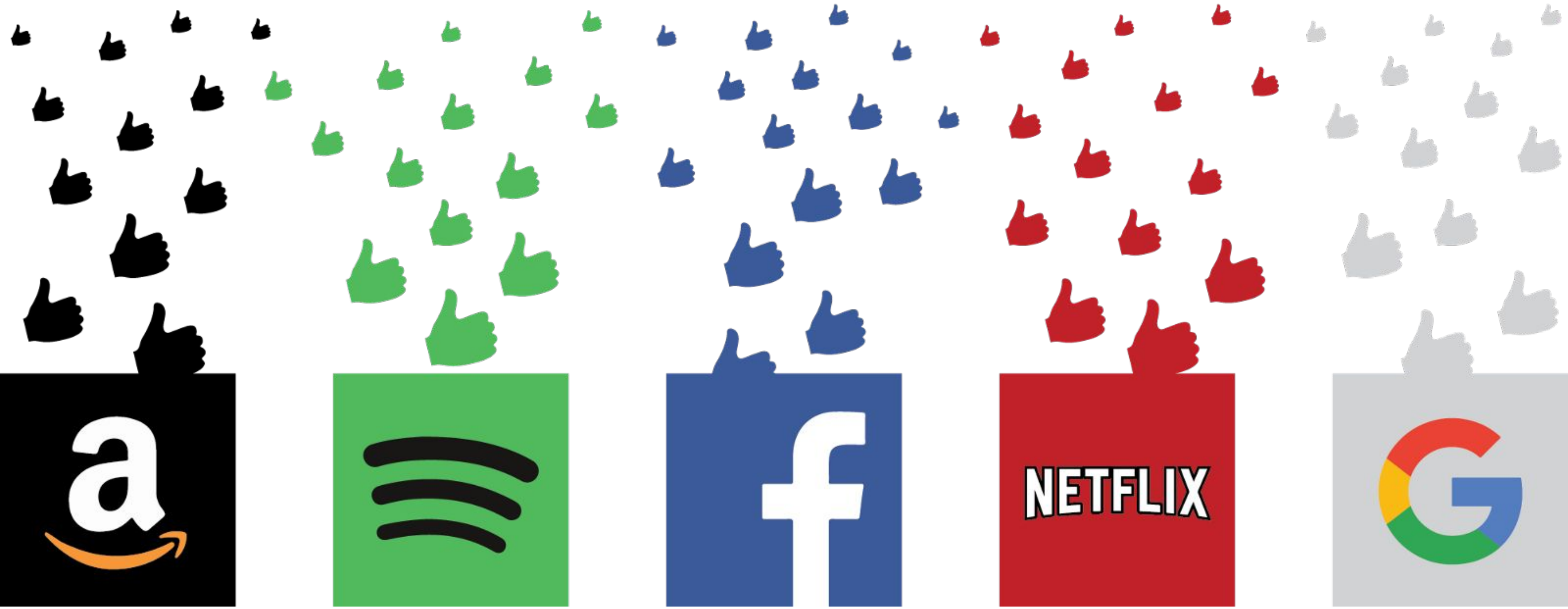
- Home page recommendations
  - **Personalized**
  - Depends on the user history + Similar items

Applications

NETFLIX:

Most famous and early adopters

Personalized data everywhere!



Applications

Ads
Online shopping
Friends suggestions (People you may know)
Content suggestion:

- YouTube
- Soundcloud
- Spotify
- Education courses

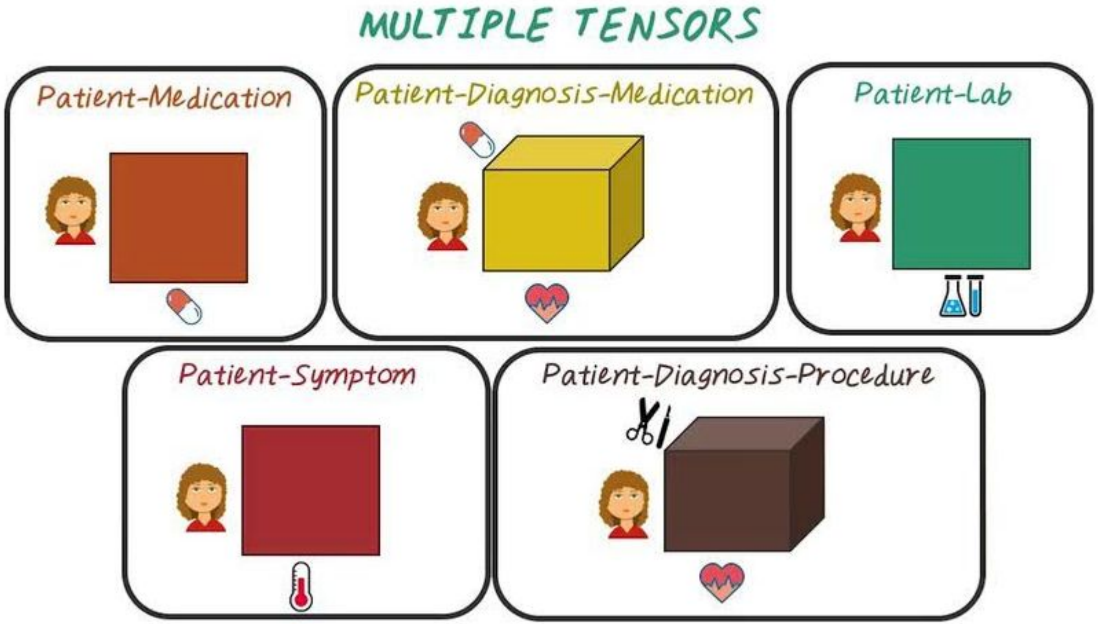
Elections!

Other applications outside the “Media” and Entertainment industry

Personalized medicine

Medicine description:

Which medicine worked with similar patient condition?

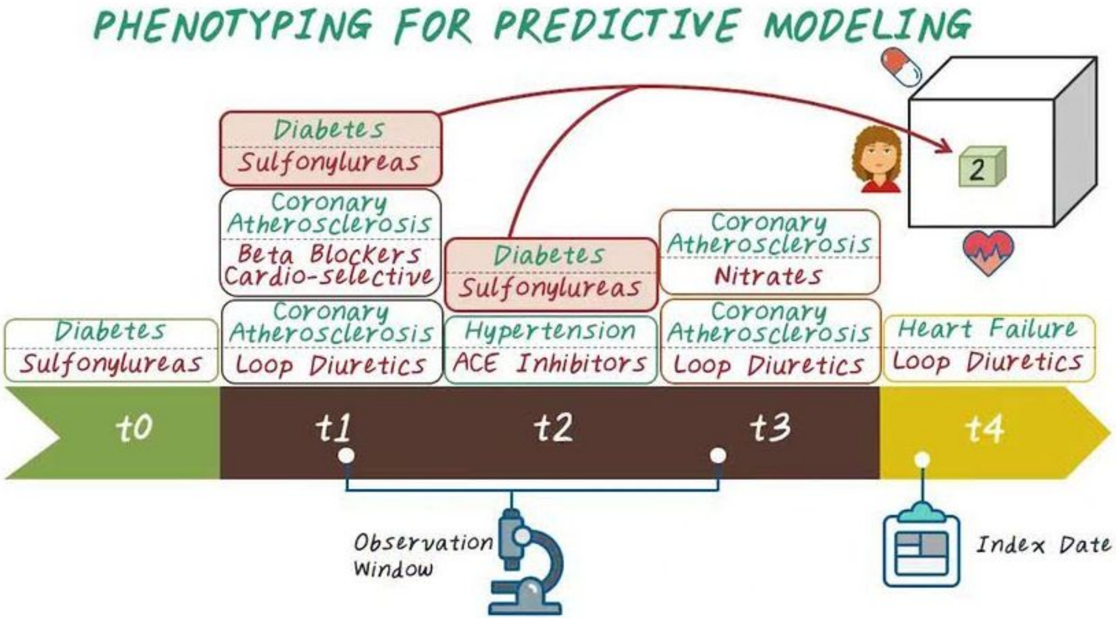


Other applications outside the "Media" and Entertainment industry

Personalized medicine

Diagnosis:

Which disease phenotype happened with similar patient condition?

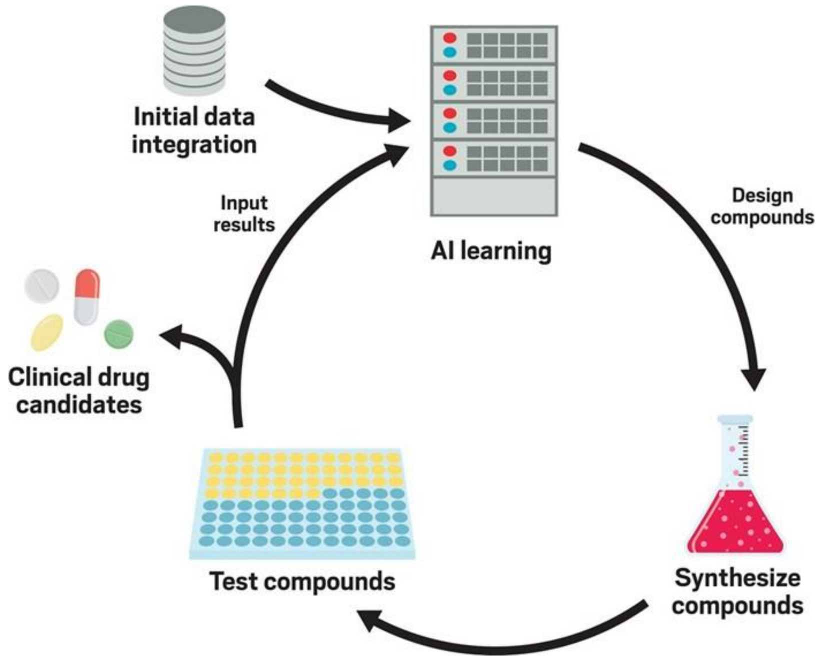


Other applications outside the “Media” and Entertainment industry

Drug Discovery:

Drug discovery is a long experimental process

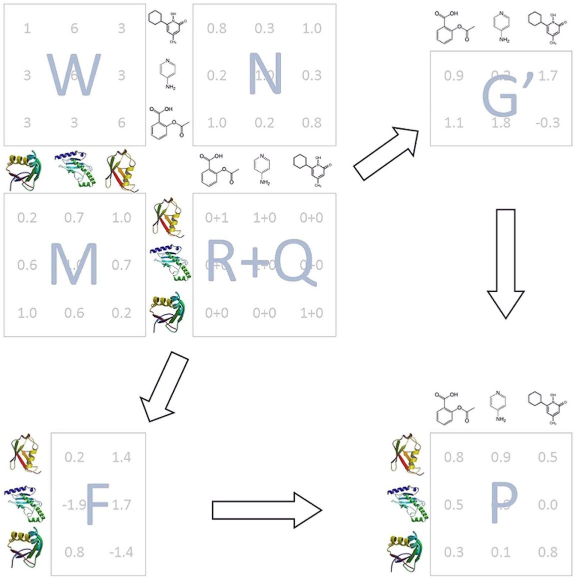
Game: reduce the space of possible drug compounds!

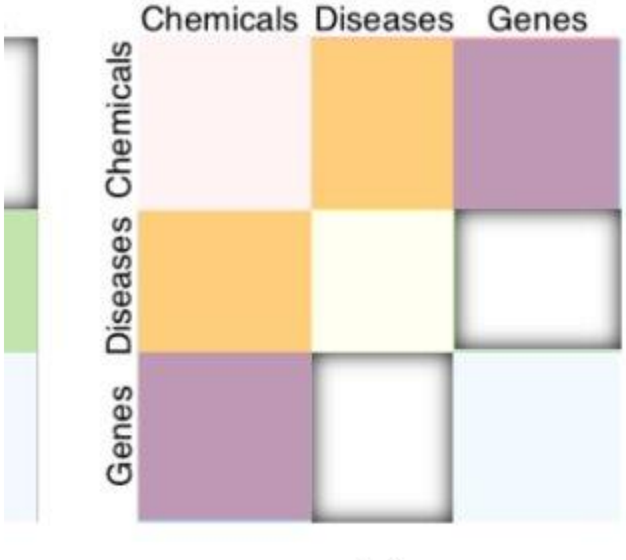


Other applications outside the “Media” and Entertainment industry

Drug Discovery:

Which compounds worked with similar disease?





(c)

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
- Build simple recommender system
- Build content-based recommender system
- Collaborative filtering
  - Matrix Factorization
  - Entity Embedding
  - Vector Similarity measures
  - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
  - Scaling
  - Movie and User Embedding
  - Meta data integration
  - Deeper model
  - T-SNE visualization

Structure of a Recommendation Engine

Candidate Generation

- Search DB

Scoring

- How likely it matches?
    - User history
    - Similar items

Re-ranking

- Filter out results

Candidate generation approaches

- Absolute ⇒ **Simple**
  - Recommend an item based on its **own** popularity, regardless of the user
- Experience centric
  - **Content based**
    - Based on **Meta data.**
    - Based on **history** of the user preferences.
    - Not taking advantage of other users similarity
  - **Collaborative**
    - takes advantages of similar users history,
    - in addition to the same user past preferences, to recommend new items

Candidate generation approaches

Type	Definition	Example
content-based filtering	Uses *similarity between items* to recommend items similar to what the user likes.	If user A watches two cute cat videos, then the system can recommend cute animal videos to that user.
collaborative filtering	Uses *similarities between queries and items simultaneously* to provide recommendations.	If user A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A (even if user A hasn't seen any videos similar to video 1).

Can take advantage of other users history even if this history is not of the current user! → By user similarity

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
- Build simple recommender system
- Build content-based recommender system
- Collaborative filtering
    - Matrix Factorization
    - Entity Embedding
    - Vector Similarity measures
    - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
    - Scaling
    - Movie and User Embedding
    - Meta data integration
    - Deeper model
    - T-SNE visualization

Simple

We will use IMBD

- **Decide on the metric or score to rate movies on. → Votes**
- Calculate the score for every movie.
- Sort the movies based on the score and output the top results.

Weighted Rating (WR) = $(\frac{v}{v+m} . R) + (\frac{m}{v+m} . C)$

where,

- v is the number of votes for the movie;
- m is the minimum votes required to be listed in the chart;
- R is the average rating of the movie; And
- C is the mean vote across the whole report

Simple

We will use IMBD

- Decide on the metric or score to rate movies on.
- **Calculate the score for every movie.**
- **Sort the movies based on the score and output the top results.**

```
#Print the top 15 movies
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(15)
```

	title	vote_count	vote_average	score
314	The Shawshank Redemption	8358.0	8.5	8.445869
834	The Godfather	6024.0	8.5	8.425439
10309	Dilwale Dulhania Le Jayenge	661.0	9.1	8.421453
12481	The Dark Knight	12269.0	8.3	8.265477
2843	Fight Club	9678.0	8.3	8.256385
292	Pulp Fiction	8670.0	8.3	8.251406
522	Schindler's List	4436.0	8.3	8.206639
23673	Whiplash	4376.0	8.3	8.205404
5481	Spirited Away	3968.0	8.3	8.196055
2211	Life Is Beautiful	3643.0	8.3	8.187171
1178	The Godfather: Part II	3418.0	8.3	8.180076
1152	One Flew Over the Cuckoo's Nest	3001.0	8.3	8.164256
351	Forrest Gump	8147.0	8.2	8.150272
1154	The Empire Strikes Back	5998.0	8.2	8.132919
1176	Psycho	2405.0	8.3	8.132715

Let’s code!

https://colab.research.google.com/drive/1gKmqbo9Wr7Np4LI0S9cMiOpPZ_ZvOx6t#scrollTo=AikyGCXpismw

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
- Build simple recommender system
- Build content-based recommender system
- Collaborative filtering
    - Matrix Factorization
    - Entity Embedding
    - Vector Similarity measures
    - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
    - Scaling
    - Movie and User Embedding
    - Meta data integration
    - Deeper model
    - T-SNE visualization

Content based

Recommend Apps to users on Google Play

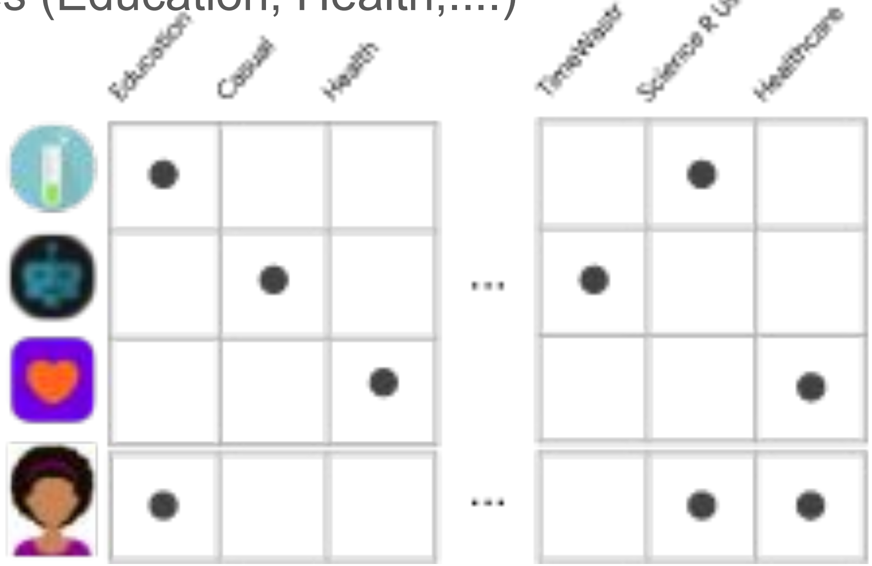
Simplest form → Decide on certain attributes (Education, Health,....)

For each App→ 1/0 if it has this attribute

For each user → 1/0 if he likes this attribute

Measure similarity of the user to all

Return top k that matches



Meta data

Suppose a user clicks a certain movie→ **Can we recommend similar movies?**

Similar based on what? → Meta data

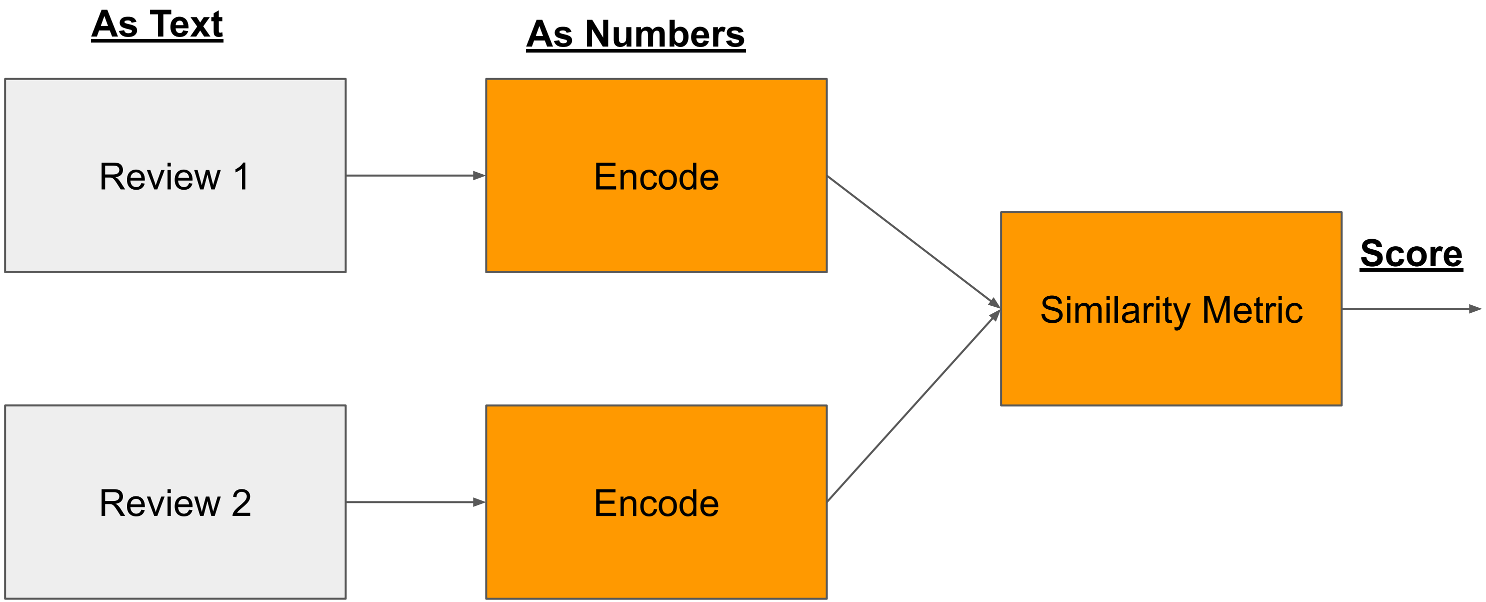
Here we assume the user vector = the clicked movie Meta data

```
#Print plot overviews of the first 5 movies.
metadata['overview'].head()
```

```
0    Led by Woody, Andy's toys live happily in his ...
1    When siblings Judy and Peter discover an encha...
2    A family wedding reignites the ancient feud be...
3    Cheated on, mistreated and stepped on, the wom...
4    Just when George Banks has recovered from his ...
Name: overview, dtype: object
```

How to deal with text similarity?

How similar are two reviews?



Then what?

Score table (if needed)

	Movie 1	Movie 2	Movie 3
Movie 1		0.9	0.1
Movie 2			
Movie 3			

Most similar to Movie 1 is Movie 2

How to encode text data?

First build a vocabulary for your data!

Many ways to encode a review

- BoW: Each sentence will have |V| numbers
  - **Binary:** Put 1/0 if the word is present/absent in a review
  - **Count:** Put the number of times the word is mentioned in a review
  - **Freq:** Normalized counts by total words counts
  - **TF-IDF:** Normalize by the total mentions in all reviews (frequent words are not important) = TF(in rev) x IDF (in ALL revs)
- Sequence (Advanced)

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

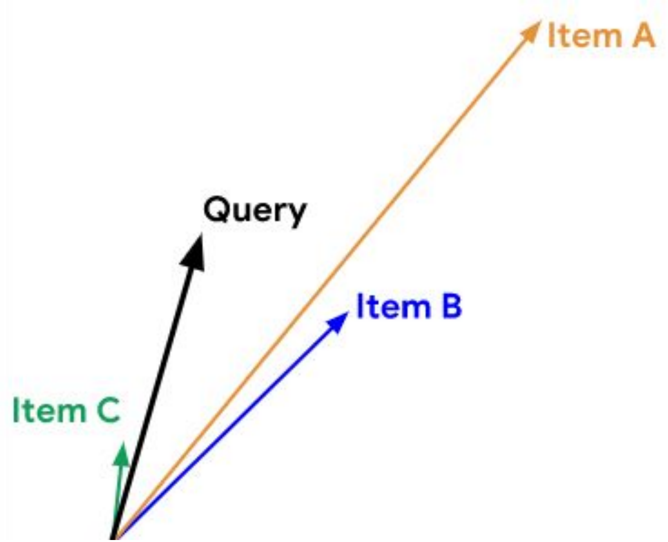
How to measure similarities

Now we represent each review with a vector |V|

How we know 2 vectors are similar?

- Dot $s(q, x) = \langle q, x \rangle = \sum_{i=1}^{d} q_i x_i.$
- Cosine similarity $s(q, x) = \|x\| \|q\| \cos(q, x)$
- Euclidean Distance $s(q, x) = \|q - x\| = \left[\sum_{i=1}^{d} (q_i - x_i)^2\right]^{\frac{1}{2}}.$

All are views of the same equation



Which one to use?

Dot and Euclidean are almost the same. Euclidean is rarely used.

The dot product similarity is sensitive to the norm of the embedding.

Items that appear very frequently in the training set (for example, popular YouTube videos) tend to have embeddings with large norms.

If capturing popularity information is desirable, then you should prefer dot product.

--> Else cosine, or encode as TF-TDF

Putting it all together

```
print(get_recommendations('The Dark Knight Rises'))
```

```
12481                                      The Dark Knight
150                                         Batman Forever
1328                                        Batman Returns
15511                           Batman: Under the Red Hood
585                                                 Batman
21194    Batman Unmasked: The Psychology of the Dark Kn...
9230                     Batman Beyond: Return of the Joker
18035                                     Batman: Year One
19792              Batman: The Dark Knight Returns, Part 1
3095                          Batman: Mask of the Phantasm
Name: title, dtype: object
```

```
get_recommendations('The Godfather')
```

```
1178                 The Godfather: Part II
44030    The Godfather Trilogy: 1972-1990
1914                The Godfather: Part III
23126                           Blood Ties
11297                     Household Saints
34717                    Start Liquidation
10821                             Election
38030              A Mother Should Be Loved
17729                    Short Sharp Shock
26293                   Beck 28 - Familjen
Name: title, dtype: object
```

```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return metadata['title'].iloc[movie_indices]
```

What if Meta Data is not Text?

- Flags
- Scalars
- Images
- ...etc

```
# Print the first two movies of your newly merged metadata
metadata.head(2)
```

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_language	original_title	overview	popularity	poster_
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	en	Toy Story	Led by Woody, Andy's toys live happily in his ...	21.9469	/rhIRbceoE9lR4veEXuwCC2wARt
1	False	NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...	NaN	8844	tt0113497	en	Jumanji	When siblings Judy and Peter discover an encha...	17.0155	/vzmL6fP7aPKNKPRTFnZmiUfciy

What if Meta Data is not Text?

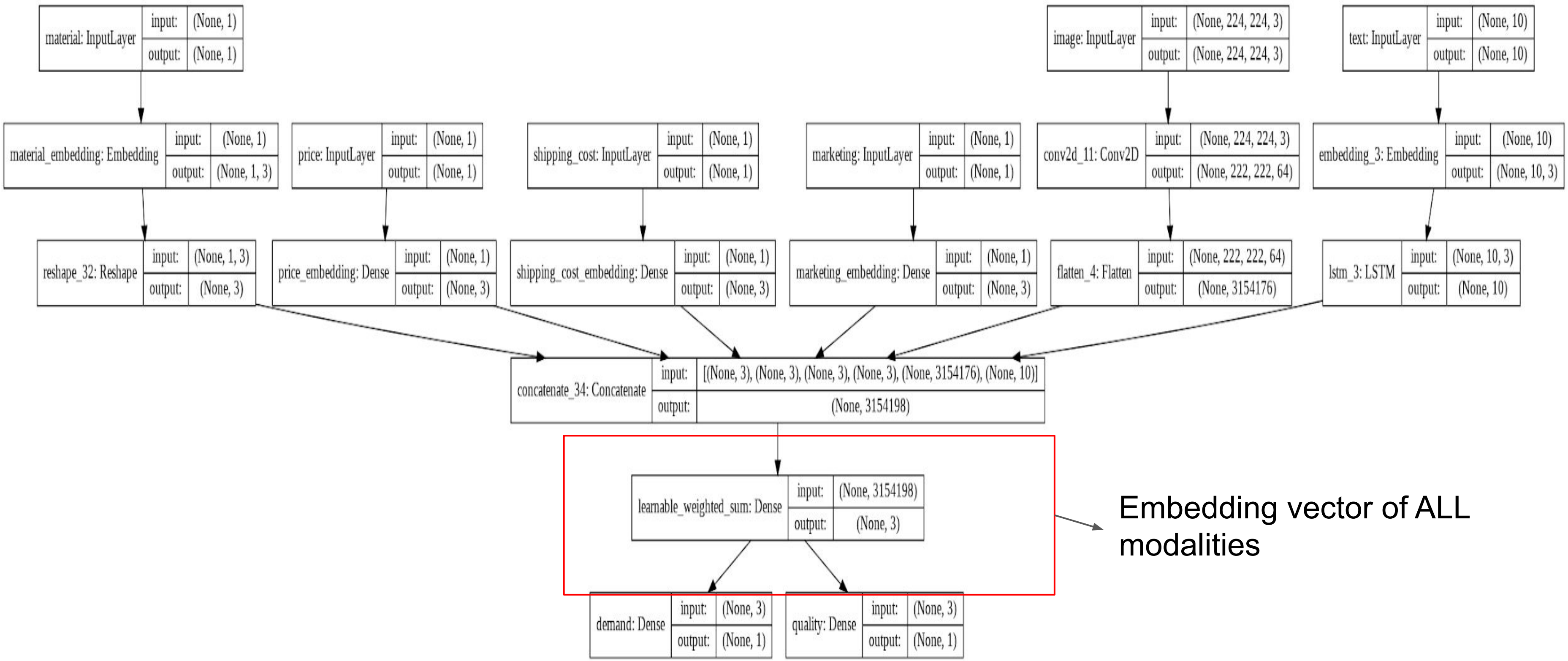
- Flags
- Scalars
- Images
- ...etc

Structured DL!

Tabular/variables	Dense/Embedding
Image	Conv2D
Text	LSTM/GRU
Video	VideoCNN, VideoDNN, ConvLSTM, CNN1D, CNN-LSTM

+ TimeSeries! → (n_samples, n_time_steps, n_features)

What if Meta Data is not Text?



Workaround → Encode all as "soup" of Text

Then apply TF-IDF + cosine similarity as before:

Simple

Doesn't work with numericals, images and videos

```
# Print the new features of the first 3 films
metadata[['title', 'cast', 'director', 'keywords', 'genres']].head(10)
```

	title	cast	director	keywords	genres
0	Toy Story	[tomhanks, timallen, donrickles]	johnlasseter	[jealousy, toy, boy]	[animation, comedy, family]
1	Jumanji	[robinwilliams, jonathanhyde, kirstendunst]	joejohnston	[boardgame, disappearance, basedonchildren'sbook]	[adventure, fantasy, family]
2	Grumpier Old Men	[waltermatthau, jacklemmon, ann-margret]	howarddeutch	[fishing, bestfriend, duringcreditsstinger]	[romance, comedy]
3	Waiting to Exhale	[whitneyhouston, angelabassett, lorettadevine]	forestwhitaker	[basedonnovel, interracialrelationship, single...	[comedy, drama, romance]
4	Father of the Bride Part II	[stevemartin, dianekeaton, martinshort]	charlesshyer	[baby, midlifecrisis, confidence]	[comedy]
5	Heat	[alpacino, robertdeniro, valkilmer]	michaelmann	[robbery, detective, bank]	[action, crime, drama]
6	Sabrina	[harrisonford, juliaormond, gregkinnear]	sydneypollack	[paris, brotherbrotherrelationship, chauffeur]	[comedy, romance]
7	Tom and Huck	[jonathantaylorthomas, bradrenfro, rachaelleig...	peterhewitt	[]	[action, adventure, drama]
8	Sudden Death	[jean-claudevandamme, powersboothe, dorianhare...	peterhyams	[terrorist, hostage, explosive]	[action, adventure, thriller]
9	GoldenEye	[piercebrosnan, seanbean, izabellascorupco]	martincampbell	[cuba, falselyaccused, secretidentity]	[adventure, action, thriller]

Pros and cons of Content based

No data needed about other users → Easy scaling to whatever number of users (less complexity)

No data needed about other users! → Other **similar** users experience is not used!

Features are hand-engineered (meta data) → Can be overcome with structure DL→ Learn the features→ But again, which Meta data to use?

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
- Build simple recommender system
- Build content-based recommender system
- Collaborative filtering
  - Matrix Factorization
  - Entity Embedding
  - Vector Similarity measures
  - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
  - Scaling
  - Movie and User Embedding
  - Meta data integration
  - Deeper model
  - T-SNE visualization

Basic idea

Similar users favor similar items!

	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
User 1	✓		✓	✓	
User 2		✓			✓
User 3	✓	✓	✓		
User 4				✓	✓

Why?

Latent Factors!

Analysis shows that the younger users category

Prefer similar movies

In other words → Age groups prefer similar movie groups

<u>Factor = Age</u>

Embeddings

A mapping from discrete (categorical) space
⇒ **Vector space**

Word ⇒ Vectors → Word Embedding

Sentence ⇒ Vectors → Sentence Embedding

User ⇒ Vectors → User Embedding

Movies/Items ⇒ Vectors → Item Embedding

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

How to calculate the Embedding? Big topic!
Core of the Recommendation Engine

Why?

Latent Factors!

Factor = Age

Give each user a factor index according to age = [-1,1].

Do the same for movies.

1D Embedding!

Why?

Latent Factors!

Factor = Age

Give each user a factor index according to age = [-1,1] → **User Embedding Vector (1D)**

Do the same for movies.→ **Movie Embedding Vector (1D)**

1D Embedding!

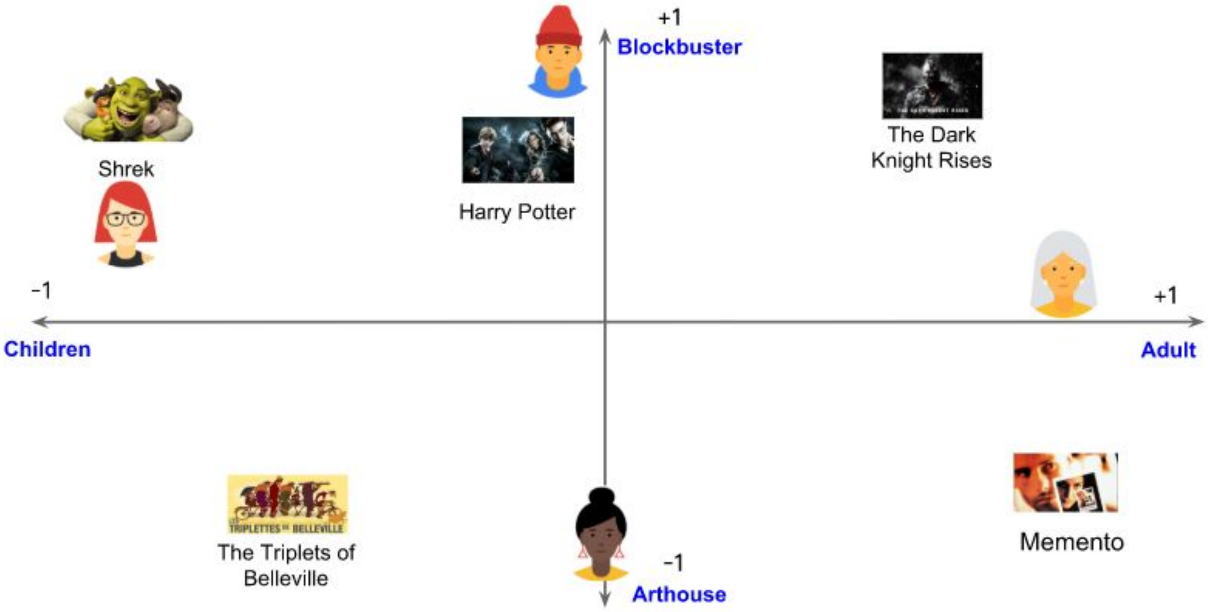
What of we have more Latent Factors?

Each user and movie are attributed to two factors

[Blockbuster, Age]

Each of the 2 attributes range from [-1,1]

So each user or movie is represented in 2D space with a 2D vector say [0.6, 0.2]



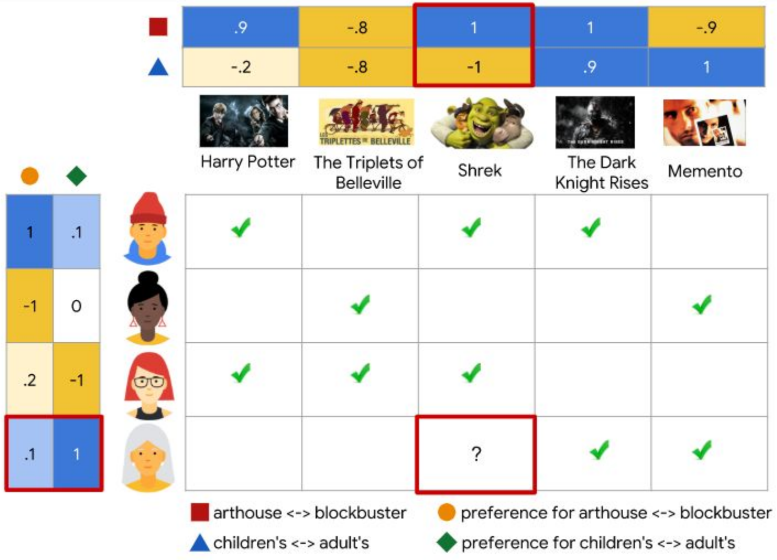
2D Embedding!

Who decides on the factors?

In the example, we decided: Blockbuster, Age

Can we learn them?

From what? → Data



Matrix Factorization

Loss = min (A - A_pred) → Find U and V

A_pred = U.V^T (outer product)

$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2.$$

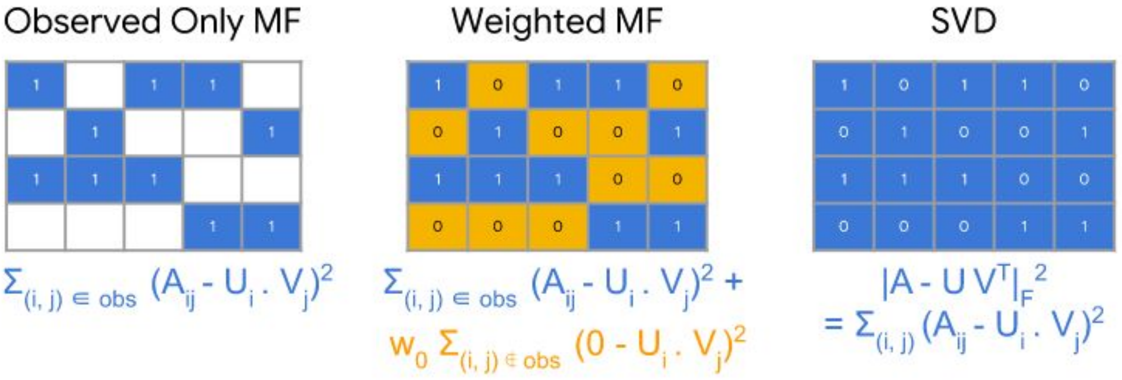
SVD

The GT A can be only the "observed" or rated movies by a user

This will be very sparse

SVD is a linear algebra technique to **<u>factorize</u>** a given matrix to it's components.

Given A (GT) → factorize to get U and V → Slow and might not work due to large A and sparsity (many 0's)



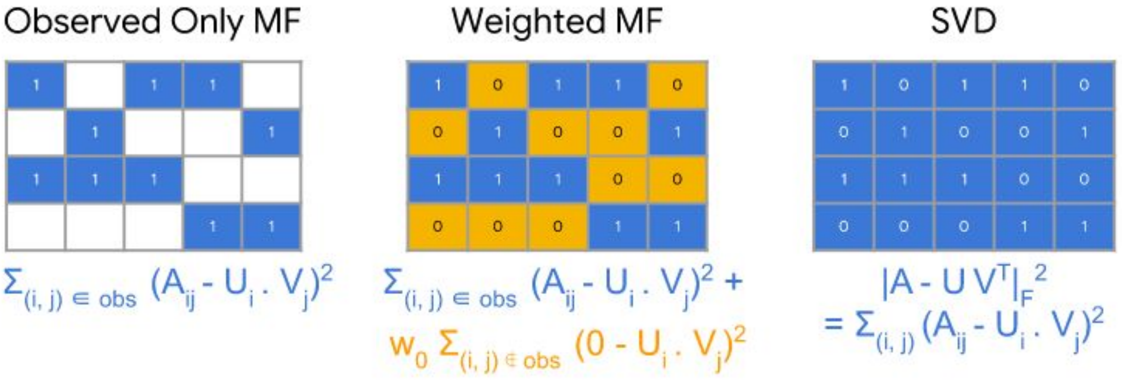
$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2.$$

SVD

The GT A can be only the "observed" or rated movies by a user

This will be very sparse

Weighted MF → Unobserved = 0's



Weighted Matrix Factorization decomposes the objective into the following two sums:

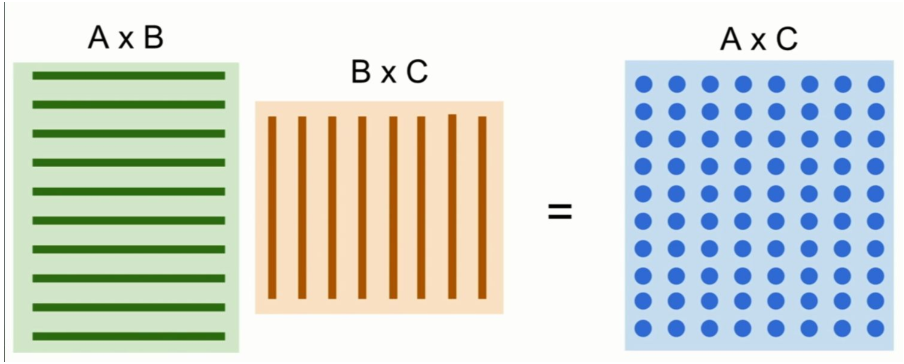
$$\min_{U \in \mathbb{R}^{m \times d},\ V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (A_{ij} - \langle U_i, V_j \rangle)^2 + w_0 \sum_{(i,j) \notin \text{obs}} (\langle U_i, V_j \rangle)^2.$$

- A sum over observed entries.
- A sum over unobserved entries (treated as zeroes).

Cost of SVD

$$\min_{U\in\mathbb{R}^{m\times d},\ V\in\mathbb{R}^{n\times d}} \|A - UV^T\|_F^2.$$

n=vocab_sz
m=n_latent=emb_sz
U⇒ nxm
V⇒ nxm



MATRIX MULTIPLICATION

The product of an m x n matrix A by an n x k matrix B is an m x k matrix C.

The pseudo code is :

```
procedure matrix_multiplication(A, B, C) is
    for i = 1 to m do
        for j = 1 to k do
            cij = 0;
            for s = 1 to n do
                cij = cij + (ais * bsj);
            end for;
        end for;
    end for;
```

This procedure takes $m*k*n$ steps.

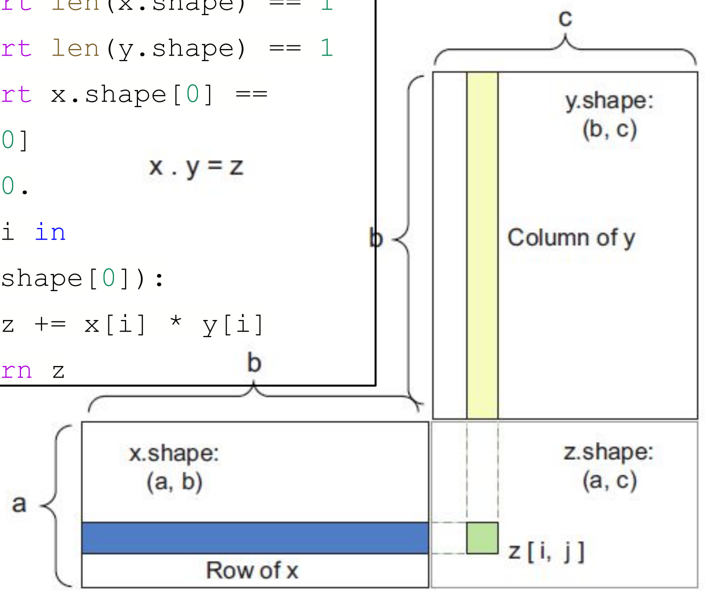
A ⇒ nxn elements
Each comes from a dot operation ⇒ m
Overall cost is O(mn^2)

Cost of SVD

$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \|A - UV^T\|_F^2.$$

n=n_movies
k=n_users
m=n_latent=emb_sz
U⇒ kxm
V⇒ nxm

```
def naive_vector_dot(x, y):
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] ==
y.shape[0]
    z = 0.
    for i in
range(x.shape[0]):
        z += x[i] * y[i]
    return z
```



Scales quadratic with nxk!

MATRIX MULTIPLICATION

The product of an m x n matrix A by an n x k matrix B is an m x k matrix C.

The pseudo code is :

procedure matrix_multiplication(A, B, C) is
    for $i = 1$ to m do
        for $j = 1$ to k do
            $c_{ij} = 0$;
            for $s = 1$ to n do
                $c_{ij} = c_{ij} + (a_{is} * b_{sj})$;
            end for;
        end for;
    end for;

This procedure takes $m*k*n$ steps.

A ⇒ nxn elements
Each comes from a dot operation ⇒ m
Overall cost is O(mnk)

Issues with Matrix Factorization

- Cost of matrix operations: Scales quadratic with vocab_sz (millions or words, or movies, or users)
- What happens when new movies or users appear?

Gradient based optimization

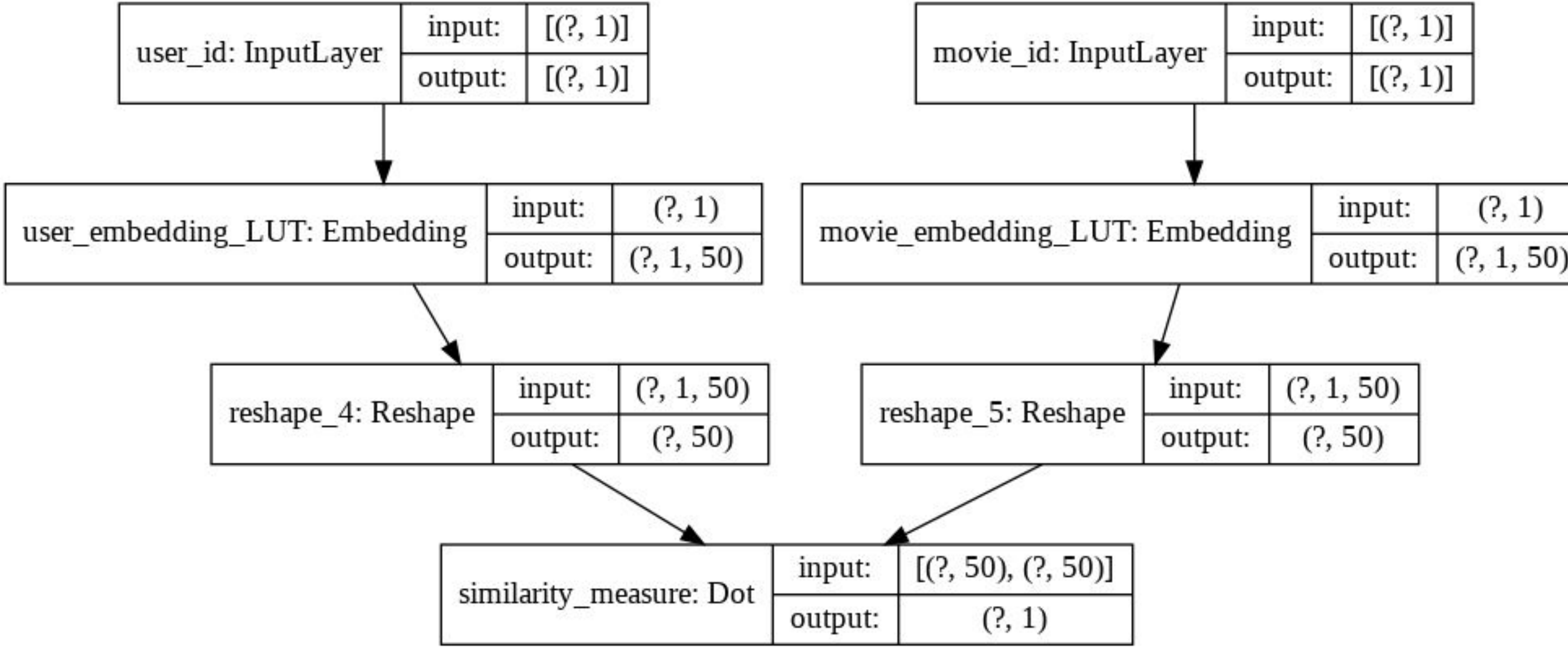
In the example, we decided: Blockbuster, Age

Can we learn them?

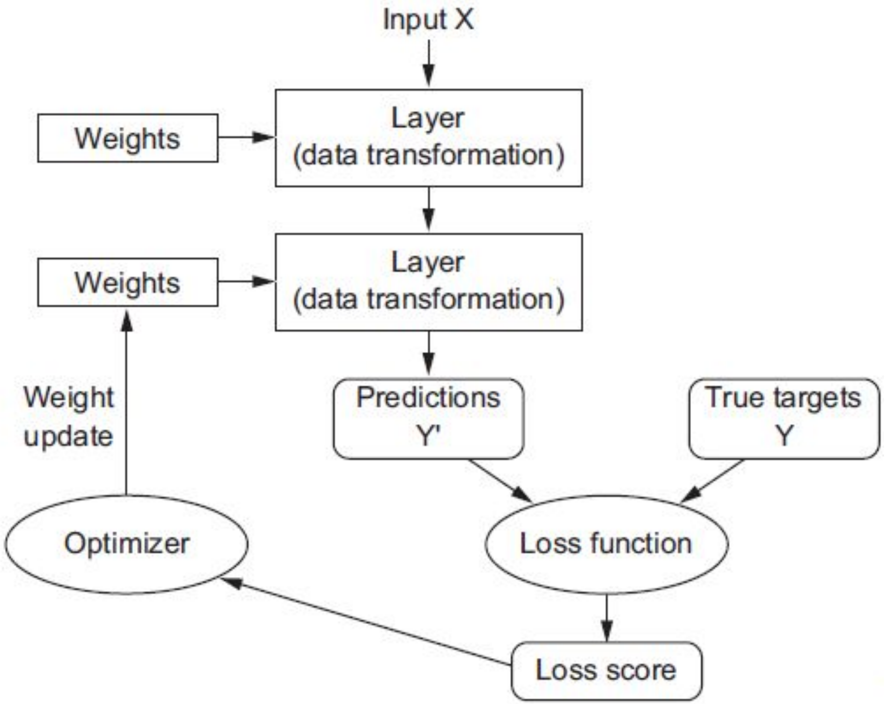
From what? → Data

- **For each [(user,movie)] pair we have a rating**
- **If we give each user/movie a random vector**
- **Dot them → Get a scalar**
- **Try to fit this scalar to the (user,movie)-->rating**

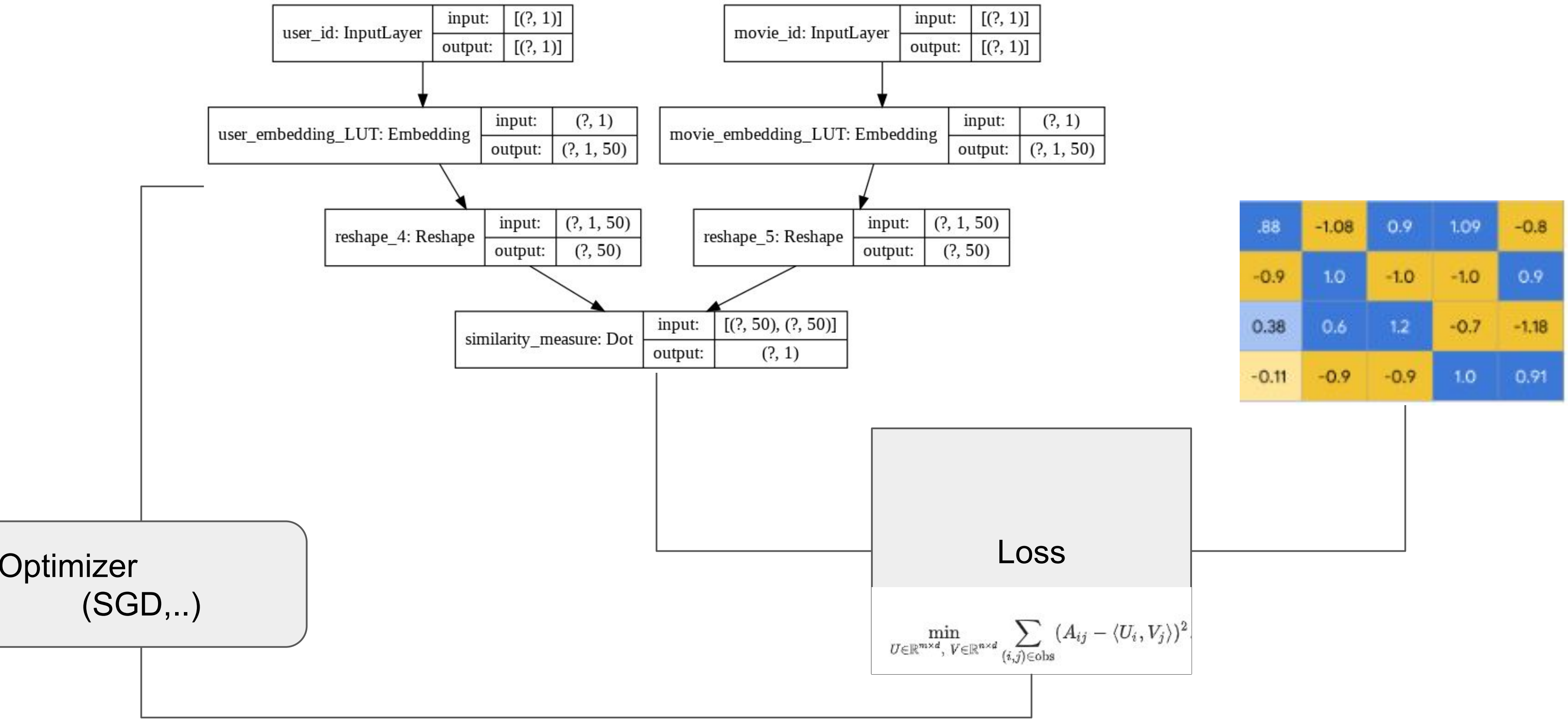
Gradient based optimization



Gradient based optimization



Gradient based optimization



Let’s code!

https://colab.research.google.com/drive/1GLofcB4RMPq8yQLyQcb_KA_mB5LRwYMa

```
# Number of latent factors
emb_sz = 50

# User embeddings
user = layers.Input(shape=(1,), name='user_id')
user_emb = layers.Embedding(n_users, emb_sz, embeddings_regularizer=regularizers.l2(1e-6), name='user_embedding_LUT')(user)
user_emb = layers.Reshape((emb_sz,))(user_emb)

# Movie embeddings
movie = layers.Input(shape=(1,), name='movie_id')
movie_emb = layers.Embedding(n_movies, emb_sz, embeddings_regularizer=regularizers.l2(1e-6), name='movie_embedding_LUT')(movie)
movie_emb = layers.Reshape((emb_sz,))(movie_emb)

# Dot product
rating = layers.Dot(axes=1, name='similarity_measure')([user_emb, movie_emb])


# Model
model = models.Model([user, movie], rating)

# Compile the model
model.compile(loss='mse',  metrics=metrics.RootMeanSquaredError(),
              optimizer=optimizers.Adam(lr=0.001))

# Show model summary
model.summary()
plot_model(model, show_shapes=True)
```

Results

Benchmark results show lowest RMSE of 0.89 on the 100K dataset as we are using.

We reach 0.85!

Only in few epochs

MovieLens (100K)

Algorithm	MAE			RMSE			Train Time (s)			Test Time (s)		
	MMLite	PREA	LibRec	MMLite	PREA	LibRec	MMLite	PREA	LibRec	MMLite	PREA	LibRec
GlobalAvg	0.945	0.949	0.945	1.126	1.128	1.126	00:00	00:00	00:00	00:00	00:00	00:00
UserAvg	0.835	0.838	0.835	1.041	1.043	1.042	00:00	00:00	00:00	00:00	00:00	00:00
ItemAvg	0.817	0.823	0.817	1.024	1.030	1.025	00:00	00:00	00:00	00:00	00:00	00:00
PD	N/A	N/A	0.794	N/A	N/A	1.094	N/A	N/A	00:00	N/A	N/A	14:26
	sigma=2.5											
UserKNN	0.721	0.732	0.737	0.921	0.937	0.944	00:02	00:00	00:05	01:38	(00:20)x5	00:03
	neighbors=60, shrinkage=25, similarity=pcc; MMLite: reg_u=12, reg_i=1											
ItemKNN	0.703	0.716	0.723	0.899	0.914	0.924	00:03	00:00	00:05	02:00	(01:47)x5	00:06
	neighbors=40, shrinkage=2500, similarity=pcc; MMLite: reg_u=12, reg_i=1											

Agenda

- What is a Recommender System?
- Why we need it? Applications
- Types of Recommender Systems
- Build simple recommender system
- Build content-based recommender system
- Collaborative filtering
  - Matrix Factorization
  - Entity Embedding
  - Vector Similarity measures
  - Build State-of-The Art Collaborative filter using DL
- Analysis and Improvements
  - Scaling
  - Movie and User Embedding
  - Meta data integration
  - Deeper model
  - T-SNE visualization

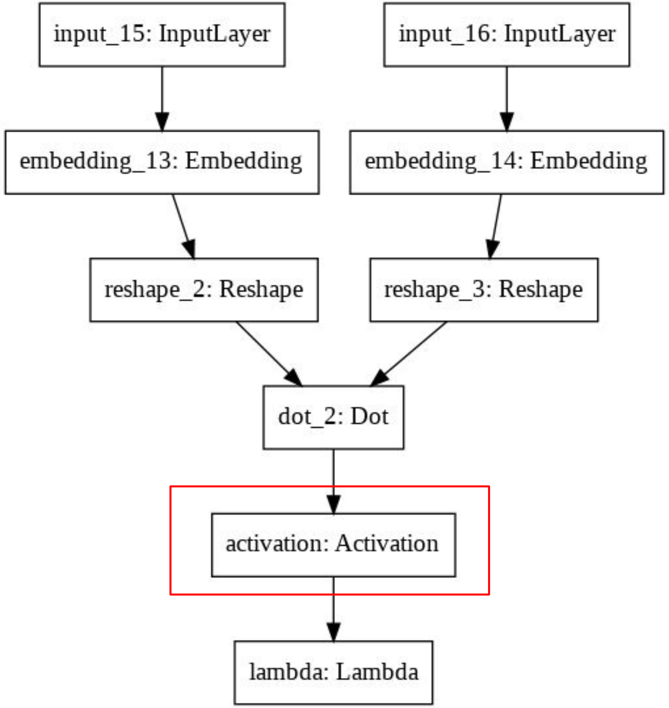
Scaling

To help the model to scale the output within the required range, we pass the output of the Dot product to a sigmoid, which ranges from [0,1], then we scale that up to the range of min_rating to max_rating.

```
# Number of latent factors
emb_sz = 50

# User embeddings
user = layers.Input(shape=(1,))
user_emb = layers.Embedding(n_users, emb_sz, embeddings_regularizer=regularizers.l2(1e-6))(user)
user_emb = layers.Reshape((emb_sz,))(user_emb)

# Movie embeddings
movie = layers.Input(shape=(1,))
movie_emb = layers.Embedding(n_movies, emb_sz, embeddings_regularizer=regularizers.l2(1e-6))(movie)
movie_emb = layers.Reshape((emb_sz,))(movie_emb)

# Dot product
rating = layers.Dot(axes=1)([user_emb, movie_emb])
rating = layers.Activation('sigmoid')(rating)
rating = layers.Lambda(lambda x:x*(max_rating - min_rating) + min_rating)(rating)

# Model
model = models.Model([user, movie], rating)
```

Scaling

To help the model to scale the output within the required range, we pass the output of the Dot product to a sigmoid, which ranges from [0,1], then we scale that up to the range of min_rating to max_rating.



User and movie bias

The first simple approach was based on recommending top movies, **regardless of the user(s) preferences. → Similar to item popularity Simple Recommender**

The latent factors are dependent on the dot product of users and movies ratings.

But there might be some prior movie or user factor, that does not depend on the relation between the user and his ratings.

Like for example, how much a **movie is popular** independent of the specific user rating. Say a user never clicks a certain category of movies (popular), how can we recommend to him?

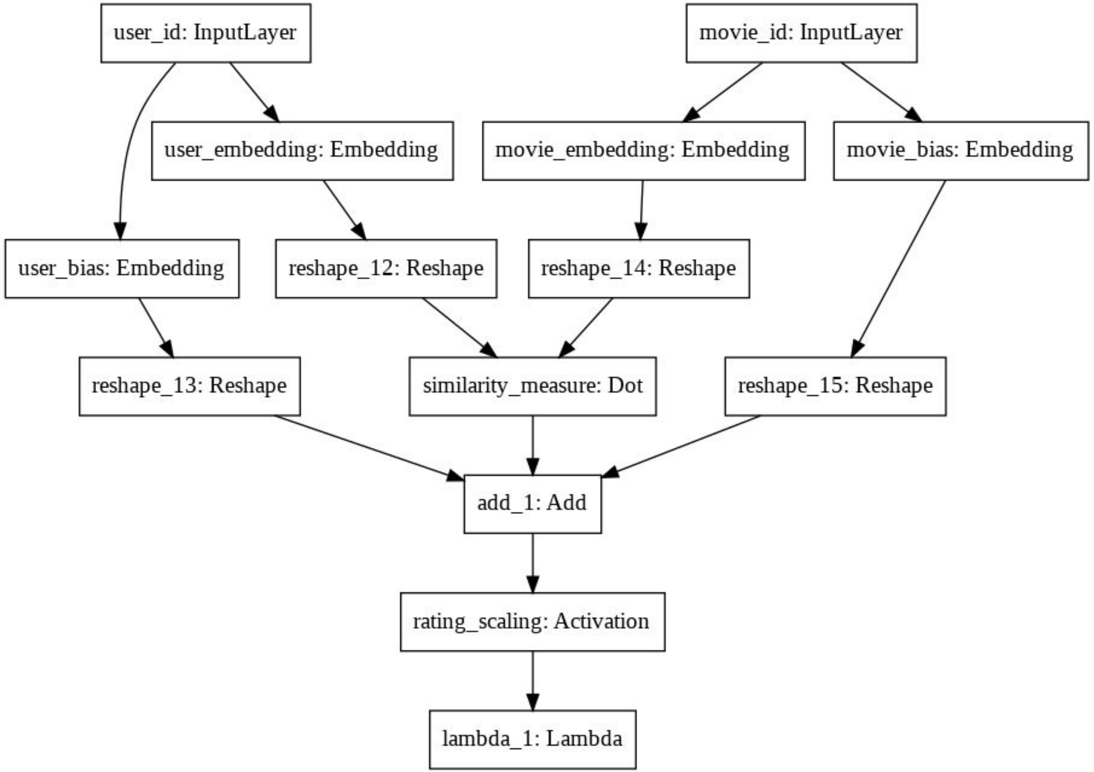
or how much a **user likes movies**, independent of his rating to a specific movie.

Here where the bias role comes in.

User and movie bias

```
# User embeddings
user = layers.Input(shape=(1),name='user_id')
user_emb = layers.Embedding(n_users, emb_sz, embeddings_regularizer=regularizers.l2(1e-6),name='user_embedding')(user)
user_emb = layers.Reshape((emb_sz,))(user_emb)

# User bias
user_bias = layers.Embedding(n_users, 1, embeddings_regularizer=regularizers.l2(1e-6),name='user_bias')(user)
user_bias = layers.Reshape((1,))(user_bias)

# Movie embeddings
movie = layers.Input(shape=(1,),name='movie_id')
movie_emb = layers.Embedding(n_movies, emb_sz, embeddings_regularizer=regularizers.l2(1e-6),name='movie_embedding')(movie)
movie_emb = layers.Reshape((emb_sz,))(movie_emb)

# Movie bias
movie_bias = layers.Embedding(n_movies, 1, embeddings_regularizer=regularizers.l2(1e-6),name='movie_bias')(movie)
movie_bias = layers.Reshape((1,))(movie_bias)

# Dot product
rating = layers.Dot(axes=1,name='similarity_measure')([user_emb, movie_emb])

# Add biases
rating = layers.Add()([rating, user_bias, movie_bias])
rating = layers.Activation('sigmoid',name='rating_scaling')(rating)
rating = layers.Lambda(lambda x:x*(max_rating - min_rating) + min_rating)(rating)

# Model
model = models.Model([user, movie], rating)
```

User and movie bias



In the same way we could add any other meta data

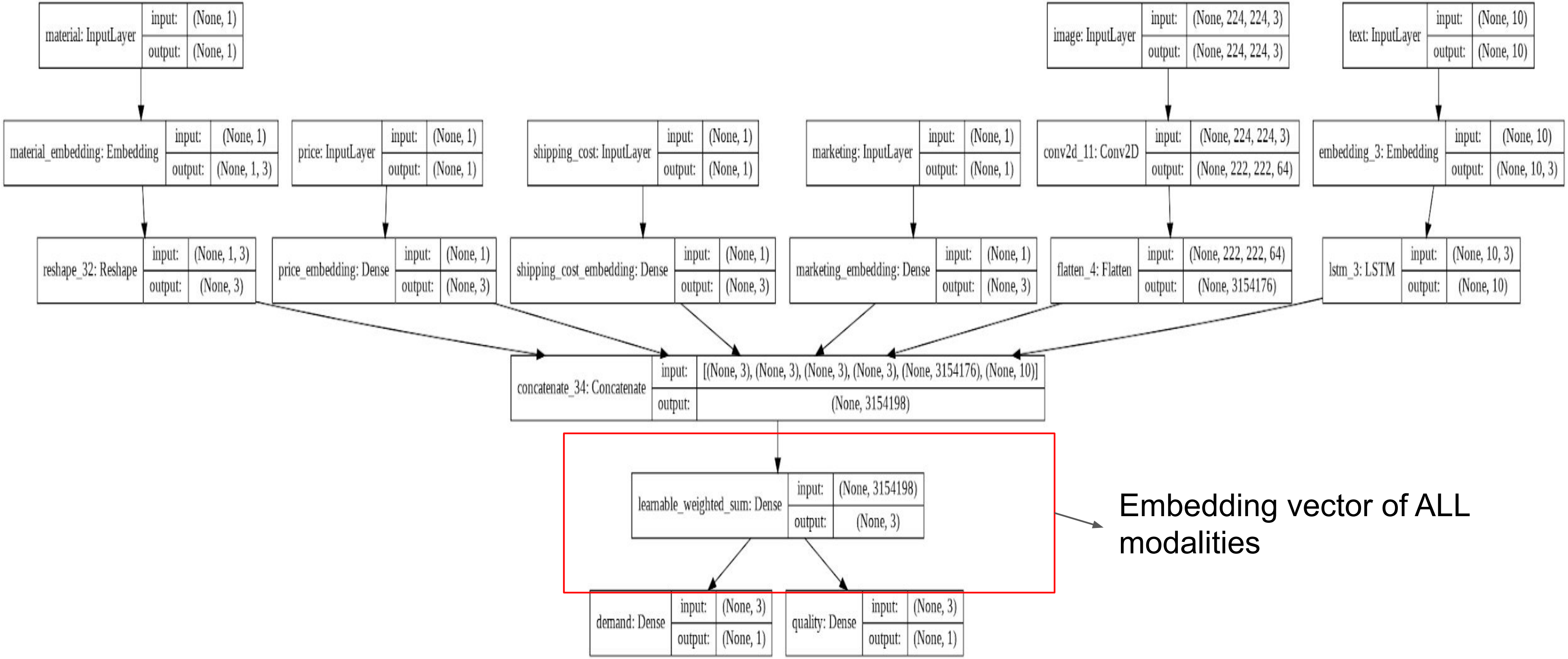
- Flags
- Scalars
- Images
- ...etc

Structured DL!

Tabular/variables	Dense/Embedding
Image	Conv2D
Text	LSTM/GRU
Video	VideoCNN, VideoDNN, ConvLSTM, CNN1D, CNN-LSTM

+ TimeSeries! → (n_samples, n_time_steps, n_features)

What if Meta Data is not Text?



Embedding vector of ALL modalities

Analysis of movie bias

How popular is a certain movie, independent of the user preference?

```
movie_bias_model = models.Model(movie, movie_bias)
movie_bias_model.summary()
```

```
mbs = movie_bias_model.predict(movies_in)
```

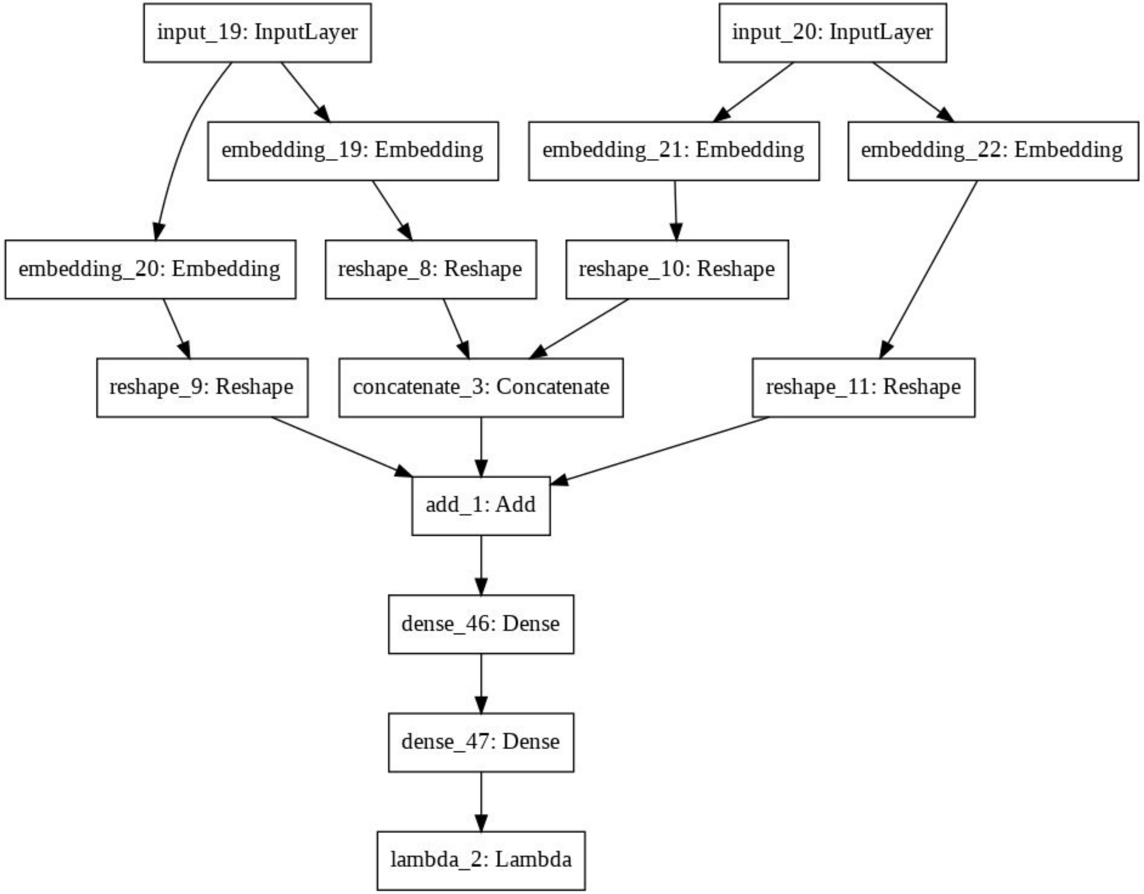
```
movie_bias_df.sort_values('bias', ascending=False)
```

	title	bias
1	Jumanji (1995)	0.560876
0	Toy Story (1995)	0.501703
198	Eat Drink Man Woman (Yin shi nan nu) (1994)	0.391028
6	Sabrina (1995)	0.363988
2008	Thirteenth Floor, The (1999)	0.356012
...	...	...
3915	Signs (2002)	-0.239535
6332	Flags of Our Fathers (2006)	-0.242444
1493	Bambi (1942)	-0.254248
6642	Bucket List, The (2007)	-0.258685
8453	Boyhood (2014)	-0.276169

2000 rows × 2 columns

Going Deeper

We could build multi Dense layers after the Dot



Use regularization to help model generalization

Embedding Regularizer → The embedding is the biggest layer → Source of model complexity → Overfitting

→ Penalize by regularization

```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_21 (InputLayer)           [(None, 1)]          0
__________________________________________________________________________________________________
input_22 (InputLayer)           [(None, 1)]          0
__________________________________________________________________________________________________
embedding_23 (Embedding)        (None, 1, 50)        30500       input_21[0][0]
__________________________________________________________________________________________________
embedding_25 (Embedding)        (None, 1, 50)        486200      input_22[0][0]
__________________________________________________________________________________________________
reshape_12 (Reshape)            (None, 50)           0           embedding_23[0][0]
__________________________________________________________________________________________________
reshape_14 (Reshape)            (None, 50)           0           embedding_25[0][0]
__________________________________________________________________________________________________
embedding_24 (Embedding)        (None, 1, 1)         610         input_21[0][0]
__________________________________________________________________________________________________
embedding_26 (Embedding)        (None, 1, 1)         9724        input_22[0][0]
__________________________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 100)          0           reshape_12[0][0]
                                                                 reshape_14[0][0]
__________________________________________________________________________________________________
reshape_13 (Reshape)            (None, 1)            0           embedding_24[0][0]
__________________________________________________________________________________________________
reshape_15 (Reshape)            (None, 1)            0           embedding_26[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 100)          0           concatenate_4[0][0]
                                                                 reshape_13[0][0]
                                                                 reshape_15[0][0]
__________________________________________________________________________________________________
dense_48 (Dense)                (None, 10)           1010        add_2[0][0]
__________________________________________________________________________________________________
dense_49 (Dense)                (None, 1)            11          dense_48[0][0]
__________________________________________________________________________________________________
lambda_3 (Lambda)               (None, 1)            0           dense_49[0][0]
==================================================================================================
Total params: 528,055
Trainable params: 528,055
Non-trainable params: 0
__________________________________________________________________________________________________
```

Use regularization to help model generalization

Embedding Regularizer → The embedding is the biggest layer → Source of model complexity → Overfitting

→ Penalize by regularization

Layer (type)	Output Shape	Param #	Connected to
input_21 (InputLayer)	[(None, 1)]	0	
input_22 (InputLayer)	[(None, 1)]	0	
embedding_23 (Embedding)	(None, 1, 50)	30500	input_21[0][0]
embedding_25 (Embedding)	(None, 1, 50)	486200	input_22[0][0]
reshape_12 (Reshape)	(None, 50)	0	embedding_23[0][0]
reshape_14 (Reshape)	(None, 50)	0	embedding_25[0][0]
embedding_24 (Embedding)	(None, 1, 1)	610	input_21[0][0]
embedding_26 (Embedding)	(None, 1, 1)	9724	input_22[0][0]
concatenate_4 (Concatenate)	(None, 100)	0	reshape_12[0][0] reshape_14[0][0]
reshape_13 (Reshape)	(None, 1)	0	embedding_24[0][0]
reshape_15 (Reshape)	(None, 1)	0	embedding_26[0][0]
		0	concatenate_4[0][0] reshape_13[0][0] reshape_15[0][0]
		1010	add_2[0][0]
		11	dense_48[0][0]
		0	dense_49[0][0]

```python
# User embeddings
user = layers.Input(shape=(1,))
user_emb = layers.Embedding(n_users, emb_sz, embeddings_regularizer=regularizers.l2(1e-6))(user)
user_emb = layers.Reshape((emb_sz,))(user_emb)

# User bias
user_bias = layers.Embedding(n_users, 1, embeddings_regularizer=regularizers.l2(1e-6))(user)
user_bias = layers.Reshape((1,))(user_bias)
```

Analysis of Latent Factors

So far we don’t control the latent factors:

(Age, Year, Blockbuster,...etc)

They are learnt! We just set emb_sz

But can we understand what they represent?

Use PCA to summarize them to the top K factors → Say 3

(PCA → project to the most changing direction that holds the info

```
top_movies_embs = movie_emb_model.predict(movies_in)
```

```
top_movies_embs.shape
```

```
(2000, 50)
```

```
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
result = pca.fit_transform(top_movies_embs)
```

try tpo figure out what are the latent factors:

```
movie_latent_factors = pd.DataFrame(movies_titles)
movie_latent_factors['factor_0'] = result[:,0]
movie_latent_factors['factor_1'] = result[:,1]
movie_latent_factors['factor_2'] = result[:,2]
movie_latent_factors
```

Analysis of Latent Factors

If we sort by factor_0, we see on one end it captures classics, on the other it captures action movies

Let's sort by the factors:

```
[ ] movie_latent_factors.sort_values('factor_0')
```

	title	factor_0	factor_1	factor_2
2	Grumpier Old Men (1995)	-1.279781	-0.244140	-0.453658
24	Leaving Las Vegas (1995)	-1.226336	-0.033326	0.202144
134	Crimson Tide (1995)	-1.151981	-0.268237	0.227814
61	Friday (1995)	-1.081181	-0.283692	-0.473304
777	20,000 Leagues Under the Sea (1954)	-1.046890	-0.085251	0.516375
...	...	...	...	...
5923	Bewitched (2005)	1.161033	-0.146358	0.087674
8691	Deadpool (2016)	1.211176	-0.771655	0.063774
2224	Home Alone 2: Lost in New York (1992)	1.321183	-0.430576	-0.046365
7888	Prometheus (2012)	1.328679	-0.168954	0.467732
3915	Signs (2002)	1.508746	-0.404520	0.125356

2000 rows × 4 columns

Analysis of Latent Factors

If we sort by factor_2, it seems to caputre 80's and 90's movies

```
movie_latent_factors.sort_values('factor_2')
```

	title	factor_0	factor_1	factor_2
1347	Mercury Rising (1998)	0.332178	0.076090	-0.702017
117	Brothers McMullen, The (1995)	-0.589981	0.178241	-0.691810
946	Graduate, The (1967)	-0.338809	-0.401788	-0.658833
199	Exotica (1994)	-0.647799	-0.192046	-0.658026
2044	Mystery Men (1999)	0.481943	0.023822	-0.636760
...	...	...	...	...
830	Weekend at Bernie's (1989)	0.729404	0.545231	0.678696
630	Phenomenon (1996)	0.583151	0.238616	0.682040
1056	Star Trek V: The Final Frontier (1989)	-0.055650	0.001553	0.704540
598	Spy Hard (1996)	-0.570099	0.124406	0.772339
1435	Terms of Endearment (1983)	0.483175	0.176901	0.797862

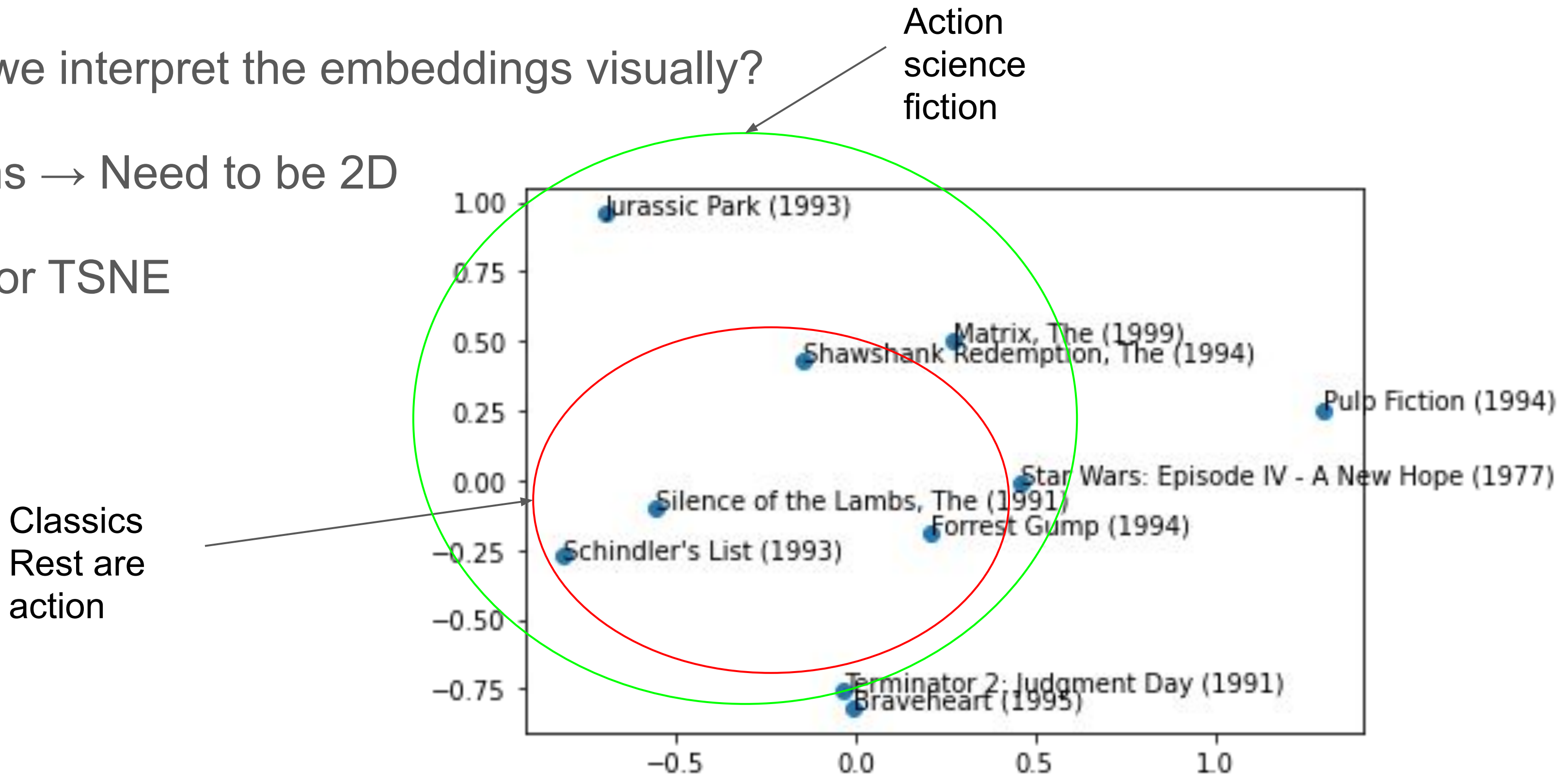
2000 rows × 4 columns

Visualization

How can we interpret the embeddings visually?

It’s 50 dims → Need to be 2D

Use PCA or TSNE



References

https://developers.google.com/machine-learning/recommendation/collaborative/basics

https://colab.research.google.com/drive/1GLofcB4RMPq8yQLyQcb_KA_mB5LRwYMa

https://colab.research.google.com/drive/1gKmqbo9Wr7Np4LI0S9cMiOpPZ_ZvOx6t#scrollTo=DDnVz1pkh5hf