

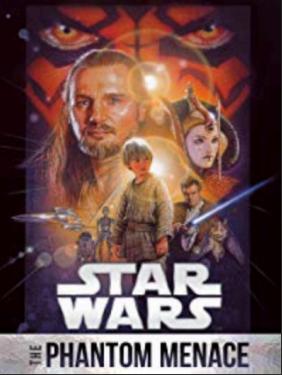
# Recommender System

Shusen Wang

# Recommender System

Secure | [https://www.amazon.com/Star-Wars-Phantom-Liam-Neeson/dp/B00VMHYKBA/ref=sr\\_1\\_6?ie=UTF8&qid=1532402082&sr=8-6&keywords=star+wars](https://www.amazon.com/Star-Wars-Phantom-Liam-Neeson/dp/B00VMHYKBA/ref=sr_1_6?ie=UTF8&qid=1532402082&sr=8-6&keywords=star+wars)

Star Wars: The Phantom Menace 1999 PG



★★★★★ (3,117) **IMDb** 6.5/10

[Watch Trailer](#)

Jedi Knights Obi-Wan Kenobi and Qui-Gon Jinn rescue Queen Amidala, ruler of a peaceful planet invaded by dark forces. On their escape, they discover nine-year-old Anakin Skywalker, a child prodigy who is unusually strong in the Force. Purchase also includes bonus features, available to watch in Your Video Library.

Starring: Liam Neeson, Ewan McGregor, Natalie Portman  
Runtime: 2 hours, 16 minutes  
Available to watch on [supported devices](#).

**Buy Movie HD \$19.99**

[More purchase options](#)

[Add to Watchlist](#)

Are you a student?  
**prime student**  Click here to start your six-month trial to get unlimited access to thousands of movies and TV shows, including award-winning Prime Original Series and movies.

Share    

Send us Feedback | Get Help

By placing your order or playing a video, you agree to our [Terms of Use](#). Sold by Amazon Digital Services LLC. Additional taxes may apply.

## Customers who watched this item also watched



# Basis of Recommendation

- Users' properties
  - gender, age, income, address, profession, etc.
- Items' properties
  - genre, name, brand, model, price, etc.
- User-item interaction history
  - browse, purchase, rating (score), review (text).

# Collaborative Filtering

# Basis of Recommendation

- Users' properties
  - gender, age, income, address, profession, etc.
- Items' properties
  - genre, name, brand, model, price, etc.
- User-item interaction history
  - browse, purchase, rating (score), review (text).

# Collaborative Filtering

## Model assumptions:

- Every user rates one or more items.
- Make recommendation purely based on the ratings.

# Collaborative Filtering

## Model assumptions:

- Every user rates one or more items.
- Make recommendation purely based on the ratings.

	Item1	Item2	Item3	Item4	Item5
User1	?	?	4	?	5
User2	3	?	2	?	?
User3	4	?	4	3	5
User4	?	3	?	5	?

# Collaborative Filtering

## Approach:

- Fill the missing entries based on the observed entries. (Matrix Completion.)
- Rank the filled entries to make recommendation.

	Item1	Item2	Item3	Item4	Item5
User1	?	?	4	?	5
User2	3	?	2	?	?
User3	4	?	4	3	5
User4	?	3	?	5	?

# Collaborative Filtering

## Approach:

- Fill the missing entries based on the observed entries. (Matrix Completion.)
- Rank the filled entries to make recommendation.

	Item1	Item2	Item3	Item4	Item5
User1	4.87	2.91	4	4.66	5
User2	3	?	2	?	?
User3	4	?	4	3	5
User4	?	3	?	5	?

# **Baseline**

# Beer Review Data (Kaggle)

- Source: <https://www.kaggle.com/c/beer-ratings/data>

- Number of users: 7442
- Number of beers: 1731
- Number of reviews: 37500

One review

```
'beer/beerId'  
'beer/brewerId'  
'beer/name'  
'beer/style'  
'user/ageInSeconds'  
'user/birthdayRaw'  
'user/birthdayUnix'  
'user/gender'  
'user/userId'  
'review/appearance'  
'review/aroma'  
'review/overall'  
'review/taste'  
'review/text'  
'review/timeStruct'  
'review/timeUnix'
```

# Beer Review Data (Kaggle)

- Source: <https://www.kaggle.com/c/beer-ratings/data>

- Number of users: 7442
- Number of beers: 1731
- Number of reviews: 37500

One review

```
'beer/beerId'  
'beer/brewerId'  
'beer/name'  
'beer/style'  
'user/ageInSeconds'  
'user/birthdayRaw'  
'user/birthdayUnix'  
'user/gender'  
'user/userId'  
'review/appearance'  
'review/aroma'  
'review/overall'  
'review/taste'  
'review/text'  
'review/timeStruct'  
'review/timeUnix'
```

# Beer Review Data (Kaggle)

- Source: <https://www.kaggle.com/c/beer-ratings/data>

- Number of users: 7442
- Number of beers: 1731
- Number of reviews: 37500
- 80% for training,
- 10% for validation
- 10% for test.

```
print(x_train)
User IDs  Beer IDs  Ratings
[ [ 479.    384.    4.5]
  [ 97.     159.    4.   ]
  [ 69.     500.    4.   ]
  ...
  [ 220.     27.    4.   ]
  [ 198.     36.    4.   ]
  [ 2637.    266.    5.   ] ]
```

```
print(x_train.shape)
```

(30000, 3)

# A Naïve Baseline Model

- Averaged rating on the training set: 3.889.
- For whatever user and beer, the predicted rating is 3.889.

```
rating_avg = np.mean(x_train[:, 2])
print('averaged rating is ' + str(rating_avg))
```

averaged rating is 3.8892333333333333

# A Naïve Baseline Model

- Averaged rating on the training set: 3.889.
- For whatever user and beer, the predicted rating is 3.889.
- The mean squared error (MSE) on the test set is 0.500.

```
rating_avg = np.mean(x_train[:, 2])
print('averaged rating is ' + str(rating_avg))
```

averaged rating is 3.8892333333333333

```
test_res = x_test[:, 2] - rating_avg
test_error_baseline = np.mean(np.multiply(test_res, test_res))
print('baseline test MSE: ' + str(test_error_baseline))
```

baseline test MSE: 0.4997916811111105

# A Naïve Baseline Model

- The naïve baseline model itself is **not useful**.
  - We cannot make recommendation using the baseline model.

	Item1	Item2	Item3	Item4	Item5
User1	3.889	3.889	4	3.889	5
User2	3	?	2	?	?
User3	4	?	4	3	5
User4	?	3	?	5	?

# A Naïve Baseline Model

- The naïve baseline model itself is not useful.
  - We cannot make recommendation using the baseline model.
- But the baseline MSE is an important reference.
  - Compare our models with the baseline.
  - Some models may not beat the baseline.

# The Sum Model

# The Sum Model

- Each item has a score,  $a_i$ , indicating its quality.
  - E.g., most people think the movies *Lord of Rings* are better than *Warcraft*.
- Each user has a score,  $b_u$ , indicating her rating behavior.
  - E.g., some people are “nice” reviewers on IMDB, Amazon, etc.
  - A “nice” reviewer has a higher  $b$  than a “critical” reviewer.

# The Sum Model

- Each item has a score,  $a_i$ , indicating its quality.
  - E.g., most people think the movies *Lord of Rings* are better than *Warcraft*.
- Each user has a score,  $b_u$ , indicating her rating behavior.
  - E.g., some people are “nice” reviewers on IMDB, Amazon, etc.
  - A “nice” reviewer has a higher  $b$  than a “critical” reviewer.
- Predict user- $u$ ’s rating of item- $i$  by:

$$f_{i,u} = a_i + b_u + c.$$

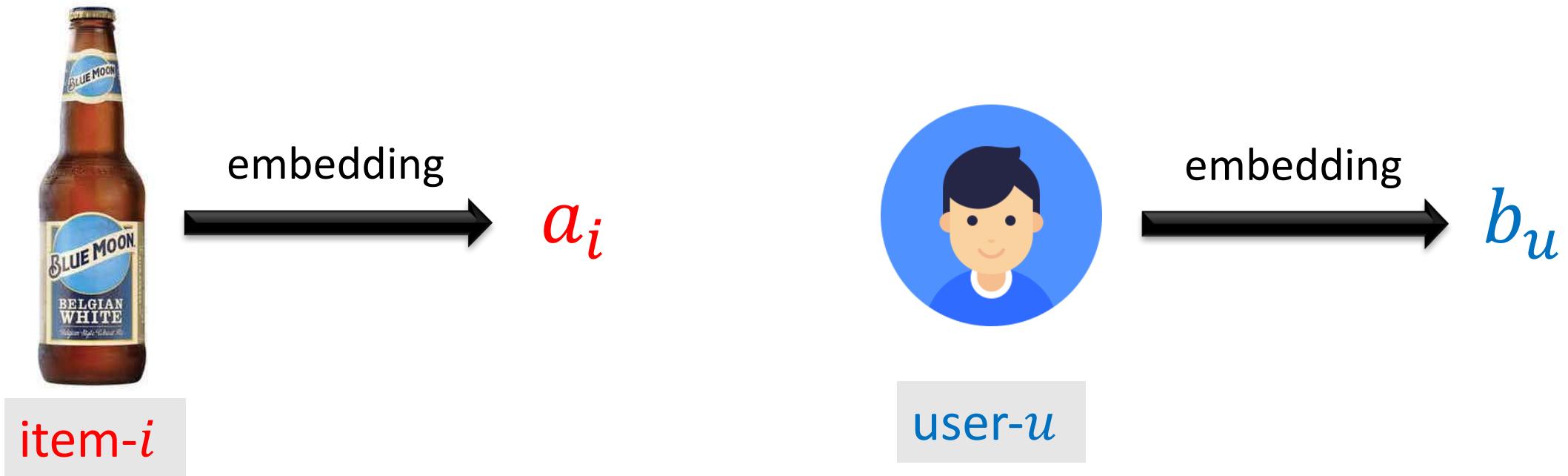
Common among all  
the items and users.

# Train the Sum Model

- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .
- There are  $m$  items and  $n$  users.
  - In the beer dataset,  $m = 1731$  and  $n = 7442$ .
  - Trainable parameters:  $a_1, \dots, a_m, b_1, \dots, b_n$ , and  $c$ .

# Train the Sum Model

- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .
- There are  $m$  items and  $n$  users.
  - In the beer dataset,  $m = 1731$  and  $n = 7442$ .
  - Trainable parameters:  $a_1, \dots, a_m$ ,  $b_1, \dots, b_n$ , and  $c$ .



# Train the Sum Model

- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .
- There are  $m$  items and  $n$  users.
  - In the beer dataset,  $m = 1731$  and  $n = 7442$ .
  - Trainable parameters:  $a_1, \dots, a_m, b_1, \dots, b_n$ , and  $c$ .
- The set  $\Omega = \{(i, u)\}$  indexes all the existing ratings.
  - We know the actual rating  $\{r_{i,u}\}$  for all  $(i, u) \in \Omega$ .
  - In the beer dataset, the cardinality is  $|\Omega| = 37,500$

# Train the Sum Model

- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .
- There are  $m$  items and  $n$  users.
  - In the beer dataset,  $m = 1731$  and  $n = 7442$ .
  - Trainable parameters:  $a_1, \dots, a_m, b_1, \dots, b_n$ , and  $c$ .
- The set  $\Omega = \{(i, u)\}$  indexes all the existing ratings.
  - We know the actual rating  $\{r_{i,u}\}$  for all  $(i, u) \in \Omega$ .
  - In the beer dataset, the cardinality is  $|\Omega| = 37,500 = 0.3\% mn$ .

If every user has rated every beer, then the number of reviews is  $mn$ .

# Train the Sum Model

- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .
- There are  $m$  items and  $n$  users.
  - In the beer dataset,  $m = 1731$  and  $n = 7442$ .
  - Trainable parameters:  $a_1, \dots, a_m, b_1, \dots, b_n$ , and  $c$ .
- The set  $\Omega = \{(i, u)\}$  indexes all the existing ratings.
  - We know the actual rating  $\{r_{i,u}\}$  for all  $(i, u) \in \Omega$ .
  - In the beer dataset, the cardinality is  $|\Omega| = 37,500 = 0.3\% mn$ .
- Find  $a_1, \dots, a_m, b_1, \dots, b_n$ , and  $c$  by minimizing

$$\frac{1}{|\Omega|} \sum_{(i,u) \in \Omega} (f_{i,u} - r_{i,u})^2.$$

- $m + n + 1 = 9,174$  trainable parameters.
- $|\Omega| = 37,500$  samples (80% for training).

# Implement the Sum Model Using Keras

- **Embedding of items:** map the index  $i$  to a real number  $a_i$ .

```
item_id = Input(shape=[1], name='item_id')  
embed_dim = 1  
item_num = 1731  
  
item_embed = Embedding(item_num, 1, input_length=1, name='item_embed')(item_id)  
item_flat = Flatten(name='item_flat')(item_embed)
```

scalar



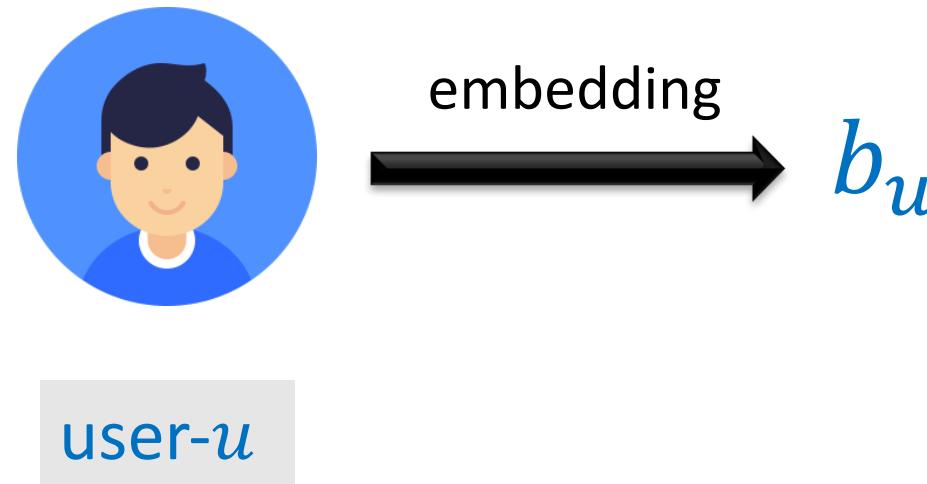
embedding

$a_i$

item- $i$

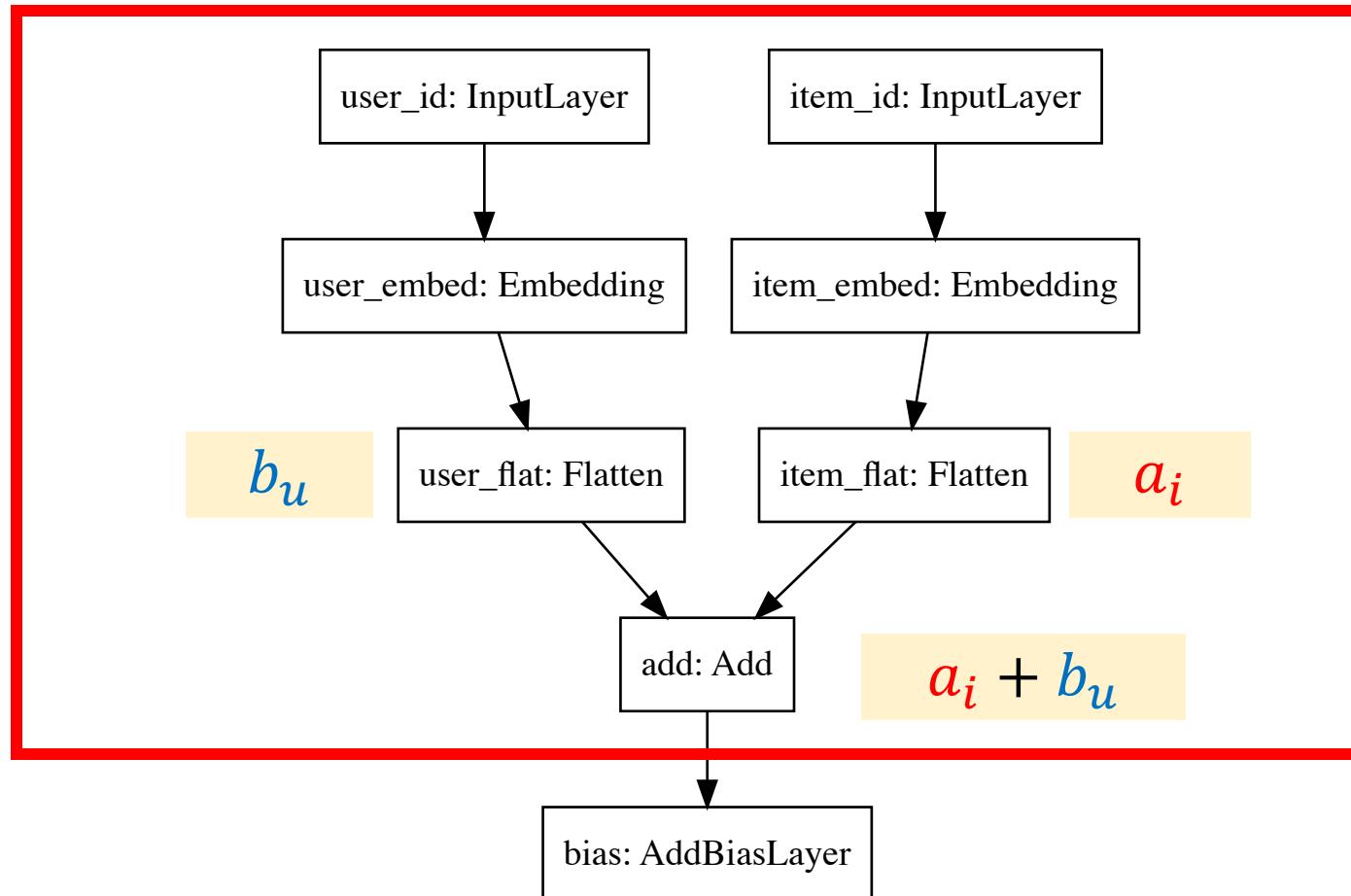
# Implement the Sum Model Using Keras

- **Embedding of items:** map the index  $i$  to a real number  $a_i$ .
- **Embedding of users:** map the index  $u$  to a real number  $b_u$ .
  - Similarly implemented.



# Implement the Sum Model Using Keras

The sum model: predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .



# Implement the Sum Model Using Keras

The sum model: predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .

```
from keras.models import Model
from keras.layers import Input, Embedding, Flatten, Add

embed_dim = 1

user_id = Input(shape=[1], name='user_id')
item_id = Input(shape=[1], name='item_id')

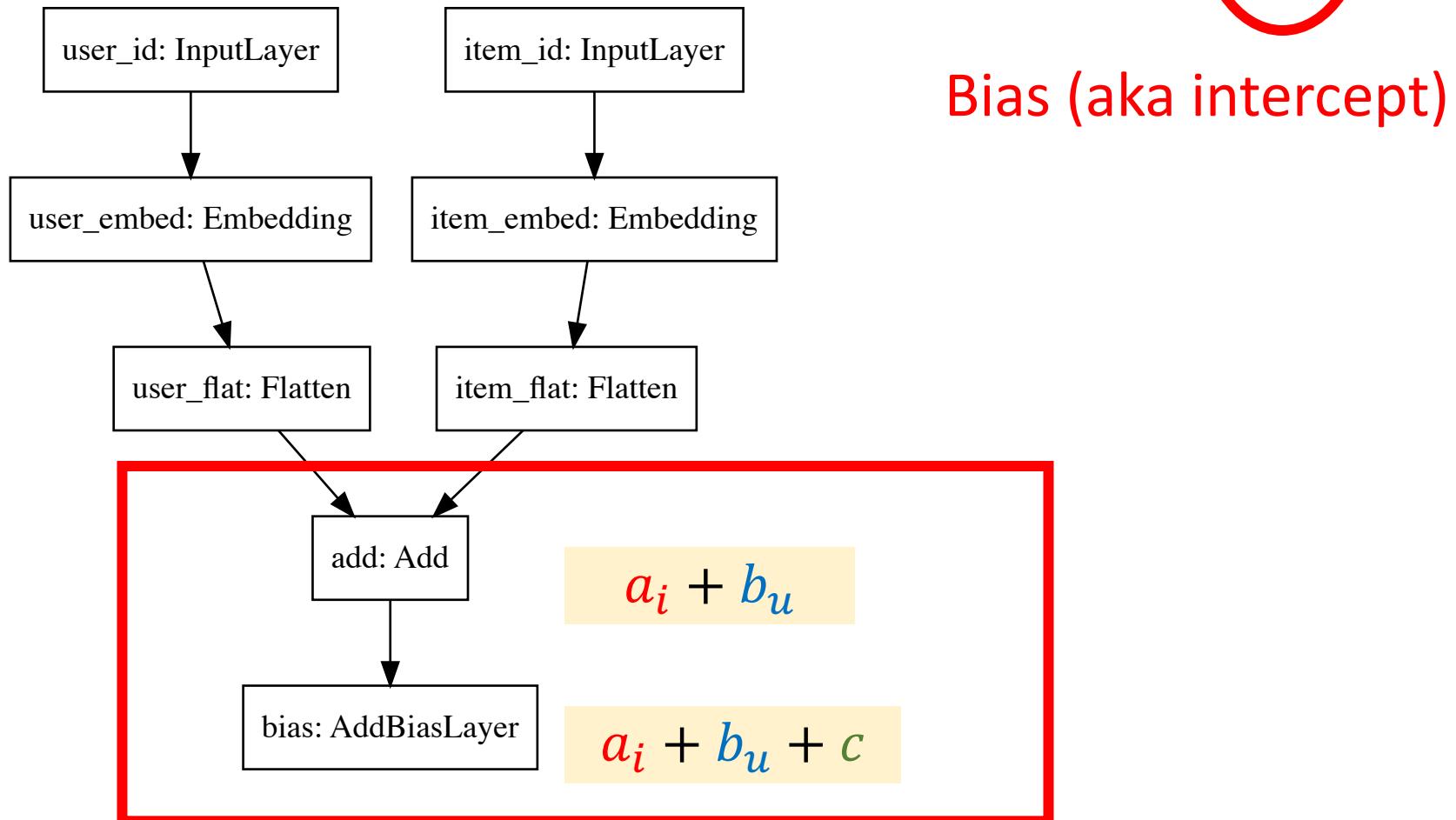
user_num = 7442
user_embed = Embedding(user_num, embed_dim, input_length=1, name='user_embed')(user_id)
user_flat = Flatten(name='user_flat')(user_embed) = bu

item_num = 1731
item_embed = Embedding(item_num, embed_dim, input_length=1, name='item_embed')(item_id)
item_flat = Flatten(name='item_flat')(item_embed) = ai

added = Add(name='add')([user_flat, item_flat]) = ai + bu
```

# Implement the Sum Model Using Keras

The sum model: predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .



# Implement the Sum Model Using Keras

The sum model: predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .

```
from keras import backend as K
from keras.engine.topology import Layer

class AddBiasLayer(Layer):
    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(AddBiasLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.bias = self.add_weight(name='bias', shape=(self.output_dim,),
                                   initializer='uniform', trainable=True)
        super(AddBiasLayer, self).build(input_shape)

    def call(self, inputs):
        c = K.bias_add(inputs, self.bias)
        return c

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)
```

# Implement the Sum Model Using Keras

The sum model: predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = a_i + b_u + c$ .

```
added = Add(name='add')([user_flat, item_flat])          =  $a_i + b_u$ 
pred = AddBiasLayer(output_dim=1, name='bias')(added)    =  $a_i + b_u + c$ 
sum_model = Model(inputs=[user_id, item_id], outputs=pred, name='sum_model')
```

# Implement the Sum Model Using Keras

```
sum_model.compile(loss='mse', optimizer='RMSprop')

history = sum_model.fit([x_train[:, 0], x_train[:, 1],
                        x_train[:, 2], batch_size=64,
                        epochs=30, shuffle=True,
                        validation_data=([x_val[:, 0], x_val[:, 1]], x_val[:, 2]))
```

Train on 30000 samples, validate on 3750 samples

Epoch 1/30

30000/30000 [=====] - 1s 22us/step - loss: 12.7309 - val\_loss: 10.276

Epoch 2/30

30000/30000 [=====] - 1s 17us/step - loss: 8.1294 - val\_loss: 6.2546

Epoch 3/30

30000/30000 [=====] - 1s 17us/step - loss: 4.6506 - val\_loss: 3.3405

•  
•  
•

Epoch 29/30

30000/30000 [=====] - 1s 18us/step - loss: 0.3283 - val\_loss: 0.3754

Epoch 30/30

30000/30000 [=====] - 0s 16us/step - loss: 0.3266 - val\_loss: 0.3753

# The Sum Model

- The baseline model
  - Validation error: 0.496
  - Test error: 0.4998
- The sum model
  - Validation error: 0.375
  - Test error: 0.393
- The sum model beats the baseline.
- The sum model is simple but hard to beat!

# The Sum Model Is Not Very Useful

- Predict user- $u$ 's rating of item- $i$  by:

$$f_{i,u} = a_i + b_u + c.$$

A diagram illustrating the components of the sum model. The equation  $f_{i,u} = a_i + b_u + c.$  is centered. A red curved arrow points from the term  $a_i$  to the text "Item's quality" located below it. A blue curly brace points from the term  $b_u$  to the text "User's rating behavior" located to its right.

Item's quality      User's rating behavior

- Recommendation is purely based on item's quality.
- Same recommendation for every user.

# The Sum Model Is Not Very Useful

- Predict user- $u$ 's rating of item- $i$  by:

$$f_{i,u} = a_i + b_u + c.$$

Item's quality      User's rating behavior

- Recommendation is purely based on item's quality.
- Same recommendation for every user.
- Whether an item **matches** a user is not considered.
  - Recommend the movie *HP2* to a user who strongly dislike *HP1*.
  - Simply because *HP2*'s own quality is high.

# Product Model

# Product Model

- Each item is represented by a feature vector  $\mathbf{a}_i \in \mathbb{R}^k$ .
  - $\mathbf{a}_i$ : item's property.



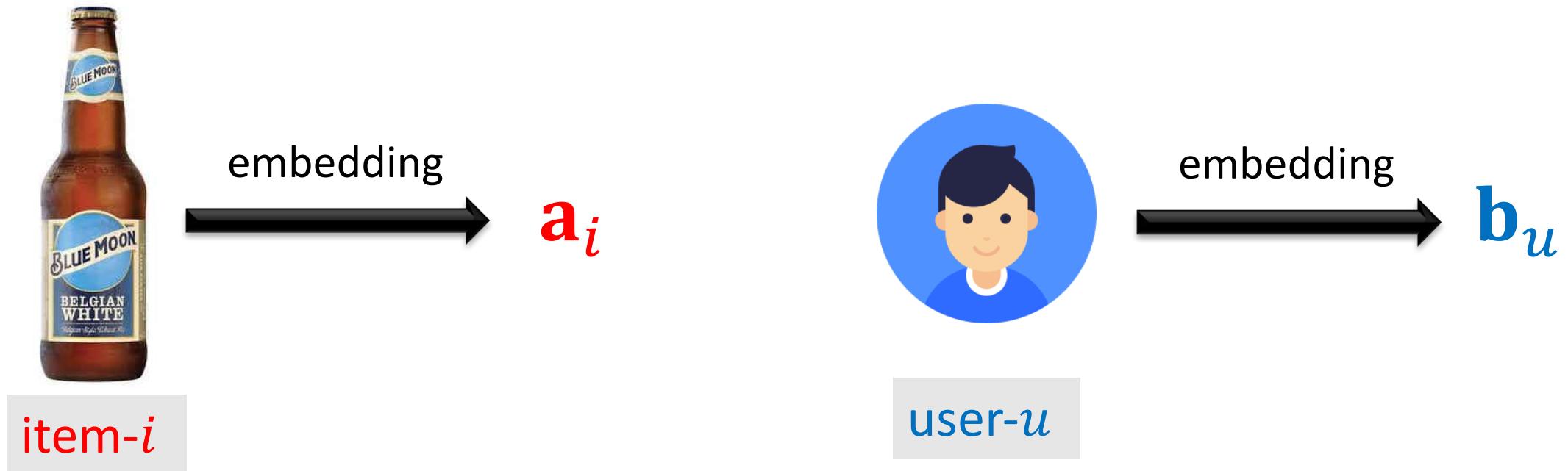
embedding

$\mathbf{a}_i$

item-*i*

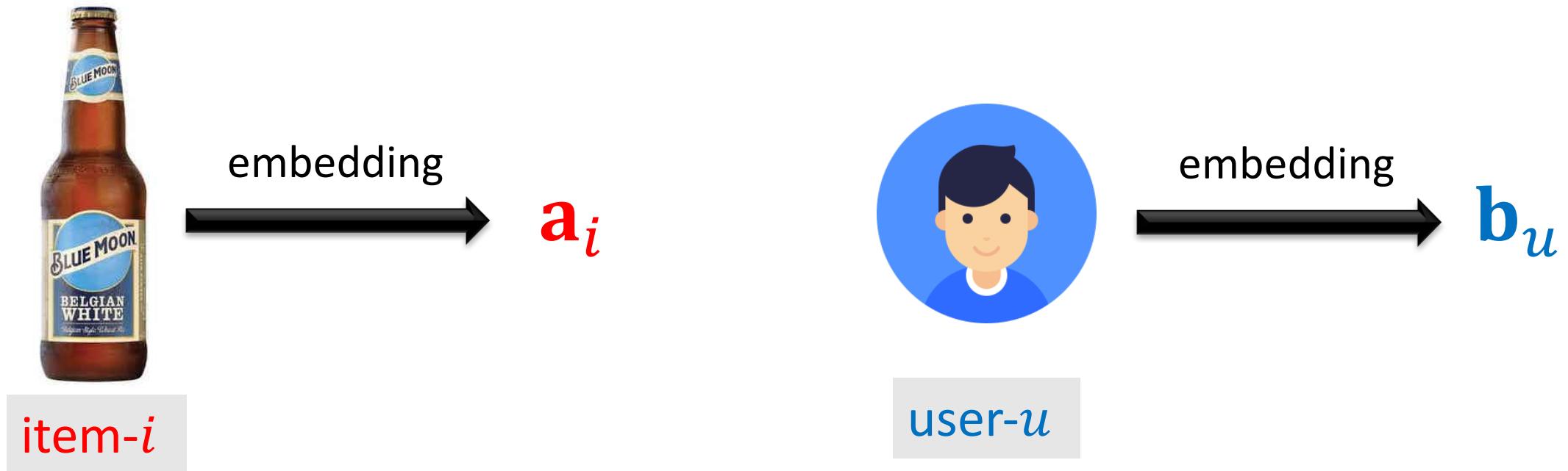
# Product Model

- Each item is represented by a feature vector  $\mathbf{a}_i \in \mathbb{R}^k$ .
- Each user is represented by a feature vector  $\mathbf{b}_u \in \mathbb{R}^k$ .
  - $\mathbf{b}_u$ : user's preference/taste.



# Product Model

- Each item is represented by a feature vector  $\mathbf{a}_i \in \mathbb{R}^k$ .
- Each user is represented by a feature vector  $\mathbf{b}_u \in \mathbb{R}^k$ .
- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = \mathbf{a}_i^T \mathbf{b}_u + c$ .
  - How well the item's property matches the user taste.



# Train the Product Model

- Each item is represented by a feature vector  $\mathbf{a}_i \in \mathbb{R}^k$ .
- Each user is represented by a feature vector  $\mathbf{b}_u \in \mathbb{R}^k$ .
- Predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = \mathbf{a}_i^T \mathbf{b}_u + c$ .
- Find  $\mathbf{a}_1, \dots, \mathbf{a}_m, \mathbf{b}_1, \dots, \mathbf{b}_n \in \mathbb{R}^k$  and  $c \in \mathbb{R}$  by minimizing

$$\frac{1}{|\Omega|} \sum_{(i,u) \in \Omega} (r_{i,u} - f_{i,u})^2.$$

The diagram shows the equation  $\frac{1}{|\Omega|} \sum_{(i,u) \in \Omega} (r_{i,u} - f_{i,u})^2$ . Two arrows point downwards from the terms  $r_{i,u}$  and  $f_{i,u}$  in the equation to the labels "Target (label)" and "Model's prediction" respectively, positioned below the equation.

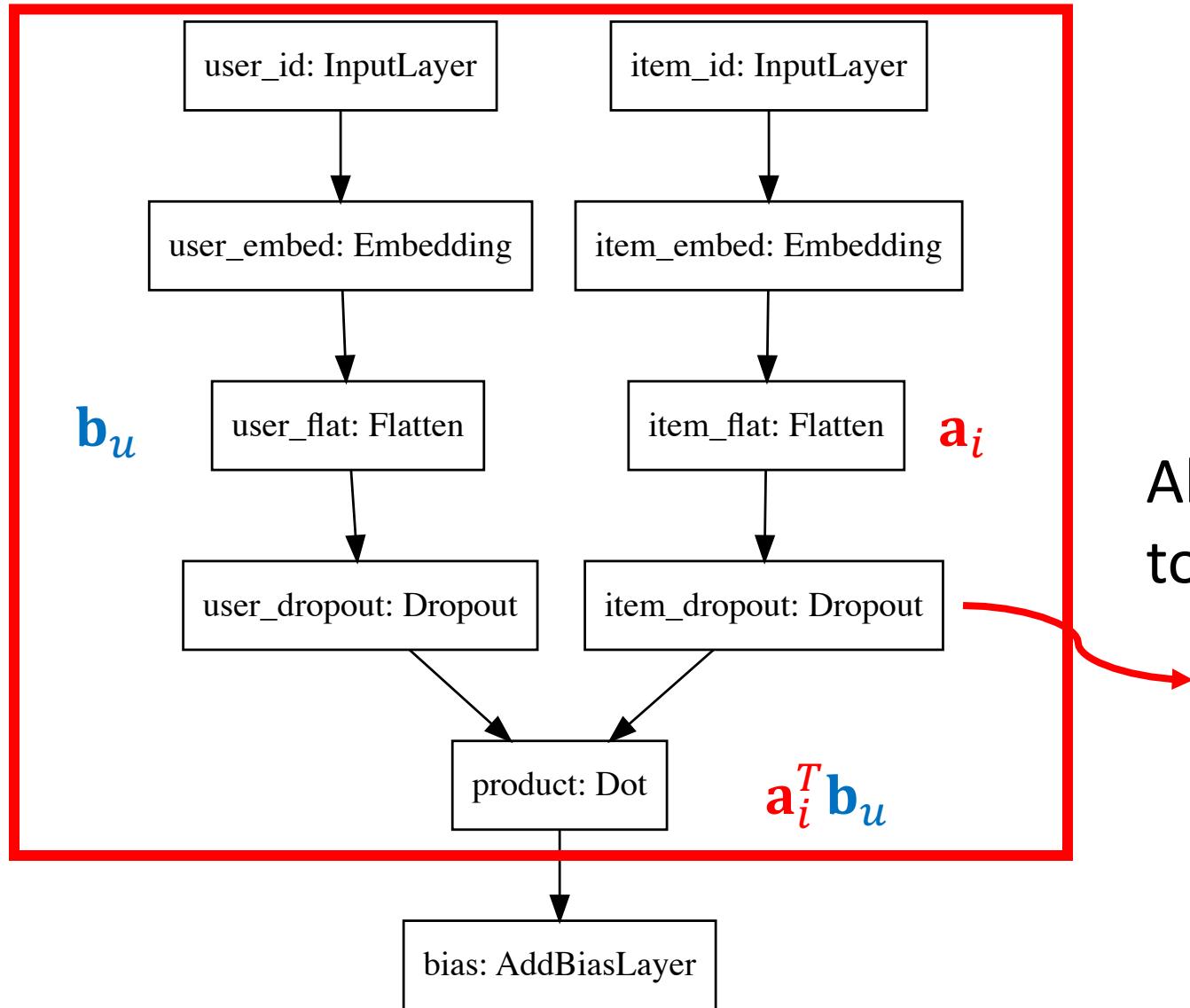
Target (label)      Model's prediction

# Product Model

Also known as “Matrix Factorization”.

- Let the full item-user rating matrix be  $\mathbf{R} \in \mathbb{R}^{m \times n}$ .
- Observe part of  $\mathbf{R}$  (entries indexed by  $\Omega$ ) .
- Seek to learn such  $\mathbf{A} \in \mathbb{R}^{m \times k}$  and  $\mathbf{B} \in \mathbb{R}^{n \times k}$  that  $\mathbf{AB}^T \approx \mathbf{R}$ .
- $m k + n k$  trainable parameters and  $|\Omega|$  samples.

# Implement the Product Model using Keras



Alternatively, added the regularizations to the objective function:

$$\lambda \|\mathbf{a}_i\|_2^2 + \lambda \|\mathbf{b}_u\|_2^2.$$

# Implement the Product Model using Keras

```
from keras.models import Model
from keras.layers import Input, Embedding, Flatten, Dot, Dropout

embed_dim = 10
dropout_rate = 0.5

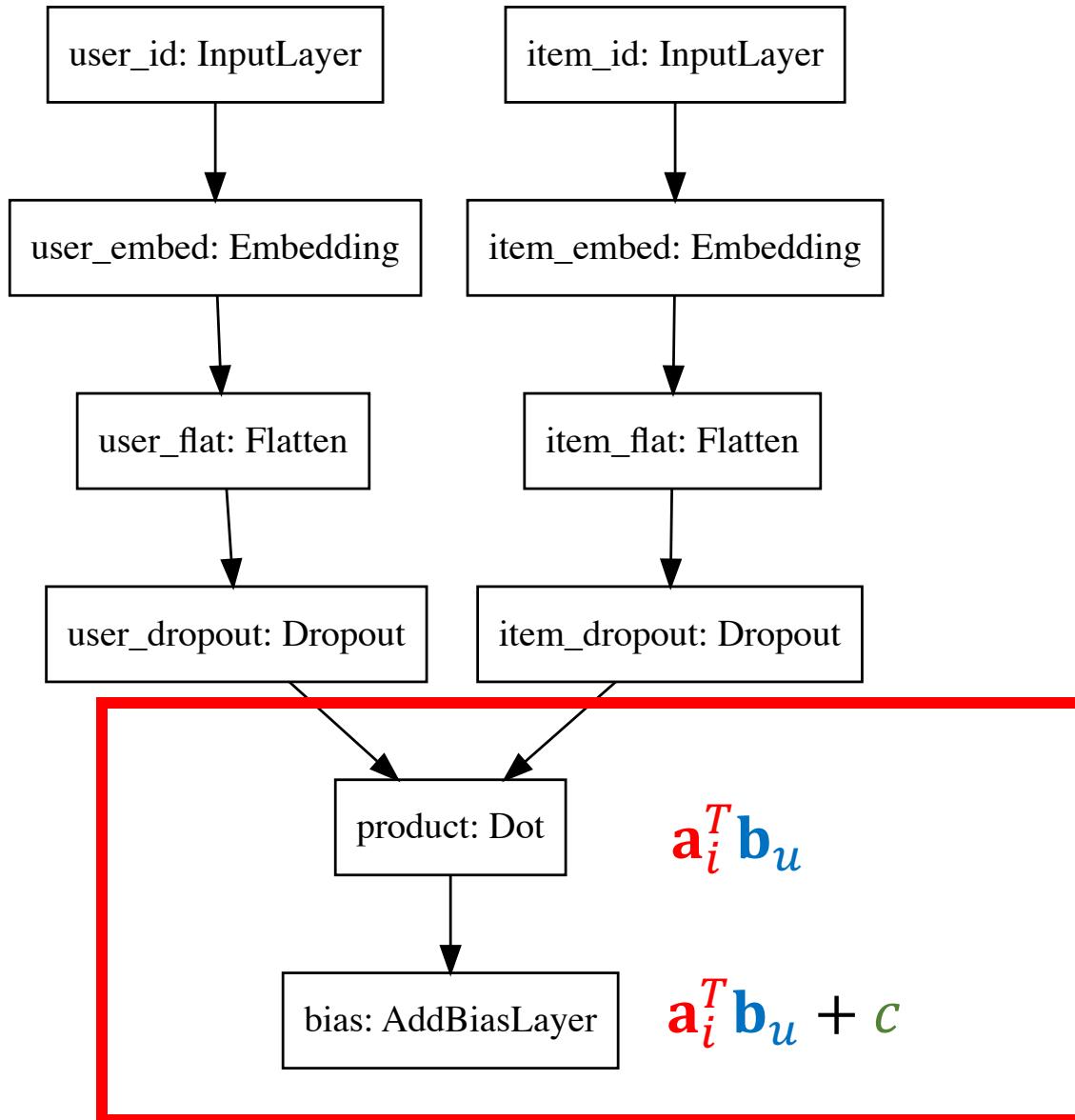
user_id = Input(shape=[1], name='user_id')
item_id = Input(shape=[1], name='item_id')

user_num = len(user_name_index)
user_embed = Embedding(user_num, embed_dim, input_length=1, name='user_embed')(user_id)
user_flat = Flatten(name='user_flat')(user_embed) =  $\mathbf{b}_u$ 
user_dropout = Dropout(rate=dropout_rate, name='user_dropout')(user_flat)

item_num = len(beer_id_index)
item_embed = Embedding(item_num, embed_dim, input_length=1, name='item_embed')(item_id)
item_flat = Flatten(name='item_flat')(item_embed) =  $\mathbf{a}_i$ 
item_dropout = Dropout(rate=dropout_rate, name='item_dropout')(item_flat)

product = Dot(1, normalize=False, name='product')([user_dropout, item_dropout]) =  $\mathbf{a}_i^T \mathbf{b}_u$ 
```

# Implement the Product Model using Keras



# Implement the Product Model using Keras

The product model: predict user- $u$ 's rating of item- $i$  by:  $f_{i,u} = \mathbf{a}_i^T \mathbf{b}_u + c$ .

```
product = Dot(1, normalize=False, name='product')([user_dropout, item_dropout])  
=  $\mathbf{a}_i^T \mathbf{b}_u$   
pred = AddBiasLayer(output_dim=1, name='bias')(product)  
=  $\mathbf{a}_i^T \mathbf{b}_u + c$   
prod_model = Model(inputs=[user_id, item_id], outputs=pred, name='product_model')
```

# Implement the Product Model using Keras

```
prod_model.compile(loss='mse', optimizer='RMSprop')

history = prod_model.fit([x_train[:, 0], x_train[:, 1],
                          x_train[:, 2], batch_size=64,
                          epochs=50, shuffle=True,
                          validation_data=([x_val[:, 0], x_val[:, 1]], x_val[:, 2]))
```

Train on 30000 samples, validate on 3750 samples

Epoch 1/50

30000/30000 [=====] - 1s 35us/step - loss: 13.8177 - val\_loss: 12.106

Epoch 2/50

30000/30000 [=====] - 1s 23us/step - loss: 10.6212 - val\_loss: 9.1376

Epoch 3/50

30000/30000 [=====] - 1s 24us/step - loss: 7.8628 - val\_loss: 6.6029

•  
•  
•

Epoch 49/50

30000/30000 [=====] - 1s 21us/step - loss: 0.3943 - val\_loss: 0.4338

Epoch 50/50

30000/30000 [=====] - 1s 22us/step - loss: 0.3942 - val\_loss: 0.4338

# Product Model

- The baseline model
  - Validation error: 0.496
  - Test error: 0.50

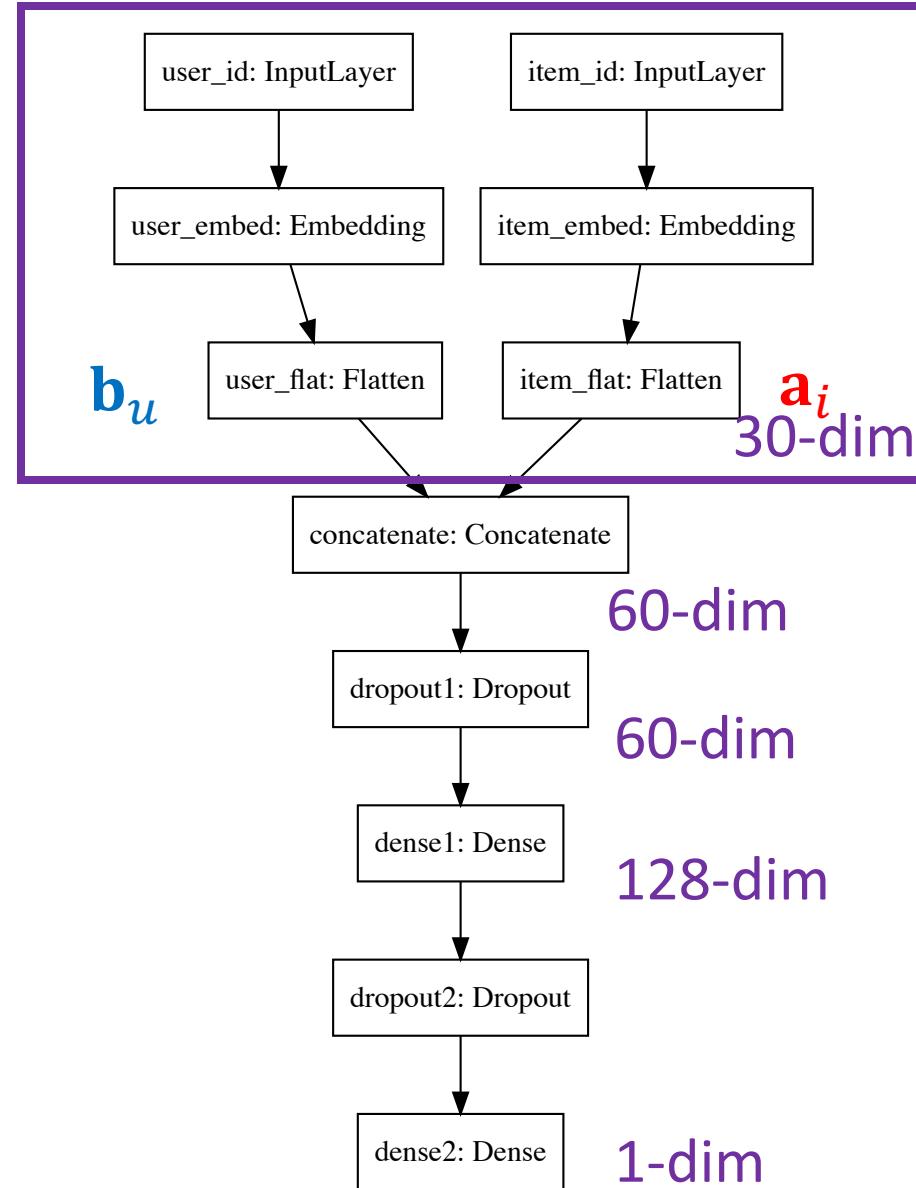
- The sum model
  - Validation error: 0.3753
  - Test error: 0.3928

- The product model
  - Validation error: 0.4338
  - Test error: 0.4468

# Neural Networks

# Neural Networks

The same as the product model



# Neural Networks

```
from keras.models import Model
from keras.layers import Input, Embedding, Flatten, Dense, Dropout, Concatenate

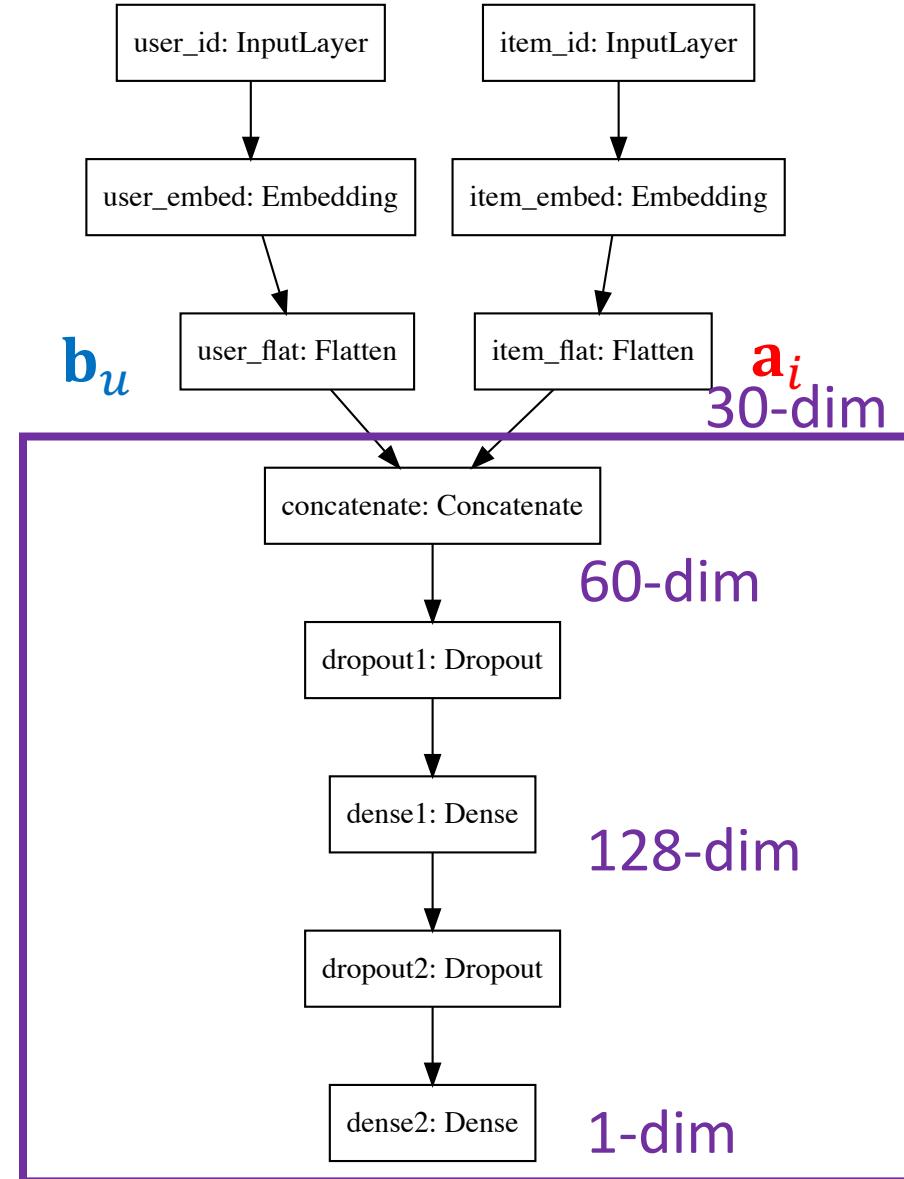
embed_dim = 30
dropout_rate = 0.5

user_id = Input(shape=[1], name='user_id')
item_id = Input(shape=[1], name='item_id')

user_num = len(user_name_index)
user_embed = Embedding(user_num, embed_dim, input_length=1, name='user_embed')(user_id)
user_flat = Flatten(name='user_flat')(user_embed) =  $\mathbf{b}_u \in \mathbb{R}^{30}$ 

item_num = len(beer_id_index)
item_embed = Embedding(item_num, embed_dim, input_length=1, name='item_embed')(item_id)
item_flat = Flatten(name='item_flat')(item_embed) =  $\mathbf{a}_i \in \mathbb{R}^{30}$ 
```

# Neural Networks

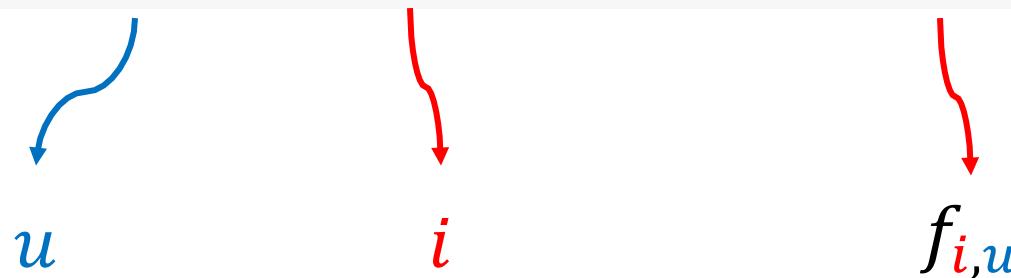


A standard fully-connected  
neural network

# Neural Networks

```
concatenate = Concatenate(name='concatenate')([user_flat, item_flat])
dropout1 = Dropout(rate=dropout_rate, name='dropout1')(concatenate)
dense1 = Dense(128, activation='relu', name='dense1')(dropout1)
dropout2 = Dropout(rate=dropout_rate, name='dropout2')(dense1)
pred = Dense(1, activation='relu', name='dense2')(dropout2)

neuralnet = Model(inputs=[user_id, item_id], outputs=pred, name='neural_net')
```



# Neural Networks

```
neuralnet.compile(loss='mse', optimizer='RMSprop')

history = neuralnet.fit([x_train[:, 0], x_train[:, 1],
                        x_train[:, 2], batch_size=16,
                        epochs=10, shuffle=True,
                        validation_data=(x_val[:, 0], x_val[:, 1]), x_val[:, 2]))
```

Train on 30000 samples, validate on 3750 samples

Epoch 1/10

30000/30000 [=====] - 4s 149us/step - loss: 1.1534 - val\_loss: 0.3959

Epoch 2/10

30000/30000 [=====] - 4s 142us/step - loss: 0.5304 - val\_loss: 0.3833

Epoch 3/10

30000/30000 [=====] - 4s 134us/step - loss: 0.4830 - val\_loss: 0.3657

•  
•  
•

Epoch 9/10

30000/30000 [=====] - 4s 122us/step - loss: 0.3522 - val\_loss: 0.3637

Epoch 10/10

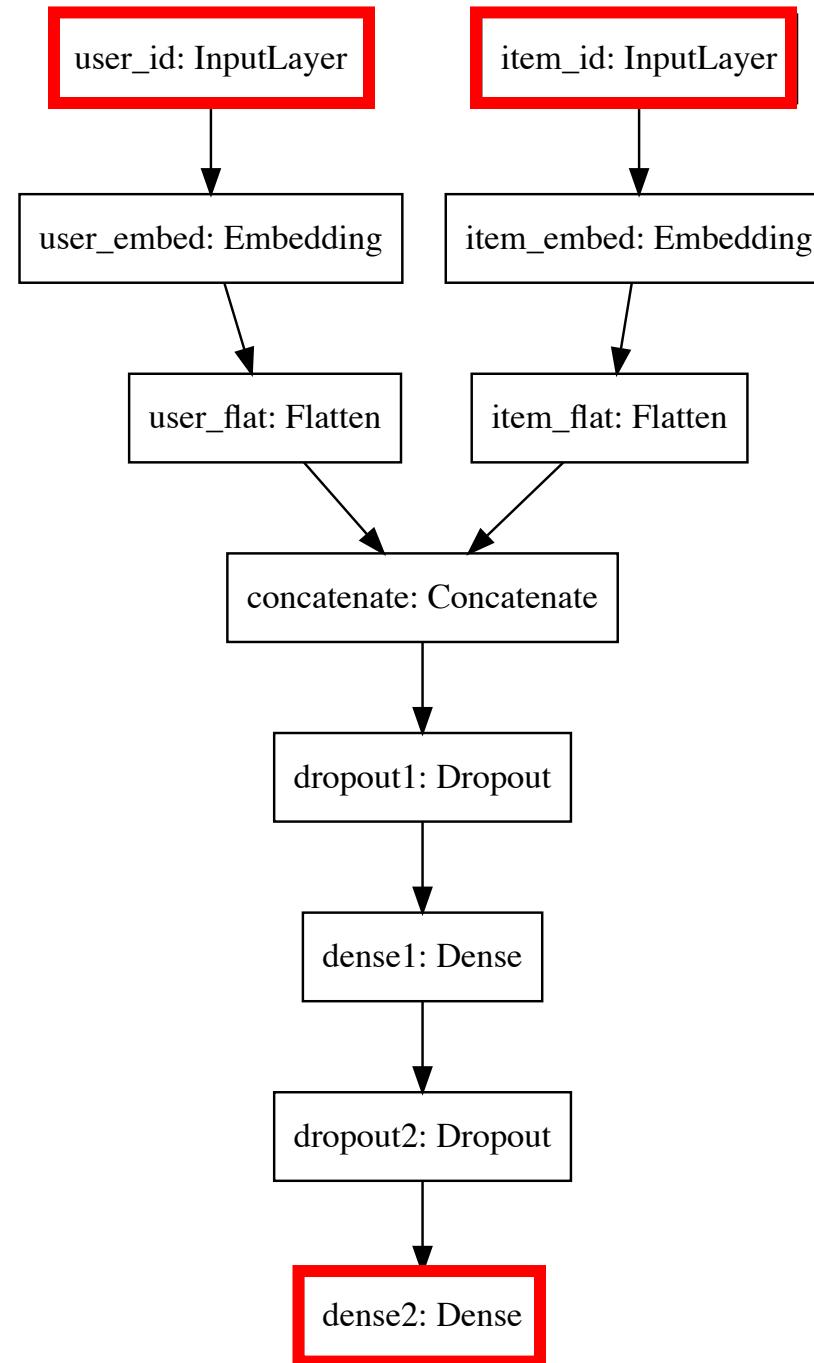
30000/30000 [=====] - 4s 120us/step - loss: 0.3441 - val\_loss: 0.3697

# Comparisons on the Beer Dataset

- The baseline model
  - Validation error: 0.496
  - Test error: 0.50
- The product model
  - Validation error: 0.4338
  - Test error: 0.4468
- The sum model
  - Validation error: 0.3753
  - Test error: 0.3928
- Neural network
  - Validation error: 0.3697
  - Test error: 0.3891

# **Discussions**

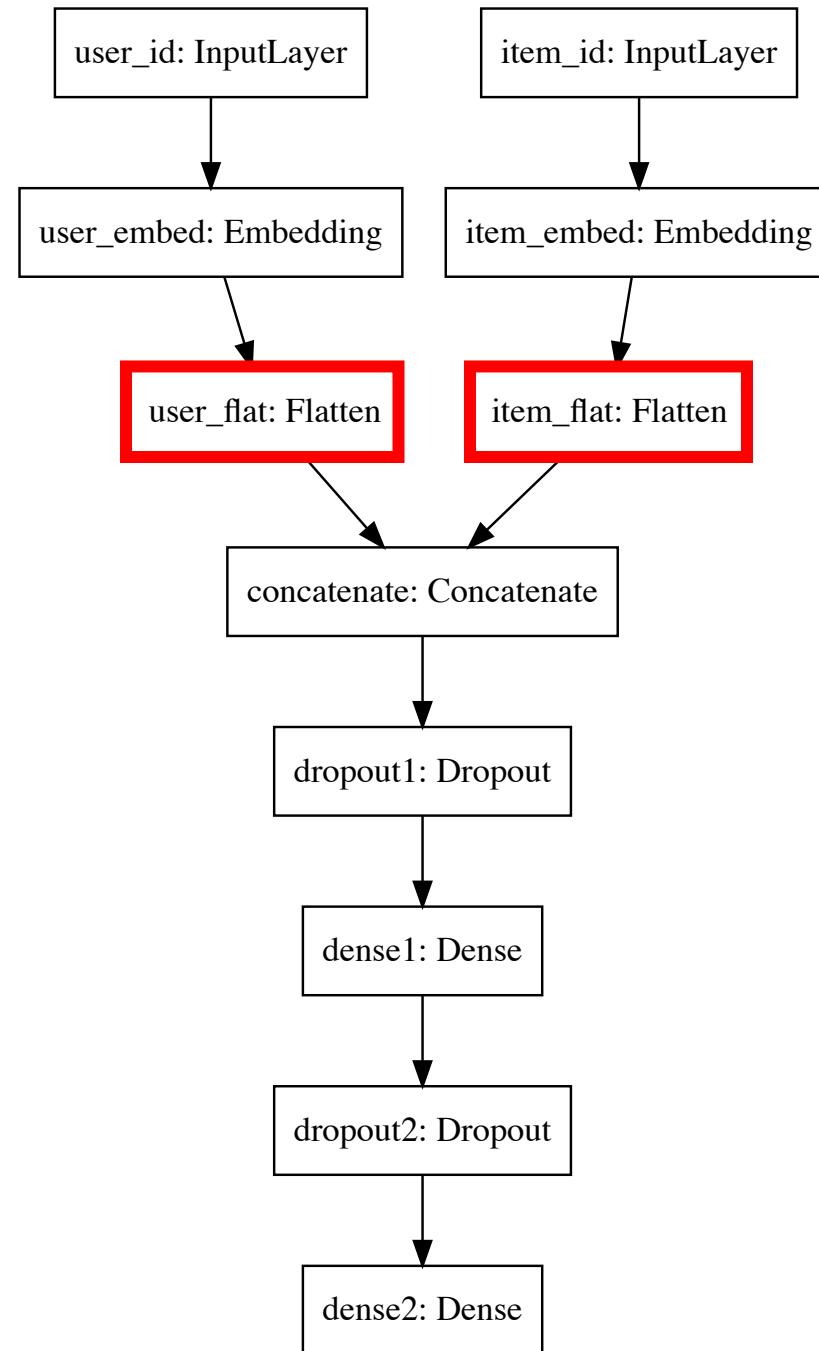
# Collaborative Filtering



- Trained a model for predicting user-item rating.
- Did not use items and users' features.

'beer/beerId'  
'beer/brewerId'  
'beer/name'  
'beer/style'  
'user/ageInSeconds'  
'user/birthdayRaw'  
'user/birthdayUnix'  
'user/gender'  
**'user/userId'**  
'review/appearance'  
'review/aroma'  
**'review/overall'**  
'review/taste'  
'review/text'  
'review/timeStruct'  
**'review/timeUnix'**

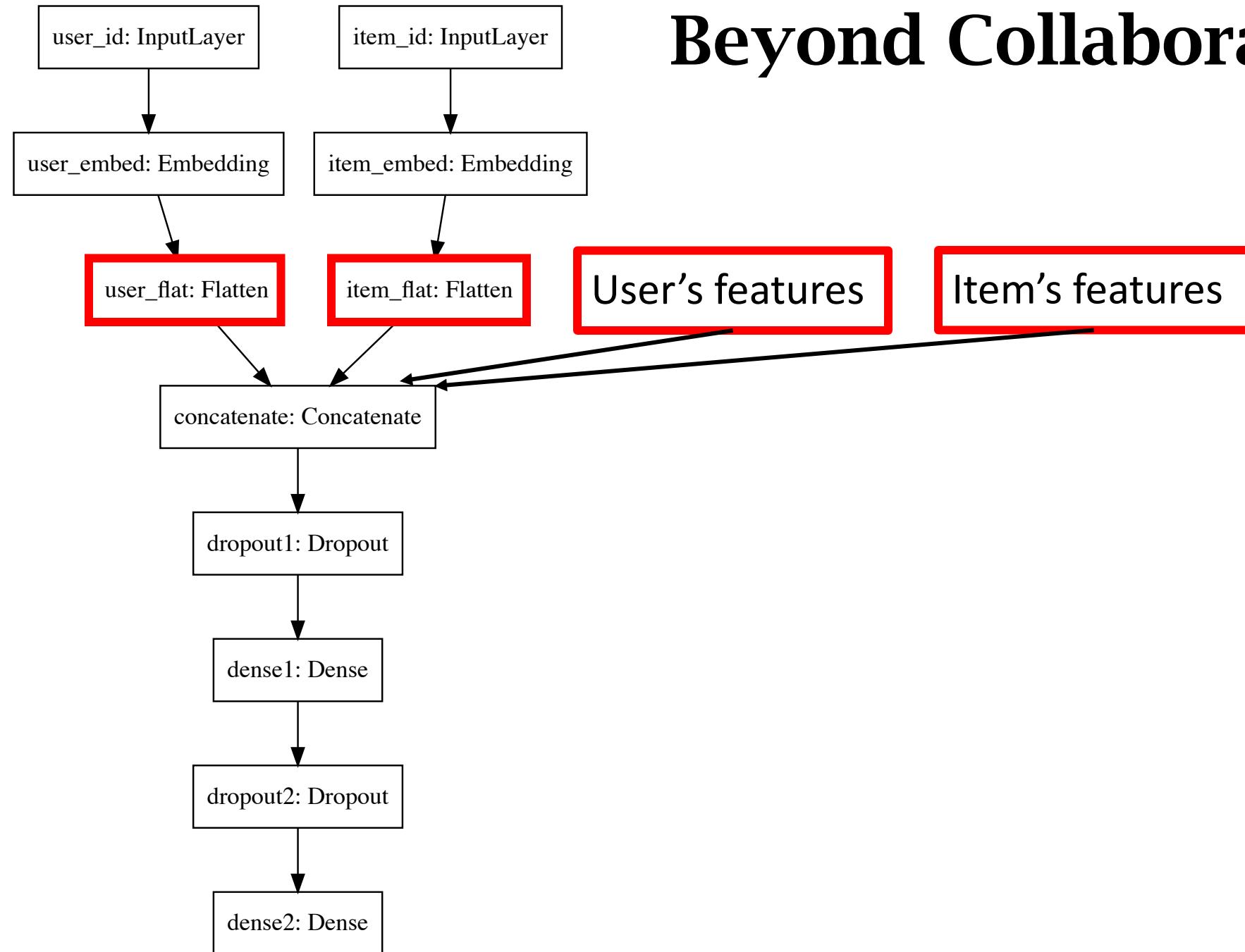
# Collaborative Filtering



- Trained a model for predicting user-item rating.
- Did not use items and users' features.
- The learned features are effective. (Beat the baselines.)

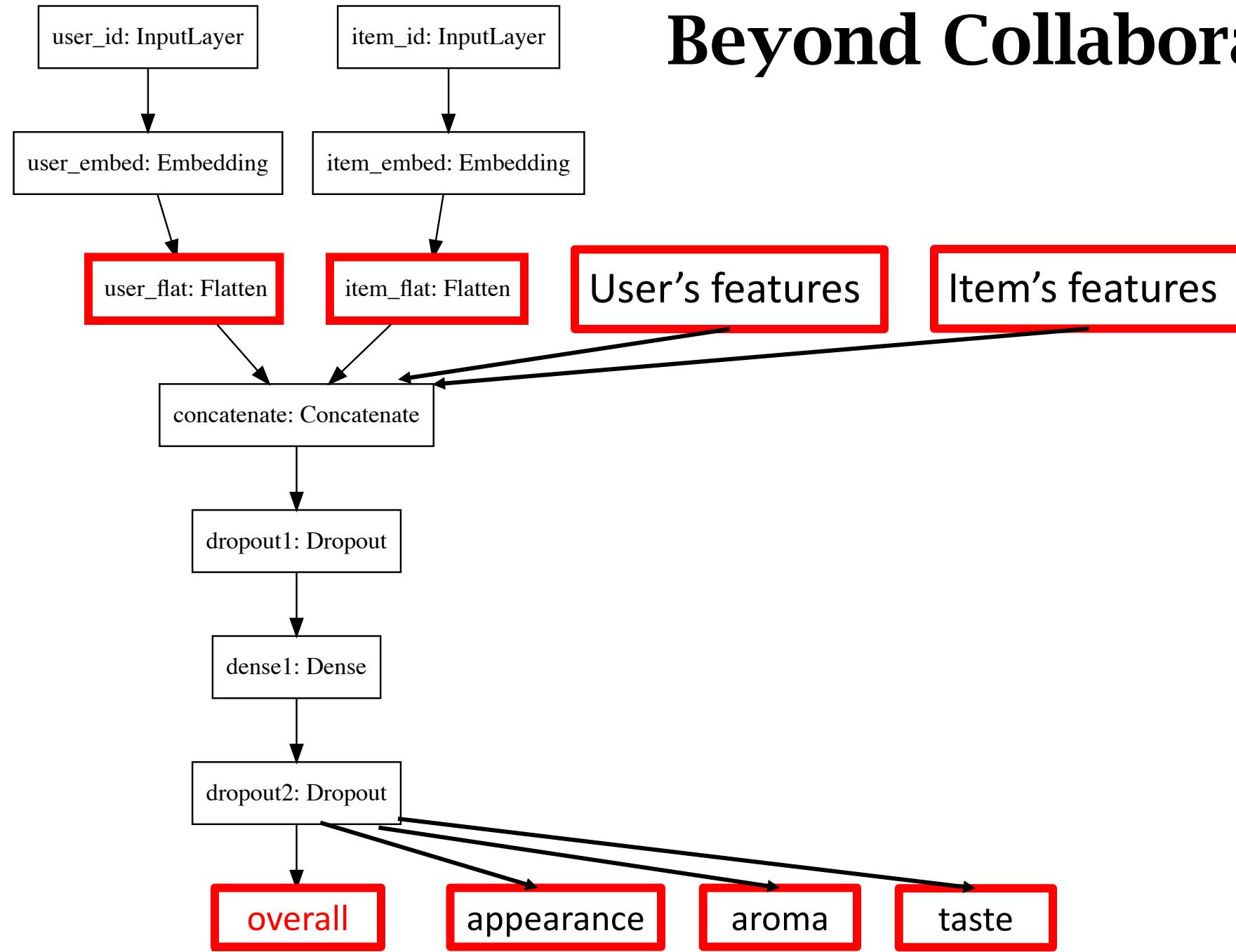
```
'beer/beerId'  
'beer/brewerId'  
'beer/name'  
'beer/style'  
'user/ageInSeconds'  
'user/birthdayRaw'  
'user/birthdayUnix'  
'user/gender'  
'user/userId'  
'review/appearance'  
'review/aroma'  
'review/overall'  
'review/taste'  
'review/text'  
'review/timeStruct'  
'review/timeUnix'
```

# Beyond Collaborative Filtering



'beer/beerId'  
'beer/brewerId'  
'beer/name'  
'beer/style'  
'user/ageInSeconds'  
'user/birthdayRaw'  
'user/birthdayUnix'  
'user/gender'  
**'user/userId'**  
'review/appearance'  
'review/aroma'  
**'review/overall'**  
'review/taste'  
'review/text'  
'review/timeStruct'  
**'review/timeUnix'**

# Beyond Collaborative Filtering



'beer/beerId'  
'beer/brewerId'  
'beer/name'  
'beer/style'  
'user/ageInSeconds'  
'user/birthdayRaw'  
'user/birthdayUnix'  
'user/gender'  
**'user/userId'**  
'review/appearance'  
'review/aroma'  
**'review/overall'**  
'review/taste'  
'review/text'  
'review/timeStruct'  
**'review/timeUnix'**