# Building a Recommender System Using Drupal and TensorFlow.js

Adapted from *Build, Train, and Deploy a Book Recommender System Using Keras, TensorFlow.js, Node.js, and Firebase (Part 1)*, https://heartbeat.fritz.ai/build-train-and-deploy-a-book-recommender-system-using-keras-tensorflow-js-b96944b936a7. Instead of using Node.js and Firebase, I will be using Drupal 9, the Component module, and the TensorFlow.js module.

The loading of the data, preprocessing the data, model building and model training will be done using Colab (or Jupyter Notebook), and the model will be exported and used by Drupal to do inference using TensorFlow.js.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import warnings

import tensorflow.keras as tf
```

Read the data downloaded from Zygmuntz's Github repository

```
ratings_df = pd.read_csv("book-data/ratings.csv")
books_df = pd.read_csv("book-data/books.csv")
```

Display the first few lines of the ratings csv file.

```
ratings_df.head()
```

| | user_id | book_id | rating |
|---|---|---|---|
| **0** | 1 | 258 | 5 |
| **1** | 2 | 4081 | 4 |
| **2** | 2 | 260 | 5 |
| **3** | 2 | 9296 | 5 |
| **4** | 2 | 2318 | 3 |

```
books_df.head()
```

| | book_id | goodreads_book_id | best_book_id | work_id | books_count | isbn |
|---|---|---|---|---|---|---|
| **0** | 1 | 2767052 | 2767052 | 2792775 | 272 | 439023483 |
| **1** | 2 | 3 | 3 | 4640799 | 491 | 439554934 |
| **2** | 3 | 41865 | 41865 | 3212258 | 226 | 316015849 |
| **3** | 4 | 2657 | 2657 | 3275794 | 487 | 61120081 |
| **4** | 5 | 4671 | 4671 | 245494 | 1356 | 743273567 |

```
print(ratings_df.shape)
print(ratings_df.user_id.nunique())
print(ratings_df.book_id.nunique())
```

```
print(ratings_df.book_id.nunique())
ratings_df.isna().sum()
```

```
(5976479, 3)
53424
10000
user_id    0
book_id    0
rating     0
dtype: int64
```

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest = train_test_split(ratings_df, test_size=0.2, random_state=1)
print(f"Shape of train data: {Xtrain.shape}")
print(f"Shape of test data: {Xtest.shape}")
```

```
Shape of train data: (4781183, 3)
Shape of test data: (1195296, 3)
```

```
#Get the number of unique entities in books and users columns
nbook_id = ratings_df.book_id.nunique()
nuser_id = ratings_df.user_id.nunique()
```

```
#Book input network
input_books = tf.layers.Input(shape=[1])
embed_books = tf.layers.Embedding(nbook_id + 1,15)(input_books)
books_out = tf.layers.Flatten()(embed_books)

#user input network
input_users = tf.layers.Input(shape=[1])
embed_users = tf.layers.Embedding(nuser_id + 1,15)(input_users)
users_out = tf.layers.Flatten()(embed_users)

conc_layer = tf.layers.Concatenate()([books_out, users_out])
x = tf.layers.Dense(128, activation='relu')(conc_layer)
x_out = x = tf.layers.Dense(1, activation='relu')(x)
model = tf.Model([input_books, input_users], x_out)
```

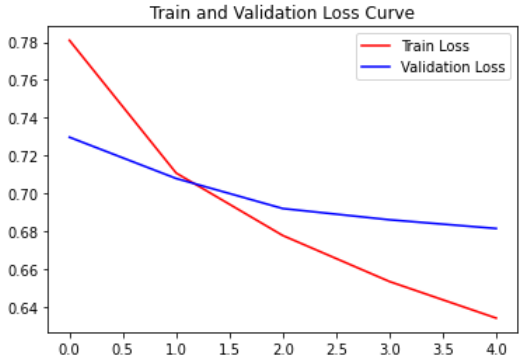
```
opt = tf.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='mean_squared_error')
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 1, 15)        150015      input_1[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 15)        801375      input_2[0][0]
__________________________________________________________________________________________________
flatten (Flatten)               (None, 15)           0           embedding[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 15)           0           embedding_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 30)           0           flatten[0][0]
                                                                 flatten_1[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          3968        concatenate[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            129         dense[0][0]
==================================================================================================
Total params: 955,487
Trainable params: 955,487
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
hist = model.fit([Xtrain.book_id, Xtrain.user_id], Xtrain.rating,
                 batch_size=64,
                 epochs=5,
                 verbose=1,
                 validation_data=([Xtest.book_id, Xtest.user_id], Xtest.rating))
```

```
Epoch 1/5
74706/74706 [==============================] - 658s 9ms/step - loss: 0.7809 - val_loss: 0.7297
Epoch 2/5
74706/74706 [==============================] - 674s 9ms/step - loss: 0.7108 - val_loss: 0.7079
Epoch 3/5
74706/74706 [==============================] - 667s 9ms/step - loss: 0.6777 - val_loss: 0.6920
Epoch 4/5
74706/74706 [==============================] - 657s 9ms/step - loss: 0.6534 - val_loss: 0.6861
Epoch 5/5
74706/74706 [==============================] - 646s 9ms/step - loss: 0.6341 - val_loss: 0.6814
```

```
train_loss = hist.history['loss']
val_loss = hist.history['val_loss']
plt.plot(train_loss, color='r', label='Train Loss')
plt.plot(val_loss, color='b', label='Validation Loss')
plt.title("Train and Validation Loss Curve")
plt.legend()
plt.show()
```




Save the model. This saves the model as a Tensorflow / Keras model. Note this format, as you'll be referencing it during model conversion in the next tutorial.

```
#save the model
model.save('model')
```

```
INFO:tensorflow:Assets written to: model/assets
```

✓ 0s completed at 2:04 AM