```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Embedding, Input, Flatten
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
```

```
# Load and view the dataset
data = pd.read_csv('/content/movie_night_utilitymatrix_F24.cs
```

```
# Initialize label encoders
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()

# Encode user and movie IDs
data['user'] = user_encoder.fit_transform(data['user'])
data['movie'] = movie_encoder.fit_transform(data['movie'])

# Split the data into training and validation sets
train_data, val_data = train_test_split(data, test_size=0.2,
```

```
# Normalize the ratings to a range between 0 and 1
data['rating_norm'] = data['rating'] / data['rating'].max()

# Split the data into training and validation sets
train_data, test_data = train_test_split(data, test_size=0.2,

# Prepare the arrays for the model input and output
X_train_array = [train_data['user'].values, train_data['movie
y_train = train_data['rating_norm'].values
```

```
X_test_array = [test_data['user'].values, test_data['movie'].
y_test = test_data['rating_norm'].values
```

```
def GMFact(in_u_dim=510, in_m_dim=20, latent_out_dim=50):

    # input layers(sparse)
    user = Input(name='u_in', shape=[1])  # User input
    movie = Input(name='m_in', shape=[1])  # Movie input

    # Embedding layers
    user_embedding = Embedding(name='u_emb',
                               input_dim = in_u_dim,  #what is
                               output_dim = latent_out_dim)(us
    movie_embedding = Embedding(name='s_emb',
                                input_dim=in_m_dim,
                                output_dim=latent_out_dim)(mov
    #Element-wise Multiply layer
    x = tf.keras.layers.Multiply()([user_embedding, movie_embe
    x = Flatten()(x)

    # Single neuron layer
    x = Dense(1, kernel_initializer='lecun_uniform')(x)  #add
    x = Activation("sigmoid")(x)  # Sigmoid activation

    # model creation
    model = Model(inputs=[user, movie], outputs=x)

    # compile the model
    model.compile(
        optimizer='sgd',
        loss='mse',
        metrics=[tf.keras.metrics.RootMeanSquaredError()])

    return model

model = GMFact()
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape
=================================================================
 u_in (InputLayer)           [(None, 1)]

 m_in (InputLayer)           [(None, 1)]

 u_emb (Embedding)           (None, 1, 50)

 s_emb (Embedding)           (None, 1, 50)

 multiply (Multiply)         (None, 1, 50)


 flatten (Flatten)           (None, 50)

 dense (Dense)               (None, 1)

 activation (Activation)     (None, 1)

=================================================================
Total params: 26551 (103.71 KB)
Trainable params: 26551 (103.71 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# train the model
batch_size = 50
history = model.fit(
    x=X_train_array,
    y=y_train,
    batch_size=batch_size,
    epochs=1,
    verbose=1,
    validation_data=(X_test_array, y_test)
)

# save the model weights
model.save_weights('gmf.h5')
```

```
237/237 [==============================] - 1s 3ms/step -
```

```
final_RMSE = (history.history['root_mean_squared_error'])[-1
final_RMSE
```

```
0.32662439346313477
```

Start coding or generate with AI.