Тихоновс Василийс,
Выпускник Санкт-Петербургского политехнического университета
Петра Великого

Сечинский Егор Валерьевич,
Выпускник Санкт-Петербургского политехнического университета
Петра Великого

ПРИМЕНЕНИЕ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ПОСТРОЕНИЯ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

Аннотация: в статье отображены результаты исследования рекомендательного алгоритма на основе сетей глубокого обучения. В исследовании оценивается оптимальность данной системы.

Ключевые слова: рекомендационная система, оценка, фильтрация, прогноз, машинное обучение, RNN, нейросеть, основа ная на содержимом, гибридная фильтрация, PYTHON.

ВВЕДЕНИЕ

Благодаря гибкости входного слоя сети, DNN легко может включать характеристики запросов и характеристики элементов, что помогает уловить конкретные интересы пользователя и улучшить релевантность рекомендаций.

В данной работе использовались DNN с Softmax-слоем для рекомендации фильмов. Пользователи и фильмы представлены в виде векторов, закодированных с помощью one-hot-кодирования, и подаются на вход глубокой нейронной сети в качестве отдельных и различных входных данных, а рейтинги предоставляются в качестве выходных данных.

Рекомендательные сети на основе сетей глубокого обучения

Задача прогнозирования релевантного набора фильмов – это задача с последовательным набором данных. Одной из самых популярных архитектур глубокого обучения для работы с такими данным является рекурентная нейронная сеть (RNN). RNN показывают себя лучше, чем классические подходы глубокого обучения, благодаря последовательной архитектуре. RNN отличаются от других нейронных сетей тем, что реализуют концепцию памяти, благодаря которой сохраняется информация о предыдущих входах.

Алгоритм работы RNN представлен на рисунке 1. Каждый слой генерирует ответ y_t, в определенный момент времени t, получая информацию из предыдущего состояния h_{t-1}.

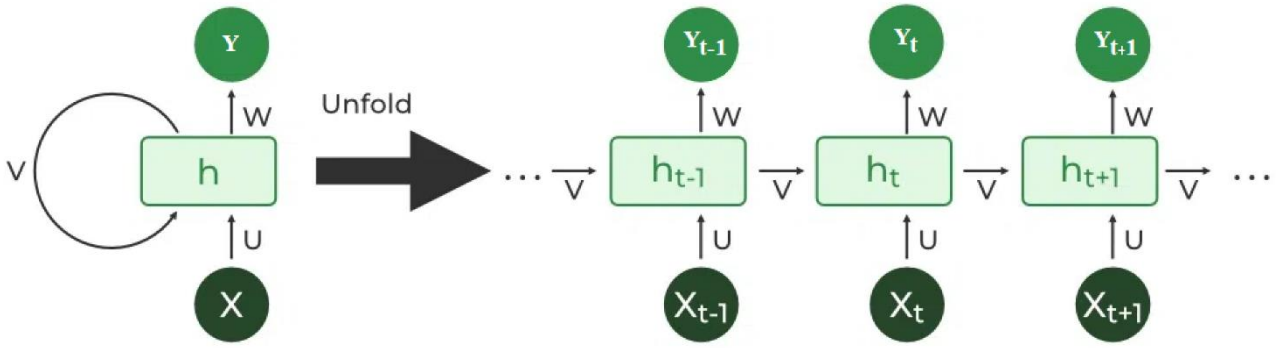

Рисунок 1 – Алгоритм работы сети

Формально RNN можно описать таким образом:

$$h_t = f(Vx_t + Wh_{t-1} + b), \quad (1)$$

где h_t – скрытое состояние в момент времени t, f – нелинейная функция активации, V – матрица весов, связанных с входными данными, W – матрица весов, связанных со скрытым состоянием, b – смещение.

Выход нейронной сети в момент времени t равен:

$$\hat{y}_t = g\left(Uh_t + \hat{b}\right), \quad (2)$$

где g – функция активации, U – матрица весов, связанных с выходным состоянием.

При обучении рекуррентной нейронной сети используется метод градиентного спуска в сочетании с алгоритмом обратного распространения ошибки во времени для определения градиентов. Этот подход отличается от классического тем, что подразумевает суммирование ошибки на каждом шаге, в то время как в традиционных нейронных сетях нет необходимости суммировать ошибки, поскольку они не используют общие параметры на каждом уровне.

Функции активации вносят нелинейность в нейроны, а также вляют на выходное значение.

Функция активации сигмоида представляет собой плавно возрастающую функцию:

$$\sigma(z) = \frac{1}{1 - e^{-z}}. \quad (3)$$

На рисунке 2 представлен график функции сигмоида:

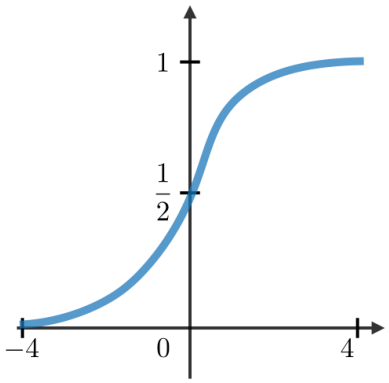

Рисунок 2 – Сигмоидная функции

Основное преимущество использования сигмоидной функции состоит в том, что она приводит выходные значения к нормализованному виду.

Гиперболический тангенс представляет собой модифицированную функцию сигмоида.

$$\sigma(z) = \frac{2}{1 + e^{-2z}} - 1. \quad (4)$$

На рисунке 3 показан график функции гиперболического тангенса:

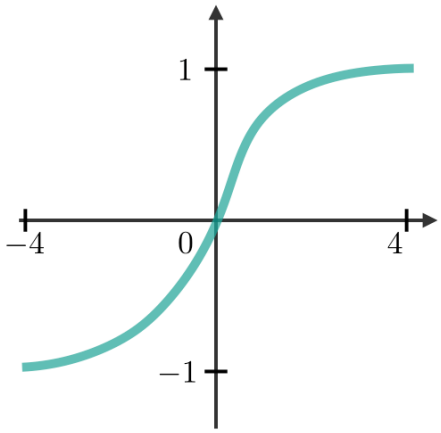

Рисунок 3 – Функция гиперболического тангенса

Функция гиперболического тангенса применяется, когда нормализация выходных значений не требуется, так как её область значений (-1, 1) обеспечивает большую амплитуду для алгоритма градиентного спуска.

Функция ReLU возвращает ноль, если входное значение отрицательно, иначе она возвращает само входное значение.

$$\mathrm{ReLU}(z) = \max(0, z). \tag{5}$$

На рисунке 4 показан график функции ReLU:

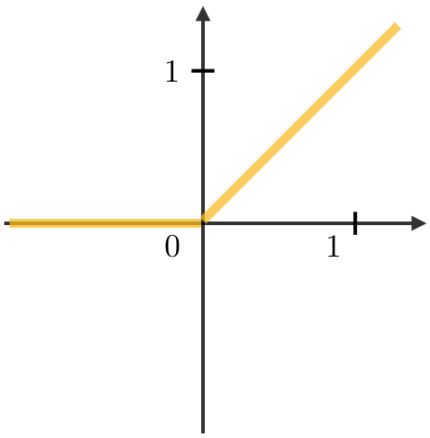

Рисунок 4 – Функция ReLU

Преимущество этой функции заключается в возможности применения для быстрого вычисление производной.

Необходимо разделить данные на обучающую и тестовую выборку

```
X = refined_dataset[['user', 'movie']].values
y = refined_dataset['rating'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=50)
X_train.shape, X_test.shape, y_train.shape, y_test.shape.
```

Далее опишем структуру сети:

– Инициализация входного слоя: user = tf.keras.layers.Input(shape = (1,));

– Инициализация входного слоя для фильмов: movie = tf.keras.layers.Input(shape = (1,));

– Определим число факторов, которые будут рассматриваться встраиваемым слоем:

```
u = keras.layers.embeddings.Embedding(n_users, n_factors, embeddings_initializer = 'he_normal', embeddings_regularizer = tf.keras.regularizers.l2(1e-6))(user)
u = tf.keras.layers.Reshape((n_factors,))(u);
```

– Выполним сложение встраиваний пользователей и фильмов:

```
x = tf.keras.layers.Concatenate()([u,m])
x = tf.keras.layers.Dropout(0.05)(x);
```

– Добавляем слои в архитектуру:

```
x = tf.keras.layers.Dense(32, kernel_initializer='he_normal')(x)
x = tf.keras.layers.Activation(activation='relu')(x)
x = tf.keras.layers.Dropout(0.05)(x)
x = tf.keras.layers.Dense(16, kernel_initializer='he_normal')(x)
x = tf.keras.layers.Activation(activation='relu')(x)
x = tf.keras.layers.Dropout(0.05)(x);
```

– Выходной слой:

```
x = tf.keras.layers.Dense(9)(x)
x = tf.keras.layers.Activation(activation='softmax')(x).
```

Далее можно скомпилировать модель нейросети (рисунок 5).

```
model.compile(optimizer='sgd', loss=tf.keras.losses.SparseCategoricalCrossentropy(), metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "functional_5"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 1, 150)       141450      input_5[0][0]
__________________________________________________________________________________________________
embedding_5 (Embedding)         (None, 1, 150)       249600      input_6[0][0]
__________________________________________________________________________________________________
reshape_4 (Reshape)             (None, 150)          0           embedding_4[0][0]
__________________________________________________________________________________________________
reshape_5 (Reshape)             (None, 150)          0           embedding_5[0][0]
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 300)          0           reshape_4[0][0]
                                                                 reshape_5[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 300)          0           concatenate_2[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 32)           9632        dropout_6[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 32)           0           dense_6[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 32)           0           activation_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 16)           528         dropout_7[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 16)           0           dense_7[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 16)           0           activation_7[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 9)            153         dropout_8[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, 9)            0           dense_8[0][0]
==================================================================================================
Total params: 401,363
Trainable params: 401,363
Non-trainable params: 0
```

Рисунок 5 – Компиляция модели

Зависимость loss от количества эпох представлена на рисунке 6.

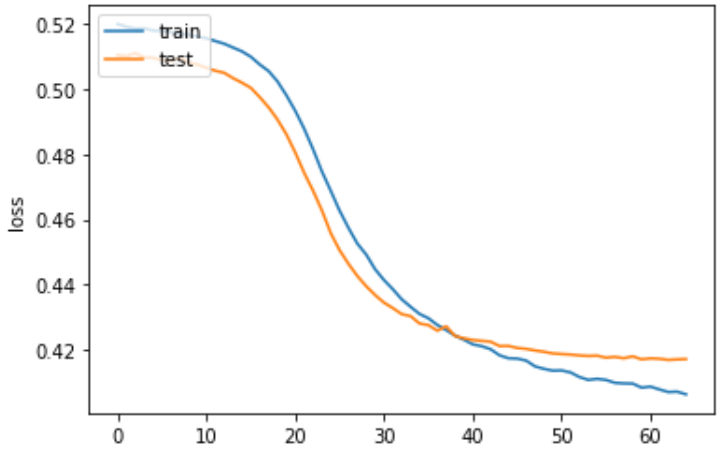

Рисунок 6 – Зависимость потери от количества эпох

Полученная модель имеет два входа, один из входов содержит идентификаторы пользователей, а другой - соответствующие идентификаторы фильмов. Модель пытается предсказать оценки комбинации пользователь-фильм. Таким образом, можно ввести конкретный идентификатор пользователя и идентификатор непросмотренного фильма и ожидать, что модель предскажет оценки фильмов, которые бы соответствовали оценками, схожими с оценками пользователя. В данном случае оценки уже нормализованы, и поскольку нам нужны фильмы, которые больше интересуют пользователя, оценки не возвращаются к диапазону 0 – 5.

Результат применения модели для прогнозирования оценок непросмотренных фильмов показан на рисунке 7.

```
predicted_ratings = model.predict(model_input)
```

```
print(predicted_ratings.shape)
```

```
(1628, 9)
```

```
print(predicted_ratings)
```

```
[[6.28711879e-01 3.71125787e-01 1.93846718e-05 ... 2.48171236e-05
  2.07571484e-05 3.11595759e-05]
 [5.16196430e-01 4.83636826e-01 2.06636632e-05 ... 2.40022491e-05
  2.14833890e-05 3.09596326e-05]
 [8.92104924e-01 1.07851624e-01 4.90856564e-06 ... 8.67088238e-06
  4.84645898e-06 9.25974837e-06]
 ...
 [6.53564811e-01 3.46285373e-01 1.90432311e-05 ... 2.25746426e-05
  1.92296520e-05 2.91551714e-05]
 [5.77207983e-01 4.22657400e-01 1.69154791e-05 ... 1.97136451e-05
  1.75365039e-05 2.56826916e-05]
 [6.86957419e-01 3.12936485e-01 1.36032540e-05 ... 1.61753997e-05
  1.37795878e-05 2.07246703e-05]]
```

Рисунок 7 – Прогноз оценок

Полученный вывод – это вероятность каждой возможной оценки пользователем от 1 до 5 для каждого фильма. Можно извлечь конкретную оценку, которую пользователь мог бы дать фильму, и отсортировать их в порядке убывания. Далее остается просто вывести список фильмов (рисунок 8).

```
pprint(list(recommended_movies[:20]))
```

```
['Sword in the Stone, The (1963)',
 'Baton Rouge (1988)',
 'Meet Wally Sparks (1997)',
 'Grosse Fatigue (1994)',
 'In the Line of Duty 2 (1987)',
 'Conspiracy Theory (1997)',
 'Red Firecracker, Green Firecracker (1994)',
 'Striking Distance (1993)',
 'Two or Three Things I Know About Her (1966)',
 'Phat Beach (1996)',
 'Diva (1981)',
 'Getaway, The (1994)',
 'Jaws 2 (1978)',
 'Welcome to the Dollhouse (1995)',
 'Basic Instinct (1992)',
 'Saint, The (1997)',
 'Critical Care (1997)',
 'Jude (1996)',
 'Mediterraneo (1991)',
 'Month by the Lake, A (1995)']
```

Рисунок 8 – Прогноз оценок

Для определения качества предсказанных оценок воспользуемся мерой RMSE. Для этого нужно написать функцию оценки (рисунок 9).

```
def rmse(prediction, ground_truth):
    # Оставим оценки, предсказанные алгоритмом, только для соотвествующего набора данных
    prediction = np.nan_to_num(prediction)[ground_truth.nonzero()].flatten()
    # Оставим оценки, которые реально поставил пользователь, только для соотвествующего набора данных
    ground_truth = np.nan_to_num(ground_truth)[ground_truth.nonzero()].flatten()

    mse = mean_squared_error(prediction, ground_truth)
    return sqrt(mse)
```

Рисунок 9 – Функция для оценки

Здесь использована встроенная в библиотеку sklearn функция mean_squared_error. Оценка точности проведена на тестовой выборке, которая была построена из части исходного датасета. Для проверки работы реализованных методов расчитывается оценка релевантности рекомендации. Данный алгоритм показал ошибку 1.22

Заключение

В статье описан один из методов реализации рекомендательных систем – RNN. Приведен пример разработки алгоритма и его реализация. Оценка по метрике RMSE демонстрирует высокую точность данного подхода.

Список литературы:

1. J. Hopcroft, R. Kannan Foundations of Data Science: учебное пособие // J. Hopcroft, R. Kannan - Cambridge University Press, 2020.
2. Козлов В. Н. Системный анализ, оптимизация и принятие решений: учебное пособие // В. Н. Козлов. – Санкт- Петербург: Издательство Политехнического университета, 2011.
3. Liu et al., 2011 - "Personalized news recommendation based on click behavior": A survey
4. G. Linden. IEEE Internet Computing: Amazon.com recommendations: item-to-item collaborative filtering // G. Linden, B. Smith, J. York. – New York, 2003.
5. J. Bobadilla. Recommender systems survey // J. Bobadilla, F. Ortega, A. Hernando, A. GutieRrez. – Canada, 2013.
6. S. Zhang. Deep learning based recommender system: A survey and new perspectives // S. Zhang, L. Yoa, A. Sun, Y. Tay. – Nayang: ACM Computing Survey, 2019.
7. J. Davidson. The YouTube video recommendation system // J. Davidson, B. Liebald, J. Liu, P. Nandy, Taylor V. Vleet. - Proceedings of the fourth ACM conference on Recommender systems. – Barcelona, 2010.