DOI 10.37882/2223-2966.2024.05.14

ИССЛЕДОВАНИЕ АРХИТЕКТУРЫ GRU В КОНТЕКСТЕ PYTHON: ПОДХОДЫ И ПРИМЕНЕНИЯ

GRU ARCHITECTURE RESEARCH IN THE CONTEXT OF PYTHON: APPROACHES AND APPLICATIONS

H. Istamqulov

Summary. This article explores the analysis of Gated Recurrent Unit (GRU) networks using Python, focusing on their application in deep learning for processing sequential data. It begins with an overview of recurrent neural networks (RNNs) and the rationale behind GRUs. The article then delves into the architecture of GRUs, illustrating their unique features such as reset and update gates. It provides a practical guide to implementing GRU networks in Python with popular libraries like TensorFlow and Keras, including sample code for building and evaluating models. The article also compares GRUs with other RNN variants, highlighting their advantages in various scenarios. In conclusion, it underscores the significance of GRUs in handling temporal dependencies and their potential in future deep learning research.

Keywords: Gated Recurrent Unit (GRU), Recurrent Neural Networks (RNN), Python, Deep learning, Temporal dependencies, TensorFlow, Keras, Model evaluation, Neural network architecture.

Истамкулов Хасанжон Саиджонович
К.т.н., Худжандский Государственный Университет им. Ак. Б. Гафурова, Таджикистан
istamqulov@gmail.com

Аннотация. В этой статье рассматривается анализ сетей Gated Recurrent Unit (GRU) с помощью Python с упором на их применение в глубоком обучении для обработки последовательных данных. Статья начинается с обзора рекуррентных нейронных сетей (РНС) и обоснования GRU. Затем статья углубляется в архитектуру GRU, иллюстрируя их уникальные особенности, такие как ворота сброса и обновления. В статье представлено практическое руководство по реализации GRU-сетей на Python с помощью популярных библиотек TensorFlow и Keras, включая примеры кода для построения и оценки моделей. В заключение подчеркивается важность GRU для работы с временными зависимостями и их потенциал в будущих исследованиях в области глубокого обучения.

Ключевые слова: Gated Recurrent Unit (GRU), Рекуррентные нейронные сети, RNN, Python, глубокое обучение, последовательная обработка данных, TensorFlow, Keras, оценка моделей, архитектура нейронных сетей.

Введение

В последние годы в области глубокого обучения наблюдаются значительные успехи, особенно в области обработки последовательных данных. Рекуррентные нейронные сети (RNN, РНС рус.) стали мощным инструментом для обработки данных временных рядов, обработки естественного языка и других задач, связанных с временными зависимостями. Однако традиционные RNN часто сталкиваются с проблемой исчезающего градиента, что препятствует их способности к изучению дальних зависимостей в последовательностях данных.

Для решения этой проблемы были созданы сети с управляемыми рекуррентными блоками (Gated Recurrent Unit, GRU), которые являются разновидностью RNN. В GRU встроены механизмы регулировки потока информации, что позволяет им сохранять релевантную информацию в длинных последовательностях данных и смягчать проблему исчезающего градиента. Это делает GRU привлекательным выбором для широкого спектра приложений, от распознавания речи до прогнозирования временных рядов.

Язык программирования Python с его богатой экосистемой библиотек и фреймворков, таких как TensorFlow, Keras и PyTorch, стал популярным выбором для реализации и анализа сетей GRU. Простота и гибкость Python позволяют исследователям и практикам эффективно строить и экспериментировать с моделями GRU.

Цель данной статьи — провести глубокий анализ GRU-сетей, подчеркнуть их архитектуру, преимущества и практическую реализацию с помощью Python. Исследуя нюансы сетей GRU, мы стремимся пролить свет на их потенциал и помочь читателям использовать эти модели для решения своих собственных задач последовательной обработки данных.

Описание модели GRU

Сети GRU, представленные в 2014 году, представляют собой упрощенную версию сетей LSTM, которая сохраняет возможность моделирования долгосрочных зависимостей при снижении вычислительной сложности. Ячейка GRU характеризуется двумя фильтрами: фильтрами обновления и фильтрами сброса. Фильтры обновления контролируют степень переноса предыдущего скрытого состояния, а фильтры сброса определяют, какую часть прошлой информации следует забыть. Математически ячейка GRU может быть представлена следующим образом [1]:

$$z_t = \sigma_g\left(W_z x_t + U_z h_{t-1} + b_z\right)$$
$$r_t = \sigma_g\left(W_r x_t + U_r h_{t-1} + b_r\right)$$
$$h_t = z_t \circ h_{t-1} + \left(1 - z_t\right) \circ \sigma_h\left(W_h x_t + U_h\left(r_t \circ h_{t-1}\right) + b_h\right)$$

где x_t — Входной вектор;
h_t — Выходной вектор;
z_t — вектор вентиля обновления;
r_t — вектор вентиля сброса;
W, U, и b — матрицы параметров и вектор;
σ_g — Функция активации на основе сигмоида;
σ_h — Функция активации на основе гиперболического тангенса.

Ниже приведено визуальное описание структуры ячейки GRU сети:

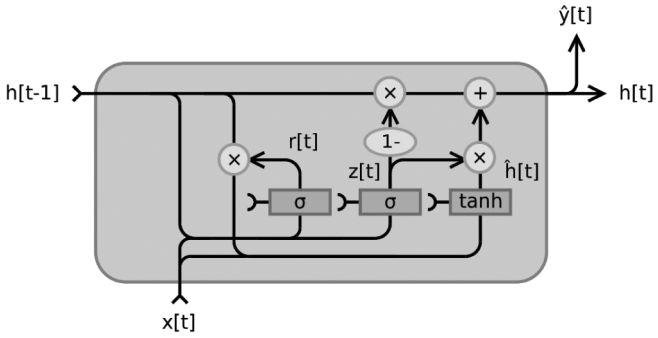

Рис. 1. Схема работы ячейки GRU

1. Входной слой: Входной слой — это первая точка соприкосновения входных данных в сети GRU. В контексте обработки естественного языка входные данные обычно представляют собой последовательность слов или лексем. Эти лексемы обычно преобразуются в плотные векторные представления, известные как вкрапления слов, которые отражают семантическое значение слов. Входной слой получает эти вкрапления и передает их в слой GRU для дальнейшей обработки. Размерность входного слоя соответствует размеру вкраплений слов.
2. Слой(и) GRU: Слой GRU — это основной компонент сети, в котором происходит последовательная обработка. Блок GRU — это усовершенствованный вариант стандартной ячейки рекуррентной нейронной сети (RNN), разработанный для решения проблемы исчезающего градиента, часто встречающейся в традиционных RNN. Блок GRU состоит из двух вентилей: вентиля обновления и вентиля сброса.

- Вентиль обновления контролируют степень переноса информации из предыдущего состояния в текущее. Он помогает модели решить, какой объем прошлой информации следует сохранить для будущих вычислений. Это очень важно для улавливания долгосрочных зависимостей в данных.
- Вентиль сброса: Ворота сброса определяют, какая часть прошлой информации должна быть забыта. Это позволяет модели отбросить неактуальную информацию и сосредоточиться на наиболее важных деталях для текущего прогноза.

Слой GRU может быть сложен, то есть несколько слоев GRU могут быть размещены друг над другом, чтобы увеличить способность модели улавливать сложные закономерности в данных. Выход каждого блока GRU передается следующему блоку в последовательности, а также следующему слою в стеке, если таковой имеется [2].

3. Выходной слой: Выходной слой отвечает за получение конечного результата работы сети. Характер выходного слоя зависит от конкретной задачи, для выполнения которой предназначена сеть GRU. Например, в задаче классификации выходной слой может состоять из активационной функции ***softmax***, которая генерирует распределение вероятностей по возможным классам. В задаче регрессии выходной слой может состоять из линейной функции активации, которая генерирует непрерывный выход. В задаче генерации последовательности выходной слой может генерировать последовательность лексем или слов, часто используя функцию ***softmax*** на каждом временном шаге.

Архитектура сети GRU позволяет ей эффективно обрабатывать последовательные данные, что делает ее пригодной для широкого спектра приложений, включая моделирование языка, машинный перевод, распознавание речи и прогнозирование временных рядов. Способность GRU улавливать долгосрочные зависимости и обрабатывать входные последовательности переменной длины делает ее мощным инструментом для моделирования последовательных данных [3].

Процесс обучения сети GRU включает в себя несколько этапов и гиперпараметров:

1. **Гиперпараметры**: это параметры, которые задаются перед началом процесса обучения и включают в себя количество слоев GRU, количество единиц в каждом слое GRU, размер вкраплений слов, скорость обучения, размер партии и количество эпох.
2. **Метод оптимизации**: это метод, используемый для обновления весов сети во время обучения. К распространенным методам оптимизации относятся стохастический градиентный спуск (SGD), Adam, RMSprop и Adagrad. Каждый из этих оптимизаторов имеет свои преимущества и выбирается в зависимости от особенностей данных и задачи.
3. **Функция потерь**: это функция, которая измеряет разницу между прогнозируемым выходом сети и фактическим целевым выходом. Выбор функции

потерь зависит от типа задачи. Например, в задаче классификации обычно используются потери от перекрестной энтропии, в то время как в задаче регрессии более подходящей может быть средняя квадратичная ошибка.

4. **Процесс обучения**: Процесс обучения включает в себя подачу входных данных в сеть, вычисление потерь с помощью функции потерь, а затем обновление весов сети с помощью техники оптимизации. Этот процесс повторяется в течение определенного количества эпох или до тех пор, пока потери не придут к минимальному значению.
5. **Оценка**: после обучения сети она оценивается на отдельном проверочном или тестовом наборе для определения ее производительности. Общие метрики оценки включают точность, прецизионность, отзыв и F1 оценка для задач классификации, а также среднюю абсолютную ошибку или среднюю квадратичную ошибку для задач регрессии.

Тщательно настроив эти гиперпараметры и выполнив процесс обучения, можно эффективно обучить сеть GRU для выполнения широкого спектра задач с последовательными данными, таких как классификация текстов, языковое моделирование и машинный перевод.

Создание сети GRU на языке Python

Есть два варианта создания сетей GRU — с нуля и на основе готовой модели.

Создание GRU-сети с нуля на Python без использования готовых моделей, как в TensorFlow или PyTorch, — сложная задача, которая обычно не делается, в угоду использования готовых решение. Для создания модели GRU нужно будет создать сущности ячеек GRU, функций активации, и логики перехода между слоями:

```
import numpy as np

def sigmoid(x):
return 1 / (1 + np.exp(-x))

def tanh(x):
return np.tanh(x)

class GRUCell:
def __init__(self, input_size, hidden_size):
self.input_size = input_size
self.hidden_size = hidden_size

 # Initialize weights
 self.Wz = np.random.randn(hidden_size, input_size)
 self.Wr = np.random.randn(hidden_size, input_size)
 self.Wh = np.random.randn(hidden_size, input_size)
 self.Uz = np.random.randn(hidden_size, hidden_size)
 self.Ur = np.random.randn(hidden_size, hidden_size)
 self.Uh = np.random.randn(hidden_size, hidden_size)
 self.bz = np.zeros((hidden_size, 1))
 self.br = np.zeros((hidden_size, 1))
 self.bh = np.zeros((hidden_size, 1))

 def forward(self, x, h_prev):
 z = sigmoid(np.dot(self.Wz, x) + np.dot(self.Uz, h_prev) + self.bz)
 r = sigmoid(np.dot(self.Wr, x) + np.dot(self.Ur, h_prev) + self.br)
 h_tilde = tanh(np.dot(self.Wh, x) + r * np.dot(self.Uh, h_prev) + self.bh)
 h_next = (1 — z) * h_tilde + z * h_prev
 return h_next
```

В этом примере класс GRUCell определяет одну ячейку GRU с необходимыми весами и смещениями. Метод forward принимает входной вектор x и предыдущее скрытое состояние *h_prev*, и вычисляет следующее скрытое состояние *h_next*, используя уравнения GRU [8]. Функции активации *sigmoid* и *tanh* созданы на основе математических функций пакета *numpy*.

Объекты класса GRUCell могут быть использованы как компоненты сети GRU. Для этого будет необходимо создать класс модели GRU:

```
class GRUNetwork:
 def __init__(self, input_size, hidden_size, output_size):
 self.input_size = input_size
 self.hidden_size = hidden_size
 self.output_size = output_size

 # Создание
 self.gru_cell = GRUCell(input_size, hidden_size)

 # Веса слоев и базисас
 self.Wo = np.random.randn(output_size, hidden_size)
 self.bo = np.zeros((output_size, 1))

 def forward(self, input_sequence):
 h = np.zeros((self.hidden_size, 1))

 for x in input_sequence:
 h = self.gru_cell.forward(x, h)

 output = np.dot(self.Wo, h) + self.bo
 return output

# Пример использования сети GRUNetwork:
input_size = 5
hidden_size = 32 # количество юнитов GRU
output_size = 2
sequence_length = 10
gru_network = GRUNetwork(input_size, hidden_size, output_size)

# Входные данные
input_sequence = [np.random.randn(input_size, 1) for _ in range(sequence_length)]
```

```
# Процесс обработки сети
output = gru_network.forward(input_sequence)
print(output)
```

На момент написание данной статьи, почти все известные библиотеки в сфере ИИ, имеют готовые к использованию модели GRU с настроенной логикой. Ниже приведен пример создания GRU сети на основе Keras [5]:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense

# Настройка параметров модели
vocab_size = 10000 # Размер словаря
embedding_dim = 64 # Размерность вектора
gru_units = 32 # Количество юнитов в слое GRU
max_length = 100 # Максимальная длина входных данных
num_classes = 2 # Количество классов вывода

# Создание GRU модели
model = Sequential([
Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length),
 GRU(units=gru_units),
 Dense(units=num_classes, activation='softmax')
])

# Компиляция модели
model.compile(
optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']
)

# Суммаризация модели
model.summary()
```

При построении сети GRU с нуля, как показано в первом примере, необходимо глубокое понимание математических операций и структуры ячеек GRU. Такой подход обеспечивает высокую степень гибкости и позволяет настраивать архитектуру GRU в соответствии с конкретными требованиями. Однако он также требует тщательного управления весами и смещениями сети, а также тщательной реализации алгоритма прямого распространения для обеспечения правильного вычисления скрытых состояний и выхода.

С другой стороны, использование готовых GRU-моделей из библиотек глубокого обучения, таких как Keras, как показано во втором примере, значительно упрощает процесс создания и обучения GRU-сетей. Эти библиотеки предоставляют предопределенные GRU-слои с оптимизированными реализациями, которые можно легко интегрировать в модель нейронной сети [6]. Такой подход не только сокращает время разработки, но и гарантирует, что сеть GRU будет использовать последние оптимизации и улучшения в базовой библиотеке.

При выборе между созданием сети GRU с нуля или использованием готовой модели важно учитывать такие факторы, как уровень необходимой настройки, сложность задачи и доступные вычислительные ресурсы. Для большинства практических приложений использование готовых слоев GRU из известных библиотек глубокого обучения является более эффективным и надежным выбором. Однако для исследовательских целей или узкоспециализированных приложений построение GRU-сети с нуля может дать ценные сведения и больший контроль над поведением сети [4].

В заключение следует отметить, что сети GRU являются мощным инструментом для обработки последовательных данных, а Python предоставляет универсальную платформу для реализации и анализа таких сетей. Созданные с нуля или построенные на основе готовых моделей, GRU-сети могут быть адаптированы к широкому спектру приложений, от обработки естественного языка до прогнозирования временных рядов. Понимая сильные стороны и ограничения каждого подхода, разработчики и исследователи могут эффективно использовать сети GRU для раскрытия потенциала последовательных данных в своих проектах.

ЛИТЕРАТУРА

1. W. Foundation, «Gated recurrent unit,» [Online]. Электронный источник: https://en.wikipedia.org/wiki/Gated_recurrent_unit. [Дата обращения: 15 1 2024].
2. С. Чжан, « A GRU-Gated Attention Model for Neural Machine Translation» // IEEE Transactions on Neural Networks and Learning Systems, 2017 г. С. 4688–4698.
3. Чо, К., Ван Меррьенбоер, Б., Гульчехр, К., Бахданау, Д., Бугарес, Ф., Швенк, Х., и Бенгио, Й. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine. // Association for Computational Linguistics. 2014г. С. 1724–1734
4. Patrick Littell, «Named Entity Recognition for Linguistic Rapid Response in Low-Resource Languages: Sorani Kurdish and Tajik,» Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, стр. 998–1006, 2016 г.
5. Keras Documentation. / Электронный ресурс. — URL: https://keras.io/. — (дата обращения: 17.02.24)
6. PyTorch Documentation. / Электронный ресурс. — URL: https://pytorch.org/. — (дата обращения: 17.02.24)
7. TensorFlow Documentation. / Электронный ресурс. — URL: https://www.tensorflow.org/ — (дата обращения: 17.02.24)
8. Карпаты, А. «A Recipe for Training Neural Networks», дата публ. 25.04.19, / электронный ресурс. URL: https://karpathy.github.io/2019/04/25/recipe/

Журнал «Современная наука: актуальные проблемы теории и практики»