Министерство образования и науки Российской Федерации
ФГБОУ «Петрозаводский государственный университет»
Институт математики и информационных технологий
Кафедра информатики и математического обеспечения

09.03.04 Программная инженерия

Профиль направления подготовки бакалавриата

«Системное и прикладное программное обеспечение»

Отчёт о прохождении производственной практики

Выполнил:
студент 2 курса группы 22207

П. А. Данило _____
подпись

Место прохождения практики:
Кафедра Информатики и Математического Обеспечения

Период прохождения практики:
29.05.2023-11.06.2023

Руководитель:
О.Ю. Богоявленская, к.т.н, доцент

_____
подпись

Итоговая оценка:

_____
оценка

Петрозаводск — 2023

Содержание

Введение

Для расширения функционала базового решения программы генерации случайного текста с использованием алгоритма цепи Маркова можно использовать библиотеки в области машинного обучения, такие как TensorFlow или PyTorch. Эти инструменты могут быть использованы для обучения модели генерации текста на основе большого объема текстовых данных и для предварительной обработки данных, такой как удаление стоп-слов, лемматизация и токенизация. Также возможно определение тональности текста с помощью библиотек машинного обучения, таких как NLTK или TextBlob. Интеграция машинного обучения в код может быть продиктована конкретными задачами и требованиями.

Цель практики:

Исследование модуля разбора текста, использующего алгоритм случайной генерации текста на основе цепей Маркова и внедрение функционала машинного обучения.

Задачи практики:

1. Изучить метод генерации случайного текста с помощью цепей Маркова.
2. Добавить функции, связанные с машинным обучением.

1 Генерация случайного текста с использованием цепей Маркова

Задача производственной практики заключается в разработке алгоритма, который бы мог генерировать случайный текст. Для этого можно использовать цепь Маркова.
Входные данные представляются в виде перекрывающихся словосочетаний. Алгоритм разделяет каждую фразу на две части: префикс, состоящий из нескольких слов, и суффикс, состоящий из одного слова, следующего за префиксом. Генерация фраз происходит случайным образом путем выбора для каждого префикса следующего за ним суффикса на основе статистики исходного текста. Для этой цели вполне подходят сочетания из трех слов. То есть, по префиксу из двух слов выбирается третье слово-суффикс. Алгоритм работает следующим образом:

```
Поместить первые два слова текста в переменные w1 и w2.
Вывести значения переменных w1 и w2.
Начать цикл:
Случайным образом выбрать w3 из суффиксов, соответствующих префиксу w1 и w2.
Заменить значения переменных w1 и w2 на w2 и w3.
Повторить цикл.
```

Программа будет считывать фрагмент текста и использовать марковский алгоритм для генерирования нового текста на основе частот появления словосочетаний фиксированной длины. Количество слов в префиксе - параметр задачи. В данном случае используется префикс из двух слов. Чем короче префикс, тем более бессвязным оказывается новый текст; чем этот префикс длиннее, тем больше программа склонна повторять текст практически дословно. Для сохранения знаков препинания и грамматики в сгенерированном тексте желательно оставлять его пунктуацию. Поэтому можно считать слова "word"и "word."различными.

2 Описание работы функционала машинного обучения

Модуль содержит определение и обучение рекуррентной нейронной сети для генерации текста. Класс RNN наследуется от nn.Module и определяет архитектуру модели. Он состоит из трех слоев: Embedding, RNN и Linear. Метод forward используется для вычисления прямого прохода через модель. Затем задается длина входной последовательности, размер скрытого слоя и размер выходного слоя. Далее определяются функция потерь (criterion) и оптимизатор (optimizer). Затем происходит цикл обучения модели с помощью метода backward() и метода step(). Обновление параметров модели производится с помощью оптимизатора. После обучения модели генерируется новый текст на основе начального префикса "because". Выбор следующего символа осуществляется с использованием softmax-функции и метода np.random.choice(), который выбирает индекс следующего символа с учетом вероятностей его появления.

3 Реализация

Класс RNN, который наследуется от nn.Module.

```
import torch
from torch import nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.RNN(hidden_size, hidden_size)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x, hidden):
        x = self.embedding(x)
        out, hidden = self.rnn(x, hidden)
        out = self.fc(out)
        return out, hidden

input_size = len(vocabulary)
hidden_size = 256
output_size = len(vocabulary)

model = RNN(input_size, hidden_size, output_size)
```

Определение функции потерь и оптимизатора

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
```

Тренировка модели

```

for epoch in range(100):
    running_loss = 0.0
    hidden = torch.zeros(1, 1, hidden_size)
    for i in range(len(data) - sequence_length):
        inputs = data[i:i+sequence_length]
        targets = data[i+sequence_length]

        inputs_tensor = torch.tensor([vocabulary.index(char) for char in inputs], dtype=torch.long)
        target_tensor = torch.tensor([vocabulary.index(targets)], dtype=torch.long)

        optimizer.zero_grad()

        outputs, hidden = model(inputs_tensor, hidden)
        loss = criterion(outputs[-1], target_tensor)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```

Генерация нового текста

```
initial_prefix = "because"
hidden = torch.zeros(1, 1, hidden_size)
generated_text = list(initial_prefix)

for _ in range(text_length - len(generated_text)):
    prefix_tensor = torch.tensor([vocabulary.index(char) for char in generated_text[-sequence_length:]], dtype=torch.long)
    _, hidden = model(prefix_tensor, hidden)
    output = nn.functional.softmax(_.squeeze()).data.numpy()
    next_index = np.random.choice(len(output), p=output)
    next_char = vocabulary[next_index]
    generated_text.append(next_char)
```

Заключение

В процессе производственной практики был изучен метод случайной генерации текста с использованием цепей Маркова, а так же реализованы функции, связанные с машинным обучением.

Список литературы

1. Керниган, Брайан У., Пайк, Роб. Практика программирования. : Пер. с англ. - М. : ООО «И.Д. Вильямс», -288 с.

2. PYTORCH DOCUMENTATION [Электронный ресурс]. -
URL: https://pytorch.org/docs/stable/index.html