

Использование библиотеки `flow_matching` для реализации алгоритмов сопоставления потоков

Анишкова Анастасия Сергеевна

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

Цель статьи — показать практическое применение библиотеки `flow_matching` для решения задач сопоставления потоков, начиная с подготовки данных и заканчивая визуализацией результатов работы модели. Работа выполнялась в облачной среде Google Colab с использованием языка Python, что обеспечило удобство и гибкость в проведении экспериментов. `Flow_matching` — это мощный инструмент для исследователей и разработчиков, работающих в области машинного обучения и глубокого обучения, особенно в задачах, связанных с генерацией данных и моделированием сложных распределений. Результат может быть применён в широком спектре задач — от генерации данных и оптимизации до научных исследований и обработки естественного языка. Её гибкость и интеграция с PyTorch делают её удобной для экспериментов и внедрения в реальные проекты.

Ключевые слова: Google Colab, `flow_matching`, реализация алгоритмов, сопоставления потоков.

Using the `flow_matching` library to implement flow matching algorithms

Anishkova Anastasia Sergeevna

Sholom-Aleichem Priamurskiy State University

Student

Abstract

The purpose of the article is to show the practical application of the `flow_matching` library for solving flow matching tasks, starting with data preparation and ending with visualization of the results of the model. The work was performed in the Google Colab cloud environment using Python, which provided convenience and flexibility in conducting experiments. `Flow_matching` is a powerful tool for researchers and developers working in the field of machine learning and deep learning, especially in tasks related to data generation and modeling of complex distributions. The result can be applied in a wide range of tasks, from data generation and optimization to scientific research and natural language processing. Its flexibility and integration with PyTorch make it convenient for experimentation and implementation in real projects.

Keywords: Google Colab, `flow_matching`, implementation of algorithms, flow matching.

1 Введение

1.1 Актуальность

Сопоставление потоков данных является важной задачей в современных приложениях, таких как анализ временных рядов, обработка событий, мониторинг систем и другие. Библиотека `flow_matching` предоставляет инструменты для эффективного решения таких задач, что делает её изучение и применение актуальным для исследователей и разработчиков.

1.2 Обзор исследований

С. Имамби, К. Б. Пракаш, Г.Р. Канагачидамбаресан рассмотрел ключевые моменты библиотека для программ на Python, которая упрощает работу с программами для глубокого обучения [1], практические аспекты внедрения решений на основе глубокого обучения с помощью PyTorch, используя практический подход к пониманию теории и практики Изучил Н. Кеткар [2], О. С. Новак представил анализ важных аспектов, возникающих при разработке приложений на основе нейронных сетей [3], набор инструментов с открытым исходным кодом под названием MMRotate, который обеспечивает согласованную структуру алгоритмов обучения, вывода и оценки для популярного алгоритма обнаружения вращающихся объектов, основанного на глубоком обучении представил И. Чжоу [4], Юань Джей проанализировал эффективность задач распознавания изображений с помощью алгоритмов глубокого обучения на основе PyTorch [5].

1.3 Цель исследования

Цель статьи — показать практическое применение библиотеки `flow_matching` для решения задач сопоставления потоков, начиная с подготовки данных и заканчивая визуализацией результатов работы модели.

2 Материалы и методы

В данном исследовании используется среда Google Colab и язык программирования Python.

3 Результаты

Установим библиотеки, необходимые для работы с PyTorch, визуализацией данных и алгоритмами сопоставления потоков (рис.1).

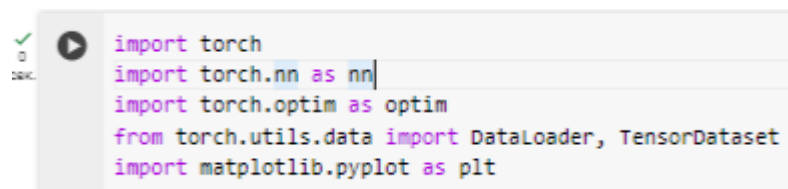


```
!pip install torch torchvision matplotlib flow_matching

Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.5.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-packages (0.20.1+cu121)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: flow_matching in /usr/local/lib/python3.11/dist-packages (from torch) (1.0.10)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.11/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.11/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch) (2024.10.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.1.105 in /usr/local/lib/python3.11/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.1.105 in /usr/local/lib/python3.11/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.1.105 in /usr/local/lib/python3.11/dist-packages (from torch) (12.1.105)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.1.3.1 in /usr/local/lib/python3.11/dist-packages (from torch) (12.1.3.1)
Requirement already satisfied: nvidia-cufft-cu12==11.0.2.54 in /usr/local/lib/python3.11/dist-packages (from torch) (11.0.2.54)
Requirement already satisfied: nvidia-curand-cu12==10.3.2.106 in /usr/local/lib/python3.11/dist-packages (from torch) (10.3.2.106)
Requirement already satisfied: nvidia-cusolver-cu12==11.4.5.107 in /usr/local/lib/python3.11/dist-packages (from torch) (11.4.5.107)
```

Рисунок 1. Установка библиотеки

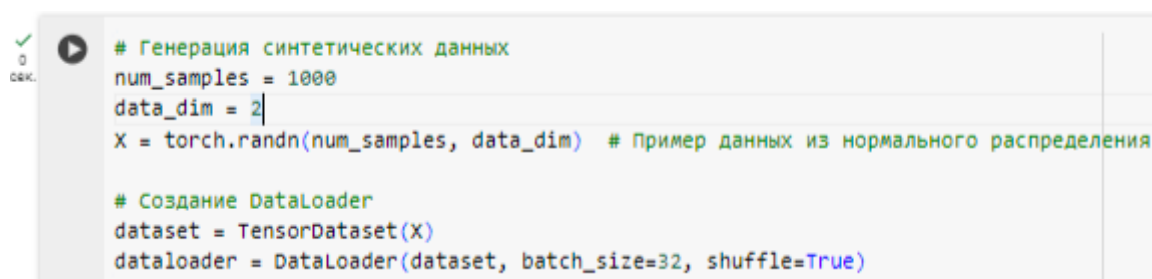
Импортируем необходимые библиотеки и модули для работы с PyTorch и визуализацией данных (рис.2): torch (основная библиотека для работы с тензорами и нейронными сетями), torch.nn (модуль для создания нейронных сетей, включая слои и функции активации), torch.optim (модуль для оптимизации моделей, например, с использованием SGD или Adam), DataLoader и TensorDataset (инструменты для удобной работы с данными, такие как загрузка, батчи и итерация), а также matplotlib.pyplot (библиотека для визуализации данных, включая построение графиков и диаграмм).



```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import matplotlib.pyplot as plt
```

Рисунок 2. Импорт библиотек

Сгенерируем синтетические данные и подготовим их для итеративной обработки в процессе обучения модели (рис.3).

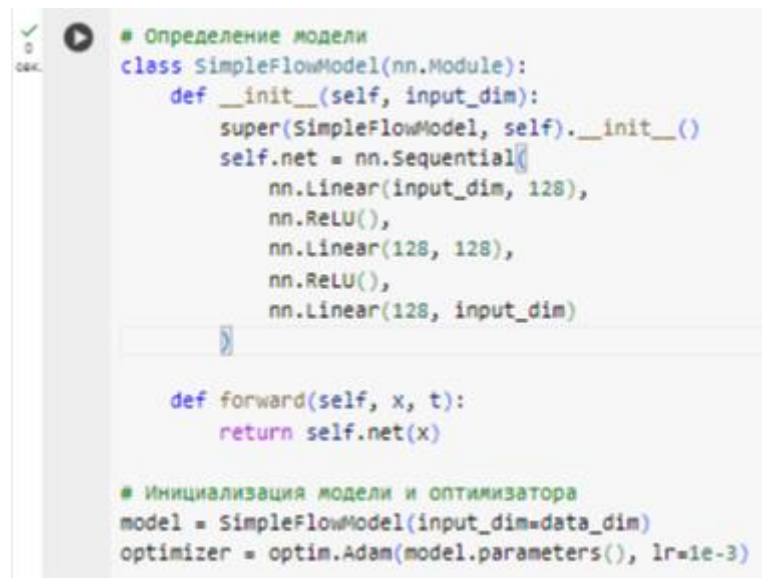


```
# Генерация синтетических данных
num_samples = 1000
data_dim = 2
X = torch.randn(num_samples, data_dim) # Пример данных из нормального распределения

# Создание DataLoader
dataset = TensorDataset(X)
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

Рисунок 3. Подготовка данных

Определим архитектуру нейронной сети, создавая её экземпляры и настраивает оптимизатор для обучения (рис.4).



```

# Определение модели
class SimpleFlowModel(nn.Module):
    def __init__(self, input_dim):
        super(SimpleFlowModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, input_dim)
        )

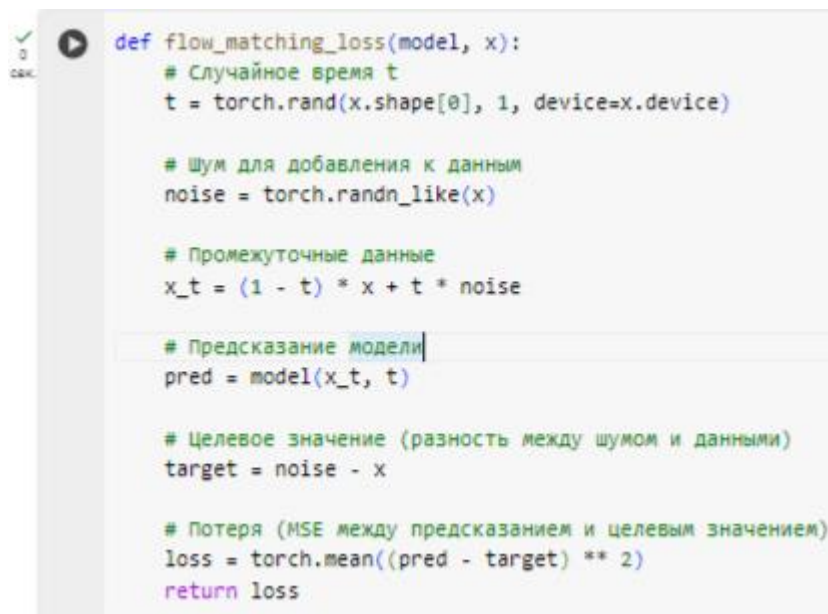
    def forward(self, x, t):
        return self.net(x)

# Инициализация модели и оптимизатора
model = SimpleFlowModel(input_dim=data_dim)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

```

Рисунок 4. Архитектура нейронной сети

Вычислим потерю для Flow Matching, сравнивая предсказания модели с целевыми значениями (рис.5), которые представляют разность между шумом и исходными данными.



```

def flow_matching_loss(model, x):
    # Случайное время t
    t = torch.rand(x.shape[0], 1, device=x.device)

    # Шум для добавления к данным
    noise = torch.randn_like(x)

    # Промежуточные данные
    x_t = (1 - t) * x + t * noise

    # Предсказание модели
    pred = model(x_t, t)

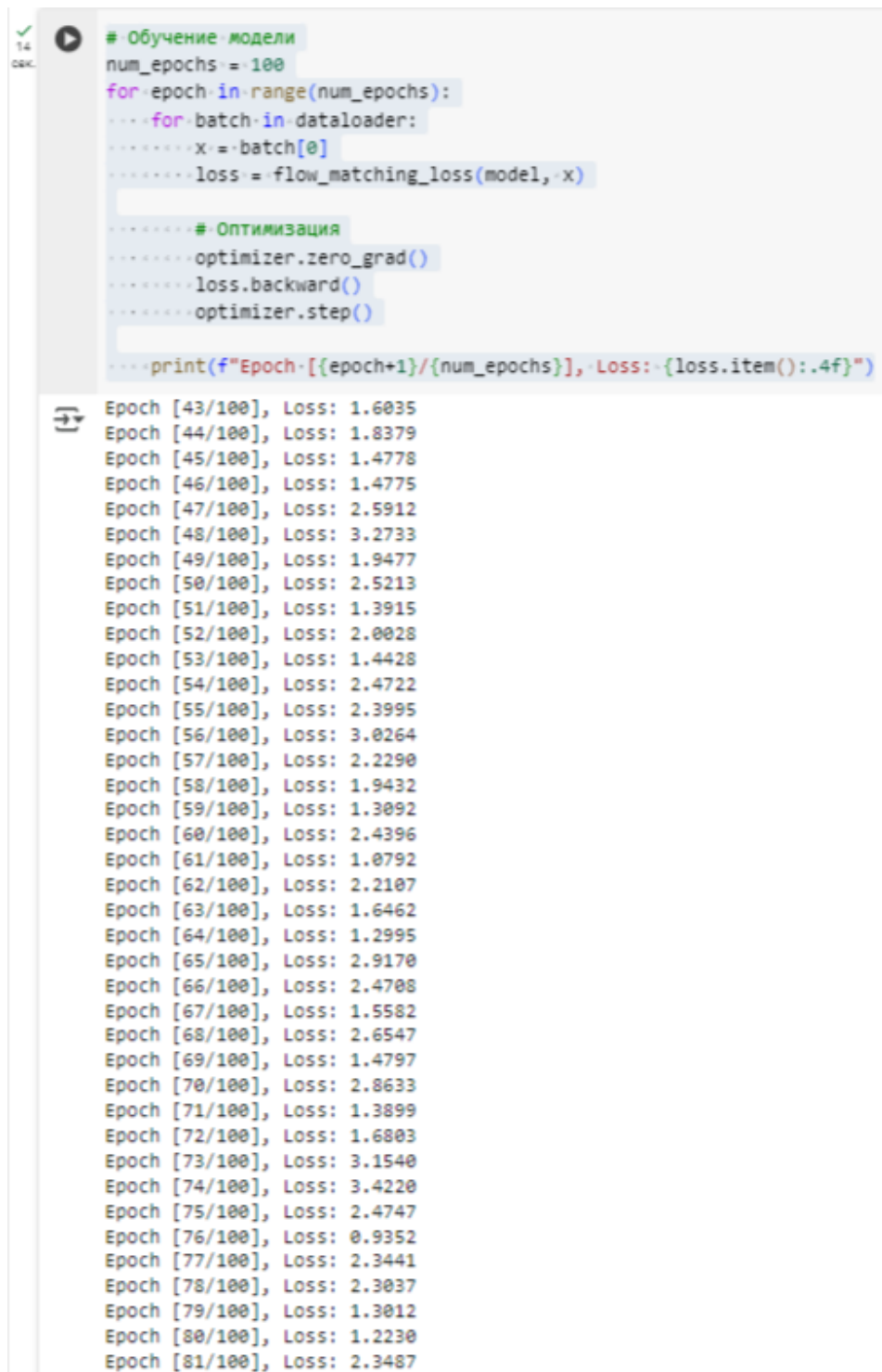
    # Целевое значение (разность между шумом и данными)
    target = noise - x

    # Потеря (MSE между предсказанием и целевым значением)
    loss = torch.mean((pred - target) ** 2)
    return loss

```

Рисунок 5. Сравнение предсказаний

Обучим модель, обновляя её параметры на каждом батче данных, и выведем значение потерь после каждой эпохи (рис.6).



```
# Обучение модели
num_epochs = 100
for epoch in range(num_epochs):
    for batch in dataloader:
        x = batch[0]
        loss = flow_matching_loss(model, x)

        # Оптимизация
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")
```

Epoch [43/100], Loss: 1.6035
Epoch [44/100], Loss: 1.8379
Epoch [45/100], Loss: 1.4778
Epoch [46/100], Loss: 1.4775
Epoch [47/100], Loss: 2.5912
Epoch [48/100], Loss: 3.2733
Epoch [49/100], Loss: 1.9477
Epoch [50/100], Loss: 2.5213
Epoch [51/100], Loss: 1.3915
Epoch [52/100], Loss: 2.0028
Epoch [53/100], Loss: 1.4428
Epoch [54/100], Loss: 2.4722
Epoch [55/100], Loss: 2.3995
Epoch [56/100], Loss: 3.0264
Epoch [57/100], Loss: 2.2290
Epoch [58/100], Loss: 1.9432
Epoch [59/100], Loss: 1.3092
Epoch [60/100], Loss: 2.4396
Epoch [61/100], Loss: 1.0792
Epoch [62/100], Loss: 2.2107
Epoch [63/100], Loss: 1.6462
Epoch [64/100], Loss: 1.2995
Epoch [65/100], Loss: 2.9170
Epoch [66/100], Loss: 2.4708
Epoch [67/100], Loss: 1.5582
Epoch [68/100], Loss: 2.6547
Epoch [69/100], Loss: 1.4797
Epoch [70/100], Loss: 2.8633
Epoch [71/100], Loss: 1.3899
Epoch [72/100], Loss: 1.6803
Epoch [73/100], Loss: 3.1540
Epoch [74/100], Loss: 3.4220
Epoch [75/100], Loss: 2.4747
Epoch [76/100], Loss: 0.9352
Epoch [77/100], Loss: 2.3441
Epoch [78/100], Loss: 2.3037
Epoch [79/100], Loss: 1.3012
Epoch [80/100], Loss: 1.2230
Epoch [81/100], Loss: 2.3487

Рисунок 6. Обучение модели

Создадим начальные точки (рис7), генерируя случайные значения $\mathbf{z}$ из нормального распределения, затем пропускает их через обученную модель без вычисления градиентов (с использованием `torch.no_grad()`) и возвращает сгенерированные данные.

```
def generate_samples(model, num_samples, data_dim):  
    # Начальные точки из простого распределения (например, гауссово)  
    z = torch.randn(num_samples, data_dim)  
  
    # Проход через модель  
    with torch.no_grad():  
        generated_data = model(z, torch.ones(num_samples, 1))  
  
    return generated_data  
  
# Генерация данных  
num_generated_samples = 100  
generated_data = generate_samples(model, num_generated_samples, data_dim)
```

Рисунок 7. Генерацию данных с помощью обученной модели

Отобразим scatter-график для визуализации сгенерированных данных (рис.8).

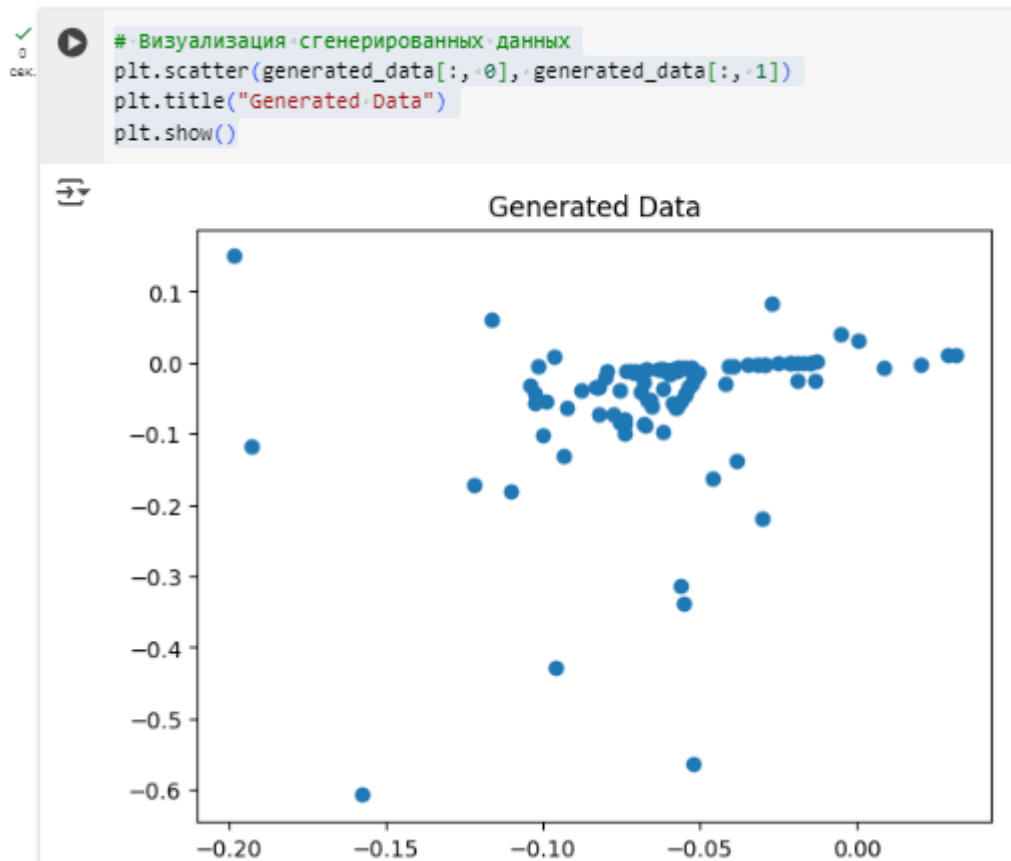


Рисунок 8. График

Библиографический список

1. Imambi S., Prakash K. B., Kanagachidambaresan G. R. PyTorch //Programming with TensorFlow: solution for edge computing applications.2021. С. 87-104.
2. Ketkar N. et al. Introduction to pytorch //Deep learning with python: learn best practices of deep learning models with PyTorch. 2021. С. 27-91.
3. Novac O. C. et al. Analysis of the application efficiency of TensorFlow and

-
- PyTorch in convolutional neural network //Sensors. 2022.№. 22.C. 8872.
4. Zhou Y. et al. Mmrotate: A rotated object detection benchmark using pytorch //Proceedings of the 30th ACM International Conference on Multimedia.2022. C. 7331-7334.
 5. Yuan J. Performance Analysis of Deep Learning Algorithms Implemented Using PyTorch in Image Recognition //Procedia Computer Science. 2024.C. 61-69.