Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Рекуррентные нейронные сети»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Туз И. С.

Брест 2025

1. Цель работы: Реализовать и обучить рекуррентную нейронную сеть(batch\_learning, реализовать вручную, но так, чтоб вычисления можно было проводить на gpu. Например на PyTorch).
2. Реализовать сеть с использованием RNN слоя(с использованием ML-фреймворка)
3. Реализовать сеть с использованием GRU слоя(с использованием ML-фреймворка)
4. Реализовать сеть с использованием LSTM слоя(с использованием ML-фреймворка)

**Ход работы**

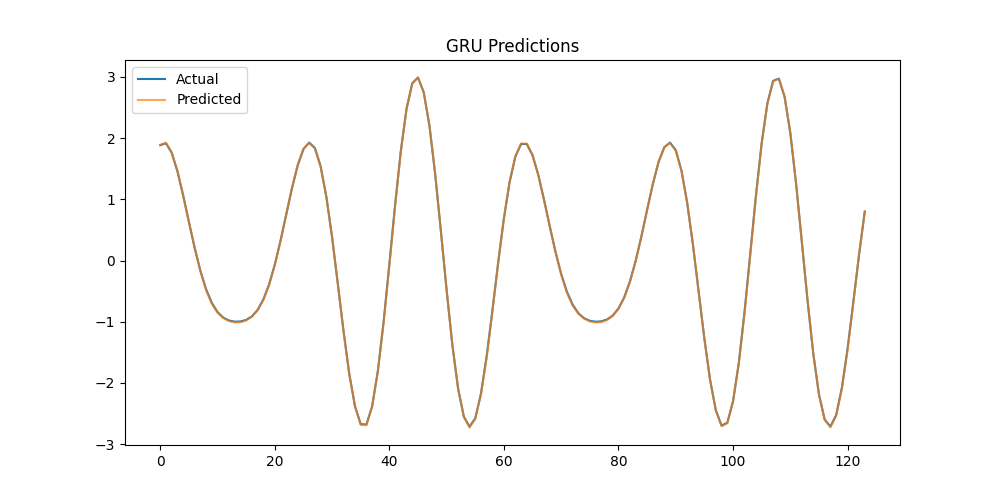
Manual\_rnn.py:

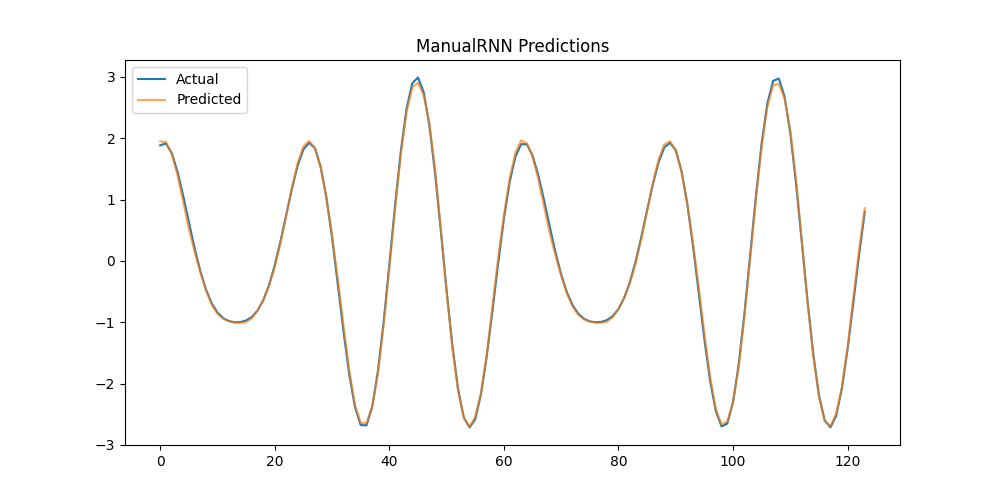
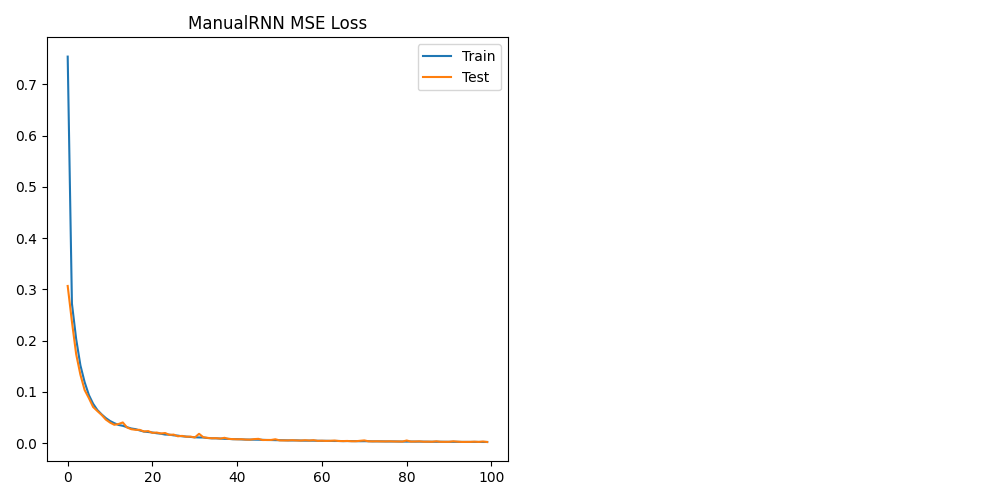
Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

import torch  
from config import \*  
  
  
class ManualRNN:  
 def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):  
 # Инициализация с Xavier и нормальными весами  
 self.W\_ih = torch.randn(input\_size, hidden\_size)  
 self.W\_hh = torch.randn(hidden\_size, hidden\_size)  
 self.W\_ho = torch.randn(hidden\_size, output\_size)  
 torch.nn.init.xavier\_normal\_(self.W\_ih)  
 torch.nn.init.xavier\_normal\_(self.W\_hh)  
 torch.nn.init.xavier\_normal\_(self.W\_ho)  
  
 self.b\_h = torch.zeros(hidden\_size)  
 self.b\_o = torch.zeros(output\_size)  
  
 self.cache = {  
 'inputs': [],  
 'hidden\_states': [],  
 'outputs': []  
 }  
  
 def tanh(self, x):  
 # Стабилизированная реализация tanh  
 return torch.clip((torch.exp(x) - torch.exp(-x)) /  
 (torch.exp(x) + torch.exp(-x)), -0.999, 0.999)  
  
 def forward(self, x):  
 self.cache = {'inputs': [], 'hidden\_states': [torch.zeros(x.size(0), HIDDEN\_SIZE)], 'outputs': []}  
  
 for t in range(x.size(1)):  
 x\_t = x[:, t, :]  
 h\_prev = self.cache['hidden\_states'][-1]  
  
 pre\_activation = x\_t @ self.W\_ih + h\_prev @ self.W\_hh + self.b\_h  
 pre\_activation = torch.clip(pre\_activation, -10, 10)  
  
 h = self.tanh(pre\_activation)  
  
 self.cache['inputs'].append(x\_t)  
 self.cache['hidden\_states'].append(h)  
  
 output = self.cache['hidden\_states'][-1] @ self.W\_ho + self.b\_o  
 self.cache['outputs'].append(output)  
  
 return output  
  
 def backward(self, dL\_dy, learning\_rate=0.01):  
 batch\_size = dL\_dy.shape[0]  
  
 dW\_ih = torch.zeros\_like(self.W\_ih)  
 dW\_hh = torch.zeros\_like(self.W\_hh)  
 db\_h = torch.zeros\_like(self.b\_h)  
 dW\_ho = torch.zeros\_like(self.W\_ho)  
 db\_o = torch.zeros\_like(self.b\_o)  
 h\_last = self.cache['hidden\_states'][-1]  
 dW\_ho = h\_last.T @ dL\_dy / batch\_size  
 db\_o = dL\_dy.mean(dim=0)  
 dL\_dh = dL\_dy @ self.W\_ho.T  
 for t in reversed(range(len(self.cache['inputs']))):  
 x\_t = self.cache['inputs'][t]  
 h\_prev = self.cache['hidden\_states'][t]  
 h\_curr = self.cache['hidden\_states'][t + 1]  
 dtanh = (1 - h\_curr \*\* 2) \* dL\_dh  
 dW\_ih += x\_t.T @ dtanh / batch\_size  
 dW\_hh += h\_prev.T @ dtanh / batch\_size  
 db\_h += dtanh.mean(dim=0)  
 dL\_dh = dtanh @ self.W\_hh.T  
 dL\_dh = torch.clip(dL\_dh, -1, 1)  
 self.W\_ih -= learning\_rate \* dW\_ih  
 self.W\_hh -= learning\_rate \* dW\_hh  
 self.b\_h -= learning\_rate \* db\_h  
 self.W\_ho -= learning\_rate \* dW\_ho  
 self.b\_o -= learning\_rate \* db\_o  
  
 def zero\_grad(self):  
 self.cache = {'inputs': [], 'hidden\_states': [torch.zeros\_like(self.cache['hidden\_states'][0])], 'outputs': []}

Изображение выглядит как текст, снимок экрана, диаграмма, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.



**Вывод:** в ходе лабораторной работы я научился реализовывать PCA и автоэнкодеры.