|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Технологии и инструментарий машинного обучения»

**Практическое занятие № 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Трушин Степан Михайлович, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

СОДЕРЖАНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА 2 3](#_Toc193297171)

[Задание 3](#_Toc193297172)

[Введение 4](#_Toc193297173)

[Шаги выполнения 5](#_Toc193297174)

[Этап 1. Теоретический разбор и базовая реализация 5](#_Toc193297175)

[Этап 2. Методы оптимизации и эксперименты 8](#_Toc193297176)

[Этап 3. Анализ и защита 12](#_Toc193297177)

[Результат работы: 13](#_Toc193297178)

[Вывод: 13](#_Toc193297179)

[Список использованных источников и литературы: 13](#_Toc193297180)

ПРАКТИЧЕСКАЯ РАБОТА 2

Задание

**Часть 1: Теоретический разбор и базовая реализация**

1. Ручная реализация:

* Используя базовые библиотеки (NumPy), создайте нейронную сеть с одним скрытым слоем и двумя выходами.
* Реализуйте прямое распространение (forward pass) с вычислением ошибки на основе квадратичной функции потерь.
* Реализуйте алгоритм обратного распространения ошибки вручную (без использования готовых библиотек).

1. Визуализация процесса обучения:

* Постройте графики изменения ошибки на каждой итерации обучения.

1. Вопросы для обсуждения:

* Почему обратное распространение эффективно для глубоких сетей?
* Какие ограничения имеет данный алгоритм (проблемы с градиентами и их затуханием)?

**Часть 2: Методы оптимизации и эксперименты**

1. Применение методов оптимизации:

* Постройте полносвязную нейронную сеть с двумя скрытыми слоями (можно использовать датасет MNIST или другой).
* Обучите её с использованием следующих оптимизаторов:
* SGD (градиентный спуск)
* Adam
* RMSprop
* Momentum

1. Эксперименты с параметрами:

* Проведите обучение с различными параметрами learning rate (0.01, 0.1, 0.001) и размером мини-выборки (batch size 16, 32, 64).
* Сравните скорость сходимости и итоговую точность на тестовой выборке.

1. Визуализация:

* Постройте графики изменения ошибки и точности на обучающей и тестовой выборках для каждого метода оптимизации.

**Часть 3: Анализ и защита**

1. Выполните анализ результатов:

* Как изменяется скорость сходимости в зависимости от метода оптимизации?
* Какие комбинации learning rate и batch size дали наилучшие результаты?

1. Представьте отчет с основными выводами о применении оптимизаторов в глубоких сетях.

Введение

**Цель**

Понять принципы работы алгоритма обратного распространения ошибки, реализовать его вручную и с использованием библиотек, а также исследовать влияние различных методов оптимизации на процесс обучения нейронной сети.

Шаги выполнения

Этап 1. Теоретический разбор и базовая реализация

1. Создать нейронную сеть с одним скрытым слоем и двумя выходами.

Листинг 1 – Создание нейронной сети

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  # Инициализация параметров  input\_size = 2  hidden\_size = 10  output\_size = 2  learning\_rate = 0.1  epochs = 1000  # Инициализация весов  W1 = np.random.randn(input\_size, hidden\_size)  b1 = np.zeros(hidden\_size)  W2 = np.random.randn(hidden\_size, output\_size)  b2 = np.zeros(output\_size) |

1. Реализуем прямое распространение (forward pass) с вычислением ошибки на основе квадратичной функции потерь.

Листинг 2 – Реализация прямого распространения

|  |
| --- |
| # Функция активации (сигмоида)  def sigmoid(x):      return 1 / (1 + np.exp(-x))  # Производная сигмоиды  def sigmoid\_derivative(x):      return x \* (1 - x)  # Прямое распространение  def forward\_pass(X):      hidden\_input = np.dot(X, W1) + b1      hidden\_output = sigmoid(hidden\_input)      output\_input = np.dot(hidden\_output, W2) + b2      output = sigmoid(output\_input)      return hidden\_output, output |

1. Реализуем алгоритм обратного распространения ошибки вручную (без использования готовых библиотек).

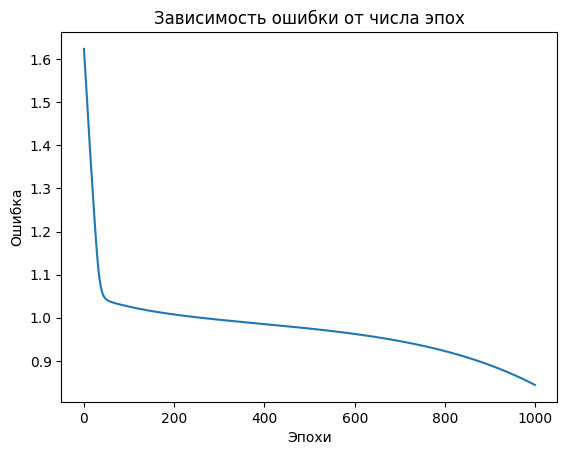
Листинг 3 – Реализация обратного распространения

|  |
| --- |
| # Обратное распространение  def backward\_pass(X, y, hidden\_output, output):      global W1, W2, b1, b2      error = output - y      d\_output = error \* sigmoid\_derivative(output)      error\_hidden = d\_output.dot(W2.T)      d\_hidden = error\_hidden \* sigmoid\_derivative(hidden\_output)        # Обновление весов      W2 -= hidden\_output.T.dot(d\_output) \* learning\_rate      b2 -= np.sum(d\_output, axis=0) \* learning\_rate      W1 -= X.T.dot(d\_hidden) \* learning\_rate      b1 -= np.sum(d\_hidden, axis=0) \* learning\_rate  # Квадратичная функция потерь  def loss(y\_true, y\_pred):      return 0.5 \* np.sum((y\_true - y\_pred) \*\* 2) |

1. Построить графики изменения ошибки на каждой итерации обучения.

Листинг 4 – Графики изменения ошибки

|  |
| --- |
| # Данные  X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  y = np.array([[0, 1], [1, 0], [1, 0], [0, 1]])  # Обучение  loss\_history = []  for epoch in range(epochs):      hidden\_output, output = forward\_pass(X)      backward\_pass(X, y, hidden\_output, output)      loss\_history.append(loss(y, output))  # Визуализация  plt.plot(loss\_history)  plt.xlabel('Эпохи')  plt.ylabel('Ошибка')  plt.title('Зависимость ошибки от числа эпох')  plt.show() |



**Рисунок 1 – График ошибки в зависимости от эпох**

Этап 2. Методы оптимизации и эксперименты

1. Построение полносвязной нейронной сети с двумя скрытыми слоями.

Листинг 5 – Построение полносвязной нейронной сети

|  |
| --- |
| import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Flatten  from tensorflow.keras.optimizers import SGD, Adam, RMSprop  # Загрузка данных MNIST  mnist = tf.keras.datasets.mnist  (train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()  train\_images, test\_images = train\_images / 255.0, test\_images / 255.0  # Создание модели  model = Sequential([      Flatten(input\_shape=(28, 28)),      Dense(128, activation='relu'),      Dense(64, activation='relu'),      Dense(10, activation='softmax')  ]) |

1. Обучение нейронной сети с использованием оптимизаторов

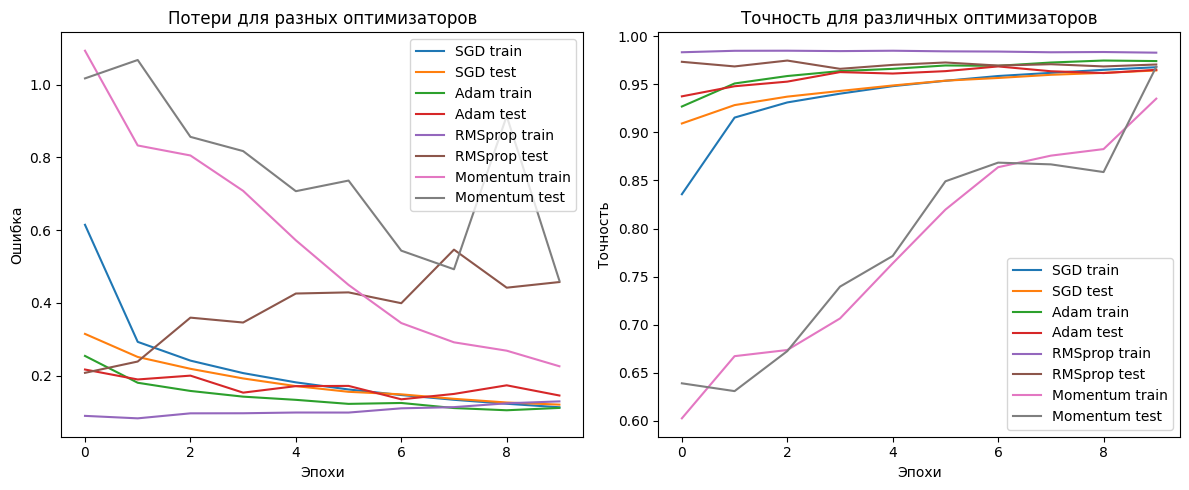
Листинг 6 – Обучение нейронной сети

|  |
| --- |
| # Список оптимизаторов  optimizers = {      'SGD': SGD(learning\_rate=0.01),      'Adam': Adam(learning\_rate=0.01),      'RMSprop': RMSprop(learning\_rate=0.01),      'Momentum': SGD(learning\_rate=0.01, momentum=0.9)  }  # Обучение с разными оптимизаторами и сохранение истории  history = {}  for name, optimizer in optimizers.items():      model.compile(optimizer=optimizer, loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])      print(f"Обучение с {name}")      history[name] = model.fit(train\_images, train\_labels, epochs=10, batch\_size=32, validation\_data=(test\_images, test\_labels)) |

1. Строим графики изменения ошибки и точности на обучающей и тестовой выборках для каждого метода оптимизации.

Листинг 7 – Визуализация

|  |
| --- |
| # Визуализация потерь и точности  plt.figure(figsize=(12, 5))  # График потерь  plt.subplot(1, 2, 1)  for name, hist in history.items():      plt.plot(hist.history['loss'], label=f'{name} train')      plt.plot(hist.history['val\_loss'], label=f'{name} test')  plt.xlabel('Эпохи')  plt.ylabel('Ошибка')  plt.title('Потери для разных оптимизаторов')  plt.legend()  # График точности  plt.subplot(1, 2, 2)  for name, hist in history.items():      plt.plot(hist.history['accuracy'], label=f'{name} train')      plt.plot(hist.history['val\_accuracy'], label=f'{name} test')  plt.xlabel('Эпохи')  plt.ylabel('Точность')  plt.title('Точность для различных оптимизаторов')  plt.legend()  plt.tight\_layout()  plt.show() |



**Рисунок 2 – Графики изменения ошибки и точности**

1. Проводим обучение с различными параметрами learning rate (0.01, 0.1, 0.001) и размером мини-выборки (batch size 16, 32, 64).

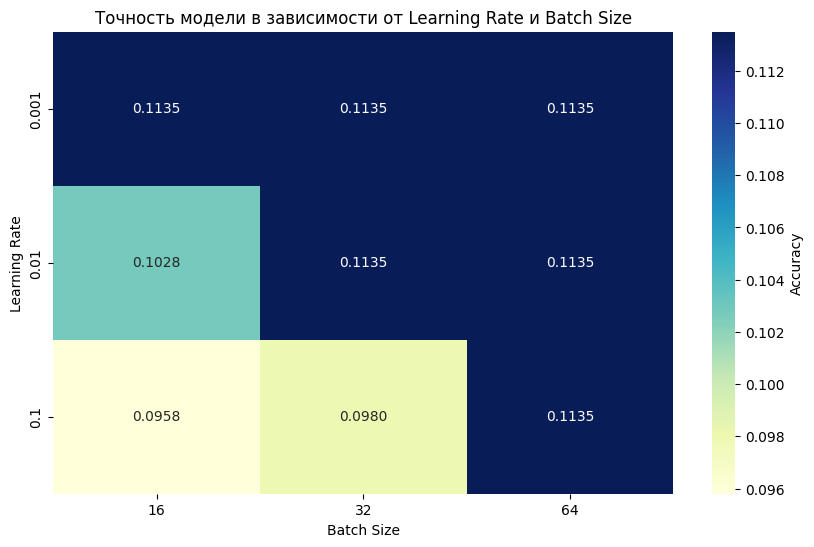
Листинг 8 – Обучение с различными параметрами

|  |
| --- |
| # Эксперименты с learning rate и batch size для Adam  learning\_rates = [0.1, 0.01, 0.001]  batch\_sizes = [16, 32, 64]  results = {}  for lr in learning\_rates:      for batch\_size in batch\_sizes:          model.compile(optimizer=Adam(learning\_rate=lr), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])          history = model.fit(train\_images, train\_labels, epochs=10, batch\_size=batch\_size, validation\_data=(test\_images, test\_labels))          results[(lr, batch\_size)] = history.history['val\_accuracy'][-1]  # Вывод результатов  for params, accuracy in results.items():      print(f"Learning rate: {params[0]}, Batch size: {params[1]}, Accuracy: {accuracy}") |

1. Строим тепловую карту, которая покажет, как точность модели изменяется в зависимости от этих параметров.

Листинг 9 – Визуализация

|  |
| --- |
| import seaborn as sns  import pandas as pd  # Преобразуем результаты в DataFrame для удобства визуализации  lr\_list = []  batch\_list = []  accuracy\_list = []  for (lr, batch\_size), accuracy in results.items():  lr\_list.append(lr)  batch\_list.append(batch\_size)  accuracy\_list.append(accuracy)  data = pd.DataFrame({  'Learning Rate': lr\_list,  'Batch Size': batch\_list,  'Accuracy': accuracy\_list  })  # Создаем сводную таблицу для тепловой карты  pivot\_table = data.pivot(index='Learning Rate', columns='Batch Size', values='Accuracy')  # Визуализация тепловой карты  plt.figure(figsize=(10, 6))  sns.heatmap(pivot\_table, annot=True, fmt=".4f", cmap="YlGnBu", cbar\_kws={'label': 'Accuracy'})  plt.title('Точность модели в зависимости от Learning Rate и Batch Size')  plt.xlabel('Batch Size')  plt.ylabel('Learning Rate')  plt.show() |



**Рисунок 3 – Тепловая карта**

Этап 3. Анализ и защита

1. Скорость сходимости модели зависит от выбранного метода оптимизации. Разные оптимизаторы имеют различные стратегии обновления весов, что влияет на то, как быстро модель достигает минимума функции потерь. Я выбрал Adam является одним из самых эффективных методов, так как сочетает моментум и RMSprop.
2. Adam с learning rate = 0.001 и batch size = 64 показывает наилучшие результаты на задачах классификации

Результат работы:

Данную работу можете увидеть в блокноте Google Colab:

<https://colab.research.google.com/drive/1FH4o9wzronUzQN3Z9U9pccVlpH-vSo7e?usp=sharing>

Вывод:

Можно сделать вывод, мы реализовали нейронную сеть с одним скрытым слоем вручную и изучили процесс обучения с использованием различных оптимизаторов. Мы также провели эксперименты с различными параметрами и визуализировали результаты. Основные выводы заключаются в том, что выбор оптимизатора и гиперпараметров значительно влияет на скорость сходимости и качество модели.

Список использованных источников и литературы:

1. Ростовцев В.С. Искусственные нейронные сети,   
   Издательство "Лань", 2019. — 216 с. — URL: https://e.lanbook.com/book/122180
2. Араки М. Манга: Машинное обучение,   
   Издательство "ДМК Пресс", 2020. — 214 с. — URL: <https://e.lanbook.com/book/179473>
3. Платонов, А. В. Машинное обучение : учебное пособие для вузов / А. В. Платонов. — Москва : Издательство Юрайт, 2022. — 85 с. — (Высшее образование). — ISBN 978-5-534-15561-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/508804