

# **ПРАКТИЧЕСКАЯ РАБОТА №2: АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ И МЕТОДЫ ОПТИМИЗАЦИИ**

## **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.

## **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).
2. Визуализация процесса обучения:
  - Постройте графики изменения ошибки на каждой итерации обучения.
3. 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
2. 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.
3. 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?
2. Представьте отчет с основными выводами о применении оптимизаторов в глубоких сетях.

### **Что нужно вставить в отчет:**

1. Введение:
  - Описание целей работы.
  - Краткое описание алгоритма обратного распространения ошибки и методов оптимизации.
2. Ручная реализация алгоритма:
  - Описание архитектуры простой сети.
  - Формулы для вычисления градиентов на каждом слое.
  - Графики изменения ошибки на каждом шаге итерации.
3. Методы оптимизации:
  - Сравнительная таблица настроек для каждого метода (SGD, Adam, RMSprop и др.).
  - Графики изменения ошибки и точности для каждого метода.
4. Эксперименты:
  - Сравнительный анализ различных параметров learning rate и batch size.
  - Графики скорости сходимости и итоговых метрик.
5. Заключение:
  - Влияние различных методов оптимизации на обучение сети.
  - Рекомендации по выбору метода для конкретных задач (мелкие датасеты, глубокие сети и т. д.).
  - Проблемы, с которыми столкнулись студенты, и их решения.

# ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ ПО 2 ПРАКТИЧЕСКОЙ

## Теоретический материал по 1 части

Алгоритм обратного распространения ошибки — это основной механизм обучения многослойных нейронных сетей. Он позволяет эффективно обновлять веса сети путем вычисления градиентов функции потерь с использованием метода градиентного спуска.

Чтобы понять работу алгоритма обратного распространения, сначала рассмотрим, как нейронная сеть обрабатывает входные данные:

1. Прямое распространение (forward pass):
  - Входные данные проходят через слои сети, преобразовываясь на каждом уровне с помощью весов, функций активации и суммирования.
  - На выходе сети получается предсказание  $\hat{y}$ .
2. Вычисление ошибки:
  - Ошибка измеряется с помощью функции потерь. В данном случае мы используем квадратичную функцию потерь, представлено на формуле 13:

$$L = \frac{1}{2} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (13)$$

где  $y_i$  — истинное значение,  $\hat{y}_i$  — предсказание модели.

3. Обратное распространение (backward pass):
  - Градиенты функции потерь вычисляются для каждого веса сети с использованием правила цепного дифференцирования (chain rule).
  - Градиенты используются для корректировки весов в направлении, уменьшающем ошибку.

## Математическое описание обратного распространения

### Прямое распространение:

Рассмотрим простой случай сети с одним скрытым слоем.

- Входной слой:  $x = [x_1, x_2, \dots, x_n]$
- Скрытый слой : нейроны  $h_j$  с весами  $\omega_{ij}$
- Выходной слой:  $o_k$  с весами  $v_{jk}$

### Шаги

1. Входной сигнал передается на скрытый слой (представлено на формуле 14):

$$h_j = f\left(\sum_{i=1}^n \omega_{ij}x_i + b_j\right) \quad (14)$$

Где  $b_j$  - смещений (bias),  $f(x)$  - функция активации (напрмер, ReLU или Sigmoid)

2. Сигнал скрытого слоя передается на выходной слой (представлено на формуле 15):

$$o_k = g \times \left(\sum_{j=1}^m v_{jk}h_j + b_k\right) \quad (15)$$

Где  $g(x)$  – функция активации выходного слоя (например, Softmax или Sigmoid)

3. Предсказание сравнивается с истинным значением  $y$ , и вычисляется ошибка с использованием функции потерь.

### Обратное распространение:

На этом этапе вычисляются частные производные функции потерь по каждому весу с использованием правила цепочки.

1. На выходном слое:

Вычисляем градиент функции потерь относительно весов  $v_{jk}$ , представлено на формуле 16.

$$\frac{\partial L}{\partial L_{jk}} = \delta_k \times h_j \quad (16)$$

Где  $\delta_k = \frac{\partial L}{\partial o_k} \times g'(o_k)$

2. На скрытом слое:

Градиенты распространяются от выходного слоя к скрытому слою, представлено на формуле 17

$$\delta_j = \left( \sum_{k=1}^p \delta_k v_{jk} \right) \times \dot{f}(h_j) \quad (17)$$

где  $\dot{f}(h_j)$  — производная функции активации скрытого слоя.

3. Обновление весов

Веса обновляются с использованием правила градиентного спуска, представлено на формуле 18

$$w_{ij} \leftarrow w_{ij} - \eta \times \frac{\partial L}{\partial w_{ij}} \quad (18)$$

где  $\eta$  — скорость обучения (learning rate).

### **Пример ручной реализации на простом случае**

Рассмотрим простую нейронную сеть с одним входным нейроном  $x$ , одним нейроном в скрытом слое  $h$  и одним выходным нейроном  $o$ .

Параметры сети:

- Вход  $x = 0.5$
- Истинное значение  $y = 1$
- Вес  $w = 0.4$
- Скорость обучения  $\eta = 0.1$
- Функция активации: Sigmoid  $f(x) = \frac{1}{1 + e^{-x}}$

Шаги:

1. Прямое распространение:

• Вычисляем выход нейрона скрытого слоя, представлено в формуле 19:

$$h = f(\omega \times x) = \frac{1}{1 + e^{-0,4 \times 0,5}} \quad (19)$$

2. Вычисляем предсказание, представлено в формуле 20:

$$o = h \times v \quad (20)$$

где  $v$  — вес между скрытым и выходным слоями.

3. Обратное распространение:

- Вычисляем градиент ошибки по формуле 21:

$$\delta_o = (o - y) \times f'(o) \quad (21)$$

- Вычисляем градиент для весов  $v$  и  $w$ , формула 22 и 23 соответственно:

$$\frac{\partial L}{\partial v} = \delta_o \times h \quad (22)$$

$$\frac{\partial}{\partial w} = \delta_o \times \frac{\partial h}{\partial w} \quad (23)$$

4. Обновляем веса, формулы 24 и 25 соответственно:

$$v \leftarrow v - \eta \times \frac{\partial L}{\partial v} \quad (24)$$

$$w \leftarrow w - \eta \times \frac{\partial L}{\partial w} \quad (25)$$

### Визуализация процесса обучения

После выполнения нескольких итераций обновления весов можно построить график, показывающий изменение ошибки на каждом шаге обучения. Это поможет понять, как быстро сеть обучается и приближается ли она к оптимуму.

Пример графика с использованием Matplotlib представлен в листинге 3:

*Листинг 3 – Пример графика с использованием Matplotlib*

```
import matplotlib.pyplot as plt
# Пример данных для графика
iterations = [1, 2, 3, 4, 5]
loss_values = [0.9, 0.7, 0.5, 0.3, 0.2]
plt.plot(iterations, loss_values, marker='o')
plt.xlabel('Итерации')
plt.ylabel('Ошибка (Loss)')
```

```
plt.title('Изменение ошибки в процессе обучения')
plt.grid(True)
plt.show()
```

## Итоговые выводы

Алгоритм обратного распространения ошибки — это основной метод оптимизации в нейронных сетях.

Эффективность обучения зависит от правильного выбора гиперпараметров, таких как learning rate.

Для глубоких сетей важно использовать методы борьбы с затухающими градиентами (например, ReLU, оптимизаторы Adam и RMSprop).

## Теоретический материал по 2 части

### 1. Введение в оптимизацию нейронных сетей

В процессе обучения нейронной сети необходимо минимизировать функцию потерь  $L$  за счет обновления весов с использованием алгоритмов оптимизации. Основная цель оптимизации — найти такие параметры  $\theta$  (веса и смещения), при которых ошибка сети минимальна.

Оптимизаторы влияют на скорость сходимости, точность и стабильность обучения. В этой части мы рассмотрим наиболее распространенные методы оптимизации и проведем эксперименты с гиперпараметрами.



## 2. Основные методы оптимизации

### 2.1 Стохастический градиентный спуск (SGD - Stochastic Gradient Descent)

SGD — это один из базовых методов оптимизации, в котором обновление весов происходит на основе градиента функции потерь, рассчитанного на случайной мини-выборке данных.

Формула обновления весов представлена на рисунке 1.

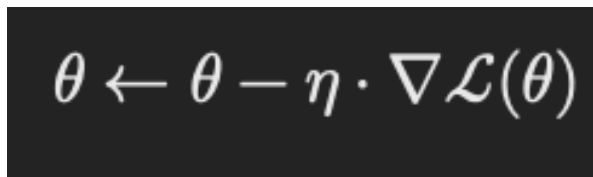

$$\theta \leftarrow \theta - \eta \cdot \nabla \mathcal{L}(\theta)$$

Рисунок 1 – Формула обновления весов

Преимущества:

- Простая реализация.
- Хорошо работает с малыми датасетами.

Недостатки:

- Может сталкиваться с колебаниями около оптимума, что замедляет сходимость.
- Зависит от выбора скорости обучения.

### 2.2 Градиентный спуск с моментумом (Momentum)

Метод моментума добавляет к обновлению весов накопленную информацию о предыдущих градиентах. Это позволяет ускорить сходимость и избежать колебаний.

Формула представлена на рисунке 2.

$$v_t = \gamma v_{t-1} + \eta \cdot \nabla \mathcal{L}(\theta)$$
$$\theta \leftarrow \theta - v_t$$

Рисунок 2 – Метод моментума

Преимущества:

- Ускоряет обучение.
- Помогает перескакивать через локальные минимумы.

Недостатки:

- Требуется настройка дополнительного гиперпараметра.

### 2.3 RMSprop (Root Mean Square Propagation)

RMSprop решает проблему затухающих градиентов за счет адаптивного выбора скорости обучения для каждого параметра. Метод основывается на сохранении скользящего среднего квадрата градиентов.

Формула представлена на рисунке 3.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta) \nabla \mathcal{L}(\theta)^2$$
$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla \mathcal{L}(\theta)$$

Рисунок 3 - RMSprop

Преимущества:

- Устраняет проблему затухающих градиентов.
- Хорошо работает в задачах с шумными градиентами.

Недостатки:

- Зависит от параметров.

## 2.4 Adam (Adaptive Moment Estimation)

Adam объединяет идеи моментума и RMSprop. Он сохраняет информацию о среднем значении градиентов и их квадратах, позволяя эффективно адаптировать скорость обучения.

Формулы представлены на рисунке 4.

$$\begin{aligned}m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla \mathcal{L}(\theta) \\v_t &= \beta_2 v_{t-1} + (1 - \beta_2) \nabla \mathcal{L}(\theta)^2 \\ \theta &\leftarrow \theta - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t\end{aligned}$$

Рисунок 4 - Adam

Преимущества:

- Быстрая сходимость.
- Хорошо работает в задачах с большим числом параметров.

Недостатки:

- Может приводить к переобучению на малых датасетах.

## 3. Гиперпараметры оптимизаторов

### 1. Learning rate (скорость обучения):

Это один из ключевых параметров, определяющий размер шага при обновлении весов.

- Слишком маленькое значение приводит к медленной сходимости.
- Слишком большое значение может привести к колебаниям или даже расходящейся модели.

## Значения для экспериментов:

$\eta = 0.1, 0.01, 0.001.$

## 2. Batch size (размер мини-выборки):

- Маленький batch size (16 или 32) делает процесс обучения быстрее, но менее стабильным.
- Большие значения (64, 128) обеспечивают более устойчивую сходимость, но требуют больше вычислительных ресурсов.

## 4. Эксперименты с различными методами оптимизации

Шаги проведения экспериментов:

1. Постройте полносвязную нейронную сеть с двумя скрытыми слоями (например, с 128 и 64 нейронами). Используйте датасет MNIST или CIFAR-10.
2. Обучите модель, используя каждый из оптимизаторов: SGD, Momentum, RMSprop и Adam.
3. Проведите эксперименты с тремя значениями learning rate: 0.1, 0.01, 0.001.
4. Используйте три значения batch size: 16, 32, 64.

Пример кода для запуска обучения с различными оптимизаторами:

*Листинг 4 – Обучение с различными оптимизаторами*

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

# Создание модели
model = Sequential([
```

```

        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu'),
        Dense(64, activation='relu'),
        Dense(10, activation='softmax')
    ])
    # Список оптимизаторов
    optimizers = {
        'SGD': tf.keras.optimizers.SGD(learning_rate=0.01),
        'Momentum': tf.keras.optimizers.SGD(learning_rate=0.01,
momentum=0.9),
        'RMSprop': tf.keras.optimizers.RMSprop(learning_rate=0.01),
        'Adam': tf.keras.optimizers.Adam(learning_rate=0.01)
    }
    # Обучение с разными оптимизаторами
    for name, optimizer in optimizers.items():
        model.compile(optimizer=optimizer,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        print(f"Обучение с {name}")
        model.fit(train_images, train_labels, epochs=10,
batch_size=32, validation_data=(test_images, test_labels))

```

## 5. Визуализация и анализ результатов

Для каждого оптимизатора необходимо построить графики:

- Потерь (loss) на обучающей и тестовой выборках.
- Точности на обучении и тестировании.

Пример кода для построения графиков:

*Листинг 5 – Построение графиков*

```

import matplotlib.pyplot as plt

# Предположим, что есть списки потерь и точности
epochs = range(1, 11)
loss_sgd = [0.5, 0.4, 0.35, 0.3, 0.25, 0.22, 0.2, 0.18, 0.16, 0.15]
loss_adam = [0.45, 0.35, 0.28, 0.22, 0.2, 0.18, 0.16, 0.14, 0.13, 0.12]

plt.plot(epochs, loss_sgd, label='SGD')
plt.plot(epochs, loss_adam, label='Adam')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Сравнение потерь для различных оптимизаторов')
plt.legend()
plt.show()

```

## **6. Основные выводы**

- SGD подходит для малых датасетов, но может быть медленным на сложных задачах.
- Momentum ускоряет сходимость и позволяет преодолевать локальные минимумы.
- RMSprop и Adam хорошо работают в задачах с шумными градиентами и сложными архитектурами.

### **Рекомендации:**

- Для малых датасетов или задач с небольшим количеством параметров может быть достаточно SGD с моментумом.
- В задачах с глубокими сетями или большими объемами данных лучше использовать Adam или RMSprop.

## **Теоретический материал по 3 части**

### **1. Введение в анализ методов оптимизации**

После обучения нейронной сети с различными методами оптимизации и гиперпараметрами (learning rate, batch size) необходимо оценить, насколько эффективно работала каждая модель. Анализ позволяет:

- Определить, какой метод оптимизации быстрее сходится.
- Выявить, какие комбинации гиперпараметров обеспечивают лучшую точность.
- Оценить устойчивость модели к переобучению.

На этом этапе важно интерпретировать полученные результаты, представить их в виде таблиц, графиков и сделать обоснованные выводы.

## 2. Скорость сходимости: анализ графиков потерь

Одним из ключевых аспектов является **скорость сходимости** — то, как быстро модель уменьшает ошибку на обучающей и тестовой выборках.

### 2.1 Как интерпретировать график потерь

- Плавное снижение loss на обучающей и тестовой выборках → хороший показатель обучения.
- Разрыв между loss на обучающей и тестовой выборках → может указывать на переобучение.
- Пилообразные скачки loss → могут указывать на слишком высокий learning rate.

Пример графика потерь представлен в Листинге 6

*Листинг 6 – График потерь*

```
import matplotlib.pyplot as plt

epochs = range(1, 11)
loss_sgd = [0.5, 0.4, 0.35, 0.3, 0.25, 0.22, 0.2, 0.18, 0.16, 0.15]
loss_adam = [0.45, 0.35, 0.28, 0.22, 0.2, 0.18, 0.16, 0.14, 0.13, 0.12]

plt.plot(epochs, loss_sgd, label='SGD')
plt.plot(epochs, loss_adam, label='Adam')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Сравнение потерь для различных оптимизаторов')
plt.legend()
plt.show()
```

Если на графике виден резкий спад, значит модель быстро нашла

минимум. Если кривая loss остается высокой после нескольких эпох, это может указывать на неподходящий learning rate или неэффективный оптимизатор.

### 3. Анализ точности на обучении и тестировании

Второй важный показатель — точность (accuracy) или другие метрики качества, такие как F1-score. Эти метрики помогают понять, насколько хорошо модель классифицирует новые данные.

Пример графика точности представлен в листинге 7:

*Листинг 7 – График точности*

```
epochs = range(1, 11)
acc_sgd = [0.6, 0.65, 0.7, 0.72, 0.75, 0.77, 0.79, 0.8, 0.81, 0.82]
acc_adam = [0.65, 0.7, 0.75, 0.78, 0.8, 0.82, 0.83, 0.84, 0.85, 0.86]

plt.plot(epochs, acc_sgd, label='SGD')
plt.plot(epochs, acc_adam, label='Adam')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Сравнение точности для различных оптимизаторов')
plt.legend()
plt.show()
```

#### 3.1 Выявление переобучения

Если точность на обучающей выборке выше 95%, но на тестовой выборке остаётся низкой, модель переобучилась. Это может быть связано с:

- Слишком большим числом нейронов.
- Отсутствием регуляризации (Dropout, L2-регуляризация).
- Неподходящими значениями learning rate.



#### 4. Влияние batch size на обучение

Размер мини-выборки (batch size) влияет на стабильность градиента и вычислительную сложность:

- Маленькие batch size (16, 32)
- Быстро обновляют веса.
- Более шумные градиенты, что может усложнять сходимость.
- Большие batch size (64, 128, 256)
- Более точные градиенты, устойчивое обучение.
- Долгое обучение.

Результаты необходимо представить в виде таблицы 1:

Таблица 1 – Влияние на обучение

Batch Size	Скорость обучения (сек.)	Итоговая точность (%)
16	?	?
32	?	?
64	?	?

#### 5. Сравнительный анализ оптимизаторов

Таблица 2 – Сравнительный анализ

Оптимизатор	Время обучения (сек.)	Итоговая точность (%)	Стабильность
SGD	X (долгое/ быстрое/ среднее/ очень быстрое)		Колебания / Хорошая / Плохая / Отличная / Лучшая
Momentum	X (долгое/ быстрое/ среднее/ очень быстрое)		Колебания / Хорошая / Плохая / Отличная / Лучшая
RMSprop	X с (долгое/ быстрое/ среднее/ очень быстрое)		Колебания / Хорошая / Плохая / Отличная / Лучшая
Adam	X с (долгое/ быстрое/ среднее/ очень быстрое)		Колебания / Хорошая / Плохая / Отличная / Лучшая

Выводы по методам оптимизации:

- SGD подходит для небольших задач, но может быть медленным.

- Momentum улучшает сходимость и уменьшает колебания.
- RMSprop хорошо работает на сложных задачах.
- Adam является одним из самых эффективных методов, так как сочетает моментум и RMSprop.

## **6. Подготовка к защите**

На этапе защиты студенты должны:

1. Представить архитектуру своей сети.
2. Обосновать выбор метода оптимизации.
3. Показать сравнительные графики потерь и точности.
4. Объяснить, какие комбинации learning rate и batch size оказались лучшими.
5. Рассказать о сложностях, с которыми они столкнулись, и о том, как их решили.

## **7. Итоговые выводы**

- Выбор метода оптимизации влияет на скорость сходимости и качество модели.
- Подбор learning rate и batch size критически важен для успешного обучения.
- Adam и RMSprop показывают наилучшие результаты, но их стоит тестировать на разных задачах.
- Визуализация результатов позволяет легко выявить проблемы, такие как переобучение.