ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Лекция 5

Оптимизация обучения нейронных сетей

SGD, Momentum, RMSProp, Adam, learning-rate schedules и подбор гиперпараметров

Преподаватель: Аршын Алтыбай

Кафедра компьютерной инженерии,
Алматинский технологический университет, Алматы, Казахстан

Учебный год: 2026

> **Связь с лекцией 4.** Алгоритм обратного распространения ошибки позволяет вычислить $\nabla_\theta J$. Однако знание градиента ещё не определяет, *как именно* изменять параметры. Эту задачу решает оптимизатор. Мы разберём поведение нескольких алгоритмов, вычислим шаги вручную и научимся проектировать воспроизводимый эксперимент [6, 3].

1. Цель, результаты обучения и план

Цель: понять математические механизмы современных оптимизаторов и научиться выбирать режим обучения нейронной сети по измеримым данным, а не по догадке. По окончании лекции студент должен уметь:

1. выводить обновления SGD, Momentum, RMSProp, Adam и AdamW;
2. рассчитывать первые итерации оптимизатора вручную;
3. объяснять влияние масштаба градиента, обусловленности задачи и мини-пакета;
4. различать гиперпараметры алгоритма, параметры модели и параметры эксперимента;
5. применять scheduling, warm-up, gradient clipping и early stopping;
6. проводить корректное сравнение оптимизаторов при фиксированных данных и бюджете;
7. реализовывать оптимизаторы на NumPy и использовать их в PyTorch;
8. обнаруживать утечки данных, нестабильность обучения и ошибки конфигурации.

План: геометрия оптимизации; SGD; Momentum и Nesterov; адаптивные методы; Adam/AdamW; управление learning rate; инженерия обучения; программные эксперименты и вопросы для закрепления.

2. Постановка задачи и геометрия оптимизации

Обучение нейронной сети сводится к приближённому решению задачи

$$\theta^* \in \arg\min_{\theta \in \mathbb{R}^p} J(\theta), \qquad J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) + \lambda \Omega(\theta). \tag{1}$$

Здесь θ содержит веса и смещения, ℓ задаёт ошибку одного объекта, Ω — регуляризатор. Обычно глубокая сеть порождает невыпуклую J; глобальную сходимость для произвольной архитектуры и данных гарантировать нельзя [6].

2.1 Градиент, гессиан и локальная аппроксимация

Для малого перемещения $\Delta\theta$ справедливо разложение Тейлора

$$J(\theta + \Delta\theta) \approx J(\theta) + \nabla J(\theta)^\top \Delta\theta + \tfrac{1}{2}\Delta\theta^\top H(\theta)\Delta\theta, \tag{2}$$

где $H = \nabla^2 J$ — гессиан. Направление $-\nabla J$ обеспечивает уменьшение линейной части, но при слишком большом шаге квадратичный член может преобладать: поэтому большой learning rate способен приводить к увеличению потерь.

> **Важное различие** **Градиент** определяет локальное направление роста функции; **оптимизатор** задаёт правило шага; **скорость обучения** задаёт масштаб изменения параметров. Даже корректные градиенты не гарантируют устойчивого обучения при неверно выбранной скорости.

2.2 Пример: вытянутая квадратичная функция

Рассмотрим $J(w_1, w_2) = \frac{1}{2}(w_1^2 + 100w_2^2)$. Тогда

$$\nabla J = (w_1, 100w_2)^\top, \qquad H = \mathrm{diag}(1, 100). \tag{3}$$

Вторая координата имеет значительно более высокую кривизну. Для градиентного шага $w_1^{(t+1)} = (1-\eta)w_1^{(t)}$, $w_2^{(t+1)} = (1-100\eta)w_2^{(t)}$. Чтобы обе координаты убывали по модулю, необходимо $0 < \eta < 2/100 = 0.02$. При $\eta = 0.05$ вторая координата умножается на -4, то есть расходится. Узкие «долины» поверхности потерь затрудняют подбор единого шага.

3. Градиентный спуск и его стохастические варианты

3.1 Полный градиентный спуск (batch gradient descent)

Классическая итерация имеет вид

$$\theta_{t+1} = \theta_t - \eta_t \nabla J(\theta_t). \tag{4}$$

Если J имеет L-липшицев градиент, то для полного градиента при постоянном шаге $0 < \eta < 2/L$ выполняется неувеличение J; в невыпуклом случае это не обещает глобальный минимум. Вычисление полного градиента требует прохода по всей выборке, что для больших N дорого [6].

3.2 Stochastic gradient descent (SGD)

В SGD на итерации случайным образом выбирают наблюдение i_t:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \ell(f_{\theta_t}(x_{i_t}), y_{i_t}). \tag{5}$$

При независимой равномерной выборке и корректных условиях стохастический градиент является несмещённой оценкой градиента *средней эмпирической потери* (без регуляризации): $\mathbb{E}[g_t \mid \theta_t] = \nabla J_{\mathrm{data}}(\theta_t)$. Шум градиента может помогать выходить из узких областей, но также увеличивает колебания.

3.3 Mini-batch SGD

Практически используют мини-пакет B_t, $|B_t| = m$:

$$g_t = \frac{1}{m} \sum_{i \in B_t} \nabla_\theta \ell(f_{\theta_t}(x_i), y_i), \qquad \theta_{t+1} = \theta_t - \eta_t g_t. \tag{6}$$

При независимых одинаково распределённых градиентах дисперсия среднего примерно пропорциональна $1/m$; при выборке без возвращения точная формула учитывает конечность генеральной совокупности.

Метод	Размер пакета	Плюсы	Ограничения
Batch GD	N	Детерминированны эмпирический градиент	Дорогой шаг на большой выборке
SGD	1	Дешёвый шаг; шум может быть полезен	Высокая вариативность оценок
Mini-batch	$1 < m < N$	Векторизация, баланс затрат и шума	Требуется настроить m и η

3.4 Как считать количество обновлений

Пусть $N = 1000$, batch size $m = 64$, 10 эпох. При сохранении последнего неполного мини-пакета число шагов за эпоху $\lceil 1000/64 \rceil = 16$, всего 160 шагов. Если последний пакет отбрасывается, шагов будет 15 за эпоху. Сравнивать два обучения только по числу эпох недостаточно: у них может различаться число обновлений.

4. Momentum: накопление направления движения

4.1 Мотивация

Обычный SGD может «зигзагообразно» двигаться в узкой долине поверхности потерь. Momentum сглаживает краткосрочные изменения направления, накапливая вклад предыдущих градиентов [6, 7].

$$v_t = \beta v_{t-1} + g_t, \qquad \theta_{t+1} = \theta_t - \eta v_t, \quad v_0 = 0, \quad 0 \le \beta < 1. \tag{7}$$

Существует и другое распространённое соглашение $v_t = \beta v_{t-1} + (1-\beta) g_t$; оно меняет масштаб скорости и потому не должно незаметно смешиваться с формулой (7).

4.2 Разворачивание рекурсии

При $v_0 = 0$:

$$v_t = g_t + \beta g_{t-1} + \beta^2 g_{t-2} + \cdots + \beta^{t-1} g_1. \tag{8}$$

Следовательно, старые градиенты затухают геометрически. При $\beta = 0.9$ вклад, сделанный десять шагов назад, примерно равен $0.9^{10} \approx 0.349$ от соответствующего нового градиента.

> **Численный пример Momentum** Возьмём скалярный параметр, $g_1 = 2$, $g_2 = 1$, $\beta = 0.9$, $\eta = 0.1$, $\theta_0 = 3$. Тогда $v_1 = 2$, $\theta_1 = 3 - 0.1 \cdot 2 = 2.8$. Далее $v_2 = 0.9 \cdot 2 + 1 = 2.8$, $\theta_2 = 2.8 - 0.1 \cdot 2.8 = 2.52$. Второй шаг учитывает накопленное направление.

4.3 Nesterov accelerated gradient

Одна из распространённых форм Nesterov Momentum для полного градиента вычисляет градиент в прогнозируемой точке:

$$v_t = \beta v_{t-1} + \nabla J(\theta_t - \eta \beta v_{t-1}), \qquad \theta_{t+1} = \theta_t - \eta v_t. \tag{9}$$

У библиотек возможны алгебраически эквивалентные, но по-другому записанные варианты. При обсуждении «Nesterov» важно указать конкретную реализацию и способ обработки dampening/weight decay.

5. Адаптивные методы: AdaGrad и RMSProp

5.1 Проблема одного шага для всех координат

При сильно отличающихся масштабах градиентов полезно корректировать шаг каждой координаты отдельно. Это особенно значимо для редких признаков, разреженных embeddings и плохо обусловленных задач.

5.2 AdaGrad

Для каждой координаты j накапливается сумма квадратов градиентов:

$$s_{t,j} = s_{t-1,j} + g_{t,j}^2, \qquad \theta_{t+1,j} = \theta_{t,j} - \frac{\eta}{\sqrt{s_{t,j}} + \varepsilon} g_{t,j}. \tag{10}$$

Большие или часто встречающиеся градиенты увеличивают $s_{t,j}$ и уменьшают эффективный шаг. Недостаток: $s_{t,j}$ не убывает, и на длительном обучении шаг может стать слишком маленьким.

5.3 RMSProp

RMSProp заменяет накопление за всё время экспоненциальным скользящим средним:

$$s_t = \rho s_{t-1} + (1-\rho) g_t^{\odot 2}, \qquad \theta_{t+1} = \theta_t - \eta \frac{g_t}{\sqrt{s_t} + \varepsilon}. \tag{11}$$

Здесь операции применяются покомпонентно; $\rho \in [0, 1)$, например 0.9. RMSProp реагирует на недавнюю статистику квадратов градиентов и не накапливает все прошлые значения с единичным весом [3, 4].

> **Ручной шаг RMSProp** При $g_1 = 2$, $s_0 = 0$, $\rho = 0.9$ получим $s_1 = 0.1 \cdot 4 = 0.4$. Если $\eta = 0.01$ и ε мало, шаг $\Delta\theta \approx -0.01 \cdot 2/\sqrt{0.4} \approx -0.03162$. Начальный шаг значительно отличается от обычного SGD при $\eta = 0.01$; настройка одинакового η для разных оптимизаторов *не* делает их шаги эквивалентными.

6. Adam: Momentum и адаптивная шкала шага

Adam объединяет экспоненциальное сглаживание первого и второго моментов градиента [6, 3].

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \tag{12}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^{\odot 2}, \tag{13}$$

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \qquad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{14}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}. \tag{15}$$

Обычно $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. Коррекция смещения (bias correction) компенсирует старт обоих моментов с нуля: иначе оценки будут систематически занижены в первые итерации.

6.1 Ручной расчёт первого шага Adam

Возьмём $g_1 = 2$, $m_0 = v_0 = 0$. Тогда

$$m_1 = 0.1 \cdot 2 = 0.2, \qquad \widehat{m}_1 = 0.2/(1-0.9) = 2,$$

$$v_1 = 0.001 \cdot 4 = 0.004, \qquad \widehat{v}_1 = 0.004/(1-0.999) = 4.$$

При $\eta = 0.01$ получим $\Delta\theta = -0.01 \cdot 2/(2+10^{-8}) \approx -0.01$. Это частный пример для ненулевого скалярного градиента; последующие шаги зависят от истории g_t.

6.2 Проблемы и ограничения Adam

Adam часто эффективен на первых этапах обучения, но не является универсально лучшим методом. Для разных задач лучшие результаты на отложенной выборке может дать SGD с momentum. Кроме того, тип оптимизатора не заменяет нормализацию данных, корректное разделение выборки и выбор функции потерь. Сравнивать методы следует при разумной настройке learning rate каждого [5, 4].

7. AdamW и смысл weight decay

7.1 L2-регуляризация в обычном градиентном спуске

Для $J(\theta) = L(\theta) + \frac{\lambda}{2}\|\theta\|_2^2$ градиентный шаг равен

$$\theta_{t+1} = \theta_t - \eta(\nabla L(\theta_t) + \lambda\theta_t) = (1 - \eta\lambda)\theta_t - \eta\nabla L(\theta_t). \tag{16}$$

Для обычного SGD L2-регуляризация и multiplicative weight decay алгебраически совпадают при таком определении коэффициентов. В Adam они в общем случае *неэквивалентны*: L2-градиент участвует в оценках моментов и адаптивном масштабировании.

7.2 Decoupled weight decay (AdamW)

В AdamW отдельно выполняют адаптивный шаг по градиенту функции потерь и сжатие параметров:

$$\theta_{t+1} = (1 - \eta\lambda)\theta_t - \eta\frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}, \tag{17}$$

где m_t, v_t рассчитаны по ∇L, а не по градиенту $L + \lambda\|\theta\|^2/2$. В прикладной практике часто исключают смещения и параметры нормализации из weight decay; это инженерный выбор, который нужно документировать.

> **Не путайте две настройки** **Learning rate** определяет размер оптимизационного шага, а **weight decay** стремится уменьшить нормы некоторых параметров. Слишком сильное сжатие может привести к недообучению; отсутствие регуляризации в больших сетях иногда ухудшает обобщение.

8. Сравнение оптимизаторов и выбор метода

Метод	Идея	Типичная сила	Что настраивать
SGD	Шаг по шумному градиенту	Простая базовая линия	η, batch size
Momentu	Накопление направлений	Меньше зигзагов	η, β
AdaGrad	Сумма квадратов градиентов	Разреженные признаки	η, ε
RMSProp	EMA квадратов градиентов	Неодинаковые масштабы	η, ρ
Adam	EMA первого и второго моментов	Быстрый практический старт	η, β_1, β_2
AdamW	Adam + отделённый weight decay	Контролируемая регуляризация	η, λ

9. Learning rate: основной гиперпараметр

9.1 Три режима поведения

При чрезмерно малом η loss уменьшается медленно и ограниченный бюджет обучения расходуется неэффективно. При разумном η обучение обычно идёт быстрее. При слишком большом η потери колеблются, резко растут или становятся NaN. На практике полезно отслеживать не только конечный loss, но и кривую по *числу шагов оптимизации*.

9.2 Piecewise schedule и StepLR

Один вариант уменьшения шага:

$$\eta_t = \eta_0 \gamma^{\lfloor t/K \rfloor}, \qquad 0 < \gamma < 1, \tag{18}$$

где через каждые K эпох или шагов ставка уменьшается в γ^{-1} раз. Указание единицы измерения t обязательно: в библиотеках scheduler иногда вызывают после каждой эпохи, иногда после каждого мини-пакета.

9.3 Экспоненциальное убывание

$$\eta_t = \eta_0 e^{-kt}, \qquad k > 0. \tag{19}$$

Такой закон гладок, но при слишком большом k скорость быстро станет незначительной.

9.4 Cosine annealing

Для $t \in [0, T]$ часто используется

$$\eta_t = \eta_{\min} + \frac{\eta_{\max} - \eta_{\min}}{2} \left(1 + \cos \frac{\pi t}{T}\right). \tag{20}$$

В начале $\eta_0 = \eta_{\max}$, в конце $\eta_T = \eta_{\min}$. Cosine schedule не обещает оптимальности сам по себе, но обеспечивает плавную смену скорости [3, 5].

9.5 Warm-up

На первых W шагах можно линейно увеличивать скорость от 0 до $\eta_{\max}$:

$$\eta_t = \eta_{\max} \frac{t}{W}, \qquad t = 1, \ldots, W. \tag{21}$$

Warm-up особенно полезен в некоторых режимах с большим batch size и глубокими архитектурами. После прогрева задают основной schedule.

10. Batch size, число эпох и количество шагов

Гиперпараметры связаны. Большой mini-batch обеспечивает более стабильную оценку градиента, но требует больше памяти. Малый mini-batch может ухудшать загрузку ускорителя и усиливать шум. Число эпох само по себе не отражает вычислительные затраты без размера выборки и batch size.

Для N образцов и размера mini-batch m число обновлений за эпоху равно $\lceil N/m \rceil$ при сохранении последнего неполного пакета. Если $N = 5000$ и $m = 128$, это 40 обновлений, а при $m = 32$ — 157. Поэтому сравнение двух режимов по одним только эпохам может быть некорректно; полезно учитывать общее число обработанных образцов, шагов и wall-clock time.

10.1 Рекомендуемый эксперимент

Возьмите фиксированный датасет, архитектуру, seed и train/validation split. Обучите сеть с batch size 16, 64, 256, фиксируя метрики по шагам и по реальному времени. Повторите минимум для трёх seeds и сообщите среднее и разброс. Вывод «большой batch лучше» не следует делать по одной кривой.

11. Численная устойчивость и защита обучения

11.1 Gradient clipping

Для глобальной нормы градиента $\|g_t\|_2$ применяется правило

$$\widetilde{g}_t = g_t \min\left(1, \frac{c}{\|g_t\|_2}\right), \qquad c > 0. \tag{22}$$

Если $\|g_t\|_2 \le c$, градиент не меняется; иначе его длина уменьшается до c без изменения направления. Gradient clipping помогает контролировать большие шаги, но не решает все проблемы выбора архитектуры и learning rate.

11.2 Gradient accumulation

При ограниченной GPU-памяти можно суммировать градиенты нескольких малых мини-пакетов до одного optimizer.step(). Чтобы имитировать усреднение loss по объединённому пакету одинаковых размеров, обычно делят каждый loss на число шагов накопления. При неравных размерах нужно учитывать количество объектов; при BatchNorm и стохастических слоях накопление не обязательно идентично физически большому mini-batch.

11.3 Аномальные значения и логирование

Признаки нестабильности: NaN или ∞ в loss, резкий рост нормы градиента, постоянная вероятность одного класса, сильные колебания validation loss. Полезные проверки:

1. корректность нормализации входов и меток;
2. согласование функции потерь и выхода сети;
3. отсутствие повторного применения sigmoid перед `BCEWithLogitsLoss`;
4. разумный диапазон learning rate;
5. контроль формы тензоров и сохранение seed/версий библиотек.

12. Early stopping и проблема переобучения

Early stopping контролирует качество на validation-данных и прекращает обучение, когда существенного улучшения нет на протяжении заданного числа проверок (patience). Типичная процедура сохраняет *лучшую* модель по validation metric, а не просто параметры последней эпохи. Использовать test для выбора момента остановки нельзя: это приведёт к утечке информации из тестовой выборки.

> **Важно для воспроизводимости** Сохраните также optimizer state и scheduler state, если планируете *продолжать* обучение из checkpoint: одних весов модели для точного возобновления Adam недостаточно, так как нужны m_t, v_t и счётчик шагов.

13. Подбор гиперпараметров как экспериментальная задача

13.1 Какие величины подбирают

К *гиперпараметрам* относятся learning rate, batch size, число эпох, архитектура сети, weight decay, dropout rate и коэффициенты оптимизатора. *Параметры* — веса и смещения, полученные в результате обучения. *Метапараметры эксперимента* — seed, версии данных, распределение вычислительного бюджета и критерии остановки.

13.2 Grid Search и Random Search

Grid Search перебирает заранее заданную сетку значений. Random Search выбирает конфигурации из распределений. Для параметров, масштаб которых может отличаться на порядки, полезна логарифмическая шкала: например $\eta \in \{10^{-4}, 3 \cdot 10^{-4}, 10^{-3}, 3 \cdot 10^{-3}\}$. При ограниченном бюджете случайный поиск нередко эффективнее полной сетки, потому что не все параметры одинаково влияют на результат [3].

13.3 Разделение данных и добросовестное сравнение

1. **Train** — оценка градиентов и обновление весов.
2. **Validation** — выбор гиперпараметров, early stopping, выбор checkpoint.
3. **Test** — независимая финальная проверка уже выбранной конфигурации.

При нескольких сравниваемых запусках следует фиксировать предварительно определённую метрику, одинаковую подготовку данных и сопоставимый вычислительный бюджет. Для стохастических процедур полезно указывать среднее и стандартное отклонение результатов разных seed.

14. Реализация оптимизаторов с нуля на NumPy

В следующем примере минимизируется квадратичная функция $J(w_1, w_2) = \frac{1}{2}(w_1^2 + 25w_2^2)$. Реализация содержит полный GD, Momentum, RMSProp и Adam. Такой небольшой пример позволяет увидеть различия именно в правиле обновления, не смешивая их с backpropagation [1, 7].

Пример 1. Четыре оптимизатора для двумерной функции

```
import numpy as np

def grad(w):
    return np.array([w[0], 25.0 * w[1]])

def loss(w):
    return 0.5 * (w[0]**2 + 25.0 * w[1]**2)

def run(method, lr, steps=100):
    w = np.array([2.0, 2.0], dtype=float)
    m = np.zeros_like(w)
    v = np.zeros_like(w)
    history = [loss(w)]
    for t in range(1, steps + 1):
        g = grad(w)
```

Пример 1 (продолжение)

```
        if method == "sgd":
            step = g
        elif method == "momentum":
            m = 0.9 * m + g
            step = m
        elif method == "rmsprop":
            v = 0.9 * v + 0.1 * g**2
            step = g / (np.sqrt(v) + 1e-8)
        elif method == "adam":
            m = 0.9 * m + 0.1 * g
            v = 0.999 * v + 0.001 * g**2
            mh = m / (1 - 0.9**t)
            vh = v / (1 - 0.999**t)
            step = mh / (np.sqrt(vh) + 1e-8)
        else:
            raise ValueError(method)
        w -= lr * step
        history.append(loss(w))
    return w, np.array(history)

for method, lr in [("sgd", .04), ("momentum", .01),
                   ("rmsprop", .05), ("adam", .08)]:
    w, hist = run(method, lr)
    print(method, "loss:", f"{hist[-1]:.6f}")
```

Приведённые скорости обучения различаются намеренно: одинаковый η не означает одинаковую величину эффективного шага. Код показывает *пример траектории*, а не универсальный рейтинг оптимизаторов. Для честного сравнения на конкретной задаче следует подбирать learning rate отдельно и использовать одинаковый бюджет.

15. Практическое обучение сети в PyTorch

Следующий пример использует синтетические двумерные данные и `BCEWithLogitsLoss`. На выходе сети один *logit*; функция потерь сама выполняет численно устойчивую комбинацию sigmoid и binary cross-entropy. Отдельная sigmoid применяется только при преобразовании logit в вероятность на этапе оценки [5, 4].

Пример 2. MLP с AdamW и validation

```
import torch
from torch import nn
from torch.utils.data import DataLoader, TensorDataset
```

Пример 2 (продолжение)

```

torch.manual_seed(7)
X = torch.randn(1200, 2)
y = ((X[:, 0] * X[:, 1]) > 0).float().unsqueeze(1)

train_ds = TensorDataset(X[:900], y[:900])
val_ds = TensorDataset(X[900:], y[900:])
train_loader = DataLoader(train_ds, batch_size=64, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=128)

model = nn.Sequential(nn.Linear(2, 32), nn.ReLU(),
                      nn.Linear(32, 32), nn.ReLU(),
                      nn.Linear(32, 1))
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3,
                              weight_decay=1e-3)

best_val = float("inf")
best_state = None
for epoch in range(30):
    model.train()
    train_total = 0.0
    for xb, yb in train_loader:
        optimizer.zero_grad()
        logits = model(xb)
        loss = criterion(logits, yb)
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 2.0)
        optimizer.step()
        train_total += loss.item() * len(xb)
    model.eval()
    val_total = 0.0
    with torch.no_grad():
        for xb, yb in val_loader:
            val_total += criterion(model(xb), yb).item() * len(xb)
    val_loss = val_total / len(val_ds)
    if val_loss < best_val:
        best_val = val_loss
        best_state = {k: v.detach().clone()
                      for k, v in model.state_dict().items()}
    if (epoch + 1) % 5 == 0:
        print(epoch + 1, round(train_total / len(train_ds), 4),
              round(val_loss, 4))
model.load_state_dict(best_state)
```

15.1 Объяснение критических строк

`optimizer.zero_grad()` очищает накопленные градиенты прошлых мини-пакетов. `loss.backward()` вычисляет градиенты методом backpropagation. `clip_grad_norm_` ограничивает общую норму до обновления. `optimizer.step()` изменяет веса. Режимы `train()` и `eval()` имеют значение для Dropout и BatchNorm, а `torch.no_grad()` отключает создание графа вычислений при валидации.

В коде лучшие веса сохраняются по *validation loss*. Чтобы продолжать прерванное обучение буквально с той же стадии, необходимо дополнительно хранить состояния оптимизатора, scheduler и генераторов случайных чисел.

16. Практикум: дизайн исследования оптимизаторо

Предлагается следующая серия экспериментов с одной архитектурой MLP и фиксированным train/validation/test split.

1. Обучить с SGD (без momentum), SGD+Momentum и AdamW.
2. Для каждого алгоритма протестировать не менее трёх значений learning rate.
3. Выбрать лучшую конфигурацию по validation loss, не просматривая test.
4. Повторить лучшие конфигурации при трёх случайных seed.
5. Сравнить значение loss, accuracy, время, число шагов и устойчивость обучения.
6. Провести дополнительный эксперимент с weight decay и с clipping.
7. Представить результаты в таблице и построить график validation loss относительно числа optimizer steps.

Метод	LR	Val loss	Время, с	Test metric
SGD	настроить	после запуска	после запуска	один раз после выбора
SGD+Mom	настроить	после запуска	после запуска	один раз после выбора
AdamW	настроить	после запуска	после запуска	один раз после выбора

17. Частые ошибки и способы диагностики

1. **Loss не уменьшается**: проверьте learning rate, целевые метки, масштаб признаков, получение градиентов.
2. **Loss стал NaN**: уменьшите скорость, проверьте деление на нуль, неправильный log и выход модели.
3. **Train loss падает, validation растёт**: возможное переобучение; примените early stopping, регуляризацию, дополнительную проверку данных.
4. **У Adam и SGD совпадает LR, но результаты разные**: совпадающий численный LR не означает совпадающие шаги.
5. **Не удаётся воспроизвести результат**: фиксируйте seed, split, версии и состояние оптимизатора.
6. **Test улучшается после десятков экспериментов**: вероятно, тестовая выборка стала частью процесса подбора; нужен новый независимый test.

18. Основные выводы

1. Оптимизация и обратное распространение решают разные задачи: второе вычисляет градиенты, первая использует их для обновления весов.
2. Momentum использует историю градиентов, RMSProp адаптирует масштаб шага, Adam сочетает оба механизма.
3. AdamW отделяет weight decay от адаптивного градиентного обновления.
4. Learning rate, batch size и число шагов следует рассматривать совместно.
5. Важны корректная валидация, сохранение лучшей модели, воспроизводимость и осмысленная оценка вычислительных затрат.
6. Нет оптимизатора, одинаково лучшего для всех архитектур и датасетов.

19. Контрольные вопросы

1. Что такое функция цели $J(\theta)$ и какую роль играет регуляризатор?
2. Почему противоположное градиенту направление локально уменьшает функцию?
3. Как кривизна поверхности потерь влияет на максимальный допустимый learning rate?

4. Чем отличаются batch GD, SGD и mini-batch SGD?
5. Почему дисперсия мини-пакетного градиента уменьшается при росте batch size?
6. Выведите первые две итерации Momentum.
7. Что происходит при $\beta = 0$ в Momentum?
8. В чём интуиция метода Nesterov?
9. Как AdaGrad изменяет эффективный шаг со временем?
10. Чем RMSProp отличается от AdaGrad?
11. Запишите все четыре ключевые формулы Adam.
12. Для чего Adam использует bias correction?
13. Как интерпретировать β_1 и β_2?
14. В чём разница L2-регуляризации и decoupled weight decay при Adam?
15. Почему одинаковый learning rate разных оптимизаторов не делает эксперименты эквивалентными?
16. Что такое learning-rate schedule?
17. Как вычислить cosine annealing на половине периода?
18. Зачем применяют warm-up?
19. Как batch size влияет на число обновлений за эпоху?
20. Что такое gradient clipping и какие проблемы оно не решает?
21. Как работает gradient accumulation?
22. Что следует сохранять в checkpoint для продолжения обучения Adam?
23. Как работает early stopping и почему нельзя использовать test?
24. В чём разница Grid Search и Random Search?
25. Как организовать честное сравнение двух оптимизаторов при разных seed?

20. Задания для самостоятельной работы

1. Для $J(w_1, w_2) = \frac{1}{2}(w_1^2 + 36w_2^2)$ найдите диапазон постоянных η, обеспечивающий сжатие обеих координат при GD.
2. Реализуйте Momentum на NumPy и проверьте первые три шага вручную на скалярном градиенте.
3. Сравните AdaGrad и RMSProp при последовательности $g = (2, 2, 0, 0, 2)$.
4. Выведите bias-corrected оценки Adam при первых двух одинаковых градиентах.
5. Проведите обучение MLP в PyTorch с SGD, Momentum, AdamW на одинаковом датасете.
6. Постройте графики train/validation loss для трёх вариантов learning rate.

7. Проверьте влияние gradient clipping при искусственно завышенной скорости обучения.
8. Реализуйте early stopping и сохранение лучшей модели по validation loss.
9. Подготовьте таблицу экспериментов с не менее тремя seeds.
10. Напишите аналитическое заключение (1–2 страницы): какие факторы влияли на качество и вычислительные затраты сильнее остальных?

21. Рекомендуемая литература

Для интуитивного знакомства с градиентными шагами особенно полезны Траск [1] и Нильсен [7]. Практическая настройка обучения подробно рассматривается у Жерона [3], Шолле [4], Стивенса и соавторов [5]. Для более строгого анализа ландшафта оптимизации и стохастических методов см. Гудфеллоу, Бенджио и Курвилля [6]. Для практики простых сетей дополнительно рекомендуем руководство Рашида [2].

References

[1] Trask A. W. *Grokking Deep Learning*. Manning Publications, 2019. Русское издание: «Грокаем глубокое обучение». https://www.manning.com/books/grokking-deep-learning.

[2] Rashid T. *Make Your Own Neural Network*. 2016. Русское издание: «Глубокое обучение. Лёгкая теория и практика на Python» (название может варьироваться по изданию). https://makeyourownneuralnetwork.blogspot.com/.

[3] Géron A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 3rd ed., O'Reilly Media, 2022. Русское издание: «Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow». https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/.

[4] Chollet F. *Deep Learning with Python*. 2nd ed., Manning Publications, 2021. Русское издание: «Глубокое обучение на Python». https://www.manning.com/books/deep-learning-with-python-second-edition.

[5] Stevens E., Antiga L., Viehmann T. *Deep Learning with PyTorch*. Manning Publications, 2020. Русское издание: «Глубокое обучение с PyTorch». https://www.manning.com/books/deep-learning-with-pytorch.

[6] Goodfellow I., Bengio Y., Courville A. *Deep Learning*. MIT Press, 2016. Русское издание: «Глубокое обучение». https://www.deeplearningbook.org/.

[7] Nielsen M. A. *Neural Networks and Deep Learning*. 2015. Бесплатный онлайн-учебник: «Нейронные сети и глубокое обучение». http://neuralnetworksanddeeplearning.com/.