ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Лекция 6

Регуляризация нейронных сетей

Переобучение, L1/L2, Dropout, Batch Normalization и обобщающая способность

Преподаватель: Аршын Алтыбай

Кафедра компьютерной инженерии,
Алматинский технологический университет, Алматы, Казахстан

Учебный год: 2026

> **Связь с лекциями 4–5.** Backpropagation вычисляет градиенты, а SGD/Adam обновляют веса. Но минимальная ошибка на обучающей выборке *не равнозначна* способности правильно работать на новых данных. В этой лекции мы переходим от задачи оптимизации к задаче **обобщения** и изучаем способы борьбы с переобучением [6, 3].

1. Цель, результаты обучения и структура

Цель: научиться распознавать переобучение и недостаточное обучение, математически понимать методы регуляризации и применять их в воспроизводимом эксперименте. По завершении лекции студент должен уметь:

1. различать обучение и обобщение, эмпирический и ожидаемый риск;
2. читать training/validation curves и диагностировать underfitting/overfitting;
3. выводить градиенты L1 и L2 штрафов и понимать эффект weight decay;
4. объяснять Dropout на этапе обучения и оценки;
5. отличать Batch Normalization от явной регуляризации;
6. применять data augmentation, early stopping и разделение данных без утечек;
7. сравнивать методы по независимому тестовому протоколу;
8. использовать NumPy и PyTorch для проведения контролируемых экспериментов.

План: обобщение; диагностика кривых; L1/L2 и нормы; Dropout; нормализация; аугментация; остановка обучения; практические эксперименты; контрольные вопросы.

2. Обобщающая способность: главная цель машинного обучения

Пусть наблюдения (x_i, y_i) получены из неизвестного распределения $P(X, Y)$ и модель f_θ предсказывает y по x. **Эмпирический риск** на обучающей выборке $S = \{(x_i, y_i)\}_{i=1}^N$ равен

$$\widehat{R}_S(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i). \tag{1}$$

Популяционный (ожидаемый) риск определяется как

$$R(\theta) = \mathbb{E}_{(X,Y)\sim P}\big[\ell(f_\theta(X), Y)\big]. \tag{2}$$

Обучение оптимизирует доступную величину $\widehat{R}_S$, однако практическая цель — уменьшить R. Разность $R(\theta) - \widehat{R}_S(\theta)$ называется **разрывом обобщения** (generalization gap). Поскольку P неизвестно, R приближённо оценивают на независимых данных. Это принципиальное различие между *подгонкой* и *обучением закономерности* [6, 7].

> **Важное ограничение** Нельзя по одной только малой training loss делать вывод, что модель хорошо работает вне выборки. Модель может запомнить случайные особенности и шум.

2.1 Разделение train, validation и test

- **Train:** обновление весов и параметров preprocessing.
- **Validation:** выбор гиперпараметров, early stopping, выбор архитектуры и регуляризатора.
- **Test:** независимая финальная оценка *после* выбора модели.

При малой выборке используют кросс-валидацию. Для временных рядов необходимы хронологические разбиения. Для данных одного пациента или пользователя предотвращают попадание связанных наблюдений в разные части разбиения.

2.2 Численный пример разрыва обобщения

Модель A имеет accuracy на train 99.5%, на validation 70%; модель B — 92% и 89% соответственно. При одинаковых условиях сбора данных и корректном протоколе модель B выглядит предпочтительнее, хотя её training accuracy ниже. Для классификации с дисбалансом классов следует анализировать также precision/recall/F1 и метрики калибровки, а не только accuracy.

3. Недостаточное обучение, переобучение и баланс смещения и вариативности

Underfitting проявляется, когда модель плохо описывает даже обучающие данные. Возможные причины: слишком простая архитектура, недостаточное обучение, слишком сильная регуляризация, неверные признаки. **Overfitting** проявляется, когда на обучении качество улучшается, но на validation начинает ухудшаться. Причины: слишком большая гибкость, малая выборка, шум, утечки или чрезмерное число настроек по validation.

3.1 Кривые обучения

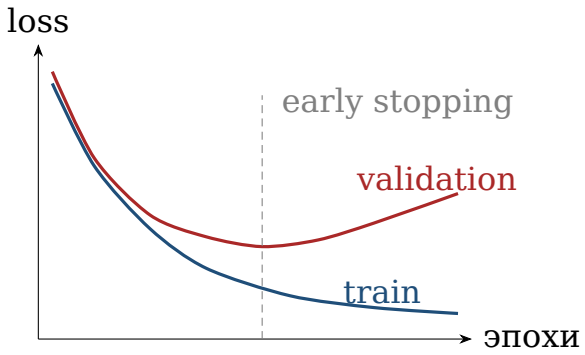


График схематичен: точный минимум validation loss зависит от данных и не обязан наступать после фиксированного числа эпох. Оптимальную эпоху выбирают по заранее определённому критерию.

3.2 Bias–variance: интуиция и ограничения

В квадратной ошибке регрессии при определённых предпосылках можно получить разложение ожидаемой ошибки на квадрат смещения, дисперсию оценки и неизбежный шум. Высокое смещение связано с систематическими ошибками простой модели; высокая вариативность — с чувствительностью к конкретной обучающей выборке. Для современных глубоких сетей зависимость между сложностью и ошибкой не всегда монотонна;

это полезная интуиция, но не универсальная теорема [6].

4. Общая формула регуляризованного обучения

Регуляризованная функция цели:

$$J(\theta) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)}_{\text{ошибка на данных}} + \lambda \underbrace{\Omega(\theta)}_{\text{штраф за сложность}}, \qquad \lambda \geq 0. \tag{3}$$

Параметр λ — гиперпараметр: при $\lambda = 0$ явного штрафа нет; увеличение λ усиливает давление на веса. Точное определение штрафа и масштаба λ нужно согласовывать с реализацией loss (сумма или среднее) [6, 3].

5. L2-регуляризация и weight decay

Пусть $\Omega_2(w) = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}\sum_{j=1}^{p} w_j^2$. Тогда

$$J(w) = L(w) + \frac{\lambda}{2}\|w\|_2^2, \qquad \nabla_w J = \nabla_w L + \lambda w. \tag{4}$$

Для обычного градиентного спуска

$$w_{t+1} = w_t - \eta(\nabla L(w_t) + \lambda w_t) = (1 - \eta\lambda)w_t - \eta\nabla L(w_t). \tag{5}$$

Множитель $(1 - \eta\lambda)$ сжимает веса, если $0 < \eta\lambda < 1$. Это связано со сглаживанием зависимости выходов от весов, хотя меньшие веса сами по себе не гарантируют лучшего обобщения.

> **L2 и AdamW не одно и то же** Для SGD без адаптивной предобработки градиента L2-штраф и классический weight decay могут давать эквивалентные обновления при согласованных коэффициентах. Для Adam добавление λw к градиенту и *decoupled weight decay* (AdamW) не совпадают: адаптивное масштабирование действует по-разному [3, 5].

5.1 Ручной расчёт L2

Пусть $w = 2$, градиент данных $\nabla L(w) = 0.3$, $\eta = 0.1$, $\lambda = 0.2$. Тогда $\nabla J = 0.3 + 0.2 \cdot 2 = 0.7$ и $w_{\text{new}} = 2 - 0.1 \cdot 0.7 = 1.93$. Без регуляризации вес стал бы 1.97. Числа иллюстрируют один шаг, но не доказывают преимущества на тестовой выборке.

5.2 Какие параметры штрафовать?

Обычно применяют штраф к матрицам весов линейных/сверточных слоёв. Смещения и параметры масштабирования нормализации часто исключают из weight decay. Выбор зависит от архитектуры и практики фреймворка.

6. L1-регуляризация и разреженные модели

L1-штраф определяется как

$$\Omega_1(w) = \|w\|_1 = \sum_{j=1}^{p} |w_j|, \qquad J(w) = L(w) + \lambda \|w\|_1. \tag{6}$$

Для $w_j \neq 0$: $\partial |w_j| / \partial w_j = \operatorname{sign}(w_j)$. В нуле классическая производная отсутствует, но существует *субдифференциал* $\partial |0| = [-1, 1]$. В частности,

$$0 \in \nabla L(w) + \lambda \partial \|w\|_1 \tag{7}$$

является необходимым условием оптимальности для выпуклой дифференцируемой L. L1 способен приводить к нулевым коэффициентам и используется для отбора признаков или получения разреженности.

6.1 Почему L1 способствует нулевым коэффициентам?

Для простой квадратичной задачи

$$\min_w \frac{1}{2}(w - a)^2 + \lambda |w| \tag{8}$$

решение имеет вид **soft-thresholding**

$$w^* = \operatorname{sign}(a) \max(|a| - \lambda, 0). \tag{9}$$

Если $a = 0.3$ и $\lambda = 0.5$, то $w^* = 0$; если $a = 2$ и $\lambda = 0.5$, то $w^* = 1.5$. Это точный результат именно для указанной задачи, а не для произвольной глубокой сети.

6.2 Сравнение L1 и L2

Свойство	L1	L2
Штраф	$\sum_j \lvert w_j \rvert$	$\frac{1}{2}\sum_j w_j^2$
Градиент	Субградиент в нуле	Гладкий: w
Характерны эффект	Поощрение разреженности	Сжатие весов
Ограничени	Негладкость; настройка λ	Не делает веса строго нулевыми обычным GD

7. Dropout как стохастическая регуляризация

Во время обучения случайно отключают часть активаций. При **inverted dropout** с вероятностью отключения p и вероятностью сохранения $q = 1-p$:

$$\widetilde{h}_j = \frac{m_j}{q} h_j, \qquad m_j \sim \text{Bernoulli}(q). \tag{10}$$

Тогда при фиксированной активации h_j получаем $\mathbb{E}[\widetilde{h}_j] = h_j$. Поэтому на этапе inference маска отключается, и дополнительное масштабирование обычно не требуется. Dropout уменьшает коадаптацию отдельных нейронов и действует как шум в обучении [8, 4].

7.1 Численный пример Dropout

Пусть $h = (2, 4, 6)$ и $p = 0.5$, так что $q = 0.5$. Если маска $m = (1, 0, 1)$, то $\widetilde{h} = (4, 0, 12)$. Это *одна случайная реализация*; в среднем по многим маскам математическое ожидание каждой координаты равно исходной активации.

7.2 Дисперсия и правильное место применения

Для независимой Bernoulli-маски

$$\text{Var}(\widetilde{h}_j) = h_j^2 \frac{p}{1-p}. \tag{11}$$

При $p \to 1$ шум становится чрезвычайно сильным. Значение p подбирают по validation, а не назначают максимально большим. Не ко всем слоям Dropout одинаково полезен: при наличии BatchNorm, небольших CNN или attention-моделей настройка требует эксперимента.

Пример 1. Inverted Dropout на NumPy

```python
import numpy as np
rng = np.random.default_rng(42)
h = np.array([2.0, 4.0, 6.0])
p = 0.5
mask = rng.binomial(1, 1-p, size=h.shape)
h_train = h * mask / (1-p)
h_eval = h.copy()
print("mask:", mask)
print("train:", h_train)
print("eval:", h_eval)
```

Код использует один фиксированный генератор и демонстрирует именно способ вычисления; полученная маска зависит от реализации генератора NumPy. На этапе оценки нельзя оставлять модель в режиме обучения.

8. Batch Normalization: стабилизация внутренних представлений

Batch Normalization (BN) нормализует активации по мини-пакету. Для одного признака $x_1, \dots, x_m$:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2, \tag{12}$$

$$\widehat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \qquad y_i = \gamma \widehat{x}_i + \beta. \tag{13}$$

Здесь $\varepsilon > 0$ обеспечивает численную устойчивость; γ и β обучаются. Для многомерных тензоров оси нормализации зависят от реализации (например, каналы в CNN) [9, 6].

8.1 Ручной расчёт BN

Пусть мини-пакет содержит $x = (2, 4, 6, 8)$. Тогда $\mu_B = 5$, а дисперсия с делителем m:

$$\sigma_B^2 = \frac{9 + 1 + 1 + 9}{4} = 5. \tag{14}$$

При $\varepsilon \approx 0$, $\gamma = 1$, $\beta = 0$ получаем приближённо $\widehat{x} = (-1.342, -0.447, 0.447, 1.342)$. На практике ε всегда положительно.

8.2 Train и evaluation — разные вычисления

Во время обучения BN использует статистики текущего мини-пакета и обновляет running estimates. При inference обычно применяют накопленные оценки среднего и дисперсии. Поэтому режимы `model.train()` и `model.eval()` необходимо переключать осмысленно.

8.3 BatchNorm не эквивалентен Dropout

Dropout добавляет шум через случайные маски; BN нормализует промежуточные активации. В ряде моделей BN создаёт сопутствующий регуляризующий эффект из-за шумных batch-статистик, но его главная роль связана с изменением геометрии/стабильности оптимизации. Объяснение преимуществ BN исключительно «устранением internal covariate shift» не является исчерпывающим [9].

8.4 Layer Normalization и малые пакеты

LayerNorm нормализует признаки отдельного объекта и не зависит от распределения объектов внутри batch так же, как BN; поэтому часто применяется в трансформерах. GroupNorm подходит, например, для CNN с маленьким batch size. Выбор нормализации определяется архитектурой и ограничениями вычислений.

9. Аугментация данных: расширение эффективной выборки

Data augmentation создаёт допустимые преобразования обучающих объектов при сохранении целевой метки. Для изображений используют отражения, кадрирование, малые повороты и вариации освещённости, но не все преобразования сохраняют смысл задачи. Например, зеркальное отражение может изменить смысл изображения медицинской стороны тела или дорожного знака.

9.1 Аугментация как регуляризация

Пусть $T \sim \mathcal{T}$ — случайное преобразование. Тогда модель минимизирует приближение риска

$$\widehat{R}_{\text{aug}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{T \sim \mathcal{T}} \left[\ell(f_\theta(T(x_i)), y_i) \right]. \tag{15}$$

Практически математическое ожидание приближают выборкой случайных преобразований в эпохах обучения. *Validation/test обычно не подвергаются случайной обучающей аугментации*; к ним применяют детерминированную подготовку входа.

9.2 Специфика текста и временных рядов

В NLP замена слов на синонимы может менять эмоциональный тон, логическую структуру и класс. Во временных рядах инверсия порядка времени или перестановка значений часто уничтожает причинную структуру. Допустимость аугментации должна обосновываться предметной областью.

10. Early Stopping как неявное управление сложностью

Если validation loss перестала улучшаться, можно остановить обучение и восстановить веса лучшей эпохи. **Patience** задаёт число последовательных проверок без улучшения; **min_delta** — минимальную величину улучшения. Нельзя забывать, что лучшая *эпоха* и последняя *эпоха* не обязательно совпадают [3, 4].

10.1 Пример выбора эпохи

Эпоха	1	2	3	4	5	6	7
Validation loss	0.60	0.49	0.42	0.39	0.41	0.43	0.45

При отсутствии ограничений по минимальному улучшению лучшая эпоха — 4. При patience $= 3$ остановка произойдёт после трёх следующих проверок без улучшения, то есть после эпохи 7, но веса рекомендуется восстановить до состояния эпохи 4.

11. Другие инструменты контроля сложности

11.1 Уменьшение архитектуры и ограничение времени обучения

Меньше слоёв и нейронов зачастую сокращают вычислительную стоимость и могут уменьшать переобучение, но слишком малая архитектура приводит к underfitting. Ограничение числа эпох иногда действует как неявная регуляризация.

11.2 Шум и label smoothing

Добавление небольшого шума в допустимых признаках может повысить устойчивость к вариациям входа. Для K классов label smoothing заменяет one-hot целевое распределение q на

$$q'_k = (1-\alpha)q_k + \frac{\alpha}{K}, \qquad 0 \le \alpha < 1. \tag{16}$$

Это уменьшает тенденцию чрезмерно уверенных предсказаний, но не гарантирует лучшую калибровку в любой задаче.

11.3 Ансамбли и transfer learning

Усреднение предсказаний нескольких моделей иногда уменьшает дисперсию, но увеличивает стоимость inference. Transfer learning использует ранее обученные представления и может быть полезен при недостатке размеченных данных, однако источник и целевой домен должны быть сопоставимы [4, 3].

12. Что выбрать на практике: диагностическая таблица

Наблюдение	Возможная причина	Разумный эксперимент
Высокие train/val loss	Недостаточная мощность/неудачная оптимизация	Увеличить мощность или улучшить обучение
Train loss падает, val растёт	Переобучение	Early stopping, weight decay, аугментация
Обучение нестабильно	Большой шаг, шумный batch	Уменьшить learning rate, нормализовать
Большой разрыв на подгруппах	Shift, дисбаланс	Проверить разбиение и группы, собрать данные
Переобучение при малых данных	Нехватка разнообразия	Допустимая augmentation, transfer learning

13. Практика на PyTorch: контролируемое сравнение

В следующем примере мы генерируем двумерные данные и сравниваем несколько режимов MLP: без регуляризации, с weight decay и с Dropout. Эксперимент учебный: целью является *дизайн сравнения*, а не гарантия, что один метод всегда окажется лучше.

Пример 2. Генерация данных и корректное разбиение

```
import numpy as np
import torch
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

seed = 17
np.random.seed(seed)
torch.manual_seed(seed)
X, y = make_moons(n_samples=600, noise=0.30,
                  random_state=seed)
X_train, X_tmp, y_train, y_tmp = train_test_split(
```

Пример 2 (продолжение)

```
    X, y, test_size=0.40, stratify=y, random_state=seed)
X_val, X_test, y_val, y_test = train_test_split(
    X_tmp, y_tmp, test_size=0.50,
    stratify=y_tmp, random_state=seed)

scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

def to_tensors(X, y):
    return (torch.tensor(X, dtype=torch.float32),
            torch.tensor(y, dtype=torch.float32).view(-1, 1))

train_x, train_y = to_tensors(X_train, y_train)
val_x, val_y = to_tensors(X_val, y_val)
test_x, test_y = to_tensors(X_test, y_test)
print(train_x.shape, val_x.shape, test_x.shape)
```

Ожидаемый вывод

```
torch.Size([360, 2]) torch.Size([120, 2]) torch.Size([120, 2])
```

Мы обучаем StandardScaler *только* на train-выборке; это исключает утечку информации о статистиках validation/test. Генерация примера фиксирует random seed, однако точное повторение числовых метрик между версиями библиотек и оборудованием не гарантируется.

Пример 3. MLP и три режима регуляризации

```
from torch import nn
from copy import deepcopy

class MLP(nn.Module):
    def __init__(self, dropout=0.0, batchnorm=False):
        super().__init__()
        parts = [nn.Linear(2, 64)]
        if batchnorm:
            parts.append(nn.BatchNorm1d(64))
        parts += [nn.ReLU(), nn.Dropout(dropout),
                  nn.Linear(64, 64), nn.ReLU(),
                  nn.Linear(64, 1)]
        self.net = nn.Sequential(*parts)

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

configs = {
    "baseline": dict(dropout=0.0, wd=0.0),
    "weight_decay": dict(dropout=0.0, wd=1e-2),
```

Пример 3 (продолжение)

```
    "dropout": dict(dropout=0.3, wd=0.0),
}

def train_one(cfg, epochs=250, patience=25):
    torch.manual_seed(123)  # comparable initialization
    model = MLP(dropout=cfg["dropout"])
    loss_fn = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.AdamW(
        model.parameters(), lr=0.01,
        weight_decay=cfg["wd"])
    best_val = float("inf")
    best_state = None
    stale = 0
    history = {"train": [], "val": []}

    for epoch in range(epochs):
        model.train()
        optimizer.zero_grad()
        logits = model(train_x)
        train_loss = loss_fn(logits, train_y)
        train_loss.backward()
        optimizer.step()

        model.eval()
        with torch.no_grad():
            val_loss = loss_fn(model(val_x), val_y).item()
        history["train"].append(train_loss.item())
        history["val"].append(val_loss)

        if val_loss < best_val - 1e-5:
            best_val = val_loss
            best_state = deepcopy(model.state_dict())
            stale = 0
        else:
            stale += 1
        if stale >= patience:
            break

    model.load_state_dict(best_state)
    model.eval()
    with torch.no_grad():
        preds = (torch.sigmoid(model(test_x)) >= 0.5)
        test_acc = (preds == test_y.bool()).float().mean().item()
    return model, history, test_acc

results = {}
for name, cfg in configs.items():
    model, history, acc = train_one(cfg)
    results[name] = (history, acc)
    print(name, "epochs", len(history["val"]),
          "test_accuracy", round(acc, 3))
```

Методологическое замечание В этом компактном примере тестовая оценка печатается для наглядности. В настоящем исследовании нельзя многократно менять гиперпараметры по увиденному test accuracy. Нужно выбрать конфигурацию *по validation*, затем *однократно* оценить выбранную модель на test. Для честного сравнения следует повторять эксперименты по нескольким seed и указывать среднее и разброс метрик.

13.1 Визуализация потерь

Пример 4. Графики train/validation loss

```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 3, figsize=(12, 3.3))
for ax, (name, (hist, acc)) in zip(axes, results.items()):
    ax.plot(hist["train"], label="train")
    ax.plot(hist["val"], label="validation")
    ax.set_title(f"{name}, test acc={acc:.3f}")
    ax.set_xlabel("epoch")
    ax.set_ylabel("BCE loss")
    ax.legend()
plt.tight_layout()
plt.show()
```

14. Практика: проверка Batch Normalization

Вставка BatchNorm между Linear и ReLU может улучшить устойчивость обучения, но выигрыш зависит от batch size, learning rate и задачи. Следующий фрагмент показывает явное различие между train и eval:

Пример 5. BatchNorm: различие режимов

```
model = MLP(dropout=0.25, batchnorm=True)
model.train()
output_train = model(train_x[:32])
model.eval()
with torch.no_grad():
    output_eval = model(train_x[:32])
print(output_train.shape, output_eval.shape)
```

Ожидаемый вывод

```
torch.Size([32, 1]) torch.Size([32, 1])
```

Обе формы совпадают, но численные значения могут различаться вследствие dropout и использования batch-статистик против накопленных средних. Для BatchNorm число примеров в пакете должно соответствовать требованиям реализации; при крайне малых пакетах часто выбирают LayerNorm или GroupNorm.

15. План лабораторного исследования

1. Выберите задачу бинарной классификации с достаточным числом наблюдений.
2. Зафиксируйте train/validation/test, preprocessing и метрики.
3. Обучите baseline и сохраните графики потерь.
4. Сравните $\lambda \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ для weight decay.
5. Сравните Dropout $p \in \{0, 0.2, 0.4\}$.
6. Сравните MLP с BatchNorm и без неё при прочих равных.
7. Примените early stopping, сохраните лучшую эпоху.
8. Повторите выбранные конфигурации минимум с тремя независимыми seed.
9. По validation выберите модель, затем оцените её на test.
10. Постройте итоговую таблицу: train/val метрики, число эпох, время, среднее и разброс результатов.

В отчёте обязательно поясните, *почему* улучшение или ухудшение каче-

ства на конкретных данных может отличаться от общего ожидания.

16. Частые ошибки и диагностика

1. **Нормализация до разбиения данных.** Это способна вызвать data leakage.
2. **Настройка по test.** Результат становится оптимистично смещённым.
3. **Dropout не отключён на inference.** Предсказания остаются случайными.
4. **BN оставлена в train-режиме.** Ответы зависят от состава batch.
5. **Слишком сильная регуляризация.** Возникает underfitting.
6. **Разные архитектуры/seed в сравнении.** Трудно приписать эффект одному методу.
7. **Использование accuracy при дисбалансе.** Может скрыть ошибки на редком классе.
8. **Не сохранена лучшая модель.** Теряются результаты early stopping.
9. **Ложная интерпретация BN как универсальной регуляризации.** Её основная функция иная.

17. Ключевые выводы

- Цель обучения — обобщение на новые данные, а не минимизация тренировочной ошибки любой ценой.
- L2 ограничивает масштаб весов; L1 способствует разреженности.
- Dropout вводит случайные маски, и его поведение различается между train и eval.
- BatchNorm стабилизирует представления/оптимизацию; сама по себе не гарантирует отсутствие overfitting.
- Data augmentation должна сохранять смысл целевой метки.
- Early stopping выбирает лучшую эпоху по validation, не по test.
- Все выводы о пользе методов требуют честного протокола и воспроизводимого сравнения.

18. Контрольные вопросы

1. Что такое ожидаемый риск и чем он отличается от эмпирического?
2. Дайте определение generalization gap.
3. Что такое underfitting и overfitting?
4. Почему маленькая training loss не гарантирует хорошую test accuracy?
5. Для чего нужна validation-выборка?
6. Почему тестовый набор нельзя использовать для настройки гиперпараметров?
7. Запишите формулу L2-регуляризованной функции потерь.
8. Выведите один шаг SGD с L2 штрафом.
9. Чем weight decay в AdamW отличается от L2-добавки к градиенту Adam?
10. Как определяется L1-норма?
11. Что такое субградиент функции $|w|$ в точке $w = 0$?
12. Почему L1 может порождать нулевые коэффициенты?
13. В чём смысл inverted dropout?
14. Почему деление на $1 - p$ нужно во время обучения?
15. Как связаны dropout probability и дисперсия активаций?
16. Как следует обращаться с Dropout при inference?
17. Запишите формулы BatchNorm для мини-пакета.
18. Для чего нужны параметры γ и β в BN?
19. Почему режимы train/eval важны для BN?
20. В чём отличие BatchNorm и LayerNorm?
21. Что такое data augmentation и какие преобразования могут менять метку?
22. Для чего нужны patience и min_delta?
23. Почему нужно восстанавливать веса лучшей эпохи?
24. Как подобрать коэффициент регуляризации без утечки данных?
25. Почему однократное улучшение метрики не доказывает эффективности метода?

19. Задания для самостоятельной работы

1. Для $w = (1, -2, 3)$ вычислите $\|w\|_1$, $\|w\|_2^2$ и обе штрафные функции при $\lambda = 0.1$.
2. Повторите численный расчёт L2-обновления для $\eta = 0.05$ и $\lambda = 0.5$.
3. Докажите формулу дисперсии inverted dropout для одной фиксированной активации.
4. Выполните ручной расчёт BatchNorm для данных $(1, 1, 3, 3)$ и $\varepsilon = 10^{-5}$.
5. Сравните L1 и L2 на простой регрессионной задаче с коррелированными признаками.
6. Реализуйте собственный цикл early stopping и сохранение лучшей модели.
7. Постройте training/validation curves для трёх коэффициентов weight decay.
8. Исследуйте влияние Dropout $p = 0, 0.2, 0.5$ при одинаковой архитектуре.
9. Проведите абляцию: baseline, +L2, +Dropout, +BN, +augmentation.
10. Подготовьте краткий доклад «Почему regularization не заменяет качественный dataset и корректный протокол оценки?».

20. Рекомендуемая литература

Основной текст опирается на математическую базу [6, 7], практические подходы [3, 4, 5] и вводные руководства [1, 2]. Две оригинальные научные статьи приведены отдельно для Dropout и Batch Normalization.

Список литературы

[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 зависит от издания). 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 учебник. http://neuralnetworksanddeeplearning.com/.

[8] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15 (2014), 1929–1958. https://jmlr.org/papers/v15/srivastava14a.html.

[9] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of ICML*, 2015, 448–456. https://proceedings.mlr.press/v37/ioffe15.html.