МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

**Лабораторная работа №1**

по дисциплине«Разработка нейросетевых систем»

Тема: «Введение в DL»  
Вариант 9

ИСПОЛНИТЕЛЬ: \_\_\_Очеретная С.В.\_\_

ФИО

группа ИУ5-25М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

ПРЕПОДАВАТЕЛЬ: \_\_\_\_\_Канев А.И.\_\_\_\_

ФИО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

"\_\_"\_\_\_\_\_\_\_\_\_2024 г.

Москва - 2024

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Задание

Ознакомиться с фреймворком машинного обучения PyTorch и выполнить три задания:

1. Регрессия по теореме универсальной аппроксимации, ручное дифференцирование;
2. Бинарная классификация с помощью автодиффиренцирования PyTorch;
3. Обучить полносвязную нейронную сеть классификации 3 классов изображений из набора данных CIFAR100 и затем повысить точность на тестовой выборке. Классы из CIFAR100 по варианту 9:

1 класс = номер группы + 15 = 25 + 15 = **40**

2 класс = номер варианта + 56 = 9 + 56 = **65**

3 класс = номер варианта + 21 = 9 + 21 = **30**

# Ход работы

## Часть 1. Задача регрессии по теореме универсальной аппроксимации, ручное дифференцирование

### Генерация выборки и инициализация параметров нейронной сети

Генерируем выборку в виде возрастающей арифметической прогрессии. Получим X – массив из 500 элементов в интервале [-0,5; 0,5) с шагом 0,01 (каждый элемент повторяется 5 раз).

Генерируем целевой признак y (массив из 500 элементов).

Генерируем массив, заполненный случайными нормальными величинами со стандартным отклонением 0,05. К элементам этого массива прибавляем элементы массива y.

Строим 2 графика (рис. 1) для зависимостей X от y (черная пунктирная линия) и X от yn (синяя сплошная линия)..44

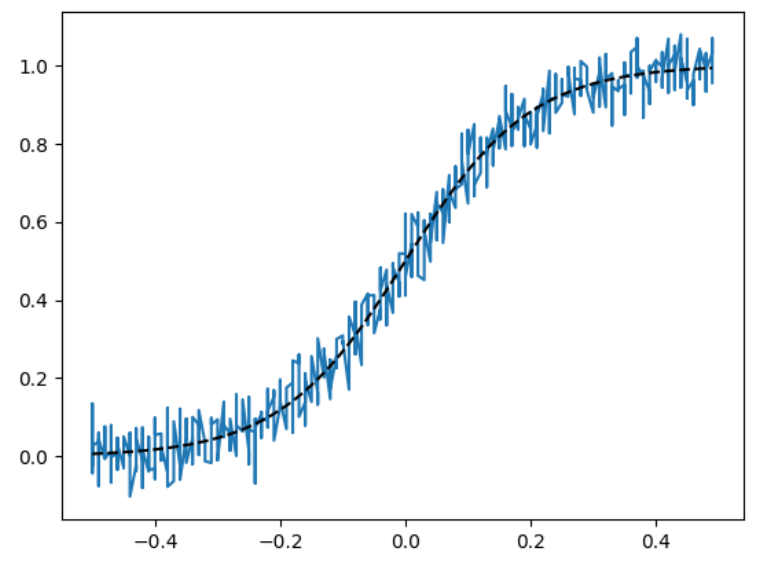


Рис. 1. Зависимости X от y, X от yn

Создаем 2 тензора из многомерных массивов X, yn (с помощью функции reshape меняем размерность массивов c 1x500 на 500x1 (то есть данные теперь не в 1ой строке, а в 1ом столбце, число строк с помощью -1 высчитывается автоматически).

Генерируем weights\_1, weights\_2 - тензоры размерностью 1х64 в диапазоне от -0,05 до 0,05 и тензоры bias\_1 (64 нуля), bias\_2 (1 ноль).

### Обучение нейронной сети задачи регрессии

Определяем функцию нелинейности и функцию потерь. Задаем функции для прямого и обратного прохода. Определяем функцию градиентного спуска по всей обучающей выборке.

Проводим обучение при 50 тысячах итераций градиентного спуска (50 тысяч эпох).

В цикле для каждых 10к эпох построили графики (рис. 2). Пунктирной линией здесь обозначена исходная выборка. На рис. 2 видно, что с увеличением числа эпох точность повышается (график с выходными данными обучения становится все более похож на график с исходными данными). Также можно заметить, что при большом количестве эпох график начинает полностью повторять исходный – это означает, что возникло переобучение.

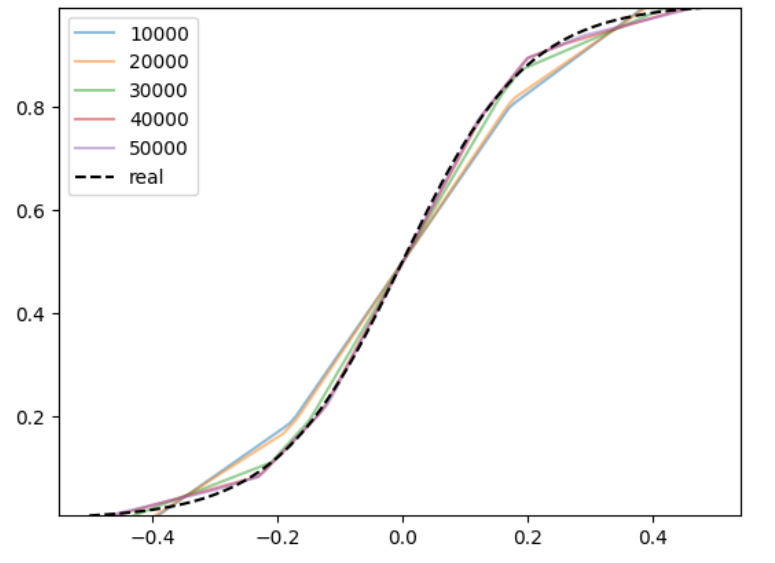


Рис. 2. Результаты работы алгоритма

Среднее значение потерь cur\_loss.numpy().mean() получилось 0,0023311719

## Часть 2. Бинарная классификация с помощью автодиффиренцирования PyTorch

### Генерация выборки и инициализация параметров нейронной сети

Генерируем выборку и создаем модель нейронной сети для решения задачи классификации, у которой 2 входа (x, y), скрытый слой (16 нейронов) и 1 выход (прогнозируемый класс).

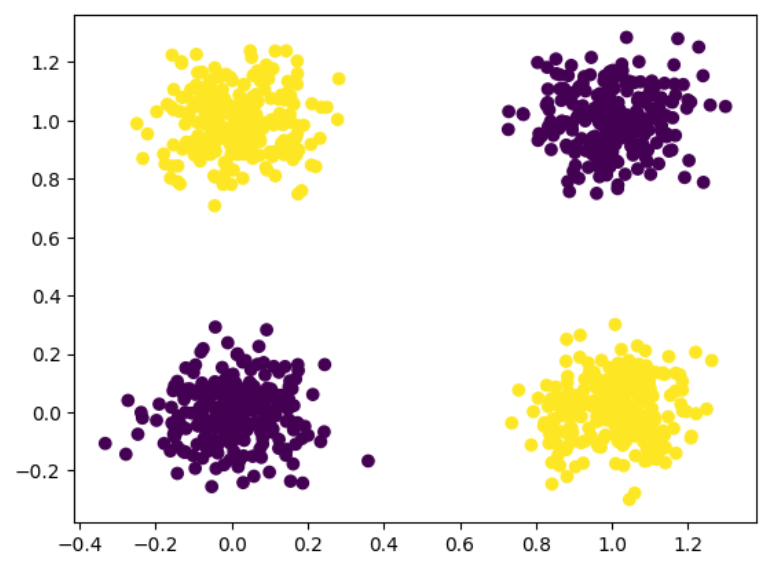


Рис. 3. Визуализация сгенерированной выборки для части 2

### Обучение нейронной сети задачи классификации

Далее определяем функцию нелинейности, функцию потерь и параметры НС (шаг обучения, число итераций). После этого проводим обучение.

Выводим график функции потерь, являющийся логарифмической функцией (рис. 4):

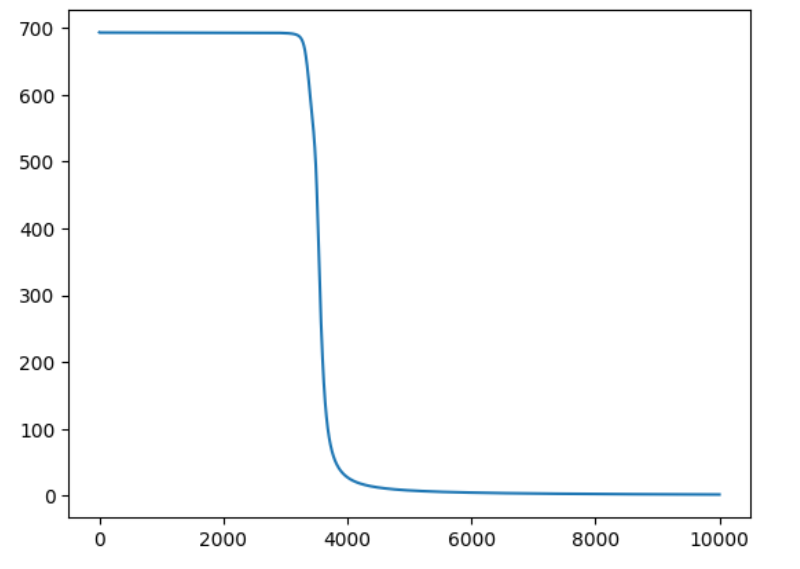
plt.plot(losses)  


Рис. 4. График функции потерь

По графику на рис. 4 видно, что после 4к итераций значения функции начали уменьшаться, что означает нахождение верного направления градиента.

### Проверка результатов обучения

Далее скомпонуем результаты обучения модели и выведем рисунок с классификацией классов нейросетью (рис. 5).

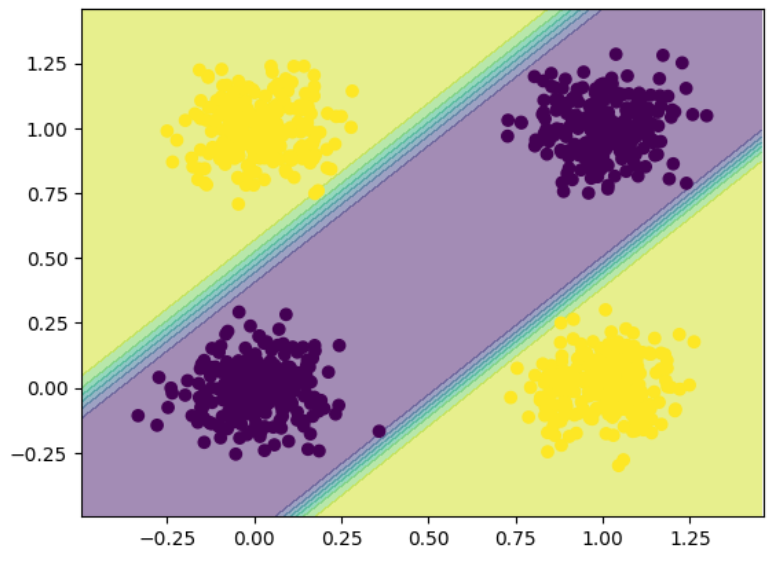


Рис. 5. Проверка классификации классов

## Часть 3. Классификация изображений CIFAR100

Загружаем и распаковываем набор данных CIFAR100

Далее читаем датасет и выберем из него данные, относящиеся к классам по варианту 9 (группа иу5-25м). Это классы 40, 65, 30 (dolphin, lamp, rabbit).

Далее создаем Pytorch DataLoader.

Далее создаем Pytorch модель многослойного перцептрона с одним скрытым слоем

self.seq = nn.Sequential(

nn.Linear(32\*32\*3, hidden\_size),

nn.ReLU(),

nn.Linear(hidden\_size, classes),

)

Итого получаем следующую модель:

Cifar100\_MLP(

(norm): Normalize()

(seq): Sequential(

(0): Linear(in\_features=3072, out\_features=10, bias=True)

(1): ReLU()

(2): Linear(in\_features=10, out\_features=3, bias=True)

)

)

Далее выбираем функцию потерь кросс-энтропию и оптимизатор градиентного спуска.

Далее обучаем модель по эпохам. Результаты отображены на рис. 6.

Далее проверяем качества модели по классам на обучающей и тестовой выборках. Результаты отображены на рис. 7.

## Задачи для самостоятельной работы

1. Проанализируйте результаты обучения модели.

Что говорит о ней точность на обучающей и тестовой выборке?

Значения точностей модели на обучающей (оранжевый график) и тестовой выборке (синий график):

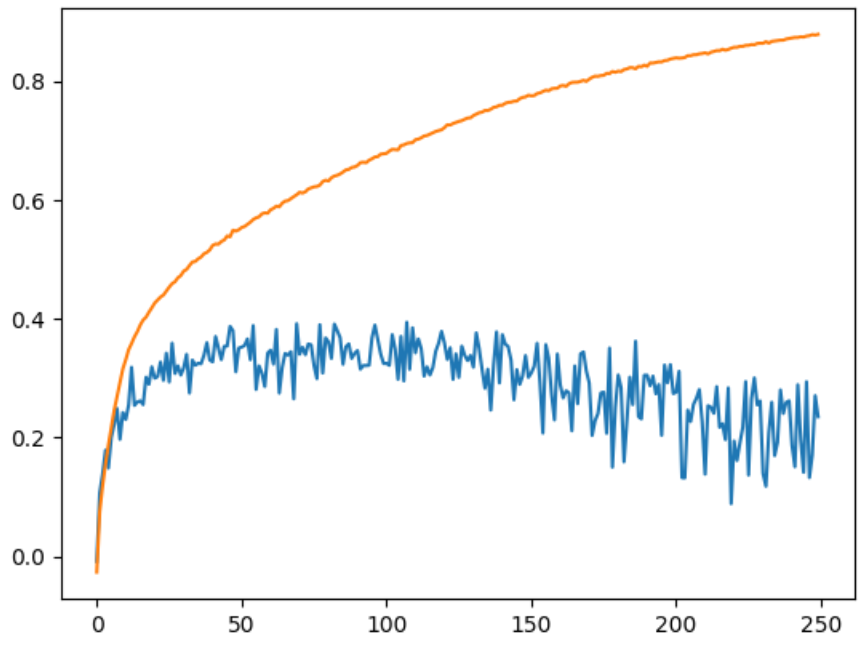


Рис. 6. Точность исходной модели на обучающей и тестовой выборке

По рис. 6 видно, что точность на обучающей выборке постоянно растет до абсолютного значения, а на тестовой сначала растет, а затем постепенно падает. Момент, когда точность на тестовой выборке начинает падать является переломным, поскольку после него возникает переобучение.

С какими классами модель справляется лучше и почему?

На основе результатов на рис. 7 можно сказать, что модель лучше всего справляется с классом 40 (lamp), далее по точности идет класс 65 (rabbit) и хуже всего модель справляется с классом 30 (dolphin).

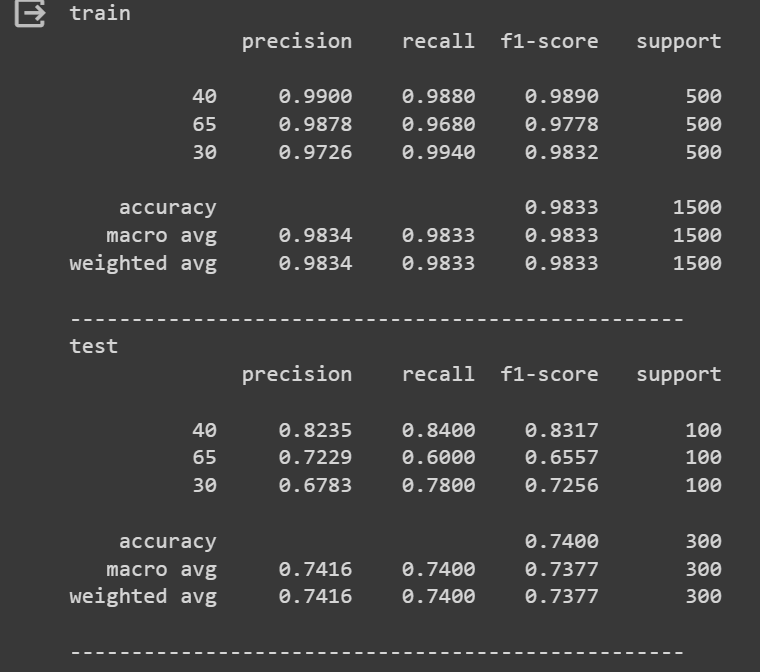


Рис. 7. Результаты обучения исходной модели (для 250 эпох)

Просмотрим несколько картинок (рис. 8):

fig = plt.figure(figsize=(10, 10))

columns = 8

rows = 8

for i in range(1, columns\*rows +1):

    img = Image.fromarray(train\_X[i]).resize((256,256))

    fig.add\_subplot(rows, columns, i)

    plt.imshow(img)

plt.show()

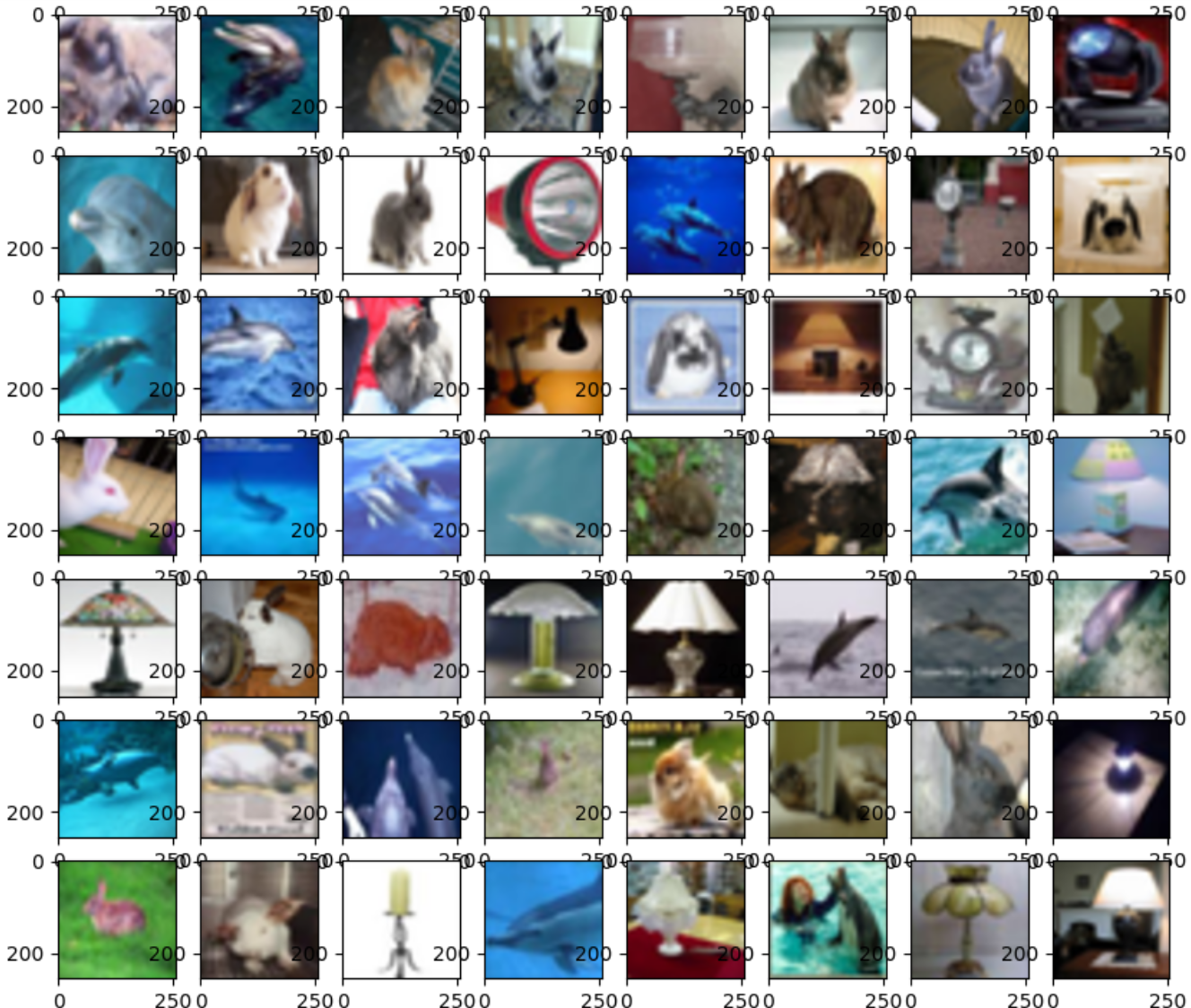


Рис. 8. Примеры картинок выборки

По картинкам видно, что лампы часто имеют отчетливую похожую форму (равнобедренная трапеция + ножка лампы), что, скорее всего, позволяет лучше отличать её от других картинок. Самая низкая precision у картинок с дельфином (то есть, когда дельфина на картинках нет, нейросеть все равно считает, что это дельфин). Скорее всего это из-за того, что дельфин часто изображен мелким среди пустого свободного пространства, при этом это может быть свойственно и картинкам кролика. У картинок кролика хуже всего значение recall, то есть кролики часто не обнаруживаются. Это может быть из-за того, что кролики могут быть разных цветов, изображены в разном масштабе на различных местностях, что может быть причиной спутывания их с лампами и дельфинами.

1. Проанализируйте результаты обучения. Возникает ли переобучение вашей модели? Что необходимо сделать, чтобы нивелировать это (не используя регуляризацию)?

Да, переобучение модели возникает, что видно на рис. 6. Минимальное значение ошибки на тестовой выборке на 108 эпохе (на рис.9 это значение 0,606).

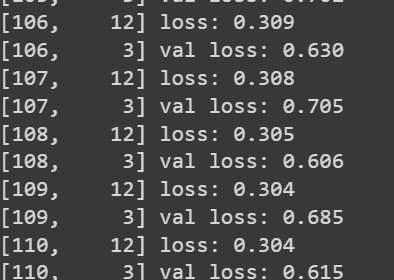


Рис. 9. Значения потерь по эпохам

Чтобы избежать переобучения обучим модель до 106 эпох. В этом случае общая точность модели поднялась с 0,74 до 0,78 (рис. 10).

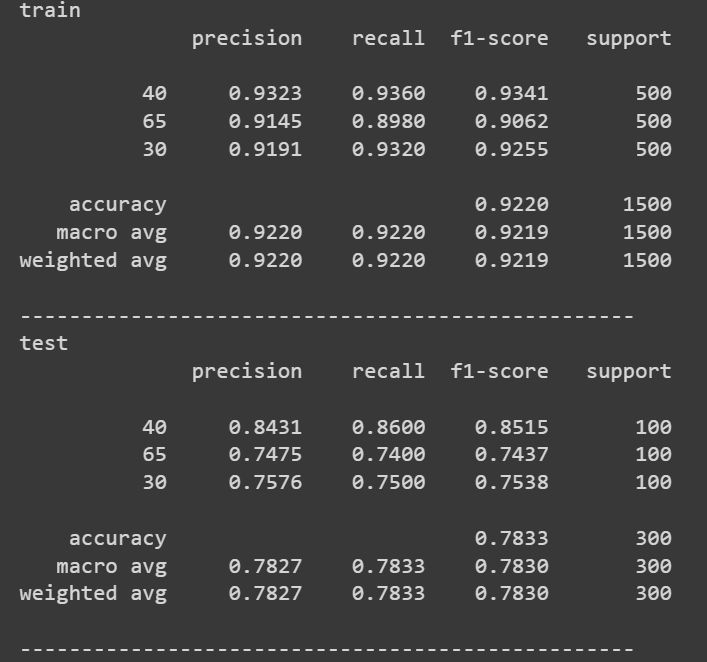
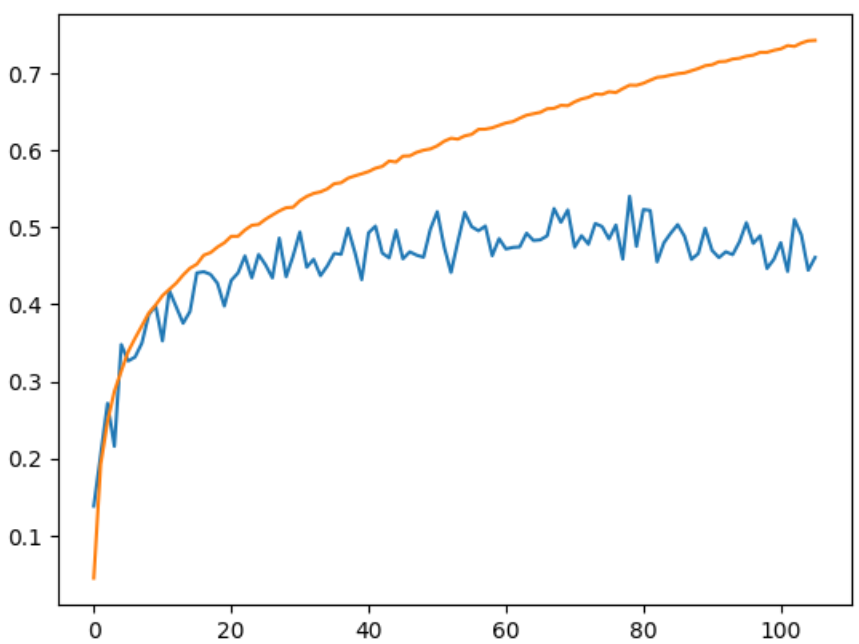
 

Рис. 10. Точность модели для 106 эпох

1. Измените размер батча, но сохраните общее количество итераций. Проанализируйте результаты обучения с новыми гиперпараметрами. Что изменилось и почему?

Посмотрим точность модели при маленьком (10) и большом (500) размерах батча. Изменим в коде переменную batch\_size (исходное значение 128). Результаты обучения представлены на рис. 11-13.

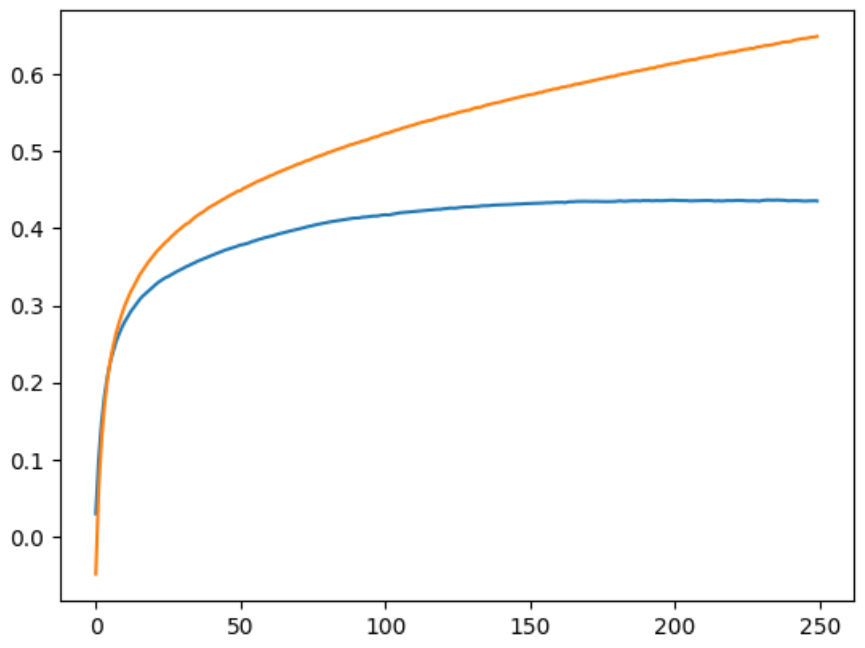
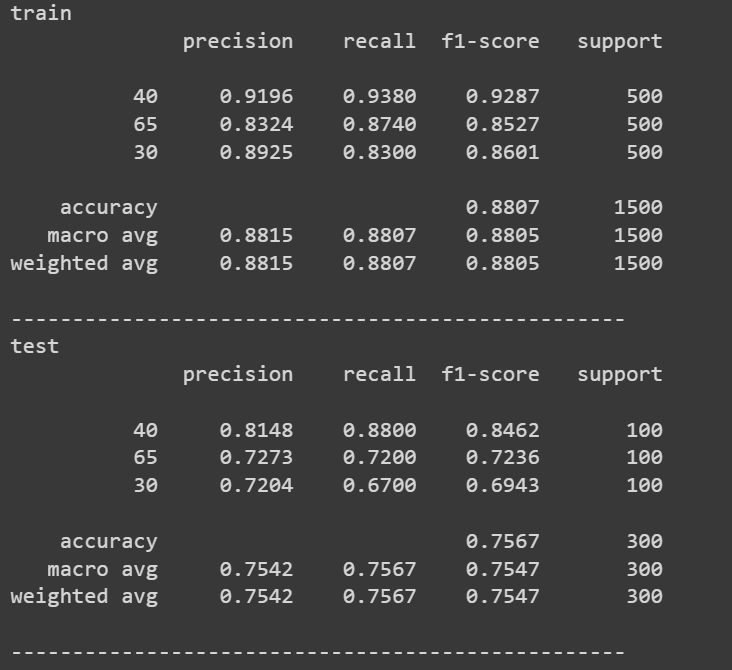


Рис. 11. Точность модели при размере батча 500

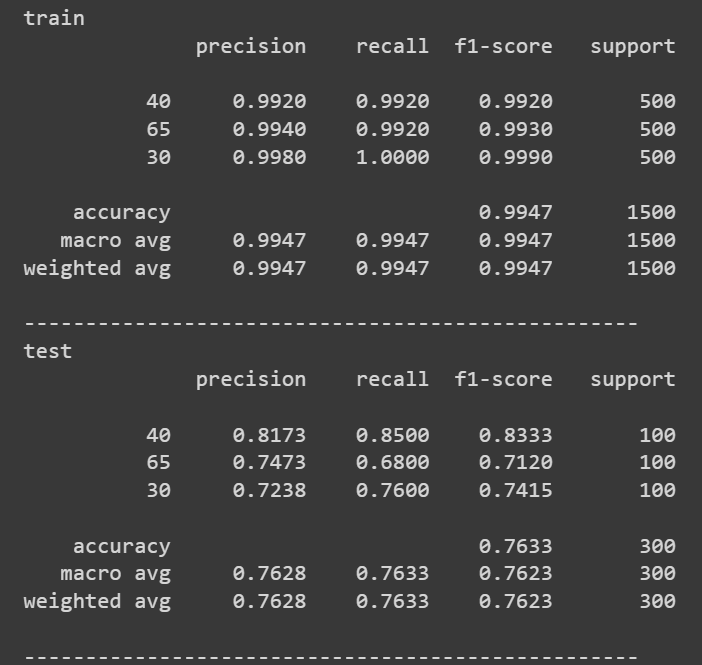
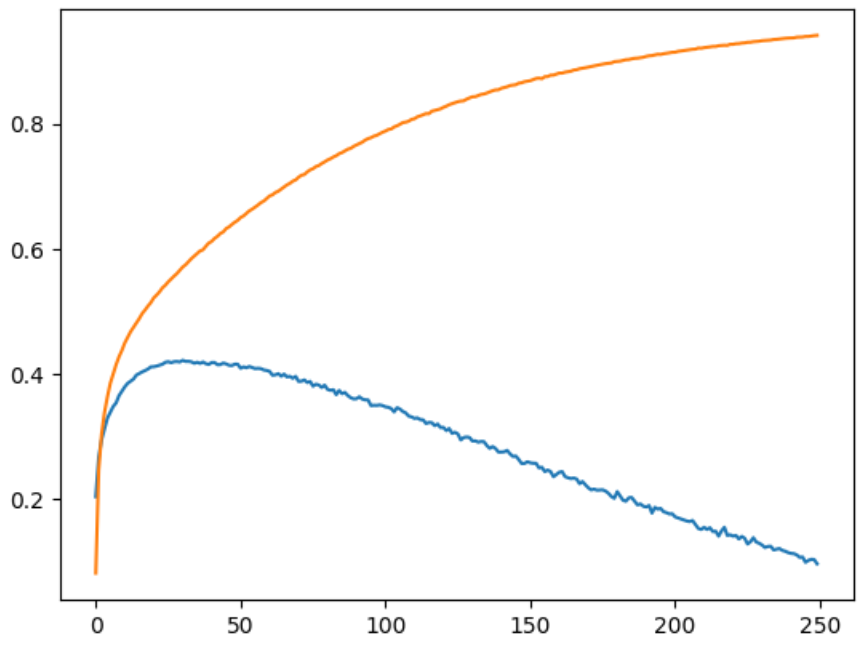
 

Рис. 12. Точность модели при размере батча 100

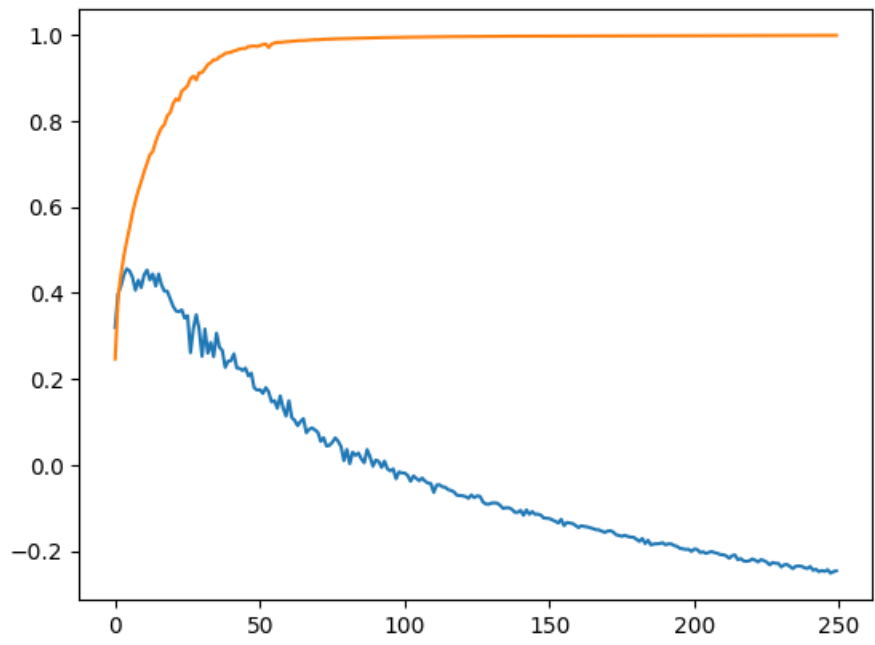
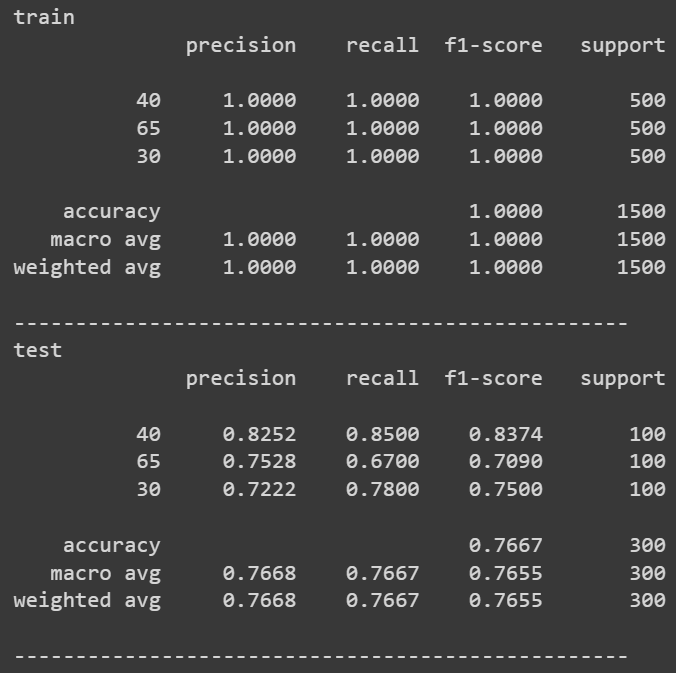


Рис. 13. Точность модели при размере батча 10

Таким образом, можно сказать, что при маленьком размере батча быстрее возникает переобучение и само обучение модели идет долго. При большом размере батча обучение идет быстро, но точность все же ниже. В итоге остановимся на среднем размере батча = 100. Для него уменьшим число эпох до 50 и получим следующие результаты (рис. 14):

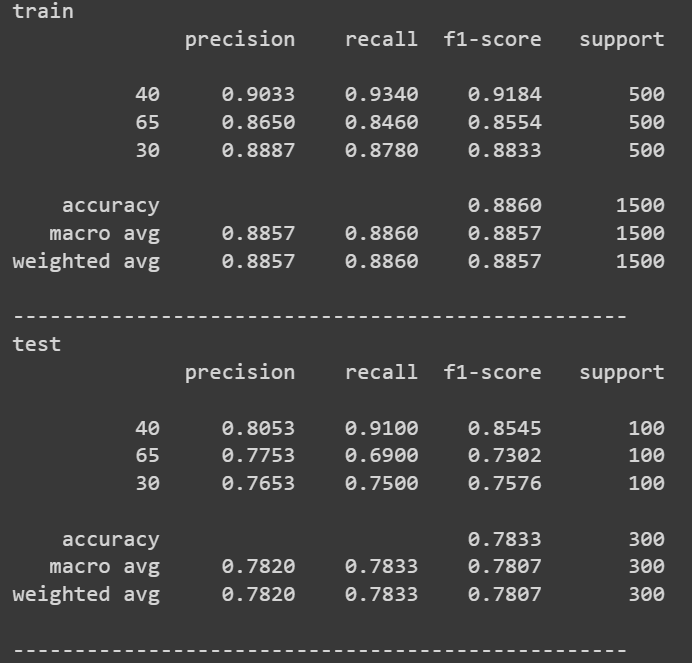
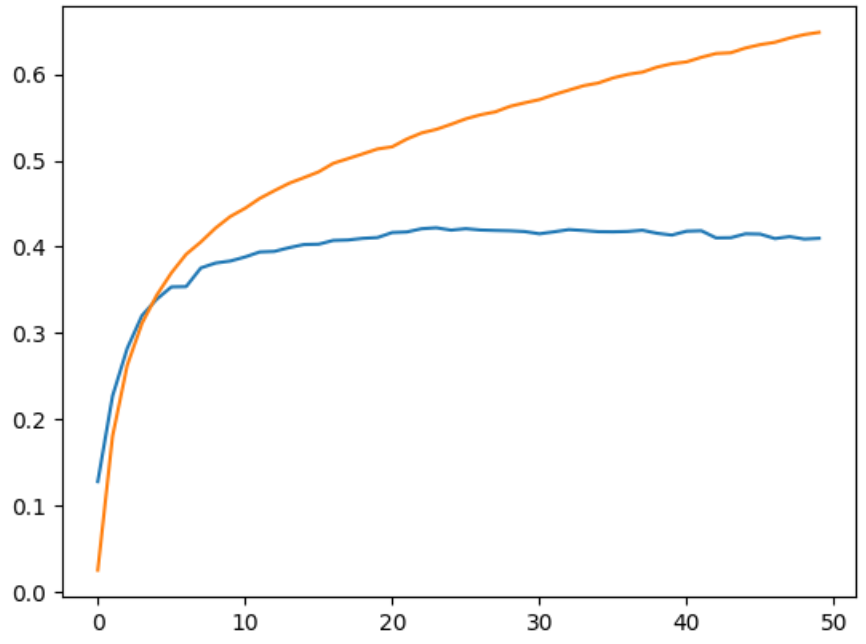
 

Рис. 14. Точность модели при размере батча 100 и числе эпох 50

1. Уменьшите скорость обучения и увеличьте общее количество итераций, чтобы повысить точность модели.

Итерации — число батчей, необходимых для завершения одной эпохи. То есть, при размере обучающей выборки = 1500 и размере батча = 128, кол-во итераций равно 1500/128 = 11,72 (округляем в большую сторону) => 12. Чтобы увеличить общее количество итераций нужно уменьшить размер батча, при этом уменьшится скорость обучения. Это мы делали в предыдущем пункте 3 в случае с размером батча = 10. Найдем для этого случая момент переобучения и уменьшим число эпох, тогда получим следующую точность модели (рис. 15):

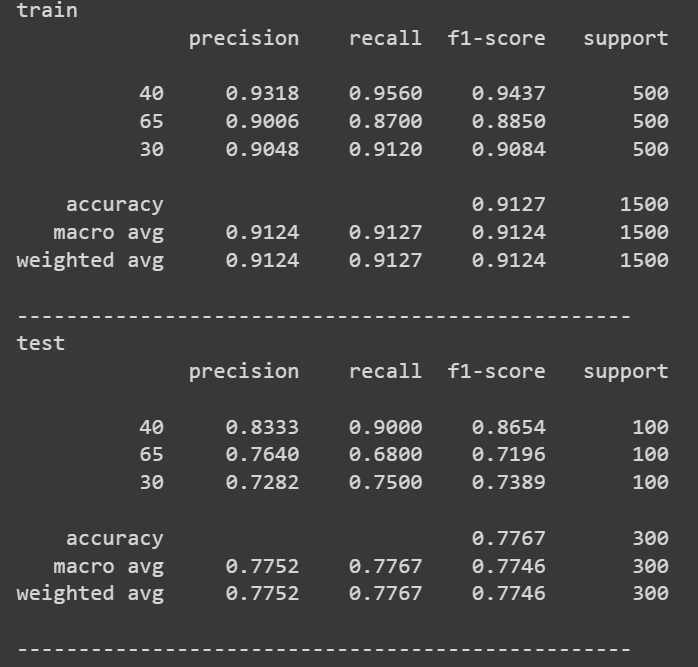
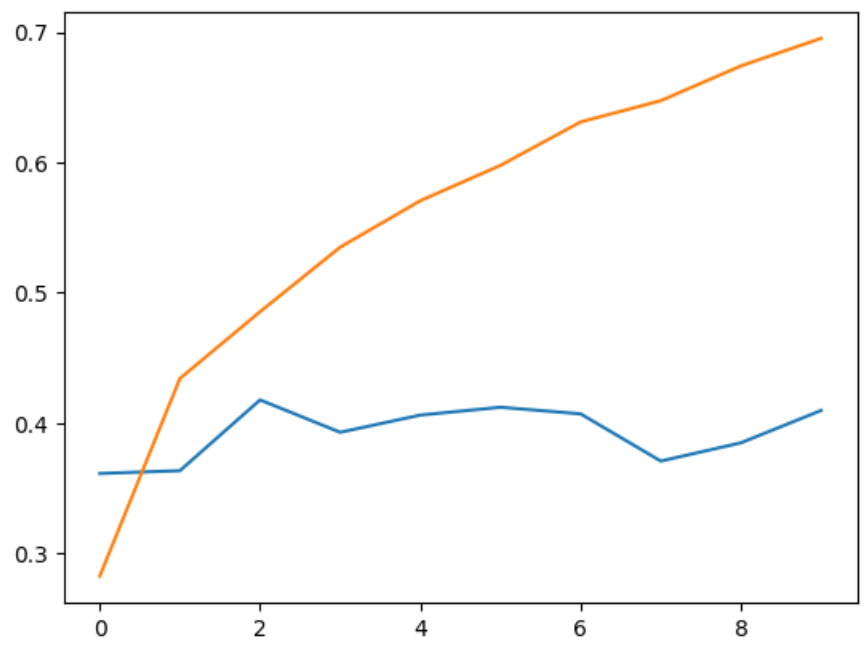
 

Рис. 15. Точность модели при размере батча 10 и числе эпох = 10.

Результаты получились хуже, чем для размера батча = 100, поэтому далее будем использовать параметры, установленные в предыдущем пункте.

1. Измените вашу модель - поменяйте количество нейронов и слоев. Проанализируйте результаты обучения новой модели. Найдите лучшие гиперпараметры для этой модели.

Изменим число нейронов в скрытых слоях HIDDEN\_SIZE до 30 и получим результаты, представленные на рис. 16. При дальнейшем увеличении числа скрытых слоев, точность выше не становится, поэтому остановимся на 30.

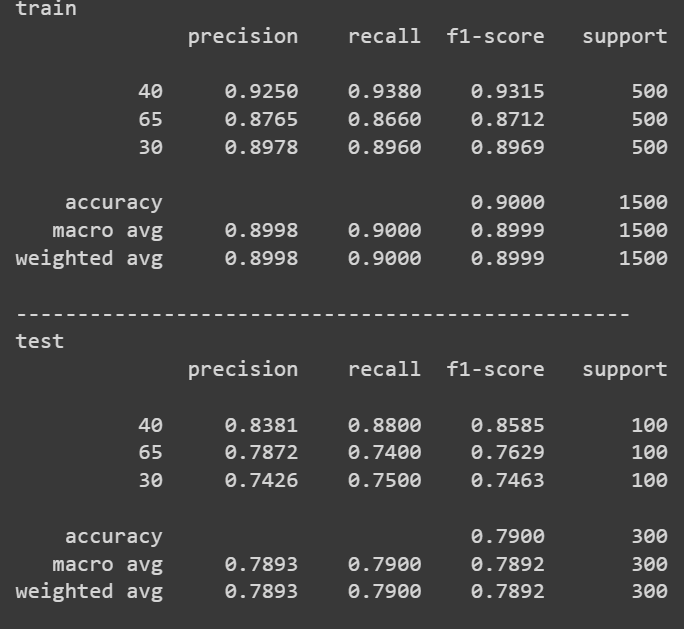
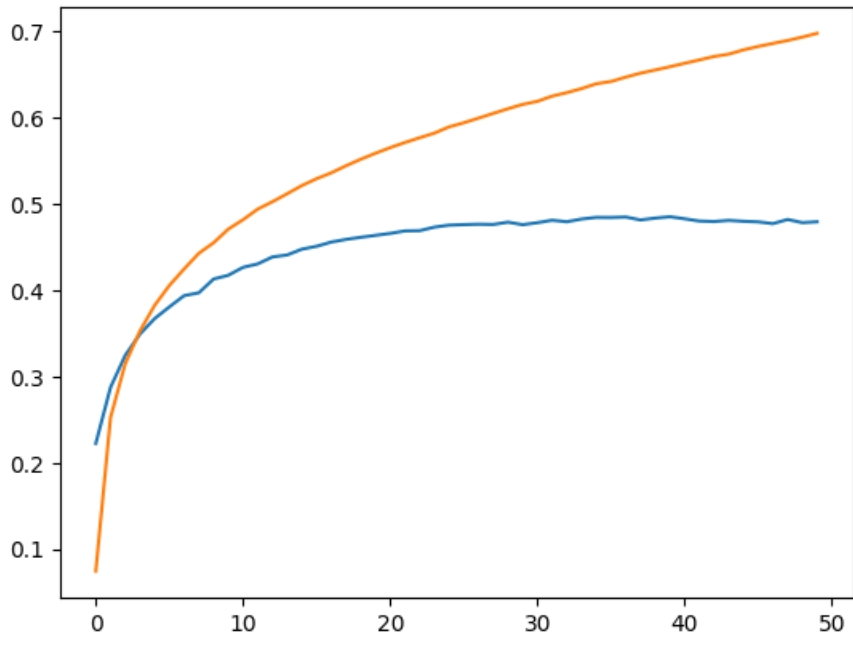
 

Рис. 16. Точность модели при размере батча 100, числе эпох 50, hidden\_size=30

Далее изменим параметр оптимизатора градиентного спуска:

optimizer = optim.SGD(model.parameters(), lr=0.006)

Получим следующие результаты (рис. 17):

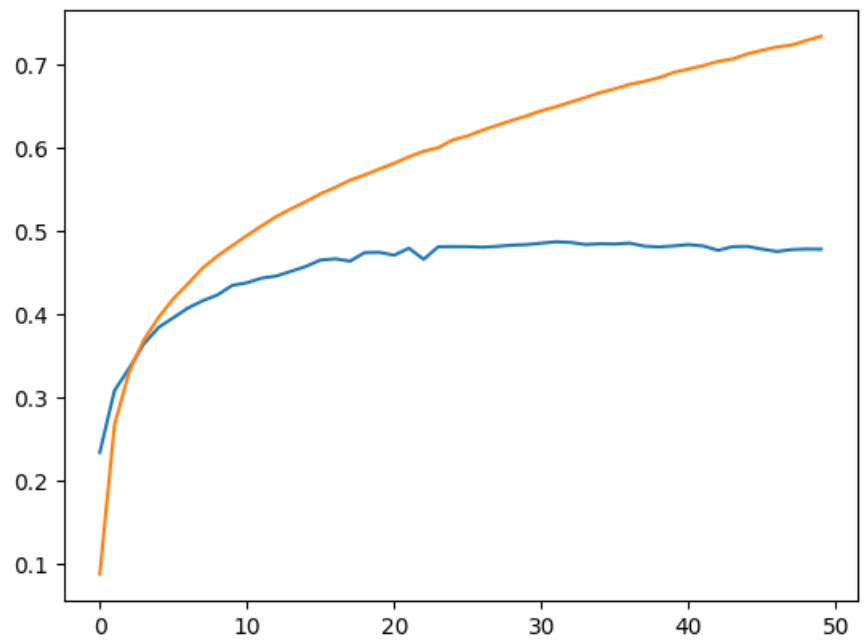
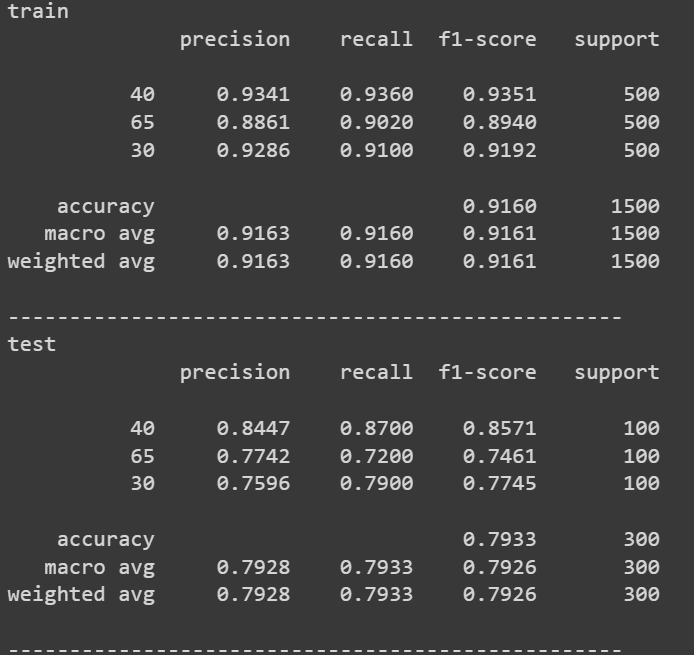


Рис. 17. Точность модели при размере батча 100, числе эпох 50, hidden\_size=30, lr=0,006

При дальнейших изменениях lr точность не повышается, поэтому оставим lr=0,006.

Далее изменим число скрытых слоев до 2х и число нейронов в них до 12. Получим результаты, представленные на рис. 18.

  self.seq = nn.Sequential(

            nn.Linear(32\*32\*3, hidden\_size),

            nn.ReLU(),

            nn.Linear(hidden\_size, hidden\_size),

            nn.ReLU(),

            nn.Linear(hidden\_size, classes),

        )

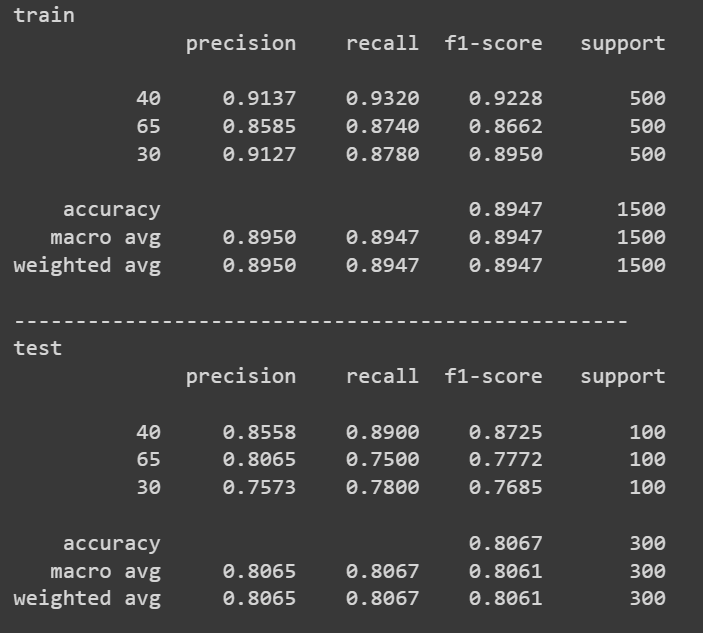
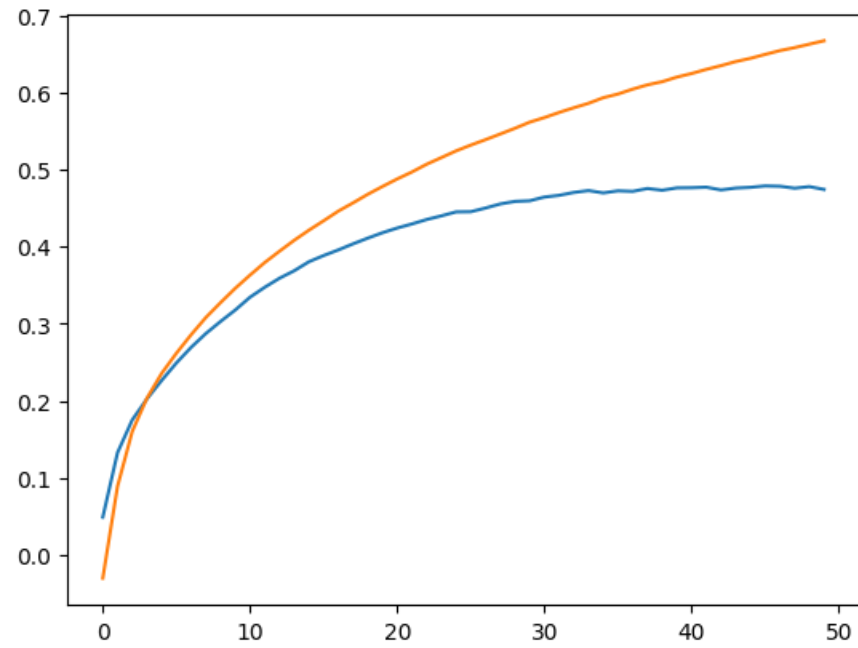
 

Рис. 18. Точность модели при размере батча 100, числе эпох 50, hidden\_size=12, lr=0,006, число скрытых слоев = 2

Таким образом, наивысшая точность, которую удалось получить, равна 80,67%.

1. Укажите, какие действия помогли повысить точность вашей модели и объясните почему?

Для повышения точности модели изменили размер батча, кол-во эпох, число скрытых слоев, число нейронов в скрытых слоях и параметр градиентного спуска. Все пояснения описали в пунктах выше.

# Итоговая таблица

Итоговая таблица 1 с результатами для всех вариантов обучения:

Таблица 1. Результаты обучения модели

| **Конфигурация нейросети** | **Гиперпараметры** | **Точность** | **Комментарий** |
| --- | --- | --- | --- |
| FC(10), FC(3) | batch\_size=128, epochs=250, lr=0,005 | train=98,33%, test=74% | Исходная модель |
| FC(10), FC(3) | batch\_size=128, epochs=**106**, lr=0,005 | train=92,2%, test=78,33% | Изменили число эпох |
| FC(10), FC(3) | batch\_size=**10**, epochs=250, lr=0,005 | train=100%, test=76,67% | Уменьшили размер батча |
| FC(10), FC(3) | batch\_size=**500**, epochs=250, lr=0,005 | train=88,05%, test=75,67% | Увеличили размер батча |
| FC(10), FC(3) | batch\_size=**100**, epochs=250, lr=0.005 | train=99,47%, test=76,33% | Нашли оптимальный размер батча |
| FC(10), FC(3) | batch\_size=**100**, epochs=**50**, lr=0,005 | train=88,57%, test=78,33% | Уменьшили число эпох, оставив размер батча |
| FC(**30**), FC(3) | batch\_size=**100**, epochs=**50**, lr=0,005 | train=90%, test=79% | Изменили число нейронов в скрытом слоев |
| FC(**30**), FC(3) | batch\_size=**100**, epochs=**50**, lr=**0,006** | train=91,6%, test=79,33% | Изменили параметр градиентного спуска |
| **FC(12), FC(12),** FC(3) | batch\_size=**100**, epochs=**50**, lr=**0,006** | train=89,47%, test=80,67% | Изменили число скрытых слоев и число нейронов в них |

# Контрольные вопросы для защиты

1. Полносвязная нейронная сеть, объясните структуру, вычисления и назначение слоев и составляющих нейронов.

Полносвязная НС состоит из нескольких слоев (входной, выходной и скрытые). На входной слой поступают данные исходной выборки, далее происходит обучение на скрытых слоях и в итоге получаем результаты обучения на выходном слое. Каждый слой состоит из нейронов. Для полносвязной НС **каждый** нейрон слоя связан со **всеми** нейронами предыдущего слоя (каждый нейрон является сумматоров весов нейронов предыдущего слоя). В этом и заключается процесс обучения (в вычислении весов). Также после каждого скрытого слоя идет функция активации.

Полносвязная НС хорошо классифицирует данные при их малом объеме, а при большем объем в ней возникает очень много связей.

1. Укажите количество нейронов, связей и весов в полносвязной нейронной сети.

Количество нейронов входного слоя = кол-ву параметров у данных. (3072)

Количество нейронов выходного слоя = кол-ву прогнозируемых классов. (3)

Для скрытых слоев можем устанавливать разное число нейронов. (12)

Количество связей для каждого слоя = число нейронов текущего слоя \* число нейронов предыдущего слоя.

Веса вычисляются для каждого нейрона и далее проходят через функцию активации.

Сколько параметров в полносвязной нейросети?

(norm): Normalize()

(seq): Sequential(

(0): Linear(in\_features=3072, out\_features=12, bias=True)

(1): ReLU()

(2): Linear(in\_features=12, out\_features=12, bias=True)

(3): ReLU()

(4): Linear(in\_features=12, out\_features=3, bias=True)

)

)

**Параметры нейронной сети** — это набор весов (*weights*) у каждого нейрона и их *biases* (у каждого нейрона — один bias).

Входной слой. Число входов: 32\*32\*3=3072 (размер изображения \* 3 класса). Число выходов=12.

Промежуточный слой: число входов: 12, число выходов 12. (таких 2 слоя)

Выходной слой: число входов 12, число выходов 3 (класса).

Всего: 3072 + 12 + 12 + 3 = 3099 нейронов

Связи: 12\*3072+12\*12+3\*12=37044 (веса)

**Число параметров:** 12\*3072 + 12 (смещения) + 12\*12 + 12 (смещения) + 3\*12 + 3 (смещения) = 36864 + 12 + 144 + 12 + 36 + 3 = 37071.

Проверка числа параметров в коде:

sum([p.numel() for p in model.parameters() if p.requires\_grad])

1. Опишите задачи регрессии и классификации. Какие функции потерь применяются в этих задачах?

**Задача классификации** – отнести объекты к заранее известным классам. Функция потерь: **логарифмическая функция** (измеряет разницу между предсказанным распределением вероятностей и истинными метками).

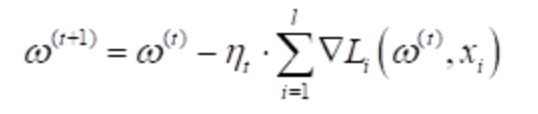
**Задача регрессии** – предсказать значения некоторой непрерывной величины. Функция потерь: **среднеквадратичная ошибка (MSE)** (измеряет среднеквадратичную разницу между прогнозируемым выходом и истинным выходом).

1. Опишите структуру набора данных, назначение его частей

В качестве набора данных был взят набор cifar-100, предназначенный для классификации изображений. Из cifar-100 были выбраны 3 класса (с дельфинами, лампами, кроликами). Пиксели картинок были преобразованы в массивы параметров для обучения. Полученный набор был разделен на обучающую (1500 объектов) и тестовую (300 объектов) выборки.

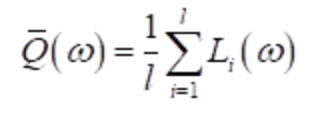
1. Опишите алгоритм стохастического градиентного спуска. Укажите назначение гиперпараметров. В чем отличие пакетного и стохастического спуска? (И мб отличие от градиентного)

Для стохастического градиентного спуска (СГС) вычисляется сумма градиентов функции потерь по обучающей выборке:



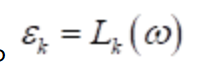
Алгоритм СГС. Вход: выборка X, шаг сходимости η, скорость «забывания» λ. Выход: вектор весовых коэффициентов ω. Шаги алгоритма:

1. инициализация весов
2. начальное вычисление функционала качества:

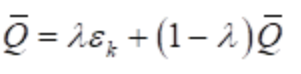


1. цикл

- случайный выбор наблюдения из X

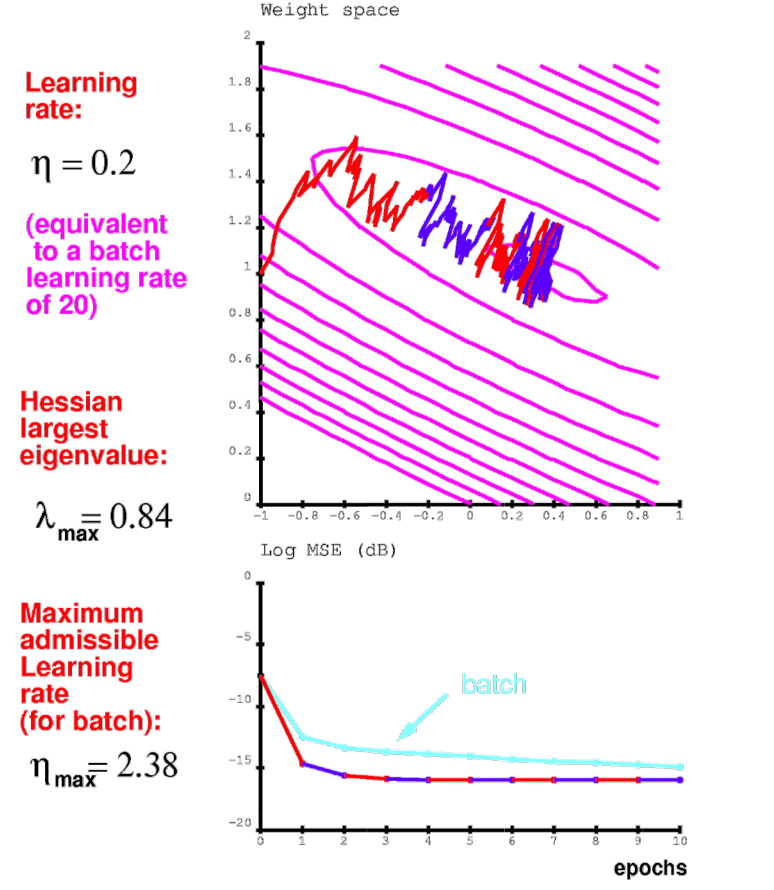
- вычисление функции потерь 

- шаг псевдоградиентного алгоритма: 

- пересчет функционала качества: 

- повторять цикл пока  и/или ω не достигнут заданных значений

Пример из лекции для СГС: набор данных: набор-1 (100 примеров, 2 гауссова распределения) сеть: 1 линейный нейрон, 2 входа, 1 выход. 2 веса, 1 отступ.



**Отличия между пакетным и стохастическим градиентным спуском:**

Стохастическое обновление обычно НАМНОГО быстрее, чем пакетное обновление. Особенно на больших избыточных наборах данных.

Вот почему:

– Представьте, что у вас есть обучающий набор из 1000 примеров.

– Этот обучающий набор состоит из 10 экземпляров по 100 примеров.

– Пакет (батч): вычисление для одного обновления будет в 10 раз больше необходимого

– Стохастическое: будет использоваться избыточность в учебном наборе для получения преимущества обучения. Одна эпоха на большом наборе будет похожа на 10 эпох на меньшем наборе.

– Пакетное обновление будет КАК МИНИМУМ в 10 раз медленнее стохастичного

– В реальной жизни повторения редко происходят, но очень часто обучающие примеры очень избыточны (много примеров похожи друг на друга), что имеет тот же эффект.

– На практике нередки разницы скорости (на порядки) между пакетным и стохастическим обновлением.

– Маленькие пакеты могут использоваться без штрафа, если примеры в минибатче не слишком похожи.

**Стохастическое.**

**Преимущества:**

– Быстрая сходимость на больших избыточных данных

– Стохастическая траектория позволяет избежать локальных минимумов

–**Недостатки:**

– Продолжает «прыгать», если скорость обучения не уменьшается

– Теоретические условия сходимости не так понятны, как для пакетного обновления

– Доказательства сходимости вероятностны

– Большинство хороших способов ускорения или методов второго порядка не работают со стохастическим градиентом

– Сложнее распараллелить, чем пакетное обновление

**Пакетное**

**Преимущества:**

– Гарантированное сходимость к локальному минимуму в простых условиях

– Много способов и методов второго порядка для ускорения

– Простые доказательства сходимости

–**Недостатки:**

– Болезненно медленный на больших задчах

– Несмотря на длинный список недостатков для стохастичного обновления, это то, что большинство людей используют (что справедливо, по крайней мере, для больших задач)

Стохастический градиентный спуск — это радикальное упрощение градиентного спуска, которое преодолевает некоторые из его трудностей. Каждая итерация SGD вычисляет градиент на основе одного случайно выбранного раздела набора данных, который был перетасован, вместо использования всей части наблюдений.

В градиентном спуске мы рассматриваем все точки при расчете потерь и производной, в то время как в стохастическом градиентном спуске мы используем одну точку в функции потерь и ее производной случайным образом.

**Отличия:**

Градиентный – смотрим весь набор при вычислении градиента. Стохастический градиентный – берем случайные разделы. Пакетный градиентный – делим весь набор на батчи и рассматриваем их. С моментом – учитываем предыдущие параметры модели на каждой новой итерации.

1. Что такое эпоха, итерация, батч обучения. Как они взаимосвязаны?

**Эпоха** – весь датасет прошел через нейронную сеть один раз.

**Батч** – часть датасета (т.е. датасет делится на батчи для улучшения производительности).

**Итерация** – число батчей, необходимых для завершения одной эпохи.

1. Что такое обучение с учителем, без учителя, с подкреплением? Приведите примеры методов и задач для каждого вида обучения.

**Обучение с учителем** – используется специальный размеченный набор данных, где указано, за что эти данные отвечают (в нашем случае это обучающая выборка).

**Обучение без учителя** – у нейросети нет размеченного набора данных, и она старается сама найти общие признаки и зависимости между данными.

**Обучение с подкреплением** – после каждого обучения нейросети результаты ее обучения оцениваются и далее НС старается их улучшить.