Лабораторная работа №2
по дисциплине «Системы поддержки принятия решений»
«Исследование методов классификации данных с помощью нейронной сети»

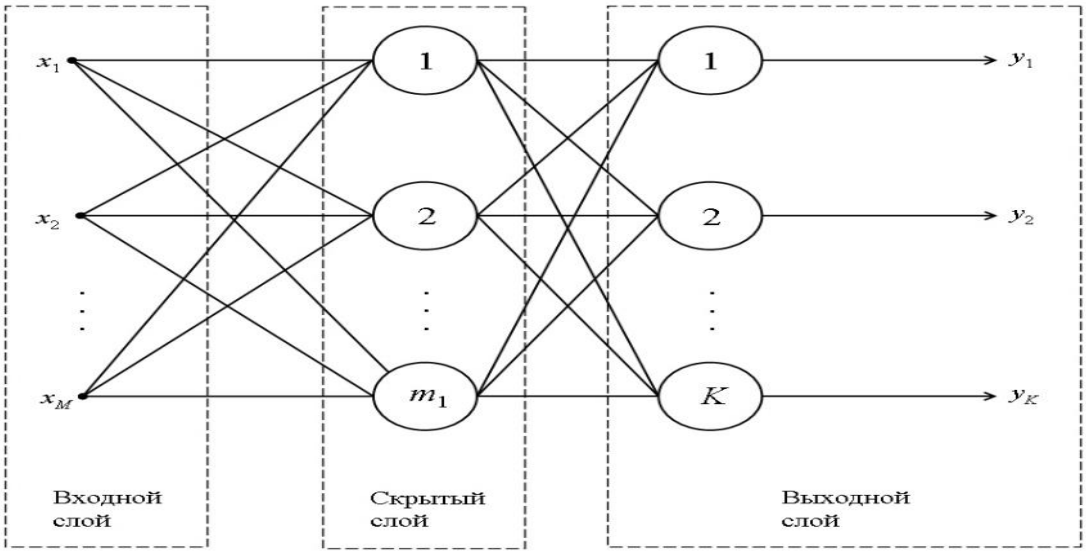
Цель работы – исследование принципов разработки нейронной сети на примере задачи классификации данных в PyTorch.

Классификация — понятие в науке, обозначающее разновидность деления объёма понятия по определённому основанию (признаку, критерию), при котором объём родового понятия (класс, множество) делится на виды (подклассы, подмножества), а виды, в свою очередь делятся на подвиды и т.д.

Общие положения

Текст программы представлен ниже.

Общая структура полносвязной нейронной сети представлена на рисунке.



Для примера используем датасет классификации вин. Известный датасет имеется в библиотеке scikit.learn. Мы можем его загрузить через sklearn.datasets.load_wine(). Этот датасет будет содержать 178 различных бутылок вин, у каждой бутылки измерено 13 параметров, это вещественные числа. Конкретную бутылку можно будет классифицировать на три класса.

Если взять один и тот же код и запустить его несколько раз, то получим разные результаты. Это происходит потому, что каждый раз веса нейронной сети инициализируются заново, происходит по-разному обучение нейронной сети, и это всё зависит от того, как отработал генератор случайных чисел в нашем процессоре.

Возможно, мы хотим, чтобы эксперименты были воспроизводимы: чтобы, взяв один и тот же питоновский файл и выполнив его, мы получили бы тот же самый результат, как и раньше. Например, это нужно для того чтобы понимать: а правда ли те изменения, которые мы делаем с нейросетью, улучшают наши результаты, или это результат некоторой случайности. Зафиксируем эту случайность. Есть такое понятие как **random seed**, это можно интерпретировать, как номер последовательности случайных чисел, которую выдаст нам случайный генератор, если его попросить выдать нам последовательность. Случайных

генераторов есть несколько: есть случайный генератор модуля random в Питоне, есть случайный генератор модуля numpy.random (библиотеки numpy), есть также случайные генераторы библиотеки PyTorch и другие.

Зафиксируем их все. Для этого возьмём random seed и поставим его в конкретное значение. Таким образом, мы всегда будем использовать нулевую последовательность при вызове случайного генератора библиотеки random (это библиотека языка python).

Также нам нужно зафиксировать сиды в numpy и в PyTorch, случайные сиды, которые отвечают за обсчёт и CPU и на GPU – они разные, соответственно нужно ещё зафиксировать случайный seed подмодуля CUDA.

Скорее всего, если вы рассчитываете на видеокарте, вы используете библиотеку cudnn, и она может выполняться в детерминистичном режиме, а может в недетерминистичном. Недетерминистичный режим гораздо быстрее, но если мы действительно, в угоду скорости, хотим детерминистичность, хотим, чтобы была воспроизводимость, то нам нужно выставить этот параметр в "True".

Выставив все эти параметры, мы можем практически гарантировать то, что у нас эксперименты будут воспроизводимы.

Теперь нужно этот датасет разбить на две части: на трейновую часть, на которой мы будем обучаться, и на тестовую, на которой мы будем считать метрики. Воспользуемся функцией train_test_split из той же библиотеки scikit_learn. Если мы передадим в функцию train_test_split первым параметром, собственно dataset (тут мы используем только первые две колонки, колонок 13, столько же, сколько у нас параметров вина, мы используем всего две для удобства последующей визуализации), вторым параметром мы передадим таргеты, то есть те классы, которые нам нужно предсказать – это будет номер класса (такая колонка).

Отведем 30 процентов на тест, и перед тем, как этот датасет делить на две части, нужно его перемешать, чтобы *удостовериться, что если он был отсортирован, например, по номеру класса, то теперь эта сортировка не работает*.

После этого мы все "фолды": X_train, X_test, Y_train и Y_test обернём в torch тензоры: дробные числа, мы обернём в float тензор, если числа не дробные, обернём в long тензор.

Далее реализуем класс, назовём его WineNet, это будет нейросеть для классификации. Точно так же, как нейросеть для синуса, которая была работе №1, отнаследуем класс от torch.nn.Module, в функции __init__ (в конструкторе этого класса) будет аргумент – количество скрытых нейронов n_hidden_neurons.

Реализуем здесь два скрытых слоя, нейросеть будет состоять всего из трёх слоёв, и два из них будут скрыты. Первый слой – это fully connected (полносвязный) слой, из двух входов (у нас две колонки для каждой бутылки вина), на выходе N скрытых нейронов, дальше активация: сигмоида, можно поставить любую другую, если хотите. После этого – скрытый слой, который из N нейронов, превращает их тоже в N нейронов. Снова сигмоидная активация. После этого снова fully connected слой, который выдаёт нам три нейрона, каждый нейрон будет отвечать за свой класс.

Таким образом, на выходе этих трёх нейронов будут некоторые числа, которые после этого мы передадим в **софтмакс**, и получим вероятности классов. Напишем функцию "forward" – она будет реализовывать граф нашей нейронной сети. Передаём двухмерный тензор с двумя колонками в первый fully connected слой, после этого в первую активацию, во второй fully connected слой, во вторую активацию, в третий fully connected слой, у которого три выхода.

Заметьте, здесь не используется softmax. Почему мы не прогнали выход из третьего слоя через softmax? Дело в том, что после того, как мы посчитаем выходы нейронной сети, мы хотим прогнать их через softmax и посчитать **кросс-энтропию**. Но в формуле кросс-энтропии есть логарифм, то есть выходы нейронной сети прогоняются через логарифм. А в формуле softmax участвуют экспоненты. Эти экспоненты и логарифмы взаимно уничтожаются, и получается, что нам не нужно вычислять экспоненты, чтобы посчитать кросс-энтропию. Мы можем её посчитать, не считая softmax. Соответственно, если мы хотим просто считать ошибку (лосс), нам softmax не нужен. Если мы хотим посчитать вероятности, то нам придётся использовать softmax.

Softmax – функция довольно долгого вычисления, поэтому мы стараемся её избегать. Для того, чтобы считать вероятности, напишем функцию inference, которая будет вызывать функцию "forward", и прогонять её через softmax.

Инициализируем нашу нейронную сеть с количеством скрытых нейронов, равным пяти. Нейросеть имеет имя wine_net. Осталось только инициировать функцию потерь – бинарную кросс-энтропию (torch.nn.CrossEntropyLoss), которая использует не выходы после софтмакса, а выходы нейронной сети, не пропущенные ещё через софтмакс.

Далее задается оптимайзер – тот метод, который будет использоваться для вычисления градиентных шагов. В оптимайзер мы передаём все параметры нейронной сети – ее веса. Это те скрытые значения, которые находятся в нейронах, которые мы хотим подбирать. Learning rate выбираем 0.001 (как правило, стандартное значение по умолчанию).

Усложним обучение нейронной сети. Ранее мы брали весь датасет, считали по нему loss-функцию, дальше делали градиентный шаг, и повторяли этот процесс многократно. Но в реальной жизни вряд ли поместится в памяти весь датасет. Обучение в реальной сети *происходит по частям данных* – они называются **батчи** (batch).

Мы должны «отрезать» некоторый кусочек данных, посчитать по нему loss, посчитать по нему градиентный шаг, сделать градиентный шаг, взять следующий кусочек, и так далее, повторять этот процесс. Соответственно, одна эпоха, то есть итерация просмотра всего датасета, бьётся на много маленьких частей.

Эту разбивку осуществим следующим образом. Нам понадобится функция numpy.random.permutation. Что делает эта функция? Если мы вызовем её с аргументом "5", она даст нам numpy.array размером "5", с числами от 0 до 4 включительно, случайно перемешанными (см. пример). Если в нее подставить размер нашего трейнового датасета, получим некоторые индексы в случайном порядке.

```
In [8]: np.random.permutation(5)
Out[8]: array([2, 4, 1, 0, 3])
```

Если от датасета взять эти индексы, то мы получим "пошафленный", перемешанный датасет. Таким образом, каждую эпоху мы будем «шаффлить» датасет, и потом резать его на части.

Пусть эти части будут размером десять элементов. Можно взять любое другое значение. Итак, каждую эпоху мы будем перемешивать датасет, у нас есть переменная

"order", которая определяется каждую эпоху, задает порядок индексов, применяемый к датасету.

Итак, будем вырезать участки длиной batch_size. Каждую эпоху будем делать перемешивание датасета, определять переменную order, которая отвечает за порядок элементов. После этого будем вычислять некоторое подмножество, начиная со start_index, который будет 0, 10, 20 и так далее, до конца батча. Batch_indexes – это некоторые индексы, которые соответствуют текущему батчу.

Таким образом, каждую эпоху гарантированно проходим все значения в датасете, при этом каждая итерация обучения происходит по десяти элементам.

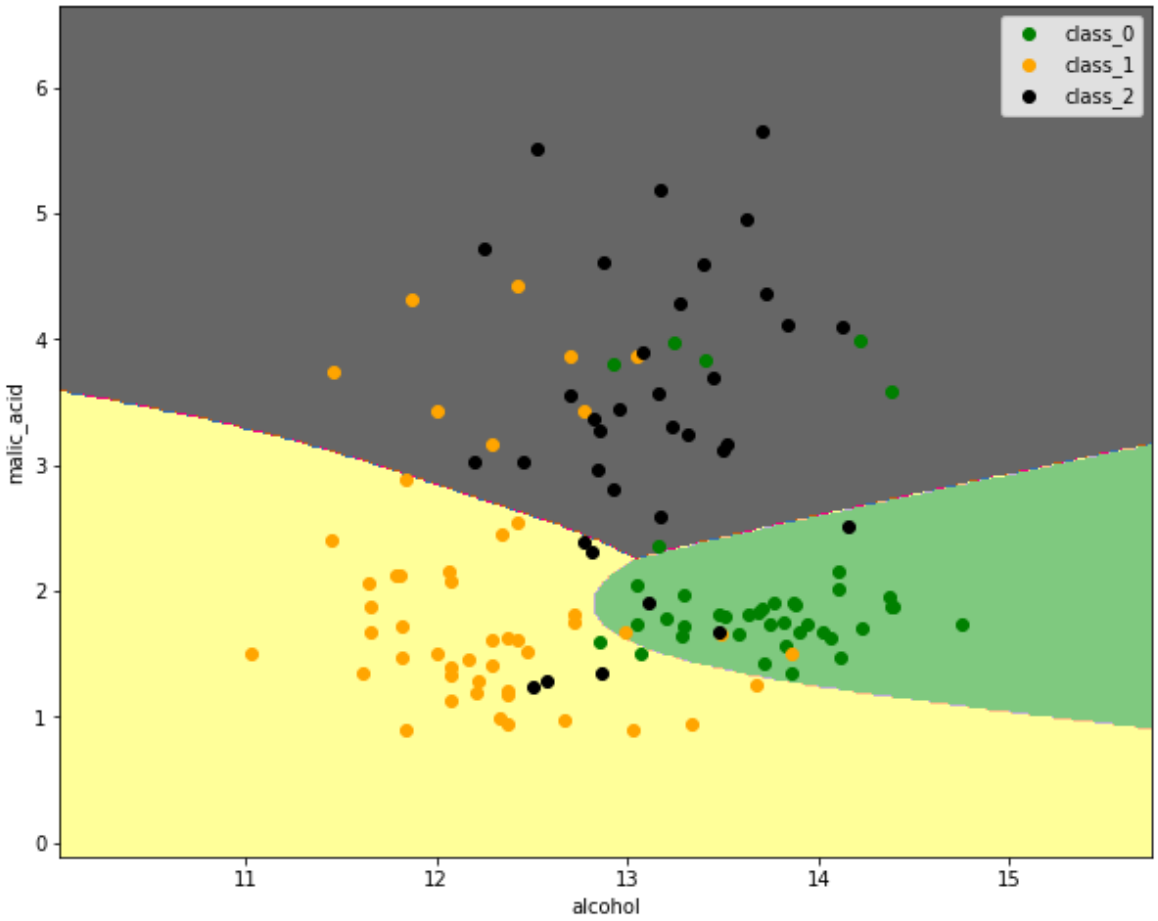
Каждые 100 эпох будем вычислять метрики на тестовом датасете чтобы посмотреть, обучается у нас нейросеть или нет. Таким образом, каждые 100 эпох мы делаем forward по тестовым данным, получаем тестовые prediction и вычисляем, какой выход был максимальный. На самом деле, чтобы понять, какой класс предсказывает нейросеть, не обязательно вычислять софтмакс, не обязательно вычислять вероятности. Достаточно посмотреть, какой выход был наибольший, и он же будет впоследствии выходом с максимальной вероятностью.

Необходимо посчитать argmax у выходов нейронной сети, это будет номер нейрона, затем сравнить его с тем номером класса, который находится в Y_test. После этого можно посчитать, какова доля этого совпадения, когда нейрон с максимальным выходом совпал с реально правильным классом. Следует посчитать среднее значение, но среднее значение нельзя посчитать у целочисленного тензора, который получается в результате этого сравнения, поэтому сначала надо преобразовать его к дробному тензору и вызвать метод mean().

Запустим обучение и посмотрим, что получится. Проходит по 100 эпох, получаем некоторое новое значение accuracy. Обычно оно растёт, правда иногда осциллирует в некоторых значениях и дальше увеличивается. Не обязательно ждать, пока обучение закончится, можно в любой момент его остановить вручную, если видно, что нейросеть уже сошлась и значения не изменяются.

Сделаем kernel interrupt и остановимся на accuracy где-то 0.85. Попробуем визуализировать результат. На графике точками обозначен трейновый датасет, то есть те точки, на которых обучалась нейронная сеть, а заполненными областями – то, как нейросеть классифицировала точки в соответствующих значениях. Видно, что нейросеть довольно неплохо справляется с тем, чтобы отделять точки разных классов.

```
tensor(0.8333)
tensor(0.8333)
tensor(0.8333)
tensor(0.8333)
tensor(0.8333)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)
tensor(0.8519)

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-9-f78af83497a6> in <module>
```



Если изменить архитектуру нейронной сети, можно попытаться улучшить результаты классификации. Например, можно изменить число нейронов в скрытых слоях, само количество скрытых слоёв, метод градиентного спуска, Learning rate, функции активации.

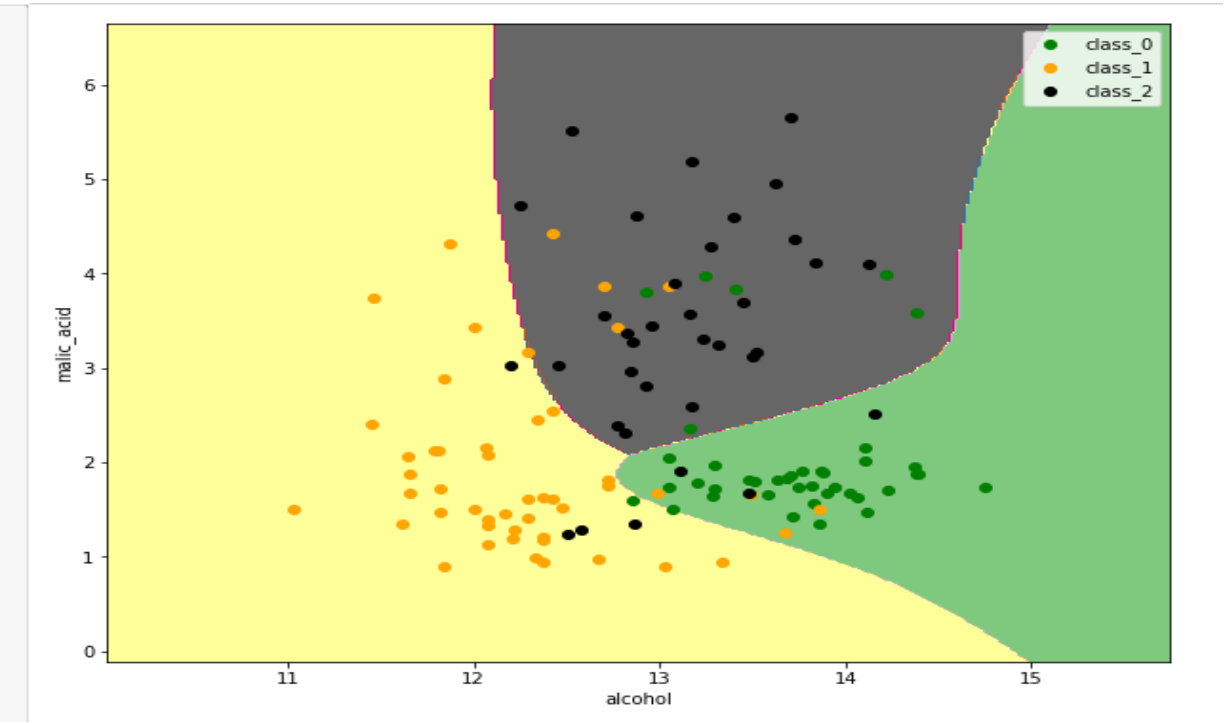
Например, изменим количество скрытых слоёв на один. Закомментируем слой.

In [11]:
```
class WineNet(torch.nn.Module):
    def __init__(self, n_hidden_neurons):
        super(WineNet, self).__init__()

        self.fc1 = torch.nn.Linear(2, n_hidden_neurons)
        self.activ1 = torch.nn.Sigmoid()
        #self.fc2 = torch.nn.Linear(n_hidden_neurons, n_hidden_neurons)
        #self.activ2 = torch.nn.Sigmoid()
        self.fc3 = torch.nn.Linear(n_hidden_neurons, 3)
        self.sm = torch.nn.Softmax(dim=1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.activ1(x)
        #x = self.fc2(x)
        #x = self.activ2(x)
        x = self.fc3(x)
        return x

    def inference(self, x):
        x = self.forward(x)
        x = self.sm(x)
        return x

wine_net = WineNet(5)
```

Увидим, что обучение пошло быстрее, потому что у нас меньше вычислений, меньше слоёв, и довольно быстро нейронная сеть выходит на те же значения.

```
tensor(0.8889)
tensor(0.8889)
tensor(0.8889)
tensor(0.8889)
tensor(0.8704)
tensor(0.8889)
tensor(0.8704)
tensor(0.9074)
tensor(0.9074)
tensor(0.9074)
tensor(0.8704)

---------------------------------------------
KeyboardInterrupt                         Trace
<ipython-input-17-f78af83497a6> in <module>
```

Видим, что картина поменялась. Возможно, эта картина более адекватная. Судя по метрике, она "угадывает на валидации" немножко лучше, отделяет классы друг от друга, и можно сделать вывод, что для данной задачи два скрытых слоя – это слишком много, это переусложнение, и хватает всего одного скрытого слоя, даже более того: кажется, что от уменьшения сложности нейронной сети мы выигрываем в качестве.




```
import torch
import random
import numpy as np
##
random.seed(0)
np.random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
torch.backends.cudnn.deterministic = True
##
```

```
import sklearn.datasets
wine = sklearn.datasets.load_wine()
wine.data.shape
##
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    wine.data[:, :2],
    wine.target,
    test_size=0.3,
    shuffle=True)
X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.LongTensor(y_train)
y_test = torch.LongTensor(y_test)
##
class WineNet(torch.nn.Module):
    def __init__(self, n_hidden_neurons):
```

```
        super(WineNet, self).__init__()

        self.fc1 = torch.nn.Linear(2, n_hidden_neurons)
        self.activ1 = torch.nn.Sigmoid()
        self.fc2 = torch.nn.Linear(n_hidden_neurons, n_hidden_neurons)
        self.activ2 = torch.nn.Sigmoid()
        self.fc3 = torch.nn.Linear(n_hidden_neurons, 3)
        self.sm = torch.nn.Softmax(dim=1)

    def forward(self, x):
        x = self.fc1(x)
        x = self.activ1(x)
        x = self.fc2(x)
        x = self.activ2(x)
        x = self.fc3(x)
        return x

    def inference(self, x):
        x = self.forward(x)
        x = self.sm(x)
        return x

wine_net = WineNet(5)
##




##
loss = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(wine_net.parameters(),
                             lr=1.0e-3)

##


batch_size = 10

for epoch in range(5000):
    order = np.random.permutation(len(X_train))
    for start_index in range(0, len(X_train), batch_size):
        optimizer.zero_grad()

        batch_indexes = order[start_index:start_index+batch_size]

        x_batch = X_train[batch_indexes]
        y_batch = y_train[batch_indexes]

        preds = wine_net.forward(x_batch)

        loss_value = loss(preds, y_batch)
        loss_value.backward()

        optimizer.step()

    if epoch % 100 == 0:
        test_preds = wine_net.forward(X_test)
        test_preds = test_preds.argmax(dim=1)
        print((test_preds == y_test).float().mean())
```

```
##




import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams['figure.figsize'] = (10, 8)

n_classes = 3
plot_colors = ['g', 'orange', 'black']
plot_step = 0.02

x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1

xx, yy =  torch.meshgrid(torch.arange(x_min, x_max, plot_step),
                         torch.arange(y_min, y_max, plot_step))

preds = wine_net.inference(
    torch.cat([xx.reshape(-1, 1), yy.reshape(-1, 1)], dim=1))

preds_class = preds.data.numpy().argmax(axis=1)
preds_class = preds_class.reshape(xx.shape)
plt.contourf(xx, yy, preds_class, cmap='Accent')

for i, color in zip(range(n_classes), plot_colors):
    indexes = np.where(y_train == i)
    plt.scatter(X_train[indexes, 0],
                X_train[indexes, 1],
                c=color,
                label=wine.target_names[i],
                cmap='Accent')
    plt.xlabel(wine.feature_names[0])
    plt.ylabel(wine.feature_names[1])
    plt.legend()
##
```

Задание на лабораторную работу

1. Изучить понятие **кросс-энтропии** и **Softmax**.
2. Исследовать нейронную сеть при заданных начальных параметрах (см. таблицу). Найти минимальное значение ***n_hidden_neurons***, при котором сеть дает неудовлетворительные результаты, т.е. обучение невозможно.
3. Исследовать зависимость точности распознавания от количества нейронов в скрытом слое, количества слоев, метода активации.
4. При каком значении test_size сеть предсказывает хуже, чем Base Rate (BaseRate – это вероятность самого многочисленного класса в исходных данных)? И какой Base Rate у датасета вин?

   Примечание: самый многочисленный класс датасета – первый.

   Base Rate = len(wine.target[wine.target == 1]) / len(wine.target)

5. Исследовать зависимость времени обучения от размера батча.

Таблица. Начальные значения гиперпараметров нейронной сети

Вариант	**Метод оптимизации**	**Число нейронов в скрытом слое** ***n_hidden_neurons***	**Шаг градиентного спуска** ***lr***
0	ADAM	10	0.01
1	ADAM	20	0.001
2	ADAM	30	0.01
3	ADAM	40	0.001
4	ADAM	5	0.01
5	SGD	10	0.001
6	SGD	20	0.01
7	SGD	30	0.001
8	SGD	40	0.01
9	SGD	50	0.001

Содержание отчета

1. Титульный лист
2. Цель работы, постановка задачи исследования.
3. Описание методики исследования.
4. Результаты исследования в соответствии с заданием.
5. Выводы по работе.