

Задача № 2

по курсу "Нейронные сети: задачи и вычисления"

студент: Бочкарев Фёдор Сергеевич

Загрузка библиотек

```
In [ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
```

Загрузка данных и выбор "предпочитаемой" страны

```
In [ ]: world_cities = pd.read_csv("worldcities.csv", header=None, skiprows=1)
world_cities = world_cities.to_numpy()

target_country = 'Russia'
world_cities[:,3] = np.where(world_cities[:,2] == target_country, 1, 0)

print(world_cities)
print(world_cities.shape)

X = world_cities[:,0:2]
y = world_cities[:,3]

X = X.astype(np.float32)
y = y.astype(np.float32)
```

```
[[35.685 139.7514 'Japan' 0]
 [40.6943 -73.9249 'United States' 0]
 [19.4424 -99.131 'Mexico' 0]
 ...
 [69.651 162.3336 'Russia' 1]
 [74.0165 111.51 'Russia' 1]
 [61.1333 -100.8833 'Canada' 0]]
(15493, 4)
```

1. Обучение

Структура нейронной сети:

- Входной слой 2
- Внутренний слой 128

- Активационный слой ReLU
- Слой нормализации (по batch)
- Слой Dropout (0.3)
- Внутренний слой 64
- Активационный слой ReLU
- Слой Dropout (0.3)
- Внутренний слой 1
- Активационный слой Sigmoid

Также важно отметить, что модель обучается на 80% данных (тех.задание) и для обучения используется предподготовка данных - происходит нормализация данных, а так же во время обучения влияние верных значений (с меткой 1) имеет больший вес (10 кратный), по сравнению с неверным значением (с меткой 0), это сделано исходя из того, что данные имеют явный дисбаланс.

Нормализуем данные

```
In [ ]: scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(12394, 2) (12394,)
(3099, 2) (3099,)
```

Структура модели
































```
In [ ]: model = Sequential()
model.add(Dense(128, input_dim=2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer=tf.optimizers.Adam(learning_rate=0.001),
              class_weights = {0: 1, 1: 10})
```

Обучение и проверка

```
In [ ]: history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=1, class_weight=class_weight)

y_pred = (model.predict(X_test) > 0.5).astype(int)
```

Epoch 1/50			
388/388		5s 4ms/step	- accuracy: 0.8662 - loss: 0.4059
Epoch 2/50			
388/388		1s 3ms/step	- accuracy: 0.9621 - loss: 0.1880
Epoch 3/50			
388/388		2s 4ms/step	- accuracy: 0.9605 - loss: 0.1651
Epoch 4/50			
388/388		1s 4ms/step	- accuracy: 0.9694 - loss: 0.1356
Epoch 5/50			
388/388		1s 3ms/step	- accuracy: 0.9652 - loss: 0.1305
Epoch 6/50			
388/388		1s 3ms/step	- accuracy: 0.9636 - loss: 0.1380
Epoch 7/50			
388/388		1s 4ms/step	- accuracy: 0.9695 - loss: 0.1306
Epoch 8/50			
388/388		1s 3ms/step	- accuracy: 0.9627 - loss: 0.1497
Epoch 9/50			
388/388		1s 4ms/step	- accuracy: 0.9612 - loss: 0.1464
Epoch 10/50			
388/388		1s 4ms/step	- accuracy: 0.9715 - loss: 0.1167
Epoch 11/50			
388/388		3s 4ms/step	- accuracy: 0.9693 - loss: 0.1386
Epoch 12/50			
388/388		1s 3ms/step	- accuracy: 0.9628 - loss: 0.1334
Epoch 13/50			
388/388		1s 3ms/step	- accuracy: 0.9676 - loss: 0.1240
Epoch 14/50			
388/388		2s 4ms/step	- accuracy: 0.9713 - loss: 0.1137
Epoch 15/50			
388/388		2s 4ms/step	- accuracy: 0.9673 - loss: 0.1203
Epoch 16/50			
388/388		1s 4ms/step	- accuracy: 0.9666 - loss: 0.1205
Epoch 17/50			
388/388		1s 4ms/step	- accuracy: 0.9712 - loss: 0.1179
Epoch 18/50			
388/388		1s 3ms/step	- accuracy: 0.9684 - loss: 0.1177
Epoch 19/50			
388/388		3s 3ms/step	- accuracy: 0.9735 - loss: 0.1009
Epoch 20/50			
388/388		1s 4ms/step	- accuracy: 0.9725 - loss: 0.0994
Epoch 21/50			
388/388		1s 3ms/step	- accuracy: 0.9669 - loss: 0.1086
Epoch 22/50			
388/388		1s 4ms/step	- accuracy: 0.9645 - loss: 0.1215
Epoch 23/50			
388/388		2s 4ms/step	- accuracy: 0.9709 - loss: 0.0972
Epoch 24/50			
388/388		1s 3ms/step	- accuracy: 0.9749 - loss: 0.0898
Epoch 25/50			
388/388		1s 4ms/step	- accuracy: 0.9720 - loss: 0.1000
Epoch 26/50			
388/388		1s 3ms/step	- accuracy: 0.9699 - loss: 0.1125
Epoch 27/50			
388/388		1s 3ms/step	- accuracy: 0.9653 - loss: 0.1141
Epoch 28/50			
388/388		1s 4ms/step	- accuracy: 0.9723 - loss: 0.1037
Epoch 29/50			
388/388		1s 3ms/step	- accuracy: 0.9633 - loss: 0.1065
Epoch 30/50			
388/388		1s 4ms/step	- accuracy: 0.9720 - loss: 0.1079
Epoch 31/50			
388/388		1s 3ms/step	- accuracy: 0.9753 - loss: 0.0996
Epoch 32/50			

```

388/388 ————— 1s 3ms/step - accuracy: 0.9708 - loss: 0.1072
Epoch 33/50
388/388 ————— 1s 4ms/step - accuracy: 0.9711 - loss: 0.0995
Epoch 34/50
388/388 ————— 1s 3ms/step - accuracy: 0.9701 - loss: 0.1120
Epoch 35/50
388/388 ————— 1s 3ms/step - accuracy: 0.9710 - loss: 0.0933
Epoch 36/50
388/388 ————— 1s 3ms/step - accuracy: 0.9752 - loss: 0.0901
Epoch 37/50
388/388 ————— 1s 4ms/step - accuracy: 0.9580 - loss: 0.1215
Epoch 38/50
388/388 ————— 1s 3ms/step - accuracy: 0.9753 - loss: 0.1010
Epoch 39/50
388/388 ————— 1s 4ms/step - accuracy: 0.9711 - loss: 0.1021
Epoch 40/50
388/388 ————— 1s 3ms/step - accuracy: 0.9767 - loss: 0.0963
Epoch 41/50
388/388 ————— 1s 3ms/step - accuracy: 0.9761 - loss: 0.0823
Epoch 42/50
388/388 ————— 1s 3ms/step - accuracy: 0.9741 - loss: 0.0891
Epoch 43/50
388/388 ————— 1s 4ms/step - accuracy: 0.9727 - loss: 0.0975
Epoch 44/50
388/388 ————— 1s 4ms/step - accuracy: 0.9723 - loss: 0.1062
Epoch 45/50
388/388 ————— 1s 3ms/step - accuracy: 0.9774 - loss: 0.0834
Epoch 46/50
388/388 ————— 1s 4ms/step - accuracy: 0.9746 - loss: 0.0922
Epoch 47/50
388/388 ————— 1s 4ms/step - accuracy: 0.9730 - loss: 0.1013
Epoch 48/50
388/388 ————— 1s 3ms/step - accuracy: 0.9707 - loss: 0.0988
Epoch 49/50
388/388 ————— 1s 4ms/step - accuracy: 0.9754 - loss: 0.0925
Epoch 50/50
388/388 ————— 1s 3ms/step - accuracy: 0.9746 - loss: 0.0853
97/97 ————— 0s 3ms/step

```

Графики потерь и точности

(чтобы убедиться что мы не переобучились/недообучились)

```

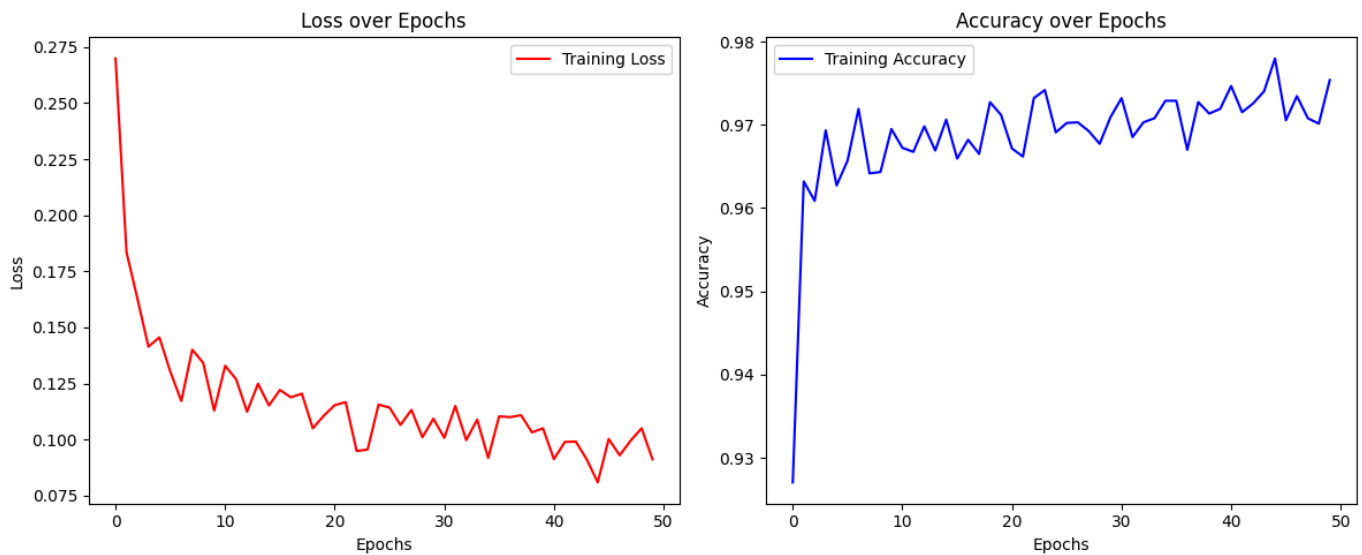
In [ ]: fig, axs = plt.subplots(1, 2, figsize=(12, 5))

axs[0].plot(history.history['loss'], label='Training Loss', color='red')
axs[0].set_title('Loss over Epochs')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
axs[0].legend()

axs[1].plot(history.history['accuracy'], label='Training Accuracy', color='blue')
axs[1].set_title('Accuracy over Epochs')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
axs[1].legend()

plt.tight_layout()
plt.show()

```



Расчёт метрики p

```
In [ ]: TN_d = np.empty((0,2),dtype=float)
TP_d = np.empty((0,2),dtype=float)
FN_d = np.empty((0,2),dtype=float)
FP_d = np.empty((0,2),dtype=float)

y_pred = y_pred.T[0]
X_test = scaler.inverse_transform(X_test)

TP_d = X_test[(y_test == 1) & (y_pred == 1)]
TN_d = X_test[(y_test == 0) & (y_pred == 0)]
FP_d = X_test[(y_test == 0) & (y_pred == 1)]
FN_d = X_test[(y_test == 1) & (y_pred == 0)]

TP = len(TP_d)
TN = len(TN_d)
FP = len(FP_d)
FN = len(FN_d)

print(TN,TP,FN,FP)
print(TN+TP+FN+FP)

p = 0.5 * ((TP / (TP + FN)) + (TN / (TN + FP)))
print(f'Metric p: {p}')
```

```
2936 105 1 57
3099
Metric p: 0.9857608003580682
```

Рассмотрение итогового результата

```
In [ ]: X_0 = X[y == 0]
X_1 = X[y == 1]

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(18, 12))

axes[0, 0].plot(TN_d[:,1], TN_d[:,0], "b.", label='TN', ms=3)
axes[0, 0].plot(TP_d[:,1], TP_d[:,0], "g.", label='TP', ms=3)
axes[0, 0].plot(FN_d[:,1], FN_d[:,0], "r.", label='FN', ms=3)
axes[0, 0].plot(FP_d[:,1], FP_d[:,0], "k.", label='FP', ms=3)
axes[0, 0].set_title('Тестовая выборка - целая карта')
axes[0, 0].grid(True)
axes[0, 0].set_xlim(-180, 180)
```

```

axes[0, 0].set_ylim(-60, 90)

axes[0, 1].plot(X_0[:,1], X_0[:,0], "b.", label='T', ms=3)
axes[0, 1].plot(X_1[:,1], X_1[:,0], "g.", label='F', ms=3)
axes[0, 1].set_title('Полная выборка - целая карта')
axes[0, 1].grid(True)
axes[0, 1].set_xlim(-180, 180)
axes[0, 1].set_ylim(-60, 90)

axes[1, 0].plot(TN_d[:,1], TN_d[:,0], "b.", label='TN', ms=9)
axes[1, 0].plot(TP_d[:,1], TP_d[:,0], "g.", label='TP', ms=9)
axes[1, 0].plot(FN_d[:,1], FN_d[:,0], "r.", label='FN', ms=9)
axes[1, 0].plot(FP_d[:,1], FP_d[:,0], "k.", label='FP', ms=9)
axes[1, 0].set_title('Тестовая выборка - Россия')
axes[1, 0].grid(True)
axes[1, 0].set_xlim(0, 180)
axes[1, 0].set_ylim(30, 90)

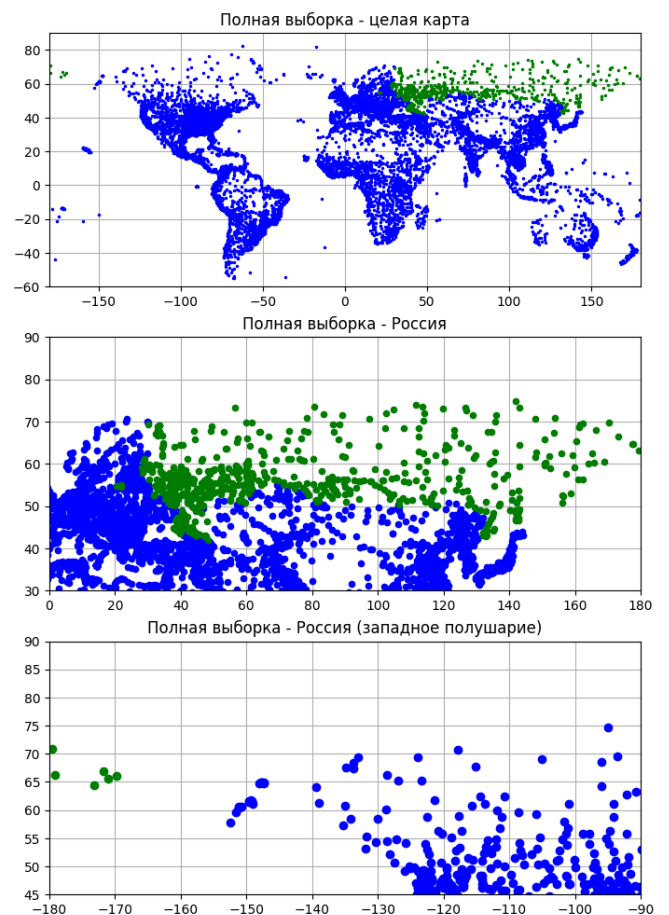
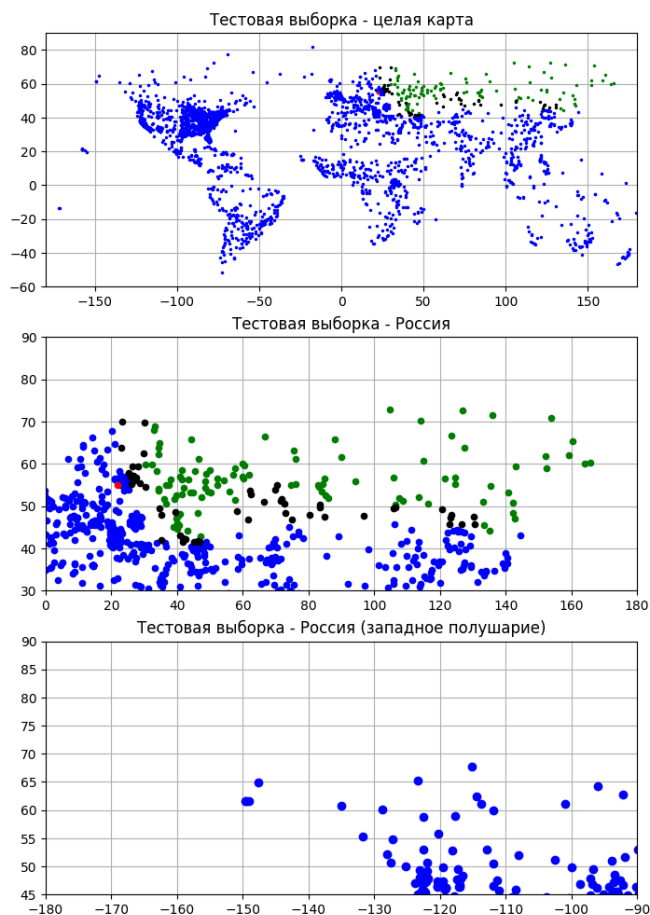
axes[1, 1].plot(X_0[:,1], X_0[:,0], "b.", label='T', ms=9)
axes[1, 1].plot(X_1[:,1], X_1[:,0], "g.", label='F', ms=9)
axes[1, 1].set_title('Полная выборка - Россия')
axes[1, 1].grid(True)
axes[1, 1].set_xlim(0, 180)
axes[1, 1].set_ylim(30, 90)

axes[2, 0].plot(TN_d[:,1], TN_d[:,0], "b.", label='TN', ms=12)
axes[2, 0].plot(TP_d[:,1], TP_d[:,0], "g.", label='TP', ms=12)
axes[2, 0].plot(FN_d[:,1], FN_d[:,0], "r.", label='FN', ms=12)
axes[2, 0].plot(FP_d[:,1], FP_d[:,0], "k.", label='FP', ms=12)
axes[2, 0].set_title('Тестовая выборка - Россия (западное полушарие)')
axes[2, 0].grid(True)
axes[2, 0].set_xlim(-180, -90)
axes[2, 0].set_ylim(45, 90)

axes[2, 1].plot(X_0[:,1], X_0[:,0], "b.", label='T', ms=12)
axes[2, 1].plot(X_1[:,1], X_1[:,0], "g.", label='F', ms=12)
# axes[2, 1].set_xlim(0, 6)
# axes[2, 1].set_ylim(0, 40)
axes[2, 1].set_title('Полная выборка - Россия (западное полушарие)')
axes[2, 1].grid(True)
axes[2, 1].set_xlim(-180, -90)
axes[2, 1].set_ylim(45, 90)

plt.show()

```



Итог

Нейронная сеть с данной структурой способна обучиться за 50 эпох и выдавать стабильный результат $p = 0.98-0.99$.

2. Расчёт размера

Напомню структуру нейронной сети:

- Входной слой 2
- Внутренний слой 128
- Активационный слой ReLU
- Слой нормализации (по batch)
- Слой Dropout (0.3)
- Внутренний слой 64
- Активационный слой ReLU
- Слой Dropout (0.3)
- Внутренний слой 1
- Активационный слой Sigmoid
- Оптимизатор Adam
- Скорость обучения 0.001

1. ReLU

Не имеет операций умножения.

2. Полносвязные слои

Каждый такой слой содержит в себе матрицу умножения, как например с 2 на 128 нейронов будет:

$$2 * 128 = 256 \text{ умножений}$$

3. Слои Dropout

Они выключают часть нейронов с предшествующего слоя (уменьшая общее число умножений), как например с слоя 64 через Dropout(0.3) на слой с 1 нейроном

$$64 * (1 - 0.3) * 1 \approx 45 \text{ умножений}$$

4. BatchNormalization

Данный слой довольно сложный, для расчёта числа умножений надо рассмотреть его структуру. Сперва вычисляется среднее

$$\mu = \frac{1}{m} \sum_{i=1}^m x_i \text{ (1 умножение на признак)}$$

Затем дисперсию:

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2 \text{ (} m + 1 \text{ умножений на признак)}$$

Нормализация:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \text{ где } \epsilon - \text{маленькое число (1 умножение на признак)}$$

Масштабирование и сдвиг:

$$\hat{y}_i = \gamma \hat{x}_i + \beta, \text{ где } \gamma, \beta - \text{параметры (1 умножение на признак)}$$

Получается в таком слое происходит суммарно $m + 4$ умножений на признак, где m - размер батча.

На самом деле в реализациях чаще всего используют не просто рассчитанные значения μ и σ^2 , а их скользящее среднее (что ещё больше увеличит число умножений)

5. Оптимизатор Adam

Внутри оптимизатора, на каждом слое происходит:

- Вычисление первого момента

$$m_{t+1} = \beta_1 * m_t + (1 - \beta_1) * g_t$$

- Вычисление второго момента

$$v_{t+1} = \beta_2 * v_t + (1 - \beta_2) * g_t^2$$

- Коррекция смещения

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

- Обновление параметров

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

С учётом того, что для каждого слоя надо ещё предварительно посчитать g_t

Подсчёт

- $2 * 128 = 256$ - переход с 2 нейронов на 128
- $128 * (32 + 4) = 4608$ - нормализация по батчу (на 32 примера)
- $128 * (1 - 0.3) * 64 = 5734.4$ - переход с 128 нейронов на 64 с выключением 30%
- $64 * (1 - 0.3) * 1 = 44.8$ - переход с 64 нейронов на 1 с выключением 30%
- $256 + 5734.4 + 44.8 = 6035.2$ - расчёт градиента ошибки для слоёв нейронов
- $128 * 7 = 896$ - расчёт градиента ошибки для нормализации по батчу ((1 на \hat{x} , 1 на μ , 2 на σ , 3 на x) \times признаков)
- $6035.2 * 9 = 54316.8$ - применение оптимизатора Adam ((2 на 1-й момент, 3 на 2-й момент, 2 на коррекцию, 2 на обновление веса) \times градиент ошибки)

На одну эпоху, с учётом того, что всего тестовая выборка состоит из 12394 примеров и размер батча 32 будет

$$(256 + 5734.4 + 44.8 + 6035.2 + 896 + 54316.8) * 12394 + 4608 * \frac{12394}{32} = 835692716.8$$

Итого 41784635840 умножений

3. Оптимизация

Попробуем теперь максимально снизить число умножений в нейронной сети.

Процесс поиска представлял обрезание и обнищание нейронной сети со структурой выше: сначала это были объективные вещи, такие как: отказ от нормализации по батчу и использование простого оптимизатора SGD, а не Adam. Далее было решено отбросить и регуляризацию, так как большого эффекта от неё не будет на малом количестве эпох.

В остатке, происходило уменьшение числа нейронов и даже слоёв пока ещё средний результат метрики p оставался в среднем выше 0.97 и в итоге была получена такая структура нейронной сети

Структура

- 2 входных нейрона
- 16 внутренних, активация ReLU
- 1 выходной, активация Sigmoid

Обучение с дисбалансом классов в 20 раз, количество эпох 20, оптимизатор SGD, скорость обучения 0.4

Так как нейронная сеть не всегда стабильно и чётко отрабатывает и результат очень сильно зависит от разбиения начальных данных и выборке примеров для обучения (всего обучение может быть на $15474 * 0.8 \approx 11700$ примерах, а в данном случае будет задействовано $32 * 20 = 640$ случайно выбранных). Поэтому для справедливой оценки качества предсказаний

данной нейронной сети, будем усреднять запуски обучения, пока не придём к определённому пределу (5 запусков без изменений среднего значения метрики p)

```
In [ ]: import warnings
warnings.filterwarnings('ignore', category=UserWarning)

df = pd.read_csv('worldcities.csv')
target_country = "Russia"
df['new_target'] = (df['country'] == target_country).astype(int)

X = df[['lat', 'lng']].values
y = df['new_target'].values

layer_1 = 16
true_class_weight = 20

num_epochs = 20

learning_rate = 0.4
model_optimizer = tf.optimizers.SGD(learning_rate=learning_rate)

num_runs = 0

losses = []
accuracies = []

TN_d_x = []
TN_d_y = []
TP_d_x = []
TP_d_y = []
FN_d_x = []
FN_d_y = []
FP_d_x = []
FP_d_y = []

pp = []
prev_rounded_mean_p = 0
while (num_runs < 5):
    # делим данные
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2)

    # строим модель
    model = Sequential()
    model.add(Dense(layer_1, input_dim=2, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    model_optimizer = tf.optimizers.SGD(learning_rate=learning_rate)
    model.compile(loss='binary_crossentropy', optimizer=model_optimizer, metrics=['accuracy'])

    # обучаем
    class_weights = {0: 1, 1: true_class_weight}
    history = model.fit(X_train, y_train, epochs=num_epochs, batch_size=32, verbose=0, class_weight=class_weights)

    # тестируем
    y_pred = (model.predict(X_test) > 0.5).astype(int)

    # запоминаем точность и потери
    losses.append(history.history['loss'])
    accuracies.append(history.history['accuracy'])
```

```

TN_d = np.empty((0,2),dtype=float)
TP_d = np.empty((0,2),dtype=float)
FN_d = np.empty((0,2),dtype=float)
FP_d = np.empty((0,2),dtype=float)

y_pred = y_pred.T[0]
X_test = scaler.inverse_transform(X_test)

TP_d = X_test[(y_test == 1) & (y_pred == 1)]
TN_d = X_test[(y_test == 0) & (y_pred == 0)]
FP_d = X_test[(y_test == 0) & (y_pred == 1)]
FN_d = X_test[(y_test == 1) & (y_pred == 0)]

TN_d_x.append(TN_d[:,1])
TN_d_y.append(TN_d[:,0])
TP_d_x.append(TP_d[:,1])
TP_d_y.append(TP_d[:,0])
FP_d_x.append(FP_d[:,1])
FP_d_y.append(FP_d[:,0])
FN_d_x.append(FN_d[:,1])
FN_d_y.append(FN_d[:,0])

TP = len(TP_d)
TN = len(TN_d)
FP = len(FP_d)
FN = len(FN_d)

print(TN,TP,FN,FP)
print(TN+TP+FN+FP)

p = 0.5 * ((TP / (TP + FN)) + (TN / (TN + FP)))
pp.append(p)
mean_p = sum(pp) / len(pp)
print(f'Metric p: {p}')
print("Prev pp:", prev_rounded_mean_p, "now", mean_p)
if (prev_rounded_mean_p == round(mean_p, 2)):
    num_runs += 1
else:
    num_runs = 0
prev_rounded_mean_p = round(mean_p, 2)

```

```

97/97 ----- 0s 3ms/step
2953 102 4 40
3099
Metric p: 0.9744498168682902
Prev pp: 0 now 0.9744498168682902
97/97 ----- 0s 3ms/step
2878 122 2 97
3099
Metric p: 0.9756329628625644
Prev pp: 0.97 now 0.9750413898654273
97/97 ----- 0s 3ms/step
2832 117 0 150
3099
Metric p: 0.9748490945674044
Prev pp: 0.98 now 0.974977291432753
97/97 ----- 0s 3ms/step
2826 121 2 150
3099
Metric p: 0.9666683057959612
Prev pp: 0.97 now 0.9729000450235551
97/97 ----- 0s 3ms/step
2846 103 1 149
3099
Metric p: 0.970317516373443
Prev pp: 0.97 now 0.9723835392935326
97/97 ----- 0s 3ms/step
2877 106 2 114
3099
Metric p: 0.9716835692261971
Prev pp: 0.97 now 0.9722668776156435
97/97 ----- 0s 3ms/step
2869 112 3 115
3099
Metric p: 0.9676870847418114
Prev pp: 0.97 now 0.9716126214908103
97/97 ----- 0s 3ms/step
2859 105 0 135
3099
Metric p: 0.9774549098196392
Prev pp: 0.97 now 0.9723429075319139

```

```

In [ ]: mean_p = sum(pp) / len(pp)
        print(f'Mean Metric p: {mean_p}')

fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# График потерь (Loss)
for i in range(len(pp)):
    axs[0].plot(losses[i])
axs[0].set_title('Loss over Epochs')
axs[0].set_xlabel('Epochs')
axs[0].set_ylabel('Loss')
# axs[0].legend()

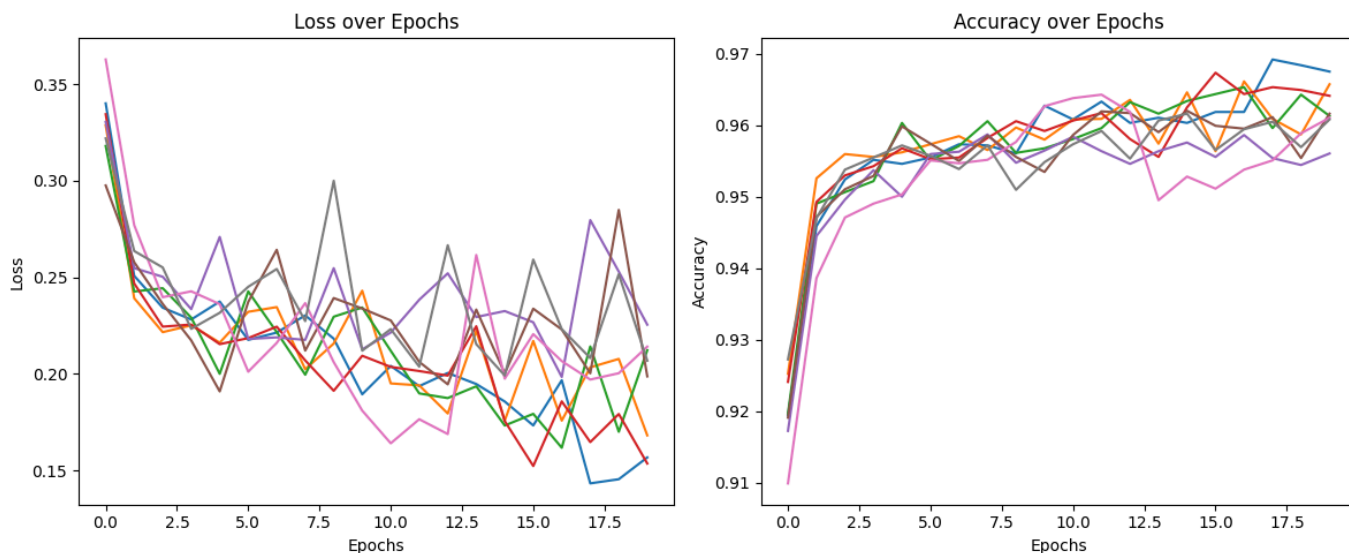
# График точности (accuracy)
for i in range(len(pp)):
    axs[1].plot(accuracies[i])
axs[1].set_title('Accuracy over Epochs')
axs[1].set_xlabel('Epochs')
axs[1].set_ylabel('Accuracy')
# axs[1].legend()

# Показать графики

```

```
plt.tight_layout()
plt.show()
```

Mean Metric p: 0.9723429075319139



```
In [ ]: X_0 = X[y == 0]
X_1 = X[y == 1]

TN_d_x = np.concatenate(TN_d_x)
TN_d_y = np.concatenate(TN_d_y)
TP_d_x = np.concatenate(TP_d_x)
TP_d_y = np.concatenate(TP_d_y)
FN_d_x = np.concatenate(FN_d_x)
FN_d_y = np.concatenate(FN_d_y)
FP_d_x = np.concatenate(FP_d_x)
FP_d_y = np.concatenate(FP_d_y)

fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(18, 12))

axes[0, 0].plot(TN_d_x, TN_d_y, "b.", label='TN', ms=3)
axes[0, 0].plot(TP_d_x, TP_d_y, "g.", label='TP', ms=3)
axes[0, 0].plot(FN_d_x, FN_d_y, "r.", label='FN', ms=3)
axes[0, 0].plot(FP_d_x, FP_d_y, "k.", label='FP', ms=3)
axes[0, 0].set_title('Тестовая выборка - целая карта')
axes[0, 0].grid(True)
axes[0, 0].set_xlim(-180, 180)
axes[0, 0].set_ylim(-60, 90)

axes[0, 1].plot(X_0[:,1], X_0[:,0], "b.", label='T', ms=3)
axes[0, 1].plot(X_1[:,1], X_1[:,0], "g.", label='F', ms=3)
axes[0, 1].set_title('Полная выборка - целая карта')
axes[0, 1].grid(True)
axes[0, 1].set_xlim(-180, 180)
axes[0, 1].set_ylim(-60, 90)

axes[1, 0].plot(TN_d_x, TN_d_y, "b.", label='TN', ms=9)
axes[1, 0].plot(TP_d_x, TP_d_y, "g.", label='TP', ms=9)
axes[1, 0].plot(FN_d_x, FN_d_y, "r.", label='FN', ms=9)
axes[1, 0].plot(FP_d_x, FP_d_y, "k.", label='FP', ms=9)
axes[1, 0].set_title('Тестовая выборка - Россия')
axes[1, 0].grid(True)
axes[1, 0].set_xlim(0, 180)
axes[1, 0].set_ylim(30, 90)

axes[1, 1].plot(X_0[:,1], X_0[:,0], "b.", label='T', ms=9)
axes[1, 1].plot(X_1[:,1], X_1[:,0], "g.", label='F', ms=9)
axes[1, 1].set_title('Полная выборка - Россия')
```

```

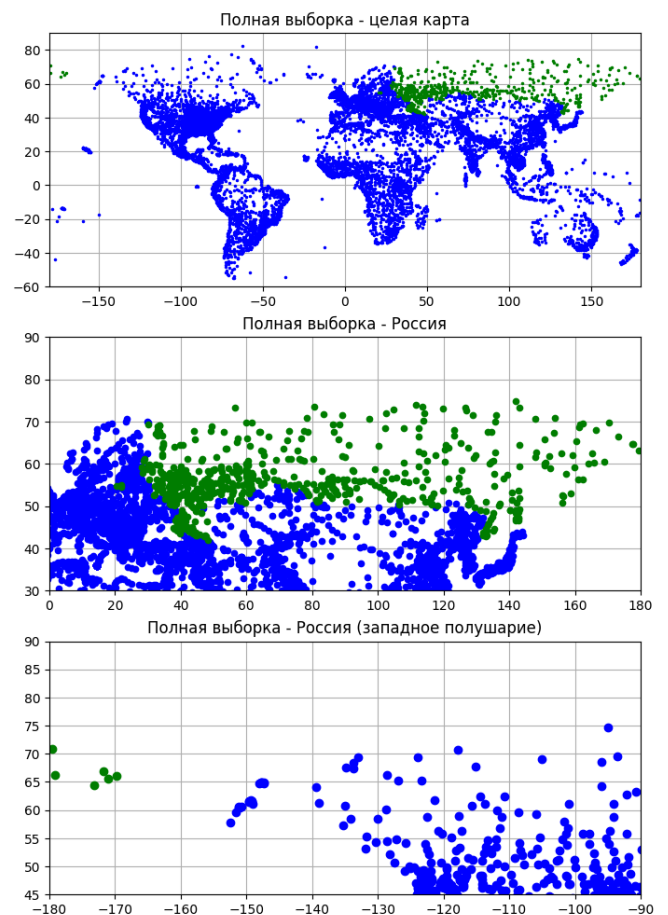
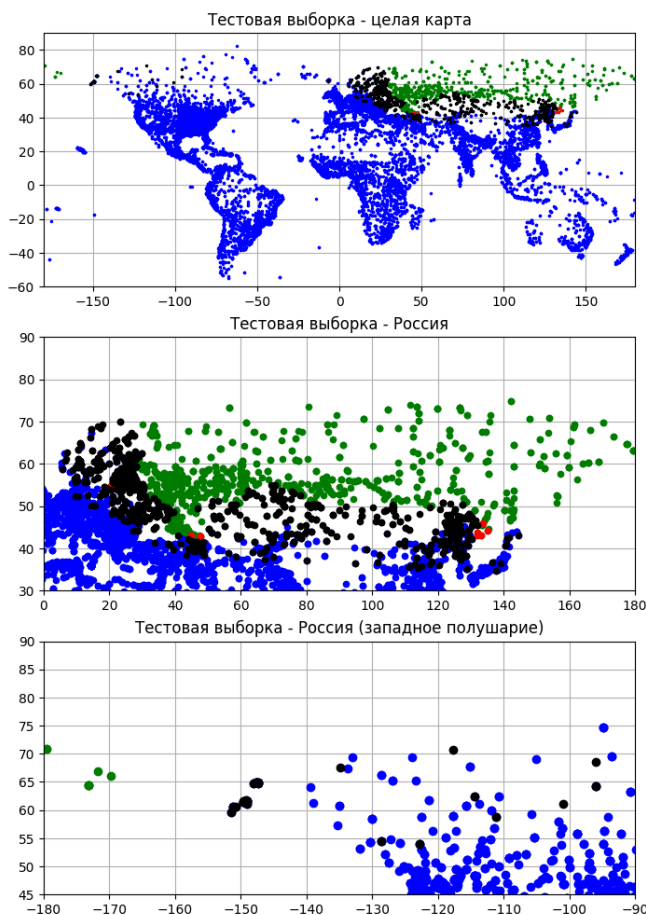
axes[1, 1].grid(True)
axes[1, 1].set_xlim(0, 180)
axes[1, 1].set_ylim(30, 90)

axes[2, 0].plot(TN_d_x, TN_d_y, "b.", label='TN', ms=12)
axes[2, 0].plot(TP_d_x, TP_d_y, "g.", label='TP', ms=12)
axes[2, 0].plot(FN_d_x, FN_d_y, "r.", label='FN', ms=12)
axes[2, 0].plot(FP_d_x, FP_d_y, "k.", label='FP', ms=12)
axes[2, 0].set_title('Тестовая выборка - Россия (западное полушарие)')
axes[2, 0].grid(True)
axes[2, 0].set_xlim(-180, -90)
axes[2, 0].set_ylim(45, 90)

axes[2, 1].plot(X_0[:,1], X_0[:,0], "b.", label='T', ms=12)
axes[2, 1].plot(X_1[:,1], X_1[:,0], "g.", label='F', ms=12)
# axes[2, 1].set_xlim(0, 6)
# axes[2, 1].set_ylim(0, 40)
axes[2, 1].set_title('Полная выборка - Россия (западное полушарие)')
axes[2, 1].grid(True)
axes[2, 1].set_xlim(-180, -90)
axes[2, 1].set_ylim(45, 90)

plt.show()

```



Подсчёт

- $2 * 16 = 32$ - переход с 2 нейронов на 16
- $16 * 1 = 16$ - переход с 16 нейронов на 1
- $32 + 16 = 48$ - расчёт градиента ошибки для слоёв нейронов
- 48 - применение оптимизатора SGD (каждый градиент на скорость обучения)

На одну эпоху, с учётом того, что размер батча 32 будет

$$12394 * (32 + 16 + 48 + 48) = 1784736$$

Итого 35694720 умножений (результат улучшился более чем в 1000 раз!)

Такой результат был достигнут из-за (1) оптимизаций, (2) небольшого снижения требований, что на самом деле тоже имеет весьма большое значение, так как достигнуть результата $p = 0.97$ в десятки раз проще чем $p = 0.98$