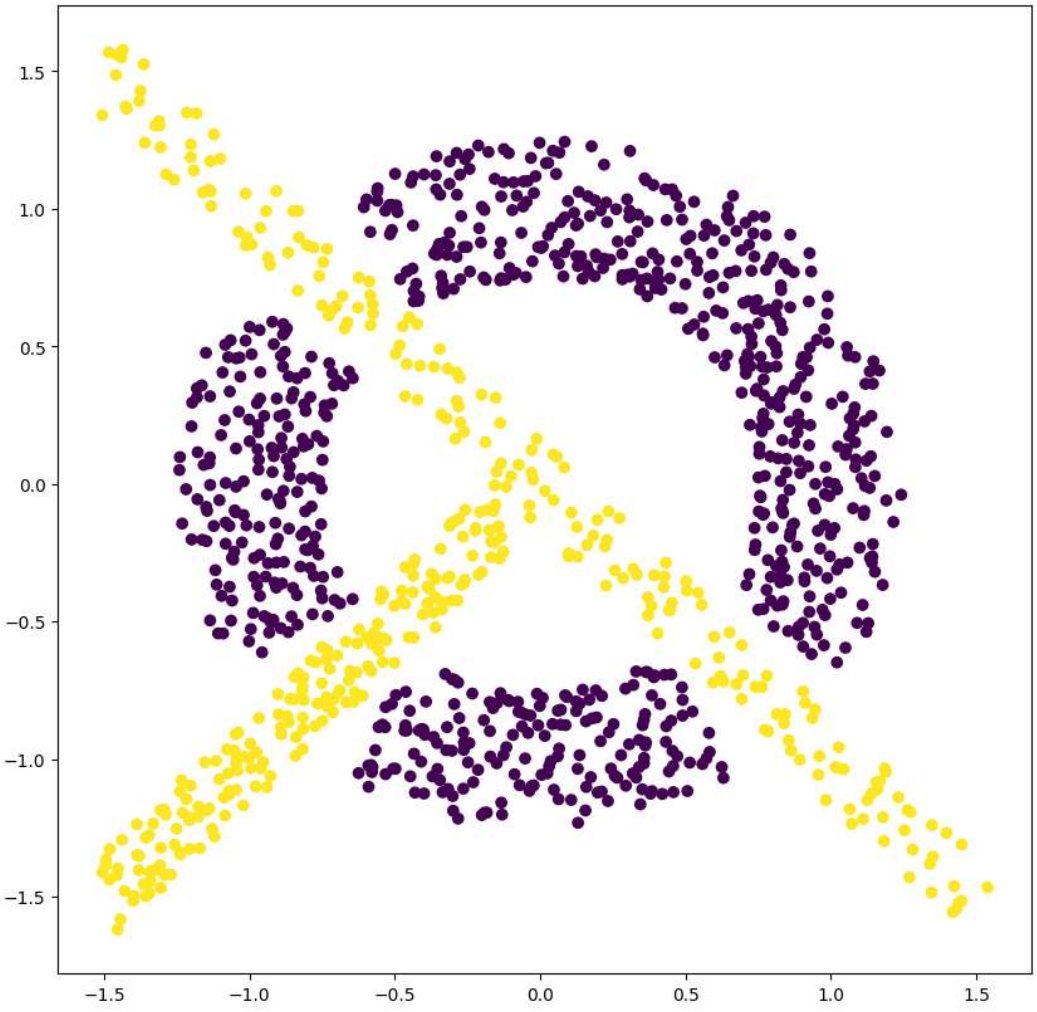
Задание.

Создать нейросеть, выполняющую разделение двух фигур на изображении, используя библиотеку keras. Выбрать для поставленной задачи наилучший оптимайзер, подобрать оптимальные коэффициенты для инициализации нейросети. Изучить работу функций обратного вызова EarlyStopping и ModelCheckpoint

Решение.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def plot_decision_boundary_keras(clf, X, Y, h=0.01, strict=False):
    x_min, x_max = X[:, 0].min(), X[:, 0].max()
    y_min, y_max = X[:, 1].min(), X[:, 1].max()

    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])[:, 0]
    Z = Z.reshape(xx.shape)
    Z_strict = Z >= 0.5

    plt.figure(figsize=(15, 15))
    if strict:
        plt.contourf(xx, yy, Z_strict, cmap=plt.cm.Blues, alpha=0.8);
    else:
        plt.contourf(xx, yy, Z, cmap=plt.cm.Blues, alpha=0.8);

    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.autumn);
    plt.title('Decision boundary');

def generate_dataset():
    ps = []
    for a in np.linspace(0, 2*np.pi, 1000):
        if (0.75*np.pi - 0.2) < a < (0.75*np.pi + 0.2):
            continue
        if (7.0*np.pi/4.0 - 0.2) < a < (7.0*np.pi/4.0 + 0.2):
            continue
        if (5.0*np.pi/4.0 - 0.2) < a < (5.0*np.pi/4.0 + 0.2):
            continue
        r = 1 + (np.random.rand()-0.5)*0.5
        x = np.cos(a)*r
        y = np.sin(a)*r
        ps.append([x,y])
    ps = np.array(ps)
```

```
    xs = np.linspace(-1.5, 1.5, 200)
    xs = xs + (np.random.rand(len(xs)) - 0.5)/10.0
    ys = -xs + (np.random.rand(len(xs)) - 0.5)/3.0

    xs2 = np.linspace(-1.5, -0.1, 200)
    xs2 = xs2 + (np.random.rand(len(xs2)) - 0.5)/10.0
    ys2 = xs2 + (np.random.rand(len(xs2)) - 0.5)/3.0

    ps2 = np.vstack([np.array([xs, ys]).T,
                     np.array([xs2, ys2]).T])

    X = np.vstack([ps, ps2])
    y = np.array([0]*ps.shape[0] + [1]*ps2.shape[0])

    return X, y

X, y = generate_dataset()

plt.figure(figsize=(10,10))
plt.scatter(X[:,0], X[:,1], c=y);
```




```
!pip install livelossplot
```

```
Collecting livelossplot
  Downloading livelossplot-0.5.5-py3-none-any.whl (22 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from livelossplot) (3.7.1)
Requirement already satisfied: bokeh in /usr/local/lib/python3.10/dist-
packages (from livelossplot) (3.2.2)
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.10/dist-
packages (from bokeh->livelossplot) (3.1.2)
Requirement already satisfied: contourpy>=1 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (1.1.1)
Requirement already satisfied: numpy>=1.16 in /usr/local/lib/python3.10/dist-
packages (from bokeh->livelossplot) (1.23.5)
Requirement already satisfied: packaging>=16.8 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (23.2)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-
packages (from bokeh->livelossplot) (1.5.3)
```

```
Requirement already satisfied: pillow>=7.1.0 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (9.4.0)
Requirement already satisfied: PyYAML>=3.10 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (6.0.1)
Requirement already satisfied: tornado>=5.1 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (6.3.2)
Requirement already satisfied: xyzservices>=2021.09.1 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
(2023.10.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->livelossplot)
(0.12.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->livelossplot)
(4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->livelossplot)
(1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->livelossplot)
(3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->livelossplot)
(2.8.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9->bokeh-
>livelossplot) (2.1.3)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2->bokeh-
>livelossplot) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib->livelossplot) (1.16.0)
Installing collected packages: livelossplot
Successfully installed livelossplot-0.5.5
```

```
from keras.models import Sequential
from keras.layers import Dense
from livelossplot import PlotLossesKeras
from keras.callbacks import Callback
from keras.optimizers import SGD, Adam
from livelossplot.inputs.keras import PlotLossesCallback
```

Добавьте в модель несколько полносвязных (Dense) слоёв. Определите оптимальное количество. Попробуйте использовать различные функции активации (sigmoid, relu, ..) и посмотрите, что будет получаться.

```
# пример модели
model = Sequential()
model.add(Dense(32, input_dim=2, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
```

```
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_46"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_165 (Dense)           (None, 32)                96

 dense_166 (Dense)           (None, 32)                1056

 dense_167 (Dense)           (None, 32)                1056

 dense_168 (Dense)           (None, 32)                1056

 dense_169 (Dense)           (None, 32)                1056

 dense_170 (Dense)           (None, 1)                 33

=================================================================
Total params: 4353 (17.00 KB)
Trainable params: 4353 (17.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

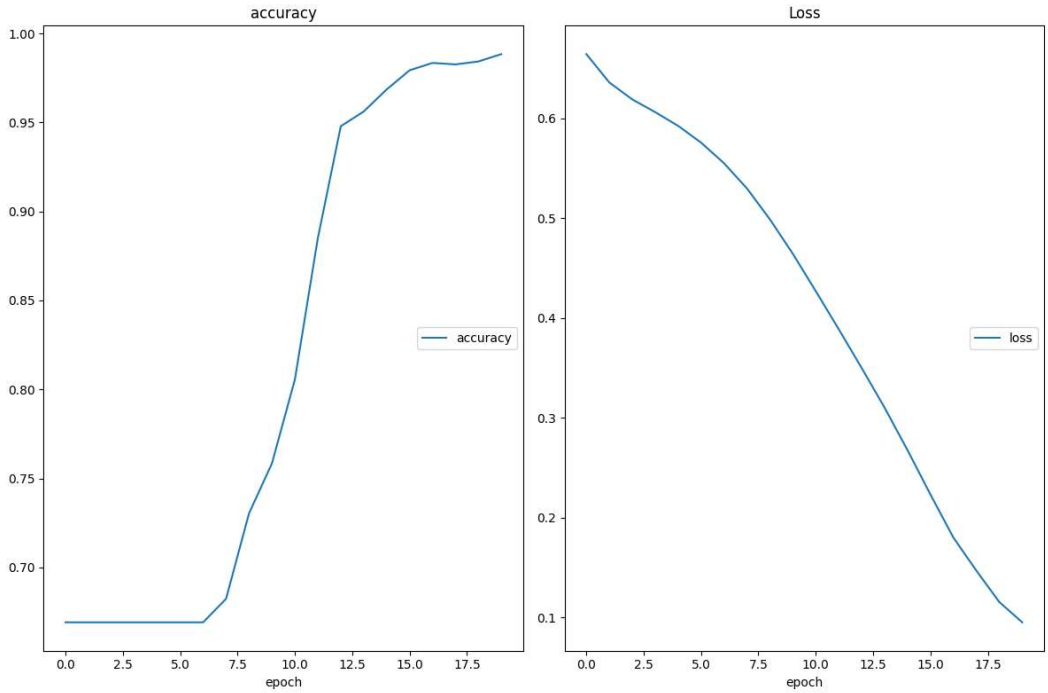
Подберите оптимальный размер батча, количество эпох и оптимайзер. Размер батча будет влиять на скорость обучения. Количество эпох должно быть достаточным для того, чтобы лосс перестал падать. Добейтесь того, чтобы две фигуры на изображении разделялись нейронной сетью.

```
def create_model():
    model = Sequential()
    model.add(Dense(32, input_dim=2, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Варианты гиперпараметров для проверки
batch_sizes = [10, 20, 30]
epochs_list = [10, 15, 20]
```

```
best_params = None
best_accuracy = 0

# Циклы для проверки различных комбинаций гиперпараметров
for batch_size in batch_sizes:
    for epochs in epochs_list:
        model = create_model()
        history = model.fit(X, y, batch_size=batch_size, epochs=epochs,
verbose=0)
        accuracy = history.history['accuracy'][-1]  # последняя точность
после всех эпох

        # Если текущая точность лучше предыдущей лучшей точности
        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_params = { 'batch_size': batch_size, 'epochs': epochs}

print(f"Лучшие параметры: {best_params} с точностью {best_accuracy *
100:.2f}%")
```

```
Лучшие параметры: {'batch_size': 10, 'epochs': 20} с точностью 99.34%
```

```
batch_size = 10
epochs = 20

model.compile('sgd', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X, y, batch_size=batch_size, epochs=epochs, verbose=0,
          callbacks=[PlotLossesKeras()])
```

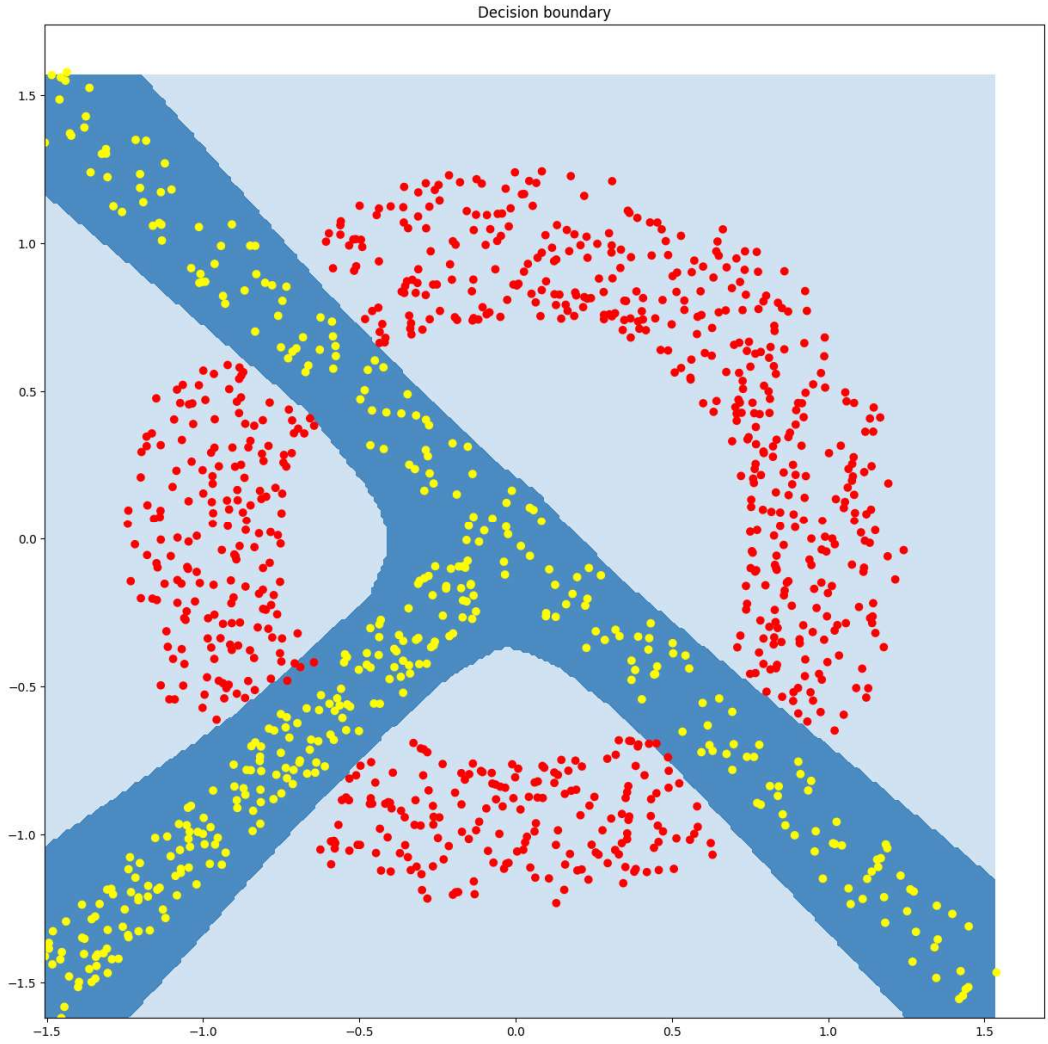


```
accuracy
      accuracy            (min:    0.669, max:    0.988, cur:    0.988)
Loss
      loss                (min:    0.095, max:    0.664, cur:    0.095)
```

```
<keras.src.callbacks.History at 0x77fcea664a60>
```

```
plot_decision_boundary_keras(model, X, y, strict=True)
```

```
3050/3050 [==============================] - 4s 1ms/step
```



Подсказка: в данном лучше можно начать с б__О__льших сетей и постепенно уменьшать. Посмотрите, какой минимально возможной сетью можно разделить точки.

```
from keras.optimizers import *
```

```
# Функция создания нейронной сети, с параметрами, найденными выше

def create_model():
    model = Sequential()
    model.add(Dense(32, input_dim=2, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Используя оптимайзер SGD, необходимо вручную подбирать параметр `learning_rate` (`lr`). Обычно он принимает значение от `1e-6` до `1e-1`. Подберите оптимальное значение `lr` для конкретной нейронной сети.

```
learning_rates = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
histories = []

for lr in learning_rates:
    # Создание новой модели
    model = create_model()

    # Определение оптимизатора SGD с текущим значением lr
    optimizer = SGD(learning_rate=lr)

    # Компиляция и обучение модели
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    history = model.fit(X, y, epochs=20, verbose=0)

    # Сохранение истории обучения
    histories.append(history)

histories
```

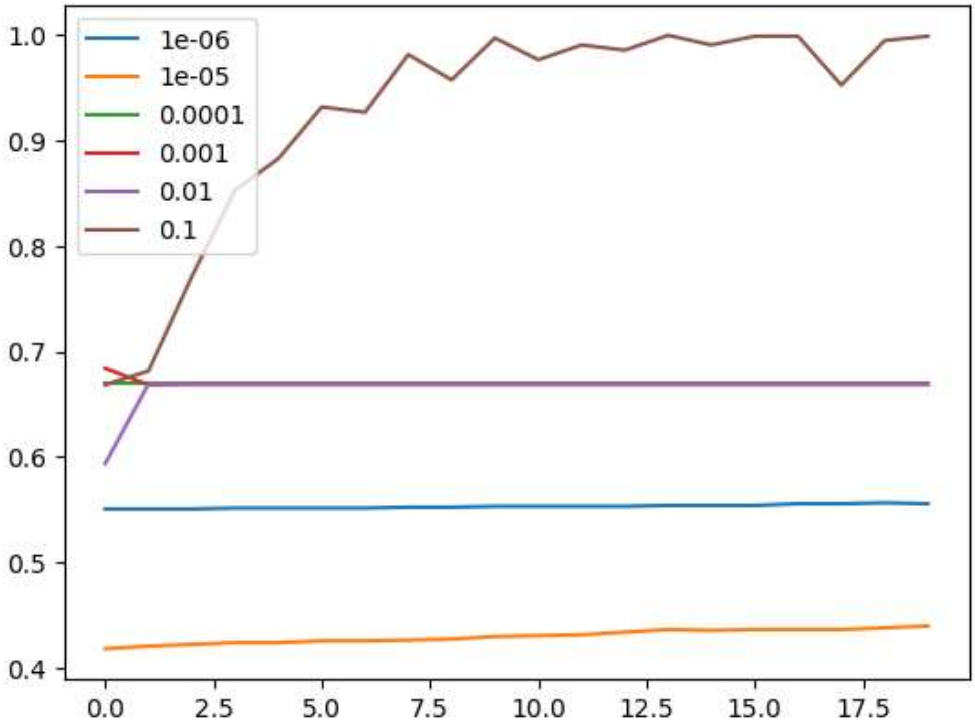
```
 [<keras.src.callbacks.History at 0x77fcc0284d90>,
 <keras.src.callbacks.History at 0x77fca5634df0>,
 <keras.src.callbacks.History at 0x77fd148c8250>,
 <keras.src.callbacks.History at 0x77fce810aa40>,
 <keras.src.callbacks.History at 0x77fcf010d3f0>,
 <keras.src.callbacks.History at 0x77fd1477b790>]
```

После вызова у модели метода `fit()` и завершения обучения возвращается объект `history` в котором можно смотреть, как падал лосс или другие заданные метрики.

```
accuracies = [h.history['accuracy'] for h in histories]
```

```
for acc, l in zip(accuracies, learning_rates):
    plt.plot(acc, label=l)
plt.legend()
```

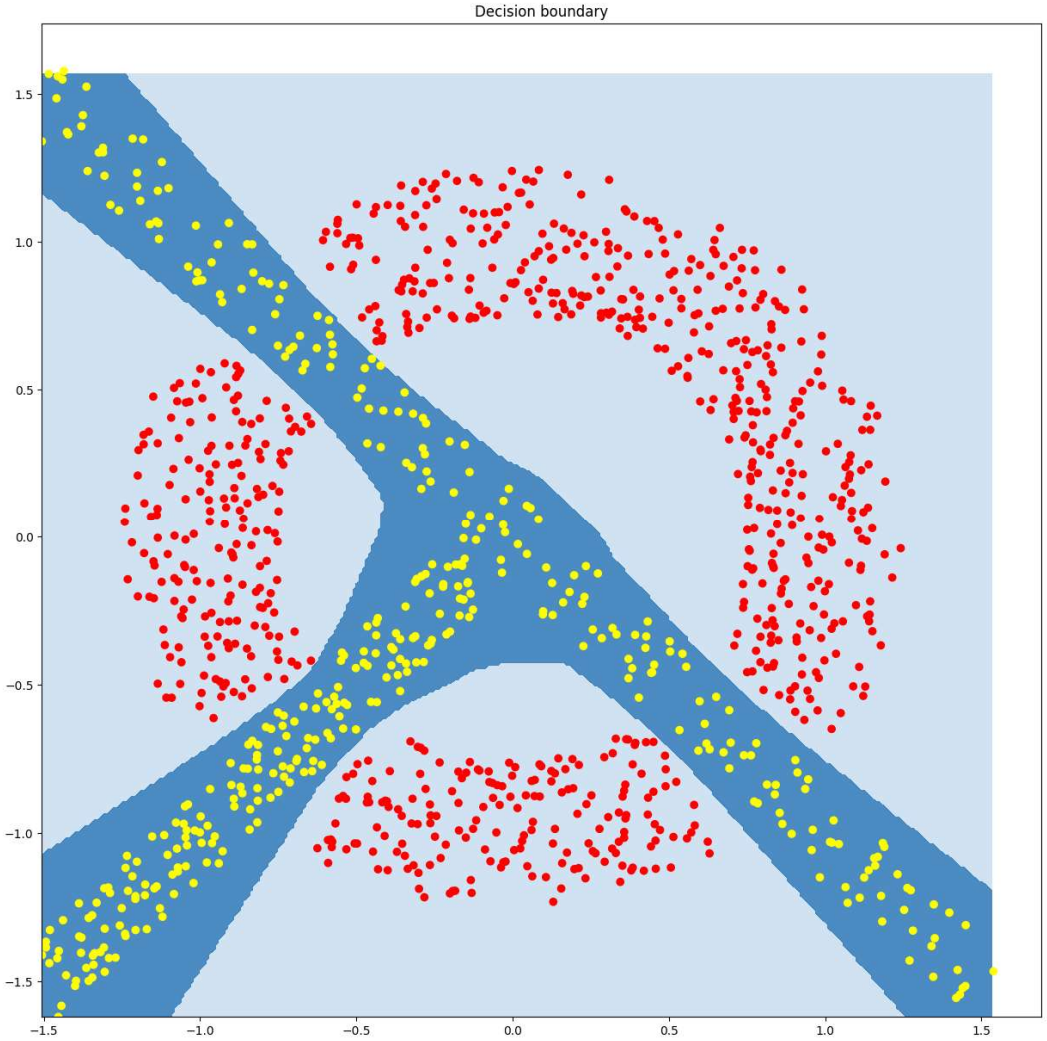
```
<matplotlib.legend.Legend at 0x77fd4a2ca230>
```



Наилучший оптимайзер: *SGD*

```
plot_decision_boundary_keras(model, X, y, strict=True)
```

```
3050/3050 [==============================] - 4s 1ms/step
```



Также можно подбирать параметр momentum, который поможет не застрять в локальном минимуме. Но если нет времени подбирать параметры SGD, то достаточно использовать оптимайзер `Adam` - он адаптивно подбирает `lr` сам. От выбранного оптимайзера будет зависеть скорость сходимости и минимально достижимое значение ошибки.

Список доступных оптимайзеров в keras: https://keras.io/optimizers/

После создания архитектуры и перед началом обучения нейронной сети её веса необходимо инициализировать. Веса инициализируют небольшими случайными значениями, но какими именно? От правильно выбора зависит сходимость сети и отсутствие затухающего или взрывающегося градиента.

Обычно подбирают только веса (), а отступы () инициализируют нулями.

Со списком всех возможных инициализаторов можно ознакомиться тут https://keras.io/initializers.

Необходимо выбрать наилучший.

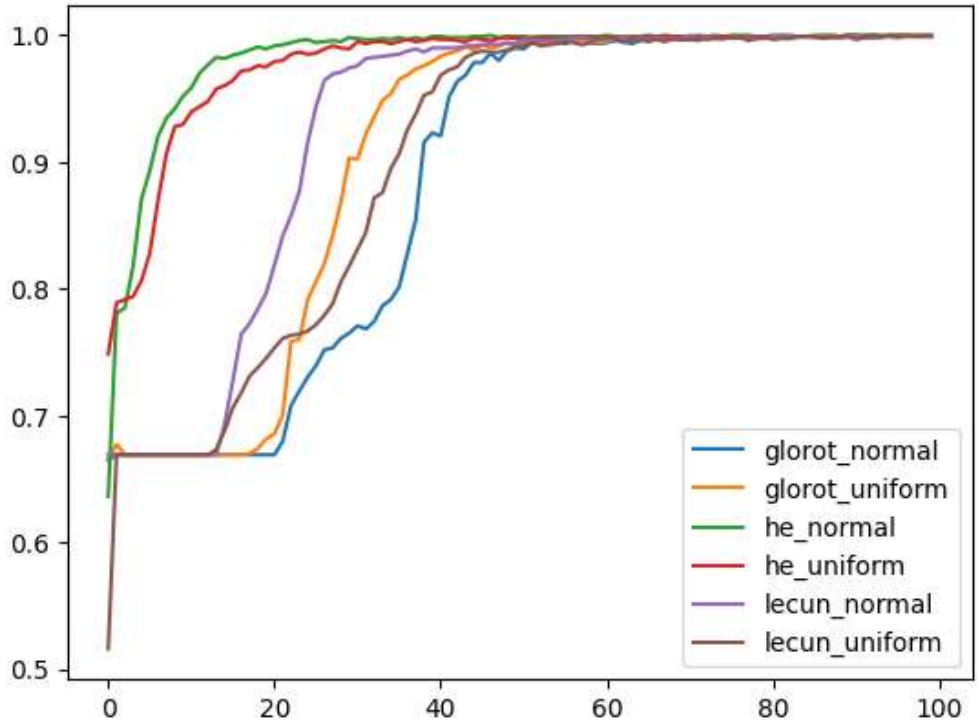
```
# Ваш код здесь
# Функция создания нейронной сети, с параметрами, найденными выше
# k_init - конкретное значение инициалайзера для параметра kernel_initializer
у каждого слоя
def create_model(k_init='glorot_uniform'):
    model = Sequential()
    model.add(Dense(32, input_dim=2, activation='relu',
kernel_initializer=k_init, bias_initializer='zeros'))
    model.add(Dense(32, activation='relu', kernel_initializer=k_init,
bias_initializer='zeros'))
    model.add(Dense(32, activation='relu', kernel_initializer=k_init,
bias_initializer='zeros'))
    model.add(Dense(32, activation='relu', kernel_initializer=k_init,
bias_initializer='zeros'))
    model.add(Dense(32, activation='relu', kernel_initializer=k_init,
bias_initializer='zeros'))
    model.add(Dense(1, activation='sigmoid', kernel_initializer=k_init,
bias_initializer='zeros'))
    return model
```

Самостоятельно найдите минмимум 4 различных варианта инициалайзера тут https://keras.io/initializers и сравните графики лосса для них

```
k_init = ['glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform',
'lecun_normal', 'lecun_uniform']
histories = []

for k in k_init:
    # Ваш код здесь
    # создайте модель с конкретным значением k_init обучите модель и запишите
history обучения модели в массив
    # в каждом цикле необходимо пересоздавать модель функцией create_model(),
чтобы обнулять веса

    # Создание новой модели с текущим инициализатором
    model = create_model(k_init=k)

    # Компиляция и обучение модели
    model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
    history = model.fit(X, y, epochs=100, verbose=0)

    # Сохранение истории обучения
    histories.append(history)

accuracies = [h.history['accuracy'] for h in histories]
```

```
for acc, l in zip(accuracies, k_init):
    plt.plot(acc, label=l)
plt.legend()
```

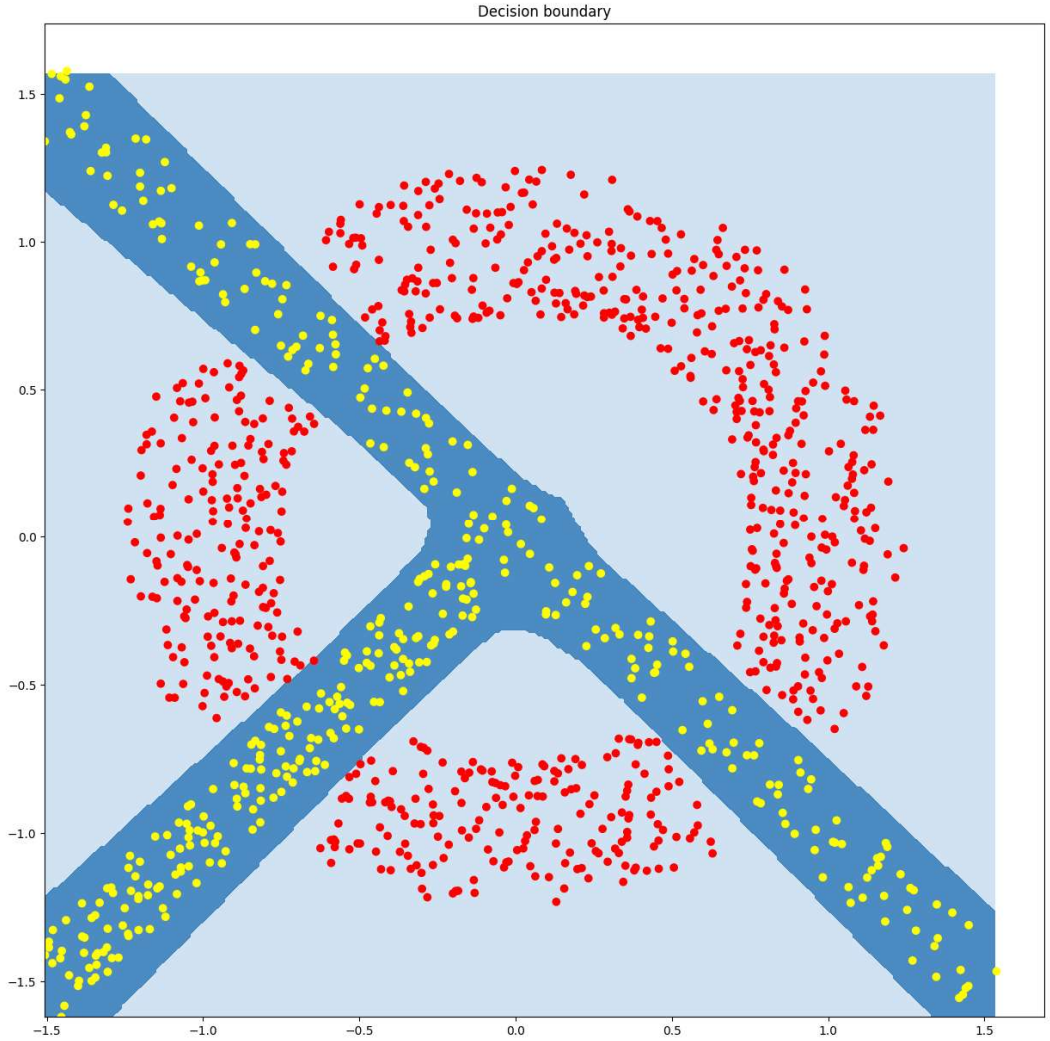
```
<matplotlib.legend.Legend at 0x77fcc01feda0>
```



Наилучший инициалайзер: *he_normal*

```
plot_decision_boundary_keras(model, X, y, strict=True)
```

```
3050/3050 [==============================] - 4s 1ms/step
```



Колбек - функция, которая выполняется после завершения эпохи или после каждого батча.

С одним из них вы уже знакомы: **PlotLossesKeras** рисует график лосса прямо во время обучения.

Попробуйте самстоятельно еще два самых полезных колбека:

- **EarlyStopping** останавливает обучение если лосс переставает падать
- **ModelCheckpoint** сохраняет веса модели по мере обучения. Это очень полезно, когда модель фитится по несколько дней

Подробнее про них тут https://keras.io/callbacks/

```
from keras.callbacks import EarlyStopping, ModelCheckpoint

# Создание модели
model = create_model()
```

```
model.compile('adam', loss='binary_crossentropy', metrics=['accuracy'])

# Определение колбеков
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1,
mode='min')  # остановка, если val_loss не улучшается в течение 10 эпох
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',
save_best_only=True, mode='min', verbose=1)  # сохранение лучшей модели по
val_loss

# Обучение модели с колбеками
model.fit(X, y, validation_split=0.2, epochs=100, batch_size=32,
callbacks=[early_stopping, model_checkpoint])

Epoch 1/100

16/31 [==============>..............] - ETA: 0s - loss: 0.6482 - accuracy:
0.8184

Epoch 1: val_loss improved from inf to 1.36957, saving model to best_model.h5
31/31 [==============================] - 2s 11ms/step - loss: 0.5972 -
accuracy: 0.8294 - val_loss: 1.3696 - val_accuracy: 0.0000e+00
Epoch 2/100
 1/31 [..............................] - ETA: 0s - loss: 0.3901 - accuracy:
0.9375

/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.

31/31 [==============================] - ETA: 0s - loss: 0.3843 - accuracy:
0.8366
Epoch 2: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.3843 -
accuracy: 0.8366 - val_loss: 3.0027 - val_accuracy: 0.0000e+00
Epoch 3/100
16/31 [==============>..............] - ETA: 0s - loss: 0.2934 - accuracy:
0.8359
Epoch 3: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.2771 -
accuracy: 0.8366 - val_loss: 4.2019 - val_accuracy: 0.0000e+00
Epoch 4/100
17/31 [===============>.............] - ETA: 0s - loss: 0.2086 - accuracy:
0.8787
Epoch 4: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.1898 -
accuracy: 0.8925 - val_loss: 6.0629 - val_accuracy: 0.0000e+00
Epoch 5/100
16/31 [==============>..............] - ETA: 0s - loss: 0.1345 - accuracy:
0.9434
Epoch 5: val_loss did not improve from 1.36957
```

```
31/31 [==============================] - 0s 5ms/step - loss: 0.1120 -
accuracy: 0.9679 - val_loss: 9.1351 - val_accuracy: 0.1240
Epoch 6/100
17/31 [===============>.............] - ETA: 0s - loss: 0.0571 - accuracy:
0.9982
Epoch 6: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.0529 -
accuracy: 0.9990 - val_loss: 11.9555 - val_accuracy: 0.1818
Epoch 7/100
31/31 [==============================] - ETA: 0s - loss: 0.0259 - accuracy:
0.9990
Epoch 7: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.0259 -
accuracy: 0.9990 - val_loss: 15.2037 - val_accuracy: 0.1818
Epoch 8/100
31/31 [==============================] - ETA: 0s - loss: 0.0124 - accuracy:
1.0000
Epoch 8: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.0124 -
accuracy: 1.0000 - val_loss: 17.1091 - val_accuracy: 0.1818
Epoch 9/100
16/31 [==============>.............] - ETA: 0s - loss: 0.0079 - accuracy:
1.0000
Epoch 9: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.0075 -
accuracy: 1.0000 - val_loss: 18.6695 - val_accuracy: 0.1860
Epoch 10/100
31/31 [==============================] - ETA: 0s - loss: 0.0045 - accuracy:
1.0000
Epoch 10: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.0045 -
accuracy: 1.0000 - val_loss: 19.9926 - val_accuracy: 0.1818
Epoch 11/100
31/31 [==============================] - ETA: 0s - loss: 0.0036 - accuracy:
1.0000
Epoch 11: val_loss did not improve from 1.36957
31/31 [==============================] - 0s 5ms/step - loss: 0.0036 -
accuracy: 1.0000 - val_loss: 20.8196 - val_accuracy: 0.1860
Epoch 11: early stopping

<keras.src.callbacks.History at 0x77fd14129690>

plot_decision_boundary_keras(model, X, y, strict=True)

3050/3050 [==============================] - 4s 1ms/step
```