Redes Neurais Artificiais - Exercício 5

January 19, 2021

Aluno: Victor São Paulo Ruela

[1]:
```
%load_ext autoreload
%autoreload 2

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_iris, load_breast_cancer
from sklearn.preprocessing import MinMaxScaler
from mpl_toolkits.mplot3d import Axes3D
```

1 Benchmark ELMs

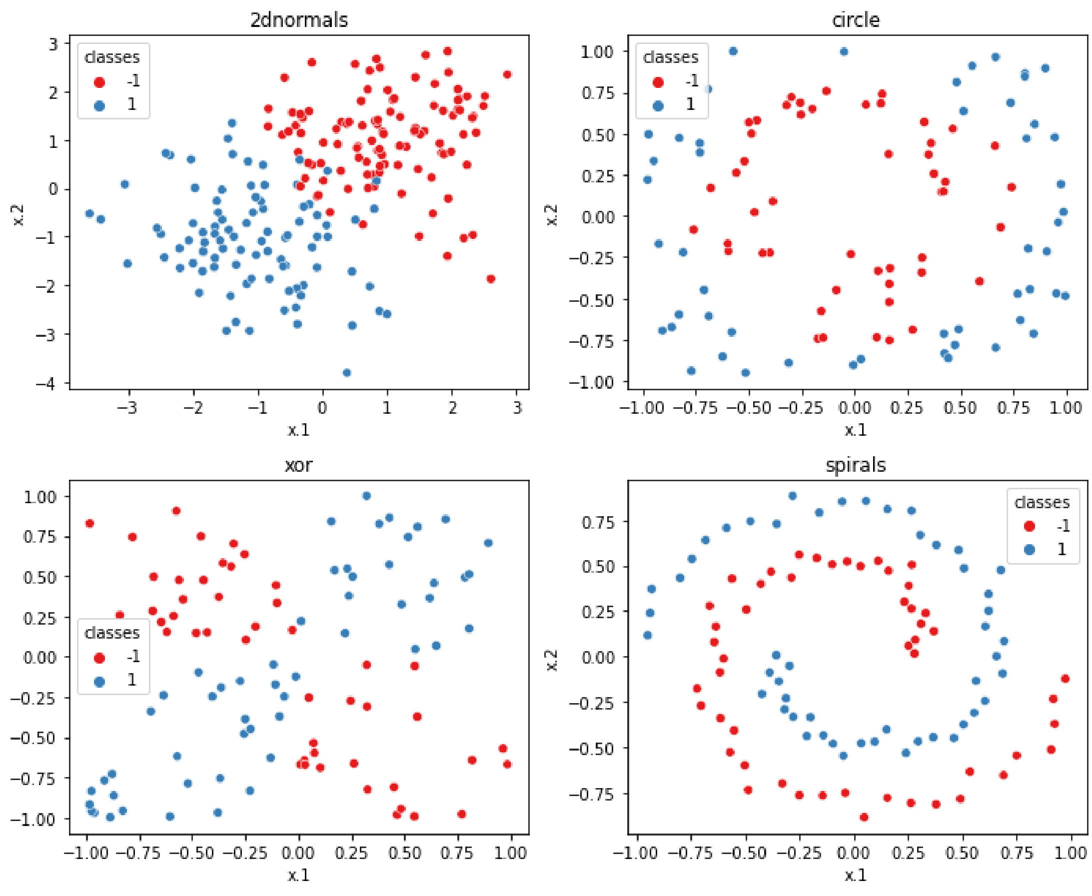
Inicialmente, carrega-se os dados para as funções indicadas no enunciado. Eles foram gerados utilizando o pacote R indicado e exportados para arquivos CSV.

[2]:
```
normals = pd.read_csv('2dnormals.csv')
xor = pd.read_csv('xor.csv')
circle = pd.read_csv('circle.csv')
spirals = pd.read_csv('spirals.csv')

# map classes as -1 or 1
normals['classes'] = normals['classes'].map({2:1, 1:-1})
xor['classes'] = xor['classes'].map({2:1, 1:-1})
circle['classes'] = circle['classes'].map({2:1, 1:-1})
spirals['classes'] = spirals['classes'].map({2:1, 1:-1})
```

Os dados gerados são exibidos na figura a seguir. Note que somente a base de dados `2dnormals` pode ser lineramente separável.

[3]:
```
# plot the functions
fig, ax = plt.subplots(2,2, figsize=(10,8))
```

```
sns.scatterplot(data=normals, x="x.1", y="x.2", hue='classes', ax=ax[0,0],␣
 ↪palette='Set1')
sns.scatterplot(data=xor, x="x.1", y="x.2", hue='classes',␣
 ↪ax=ax[1,0],palette='Set1')
sns.scatterplot(data=circle, x="x.1", y="x.2", hue='classes',␣
 ↪ax=ax[0,1],palette='Set1')
sns.scatterplot(data=spirals, x="x.1", y="x.2", hue='classes',␣
 ↪ax=ax[1,1],palette='Set1')

ax[0,0].set_title('2dnormals')
ax[1,0].set_title('xor')
ax[0,1].set_title('circle')
ax[1,1].set_title('spirals')

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



A seguir, é feita a implementação do algoritmo ELM.

[5]:
```python
# Implementação do perceptron simples para um problema de classificação binário
class ELM:
    def __init__(self, p=5):
        self.p = p

    def predict(self, x, w, H, Z):
        N, _ = x.shape
        x_aug = np.hstack((-np.ones((N, 1)), x))
        H = np.tanh(x_aug @ Z)
        u = np.sign(H @ w)
        return u

    def fit(self, x_train, y_train):
        N, n = x_train.shape
        # augment X
        x_aug = np.hstack((-np.ones((N, 1)), x_train))
        # create initial Z matrix
        Z = np.random.uniform(-0.5, 0.5, (n+1, self.p))
        # apply activation function: tanh
        H = np.tanh(x_aug @ Z)
        # calculate the weights
        w = np.linalg.pinv(H) @ y_train
        return w, H, Z
```

Em seguida, é criada uma rotina que recebe um conjunto de dados de entrada e desenha a sua superfície de separação conforme a sugestão do enunciado do exercício.

[6]:
```python
def plot_decision_boundary(data, p=5, plot_error=False):
    fig = plt.figure(figsize=(10,4))
    ax1 = fig.add_subplot(1, 2, 1)
    ax2 = fig.add_subplot(1, 2, 2, projection='3d')

    x1 = np.arange(np.min(data['x.1']) - 1, np.max(data['x.1']) + 1, step=0.1)
    x2 = np.arange(np.min(data['x.2']) - 1, np.max(data['x.2']) + 1, step=0.1)

    xx, yy = np.meshgrid(x1, x2)
    # flatten each grid to a vector
    r1, r2 = xx.flatten(), yy.flatten()
    r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))

    # horizontal stack vectors to create x1,x2 input for the model
    grid = np.hstack((r1,r2))

    # extract the data
    X, y = data[['x.1', 'x.2']].to_numpy(), data['classes'].to_numpy()

    # train the model
```

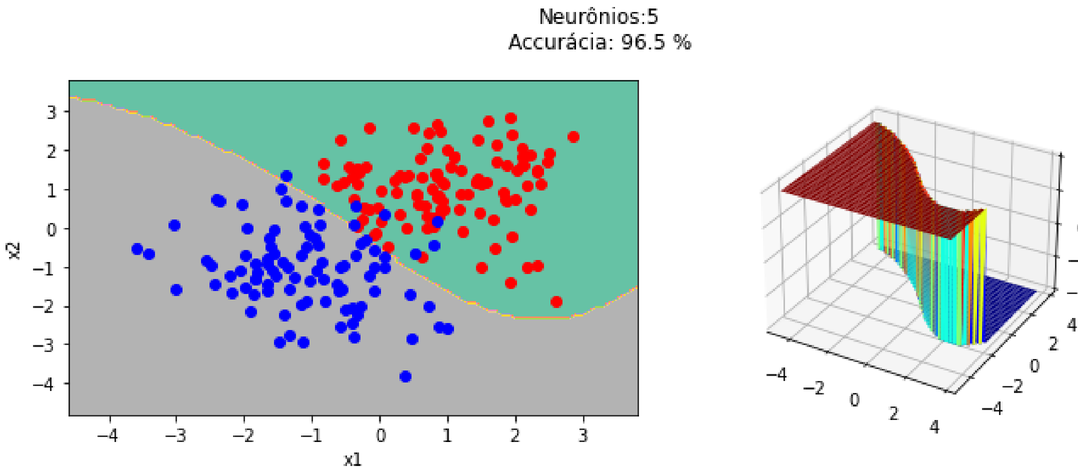
```
    model = ELM(p=p)
    w, H, Z = model.fit(X, y)

    # make predictions for the grid
    yhat = model.predict(grid, w, H, Z)
    # reshape the predictions back into a grid
    zz = yhat.reshape(xx.shape)
    ax1.contourf(xx, yy, zz, cmap='Set2')

    t_class0 = data['classes'] == -1
    t_class1 = data['classes'] == 1
    ax1.scatter(data.loc[t_class0, 'x.1'],
                data.loc[t_class0, 'x.2'], color='red')
    ax1.scatter(data.loc[t_class1, 'x.1'], data.loc[t_class1, 'x.2'],␣
↪color='blue')
    ax1.set_xlabel('x1')
    ax1.set_ylabel('x2')
    fig.suptitle(f'Neurônios:{p}\nAccurácia: {100 * np.sum(y == model.
↪predict(X,w, H, Z))/len(y)} %')

    surf = ax2.plot_surface(xx, yy, zz, cmap='jet')
    fig.tight_layout()
    fig.show()
```

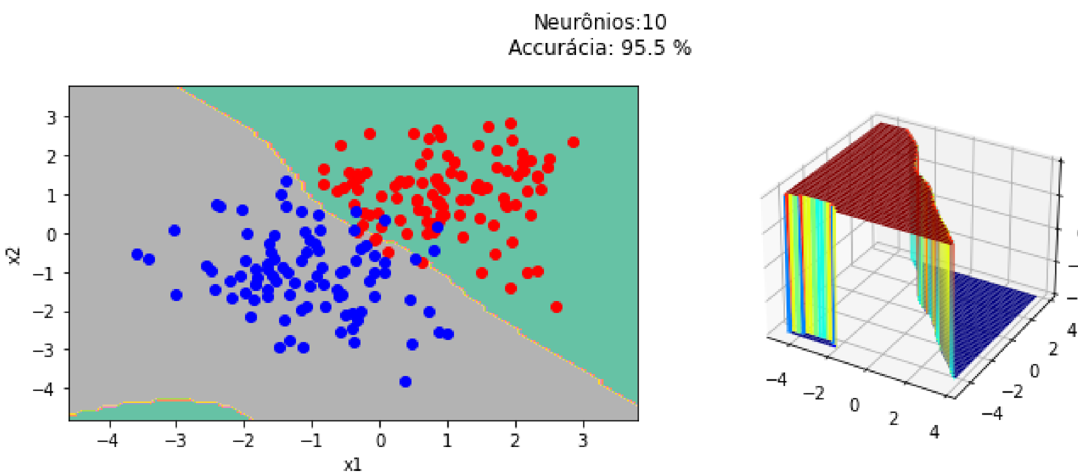
1.1 Análise dos Resultados

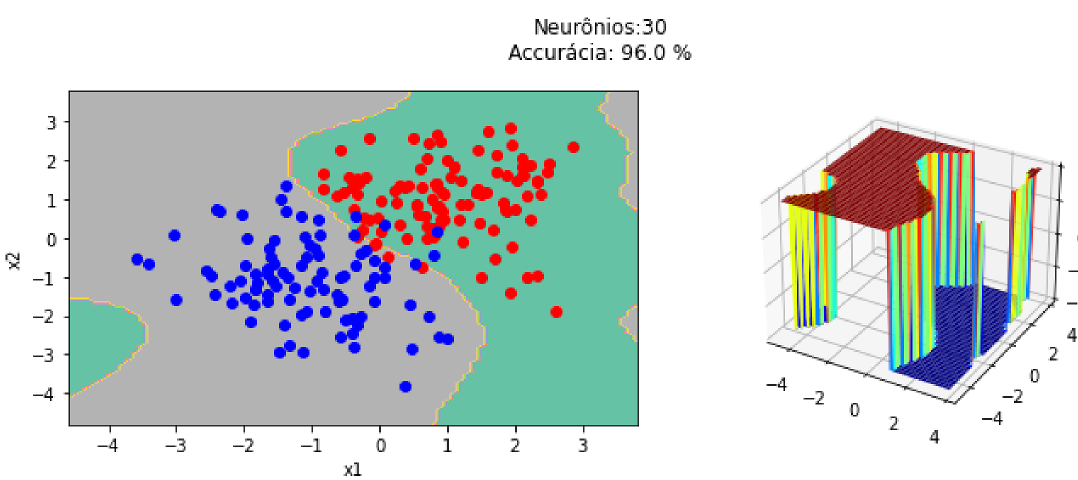
Para cada base de dados, são geradas superfícies de decisão considerando 5, 10 e 30 neurônios. Os resultados são discutidos a seguir.

1.1.1 Base de dados 2dnormals

[11]:
```
# define the neuros array:
neurons = [5, 10, 30]
# 2dnormals
for p in neurons:
    plot_decision_boundary(normals, p=p)
```


Neurônios:5
Accurácia: 96.5 %
x1
x2

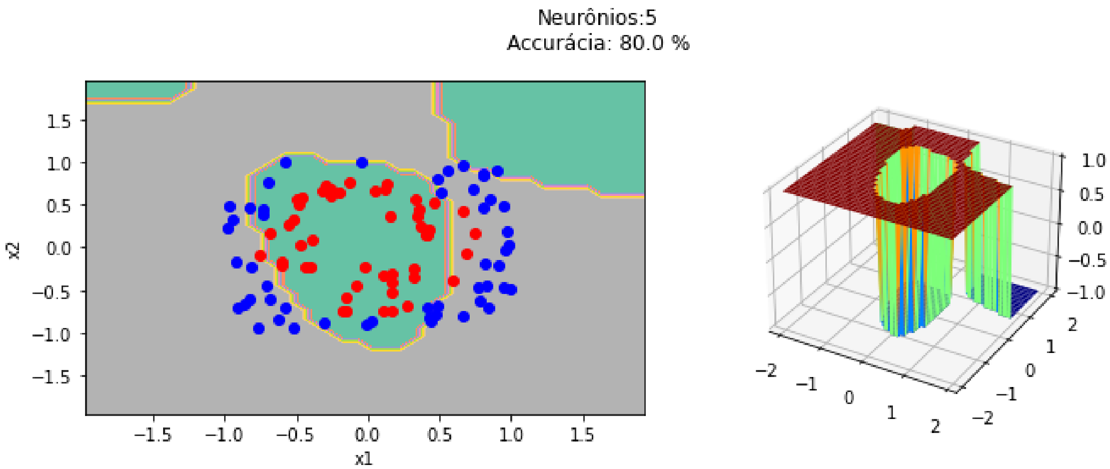

Neurônios:10
Accurácia: 95.5 %
x1
x2

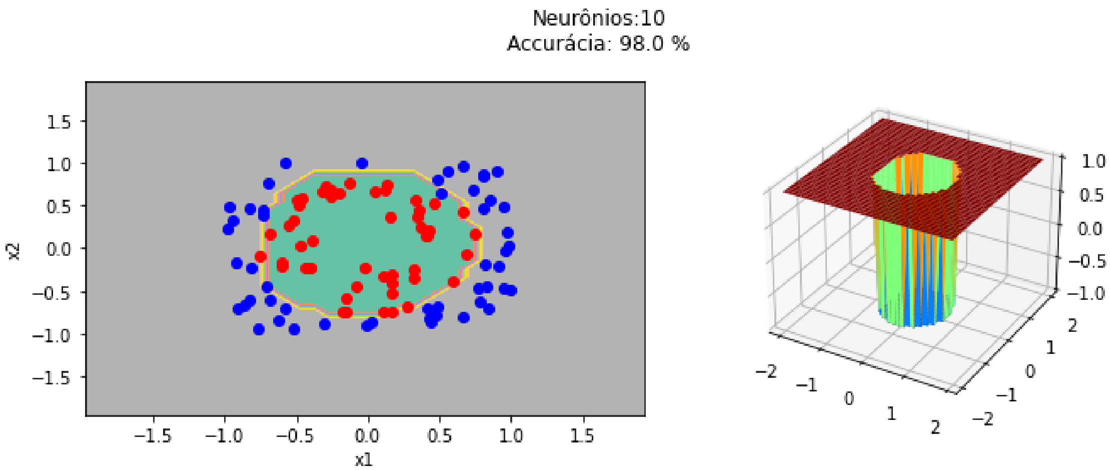

Neurônios:30
Accurácia: 96.0 %
x1
x2

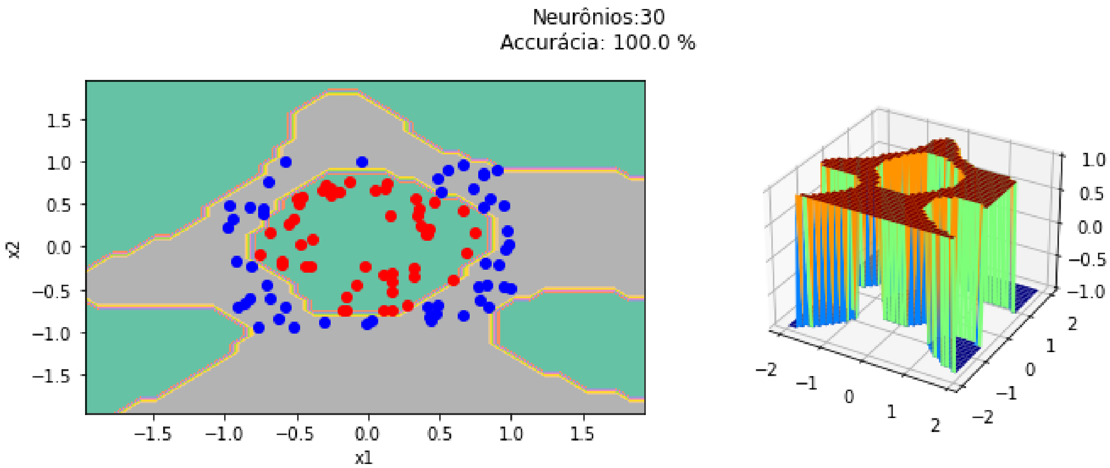
Esta base de dados é a mais simples, de forma que um modelo linear seria capaz de definir uma superfície de separação com alta acurácia. Logo, o ELM com 5 e 10 neurônios foi capaz de definir uma superfície de separação bastante eficiente, possivelmente com maior generalizaçao que os demais modelos. Através dos gráficos, é possível notar que o aumento do número de neurônios para 30 leva a um possível overfitting sobre os dados, resultando em uma superfície de separação bastante irregular.

1.1.2 Base de dados `circle`

[12]:
```python
# circle
for p in neurons:
    plot_decision_boundary(circle, p=p)
```





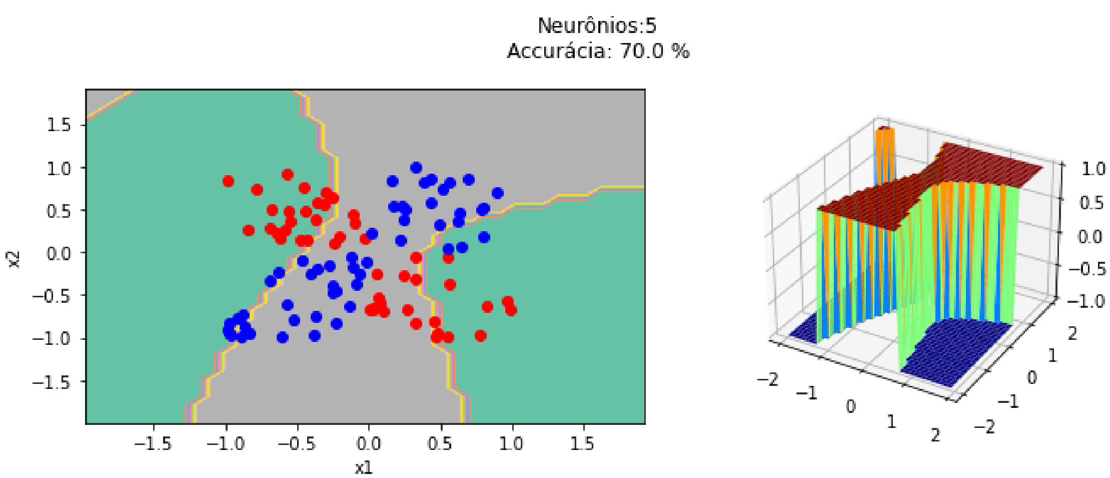


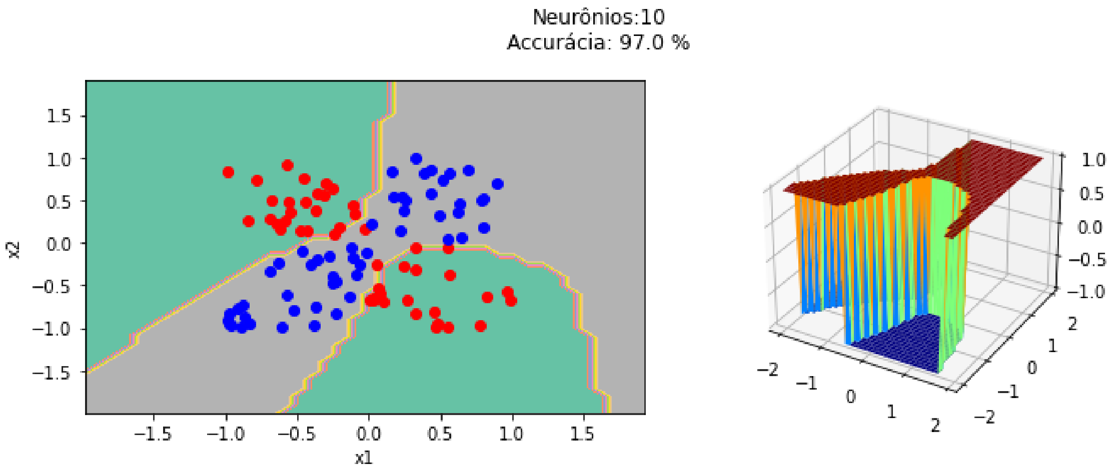
Por ser não-linermente separável, era de ser esperar que um número maior de neurônios fosse necessário para a obtenção de um modelo adequado com esta base dedos. Conforme visto na figura, o uso de 5 neurônios resulta em um underfitting dos dados, resultando em uma acurácia baixa. Aumentando o número de neurônios para 10 foram obtidos os melhores resultados, uma vez que a superfície de separação consegue separar perfeitamente os dados com boa generalização.

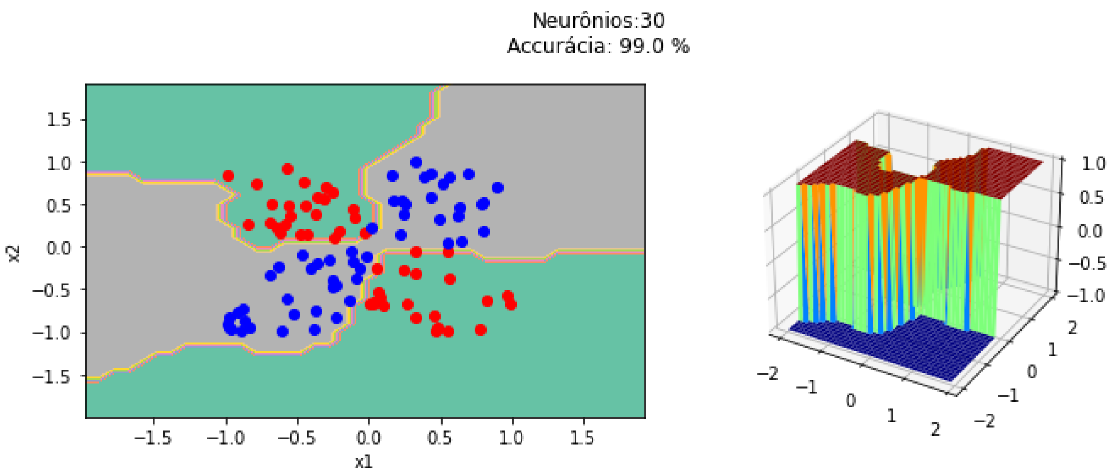
O uso de 30 neurônios sugere um overfitting aos dados, uma vez que a superfície de separação é bem irregular, apresentando uma estrutura diferente de um círculo (similar a um anel envolvendo os dados da classe azul). Em bora apresentou 100% de acurácia, é fácil ver que ela não teria uma generalização adequada para dados ainda não vistos.

1.1.3 Base de dados `xor`

[14]:
```
# xor
for p in neurons:
    plot_decision_boundary(xor, p=p)
```



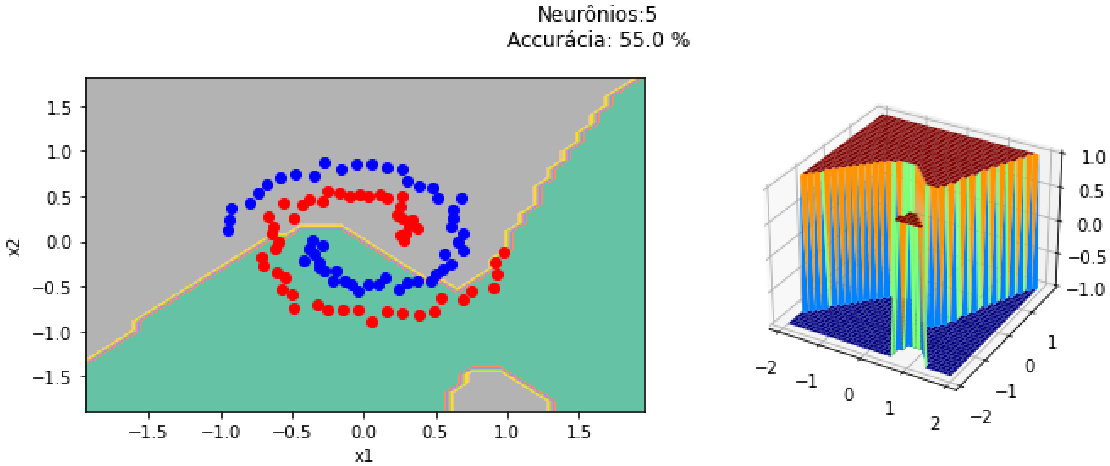


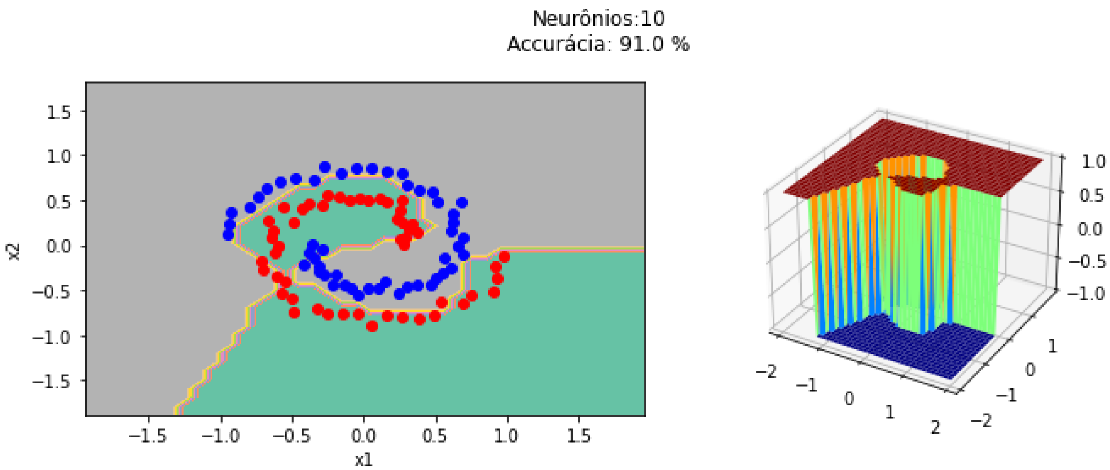


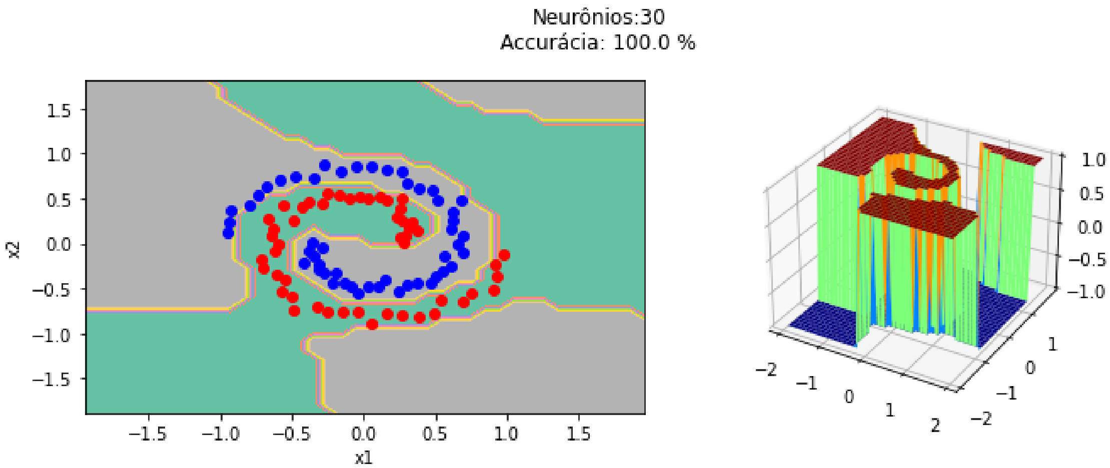
Para este modelo, nota-se que os melhores resultados foram obtidos para 10 e 30 neurônios. O uso de 5 neurônios, embora tenha aproximado com boa qualidade a superfície de separação, deixou um pouco a desejar na acurácia pois não foi capaz de separar bem os dados mais próximos à região central dos dados. O uso de uma quantidade maior de neurônios foi capaz de separar com mais eficiência esta região, conforme pode ser ser vista nas demais figuras.

1.1.4 Base de dados spirals

[15]:
```
# spirals
for p in neurons:
    plot_decision_boundary(spirals, p=p)
```







Conforme pode ser visto no gráfico, é necessária uma quantidade muito grande de neurônios para obter uma boa aproximação da superfície de separação destes dados. O uso de 5 neurônios não foi suficiente para obter uma boa aproximação, enquanto que a partir de 10 é possível ver que o algoritmo começa a se aproximar mais da superfície desejada. O uso de 30 neurônios obeteve os melhores resultados para esta base de dados.