Trabalho 4 de Inteligência Artificial – Manha

NOME	MATRICULA
Fabricio Baptista de Castro	0050481821007
Mario Celso Zanin	0050481921023

Código tp4_1a

```
from __future__ import division

import math
import random

import matplotlib.pyplot as plt
import numpy as np
from linear_algebra import dot


def sigmoid(t):
    return ((2 / (1 + math.exp(-t)))-1)

def neuron_output(weights, inputs):
    return sigmoid(dot(weights, inputs))

def feed_forward(neural_network, input_vector):
    outputs = []
    for layer in neural_network:
        input_with_bias = input_vector + [1]              # adiciona bias à entrada
        output = [neuron_output(neuron, input_with_bias) # calcula a saída do
neurônio
                  for neuron in layer]                    # para cada camada
        #print(output)
        outputs.append(output)

        # a saída de uma camada de neurônio é a entrada da próxima camada
        input_vector = output

    return outputs

alpha = 0.08

def backpropagate(network, input_vector, target):
    # feed_forward calcula a saída dos neurônios usando sigmóide
    hidden_outputs, outputs = feed_forward(network, input_vector)
    # 0.5 *alpha* (1 + output) * (1 - output) cálculo de derivada de sigmóide
    output_deltas = [0.5 * (1 + output) * (1 - output) * (output - target[i]) *
alpha
```

```
                    for i, output in enumerate(outputs)]

    # ajuste dos pesos sinápticos para camadas de saída (network[-1])
    for i, output_neuron in enumerate(network[-1]):
        for j, hidden_output in enumerate(hidden_outputs + [1]):
            output_neuron[j] -= output_deltas[i] * hidden_output

    # 0.5 *alpha* (1 +output)*(1-output) cálculo de derivada da sigmóide
    # retro-programação do erro para camadas intermediárias
    hidden_deltas = [ 0.5 * alpha * (1 + hidden_output) * (1 - hidden_output) *
                     dot(output_deltas, [n[i] for n in network[-1]])
                    for i, hidden_output in enumerate(hidden_outputs)]

    # ajuste dos pesos sinápticos para camadas intermediárias (network[0])
    for i, hidden_neuron in enumerate(network[0]):
        for j, input in enumerate(input_vector + [1]):
            hidden_neuron[j] -= hidden_deltas[i] * input

def seno(x): # função a ser aproximada pela rede neural
       seno = [(math.sin(2*math.pi/180*x)*math.sin(math.pi/180*x))]
       # seno é uma lista
       # [(0.8+(math.sin(math.pi/180*x)*math.sin(2*math.pi/180*x)))*0.5]
       return [seno]

def predict(inputs):
     return feed_forward(network, inputs)[-1]

inputs = []
targets = []
for x in range(360):
    seno_a = seno(x)

# TREINAMENTO DA REDE NEURAL

random.seed(0) # valores iniciais de pesos sinápticos
input_size = 1 # dimensão do vetor de entrada
num_hidden = 2 # número de neurônios na camada intermediária
output_size = 1 # dimensão das camadas de saída = 1 neurônio

"""
inserindo manualmente os vetores relativos à camada intermediária e a saída da
Rede Neural
hidden_layer = [[-0.085, -0.09], [-0.033, -0.08], [-0.074, -0.063], [-0.075,
-0.065], [-0.088, -0.076], [-0.077, -0.072]]
output_layer = [[0.082, -0.09, 0.064, -0.08, 0.084, -0.075, 0.099]] """

# cada neurônio da camada intermediária tem um peso sináptico associado à entrada
# e adicionado o peso do bias
hidden_layer = [[random.random() for __ in range(input_size + 1)]
                for __ in range(num_hidden)]

#print(hidden_layer)
# neurônio de saída tem um peso sináptico associado a cada neurôio da camada
intermediária
```

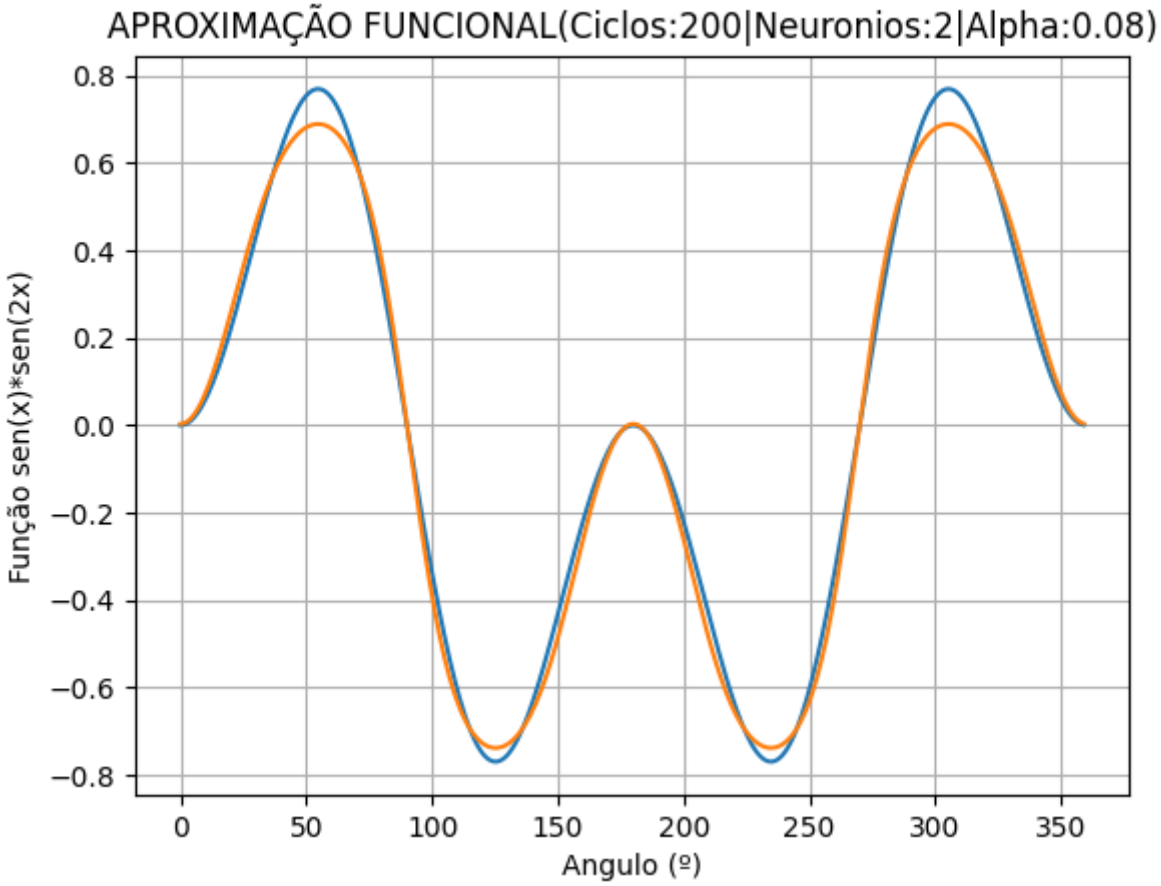
```
# e adicionado o peso do bias
output_layer = [[random.random() for __ in range(num_hidden +1)]
                for __ in range(output_size)]

# a rede inicializa com pesos sinápticos randômicos
network = [hidden_layer, output_layer]
# print(network)
for __ in range(200): # número de ciclos de treinamento
    for x in range(360):
        inputs = seno(x)
        targets = seno(x)
        for input_vector, target_vector in zip(inputs, targets):
            backpropagate(network, input_vector, target_vector)

#TERINAMENTO DA REDE NEURAL
# formação do gráfico
fig, ax = plt.subplots()
ax.set(xlabel='Angulo (º)', ylabel='Função sen(x)*sen(2x)',
       title='APROXIMAÇÃO FUNCIONAL(Ciclos:200|Neuronios:2|Alpha:0.08)')
ax.grid()
t = np.arange(0, 360,1)

#teste da rede através de predict()
saida = []
for x in range(360):
    inputs = seno(x)
    targets = seno(x)
    for input_vector, target_vector in zip(inputs, targets):
        sinal_saida = predict(input_vector)
        saida.extend(sinal_saida)

entrada = []
for x in range(360):
    entrada += seno(x) # criando o arranjo da função de entrada para o gráfico
ax.plot(t, entrada)
ax.plot(t, saida)

print ("comando entrada", hidden_layer)
print ("comando saída", output_layer)

plt.show()
fig.savefig('./img/aprox_func_1a.png')
```

CONSOLE tp4_1a

```
PS D:\workspace\IA> &
C:/Users/bapti/AppData/Local/Programs/Python/Python310/python.exe
d:/workspace/IA/tp4/tp4_1a.py
comando entrada [[1.0408393883858735, 0.4525690083234288], [0.7524377929122791,
0.1291888136629787]]
```

```
comando saída [[3.047923627308438, 2.435423673493166, -0.8289472821150731]]
PS D:\workspace\IA>
```

Gráfico tp4_1a



Código tp4_1b

```python
from __future__ import division

import math  # pg 56 - Summerfield
import random

import matplotlib.pyplot as plt
import numpy as np
from linear_algebra import dot


def sigmoid(t):
    return ((2 / (1 + math.exp(-t)))-1)

def neuron_output(weights, inputs):
    return sigmoid(dot(weights, inputs))
```

```
def feed_forward(neural_network, input_vector):
    outputs = []
    for layer in neural_network:
        input_with_bias = input_vector + [1]              # adiciona bias à entrada
        output = [neuron_output(neuron, input_with_bias) # calcula a saída do
neurônio
                  for neuron in layer]                    # para cada camada
        #print(output)
        outputs.append(output)

        # a saída de uma camada de neurônio é a entrada da próxima camada
        input_vector = output

    return outputs

alpha = 0.08

def backpropagate(network, input_vector, target):
    # feed_forward calcula a saída dos neurônios usando sigmóide
    hidden_outputs, outputs = feed_forward(network, input_vector)
    # 0.5 *alpha* (1 + output) * (1 - output) cálculo de derivada de sigmóide
    output_deltas = [0.5 * (1 + output) * (1 - output) * (output - target[i]) *
alpha
                     for i, output in enumerate(outputs)]

    # ajuste dos pesos sinápticos para camadas de saída (network[-1])
    for i, output_neuron in enumerate(network[-1]):
        for j, hidden_output in enumerate(hidden_outputs + [1]):
            output_neuron[j] -= output_deltas[i] * hidden_output

    # 0.5 *alpha* (1 +output)*(1-output) cálculo de derivada da sigmóide
    # retro-programação do erro para camadas intermediárias
    hidden_deltas = [ 0.5 * alpha * (1 + hidden_output) * (1 - hidden_output) *
                      dot(output_deltas, [n[i] for n in network[-1]])
                     for i, hidden_output in enumerate(hidden_outputs)]

    # ajuste dos pesos sinápticos para camadas intermediárias (network[0])
    for i, hidden_neuron in enumerate(network[0]):
        for j, input in enumerate(input_vector + [1]):
            hidden_neuron[j] -= hidden_deltas[i] * input

def seno(x): # função a ser aproximada pela rede neural
       seno = [(math.sin(2*math.pi/180*x)*math.sin(math.pi/180*x))]
       # seno é uma lista
       # [(0.8+(math.sin(math.pi/180*x)*math.sin(2*math.pi/180*x)))*0.5]
       return [seno]

def predict(inputs):
     return feed_forward(network, inputs)[-1]

inputs = []
targets = []
for x in range(360):
```

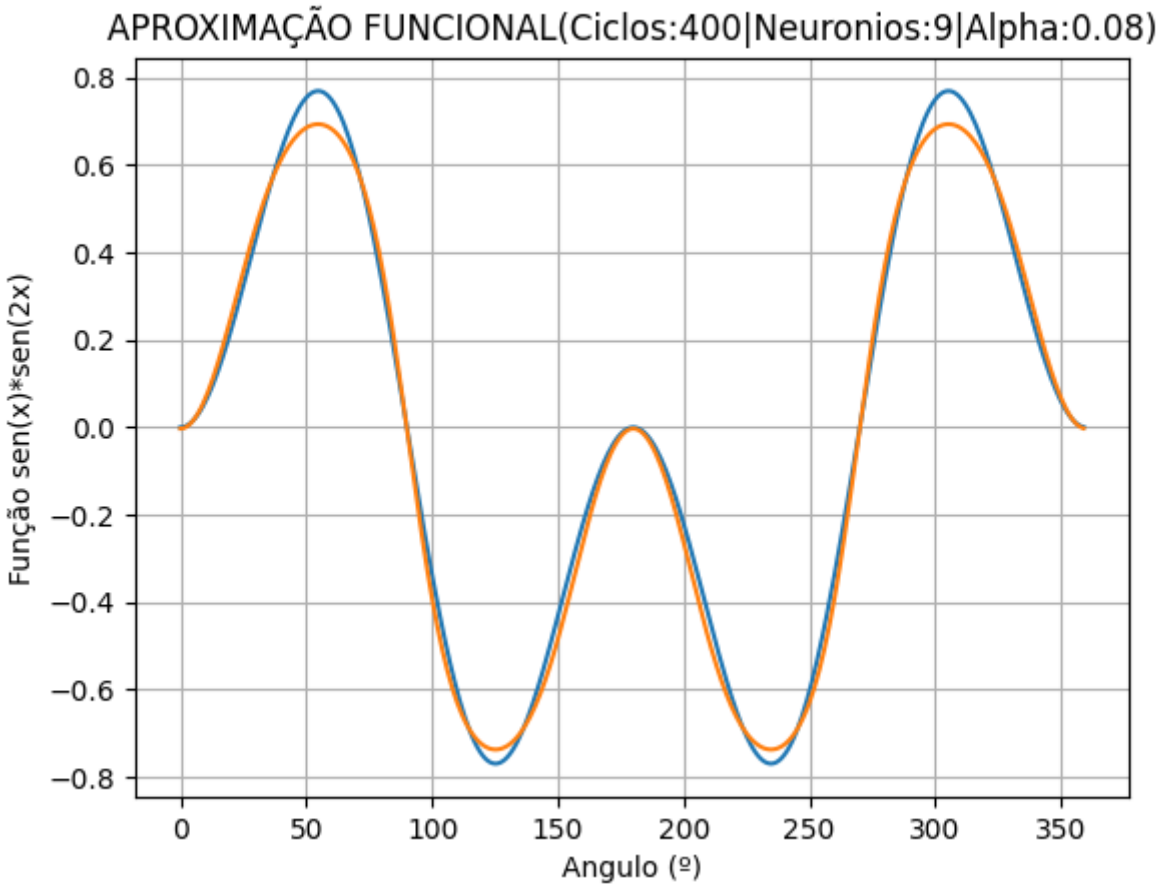
```
    seno_a = seno(x)

# TREINAMENTO DA REDE NEURAL
random.seed(0) # valores iniciais de pesos sinápticos
input_size = 1 # dimensão do vetor de entrada
num_hidden = 9 # número de neurônios na camada intermediária
output_size = 1 # dimensão das camadas de saída = 1 neurônio

"""
inserindo manualmente os vetores relativos à camada intermediária e a saída da
Rede Neural
hidden_layer = [[-0.085, -0.09], [-0.033, -0.08], [-0.074, -0.063], [-0.075,
-0.065], [-0.088, -0.076], [-0.077, -0.072]]
output_layer = [[0.082, -0.09, 0.064, -0.08, 0.084, -0.075, 0.099]] """

# cada neurônio da camada intermediária tem um peso sináptico associado à entrada
# e adicionado o peso do bias
hidden_layer = [[random.random() for __ in range(input_size + 1)]
                for __ in range(num_hidden)]

# neurônio de saída tem um peso sináptico associado a cada neurôio da camada
intermediária
# e adicionado o peso do bias
output_layer = [[random.random() for __ in range(num_hidden +1)]
                for __ in range(output_size)]

# a rede inicializa com pesos sinápticos randômicos
network = [hidden_layer, output_layer]
# print(network)
for __ in range(400): # número de ciclos de treinamento
    for x in range(360):
        inputs = seno(x)
        targets = seno(x)
        for input_vector, target_vector in zip(inputs, targets):
            backpropagate(network, input_vector, target_vector)

#TERINAMENTO DA REDE NEURAL
# formação do gráfico
fig, ax = plt.subplots()
ax.set(xlabel='Angulo (º)', ylabel='Função sen(x)*sen(2x)',
       title='APROXIMAÇÃO FUNCIONAL(Ciclos:400|Neuronios:9|Alpha:0.08)')
ax.grid()
t = np.arange(0, 360,1)

#teste da rede através de predict()
saida = []
for x in range(360):
    inputs = seno(x)
    targets = seno(x)
    for input_vector, target_vector in zip(inputs, targets):
        sinal_saida = predict(input_vector)
        saida.extend(sinal_saida)

entrada = []
```

```
for x in range(360):
    entrada += seno(x) # criando o arranjo da função de entrada para o gráfico
ax.plot(t, entrada)
ax.plot(t, saida)

print ("comando entrada", hidden_layer)
print ("comando saída", output_layer)

plt.show()
plt.show()
fig.savefig('./tp4/img/aprox_func_1b.png')
```

CONSOLE 1b

```
PS D:\workspace\IA> &
C:/Users/bapti/AppData/Local/Programs/Python/Python310/python.exe
d:/workspace/IA/tp4/tp4_1b.py
comando entrada [[0.8241839825342936, 0.6779552357977129], [0.5129276665813426,
0.16428832275156816], [0.5277483686843542, 0.3770584926586358],
[0.8069063410355166, 0.17127803784138376], [0.5175852235255326,
0.5088302360372114], [0.8849956505370526, 0.39288563704486107],
[0.2845392063746386, 0.744754161372949], [0.6302343916224499,
0.22411623889245197], [0.8963678131159108, 0.969199189263095]]
comando saída [[0.8301778027924777, 1.3765665195222165, 0.62403027929275,
1.459084871764951, 1.0027979800177576, 1.1131499967279392, 0.15500174280690945,
0.7079369881450845, 0.0989772806473804, -1.2753362039463165]]
PS D:\workspace\IA>
```

Gráfico tp4_1b



Codigo tp4_1c

```
from __future__ import division

import math  # pg 56 - Summerfield
import random

import matplotlib.pyplot as plt
import numpy as np
from linear_algebra import dot


def sigmoid(t):
    return ((2 / (1 + math.exp(-t)))-1)

def neuron_output(weights, inputs):
    return sigmoid(dot(weights, inputs))

def feed_forward(neural_network, input_vector):
    outputs = []
    for layer in neural_network:
        input_with_bias = input_vector + [1]             # adiciona bias à entrada
        output = [neuron_output(neuron, input_with_bias) # calcula a saída do
neurônio
```

```
                    for neuron in layer]                    # para cada camada
            #print(output)
            outputs.append(output)

            # a saída de uma camada de neurônio é a entrada da próxima camada
            input_vector = output

        return outputs

    alpha = 0.15

    def backpropagate(network, input_vector, target):
        # feed_forward calcula a saída dos neurônios usando sigmóide
        hidden_outputs, outputs = feed_forward(network, input_vector)
        # 0.5 *alpha* (1 + output) * (1 - output) cálculo de derivada de sigmóide
        output_deltas = [0.5 * (1 + output) * (1 - output) * (output - target[i]) *
    alpha
                         for i, output in enumerate(outputs)]

        # ajuste dos pesos sinápticos para camadas de saída (network[-1])
        for i, output_neuron in enumerate(network[-1]):
            for j, hidden_output in enumerate(hidden_outputs + [1]):
                output_neuron[j] -= output_deltas[i] * hidden_output

        # 0.5 *alpha* (1 +output)*(1-output) cálculo de derivada da sigmóide
        # retro-programação do erro para camadas intermediárias
        hidden_deltas = [ 0.5 * alpha * (1 + hidden_output) * (1 - hidden_output) *
                          dot(output_deltas, [n[i] for n in network[-1]])
                         for i, hidden_output in enumerate(hidden_outputs)]

        # ajuste dos pesos sinápticos para camadas intermediárias (network[0])
        for i, hidden_neuron in enumerate(network[0]):
            for j, input in enumerate(input_vector + [1]):
                hidden_neuron[j] -= hidden_deltas[i] * input

    def seno(x): # função a ser aproximada pela rede neural
            seno = [(math.sin(2*math.pi/180*x)*math.sin(math.pi/180*x))]
            # seno é uma lista
            # [(0.8+(math.sin(math.pi/180*x)*math.sin(2*math.pi/180*x)))*0.5]
            return [seno]

    def predict(inputs):
         return feed_forward(network, inputs)[-1]

    inputs = []
    targets = []
    for x in range(360):
        seno_a = seno(x)

    # TREINAMENTO DA REDE NEURAL
    random.seed(0) # valores iniciais de pesos sinápticos
    input_size = 1 # dimensão do vetor de entrada
    num_hidden = 9 # número de neurônios na camada intermediária
    output_size = 1 # dimensão das camadas de saída = 1 neurônio
```

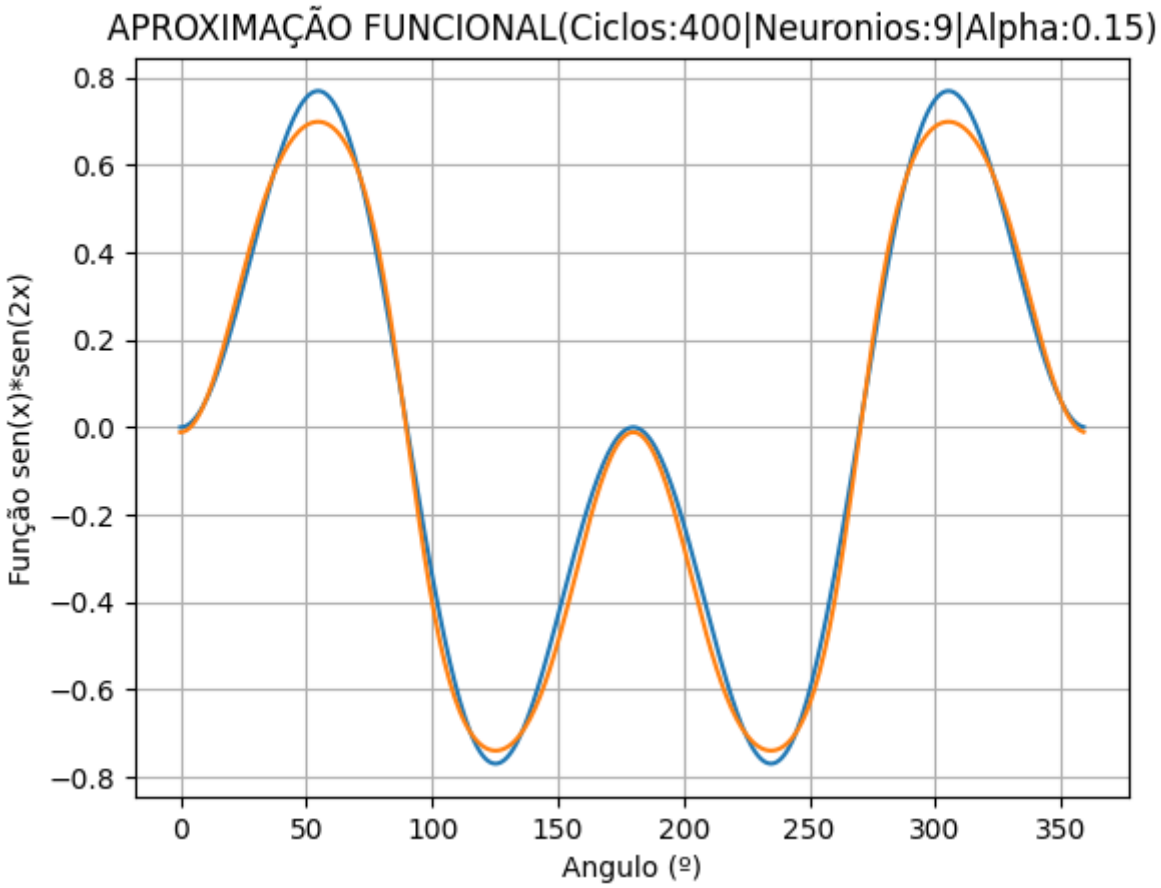
```
"""
inserindo manualmente os vetores relativos à camada intermediária e a saída da
Rede Neural
hidden_layer = [[-0.085, -0.09], [-0.033, -0.08], [-0.074, -0.063], [-0.075,
-0.065], [-0.088, -0.076], [-0.077, -0.072]]
output_layer = [[0.082, -0.09, 0.064, -0.08, 0.084, -0.075, 0.099]] """

# cada neurônio da camada intermediária tem um peso sináptico associado à entrada
# e adicionado o peso do bias
hidden_layer = [[random.random() for __ in range(input_size + 1)]
                for __ in range(num_hidden)]

#print(hidden_layer)
# neurônio de saída tem um peso sináptico associado a cada neurôio da camada
intermediária
# e adicionado o peso do bias
output_layer = [[random.random() for __ in range(num_hidden +1)]
                for __ in range(output_size)]

# a rede inicializa com pesos sinápticos randômicos
network = [hidden_layer, output_layer]
# print(network)
for __ in range(400): # número de ciclos de treinamento
    for x in range(360):
        inputs = seno(x)
        targets = seno(x)
        for input_vector, target_vector in zip(inputs, targets):
            backpropagate(network, input_vector, target_vector)

#TERINAMENTO DA REDE NEURAL
# formação do gráfico
fig, ax = plt.subplots()
ax.set(xlabel='Angulo (º)', ylabel='Função sen(x)*sen(2x)',
       title='APROXIMAÇÃO FUNCIONAL(Ciclos:400|Neuronios:9|Alpha:0.15)')
ax.grid()
t = np.arange(0, 360,1)

#teste da rede através de predict()
saida = []
for x in range(360):
    inputs = seno(x)
    targets = seno(x)
    for input_vector, target_vector in zip(inputs, targets):
        sinal_saida = predict(input_vector)
        saida.extend(sinal_saida)

entrada = []
for x in range(360):
    entrada += seno(x) # criando o arranjo da função de entrada para o gráfico
ax.plot(t, entrada)
ax.plot(t, saida)

print ("comando entrada", hidden_layer)
```

```
print ("comando saída", output_layer)

plt.show()
plt.show()
fig.savefig('aprox_func_1c.png')
```

Conosole tp4_1c

```
PS D:\workspace\IA> &
C:/Users/bapti/AppData/Local/Programs/Python/Python310/python.exe
d:/workspace/IA/tp4/tp4_1c.py
comando entrada [[0.7640532342489892, 0.6257546628541283], [0.5652244989241987,
0.07647916420982044], [0.529554514750517, 0.3587375268660623],
[0.7692030864376894, 0.05595708403030226], [0.5285373445540341,
0.4487212654289582], [0.8088960130981689, 0.3116653157817971],
[0.2879585415117993, 0.7385724817768965], [0.6218990409431315,
0.20831658619338947], [0.8745707370522826, 0.9702492052875391]]
comando saída [[0.8411590370150338, 1.4563718006553463, 0.6530917109899849,
1.4653041412788508, 1.0723761422861993, 1.0918939443648357, 0.23057069278727388,
0.7000174601662565, 0.08995053109461301, -1.0900295144856231]]
PS D:\workspace\IA>
```

Gráfico tp4_1c


APROXIMAÇÃO FUNCIONAL(Ciclos:400|Neuronios:9|Alpha:0.15)
Função sen(x)*sen(2x)
Angulo (º)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
50
100
150
200
250
300
350