Аппроксимация – Монте-Карло

# Approx - Monte-Karlo

import numpy as np

import matplotlib.pyplot as plt

from math import sin

from math import pi

from math import sqrt

DIM\_N = 1

NEURON\_N1 = 20

NEURON\_N2 = 20

NN = 10

data = []

for i in range(NN+1):

data.append([2.0\*pi/NN \* i])

data = np.array(data)

data = np.random.permutation(data)

targ = np.sin(data)

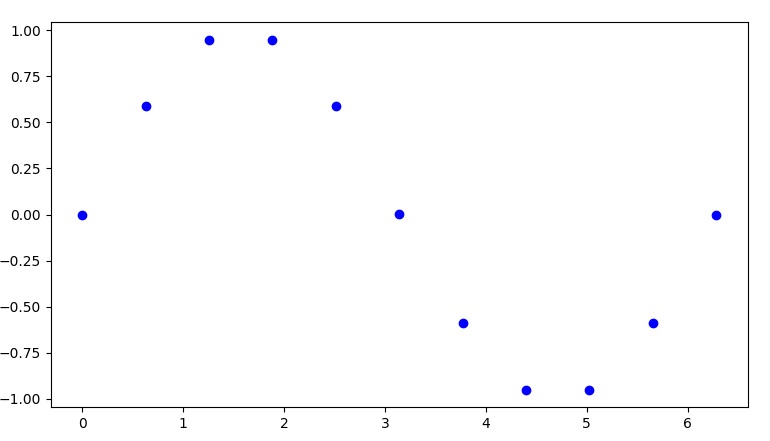
#targ = np.add(targ, 1)

plt.figure(figsize=(9,5))

for i in range(NN+1):

plt.scatter(data[i], targ[i], c='blue', marker='o')

plt.show()



class LAYER:

def \_\_init\_\_(self, n\_neurons, n\_input):

self.n\_neurons = n\_neurons

self.n\_input = n\_input

# self.w = np.random.rand(self.n\_neurons,self.n\_input) - 0.5

self.w = np.random.normal( 0.0, self.n\_input\*\*-0.5, (self.n\_neurons,self.n\_input) )

self.dw = np.zeros([self.n\_neurons,self.n\_input], dtype=float)

self.b = np.zeros(self.n\_neurons, dtype=float)

self.db = np.zeros(self.n\_neurons, dtype=float)

def forward(self, inputs):

res = np.dot(self.w, inputs) + self.b

return res

def forward\_add(self, inputs):

res = np.dot(self.w+self.dw, inputs) + self.b+self.db

return res

Layer = [LAYER(NEURON\_N1, DIM\_N), LAYER(NEURON\_N2, NEURON\_N1), LAYER(1, NEURON\_N2)]

def Sigma(x):

return 1.0 / (1.0 + np.exp(-1.0\*x))

def Forward (point):

r = Layer[0].forward(point)

r = Sigma(r)

r = Layer[1].forward(r)

r = Sigma(r)

r = Layer[2].forward(r)

return r

def Forward\_add (point):

r = Layer[0].forward\_add(point)

r = Sigma(r)

r = Layer[1].forward\_add(r)

r = Sigma(r)

r = Layer[2].forward\_add(r)

return r

a0 = 1.0e+99

pred = []

for i in range(10000):

Layer[0].dw = np.random.normal( 0.0, Layer[0].n\_input\*\*-0.5, (Layer[0].n\_neurons,Layer[0].n\_input) )

Layer[0].db = np.random.normal( 0.0, Layer[0].n\_input\*\*-0.5, (Layer[0].n\_neurons) )

Layer[1].dw = np.random.normal( 0.0, Layer[1].n\_input\*\*-0.5, (Layer[1].n\_neurons,Layer[1].n\_input) )

Layer[1].db = np.random.normal( 0.0, Layer[1].n\_input\*\*-0.5, (Layer[1].n\_neurons) )

Layer[2].dw = np.random.normal( 0.0, Layer[2].n\_input\*\*-0.5, (Layer[2].n\_neurons,Layer[2].n\_input) )

Layer[2].db = np.random.normal( 0.0, Layer[2].n\_input\*\*-0.5, (Layer[2].n\_neurons) )

a = 0.0

for n in range(data.shape[0]):

a += ( targ[n] - Forward\_add(data[n])[0] )\*\*2

if a<a0 :

a0 = a

Layer[0].w = np.add(Layer[0].w, Layer[0].dw)

Layer[0].b = np.add(Layer[0].b, Layer[0].db)

Layer[1].w = np.add(Layer[1].w, Layer[1].dw)

Layer[1].b = np.add(Layer[1].b, Layer[1].db)

Layer[2].w = np.add(Layer[2].w, Layer[2].dw)

Layer[2].b = np.add(Layer[2].b, Layer[2].db)

print ('err= ', sqrt(a0)/NN )

res = []

for n in range(data.shape[0]):

res.append(Forward(data[n]))

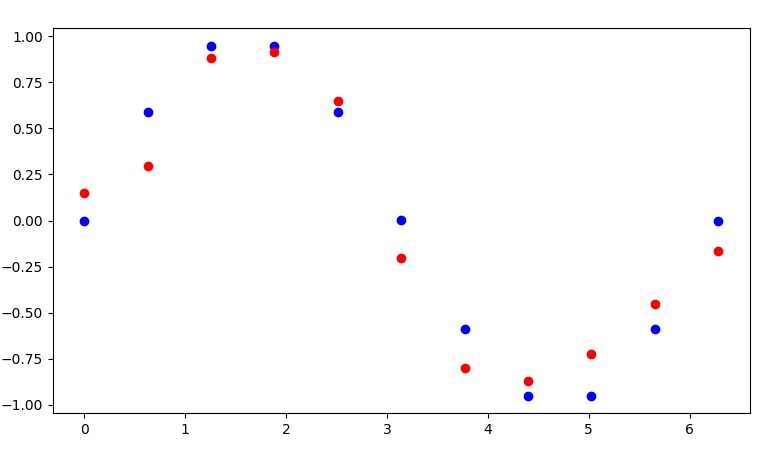
plt.figure(figsize=(9,5))

for i in range(NN+1):

plt.scatter(data[i], targ[i], c='blue', marker='o')

plt.scatter(data[i], res[i], c='red', marker='o')

plt.show()



Аппроксимация – библиотека Keras

# Approx - Monte-Karlo

import numpy as np

import matplotlib.pyplot as plt

from math import sin

from math import pi

from math import sqrt

DIM\_N = 1

NEURON\_N1 = 20

NEURON\_N2 = 20

NN = 10

data = []

for i in range(NN+1):

data.append([2.0\*pi/NN \* i])

data = np.array(data)

data = np.random.permutation(data)

targ = np.sin(data)

plt.figure(figsize=(9,5))

for i in range(NN+1):

plt.scatter(data[i], targ[i], c='blue', marker='o')

plt.show()

#Neural network module

import keras

from keras.models import Sequential

from keras.layers import Dense

model=Sequential()

model.add(Dense(NEURON\_N1, input\_dim=DIM\_N, activation='sigmoid'))

model.add(Dense(NEURON\_N2, activation='sigmoid'))

model.add(Dense(1,activation='linear'))

model.compile(loss='mse', optimizer='adam')

callback = keras.callbacks.EarlyStopping(monitor="loss", mode="min", verbose=1, patience=300, min\_delta=0.001)

model.fit(data, targ, batch\_size=int(data.shape[0]/2), epochs=5000, verbose=1, callbacks=[callback])

res = model.predict(data)

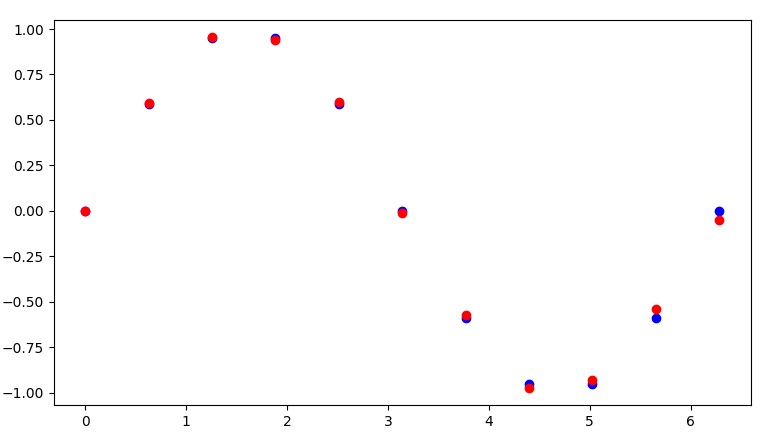
plt.figure(figsize=(9,5))

for i in range(NN+1):

plt.scatter(data[i], targ[i], c='blue', marker='o')

plt.scatter(data[i], res[i], c='red', marker='o')

plt.show()



Keras — это набор функций API для построения и использования нейросетей, написанный на Python и работающий поверх платформы машинного обучения TensorFlow.

Подключение библиотек:

import tensorflow

import keras

Основная компонента – уровень нейронной сети:

from keras import layers

layer1 = layers.Dense(2, activation="sigmoid", input\_dim=2)

layer2 = layers.Dense(3, activation="sigmoid")

layer3 = layers.Dense(1)

Создаём уровни с указанными именами. Dense создаёт уровни с указанном числом нейронов – первый аргумент функции Dense. Другие аргументы: активационная функция, число входов нейрона, … .

Прямое распространение данных через уровни сети:

y = layer3(layer2(layer1(x)))

x – матрица, число столбцов должно соответствовать числу входов соответствующего уровня сети (аргумент input\_dim).

Модель сети. Последовательная сеть:

model = keras.Sequential([

layers.Dense(2, activation="sigmoid", input\_dim=2),

layers.Dense(3, activation="sigmoid"),

layers.Dense(1)

])

Прямое распространение данных через сеть:

y = model(x)

Функция

model.summary()

показывает структуру построенной сети:

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

layer1 (Dense) (3, 2) 8

layer2 (Dense) (3, 3) 9

layer3 (Dense) (3, 4) 16

=================================================================

Total params: 33

Trainable params: 33

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Обучение сети:

model.compile(optimizer='RMSprop', loss='mean\_squared\_error')

model.fit(x, y, epochs=5000, batch\_size=100, verbose=1)

(большинство параметров по умолчанию)