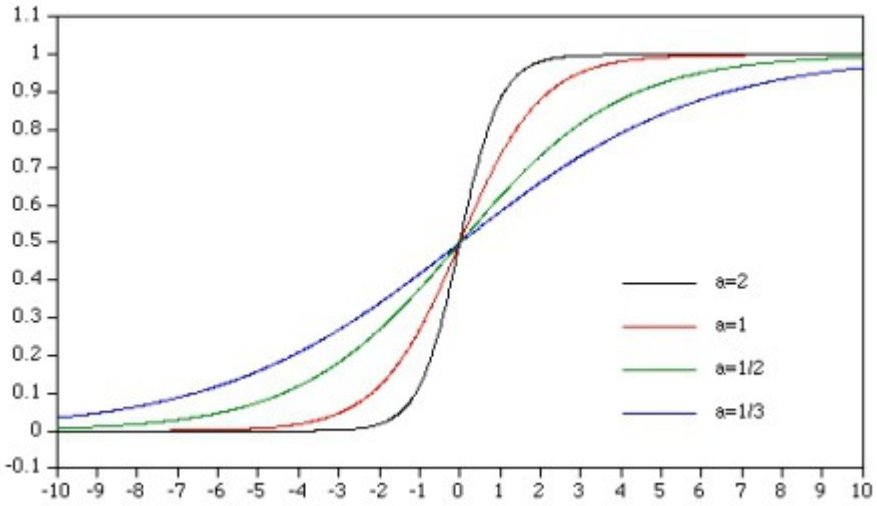
Sprawozdanie – ćwiczenie nr 3
Sieć wielowarstwowa.

Piotr Tutak

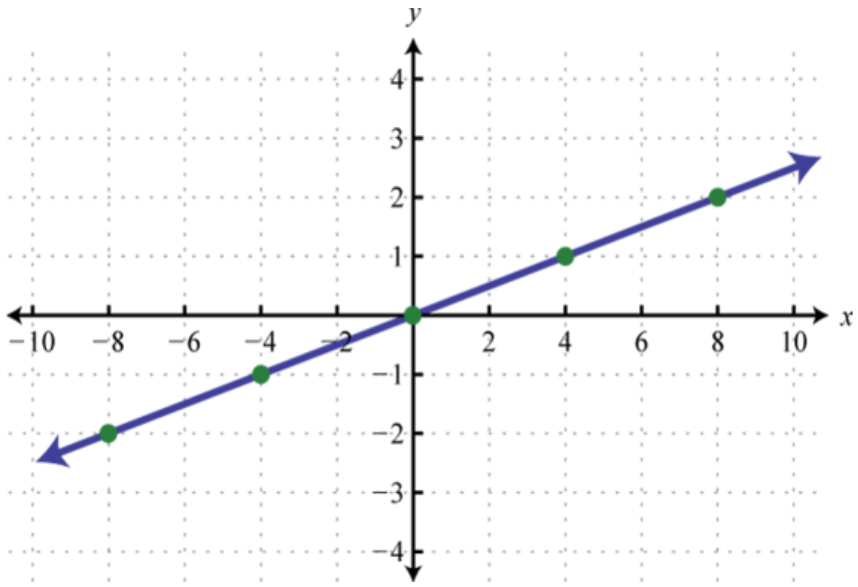
1. Syntetyczny opis budowy sieci i algorytmu uczenia

Wykorzystana sieć neuronowa została oparta o moduł Keras zaimplementowany w Pythonie.
Jest to standardowa sieć neuronowa wielowarstwowa sekwencyjna.

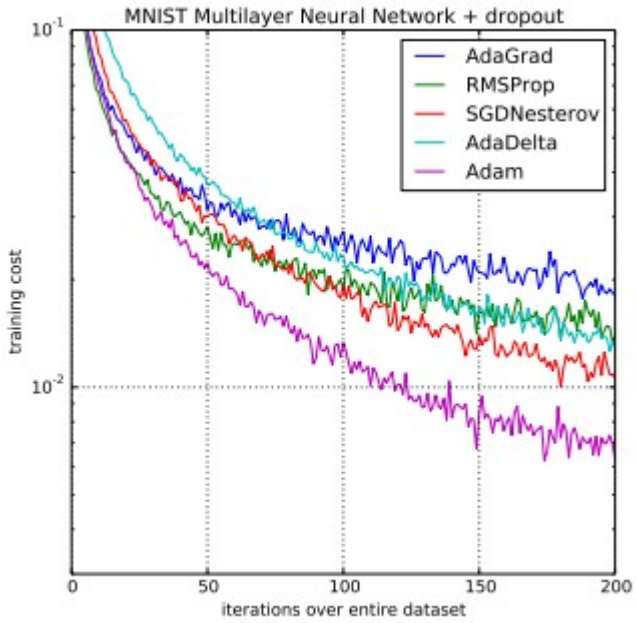
Sieć zawierała od 2 – 5 warstw, po od 1 – 30 neuronów w każdej warstwie. Użyte funkcje aktywacji w każdej warstwie to funkcja sigmoidalna dla warstw początkowych:



Oraz funkcja liniowa dla warstwy ostatniej:



Użyty algorytm uczenia korzystał z metody optymalizacji ADAM. Jest to bardzo wydajna i stosunkowo nowa metoda optymalizacji uczenia, konkurująca z innymi bardzo popularnymi jak SGD, AdaGrad itp.
Porównanie ADAM i innych algorytmów uczących:



Więcej o ADAM można poczytać tutaj:

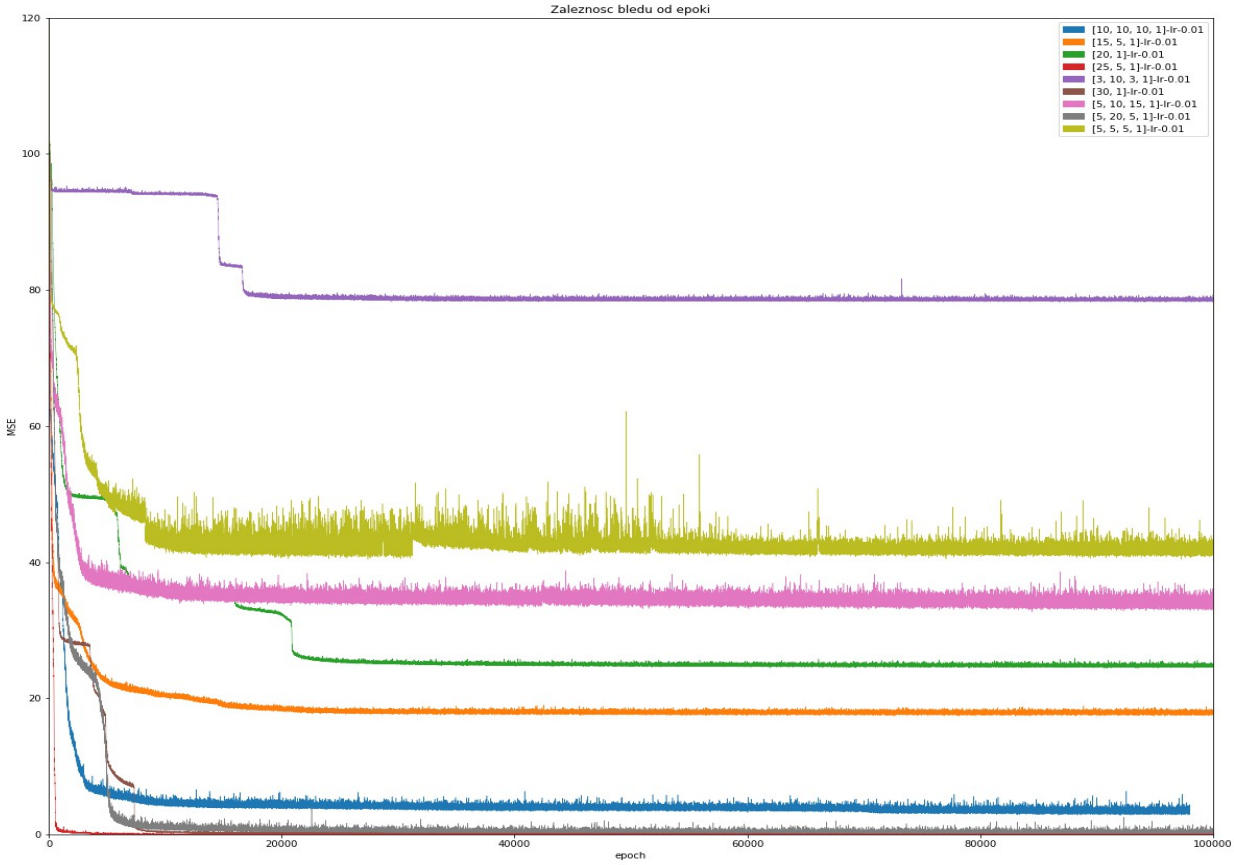
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
https://arxiv.org/pdf/1412.6980.pdf

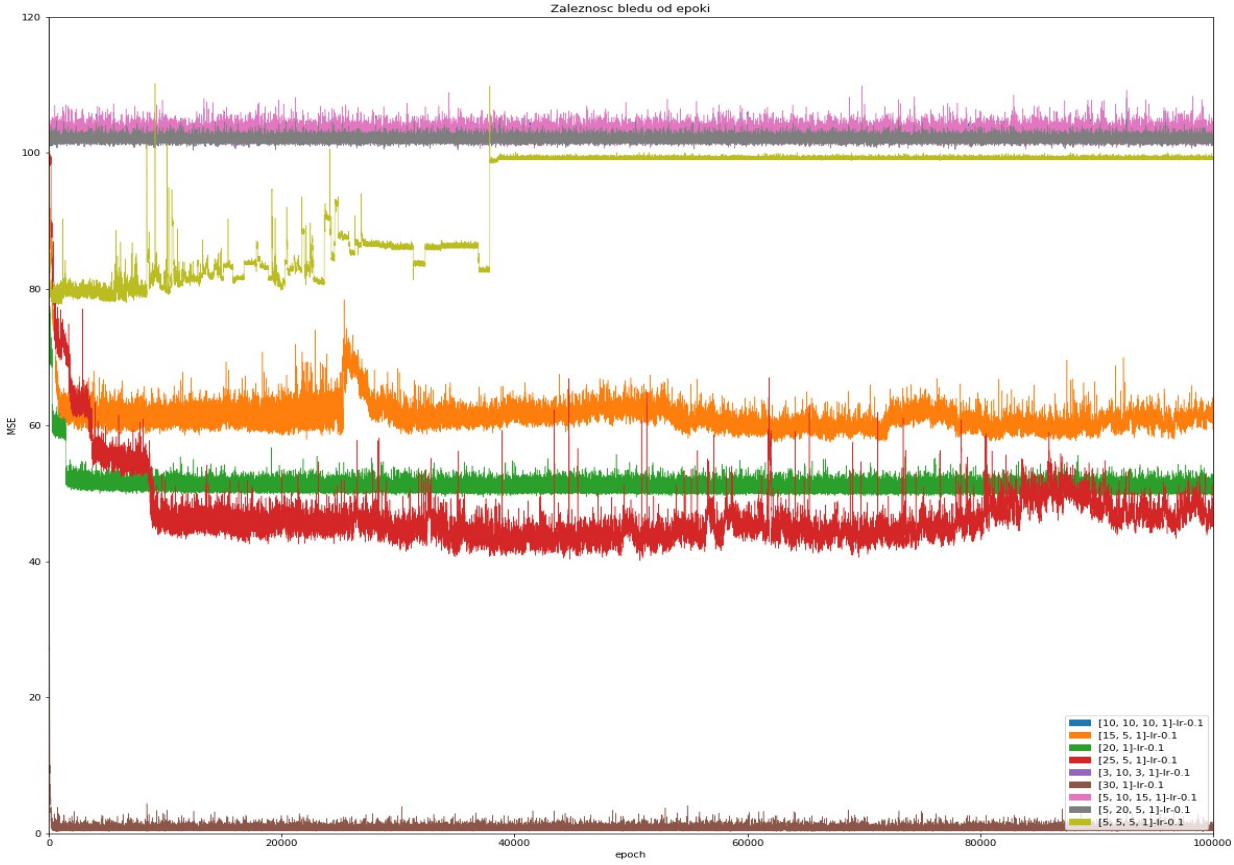
Syntetycznie można napisać, że jest to algorytm znajdujący optymalnie szybko minima badanej funkcji.

Badany błąd, który podlegał minimalizacji to MSE, czyli „mean square error” - błąd średniokwadratowy.

Sieć była uczona ze współczynnikiem batch=20, czyli wagi były aktualizowane po 20 przypadkach uczenia.

2. Wyniki uczenia:





3. Analiza:

Jak widać z otrzymanych wyników sieci które się najlepiej uczą to te o konfiguracjach:
- [5,20,5,1], lr=0.01
- [10,10,10,1], lr=0.01
- [25,5,1], lr=0.01
- [30,1], lr=0.01
- [30,1], lr=0.1

W większości przypadków ostateczna wartość błędu uczenia była uzyskiwana po ok. 40000 iteracji.
W znakomitej większości przypadków wartość błędu uczenia była niższa po zastosowaniu procesu uczenia, niż przed zastosowaniem tego algorytmu.
W jednym przypadku wartość błędu sieci po zastosowaniu procesu uczenia była wyższa, niż przed zastosowaniem tego procesu (sieć [20,1], lr=0.1).
Zwiększenie współczynnika uczenia z 0.01 do 0.1 we wszystkich przypadkach doprowadziło do pogorszenia wyników.
Niektóre sieci przy współczynniku uczenia=0.1 nie nauczyły się praktycznie nic.

Minimalny uzyskany błąd MSE~= 0.03 – uzyskany dla sieci [25,5,1], lr=0.01

W wielu przypadkach uczenia sieci widać skokowy postęp uczenia. Wiedząc, że funkcja rastrigin jest funkcją o wielu minimach i maksimach, można przypuszczać, że są to punkty w których funkcja optymalizacji przeszła kolejne „wzniesienie" i mogła odnaleźć kolejny punkt lepiej pasujący i oddający ostateczną wartość funkcji.
Średni czas uczenia jednej sieci neuronowej wynosił ok. 1 godz.

4. Wnioski:

- Współczynnik uczenia dla tak badanej i estymowanej funkcji powinien być raczej niższy niż wyższy
- Najlepsze wyniki uzyskuje się dla sieci o dużej ilości neuronów wejściowych, albo posiadających średnią ilość neuronów ale w większej ilości warstw
- W przypadku tak dobranej badanej funkcji uczenie można z powodzeniem ograniczyć do 40000 epok

Listing kodu:

```
# -*- coding: utf-8 -*-

import os
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

import numpy as np
import sys

"""
Rastrigin
"""

def rastrigin(x,y):
    return 20+x**2-10*np.cos(2*np.pi*x)+y**2-10*np.cos(2*np.pi*y)

np.random.seed(7)

"""
Generowanie danych uczących i testujących
"""

with open('training_data.csv','w') as f:
    for i in range(1000):
        x=np.random.sample()*4-2
        y=np.random.sample()*4-2
        print('{0},{1},{2}'.format(x,y,rastrigin(x,y)),file=f)

with open('test_data.csv','w') as f:
    for i in range(1000):
        x=np.random.sample()*4-2
        y=np.random.sample()*4-2
        print('{0},{1},{2}'.format(x,y,rastrigin(x,y)),file=f)


lr=0.1
decay=0.0
layers=[30,1]


"""
Przekierowanie wyjscia
"""
STDOUT=sys.stdout
try:
    f=open('results'+str(layers)+'-lr-'+str(lr)+'-decay-'+str(decay)+'.txt','w');
```

```
    sys.stdout=f

    np.random.seed(7)

    dataSet=np.loadtxt('training_data.csv',delimiter=',')
    inputData = dataSet[:,0:2]
    expected = dataSet[:,2]

    testDataSet=np.loadtxt('test_data.csv',delimiter=',')
    inputTestData = testDataSet[:,0:2]
    expectedTestData = testDataSet[:,2]

    """
    Dodawanie warstw do modelu
    """
    model=Sequential()
    for i in range(len(layers)):
        if i==0:
            model.add(Dense(layers[i], input_dim=2,activation='sigmoid'))
        elif i==len(layers)-1:
            model.add(Dense(layers[i],activation='linear'))
        else:
            model.add(Dense(layers[i],activation='sigmoid'))

    """
    Optymalizator
    """
    adam = optimizers.Adam(lr=lr, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=decay)

    model.compile(loss='mean_squared_error', optimizer=adam, metrics=['accuracy'])

    """
    Trenowanie modelu
    """
    model.fit(inputData,expected,epochs=100000,batch_size=20)

    scores=model.evaluate(inputTestData,expectedTestData)
    print("scores: ",scores)
    print("metrics_names:",model.metrics_names)

    print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

    print(model.summary())
    """
    zapis modelu
    """
    model.save('model_sieci-'+str(layers)+'-lr-'+str(lr)+'-decay-'+str(decay)+'.h5')
finally:
    sys.stdout=STDOUT
    f.close()
```