Wydział Informatyki

Metody sztucznej inteligencji

Laboratorium 02 IUz-22 Urbaniak

Sprawozdanie

Autor: Sergiusz Urbaniak
Grupa: IUz-22
Data: 5 stycznia 2010

Spis treści

1 Wprowadzenie

Następujące zadania nie zostały rozwiązane za pomocą oprogramowania Matlab lecz dzięki alternatywnych i wolnodostępnych rozwiązań. Jest to spowodowane faktem że autor niestety nie posiada stałego dostępu do internetu i jest skazany do odrabiania sprawozdań "w drodze"(w pociągach). Poza tym jest zwolennikiem Ópen Source".

Istnieje ciekawy projekt o nazwie FANN (Fast Artificial Neural Network Library)[1]. Biblioteka ta udostępnia algorytmy związane z sieciami neuronowymi pod różnymi językami. Dla następującego sprawozdania został wybrany język Python. Jest to język przede wszystkim obiektowy (w odróżnieniu do języka w Matlabie/Octavie) i posiada znakomite biblioteki naukowe (scipy, numpy, matplotlib).

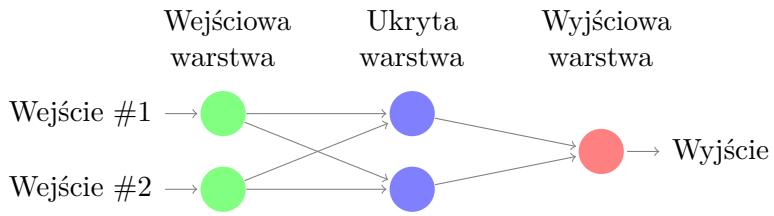
2 Bramka logiczna XOR

Bramka XOR posiada tablice prawdy pokazaną w tablicy 4.

Wejście	Wyjście
0 0	0
0 1	1
1 0	1
1 1	0

Tablica 1: Tablica prawdy bramki XOR

Zamodelowac można ją za pomocą dwóch neuronów w warstwie ukrytej. Sieć neuronowa jest pokazana w rysunku 1.



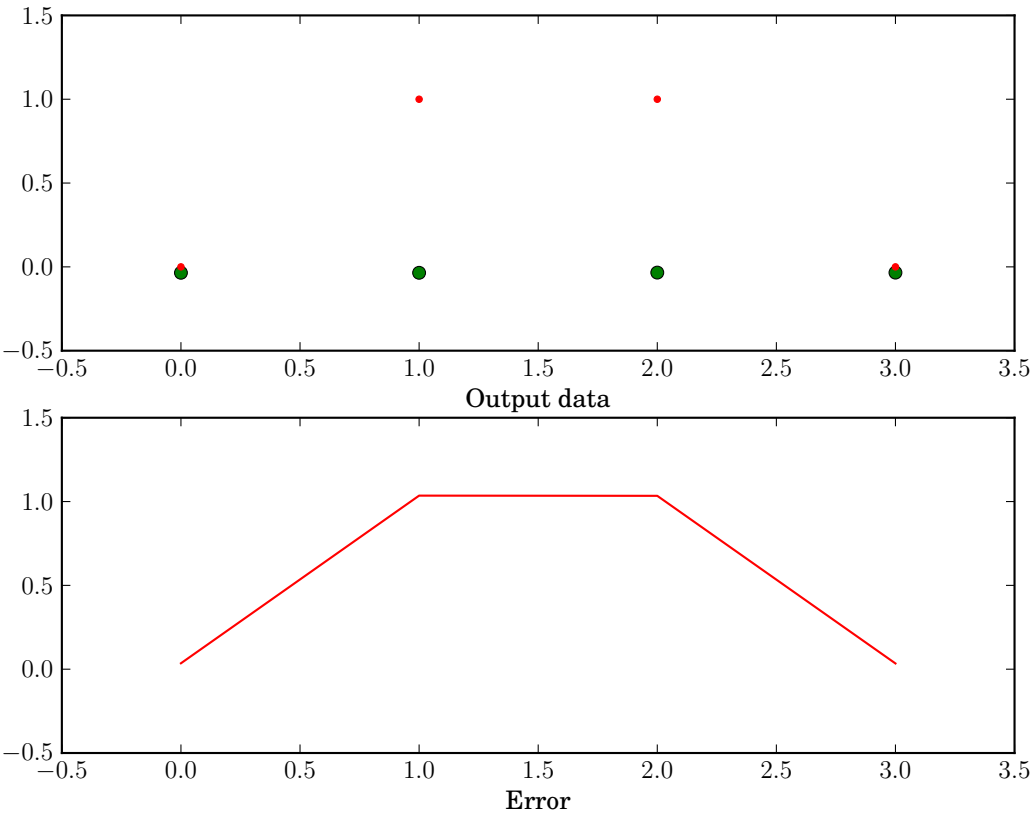
Rysunek 1: Siec neuronowa wielowarstwowa

Nie uczona sieć jest widoczna w rysunku 2. W diagramie widać na czerwonych punktach właściwe dane wyjściowe. Na zielonych punktach widoczna jest wyjście z sieci. Można zauważyć że istnieje błąd widoczny w dolnym diagramie.

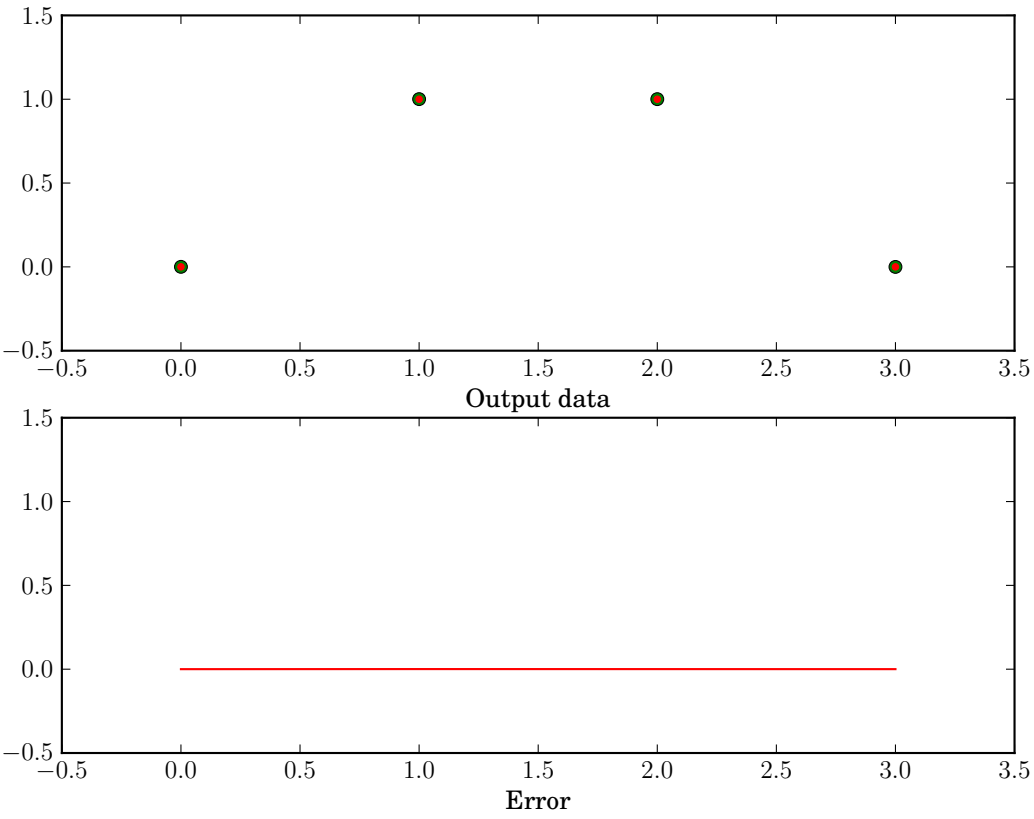
Wynik po uczeniu widać na rysunku 3. Widać że siec z tylko dwoma neuronami była w stanie nauczyć się bramki XOR. Wyjście sieci (zielone punkty) zbliżyły się do właściwych wartości i błąd wynosi 0.

Kod uczenia jak i wykresów jest widoczny w listingu 1.

[1] http://leenissen.dk/fann



Rysunek 2: Wyniki przed uczeniu



Rysunek 3: Wyniki po uczeniu

```
import pyfann.libfann as fann
from matplotlib import pyplot, rc
import numpy

# init pyplot
rc('text', usetex=True)
rc('font', family='serif')

# create an empty simple network
def create_net(in_layers, hidden_layers, out_layers):
  net = fann.neural_net()
  net.create_sparse_array(1, (in_layers, hidden_layers, out_layers))
  net.set_activation_function_hidden(fann.SIGMOID_SYMMETRIC)
  net.set_activation_function_output(fann.LINEAR)

  return net

# run the network on some data
def run(net, data):
  out = numpy.zeros((len(data), 1))
  for i, y in enumerate(data):
    out[i] = net.run(y)

  return out

# plot error and data
def plot_output(y_learned, y, rowcol):
  pyplot.subplot(rowcol)
  pyplot.plot(y_learned, 'go')
  pyplot.plot(y, 'r.')
  pyplot.axis([-0.5, 3.5, -0.5, 1.5])
  pyplot.xlabel("Output data")

  error = abs(y_learned - y)

  pyplot.subplot(rowcol + 1)
  pyplot.plot(error, 'r')
  pyplot.axis([-0.5, 3.5, -0.5, 1.5])
  pyplot.xlabel("Error")

# create xor training data
data = fann.training_data()
inp = [[1,1],[0,1],[1,0],[0,0]]
out = [[0],[1],[1],[0]]
data.set_train_data(inp,out)

# create network
net = create_net(2, 2, 1)

# run on untrained network
out_learned = run(net, inp)

# save and plot
net.save("xor_untrained.net")
plot_output(out_learned, out, 211)
pyplot.savefig("xor_untrained.pdf", format = "pdf")

# train the network
```

```
net.train_on_data(data, 10000, 1000, 0.0000001)

# run on trained network
out_learned = run(net, inp)

# save and plot
net.save("xor_trained.net")
pyplot.clf()
plot_output(out_learned, out, 211)
pyplot.savefig("xor_trained.pdf", format = "pdf")
```

Listing 1: Kod uczenia bramki XOR

3 Uczenie plików

Uczenie podanymi plikami okazało się już trudniejszym zadaniem. Okazało się że siec zbudowana z FANN zbyt nie dokładnie się uczy danych. Wybrano jednak te pliki gdzie uzyskano najmniejsze błędy. Zostały wybrane następujące zestawy:

- parity_i.txt, parity_o.txt
- sincos_i, sincos_o.txt
- glass.txt

Następujące rozdziały są organizowane w trzy części:

1. Opis optymalnych danych dla danego pliku
2. Tabela badań z różnymi funkcjami aktywacji. Skonfigurowano bibliotekę FANN w taki sposób aby przestała uczyć siec jak wystąpi jeden z następujących warunków:
   - Aktualna iteracja (epoka) wynosi nie więcej niż 1000
   - Błąd jest poniżej 0.0001
3. Wykresy badan:
   - Dane wyjściowe z pliku (Output data)
   - Dane wyjściowe po uczeniu (Network output on input data)
   - Błąd: Rożnica powyższych danych

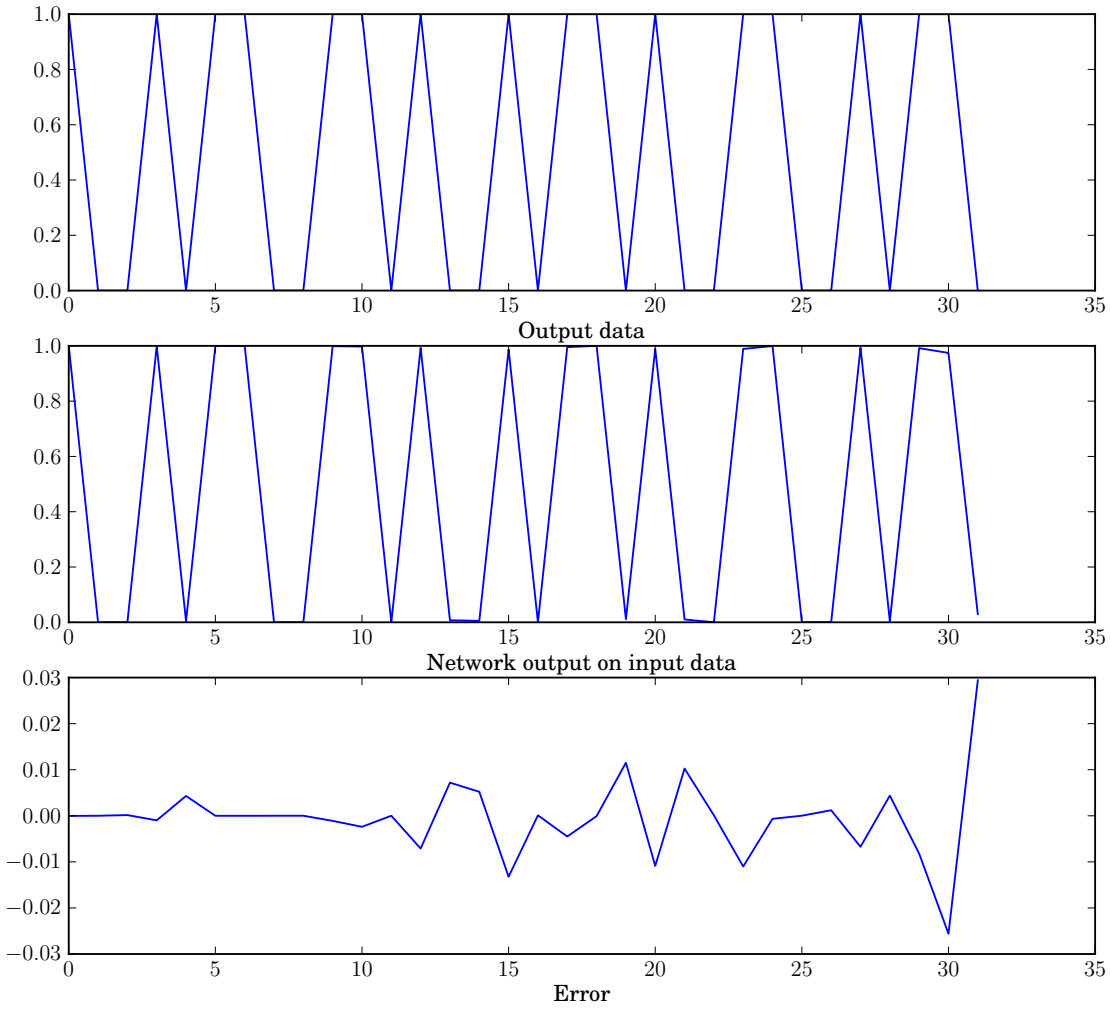
3.1 Plik `parity`

Dane okazały najmniejszy błąd za pomocą następujących funkcji aktywacji:

- Funkcja ukryta: sigmoidalna
- Funkcja wyjściowa: sigmoidalna
- Ilość neuronów ukrytej warstwy: 15
- Iteracje: 491
- Błąd: Poniżej 0.0001

Funkcja ukryta	Funkcja wyjściowa	Ilość ukryta	Iteracje	Błąd
sigmoid	sigmoid	5	1000	0.03
sigmoid	sigmoid	15	491	
sigmoid	sigmoid	40	647	
sigmoid	tangential	5	1000	0.062
sigmoid	tangential	15	1000	0.007
sigmoid	tangential	40	773	0.0001
sigmoid	linear	5	1000	0.25
sigmoid	linear	15	1000	0.024
sigmoid	linear	40	586	
linear	sigmoid	5	1000	0.25
linear	sigmoid	15	1000	0.25
linear	sigmoid	40	1000	0.25
linear	tangential	5	1000	0.063
linear	tangential	15	1000	0.063
linear	tangential	40	1000	0.063
linear	linear	5	1000	0.25
linear	linear	15	1000	0.25
linear	linear	40	1000	0.25
tangential	sigmoid	5	1000	0.123
tangential	sigmoid	15	506	
tangential	sigmoid	40	477	
tangential	tangential	5	1000	0.008
tangential	tangential	15	1000	0.0005
tangential	tangential	40	1000	0.008
tangential	linear	5	1000	0.08
tangential	linear	15	1000	0.004
tangential	linear	40	1000	0.002

Tablica 2: Wyniki badan nad plikami `parity`



Rysunek 4: Wyniki po uczeniu plików `parity`

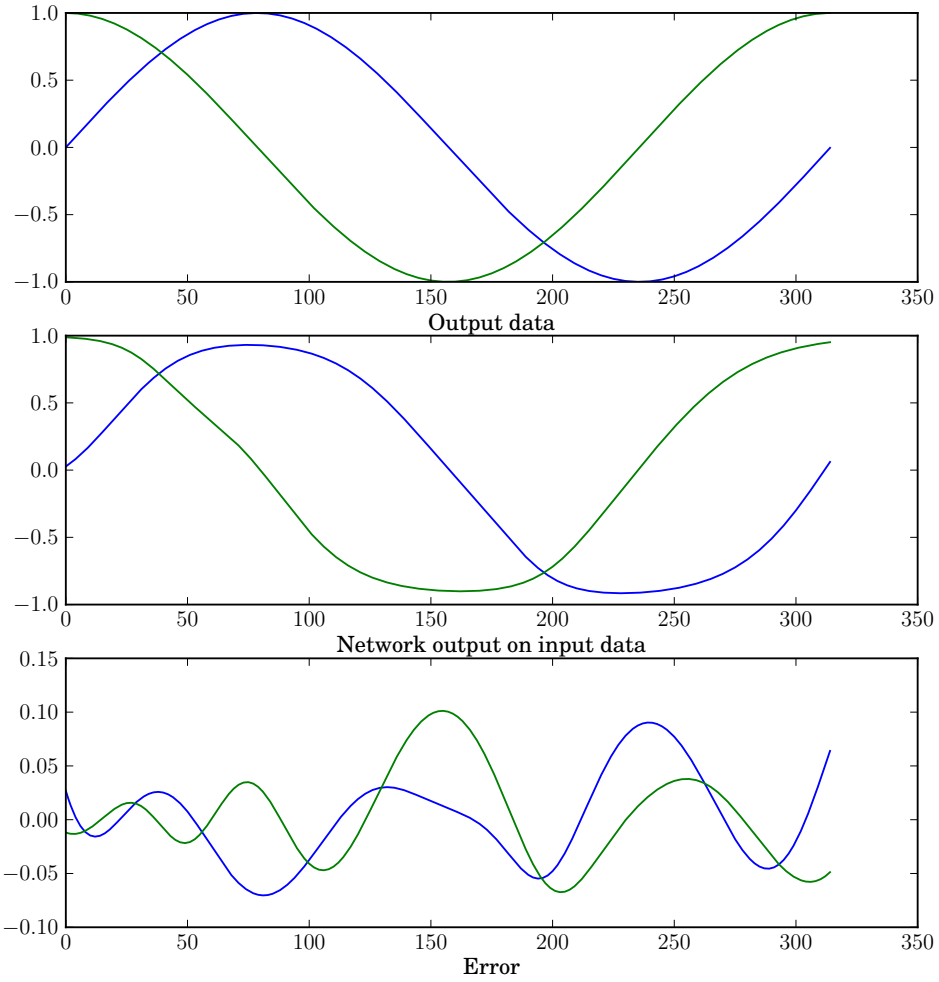
3.2 Plik `sincos`

Dane okazały najmniejszy błąd za pomocą następujących funkcji aktywacji:

- Funkcja ukryta: sigmoidalna
- Funkcja wyjściowa: tangencjalna
- Ilość neuronów ukrytej warstwy: 40
- Iteracje: 1000
- Błąd: Poniżej 0.0004

Funkcja ukryta	Funkcja wyjściowa	Ilość ukryta	Iteracje	Błąd
sigmoid	sigmoid	5	1000	0.5
sigmoid	sigmoid	15	1000	0.5
sigmoid	sigmoid	40	1000	0.47
sigmoid	tangential	5	1000	0.0017
sigmoid	tangential	15	1000	0.0005
sigmoid	tangential	40	1000	0.0004
sigmoid	linear	5	1000	0.35
sigmoid	linear	15	1000	0.35
sigmoid	linear	40	1000	0.35
linear	sigmoid	5	1000	0.5
linear	sigmoid	15	1000	0.5
linear	sigmoid	40	1000	0.5
linear	tangential	5	1000	0.08
linear	tangential	15	1000	0.08
linear	tangential	40	1000	0.08
linear	linear	5	1000	0.35
linear	linear	15	1000	0.35
linear	linear	40	1000	0.35
tangential	sigmoid	5	1000	0.5
tangential	sigmoid	15	1000	0.5
tangential	sigmoid	40	1000	0.5
tangential	tangential	5	1000	0.0016
tangential	tangential	15	1000	0.0009
tangential	tangential	40	1000	0.0009
tangential	linear	5	1000	0.07
tangential	linear	15	1000	0.002
tangential	linear	40	1000	0.0006

Tablica 3: Tablica prawdy bramki XOR



Rysunek 5: Wyniki po uczeniu

3.3 Plik `glass.txt`

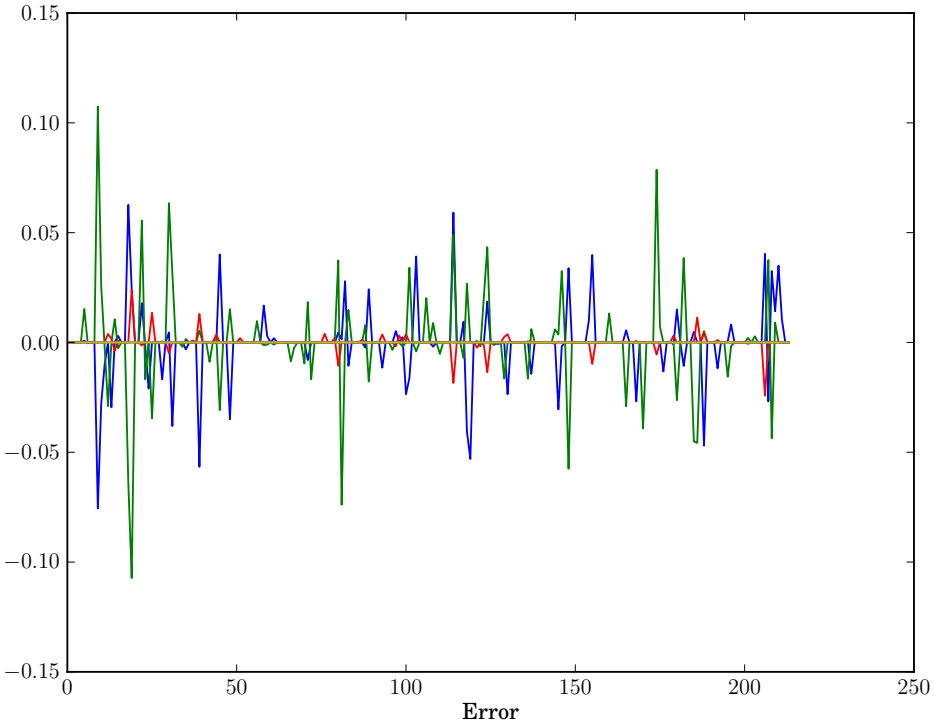
Dane okazały najmniejszy błąd za pomocą następujących funkcji aktywacji:

- Funkcja ukryta: sigmoidalna
- Funkcja wyjściowa: sigmoidalna
- Ilość neuronów ukrytej warstwy: 40
- Iteracje: 10000
- Błąd: Poniżej 0.0004

Dla tego zestawu danych trzeba było dopisać nowa funkcje wczytania danych ponieważ wyjście i wejście znajdują się w tym samym pliku. Okazało się ze trzeba tez było drastycznie powiększyć maksymalna ilość iteracji aby osiągnąć sensownie niskie błędy sieci.

Funkcja ukryta	Funkcja wyjściowa	Ilość ukryta	Iteracje	Błąd
sigmoid	sigmoid	5	10000	0.05
sigmoid	sigmoid	15	10000	0.009
sigmoid	sigmoid	40	9229	
sigmoid	tangential	5	10000	0.013
sigmoid	tangential	15	10000	0.007
sigmoid	tangential	40	10000	0.004
sigmoid	linear	5	10000	0.06
sigmoid	linear	15	10000	0.03
sigmoid	linear	40	10000	0.017
linear	sigmoid	5	10000	0.069
linear	sigmoid	15	10000	0.069
linear	sigmoid	40	10000	0.069
linear	tangential	5	10000	0.02
linear	tangential	15	10000	0.02
linear	tangential	40	10000	0.02
linear	linear	5	10000	0.09
linear	linear	15	10000	0.08
linear	linear	40	10000	0.08
tangential	sigmoid	5	10000	0.04
tangential	sigmoid	15	10000	0.01
tangential	sigmoid	40	10000	0.007
tangential	tangential	5	10000	0.014
tangential	tangential	15	10000	0.01
tangential	tangential	40	10000	0.007
tangential	linear	5	10000	0.07
tangential	linear	15	10000	0.037
tangential	linear	40	10000	0.025

Tablica 4: Tablica prawdy bramki XOR



Rysunek 6: Wyniki po uczeniu

```
import pyfann.libfann as fann
import numpy
from matplotlib import pyplot, rc

# init pyplot
rc('text', usetex=True)
rc('font', family='serif')

# create an empty simple network
def create_net(layers, funcs):
  net = fann.neural_net()
  net.create_sparse_array(1, layers)
  net.set_activation_function_hidden(funcs[0])
  net.set_activation_function_output(funcs[1])

  return net

# convinience method for 1-dimensional arrays
# fann cannot handle those and leaves with a segfault :(
def check_matrix(matrix):
  if matrix.ndim == 1:
    new = numpy.empty((matrix.shape[0], 1))

    for i, x in enumerate(matrix):
      new[(i, 0)] = x

    return new

  return matrix

def load_data(filename, in_outs):
  a = numpy.loadtxt(filename)
  inp = numpy.compress(numpy.ones(in_outs[0]), a, axis=1)
  inp = check_matrix(inp)
  out = numpy.compress(numpy.concatenate([numpy.zeros(in_outs[0]), numpy.
      ones(in_outs[1])]), a, axis=1)
  out = check_matrix(out)

  data = fann.training_data()
  data.set_train_data(inp,out)

  return data

# load data and create training set
def load_data_prefix(prefix):
  inp = numpy.loadtxt(prefix + "_i.txt")
  inp = check_matrix(inp)
  out = numpy.loadtxt(prefix + "_o.txt")
  out = check_matrix(out)

  data = fann.training_data()
  data.set_train_data(inp,out)

  return data

# learning routine
def learn(data, layers, funcs):
  net = create_net((len(data.get_input()[0]), layers, len(data.get_output
```

```
      ()[0])), funcs)
  net.train_on_data(data, 10000, 1000, 0.0001)

  return net

# run the net with some data
def run(net, data):
  out = numpy.zeros((data.length_train_data(), data.num_output_train_data
      ()))
  for i, y in enumerate(data.get_input()):
    out[i] = net.run(y)

  return out

def save_plot(filename):
  pyplot.savefig(filename, format = "pdf")

def run_suite(filename, num_hidden, func, in_outs):
  data = load_data(filename, in_outs)
  net = learn(data, num_hidden, func)
  out = run(net, data)

  #pyplot.subplot(311)
  #pyplot.plot(data.get_output())
  #pyplot.xlabel("Output data")

  #pyplot.subplot(312)
  #pyplot.plot(out)
  #pyplot.xlabel("Network output on input data")

  error = out - data.get_output()
  pyplot.subplot(111)
  pyplot.plot(error)
  pyplot.xlabel("Error")
  pyplot.show()
```

Listing 2: Kod uczenia plików

4 Podsumowanie

Instalacja, nauczenie się funkcji dostępnych przez FANN, numpy i matplotlib pod Python zajęło dużo czasu ale zdaniem autora było warto. Nawet udało się w ramach tego sprawozdania znaleźć błąd biblioteki FANN, który występuje tylko przy wykorzystaniu języka Python. Autor biblioteki FANN został skontaktowany.

Pozostają jednak pytania dlaczego FANN stosunkowo źle się uczył w ramach sprawozdania zadanych danych.