Artificial Neural Networks

Prof. Dr. Sen Cheng

Nov 18, 2019

Problem Set 7: Artifical Neural Networks

Tutors: Vinita Samarasinghe (samarasinghe@ini.rub.de), Thomas Walther (thomas.walther@rub.de)

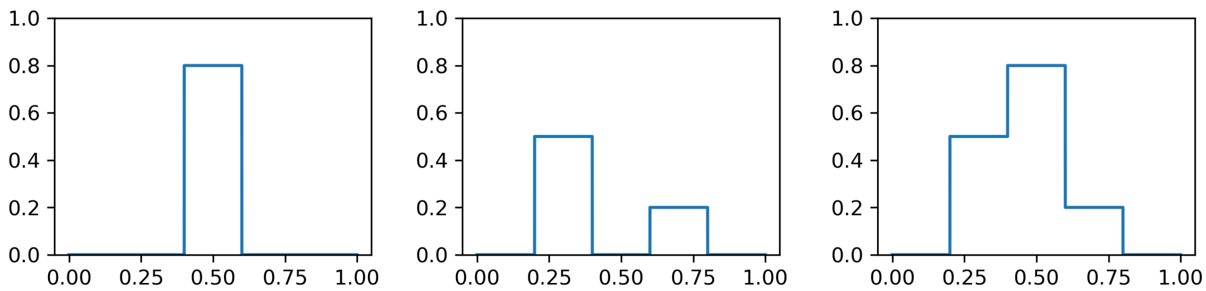
1. The universal approximation theorem states that any function can be approximated by a neural network with one hidden layer.

$$f(x) = \sum_{i=1}^{N} \nu_i \phi\left(w_i^T x + b_i\right) \tag{1}$$

   Implement this network in a Python function using only elementary programming operations. For the activation function $\phi(\cdot)$, use the sigmoid function $\sigma(\cdot)$. For the latter, use the `expit` function from `scipy.special`.

2. Using the previously implemented function $f(x)$, manually set the parameters ν_i, w_i, b_i, N in your program to replicate the output of $f(x)$ shown in the Figures 1a, 1b, and 1c.

   Useful applet that visualizes the impact of the network parameters: `http://neuralnetworksanddeeplearning.com/chap4.html`



(a) (b) (c)

Figure 1: Sample outputs of $f(x)$.

3. Given $g(x) = \sin(2\pi x)$ on the domain $[0;1]$.

   (a) Approximate $g(x)$ with $f(x)$ using N=10, by computing ν_i, w_i, b_i in a program. Plot the functions $g(x)$ and $f(x)$ in a single figure.

   (b) Compute the residual error $|f(x) - g(x)|$ using elementary programming operations. Repeat the approximation for several larger values of N. Plot the residual error against N.

Solutions

1. The definition of $\phi(\cdot) = \sigma(x)$ can be done as follows:

```
from scipy.special import expit as sigmoid
```

With that, 'sigmoid(x)' represents the function $\sigma(x)$ that can be used as activation function $\phi(\cdot)$.
Note: do not forget to import other necessary libraries:

```
import numpy as np
import matplotlib.pyplot as graph
```

In your program define $f(x)$, taking into account the necessary adaptation of w_i and b_i:

```
def f(x,w,b,v,N):
    output=0.0
    for i in range(N):
        output+=v[i]*sigmoid(w[i]*(x+b[i]))
    return output
```

2. In your program, define the x-domain:

```
x=np.linspace(0.0,1.0,1e4)
```

Generate output shown in Fig. 1a:

```
# number of units
N=2
# vector of weights (hidden)
w=np.array([1e4,1e4])
# vector of biases (hidden)
b=np.array([−0.4,−0.6])
# vector of weights (output)
v=np.array([0.8,−0.8])

# plot the function
graph.plot(x,f(x,w,b,v,N))
# display the plot
graph.show()
```

Generate output shown in Fig. 1b:

```
N=4
w=np.array([1e4,1e4,1e4,1e4])
b=np.array([−0.2,−0.4,−0.6,−0.8])
v=np.array([0.5,−0.5,0.2,−0.2])

graph.plot(x,f(x,w,b,v,N))
graph.show()
```

Generate the output shown in Fig. 1c:

```
N=6
w=np.array([1e4,1e4,1e4,1e4,1e4,1e4])
b=np.array([−0.2,−0.4,−0.4,−0.6,−0.6,−0.8])
v=np.array([0.5,−0.5,0.8,−0.8,0.2,−0.2])
```

```
graph.plot(x,f(x,w,b,v,N))
graph.show()
```

3. In your program, define $g(x) = \sin(2\pi x)$:

```
def g(x):
    return np.sin(x*2.0*np.pi)
```

(a) Approximate $g(x)$ with $f(x)$, using $N = 10$, by choosing ν_i, w_i, b_i automatically. For that, first program the function:

```
def setParameters(N):
    w=np.ones ((N),dtype=float)*1e4
    b=np.zeros((N),dtype=float)
    v=np.zeros((N),dtype=float)

    intervalDomain=np.linspace(0.0,1.0,N/2+1)

    for index in range(int(N/2)):
        leftBorder=intervalDomain[index]
        rightBorder=intervalDomain[index+1]
        b[index*2]=-leftBorder
        b[index*2+1]=-rightBorder
        mid=(rightBorder-leftBorder)/2.0+leftBorder
        v[index*2]=g(mid)
        v[index*2+1]=-g(mid)

    return w,b,v
```

Then use the programmed function:

```
N=10
w,b,v=setParameters(N)

graph.plot(x,g(x))
graph.plot(x,f(x,w,b,v,N))
graph.show()
```

(b) Compute the residual error using elementary programming operations.

```
residualError=np.sum(np.abs(f(x,w,b,v,N)-g(x)))
```

Using $N = 10$, the residual error is ≈ 1963.96. To plot the residual error against N:

```
errors=np.zeros((10,1),dtype=float)
N=np.linspace(10,100,10)
for nIndex in range(N.shape[0]):
    n=int(N[nIndex])
    w,b,v=setParameters(n)
    residualError=np.sum(np.abs(f(x,w,b,v,n)-g(x)))
    errors[nIndex]=residualError

graph.plot(N,errors)
graph.show()
```

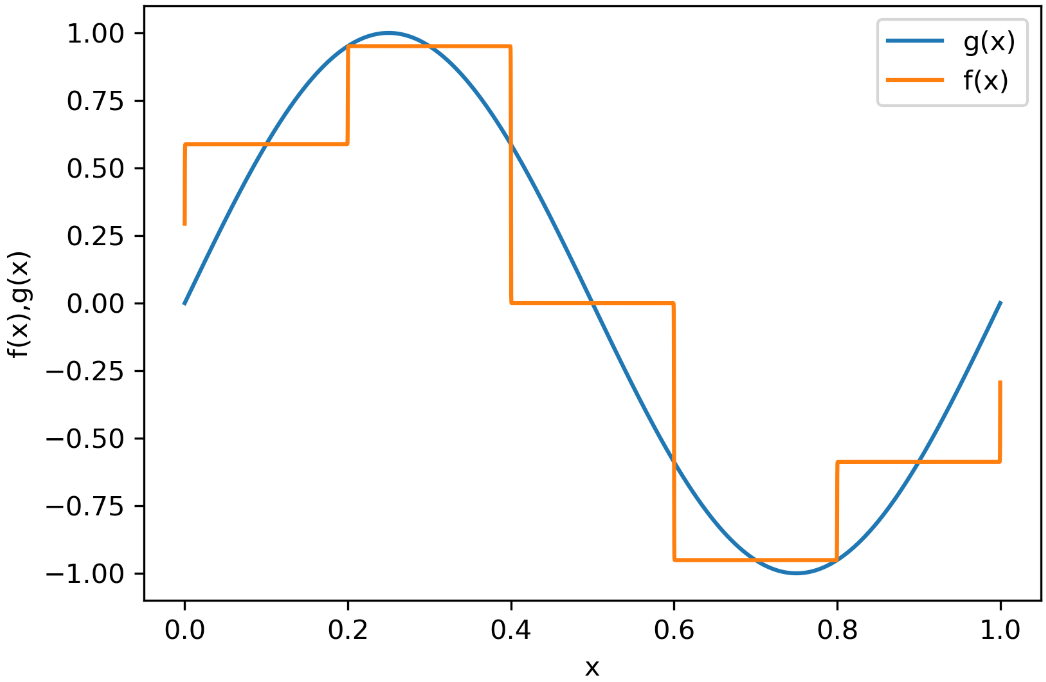


Figure 2: Solution output for task 3, subtask (a).

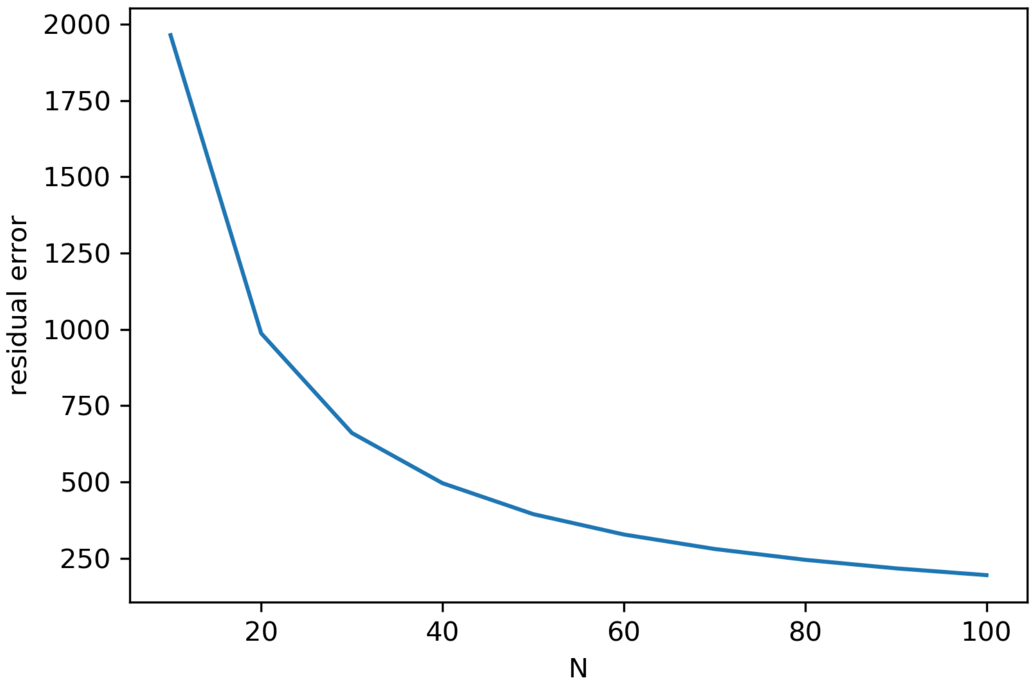


Figure 3: Solution output for task 3, subtask (b).