Artificial Neural Networks

Prof. Dr. Sen Cheng

Oct 14, 2019

Problem Set 1: Introduction

Tutors: Filippos Panagiotou (filippos.panagiotou@ini.rub.de), Mohammad M. Nejad (m.mohagheghi@ini.rub.de)

1. **Function**

   In the following you find two relations, f and g, that map x onto y. Determine which one is a function and why.

   (a) $f : x \to y$ such that $y = x^2$

   (b) $g : x \to y$ such that $y^2 = x$

   Also, plot y vs. x for both relationships in Python and try to visually determine which one is a function. Note that the range of x in your plot is limited and therefore you may not see the full characteristics of the functions, so choose the range carefully!

2. **Types of functions**

   Plot the following functions for $x \in [-10, 10]$. Be sure to plot them in a way that you can change the parameters easily:

   (a) $f(x) = 2x + 2$

   (b) $f(x) = 5x^2 + x$

   (c) $f(x) = 11x^3 + 2x^2 + 2x + 3$

   (d) $f(x) = e^x$

   Try the following adjustments:

   i. Take the linear function from above in its generalized form $f(x) = ax + b$. It has 2 parameters. Adjust each of them and plot the result. Observe how do they change the behavior of the function.

   ii. Take the quadratic function from above in its generalized form $f(x) = ax^2 + bx + c$. Can you tune the parameters in such a way that the minimum of the function lies at -2? What happens if you multiply the quadratic term with a large constant? Extra: can you explain why the linear factor does not contribute to the result?

Solutions

1. **Definition of functions**

   f is a function because every x is mapped onto exactly one y (left panel, Fig. 1). g, however, is not a function. If we rewrite $y^2 = x$ as $y = \pm\sqrt{x}$, it is clear that all $x > 0$ are mapped onto two y (right panel, Fig. 1).

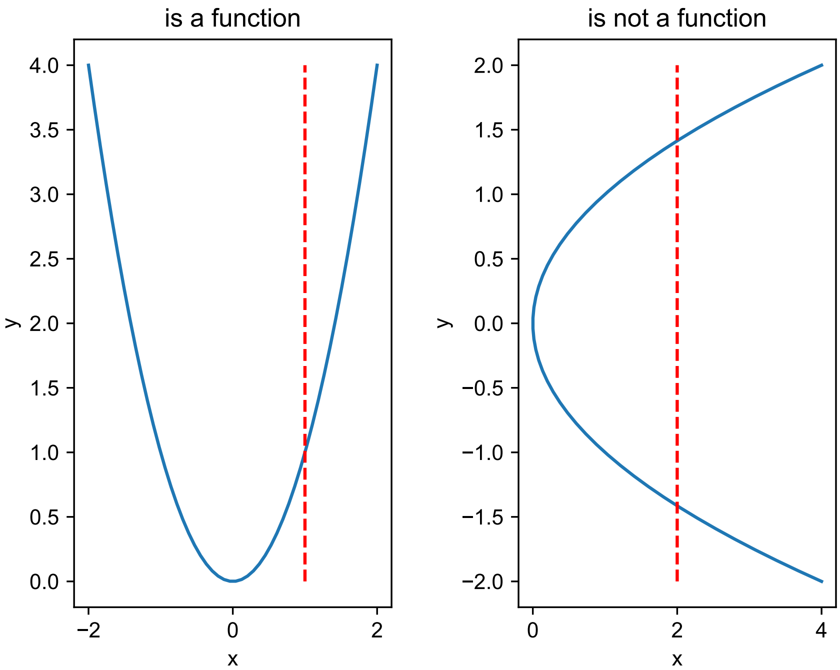


Figure 1: Plotting the mappings f (left panel) and g (right panel) vs. x. Any line parallel to the vertical axis has only one intersection with f, so $f(x)$ is a function. Any vertical line for $x > 0$ has two intersections with g and hence g is not a function.

```
# Python packages required for this assignment

import numpy as np
import matplotlib.pyplot as plt

# Declared functions

def fx(x):
    return x**2

# For g: y^2=x it is easier to think of computing x for different y and
# then plot the computed x on the horizontal axis and y on vertical axis.
def ginv(gy):
    return gy**2

# Main body of script the


# Costumizing the fonts used in plots
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = 'Arial'
```

```
plt.rcParams['font.size'] = 10

x = np.linspace(start=-2, stop=2, num=50)#Generate 50 linearly-spaced
                                          #numbers in [-2, 2]
fy = fx(x)
gy = x
gx = ginv(gy)  #Computing x for different values of g(x)

#make figure with two subplots
fig, ax = plt.subplots(nrows=1, ncols=2,
                       gridspec_kw={'wspace':0.4})

ax[0].plot(x, fy)   #Choosing the first subplot and plot the function
ax[0].plot([1, 1], [0, 4], '--r')
ax[0].set_xlabel('x')        #Set x label for the first subplot
ax[0].set_ylabel('y')  #Set y label ...
ax[0].set_title('is a function')

ax[1].plot(gx, gy)  #Choosing the second subplot and ...
ax[1].plot([2, 2], [-2, 2], '--r')
ax[1].set_xlabel('x')
ax[1].set_ylabel('y')
ax[1].set_title('is not a function')
# Save the figure
fig.savefig('a1-e1-function.pdf', format='pdf')
```

2. **Types of functions**

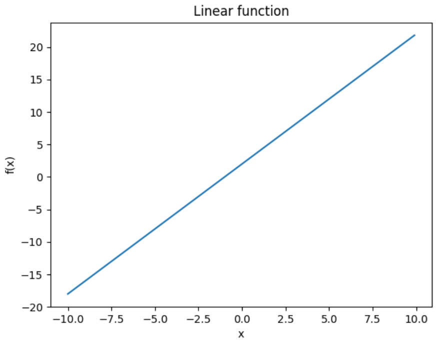
   The graphs of the functions can be found in Fig. 2.

   i. **Parametrized functions:** It is easy to determine what the parameters do in a linear function by changing them by hand (Fig. 3). Changing the bias causes the linear function to be offset in the y-axis. Changing the scaling factor in the linear function (the a in $ax+b$) will cause the slope of the line to change.

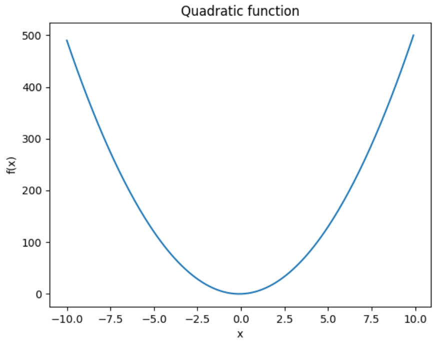
   ii. In quadratic functions, changing the bias causes the function to be offset in the y-axis and changing the quadratic term (the a in ax^2+bx+c) causes the parabola to become narrower (Fig. 4).

   Extra: The quadratic function (c) includes also a linear term that somehow does not contribute to the output. The reason is that the quadratic term has a higher multiplier, and moreover grows faster than the linear term. If you adjust the values the other way around and plot(x^2+5x) you will see a different story. Could you explain why the function has become imbalanced?

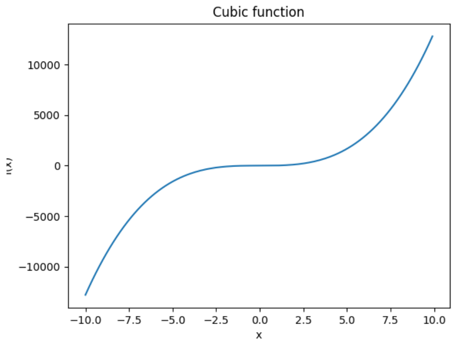
```
# python libraries required for the assignment
import numpy as np
import matplotlib.pyplot as plt


# define exponential function
def exponential(x, bias=0):
    return np.exp(x)+bias

# define linear function
def linear(x, scaling_factor=2, bias=2):
```



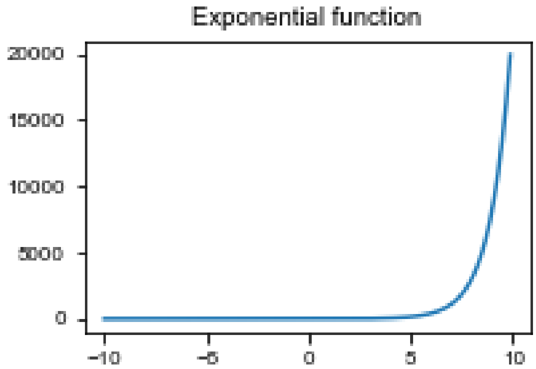
(a) Linear function



(b) Quadratic function

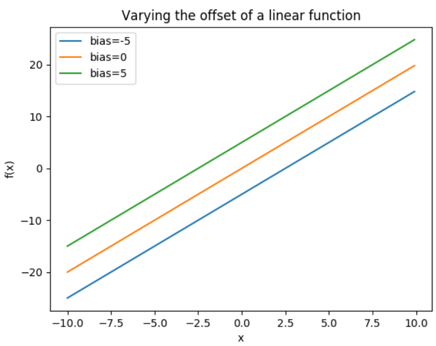


(c) Cubic function

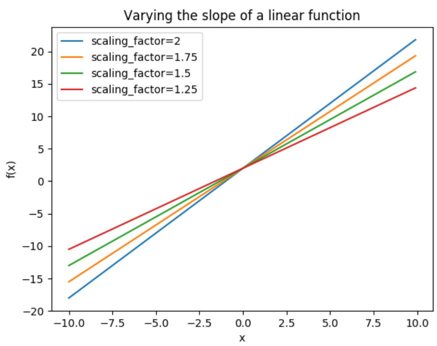


(d) Exponential function

Figure 2: Functions

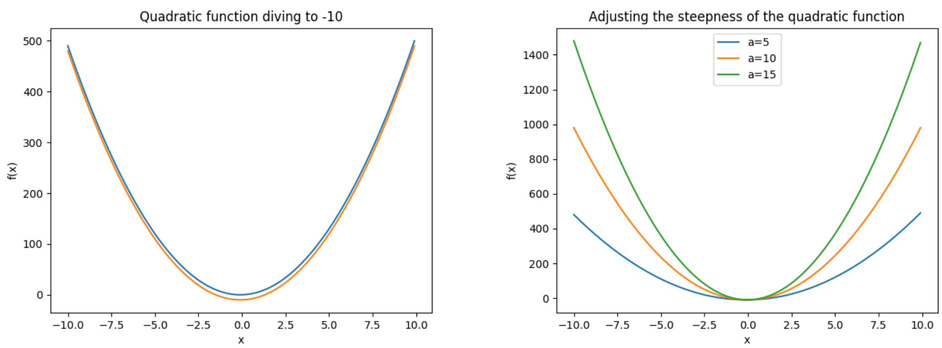


(a) Changing the bias



(b) Changing the scaling factor

Figure 3: Parametrization of linear function.



(a) Changing the bias

(b) Changing the quadratic term

Figure 4: Parametrization of quadratic function.

```
    return scaling_factor*x + bias

# define quadratic function
def quadratic(x, scaling_factors=[11,2],bias=0):
    # making sure that this function is used correctly
    assert isinstance(scaling_factors, list), \
        "scaling factors should be a list of two elements"
    assert len(scaling_factors)==2,  \
        "scaling factors should be a list of two elements"

    # unpack scaling factors
    a, b = scaling_factors

    # actual definition
    return a*x**2 + b*x + bias

# define polynomial with x^3
def cubic(x, scaling_factors=[11, 2, 2], bias=0):
    # sanity checks
    assert isinstance(scaling_factors, list), \
        "scaling factors should be a list of three elements"
    assert len(scaling_factors)==3,  \
        "scaling factors should be a list of three elements"

    # unpack scaling factors
    a, b, c = scaling_factors

    # actual definition
    return a*x**3 + b*x**2 + c*x**3 + bias

# define x as an interval between -10 and 10
x=np.arange(start=-10,step=0.1,stop=10)

# plot the functions

# Costumizing the fonts used in plots
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.sans-serif'] = 'Arial'
```

```
plt.rcParams['font.size'] = 8

# exponential
fig=plt.figure(figsize=(3,2)) # set up figure
plt.title('Exponential function')
plt.plot(x,exponential(x,bias=0)) # try the parameter here!

# linear
fig2=plt.figure() # set up figure
plt.title('Linear function')
plt.plot(x,linear(x,scaling_factor=2,bias=2)) # try the parameters here!


# quadratic
fig3=plt.figure() # set up figure
plt.title('Quadratic function')
plt.plot(x,quadratic(x,scaling_factors=[5,1], bias=0)) # try the parameters here

# cubic
fig4=plt.figure() # set up figure
plt.title('Cubic function')
plt.plot(x,cubic(x,scaling_factors=[11,2,2], bias=0)) # try the parameters here!


# changing offset of linear function
fig5=plt.figure() # set up figure
plt.title('Varying the offset of a linear function')
plt.plot(x,linear(x,scaling_factor=2,bias=-5))
plt.plot(x,linear(x,scaling_factor=2,bias=0))
plt.plot(x,linear(x,scaling_factor=2,bias=5))
plt.legend(labels=['b=-5', 'b=0','b=5'])

# changing scaling factor
fig6=plt.figure() # set up figure
plt.title('Varying the slope of a linear function')
plt.plot(x,linear(x,scaling_factor=2,bias=2))
plt.plot(x,linear(x,scaling_factor=1.75,bias=2))
plt.plot(x,linear(x,scaling_factor=1.5,bias=2))
plt.plot(x,linear(x,scaling_factor=1.25,bias=2))
plt.legend(labels=['a=2', 'a=1.75',
                   'a=1.5', 'a=1.25'])


# changing offset of quadratic function
fig7=plt.figure() # set up figure
plt.title('Quadratic function diving to -10')
plt.plot(x,quadratic(x,scaling_factors=[5,1],bias=-10))

# changing steepness of quadratic function
fig8=plt.figure() # set up figure
plt.title('Adjusting the steepness of the quadratic function')
plt.plot(x,quadratic(x,scaling_factors=[5,1],bias=-10))
plt.plot(x,quadratic(x,scaling_factors=[10,1],bias=-10))
```

```
plt.plot(x, quadratic(x, scaling_factors=[15,1], bias=-10))
plt.legend(labels=['a=5', 'a=10','a=15'])
```