Notebook 1.1 -- Background Mathematics

The purpose of this Python notebook is to make sure you can use CoLab and to familiarize yourself with some of the background mathematical concepts that you are going to need to understand deep learning.

It's not meant to be difficult and it may be that you know some or all of this information already.

Math is *NOT* a spectator sport. You won't learn it by just listening to lectures or reading books. It really helps to interact with it and explore yourself.

Work through the cells below, running each cell in turn. In various places you will see the words **"TO DO"**. Follow the instructions at these places and write code to complete the functions. There are also questions interspersed in the text.

Contact me at udlbookmail@gmail.com if you find any mistakes or have any suggestions.

```
# Imports math library
import numpy as np
# Imports plotting library
import matplotlib.pyplot as plt
```

Linear functions

We will be using the term *linear equation* to mean a weighted sum of inputs plus an offset. If there is just one input x, then this is a straight line:

$$y = \beta + \omega x,$$

where β is the y-intercept of the linear and ω is the slope of the line. When there are two inputs x_1 and x_2, then this becomes:

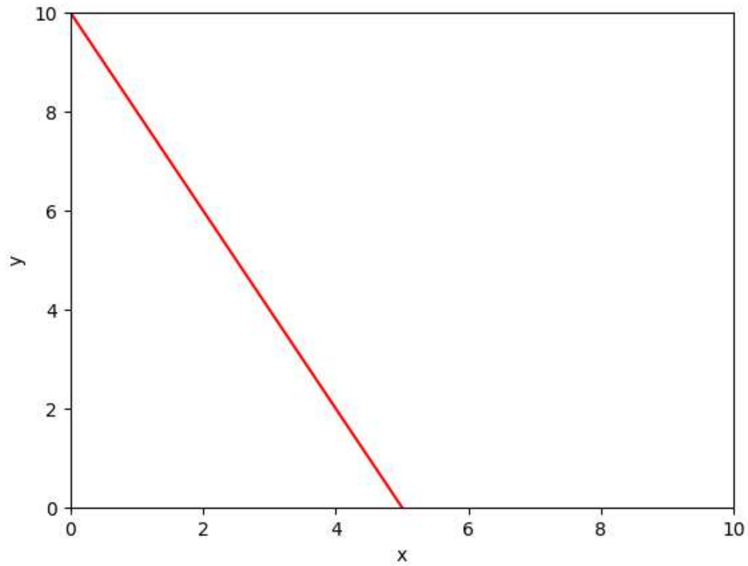
$$y = \beta + \omega_1 x_1 + \omega_2 x_2.$$

Any other functions are by definition **non-linear**.

```
# Define a linear function with just one input, x
def linear_function_1D(x,beta,omega):
  # TODO -- replace the code line below with formula for 1D linear equation
  y = beta + omega*x

  return y
```

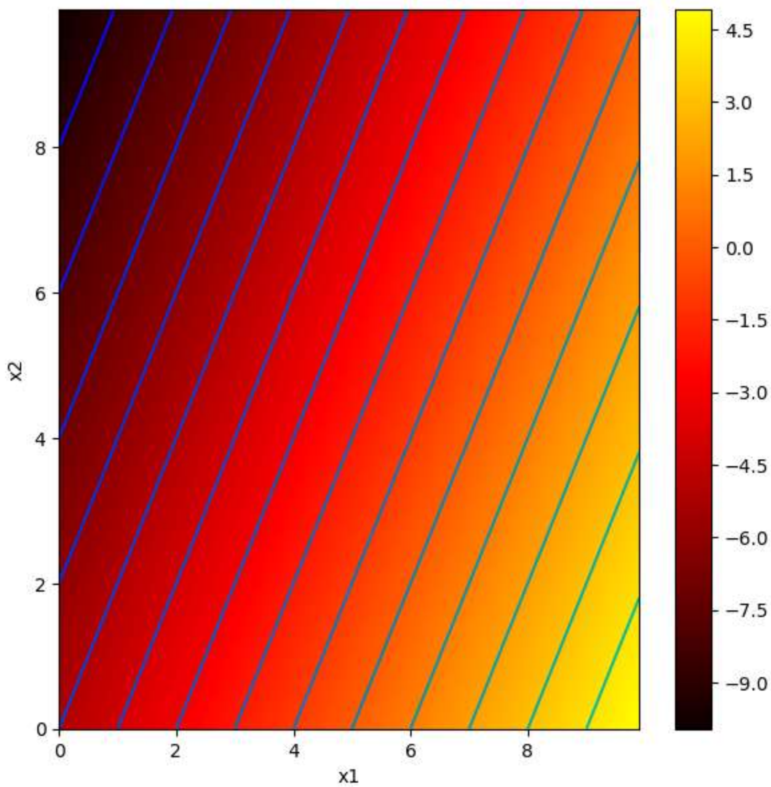
```
# Plot the 1D linear function

# Define an array of x values from 0 to 10 with increments of 0.01
# https://numpy.org/doc/stable/reference/generated/numpy.arange.html
x = np.arange(0.0,10.0, 0.01)
# Compute y using the function you filled in above
beta = 10; omega = -2

y = linear_function_1D(x,beta,omega)

# Plot this function
fig, ax = plt.subplots()
ax.plot(x,y,'r-')
ax.set_ylim([0,10]);ax.set_xlim([0,10])
ax.set_xlabel('x'); ax.set_ylabel('y')
plt.show

# TODO -- experiment with changing the values of beta and omega
# to understand what they do.  Try to make a line
# that crosses the y-axis at y=10 and the x-axis at x=5
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Now let's investigate a 2D linear function

```
# Code to draw 2D function -- read it so you know what is going on, but you don't have to change it
def draw_2D_function(x1_mesh, x2_mesh, y):
    fig, ax = plt.subplots()
    fig.set_size_inches(7,7)
    pos = ax.contourf(x1_mesh, x2_mesh, y, levels=256 ,cmap = 'hot', vmin=-10,vmax=10.0)
    fig.colorbar(pos, ax=ax)
    ax.set_xlabel('x1');ax.set_ylabel('x2')
    levels = np.arange(-10,10,1.0)
    ax.contour(x1_mesh, x2_mesh, y, levels, cmap='winter')
    plt.show()


# Define a linear function with two inputs, x1 and x2
def linear_function_2D(x1,x2,beta,omega1,omega2):
  # TODO -- replace the code line below with formula for 2D linear equation
  y = x1*omega1 + x2*omega2 + beta

  return y


# Plot the 2D function

# Make 2D array of x and y points
x1 = np.arange(0.0, 10.0, 0.1)
x2 = np.arange(0.0, 10.0, 0.1)
x1,x2 = np.meshgrid(x1,x2)  # https://www.geeksforgeeks.org/numpy-meshgrid-function/

# Compute the 2D function for given values of omega1, omega2
beta = -5; omega1 = 1.0; omega2 = -0.5
y  = linear_function_2D(x1,x2,beta, omega1, omega2)

# Draw the function.
# Color represents y value (brighter = higher value)
# Black = -10 or less, White = +10 or more
# 0 = mid orange
# Lines are contours where value is equal
draw_2D_function(x1,x2,y)

# TODO
# Predict what this plot will look like if you set omega_1 to zero
# Change the code and see if you are right.

# TODO
# Predict what this plot will look like if you set omega_2 to zero
# Change the code and see if you are right.

# TODO
# Predict what this plot will look like if you set beta to -5
# Change the code and see if you are correct
```



Often we will want to compute many linear functions at the same time. For example, we might have three inputs, x_1, x_2, and x_3 and want to compute two linear functions giving y_1 and y_2. Of course, we could do this by just running each equation separately,

$$\begin{aligned} y_1 &= \beta_1 + \omega_{11}x_1 + \omega_{12}x_2 + \omega_{13}x_3 \\ y_2 &= \beta_2 + \omega_{21}x_1 + \omega_{22}x_2 + \omega_{23}x_3. \end{aligned}$$

However, we can write it more compactly with vectors and matrices:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \omega_{11} & \omega_{12} & \omega_{13} \\ \omega_{21} & \omega_{22} & \omega_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix},$$

or

$$\mathbf{y} = \boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{x}.$$

for short. Here, lowercase bold symbols are used for vectors. Upper case bold symbols are used for matrices.

```
# Define a linear function with three inputs, x1, x2, and x_3
def linear_function_3D(x1,x2,x3,beta,omega1,omega2,omega3):
  # TODO -- replace the code below with formula for a single 3D linear equation
  y = x1*omega1+x2*omega2+x3*omega3+beta

  return y
```

Let's compute two linear equations, using both the individual equations and the vector / matrix form and check they give the same result

```
# Define the parameters
beta1 = 0.5; beta2 = 0.2
omega11 =  -1.0 ; omega12 = 0.4; omega13 = -0.3
omega21 =  0.1  ; omega22 = 0.1; omega23 = 1.2

# Define the inputs
x1 = 4 ; x2 =-1; x3 = 2

# Compute using the individual equations
y1 = linear_function_3D(x1,x2,x3,beta1,omega11,omega12,omega13)
y2 = linear_function_3D(x1,x2,x3,beta2,omega21,omega22,omega23)
print("Individual equations")
print('y1 = %3.3f\ny2 = %3.3f'%((y1,y2)))

# Define vectors and matrices
beta_vec = np.array([[beta1],[beta2]])
omega_mat = np.array([[omega11,omega12,omega13],[omega21,omega22,omega23]])
x_vec = np.array([[x1], [x2], [x3]])

# Compute with vector/matrix form
y_vec = beta_vec+np.matmul(omega_mat, x_vec)
print("Matrix/vector form")
print('y1= %3.3f\ny2 = %3.3f'%((y_vec[0],y_vec[1])))
```

```
Individual equations
y1 = -4.500
y2 = 2.900
Matrix/vector form
y1= -4.500
y2 = 2.900
<ipython-input-16-a1e5bfd57f13>:23: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error i
  print('y1= %3.3f\ny2 = %3.3f'%((y_vec[0],y_vec[1])))
```

Questions

1. A single linear equation with three inputs (i.e. **linear_function_3D()**) associates a value y with each point in a 3D space (x_1,x_2,x_3). Is it possible to visualize this? What value is at position (0,0,0)?

Answer: We cannot see it cause it would have 4-D, but we could try to imagine 3-D level curves. At position (0,0,0), we would het the value beta.

2. Write code to compute three linear equations with two inputs (x_1, x_2) using both the individual equations and the matrix form (you can make up any values for the inputs β_i and the slopes ω_{ij}.

```
beta1 = 0.5; beta2 = 0.2; beta3=0.4
omega11 =  -1.0 ; omega12 = 0.4; omega13 = -0.3
omega21 =  0.1  ; omega22 = 0.1; omega23 = 1.2
omega31 =  4.5  ; omega32 = 2.6; omega33=1.27

# Define the inputs
x1 = 4 ; x2 =-1; x3 = 2

# Compute using the individual equations
y1 = linear_function_3D(x1,x2,x3,beta1,omega11,omega12,omega13)
y2 = linear_function_3D(x1,x2,x3,beta2,omega21,omega22,omega23)
y3= linear_function_3D(x1,x2,x3,beta3,omega31,omega32,omega33)
print("Individual equations")
print('y1 = %3.3f\ny2 = %3.3f\ny3 = %3.3f'%((y1,y2,y3)))

# Define vectors and matrices
beta_vec = np.array([[beta1],[beta2],[beta3]])
omega_mat = np.array([[omega11,omega12,omega13],[omega21,omega22,omega23],[omega31,omega32,omega33]])
x_vec = np.array([[x1], [x2], [x3]])

# Compute with vector/matrix form
y_vec = beta_vec+np.matmul(omega_mat, x_vec)
print("Matrix/vector form")
print('y1= %3.3f\ny2 = %3.3f\ny3 = %3.3f'%((y_vec[0],y_vec[1],y_vec[2])))
```

```
Individual equations
y1 = -4.500
y2 = 2.900
y3 = 18.340
```

```
Matrix/vector form
y1= -4.500
y2 = 2.900
y3 = 18.340
<ipython-input-19-f03684df1dd4>:24: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error i
  print('y1= %3.3f\ny2 = %3.3f\ny3 = %3.3f'%((y_vec[0],y_vec[1],y_vec[2])))
```

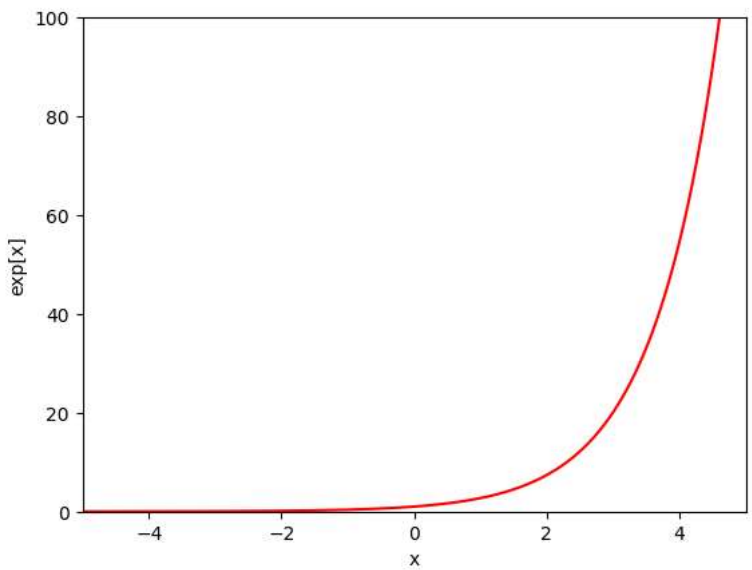
Special functions

Throughout the book, we'll be using some special functions (see Appendix B.1.3). The most important of these are the logarithm and exponential functions. Let's investigate their properties.

We'll start with the exponential function $y = \exp[x] = e^x$ which maps the real line $[-\infty, +\infty]$ to non-negative numbers $[0, +\infty]$.

```
# Draw the exponential function

# Define an array of x values from -5 to 5 with increments of 0.01
x = np.arange(-5.0,5.0, 0.01)
y = np.exp(x) ;

# Plot this function
fig, ax = plt.subplots()
ax.plot(x,y,'r-')
ax.set_ylim([0,100]);ax.set_xlim([-5,5])
ax.set_xlabel('x'); ax.set_ylabel('exp[x]')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



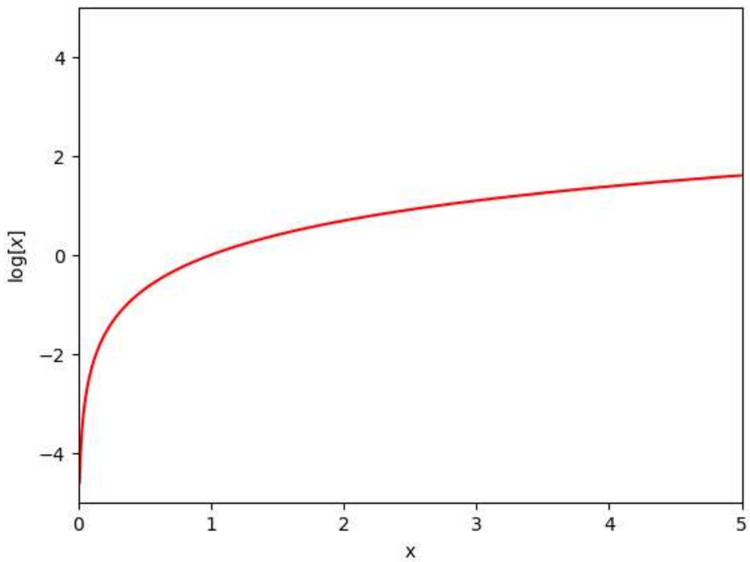
Questions

1. What is $\exp[0]$? Answer: 1
2. What is $\exp[1]$? Answer: e
3. What is $\exp[-\infty]$? Answer: 0
4. What is $\exp[+\infty]$? Answer: ∞
5. A function is convex if we can draw a straight line between any two points on the function, and this line always lies above the function. Similarly, a function is concave if a straight line between any two points always lies below the function. Is the exponential function convex or concave or neither? Answer: It is convex

Now let's consider the logarithm function $y = \log[x]$. Throughout the book we always use natural (base e) logarithms. The log function maps non-negative numbers $[0, \infty]$ to real numbers $[-\infty, \infty]$. It is the inverse of the exponential function. So when we compute $\log[x]$ we are really asking "What is the number y so that $e^y = x$?"

```
# Draw the logarithm function

# Define an array of x values from -5 to 5 with increments of 0.01
x = np.arange(0.01,5.0, 0.01)
y = np.log(x) ;

# Plot this function
fig, ax = plt.subplots()
ax.plot(x,y,'r-')
ax.set_ylim([-5,5]);ax.set_xlim([0,5])
ax.set_xlabel('x'); ax.set_ylabel('$\log[x]$')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Questions

1. What is $\log[0]$? Answer: Not definded, it goes to $-\infty$
2. What is $\log[1]$? Answer: 0
3. What is $\log[e]$? Answer: 1
4. What is $\log[\exp[3]]$? Answer: 3
5. What is $\exp[\log[4]]$? Answer: 4
6. What is $\log[-1]$? Answer: Not definded
7. Is the logarithm function concave or convex? Answer: It is concave