Topics:

1. Plot some graphics in Python.
Talk about single derivative for function of one argument.
Derive things relative to derivative

2. Plot some graphics in Google Chrome Web Browser of 2 variables
Talk about single derivative for function of one argument.
Play with photo of Neo from Matrix.

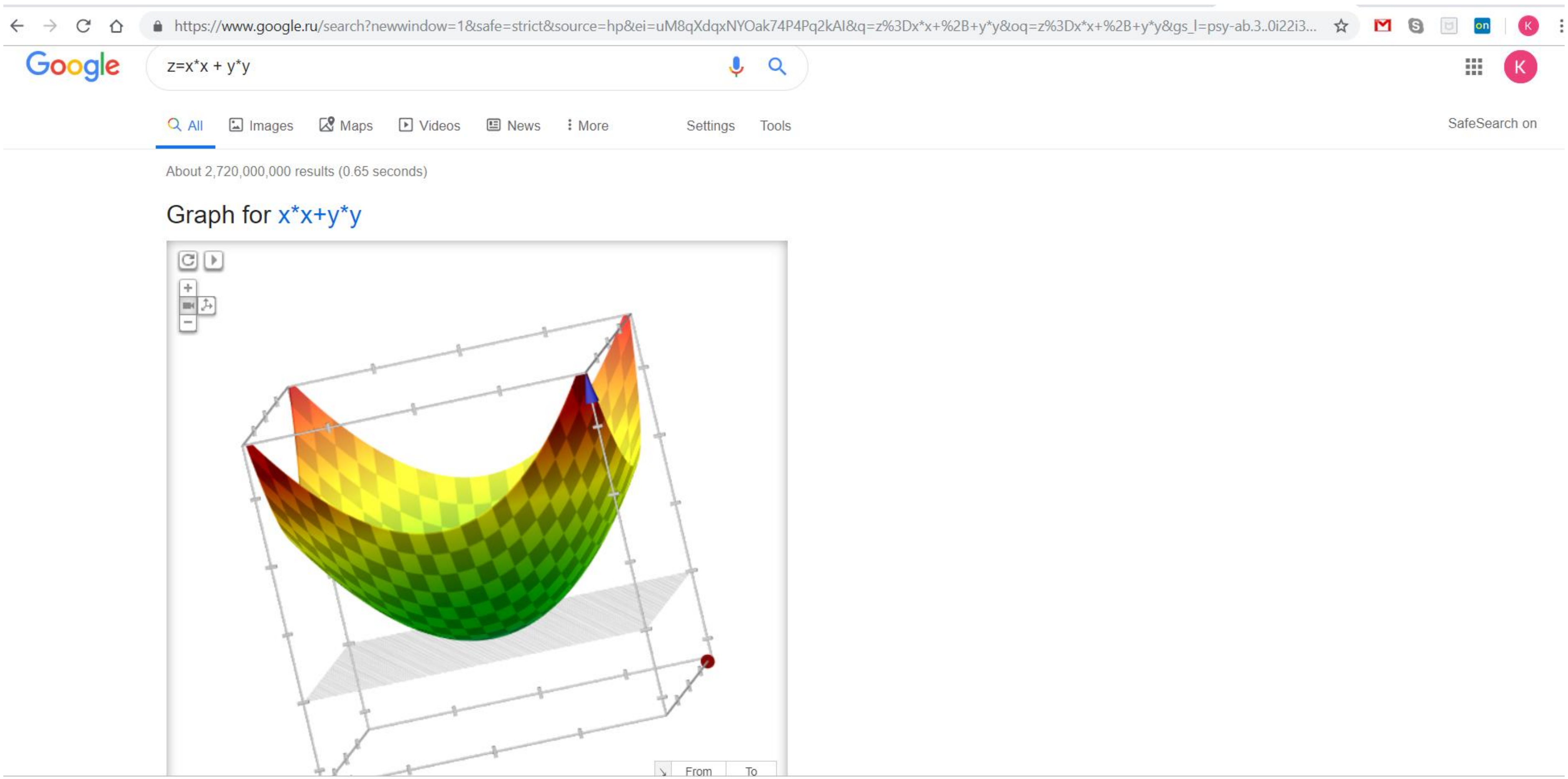
3. Single Layer Neural Net – it's construction. Various Transfer functions.

4. Talk about derivatives

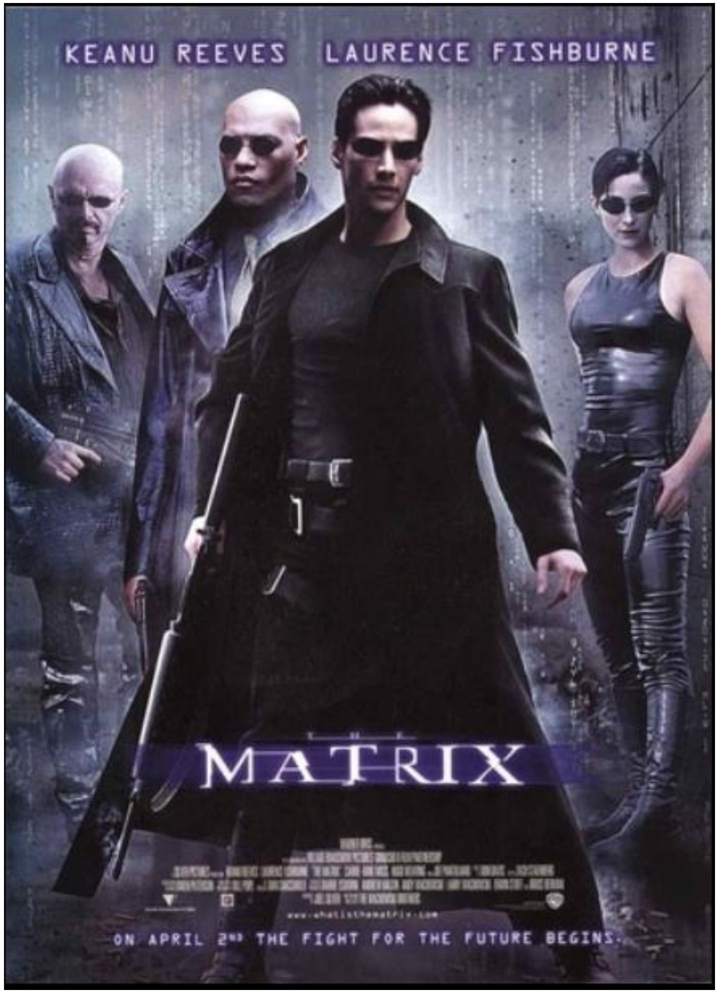
Plot some graphics in Python

```
#!/usr/bin/env python3
# L2_plot_snippet.py

import numpy as np
import matplotlib.pyplot as plt
import sys

x = np.arange(0.0, 10.0, 0.1)

#==================================================
beta = 2.0
y1 = (beta-1.0)*0.5*x*x + (2.0-beta)*np.abs(x)
plt.plot(x, y1, 'g-', label='EN beta=' + str(beta))
#==================================================

#==================================================
beta = 1.6
y2 = (beta-1.0)*0.5*x*x + (2.0-beta)*np.abs(x)
plt.plot(x, y2, 'r-', label='EN beta=' + str(beta))
#==================================================

#==================================================
beta = 1.1
y3 = (beta-1.0)*0.5*x*x + (2.0-beta)*np.abs(x)
plt.plot(x, y3, 'b-', label='EN beta=' + str(beta))
#==================================================

yMin = np.min(np.concatenate( (y1, y2, y3) ))
yMax = np.max(np.concatenate( (y1, y2, y3) ))

#==================================================
plt.xlim(x[0], x[-1])
plt.ylim(x[0], x[-1])
plt.gca().set_aspect('equal', adjustable='box')
#==================================================

plt.xlabel('x')
plt.ylabel('y(x)')
plt.legend(loc='upper left')
plt.grid(True)
plt.title('Example scalar functions')
plt.show()
```

Plot some graphics in Google



Functions, points, curves, discontinuity



Set of functions

- We defined set of real number, etc. at the beginning
- But we can consider sets of more complicated objects like functions
- If $f_1: X \to Y$ and $f_2: X \to Y$ then we can take this two functions and consider a set $\{f_1, f_2\}$

Function class

- Instead of enumerating functions directly we can do one trick to consider infinite set of functions
- We have a form $F(X,W)=Y$ where X is variable, but W is something which encode selection of function

What we want – we want to approximate

<u>1-st step Model (or pattern) structure</u>

$y = F(x)$ *is real function, which* ***we don't know,***
but we want to know it very much, because it help us to make a decision.

$\hat{F}(x) = \hat{F}(x; a) \in \mathcal{F}(a)$ *instead we construct class(or set) of functions.*

Structure Model

$\widetilde{F}(x) = \sum_{m=1}^{M} b_m \cdot S_m(a_{0m} + \sum_{j=1}^{n} a_{jm} x_j) + b_0$
is example of single layer Neural Network

$\boldsymbol{S_m}$ is called activation function or transfer function or squashing function

$\boldsymbol{a_{[M\ x\ n]}}$ - is weight matrix for first layer.

$\boldsymbol{a_0}$ - is bias term for first layer

$\boldsymbol{b_{[1,M]}}$ – is weight matrix for output layer.

$\boldsymbol{b_0}$ – is bias term for output layer.

Structure Model - Massaging

$$\widetilde{F}(x) = \sum_{m=1}^{M} b_m \cdot S(a_{0m} + \sum_{j=1}^{n} a_{jm} x_j) + b_0$$

For compress this formula we can assume that:

- $x_0 = 1$ always (assumption)
- $S_0 = 1$ always (assumption)

So we can more compactly write:

$$\widetilde{F}(x; a, b) = \sum_{m=0}^{M} b_m \cdot S_m(\sum_{j=0}^{n} a_{jm} x_j)$$

What we need to select specific function?

- All a_{ij}, b_j are parameters to be learned.
- In optimization it called variables, but in Machine Learning the variables which are needed to be find by learning procedure called parameters.
- This function class have at least continuous space of parameters.

Why this call Neural Network, where is a Network?

- People who worked in this area look not into formula described this model
- But into graph interpretation of the model they draw a graph and say “network”.
- Without loss of generality we can work with Alegebraic form

Derivative of some Transfer Functions

Function name	Algebraic expression for transfer function	Evaluated derivative (gradient)
Sigmod or Logistic	$s(z) = \frac{1}{1+\exp(-z)}$	$s'(z) = s(z)(1 - s(z))$
Identity function	$s(z) = z$	$s'(z) = 1$
RELU	$s(z) = \max(0, z)$	$s'(z) = \begin{cases} 1, z > 0 \\ 0, z < 0 \\ \text{not esist}, z = 0 \end{cases}$
Hyperbolic tangent	$s(z) = \frac{\sinh(x)}{\cosh(x)} = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}}$	$s'(z) = 1 - s(z)^2$
Parametric Relu (PreLU)	$s(z) = \begin{cases} z, z > 0 \\ az, z \leq 0 \end{cases}$	$s'(z) = \begin{cases} 1, z > 0 \\ a, z < 0 \\ \text{not exis}, z = 0 \end{cases}$

Derivative of Sigmoid Function

$$sigmoid(z)' = \left(\frac{1}{1+\exp(-z)}\right)' = -\left(\frac{1}{1+\exp(-z)}\right)^2 \exp(-z)\,(-1) =$$
$$\left(\frac{1}{1+\exp(-z)}\right)\frac{\exp(-z)}{1+\exp(-z)} = S(z)\left(\frac{1+\exp(-z)-1}{1+\exp(-z)}\right) = sigmoid(z)(1 - sigmoid(z))$$

Discussion About Derivative

On the Board

Taylor series

1. Named for the English mathematician *Taylor* (1685-1731)
2. We can control **p** and **n**. But θ $from$ $[0,1]$ is not our control.
3. $\boldsymbol{f^k}$ is take derivative to function k times
4. $\boldsymbol{f}: \mathbb{R} \to \mathbb{R}$

$$f(x) = \sum_{k=0}^{n} \frac{f^k(x_0)}{k!}(x - x_0)^k + R_n(x)$$

$$R_n(x) = \frac{(x - x_0)^{n+1}(1 - \theta)^{n-p+1}}{pn!} f^{n+1}(x_0 + \theta(x - x_0))$$

Example of Apply Taylor Series

- Original sigmoid function looks like $(x) = \frac{1}{1+\exp(-x)}$.
- Now let's use simple algebra to understand some symmetry properties:

$$1 - f(x) = 1 - \frac{1}{1+\exp(-x)} = \frac{1+\exp(-x)-1}{1+\exp(-x)} = \frac{\exp(-x)}{1+\exp(-x)} = \frac{1}{1+\exp(x)}$$

$$=> 1 - f(x) = f(-x)$$

- In principle we can replace exponent with it's Taylor expansion near point $x_0 = 0$ via considering only two first terms
- $\overset{\frown}{f}(x) = \frac{1}{1+\exp(-x)} = \frac{1}{1+\left(1+\frac{-x}{1!}+\frac{(-x)^2}{2!}+\cdots\right)}$

Example of Apply Taylor Series

Script:

```
#!/usr/bin/env python3
# test_sigmoid.py, Konstantin Burlachenko
import numpy as np
import matplotlib.pyplot as plt
def y_aprox(x):
    if (x < 0):
        compute = 1/(1+(1-x+x*x/2))
        return compute
    else:
        return 1 - y_aprox(-x)
x = np.arange(-20, 20, 0.1)
y = 1/(1+np.exp(-x))
yapr = [y_aprox(xi) for xi in x]
plt.plot(x, y)
plt.plot(x, yapr)
plt.title('Sigmoid and it approximation')
plt.legend(['Sigmoid', 'Sigmoid approximation'])
plt.show()
```