**Natural Language Processing** **Tel Aviv University**

# Assignment 0: Getting to know the basic tools

Due Date: *Nov 14 , 2024* Lecturer: Mor Geva & Maor Ivgi, TA: Noa Mark

## Preliminaries

**Submission Instructions** Submit your solution through Moodle. Your submission should consist of a single zip file named `<id1>_<id2>.zip` (where `id1` refers to the ID of the first student). This zip file should include both the Colab file as well as a tex one. The Colab file should include the answers for the first (Google Colab) and second (Numpy + Torch) sections, in an ipynb format, which has been downloaded from Colab (File → Download → Download .ipynb)

## 1 Google Colab

In the following sections you will be getting to know Google Colab - a web framework that enables creating Python notebooks which can be easily shared among collaborators, as well as running them on a cloud platform, which apart from a CPU, also offers an option to use a GPU (and even a TPU).

Google Colab is very useful tool in research, and will be used throughout the class to submit home assignments. Therefore, it is important for you to get to know it as soon as possible. We will see in this exercise basic tools Google Colab provide including: mounting your drive, for easy file management; writing terminal command in its cells, to download additional libraries; using online computational power, to work with GPU.

If you are unfamiliar with Google Colab, we recommend on Stanford cs231n notes on "Working remotely on Google Colaboratory", and the following nice snippets notebook.

Before this exercise, verify you have Google account with an associated Google Drive and Colab apps.

### 1.1 Mounting your drive

Mounting your drive enable you to manage your project files online very similar to your local file system. It's a comfortable way to work on your project (You can also work with files in a temporary memory, hopefully you will remember to save them! Or work with Google Cloud storage, as explained here.)

Create *'NLP_ex0'* directory in your Drive (under My Drive). Download a smiley image from the browser and upload it to this directory as *'enjoying_nlp.png'*.

Create your first cell with the following lines:

```
from google.colab import drive
import os
```

```
drive.mount('/content/drive', force_remount=True)
drive_directory = '/content/drive/My Drive/NLP_ex0'
os.chdir(drive_directory)
```

Hopefully, now your google-drive is connected to your colab notebook! You can work with it very similarly to the way you work with your local file system. Lets verify: load and show the smiley image:

```
from IPython.display import Image, display
display(Image('enjoying_nlp.png'))
```

### 1.2 Terminal commands and libraries download

You can write terminal commands in colab. For example, we could define the session directory (as we did before using os library) using a terminal command:

```
%cd /content/drive/My Drive/'NLP_ex0'
```

The default environment in colab comes with many preinstalled packages. To download new libraries we can run pip install command with the ! prefix:

```
!pip install transformers
```

We could also downloaded the image through the command line as follow:

```
!wget https://en.wikipedia.org/wiki/Smiley#/media/File:SmileyFace.png
```

Google colab offer many tools to help you develop your research projects! We hoped this section helped you to get comfortable with the basic framework, and encouraged you to learn further.

## 2 Numpy + Torch

Throughout this section, we will be recalling some basics from NumPy, Matplotlib and PyTorch, three Python frameworks that will be definitely be used for future home assignments.

Please continue to work on the Colab notebook you started in the previous section.

1. Plot the curve of the function $y = 2x + 3$ within the $(0, 10)$ interval, using *numpy.linspace*. Try to use the minimal $x$ samples that's possible to accurately plot this function.
2. Plot the curve of the function $y = x^2$, again using *numpy.linspace*. Here you might need a larger amount of $x$ axis samples.
3. Plot 2 again, but this time add standard Gaussian noise to the samples. Namely, $y = x^2 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 1)$.

4. Randomly sample 100 real numbers from the interval $[0, 1)$ uniformly (using *numpy.random*. Computes the empirical mean and variance and print these. (what would you expect that to be?).

5. Print the indices of the 10 largest floats (within the original random array from previous subsection), using *numpy.argpartition*.

6. The sigmoid function $\sigma : \mathbb{R} \to (0, 1)$ is given by

$$\sigma(x) \triangleq \frac{1}{1 + e^{-x}} \equiv \frac{e^x}{1 + e^x}$$

   It is very important in machine learning as it maps any real number to the range $(0, 1)$ and is thus often used to induce a probability distribution. On a single figure, plot both $\sigma(x)$ and $\sigma(-x)$ in the range of $[-10, 10]$. What is the relation between $\sigma(-x)$ and $\sigma(x)$? What is the derivative of the sigmoid with respect to $x$? Answer in a comment in the cell (you can use $f(x)$ instead of $\sigma(x)$ for convenience) and add to the same plot the derivative of $\sigma(x)$ in the same x-range.

7. Now let's move to mess around a bit with the GPU. Import torch, then check whether the GPU is visible to you. Here you can use *torch.cuda.is_availiable*().

8. If not, make it happen. Google Colab offers a free GPU (and also some paid upgraded ones) to every user. (help: go to Runtime → Change runtime type)

9. Now, create a random 3-dimensional tensor using *torch.rand*, and print its current device using the *.device* function. Now, move it to the GPU. Here you may use the *.to*('*cuda*') method of torch tensors. Now print its device again to verify it has indeed been moved to the GPU.

# 3 Theoretical background

Understanding the mathematical properties of different functions is significant to gain dipper understanding of the way the networks works. In this section, we encourage you to verify you are familiar with some of Softmax properties. You do not need to submit it.

1. Softmax function is often used in neural networks. It's define as follow: $softmax : \mathbb{R}^N \to (0, 1)^N$

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

Prove that softmax is invariant to constant offset in the input, i.e prove that for any input vector $\boldsymbol{x}$ and any constant $c$,

$$\text{softmax}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{x} + c)$$

# 4 Latex

In this part you are asked to use Latex in order to create a file which is visible to you only as a PDF. LaTeX is a bit different from word processors you may be used to in that you write in a markup language that is then compiled into the final document. To do so, you will need to use a LaTeX editor. We recommend using an online editor such as Overleaf, which is a web platform that enables easily creating

files which are making use of Latex. Visual Studio Code is also a strong choice if you prefer to work offline, though it may require you to install many dependencies.

Since you are going to be asked to submit your written answers (to the home assignments) in a digital format (i.e. PDF), becoming familiar with latex early on should come in handy.

Latex has a very wide community so many tutorial and forums exist (including a lively TeX exchange), but we recommend starting by looking at the overleaf's documentation or watch Dr Trefor bazzet short introduction video. You might also find Detexify useful to find the commands needed to draw a particular symbol. Another useful tool is latexify which converts python code to formatted latex script.

In the next two pages of this PDF, you will find a brief recap on gradient descent. Use overleaf (or any tex editor you prefer) to create a .tex file that is as identical as possible (in content, even if the exact locations of sentences differ) to the given document. Feel free to copy any text directly from the given PDF to avoid retyping everything, just make sure you correct any symbol to its correct form. You can extract the figure from the PDF by right-clicking it and export/save the image locally. Note, your PDF should show the date when you compiled this file , instead of "March 14, 2023".

When done, you can just download the *.tex* file from overleaf (menu → source), and attach that to your submission *.zip* file, along with the generated PDF file you created. Make sure to change the author names, IDs and submission date!

# Gradient Descent Very Brief Review

Student 1 name
Student 1 ID

Student 2 name
Student 2 ID

March 14, 2023

**Abstract**

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function. It and its variants are with no doubt the most common optimization algorithms which are used in ML.

## 1 Introduction

Optimization refers to the task of minimizing/maximizing an objective function $f(x)$ parameterized by $x$. In machine/deep learning terminology, it's the task of minimizing the cost/loss function $\mathcal{L}(\theta)$ parameterized by the model's parameters $\theta \in \mathbb{R}^d$. This objective function might be represented in many ways - it could be a simple linear function such that $\theta$ is just one matrix, as well as a 175B parameters transformer architecture, such that $\theta$ is a collection of large number of matrices and bias terms. Optimization algorithms (in the case of minimization) have one of the following goals:

- Find the global minimum of the objective function. This is feasible if the objective function is convex, i.e. any local minimum is a global minimum.
- Find the lowest possible value of the objective function within its neighborhood. That's usually the case if the objective function is not convex as the case in most deep learning problems.

## 2 Algorithm

Let's denote our objective function by $\mathcal{L}$ (it is often referred to it as the loss function), and let's say it is parameterized with some parameters which we will denote with $\theta$. Additionally, let's denote an additional hyperparameter the algorithm uses, called the *learning rate*, with $\eta$. The Vanilla Gradient Descent Algorithm suggests following these following steps:

1. Start by Randomly initialize values for $\theta_0$
2. Update $\theta$ values: $\theta_{t+1} = \theta_t - \eta \nabla L(\theta)$
3. Repeat until the slope is approximately flat. Namely, $\frac{\partial \mathcal{L}(\theta)}{\partial \theta} \approx 0$

### 2.1 Learning Rate

How big the steps gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights. To illustrate the effects of a wrong/right choice of the learning rate, Figure 1 is attached.

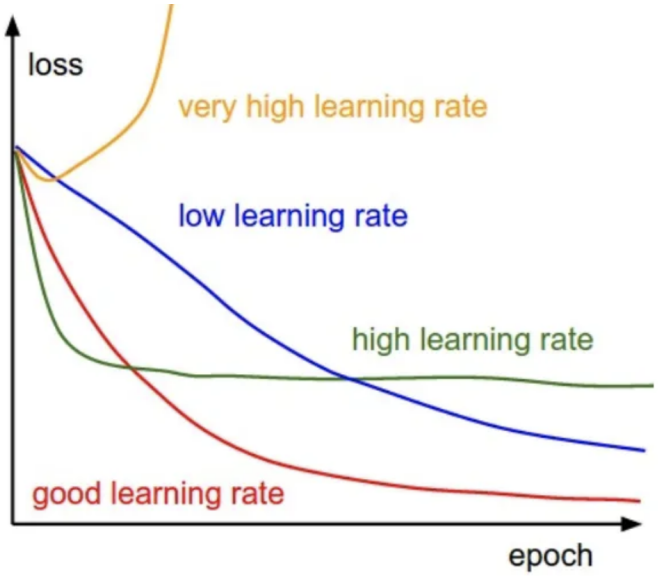


Figure 1: Effects of using too large/small learning rate while applying the GD algorithm