# Neural Networks and Deep Learning
# 26:198:685:01 Spring 2020
# Homework 2

**Instructor:** Farid Alizadeh
**Due Date:** Wednesday March 25, 2020, at 11:50PM

last updated on March 19, 2020

Please answer the following questions in **electronic** form and upload and submit your files to Sakai Assignment site, before the due date. Make sure to click on the **submit** button. The file format should be **pdf**. You may either typeset or hand write your answers. In case of hand-write transform your work into a **pdf** file using apps such as **CamScanner**. You should have only a single pdf file.
For this homework, you also have Python scripts, include the text of the script with your pdf submission (no separate file is needed).

1. Consider the function $f$ of three variables $x, y$ and $z$ defined as follows:

$$
\begin{aligned}
n1 &= x * y \\
n2 &= n1 * z \\
n3 &= \text{sigmoid}(n1 + y) = \tfrac{1}{1+\exp(-(n1+y))} \\
n4 &= 2 * n3 + x \\
f1 &= \log\big(1 + \exp(n1 + n2 + n4)\big) \\
f2 &= \tfrac{\exp(n3)}{\exp(n1)+\exp(n3)+\exp(n4)}
\end{aligned}
$$

   1a) Draw the computation graph of $f_1$ and $f_2$.

   1b) Set $x = 1, y = 2, z = 3$ and compute all nodes values, using these inputs and show the results neatly on a copy of your graph.

   1c) Now compute the gradient $\nabla_{x,y,z} f_1(x, y, z)$ at $x = 1, y = 2$ and $z = 3$, using the back propagation algorithm. Show the values of partial derivatives at each node on a copy of the graph.

   1d) Now compute the partial derivatives of both $f_1$ and $f_2$ with respect to the variable $y = 1$, only, with $x = z = 1$, using the forward method. Show both the values of each node, as well as the values of the partial derivatives with respect to $y$ for the node.

   Sbmit your work in pdf format.

2. Now, using Tensorflow you are to set up the graph of the $f_2$ function as described in Q1, and then compute the derivatives with respect to variables $x, y$ and $z$. To do so, create a python file called `hw2Q2.py`.

   **Please note:** There are some important differences between Tensorflow version 1.xx and Tensorflow version 2.xx. In particular, version 2 and above do not require initialization since they now by default implement *eager execution*, which means, Tensorflow evaluates expressions as needed without waiting for the graph to be built. Also, for gradient computing a construct known as `GradientTape` has to be used. See the Tensorflow tutorial webpage for more information. Also in the Lecture Notes tab at Sakai, I have updated the examples and added version 2 examples as well.

   In the file `hw2Q2.py` at the top, make sure to both state the version of Tensorflow software you are using and print the version, as follows:

```
# Tensorflow version 1.14 is used
print("Tensorflow version:", tf.__version__)
```

(a) Write python script building the Tensorflow graph implementing the function $f_2$ as described in Q1. Print both the value of $f(x, y, z)$ at $x = 1, y = 1, z = 1$, and also the value of the gradient vector of $f_2$ at these values.

3. **Keras/Python Project:** This exercise continues the fashion-nist data you worked on in the last homework. We are going to add various regularization techniques and see if there is a significant improvement in the plain approach.

   3a) Do not use the entire training data. Instead randomly select 50,000 of the 60,000 data points as your actual training set, and the remaining 10,000 points as your validation data. Experiment both with $\ell_2$ and $\ell_1$ norms for regularization. Choose $\alpha = .01$, and $\alpha = .001$. Use 20 epochs. Once done, graph the error rate with respect to the training data, and with respect to the validation data Does any of these combinations perform better than the results you obtained from homework 1? Also, make sure to both print and darw as a heat map the confusion matrix of the error rate for the test set. Submit the answer for this homework in a file named `hw2Q3a.py`.

   3b) Repeat part a) but instead of norm regularization, use Dropout layers between all layers. Use a dropout rate of only 20% for the input layer, and a rate of 50% for the others. Submit your work in a file called `hw2Q3b.py`

   3c) Repeate part a) but instead of norm regularization, use early stopping. set the epoch parameter to 100. At what epoch number did the process stop? Is the results better than those you obtained in homework 1? Submit your work in a file called `hw2Q3c.py`.

**Running your code on Google Colab:** Some programs may require a long time for each to run. You can use Google's *Colab* project and use their servers with GPU and even TPU (Tensor Processing Unit). You need to have a Google account. Note that with Rutgers *ScarletMail* you will have access to Google services with additional privileges. Here are the steps you need to take:

A. Open a web browser tab and go to the page https://colab.research.google.com. You should see a pop-up page and at the bottom left you should see link called **New Notebook** followed by **Cancel**. For the first time choose cancel to see some introductory tutorial (not very long.)

B. Once you read the tutorial introduction and have an idea how Google Colab works (essentially a Python notebook that allows you to run your code on their servers,) you can visit the page again and choose **New Notebook**.

C. In the notebook you can start typing your code, or cut and paste it from a local file.

D. You can click on the button on the left side when you are ready to run your code.

E. By default your code is run on a regular server, much like your own personal computer, and thus, it will be slow. To access their GPU or TPU servers, select the **Edit** tab and from there choose **Notebook Setting**. On the second line you may choose GPU. This should run your code much faster.

F. At the time of writing there are bugs in the Colab. By default it runs on Tensorflow 1.14, and may not work on code using Tensorflow 2.1 or higher. You can enforce Tensorflow 2.1 behavior in your Colab notebook by typing at the beginning:
`%tensorflow_version 2.x`
However, beware, this is buggy, especially when running with Keras.

G. Happy deep learning!