Assignment 3: Neural Networks

Due November 22 at 11:59pm

This assignment is to be done individually.

Important Note: The university policy on academic dishonesty (cheating) will be taken very seriously in this course. You may not provide or use any solution, in whole or in part, to or by another student.

You are encouraged to discuss the concepts involved in the questions with other students. If you are in doubt as to what constitutes acceptable discussion, please ask! Further, please take advantage of office hours offered by the instructor and the TA if you are having difficulties with this assignment.

DO NOT:

- Give/receive code or proofs to/from other students
- Use Google to find solutions for assignment

DO:

- Meet with other students to discuss assignment (it is best not to take any notes during such meetings, and to re-work assignment on your own)
- Use online resources (e.g. Wikipedia) to understand the concepts needed to solve the assignment.

Submitting Your Assignment

The assignment must be submitted online at `https://canvas.sfu.ca`. You must submit two files:

1. An assignment report in **PDF format**, named `report.pdf`. This report should contain the solutions to Error Propagation part and a brief explanation for Fine-tuning part .
2. Your code for Fine-Tuning part.

---

1 Neural Networks (5 marks)

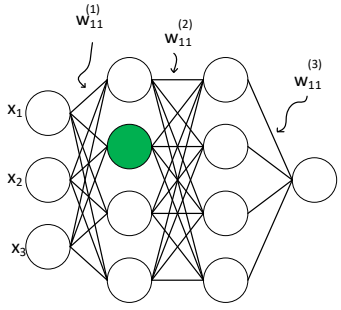
Error Propagation [2 marks]

We will derive error derivatives using back-propagation on the network below.

Notation: Please use notation following the examples of names for weights given in the figure. For activations/outputs, the green node would have activation $a_2^{(2)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3$ and output $z_2^{(2)} = h(a_2^{(2)})$.

Activation functions: Assume the activation functions $h(\cdot)$ for the hidden layers are logistics. For the final output node assume the activation function is an identity function $h(a) = a$.

Error function: Assume this network is doing regression, trained using the standard squared error so that $E_n(w) = \frac{1}{2}(y(\boldsymbol{x}_n, w) - t_n)^2$.



[1 mark] Consider the output layer.

- Calculate $\frac{\partial E_n(w)}{\partial a_1^{(4)}}$. Note that $a_1^{(4)}$ is the activation of the output node, and that $\frac{\partial E_n(w)}{\partial a_1^{(4)}} \equiv \delta_1^{(4)}$.
- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{12}^{(3)}}$.

[1 mark] Next, consider the penultimate layer of nodes.

- Write an expression for $\frac{\partial E_n(w)}{\partial a_1^{(3)}}$. Use $\delta_1^{(4)}$ in this expression.
- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(2)}}$.

[1 mark] Finally, consider the weights connecting from the inputs.

- Write an expression for $\frac{\partial E_n(w)}{\partial a_1^{(2)}}$. Use the set of $\delta_k^{(3)}$ in this expression.

- Use this result to calculate $\frac{\partial E_n(w)}{\partial w_{11}^{(1)}}$.

Fine-Tuning a Pre-Trained Network [2 marks]

In this question you will experiment with fine-tuning a pre-trained network. This is a standard workflow in adapting existing deep networks to a new task.

We will utilize PyTorch (`https://pytorch.org`) a machine learning library for python.

The provided code builds upon ResNet 50, a state of the art deep network for image classification. ResNet 50 has been designed for ImageNet image classification with 1000 output classes.

The ResNet 50 model has been adapted to solve a (simpler) different task, classifying an image as one of 10 classes on CIFAR10 dataset.

The code `cifar_finetune.py` does the following:

- Constructs a deep network. This network starts with ResNet 50 up to its average pooling layer. Then, a small network with 32 hidden nodes then 10 output nodes (dense connections) is added on top.
- Initializes the weights of the ResNet 50 portion with the parameters from training on ImageNet.
- Performs training on only the new layers using CIFAR10 dataset – all other weights are fixed to their values learned on ImageNet.

The code can be downloaded from Canvas. You can either use Google Colaboratory (`https://colab.research.google.com/`) to run the code or setup virtual environment in your local machine. If you are setting up virtual environment locally, Anaconda (`https://www.anaconda.com`) environment config files with the latest stable release of PyTorch and torchvision are provided for Python 3.8 for Linux and macOs users. You can use the config file to create virtual environments and test your code. To set up the virtual environment, install Anaconda and run the following command

```
conda env create -f CONFIG_FILE.
```

Replace `CONFIG_FILE` with the path to the config file you downloaded. To activate the virtual environment, run the following command

```
source activate ENV_NAME
```

Replacing `ENV_NAME` with `cmpt726-pytorch-python38`.

Windows users please follow the instructions on PyTorch website (`https://pytorch.org`) to install manually. PyTorch only supports Python3 on Windows!

If you wish to download and install PyTorch by yourself, you will need PyTorch (v 0.4.1), torchvision (v 0.2.1), and their dependencies.

What to do:

Start by running the code provided. It will be *very* slow to train since the code runs on a CPU. You can try figuring out how to change the code to train on a GPU if you have a good GPU and want to accelerate training. Do the following tasks:

- Run validation of the model every few training epochs on validation or test set of the dataset and save the model with the best validation error.
- Try applying $L2$ regularization to the coefficients in the small networks we added.

Put your code and a readme file for Question 1 under a separate directory named Q1 in the code.zip file you submit for this assignment. The readme file should describe what you implemented for the questions. It should also include the command to run your code.