ECE194N: Homework 2 (due Feb 20)

Part I: Written Part

Turn in this part during the lecture on Feb 20.

1. Given the input x_i and its label y, we use the following multi layer network and the loss function as a regression problem.

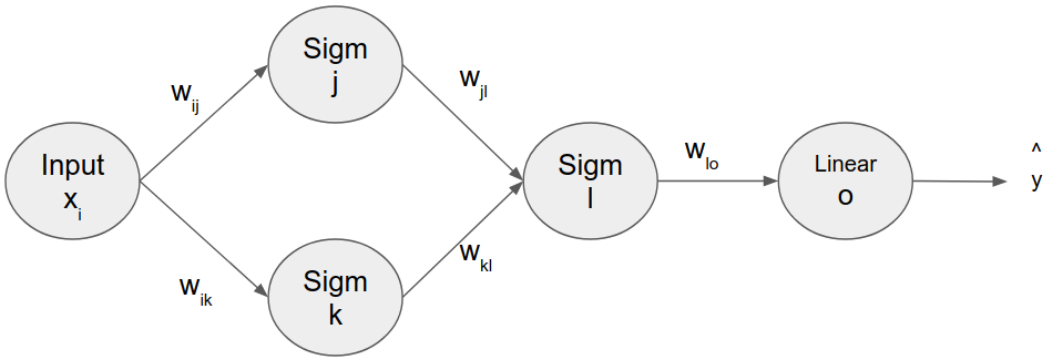

Figure 1: Multi layer network

$$L = \frac{1}{2}(\hat{y} - y)^2 \tag{1}$$

Derive the expressions

- $\frac{\partial L}{\partial w_{ij}}$,
- $\frac{\partial L}{\partial w_{ik}}$,
- $\frac{\partial L}{\partial w_{jl}}$,
- $\frac{\partial L}{\partial w_{kl}}$,
- $\frac{\partial L}{\partial w_{lo}}$

Sigm functions are sigmoid functions: $\sigma(x) = \frac{1}{1+e^{-x}}$.

Linear defines the linear function: $linear(x) = x$

2. If an input image goes through 4 consecutive 3x3 convolutional layers, what is the size of the area in the input image that affects a single pixel in the output feature map? This is called the receptive field of a single pixel in a feature map.

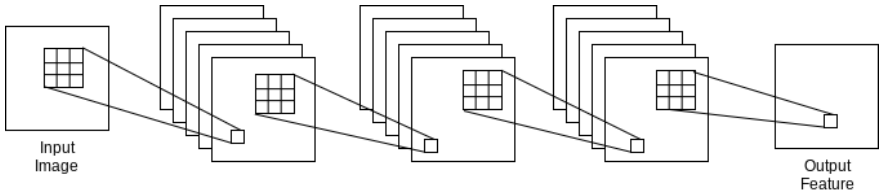

Figure 2: 4 consecutive 3x3 CNNs.

3. Given the image matrix

$$\mathbf{M} = \begin{bmatrix} [0] & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

and the the vertical and horizontal Sobel edge masks, $\mathbf{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & [0] & 2 \\ -1 & 0 & 1 \end{bmatrix}$, and $\mathbf{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & [0] & 0 \\ 1 & 2 & 1 \end{bmatrix}$, and the Laplacian operator $\mathbf{L} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & [4] & -1 \\ 0 & -1 & 0 \end{bmatrix}$. (NOTE: [.] represents the origin of the matrix). To handle the boundaries, pad the image with the boundary values. Since we are convoluting the image with 3×3 filters, we need to pad 2 additional values on each dimension and side. Which will give us:

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & [0] & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

   (a) Compute $\mathbf{A} = \mathbf{S}_x * \mathbf{M}$ and $\mathbf{B} = \mathbf{S}_y * \mathbf{M}$, where $*$ represents the linear convolution operation. Mark the origin of the resulting matrices. Locate the edges using **A** and **B**, and explain your reasoning.

   (b) Compute $\mathbf{C} = \mathbf{L} * \mathbf{M}$, where $*$ represents the linear convolution operation. Mark the origin of the resulting matrix. Locate the edges using **C** and explain your reasoning.

Part II: Programming Part

(Upload the zip file by 5pm, Friday, FEB 22, on Gauchospace).

You will write a report on each of the questions below. Upload your report together with your code as a zip file to Gauchospace. Organize the main folder into three sub-folders and name them as "XOR", "CNN". Name the main folder as "yourlastname_firstname". Include the individual reports for each of the sub-problems in the respective sub-folders. Organize the corresponding code into a subfolder named "code" inside the resepctive sub-folders.

1. **XOR:** Given following samples, we will use multi-layer networks to approximate the functions defined by the samples.

$$\begin{aligned} \boldsymbol{x_1} &= [1,1]^T, \quad y_1 = +1 \\ \boldsymbol{x_2} &= [0,0]^T, \quad y_2 = +1 \\ \boldsymbol{x_3} &= [1,0]^T, \quad y_3 = -1 \\ \boldsymbol{x_4} &= [0,1]^T, \quad y_4 = -1 \end{aligned}$$

   (a) Visualize the samples in a 2 dimensional space. Indicate their classes, y.

   (b) Implement a network to estimate the function that is generating these samples. You need to choose parameters of your model architecture such as number of layers, number of neurons and the loss function. Implement your model in python with Numpy without using Tensorflow. This will require you to derive update rules manually. Comment on how you choose your parameters.

   (c) Visualize the final classification regions on the 2 dimensional space. Hint: Since this will be a multi-layer network, classification region won't be a single line. How can you visualize such regions?

   (d) In real applications, we almost never work with data without noise. Now instead of using the above points generate Gaussian random noise centered on these locations.

$$\begin{aligned} \boldsymbol{x_1} \sim \boldsymbol{\mu_1} = [1,1]^T, \Sigma_1 = \Sigma \quad y_1 = +1 \\ \boldsymbol{x_2} \sim \boldsymbol{\mu_1} = [0,0]^T, \Sigma_2 = \Sigma \quad y_1 = +1 \\ \boldsymbol{x_3} \sim \boldsymbol{\mu_1} = [1,0]^T, \Sigma_3 = \Sigma \quad y_1 = -1 \\ \boldsymbol{x_4} \sim \boldsymbol{\mu_1} = [0,1]^T, \Sigma_4 = \Sigma \quad y_1 = -1 \end{aligned}$$

$$\Sigma = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$$

   Re-train and visualize your classification regions for $\sigma = 0.5, 1, 2$. Comment on your results.

2. In the previous homework, we used K-nearest neighbors to classify images in CIFAR-10 dataset. In this part we will be using the same dataset but use CNNs to classify the model using Tensorflow. Dataset is located at `https://www.cs.toronto.edu/~kriz/cifar.html`.

   (a) Visualize one sample image for each of the 10 classes.

   (b) Design a model to classify these 10 classes. Choose parameters of your model architecture such as number of layers, number of neurons and the loss function. Comment on these choices. Implement this model in Tensorflow and train the model weights using the training set. Plot the training and testing loss and accuracy over time and observe the network's improvement with every iteration.

   (c) Calculate the total number of free, trainable parameters in your model, i.e., total number of weights, biases.

   (d) Visualize one wrongly-classified sample from each of the 10 classes. Choose these samples from the test set.