EECE-571T Assignment 2

Release date: Friday Feb. 26th, 2021 **Deadline:** Thursday Mar. 25th, 2021, 11:59 PM

Submissions will be made through Canvas as PDF attachments of your answers to Problems 1-3 and the completed version of notebooks both link to the notebook or the soft copy is acceptable.

Note: it is important to view the digital version of this PDF as it contains important hyperlinks.

We encourage you to type up your answers in LaTeX. To use the LaTeXfile for this assignment as a template, you can go to this Overleaf project.

1 Automatic Differentiation - 25 points

The key to learning in deep neural networks is ability to compute the derivative of a vector function f with respect to the parameters of the deep neural network. Computing such derivatives as closed-form expressions for complex functions f is dificult and computationally expensive. Therefore, instead, deep learning packages define computational graphs and use automatic differentiation algorithms (typically backpropagation) to compute gradients progressively through the network using the chain rule.

Let us define the following vector functions:

$$y_1 = f_1(x_1, x_2) = e^{2x_2} + x_1 \sin\left(3x_2^2\right) \tag{1}$$

$$y_2 = f_2(x_1, x_2) = x_1 x_2 + \sigma(x_2), \tag{2}$$

where $\sigma(.)$ denotes the standard sigmoid function. This is equivalent to a network with two inputs $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and two outputs $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f(x)$ and a set of intermediate layers.

(i) Draw a computational graph. Computational graph should be at the level of elemental mathematical oprations (multiplication, square, sine, etc.) and constants/variables.

Solution:
The computation graph for the vectors is shown in figure 1:

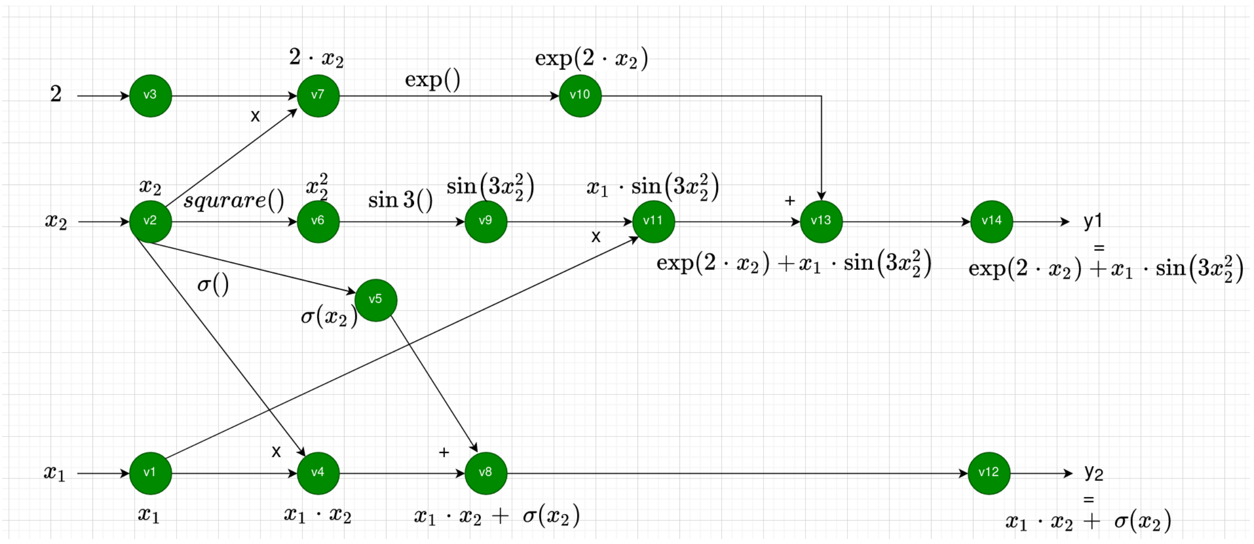

Figure 1: Computational graph

(ii) Draw backpropagation graph, based on computational graph above.

Solution:
The backpropagation graph for the computational graph in figure 2:

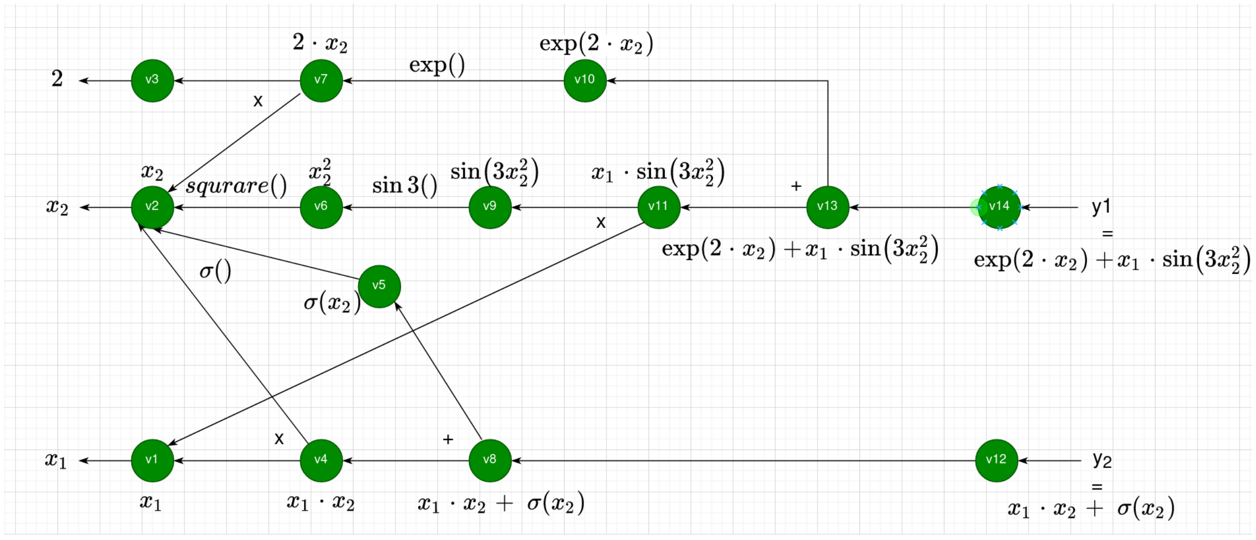

Figure 2: Backpropagation graph

>

(iii) Compute the value of f(x) at $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ using forward propagation. Please do not just plug in numbers directly into Eq. (1) and (2), this is not the intent here.

Solution:

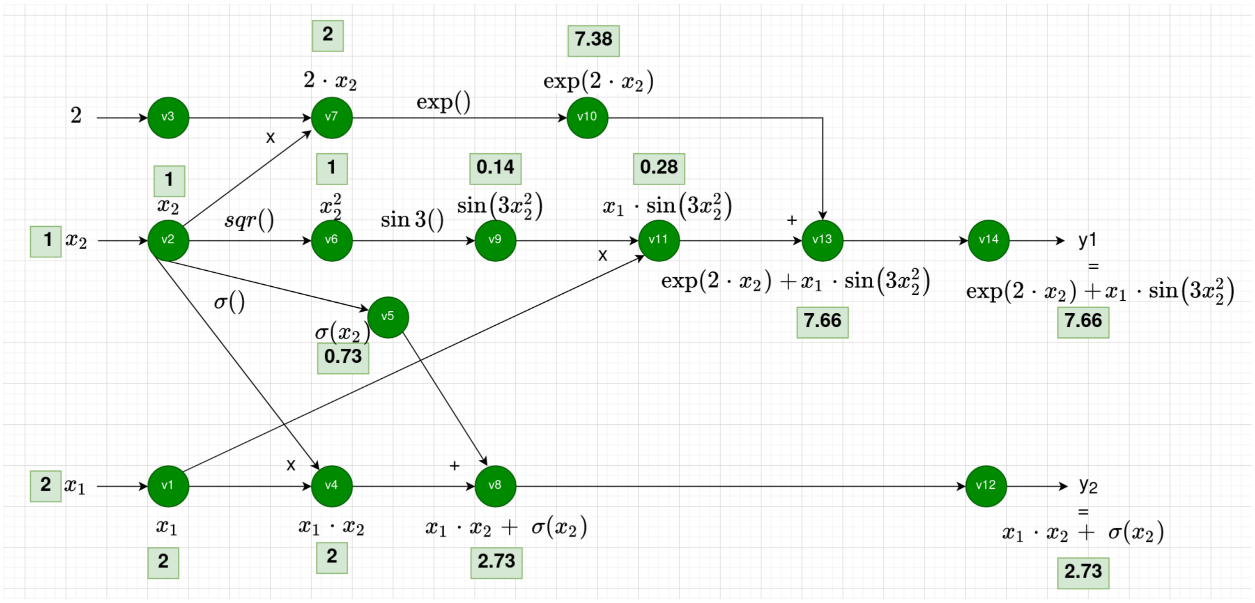

Figure 3: Calculation in computation graph

The value of $f(x) = \begin{pmatrix} 7.66 \\ 2.73 \end{pmatrix}$

(iv) At $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, compute Jacobian using forward mode auto-differentiation. Show all steps.

(v) At $x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$, compute Jacobian using backward mode auto-differentiation. Show all steps.

$$\frac{\partial v_1}{\partial x_2} = 0$$
$$\frac{\partial v_2}{\partial x_2} = 1$$
$$\frac{\partial v_3}{\partial x_2} = 0$$
$$\frac{\partial v_4}{\partial x_2} = \frac{\partial (v_2 \cdot v_1)}{\partial x_2} = v_1 \cdot \frac{\partial v_2}{\partial x_2} + v_2 \cdot \frac{\partial v_1}{\partial x_2} = 0 \cdot 1 + 1 \cdot 0 = 0$$
$$\frac{\partial v_5}{\partial x_2} = \frac{\partial v_5}{\partial v_2} \cdot \frac{\partial v_2}{\partial x_2} = \frac{\sigma(v_2)}{1 - \sigma(v_2)} \cdot 1 = \frac{0.73}{1 - 0.73} = 2.70$$
$$\frac{\partial v_6}{\partial x_2} = \frac{\partial v_6}{\partial v_2} \cdot \frac{\partial v_2}{\partial x_2} = 2v_2 \cdot 1 = 2 \cdot 1 \cdot 1 = 2$$
$$\frac{\partial v_7}{\partial x_2} = \frac{\partial v_3 \cdot v_2}{\partial x_2} = v_2 \cdot \frac{\partial v_3}{\partial x_2} + v_3 \cdot \frac{\partial v_2}{\partial x_2} = 1 \cdot 0 + 2 \cdot 1 = 2$$
$$\frac{\partial v_8}{\partial x_2} = \frac{\partial (v_4 + v_5)}{\partial x_2} = 0 + 2.70 = 2.70$$
$$\frac{\partial v_9}{\partial x_2} = \frac{\partial v_9}{\partial v_6} \cdot \frac{\partial v_6}{\partial x_2} = 3 \cdot \cos(3v_6) \cdot 2 = 6 \cdot (-0.98) = -5.88$$
$$\frac{\partial v_{10}}{\partial x_2} = \frac{\partial v_{10}}{\partial v_7} \cdot \frac{\partial v_7}{\partial x_2} = \exp(v_7) \cdot 2 = 15.76$$
$$\frac{\partial v_{11}}{\partial x_2} = \frac{\partial (v_1 \cdot v_9)}{\partial x_2} = v_9 \cdot \frac{\partial v_1}{\partial x_2} + v_1 \cdot \frac{\partial v_9}{\partial x_2} = 0.14 \cdot 0 + 2 \cdot (-5.88) = -11.76$$
$$\frac{\partial v_{12}}{\partial x_2} = \frac{\partial v_8}{\partial x_2} = 2.70$$
$$\frac{\partial v_{13}}{\partial x_2} = \frac{\partial (v_{10} + v_1 1)}{\partial x_2} = 15.76$$
$$\frac{\partial v_{14}}{\partial x_2} = \frac{\partial v_{13}}{\partial x_2} = 15.76$$

Note: The goal here is not the final result, it could be obtained easily by any symbolic differentiation package (Mathematica, Matlab, etc.) and you will loose points if this is what you do. The goal is to show that you understand how backpropagation actually works and can do this by hand. This will be a bit painful, but in fact that is part of the point: ability of AutoDiff algorithms to do this automatically is a huge benefit. Use the notation of this example.

2 Convolutional Neural Network - 30 points

In this assignment we will be using the MNIST digit dataset. The dataset contains images of hand-written digits ($0 - 9$), and the corresponding labels. The images have a resolution of 28×28 pixels. Please go to the COLAB notebook here, and create a new copy on your drive. In this notebook we will train a simple convolutional neural networks (CNNs) for classifying hand-written digit images from MNIST data set. Follow the steps and fill in the blanks with your code to build, train and test a CNN model on MNIST. **(10 point)**

Give short answers to the following questions:

(i) Using MINIST dataset as input (input size 28×28), what is the output size after applying convolution operation with filter size 3×3, padding 2, and stride 1? **(5 points)**

(ii) Why do we have max-pooling in classification CNNs?**(5 points)**

(iii) Why do we use convolutions for images rather than just FC layers?**(5 points)**

(iv) What is the difference between Sigmoid and Softmax activation functions?**(5 points)**

3 Recurrent Models - 45 points

Here we use the same dataset as in question 2, but we will be using this data a bit differently this time around. Please go to the COLAB notebook here, and create a new copy on your drive. Since this assignment will be focusing on recurrent networks that model sequential data, we will be looking at each image as a sequence: the networks you train will be "reading" the image one row at a time, from top to bottom (we could even do pixel-by-pixel, but in the interest of time we'll do row-by-row which is faster). Also, we will work with a binarized version of MNIST – we constrain the values of the pixels to be either 0 or 1. You can do this by applying the method binarize, to the raw images.

There are various ways and tasks for which we can use recurrent models. A depiction of the most common scenarios is available in the Figure below. In this assignment we will look at many-to-one (sequence to label/decision). You can use this to solve the classification task.

3.1 Understanding LSTM vs GRU - 15 points

Before going deeper into your practical tasks, take some time to revise and make sure you understand the two major types of recurrent cells you will be using in this assignment: Long-Short Term Memory Units (LSTM) first introduced by Hochreiter and Schmidhuber [1997] and the more recent Gated Recurrent Units (GRU) by Cho et al. [2014]. Once you have done this, answer the following questions:

(i) Explain the vanishing gradient issue in RNNs and why it is important.**(5 points)**

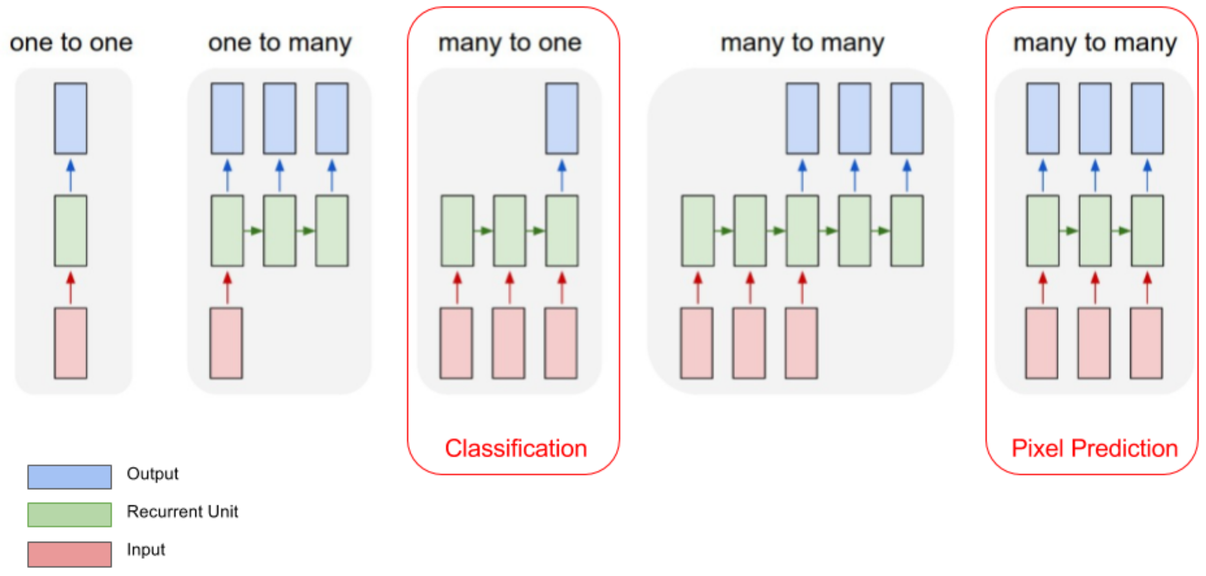

Figure 4: You will be implementing variants of many-to-one for classification

(ii) Explain the importance of gates in LSTM and GRU.**(5 points)**

(iii) Explain the difference between LSTM and GRU.**(5 points)**

3.2 Implementation - 30 points

In this part you will train a number of many-to-one recurrent models that takes as input: an image (or part of an image) as a sequence (row by row) and after the last input row produces, as output, a probability distribution over the 10 possible labels $(0 - 9)$. The models will be trained using a cross-entropy loss function over these output probabilities. Use the Adam optimizer (with default settings other than the learning rate) for training. [Optional] Sometimes dropout has been shown to be beneficial in training recurrent models, so feel free to use it or any other form of regularization that seems to improve performance. It might be also worth trying out batch-normalization.

Models: Your models will have the following structure:

- The input (current binarised row of pixels) can be fed directly into the recurrent connection without much further pre-processing. The only thing you need to do is have an affine transformation to match the dimensionality of the recurrent unit, i.e. one of $(32, 64)$.

- The output (probabilities over the 10 classes) is produced by looking at the last output of the recurrent units, transforming them via an affine transformation.
- For the recurrent part of the network, please implement and compare the following architectures:

  (i) LSTM with 32, 64 units. **(10 points)**

  (ii) GRU with 32, 64 units. **(10 points)**

  (iii) stacked LSTM: 3 recurrent layers with 32 units each. **(5 points)**

  (iv) stacked GRU: 3 recurrent layers with 32 units each. **(5 points)**

Your network should look like:
Input $\Rightarrow$ RNN cell $\Rightarrow$ Relu $\Rightarrow$ Fully connected $\Rightarrow$ Relu $\Rightarrow$ Fully connected $\Rightarrow$ Output

For all cases train the model with these hyper-parameter settings epochs=10, learning rate=0.001, batch size=256, fully connected hidden units=64

With these hyper-parameters you should be comfortably above 95% test set accuracy on all tasks. (Feel free to try other settings, there are certainly better choices, but please report the results with these exact hyper-parameters).

Please report the cross-entropy and the classification accuracy for the test set of the models trained. Your comparison should also include the computation time, and difficulty of implementation.