Due Date: October 21th, 2023 at 11:00 pm

The use of AI tools like Chat-GPT to find answers or parts of answers for any question in this assignment is not allowed. However, you can use these tools to improve the quality of your writing, like fixing grammar or making it more understandable. If you do use these tools, you must clearly explain how you used them and which questions or parts of questions you applied them to. Failing to do so or using these tools to find answers or parts of answers may result in your work being completely rejected, which means you'll receive a score of 0 for the entire practical section.

- *This assignment is intensive, start well ahead of time !*
- *For all questions, show your work.*
- *The practical part has been provided to you as a Jupyter Notebook on Google Colab. You must fill in your answers in the notebook and export the notebook as a Python file named* `solution.py` *(File→Download→Download .py) and submit it via Gradescope for autograding.*
- *You must also submit your report (PDF) on Gradescope. Your report must contain answers to Questions 1.1 d), 1.2 j) k) l), and 3.5 to 3.9. You do not need to submit code for these questions, just the report will suffice.*
- *TAs for this assignment are* ***Shuo Zhang and Thomas Jiralerspong****.*

Link to Notebook: Click here.

Question 1 (30)**. (Implementing Perceptron and MLP with NumPy)**

(1.1) Implement a perceptron, follow the provided Colab Notebook for instructions. (5)

The following parts will be automatically graded:

(a) Implement the **relu** activation function, which is defined as $\max(x, 0)$ for any input vector x. (0.5)

(b) Implement the **fire** function, which determines whether the perceptron is activated/fired given a stimulus x. (1)

(c) Implement the **train** function. It takes an input vector x, a label y, and a learning rate lr. The train function then performs **one** iteration of backpropagation. It calculates the gradients of the loss w.r.t. weights w and biases b, then updates them using the given learning rate lr. (2.5)

The following parts should be submitted in the PDF report and will be graded manually:

(d) Using the provided data and graph function, train the perceptron for 3 epochs using the learning rate of 0.001. Show your results (decision boundary graph) in the PDF report. (1)

(1.2) Implement MLP, follow the provided Colab Notebook for instructions. (25)

The following parts will be automatically graded:

(a) Implement the function **init_parameters**. This initializes the weights w of the MLP with values sampled from a uniform distribution $\mathcal{U}\left[-\frac{1}{\sqrt{h_0}}, \frac{1}{\sqrt{h_0}}\right]$, where h_0 is the dimension of the input embedding. Then, initialize the biases b of the MLP with zeros. (1)

(b) Implement the function **gradient_activation_fn**. In this function, you need to implement the gradient computation functions for the activation functions relu, sigmoid and tanh respectively. (1)

(c) Implement the function **softmax**. Make sure to avoid numerical instability using the translation invariance property of softmax: $\text{softmax}(x) = \text{softmax}(x + c)$, where c is constant. (1)

(d) Implement the function **layer_forward**. In this function, you will do forward propagation for **one** layer. This function will then be called in a *for* loop across all layers. Remember to treat the output layer differently in order to have probabilities as output. Note that this function stores its outputs to be used later during backpropagation. (3)

(e) Implement the function **forward**. In this function, you will apply the *layer_forward* function implemented above across all layers. (2)

(f) Implement the function **cross_entropy_loss**. In this function, you will implement the cross-entropy loss between model predictions and true labels. Cross-entropy loss is the primary loss for classification. (2)

(g) Implement the function **layer_backward**. In this function, you will implement the backpropagation algorithm for **one** layer. This function will then be called in a *for* loop across all layers from the output layer till the first layer. The necessary information a, a_{prev} and w have been coded for you, use them to calculate the gradients. a is the postactivation vector of the current layer, a_{prev} is the postactivation vector of the previous layer, w is the weight matrix of the current layer. The output layer has been treated differently, you only need to add the code for the remaining layers. Note that this function only calculates the gradients and does not update weights and biases. (5)

(h) Implement the function **backward**. In this function, you will take the gradients calculated with *layer_backward* above and update the model weights and biases. (2)

(i) Implement the function **one_hot_encode**. In this function, you will convert an input array of class indices to their corresponding one-hot vectors. For example, for the input array [0, 2, 1], where 0, 2 and 1 are class labels for each of the three input sample, the output should be [[1, 0, 0], [0, 0, 1], [0, 1, 0]]. For 0, only the 0th entry is 1, other entries are 0 ; for 2, the 2nd entry is 1, others are 0 ; for 1, the 1st entry is 1, others are 0. (2)

The following parts should be submitted in the PDF report and will be graded manually:

(j) Train the model using the provided MNIST data for 1 epoch with the following batch sizes: 16, 32, 64, 128. Use the default learning rate and activation function. Take the graphs produced by the train function and include them in your report. What is your observation ? Explain briefly in your report. (2)

(k) Train the model for 1 epoch with the following learning rates: 0.1, 0.01, 0.001, 0.0001. Use the default batch size and activation function. Take the graphs produced by the train function and include them in your report. What is your observation ? Explain briefly in your report. (2)

(l) Train the model for 1 epoch with the following activation functions: relu, sigmoid and tanh. Use the default learning rate and batch size. Take the graphs produced by the train

function and include them in your report. What is your observation ? Explain briefly in your report. (2)

Question 2 (20)**. (Implementing CNN layers with NumPy)**

Convolutional neural networks (CNN) differ from MLPs mainly because of two components: a convolution layer and a pooling layer. In this exercise, we have a convolution layer of filter size (3×3) and stride 1 consisting of 32 units, and a max-pooling layer of filter size (2×2). The inputs to the convolution layer are each of size $(1 \times 128 \times 128)$ (single channel) while the inputs to the max-pooling layer is the size of the convolution layer's outputs. You will code up the forward and backward passes for these two layers using NumPy. You may refer to these slides for details.

(2.1) Write the **_get_filtersizeXfiltersize_views** function for the 2d convolution layer. This function traverses its input matrix in steps of the given stride, and returns the views formed by the given filter/channel size at each step. (2)

(2.2) Write the **forward** function for the convolution layer. The input to **forward** is a *np.ndarray* of size (batch size $\times 1 \times 128 \times 128$) (we consider only 1 color channel here), and the output is the result of the convolution operation. (2)

(2.3) Write the **backward** function for the convolution layer. We assume that the input to this function is the derivative of the loss w.r.t. the output of the layer. The output of the function is the gradient of the loss with respect to the parameters of the layer. (8)

(2.4) Write the **_get_filtersizeXfiltersize_views** function for the max-pooling layer. (1)

(2.5) Write the **forward** function for the max-pooling layer. (1)

(2.6) Write the **backward** function for the max-pooling layer. We assume that the input to this function is the derivative of the loss w.r.t. the output of the layer. The output of the function is the gradient of the loss w.r.t. the input to the layer. (6)

Question 3 (50)**. (Implementation and experimentation)** An important skill for any deep learning practitioner is being able to implement a state-of-the-art architecture yourself using a deep learning library.

In this section, you will implement one of the most influential CNN architectures of all time (VGG16) using PyTorch. You will then perform several experiments with it.

(3.1) Read the first 4 pages of the original VGG16 paper.

(3.2) Use the information found in the paper to complete the code for the VGG16 model in the Colab notebook. Use the model in column D of the table on page 3. Note that since CIFAR-10 only has 10 classes compared to the 1000 classes used in the paper, the final fully connected layer will have an output dimension of 10 instead of 1000. Also, add 2D batch normalization after every convolutional layer. Do not add dropout. Don't forget the final softmax ! Do not make any changes to the data pre-processing pipeline ! Once you have finished implementing the VGG16 model, test your implementation using the autograder on Gradescope (10)

The following parts should be submitted in the PDF report and will be graded manually:

3. Write the `train_loop` function and train the model on the CIFAR-10 dataset. Also complete `valid_loop`, which is used to validate the model's performance. Perform validation after every training epoch. Each epoch should take around 15 minutes on a T4 GPU on Google Colab. You may stop training when the validation loss starts to consistently go up (early stopping)

4. Design your own architecture from scratch, implement it using PyTorch and train it with the same hyperparameters as the VGG16 model. You **may not** use an existing architecture which can be found online.
5. Provide an explanation of why you made the choices you did for your own architecture. (10)
6. Plot curves of the training loss, training accuracy, validation loss and validation accuracy across epochs for both models and include it in your report. Use ReLU activation and `xavier_normal` initialization) for the VGG16 model. (5)
7. Provide an analysis of the results and why you think one architecture performed better or worst than the other. (10)
8. Visualize a few filters from the first and last convolution layers in the trained model (helper methods have been provided). Include the images in your report and comment on the features learnt by the first and last convolution layers. How do they differ ? (5)
9. Visualize the feature maps from the same filters for the first and last convolution layers in the trained model for the image provided in the variable `vis_image` in the Colab notebook. Include the images in your report. Comment on the features from the input detected by specific filters that could help classify the image. (5)
10. Construct an MLP with approximately equal parameters as VGG16 and train it on the CIFAR-10 dataset. Include a table in your report comparing the MLP and CNN – list the number of parameters, validation accuracy, validation loss and training time for the same number of epochs (25 epochs). Which model is better and why do you think so ? (5)
11. Submit a link to your Google Colab notebook in your PDF report (at the top) as well, make sure to make it publicly viewable !