Due Date: October 17th, 2022 at 11:00 pm

Instructions

- *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 3.4, 3.5, 3.6 and 3.7. You do not need to submit code for these questions, just the report will suffice.*
- *TAs for this assignment are* ***Nanda Harishankar Krishna and Sarthak Mittal****.*

Link to Notebook: Click here.

Question 1 (30)**. (Implementing MLPs with NumPy)**

Consider an MLP with two hidden layers with $h^1 = 128$ and $h^2 = 64$ hidden units. The number of features of the input data is $h^0 = 784$. The output of the neural network is parameterized by a softmax of $h^3 = 10$ classes. Create an MLP with these specifications (do not forget to include biases!). Implement the forward and backward propagation of the MLP in NumPy without using any of the deep learning frameworks that provide automatic differentiation. Also write code to compute the *crossentropy loss*, which is used as the optimization criterion. Finally, write a function that updates the parameters of the network. Note that you do not need to train the model for this question – simply write code for the functions mentioned.

The following parts will be automatically graded:

(1.1) Write the function `initialize_weights`. This initializes the weights of the MLP with values sampled from a uniform distribution $\mathcal{U}\left[-\frac{1}{\sqrt{h_0}}, \frac{1}{\sqrt{h_0}}\right]$. Note that we have already assigned 0s to the biases, you do not need to modify this.

(1.2) Write all of the given activation functions, i.e. `relu`, `tanh`, `sigmoid` and `softmax`, and the selector function `activation` that selects the activation function specified by the input string (`activation_str`). Note that if the argument `grad` is `True`, we expect the function to return the gradient of the activation function with respect to its input.

(1.3) Write the `forward` pass function, which takes in an input to the MLP and returns a dictionary `cache` containing pre-activations (`Z`s) and activations (`A`s) for every layer. Note that you will use the weights and biases initialised earlier for these computations.

(1.4) Write the `loss` function (crossentropy for multi-class classification).

(1.5) Write the `backward` pass function. We assume that the input to this function is the true label associated with the input example considered in the `forward` pass. You may calculate

the gradient of the loss with respect to the output layer assuming that the loss is multi-class crossentropy. The function must return a dictionary `grads` containing the gradient with respect to the activations at each layer and the parameters (weights and biases).

(1.6) Write the `update` function that takes in the gradient to update the parameters.

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

Consider two layers, a convolution layer of filter size (3×3) and stride 1 consisting of 64 units, and a max-pooling layer of filter size (2×2). The inputs to the convolution layer are each of size $(1 \times 32 \times 32)$ (single channel) while the input to the max-pooling layer is the size of the convolution layer's output. 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 `initialize_weights` and `forward` functions for the convolution layer. The input to `forward` is a matrix of size $(n \times 1 \times 32 \times 32)$ (i.e. n single channel images of size (32×32)) and the output is the result of the convolution operation.

(2.2) Write the `backward` pass function for the convolution layer. We assume that the input to this function is the derivative of the loss with respect to the output of the layer. The output of the function is the gradient of the loss with respect to the parameters of the layer.

(2.3) Write the `forward` pass function for the max-pooling layer.

(2.4) Write the `backward` pass function for the max-pooling layer. We assume that the input to this function is the derivative of the loss with respect to the output of the layer. The output of the function is the gradient of the loss with respect to the input to the layer.

Question 3 (50)**. (Implementing a CNN and comparison with MLPs using PyTorch)**

In this question you will use PyTorch to implement a ResNet-18 right up from its component blocks and train it to classify images from the CIFAR-10 dataset. The architecture has been slightly modified to better suit the CIFAR-10 dataset. You will visualize the filters in the CNN and feature maps for an image, and compare the CNN with an MLP in terms of performance and training time.

(3.1) Complete the `__init__`, `activation`, `residual` and `forward` methods in the `ResidualBlock` class. You may refer to the figure below to implement this.

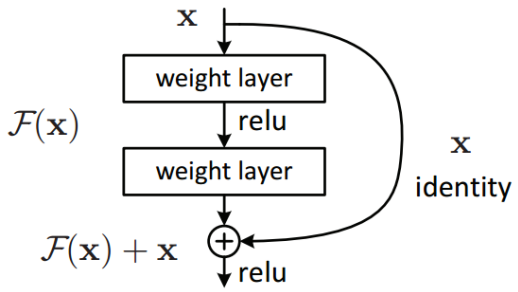

FIGURE 1 – Residual Block (source: "Deep Residual Learning for Image Recognition", He at al., CVPR 2016)

(3.2) Complete the `__init__`, `_create_layer` and `forward` methods in the `ResNet18` class. The input to the `forward` method is a batch of inputs to the network. The figure below shows the architecture of ResNet-18.

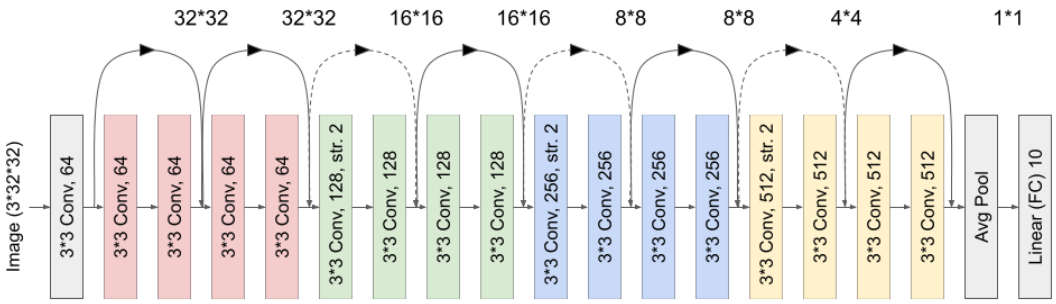

FIGURE 2 – ResNet-18 Architecture

Implementation details:

- Dotted lines represent residual connections involving downsampling. In such cases, the output of `residual` is not the identity, but the result of a convolution with kernel size (1×1), stride 2, `padding=0` and `bias=False`.
- The numbers mentioned on the top are the height and width of the feature maps after being passed through each block.
- Use `padding=1` for all convolutions except the downsampling convolution in `residual` which does not need padding (i.e. `padding=0`). Also, `bias=False` for all convolution layers.
- You must include a `BatchNorm2d` after *every* convolution and before the activation.
- The stride for convolution layers is 1 unless otherwise specified (see "str. 2" in the figure, this means the stride is 2 for those specific convolution layers).
- You need not apply softmax at the output layer as `nn.CrossEntropyLoss` takes care of this implicitly.

(3.3) 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.

(3.4) Plot curves of the training loss, training accuracy, validation loss and validation accuracy across epochs and include it in your report (ReLU activation, `xavier_normal` initialization).

(3.5) 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 ?

(3.6) 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`. Include the images in your

report. Comment on the features from the input detected by specific filters that could help classify the image.

(3.7) Consider the activation functions: (i) ReLU and (ii) tanh, and the weight initialization schemes: (i) Glorot uniform (`init.xavier_uniform_`), (ii) Glorot normal (`init.xavier_normal_`) and (ii) He normal (`init.kaiming_normal_`). Perform a grid search over these hyperparameters. Report the set of hyperparameters that give the highest validation accuracy, and also provide the values of the validation accuracy and validation loss.

(3.8) Construct an MLP with approximately equal parameters as the ResNet-18 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). Which model is better and why do you think so ?