1 Outline

In this assignment, you are asked to design MNIST classifier using convolutional neural networks.

2 Specification

In this assignment, you are asked to write a Python code to implement a neural network classifier for MNIST dataset. Specifically, complete three methods, `forward`**,** `backprop` **and** `update_weights` **for class** `nn_mnist_classifier`**. You also need to complete all the layers in nn_layers.py file.** The implementation details are provided in the following sections.

3 Convolutional Neural Network (CNN) for MNIST

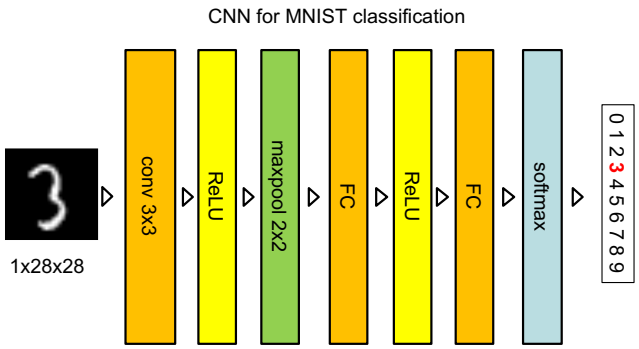

Figure 1: CNN architecture.

3.1 Network Architecture

MNIST dataset is a database of images of handwritten digits. There are handwritten numbers of 0 to 9. The dataset contains 60,000 training images and 10,000 test images. The goal of a classifier is to classify the digits correctly to numbers 0 to 9. So this is a classification problem of 10 classes (0 to 9). The size of single image is 28-by-28 pixels. The image is grayscale, so there is only one color channel. Therefore, an image has dimension 1x28x28.

The proposed CNN is given by Fig. 1. It consists of the following layers in order:

1. Convolutional layer: takes a batch of MNIST images as input. Input x will have shape

   `x.shape=(batch_size, input_channel_size, in_width, in_height)`

   where `input_channel_size`=1, `in_width=in_height`=28. The convolutional layer has 28 filters. Thus the output `y` will have shape

   `y.shape=(batch_size, num_filters, out_width, out_height)`

   where `num_filters`=28, `out_width=out_height`=26.

   This layer is an object of class `nn_convolutional_layer`

2. Activation layer: ReLU activation is used.

   This layer is an object of class `nn_activation_layer`

3. Maxpool layer: A 2x2 maxpooling is done with stride 2. Thus the output `y` will have shape

   `y.shape=(batch_size, 28, 13, 13)`

   This layer is an object of class `nn_max_pooling_layer`

4. Fully Connected (FC) layer: A linear layer producing linear scores, $y = Wx+b$, where the output channel size is 128. As input, the layer takes $(batch\_size, 28, 13, 13)$. Note however, for each input in the batch, FC layer treats each input whose 3-D volume is $28 \times 13 \times 13$ as a 'flat' vector. Thus, FC layer will regard the input as $(batch\_size, 28 * 13 * 13)$, take inner product with each flattened vector, and produce output `y` with shape

   `y.shape=(batch_size, 128)`

   This layer is an object of class `nn_fc_layer`

5. Activation layer: ReLU activation is used.

   This layer is an object of class `nn_activation_layer`

6. Fully Connected (FC) layer: Another FC layer, takes input size of 128, and produces output size of 10. These 10 numbers represent the linear scores for each category, that is, from number 0 to 9. The output `y` has shape

   `y.shape=(batch_size, 10)`

   This layer is an object of class `nn_fc_layer`

7. softmax layer: Softmax output, representing the probability scores for 10 categories, that is, from number 0 to 9. The output `y` has shape

   `y.shape=(batch_size, 10)`

   This layer is an object of class `nn_softmax_layer`

8. The loss function is the cross-entropy loss. `nn_cross_entropy_layer` is added at the end of the CNN.

3.2 class `nn_mnist_classifier`

The class have three methods to be implemented: `forward`, `backprop` and `update_weights`.

3.2.1 `forward` method

This method performs the forward pass of a batch of MNIST images; that is, the batch is passed through the CNN. The method returns the scores for the batch, and the average cross-entropy loss over the batch.

`nn_mnist_classifier.forward(x,y,backprop_req)`: Performs a forward pass of the classifier. Input parameters are as follows:

Parameters:

- x: *nd-array*.

  4-dimensional `numpy.array` input map to the convolutional layer. Each dimension of x has meaning

  ```
  x.shape=(batch_size, input_channel_size, in_width, in_height)
  ```

  For example, if `x.shape` returns $(16, 3, 32, 32)$, it means that `x` is an image/activation map of size 32×32, with three input channels (like RGB), and there are 16 of them treated as one batch (as in mini-batch SGD).

- `y`: *nd-array*

  Ground Truth (GT) labels for input image batch `x`.

  ```
  y.shape=(batch_size,)
  ```

- `backprop_req`: *boolean*

  Set this to `True` if backprop method is to be called afterwards. This will store outputs from the intermediate layers, which will be used to computing backpropagation. Set this to `False` if only forward pass (inference) is needed.

Returns: two values `score, loss`.

- `score`: *ndarray*

  2-dimensional `numpy.array` with the following shape.

  ```
  score.shape=(batch_size, 10)
  ```

  Softmax output containing scores for 10 categories (numbers from 0 to 9) for the batch.

- `loss`: *float64*

  Cross-entropy loss.

3.2.2 `backprop` method

`nn_mnist_classifier.backprop()`: Performs a backpropagation of the CNN. The method performs backprop calculation with the saved outputs from intermediate layers. The intermediate outputs are saved when `backprop_req` flag is set `True` at the time of calling `forward` method.

3.2.3 `update_weights` method

`nn_mnist_classifier.update_weights()`: Update weights according to momentum method. The velocity or momentum parameters are defined, stored and updated within the class object.

3.2.4 `__init__` method

Initialization method for the class.
Parameters:

- `mmt_friction`: *float64*.

  The friction or α parameter for momentum method. By default, this is set to 0.9.

- `lr`: *float64*.

  learning rate.

3.3 class `nn_fc_layer` (in `nn_layer.py` module)

(Note: this layer is almost identical to the previous `nn_linear_layer`, except some minor differences in initializations, and some dimensions of parameters and inputs. So you can re-use your previous implementation as much as you like.)

This method performs the forward pass of fully connected layer. For given input x, the forward layer computes

$$y = Wx + b$$

where W and b are learnable parameters of this layer. For (input_size)-dimensional input vector x, outputs (output_size)-dimensional vector. x can come in batches, so the shape of y is (batch_size, output_size). W has shape (output_size, input_size), and b has shape (output_size,)

The class have two methods to be implemented: `forward` and `backprop`.

3.3.1 `__init__` method

Initialization method for the class. Initializes W and b such that

```
W.shape=(output_size, input_size)
```

and

```
b.shape=(output_size,)
```

Also performs Xavier-He initialization of weights.

3.3.2 `forward` method

`nn_fc_layer.forward(x)`: Performs a forward pass of linear layer. Input parameters are as follows:
Parameters:

- x: *nd-array*.

  A `numpy.array` input map to the fully connected layer. For example, x may have the following shape:

  ```
  x.shape=(batch_size, input_channel_size, in_width, in_height)
  ```

  Note the axis 0 of x (or $x.shape[0]$) is always batch size. For example, if `x.shape` returns $(16, 3, 32, 32)$, it means that `x` is an image/activation map of size 32×32, with three input channels (like RGB), and there are 16 of them treated as one batch (as in mini-batch SGD).

  However, when you do linear operation $Wx + b$, here x is treated as a 'flattened' vector. For example, x is treated as $32 \times 32 \times 3$-long vector, and there are 16 batches of them. So you may need to do some reshaping in your method.

Returns: output *ndarray*-type y whose shape is `(batch_size,output_size)`.

3.3.3 `backprop` method

: backpropagation. This method takes 2 parameters:

- x: *nd-array* input.
- `dLdy`. This is the upstream gradient coming from the next layer. Specifically, this parameter represents $\frac{\partial L}{\partial y}$ where y denotes the output of this layer. Thus `backprop` need to return the following gradient, which will be passed as the upstream gradient to the previous layer:

  $$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

  We have two trainable parameters W and b. In order to perform gradient descent, the following should be evaluated: for W

  $$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial W}$$

  and for b

  $$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial b}$$

  Returns: output *ndarray*-type `dLdx, dLdW, dLdb`. Always have in mind that, `dLdW` has the same shape as `W`, and `dLdb` as `b`.

4 What to submit

You will be given two files `hw5.py` and nn_layers.py.

1. **You are asked to complete Q1–Q3 parts in `hw5.py`.**
2. **You are asked to complete all the layers in `nn_layers.py`.**

Submission instructions are

- Submit modified Python file `hw5.py` and `nn_layers.py`.
- Upload your files at Blackboard before deadline. (Please submit the file in time, no late submission will be accepted).

5 How your module will be graded

If done correctly, you will be able to achieve accuracy over 90% **within 2 epochs** of training with proper batch size and learning rate. That is, if your CNN gets trained with the whole training dataset (50000 images) for two times, the accuracy of your model will be high enough. **The goal is to achieve over 80% of accuracy under the following condition:**

Before testing, your model will be trained until one of the following two events occur, whichever comes earlier.

1. Your model will be trained for **two** epochs.
2. Your model will be trained for **two hours** under the standard Google Colab compute engine environment.

The reason we put upper limit on the training time is that, if your model uses too many `for` loops in convolutional and maxpool, the training will be very slow. So try to avoid using too many `for` loops: instead, use `view_as_windows` function and `numpy.dot` function properly. Also we have time limits for grading the homework, so if the training of your model takes excessively long, then we have to stop it at some point.

So before submitting your model, please try to train your model on the standard Google Colab environment, and check whether your model can achieve over 80% accuracy under 2 hours of training.

6 Grading

- 20 points if your classifier achieves total accuracy more than 80%.
- 6 points if your classifier achieves total accuracy between 20–80%.
- 2 points if your classifier achieves total accuracy below 20%.

- 0 point if you do not submit the file by deadline.

In the blackboard, you can upload your files as many times as you like, before the deadline. The last uploaded file will be used for grading. After deadline, the submission menu will be closed and you will not be able to make submission.