Report for PyTorch Programming Exercise

Roll No. 2019702014

Question 1: Model Building

Build a multi-layered perceptron (MLP) in Pytorch that inputs that takes the (224x224 RGB) image as input, and predicts the letter (You may need to flatten the image vector first). Your model should be a subclass of nn.Module. Explain your choice of neural network architecture: how many layers your network has? What types of layers does it contain? What about other decisions like use of dropout layers, activation functions, number of channels / hidden units.

Question 2: Training Code

Write code to train your neural network given some training data. Your training code should make it easy to tweak hyperparameters. Make sure that you are checkpointing your models from time to time (the frequency is up to you). Explain your choice of loss function. Ensure that your code runs on GPU.

Question 3: Overfit to a Small Dataset

Part (a): One way to sanity check our neural network model and training code is to check whether the model is capable of overfitting a small dataset. Construct a small dataset (e.g. 1-2 image per class). Then show that your model and training code is capable of overfitting on that small dataset. You should be able to obtain a 100% training accuracy on that small dataset relatively quickly.

If your model cannot overfit the small dataset quickly, then there is a bug in either your model code and/or your training code. Fix the issues before you proceed to the next step.

Part (b): Once you are done with the above part, try to reduce the effect of overfitting by using techniques discussed in the previous lecture.

Solution:

The solution to the above problem is obtained by sub-dividing it in sequential goal and trying out various hyper-parameters and after every goal completion the inference for the trials is made:

Goal 1: To obtain 100% training accuracy for input of size 224x224x3 for a small subset of 20 images having at least one sample of each of the 9 classes

Choice I:

- Architecture chosen has 2 hidden layers with following structure [224x224x3→500→50→9]
- Learning rate=0.01
- Batch size=20 (Hence, using Batch GD)
- Activation Function ReLu for hidden units; no activation for ouput layer

Result: training accuracy only 25%

Choice II:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→5000→500→9]
- Learning rate=0.1
- Batch size=10 (Hence, using mini-batch GD)
- Activation Function ReLu for hidden units; no activation for ouput layer

Result: training accuracy only 25%

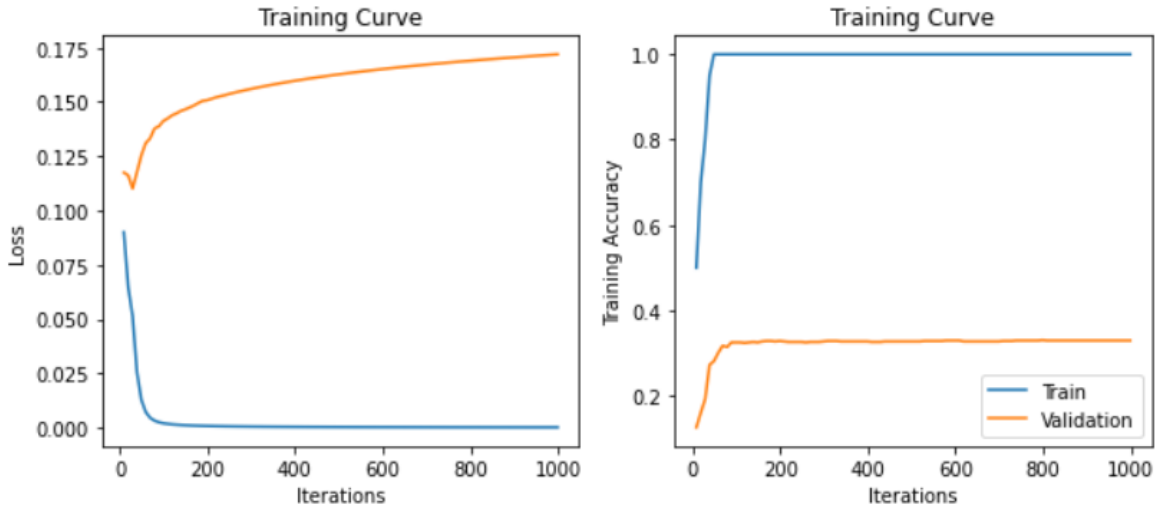
Choice III:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Learning rate=0.01
- Batch size=20 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer

Result: Training accuracy only 100%, Validation accuracy: 32.9% →Goal 1 achieved

So, architecture is finalized for the rest of the solution.

Inference: In Relu if the outputs of any layer are small then are remaining small as there is not much data introduce large changes and the NN is learning only in the region close to the initialization. While tanh is symmetrically distributing the output of hidden layers giving the accuracy of 100%




```
Final Training Accuracy: 1.0
Final Validation Accuracy: 0.32967032967032966
```

Goal 2: To reduce overfitting using dropout for the above model

Choice I:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Learning rate=0.01
- Batch size=20 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 50% on input layer and all hidden layers

Result: Training accuracy 100% Validation accuracy 30%

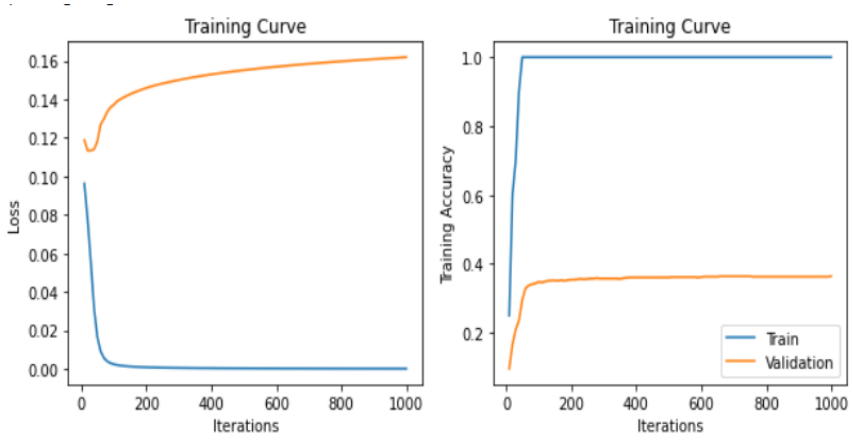
Choice II:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Learning rate=0.01
- Batch size=20 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 90% on input layer and 60% on first hidden layer and 30% on second hidden layer

Result: Training accuracy 100% Validation accuracy 36% → Goal 2 achieved

So, dropout is finalized for the rest of the solution.

Inference: As we are working on image data using fully connected layer, the method is not efficient because only neighboring pixels are related to each other that's why we use convolution layer for the image data. So equivalently if we use high dropout at input layers so the input features will be trained mode independently and as we propagate forward the features are more related so we decrease the amount of dropout and hence there is 3% increase in accuracy at such low amount of data.




```
Final Training Accuracy: 1.0
Final Validation Accuracy: 0.3637362637362637
```

Goal 3: To reduce overfitting using regularization for the above model with the given architecture and dropout

Fixed Model:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Learning rate=0.01
- Batch size=20 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 90% on input layer and 60% on first hidden layer and 30% on second hidden layer

Choice I: weight_decay of 0.01 Result: Validation accuracy→33%

Choice II: weight_decay of 0.1 Result: Validation accuracy→33%

Choice III: weight_decay of 0.5 Result: Decreased accuracy of training and validation

Inference: Regularization is not reducing the validation accuracy because the distribution of the training data is not representing the validation data distribution due to dearth of data so first we should perform data augmentation to make the model more general then the regularization of the parameters would help.

Goal 4: To reduce overfitting using Data-Augmentation for the above model with the given architecture and dropout

Choice I:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Learning rate=0.1
- Batch size=20 (Hence, using mini-Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 90% on input layer and 60% on first hidden layer and 30% on second hidden layer
- Data-augmentation was done by generating 400 images from the 20 images by adding random rotation of 20 degree.
- Number of epochs 1000

Result: Training accuracy reduced to 14% and validation accuracy was poorer

Choice II:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Learning rate=0.1
- Batch size=400 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 90% on input layer and 60% on first hidden layer and 30% on second hidden layer

- Data-augmentation was done by generating 400 images from the 20 images by adding random rotation of 20 degree.
- Number of epochs 1000
- Data-augmentation was done by generating 400 images from the 20 images by adding random rotation of 20 degree and random Horizontal flip.

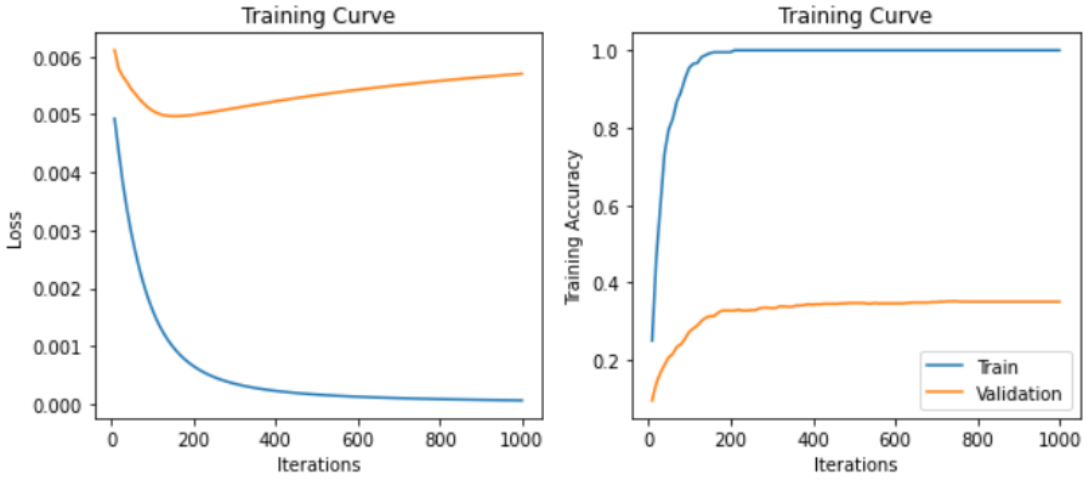
Result: Training accuracy reduced to 14% and validation accuracy was 12%.

Choice III:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Batch size=400 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 90% on input layer and 60% on first hidden layer and 30% on second hidden layer
- Data-augmentation was done by generating 400 images from the 20 images by adding random rotation of 25 degree and applying Gaussian Blurring with kernel of 3x3
- Number of epochs 1000
- Learning rate 0.1→0.001

Result: Training accuracy reduced to 100% and validation accuracy was 35.1%. →Goal 4 achieved

Inference: Proper choice of learning rate helped in achieving 100% training accuracy for a bigger dataset and Gaussian Blurring and Random rotation helped to give new samples from the same previous distribution.




```
Final Training Accuracy: 1.0
Final Validation Accuracy: 0.3505494505494505
```

Goal 5: To reduce overfitting using regularization for the above model trained on the augmented data

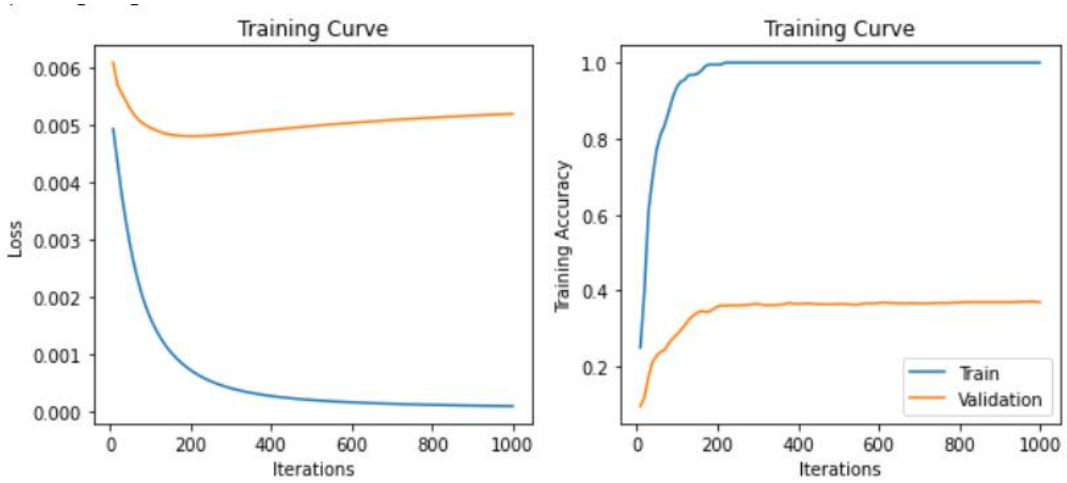
Fixed Model:

- Architecture chosen has 2 hidden layers but with more units per layer with following structure [224x224x3→500→50→9]
- Batch size=400 (Hence, using Batch GD)
- Activation Function tanh for hidden units; no activation for ouput layer
- Dropout of 90% on input layer and 60% on first hidden layer and 30% on second hidden layer
- Data-augmentation was done by generating 400 images from the 20 images by adding random rotation of 25 degree and applying Gaussian Blurring with kernel of 3x3
- Number of epochs 1000
- Learning rate 0.1→0.001

Choice I: Weight_Decay=0.2 Result: Training_accuracy =80% Validation accuracy = 21%

ChoiceII: Weight_Decay=0.01 Result: Training_accuracy =100% Validation accuracy = 37% →Goal 5 achieved

Inference: When Regularization was done after proper amount of same distribution data was present with the network then same regularization parameter helped in achieving greater accuracy



Comments about choices made for the network:

- The loss function was chosen to be Cross-Entropy loss as the data was of Categorical nature and involved multi-class classification. Moreover the class label and classes did not have any numerical significance so MSE loss was not preferred
- Code runs on GPU, hyper parameters can be tweaked and model is check-pointed from time to time

Question 4: Finetuning

For many image classification tasks, it is generally not a good idea to train a very large deep neural network model from scratch due to the enormous compute requirements and lack of sufficient amounts of training data.

In this part, you will use Transfer Learning to extract features from the hand gesture images. Then, train last few classification layers to use these features as input and classify the hand gestures. As you have learned in the previous lecture, you can use AlexNet architecture that is pretrained on 1000-class ImageNet dataset and finetune it for the task of understanding American sign language.

Question 5: Report result

Train your new network, including any hyperparameter tuning. Plot and submit the training and validation loss and accuracy of your best model only. Along with it, also submit the final validation accuracy achieved by your model.

- Last layers of the AlexNet was Fine-tuned with respect to the ASL dataset

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=9, bias=True) → Only this layer was trained rest all layers with the weights of
AlexNet were used
  )
                )
```

- The best model accuracy obtained is 72.88% on the validation set

Some Observations and Learnings :

1) If the training curve has high fluctuation the n increasing the momentum rate reduces the fluctuation
2) If the Loss is not changing at all increasing the learning rate let the model come out of the local minima
3) Data Augmentation if done with high variation may decrease training accuracy but it generalizes well for the validation set
4) As the dropout percentage increases the neurons learn more independently and it also speeds up the training of one epoch.
5) Loss generally decreases with increase in number of epochs but only upto a certain number of epochs and then it becomes constant.

Link for the Collab NB:
https://colab.research.google.com/drive/1RmlxhuFQkEVDwI7tF21EG1dRj6aEfgeI?usp=sharing