EE5178 : Modern Computer Vision

Programming Assignment 1

Instructions

- You are supposed to use Python and Pytorch for this assignment.
- Read the problem fully to understand the whole procedure.
- Comment your code generously and place appropriate titles in all figures
- Post any doubts you have on moodle discussion threads. This will be helpful to your peers as well.
- Submit the codes (the actual notebook with all the cells run and the outputs being displayed and a PDF version) and your assignment report in a **single zip file** titled PA1_RollNumber.zip in the submission link provided on moodle.

Preliminaries

The aim of this assignment is to experiment Multilayer Feedforward Neural Network (MLP) and Convolutional Neural Networks (CNN) for image classification. Please use Pytorch, an open source library for implementing deep learning models, to write your code. You can use any other libraries that you may feel are necessary.
Dataset: The aim of this assignment is to implement an image classifier on the popular CIFAR-10 dataset. This dataset contains 60000 32×32 color images in 10 different classes, with 6000 images per class. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.
Note: Pytorch provides CIFAR-10 dataset in the torchvision.datasets module, which you can use directly to load the dataset.

1 MLP

For this part of the assignment, you will implement a simple MLP baseline for image classification on CIFAR-10 dataset.

1.1 Architecture and Training

Input to the network is an image ($32 \times 32 \times 3$) flattened as 3072 dimensional vector xi . Input is followed by layers h1 (500), h2 (250), h3 (100). The number of units are mentioned beside the corresponding hidden layer. Use ReLU activation for all these hidden layers. The output unit consists of 10 units, one for each class. Let the output activation be linear. Since you want to do classification, use softmax to get probabilities of image belonging to 10 classes. This will be a 10 dimensional vector ŷi . Now use cross-entropy loss between ŷi and yi , where yi is the one hot representation of the ground-truth label. Use SGD optimizer to train the model. Deliverables:

1. For the experiment above, you are required to show the plot of training error, validation error and prediction accuracy as the training progresses.
2. At the end of training, report the average prediction accuracy for the whole test set of 10000 images.
3. You should also plot randomly selected test images showing the true class label as well as predicted class label.
4. Report the confusion matrix that shows the kind of errors that your classifier makes. In this problem, your confusion matrix is a 10×10 matrix, where the rows represent the true label of a test sample and the columns represent the predicted labels of the classifier.
5. Use batch-normalization. Does it improve the test accuracy? Does it affect training time?

2 CNN

For this part of the assignment, we will implement VGG11 architecture for image classifica- tion from scratch. The architecture of VGG11 is described in the below image. Note that max-pooling layers reduce the image/feature size by half. All the experiments should be performed on CIFAR-10 datasets. Please follow the link https://www.binarystudy.com/2021/09/how-to-load-preprocess-visualize-CIFAR-10-and-CIFAR-100.html for the dataset exploration and visualization.

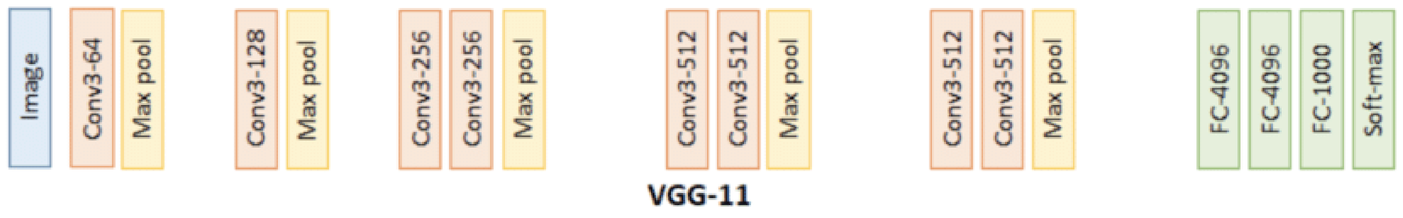

Figure 1: VGG11 architecture https://arxiv.org/pdf/1409.1556.pdf

2.1 Experimental Settings

Since the dataset contains 10 classes, modify (or add an extra layer) the last layer of VGG11. You are free to experiment with different settings of learning rates(such as 0.001,0.0001) and batch size for the training and report the set of hyperparameters, which resulted in the best performance. Use CrossEntropy loss and SGD optimizer for all of the experiments. You are also free to change and play with the fully connected layers' neurons (i.e., increase or decrease the no. of neurons in a layer or increase or decrease the fully connected layers' count). For some experiments, you may also remove the last max-pooling layer (before the fully connected layers). All the experiments must be documented in the final report irre- spective of the performance of a particular experiment. Report the result with

1. Training setting (such as learning rate and training epoch used). Report this for all the different settings that you experimented with.
2. Any bottleneck/challenges you faced during training or testing.
3. Training loss/accuracy plot (loss/accuracy versus epoch)
4. Report the accuracy of the test dataset.
5. Show visual results for 5 images for each category of test classes.
6. Now download at least 5 random images (or capture images from your smartphone, camera, or any imaging device) of appropriate classes and test those images on the trained network. What is your observation?

3 RNN

In this assignment, you will experiment with' a sample of the Dakshina Dataset released by Google. This dataset contains pairs of the following form:

x, y
ajanabee, अजनबी

i.e., a word in the native script and its corresponding transliteration in the Latin script (the way we type while chatting with our friends on WhatsApp etc). Given many such $(x_i, y_i)_{i=1}^{n}$ pairs your goal is to train a model $y = \hat{f}(x)$ which takes as input a romanized string (ghar) and produces the corresponding word in Devanagari (घर).

As you would realise this is the problem of mapping a sequence of characters in one language to a sequence of characters in another language. Notice that this is a scaled down version of the problem of translation where the goal is to translate a sequence of words in one language to a sequence of words in another language (as opposed to sequence of characters here). Read this blog to understand how to build sequence to sequence models.

1. Build a RNN based seq2seq model which contains the following layers:
   (a) input layer for character embeddings
   (b) one encoder RNN which sequentially encodes the input character sequence (Latin)

(c) one decoder RNN which takes the last state of the encoder as input and produces one output character at a time (Devanagari). For the deocder RNN:

$$\begin{aligned} s_t &= RNN(s_{t-1}, e(\hat{y}_{t-1})) \\ s_0 &= h_T \end{aligned} \tag{1}$$

where e is the character embedding, T is the length of the input sequence and h_t is the hidden state of the encoder at time step t.

The code should be flexible such that the dimension of the input character embeddings, the hidden states of the encoders and decoders, and the number of layers in the encoder and decoder can be changed.

2. Now train your model using any one language from the Aksharantar dataset (I would suggest picking a language that you can read so that it is easy to analyze the errors). Use the standard train, dev, test set from the folder dakshina_dataset_v1.0/hi/lexicons/ (replace hi with the language of your choice). Train this over the above RNN, and choose one set of hyperparameters which you feel might work the best. One suggestion could be: Hidden size of 256, number of layers: 2.

   (a) Mention the loss, optimizer used, learning rate and the number of epochs trained for.

   (b) Report the training loss, accuracy and validation loss,accuracy plots.

   (c) Show some of the interesting predictions of your network and comment on the same.

3. Repeat 3.2 with an RNN decorder whose equations are given as:

$$\begin{aligned} s_t &= RNN(s_{t-1}, [e(\hat{y}_{t-1}), h_T]) \\ s_0 &= h_T \end{aligned} \tag{2}$$

where e is the character embedding, T is the length of the input sequence and h_t is the hidden state of the encoder at time step t. Basically, you are also inputting the last hidden state of the encoder at every time step of the decoder.