AI2100/AI5100 Deep Learning, Spring 2022

Indian Institute of Technology Hyderabad

Homework 5, **10 points**. Convolutional Neural Networks (CNNs), Assigned 13.04.2022, Due **11:59 pm on 20.04.2022**

With great power comes great responsibility – Spiderman's Uncle Ben (?)

Instructions:

- It is **strongly recommended** that you work on your homework on an *individual* basis. If you have any questions or concerns, feel free to talk to the instructor or the TAs.
- **You are free to use PyTorch functions in this assignment.**
- Use grayscale images from the MNIST database. Use PyTorch to load the data and reshape the input to 28×28 patches. Please see `https://pytorch.org/vision/stable/datasets.html`.
- Refer to the PyTorch NN tutorial.
- Please turn in Python Notebooks with the following notation for the file name: `your-roll-number-hw5.ipynb`.
- Do not turn in images. Please use the same names for images in your code as in the database. The TAs will use these images to test your code.

In this assignment you will build on the functions you wrote in the previous assignment to implement an image classifier.

1. Train the following CNN for image classification. Randomly initialize your network.
   - Input image of size 28×28 (images from the MNIST dataset).
   - Convolution layer with 4 kernels of size 5×5, ReLU activation, and stride of 1. Ensure that each activation channel output from this conv layer has the same width and height as its input.
   - Max pooling layer of size 2×2 with a stride of 2 along each dimension.
   - Convolution layer with 4 kernels of size $5 \times 5 \times 4$, ReLU activation and stride of 1. As before, ensure that the input and output width and height match.
   - Max pooling layer of size 2×2 with a stride of 2 along each dimension.
   - This network has a flattening layer that is an identity matrix i.e., it simply passes the unravelled vector forward. Note that there is no need to learn the parameters of this layer.
   - An MLP with one hidden layer that accepts as input the flattening layer output maps it to a hidden layer with 49 nodes and then onto 10 output nodes. Use ReLU activation for the MLP. The output of the MLP is normalized using the softmax function.
2. Use *cross-entropy loss* to find the error at the softmax output layer. Note that the ground-truth labels are one-hot encoded vectors of 10 dimensions.
   (a) Vanilla SGD. Use learning rate $\eta = 0.001$. (1)
   (b) Momentum. Use $\eta = 0.001, \alpha = 0.9$. (1)
   (c) RMSProp. Use $\eta = 0.001, \rho = 0.9$. (1)

   For training, choose 100 images per class from the training set i.e., use 1000 images for training. Experiment with the number of images chosen per mini-batch in SGD. *Shuffle* the training data after one *epoch* i.e., after all the training data points have been passed through the network once. Do this for 15 epochs. You can experiment with the number of epochs as well.
3. Do the following after each epoch:
   (a) Compute the *error* on the training and test data sets. Choose 10 images per class from the test set for a total of 100 images. *Plot* the training and test errors as a function of epochs (at the end of 15 epochs). (1)
   (b) *Visualize* the activation maps. You can pick a couple of representative slices from the activation volumes at each of the convolution layers. (1)
   (c) Report the *accuracy* of your classifier. (1)
4. Use tSNE to visualize the bottleneck feature at the end of the first epoch and the last epoch. Use the same set of test images as in Q4. (1)
5. Compare the results with the output from HW4 and comment. For a fair comparison, use the same number of training and testing images as you used in HW4. (1)

6. Now apply dropout at the hidden layer of the MLP. Choose dropout rates of 0.2, 0.5 and 0.8. Train your model with this change and report your result for RMSProp. Compare this with the implementation without dropout and comment on the differences in training and testing performance. (1)

7. Now introduce batch normalization at the same hidden layer of the MLP and report performance. BN is done at the output of the hidden layer before applying the non-linearity. In your code, implement both BN and dropout (with $p = 0.2$). Again, compare performance with your vanilla implementation as well as the model with the dropout and comment on the training and testing performance. (1)