EE6380/AI2100/AI5100 Deep Learning, Fall 2024

Indian Institute of Technology Hyderabad

Homework 4, Convolutional Neural Networks (CNNs), Assigned 23.10.2024, Due **11:59 pm on 04.11.2024**, **25 points**

A computer would deserve to be called intelligent if it could deceive a human into believing that it was human. – Alan Turing

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.
- **The functions implemented for previous HW can be reused here.**
- Use `matplotlib` where specified - `https://matplotlib.org/tutorials/introductory/images.html`.
- Use color images from the CIFAR-10 dataset – `https://www.cs.toronto.edu/~kriz/cifar.html`. Use PyTorch to load the data. Please see `https://pytorch.org/vision/stable/datasets.html`.
- Please turn in Python Notebooks with the following notation for the file name: `your-roll-number-`hw4.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.

1 Numpy Implementation

In this assignment you will implement each of the components of a Convolutional Neural Network (CNN) from scratch (i.e., without using built-in functions for convolution, pooling, non-linearity, padding, striding). Your implementation must accept an image input and generate an output vector. Use random weights for filter kernels and fully connected layers. Specifically, implement the following:

1. **Convolution function:** It accepts as input an image, a filter kernel, stride, padding and the non-linear function. The function must correlate (convolve) the input image (after padding if specified) with the kernel (at the specified stride size) and generate an output activation after applying the specified non-linearity. Verify with the standard options for the non-linear activation functions - `sigmoid, tanh, ReLU, Parametric ReLU (PReLU)`. Display the input image, the filter kernel and the output activation map. Ensure that your function can accept multi-channel input and a corresponding kernel volume. (3)
2. **Pooling function:** It accepts as input the activation map output from the convolution function, a pooling function, and stride. The function must output the appropriately pooled activation map. Display the input activation map and the pooled output. (2)
3. **Convolution layer function:** It accepts as input a volume (image or activation maps), filter kernels, stride, padding and the non-linear function. The function must convolve the input volume (after padding if specified) with each of the kernels (at the specified stride size) and generates an output activation volume after applying the specified non-linearity. Display the input image or activation maps, the filter kernels and the output activation maps. Verify that the output of this function does indeed have the expected size ($W \times H \times C$) as discussed in class. (1)
4. **Pooling layer function:** It accepts as input the activation map volume, the pooling function, stride, and generates a pooled output volume. Display the input and output volumes. (1)
5. **Flattening (unraveling) function:** It accepts as input the activation map volume output by the pooling layer and generates a vector of a specified size. It is important to note that this function has a weight matrix associated with it whose size is chosen such that the input and desired output sizes are matched. (1)

6. **Multilayer Perceptron (MLP) function (Fully Connected):** It accepts as input a vector, the number of hidden layers, the size of each hidden layer, the non-linear function, and the size of the output layer. This function should generate an output vector of the specified size. Generate the output with and without the *softmax* function applied to the output layer. (2)

7. **Feed-forward path:** Finally, use the functions you have written to implement a CNN with the following architecture. The CNN must accept an image input and output a vector of appropriate dimension. In other words, the function must effectively implement the feed-forward path in a CNN. (3)

   - Input image of size $32 \times 32 \times 3$. Use images from the CIFAR-10 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.

   Verify that your composition of function accepts and image input and outputs a vector.

8. (a) Choose an image from each of the 10 classes and display the output vector for each case. Do you see any trend in the output vectors? (1)

   (b) Does a randomly initialized network show any discriminability? Visualize the bottleneck layer (output of flattening layer) using *builtin* tSNE plots. Choose three images per class. If you are wondering about this question, check out the Deep Image Prior paper. (1)

2 Pytorch Implementation

1. Train the CNN described in Q.7 above for image classification. Randomly initialize your network.

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 the previous question. (1)

5. 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)

6. 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. (2)