Indian Institute of Technology Hyderabad
Deep Learning (AI2100/AI5100/EE6380): Assignment-2

Topic: Convolutional Neural Networks and Sequence Models
Assigned on: 21st **February, 2025**
Deadline: 2nd **March, 2025**
Maximum Marks: 50

---

1 Instructions

- Answer all questions. We encourage best coding practices by not penalizing (i.e., you may not get full marks if you make it difficult for us to understand your submission. Hence, use intuitive names for the variables, functions, etc., and comment your code liberally. You may use the text cells in the notebook to briefly explain the objective of a code cell.)
- It is **expected** that you work on these problems individually. If you have any doubts please contact the TA or the instructor no later than 4 days before the deadline.
- You may use built-in implementations only for the basic functions such as `sqrt, log`, etc. from libraries such as `numpy` or `PyTorch`. Other high-level functionalities are expected to be implemented by the students. (Individual problem statements will make this clear.)
- For plots, you may use `matplotlib` and generate clear plots that are complete and easy to understand.
- You are expected to submit the Python Notebooks saved as A2_<your-roll-number>.ipynb
- If you are asked to report your observations, use the mark down text cells in the notebook.

2 Problems

3D Convolutional Neural Network

The objective of this problem is to perform a forward pass through a 3D Convolutional Neural Network (3D CNN) on a 3D point cloud representation of the MNIST dataset. You can do it on any datapoint available from this site. You are required to implement the core components of a 3D CNN, including 3D convolution functions, 3D pooling layers, and an MLP classifier, and then assemble them into a full 3D CNN model for classification.

1. **3D Convolution function**: Implement a function that applies 3D convolution to a given 3D point cloud input. The function should accept an input tensor representing the 3D volume, a kernel size for the convolution filter, stride, padding, number of filters, and a choice of non-linearity activation functions such as ReLU, sigmoid, tanh, or Leaky ReLU. Ensure that your function supports multi-channel 3D inputs and correctly computes the output activation maps. Display the input 3D point cloud and the corresponding output activation map, and verify that the output dimensions match expectations. (6)
2. **3D Pooling function**: Implement a function that performs 3D pooling on an activation map generated by a 3D convolution operation. The function should accept an input tensor, a choice of pooling operation (Max Pooling or Average Pooling), and stride values. The function should output the appropriately pooled activation map and verify that the output dimensions are computed correctly. Compare different pooling strategies by visualising their effect on feature retention. (3)
3. **3D Convolution layer function**: Implement a function that represents a 3D convolutional layer, which applies multiple 3D convolution filters to an input 3D volume. The function should accept a 3D input volume, apply multiple convolutional filters with the specified stride, padding, and activation function, and generate an output activation volume. Display the input, filter kernels, and resulting feature maps, ensuring that the function produces an output with the correct shape. (4)

4. **3D Pooling layer function**: Implement a function that applies a 3D pooling layer to a 3D activation map. The function should support both local pooling (Max or Average pooling) and Global Average Pooling (GAP) as a special case. Verify that the pooling layer correctly reduces the spatial dimensions and compare the effects of different pooling strategies. (3)

5. **Multilayer Perceptron (MLP) function**: Implement an MLP classifier that takes a flattened vector input from the GAP layer and maps it to a classification output. The function should support multiple hidden layers, allow selection of different activation functions, and output a 10-class vector representing the MNIST digits. The output should be generated with the softmax activation function applied to the final layer. (3)

6. **Putting-it all together**: Using the functions implemented above, construct a complete 3D CNN model that accepts 3D point clouds as input and classifies them into one of 10 MNIST classes. The model should consist of the following layers:

   - A 3D convolution layer with 16 filters of size $5 \times 5 \times 5$ and ReLU activation.
   - A Max Pooling 3D layer of size $2 \times 2 \times 2$ with stride 2.
   - A 3D convolution layer with 32 filters of size $3 \times 3 \times 3$ and ReLU activation.
   - A Max Pooling 3D layer of size $2 \times 2 \times 2$ with stride 2.
   - A Global Average Pooling (GAP) layer.
   - An MLP with one hidden layer, where the input size matches the output of the GAP layer and the output size is 10 (one for each digit class). The hidden layer should use ReLU activation, and the output layer should apply softmax activation.

   To validate the model, pass a 3D point cloud sample from the dataset through the entire pipeline and verify that the final output shape matches the expected (10-class) classification output. (6)

Sequence Models

7. **Balanced Parentheses Counting problem**: In this task, you have to create a dataset of variable-length sequences that contain different types of Parentheses ("(,)","{,}","[,]","<,>"), along with other characters such as letters, digits, or punctuation. Each sequence will be labelled by the total count of fully balanced bracket pairs it contains. To build your dataset, you may collect snippets from random text files, code assignments, or C/C++/Python/LaTex scripts on GitHub, or generate using LLM tools. Combine or merge these sources into one large text file, then sample random sequences of varying lengths (A range of 10 - 100 would be decent) from that merged text. To determine the label for each sequence, design a function (e.g., using a stack-based approach) that iterates over the characters, keeps track of opening brackets, and counts a pair as soon as it finds a matching closing bracket. You have to generate/create a big dataset (>= 5000) of such samples for training and testing(make sure your text file is big enough), where each sample is a single sequence (e.g., a snippet of length n), and each label is an integer count.

   For example,
   "((a*2)+(5%7)" has 2 balanced pairs.
   "#include<ios.h>" has 1 balanced pair.
   "main(){<}>" has 2 balanced pair, while "main(){<}>}" has 3 balanced pairs.

   You must implement three models — Elmon network, LSTM, and GRU—to predict the count of balanced brackets for each sequence. For comparison, include a baseline that always predicts 1 for any input sequence. Train each model on your dataset using a suitable regression loss (e.g., Mean Squared Error), and compare their performances along with the baseline by plotting the learning curves to show how each model converges. Although you must implement the core logic for the RNNs yourself (instead of using built-in modules), you may still inherit from `nn.Module` in PyTorch and use the standard `backward()` function for automatic differentiation. Show some sample results on test sequences. Submit your merged input text file also to reproduce the results. **Hint:** Remember to encode the input sequences appropriately before feeding them into your models. [4 (Elmon) + z8 (LSTM) + 5 (GRU)+ 8 (Dataset & baseline & comparison) = 25]