AI 1013: Programming for AI

Assignment 3

Due on: 14 April 2025, 11:59 PM IST.

Read each question carefully, and answer all parts. Write well-documented codes with contextual variable names. Put the codes (.py or .ipynb) and plots (.png, .jpg, .jpeg, .pdf) of all the questions into one single folder, compress this folder, and upload the .zip file. You should name the .zip file as `AI1013_assignment_3_<your roll number>.zip`.

1 Dataset Construction

1.1 Sinc Function Data with Noise

1.1.1 Data Generation

- Generate 1D input values using `torch.linspace()` over a suitable range (e.g., $[-10, 10]$).
- Compute the sinc function via $\text{sinc}(x) = \frac{\sin(x)}{x}$ using `torch.sin()` and element-wise division.
- Handle the $x = 0$ case using `torch.where()`.
- Add Gaussian noise generated with `torch.randn()`.

1.1.2 Visualization

- Use `matplotlib.pyplot.plot()` for the true sinc function.
- Use `matplotlib.pyplot.scatter()` to plot the noisy data.
- Include axis labels and legends for clarity.

2 Multilayer Perceptron (MLP) with Two Hidden Layers

2.1 Model Architecture & Setup

2.1.1 Defining the MLP Model

- Subclass `torch.nn.Module` to define the MLP class.
- Use `torch.nn.Linear` for the input, hidden, and output layers.
- Apply ReLU activation via `torch.nn.ReLU` or `torch.nn.functional.relu` in hidden layers.
- Implement the `forward()` method to define the forward computation flow.

2.1.2 Hyperparameters and Setup

- Set the batch size to 64, learning rate (e.g., 0.05), and number of epochs (e.g., 100).
- Use `torch.nn.MSELoss()` as the loss function.
- Choose an optimizer, such as `torch.optim.SGD`, to update model parameters.

2.2 Training the MLP

2.2.1 Data Preparation

- Convert input data from Section 1.1.1 to PyTorch tensors using `torch.tensor()`.
- Wrap the data using `torch.utils.data.TensorDataset` and create a DataLoader with `shuffle=True`.

2.2.2 Training Loop

- Perform a forward pass through the model for each batch.
- Compute the loss using MSE.
- Call `loss.backward()` to compute gradients.
- Clear gradients with `optimizer.zero_grad()`.
- Update parameters using `optimizer.step()`.
- Store the training loss for visualization.

2.2.3 Visualization of Training Loss

- Use `matplotlib.pyplot.plot()` to visualize training loss over epochs.

2.3 Decision Boundary Visualization

2.3.1 Generate a Mesh Grid

- Use `numpy.linspace()` and `numpy.meshgrid()` to span the data range.
- Convert the mesh grid to tensors via `torch.tensor()`.

2.3.2 Prediction over the Grid

- Set the model to evaluation mode using `model.eval()`.
- Use `torch.no_grad()` to compute predictions over the grid.
- Reshape predictions to fit the grid for visualization.

3 Introduction to CNN with PyTorch

In this assignment, you will explore Convolutional Neural Networks (CNNs) using PyTorch. You will work with the Rock-Paper-Scissors dataset from Kaggle (https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors) and perform a series of tasks that will guide you through data preparation, understanding convolutions, experimenting with pooling methods, and ultimately building and training a complete CNN model.
You can run on Google Colab with GPU runtime enabled.

For a brief refresher on CNNs, refer to Section 22.3 of Artifical Intelligence: A Modern Approach by Russell and Norvig (the course textbook for Intro to Modern AI from the previous semester).

3.1 Data Preparation

1. Write a custom PyTorch `Dataset` class to load and preprocess the images.
2. Split the dataset into training and test sets.

3.2 Understanding Convolutions

1. Build a modular CNN class using PyTorch's `nn.Module`.
2. Implement one convolutional layer using PyTorch's `nn.Conv2d` and verify if the output sizes match the expected dimensions.
3. Apply this model to a single image and visualize the outputs (feature maps) with different kernels such as blur, sharpen, and edge detection.
4. Experiment by changing kernel size, stride, and padding.

3.3 Pooling Functions

1. Extend your CNN class to include a pooling layer (`nn.MaxPool2d`). Visualize and compare the feature maps before and after pooling. Verify if the output sizes match the expected dimensions.
2. Implement Global Average Pooling (GAP) using `nn.AdaptiveAvgPool2d` and verify if the output sizes match the expected dimensions.

3.4 Building the Complete CNN Model

1. Implement the following CNN architecture:
   - Three convolutional layers, each followed by an activation function (`nn.ReLU`) and a MaxPool2d layer.
   - Apply 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 3 (one for each class). The hidden layer should use ReLU activation, and the output layer should apply softmax activation.
2. Print and verify the dimensions after each convolution, activation, pooling, GAP, and fully connected layer to ensure they match your expectations.

3.5 Training and Evaluation

1. Train your CNN model on the training set using the basic optimizer SGD and the loss function `nn.CrossEntropyLoss()`.
2. Evaluate your model on the test set and plot the training and test accuracy curves.
3. Experiment with learning rates, batch sizes, and epochs. Analyze and visualize how these hyperparameters affect your model's performance.