PyTorch Tensor and Autograd Exercises(Day1)

Exercises

Tensors

1. Create a 2D tensor of size 3×4 filled with ones.
2. Generate a random tensor of size 5×3 and compute its transpose.
3. Create a tensor with values from 0 to 11 and reshape it into a 3×4 matrix.
4. Create a tensor of size 4×4 with random values and replace all values less than 0.5 with zeros.
5. Concatenate two tensors of size 2×3 along the first dimension.
6. Compute the element-wise product of two tensors of the same size.
7. Perform matrix multiplication between a 2×3 tensor and a 3×4 tensor.
8. Calculate the mean and standard deviation of a tensor with random values.

Autograd

1. Create a tensor with requires_grad=True and perform a series of operations (addition, multiplication). Compute the gradient of the resulting tensor with respect to the original tensor.
2. Define a function $f(x) = x^2 + 3x + 2$. Use PyTorch autograd to compute the derivative of $f(x)$ at $x = 2$.
3. Create two tensors, perform element-wise multiplication, and then sum the result. Use autograd to compute the gradient.
4. Define a simple neural network with one linear layer. Use a random input tensor to perform a forward pass and compute the loss. Backpropagate the loss to obtain the gradients of the network parameters.
5. Compute the gradient of the function $g(x, y) = x^2y + y^3$ with respect to x and y at the point $x = 1$ and $y = 2$.

6. Implement a simple linear regression model. Define the loss function and use autograd to compute the gradients of the model parameters.

7. Create a tensor, compute its norm, and then use autograd to find the gradient of the norm with respect to the tensor.

8. Define a multi-variable function $h(x, y, z) = x^2 + y^2 + z^2$. Compute the gradient with respect to x, y, and z at the point $x = 1$, $y = 2$, and $z = 3$.

Dataset Class

1. Create a custom ‘Dataset‘ class that loads images from a directory. The class should:
   - Take the path to the directory as an argument.
   - Implement the ‘len‘ method to return the number of images.
   - Implement the ‘getitem‘ method to load and return an image and its corresponding label (assume labels are encoded in the filename).

2. Modify the custom ‘Dataset‘ class to apply a series of transformations (e.g., resizing, normalization) to each image.

3. Create a custom ‘Dataset‘ class that loads images and their corresponding bounding box annotations from a directory. The class should:
   - Take the paths to the images and annotations directories as arguments.
   - Implement the ‘len‘ method to return the number of images.
   - Implement the ‘getitem‘ method to load and return an image and its corresponding bounding box annotations.

4. Implement a custom ‘Dataset‘ class that loads grayscale images from a directory and converts them to RGB format.

5. Create a custom ‘Dataset‘ class that loads images from multiple directories and assigns a different label to each directory.

DataLoader

1. Use the custom ‘Dataset‘ class to create a ‘DataLoader‘ with a batch size of 32. Shuffle the data and use 4 worker threads.

2. Create a ‘DataLoader‘ that loads images in batches and applies data augmentation techniques such as random horizontal flip and random crop.

3. Implement a ‘DataLoader‘ that performs stratified sampling to ensure each batch has a balanced number of samples from each class.

4. Use a custom ‘Dataset‘ class and ‘DataLoader‘ to create a pipeline that loads and preprocesses images for a convolutional neural network.

5. Create a ‘DataLoader‘ that loads images in batches and applies a custom collate function to handle variable-sized images.

6. Implement a ‘DataLoader‘ with a batch sampler that ensures each batch contains a specified number of samples from each class.

7. Create a ‘DataLoader‘ that loads images in a specific order (e.g., sorted by filename).

8. Use a custom ‘Dataset‘ class and ‘DataLoader‘ to implement a training loop that trains a neural network on a dataset of images.