a.

The main benefit of using autograd in PyTorch is that it automates the process of computing gradients, which is used to optimize parameters in a neural network during training. Instead of manually implementing the forward and backward passes for each operation, as we did within the homework, PyTorch automatically tracks the forward pass and generates the backward pass using the chain rule, allowing it to be much easier and faster to develop custom neural networks. In Comparison to implementing gradients manually, autograd in PyTorch is also less error-prone, thus more efficient, as well as more flexible.

b.

PyTorch requires using its special tensor class (torch.Tensor) for most arrays and matrices to take advantage of efficient GPU acceleration for matrix operations, which is crucial for training deep neural networks. PyTorch's tensor class provides automatic differentiation and integration with other modules, making it a more comprehensive and flexible tool for building deep learning models. While numpy arrays are similar, they lack the same level of optimization and functionality as PyTorch tensors, as other array classes can lead to slower performance and errors.