Core components of pytorch

1. Tensor library for efficient computing
2. Automatic differentiation engine – utilities to differentiate computations automatically
3. Deep learning library

Defining Deep learning:

1. If cuda compatible gpu is found , torch for gpu will be automatically installed.
2. pip install torch torchvision torchaudio --index-url <https://download.pytorch.org/whl/cu126> [126 is cuda 12.6]
3. nvidia-smi =>The **nvidia-smi** (NVIDIA System Management Interface) command displays information about your **GPU**, including driver version, CUDA version, memory usage, and running processes.
4. Float 32 is generated by default and its optimized for GPU
5. Most operations are similar to GPU
6. If we carry out the computations in Pytorch , it will build a computational graph internally by default if one of its terminal nodes has requires\_grad =True
7. By default ..after calculating the gradient , the computational graph is destroyed but here using retain\_graph=True, we retain the graph as we do multiple grad calculations
8. We can ask the model not to compute the computation graph during inference , hence use torch.no\_grad()
9. For every random initialize use torch.manual\_Seed(number)
10. We don’t pass last layer to activation function in pytorch as its automatically taken care in loss functions, but for inference we can explicitly call those functions [softmax]
11. Dataset class defines how individual data records are loaded [we created tensor dataset that sits in memory]
12. DataLoader class defines dataset shuffling,batches and more
13. Cpu load and preprocess data , GPu waits if num workers =0
14. Else parallell processing can be done
15. Optimal training is num\_workers=4
16. Torch.save(model.state\_dict(),”model.pth”) -- dict object maps each layer to trainable params.. .pth , .pt are most common … model.load\_state\_dict(‘mlp.pth’) loads these params to the model of same architecture
17. GPU training – torch.cuda.is\_available() … tensor\_1.to(“cuda”) if multiple GPU … tensor\_2.to(“cuda:0”)
18. All tensors must be on same device . otherwise the computation will fail, where one tensor resides on CPU and other on GPU
19. Pytorch Distributed Data Parallel [DDP] splits input data and do the processing
20. Each gpu will receive a model,each model will receive a mini batch[distributed sampler ] non overlapping samples, get logits and gradients then synch up do weight update,
21. Multiprocessing works differently in scripts and jupyter notebook