

PRACTICE QUESTIONS

Linux Foundation / CNCF

PTCA

PyTorch Certified Associate

SHARED BY [u/Misty-Sparrow-3232](#) on aiexamcerts

EDITION Free Set 1

QUESTIONS 10

SHARED October 18, 2026

EXAM Linux Foundation / CNCF PTCA, PyTorch Certified Associate

Shared by [u/Misty-Sparrow-3232](#) on aiexamcerts

10 practice questions for the Linux Foundation / CNCF PTCA exam, PyTorch Certified Associate, each followed by its answer and a short explanation. More free sets and the full practice exam:

aiexamcerts.com/linux-foundation-cncf-ptca. Spotted a mistake? Email support@aiexamcerts.com with the exam code PTCA and the question number.

1. Mixed precision training with torch.autocast runs eligible operations in a lower precision such as float16 or _____.

Answer: bfloat16 (also accepted: bf16)

bfloat16 is the other common reduced precision type used with autocast, and it keeps float32's exponent range. Float16 has fewer exponent bits, which is why it usually needs a gradient scaler. Naming float64 here would describe higher precision, which autocast does not use to save memory.

2. Which transform converts a PIL image or a uint8 array into a float tensor scaled to the range 0 to 1?
 - A transforms.Normalize with mean 0 and standard deviation 1 applied to each channel
 - B transforms.RandomCrop with an output size of 224 pixels on each side of the image
 - C transforms.Resize, which changes the image to the nearest integer size for the batch
 - D transforms.ToTensor

Answer: D

transforms.ToTensor converts a PIL image or uint8 array into a float tensor and scales 8-bit values into the range 0 to 1. Normalize is tempting because it also produces floats, but it shifts and scales values by a mean and standard deviation rather than mapping them to the unit range. RandomCrop and Resize change geometry and do not change the value range.

3. What does the shape (batch, channels, height, width) describe in a typical PyTorch image tensor?
 - A The channels-last layout that every PyTorch operator requires for images
 - B The default NCHW layout, with the batch dimension first and the channel dimension second
 - C The NHWC layout that nn.Conv2d requires as its input format
 - D A single image with time steps stacked along the width dimension of the tensor

Answer: B

The default image layout in PyTorch is NCHW, with batch first, then channels, height and width. Channels-last is an alternative memory format, not a requirement, and nn.Conv2d accepts the NCHW layout by default. Stacking time steps along width describes a different data type and does not match the four named dimensions.

4. Which two components are usually defined inside an `nn.Module __init__` method? (Choose two.)
- A Layers such as `nn.Linear` and `nn.Conv2d`, registered as submodules of the model
 - B The loss value computed for the current batch during a training step
 - C Learnable tensors wrapped in `nn.Parameter` so that they are tracked by the module
 - D The `optimizer.step()` call that updates the weights after each backward pass
 - E The `DataLoader` that feeds training batches into the model during each epoch

Answer: A, C

Submodules and `nn.Parameter` objects assigned in `__init__` are registered automatically, which lets `parameters()`, `state_dict()` and optimizers find them. A per-batch loss and the optimizer step belong to the training loop, not the module. The `DataLoader` is outside the model entirely, so it is never a component of the module.

5. Which change to a training run most directly reduces time per epoch when the GPU is underutilized because batches load slowly?
- A Reduce the batch size to one example per step so that each step finishes sooner
 - B Increase `num_workers` in the `DataLoader` and enable `pin_memory` for faster host-to-device copies
 - C Remove the validation loop from every epoch so that less work runs per epoch
 - D Switch the loss function to a non-differentiable metric so that the backward pass is shorter

Answer: B

Raising `num_workers` loads batches in parallel, and `pin_memory` speeds up copies to the GPU, so the GPU spends less time waiting for data. Cutting the batch size to one sample makes each step process far less data, which usually lengthens the epoch rather than shortening it. Removing validation or using a non-differentiable loss does not address the loading delay and would break or change training.

6. Setting `requires_grad=False` on a parameter stops gradients from being computed for it during backward.
- A True
 - B False

Answer: A (True)

True. Autograd does not record operations that depend on a tensor with `requires_grad` set to `False` for gradient purposes, so its `.grad` stays unset. This is the standard way to freeze layers during fine-tuning.

7. Which DataLoader argument should be enabled to discard the final incomplete batch so that every batch has the same size?
- A shuffle=True, which reorders the samples at the start of every epoch
 - B num_workers=0, which loads every batch in the main process of the program
 - C pin_memory=True, which places batches in page-locked host memory for transfers
 - D drop_last=True, which discards a final batch smaller than the batch size

Answer: D

drop_last=True removes a final batch that is smaller than batch_size, so every batch has the same shape. Shuffling changes the order of samples but not the batch sizes, which makes it the most tempting wrong choice. The worker count and pinned memory affect loading speed and transfers, not the size of the last batch.

8. In gradient accumulation, the number of micro-batches processed before optimizer.step() is called is set by the accumulation _____.

Answer: steps (also accepted: step count)

Gradient accumulation runs several forward and backward passes, dividing each loss by the accumulation steps, and then calls optimizer.step() once. This simulates a larger batch on limited memory. Calling step after every micro-batch would undo the purpose of accumulating.

9. Which is the standard way to move a model's parameters onto a device before training?
- A model.to(device)
 - B Assigning model.device = 'cuda' before the forward pass runs
 - C Saving the model with torch.save to a path named cuda before training
 - D Wrapping the model in a Python list so that PyTorch moves each element automatically

Answer: A

model.to(device) moves all registered parameters and buffers to the target device and returns the module. Setting a device attribute on the model does nothing, because PyTorch does not store the device that way. Saving a file or wrapping the model in a list does not move any tensors.

10. Which container holds a list of submodules so that their parameters are registered and included in model.parameters()?
- A A plain Python list of layers stored as an attribute on the module
 - B A tuple of nn.Linear objects created once in the __init__ method
 - C nn.ModuleList, which registers each child module with the parent
 - D A dictionary stored in a local variable that exists only inside the forward method

Answer: C

nn.ModuleList registers each child, so their parameters appear in parameters(), state_dict() and the optimizer. A plain Python list hides its layers from the module, which means their weights are never trained, and the tuple has the same problem. A local dictionary inside forward is recreated each call and is not part of the model at all.