

Linux Foundation PTCA

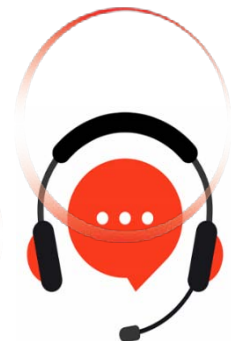
Linux Foundation PyTorch Certified Associate

For More Information – Visit link below:

<https://www.examsempire.com/>

Product Version

1. Up to Date products, reliable and verified.
2. Questions and Answers in PDF Format.



<https://examsempire.com/>

Visit us at: <https://www.examsempire.com/ptca>

Latest Version: 6.0

Question: 1

For an image classification model you want each training image converted to a tensor and then standardized per channel.

Which construction correctly chains these steps in order?

- A. `ToTensor(Normalize(mean, std))`
- B. `Compose([Normalize(mean, std), ToTensor()])`
- C. `Normalize(ToTensor(), mean, std)`
- D. `Compose([ToTensor(), Normalize(mean, std)])`

Answer: D

Question: 2

You are choosing which per-channel mean and std to pass to `Normalize` in your training transform pipeline.

Why is normalizing inputs a common preprocessing step?

- A. It increases the effective number of training samples by generating extra augmented copies of the data
- B. It rescales features to a comparable range, which keeps gradients well-behaved and stabilizes optimization
- C. It guarantees the model cannot overfit by clamping every input feature to a fixed unit range
- D. It converts the integer class labels into one-hot encoded target vectors for the loss

Answer: B

Question: 3

You already hold your features and labels as two aligned tensors, `X` and `y`, and want to feed them to a `DataLoader` without writing a custom class.

Which utility wraps existing tensors into a ready-to-use dataset?

- A. `random_split(X, y)`
- B. `Compose(X, y)`
- C. `TensorDataset(X, y)`
- D. `DataLoader(X, y)`

Answer: C

Question: 4

A colleague removes `optimizer.zero_grad()` from their training loop, keeping the forward pass, `loss.backward()`, and `optimizer.step()` each iteration. What happens as a result, and why?

- A. Nothing changes, because `loss.backward()` automatically overwrites the previous `.grad` values on every call, so each step still uses only the current batch.
- B. Gradients accumulate into `.grad`, so each step sums gradients across batches rather than using only the current one, corrupting the updates.
- C. Training halts with a runtime error, because PyTorch forbids running another backward pass while non-zero gradients from a prior iteration are still present.
- D. The parameters gradually stop updating, because the accumulated gradients from successive batches cancel out to a net value of zero.

Answer: B

Question: 5

During evaluation a developer calls `model.eval()` and then runs the validation batches, but memory usage stays high and the autograd graph is still being built. Which statement best explains the situation and the fix?

- A. `model.eval()` only switches layers like Dropout and BatchNorm; it does not disable gradient tracking, so also wrap evaluation in a `with torch.no_grad():` block.
- B. `model.eval()` already disables autograd, so the persistent memory must instead come from an unrelated leak in the data loader or metric accumulation.
- C. You must run a `loss.backward()` pass during validation to release the graph that `eval()` would otherwise leave allocated across batches.
- D. Switching back to `model.train()` during validation would disable gradient tracking and lower memory, at the cost of re-enabling Dropout.

Answer: A

Question: 6

In a standard PyTorch training loop, you call `optimizer.step()` after computing the loss and calling `loss.backward()`. What does `optimizer.step()` do?

- A. It resets all parameter gradients to zero so they are clean before the next iteration.
- B. It updates the model parameters using the gradients that were computed during backpropagation.
- C. It runs the forward pass over the input batch to produce the predictions the loss is computed from.
- D. It computes the gradients of the loss with respect to each of the model's trainable parameters.

Answer: B

Question: 7

A teammate defines a classifier whose final layer produces raw logits, then writes:
`probs = softmax(logits, dim=1)` followed by `loss = criterion(probs, targets)`, where `criterion = nn.CrossEntropyLoss()`.
Why is this a bug, and how should it be fixed?

- A. The real bug is the softmax dimension: `dim=0` normalizes across the batch, so keeping the softmax but switching it to `dim=1` makes the loss correct.
- B. The targets must also be one-hot encoded to match the softmax output, and adding that encoding aligns the shapes so the loss computes correctly.
- C. There is no bug here; `nn.CrossEntropyLoss` needs probabilities as input, so applying softmax to the logits first is exactly the required preprocessing.
- D. `nn.CrossEntropyLoss` already applies log-softmax internally, so the extra softmax double-applies it; pass the raw logits instead.

Answer: D

Question: 8

When implementing gradient accumulation over several mini-batches, a common mistake is forgetting one step of the loop.
Which step is essential to accumulate correctly?

- A. Call `optimizer.zero_grad()` only after the accumulated `optimizer.step()`, not after every mini-batch.
- B. Move the model to the CPU in between the mini-batches so that accumulated gradient memory is freed.
- C. Call `optimizer.zero_grad()` after every single mini-batch so the gradients never overlap between steps.
- D. Detach the loss with `.item()` before you call `backward()` on it each iteration.

Answer: A

Question: 9

A model applies an in-place ReLU (the variant ending in `_`) to a tensor whose original values are later needed to compute gradients during `backward()`. The run raises an autograd error about a value modified in place.

Why does the in-place operation cause this?

- A. The error happens because in-place operations are forced to run on the CPU while the rest of the model runs on the GPU.
- B. In-place ops double the tensor's memory use, triggering an out-of-memory failure that surfaces as an autograd error.
- C. The in-place op overwrote a value autograd had saved for the backward pass, so the gradient can no longer be computed.
- D. In-place operations are always disallowed anywhere inside a model that relies on autograd for its gradients.

Answer: C

Question: 10

During inference you wrap the forward pass in `torch.no_grad()`.
What does `torch.no_grad()` do?

- A. It moves the model along with all of its input tensors onto the GPU so that the forward pass runs faster.
- B. It permanently deletes the gradients already stored on the model's parameters.
- C. It switches dropout and batch-norm layers into their evaluation behavior for correct inference.
- D. It disables gradient tracking so autograd does not build the computation graph, saving memory and time.

Answer: D

Thank You for Trying Our Product

Special 16 USD Discount Coupon: **NSZUBG3X**

Email: support@examsempire.com

**Check our Customer Testimonials and ratings
available on every product page.**

Visit our website.

<https://examsempire.com/>