Your grade: 100%

Your latest: **100%** • Your highest: **100%** • To pass you need at least 80%. We keep your highest score.

Next item →

1. Which techniques does quantized low-rank adaptation (QLoRA) use for fine-tuning large language models (LLMs)? **1 / 1 point**

- LoRA adaptation
- 4-bit quantization
- Few-shot inference
- Zero-shot inference

Correct
4-bit quantization maintains high precision in the models. It reduces memory usage, making the model highly efficient for fine-tuning LLMs.

2. Which of the following techniques helps reduce the number of trainable parameters by adding low-rank matrices? **1 / 1 point**

- Additive fine-tuning
- Soft prompts
- Full fine-tuning
- LoRA

Correct
Reparameterization-based methods, such as LoRA or low-rank adaptation, use reparametrizing network weights using low-rank transformations. This reduces the number of trainable parameters while still working with high-dimensional matrices like the pre-trained parameters of the network.

3. LoRA helps fine-tune a pre-trained language model. If you introduce low-rank matrices to the model, how does it affect its parameter efficiency? **1 / 1 point**

- Introducing low-rank matrices to the existing weights helps add a small number of trainable parameters.
- Low-rank matrices keep track of the original number of parameters and use them for an efficient structure.
- Low-rank matrices replace the original weight matrices with low-rank approximation to reduce the number of trainable parameters.
- Low-rank matrices increase the number of trainable parameters by adding high-weight matrices.

Correct
Introducing low-rank matrices to the existing weights makes the fine-tuning process parameter-efficient and increases the model size.

4. Which of the following code snippets copies the original linear model and creates a LoRALayer object? **1 / 1 point**

```
class LoRALayer(torch.nn.Module):
    def __init__(self, in_dim, out_dim, rank, alpha):
        super().__init__()
        std_dev = 1 / torch.sqrt(torch.tensor(rank).float())
        self.A = torch.nn.Parameter(torch.randn(in_dim, rank) * std_dev)
        self.B = torch.nn.Parameter(torch.zeros(rank, out_dim))
        self.alpha = alpha

    def forward(self, x):
        x = self.alpha * (x @ self.A @ self.B)
        return x
```

```
from urllib.request import urlopen
import io

model_lora = TextClassifier(num_classes = 4,freeze = False)
```

```
model_lora.to(device)
path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/ \
                                   uGC04Pom651hQs1XrZ0NsQ/my-model-freeze-false.pth'

urlopened = urlopen(path)
stream = io.BytesIO(urlopened.read())
state_dict = torch.load(stream, map_location = device)
model_lora.load_state_dict(state_dict)
```

- [ ]

```
model_lora.fc2 = nn.Linear(in_features = 128, out_features = 2, bias = True).to(device)
model_lora.fc1 = LinearWithLoRA(model_lora.fc1,rank = 2, alpha = 0.1).to(device)
```

- [x]

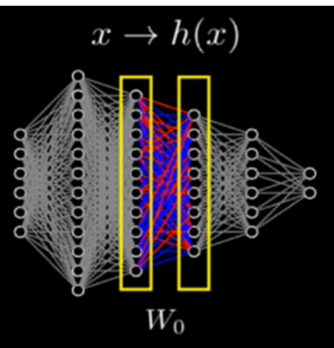
```
class LinearWithLoRA(torch.nn.Module):
    def __init__(self, linear, rank, alpha):
        super().__init__()
        self.linear = linear.to(device)
        self.lora = LoRALayer(
            linear.in_features, linear.out_features, rank, alpha
        ).to(device)

    def forward(self, x):
        return self.linear(x) + self.lora(x)
```

Correct
LinearWithLoRA applies the original linear model and LoRA model to the input X and adds the output together.

5. What would be the equation for the resultant parameter when the LoRA layer's output computes as a function of $h(x) = W_0 \times x$, where x = input from the last layer?

1 / 1 point



$$h(x) = W_0 x$$

$$\begin{bmatrix} W_{00} & W_{01} & \dots & W_{0k} \\ W_{10} & W_{11} & \dots & W_{1k} \\ \dots & \dots & \dots & \dots \\ W_{d0} & W_{d1} & \dots & W_{dk} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_k \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ \dots \\ h_d \end{bmatrix}$$

- [ ] d*x
- [ ] k*x
- [ ] d*W_0
- [x] d*k

Correct
The original neural network layer has a weight matrix W_0 with dimensions d by k, where d is the input size and k is the output size.