The resulting accuracy values are

```
Training accuracy: 46.25%
Validation accuracy: 45.00%
Test accuracy: 48.75%
```

These accuracy values are identical to the values from chapter 6. This result occurs because we initialized the LoRA matrix B with zeros. Consequently, the product of matrices AB results in a zero matrix. This ensures that the multiplication does not alter the original weights since adding zero does not change them.

Now let's move on to the exciting part—fine-tuning the model using the training function from chapter 6. The training takes about 15 minutes on an M3 MacBook Air laptop and less than half a minute on a V100 or A100 GPU.

Listing E.7 Fine-tuning a model with LoRA layers

```
import time
from chapter06 import train_classifier_simple

start_time = time.time()
torch.manual_seed(123)
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5, weight_decay=0.1)

num_epochs = 5
train_losses, val_losses, train_accs, val_accs, examples_seen = \
    train_classifier_simple(
        model, train_loader, val_loader, optimizer, device,
        num_epochs=num_epochs, eval_freq=50, eval_iter=5,
        tokenizer=tokenizer
    )

end_time = time.time()
execution_time_minutes = (end_time - start_time) / 60
print(f"Training completed in {execution_time_minutes:.2f} minutes.")
```

The output we see during the training is

```
Ep 1 (Step 000000): Train loss 3.820, Val loss 3.462
Ep 1 (Step 000050): Train loss 0.396, Val loss 0.364
Ep 1 (Step 000100): Train loss 0.111, Val loss 0.229
Training accuracy: 97.50% | Validation accuracy: 95.00%
Ep 2 (Step 000150): Train loss 0.135, Val loss 0.073
Ep 2 (Step 000200): Train loss 0.008, Val loss 0.052
Ep 2 (Step 000250): Train loss 0.021, Val loss 0.179
Training accuracy: 97.50% | Validation accuracy: 97.50%
Ep 3 (Step 000300): Train loss 0.096, Val loss 0.080
Ep 3 (Step 000350): Train loss 0.010, Val loss 0.116
Training accuracy: 97.50% | Validation accuracy: 95.00%
Ep 4 (Step 000400): Train loss 0.003, Val loss 0.151
Ep 4 (Step 000450): Train loss 0.008, Val loss 0.077
Ep 4 (Step 000500): Train loss 0.001, Val loss 0.147
Training accuracy: 100.00% | Validation accuracy: 97.50%
```

```
Ep 5 (Step 000550): Train loss 0.007, Val loss 0.094
Ep 5 (Step 000600): Train loss 0.000, Val loss 0.056
Training accuracy: 100.00% | Validation accuracy: 97.50%

Training completed in 12.10 minutes.
```

Training the model with LoRA took longer than training it without LoRA (see chapter 6) because the LoRA layers introduce an additional computation during the forward pass. However, for larger models, where backpropagation becomes more costly, models typically train faster with LoRA than without it.

As we can see, the model received perfect training and very high validation accuracy. Let's also visualize the loss curves to better see whether the training has converged:

```
from chapter06 import plot_values

epochs_tensor = torch.linspace(0, num_epochs, len(train_losses))
examples_seen_tensor = torch.linspace(0, examples_seen, len(train_losses))

plot_values(
    epochs_tensor, examples_seen_tensor,
    train_losses, val_losses, label="loss"
)
```

Figure E.5 plots the results.

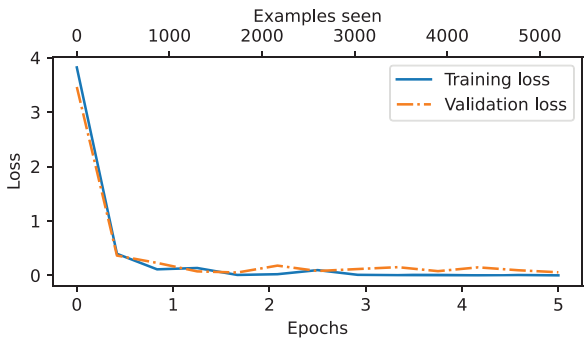

Figure E.5 The training and validation loss curves over six epochs for a machine learning model. Initially, both training and validation loss decrease sharply and then they level off, indicating the model is converging, which means that it is not expected to improve noticeably with further training.

In addition to evaluating the model based on the loss curves, let's also calculate the accuracies on the full training, validation, and test set (during the training, we approximated the training and validation set accuracies from five batches via the `eval_iter=5` setting):

```
train_accuracy = calc_accuracy_loader(train_loader, model, device)
val_accuracy = calc_accuracy_loader(val_loader, model, device)
test_accuracy = calc_accuracy_loader(test_loader, model, device)

print(f"Training accuracy: {train_accuracy*100:.2f}%")
print(f"Validation accuracy: {val_accuracy*100:.2f}%")
print(f"Test accuracy: {test_accuracy*100:.2f}%")
```

The resulting accuracy values are

```
Training accuracy: 100.00%
Validation accuracy: 96.64%
Test accuracy: 98.00%
```

These results show that the model performs well across training, validation, and test datasets. With a training accuracy of 100%, the model has perfectly learned the training data. However, the slightly lower validation and test accuracies (96.64% and 97.33%, respectively) suggest a small degree of overfitting, as the model does not generalize quite as well on unseen data compared to the training set. Overall, the results are very impressive, considering we fine-tuned only a relatively small number of model weights (2.7 million LoRA weights instead of the original 124 million model weights).