Show TranscriptSummarize Video

Note: The code in video has an issue where the `loss`variable was used to calculate the total loss as well. In this case, for every batch the loss was considered twice. Here, we should have a separate variable `total_loss`to overcome the issue. The code snippets below have it corrected.
A PyTorch training loop is an essential part of building a neural network model, which helps us teach the computer how to make predictions or decisions based on data. By using this loop, we gradually improve our model's accuracy through a process of learning from its mistakes and adjusting.
Technical Terms Explained:
Training Loop: The cycle that a neural network goes through many times to learn from the data by making predictions, checking errors, and improving itself.
Batches: Batches are small, evenly divided parts of data that the AI looks at and learns from each step of the way.
Epochs: A complete pass through the entire training dataset. The more epochs, the more the computer goes over the material to learn.
Loss functions: They measure how well a model is performing by calculating the difference between the model's predictions and the actual results.
Optimizer: Part of the neural network's brain that makes decisions on how to change the network to get better at its job.

Quiz Question
How does the model improve its prediction performance during the training loop?
- By increasing the number of nodes in the hidden layer
- By changing the activation function from ReLU to another type
- By using the loss function to measure errors and the optimizer to adjust itself
- By adding more layers to the neural network after each epoch

Submit

Code Examples
Create a Number Sum Dataset
This dataset has two features—a pair of numbers—and a target value—the sum of those two numbers.
Note that this is *not* actually a good use of deep learning. At the end of our training loop, the model still doesn't know how to add 3 + 7! The idea here is to use a simple example so it's easy to evaluate the model's performance.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

class NumberSumDataset(Dataset):
    def __init__(self, data_range=(1, 10)):
        self.numbers = list(range(data_range[0], data_range[1]))

    def __getitem__(self, index):
        number1 = float(self.numbers[index // len(self.numbers)])
        number2 = float(self.numbers[index % len(self.numbers)])
        return torch.tensor([number1, number2]), torch.tensor([number1 + numbe
```

```
    def __len__(self):
        return len(self.numbers) ** 2
```

Inspect the Dataset

```
dataset = NumberSumDataset(data_range=(1, 100))

for i in range(5):
    print(dataset[i])
# (tensor([1., 1.]), tensor([2.]))
# (tensor([1., 2.]), tensor([3.]))
# (tensor([1., 3.]), tensor([4.]))
# (tensor([1., 4.]), tensor([5.]))
# (tensor([1., 5.]), tensor([6.]))
```

Define a Simple Model

```
class MLP(nn.Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()
        self.hidden_layer = nn.Linear(input_size, 128)
        self.output_layer = nn.Linear(128, 1)
        self.activation = nn.ReLU()

    def forward(self, x):
        x = self.activation(self.hidden_layer(x))
        return self.output_layer(x)
```

Instantiate Components Needed for Training

```
dataset = NumberSumDataset(data_range=(0, 100))
dataloader = DataLoader(dataset, batch_size=100, shuffle=True)
model = MLP(input_size=2)
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Create a Training Loop

```
for epoch in range(10):
    total_loss = 0.0
    for number_pairs, sums in dataloader:  # Iterate over the batches
        predictions = model(number_pairs)  # Compute the model output
        loss = loss_function(predictions, sums)  # Compute the loss
        loss.backward()  # Perform backpropagation
        optimizer.step()  # Update the parameters
        optimizer.zero_grad()  # Zero the gradients

        total_loss += loss.item()  # Add the loss for all batches

    # Print the loss for this epoch
    print("Epoch {}: Sum of Batch Losses = {:.5f}".format(epoch, total_loss))
    # Epoch 0: Sum of Batch Losses = 118.82360
    # Epoch 1: Sum of Batch Losses = 39.75702
    # Epoch 2: Sum of Batch Losses = 2.16352
```

```
    # Epoch 3: Sum of Batch Losses = 0.25178
    # Epoch 4: Sum of Batch Losses = 0.22843
    # Epoch 5: Sum of Batch Losses = 0.19182
    # Epoch 6: Sum of Batch Losses = 0.15507
    # Epoch 7: Sum of Batch Losses = 0.07789
    # Epoch 8: Sum of Batch Losses = 0.06329
    # Epoch 9: Sum of Batch Losses = 0.04936
```

Try the Model Out

```
# Test the model on 3 + 7
model(torch.tensor([3.0, 7.0]))
# tensor([10.1067], grad_fn=<AddBackward0>)
```

Resources

PyTorch quickstart tutorial