PyTorch Loss Functions

Lesson Downloads

Show TranscriptSummarize Video

PyTorch loss functions are essential tools that help in improving the accuracy of a model by measuring errors. These functions come in different forms to tackle various problems, like deciding between categories (classification) or predicting values (regression). Understanding and using these functions correctly is key to making smart, effective models that do a great job at the tasks they're designed for!

Technical Terms Explained:

Loss functions: They measure how well a model is performing by calculating the difference between the model's predictions and the actual results.

Cross entropy loss: This is a measure used when a model needs to choose between categories (like whether an image shows a cat or a dog), and it shows how well the model's predictions align with the actual categories.

Mean squared error: This shows the average of the squares of the differences between predicted numbers (like a predicted price) and the actual numbers. It's often used for predicting continuous values rather than categories.

Quiz Question

In the context of a PyTorch model trained for clas ation, what does a wer cross-entropy loss value indicate?

- The model's predictions are more random a ess accurate
- The model's predictions are closer to the actual labels
- The model is overfitting to the training data
- The model requires more data for accurate pr ctions

Submit

Code Examples

Cross-Entropy Loss

```
import torch
import torch.nn as nn

loss_function = nn.CrossEntropyLoss()

# Our dataset contains a single image of a dog, where
# cat = 0 and dog = 1 (corresponding to index 0 and 1)
target_tensor = torch.tensor([1])
target_tensor
# tensor([1])
```

Prediction: Most likely a dog (index 1 is higher)

```
# Note that the values do not need to sum to 1
predicted_tensor = torch.tensor([[2.0, 5.0]])
loss_value = loss_function(predicted_tensor, target_tensor)
loss_value
# tensor(0.0181)
```

Prediction: Slightly more likely a cat (index 0 is higher)

```
predicted_tensor = torch.tensor([[1.5, 1.1]])
loss_value = loss_function(predicted_tensor, target_tensor)
```

```
loss_value
# tensor(0.9130)
```

Mean Squared Error Loss

```
# Define the loss function
loss_function = nn.MSELoss()

# Define the predicted and actual values as tensors
predicted_tensor = torch.tensor([320000.0])
actual_tensor = torch.tensor([300000.0])

# Compute the MSE loss
loss_value = loss_function(predicted_tensor, actual_tensor)
print(loss_value.item()) # Loss value: 20000 * 20000 / 1 = ...
# 400000000.0
```

Resources

torch.nn.CrossEntropyLoss documentation
torch.nn.MSELoss documentation
Index of PyTorch loss functions