Loading Spam Data with torch

Priyanshu Tiwari

2025-03-18

We'll create a custom dataset for the spam data from Elements of Statistical Learning.

Creating a Custom Spam Dataset

The spam dataset contains email characteristics with 57 features and a binary classification (spam=1, not-spam=0). Let's create a custom dataset class for this data:

```
library(torch)
spam_dataset <- dataset(
  name = "spam_dataset",

  initialize = function(url = "https://hastie.su.domains/ElemStatLearn/datasets/spam.data") {
    # Download and prepare the data
    self$data <- self$prepare_spam_data(url)
  },

  .getitem = function(index) {
    # Get features (all columns except the last one)
    x <- self$data[index, 1:57]

    # Get target (last column), convert to long tensor for classification
    y <- self$data[index, 58]$to(torch_long())

    list(x, y)
  },

  .length = function() {
    self$data$size()[[1]]
  },

  # Helper function to download and preprocess the data
  prepare_spam_data = function(url) {
    # Download the data
    temp_file <- tempfile()
    download.file(url, temp_file)

    # Read the data - space-separated values, no header
    spam_data <- read.table(temp_file, header = FALSE)

    # Convert to tensor
    input <- as.matrix(spam_data)
```

```
    torch_tensor(input, dtype = torch_float32())
  }
)
```

Using the Spam Dataset

Now let's create a dataset instance and explore it:

```
# Create an instance of our dataset
spam_data <- spam_dataset()

# Check the dataset size
spam_data$.length()
#> [1] 4601

# Look at the first item
first_item <- spam_data$.getitem(1)
cat("Features shape:", first_item[[1]]$shape, "\n")
#> Features shape: 57
cat("Target value:", first_item[[2]]$item(), "\n")
#> Target value: 1
```

Creating a DataLoader

To efficiently batch the data for training, we create a DataLoader:

```
# Create a dataloader with batch size of 32
dl <- dataloader(spam_data, batch_size = 32, shuffle = TRUE)

# Get the number of batches
num_batches <- dl$.length()
cat("Number of batches:", num_batches, "\n")
#> Number of batches: 144

# Inspect the first batch
iter <- dl$.iter()
batch <- iter$.next()
cat("Batch X shape:", batch[[1]]$shape, "\n")
#> Batch X shape: 32 57
cat("Batch Y shape:", batch[[2]]$shape, "\n")
#> Batch Y shape: 32
```

Model for spam classification

```
spam_net <- nn_module(
  "SpamNet",
  initialize = function() {
    self$fc1 <- nn_linear(57, 32)  # 57 input features
    self$fc2 <- nn_linear(32, 16)
```

```
    self$fc3 <- nn_linear(16, 1)    # 1 output for binary classification
  },
  forward = function(x) {
    x %>%
      self$fc1() %>%
      nnf_relu() %>%
      self$fc2() %>%
      nnf_relu() %>%
      self$fc3() %>%
      torch_sigmoid()  # Sigmoid activation for binary classification
  }
)

# Create the model
model <- spam_net()

# Binary cross-entropy loss for binary classification
loss_fn <- nn_bce_loss()

# Adam optimizer
optimizer <- optim_adam(model$parameters, lr = 0.001)
```

Train the model

```
# Training loop
num_epochs <- 5

for (epoch in 1:num_epochs) {

  model$train()  # Set model to training mode
  epoch_loss <- c()

  coro::loop(for (batch in dl) {
    # Get data and labels
    x <- batch[[1]]
    y <- batch[[2]]$float()  # Convert to float for BCE loss

    # Forward pass
    optimizer$zero_grad()
    pred <- model(x)
    loss <- loss_fn(pred, y$unsqueeze(2))  # Reshape y to match pred dimensions

    # Backward pass and optimize
    loss$backward()
    optimizer$step()

    # Store loss
    epoch_loss <- c(epoch_loss, loss$item())
  })

  # Print epoch statistics
```

```
  cat(sprintf("Epoch %d/%d, Loss: %.4f\n",
              epoch, num_epochs, mean(epoch_loss)))
}
#> Epoch 1/5, Loss: 0.6254
#> Epoch 2/5, Loss: 0.4339
#> Epoch 3/5, Loss: 0.3937
#> Epoch 4/5, Loss: 0.2995
#> Epoch 5/5, Loss: 0.2770
```

Evaluating the model

```
# Create a separate test dataloader (in practice, you would use a separate test set)
# For demonstration, we'll use the same data
test_dl <- dataloader(spam_data, batch_size = 64, shuffle = FALSE)

# Evaluate the model
model$eval()  # Set the model to evaluation mode

correct <- 0
total <- 0

# No need to track gradients during evaluation
with_no_grad({
  coro::loop(for (batch in test_dl) {
    x <- batch[[1]]
    y <- batch[[2]]

    # Forward pass
    outputs <- model(x)

    # Convert probabilities to binary predictions (threshold at 0.5)
    predicted <- (outputs > 0.5)$to(torch_long())

    # Count correct predictions
    total <- total + y$size(1)
    correct <- correct + (predicted$squeeze() == y)$sum()$item()
  })
})

accuracy <- correct / total
cat(sprintf("Accuracy: %.2f%%\n", accuracy * 100))
#> Accuracy: 91.41%
```