```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from keras.datasets import reuters
from keras.preprocessing.sequence import pad_sequences
import numpy as np

# Parameters
num_words = 10000      # Top most frequent words to consider
maxlen = 200           # Max sequence length for padding
embed_dim = 64
batch_size = 64
num_epochs = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 1. Load Reuters dataset from Keras
(x_train, y_train), (x_test, y_test) =
reuters.load_data(num_words=num_words)

# 2. Pad sequences to fixed length
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# 3. Convert to torch tensors
x_train = torch.tensor(x_train, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.long)
x_test = torch.tensor(x_test, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)

# 4. Create Dataset and DataLoader
train_ds = TensorDataset(x_train, y_train)
train_dl = DataLoader(train_ds, batch_size=batch_size, shuffle=True)

# 5. Define Model
class ReutersDNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.fc1 = nn.Linear(embed_dim * maxlen, 128)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
```

```
        x = self.embedding(x)              # [batch_size, maxlen,
embed_dim]
        x = x.view(x.size(0), -1)         # flatten: [batch_size,
maxlen*embed_dim]
        x = self.relu(self.fc1(x))
        return self.fc2(x)

num_classes = len(np.unique(y_train.numpy()))
model = ReutersDNN(num_words, embed_dim, num_classes).to(device)

# 6. Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 7. Training loop
for epoch in range(num_epochs):
    model.train()
    total_loss, correct = 0, 0
    for xb, yb in train_dl:
        xb, yb = xb.to(device), yb.to(device)
        optimizer.zero_grad()
        output = model(xb)
        loss = criterion(output, yb)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        correct += (output.argmax(1) == yb).sum().item()
    acc = correct / len(train_ds)
    print(f"Epoch {epoch+1}, Loss: {total_loss:.4f}, Accuracy: {acc:.4f}")

# 8. Evaluation on test data
model.eval()
with torch.no_grad():
    x_test = x_test.to(device)
    y_test = y_test.to(device)
    preds = model(x_test).argmax(1)
    test_acc = (preds == y_test).sum().item() / len(y_test)
    print(f"Test Accuracy: {test_acc:.4f}")
```