hw5q2

April 4, 2025

[1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import requests
import time
import math

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[ ]:
```
#sequence length 22
print(f"Using device: {device}")

# Step 1: Download the dataset
url = "https://raw.githubusercontent.com/karpathy/char-rnn/master/data/
↪tinyshakespeare/input.txt"
response = requests.get(url)
text = response.text  # This is the entire text data

# Step 2: Prepare the dataset
sequence_length = 20
text = text[:sequence_length * (len(text)//sequence_length)]  # Truncate text␣
↪to fit sequence length
# Create a character mapping to integers
chars = sorted(list(set(text)))
char_to_int = {ch: i for i, ch in enumerate(chars)}
int_to_char = {i: ch for i, ch in enumerate(chars)}

# Encode the text into integers
encoded_text = [char_to_int[ch] for ch in text]

# Create sequences and targets
sequences = []
targets = []
for i in range(0, len(encoded_text) - sequence_length):
    seq = encoded_text[i:i+sequence_length]
    target = encoded_text[i+sequence_length]
```

```
    sequences.append(seq)
    targets.append(target)

# Convert lists to PyTorch tensors and move to device
sequences = torch.tensor(sequences, dtype=torch.long).to(device)
targets = torch.tensor(targets, dtype=torch.long).to(device)

# Step 3: Create a dataset class
class CharDataset(Dataset):
    def __init__(self, sequences, targets):
        self.sequences = sequences
        self.targets = targets

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, index):
        return self.sequences[index], self.targets[index]

# Instantiate the dataset
dataset = CharDataset(sequences, targets)

# Step 4: Create data loaders
batch_size = 128
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset,
  [train_size, test_size])

train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=batch_size)

class CharModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size,
  model_type='Transformer', num_layers=2, num_heads=2, dim_feedforward=256,
  dropout=0.1):
        super(CharModel, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        if model_type == 'Transformer':
            encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_size,
  nhead=num_heads, dim_feedforward=dim_feedforward, dropout=dropout)
            self.transformer_encoder = nn.TransformerEncoder(encoder_layer,
  num_layers=num_layers)
        else:
            raise ValueError("Invalid model type. Choose 'Transformer'.")
        self.fc = nn.Linear(hidden_size, output_size)
```

```
    def forward(self, x):
        embedded = self.embedding(x)
        transformer_output = self.transformer_encoder(embedded)
        output = self.fc(transformer_output[:, -1, :])
        return output

# Train and evaluate function
def train_evaluate(model_type, train_loader, val_loader, device):
    model = CharModel(len(chars), hidden_size, len(chars), model_type).
 ↪to(device)
    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    start_time = time.time()
    for epoch in range(epochs):
        model.train()
        train_loss = 0.0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)  # Move␣
 ↪data to device
            optimizer.zero_grad()
            output = model(inputs)
            loss = criterion(output, targets)
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * inputs.size(0)

        epoch_train_loss = train_loss / len(train_loader.dataset)

        # Validation
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(device), targets.to(device)  # Move␣
 ↪data to device
                val_output = model(inputs)
                loss = criterion(val_output, targets)
                val_loss += loss.item() * inputs.size(0)
                _, predicted = torch.max(val_output, 1)
                total += targets.size(0)
                correct += (predicted == targets).sum().item()

        epoch_val_loss = val_loss / len(val_loader.dataset)
```

```
        epoch_val_accuracy = correct / total

        if (epoch+1) % 1 == 0:
            print(f'Epoch {epoch+1}, Train Loss: {epoch_train_loss}, Validation␣
↪Loss: {epoch_val_loss}, Validation Accuracy: {epoch_val_accuracy}')

    end_time = time.time()
    execution_time = end_time - start_time

    return epoch_train_loss, epoch_val_loss, epoch_val_accuracy, execution_time

# Define parameters
hidden_size = 512
num_layers = 2
num_heads = 2
dim_feedforward = 256
dropout = 0.1
learning_rate = 0.0001
epochs = 20

# Train and evaluate models for sequence length 20
print("\nTraining models for sequence length: 20")
results = {}
for model_type in ['Transformer']:
    print(f"\nTraining {model_type} model...")
    loss, val_loss, val_accuracy, execution_time = train_evaluate(model_type,␣
↪train_loader, test_loader, device)
    results[model_type] = {
        'loss': loss,
        'val_loss': val_loss,
        'val_accuracy': val_accuracy,
        'execution_time': execution_time
    }

# Print and compare results
print("\nResults for sequence length: 20")
for model_type, data in results.items():
    print(f"\n{model_type} Model:")
    print(f"Training Loss: {data['loss']}")
    print(f"Validation Loss: {data['val_loss']}")
    print(f"Validation Accuracy: {data['val_accuracy']}")
    print(f"Execution Time: {data['execution_time']} seconds")
```

```
Using device: cuda

Training models for sequence length: 20
```

```
Training Transformer model…

/home/dman/.venv/master/lib/python3.12/site-
packages/torch/nn/modules/transformer.py:385: UserWarning: enable_nested_tensor
is True, but self.use_nested_tensor is False because
encoder_layer.self_attn.batch_first was not True(use batch_first for better
inference performance)
  warnings.warn(

Epoch 1, Train Loss: 2.5119670595128847, Validation Loss: 2.479000235643607,
Validation Accuracy: 0.26984112752833167
Epoch 2, Train Loss: 2.483663074872087, Validation Loss: 2.473960489132982,
Validation Accuracy: 0.26918214746808206
Epoch 3, Train Loss: 2.4786546353783536, Validation Loss: 2.470480546601071,
Validation Accuracy: 0.2681062616554296
Epoch 4, Train Loss: 2.4755727058331845, Validation Loss: 2.4688133793605007,
Validation Accuracy: 0.26835730167838184
Epoch 5, Train Loss: 2.473518109085604, Validation Loss: 2.4662613287288067,
Validation Accuracy: 0.27017734184478553
Epoch 6, Train Loss: 2.4720310511949535, Validation Loss: 2.4647354761919047,
Validation Accuracy: 0.26998457897001865
Epoch 7, Train Loss: 2.470865965163138, Validation Loss: 2.4649354778117964,
Validation Accuracy: 0.2688235188638646
Epoch 8, Train Loss: 2.4700172492650982, Validation Loss: 2.4638854581063763,
Validation Accuracy: 0.2692673217615837
Epoch 9, Train Loss: 2.469070836985031, Validation Loss: 2.4635485943720514,
Validation Accuracy: 0.2707287333237699
Epoch 10, Train Loss: 2.4686012433601614, Validation Loss: 2.462973408712032,
Validation Accuracy: 0.26957215607516855
Epoch 11, Train Loss: 2.4680831152916367, Validation Loss: 2.463070592995166,
Validation Accuracy: 0.2690431788839478
Epoch 12, Train Loss: 2.467685655616582, Validation Loss: 2.461842031359861,
Validation Accuracy: 0.2676086644670779
Epoch 13, Train Loss: 2.4672298739893916, Validation Loss: 2.463565190610337,
Validation Accuracy: 0.2707332161813226
Epoch 14, Train Loss: 2.4667775889514183, Validation Loss: 2.4620316354877176,
Validation Accuracy: 0.2703880361497633
Epoch 15, Train Loss: 2.4664592531896736, Validation Loss: 2.4629334047011335,
Validation Accuracy: 0.2695183617845359
Epoch 16, Train Loss: 2.4663248621841154, Validation Loss: 2.461294505846174,
Validation Accuracy: 0.268944556017788
Epoch 17, Train Loss: 2.4659226315210647, Validation Loss: 2.461467037787825,
Validation Accuracy: 0.2699397503944915
Epoch 18, Train Loss: 2.4655860289151077, Validation Loss: 2.4610873909664197,
Validation Accuracy: 0.26924939033137285
Epoch 19, Train Loss: 2.465595694529482, Validation Loss: 2.460609244472617,
Validation Accuracy: 0.27054045330655574
Epoch 20, Train Loss: 2.4652549096628578, Validation Loss: 2.4615660985356476,
Validation Accuracy: 0.27000251040022955
```

```
Results for sequence length: 20

Transformer Model:
Training Loss: 2.4652549096628578
Validation Loss: 2.4615660985356476
Validation Accuracy: 0.27000251040022955
Execution Time: 934.1320867538452 seconds
```

[6]:
```python
#sequence length 30

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import requests
import time
import math

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Step 1: Download the dataset
url = "https://raw.githubusercontent.com/karpathy/char-rnn/master/data/
↪tinyshakespeare/input.txt"
response = requests.get(url)
text = response.text  # This is the entire text data

# Step 2: Prepare the dataset
sequence_length = 30
text = text[:sequence_length * (len(text)//sequence_length)]  # Truncate text␣
↪to fit sequence length
# Create a character mapping to integers
chars = sorted(list(set(text)))
char_to_int = {ch: i for i, ch in enumerate(chars)}
int_to_char = {i: ch for i, ch in enumerate(chars)}

# Encode the text into integers
encoded_text = [char_to_int[ch] for ch in text]

# Create sequences and targets
sequences = []
targets = []
for i in range(0, len(encoded_text) - sequence_length):
    seq = encoded_text[i:i+sequence_length]
    target = encoded_text[i+sequence_length]
    sequences.append(seq)
```

```python
    targets.append(target)

# Convert lists to PyTorch tensors and move to device
sequences = torch.tensor(sequences, dtype=torch.long).to(device)
targets = torch.tensor(targets, dtype=torch.long).to(device)

# Step 3: Create a dataset class
class CharDataset(Dataset):
    def __init__(self, sequences, targets):
        self.sequences = sequences
        self.targets = targets

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, index):
        return self.sequences[index], self.targets[index]

# Instantiate the dataset
dataset = CharDataset(sequences, targets)

# Step 4: Create data loaders
batch_size = 128
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size])

train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=batch_size)

class CharModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, model_type='Transformer', num_layers=2, num_heads=2, dim_feedforward=256, dropout=0.1):
        super(CharModel, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        if model_type == 'Transformer':
            encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_size, nhead=num_heads, dim_feedforward=dim_feedforward, dropout=dropout)
            self.transformer_encoder = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        else:
            raise ValueError("Invalid model type. Choose 'Transformer'.")
        self.fc = nn.Linear(hidden_size, output_size)
```

```python
    def forward(self, x):
        embedded = self.embedding(x)
        transformer_output = self.transformer_encoder(embedded)
        output = self.fc(transformer_output[:, -1, :])
        return output

# Train and evaluate function
def train_evaluate(model_type, train_loader, val_loader, device):
    model = CharModel(len(chars), hidden_size, len(chars), model_type).to(device)
    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    start_time = time.time()
    for epoch in range(epochs):
        model.train()
        train_loss = 0.0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)  # Move data to device
            optimizer.zero_grad()
            output = model(inputs)
            loss = criterion(output, targets)
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * inputs.size(0)

        epoch_train_loss = train_loss / len(train_loader.dataset)

        # Validation
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(device), targets.to(device)  # Move data to device
                val_output = model(inputs)
                loss = criterion(val_output, targets)
                val_loss += loss.item() * inputs.size(0)
                _, predicted = torch.max(val_output, 1)
                total += targets.size(0)
                correct += (predicted == targets).sum().item()

        epoch_val_loss = val_loss / len(val_loader.dataset)
        epoch_val_accuracy = correct / total
```

```python
        if (epoch+1) % 1 == 0:
            print(f'Epoch {epoch+1}, Train Loss: {epoch_train_loss}, Validation␣
↪Loss: {epoch_val_loss}, Validation Accuracy: {epoch_val_accuracy}')

    end_time = time.time()
    execution_time = end_time - start_time

    return epoch_train_loss, epoch_val_loss, epoch_val_accuracy, execution_time

# Define parameters
hidden_size = 512
num_layers = 2
num_heads = 2
dim_feedforward = 256
dropout = 0.1
learning_rate = 0.0001
epochs = 20

# Train and evaluate models for sequence length 30
print("\nTraining models for sequence length: 30")
results = {}
for model_type in ['Transformer']:
    print(f"\nTraining {model_type} model...")
    loss, val_loss, val_accuracy, execution_time = train_evaluate(model_type,␣
↪train_loader, test_loader, device)
    results[model_type] = {
        'loss': loss,
        'val_loss': val_loss,
        'val_accuracy': val_accuracy,
        'execution_time': execution_time
    }

# Print and compare results
print("\nResults for sequence length: 30")
for model_type, data in results.items():
    print(f"\n{model_type} Model:")
    print(f"Training Loss: {data['loss']}")
    print(f"Validation Loss: {data['val_loss']}")
    print(f"Validation Accuracy: {data['val_accuracy']}")
    print(f"Execution Time: {data['execution_time']} seconds")
```

```
Using device: cuda

Training models for sequence length: 30

Training Transformer model…
```

```
Epoch 1, Train Loss: 2.511884667894887, Validation Loss: 2.4799857660870193,
Validation Accuracy: 0.2682724550361325
Epoch 2, Train Loss: 2.4825677109980533, Validation Loss: 2.4747629323492117,
Validation Accuracy: 0.26891799809923433
Epoch 3, Train Loss: 2.477477387469446, Validation Loss: 2.4695597047270694,
Validation Accuracy: 0.26957250703821256
Epoch 4, Train Loss: 2.4748635443098257, Validation Loss: 2.4690307308847,
Validation Accuracy: 0.27129843814442234
Epoch 5, Train Loss: 2.4727685301916122, Validation Loss: 2.4677752525927374,
Validation Accuracy: 0.2698056198109993
Epoch 6, Train Loss: 2.47145712259995, Validation Loss: 2.465626111920603,
Validation Accuracy: 0.2697159610522352
Epoch 7, Train Loss: 2.4702125653769627, Validation Loss: 2.4674616354291232,
Validation Accuracy: 0.2700745960872918
Epoch 8, Train Loss: 2.469227096380211, Validation Loss: 2.4665705227985684,
Validation Accuracy: 0.2707335879642082
Epoch 9, Train Loss: 2.4687092175458822, Validation Loss: 2.4652078668154624,
Validation Accuracy: 0.27063048039162946
Epoch 10, Train Loss: 2.4680916146246865, Validation Loss: 2.4651610764328997,
Validation Accuracy: 0.26852349956067206
Epoch 11, Train Loss: 2.4674120754586566, Validation Loss: 2.4649974400490926,
Validation Accuracy: 0.2700835619631682
Epoch 12, Train Loss: 2.467287252531045, Validation Loss: 2.4647911143071624,
Validation Accuracy: 0.27007907902523
Epoch 13, Train Loss: 2.4666907511825067, Validation Loss: 2.465175572344073,
Validation Accuracy: 0.2698235515627522
Epoch 14, Train Loss: 2.466199736526151, Validation Loss: 2.4639305313879283,
Validation Accuracy: 0.27095773486111857
Epoch 15, Train Loss: 2.466092009425264, Validation Loss: 2.464216867291759,
Validation Accuracy: 0.2703973676188427
Epoch 16, Train Loss: 2.46574786988668, Validation Loss: 2.4623616908448946,
Validation Accuracy: 0.2706618609571969
Epoch 17, Train Loss: 2.465277184716176, Validation Loss: 2.463033082189479,
Validation Accuracy: 0.2708501443506016
Epoch 18, Train Loss: 2.465334096599519, Validation Loss: 2.4644987555542204,
Validation Accuracy: 0.2654795847006294
Epoch 19, Train Loss: 2.465086319987513, Validation Loss: 2.461927108498556,
Validation Accuracy: 0.270599099826062
Epoch 20, Train Loss: 2.4649108854089943, Validation Loss: 2.461431614623741,
Validation Accuracy: 0.27050495812935965

Results for sequence length: 30

Transformer Model:
Training Loss: 2.4649108854089943
Validation Loss: 2.461431614623741
Validation Accuracy: 0.27050495812935965
Execution Time: 1232.0165874958038 seconds
```

[7]:

```
#sequence length 50

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import requests
import time
import math

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Step 1: Download the dataset
url = "https://raw.githubusercontent.com/karpathy/char-rnn/master/data/
↪tinyshakespeare/input.txt"
response = requests.get(url)
text = response.text  # This is the entire text data

# Step 2: Prepare the dataset
sequence_length = 50
text = text[:sequence_length * (len(text)//sequence_length)]  # Truncate text␣
↪to fit sequence length
# Create a character mapping to integers
chars = sorted(list(set(text)))
char_to_int = {ch: i for i, ch in enumerate(chars)}
int_to_char = {i: ch for i, ch in enumerate(chars)}

# Encode the text into integers
encoded_text = [char_to_int[ch] for ch in text]

# Create sequences and targets
sequences = []
targets = []
for i in range(0, len(encoded_text) - sequence_length):
    seq = encoded_text[i:i+sequence_length]
    target = encoded_text[i+sequence_length]
    sequences.append(seq)
    targets.append(target)

# Convert lists to PyTorch tensors and move to device
sequences = torch.tensor(sequences, dtype=torch.long).to(device)
targets = torch.tensor(targets, dtype=torch.long).to(device)

# Step 3: Create a dataset class
class CharDataset(Dataset):
    def __init__(self, sequences, targets):
```

```
        self.sequences = sequences
        self.targets = targets

    def __len__(self):
        return len(self.sequences)

    def __getitem__(self, index):
        return self.sequences[index], self.targets[index]

# Instantiate the dataset
dataset = CharDataset(sequences, targets)

# Step 4: Create data loaders
batch_size = 128
train_size = int(len(dataset) * 0.8)
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset,␣
 ↪[train_size, test_size])

train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)
test_loader = DataLoader(test_dataset, shuffle=False, batch_size=batch_size)

class CharModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size,␣
 ↪model_type='Transformer', num_layers=2, num_heads=2, dim_feedforward=256,␣
 ↪dropout=0.1):
        super(CharModel, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        if model_type == 'Transformer':
            encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_size,␣
 ↪nhead=num_heads, dim_feedforward=dim_feedforward, dropout=dropout)
            self.transformer_encoder = nn.TransformerEncoder(encoder_layer,␣
 ↪num_layers=num_layers)
        else:
            raise ValueError("Invalid model type. Choose 'Transformer'.")
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        embedded = self.embedding(x)
        transformer_output = self.transformer_encoder(embedded)
        output = self.fc(transformer_output[:, -1, :])
        return output

# Train and evaluate function
def train_evaluate(model_type, train_loader, val_loader, device):
```

```python
    model = CharModel(len(chars), hidden_size, len(chars), model_type).to(device)
    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    start_time = time.time()
    for epoch in range(epochs):
        model.train()
        train_loss = 0.0
        for inputs, targets in train_loader:
            inputs, targets = inputs.to(device), targets.to(device)  # Move data to device
            optimizer.zero_grad()
            output = model(inputs)
            loss = criterion(output, targets)
            loss.backward()
            optimizer.step()
            train_loss += loss.item() * inputs.size(0)

        epoch_train_loss = train_loss / len(train_loader.dataset)

        # Validation
        model.eval()
        val_loss = 0.0
        correct = 0
        total = 0
        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(device), targets.to(device)  # Move data to device
                val_output = model(inputs)
                loss = criterion(val_output, targets)
                val_loss += loss.item() * inputs.size(0)
                _, predicted = torch.max(val_output, 1)
                total += targets.size(0)
                correct += (predicted == targets).sum().item()

        epoch_val_loss = val_loss / len(val_loader.dataset)
        epoch_val_accuracy = correct / total

        if (epoch+1) % 1 == 0:
            print(f'Epoch {epoch+1}, Train Loss: {epoch_train_loss}, Validation Loss: {epoch_val_loss}, Validation Accuracy: {epoch_val_accuracy}')

    end_time = time.time()
    execution_time = end_time - start_time
```

```
    return epoch_train_loss, epoch_val_loss, epoch_val_accuracy, execution_time

# Define parameters
hidden_size = 512
num_layers = 2
num_heads = 2
dim_feedforward = 256
dropout = 0.1
learning_rate = 0.0001
epochs = 20

# Train and evaluate models for sequence length 50
print("\nTraining models for sequence length: 50")
results = {}
for model_type in ['Transformer']:
    print(f"\nTraining {model_type} model...")
    loss, val_loss, val_accuracy, execution_time = train_evaluate(model_type,
 ↪train_loader, test_loader, device)
    results[model_type] = {
        'loss': loss,
        'val_loss': val_loss,
        'val_accuracy': val_accuracy,
        'execution_time': execution_time
    }

# Print and compare results
print("\nResults for sequence length: 50")
for model_type, data in results.items():
    print(f"\n{model_type} Model:")
    print(f"Training Loss: {data['loss']}")
    print(f"Validation Loss: {data['val_loss']}")
    print(f"Validation Accuracy: {data['val_accuracy']}")
    print(f"Execution Time: {data['execution_time']} seconds")
```

```
Using device: cuda

Training models for sequence length: 50

Training Transformer model…
Epoch 1, Train Loss: 2.5139466033113984, Validation Loss: 2.4760523837510324,
Validation Accuracy: 0.26515735676499597
Epoch 2, Train Loss: 2.4844192769659332, Validation Loss: 2.470618312523246,
Validation Accuracy: 0.2598493678830808
Epoch 3, Train Loss: 2.479298906818822, Validation Loss: 2.4674401754115736,
Validation Accuracy: 0.26669057652649514
Epoch 4, Train Loss: 2.4757765249554846, Validation Loss: 2.4641872418404565,
Validation Accuracy: 0.27013807944050927
Epoch 5, Train Loss: 2.474006155998752, Validation Loss: 2.4615962165829783,
```

```
Validation Accuracy: 0.271115394960997
Epoch 6, Train Loss: 2.472306903586691, Validation Loss: 2.461117958011386,
Validation Accuracy: 0.2707253653725455
Epoch 7, Train Loss: 2.4715746424160314, Validation Loss: 2.4608820137267453,
Validation Accuracy: 0.26848381601362864
Epoch 8, Train Loss: 2.4704268379189194, Validation Loss: 2.461576886952196,
Validation Accuracy: 0.2713619653904779
Epoch 9, Train Loss: 2.4698955826103917, Validation Loss: 2.459983678510945,
Validation Accuracy: 0.26928180758540304
Epoch 10, Train Loss: 2.4694360732177145, Validation Loss: 2.459338653439269,
Validation Accuracy: 0.27021429211871245
Epoch 11, Train Loss: 2.4687797129704667, Validation Loss: 2.458194596083573,
Validation Accuracy: 0.27265758091993186
Epoch 12, Train Loss: 2.4684547925437927, Validation Loss: 2.459560776056115,
Validation Accuracy: 0.271115394960997
Epoch 13, Train Loss: 2.4681183741679056, Validation Loss: 2.4588956058180345,
Validation Accuracy: 0.2712274724289429
Epoch 14, Train Loss: 2.4676741856219553, Validation Loss: 2.4589549523773164,
Validation Accuracy: 0.2696763202725724
Epoch 15, Train Loss: 2.467358334652567, Validation Loss: 2.4585148517918527,
Validation Accuracy: 0.27254102035326816
Epoch 16, Train Loss: 2.4671145867245365, Validation Loss: 2.456896469965651,
Validation Accuracy: 0.2714471442661167
Epoch 17, Train Loss: 2.466653859334217, Validation Loss: 2.4583627447016965,
Validation Accuracy: 0.26896350757643683
Epoch 18, Train Loss: 2.4667657241557923, Validation Loss: 2.456080484467216,
Validation Accuracy: 0.2713081682058639
Epoch 19, Train Loss: 2.46620395368475, Validation Loss: 2.456875953308341,
Validation Accuracy: 0.2712588541199677
Epoch 20, Train Loss: 2.466069513178295, Validation Loss: 2.4564821348846158,
Validation Accuracy: 0.27076123016228815

Results for sequence length: 50

Transformer Model:
Training Loss: 2.466069513178295
Validation Loss: 2.4564821348846158
Validation Accuracy: 0.27076123016228815
Execution Time: 2026.1996200084686 seconds
```