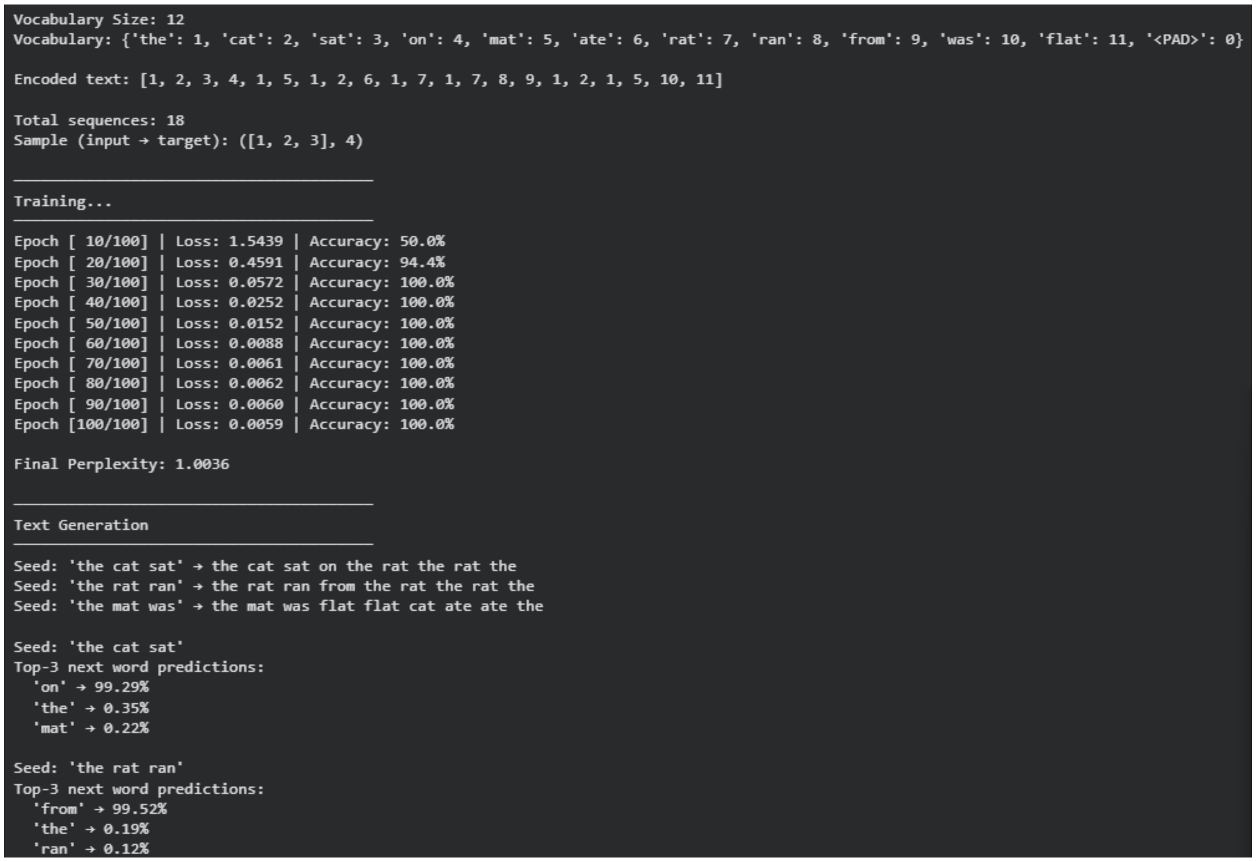
Experiment 7: Text processing, Language Modelling using RNN

```
import torch
import torch.nn as nn
import numpy as np
from collections import Counter

text = """the cat sat on the mat the cat ate the rat
the rat ran from the cat the mat was flat"""

tokens = text.lower().split()

counter = Counter(tokens)
vocab = {word: idx+1 for idx, (word, _) in
enumerate(counter.items())}
vocab['<PAD>'] = 0
idx_to_word = {v: k for k, v in vocab.items()}

VOCAB_SIZE = len(vocab)
print(f"Vocabulary Size: {VOCAB_SIZE}")
print(f"Vocabulary: {vocab}\n")

encoded = [vocab[w] for w in tokens]
print(f"Encoded text: {encoded}\n")

SEQ_LENGTH = 3

sequences = []
for i in range(len(encoded) - SEQ_LENGTH):
  seq_in  = encoded[i : i + SEQ_LENGTH]
  seq_out = encoded[i + SEQ_LENGTH]
  sequences.append((seq_in, seq_out))

print(f"Total sequences: {len(sequences)}")
print(f"Sample (input → target): {sequences[0]}\n")

X = torch.tensor([s[0] for s in sequences], dtype=torch.long)
y = torch.tensor([s[1] for s in sequences], dtype=torch.long)

class RNNLanguageModel(nn.Module):
  def __init__(self, vocab_size, embed_dim, hidden_dim,
num_layers=2, dropout=0.3):
    super().__init__()
    self.embedding = nn.Embedding(vocab_size, embed_dim,
padding_idx=0)
    self.lstm      = nn.LSTM(embed_dim, hidden_dim,
num_layers,
                    batch_first=True, dropout=dropout)
    self.dropout   = nn.Dropout(dropout)
    self.fc        = nn.Linear(hidden_dim, vocab_size)

  def forward(self, x, hidden=None):
    embeds      = self.embedding(x)
    out, hidden = self.lstm(embeds, hidden)
    out         = self.dropout(out[:, -1, :])
    logits      = self.fc(out)
    return logits, hidden

EMBED_DIM  = 32
HIDDEN_DIM = 128
NUM_LAYERS = 2
NUM_EPOCHS = 100
LR         = 0.005

model     = RNNLanguageModel(VOCAB_SIZE, EMBED_DIM,
HIDDEN_DIM, NUM_LAYERS)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LR)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
step_size=30, gamma=0.5)

print("–" * 40)
print("Training...")
print("–" * 40)

for epoch in range(NUM_EPOCHS):
  model.train()
  optimizer.zero_grad()

  logits, _ = model(X)
  loss      = criterion(logits, y)

  loss.backward()
  torch.nn.utils.clip_grad_norm_(model.parameters(),
max_norm=1.0)
  optimizer.step()
  scheduler.step()

  if (epoch + 1) % 10 == 0:
    preds    = torch.argmax(logits, dim=1)
    accuracy = (preds == y).float().mean().item() * 100
    print(f"Epoch [{epoch+1:3d}/{NUM_EPOCHS}] | Loss:
{loss.item():.4f} | Accuracy: {accuracy:.1f}%")

model.eval()
with torch.no_grad():
  logits, _ = model(X)
  loss      = criterion(logits, y)
  perplexity = torch.exp(loss).item()

print(f"\nFinal Perplexity: {perplexity:.4f}")

def generate_text(model, seed_words, num_words=10,
temperature=0.8):
  model.eval()

  seed_tokens = [vocab.get(w, 0) for w in seed_words]
  generated   = list(seed_words)
  input_seq   = torch.tensor([seed_tokens], dtype=torch.long)
  hidden      = None

  with torch.no_grad():
    for _ in range(num_words):
        logits, hidden = model(input_seq, hidden)
        logits     = logits / temperature
        probs      = torch.softmax(logits, dim=-1)
        next_token   = torch.multinomial(probs,
num_samples=1).item()
        next_word    = idx_to_word.get(next_token, '<UNK>')
        generated.append(next_word)
        input_seq    = torch.tensor([[next_token]],
dtype=torch.long)

    return ' '.join(generated)

print("\n" + "–" * 40)
print("Text Generation")
print("–" * 40)

seeds = [
    ["the", "cat", "sat"],
    ["the", "rat", "ran"],
    ["the", "mat", "was"],
]

for seed in seeds:
    output = generate_text(model, seed, num_words=6,
temperature=0.8)
    print(f"Seed: '{' '.join(seed)}' → {output}")

def predict_next_word(model, seed_words, top_k=3):
    model.eval()
    seed_tokens = [vocab.get(w, 0) for w in seed_words]
    input_seq   = torch.tensor([seed_tokens], dtype=torch.long)

    with torch.no_grad():
        logits, _ = model(input_seq)
        probs     = torch.softmax(logits, dim=-1).squeeze()
        top_probs, top_indices = torch.topk(probs, top_k)

    print(f"\nSeed: '{' '.join(seed_words)}'")
    print(f"Top-{top_k} next word predictions:")
    for prob, idx in zip(top_probs, top_indices):
        print(f"  '{idx_to_word[idx.item()]}' →
{prob.item()*100:.2f}%")

predict_next_word(model, ["the", "cat", "sat"])
predict_next_word(model, ["the", "rat", "ran"])
```

Observation

The RNN language model learned the text patterns and predicted the next word based on context. Loss decreased during training, showing effective learning. The model generated meaningful but slightly repetitive sentences due to the small dataset.



```
Vocabulary Size: 12
Vocabulary: {'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'mat': 5, 'ate': 6, 'rat': 7, 'ran': 8, 'from': 9, 'was': 10, 'flat': 11, '<PAD>': 0}

Encoded text: [1, 2, 3, 4, 1, 5, 1, 2, 6, 1, 7, 1, 7, 8, 9, 1, 2, 1, 5, 10, 11]

Total sequences: 18
Sample (input → target): ([1, 2, 3], 4)

________________________________________
Training...
________________________________________
Epoch [ 10/100] | Loss: 1.5439 | Accuracy: 50.0%
Epoch [ 20/100] | Loss: 0.4591 | Accuracy: 94.4%
Epoch [ 30/100] | Loss: 0.0572 | Accuracy: 100.0%
Epoch [ 40/100] | Loss: 0.0252 | Accuracy: 100.0%
Epoch [ 50/100] | Loss: 0.0152 | Accuracy: 100.0%
Epoch [ 60/100] | Loss: 0.0088 | Accuracy: 100.0%
Epoch [ 70/100] | Loss: 0.0061 | Accuracy: 100.0%
Epoch [ 80/100] | Loss: 0.0062 | Accuracy: 100.0%
Epoch [ 90/100] | Loss: 0.0060 | Accuracy: 100.0%
Epoch [100/100] | Loss: 0.0059 | Accuracy: 100.0%

Final Perplexity: 1.0036

________________________________________
Text Generation
________________________________________
Seed: 'the cat sat' → the cat sat on the rat the rat the
Seed: 'the rat ran' → the rat ran from the rat the rat the
Seed: 'the mat was' → the mat was flat flat cat ate ate the

Seed: 'the cat sat'
Top-3 next word predictions:
  'on' → 99.29%
  'the' → 0.35%
  'mat' → 0.22%

Seed: 'the rat ran'
Top-3 next word predictions:
  'from' → 99.52%
  'the' → 0.19%
  'ran' → 0.12%
```