Remembering the Past: Recurrent Neural Networks

In the previous lectures, we built feed-forward neural networks that process each input independently—an image goes in, a prediction comes out, and the network has no memory of what it saw before. But language doesn't work that way. To predict the next word in a sentence, you need to remember what came before. In this assignment, you'll explore why sequential data requires a fundamentally different architecture, and how Recurrent Neural Networks (RNNs) solve this problem by maintaining a hidden state that carries information forward through time.

Learning Objectives: By completing this assignment, you will (hopefully) be able to:
- Explain why feed-forward networks are insufficient for sequential data like text
- Describe the role of the hidden state in an RNN and how it carries information forward
- Trace data through the RNN equations step by step
- Interpret training dynamics of a character-level language model
- Connect the RNN architecture to the broader goal of building a text generator

Submission Guidelines:

- Please type your answers in the boxes underneath the questions.
- Submit your answers as a PDF document on Canvas.
- You may include drawings or screenshots when helpful.
- Show your work and explain your reasoning — partial credit is available.

Grading Rubric:

Section	Points
Task 01: Why Sequence Matters	1 point
Task 02: The RNN Architecture	1.5 points
Task 03: Training a Language Model	1.5 points
Task 04: Generation and the Big Picture	1 point
Total	**5 points**

Task 01: Why Sequence Matters (1 point)

In Lecture 11, we discussed a key difference between image data and text data. When classifying images of handwritten digits (MNIST), the order in which images are presented doesn't matter—each image is independent. But for text, what came before matters a LOT.

a.) Consider the sentence: "I like cute kittens and ___". What word might come next? Now consider just the word "and ___" with no prior context. Explain why the words that came before "and" are essential for predicting what comes next, and give two plausible completions that depend on the earlier context. (0.5 pts.)

b.) In our feed-forward networks (like the MNIST classifier or the CIFAR-10 network), each input is processed completely independently—the network has no memory of previous inputs. Explain why this architecture fundamentally cannot generate text. What specific capability is missing? (0.5 pts.)

Task 02: The RNN Architecture (1.5 points)

A Recurrent Neural Network solves the memory problem by maintaining a hidden state h that gets updated at each time step. The key equations are:

$$h_t = \tanh(h_{t-1} \cdot w_h + e_{x_t} \cdot w_x + b_h)$$

$$y_{t+1} = \text{softmax}(h_t \cdot w_y + b_y)$$

where h_t is the hidden state at time t, e_{x_t} is the embedding of the current input token, w_h, w_x, and w_y are weight matrices, and y_{t+1} is the predicted probability distribution over the next token.

a.) The hidden state h_t depends on both the previous hidden state h_{t-1} and the current input e_{x_t}. Explain in your own words what this means conceptually. What "information" does the hidden state carry, and why does it need both pieces (the memory and the new input)? (0.5 pts.)

b.) In the Lecture 11 notebook, the RNN model has the following architecture: (0.5 pts.)

```
self.embedding = nn.Embedding(512, 128)
self.rnn       = nn.RNN(128, 64, batch_first=True)
self.output    = nn.Linear(64, 512)
```

The vocabulary has 512 tokens. Explain what each of the three layers does, and identify the dimensions at each stage: What size vector goes in, and what size comes out? (Hint: trace a single token through the network.)

c.) Compare the RNN equation $h_t = \tanh(h_{t-1} \cdot w_t + e_{x_t} \cdot w_x + b_t)$ to the feed-forward equation $\hat{y} = X\omega^T + b$. What is similar? What is the crucial difference that gives the RNN its "memory"? How does the tanh activation function play the same role as ReLU did in our feed-forward networks? (0.5 pts.)

Task 03: Training a Language Model (1.5 points)

In Lecture 11, we trained our RNN on Shakespeare's text. At each epoch, we prompted the model with "hello there" and watched its completions evolve. Here are some snapshots:

Epoch	Loss	Perplexity	Sample Output
1	4.33	76.09	"hello there imhy."
5	3.29	26.97	"hello there ruiting constile."
10	3.16	23.52	"hello there hath way all."
25	3.04	21.00	"hello there hath sa, the clakes lost;"

a.) Describe the progression you see across these four snapshots. How does the quality of the generated text change? Be specific: compare the early outputs (epoch 1) to the later outputs (epoch 25) in terms of whether they look like real words, real sentences, and real Shakespeare. (0.5 pts.)

b.) The "perplexity" metric starts at 76.09 and drops to 21.00. Perplexity can be interpreted as: "on average, the model is as confused as if it were choosing randomly between N equally likely options." Using this interpretation, explain what a perplexity of 76 means at epoch 1 versus a perplexity of 21 at epoch 25. Is a perplexity of 21 good for a vocabulary of 512 tokens? Why? (0.5 pts.)

c.) The training process works by feeding the model a sequence of tokens and asking it to predict the next token at each position. For example, given "I like cute", the model should predict "like" after "I", "cute" after "I like", and "kittens" after "I like cute". This is called next-token prediction. Explain why this training strategy is well-suited for language: why does predicting the next word force the model to learn something useful about language? (0.5 pts.)

Task 04: Generation and the Big Picture (1 point)

Once the RNN is trained, we can use it to generate new text one token at a time.

a.) The generation process works as follows: (1) feed in a prompt, (2) take the model's output probabilities for the next token, (3) sample a token from those probabilities, (4) feed that token back in as the new input, (5) repeat. Explain why the hidden state h is critical to this process. What would happen if we reset h to zero before generating each new token instead of carrying it forward? (0.5 pts.)

b.) We've now seen three architectures in this course: single linear layers (Lecture 09), multi-layer feed-forward networks (Lecture 10), and RNNs (Lecture 11). In 3–4 sentences, explain the progression. What limitation of each architecture motivated the next one? How does each new architecture solve a problem the previous one couldn't? (0.5 pts.)