Assignment 5

CSD4999 / Protopsaltis Ioannis

June 3, 2025

1 Part A:

1.1 A

RNNs are pretty neat when you're dealing with sequential data like text or time series. The core concept revolves around maintaining a "memory" through hidden states that evolve as we process each element in the sequence.

Here's how it works at each time step:

- We feed in the current input
- Combine it with what we remember from before (the hidden state)
- Generate both a new hidden state and an output

This mechanism allows the network to build up context as it processes a sequence.

1.2 B

Vanishing Gradients: During backpropagation, gradient signals can become extremely small, making it difficult to update weights effectively. This causes the model to "forget" information from earlier time steps.

Exploding Gradients: The flip side - gradients can grow exponentially large, leading to unstable training. We handled this using gradient clipping techniques.

Limited Long-term Memory: Despite having a memory mechanism, standard RNNs struggle to maintain information over long sequences, typically focusing on recent inputs.

Sequential Processing: Each step depends on the previous one, making parallelization difficult and training slower compared to other architectures.

These limitations explain why researchers developed improved variants like LSTMs and GRUs, or moved toward attention-based models like Transformers for more complex applications.

1.3 C

For bird species classification, we're dealing with 2D spatial data where the relationships are primarily visual and spatial rather than sequential. CNNs are much better suited for this type of pattern recognition.

Movie review sentiment analysis, however, involves processing sequences of words where order is crucial. Consider the difference between "not good" versus "good" - the sequential nature of language makes RNNs (or their improved variants) the natural choice here.

1.4 D

When loss increases and weights become NaN values, we're likely experiencing exploding gradients. During backpropagation, gradient magnitudes grow exponentially, causing weight updates to overflow.

Solutions:

- Gradient clipping
- Reducing the learning rate
- Better weight initialization strategies
- Using more robust architectures like LSTMs or GRUs

1.5 E

Standard RNNs suffer from vanishing gradients, making it hard to capture long-range dependencies in sequences. GRUs address this through two gating mechanisms:

Update Gate: Controls how much of the previous hidden state to retain.

Reset Gate: Determines how much past information to discard when computing the new candidate state.

These gates create pathways for information to flow unchanged across multiple time steps, helping preserve gradients and enabling the network to learn longer-term patterns.

2 Part B:

2.1 A

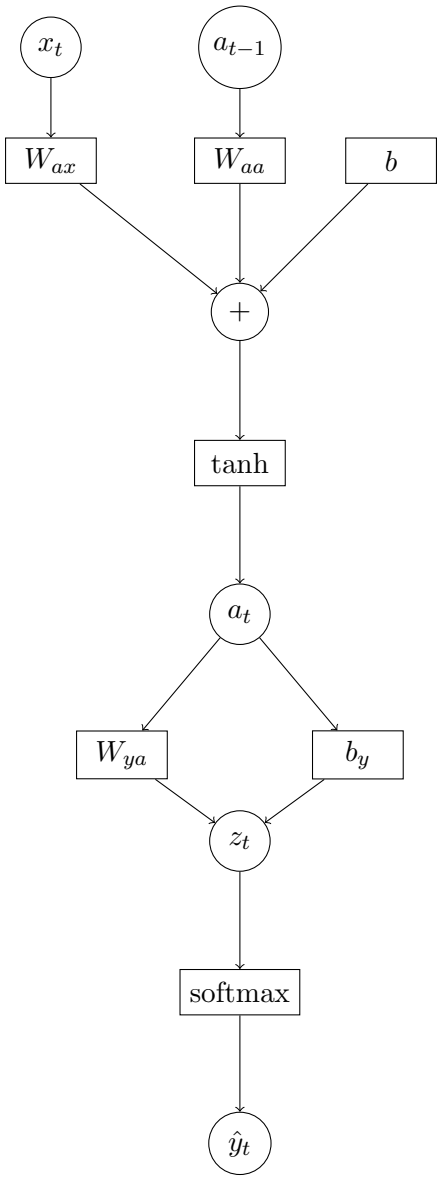

Figure 1: RNN Computational Graph

Forward Pass:

$$u^{\langle t \rangle} = W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b \tag{1}$$

$$a^{\langle t \rangle} = \tanh(u^{\langle t \rangle}) \tag{2}$$

$$z^{\langle t \rangle} = W_{ya}a^{\langle t \rangle} + b_y \tag{3}$$

$$\hat{y}^{\langle t \rangle} = \text{softmax}(z^{\langle t \rangle}) \tag{4}$$

The total cross-entropy loss across a sequence of length T:

$$L = -\sum_{t=1}^{T} \log(\hat{y}_{i_t^*}^{\langle t \rangle}) \tag{5}$$

where $y_{i_t^*}^{\langle t \rangle} = 1$ (one-hot encoding).

Backward Pass:

1. Output error: $dz^{(t)} = \hat{y}^{(t)} - y^{(t)}$
2. Output layer gradients: $dW_{ya} += dz^{(t)}(a^{(t)})^T$, $db_y += dz^{(t)}$
3. Hidden state backprop: $dtahn^{(t)} = W_{ya}^T dz^{(t)} + dtahn^{(t+1)}$
4. Activation backprop: $du^{(t)} = dtahn^{(t)} \odot (1 - (a^{(t)})^2)$
5. Weight gradients:

$$dW_{ax} += du^{(t)}(x^{(t)})^T \tag{6}$$

$$dW_{aa} += du^{(t)}(a^{(t-1)})^T \tag{7}$$

$$db += du^{(t)} \tag{8}$$

6. Gradient clipping and parameter updates

2.2 B

We implement gradient clipping to prevent the exploding gradient problem. When gradients grow too large during backpropagation, they can cause training instability and numerical overflow. By constraining each gradient component to a fixed range, we maintain stable training dynamics and reasonable parameter updates.

2.3 C

The generated chemical names show interesting patterns - many end with "-ium" suffixes, which is common in our training data and real elements. This suggests the model has learned typical formation patterns for chemical names rather than just memorizing examples.

Some generated names like "Tathermium" or "Taccadium" sound quite plausible. Overall, the model demonstrates pattern learning rather than simple copying.

2.4 D

Training on the movie titles dataset presents several difficulties. With only around 250 titles, there's insufficient data for effective learning. Movie titles are also more complex than chemical names - they contain mixed case, punctuation, numbers, and multiple words, creating a more challenging learning task.

The simple RNN architecture struggles with these complexities, often generating fragmented or nonsensical outputs. While it picks up common words like "the", "of" and "if" it can't produce coherent, realistic movie titles.

2.5 E

For better performance on movie titles, several approaches could help:

- Use LSTM or GRU cells instead of vanilla RNNs for better sequence modeling
- Increase hidden layer size or add multiple RNN layers
- Implement data augmentation techniques
- Consider pre-training on larger text corpora
- Apply regularization techniques like dropout

2.5.1 Bonus B

Both GRU and LSTM are designed to handle long-term dependencies better than standard RNNs. LSTM uses three gates (input, forget, output) plus a separate cell state for long-term memory storage. GRU simplifies this with two gates (update, reset) and no separate cell state.

This makes GRU computationally more efficient with fewer parameters, while LSTM potentially offers better long-term memory capabilities when computational resources aren't a constraint. The choice often depends on your specific application requirements and available resources.