Deep Learning

homework 4

Amirparsa Bahrami 401101332
github repository

Question 1

1. *Structure of the Network*

- **Input Unit:** The input values (x_t) are integers.
- **Linear Hidden Unit:** This unit computes a linear combination of the inputs and the previous hidden state.
- **Logistic Output Unit:** The logistic unit applies the sigmoid activation function to the weighted output of the hidden layer.

2. *Sigmoid Activation Function*

The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

3. *Computation in the Network*

At each time step t, the input x_t is fed into the network. The weights and biases are defined as:

- Weight from the input to the linear hidden unit: 1
- Weight from the previous hidden state (recurrent connection): -1
- Weight from the hidden linear unit to the logistic output unit: 10

4. *Hidden Unit Computation*

The linear hidden unit computes:

$$h_t = x_t + (-1)h_{t-1}$$

where h_t is the value of the hidden state at time t.

5. *Output Unit Computation*

The logistic output unit computes:

$$y_t = \sigma(10 \cdot h_t)$$

where σ is the sigmoid activation function.

6. *Final Time Step*

Since the sequence length n is even, at the final time step $t = n$, the output is given by:

$$y_n = \sigma(10 \cdot h_n)$$

where h_n is the final hidden state after processing the entire sequence.

7. Conclusion

The network computes a function of the inputs where the hidden state h_t is recursively calculated as a combination of the current input x_t and the previous hidden state. The final output is a scaled and squashed value (between 0 and 1) due to the sigmoid function applied to $10 \cdot h_n$. Essentially, this network models a weighted sum of inputs with recurrent feedback and outputs a non-linear transformation of the final state.

Question 2

Two issues with using one-hot encoding for word representation:

1. **High Dimensionality:**
   One-hot encoding creates a vector with a dimension equal to the size of the vocabulary. For large vocabularies (e.g., in natural language processing), this results in extremely high-dimensional vectors, which are inefficient to store and process. It increases memory usage and computational cost.

2. **Lack of Semantic Information:**
   One-hot encoded vectors do not capture any semantic or contextual relationships between words. For instance, words like "king" and "queen" or "cat" and "dog" are completely independent and equidistant in one-hot encoding, even though they may have strong semantic similarities. This limits the effectiveness of models relying on such representations.

Question 3

(a) Gradient of h_t in terms of h_{t+1}

To compute the gradient of h_t in terms of h_{t+1}, we use the chain rule. The gradient of the loss L with respect to h_t is given by:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t}.$$

The relationship between h_t and h_{t+1} depends on the weight W and the derivative of the activation function. For a sigmoid activation function, the derivative is:

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)).$$

Thus, the gradient can be expressed as:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \cdot W \cdot \sigma'(h_t),$$

where W is the weight connecting h_t to h_{t+1}, and $\sigma'(h_t)$ is the derivative of the sigmoid activation.

(b) Gradient of h_0 in terms of h_T

To express $\frac{\partial L}{\partial h_0}$ in terms of $\frac{\partial L}{\partial h_T}$, we recursively propagate the gradient through all time steps from 1 to T. Using the chain rule repeatedly:

$$\frac{\partial L}{\partial h_0} = \frac{\partial L}{\partial h_T} \cdot \prod_{t=1}^{T} \frac{\partial h_t}{\partial h_{t-1}}.$$

From part (a), we know:

$$\frac{\partial h_t}{\partial h_{t-1}} = W \cdot \sigma'(h_{t-1}).$$

Substituting this back, we get:

$$\frac{\partial L}{\partial h_0} = \frac{\partial L}{\partial h_T} \cdot \prod_{t=1}^{T} \left(W \cdot \sigma'(h_{t-1})\right).$$

This product term, $\prod_{t=1}^{T} (W \cdot \sigma'(h_{t-1}))$, shows how the gradient accumulates as it propagates backward through time. If $|W \cdot \sigma'(h_{t-1})| < 1$, the gradient will vanish exponentially. If $|W \cdot \sigma'(h_{t-1})| > 1$, the gradient will explode exponentially.

(c) Methods to Prevent Gradient Fading and Explosion

(i) Proper Weight Initialization

To prevent gradient fading or explosion, weights (W) must be initialized carefully. From part (b), the gradient at h_0 involves the term $(W \cdot \sigma'(h))^T$. To ensure stability, we need:

$$|W \cdot \sigma'(h)| \approx 1.$$

For a sigmoid activation function, $\sigma'(h) \in [0, 0.25]$. To prevent gradient explosion, the maximum value of W should satisfy:

$$|W| \leq \frac{1}{0.25} = 4.$$

Thus, initializing W within the range $[-4, 4]$ ensures that gradients neither vanish nor explode from the start.

▬ *(ii) Skip-Connections to Prevent Gradient Fading*

Skip connections allow the gradient to bypass intermediate layers, reducing the effect of gradient fading. For example, consider a skip connection where h_t is directly connected to h_{t+2} in addition to h_{t+1}. In this case, the gradient of h_t in terms of h_{t+2} becomes:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \cdot W \cdot \sigma'(h_t) + \frac{\partial L}{\partial h_{t+2}} \cdot W_s \cdot \sigma'(h_t),$$

where W_s is the weight for the skip connection.

By allowing gradients to flow more directly to earlier layers, skip connections reduce the risk of vanishing gradients, as the gradient signal can bypass intermediate transformations.

▬ *(iii) Gradient Clipping to Prevent Gradient Explosion*

Gradient clipping is used to cap the magnitude of gradients during backpropagation. It is divided into two methods:

1. Clipping by Value: In this method, each component of the gradient is clipped to lie within a specified range $[-v, v]$. For example, if a gradient component exceeds v, it is set to v. If it is less than $-v$, it is set to $-v$.

Advantage: Simple to implement and ensures that no gradient component becomes excessively large.

2. Clipping by Norm: Here, the entire gradient vector is rescaled if its norm exceeds a specified threshold v. If the norm of the gradient $\|\mathbf{g}\|$ is greater than v, the gradient is scaled as:

$$\mathbf{g} = \mathbf{g} \cdot \frac{v}{\|\mathbf{g}\|}.$$

Advantage: Preserves the relative proportions of gradient components while ensuring the overall magnitude is controlled.

Comparison: Clipping by norm is more commonly used because it maintains the directionality of the gradient vector, whereas clipping by value may distort the gradient directions.

Question 4

(a) Problems with Sequence Generation and the Teacher Forcing Method

Problems in sequence generation using the crude strategy: In the crude sequence generation strategy, at each time step $t+1$, the token generated by the decoder at time t is fed as input to the decoder for the next time step. This approach introduces the following issues:

1. **Error Accumulation:** - If the decoder generates incorrect tokens at any time step t, these incorrect tokens are used as input for subsequent time steps. - This results in a cascade of errors, leading to poor-quality outputs.

2. **Training-Testing Mismatch:** - During training, the model uses ground-truth tokens as inputs for the next step (teacher forcing), while during inference, the model relies on its own predictions. This mismatch between training and inference can degrade performance as the model does not learn to recover from its own mistakes or handle noisy inputs.

Teacher Forcing Method: Teacher forcing is a training technique where, during training, the decoder is provided with the ground-truth token from the previous time step as input, instead of the token generated by the model.

How Teacher Forcing Solves These Problems:

1. **Faster and Stable Training:** - By providing the ground-truth token at each time step, the model avoids compounding errors during training, allowing it to learn a more accurate mapping between inputs and outputs.

2. **Prevents Error Accumulation During Training:** - Since the model does not rely on its own predictions during training, it avoids the issue of cascading errors, resulting in a more stable and efficient learning process.

(b) The Exposure Bias Problem

What is Exposure Bias?

1. **Definition:** Exposure bias occurs because, during training, the model is conditioned on ground-truth tokens at every time step, but during inference, it must rely on its own generated tokens as input. This creates a mismatch between the training and inference phases.

2. **Impact:** - The model becomes highly dependent on the ground-truth tokens during training and does not learn to recover from its own mistakes. - During inference, when the model generates incorrect tokens, it cannot recover effectively, as it has never encountered such scenarios during training.

(c) Scheduled Sampling

What is Scheduled Sampling? Scheduled sampling is a technique introduced to address the exposure bias problem by gradually transitioning the model from relying on ground-truth tokens to using its own predictions during training.

How Scheduled Sampling Works:

1. **Random Sampling of Inputs:** - At each time step during training, the decoder's input is either the ground-truth token (as in teacher forcing) or the model's own predicted token from the previous time step. - The choice between these two inputs is made probabilistically, based on a scheduling function.

2. **Scheduling Function:** - Initially, the model uses the ground-truth token with high probability (e.g., $p = 1$). - Over time, the probability of using the ground-truth token decreases, while the probability of using the model's predicted token increases. - This gradual shift ensures the model learns to handle both ideal (ground-truth) and noisy (predicted) inputs.

How Scheduled Sampling Reduces Exposure Bias:

1. **Bridges the Training-Testing Gap:** - By exposing the model to its own predictions during training, scheduled sampling reduces the mismatch between training and inference conditions.

2. **Improves Robustness:** - The model learns to handle noisy inputs and recover from its mistakes, making it more robust during inference.

3. **Smooth Transition:** - Scheduled sampling ensures a smooth transition from teacher forcing to self-reliance, balancing the benefits of stable training and exposure to real-world conditions.

Question 5

Problem Analysis

The network has the following structure:

- **Input (x):** A binary value (1 or 0) at each time step.
- **Hidden State (h):** Encodes the memory of whether the network has encountered a zero in the past.
- **Output (y):** Depends on the hidden state (h) and determines whether the output is 1 or 0.

Desired behavior:

1. If $x_t = 1$ for all previous time steps, the output $y_t = 1$.
2. If $x_t = 0$ at any time step, the output $y_t = 0$ from that time step onward.

Hidden State Update Equation

The hidden state h_t is determined by:

$$h_t = \sigma(w_1 x_t + w_2 h_{t-1} + b_2),$$

where:

- $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function.

To achieve the desired behavior:

1. When $x_t = 0$, $h_t = 0$.
2. When $x_t = 1$, $h_t = h_{t-1}$.

This behavior can be achieved with the following parameter settings:

$$w_1 = 5, \quad w_2 = 5, \quad b_2 = -7.$$

Explanation:

- When $x_t = 0$, $w_1 x_t + w_2 h_{t-1} + b_2 = -7$, which is a large negative value. Thus, $\sigma(-7) \approx 0$, ensuring $h_t = 0$.
- When $x_t = 1$ and $h_{t-1} = 1$, $w_1 x_t + w_2 h_{t-1} + b_2 = 5 + 5 - 7 = 3$, which is a positive value. Thus, $\sigma(3) \approx 1$, ensuring $h_t = 1$.

Output Equation

The output y_t is determined by:

$$y_t = \sigma(w_3 h_t + b_3).$$

To ensure $y_t = h_t$, we can set:

$$w_3 = 10, \quad b_3 = -5.$$

Explanation:

- When $h_t = 1$, $w_3 h_t + b_3 = 10 - 5 = 5$. Thus, $\sigma(5) \approx 1$, ensuring $y_t = 1$.
- When $h_t = 0$, $w_3 h_t + b_3 = 0 - 5 = -5$. Thus, $\sigma(-5) \approx 0$, ensuring $y_t = 0$.

Final Parameters

To achieve the desired behavior, set:

$$w_1 = 5, \quad w_2 = 5, \quad b_2 = -7, \quad w_3 = 10, \quad b_3 = -5.$$

Verification Example

For the input sequence $x = [1, 1, 1, 0, 1, 0, 0]$:

1. At $t = 1$: $x_1 = 1$, $h_1 = 1$, $y_1 = 1$.
2. At $t = 2$: $x_2 = 1$, $h_2 = 1$, $y_2 = 1$.
3. At $t = 3$: $x_3 = 1$, $h_3 = 1$, $y_3 = 1$.
4. At $t = 4$: $x_4 = 0$, $h_4 = 0$, $y_4 = 0$.
5. At $t = 5$: $x_5 = 1$, $h_5 = 0$, $y_5 = 0$.
6. At $t = 6$: $x_6 = 0$, $h_6 = 0$, $y_6 = 0$.
7. At $t = 7$: $x_7 = 0$, $h_7 = 0$, $y_7 = 0$.

The output sequence is $y = [1, 1, 1, 0, 0, 0, 0]$, which matches the desired behavior.

Question 6

Problem Analysis

The recurrent network must determine whether two binary sequences $x_1^{(t)}$ and $x_2^{(t)}$ are equal up to time t. Specifically:

1. $y_t = 1$ if $x_1^{(t)} = x_2^{(t)}$ for all time steps up to t.
2. $y_t = 0$ if $x_1^{(t)} \neq x_2^{(t)}$ at any step up to t.

The hidden states:

- $h_1^{(t)}$: Indicates whether $x_1^{(t)} = 0$ and $x_2^{(t)} = 0$.
- $h_2^{(t)}$: Indicates whether $x_1^{(t)} = 1$ and $x_2^{(t)} = 1$.

Network Equations

Hidden State Update

$$\mathbf{h}^{(t)} = \phi(W\mathbf{x}^{(t)} + \mathbf{b}),$$

where:

- $\mathbf{h}^{(t)} = \begin{bmatrix} h_1^{(t)} \\ h_2^{(t)} \end{bmatrix}$,
- $\mathbf{x}^{(t)} = \begin{bmatrix} x_1^{(t)} \\ x_2^{(t)} \end{bmatrix}$,
- W is a 2×2 matrix,
- $\mathbf{b}$ is a 2-dimensional bias vector,
- $\phi(z) = 1 \,\text{if}\, z > 0,\ 0 \,\text{if}\, z \leq 0$ (activation function).

Output Update

$$y_t = \begin{cases} \phi(\mathbf{v}^\top \mathbf{h}^{(t)} + c_0), & \text{for } t = 1, \\ \phi(\mathbf{v}^\top \mathbf{h}^{(t)} + r y_{t-1} + c), & \text{for } t > 1, \end{cases}$$

where:

- $\mathbf{v}$ is a 2-dimensional vector,
- c_0, c, r are scalar values.

Parameter Selection

Hidden Layer (W and b) The hidden states $h_1^{(t)}$ and $h_2^{(t)}$ behave as follows:

1. $h_1^{(t)} = 1$ if $x_1^{(t)} = 0$ and $x_2^{(t)} = 0$, otherwise $h_1^{(t)} = 0$.
2. $h_2^{(t)} = 1$ if $x_1^{(t)} = 1$ and $x_2^{(t)} = 1$, otherwise $h_2^{(t)} = 0$.

This can be achieved by setting:

$$W = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Output Layer $(\mathbf{v}, c_0, r, c)$ The output y_t is determined as follows:

1. At $t = 1$:
$$y_1 = 1 \text{ if } x_1^{(1)} = x_2^{(1)}, \text{ otherwise } y_1 = 0.$$
Use:
$$\mathbf{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad c_0 = -1.$$

2. At $t > 1$:
$$y_t = 1 \text{ if } y_{t-1} = 1 \text{ and } x_1^{(t)} = x_2^{(t)}, \text{ otherwise } y_t = 0.$$
Use:
$$r = 1, \quad c = -1.$$

Explanation of Parameters

Hidden Layer

- For $h_1^{(t)} = 1$ when $x_1^{(t)} = 0$ and $x_2^{(t)} = 0$:
$$W\mathbf{x}^{(t)} + \mathbf{b} = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$
Since $1 > 0$ and $-1 \leq 0$, $h_1^{(t)} = 1, h_2^{(t)} = 0$.

- For $h_2^{(t)} = 1$ when $x_1^{(t)} = 1$ and $x_2^{(t)} = 1$:
$$W\mathbf{x}^{(t)} + \mathbf{b} = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$
Since $-1 \leq 0$ and $1 > 0$, $h_1^{(t)} = 0, h_2^{(t)} = 1$.

Output Layer

- At $t = 1$, $y_1 = 1$ if $h_1^{(1)} + h_2^{(1)} = 1$ (i.e., one of the hidden states is active), otherwise $y_1 = 0$.
- For $t > 1$, $y_t = 1$ only if $y_{t-1} = 1$ and one of the hidden states is active.

Verification Example

For the input sequences:
$$x_1 = [1, 1, 0, 0], \quad x_2 = [1, 1, 0, 1],$$
the output y_t at each time step is:
$$y = [1, 1, 1, 0].$$

1. At $t = 1$, $x_1^{(1)} = x_2^{(1)} = 1$, so $y_1 = 1$.
2. At $t = 2$, $x_1^{(2)} = x_2^{(2)} = 1$, so $y_2 = 1$.
3. At $t = 3$, $x_1^{(3)} = x_2^{(3)} = 0$, so $y_3 = 1$.
4. At $t = 4$, $x_1^{(4)} = 0 \neq x_2^{(4)} = 1$, so $y_4 = 0$.

The network behaves as expected.

End of homework 4