Deep learning

DR.Fatemi Zadeh

electrical engineering department

Ahmadreza Majlesara 400101861

computer assignment repository

assignment 4

December 31, 2024

Deep learning

assignment 4

Ahmadreza Majlesara 400101861
computer assignment repository

Question 1

Determine what the following network computes. (To be more precise, determine the output function calculated by the unit at the final time step.) The size of other outputs is not important. All biases are zero. Assume the inputs are integers, and the length of the input sequence is even. Also, consider the activation function to be a sigmoid.

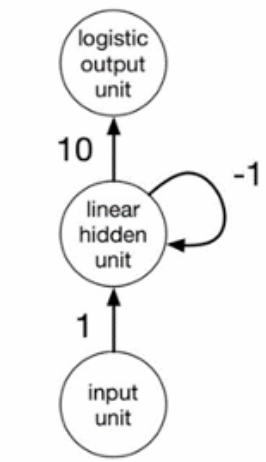

Figure 1. Network of Question 1

solution

Let the input sequence be denoted as $x_1, x_2, \ldots, x_T$, where T is the length of the input sequence (assumed to be even). The hidden state at time t is denoted as h_t, and it is updated as follows:

$$h_t = x_t - h_{t-1}$$

where $h_0 = 0$ (since all biases are zero).
At the final time step T, the hidden state is:

$$h_T = x_T - h_{T-1}$$

Substituting recursively, we find:

$$h_T = x_T - (x_{T-1} - (x_{T-2} - \cdots - (x_2 - x_1)))$$

This simplifies to:

$$h_T = x_T - x_{T-1} + x_{T-2} - x_{T-3} + \cdots + (-1)^{T+1} x_1$$

The pattern indicates that h_T is the alternating sum of the input sequence $x_1, x_2, \ldots, x_T$.
Next, the output of the logistic unit is given by:

$$y_T = \sigma(10 h_T)$$

Substituting h_T, the output becomes:

$$y_T = \sigma\left(10 \cdot \sum_{t=1}^{T} (-1)^{t+1} x_t\right)$$

where $\sigma(x)$ is the sigmoid activation function.
Thus, the network computes the alternating sum of the input sequence, scales it by 10, and applies the sigmoid function.

Question 2

Mention two issues that exist when using the one-hot vector for word representation.

solution

Firstly, one-hot encoding results in very high-dimensional vectors. The size of the vector depends on the vocabulary size, which can be enormous for natural language datasets. This high dimensionality makes computations inefficient and requires a lot of memory, which is particularly problematic for tasks involving large vocabularies.
Secondly, one-hot vectors do not capture any semantic or contextual relationships between words. For instance, the words ”king” and ”queen” are treated as entirely independent and unrelated, even though they share a semantic connection. This lack of meaning in the representation severely limits the model's ability to generalize and understand linguistic patterns.
These limitations highlight the need for more efficient and meaningful word representations, such as dense embeddings learned through methods like Word2Vec or GloVe.

Question 3

Given the following recurrent neural network diagram, answer the questions below. Note that for simplicity, all values (inputs, weights, and outputs) are scalar values. Also, assume that all activation functions are sigmoid functions.

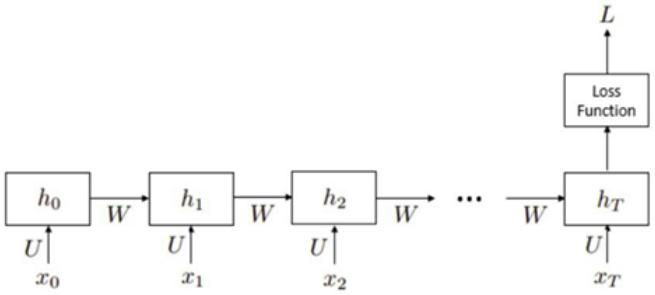

Figure 2. Network Architecture

part A

Write the gradient of h_t with respect to h_{t+1}, i.e., $\frac{\partial L}{\partial h_t}$ in terms of $\frac{\partial L}{\partial h_{t+1}}$ $(1 \leq t \leq T-1)$.

solution

based on the Architecture we have:

$$h_t = \sigma(Ux_t + Wh_{t-1}),$$

To calculate $\frac{\partial L}{\partial h_t}$, the chain rule gives:

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

Using the recurrence relation for h_{t+1}:

$$h_{t+1} = \sigma(Ux_{t+1} + Wh_t),$$

the derivative of h_{t+1} with respect to h_t is:

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

where $\sigma'(z) = \sigma(z)(1-\sigma(z))$ is the derivative of the sigmoid function.
Substituting this back, we get:

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

so the final answer is:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \cdot \sigma(Ux_{t+1} + Wh_t) \cdot (1 - \sigma(Ux_{t+1} + Wh_t)) \cdot W.$$

part B

Using the relationship from part (a), write the gradient h_t in a chain-like form with respect to the gradient h_T.

solution

From part (a), we know the relationship for the gradient of L with respect to h_t in terms of $\frac{\partial L}{\partial h_{t+1}}$ is given by:

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

To express $\frac{\partial L}{\partial h_t}$ in terms of $\frac{\partial L}{\partial h_T}$, we use the recursive nature of this relationship:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \cdot \prod_{k=t+1}^{T} \left(W \cdot \sigma'(Ux_k + Wh_{k-1})\right).$$

This equation indicates that the gradient $\frac{\partial L}{\partial h_t}$ depends on the gradient at the final time step $\frac{\partial L}{\partial h_T}$ and accumulates a product of weights W and the sigmoid derivatives $\sigma'(Ux_k + Wh_{k-1})$ for all subsequent time steps from $t+1$ to T.
Thus, the chain-like form of the gradient is:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \cdot \prod_{k=t+1}^{T} \left(W \cdot \sigma'(Ux_k + Wh_{k-1})\right),$$

part C

One important method for preventing the vanishing and exploding gradients is to initialize the network weights correctly. Describe how to set the maximum value of W such that the gradients neither vanish nor explode. (Hint: Find the upper bound for the gradient h_t.)

solution

To determine the upper bound on W, we use the expression for the gradient $\frac{\partial L}{\partial h_t}$ derived earlier:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \cdot \prod_{k=t+1}^{T} \left(W \cdot \sigma'(Ux_k + Wh_{k-1})\right).$$

The derivative of the sigmoid function, $\sigma'(z) = \sigma(z)(1-\sigma(z))$, achieves its maximum value at $\frac{1}{4}$. Therefore, for any k, we have:

$$\sigma'(Ux_k + Wh_{k-1}) \leq \frac{1}{4}.$$

Substituting this upper bound into the gradient expression, the magnitude of $\frac{\partial L}{\partial h_t}$ can be bounded as:

$$\left|\frac{\partial L}{\partial h_t}\right| \leq \left|\frac{\partial L}{\partial h_T}\right| \cdot \prod_{k=t+1}^{T} |W| \cdot \frac{1}{4}.$$

Simplifying further, we obtain:

$$\left|\frac{\partial L}{\partial h_t}\right| \leq \left|\frac{\partial L}{\partial h_T}\right| \cdot |W|^{T-t} \cdot \left(\frac{1}{4}\right)^{T-t}.$$

part D

Another method to prevent the vanishing gradient problem is to use skip-connections. The following diagram shows that at each time step t, h_t is connected to h_{t+2} as well as h_{t+1}. Now, rewrite the gradient of h_t with respect to h_{t+2} and explain why this approach effectively reduces the vanishing gradient problem $(1 \leq t \leq T-2)$.

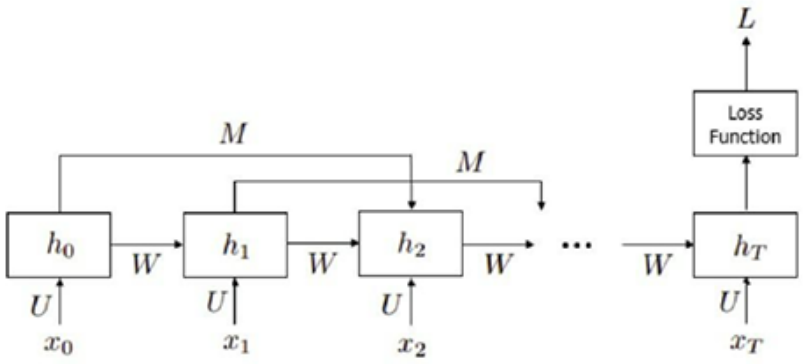

Figure 3. Network Architecture

solution

The gradient of the loss L with respect to h_t can be expressed using the chain rule, considering both the direct contribution from h_{t+2} and the indirect contribution through h_{t+1}. This is written as:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+2}} \cdot \frac{\partial h_{t+2}}{\partial h_t} + \frac{\partial L}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial h_t}.$$

The skip-connection introduces a direct path from h_t to h_{t+2}, which contributes to $\frac{\partial h_{t+2}}{\partial h_t}$. This term can be expressed as:

$$\frac{\partial h_{t+2}}{\partial h_t} = \sigma'(Wh_{t+1} + Mh_t + Ux_{t+2}) \cdot M + \sigma'(Wh_{t+1} + Mh_t + Ux_{t+2}) \cdot W \cdot \frac{\partial h_{t+1}}{\partial h_t}.$$

Substituting the gradient $\frac{\partial h_{t+1}}{\partial h_t}$, which is:

$$\frac{\partial h_{t+1}}{\partial h_t} = \sigma'(Wh_t + Mh_{t-1} + Ux_{t+1}) \cdot W,$$

we get:

$$\frac{\partial h_{t+2}}{\partial h_t} = \sigma'(Wh_{t+1} +$$

$$Mh_t + Ux_{t+2}) \cdot M + \sigma'(Wh_{t+1} + Mh_t + Ux_{t+2}) \cdot W \cdot \sigma'(Wh_t + Mh_{t-1} + Ux_{t+1}) \cdot W.$$

Finally, substituting this back into the expression for $\frac{\partial L}{\partial h_t}$, we get:

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+2}} \cdot (\sigma'(Wh_{t+1} + Mh_t + Ux_{t+2}) \cdot M$$

$$+\sigma'(Wh_{t+1} + Mh_t + Ux_{t+2}) \cdot W \cdot \sigma'(Wh_t + Mh_{t-1} + Ux_{t+1}) \cdot W)$$

$$+\frac{\partial L}{\partial h_{t+1}} \cdot \sigma'(Wh_t + Mh_{t-1} + Ux_{t+1}) \cdot W.$$

The inclusion of the term M in the gradient ensures a direct contribution to $\frac{\partial h_{t+2}}{\partial h_t}$, bypassing the dependence on repeated weight multiplications. This reduces the risk of the gradients vanishing, as the direct path provided by the skip-connection does not involve long chains of activations and weight products. This ensures more stable gradient flow and facilitates training for deeper recurrent networks or longer sequences.

part E

One of the ways to prevent gradient explosion is gradient clipping. This can be done in two ways: clipping by value and clipping by norm. Explain these two methods separately. Also, explain the advantages of clipping by norm over clipping by value.

solution

Gradient explosion occurs when gradients become too large, making training unstable. Gradient clipping solves this by limiting the size of gradients, either by value or by norm.
Clipping by value restricts each gradient component to a fixed range, such as $[-\text{threshold}, \text{threshold}]$. This method is simple but does not consider the overall gradient magnitude or direction.
Clipping by norm adjusts the entire gradient vector if its norm exceeds a threshold. The vector is scaled down to make its norm equal to the threshold, preserving the gradient's direction.
Clipping by norm is generally better because it ensures consistent gradient direction and controls the total gradient size, which is especially useful in large models.

Question 4

In this question, we want to familiarize ourselves with concepts in sequence-to-sequence (Seq2Seq) models, their advantages, and their disadvantages. In this question, we will study the concept of **teacher forcing**. To generate a sequence, we can consider a raw strategy where we generate token $t+1$ at time $t+1$ by encoding time t output as the input at time $t+1$. However, this has some problems.

part A

First, explain what these problems are, and then explain the **teacher forcing** method and how it resolves these issues.

solution

In sequence-to-sequence models, a common strategy is to use the output from time t as input for time $t+1$. This has problems because if the model makes a mistake, the error propagates to later steps, making the entire sequence worse. This issue, called error accumulation, is especially bad for long sequences. Also, during training, the model does not see its own outputs, which creates a mismatch between training and inference. This is known as exposure bias.
The **teacher forcing** method fixes these issues by feeding the true outputs (ground truth) from the training data as inputs at each step instead of the model's own outputs. This helps the model learn better because it always gets correct inputs during training. It also reduces exposure bias and speeds up learning.
However, teacher forcing has a limitation. During inference, the model still uses its own outputs as inputs, which it was not trained for. To handle this, techniques like scheduled sampling are used to mix ground truth and self-generated inputs during training.

part B

The main problem of **teacher forcing** is **exposure bias**. Explain this issue.

solution

The main problem of **teacher forcing** is exposure bias. During training, the model always uses the correct output (ground truth) as input, but during inference, it uses its own predictions. This mismatch makes the model unprepared for inference conditions, and errors in predictions can accumulate over time, especially in long sequences.

part C

One of the solutions to the **exposure bias** issue is a technique called **scheduled sampling**. Explain this technique and describe how it reduces the effect of exposure bias.

solution

Scheduled sampling is a technique used to reduce exposure bias in sequence-to-sequence models. Instead of always using the ground truth as input during training, scheduled sampling gradually introduces the model's own predictions as inputs. At the start of training, the model mostly uses the ground truth, but as training progresses, it increasingly uses its own predictions.
This helps the model adapt to inference conditions, where it relies entirely on its predictions. By exposing the model to its own outputs during training, scheduled sampling reduces the mismatch between training and inference, making the model more robust and less prone to error accumulation.

Question 5

Consider the recurrent network shown below. Set the weights and biases in such a way that the network output remains 1 as long as the input sequence contains ones, and the network output changes to 0 when the input changes to 0. For example, the network output for the input sequence $x = 1111010000$ is $y = 1111000000$.

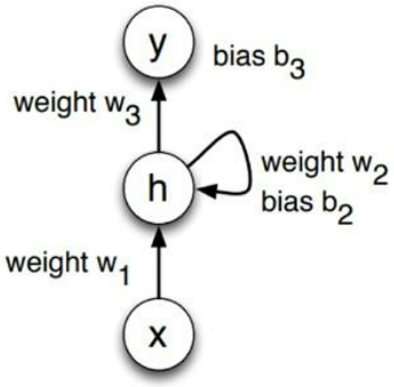

Figure 4. Recurrent network

solution

The hidden state h_t acts as a memory cell. It needs to retain the value 1 if $x_t = 1$ or if the previous hidden state h_{t-1} was already 1. When $x_t = 0$, h_t should decay to 0. To achieve this, the hidden state can be defined as:

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

where $\sigma(z)$ is the sigmoid activation function. Setting $w_1 = 1$, $w_2 = 1$, and $b_2 = -0.5$ ensures that h_t transitions smoothly. When $x_t = 1$, $h_t \approx 1$. When $x_t = 0$, h_t decays toward 0.
The output y_t should mirror the value of the hidden state h_t. The output is defined as:

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

To ensure $y_t \approx 1$ when $h_t = 1$ and $y_t \approx 0$ when $h_t = 0$, we can set $w_3 = 10$ and $b_3 = -5$. These values ensure a sharp transition between $y_t = 1$ and $y_t = 0$.
using this configuration for the input sequence $x = 1111010000$, we get this output:

$$y = 1111000000.$$

Question 6

Consider the following recurrent network. Assume that this network receives two sequences of zeros and ones and returns the number 1 if the two sequences are equal, and the number 0 otherwise.

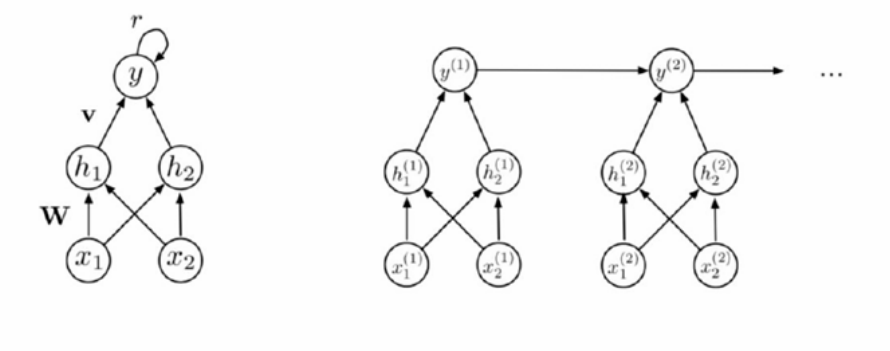

Figure 5. Recurrent network

The matrix W is a 2×2 matrix, and $\mathbf{v}$, $\mathbf{b}$ are 2-dimensional vectors, and c, r, and c_0 are scalars. Determine these parameters so that the network operates as described. (Hint: $y^{(t)}$, the output at time t, indicates whether the two sequences have been equal up to that point. The first hidden layer indicates whether the two inputs at time t were both zero, and the second hidden layer indicates whether the two inputs at time t were both one.)

$$h^{(t)} = \phi\left(Wx^{(t)} + \mathbf{b}\right)$$

$$y^{(t)} = \begin{cases} \phi\left(\mathbf{v}^\top h^{(t)} + ry^{(t-1)} + c\right) & \text{for } t > 1 \\ \phi\left(\mathbf{v}^\top h^{(t)} + c_0\right) & \text{for } t = 1 \end{cases}$$

$$\phi(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \le 0 \end{cases}$$

solution

The hidden states $h_1^{(t)}$ and $h_2^{(t)}$ represent whether the two inputs $x_1^{(t)}$ and $x_2^{(t)}$ satisfy specific conditions at time t:

$h_1^{(t)}$ indicates if both $x_1^{(t)} = 0$ and $x_2^{(t)} = 0$.
$h_2^{(t)}$ indicates if both $x_1^{(t)} = 1$ and $x_2^{(t)} = 1$.

To achieve this, we set the matrix W and bias $\mathbf{b}$ as:

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

This ensures that:

$$h_1^{(t)} = \phi(-x_1^{(t)} - x_2^{(t)} + 1),$$

which outputs 1 when $x_1^{(t)} = 0$ and $x_2^{(t)} = 0$, and

$$h_2^{(t)} = \phi(x_1^{(t)} + x_2^{(t)} - 1)$$

, which outputs 1 when $x_1^{(t)} = 1$ and $x_2^{(t)} = 1$.
The output $y^{(t)}$ should indicate whether the inputs have been equal up to time t.
For $t = 1$, the output depends only on the initial hidden state:

$$y^{(1)} = \phi(\mathbf{v}^\top h^{(1)} + c_0),$$

where $\mathbf{v} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and $c_0 = -0.5$. This ensures $y^{(1)} = 1$ if $h_1^{(1)} + h_2^{(1)} = 1$ (i.e., $x_1^{(1)} = x_2^{(1)}$), and $y^{(1)} = 0$ otherwise.
For $t > 1$, $y^{(t)}$ depends on both the current hidden state $h^{(t)}$ and the previous output $y^{(t-1)}$. The equation is:

$$y^{(t)} = \phi(\mathbf{v}^\top h^{(t)} + r y^{(t-1)} + c),$$

where $r = 2$ and $c = -2.5$. This ensures that $y^{(t)} = 1$ if the inputs match at time t and $y^{(t-1)} = 1$, maintaining consistency across all time steps.
as an example Consider the input sequences:

$$x_1 = [1,1,0,0,1], \quad x_2 = [1,1,0,0,0].$$

At each time step:
- $h_1^{(t)} = 1$ if $x_1^{(t)} = 0$ and $x_2^{(t)} = 0$,
- $h_2^{(t)} = 1$ if $x_1^{(t)} = 1$ and $x_2^{(t)} = 1$,
- $y^{(t)} = 1$ only if x_1 and x_2 have been equal up to time t.

The outputs are computed as follows:

$$h_1 = [0,0,1,1,0], \quad h_2 = [1,1,0,0,0],$$

$$y = [1,1,1,1,0].$$

The final output sequence is:

$$y = [1,1,1,1,0].$$