INF721 - Deep Learning
L13: Recurrent Neural Networks

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

1 Introduction to Sequential Problems

Sequential problems are a fundamental class of tasks in deep learning where the order and context of inputs matter. Unlike traditional problems where each input is processed independently, sequential problems require understanding temporal dependencies and maintaining context across a sequence of inputs.

2 Sequential Problems in Artificial Intelligence

Sequential problems are pervasive in AI applications, especially those involving natural language processing, speech recognition, and time series analysis. The table below shows examples of sequential problems and their input/output formats.

Examples of Sequential Problems

Problem	Input	Output
Speech Recognition	Audio signal	Word sequence
Sentiment Analysis	Text sentence	Rating (1-5)
Machine Translation	Source sentence	Target sentence
Image Captioning	Image	Description sentence
Music Generation	None/Prompt	Note sequence
Named Entity Recognition	Text	Entity labels

3 Limitations of Feed-forward Networks

Traditional feed-forward neural networks (MLPs) are inadequate for sequential problems for two key reasons:

- **Variable Input Sizes**: MLPs require fixed-size inputs, but sequences can have varying lengths.
- **Temporal Dependencies**: MLPs don't capture the order-dependent relationships between elements in a sequence.

4 Recurrent Neural Networks (RNNs)

The key idea behind RNNs is to maintain a hidden state that captures context across time steps. This allows RNNs to process sequences of varying lengths and model temporal dependencies. At each time step t:

$$\mathbf{h}^{<t>} = g_1(\mathbf{W}_h\mathbf{h}^{<t-1>} + \mathbf{W}_x\mathbf{x}^{<t>} + \mathbf{b}_h) \tag{1}$$

$$\mathbf{y}^{<t>} = g_2(\mathbf{W}_y\mathbf{h}^{<t>} + \mathbf{b}_y) \tag{2}$$

where:

- $\mathbf{x}^{<t>}$ is the input vector at time t
- $\mathbf{h}^{<t>}$ is the hidden state vector at time t
- $\mathbf{y}^{<t>}$ is the output vector at time t
- $\mathbf{W}_h$, $\mathbf{W}_x$, $\mathbf{W}_y$ are weight matrices
- $\mathbf{b}_h$, $\mathbf{b}_y$ are bias vectors
- g_1, g_2 are activation functions (typically tanh for g_1)

4.1 RNN Types

RNNs can be configured in different ways depending on the task:

- **Many-to-Many**
  - Input: Sequence $\{x^{<1>}, x^{<2>}, \ldots, x^{<T>}\}$ of size T
  - Output: Sequence $\{y^{<1>}, y^{<2>}, \ldots, y^{<T>}\}$ of same size T

- Example: Named Entity Recognition.

  In Named Entity Recognition, the input is a sentence (sequence of words), for example, "Lucas is a professor at UFV", and the output is a sequence of entity labels, for example, $\{1, 0, 0, 0, 0, 2\}$, where 1 represents a person, 2 represents an organization, and 0 represents other entities.

- **Many-to-One**

  - Input: Sequence $\{x^{<1>}, x^{<2>}, \ldots, x^{<T>}\}$ of size T_x
  - Output: A single label $y^{<}T_y >$ (binary, categorical, or continuous) after processing the entire sequence of length T_y
  - Example: Sentiment Analysis

    In Sentiment Analysis, the input is a sentence, for example, "This movie is great!", and the output is a single label, for example, 1 (positive sentiment) or 0 (negative sentiment). This problem can also be formulated as a regression task where the output is a continuous value, for example, between 0 and 5, representing the sentiment score.

- **One-to-Many**

  - Input: Single input vector x
  - Output: Sequence $\{y^{<1>}, y^{<2>}, \ldots, y^{<T>}\}$ of size T
  - Example: Image Captioning

    In Image Captioning, the input is an image, and the output is a sequence of words describing the image. For example, given an image of a cat, the output could be "a cat sitting on a chair".

- **Sequence-to-Sequence**

  - Input: Sequence $\{x^{<1>}, x^{<2>}, \ldots, x^{<T_x>}\}$ of size T_x
  - Output: Sequence $\{y^{<1>}, y^{<2>}, \ldots, y^{<T_y>}\}$ of size T_y (possibly different from T_x)
  - Example: Machine Translation

    In Machine Translation, the input is a sentence in one language, for example, "I am an student", and the output is a sentence in another language, for example, "Eu sou um estudante".

5 Training RNNs

Training RNNs involves processing the input sequence, computing the output sequence, and comparing it with the target sequence to compute the loss. Let's examine detailed calculations for both many-to-many and many-to-one scenarios.

1. **Many-to-Many**:

In many-to-many scenarios, the loss is computed for each time step and summed across all time steps. The total loss is the sum of losses at each time step:

$$L = \sum_{t=1}^{T} L_t(\mathbf{y}_t, \hat{\mathbf{y}}_t) \tag{3}$$

Example: Named Entity Recognition
Consider the sentence: "John works at Google"

- Input sequence $\mathbf{X}$: ["John", "works", "at", "Google"]
- Target sequence $\mathbf{Y}$: [1, 0, 0, 2] (1: Person, 0: Other, 2: Organization)
- Predicted probabilities $\hat{\mathbf{Y}}$ after softmax for each token:
  - "John": [0.75, 0.20, 0.05] (high confidence for Person)
  - "works": [0.15, 0.75, 0.10] (good confidence for Other)
  - "at": [0.05, 0.90, 0.05] (very high confidence for Other)
  - "Google": [0.10, 0.30, 0.60] (moderate confidence for Organization)

Using cross-entropy loss for each time step:

$$L_t = -\sum_{c=1}^{C} y_{t,c} \log(\hat{y}_{t,c}) \tag{4}$$

Let's calculate the loss for each word:
1. "John" (t=1, true label: Person):

$$\begin{aligned} L_1 &= -(1 \cdot \log(0.75) + 0 \cdot \log(0.20) + 0 \cdot \log(0.05)) \\ &= -\log(0.75) \\ &= 0.124 \end{aligned}$$

2. "works" (t=2, true label: Other):

$$\begin{aligned} L_2 &= -(0 \cdot \log(0.15) + 1 \cdot \log(0.75) + 0 \cdot \log(0.10)) \\ &= -\log(0.75) \\ &= 0.124 \end{aligned}$$

3. "at" (t=3, true label: Other):

$$\begin{aligned} L_3 &= -(0 \cdot \log(0.05) + 1 \cdot \log(0.90) + 0 \cdot \log(0.05)) \\ &= -\log(0.90) \\ &= 0.045 \end{aligned}$$

4. "Google" (t=4, true label: Organization):

$$\begin{aligned} L_4 &= -(0 \cdot \log(0.10) + 0 \cdot \log(0.30) + 1 \cdot \log(0.60)) \\ &= -\log(0.60) \\ &= 0.221 \end{aligned}$$

Total loss:

$$\begin{aligned} L &= L_1 + L_2 + L_3 + L_4 \\ &= 0.124 + 0.124 + 0.045 + 0.221 \\ &= 0.514 \end{aligned}$$

2. **Many-to-One**:

In many-to-one scenarios, the loss is computed at the final time step. The total loss is the loss at the final time step:

$$L = L_T(\mathbf{y}_T, \hat{\mathbf{y}}_T) \tag{5}$$

Example: Sentiment Analysis

Consider the review: "This movie is great!"

- Input sequence $\mathbf{X}$: ["This", "movie", "is", "great", "!"]
- Target y: 1 (Positive sentiment)
- Predicted probability $\hat{y}$: 0.85

Using binary cross-entropy loss:

$$\begin{aligned} L &= -(y \log(\hat{y}) + (1-y) \log(1-\hat{y})) \\ L &= -(1 \cdot \log(0.85) + (1-1) \log(1-0.85)) \\ &= -\log(0.85) - 0 \\ &= -(-0.07) \\ &= 0.07 \end{aligned}$$

Note: These calculations use natural logarithm (ln). In practice, many implementations use log base 2 or base 10, which would give different absolute values but maintain the same relative relationships.

5.1 Backpropagation Through Time (BPTT)

Training RNNs involves backpropagating errors through the temporal dimension. Key points:

1. Forward pass processes entire sequence
2. Backward pass computes gradients through all time steps
3. Weight updates affected by entire sequence history

Truncated BPTT

Due to computational constraints with long sequences:

1. Process sequence in chunks of size J
2. Maintain hidden state between chunks
3. Update weights after each chunk
4. Move window forward J steps

6 Practical Considerations

When implementing RNNs, it's important to consider several practical aspects to ensure efficient training and good performance.

6.1 Initialization

- Hidden state ($\mathbf{h}_0$) typically initialized to zero
- Weight matrices initialized randomly
- Bias terms typically initialized to zero

6.2 Activation Functions

- Hidden layer: typically tanh
- Output layer: depends on task
  - Classification: softmax
  - Regression: linear
  - Binary: sigmoid

7 Conclusion

RNNs are a powerful class of neural networks that can model sequential problems by capturing temporal dependencies and maintaining context across time steps. They are widely used in natural language processing, speech recognition, and time series analysis. Training RNNs involves backpropagating errors through time, which requires careful initialization and activation function choices.

In the next lecture, we will discuss the challenges of training RNNs, including gradient vanishing/exploding, and introduce advanced RNN architectures like LSTM and GRU.