# INF721 - Deep Learning
# L14: Recurrent Neural Networks (Part II)

Prof. Lucas N. Ferreira
Universidade Federal de Viçosa

2024/2

## 1 Introduction

Last lecture we introduced the concept of Recurrent Neural Networks (RNNs) and their applications in sequential data. In this lecture, we will discuss a very important application of RNNs: language modeling. We will also cover the challenges with basic RNNs and introduce advanced RNN architectures like LSTM and GRU.

## 2 Language Modeling: A Motivating Example

Language modeling is a fundamental task in natural language processing (NLP) that involves predicting the next word in a sequence of words. Formally, given a sequence of tokens $(x_1, x_2, ..., x_t)$, a language model computes the probability of the next token $x_{t+1}$:

$$P(x_{t+1}|x_t, x_{t-1}, ..., x_1) \tag{1}$$

The goal of language modeling is to learn the underlying structure of a language, which can be used for various NLP tasks like speech recognition, machine translation, and text generation.

## 3 RNN Language Model

For language modelling, we need a many-to-many RNN architecture that processes a sequence of words and predicts the next word at each time step. The basic RNN architecture consists of a single hidden layer:

$$\mathbf{h}_t = \tanh(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2}$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{W}_y\mathbf{h}_t + \mathbf{b}_y) \tag{3}$$

where:

- $\mathbf{h}_t$ is the hidden state at time t
- $\mathbf{x}_t$ is the input at time t
- $\mathbf{W}_x, \mathbf{W}_h, \mathbf{W}_y$ are weight matrices
- $\mathbf{b}_h, \mathbf{b}_y$ are bias vectors

## 4 Training an RNN for Language Modeling

We typically train language models using a very large corpus of text data, such as Wikipedia or the Gutenberg Project. Since it is unfeasible to train on the entire dataset at once, we split the data into smaller sequences of fixed length $T$ and train the model on these sequences.

### 4.1 Loss Function

For each sequence, we want to maximize the likelihood of the next token, as defined by a vocabulary $V$, so we minimize the categorical cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log(\hat{y}_{t,j}) \tag{4}$$

where:

- $T$ is the sequence length
- $|V|$ is the vocabulary size
- $y_{t,j}$ is the true probability (one-hot encoded)
- $\hat{y}_{t,j}$ is the predicted probability

### 4.2 Creating Examples and Mini-Batches

To take advantage of vectorization and speed up training, we typically optimize the parameters using Adam, which updates the weights for mini-batches of sequences. Since every sequence $s$ has the same length $T$, we can pack them into mini-batches where each example has an input sequence $x = s[0 : T-1]$ and $y = s[1 : T]$. For example, consider we wanted to model the Bob Dylan songs. For the sake of simplicity, let's assume our dataset $D$ is only the chorus of the song Like a Rolling Stone:

$$D = \text{“How does it feel, how does it feel?}$$
$$\text{To be without a home}$$
$$\text{Like a complete unknown, like a rolling stone“}$$

Consider a sequence length of site $T = 4$. To produce a mini-batchs of size 3 from this Dataset, we need to first to split the dataset into sequences of size $T = 4$. That would give as the following sentences:

$$s_1 = [\text{How, does, it, feel}]$$
$$s_2 = [\text{',', how, does, it}]$$
$$s_3 = [\text{feel, ?, }\backslash n\text{, To}]$$
$$s_4 = [\text{be, without, a, home}]$$
$$s_5 = [\backslash n\text{, Like, a, complete}]$$
$$s_6 = [\text{unknown, ',', like, a}]$$
$$s_7 = [\text{rolling, stone, }\backslash n]$$

It is common that when dividing the dataset into sequences, the last sequence will end up with a size less than $T$. There are two options to solve this problem: padding, or removing this sentence. Let's just remove it for simplicity. Now, we can create a training example for each sequence $s_i$ by using the first $T-1$ words as input and the last $T-1$ words as the target:

$$x_1 = [\text{How, does, it}],\ y_1 = [\text{does, it, feel}]$$
$$x_2 = [\text{',', how, does}],\ y_2 = [\text{how, does, it}]$$
$$x_3 = [\text{feel, ?, }\backslash n],\ y_3 = [\text{?, }\backslash n\text{, To}]$$
$$x_4 = [\text{be, without, a}],\ y_4 = [\text{without, a, home}]$$
$$x_5 = [\backslash n\text{, Like, a}],\ y_5 = [\text{Like, a, complete}]$$
$$x_6 = [\text{unknown, ',', like}],\ y_6 = [\text{', like, a}]$$

Now, we can pack these examples into mini-batches of size 3:

$$X = [[x_1, x_2, x_3], [x_4, x_5, x_6]]$$
$$Y = [[y_1, y_2, y_3], [y_4, y_5, y_6]]$$

### 4.3 Represening Words as Vectors

To train the RNN, we need to represent words as vectors. One common approach is to use one-hot encoding, where each word is represented as a binary vector with a 1 at the index of the word in the vocabulary. For example, we can uild a vocabulary $V$ from the dataset $D$ above and represent the example $x_1 = [\text{How, does, it}]$ as a matrix where each column is the one-hot encoding of the word:

$$V = \begin{bmatrix} \text{a} \\ \text{be} \\ \text{complete} \\ \text{does} \\ \text{feel} \\ \text{home} \\ \text{how} \\ \text{it} \\ \text{like} \\ \text{rolling} \\ \text{stone} \\ \text{to} \\ \text{unknown} \\ \text{without} \\ , \\ ? \\ \backslash n \end{bmatrix} \qquad \text{How, does, it} = [ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} ]$$

### 4.4 Gradient Descent and Backpropagation

With the loss function defined and the mini-batches created, we can now train the RNN using gradient descent. The gradients are computed using backpropagation through time (BPTT), which involves unrolling the RNN for each time step $1 \leq t \leq T$ and computing the gradients for each time step. The gradients are then averaged over the mini-batch and used to update the weights. The pseudo-code of the RNN training algorithm in the vectorized form is shown below:

Algorithm: Vectorized RNN Language Model

```
Initialize parameters $\mathbf{W}_x, \mathbf{W}_h, \mathbf{W}_y, \mathbf{b}_h, \mathbf{b}_y$
for each epoch do
  for each mini-batch $(X^{\{i\}}, Y^{\{i\}})$ do
    Initialize hidden states $h^{<0>}$ (batch size x hidden size) with zeros
    $\hat{H} = [], \hat{Y} = []$
    for $t = 1$ to $T$ do
      $h^{<t>} = \tanh(\mathbf{W}_x X^{\{i\}}[:, t, :] + \mathbf{W}_h h^{<t-1>} + \mathbf{b}_h)$
      $y^{<t>} = \text{softmax}(\mathbf{W}_y h + \mathbf{b}_y)$
      $\hat{H}.append(h^{<t>}), \hat{Y}.append(y^{<t>})$
    end for
    $\hat{H} = \text{stack}(\hat{H}), \hat{Y} = \text{stack}(\hat{Y})$
    Initialize gradients $\nabla_{\mathbf{W}_x}, \nabla_{\mathbf{W}_h}, \nabla_{\mathbf{W}_y}, \nabla_{\mathbf{b}_h}, \nabla_{\mathbf{b}_y}$ with zeros
    Initialize $\nabla_{\mathbf{h}^{<t-1>}} = 0$
    for $t = T$ to $1$ do
      $\nabla_y = \hat{Y}[t] - Y^{\{i\}}[t]$
      $\nabla_{\mathbf{W}_y} + = \nabla_y \cdot \hat{H}[t]$
      $\nabla_{\mathbf{b}_y} + = \nabla_y$
      $\nabla_{\mathbf{h}^{<t>}} = \nabla_y \cdot \mathbf{W}_y + \nabla_{\mathbf{h}^{<t-1>}}$
      $\nabla_{\mathbf{h}^{<t>}} = (1 - \hat{H}[t]^2) \odot \nabla_{\mathbf{h}^{<t>}}$
      $\nabla_{\mathbf{W}_h} + = \nabla_{\mathbf{h}^{<t>}} \cdot h^{<t-1>}$
      $\nabla_{\mathbf{b}_h} + = \nabla_{\mathbf{h}^{<t>}}$
      $\nabla_{\mathbf{W}_x} + = \nabla_{\mathbf{h}^{<t>}} \cdot X^{\{i\}}[:, t, :]$
      $\nabla_{\mathbf{h}^{<t-1>}} = \nabla_{\mathbf{h}^{<t>}} \cdot \mathbf{W}_h$
    end for
    Update parameters $\mathbf{W}_x, \mathbf{W}_h, \mathbf{W}_y, \mathbf{b}_h, \mathbf{b}_y$ using an optimizer
  end for
end for
```

## 5 Challenges with Basic RNNs

Basic RNNs have several limitations that make them unsuitable for learning long-term dependencies in sequences. Two common challenges are vanishing gradients and exploding gradients.

### 5.1 Vanishing Gradients

In long sequences, gradients can become extremely small due to repeated multiplication with small values during backpropagation. Looking at the RNN gradient calculation, we can see that the gradients $\nabla_{\mathbf{h}^{<t>}}$ are multiplied by the weight matrix $\mathbf{W}_h$ at each time step. If the weights are smaller than 1, the gradients will vanish. This makes it difficult for the model to learn long-term dependencies, as the gradients become too small to update the weights.

### 5.2 Exploding Gradients

Conversely, gradients can also grow exponentially if weights are larger than 1, leading to numerical instability during training.

## 6 Advanced RNN Architectures

To address the challenges with basic RNNs, several advanced architectures have been proposed, including Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). These architectures introduce gating mechanisms that control the flow of information in the network, allowing them to learn long-term dependencies more effectively.

### 6.1 Long Short-Term Memory (LSTM)

LSTM is a type of RNN that uses a more complex cell state to store information over long sequences. It has three gates (forget, input, and output) that control the flow of information in the network. The LSTM cell is defined as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{5}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{6}$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \tag{7}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{8}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{9}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{10}$$

where:

- $\mathbf{f}_t$ is the forget gate

- $\mathbf{i}_t$ is the input gate
- $\mathbf{o}_t$ is the output gate
- $\mathbf{c}_t$ is the cell state
- $\odot$ denotes element-wise multiplication

## 6.2 Gated Recurrent Unit (GRU)

GRU is a simplified version of LSTM with fewer parameters:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z) \tag{11}$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r) \tag{12}$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_h) \tag{13}$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{14}$$

## 6.3 Bidirectional RNNs

Bidirectional RNNs process sequences in both forward and backward directions, capturing context from both past and future tokens. To compute an output at time $t$, we concatenate the hidden states from the forward and backward RNNs:

$$\overrightarrow{\mathbf{h}}_t = \tanh(\overrightarrow{\mathbf{W}}_x \mathbf{x}_t + \overrightarrow{\mathbf{W}}_h \overrightarrow{\mathbf{h}}_{t-1} + \mathbf{b}_h) \tag{15}$$
$$\overleftarrow{\mathbf{h}}_t = \tanh(\overleftarrow{\mathbf{W}}_x \mathbf{x}_t + \overleftarrow{\mathbf{W}}_h \overleftarrow{\mathbf{h}}_{t-1} + \mathbf{b}_h) \tag{16}$$
$$\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] \tag{17}$$
$$\mathbf{y}_t = \text{softmax}(\mathbf{W}_y \mathbf{h}_t + \mathbf{b}_y) \tag{18}$$

Note that bidirectional RNNs are formed by two independent RNNs, one processing the sequence in the forward direction and the other in the backward direction. Thus, weight matrices $\overrightarrow{\mathbf{W}}_x, \overrightarrow{\mathbf{W}}_h, \overleftarrow{\mathbf{W}}_x, \overleftarrow{\mathbf{W}}_h$ are different for the forward and backward RNNs.

bidirectional RNNs are very useful for tasks like named entity recognition, where the context of a word is important for determining its label. However, we can't use bidirectional RNNs for tasks like language modeling, where we need to predict the next word in a sequence.

## 7 Deep RNNs

Deep RNNs stack multiple RNN layers on top of each other to learn complex patterns in the data. Each layer processes the output of the previous layer, allowing the network to capture hierarchical features in the sequence. The hidden state of the first layer is used as input to the second layer, and so on:

$$\mathbf{h}_t^{(1)} = \tanh(\mathbf{W}_x^{(1)}\mathbf{x}_t + \mathbf{W}_h^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{b}_h^{(1)}) \tag{19}$$

$$\mathbf{h}_t^{(2)} = \tanh(\mathbf{W}_x^{(2)}\mathbf{h}_t^{(1)} + \mathbf{W}_h^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{b}_h^{(2)}) \tag{20}$$

$$\mathbf{y}_t = \text{softmax}(\mathbf{W}_y\mathbf{h}_t^{(2)} + \mathbf{b}_y) \tag{21}$$

## 8 Conclusion

In this lecture, we discussed the application of RNNs in language modeling and the challenges with basic RNNs. We introduced advanced RNN architectures like LSTM and GRU, which address the vanishing and exploding gradient problems. In the next lecture, we will discuss the application of RNNs in machine translation and sequence-to-sequence learning. We will also discuss how attention mechanisms can improve the performance of RNNs in these tasks.

## Exercises

1. In a language modeling task, given the vocabulary V = ["the", "cat", "sat", "on", "mat"] and the sequence "the cat sat", calculate the cross-entropy loss for predicting the next word if the model outputs the following probabilities: ["the": 0.1, "cat": 0.2, "sat": 0.1, "on": 0.5, "mat": 0.1] and the true next word is "on". Show your calculation.

   (a) 0.693

   (b) 0.916

   (c) 1.204

   (d) 0.511

2. Consider a standard LSTM cell. If the forget gate outputs $f_t = [0.9, 0.1]$, input gate $i_t = [0.2, 0.8]$, and candidate cell state $\tilde{c}_t = [1.0, -1.0]$, what will be the new cell state $c_t$ assuming the previous cell state $c_{t-1} = [2.0, 3.0]$? Round to 2 decimal places.

   (a) $[2.00, 0.50]$

   (b) $[1.80, -0.50]$

   (c) $[2.00, -0.80]$

   (d) $[1.80, 0.30]$

3. What is the key difference between GRU and LSTM architectures?

   (a) GRU uses output gates while LSTM uses input gates

   (b) LSTM maintains a separate cell state and hidden state, while GRU combines them

   (c) GRU processes sequences bidirectionally while LSTM processes them unidirectionally

   (d) LSTM uses tanh activation while GRU uses sigmoid activation

4. For a bidirectional RNN with hidden size of 64 for both forward and backward RNNs, what will be the dimension of the concatenated hidden state $h_t$?

   (a) 64

   (b) 128

(c) 32

(d) 256

5. Given a sequence "I love deep learning" and vocabulary size $|V| = 10000$, what would be the shape of the one-hot encoded input matrix X for this sequence?

    (a) (4, 10000)

    (b) (10000, 4)

    (c) (1, 4, 10000)

    (d) (4, 1, 10000)