The National University of Computer and Emerging Sciences

Recurrent Neural Network

Deep Learning

1

Dr. Qurat Ul Ain

Department of AI & DS

Recurrent Neural Network (RNN)

Why RNN?

Why do we need Recurrent Neural Network?

1. What Problems are Normal CNNs good at?
2. What is Sequence Learning?
3. Ways to Deal with Sequence Labeling.

What Problems are CNNs normally good at?

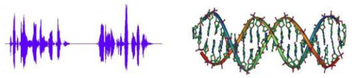
1. Image classification as a naive example
   1. Input: one image.
   2. Output: the probability distribution of classes.
   3. You need to provide one guess (output), and to do that you only need to look at one image (input).

$P(\text{Cat} \mid \text{image}) = 0.1$

$P(\text{Panda} \mid \text{image}) = 0.9$

What is Sequential Learning

1. Sequence learning is the study of machine learning algorithms designed for sequential data [1].



2. Language model is one of the most interesting topics that use sequence labeling.
    1. Language Translation
        1. Understand the meaning of each word, and the relationship between words
        2. Input: one sentence in German

           input = "Ich will stark Steuern senken"
        3. Output: one sentence in English

           output = "I want to cut taxes bigly" (big league?)

What is Sequential Learning

1. To make it easier to understand why we need RNN, let's think about a simple speaking case
    1. We are given a hidden state that encodes all the information in the sentence we want to speak.
    2. We want to generate a list of words (sentence) in an one-by-one fashion.
        1. At each time step, we can only choose a single word.
        2. The hidden state is affected by the words chosen (so we could remember what we just say and complete the sentence).

What is Sequential Learning

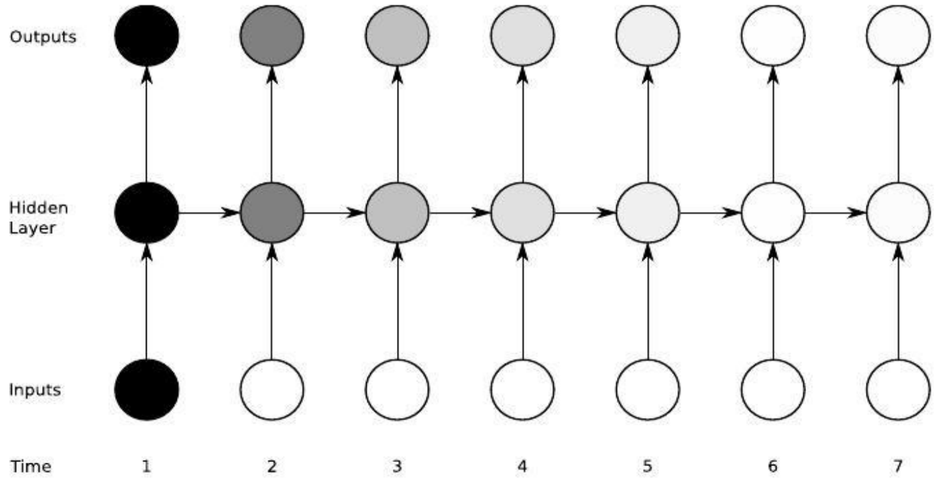
Plain CNNs are not born good at length-varying input and output.

1. Difficult to define input and output
    1. Remember that
        1. Input image is a 3D tensor (width, length, color channels)
        2. Output is a distribution on fixed number of classes.
    2. Sequence could be:
        1. "I know that you know that I know that you know that I know that you know that I know that you know that I know that you know that I know that you know that I don't know"
        2. "I don't know"

Ways to Deal with Sequence Labeling

The RNN model!

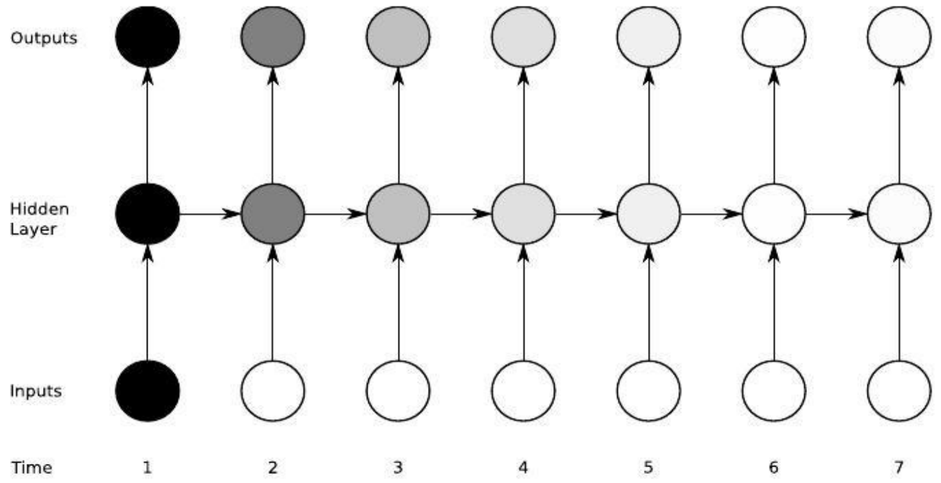
1. Update the hidden state.
2. In the simple speaking case, we send the chosen word back to the network as input.



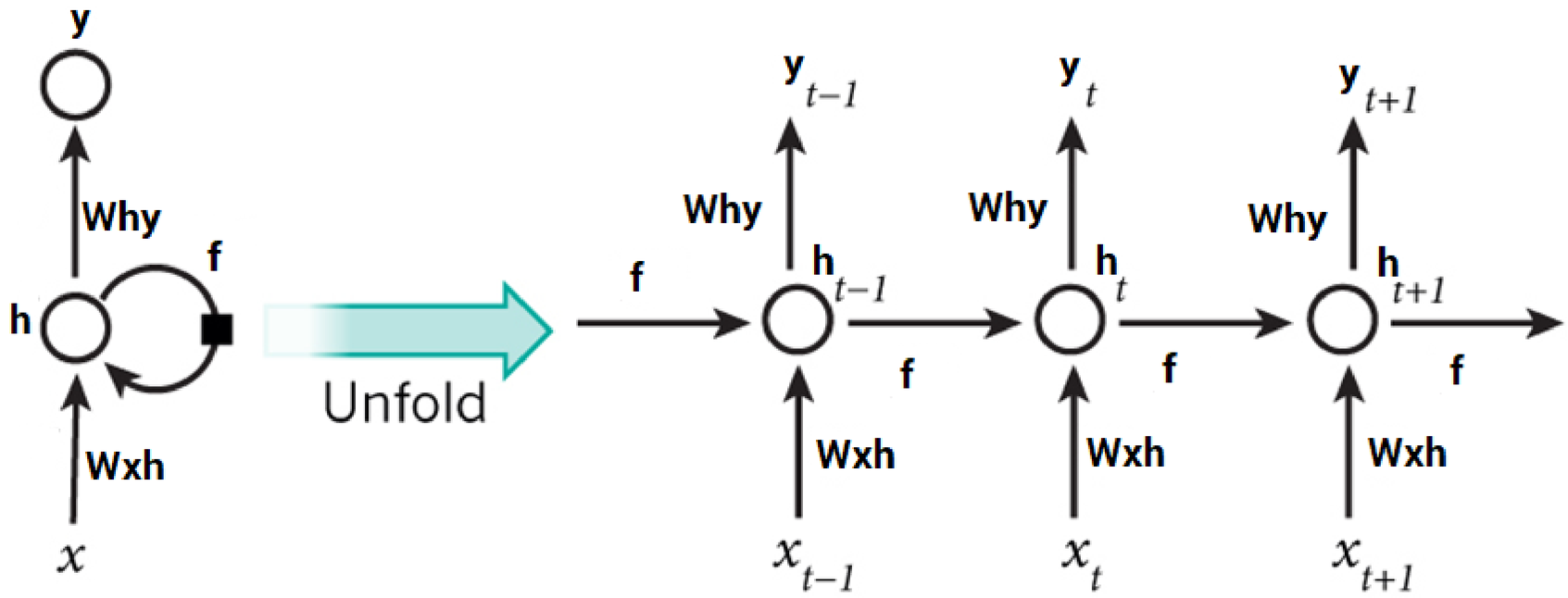
Ways to Deal with Sequence Labeling

RNNs are very powerful, because they:

1. Distributed hidden state that allows them to store a lot of information about the past efficiently.
2. Non-linear dynamics that allows them to update their hidden state in complicated ways.
3. No need to infer hidden state, pure deterministic.
4. Weight sharing



RNN Architecture



Problem Setup

The task is to predict the next word in a sequence.
Given the sentence "I love deep learning," the RNN will predict:

- Given "I," predict "love."
- Given "I love," predict "deep."
- Given "I love deep," predict "learning."

Input representation

- Each word in the vocabulary is represented as a one-hot vector.
- Assume our vocabulary consists of the words:

{I, love, deep, learning, _}

(where _ represents an unknown word or padding).

- The one-hot vectors for each word are:
  - "I": [1, 0, 0, 0, 0]
  - "love": [0, 1, 0, 0, 0]
  - "deep": [0, 0, 1, 0, 0]
  - "learning": [0, 0, 0, 1, 0]
  - "_": [0, 0, 0, 0, 1]

RNN Architecture

- The RNN processes the sequence of words one word at a time. At each time step t, the RNN takes in the current input word vector x_t and the hidden state from the previous time step h_{t-1}, and it produces a new hidden state h_t and an output y_t.
- The equations governing the RNN are:

  - $h_t = \sigma(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$
  - $y_t = \text{softmax}(W_y \cdot h_t + b_y)$

- $h_t = \sigma(W_h \cdot x_t + U_h \cdot h_{t-1} + b_h)$
- $y_t = \text{softmax}(W_y \cdot h_t + b_y)$

Where:

- W_h is the weight matrix connecting the input to the hidden state.
- U_h is the weight matrix connecting the previous hidden state to the current hidden state.
- b_h is the bias for the hidden state.
- W_y is the weight matrix connecting the hidden state to the output.
- b_y is the bias for the output.
- σ is the activation function, typically tanh or ReLU

Weight Matrices Initialization

Let’s assume:

- Input size (number of unique words in the vocabulary) =5
- Hidden state size = 3
- Output size = 5 (same as the input size since we are predicting the next word in the same vocabulary)

The weight matrices are initialized as:

$$W_h = \begin{pmatrix} 0.2 & -0.3 & 0.1 & 0.4 & 0.1 \\ 0.5 & 0.1 & -0.2 & 0.2 & -0.4 \\ -0.1 & 0.3 & 0.2 & -0.1 & 0.2 \end{pmatrix}$$

$$U_h = \begin{pmatrix} 0.1 & 0.3 & -0.2 \\ 0.4 & -0.1 & 0.5 \\ -0.3 & 0.2 & 0.1 \end{pmatrix}$$

$$b_h = \begin{pmatrix} 0.1 \\ 0.1 \\ -0.1 \end{pmatrix}$$

$$W_y = \begin{pmatrix} -0.2 & 0.1 & 0.2 \\ 0.3 & -0.4 & 0.5 \\ 0.1 & 0.2 & -0.3 \\ -0.5 & 0.4 & 0.1 \\ 0.2 & -0.1 & 0.3 \end{pmatrix}$$

$$b_y = \begin{pmatrix} 0.0 \\ 0.1 \\ 0.0 \\ 0.1 \\ 0.0 \end{pmatrix}$$

Forward Pass Example

Let’s go through the first time step to illustrate the process.

Step 1: Input (t = 1)

The input at time t=1 is "I", represented as x_1=[1,0,0,0,0].

Step 2: Initial Hidden State

Assume the initial hidden state h_0=[0,0,0].

Step 3: Calculate Hidden State h_1

Using the RNN equation:

$$h_1 = \tanh(W_h \cdot x_1 + U_h \cdot h_0 + b_h)$$

Calculating $W_h \cdot x_1$:

$$W_h \cdot x_1 = \begin{pmatrix} 0.2 & -0.3 & 0.1 & 0.4 & 0.1 \\ 0.5 & 0.1 & -0.2 & 0.2 & -0.4 \\ -0.1 & 0.3 & 0.2 & -0.1 & 0.2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.5 \\ -0.1 \end{pmatrix}$$

Calculating $U_h \cdot h_0$:

$$U_h \cdot h_0 = \begin{pmatrix} 0.1 & 0.3 & -0.2 \\ 0.4 & -0.1 & 0.5 \\ -0.3 & 0.2 & 0.1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Adding the bias b_h and applying the activation function $\tanh$:

$$h_1 = \tanh\left(\begin{pmatrix} 0.2 \\ 0.5 \\ -0.1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.1 \\ -0.1 \end{pmatrix}\right) = \tanh\left(\begin{pmatrix} 0.3 \\ 0.6 \\ -0.2 \end{pmatrix}\right) = \begin{pmatrix} 0.2913 \\ 0.5370 \\ -0.1974 \end{pmatrix}$$

Step 4: Calculate Output y_1

Using the output equation:

$$y_1 = \text{softmax}(W_y \cdot h_1 + b_y)$$

Calculating $W_y \cdot h_1$:

$$W_y \cdot h_1 = \begin{pmatrix} -0.2 & 0.1 & 0.2 \\ 0.3 & -0.4 & 0.5 \\ 0.1 & 0.2 & -0.3 \\ -0.5 & 0.4 & 0.1 \\ 0.2 & -0.1 & 0.3 \end{pmatrix} \cdot \begin{pmatrix} 0.2913 \\ 0.5370 \\ -0.1974 \end{pmatrix}$$

Performing the multiplication:

$$W_y \cdot h_1 = \begin{pmatrix} (-0.2 \cdot 0.2913) + (0.1 \cdot 0.5370) + (0.2 \cdot -0.1974) \\ (0.3 \cdot 0.2913) + (-0.4 \cdot 0.5370) + (0.5 \cdot -0.1974) \\ (0.1 \cdot 0.2913) + (0.2 \cdot 0.5370) + (-0.3 \cdot -0.1974) \\ (-0.5 \cdot 0.2913) + (0.4 \cdot 0.5370) + (0.1 \cdot -0.1974) \\ (0.2 \cdot 0.2913) + (-0.1 \cdot 0.5370) + (0.3 \cdot -0.1974) \end{pmatrix} = \begin{pmatrix} -0.05826 \\ -0.05474 \\ 0.22871 \\ 0.03783 \\ 0.01737 \end{pmatrix}$$

Adding the bias vector b_y:

$$W_y \cdot h_1 + b_y = \begin{pmatrix} -0.05826 \\ -0.05474 \\ 0.22871 \\ 0.03783 \\ 0.01737 \end{pmatrix} + \begin{pmatrix} 0.0 \\ 0.1 \\ 0.0 \\ 0.1 \\ 0.0 \end{pmatrix} = \begin{pmatrix} -0.05826 \\ 0.04526 \\ 0.22871 \\ 0.13783 \\ 0.01737 \end{pmatrix}$$

Now, apply the softmax function to convert the scores into probabilities:

$$y_1 = \text{softmax} \left(\begin{pmatrix} -0.05826 \\ 0.04526 \\ 0.22871 \\ 0.13783 \\ 0.01737 \end{pmatrix} \right)$$

The softmax function is computed as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Calculating the exponentials of each element:

$$\exp\left(\begin{pmatrix}-0.05826\\0.04526\\0.22871\\0.13783\\0.01737\end{pmatrix}\right) = \begin{pmatrix}e^{-0.05826}\\e^{0.04526}\\e^{0.22871}\\e^{0.13783}\\e^{0.01737}\end{pmatrix} \approx \begin{pmatrix}0.9434\\1.0463\\1.2570\\1.1477\\1.0176\end{pmatrix}$$

Now, normalize by dividing each value by the sum of all exponentials:

$$\sum_j e^{z_j} = 0.9434 + 1.0463 + 1.2570 + 1.1477 + 1.0176 = 5.4119$$

So, the softmax probabilities are:

$$y_1 = \begin{pmatrix}\frac{0.9434}{5.4119}\\\frac{1.0463}{5.4119}\\\frac{1.2570}{5.4119}\\\frac{1.1477}{5.4119}\\\frac{1.0176}{5.4119}\end{pmatrix} = \begin{pmatrix}0.1743\\0.1933\\0.2323\\0.2120\\0.1880\end{pmatrix}$$

These probabilities represent the likelihood of the next word in the sequence being each of the words in the vocabulary. In this case, the RNN predicts the highest probability for the word corresponding to 0.23230.23230.2323, which in this example would correspond to "love."

Summary of steps

1. **Input Processing:** Convert the input word to a one-hot vector
2. **Hidden State Calculation:** Update the hidden state using the input vector, the previous hidden state, and the hidden layer weights
3. **Output Calculation:** Compute the output vector using the hidden state and the output layer weights, then apply the softmax function to get the probability distribution over the vocabulary.
4. **Prediction:** Choose the word with the highest probability as the predicted next word.

Formation of Weight Matrices

- W_h: Connects the input word vector to the hidden state. It determines how the input word influences the hidden state.
- U_h: Connects the previous hidden state to the current hidden state. It determines how the hidden state from the previous time step influences the current hidden state.
- b_h: Bias added to the hidden state.
- W_y: Connects the hidden state to the output vector, predicting the next word.
- b_y: Bias added to the output.

Backpropagation through Time (BPTT)

- During training, the RNN adjusts these weight matrices through BPTT to minimize the difference between the predicted word probabilities and the actual words in the sequence.
- This process involves computing gradients for each weight matrix and updating them to improve the model's predictions over time.
- This completes the detailed explanation of RNNs using a sequence of words as an example.

•**What Problems are Normal CNNs Good at?**
•**Image recognition and classification**: CNNs excel at tasks involving spatial relationships, such as identifying objects in images.
•**Object detection**: They can locate and classify multiple objects within an image.
•**Image segmentation**: CNNs divide an image into regions or objects.
•**Video analysis**: They handle frame-by-frame analysis but don't inherently deal with temporal dependencies well.
•**What is Sequence Learning?**
•Sequence learning involves processing data where the order of elements matters, such as in time-series data, speech, or text.
•Examples include language modeling, machine translation, speech recognition, and stock price predictions.
•**Ways to Deal with Sequence Labeling**
•**RNNs**: Capture sequential data relationships and are used in tasks like part-of-speech tagging.
•**LSTMs/GRUs**: Handle long-term dependencies better than basic RNNs, preventing vanishing gradient issues.
•**CRFs (Conditional Random Fields)**: Often combined with neural networks for structured prediction.
•**Bi-directional RNNs/LSTMs**: Process data in both forward and backward directions, improving sequence labeling tasks like named entity recognition.
•**Attention Mechanisms**: Used to focus on important parts of a sequence, enhancing performance in tasks like translation or summarization.
•Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are typically used for this purpose, as they can capture temporal dependencies.