Recurrent Neural Network (CNN) Architecture

M. R. Hasan

Deep Learning
Readings

- Geron: 15
- Zhang: 8
- Chollet: 6
What We Will Cover

- RNN Architecture
- RNN: Signal Propagation
- Are RNNs Deep Neural Networks?
- RNN Vs FNN: Model Complexity & Computational Complexity
- RNN Architectures: Various Types of Sequence Data
RNN Architecture
RNN Architecture

- A Recurrent Neural Network (RNN) architecture looks very much like a Feedforward Neural Network (FNN) architecture.
- Except RNN has connections pointing backward.
RNN

• Let’s **generalize** the architecture of a RNN.
• In the following RNN illustration, our **input is a scalar**.
• Thus we use a **single neuron at the hidden layer**.
• It produces a scalar hidden output.
RNN

• For processing vector inputs, we create a layer of recurrent neurons.
• At each timestep $t$, every neuron receives:
  - The input vector $\tilde{x}_t$ (each token has a vector embedding) and
  - The hidden state vector $\tilde{h}_{t-1}$ from the previous timestep.
RNN: Signal Propagation
RNN: Signal Propagation

• Let’s present how a signal is propagated forward in RNNs for a single sample.
• We need to be careful about the meaning of timestep in the context of RNNs.
• In a typical optimization algorithm, such as in gradient descent, we often refer iteration \( t \) as time \( t \).
RNN: Signal Propagation

• However, **time** in RNNs refers to **steps within an iteration**.
• Example: if an input sequence contains $T$ tokens (embeddings), then there will be $T$ number of timesteps within a single iteration.

One iteration: there are 5 timesteps for an input sequence
RNN: Signal Propagation

- Usually, each recurrent neuron has **two sets of weights**:
  - Input weights $\overrightarrow{w}_{xh}$: for the input $\hat{x}_t$
  - Recurrent weights $\overrightarrow{w}_{hh}$: for the hidden state outputs of the previous timestep $\hat{h}_{t-1}$

In special cases the output connection has weights $\overrightarrow{w}_{hy}$ (output layer neuron not shown)
RNN: Signal Propagation

- Consider the **whole recurrent layer** instead of just one recurrent neuron.
- Place all the weight vectors for multiple neurons in two weight matrices: $W_{xh}$ and $W_{hh}$
• Assume that:
  - An input vector is \textit{d-dimensional} (i.e., each token embedding has \textit{d} features)
  - There are \textit{k} recurrent neurons.

• Then, the \textbf{size of the weight matrices} are defined as follows.
  - \( \mathbf{W}_{xh} \): size \( d \times k \) - containing the connection weights for the inputs of the current timestep.
  - \( \mathbf{W}_{hh} \): size \( k \times k \) - containing the connection weights for the hidden outputs of the previous timestep.
• Now consider a **single iteration** in which a sequence of timesteps is denoted by \( t = 1, \ldots, T \).

• Then, the hidden-state output for a **single input at timestep** \( t \) is calculated as follows:

\[
\hat{h}_t = \phi(\mathbf{W}_{xh}^T \tilde{x}_t + \mathbf{W}_{hh}^T \hat{h}_{t-1} + \mathbf{b}_h)
\]

• Note that \( \hat{h}_t \) is a 1D vector of size \( k \).

• Also, \( \mathbf{b}_h \) is the bias vector of size \( k \) containing each neuron’s bias term.

\[
\begin{align*}
\mathbf{W}_{xh} & : \text{size } d \times k \\
\mathbf{W}_{hh} & : \text{size } k \times k \\
\tilde{x}_t & : \text{size } d \times 1 \\
\hat{h}_t & : \text{size } k \times 1 \\
\mathbf{b}_h & : \text{size } k \times 1
\end{align*}
\]
RNN: Signal Propagation

- \( \vec{h}_t = \phi(\mathbf{W}_{xh} \vec{x}_t + \mathbf{W}_{hh} \vec{h}_{t-1} + \vec{b}_h) \)
- The \( \phi(\cdot) \) is a nonlinear activation function (e.g., tanh).
- Note that the activation function is applied element-wise.
- Since the hidden-state output \( \vec{h}_t \) is a vector, the activation needs to be applied to each component of the vector.

- \( \mathbf{W}_{xh} \): size \( d \times k \)
- \( \mathbf{W}_{hh} \): size \( k \times k \)
- \( \vec{x}_t \): size \( d \times 1 \)
- \( \vec{h}_t \): size \( k \times 1 \)
- \( \vec{b}_h \): size \( k \times 1 \)
RNN: Signal Propagation

• We can compute a recurrent layer’s output in one shot for a **whole mini-batch**.
• Placing all inputs at timestep $t$ in an input matrix $X_t$.
• Say that there are $m$ samples in the mini-batch, each sample containing $d$ features.
RNN: Signal Propagation

- We define the size of the input and hidden-state output matrices as follows.
  - $X_t$: $m \times d$ - containing the inputs for all instances.
  - $H_t$: $m \times k$ - containing the layer’s outputs at timestep $t$ for each instance in the mini-batch.

$W_{xh}$: size $d \times k$
$W_{hh}$: size $k \times k$
$X_t$: size $m \times d$
$H_t$: size $m \times k$
$\vec{b}_h$: size $k \times 1$
RNN: Signal Propagation

• Then, the hidden-state output for a mini-batch input at timestep $t$ is calculated as follows:

$$H_t = \phi \otimes (X_t W_{xh} + H_{t-1} W_{hh} + \vec{b}_h)$$

$W_{xh}$: size $d \times k$
$W_{hh}$: size $k \times k$
$X_t$: size $m \times d$
$H_t$: size $m \times k$
$\vec{b}_h$: size $k \times 1$
RNN: Signal Propagation

- \( H_t = \phi \otimes (X_t W_{xh} + H_{t-1} W_{hh} + \vec{b}_h) \)

- The weight matrices \( W_{xh} \) and \( W_{hh} \) are often concatenated vertically into a single weight matrix \( W \):
  \[
  H_t = \phi \otimes ([X_t \ H_{t-1}]W + \vec{b}_h)
  \]

- The notation \([X_t \ H_{t-1}]\) represents the horizontal concatenation of the matrices \(X_t\) and \(H_{t-1}\).

\[
W = \begin{bmatrix} W_{xh} \\ W_{hh} \end{bmatrix}
\]

The shape of \(W\) is \((d+k)\times k\)
RNN: Signal Propagation

• As mentioned earlier, in a simple RNN architecture, the output vector is equal to the hidden-state vector.
  \[ Y_t = H_t \]

• Here, the size of the output vector is:

  • \( Y_t: m \times k \) - containing the layer’s outputs at timestep \( t \) for each instance in the mini-batch.

\[
\begin{align*}
W_{xh}: & \text{ size } d \times k \\
W_{hh}: & \text{ size } k \times k \\
X_t: & \text{ size } m \times d \\
H_t: & \text{ size } m \times k \\
\vec{b}_h: & \text{ size } k \times 1 \\
Y_t: & \text{ size } m \times k
\end{align*}
\]
RNN: Signal Propagation

• In general, for complex RNN architectures, the output vector is computed as follows:

$$Y_t = \phi \otimes (H_t W_{hy} + \vec{b}_y)$$
Are RNNs Deep Neural Networks?
RNN: Deep Neural Network

- So far, we have seen that the RNN models contain only one hidden state layer.
- It is possible to stack multiple hidden state layers to create deep RNNs.
- We will discuss this type of deep RNN architecture later.
RNN: Deep Neural Network

- Is a single-hidden layer RNN a Deep Neural Network (DNN)?
- A RNN with only one hidden state layer can be a DNN.
- It is deep not in space, but in time.
- For a long sequence, we have to extend the recurrent computation deep into time.
RNN: Deep Neural Network

- In feedforward neural networks (FC or CNN), the depth of a network is defined in space, i.e., with respect to the number of hidden layers.

- More fundamentally, the network depth represents the length of the path that an input signal has to pass through to reach the final state in the computation.

- In FNNs, this length of the computational path is defined by the depth of a network.
RNN: Deep Neural Network

- But in RNNs, a signal propagates **through time** to reach the final state (last embedding of a sequence) via recurrence.
- Thus, the **amount of the recurrence** is a measure of the depth of a RNN.
- In other words, in RNNs the network depth is **defined in time**.

To process long input sequences, we need to build deep RNNs.
RNN Vs FNN: Model Complexity & Computational Complexity
RNN Vs FNN: Model Complexity & Computational Complexity

- RNNs always use the **same model parameters** (two weight matrices) for different timesteps.
- The number of RNN model parameters **does not grow** as the number of timesteps increases.
- Thus, RNNs are **memory efficient**.
RNN Vs FNN: Model Complexity & Computational Complexity

• However, the computational complexity of a RNN could explode depending on the **length of the input sequence**.

• The computational complexity will also increase with **model complexity** (e.g., multiple hidden state layers, both forward and backward processing, etc.).
RNN Architectures: Various Types of Sequence Data
RNN Architectures: Various Types of Sequence Data

• So far, we have assumed that the **length** of the input sequence and output sequence are the **same**.

• In practice, this may not hold true.
RNN Architectures: Various Types of Sequence Data

- We encounter scenarios in which the length of input sequence **varies** from that of the output sequence.
- In such case, we will have to **adapt the simple RNN** architecture.
RNN Architectures: Various Types of Sequence Data

• We will discuss **three broad cases** of input-output sequence-length variation, and give suitable RNN architectures.
  - Many-to-Many: Sequence-to-Sequence
  - Many-to-One: Sequence-to-Vector
  - One-to-Many: Vector-to-Sequence
• Many-to-Many: Sequence-to-Sequence
• Many-to-One: Sequence-to-Vector
• One-to-Many: Vector-to-Sequence

Many-to-Many: Sequence-to-Sequence

Synced: Input Sequence = Output Sequence

Many-to-One: Sequence-to-Vector

One-to-Many: Vector-to-Sequence
Many-to-Many: Sequence-to-Sequence

• Consider the problem of **parts of speech (POS) tagging**.
• In this problem, we have to **mark up a word** in a text as corresponding to a particular part of speech, based on both its definition and its context.

![Parts of Speech Tagging](image)

*Many-to-Many: Sequence-to-Sequence*

*Synced*: Input Sequence = Output Sequence
Many-to-Many: Sequence-to-Sequence

- For example, given the sentence “Life is beautiful”, we will have to infer the POS tag for each of the token in this sentence.
- In this problem the length of the input sequence is equal to the length of the output sequence.

Also both sequences are synced.
Many-to-Many: Sequence-to-Sequence

• Thus, it’s a **many-to-many mapping**.
• We can use the **simple RNN architecture** that we presented previously for solving this type of problem.

**Parts of Speech Tagging**

**Synced**: Input Sequence = Output Sequence
Many-to-Many: Sequence-to-Sequence

• Other examples that use this type of sequence-to-sequence RNN models include:
  - Stock price prediction
  - Named entity recognition
  - Pose estimation
  - Video classification where we wish to label each frame of the video

Parts of Speech Tagging

Noun  Verb  Adjective

Life  is  Beautiful

Many-to-Many: Sequence-to-Sequence
Synced: Input Sequence = Output Sequence
Many-to-Many: Sequence-to-Sequence

• Now consider another type of many-to-many mapping.
• The input and output sequence length are not the same.
• Moreover, they are not synced.
Many-to-Many: Sequence-to-Sequence

- Example: machine translation.
- The input sentence in English has 3 tokens.
- But the output (inferred) sequence (sentence in French) has 4 tokens.

The two sequences are **NOT synced**.
Many-to-Many: Sequence-to-Sequence

• This type of sequence-to-sequence problem is solved by designing **two networks**.
  - Sequence-to-vector (encoder)
  - vector-to-sequence (decoder)
• The encoder converts the English sentence into a **single vector** representation.

Then, the decoder takes this vector as input and **decodes** it into a sentence in French.
Many-to-One: Sequence-to-Vector

- In some problems, input is a sequence, but **output is a vector**.
- For example, consider the **movie sentiment analysis** problem.
- The output could be binary (positive sentiment or negative sentiment), or a moving ranking between 1 to 5.
Many-to-One: Sequence-to-Vector

- In either case, we feed the network the **entire sequence** of the review.
- At each timestep, the network produces an output.
- We don’t need outputs from all timesteps **except the last one** where the network makes an inference about the review.

This scenario can be modeled by a **sequence-to-vector** network.
One-to-Many: Vector-to-Sequence

- In some problems input is a vector, but output is a sequence.
- Consider an example of image captioning.
- The input could be an image (or the output of a CNN), which is a single vector.
- The output could be a caption for that image (i.e., a sequence of words).
Many-to-One: Sequence-to-Vector

- Another example could be **music generation**.
- As input we may provide the genre or even nothing (i.e., a null vector).
- But the output is a **sequence of notes** representing a piece of music.

This type of problems can be modeled by using a **vector-to-sequence** RNN architecture.
Next: RNN Training