Recurrent NN

MICHAEL WOLLOWSKI

SUMMARY OF CHAPTER 9: RNNS AND LSTMS

FROM: SPEECH AND LANGUAGE PROCESSING.

BY JURAFSKY AND MARTIN. HTTPS://WEB.STANFORD.EDU/~JURAFSKY/SLP3/

NLP with FFnet

Performing next word prediction with a feed-forward network and word embeddings.

At each time step *t*, the network converts N context words, each to a *d*-dimensional embedding.

It concatenates the N embeddings together to obtain the *Nd* x 1 unit input vector **x** for the network.

The output of the network is a probability distribution over the vocabulary.

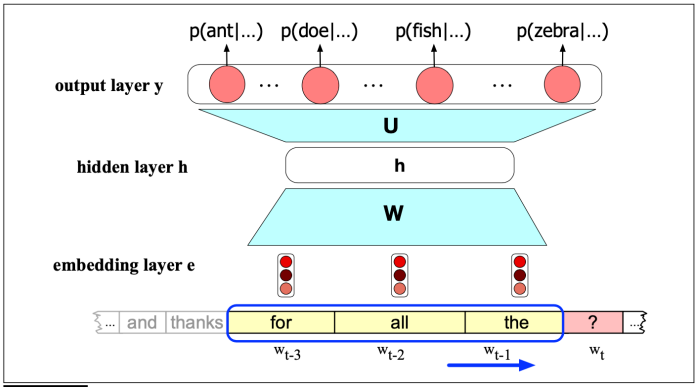

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 12, 2022.

RNN Introduction

In an RNN, a unit has a feedback loop to itself, hence “recurrent”.

This enables a unit to maintain a state.

Below is an RNN unit unfolded in time.

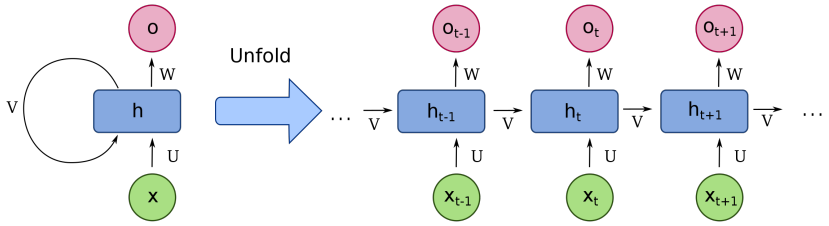

Image source: fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60109157

Applications of RNNs

RNNs can use their internal state (memory) to process variable length sequences of inputs.

RNNs are used in processing language and speed.

In language and speech, we have long sequences of inputs.

RNNs in Detail

Let's focus on the right part of the following image first.

It contains the vanilla feed-forward portion of the RNN:

- input vector x_t
- weight matrix W from input to hidden layer units
- vector h_t of the hidden layer unit activations
- weight matrix V leading from the hidden layer to output layer.
- output vector y_t

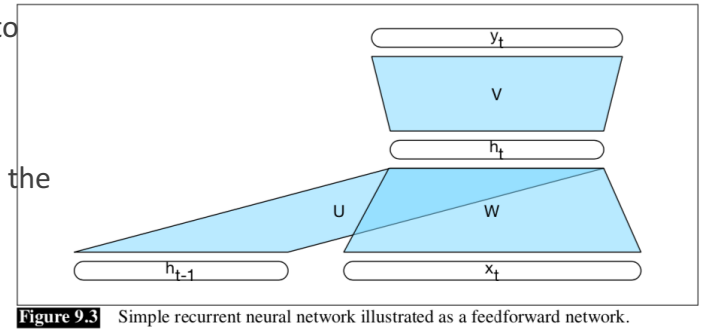

Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

RNNs in Detail

Now let's focus on the left portion of the image.

It captures the recurrent nature of RNNs

h_{t-1} is the output of the hidden layer units at the prior time step.

U is the weight vector from the hidden layer units to themselves.

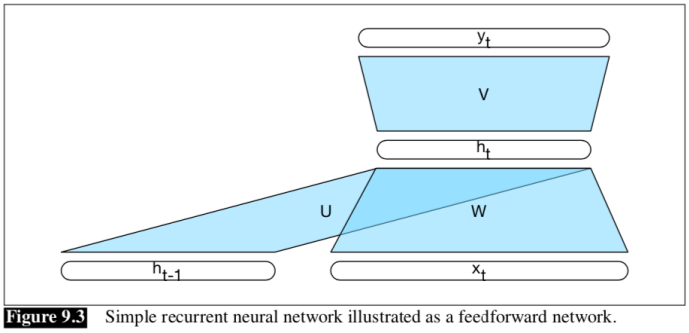

Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

Calculating Activations

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

Figure 9.3 Forward inference in a simple recurrent network. The matrices **U**, **V** and **W** are shared across time, while new values for **h** and **y** are calculated with each time step.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Chapter 9, Draft of February 3, 2024.

Next word Prediction

Texting applications and emailers attempt to do next word prediction.

Not very well, because they have limited context.

Next word prediction improves with the length of the context (and a corresponding size of the model.

Consider *"Thanks for all the"*

What word might follow?

Next word Prediction

How about "*fish*"?

We would compute: *P*(*fish*|*Thanks for all the*)

Language models give us the ability to assign such a conditional probability to every possible next word

In other words, they give us a distribution over the entire vocabulary.

RNN for Next Word Prediction

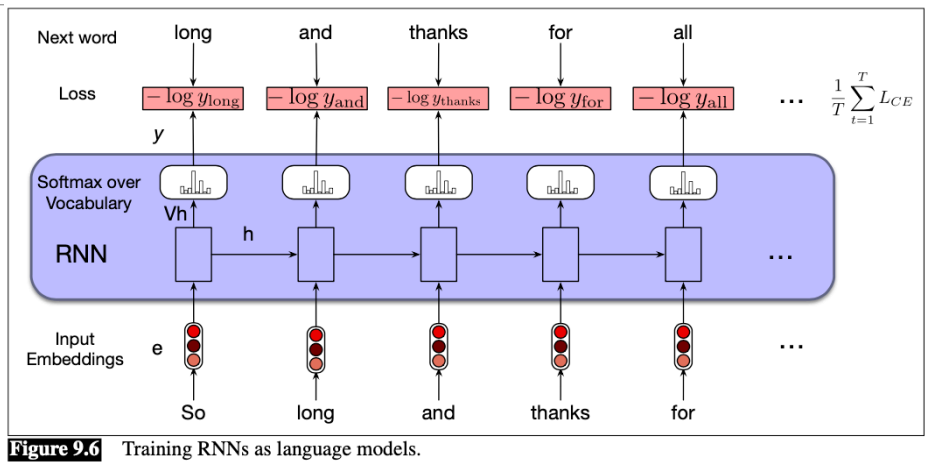

Figure 9.6 Training RNNs as language models.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

RNNs in Detail

By now, you should be aware that:

- neural networks have weights
- we adjust weights as part of wrestling a NN into submission.

Let's see what is new when it comes to RNNs.

At right is a single recurrent unit.

The dashed, blue link means an additional weight.

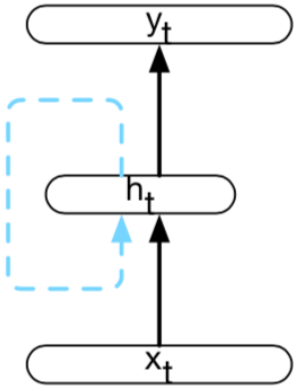

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

RNNs in Detail

The image on the right shows processing of three input sets, x_1, x_2 and x_3.

The network is unrolled in time to show how the use of the hidden layer data.

Throughout training, the weight matrices U, V and W change.

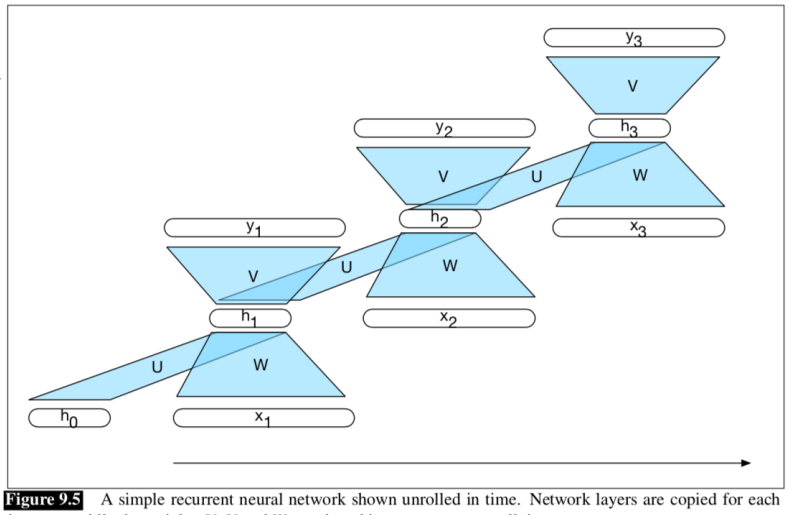

Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

RNNs in Detail

Learning in RNNs is through backpropagation.

Consider the upper right corner of the image on the right.

t_3 is the target vector and y_3 is the actual output.

Adjusting V is as with FF networks, i.e. it depends on the difference between t_3 and y_3.

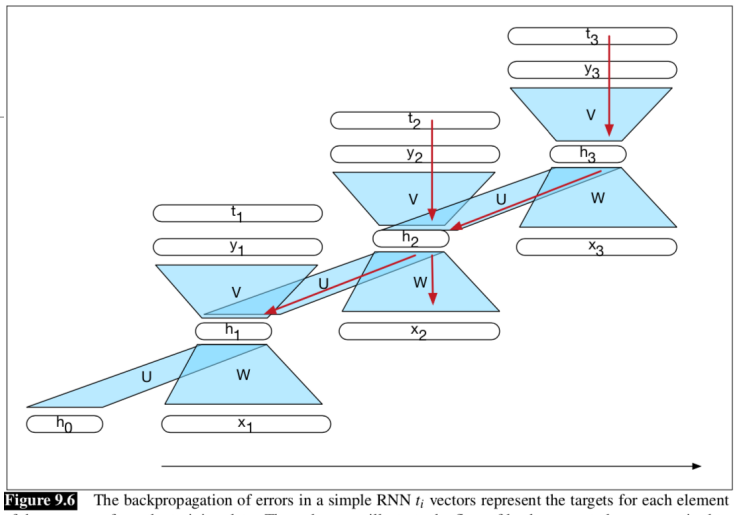

Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

RNNs in Detail

Adjusting U requires data from the errors at the output layer as well as the hidden layer.

This is shown by the red arrows into h_2.

There is a time offset though.

Again, looking at h_2, we take the current error from t_2-y_2 and add to it the error from the next time step, i.e. the one at h_3.

Recall h outputs the same value to itself and the next layer.

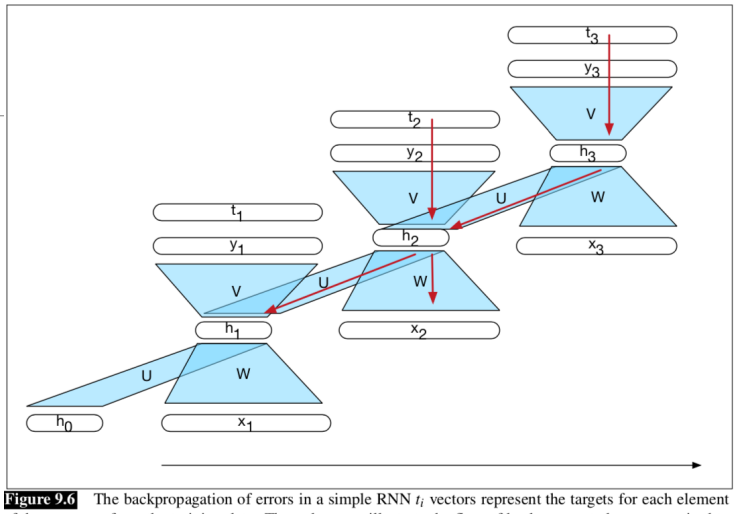

Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

Backpropagation through Time

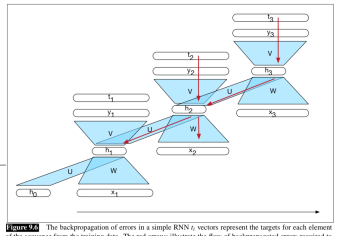

Figure 9.6 The backpropagation of errors in a simple RNN. y_t vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

Tailoring the backpropagation algorithm to this situation leads to a two-pass algorithm for training the weights in RNNs.

In the first pass, we perform forward inference, computing h_t , y_t , accumulating the loss at each step in time, saving the value of the hidden layer at each step for use at the next time step.

In the second phase, we process the sequence in reverse, computing the required gradients as we go, computing and saving the error term for use in the hidden layer for each step backward in time.

This general approach is commonly referred to as backpropagation through time.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

Backpropagation through Time

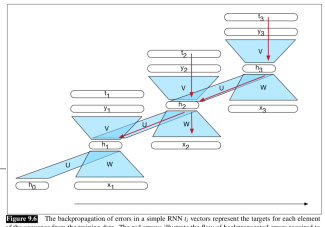

Figure 9.6 The backpropagation of errors in a simple RNN. y_t vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

With modern computational frameworks and adequate computing resources, there is no need for a specialized approach to training RNNs.

Explicitly unrolling a recurrent network into a feedforward computational graph eliminates any explicit recurrences, allowing the network weights to be trained directly.

We provide a template that specifies the basic structure of the network, including all the necessary parameters for the input, output, and hidden layers, the weight matrices, as well as the activation and output functions to be used.

When presented with a specific input sequence, we can generate an unrolled feedforward network specific to that input, and use that graph to perform forward inference or training via ordinary backpropagation.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

Backpropagation through Time

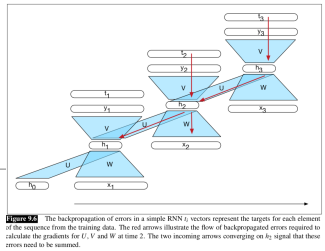

Figure 9.6 The backpropagation of errors in a simple RNN. y_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging on h_2 signal that these errors need to be summed.

For applications that involve much longer input sequences, unrolling an entire input sequence may not be feasible.

In these cases, we can unroll the input into manageable fixed-length segments and treat each segment as a distinct training item.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

Part-of-Speech (POS) Tagging with RNNs

POS tagging a sequence with a simple RNN

Pre-trained word-embeddings serve as inputs.

A softmax layer provides probability distribution over the POS tags at each time step

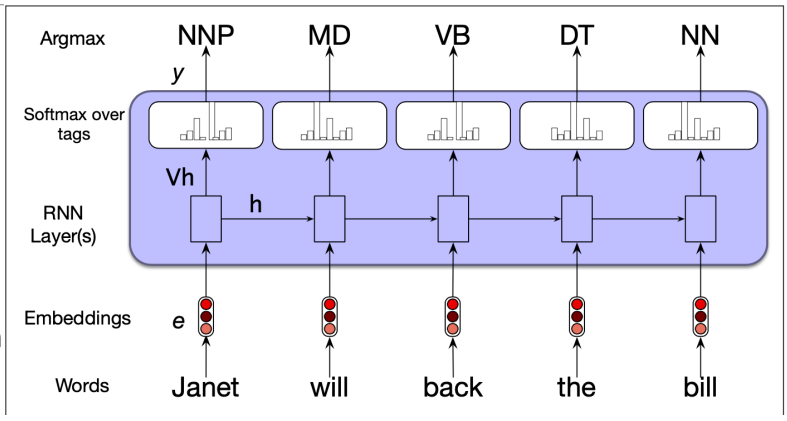

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 12, 2022.

Document Classification with RNNs

This network processes a sequence of text and then classifies it.

It can be used for SPAM detection.

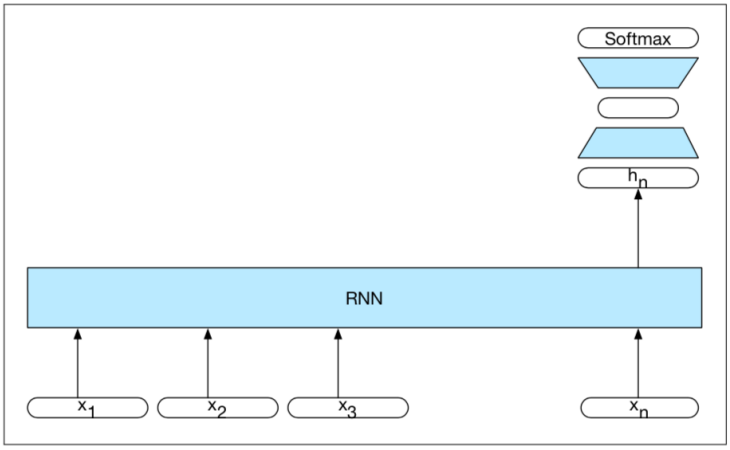

Figure 9.9 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of October 2, 2019.

Autoregressive RNN

Used for text generation.

Called ”generative AI”

ChatGPT is considered an autoregressive model.

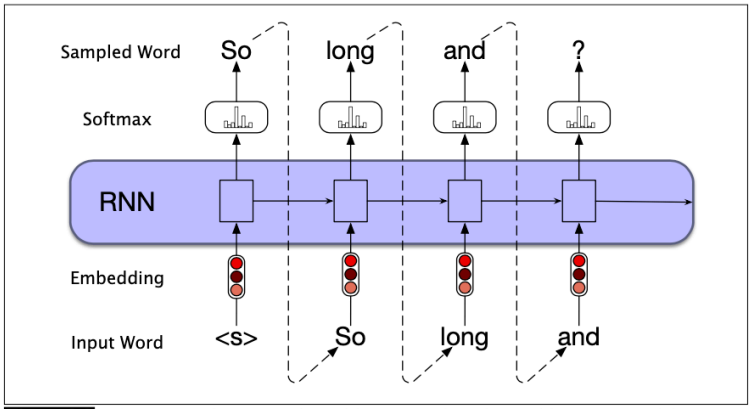

Figure 9.9 Autoregressive generation with an RNN-based neural language model.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Autoregressive RNN

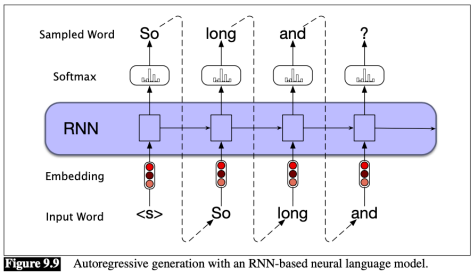

Figure 9.9 Autoregressive generation with an RNN-based neural language model.

1. Typically, the system starts with a seed.
2. In the diagram on the right, the system begins with a beginning of the sentence marker <s>
3. Next, sample a word in the output from the softmax distribution that results from using the beginning of sentence marker, <s>
4. Use the word as the input to the next time step, and sample the next word in the same fashion.
5. Continue generating until the end of sentence marker, </s>, is sampled or a fixed length limit is reached.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Stacked RNNs

So far, the inputs to the RNNs have consisted of sequences of words.

The outputs have been vectors useful for predicting words, tags or sequence labels.

A stacked RNN is a network in which the entire sequence of outputs from one RNN as an input sequence to another one.

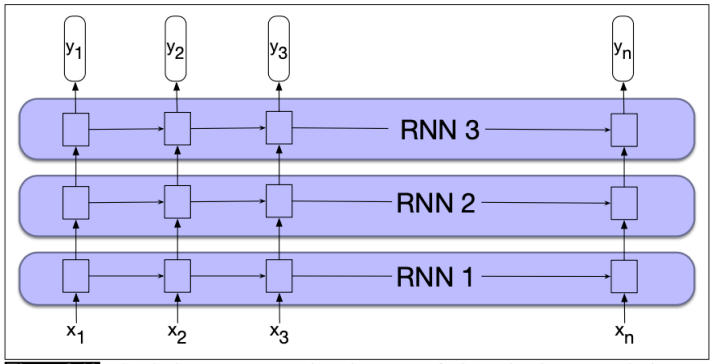

Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Stacked RNNs

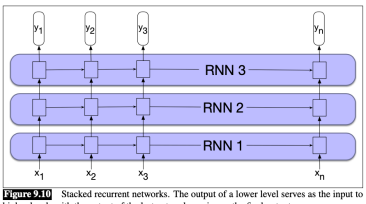

Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

Stacked RNNs generally outperform single-layer networks.

One reason for this success seems to be that the network induces representations (patterns) at differing levels of abstraction across layers.

Similar effect to what CNNs do.

The initial layers of stacked networks can induce representations that serve as useful abstractions for further layers.

The optimal number of stacked RNNs is specific to each application and to each training set.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Bidirectional RNNs

A bidirectional RNN are two RNNs:

- one processes information from beginning to end of a sentence and
- the other from end to beginning.

The information of both RNNs is then concatenated

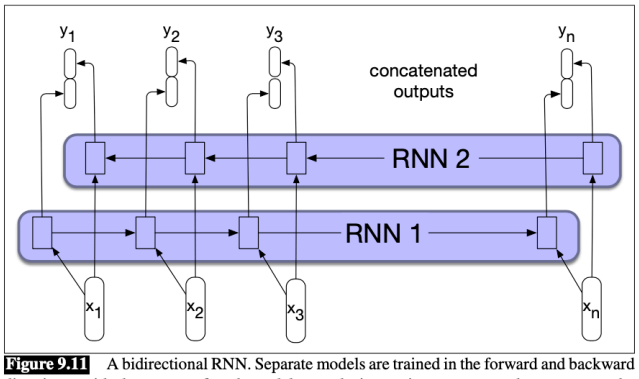

Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Bidirectional RNNs

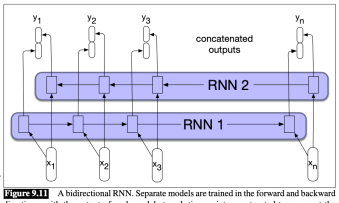

Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

An RNN uses information from the left (prior) context to make its predictions at time *t*.

In many applications we have access to the entire input sequence.

In those cases we would like to use words from the context to the right of *t*.

One way to do this is to run two separate RNNs, one left-to-right, and one right-to-left, and concatenate their representations.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Bidirectional RNNs

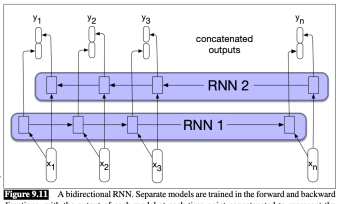

Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

In the left-to-right RNNs we have seen so far, the hidden state at a given time *t* represents everything the network knows about the sequence up to that point.

The state is a function of the inputs $x_1,...,x_t$ and represents the context of the network to the left of the current time.

To take advantage of context to the right of the current input, we can train an RNN on a *reversed* input sequence.

With this approach, the hidden state at time *t* represents information about the sequence to the *right* of the current input:

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Bidirectional RNNs

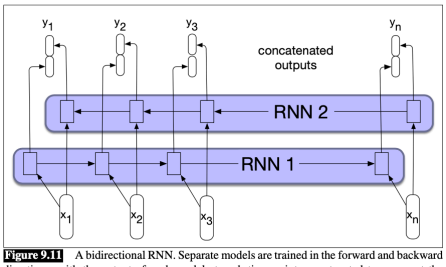

Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

A bidirectional RNN combines two independent RNNs:

- one where the input is processed from the start to the end, and
- the other from the end to the start.

The overall network then concatenates the two representations computed by the networks into a single vector.

It captures both the left and right contexts of an input at each point in time.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

Bidirectional RNN for Sequence Classification

Bidirectional RNNs have proven to be quite effective for sequence classification.

The final state naturally reflects more information about the end of the sentence than its beginning.

Simply combine the final hidden states from the forward and backward passes.

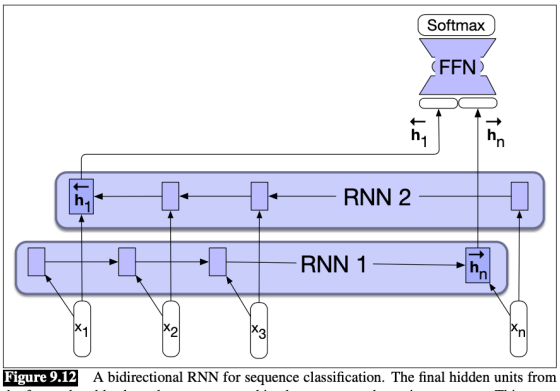

Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.