Recurrent NN

MICHAEL WOLLOWSKI

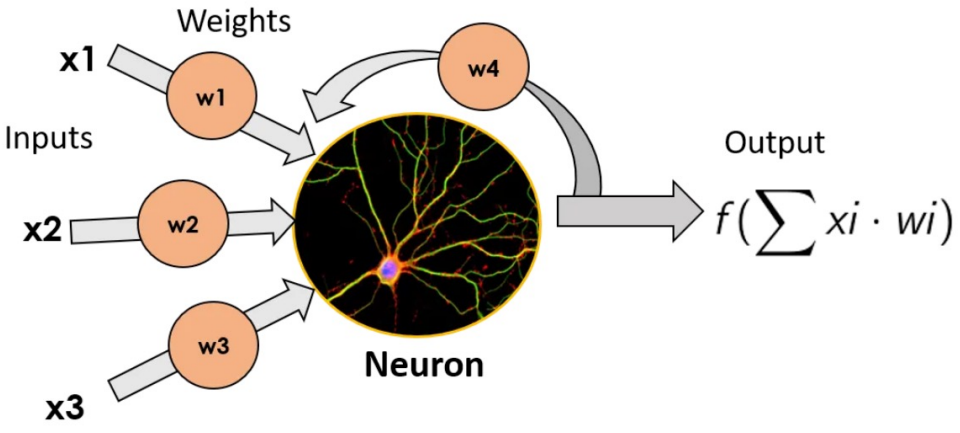
SUMMARY OF: HTTPS://MEDIUM.COM/@HUMBLE_BEE/RNN-RECURRENT-NEURAL-NETWORKS-LSTM-842BA7205BBF

AND CHAPTER 9: RNNS AND LSTMS, FROM: SPEECH AND LANGUAGE PROCESSING.

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

1

A Recurrent Neuron



2

Definition and Use of RNN

Recurrent Neural Network is a feed-forward network that has an internal memory.

They are designed to effectively deal with sequential data, such as text.

For making a decision, an RNN considers the current input and the output that it has learned from the previous input.

The internal state acts as (limited) memory.

3

Difference Between FFNets and RNNs

If using a a feed-forward neural network to process sequences:

The entire sequence has to be presented.

This is problematic because number of words in a sentence vary.

Additionally, it treats the input more like a bag of words.

It does not necessarily "perceive" the structure of the sentence.

Think "Mary loves John," vs. "John loves Mary."

4

Difference Between FFNets and RNNs

People read sentences word by word.

They keep prior words and the context in memory.

They update their understanding based on the new words encountered.

This is the basic idea of RNNs.

They iterate through the elements of input sequence while maintaining a internal “state”.

The internal state encodes everything it has seen so far.

Admittedly, it is a fairly small state.

5

Example of RNN Processing

Predicting next word.

Consider the following text string: “A girl walked into a bar, and she said ‘Can I have a drink please?’

The bartender said ‘Certainly { }”

There are many options for the next word, as indicated by the curly brackets.

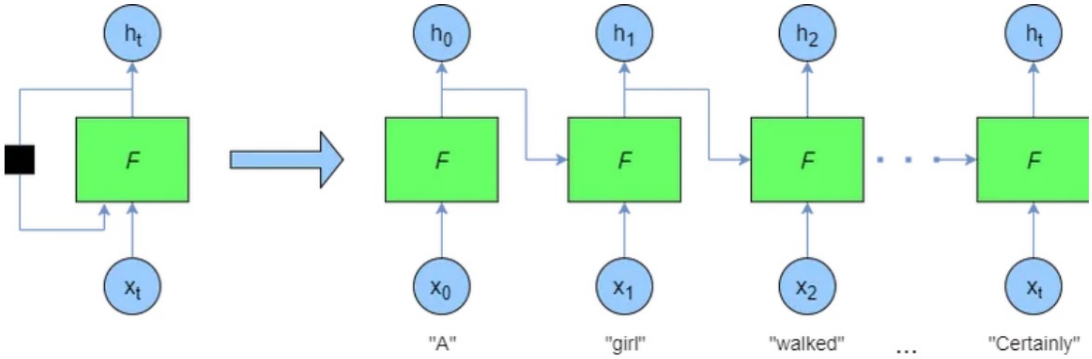
It could be: “miss” or “ma’am.”

However, other words could also fit, such as: “sir” or “mister.”

In order to get the correct gender of the noun, the neural network needs to “recall” that two prior words designating the likely gender (i.e. “girl” and “she”) were used.

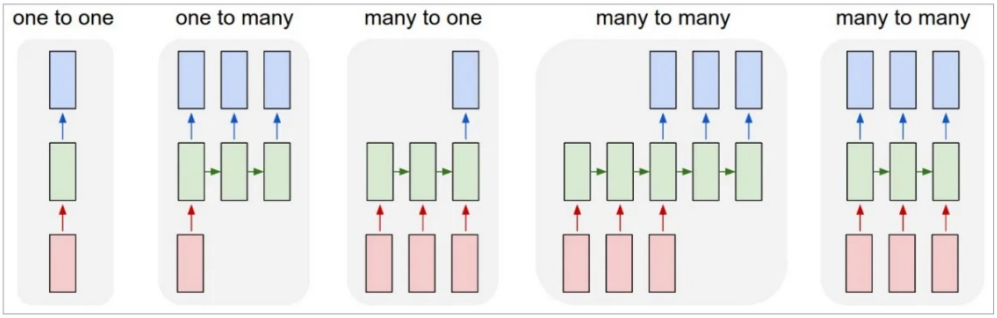
6

Unrolled RNN



7

RNNs for NLP



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

8

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.

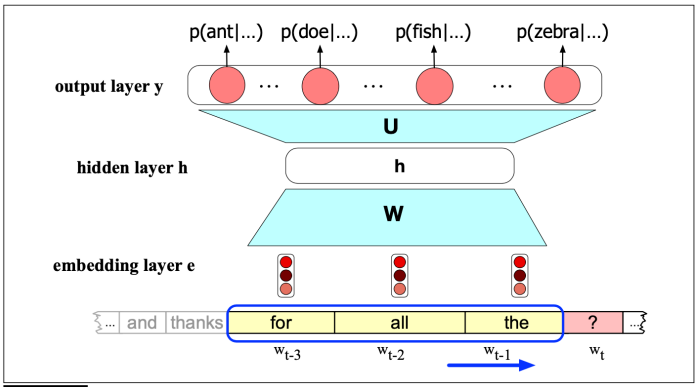


Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 12, 2022.

9

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.

10

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

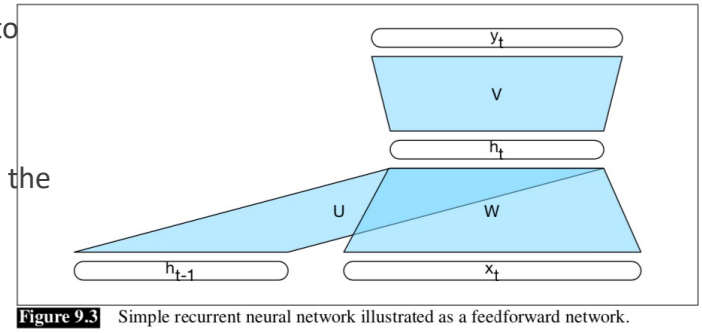


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

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

11

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.

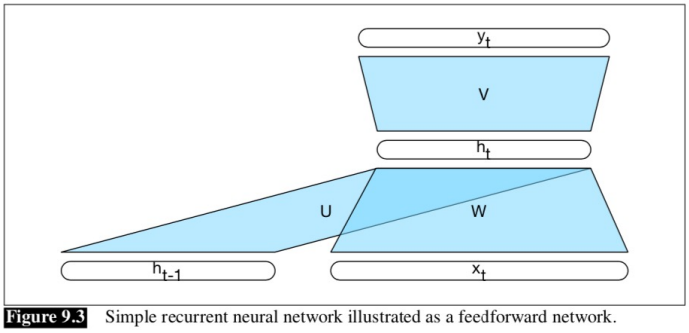


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

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

12

Calculating Activations

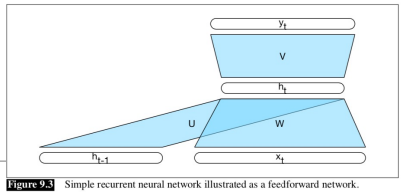


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

function FORWARDRNN(**x**, *network*) **returns** output sequence **y**

$\mathbf{h}_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(**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.

13

Calculations of new Hidden Values

```
for (int t = 0; t < seqLength; t++) {
  double[] hNew = new double[hiddenDim];
  for (int i = 0; i < hiddenDim; i++) {
    double in = 0.0;
    for (int j = 0; j < inputDim; j++) {
      in += W[i][j] * x[t][j];
    }
    for (int k = 0; k < hiddenDim; k++) {
      in += U[i][k] * hPrev[k];
    }
    in += b[i]
    hNew[i] = g(in);
  }
  hPrev = hNew;  // Update for next time step
}
```

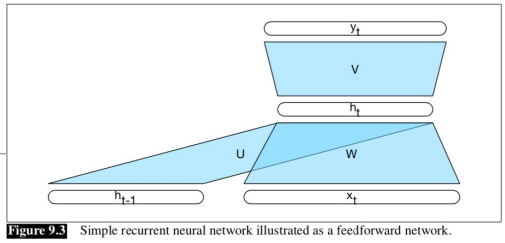


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

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

14

Calculations of new Output Values

```
for (int t = 0; t < seqLength; t++) {
  double[] y = new double[outputDim];
  for (int i = 0; i < outputDim; i++) {
    double in = 0.0;
    for (int j = 0; j < hiddenDim; j++) {
      in += V[i][j] * hNew[j];
    }
    in + by[i];
    }
  y[i] = f(in);
}
```

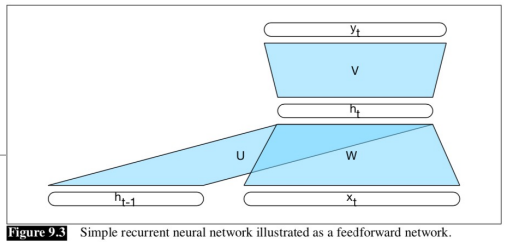


Figure 9.3 Simple recurrent neural network illustrated as a feedforward network.

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

15

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?

16

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.

17

Autoregressive RNN

Used for text generation.

Called "generative AI"

ChatGPT is considered an autoregressive model.

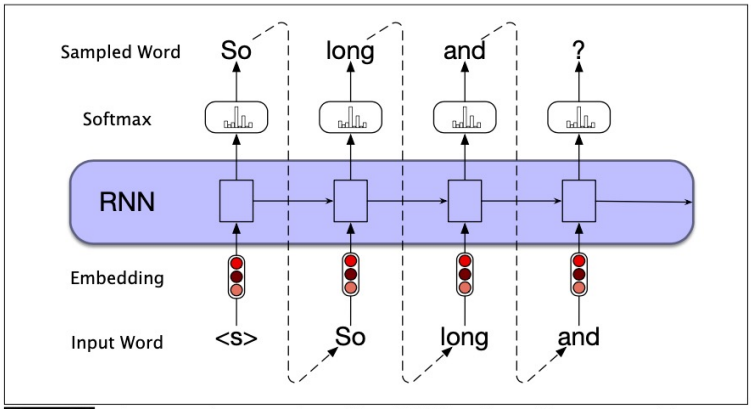


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

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

18

Autoregressive RNN

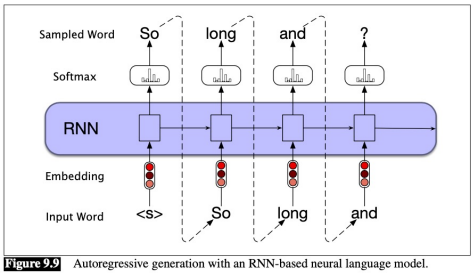


Figure 9.9 Autoregressive generation with an RNN-based neural language model.

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.

19

Example

So long and thanks for

20

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

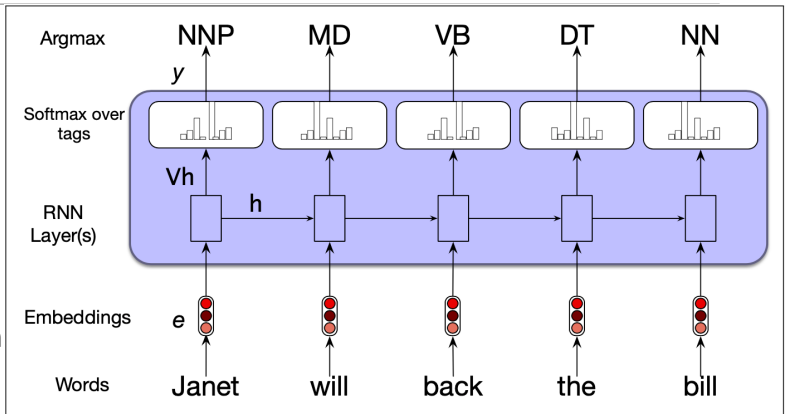


Image source: Jurafsky & Martin, Speech and Language Processing. 3rd Ed., Draft of January 12, 2022.

21

Document Classification with RNNs

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

It can be used for SPAM detection.

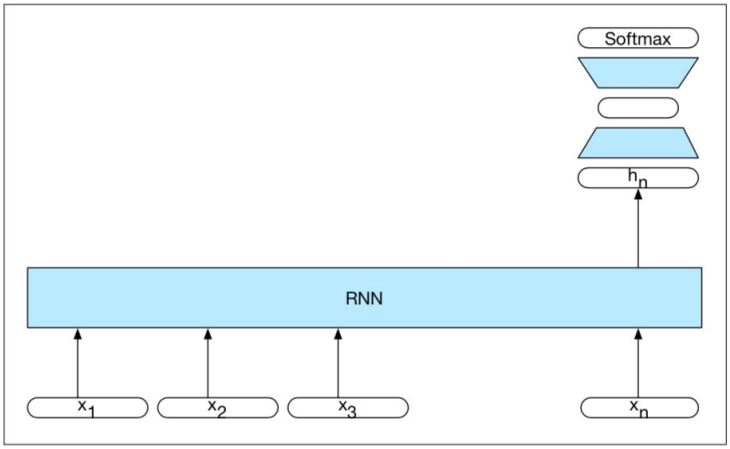


Figure 9.9 Sequence classification using a simple RNN combined with a feedforward network. The final hidden state from the RNN is used as the input to a feedforward network that performs the classification.

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

22

Training RNNs for Next Word Prediction

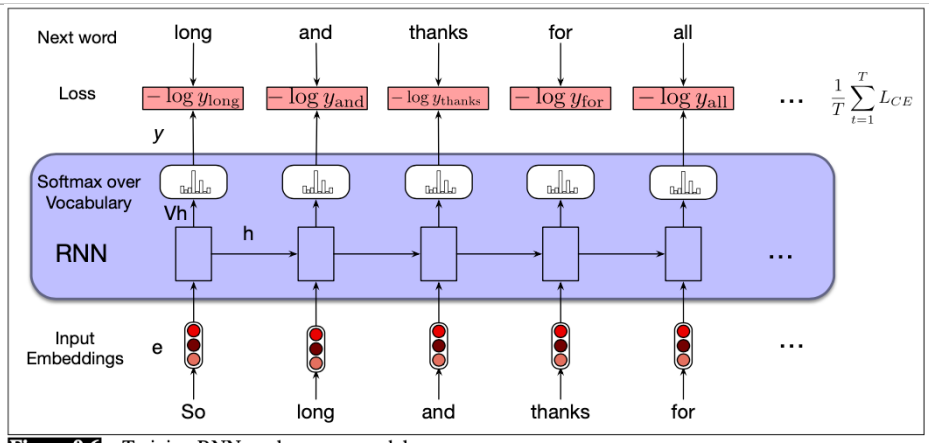


Figure 9.6 Training RNNs as language models.

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

23

Logs

- Logs measures the accuracy of a model's predicted probabilities
- They heavily penalize confident wrong predictions

Log(1) = 0 got it right, no error

Log(0.5) = -0.3

Log(0.1) = -1

Log(0.01) = -2

24

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.

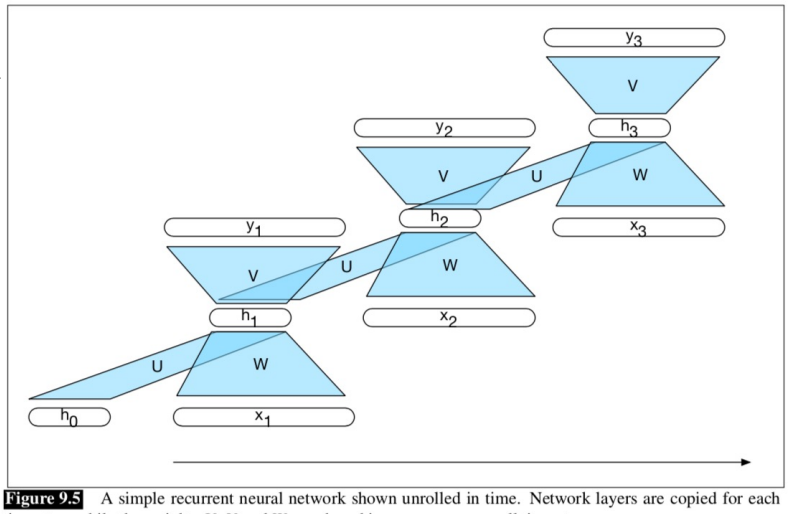


Figure 9.5 A simple recurrent neural network shown unrolled in time. Network layers are copied for each time step, while the weights U, V and W are shared in common across all time steps.

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

25

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 error or loss function for t_3 and y_3.

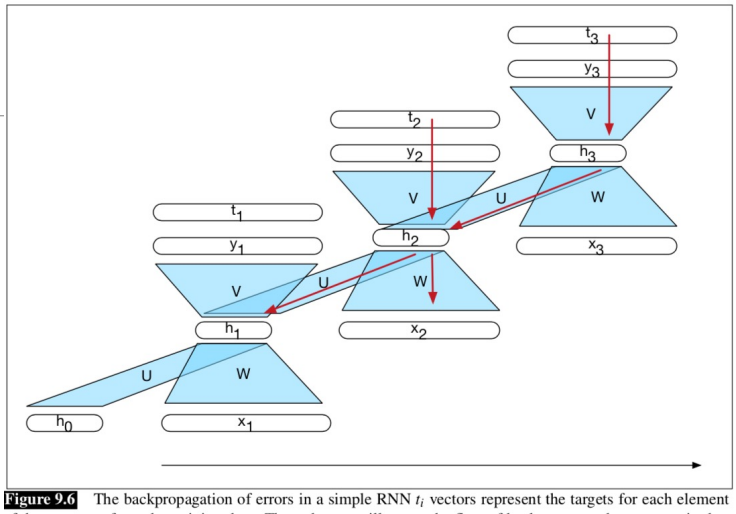


Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging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

26

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.

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

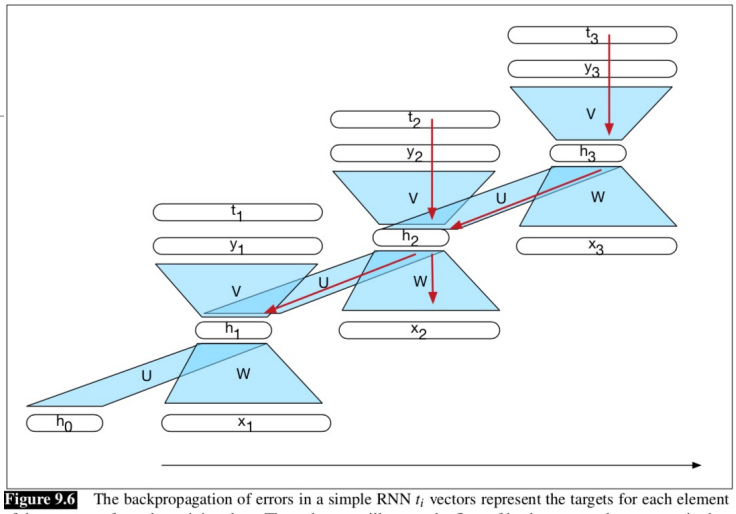
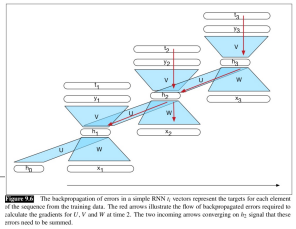


Figure 9.6 The backpropagation of errors in a simple RNN t_i vectors represent the targets for each element of the sequence from the training data. The red arrows illustrate the flow of backpropagated errors required to calculate the gradients for U, V and W at time 2. The two incoming arrows converging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

27

Backpropagation through Time

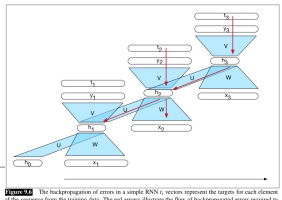


- The backpropagation algorithm has to have access to the errors at the output layer as well as the errors at the hidden layer.
- The errors at the output layer can be calculated at each step and saved.
- The errors at the hidden layer cannot be calculated on the forward pass.
- The activations of the hidden layer for each time step have to be saved.
- In the backpropagation phase, we process the saved data in reverse, computing the errors for all nodes.
- Notice that we additionally have to save the errors at the hidden layers for one time step.
- This approach is commonly referred to as *backpropagation through time.*

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

28

Backpropagation through Time



- Conceptually, we are looping back through the network.
- However, these networks are typically not very large.
- We can simply store all of the intermediate results.
- Think of it as a network with many layers.
- Backpropagation still needs to be going backwards, but it is very much like backpropagation for feed-forward networks.

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

29

Stacked RNNs

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

More weights, more patterns.

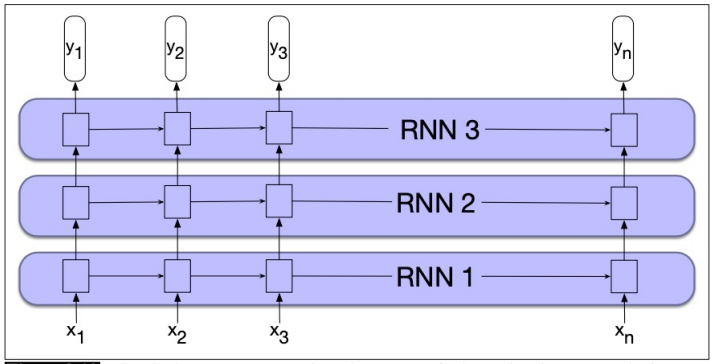


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

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

30

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

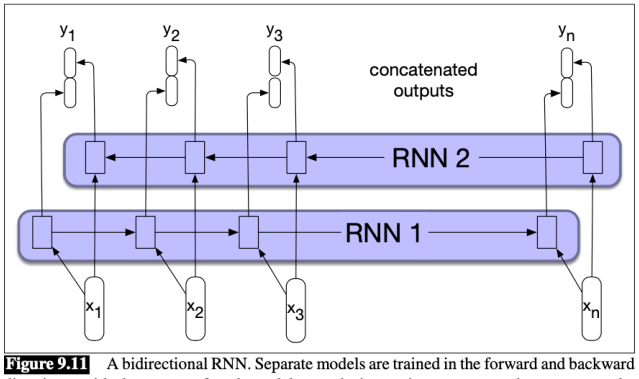


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

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

31

Bidirectional RNNs

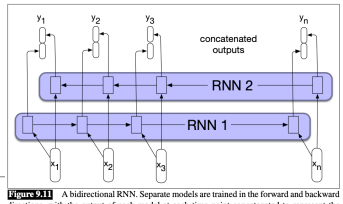


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.

The RNNs discussed so far use 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* as well.

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

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

32

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

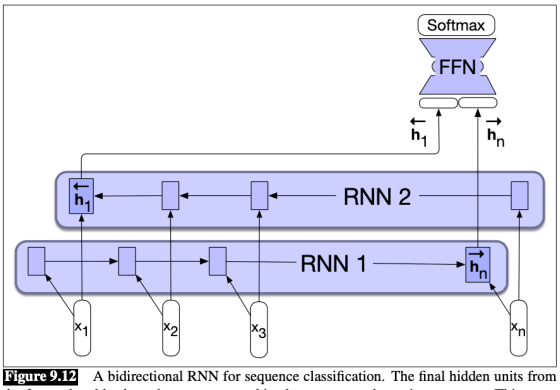


Figure 9.12 A bidirectional RNN for sequence classification. The final hidden units from the forward and backward passes are combined to represent the entire sequence. This combined representation serves as input to the subsequent classifier.

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

33