CSC321 Lecture 15: Recurrent Neural Networks

Roger Grosse

Overview

- Sometimes we're interested in predicting sequences
  - Speech-to-text and text-to-speech
  - Caption generation
  - Machine translation
- If the input is also a sequence, this setting is known as sequence-to-sequence prediction.
- We already saw one way of doing this: neural language models
  - But autoregressive models are memoryless, so they can't learn long-distance dependencies.
  - Recurrent neural networks (RNNs) are a kind of architecture which can remember things over time.

Overview

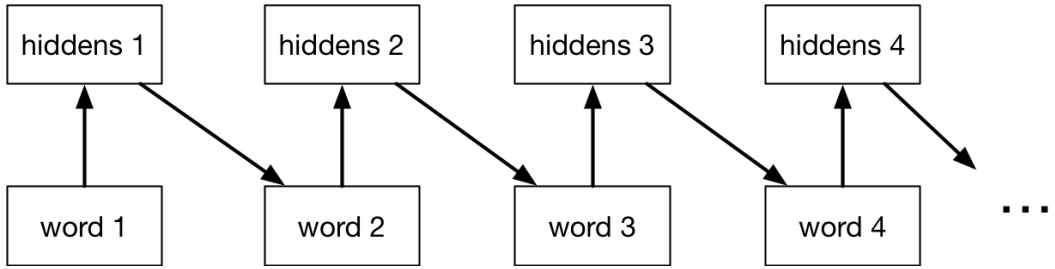
Recall that we made a Markov assumption:

$$p(w_i \mid w_1, \ldots, w_{i-1}) = p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}).$$

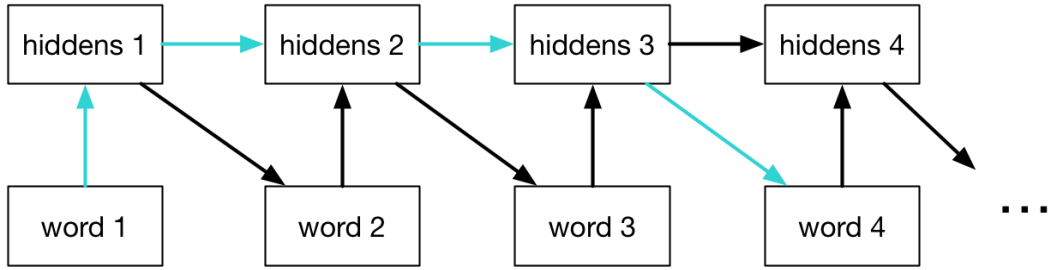
This means the model is memoryless, i.e. it has no memory of anything before the last few words. But sometimes long-distance context can be important.

Overview

- Autoregressive models such as the neural language model are memoryless, so they can only use information from their immediate context (in this figure, context length $= 1$):

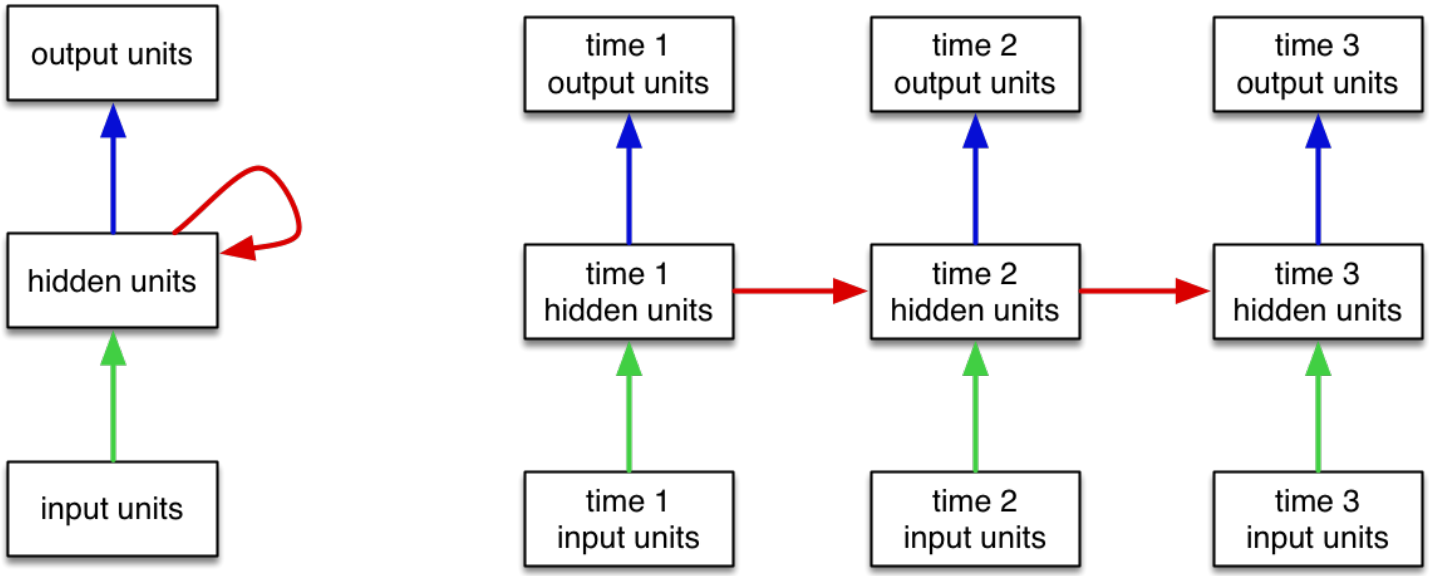


- If we add connections between the hidden units, it becomes a recurrent neural network (RNN). Having a memory lets an RNN use longer-term dependencies:



Recurrent neural nets

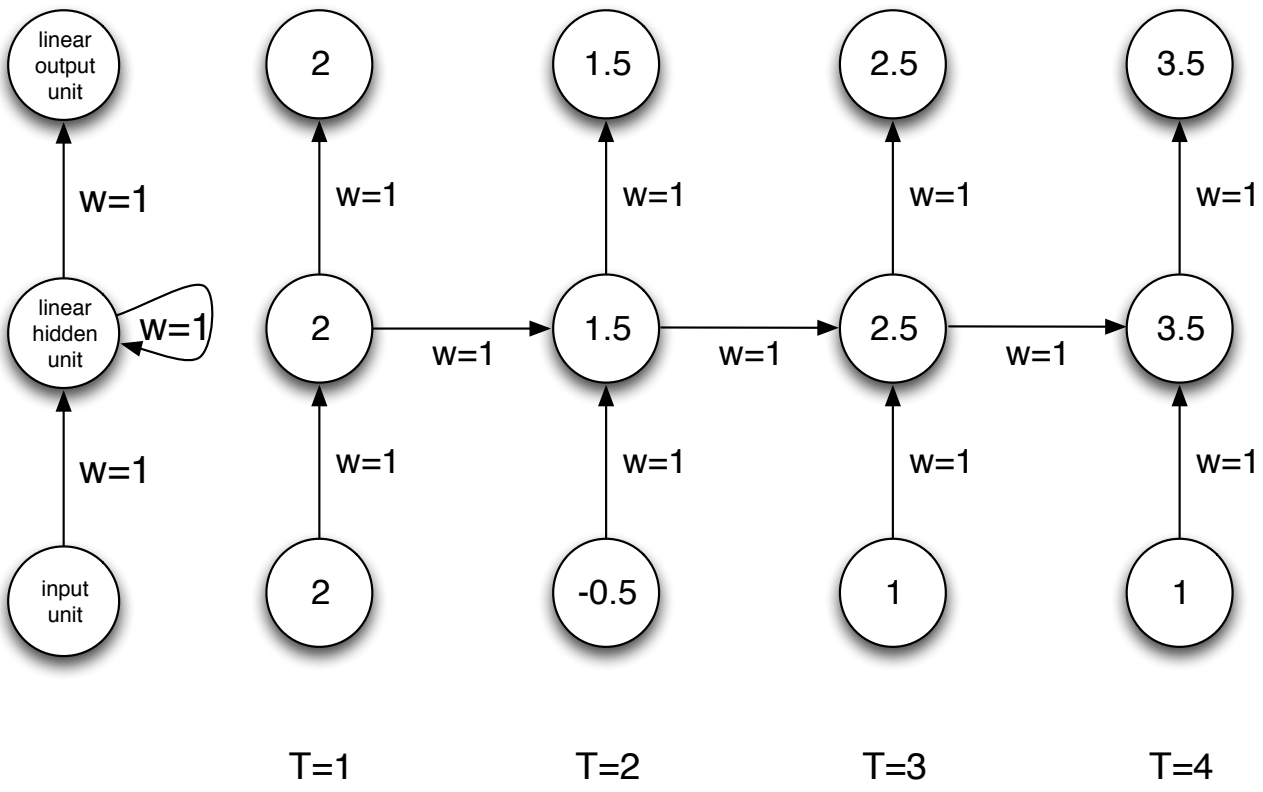
- We can think of an RNN as a dynamical system with one set of hidden units which feed into themselves. The network's graph would then have self-loops.
- We can unroll the RNN's graph by explicitly representing the units at all time steps. The weights and biases are shared between all time steps
  - Except there is typically a separate set of biases for the first time step.



RNN examples

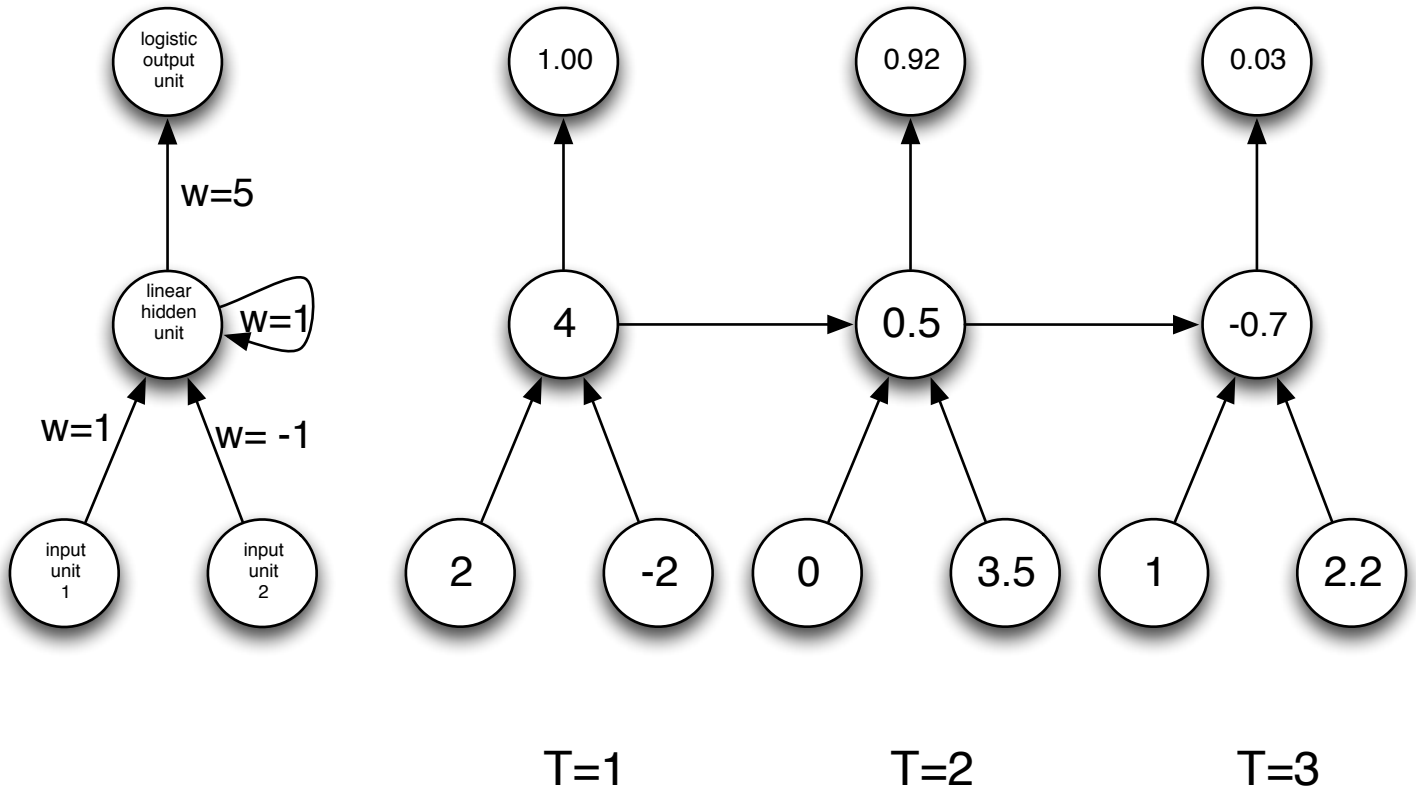
Now let's look at some simple examples of RNNs.

This one sums its inputs:



RNN examples

This one determines if the total values of the first or second input are larger:



Example: Parity

Assume we have a sequence of binary inputs. We'll consider how to determine the parity, i.e. whether the number of 1's is even or odd.

We can compute parity incrementally by keeping track of the parity of the input so far:

Parity bits:	0	1	1	0	1	1	$\longrightarrow$				
Input:	0	1	0	1	1	0	1	0	1	1	

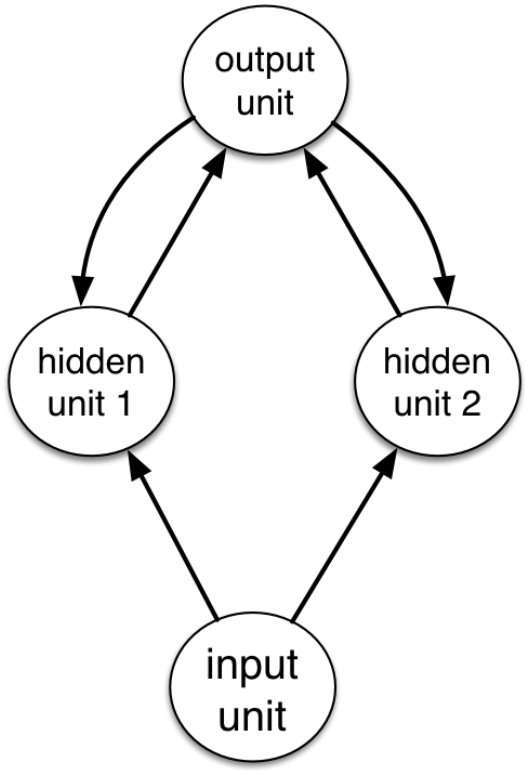
Each parity bit is the XOR of the input and the previous parity bit.

Parity is a classic example of a problem that's hard to solve with a shallow feed-forward net, but easy to solve with an RNN.

Example: Parity

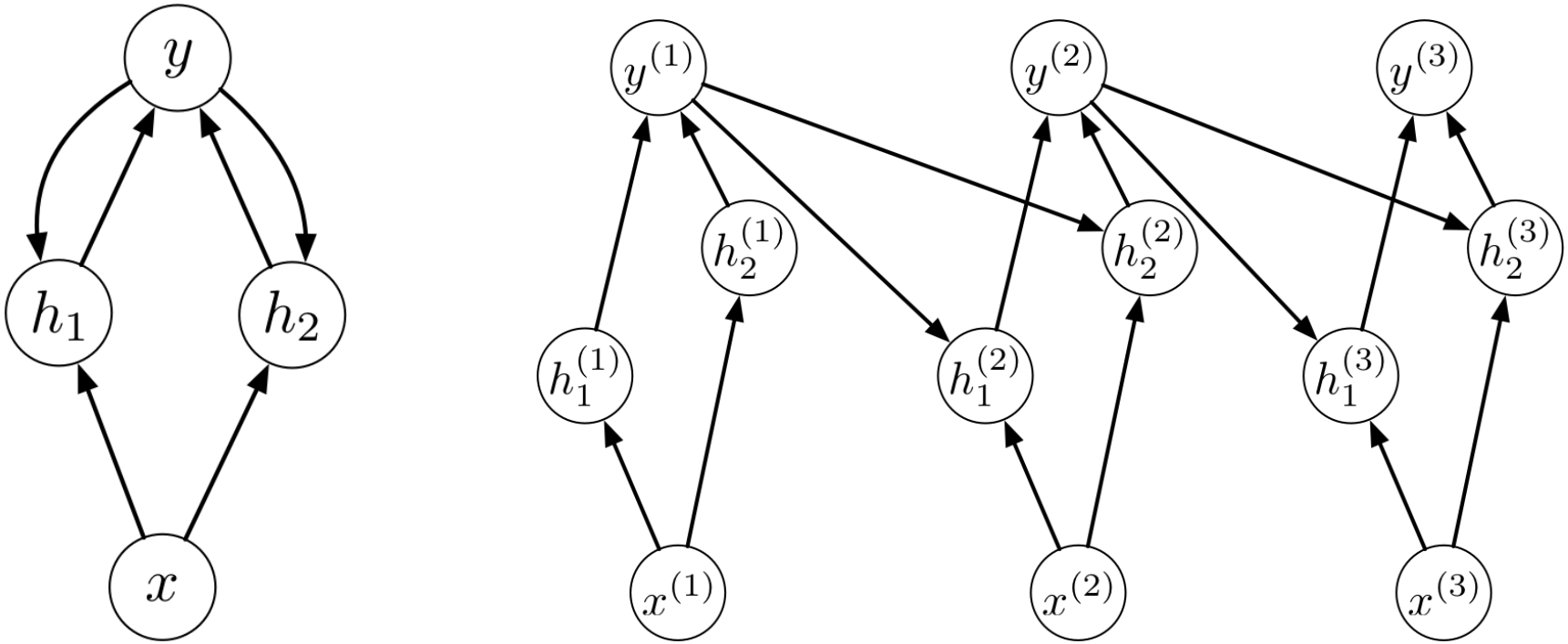
Assume we have a sequence of binary inputs. We'll consider how to determine the parity, i.e. whether the number of 1's is even or odd.

- Let's find weights and biases for the RNN on the right so that it computes the parity. All hidden and output units are **binary threshold units**.
- **Strategy:**
  - The output unit tracks the current parity, which is the XOR of the current input and previous output.
  - The hidden units help us compute the XOR.



Example: Parity

Unrolling the parity RNN:



Example: Parity

The output unit should compute the XOR of the current input and previous output:

$y^{(t-1)}$	$x^{(t)}$	$y^{(t)}$
0	0	0
0	1	1
1	0	1
1	1	0

Example: Parity

Let's use hidden units to help us compute XOR.

- Have one unit compute AND, and the other one compute OR.
- Then we can pick weights and biases just like we did for multilayer perceptrons.

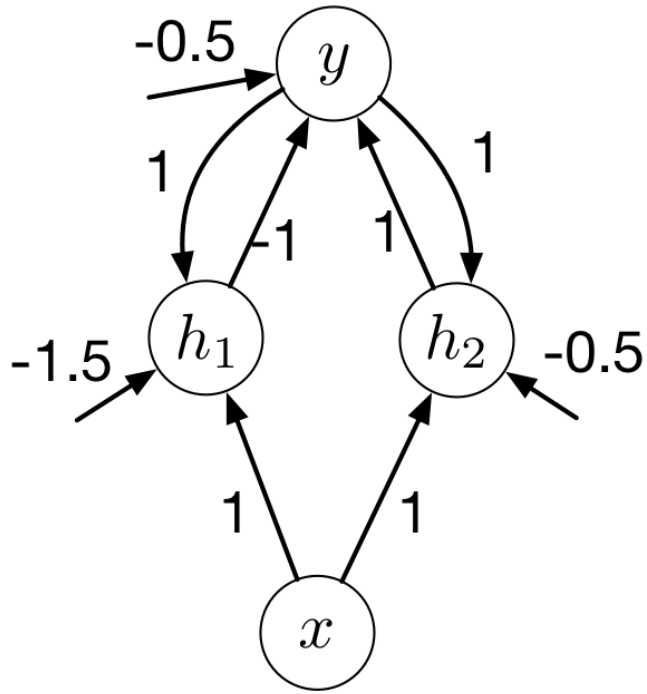
$y^{(t-1)}$	$x^{(t)}$	$h_1^{(t)}$	$h_2^{(t)}$	$y^{(t)}$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Example: Parity

Let's use hidden units to help us compute XOR.

- Have one unit compute AND, and the other one compute OR.
- Then we can pick weights and biases just like we did for multilayer perceptrons.

$y^{(t-1)}$	$x^{(t)}$	$h_1^{(t)}$	$h_2^{(t)}$	$y^{(t)}$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

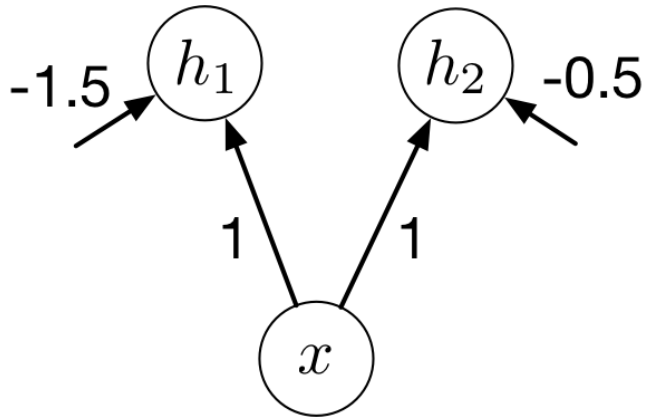


Example: Parity

We still need to determine the hidden biases for the first time step.

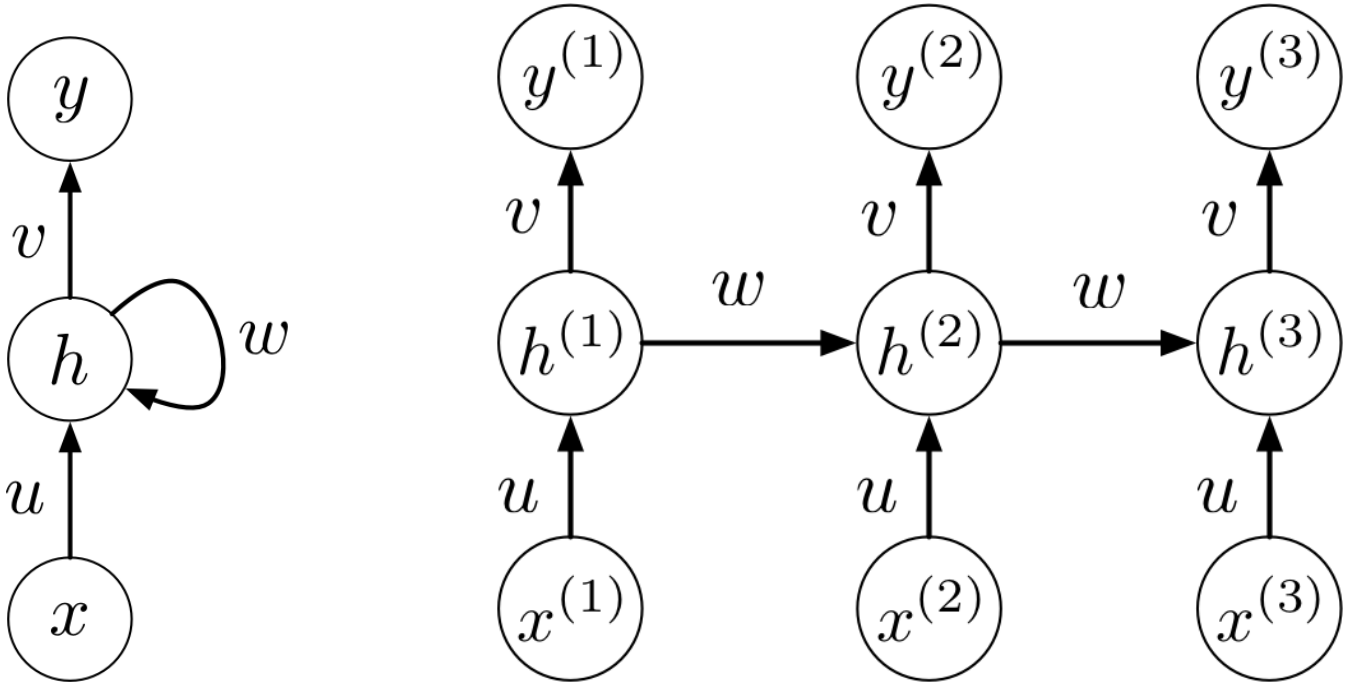
- The network should behave as if the previous output was 0. This is represented with the following table:

$x^{(1)}$	$h_1^{(1)}$	$h_2^{(1)}$
0	0	0
1	0	1



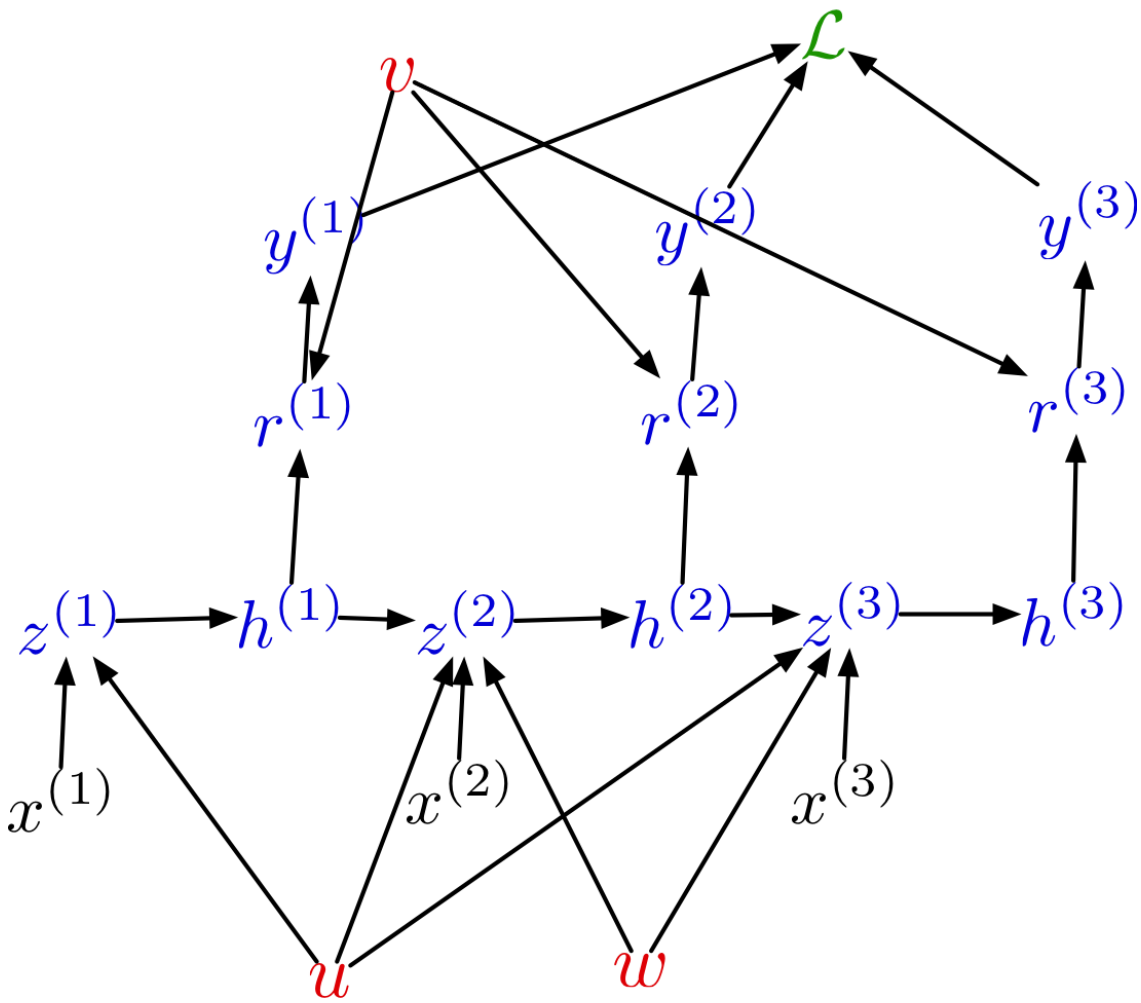
Backprop Through Time

- As you can guess, we don't usually set RNN weights by hand. Instead, we learn them using backprop.
- In particular, we do backprop on the unrolled network. This is known as backprop through time.

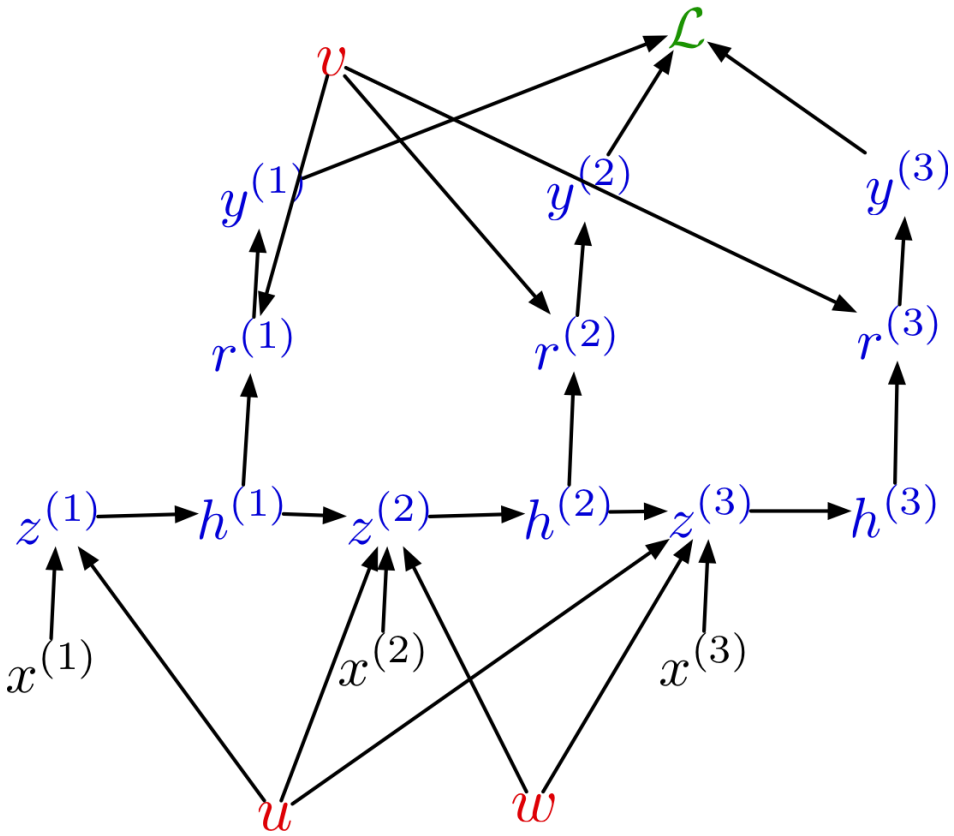


Backprop Through Time

Here's the unrolled computation graph. Notice the weight sharing.



Backprop Through Time



Activations:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y^{(t)}} = \overline{\mathcal{L}} \frac{\partial \mathcal{L}}{\partial y^{(t)}}$$

$$\overline{r^{(t)}} = \overline{y^{(t)}} \, \phi'(r^{(t)})$$

$$\overline{h^{(t)}} = \overline{r^{(t)}} \, v + \overline{z^{(t+1)}} \, w$$

$$\overline{z^{(t)}} = \overline{h^{(t)}} \, \phi'(z^{(t)})$$

Parameters:

$$\overline{u} = \sum_t \overline{z^{(t)}} \, x^{(t)}$$

$$\overline{v} = \sum_t \overline{r^{(t)}} \, h^{(t)}$$

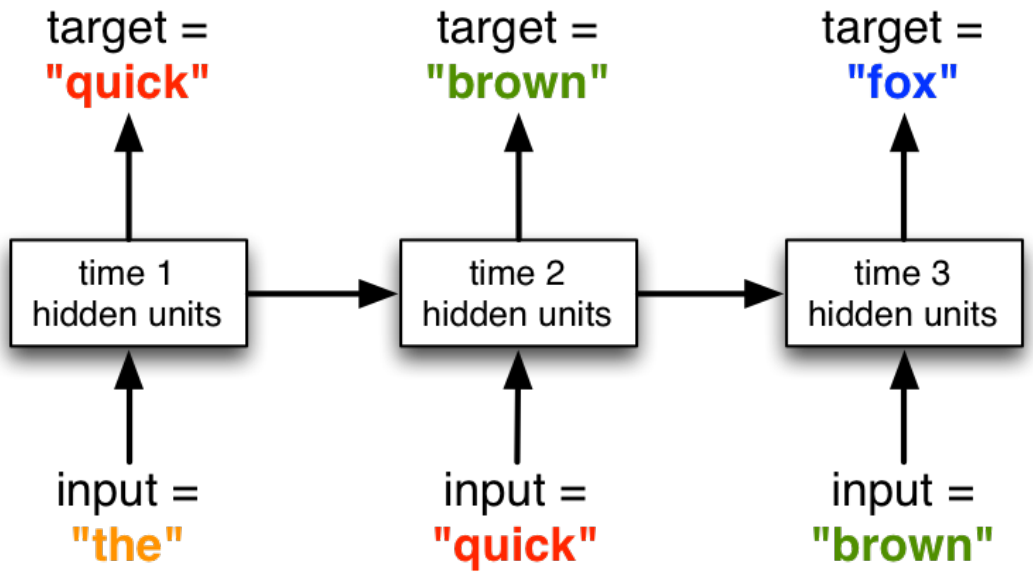
$$\overline{w} = \sum_t \overline{z^{(t+1)}} \, h^{(t)}$$

Backprop Through Time

- Now you know how to compute the derivatives using backprop through time.
- The hard part is using the derivatives in optimization. They can explode or vanish. Addressing this issue will take all of the next lecture.

Language Modeling

One way to use RNNs as a language model:

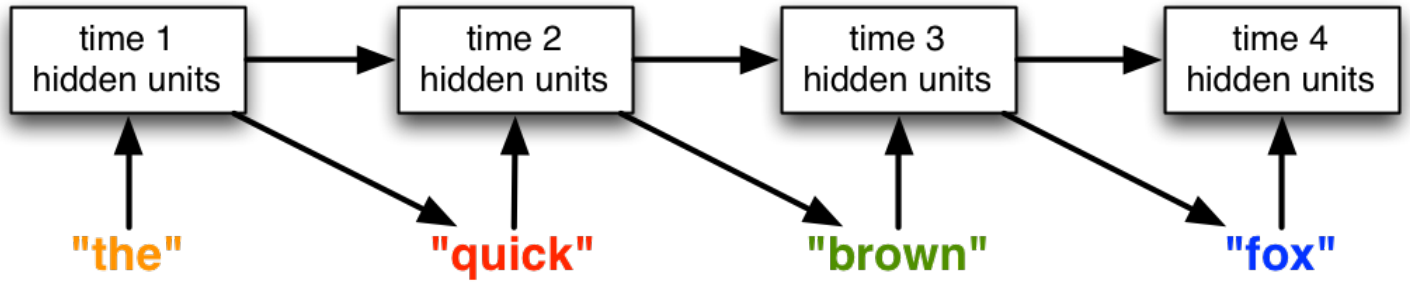


As with our language model, each word is represented as an indicator vector, the model predicts a distribution, and we can train it with cross-entropy loss.

This model can learn long-distance dependencies.

Language Modeling

When we generate from the model (i.e. compute samples from its distribution over sentences), the outputs feed back in to the network as inputs.



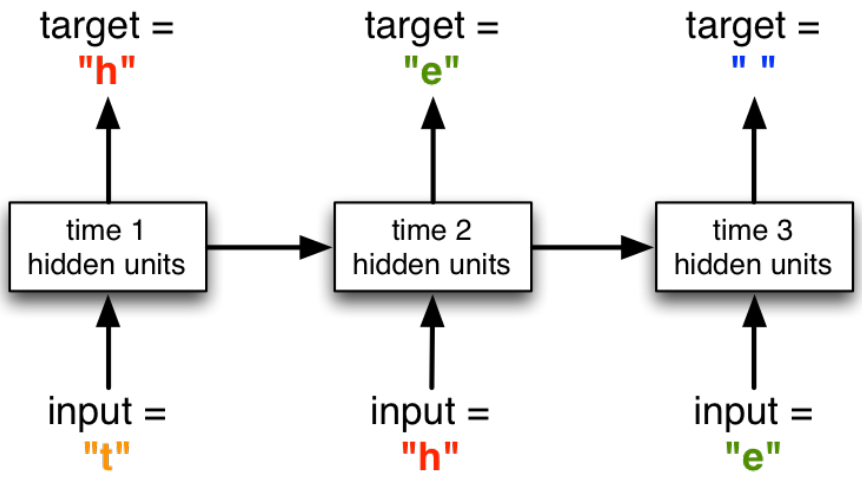
At training time, the inputs are the tokens from the training set (rather than the network's outputs). This is called teacher forcing.

Some remaining challenges:

- Vocabularies can be very large once you include people, places, etc. It's computationally difficult to predict distributions over millions of words.
- How do we deal with words we haven't seen before?
- In some languages (e.g. German), it's hard to define what should be considered a word.

Language Modeling

Another approach is to model text *one character at a time*!



This solves the problem of what to do about previously unseen words. Note that long-term memory is *essential* at the character level!

Note: modeling language well at the character level requires *multiplicative* interactions, which we're not going to talk about.

Language Modeling

From Geoff Hinton's Coursera course, an example of a paragraph generated by an RNN language model one character at a time:

> He was elected President during the Revolutionary War and forgave Opus Paul at Rome. The regime of his crew of England, is now Arab women's icons in and the demons that use something between the characters‘ sisters in lower coil trains were always operated on the line of the ephemerable street, respectively, the graphic or other facility for deformation of a given proportion of large segments at RTUS). The B every chord was a "strongly cold internal palette pour even the white blade.”

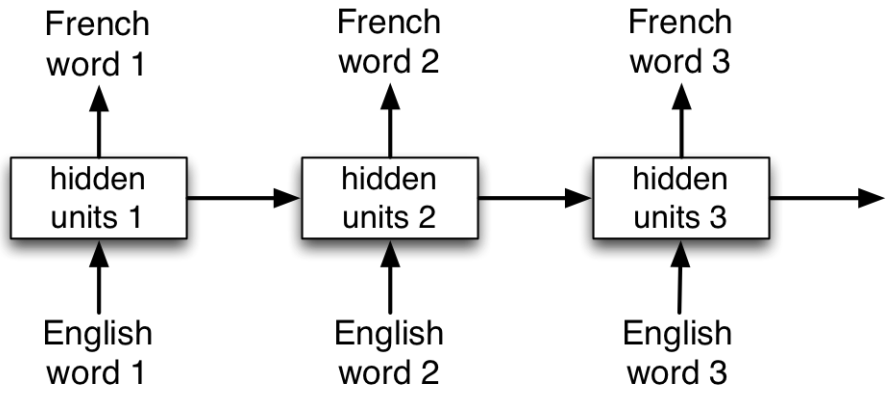
J. Martens and I. Sutskever, 2011. Learning recurrent neural networks with Hessian-free optimization.

`http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Martens_532.pdf`

Neural Machine Translation

We'd like to translate, e.g., English to French sentences, and we have pairs of translated sentences to train on.

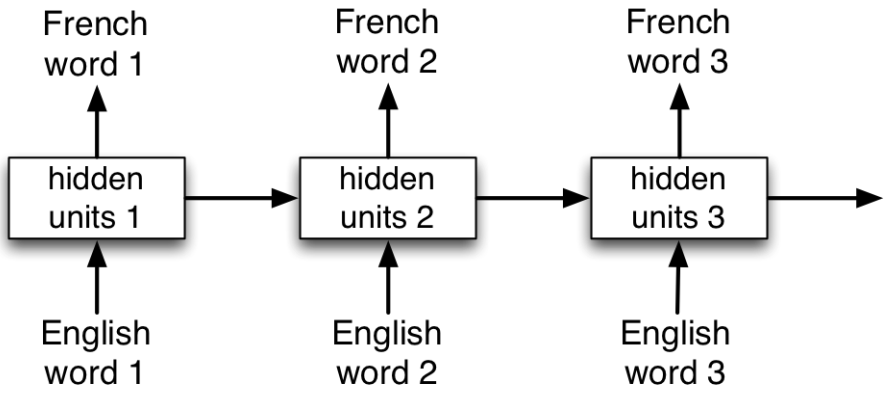
What's wrong with the following setup?



Neural Machine Translation

We'd like to translate, e.g., English to French sentences, and we have pairs of translated sentences to train on.

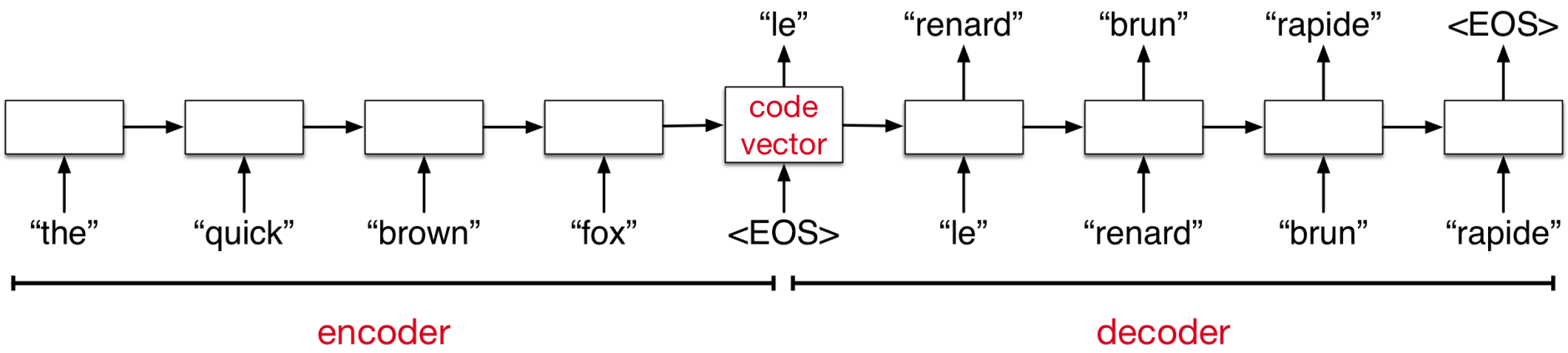
What's wrong with the following setup?



- The sentences might not be the same length, and the words might not align perfectly.
- You might need to resolve ambiguities using information from later in the sentence.

Neural Machine Translation

Encoder-decoder architecture: the network first reads and memorizes the sentence. When it sees the end token, it starts outputting the translation.



The encoder and decoder are two different networks with different weights.

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio. EMNLP 2014.

Sequence to Sequence Learning with Neural Networks, Ilya Sutskever, Oriol Vinyals and Quoc Le, NIPS 2014.

What can RNNs compute?

In 2014, Google researchers built an encoder-decoder RNN that learns to execute simple Python programs, *one character at a time*!

Input:

```
j=8584
for x in range(8):
  j+=920
b=(1500+j)
print((b+7567))
```

Target: 25011.

Input:

```
i=8827
c=(i-5347)
print((c+8704) if 2641<8500 else
    5308)
```

Target: 1218.

Example training inputs

Input:

```
vqppkn
sqdvfljmnc
y2vxdddsepnimcbvubkomhrpliibtwztbljipcc
```

Target: hkhpg

A training input with characters scrambled

W. Zaremba and I. Sutskever, "Learning to Execute." `http://arxiv.org/abs/1410.4615`

What can RNNs compute?

Some example results:

Input:

```
print(6652).
```

Target:	6652.
"Baseline" prediction:	6652.
"Naive" prediction:	6652.
"Mix" prediction:	6652.
"Combined" prediction:	6652.

Input:

```
print((5997-738)).
```

Target:	5259.
"Baseline" prediction:	5101.
"Naive" prediction:	5101.
"Mix" prediction:	5249.
"Combined" prediction:	5229.

Input:

```
d=5446
for x in range(8):d+=(2678 if 4803<2829 else 9848)
print((d if 5935<4845 else 3043)).
```

Target:	3043.
"Baseline" prediction:	3043.
"Naive" prediction:	3043.
"Mix" prediction:	3043.
"Combined" prediction:	3043.

Input:

```
print(((1090-3305)+9466)).
```

Target:	7251.
"Baseline" prediction:	7111.
"Naive" prediction:	7099.
"Mix" prediction:	7595.
"Combined" prediction:	7699.

Take a look through the results (`http://arxiv.org/pdf/1410.4615v2.pdf#page=10`). It's fun to try to guess from the mistakes what algorithms it's discovered.