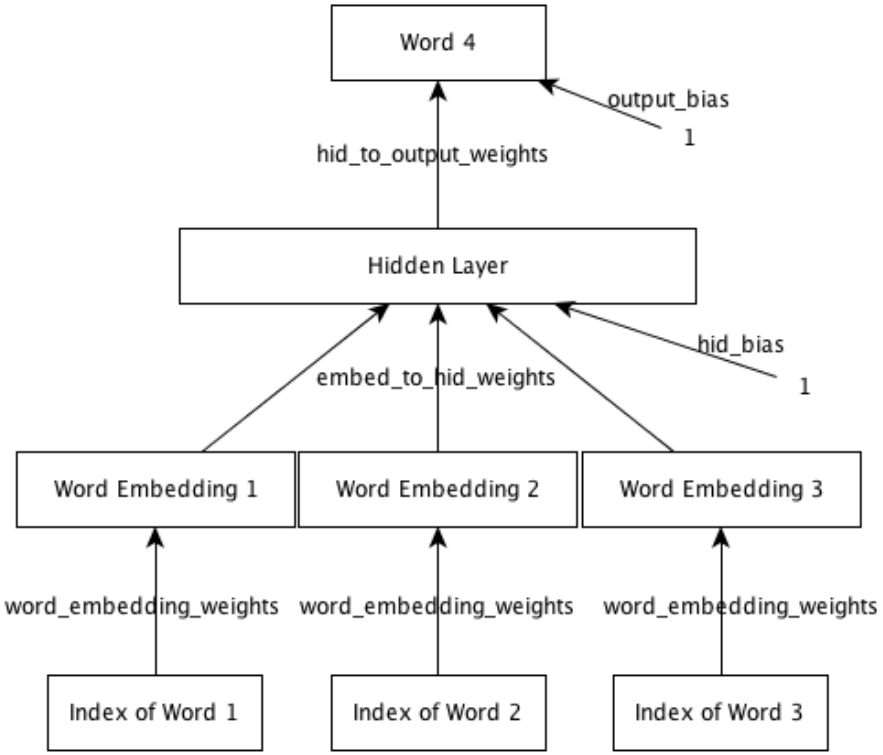
CSC321 Lecture 9
Recurrent neural nets

Roger Grosse and Nitish Srivastava

February 3, 2015

Overview

You just implemented a neural probabilistic language model for Assignment 1:



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.

Overview

Recall that we made a Markov assumption:

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

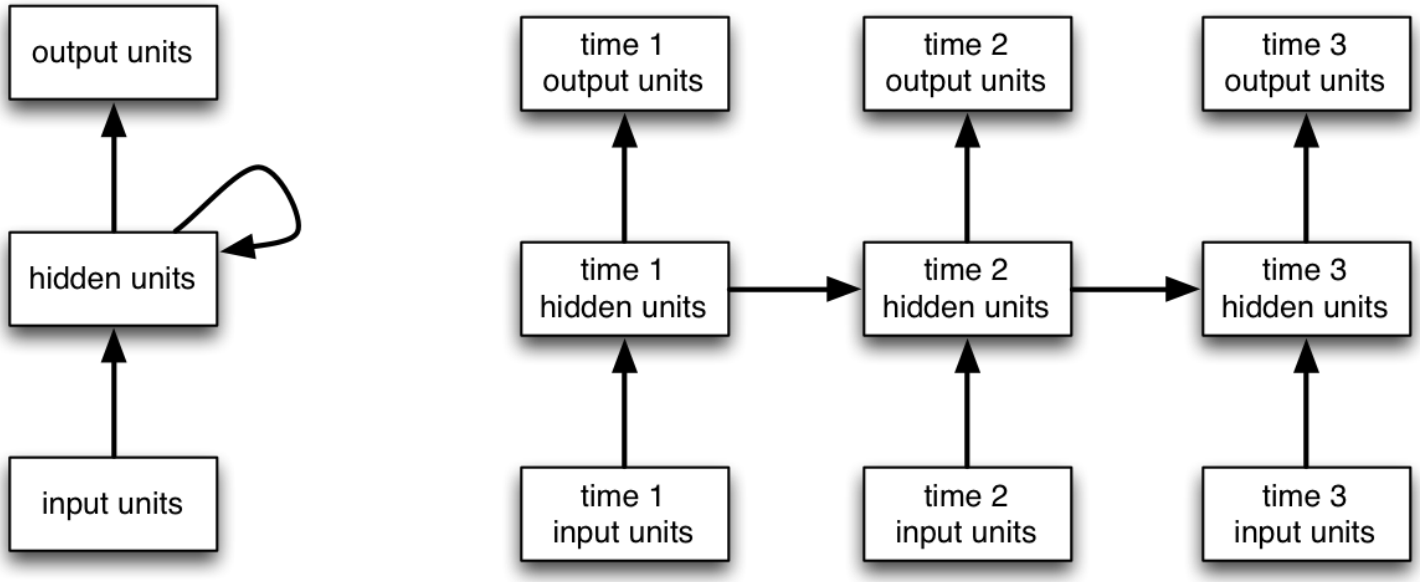
This means the model is memoryless, i.e. it has no memory of anything before the last few words.

But sometimes long-distance context can be important:

> Rob Ford told the flabbergasted reporters assembled at the press conference that _____.

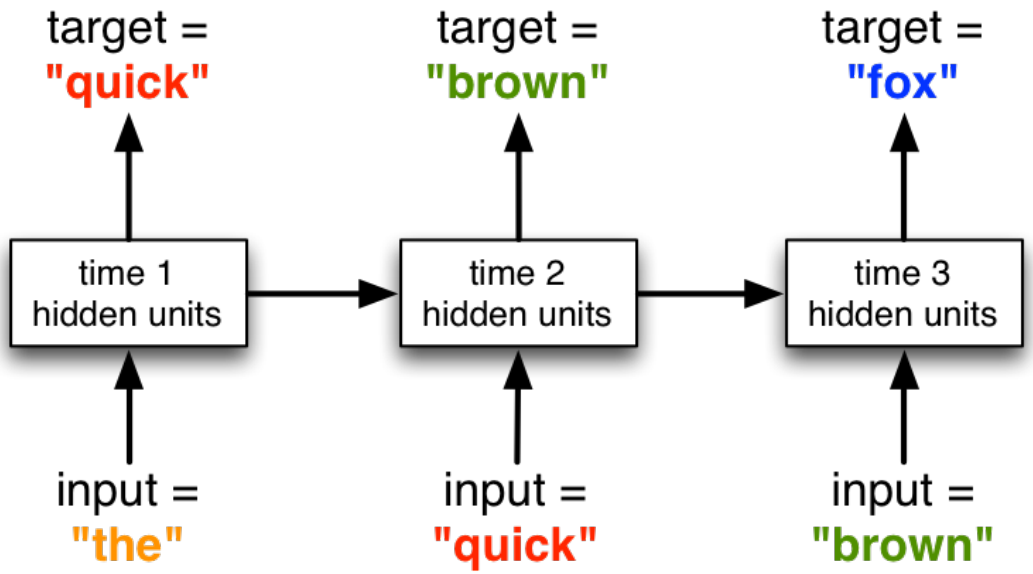
Recurrent neural nets

RNNs are a kind of neural net model which use hidden units to remember things over time. When we compute with them, we unroll them over time:



Recurrent neural nets

One way to use RNNs to model text:

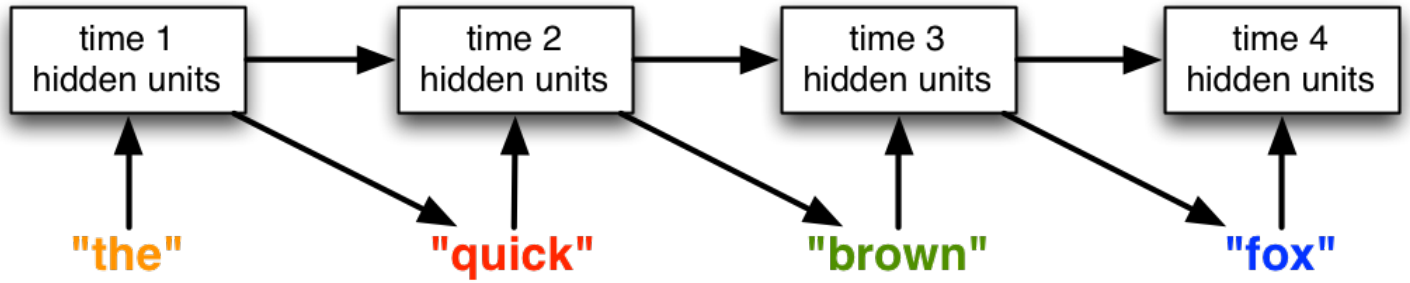


As with our language model, each word is represented as an indicator vector, the model predicts a distribution, and we can train it with cross-entropy loss.

This model can learn long-distance dependencies.

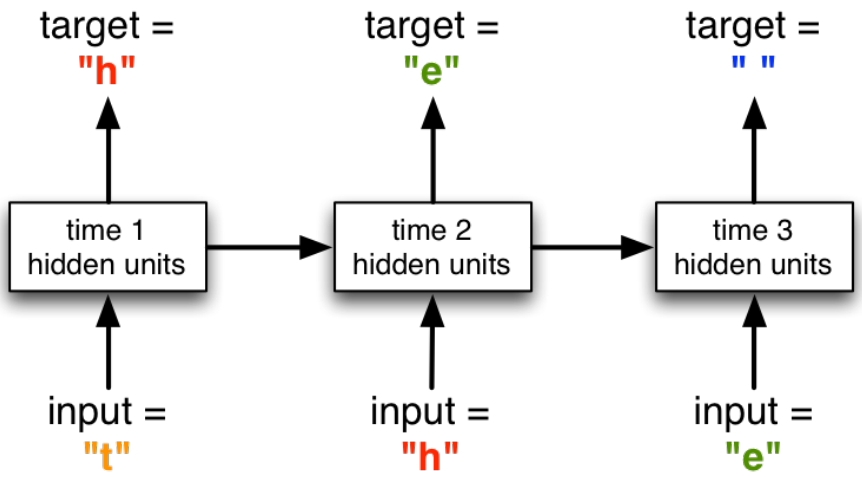
Recurrent neural nets

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.



Recurrent neural nets

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.

Recurrent neural nets

From Geoff's lecture video (optional Lecture H), 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.”

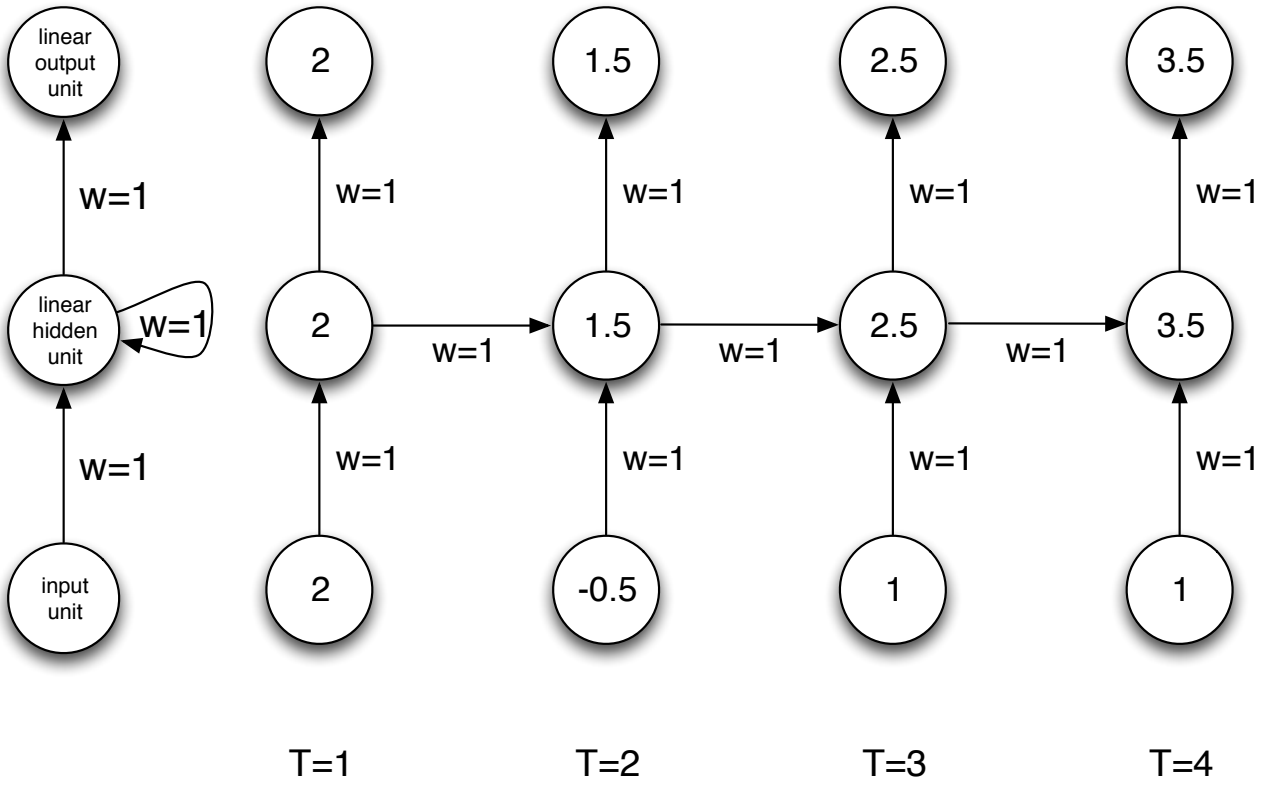
J. Martens and I. Sutskever, 2011. Learning recurrent neural networks with Hessian-free optimization.

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

Question 1: RNN examples

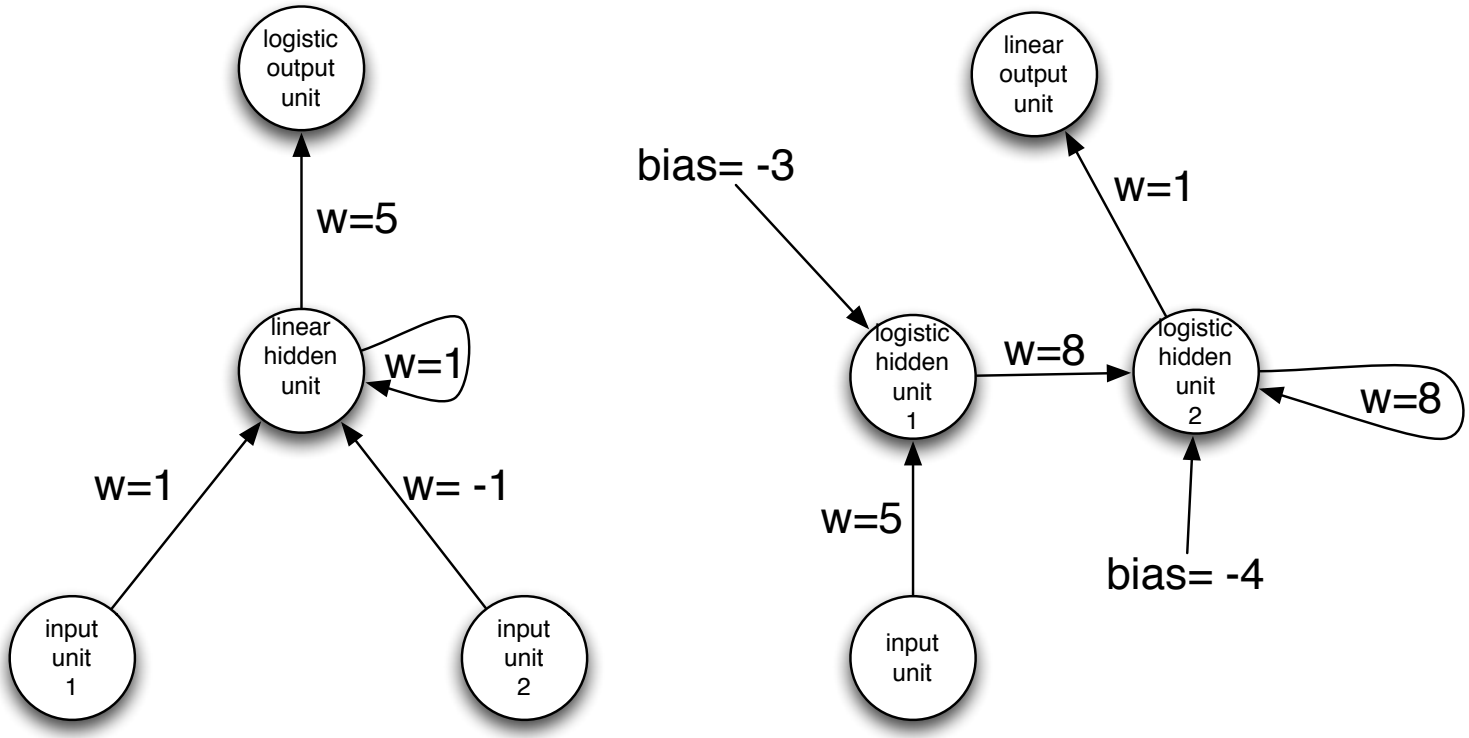
Now let's look at some simple examples of RNNs.

This one sums its inputs:



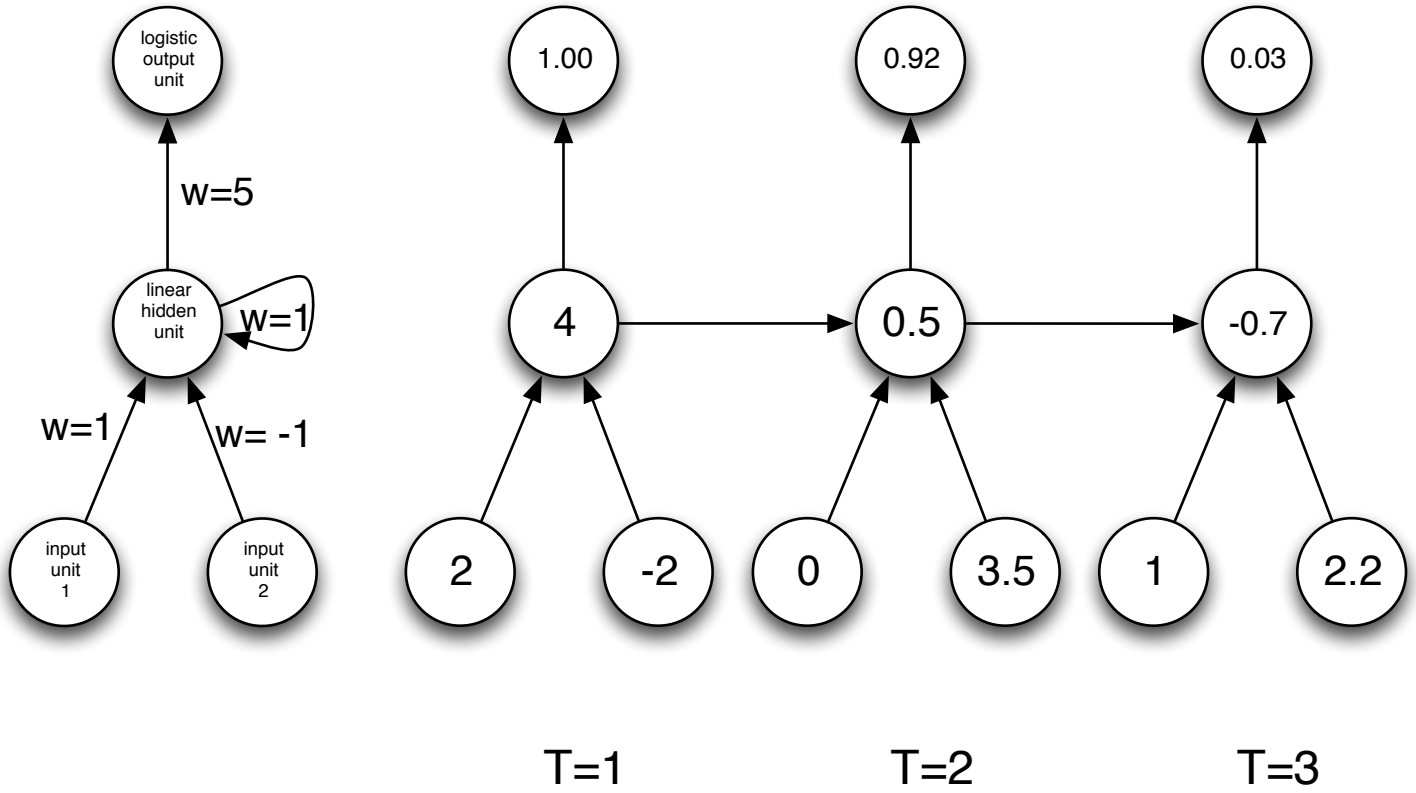
Question 1: RNN examples

What do these RNNs do?



Question 1: RNN examples

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



Finite state machines

A major motivation for RNNs is that they can perform computations. Finite state machines (FSMs) are a simple model of computation.

Intuitively, they perform computations that require only finite memory.

Finite state machines

A major motivation for RNNs is that they can perform computations. Finite state machines (FSMs) are a simple model of computation.

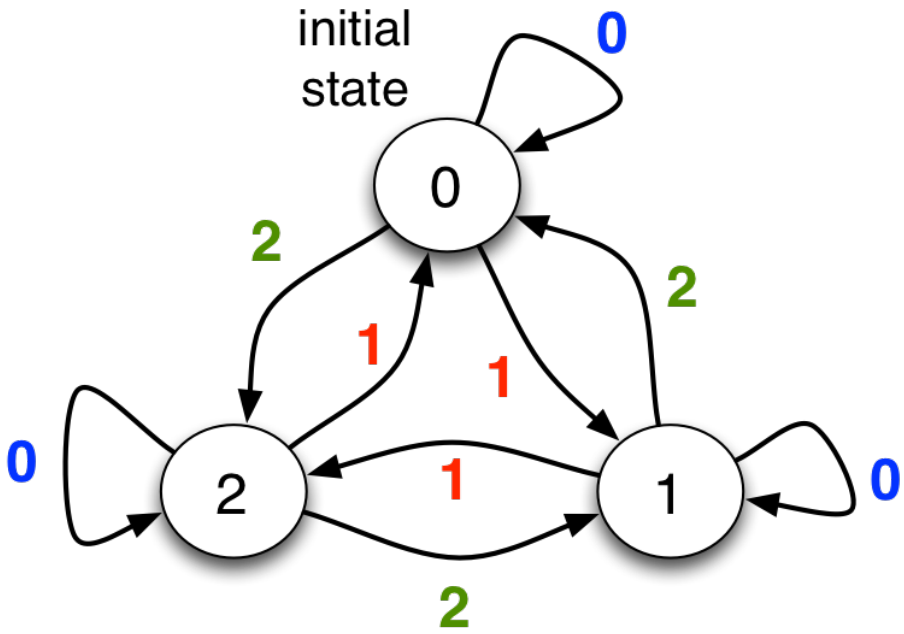
Intuitively, they perform computations that require only finite memory.

We can represent a FSM as a graph.

- Each node denotes a state that the FSM can be in.
- It reads its input one symbol at a time.
- After reading the input, it transitions to some other state according to the edge label.

Finite state machines

What does this FSM do?



Question 2: 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	

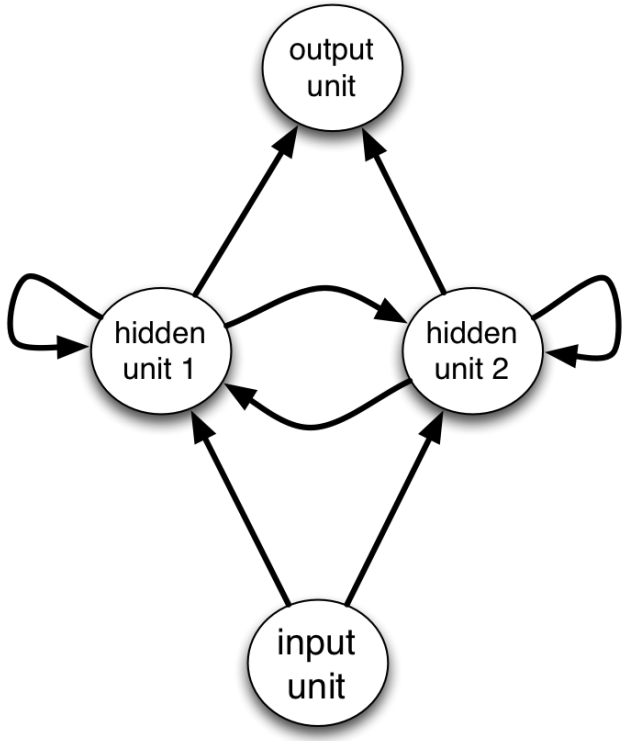
Each parity bit is the XOR of the input and the previous parity bit.

Question 2: 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.

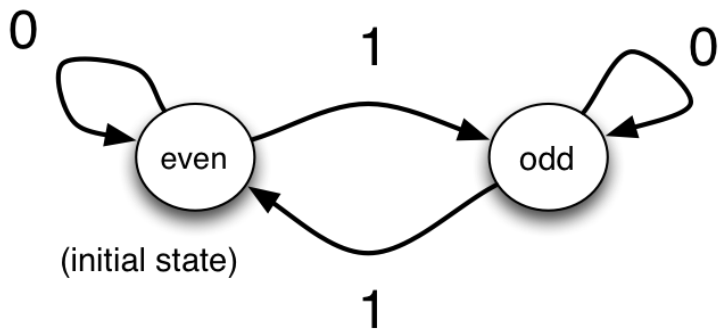
1. Write down a finite state machine which determines the parity.
2. Fill in the weights and biases for the RNN on the right so that it computes the parity. All hidden and output units are **binary threshold units**.

   Hint: Have one hidden unit represent the conjunction of the input and the previous parity bit, and the other hidden unit represent the disjunction.



Question 2: Parity

This FSM determines the parity:

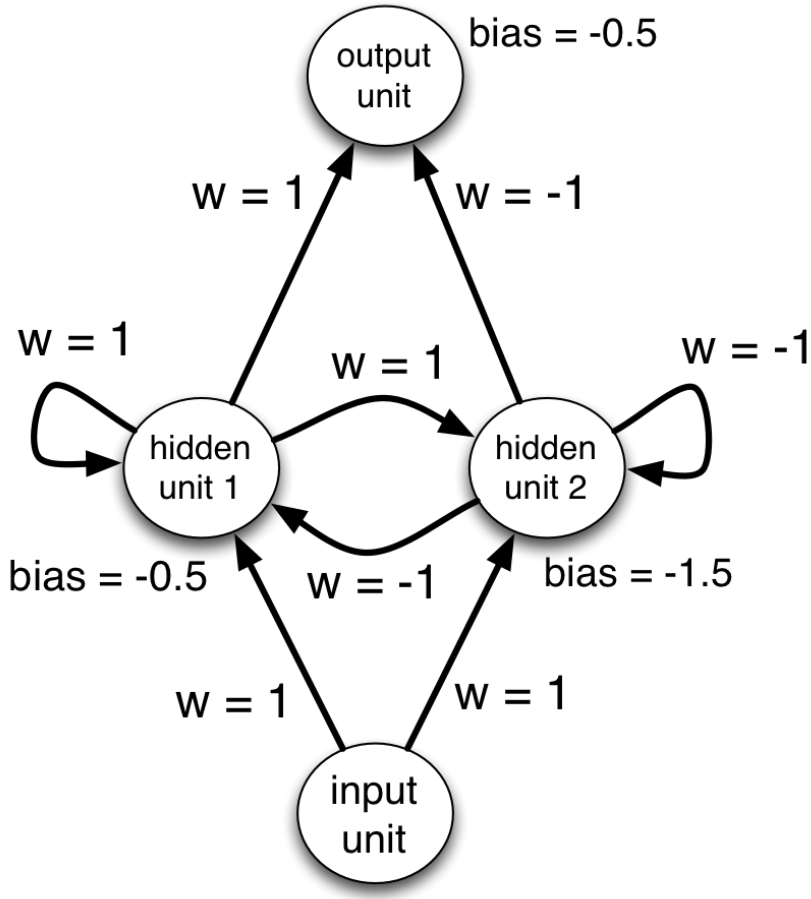


Question 2: Parity

We can't get by with just one hidden unit, since we need to solve the XOR problem at each time step.

Recall that a feed-forward network can compute XOR if we have one hidden unit compute the disjunction of its inputs and the other one the conjunction.

This RNN computes the parity using this strategy. One hidden unit computes the disjunction of the input with the previous parity bit, and the other computes the conjunction.



Question 2: Parity

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

Geoff said that "a recurrent network can emulate a finite state automaton."

- In our last example, we designed an RNN by inspection. If you want a fun challenge, try to come up with a procedure for converting a FSM into an RNN.
- The parity example makes a good test case.
- **Hint:** you'll need lots of hidden units.

What can RNNs compute?

In 2014, Google researchers built an 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.