Recurrent Neural Networks

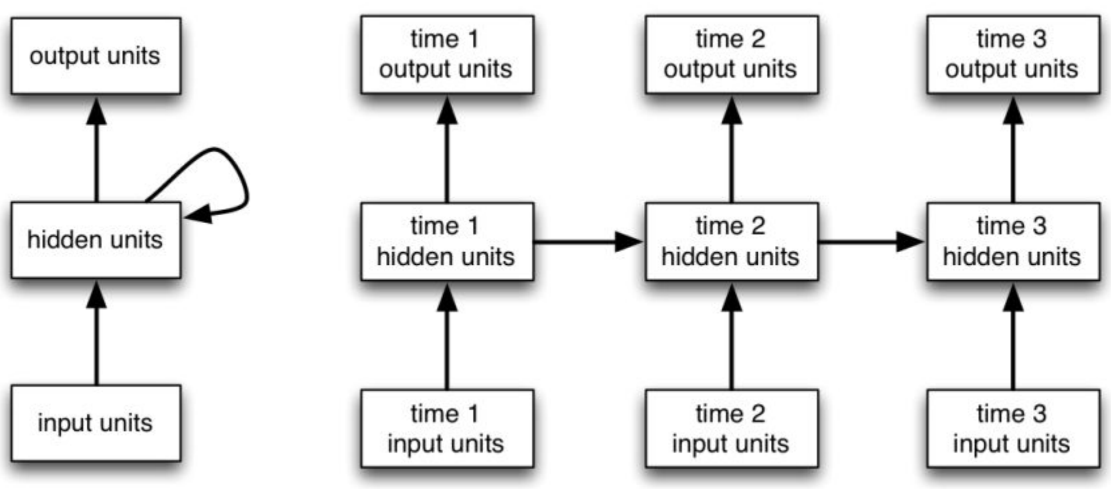
Rajat Shah (Team #18)

Why Recurrent Neural Networks (RNNs)?

- Problems requiring history/context to generate proper output: speech recognition, stock forecasting.
- Much natural way of handling sequential data.
    - Even if your data is not in form of sequences, you can still formulate and train powerful models that learn to process it sequentially. You're learning stateful programs that process your fixed-sized data.
- Has internal state, like state machine gives output for the current input depending on the *current state*.
- Can be trained with Backpropagation through time, but efficient training is a problem.

RNN Overview

- 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:

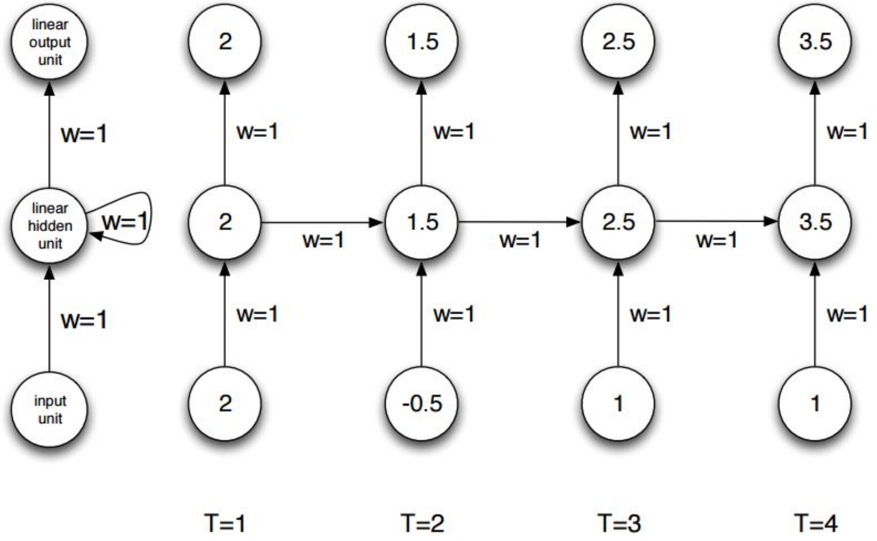


RNN over HMM

- HMM has Limitations
  - Markov Assumption: This means the model is memoryless, i.e. it has no memory of anything before the last few words.
  - At each time step it must select one of its hidden states. So with N hidden states it can only remember log(N) bits about what it generated so far.
- RNNs are very powerful, because they combine two properties:
  - Do not make the Markov assumption and so can, in theory, take into account long-term dependencies when modeling natural language.
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer!

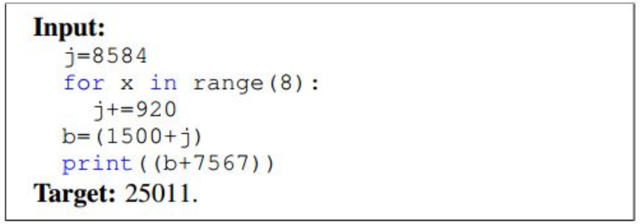
Simple RNN Example

- What does this RNN do?

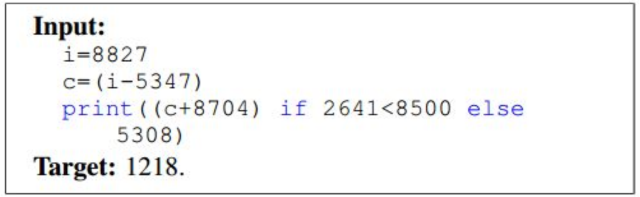


What can RRNs compute?

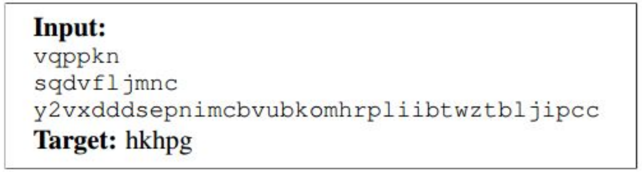
- In 2014, Google researchers built an RNN that learns to execute simple Python programs, one character at a time!



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.

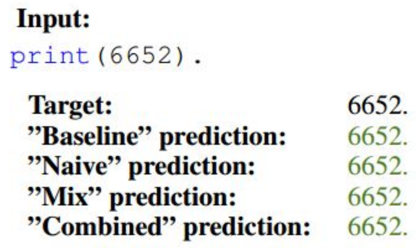


Input:
```
vqppkn
sqdvfljmnc
y2vxdddsepnimcbvubkomhrpliibtwztbljipcc
```
Target: hkhpg

A training input with characters scrambled

What can RRNs compute?

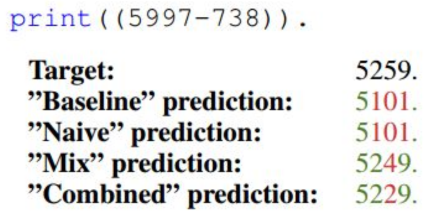
- Some example results:



Input:

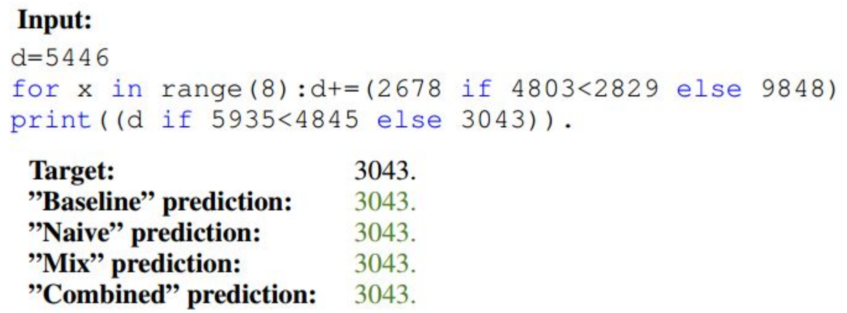
```
print(6652).
```

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



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

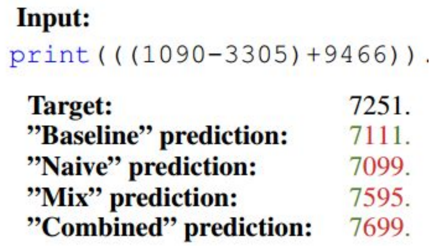
Target:	5259.
"Baseline" prediction:	5101.
"Naive" prediction:	5101.
"Mix" prediction:	5249.
"Combined" prediction:	5229.



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)).
```

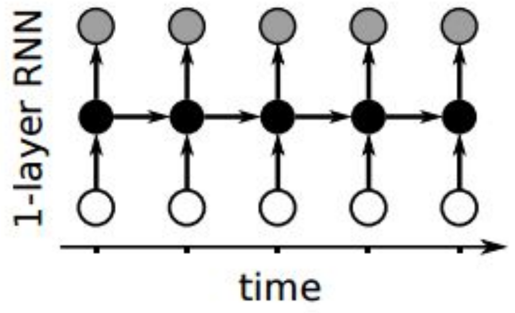
Target:	7251.
"Baseline" prediction:	7111.
"Naive" prediction:	7099.
"Mix" prediction:	7595.
"Combined" prediction:	7699.

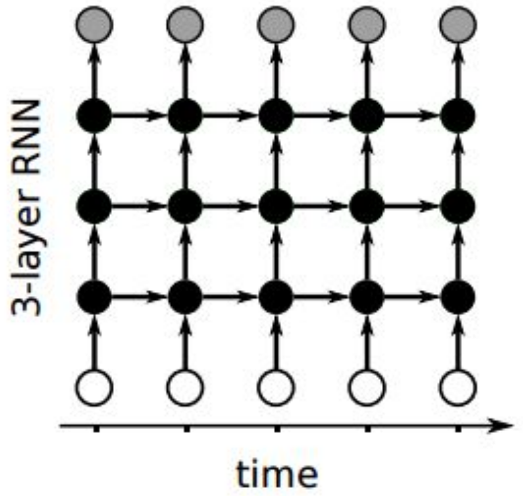
Problems with training RNN

- Method: Backpropagation through time.
- The computational power of RNNs makes them very hard to train.
- Problem of “Exploding and vanishing gradients”
  - Deal this using “Long Short Term Memory” cells.
- The memorization task exemplifies how the issue can arise in RNNs. The network must read and memorize the input sequence, then spit it out again.

Modeling Sequences with RNN

- Want to turn input sequence to output sequence in different domain
  - E. g. turn a sequence of sound pressures into a sequence of word identities.
- Training RNN by trying to predict the next term in the input sequence
  - Target sequence is the input sequence with an advance of 1 step.

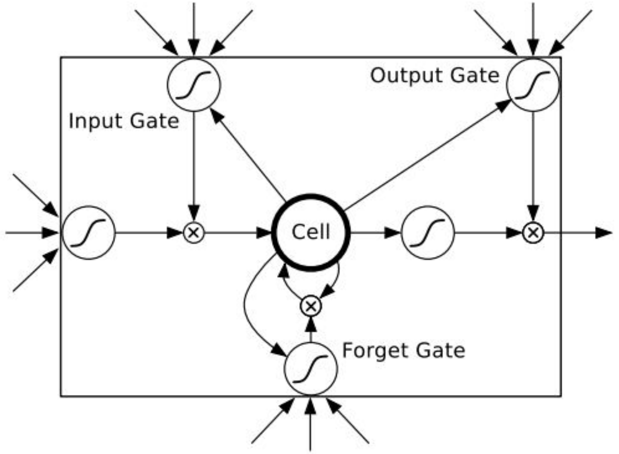




➡ white circles: input
➡ black circles: hidden state
➡ Grey circle: output

Long Short Term Memory

- LSTM is an RNN architecture designed to have a better memory. It uses linear memory cells surrounded by multiplicative gate units to store read, write and reset information.
- Replace each single unit in an RNN by a memory block

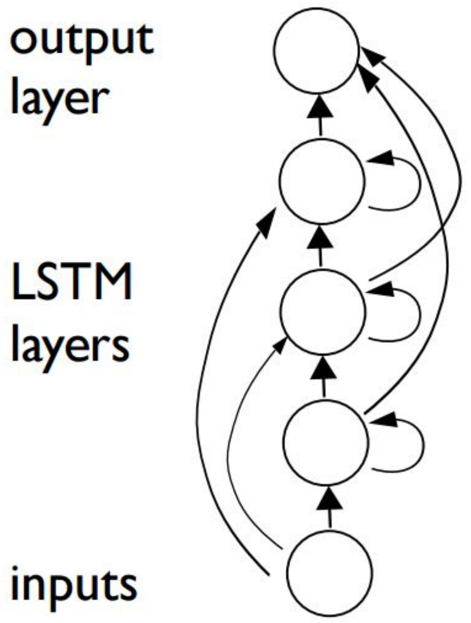


Input gate: scales input to cell (write)

Output gate: scales output from cell (read)

Forget gate: scales old cell value (reset)

General Architecture



- Deep recurrent LSTM net with skip connections
- Inputs arrive one at a time, outputs determine predictive distribution over next input
- Train by minimising log-loss:

$$\sum_{t=1}^{T} - \log \Pr(x_t | x_{1:t-1})$$

- Generate by sampling from output distribution and feeding into input

Practical Demos

1. Text-Generation demo: http://www.cs.toronto.edu/~ilya/fourth.cgi
   a. Trained with words as sequence input elements.
2. Charcater Generation demo: http://cs.stanford.edu/people/karpathy/recurrentjs/
   a. Individual characters as input for training.

References

- Figure: Basic structure of Recurrent Neural Network (taken from M. Hermans and B. Schrauwen (2013,[3]))
- http://www.cs.toronto.edu/~rgrosse/csc321/lec9.pdf
- http://www.cs.toronto.edu/~graves/gen_seq_rnn.pdf

Thank you!