DD2418 Language Engineering
9: Recurrent neural networks

Johan Boye, KTH

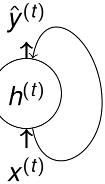
May 7, 2020

Recurrent neural networks (RNNs)

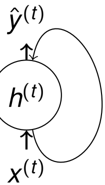
A recurrent neural network (RNN), the network maintains a hidden state, which is updated as a function of the input and the last hidden state.

The output is computed as a function of the hidden state.

input words into hidden state
then output prediction



Recurrent neural networks (RNNs)



The hidden state is updated as a function of the input and last hidden state:

Whh is example 100 x 100 matrix
Wxh is example 50 x 100 matrix

$$h^{(t)} = g(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b^{(t)})$$

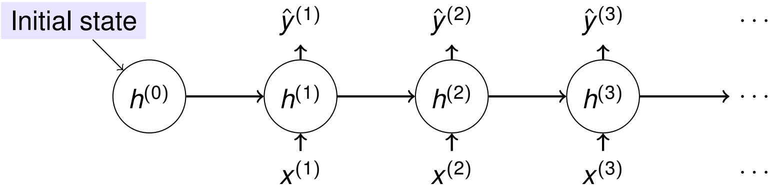
The output is a function of the current state:

$$\hat{y}^{(t)} = f(W_{hy}h^{(t)})$$

All W are trainable

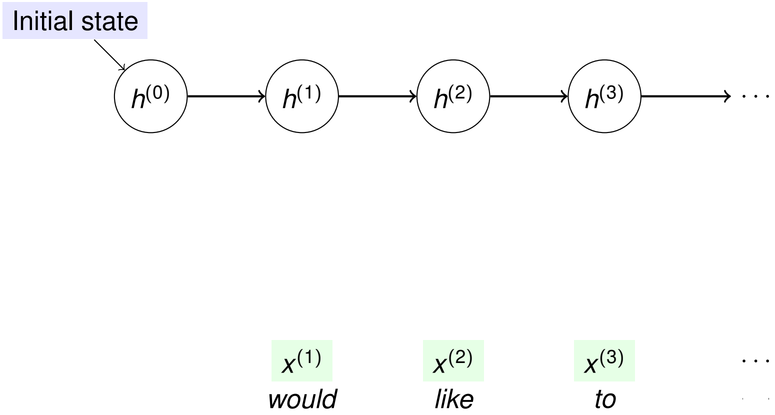
(f and g are non-linear activation functions)

Unrolling RNNs

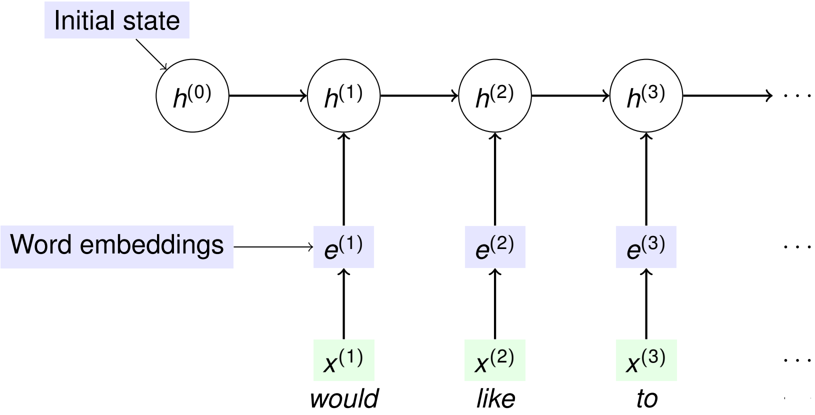


Unrolling = displaying RNNs with every time step separately.

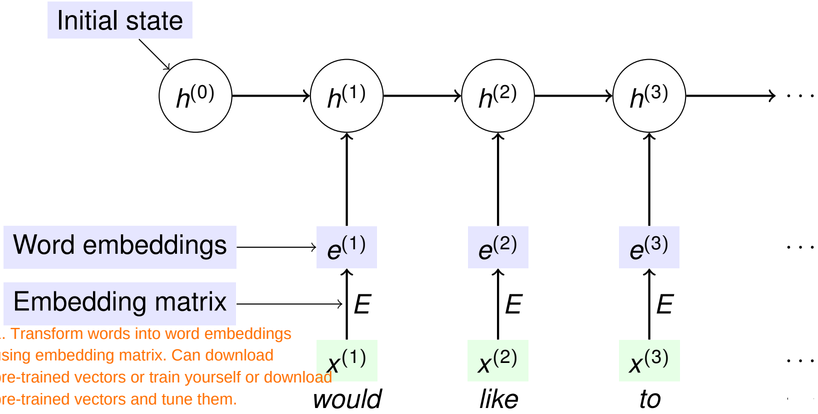
Language RNNs



Language RNNs

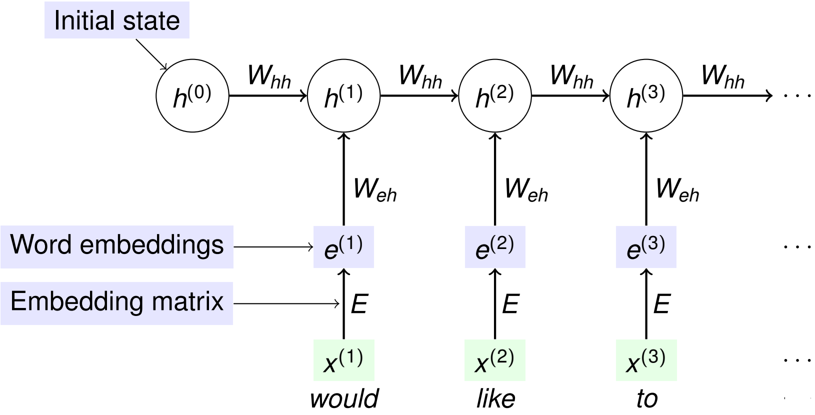


Language RNNs

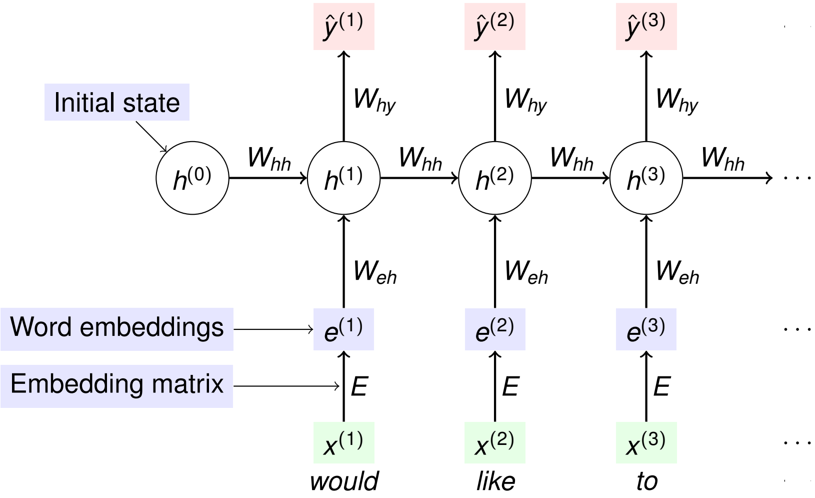


1. Transform words into word embeddings using embedding matrix. Can download pre-trained vectors or train yourself or download pre-trained vectors and tune them.

Language RNNs



Language RNNs



RNNs for language processing

Some nice things:

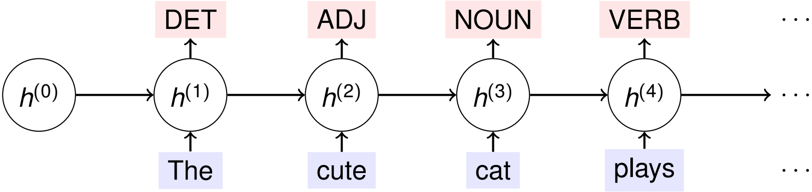
- Can handle arbitrarily long sequences
- Model size is independent on sequence length.
- Can (potentially) remember things from way back.

But:

- Can be slow to train.
- Non-parallelizable.
- Vanilla RNNs don't actually remember long-term dependencies so well (but there are extensions that do).
  Need GRU or LSTM.

Labeling words in a sequence

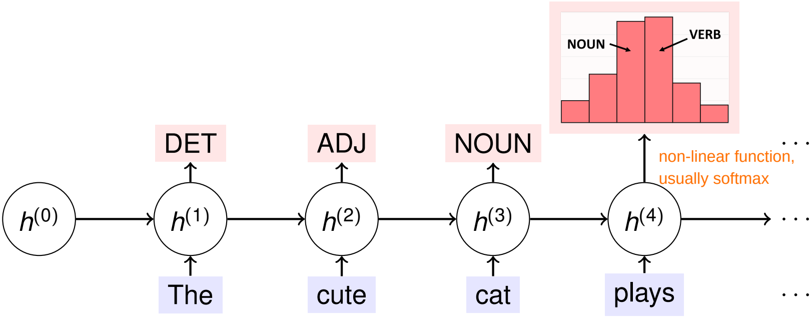
RNNs can be used for sequence labeling tasks, like POS tagging.



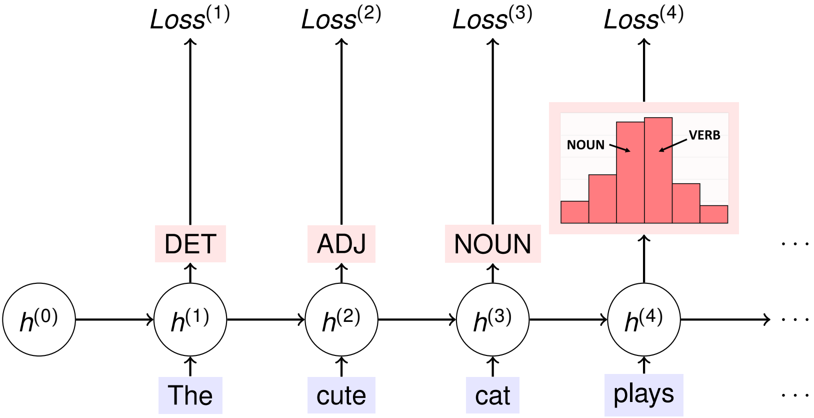
One prediction per input

Labeling words in a sequence

RNNs can be used for sequence labeling tasks, like POS tagging.

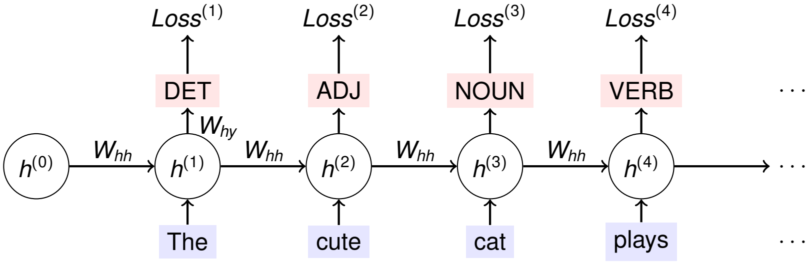


Training



$Loss^{(1)}$
$Loss^{(2)}$
$Loss^{(3)}$
$Loss^{(4)}$
NOUN
VERB
DET
ADJ
NOUN
...
$h^{(0)}$
$h^{(1)}$
$h^{(2)}$
$h^{(3)}$
$h^{(4)}$
...
The
cute
cat
plays
...

Training

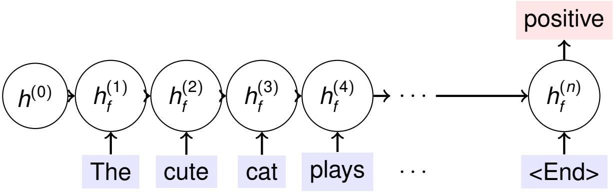


Need to compute the gradient of the loss in each timestep w.r.t. all trainable parameters, e.g. $\frac{\partial Loss^{(i)}}{\partial W_{hh}}$.

The backpropagation-through-time (BPTT) algorithm solves this problem.

Sequence classification

RNNs can be used to classify the entire sequence, e.g. sentiment analysis.

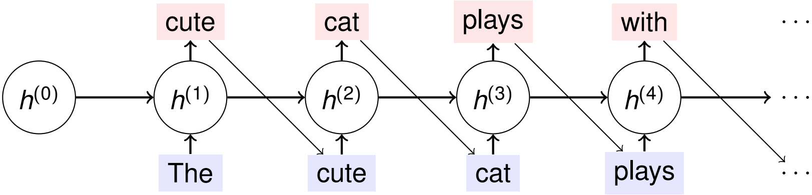


We ignore all outputs $\hat{y}^{(t)}$ except the last $\hat{y}^{(n)}$, which is considered to be the result.

Language models

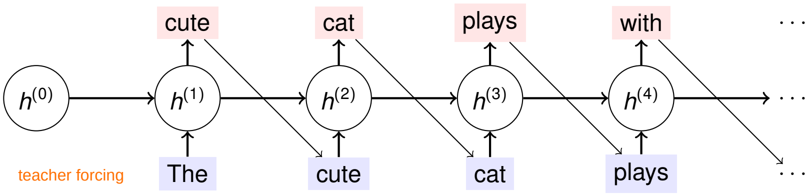
Whereas Feed-forward Neural Network is discriminative in nature, RNN is generative

RNNs can be used to predict/generate the next word.



Language models

RNNs can be used to predict/generate the next word.



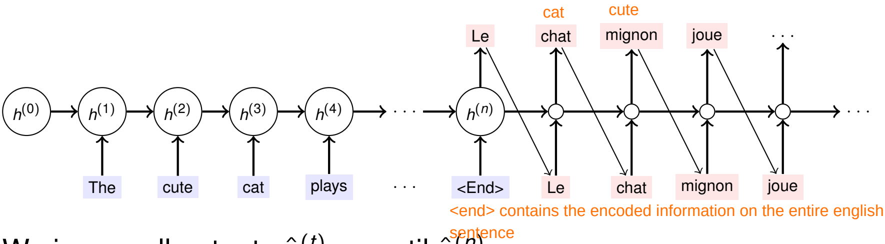
This is an autoregressive model: it takes its own previous output into account when producing the next output.

Training can be done by inputting the produced output or the correct output of the preceding time step.

The latter is called teacher forcing.

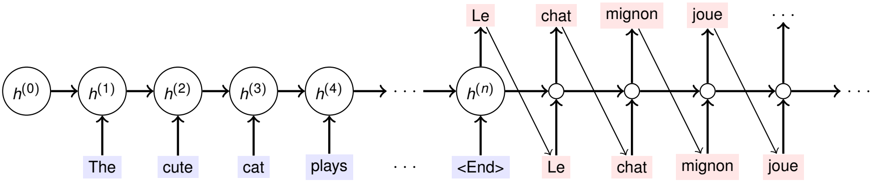
If just do teacher forcing, it will never learn to react to its own output. It will always be expected to have the correct input. So it is a good idea to use both produced output and correct output.

Translation



We ignore all outputs $\hat{y}^{(t)}$ up until $\hat{y}^{(n)}$.

Translation



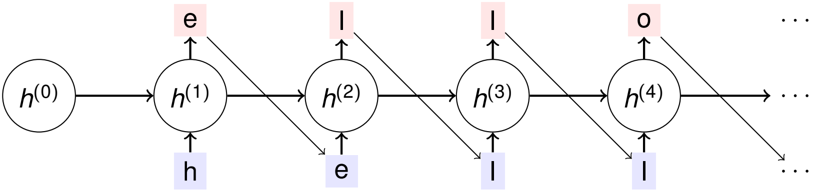
This is an example of an encoder-decoder architecture.

The hidden state $h^{(n)}$ (hopefully) encodes all necessary information about the source sentence, which can then be decoded into the target sentence.

In practice, this scheme needs to be extended (using attention information).

Language generation using an RNN

Andrej Karpathy showed in an article from 2015 how character level RNNs can be used for language generation.



Language generation using an RNN

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.
```

A. Karpathy: *The unreasonable effectiveness of Recurrent Neural Networks*, blog post

Language generation using an RNN

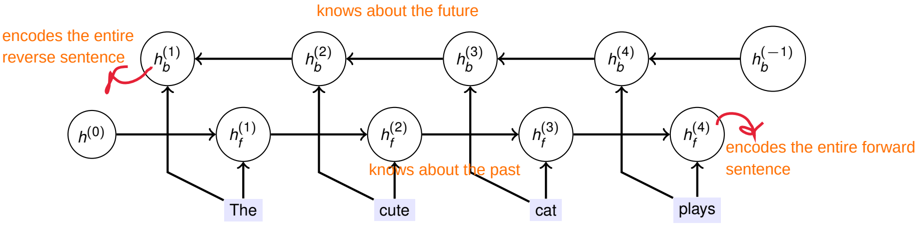
```
/*
 * If this error is set, we will need anything right after th
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_C
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
```

A. Karpathy: *The unreasonable effectiveness of Recurrent Neural Networks*, blog post

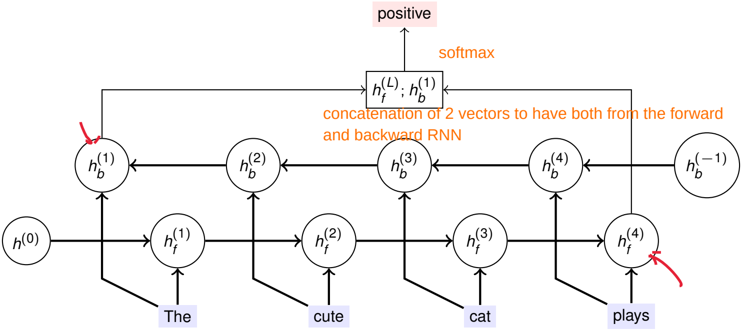
Bi-directional RNNs

Often it is useful to get information both from the left and the right context.



Bi-directional RNNs

Will be required in Assignment 4



softmax

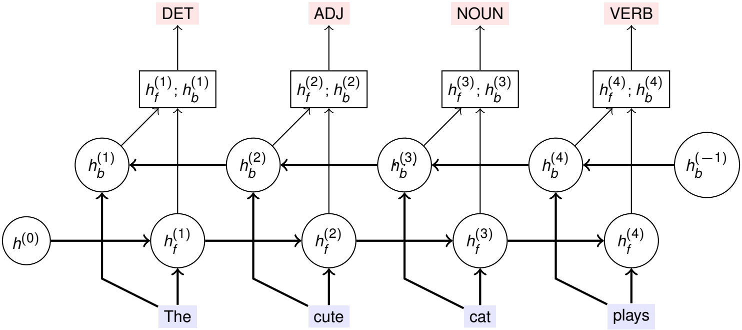
concatenation of 2 vectors to have both from the forward and backward RNN

If interested in sentiment analysis, only look at the 2 final hidden states of the 2 RNNs.

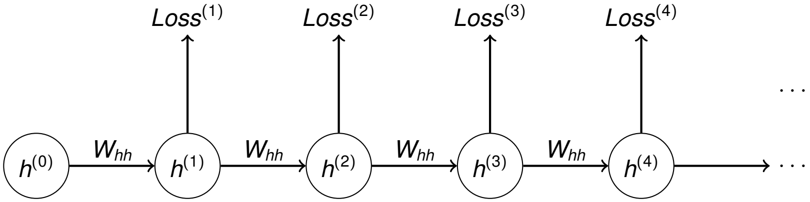
Bi-directional RNNs

POS tagging

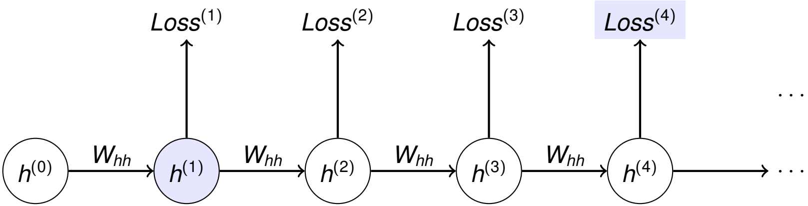
An idea we will also be using in Assignment 4



Vanishing gradient problem

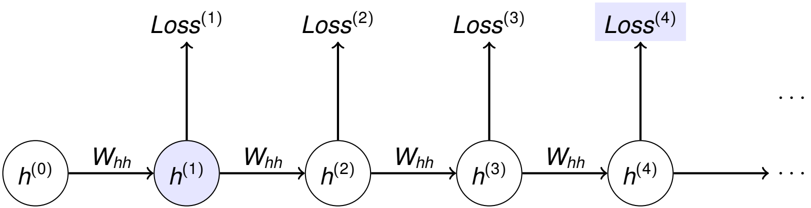


Vanishing gradient problem



$$\frac{\partial Loss^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial Loss^{(4)}}{\partial h^{(4)}}$$

Vanishing gradient problem



$$\frac{\partial Loss^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial Loss^{(4)}}{\partial h^{(4)}}$$

If all intermediate gradients are small, then the gradient signal from $Loss^{(4)}$ on $h^{(1)}$ is going to be very small...

... but the gradient signal from $Loss^{(2)}$ is going to be stronger ...

... so the RNN will have problems learning long-distance relationships!

Managing context in RNNs

To make RNNs better capture long-distance dependencies, we can add so-called gates that better control the flow of information.

Two important suggestions:

- Long Short-Term Memory (LSTM) networks (Schmidthuber and Hochreiter 1997)
- Gated Recurrent Units (GRU) networks (Cho et al. 2014)

LSTMs are more powerful, but GRUs are quicker to train and simpler to understand and implement.

Gated Recurrent Units (GRUs)

The update of the hidden states are controlled by two gates:

- the reset gate r controls what part of the previous hidden state is relevant for the current situation
- the update gate z decides what of the old previous hidden state should be retained, and what part should be updated

$$r^{(t)} = \sigma(U_r h^{(t-1)} + W_r x^{(t)})$$
$$z^{(t)} = \sigma(U_z h^{(t-1)} + W_z x^{(t)})$$

The tentative new hidden state:

$$\widetilde{h}^{(t)} = \tanh(U_h(r^{(t)} \odot h^{(t-1)}) + W_h x^{(t)})$$

The new hidden state:

$$h^{(t)} = (1 - z^{(t)})h^{(t-1)} + z^{(t)}\widetilde{h}^{(t)}$$

Hidden states are outputs from the GRU.

GRU networks

Also need this for assignment 4

Gated Recurrent Units can be then be used in RNNs instead of (just) hidden states.