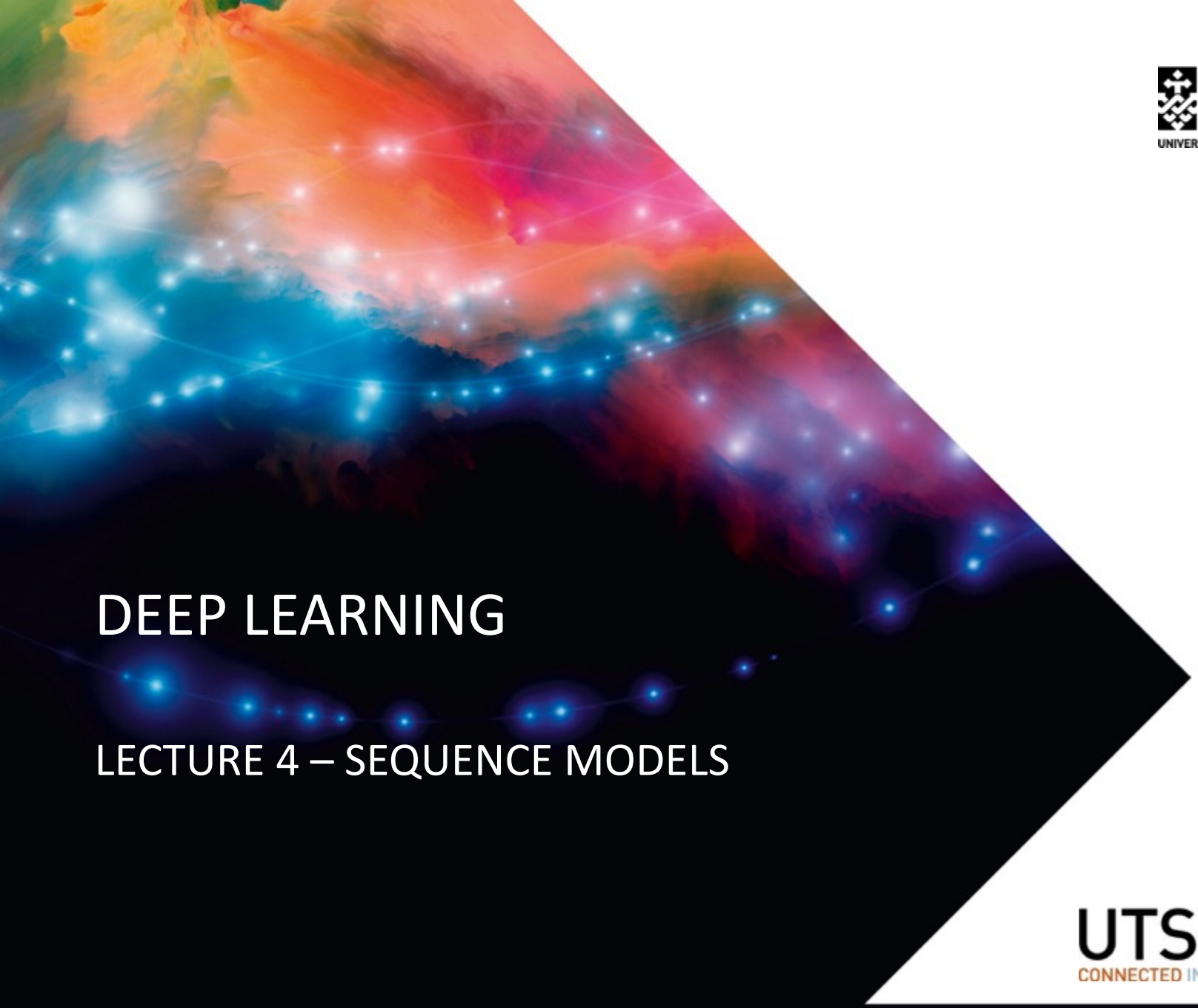


UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

DEEP LEARNING

LECTURE 4 – SEQUENCE MODELS

UTS:CIC
CONNECTED INTELLIGENCE CENTRE

OVERVIEW OF TODAY

Schedule:

> AT1B Check in

> Intro to Sequence Models

> RNN/LSTMs

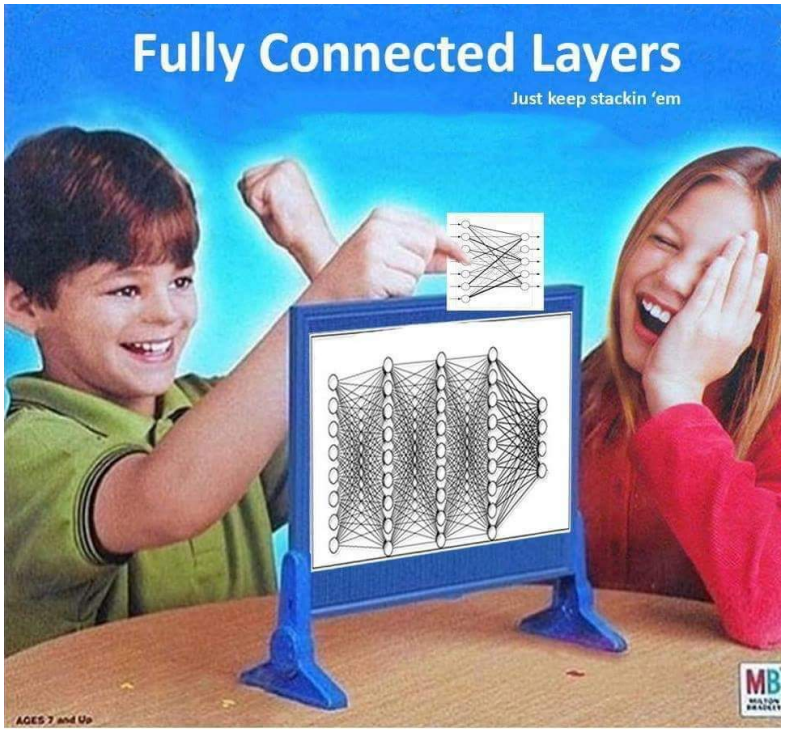
> Attention

Lab/Lecture:

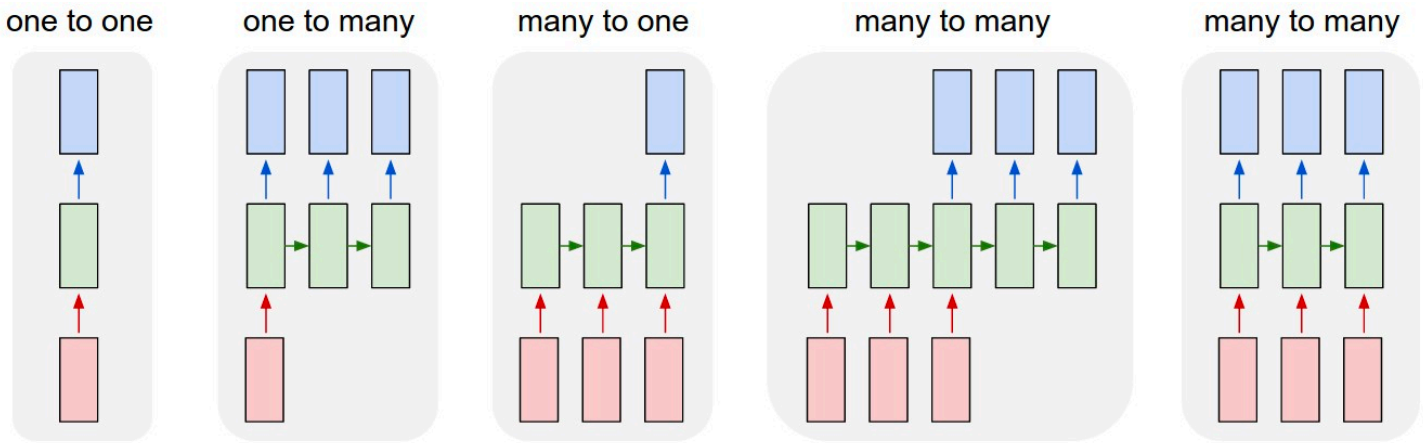
> NLP & Deep Learning in industry: Steve Nouri, Head of A.I at Nod.

AT1B CHECK IN

- How is it all going?



INTRODUCTION TO SEQUENCE MODELS



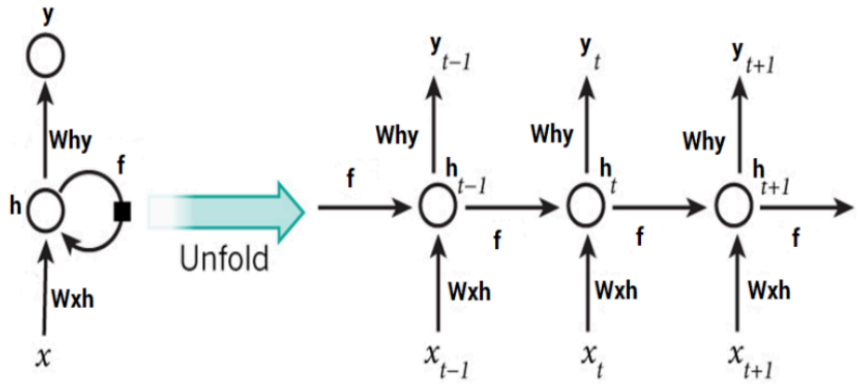
Source

- We have so far been working with one-to-one, even with out images.

There is also:

- One-to-many (Image captioning)
- Many to one (Sentiment analysis on a sentence)
- Many to many with variable length I/O (Language translation)
- Many to many with variable input but single classification per input (video frame classification)

INTRODUCING RNNS

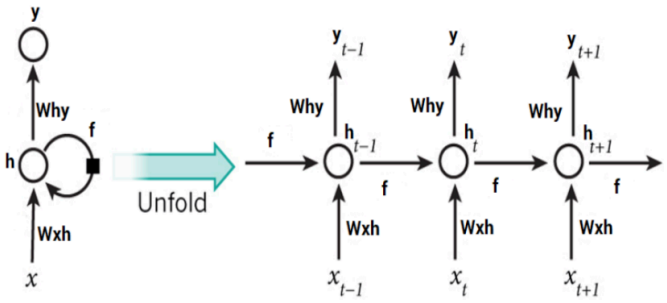


source

- An RNN is a network architecture that can deal with this.
  - Originally proposed in 'that' backprop paper (Rumelhart et al, 1986)
- A vanialla RNN is seen in compact (left) and unrolled (right) diagrams. Both represent the same thing.
  - The neuron has a hidden state
  - The hidden state is updated when an input vector X at every time point is added.
  - There are two weight matrices, one on input and one on output

INTRODUCING RNNS – THE MATHS

- We can represent the hidden state at any given time point as:
  - $h_t = f_W(h_{t-1}, x_t)$
    - Some **function** with parameters (weights **w**)
    - Is applied to the **previous hidden state** AND the **input vector** at that time
    - A **new state** is produced
- Note we use the **same** weights and function at each time step

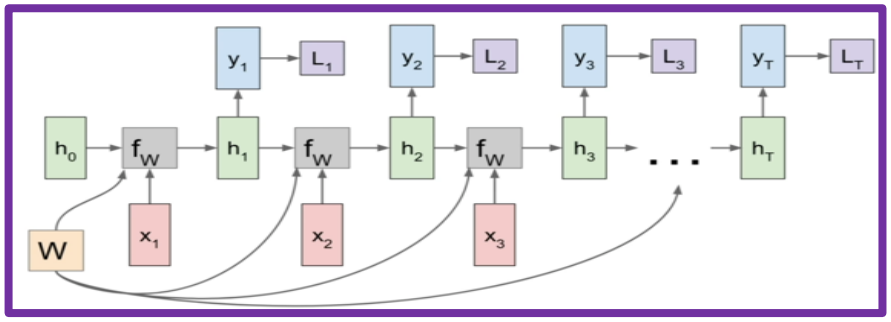


Typically H_0 initalised to zeros

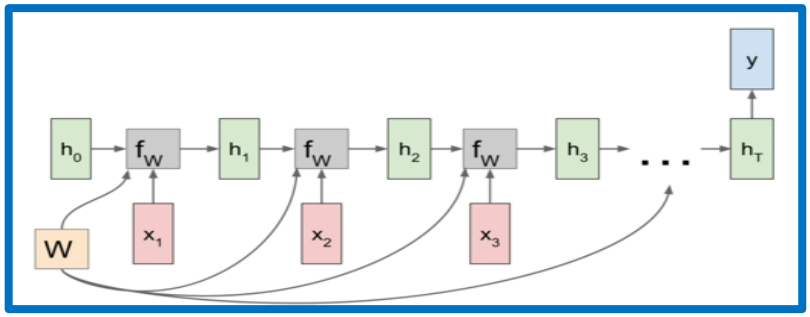
- Which might look something like this (tanhs used in earlier papers):
  - $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
- You may also want a prediction at each timestep (seen above), so you need another weight matrix (W_{hy}) to multiply this new hidden state to create class scores.

RNN COMPUTATIONAL GRAPH

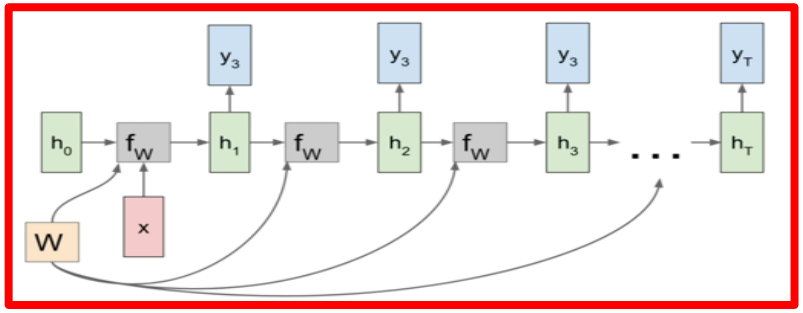
- We can represent the different RNN types as computational graphs [Source]



A **Many to Many** RNN computational graph. The multiple losses mean a final loss summation to get a total loss as well as an additive operation to sum gradients into the final weight matrix during back prop.



A **Many to One** RNN computational graph. There is only one output, but it has elements of everything that has come before it. Hence only one loss



A **One to Many** RNN computational graph. Similar to Many to Many but only one input X.

A FUN RNN EXAMPLE

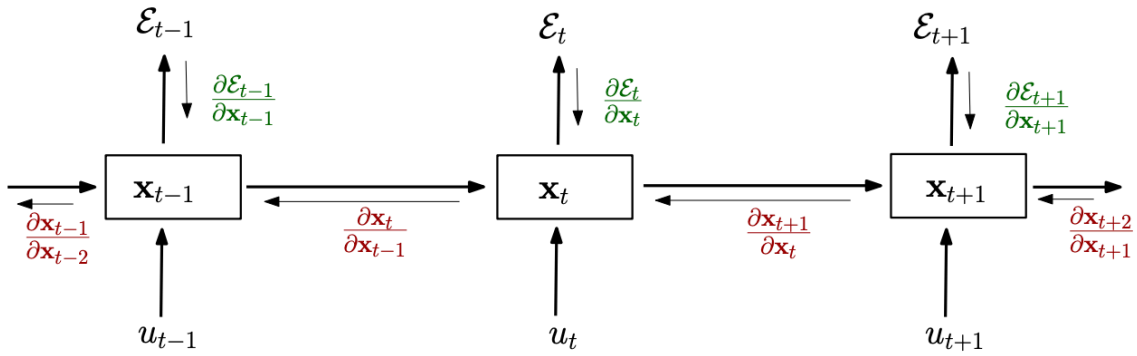
- An infamous paper/example by Andrej Karpathy – 'The unreasonable Effectiveness of RNNs' (source)
  - A character-level RNN to generate text.
  - *"To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file. We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer. After we train the network for a few hours we obtain samples such as:"*

```
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.

VIOLA:
I'll drink it.
```

TRAINING RNNS

- There is little new knowledge required here, we forward prop as noted and backprop using optimisers we know about.

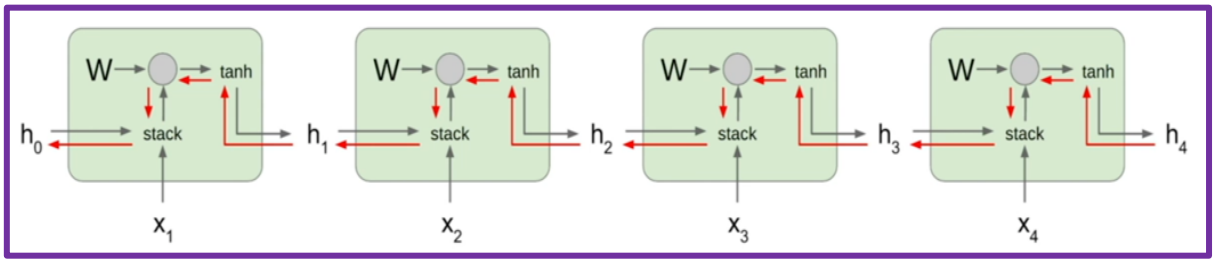
- The only twist we need to know about is *'Backpropagation Through Time'* (source) where we need to do a summation of gradients after unrolling the RNN.



source

TRAINING RNN ISSUES

- The training becomes computationally expensive (and slow!) as the number of time positions increases.
- Additionally there are **exploding gradients** and **vanishing gradients** (Bengio et al, 1994. Here). Over a long enough time period:
  - If the weight matrix is >1, the gradients explode
  - If the weight matrix is <1, the gradients vanish
- See below, the backwards (red) path, we will multiply by transpose of weight matrix for each RNN unit. Over long time periods, this becomes an issue.



source

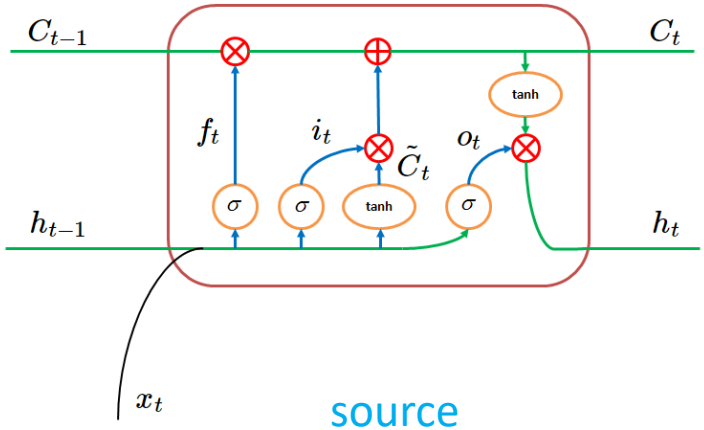
TRAINING RNN ISSUES

- We can fix this with:
  - Gradient clipping & constraints for exploding/vanishing (Source, Bengio et al 2012)
    - Truncated back prop through time (only going back or forward a certain amount)
  - Batch norm difficult (but ongoing research area) due to time component in minibatch
    - "the summed inputs to the recurrent neurons in a recurrent neural network (RNN) often vary with the length of the sequence so applying batch normalization to RNNs appears to require different statistics for different time-steps" (Source, Hinton et al 2016)
  - Change the architecture…..
    - See also: GRU (Cho et al, 2014 source)

INTRODUCING LSTM

- Long Short Term Memory (Hochreiter & Schmidhuber, 1997 source) proposes a solution to the RNN gradient vanishing problem by a slightly more complex architecture.

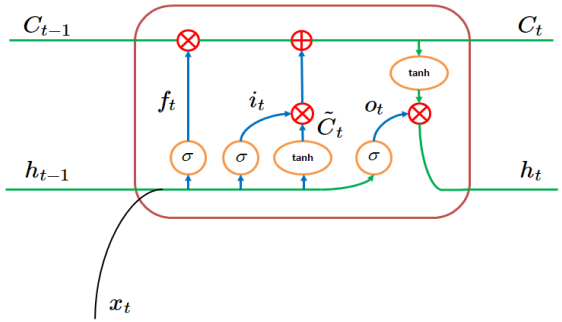
- Let's look at this new architecture:
  - C_{t-1} as the long term memory state
  - h_{t-1} as the short term memory state
  - f_t is the 'forget gate'
    - This tells you how much history you want to keep and how much to 'forget'
    - $f_t = \sigma(W_{xf}^T * x_t + W_{hf}^T * h_{t-1} + b_f)$
  - i_t is the 'input gate'
    - $i_t = \sigma(W_{xCi}^T * x_t + W_{hif}^T * h_{t-1} + b_i)$
    - How much to information to update



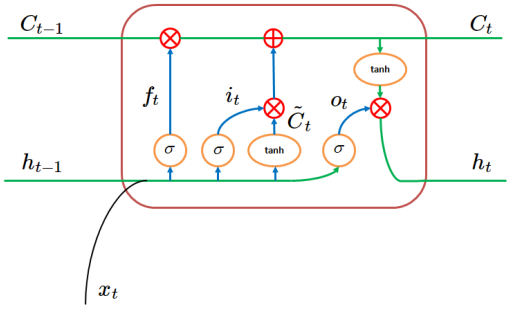
source

INTRODUCING LSTM

- f_t is the 'forget gate'
  - This tells you how much history you want to keep and how much to 'forget'
  - $f_t = \sigma(W_{xf}^T * x_t + W_{hf}^T * h_{t-1} + b_f)$
- i_t is the 'input gate'
  - $i_t = \sigma(W_{xi}^T * x_t + W_{hi}^T * h_{t-1} + b_i)$
  - How much to information to update
- $\tilde{C}_t$ is the intermediary memory state, letting through memory
  - $\tilde{C}_t = tanh(W_{xC}^T * x_t + W_{hC}^T * h_{t-1} + b_C)$
- o_t is the 'output gate'
  - What needs to be given as output
  - $o_t = \sigma(W_{xo}^T * x_t + W_{ho}^T * h_{t-1} + b_o)$
- So the final output is:
  - C_t (Cell State or long term) = $f_t \otimes C_{t-1} \oplus i_t \otimes \tilde{C}_t$
  - h_t (Hidden State or short term) $= tanh(C_t) \otimes o_t$



LSTM IN SUMMARY



- f_t is the 'forget gate' $f_t = \sigma(W_{xf}^T * x_t + W_{hf}^T * h_{t-1} + b_f)$
- i_t is the 'input gate' $i_t = \sigma(W_{xi}^T * x_t + W_{hi}^T * h_{t-1} + b_i)$
- $\tilde{C}_t$ is the 'intermediate memory' $= \tilde{C}_t = tanh(W_{xC}^T * x_t + W_{hC}^T * h_{t-1} + b_C)$
- o_t is the 'output gate' = $o_t = \sigma(W_{xo}^T * x_t + W_{ho}^T * h_{t-1} + b_o)$
- So the final output is:
  - C_t (long term) = $f_t \otimes C_{t-1} \oplus i_t \otimes \tilde{C}_t$
  - h_t (short term) $= tanh(C_t) \otimes o_t$

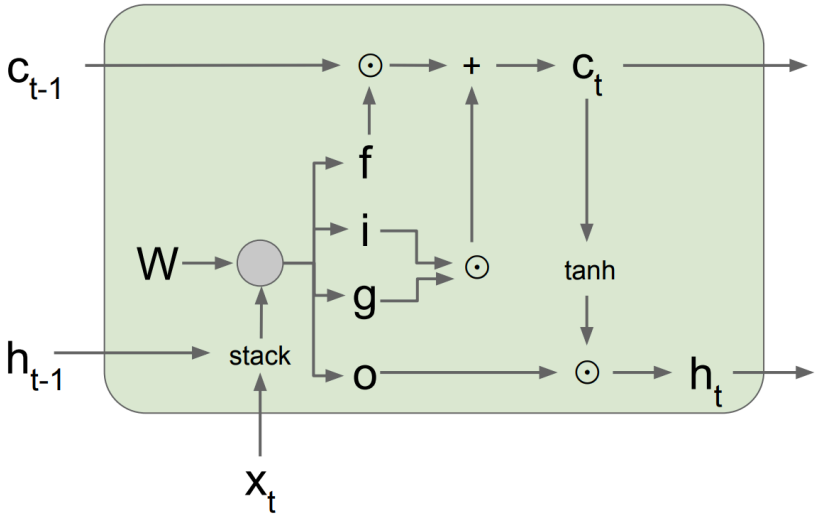
The forget gate is multiplied element wise by the intermediate cell state. Hence at extreme values (0's and 1's) it can forget which cells to forget.

Similar logic applies to input and intermediate memory gate.

AN ALTERNATE LSTM DIAGRAM

source

- We can see in the previous formulae that there are a lot of similar operations happening (Weight matrices multiplying by the same things) and then activations over that.
  - In fact this would be stacked for computation, so we can view a more compact version.
  - Note: In this version, the intermediary memory is called the 'gate gate' (….)



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

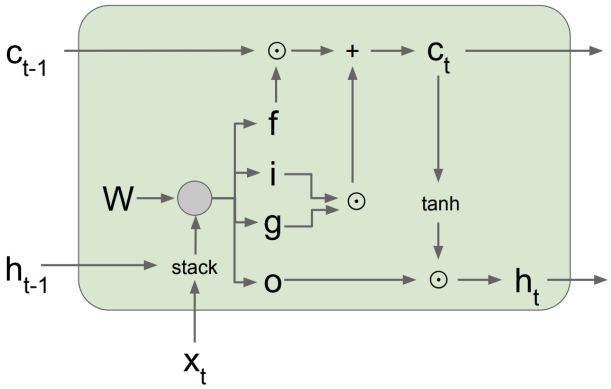
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

AN ALTERNATE LSTM DIAGRAM

source

- Let's walk through the steps:
  1. Take previous hidden state h_{t-1} and cell state C_{t-1}
  2. Stack the previous hidden state h_{t-1} and the input **X**
  3. Multiply by a large Weight matrix **W**
     > This is the 4 we had previously stacked together.
  4. The **forget gate** multiplies element wise with the previous cell state C_{t-1}
  5. **I** and **G** gates element-wise multiply
  6. Output of (5) added to output of (4) to create new cell state C_t
  7. Output of (6) is squashed through tanh
  8. Output of (7) element-wise multiplied with **O** gate to create new hidden state h_t



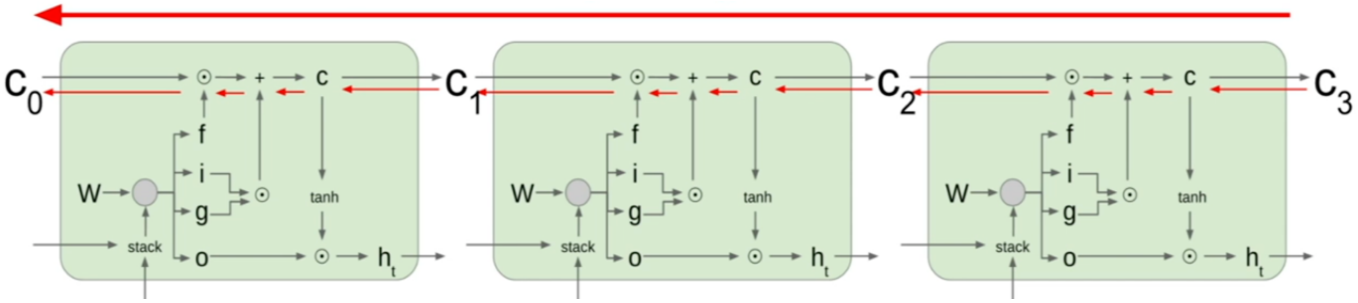
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

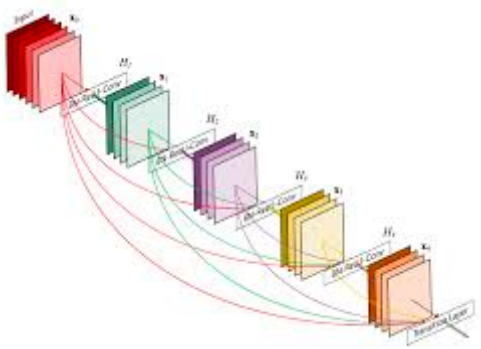
$$h_t = o \odot \tanh(c_t)$$

WHY DOES THIS WORK? source

- We have essentially created a gradient highway that can bypass the weight matrix multiplications in some levels:
  - There are interesting parallels between this and the resnet approach.



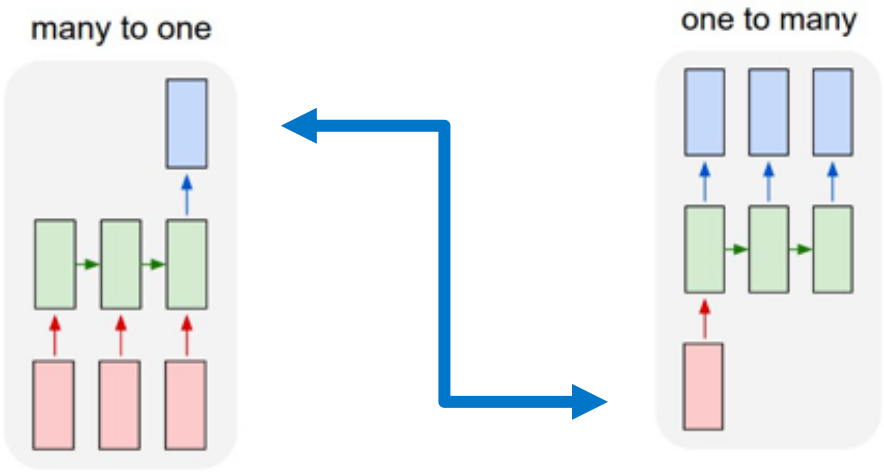
- Something else that helps is 'Peephole' connections (Source Gers et al 2002)



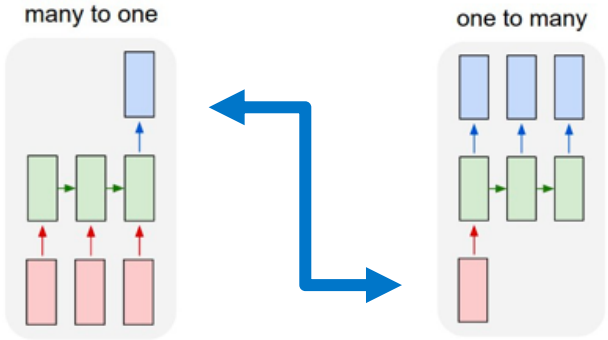
  - This allows the long term memory of the previous time to be an input to the controller gates (forget, input, intermediary) and hence there are more connections for the network to fetch information it may need.

SEQ2SEQ

- A successful variant of sequence models has been ‘seq to seq’ which is used for image captioning, nlp, text models. (Github, google)
  - Seq2Seq (source, Sutskever et al 2014) (nips paper)
    - Similar to Cho et al, 2014 (Source)
  - Seq2Vec (Source, Kimothi et al, 2016) – interesting biological application.
- It constructs two steps we have seen before.
  - An **encoder** that is a many to one
  - A **decoder** that is a one to many.



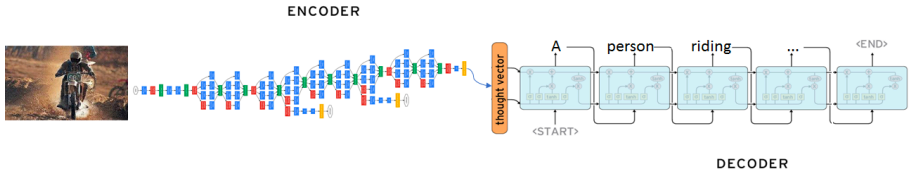
SEQ2SEQ + ATTENTION



- (Source, 2014. Bahdanau, Cho & Bengio)
- The problem with Seq2Seq is this idea of encoding a fixed length vector as the output.
  - *the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases*
- *In this paper, we conjecture that the **use of a fixed-length vector is a bottleneck** in improving the performance of this basic encoder-decoder architecture, and propose to extend this by **allowing a model to automatically (soft-)search** for parts of a source sentence that are relevant to predicting a target word….. Instead, [this network] encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation.*

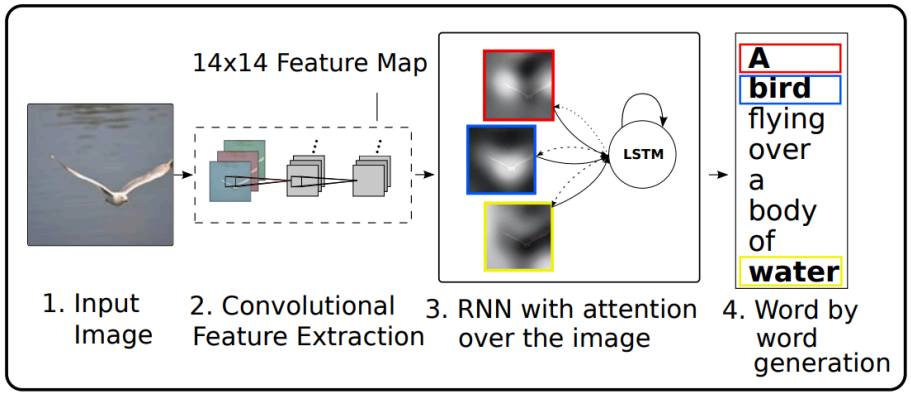
SEQ2SEQ + ATTENTION + IMAGE CAPTIONING

- Attention has also been very important in assisting with image captioning tasks.
  - (Nice video), (Nice article)



source

- Attention helped to find the areas of interest when undertaking image captioning (Xu, Cho, Bengio et al 2015 Source):
  - "Inspired by recent work in machine translation and object detection, we introduce an attention based model that automatically learns to describe the content of images"



ATTENTION + IMAGE CAPTIONING EXAMPLES

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

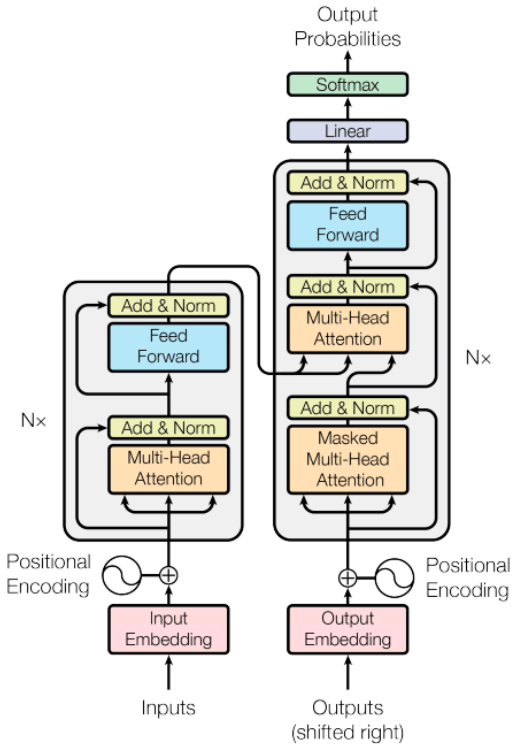
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

EXTENSION: THE TRANSFORMER UNIT

- (Source, 2017)
- Moving forward from encode-decoder networks that rely on an RNN or LSTM, this important paper introduced the ‘Transformer’ network.
- Without going into too much detail, the key innovation was to remove the RNN/LSTM component completely and purely use attention-based mechanisms.



GUEST LECTURER: STEVE NOURI FROM NOD