

Lecture 14. Recurrent Neural Networks, Attention, and the Transformer

COMP90051 Statistical Machine Learning

Christine de Kock



THE UNIVERSITY OF
MELBOURNE

This lecture

- Recurrent networks for modelling sequences
 - * recurrent units
 - * back-propagation through time
 - * long-short term memory
- Transformers and attention

Recurrent Networks

*A DNN tailored to variable length
sequential inputs*

Sequential input

- Until now, we have assumed fixed-sized input
 - * Vectors of features \mathbf{x} in d dimensions
 - * Matrices of pixels in an image
- What if our input is a **sequence**?
 - * Frames in a video clip
 - * Time steps in an audio clip
 - * Words in a sentence
 - * A protein sequence
 - * Stock prices over time ...
- How can we model this in a DNN?

FCNNs are poor for sequences

fully connected NN (vanilla NN)

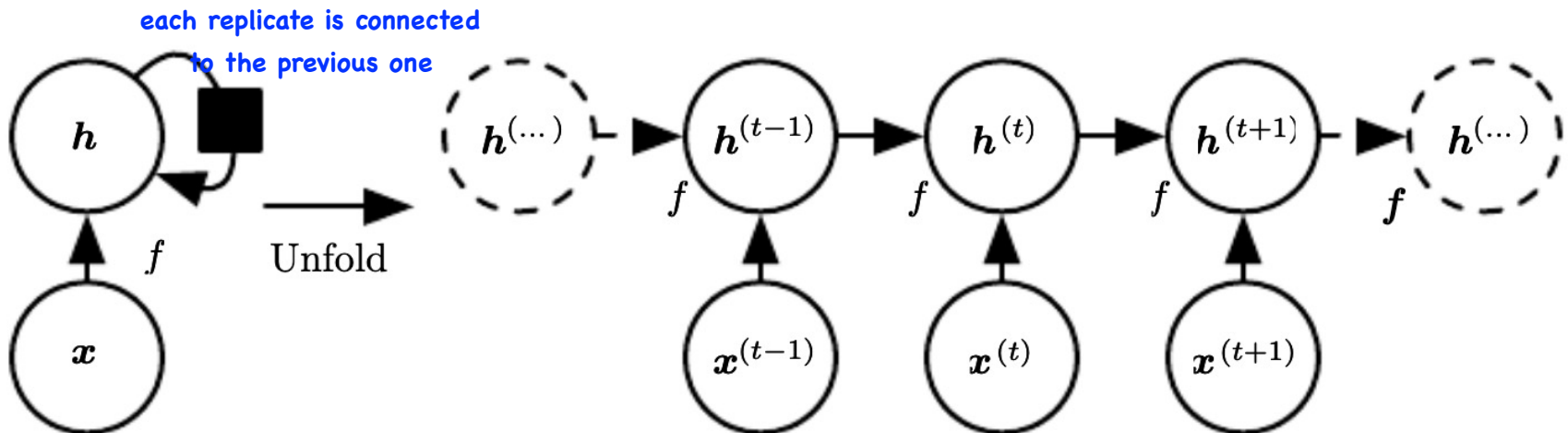
- Consider classifying sentences
 - * “This is the worst movie of all time, a real stinker” → 😞
 - * “The movie is a real stinker” → 😞
- Issue: inputs are *different lengths*
 - * **pad** them with empty “words” to be a fixed size
- Issue: how do we *represent words* as vectors?
 - * learn an “**embedding**” vector for each word
- Issue: phrases have *similar meaning even when at different locations*
 - * “a real stinker” is a key predictive feature
 - * if we naively apply FCNN needs to learn this concept repeatedly

ConvNets for Sequences?

- Sequences are just rectangular shaped images (e.g., embedding dim. times length): apply CNNs
 - * With **1D filters**
 - * The filter parameters are shared across time, and can find patterns in the input
- This is called the *time delay neural network*
- Downside:
 - * receptive field of filters are limited to finite size, i.e., the width of the convolutional filters, which can be expanded with deeper networks

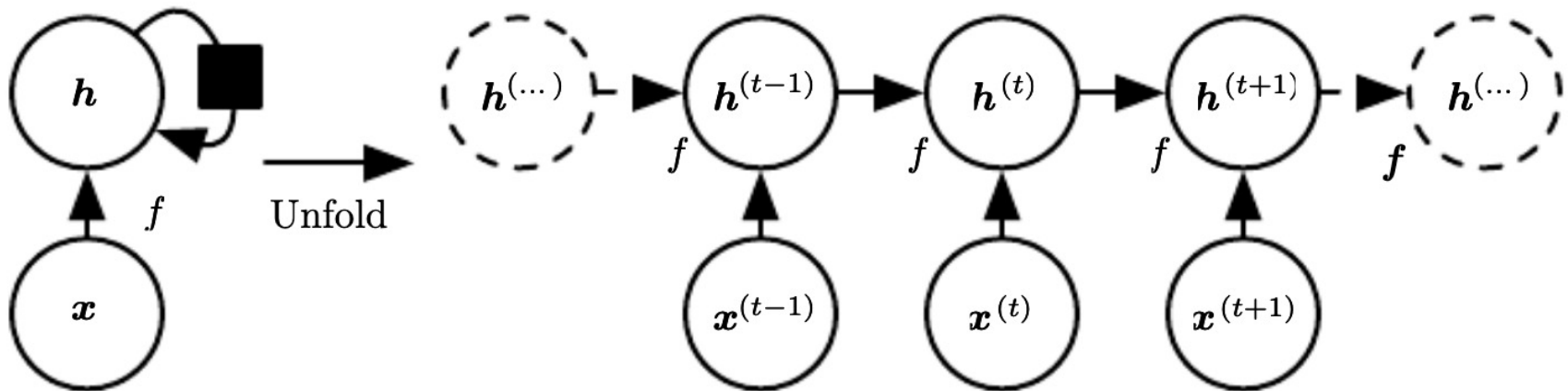
Recurrent Neural Nets (RNNs)

- RNNs create networks dynamically, based on input sequence
 - * given sequence of inputs $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$
 - * process each symbol from left to right, to form a sequence of hidden states $\mathbf{h}^{(t)}$
 - * each $\mathbf{h}^{(t)}$ encodes all inputs up to t



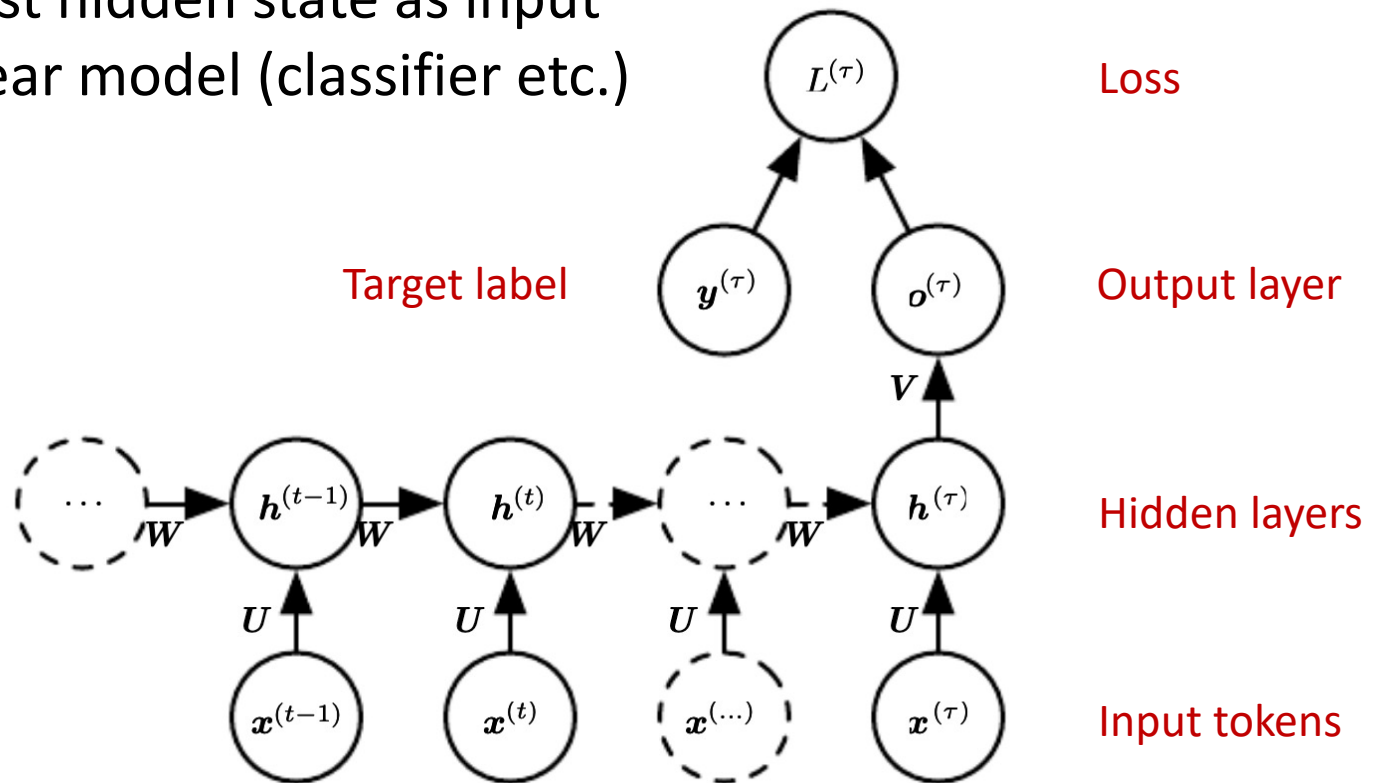
RNNs as Very Deep Networks

- Compared to NNets we've seen before:
 - * unfolded RNN has **depth equal to input sequence length**
 - * **parameters shared** between layers
- Can easily be 'unrolled' to cater to any input length



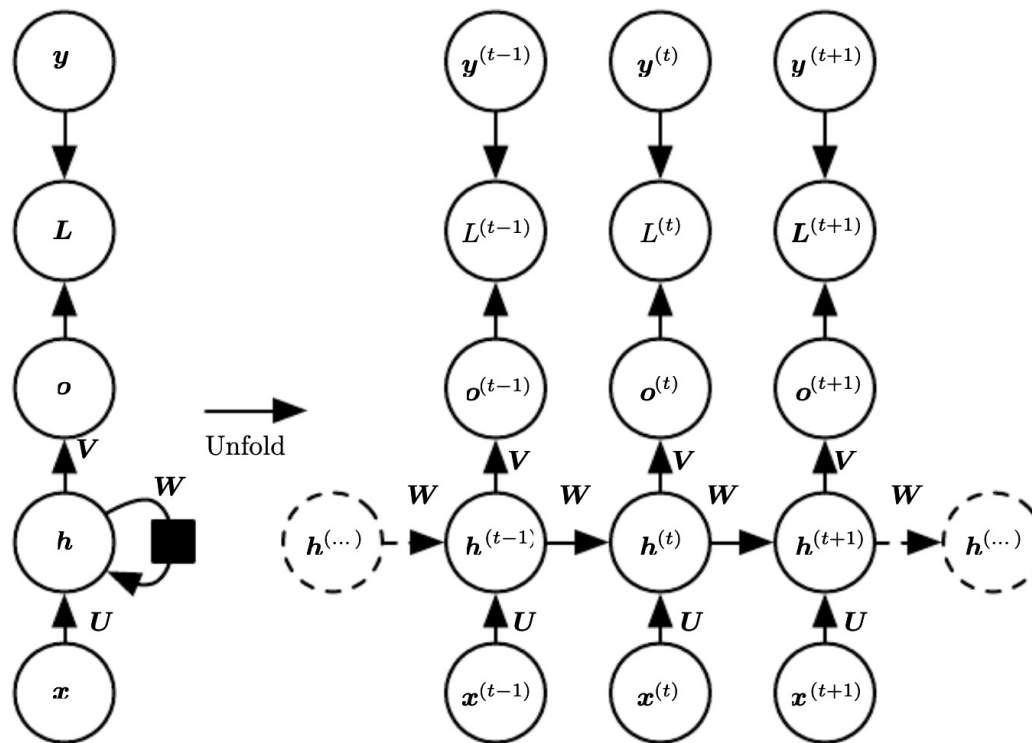
RNN Applications: Seq. Classification

- Sequence classification: labelling sequence
 - * use last hidden state as input to linear model (classifier etc.)

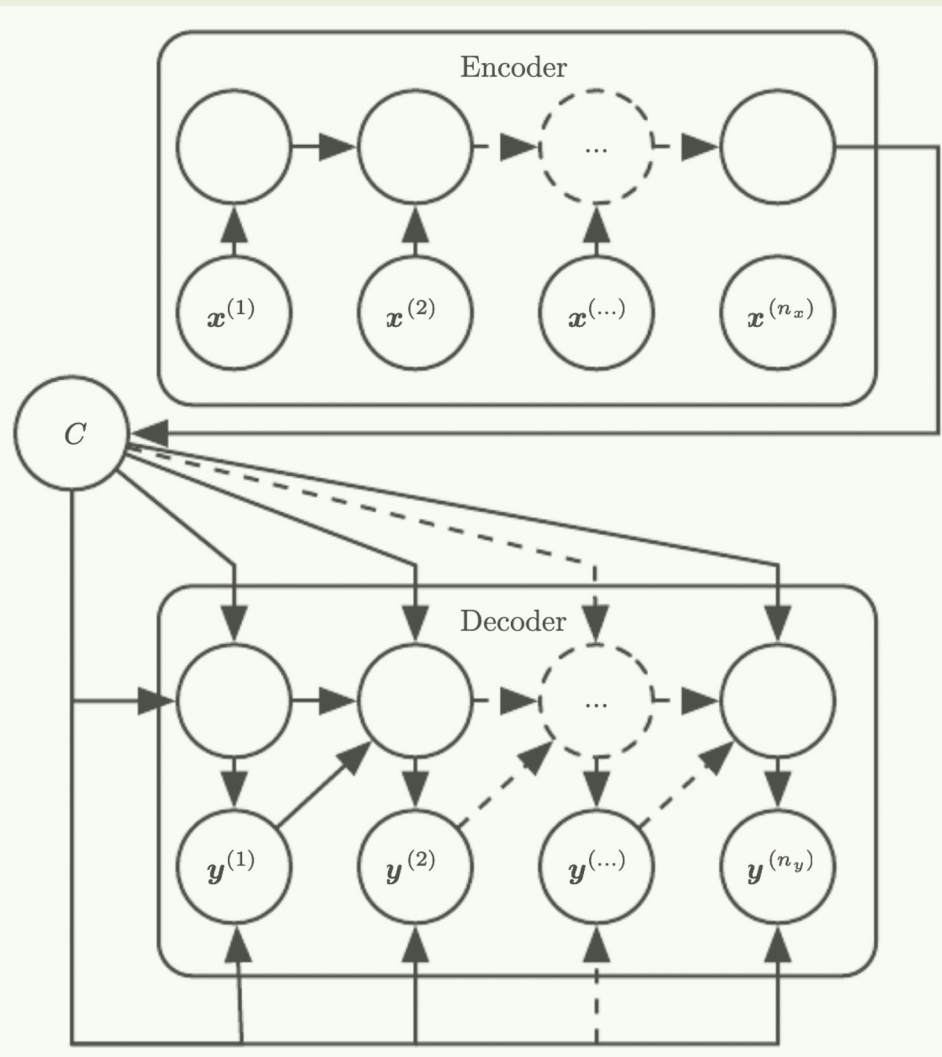


Sequence Tagging RNN

- Assign each item/token a label in sequence
 - * Given targets per item, can measure loss per item



Encoder-Decoder for Sequence Translation



E.g., English to French

Encoder RNN encodes input sequence into a context

Decoder RNN acts like a tagger, where we're trying to (re)generate next inputs

RNN Parameterisation

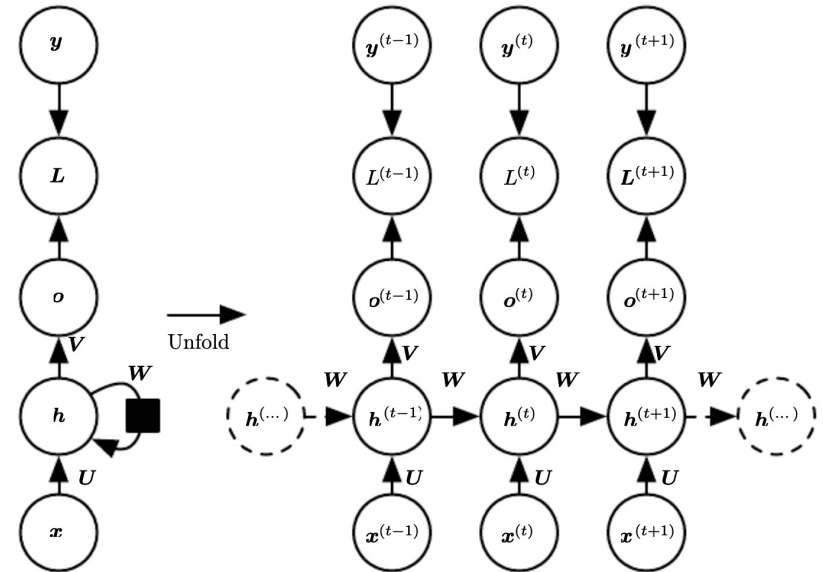
- Consider tagging RNN:
define f as follows

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}),$$



- Parameters are $\mathbf{b}, \mathbf{W}, \mathbf{U}, \mathbf{c}, \mathbf{V}$
 - * not specific to timestep t , but shared across all positions
 - * this “template” can get unrolled arbitrarily

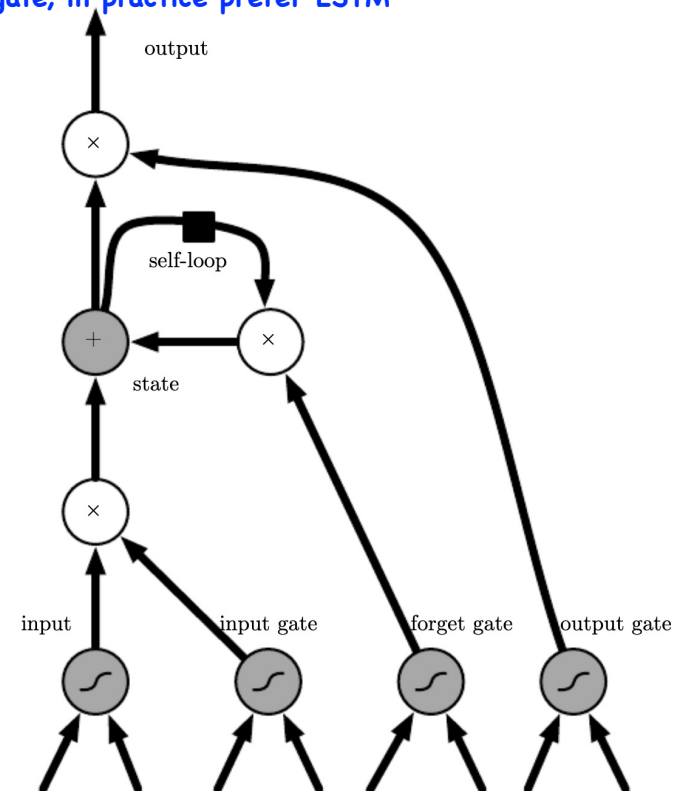
Training RNNs: Backprop. Thru. Time

- Backpropagation algorithms can be applied to network
 - * Called **backpropagation through time (BPTT)**
 - * Gradients from the loss at every position must be propagated back to the very start of the network
- Suffers from **gradient vanishing** problem
 - * Consider linear RNN, gradients of $\frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(1)}} = \mathbf{W}^{T-1}$, thus can explode or vanish with large T , depending on largest eigenvalue of \mathbf{W} (i.e., greater than / less than one).
 - * Can't *learn* long distance phenomena (over 10+ steps)

Long Short-Term Memory (LSTM)

- In RNN, previous state is provided as an input
 - * Multiplied by weight matrix, and non-linearity applied
- LSTM introduces state self-loop, based on copying
 - * Takes **copy** of previous state, scaled by sigmoid **forget gate**
- Gradient magnitude now maintained
 - * Can handle 100+ distance phenomena (vs 5-10 for RNN)

GRU: no gate, in practice prefer LSTM



Mini-summary

- Recurrent networks for modelling sequences
 - * recurrent units
 - * back-propagation through time
 - * long-short term memory
 - * applications

next: Transformers

Transformers

*A method for processing sequence inputs in highly
parallelizable manner, using **attention**.*

The hidden state acts as the memory of the RNN. It stores information from previous time steps and uses it to influence the output at the current time step and the hidden state for the next time step

Attention

which allows the model to focus on different parts of the input sequence when producing an output, rather than relying on a fixed-size hidden state.

- **RNNs** over long sequences not too good at representing properties of the full sequence
 - * **Biased** towards the end (or ends) of the sequence
 - * Last hidden layer / context: A **bottleneck!**

- **Attention** averages over hidden sequence

* $c = \sum_j \alpha_j h^{(j)}$ summary **weighted average**

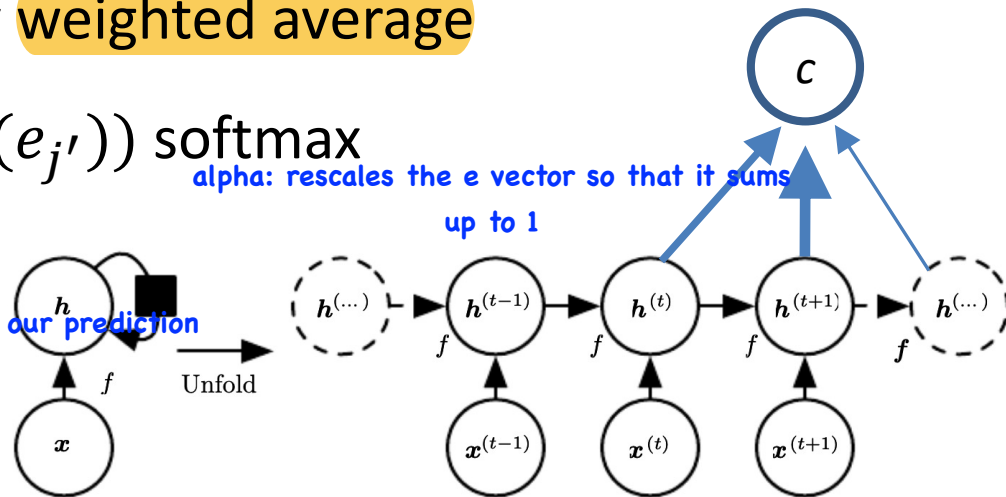
* $\alpha_j = \exp(e_j) / (\sum_{j'} \exp(e_{j'}))$ softmax

* $e_j = f(h^{(j)})$

alpha: rescales the e vector so that it sums up to 1

e: how important the hidden state is to our prediction

- E.g., key phrase in review



Repeated attention in Seq2seq models

- Consider multiple sequential outputs

$$* \mathbf{s}_i = f(\mathbf{s}^{(i-1)}, \mathbf{y}^{(i-1)}, \mathbf{c}_i)$$

a and f are NN

$$* \mathbf{c}_i = \sum_j \alpha_{ij} \mathbf{h}^{(j)}$$

context vector

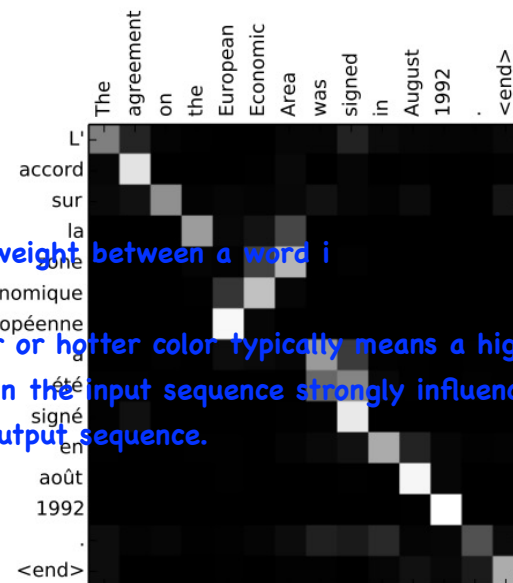
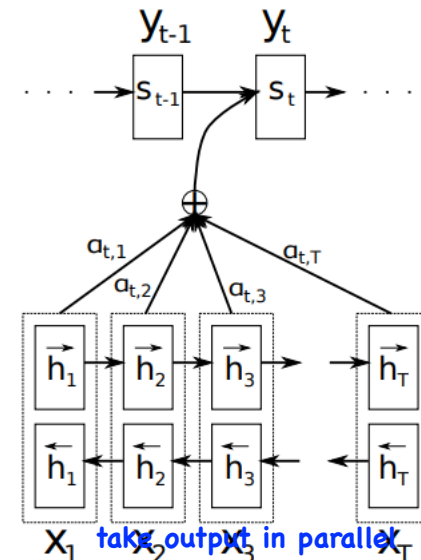
$$* \alpha_{ij} = \exp(e_{ij}) / (\sum_{j'} \exp(e_{ij'}))$$

$$* e_{ij} = a(\mathbf{s}^{(i-1)}, \mathbf{h}^{(j)})$$

a: alignment problem

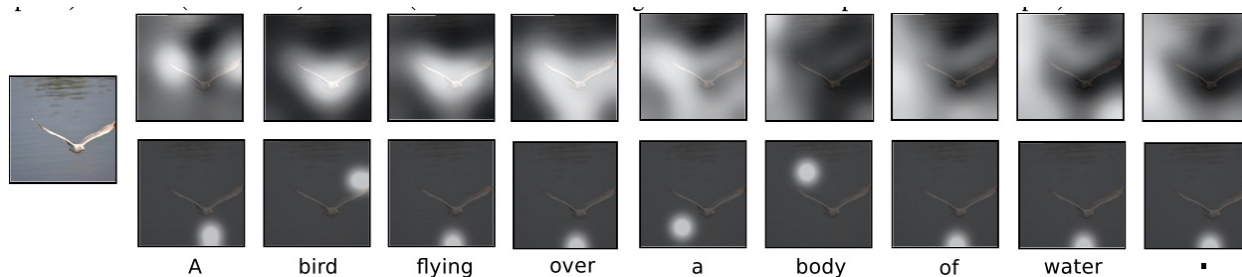
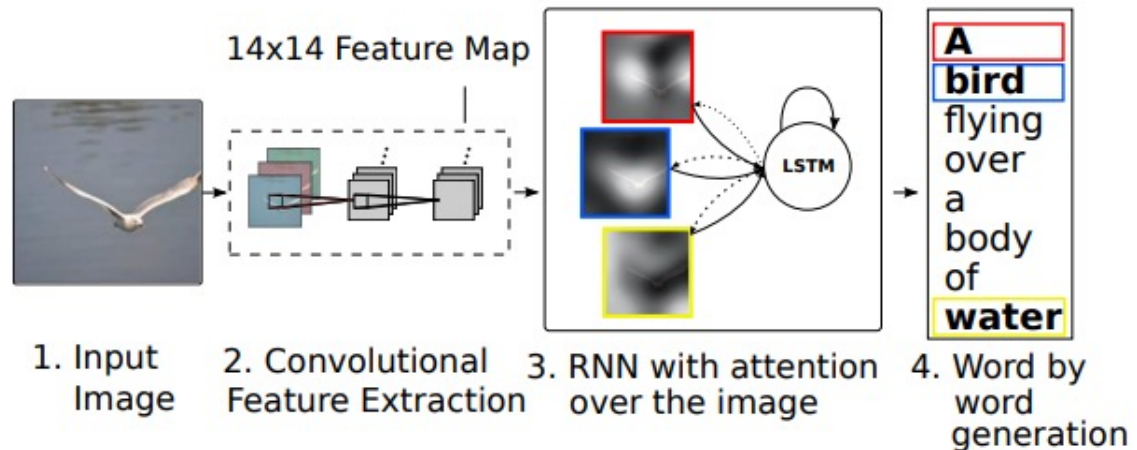
- Avoids bottleneck, and uncovers meaningful structure

cell represents attention weight between a word i magnitude of the attention weight. A darker or hotter color typically means a higher attention weight, indicating that the word in the input sequence strongly influenced the word in the output sequence.



Attention in Vision

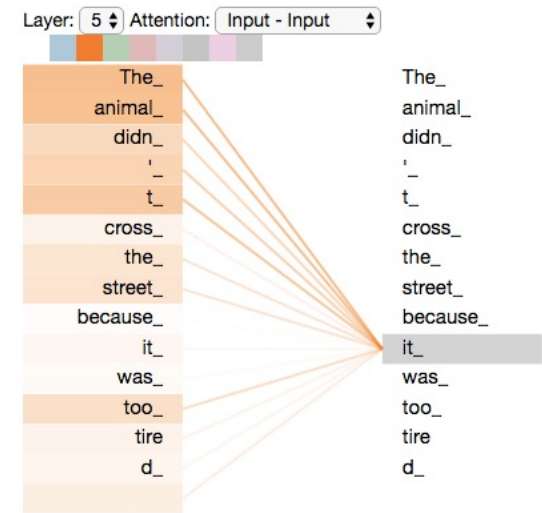
- Can attend to other representations, e.g., images
 - * Attention over matrix input
 - * RNNs during generation of caption



Show, Attend and Tell: Neural Image Caption Generation with Visual Attention,
Xu et al, ICML 2015

Self-attention

- **Transformers** use attention as means of representing sequences directly, instead of RNN
 - * Representation of item i is based on attention to the rest of the sequence
no explicit sequential modeling going on
 - * Use item i as the query in attention against all items $j \neq i$
- Compared to RNNs
 - * No explicit position information (add to each symbol position index)
 - * Cheap: easily done in parallel



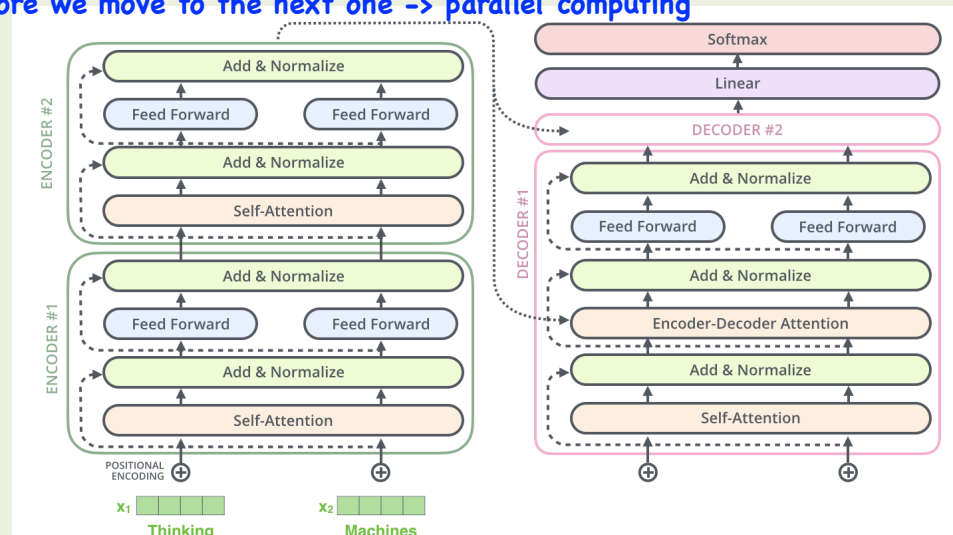
The Illustrated Transformer, <http://jalamar.github.io/illustrated-transformer/> attention is all you need -> we don't need RNN anymore!!!?

Transformer

RNN doesn't have sequential information, we have that as input in RNN
eg. positional embedding

- The Transformer uses self-attention as its main step
 - * Alongside residual, and normalization layers
 - * Using multiple “attention heads”, and deep stacking
- Applied first to translation
 - * Then raw text, e.g., BERT, RoBERTa, GPT
 - * Highly scalable
 - * Large performance gains over RNN models

we have to calculate each step before we move to the next one -> parallel computing



The Illustrated Transformer,
<http://jalamar.github.io/illustrated-transformer/>

This lecture

- Recurrent networks for modelling sequences
 - * recurrent units
 - * back-propagation through time
 - * long-short term memory
- Transformers and attention