COMP 551 – Applied Machine Learning
Lecture 17: Deep Learning (cont'd)

Instructor: Ryan Lowe *(ryan.lowe@cs.mcgill.ca)*

Slides by: Joelle Pineau & Ryan Lowe

Class web page: *www.cs.mcgill.ca/~jpineau/comp551*

Announcements

- Course schedule has been shifted a bit
  - Lectures on Bayesian inference & Gaussian processes now come before unsupervised learning
  - Sarath Chandar (lecturer for other section) will be giving unsupervised learning lecture on March 26
  - Tentative topics after midterm: "**Reinforcement learning**", and "**Frontiers in (deep) machine learning**"

Major paradigms for deep learning

- **Deep neural networks**:
  - Supervised training: Feedforward neural networks (also called **multi-layer perceptrons**, MLPs).
  - Unsupervised pre-training: Stacked autoencoders.

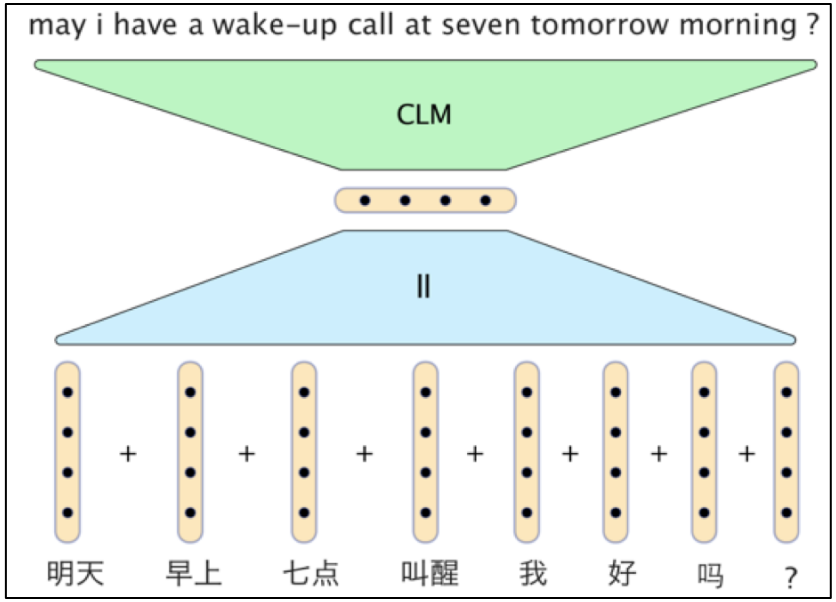
- Special architectures for different problem domains.
  - Computer vision => Convolutional neural nets.
  - Text and speech => Recurrent neural nets.

Neural models for sequences

- Several datasets contain **sequences** of data (e.g. time-series, text)

- How could we process sequences with a **feed-forward neural network**?

1. Take vectors representing the last N timesteps and **concatenate** (join) them
2. Take vectors representing the last N timesteps and **average** them



From Phil Blumson's slides

Neural models for sequences

- **<u>Problem:</u>** these approaches **don't exploit the sequential nature of the data!!**
- Also, can only consider information from a fixed-size context window
- **Temporal information is very important in sequences!!**
- E.g. machine translation:

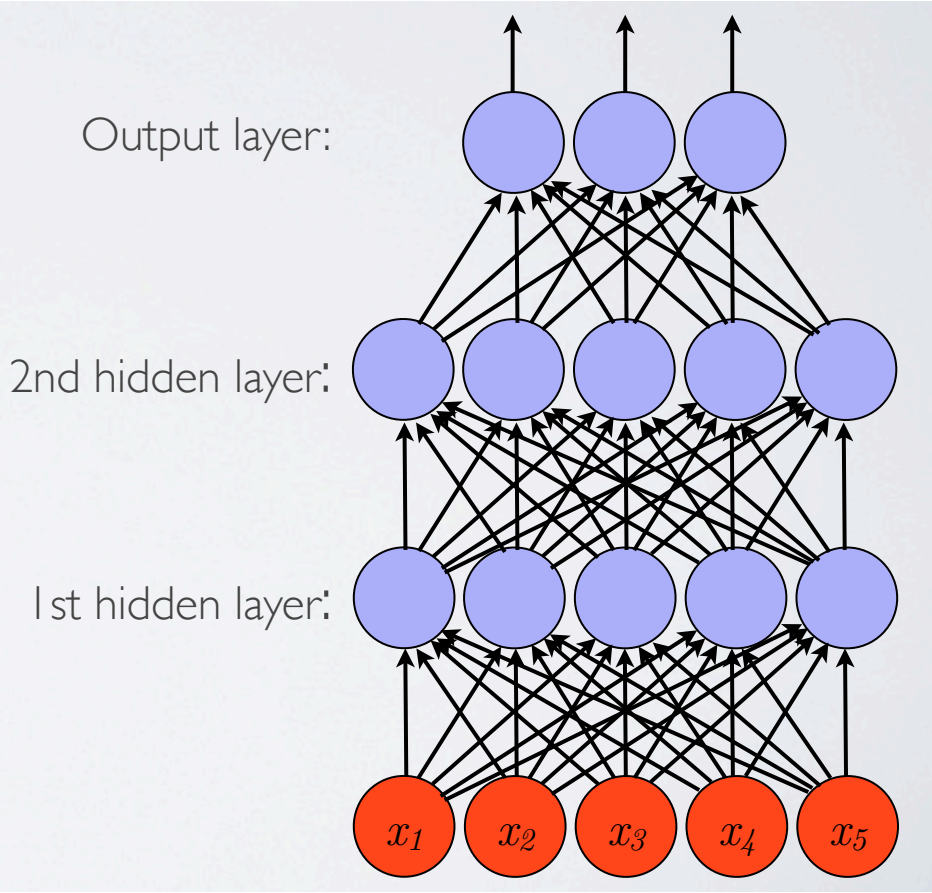
"John hit Steve on the head with a bat"

!= "Steve hit John on the bat with a head"

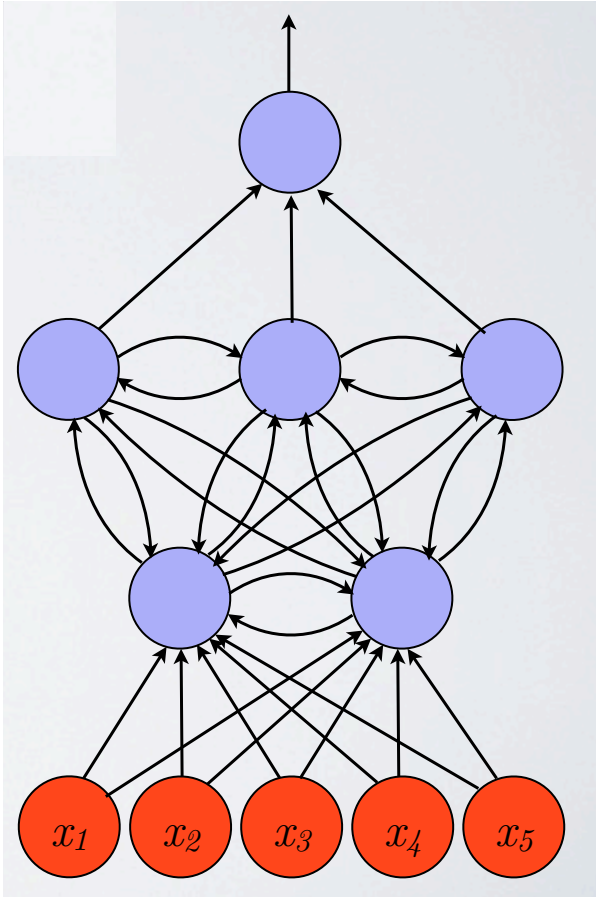
!= "Bat hit a with head on the John Steve"

Recurrent Neural Networks (RNNs)

Feed-forward neural net



Add cycles in network



Recurrent Neural Networks (RNNs)

What kind of cycles?

- Cycles with a **time delay**
- Box means that the information is sent at the **next time step** (no infinite loops)

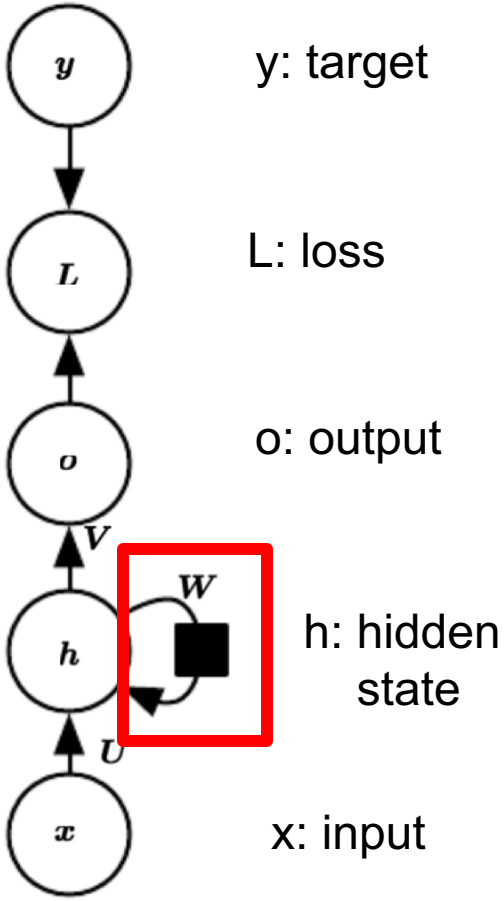


Image from deeplearningbook.org

Recurrent Neural Networks (RNNs)

What does this allow us to do?

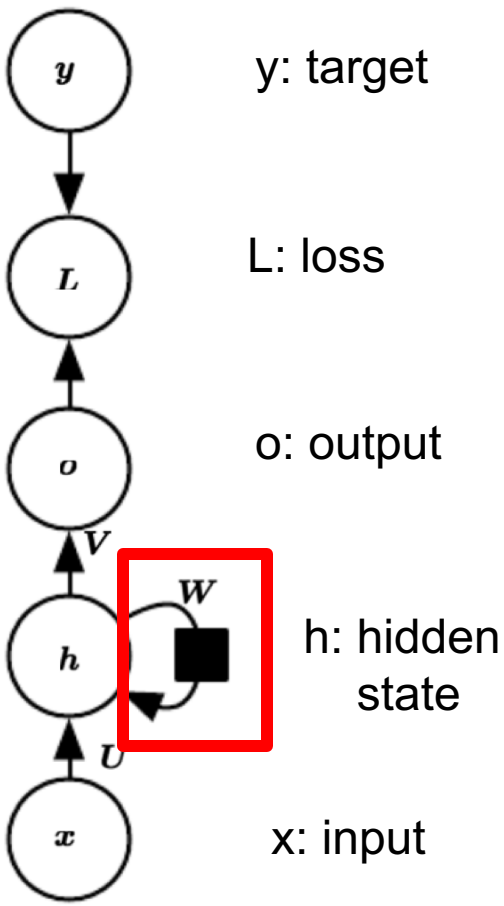
- Can view an RNN as having a ***hidden state*** *h_t* that changes over time
- *h_t* represents **'useful information'** from past inputs

$$\boldsymbol{h_{t+1} = f(h_t, x_t)}$$

Often, $h_{t+1} = \sigma(Wh_t + Ux_t + b_h)$, where sigma is the non-linearity, b is bias

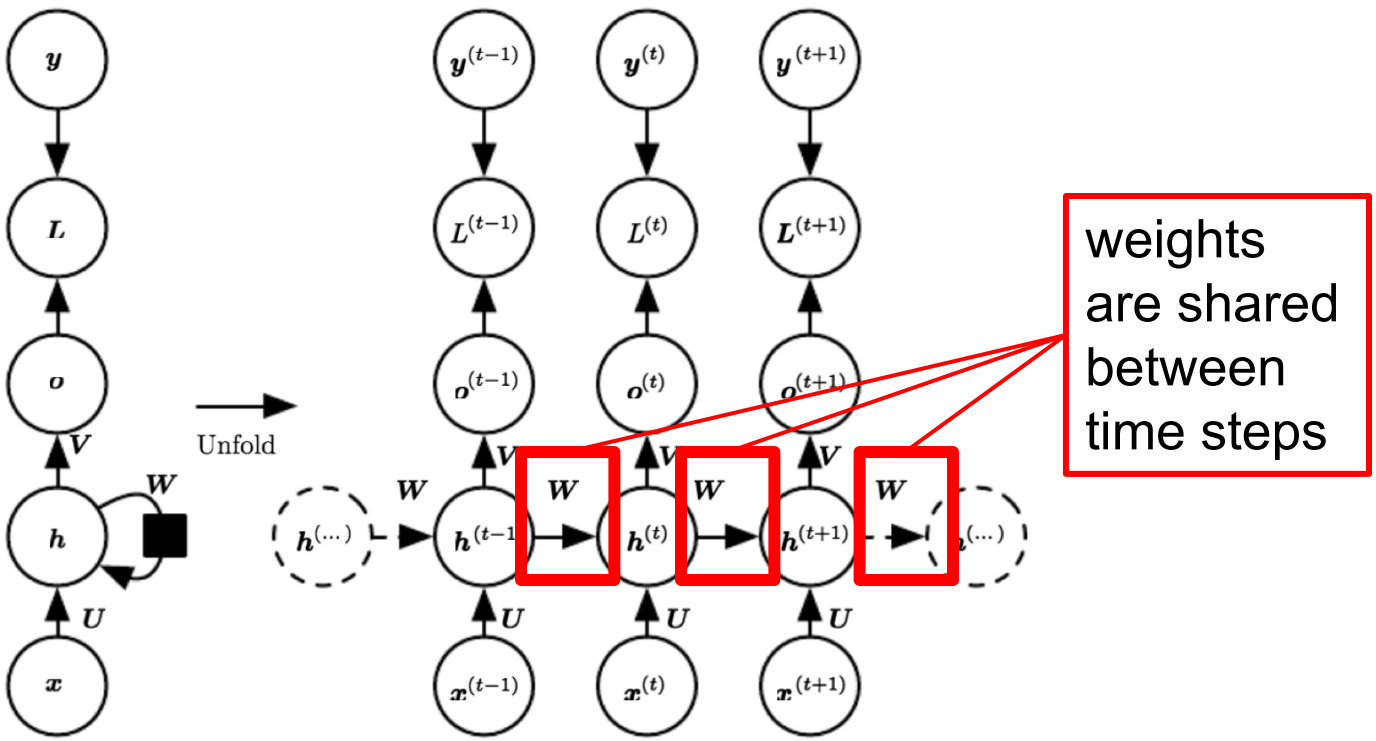
- h_t used to make predictions:

$$o_t = \sigma(Vh_t + b_o)$$



Recurrent Neural Networks (RNNs)

- Can **unroll** the RNN over time to form an acyclic graph.
- **RNN = special kind of feed-forward network**



Example: language modelling

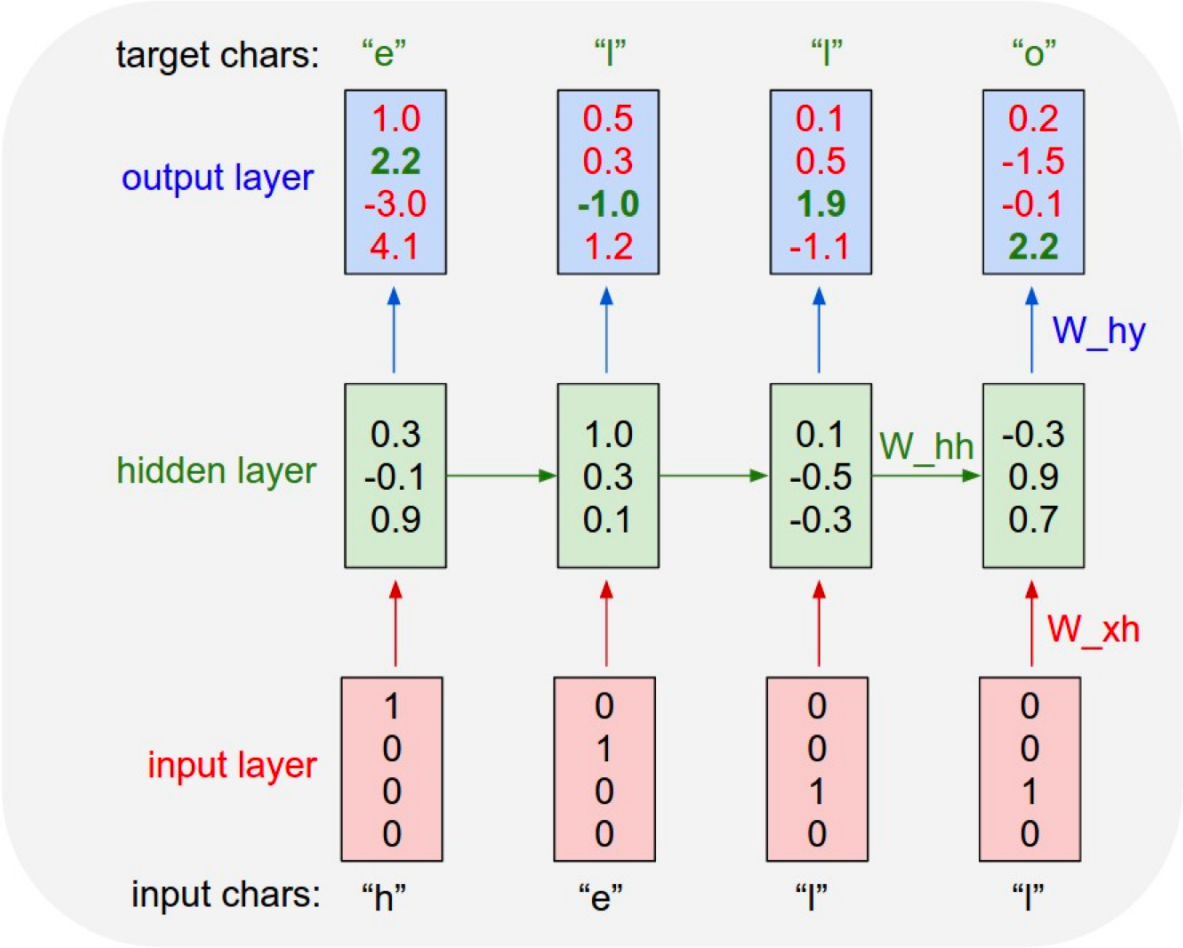
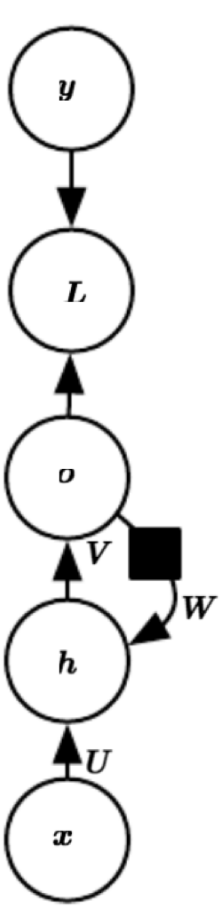


Image from: karpathy.github.io

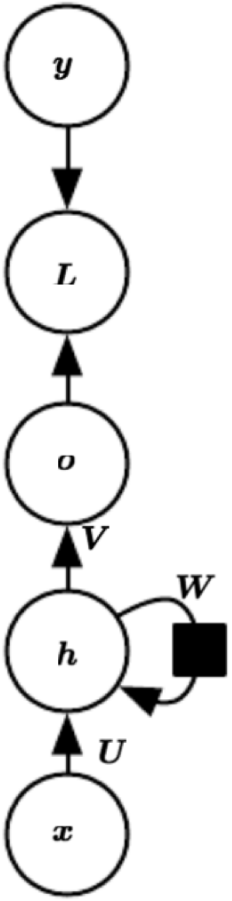
Kinds of cycles

- **Q: Which is better?**



Jordan RNN

vs.

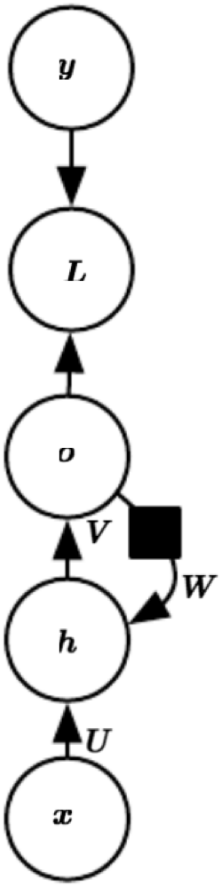


Elman RNN

Kinds of cycles

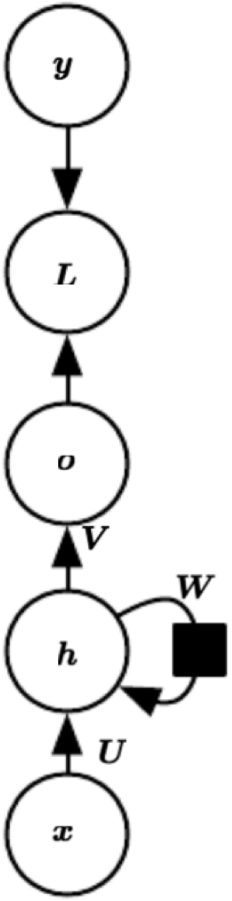
- **Q: Which is better?**
- **A:** Elman RNN. Usually output *o* is constrained in some way, and may be missing some important info from the past.

(in practice, sometimes both connections are used)



Jordan RNN

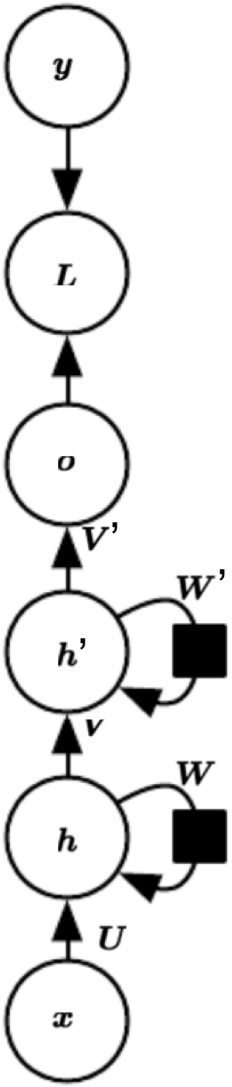
vs.



Elman RNN

Kinds of cycles

- What kinds of cycles can you have?
- <u>In theory:</u> any kind!
- <u>In practice:</u> usually self-connections are grouped **within a single hidden layer** (vs. across layers)
- Often in language modelling, also feed the previous output as input at next time step to help make predictions (more on that later)



(See ch10.5 of deep learning textbook, Goodfellow et al.)

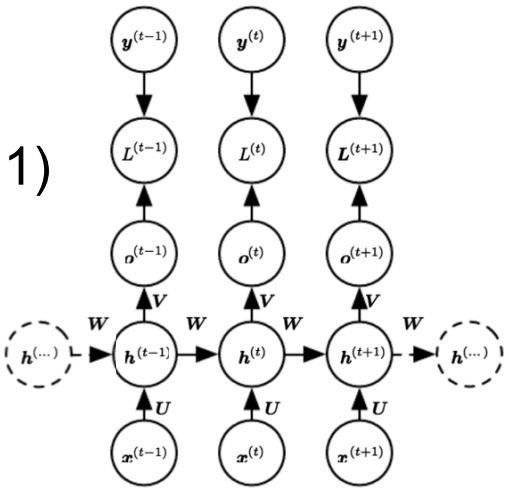
Kinds of target

- How do we specify the target output of an RNN?
- Many ways! Two main ones:

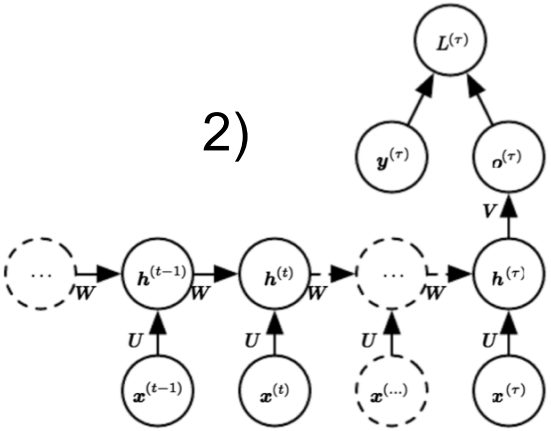
1) Can specify an target **at each time step**
   - Ex: generating language

2) Can specify one target **at the end of the sequence**
   - Ex: sentiment classification

1)

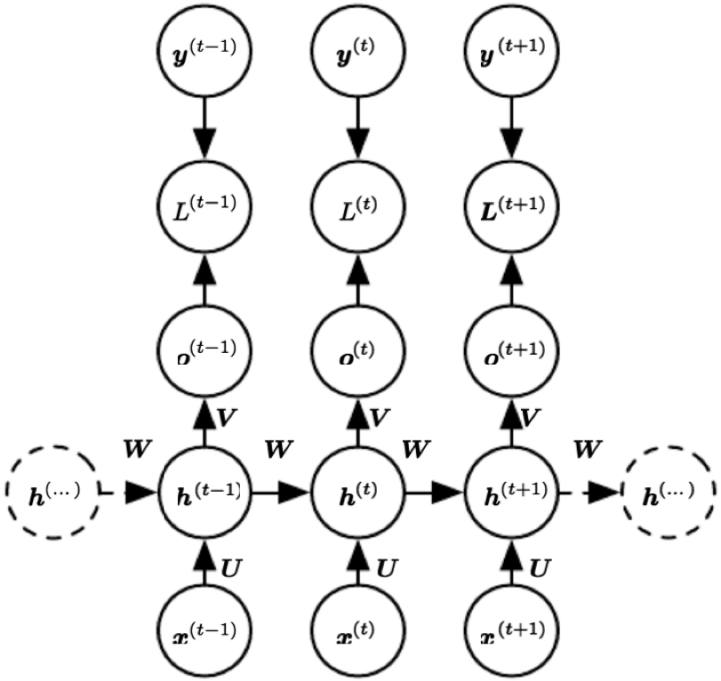


2)



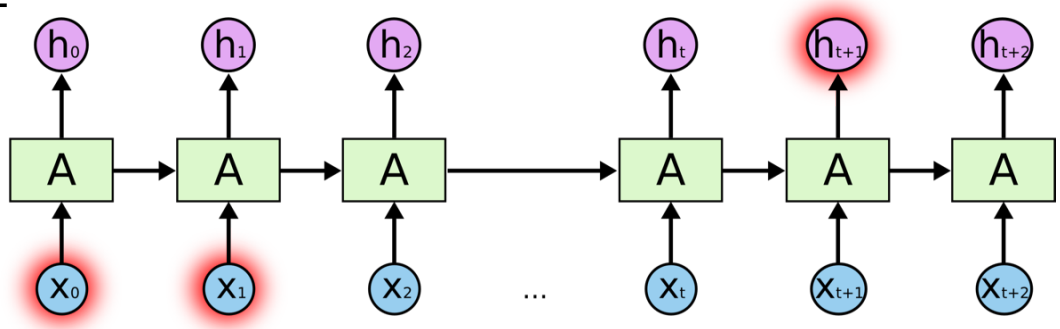
Training RNNs

- How can we train RNNs?
- Same as feed-forward networks: train with backpropagation on unrolled computation graph
- This is called **backpropagation through time** (BPTT)
- Same derivation as regular backprop (use chain rule)



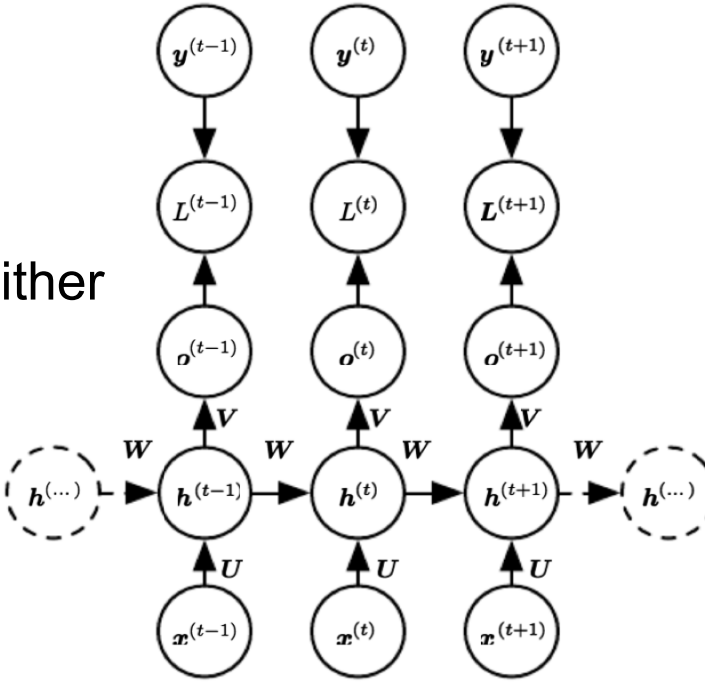
The problem of long-term dependencies

- Let's say we are doing language modelling
- Input paragraph: *"I grew up in France. I worked at […]. I speak fluent French."*
- Want to predict 'French' given words before. This can be hard!
- In practice it is very hard for RNNs to to learn dependencies lasting many time steps.
- **Why could this be?**



The problem of long-term dependencies

- Because the hidden-to-hidden transition matrix *W* is **the same for each time step**, this can cause the gradients to **explode** or **vanish**
- **<u>Intuition:</u>** Imagine multiplying a scalar number *w* by itself many times. This will either explode (if *w* > 1) or vanish (if *w* < 1)
- Similar behavior occurs if *W* is a matrix



The problem of long-term dependencies

- **<u>Recall:</u>** a way to intuitively think of backpropagating gradients
- If I change my input by a small amount, what will be the result on the output?

⇔ if I want my output (loss) to decrease, how do I change my input?

- If input is being multiplied by same *W* many times, this could cause either a huge or tiny effect on the output

=> gradient of loss wrt parameters could be huge or tiny

The problem of long-term dependencies

(Aside: an eigendecomposition view)

- Consider an RNN without inputs or activation functions: $h_t = W h_{t-1}$
- If *W* admits an eigendecomposition of the form $W = Q\Lambda Q^T$, where Λ is a diagonal matrix of eigenvalues and Q is orthogonal, then

$$h_t = W^t h_0$$

$$h_t = Q\Lambda Q^T Q\Lambda Q^T \dots h_0$$

$$h_t = Q\Lambda^t Q^T h_0$$

since $QQ^T = I$. So each eigenvalue is raised to the power of *t*, causing eigenvalues <1 to vanish and >1 to explode.

How to avoid vanishing/exploding gradients?

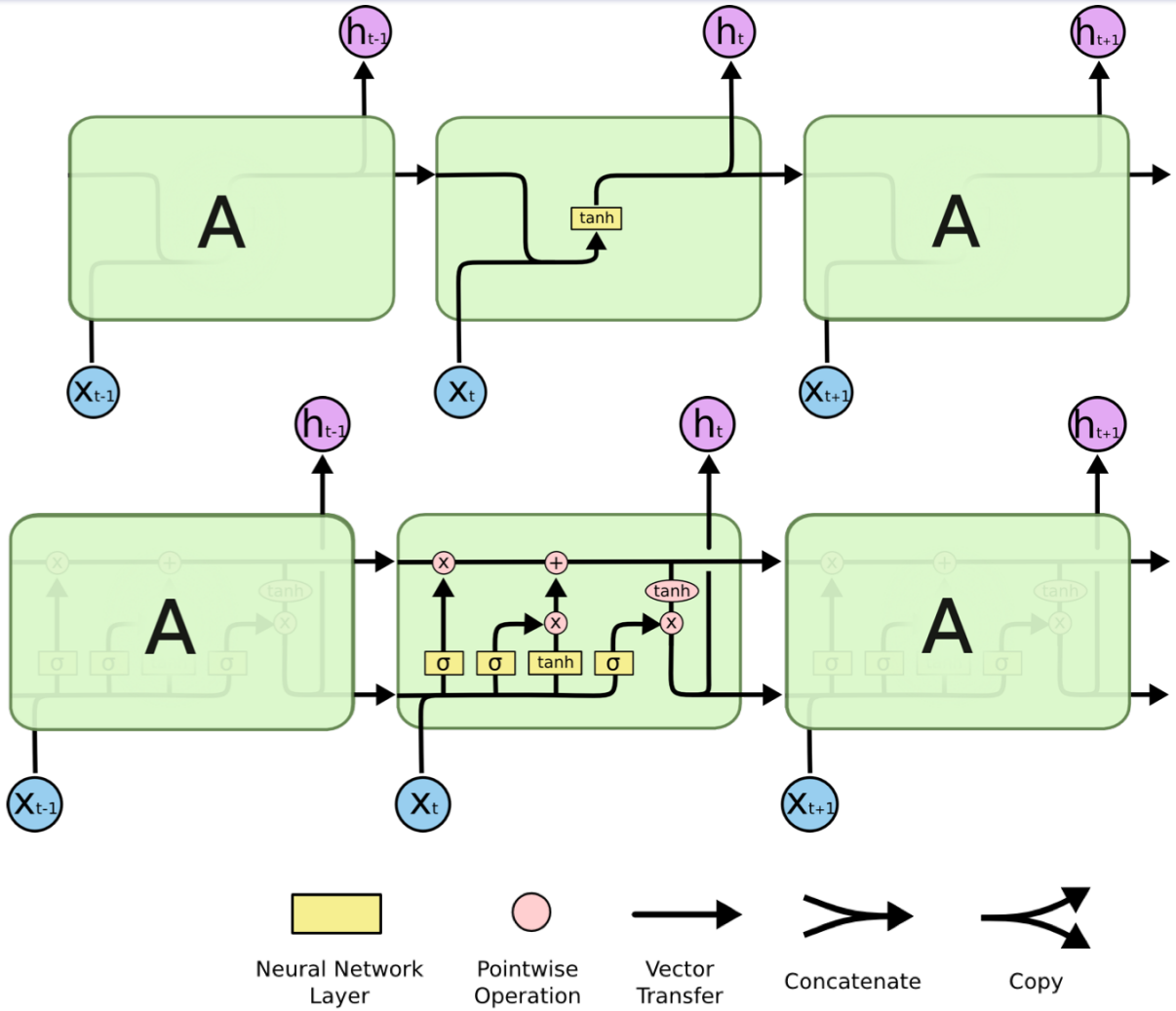
- Simple way to avoid exploding gradients: **gradient clipping**
  - if |*gradient*| > *value*:
  - *gradient* = *value* * sign(*gradient*)

- Another way: can change the architecture of the RNN so there are some non-multiplicative interactions
  - E.g.: long short-term memory (LSTM) units

Long short-term memory (LSTM) units

RNN:

LSTM:



LSTM images from: colah.github.io

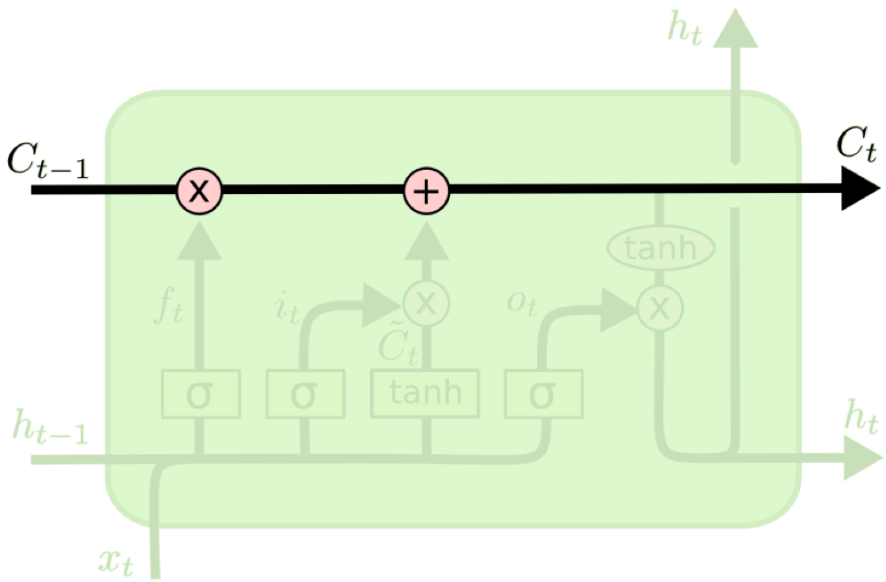
Long short-term memory (LSTM) units

- Much better at dealing with long-term dependencies
- Can think of it as a special ‘cell’
- Governed by a set of update equations:

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

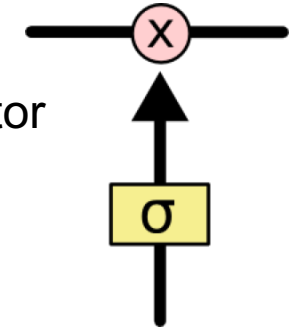
LSTMs

- <u>Core idea</u>: the **cell state**, an 'information highway'
- Cell state is updated **additively** based on input, rather than **multiplicatively** => *<u>less prone to exploding/ vanishing gradients</u>*



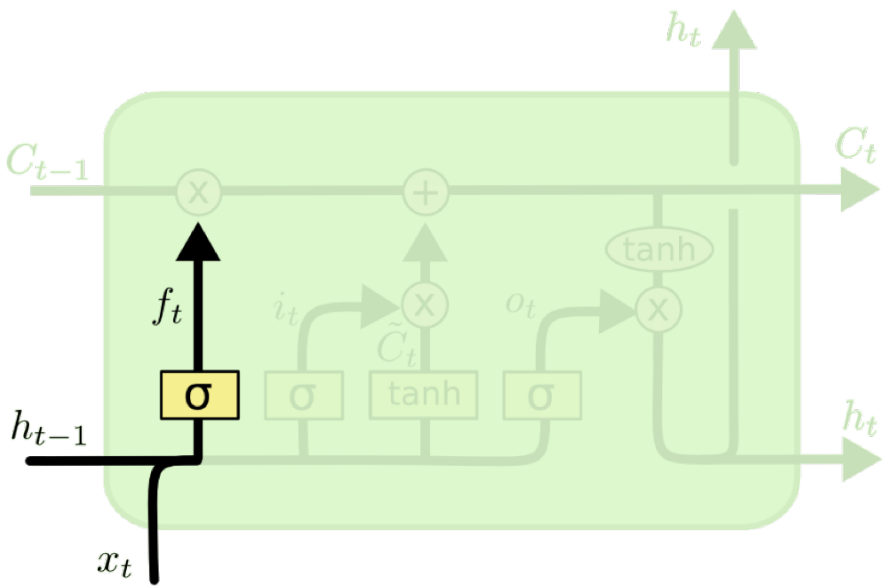
LSTMs

- Cell state vs hidden state (roughly)
  - Hidden state: what info from past do I need **to make my next prediction**?
  - Cell state: what info from past might I need **to make future predictions**?
- For regular RNN, hidden state plays both of these roles
- LSTM uses a set of '**gates**' to control information flow
  - Gate = sigmoid layer + elem.-wise multiplication. Gives vector of numbers between [0,1] that determine how much of each component to let through



LSTMs

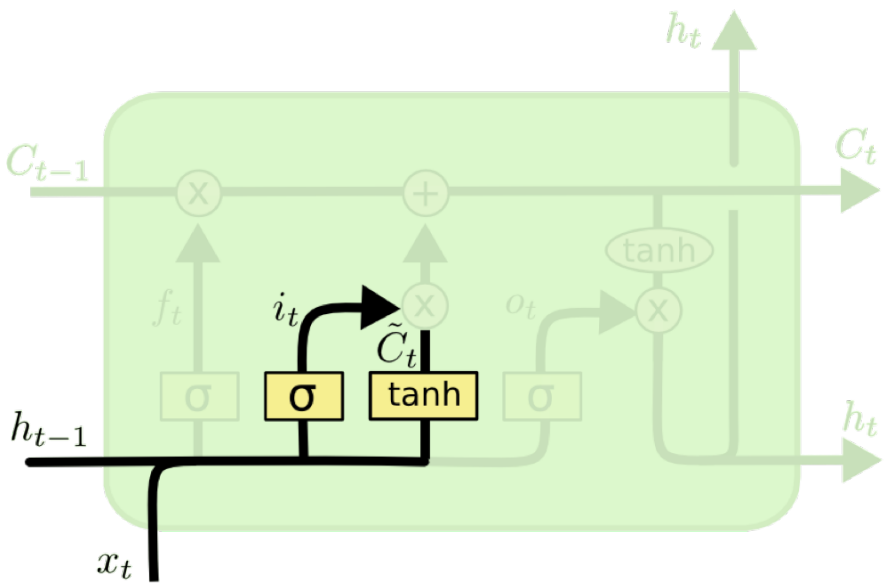
- **Forget gate**: how much information do we want to keep from the previous cell state?



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTMs

- **Input gate**: what information from the current input (and previous hidden state) do we want to transfer to the cell state?

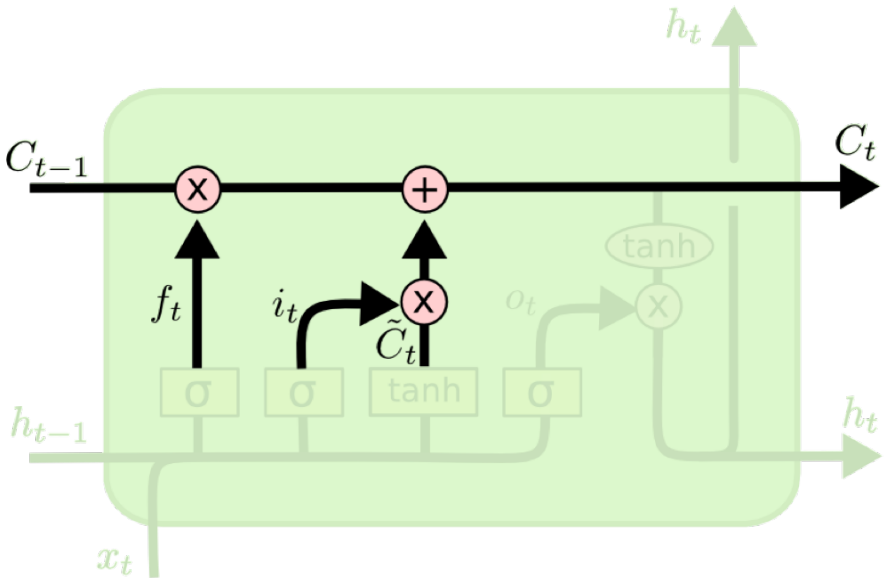


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTMs

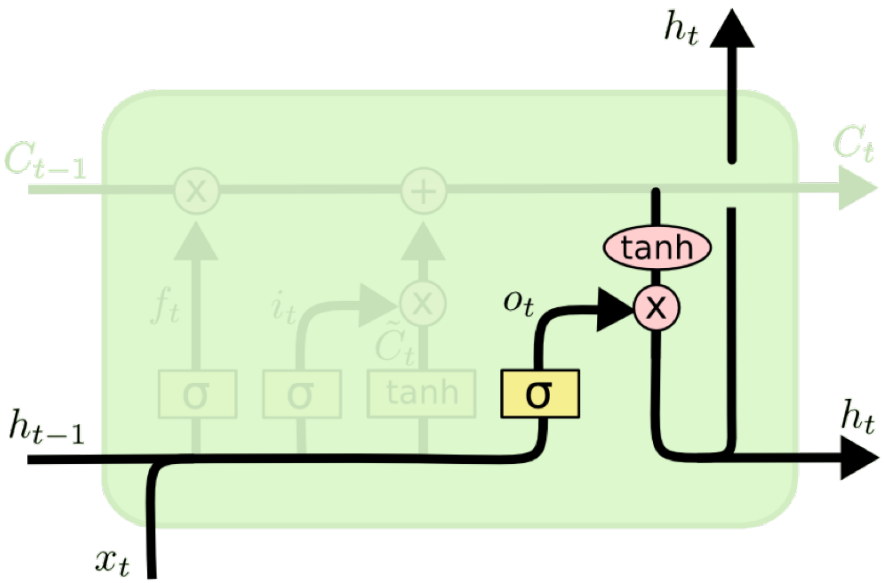
- Cell state updated as an **additive linear combination** of old cell state and processed input



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTMs

- **Output gate:** what information from the cell state do we need to make the next prediction?



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTMs

- LSTM architecture has existed for many years (Hochreiter & Schmidhuber 1997).

- Several state-of-the-art results:
  - Cursive handwriting recognition (Graves & Schmidhuber, 2009)
  - Speech recognition (Graves, Mohamed & Hinton, 2013)
  - Machine translation (Sutskever, Vinyals & Le, 2014)
  - Question-answer (Weston et al., 2015)
  - Unstructured dialogue response generation (Serban et al., 2016)

- Other similar models can be used (e.g. Gated Recurrent Units)

Still confused?

- See this great post by Chris Olah explaining LSTMs (the source of the LSTM diagrams in these slides):
  - https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural Language Modelling

- Given sequence of words: $x_1, x_2, \ldots, x_t$
- <u>Language Modelling:</u> want to estimate

  $p(x_t \mid x_{t-n}, \ldots, x_{t-1})$

Continuous space word representation

$$s_{t'} = W^\top x_{t'}, \text{ where } W \in \mathbb{R}^{|V| \times d}$$

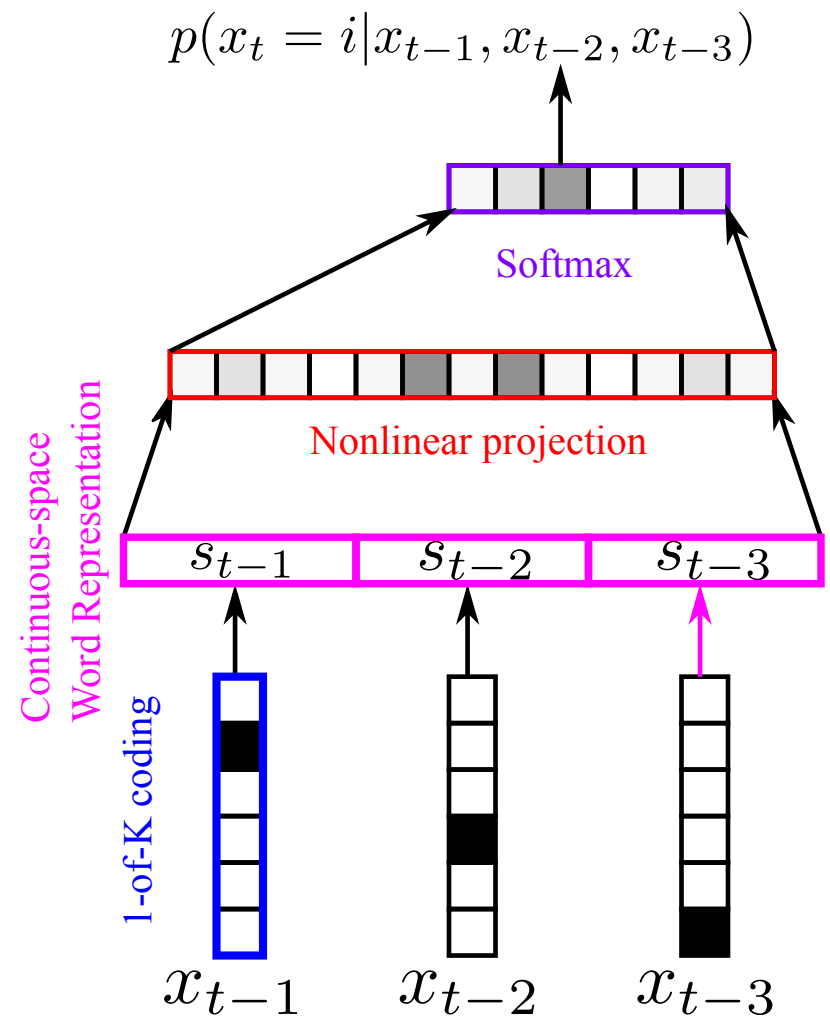
Nonlinear hidden layer

$$h = \tanh(U^\top [s_{t-1}; s_{t-2}; \cdots; s_{t-n}] + b)$$

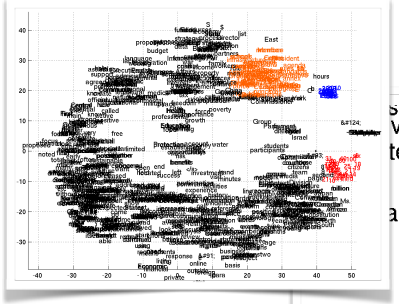
$$, \text{ where } U \in \mathbb{R}^{nd \times d'} \text{ and } b \in \mathbb{R}^{d'}$$

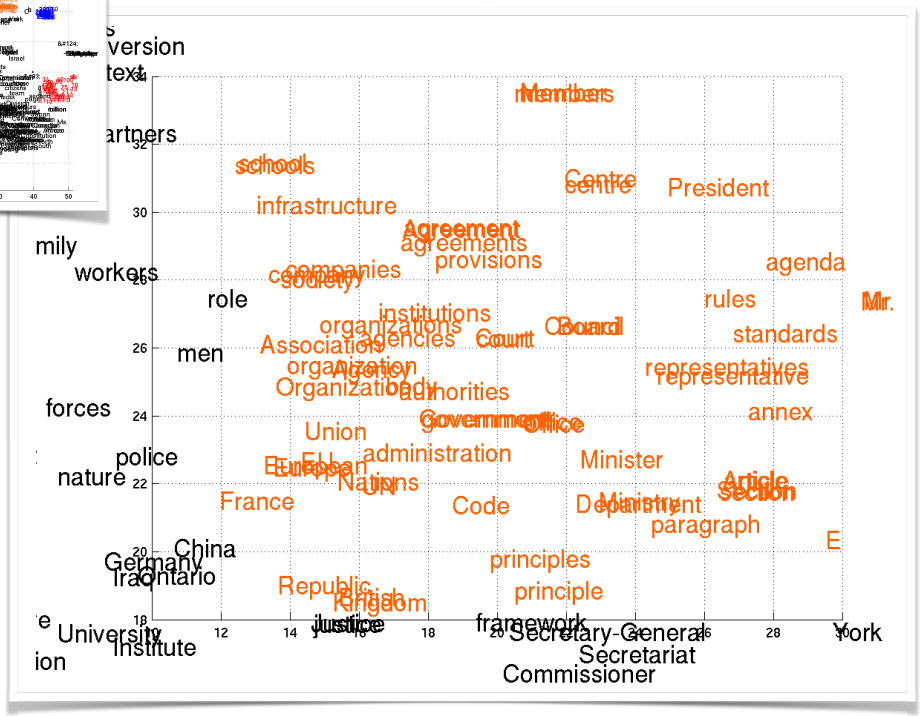
Softmax normalization

$$p(x_t = i | x_{t-n}, \ldots, x_{t-1}) = \frac{\exp(y_i)}{\sum_{j=1}^{|V|} \exp(y_j)}$$



Aside: word vectors





- Problem with using ‘one-hot’ word representation: no relationship between similar words
- **<u>Solution</u>**: use small, dense vectors that captures word semantics!
- Can learn these from data (e.g. *word2vec,* Mikolov et al. (2013))

Language modelling with RNNs

- Problem with using MLPs to generate language (or any sequence): *can only consider fixed window into the past*
- RNNs naturally work with sequences, considers entire history
- Useful information from past stored in the RNNs hidden state (or cell state of LSTM)

Language modelling with RNNs

- Want to model probability of a sequence $p(x_1, \ldots, x_T)$
- Can directly model the conditional probabilities:

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

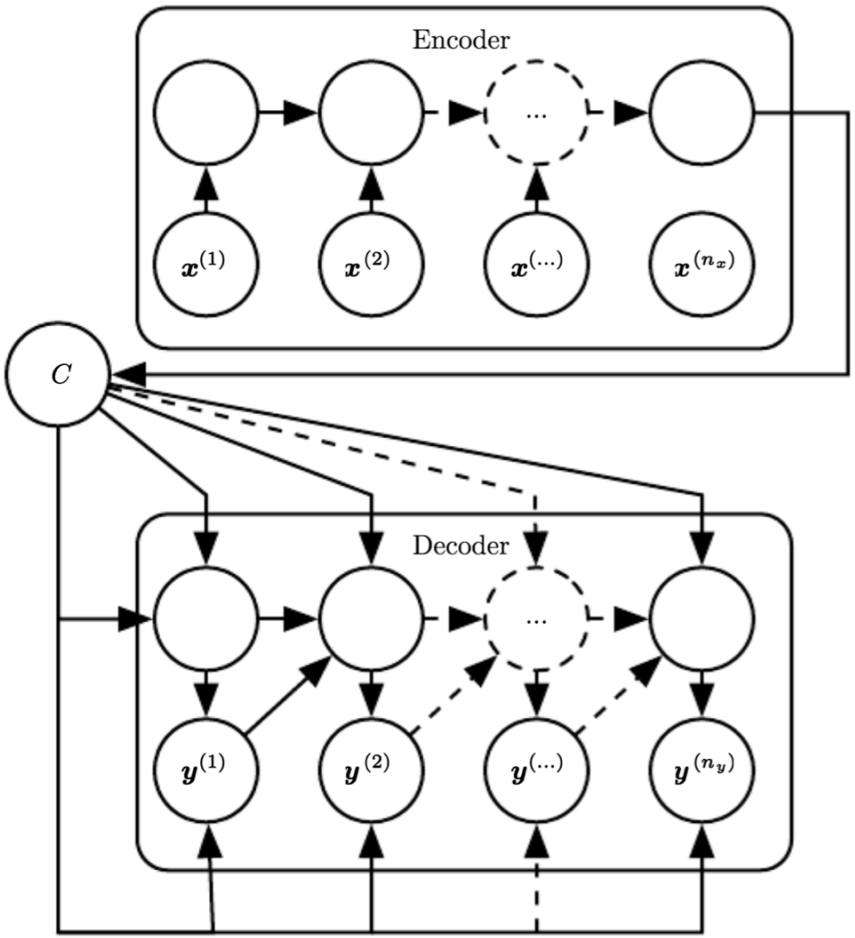
- So at each step we calculate the probability of the next token (word) given all of the previous ones
- Again use softmax as output at last layer
- Can now **generate language by sampling** from $p(x_t \mid x_1, \ldots, x_{t-1})$

Generating sequences with RNNs

- With a single RNN, can produce output sequences that have the same length as the input sequence
- **Question:** What if we want to go beyond language modelling, and produce an outputs with **a different length** from our input?
- E.g.: machine translation, image captioning
- **Solution:** ***Use two RNNs!*** One to encode the input, and one to produce the output

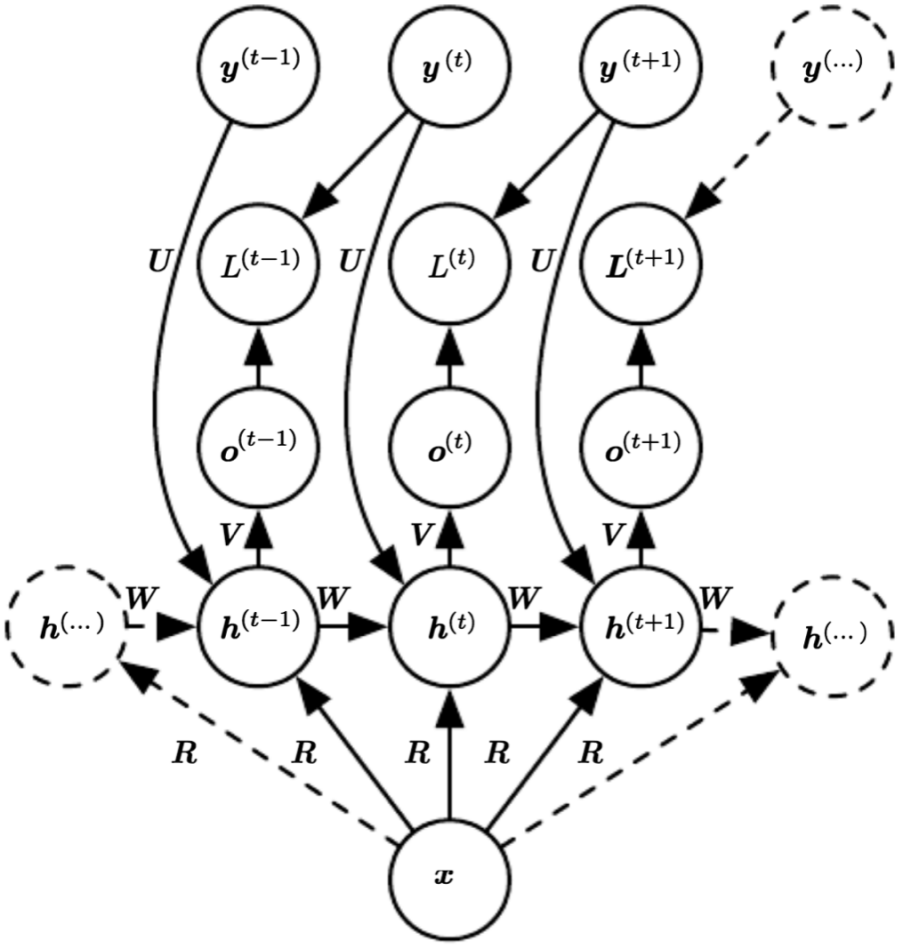
Generating Sequences with RNNs

- How do we consider an arbitrary length sequence as input?
- Use a **separate 'encoder' RNN** to transform the input into a vector!
- Usually, the *last hidden state* of the encoder RNN is used to condition the 'decoder'
- This is the **encoder-decoder model**
- Trained entirely with backprop

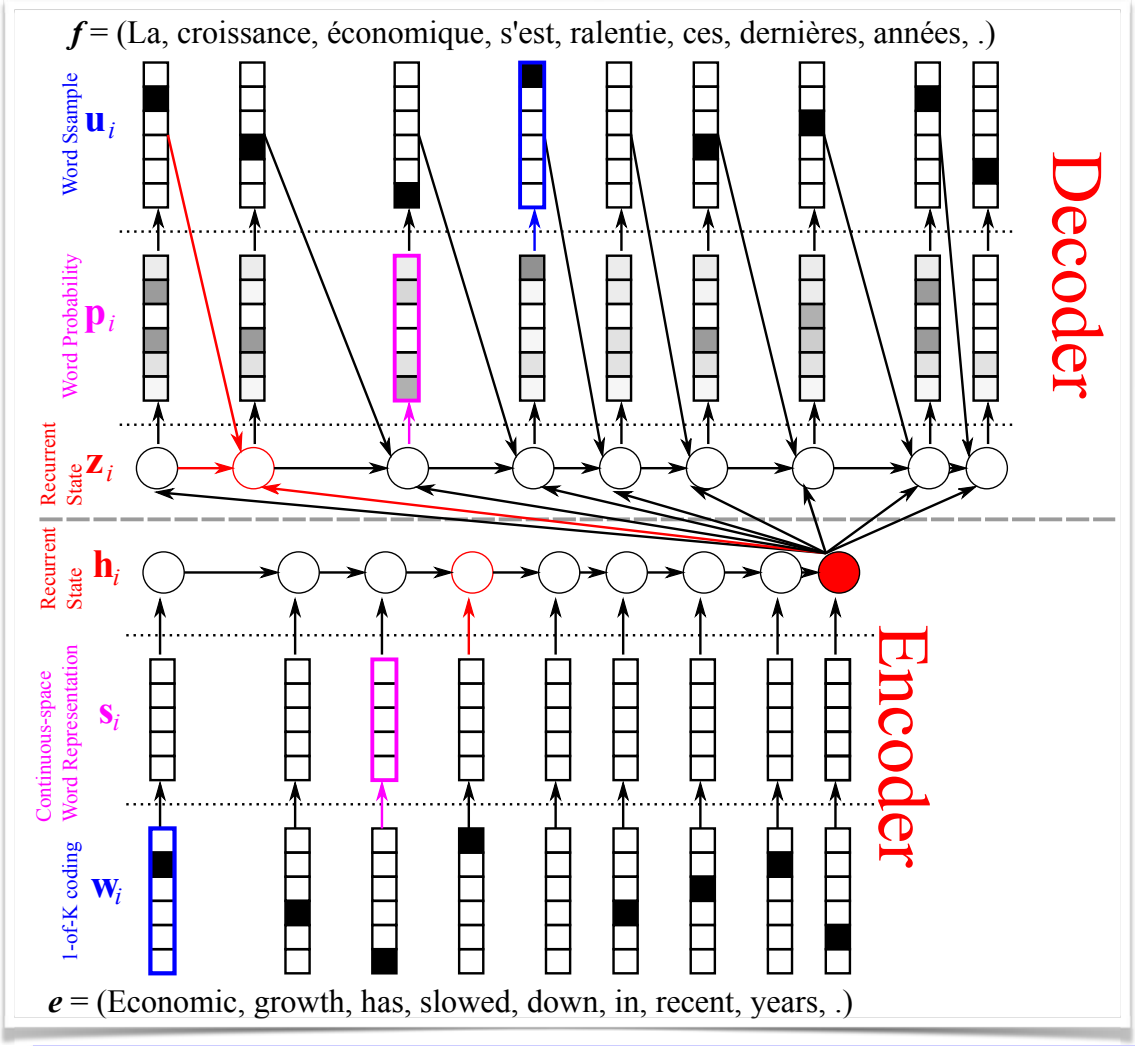


A closer look at the decoder

- A closer look at the decoder (x = C, the context)
- **Q:** How do you know when to stop generating?
- **A:** Add a special **"end-of-sequence" symbol**



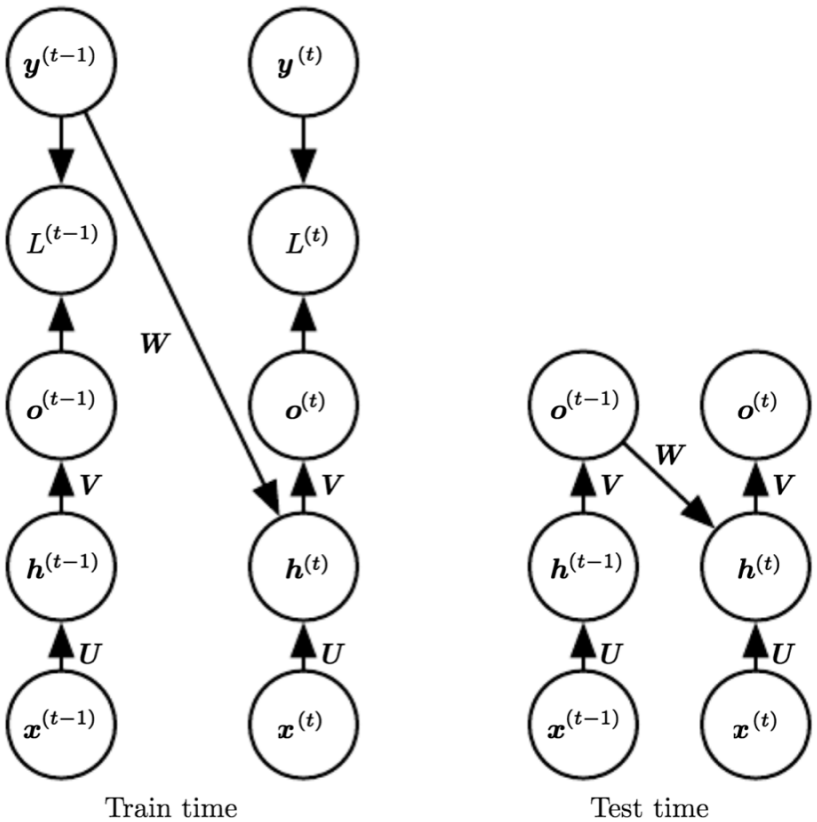
Encoder-decoder: another picture



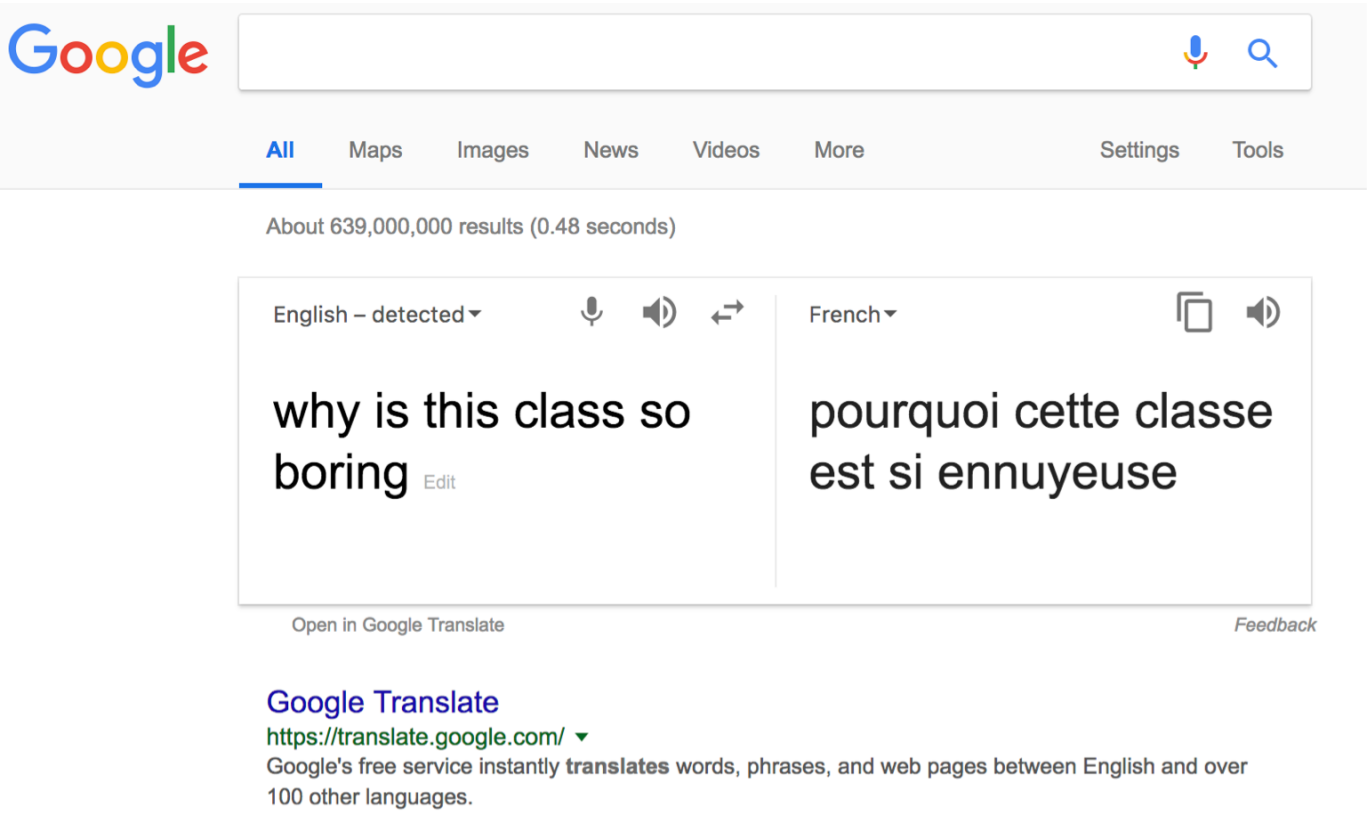
(Chrisman, 1991;
Forcada&Ñeco, 1997;
Castaño&Casacuberta, 1997;
Kalchbrenner&Blunsom, 2013;
Sutskever et al., 2014;
Cho et al., 2014)

Teacher forcing

- Method of training RNNs if the model receives ground-truth output as input
- **<u>Train time:</u>** condition on previous ground-truth output y_{t-1}
- **<u>Test time:</u>** don't have true labels, so condition on your own prediction o_{t-1}

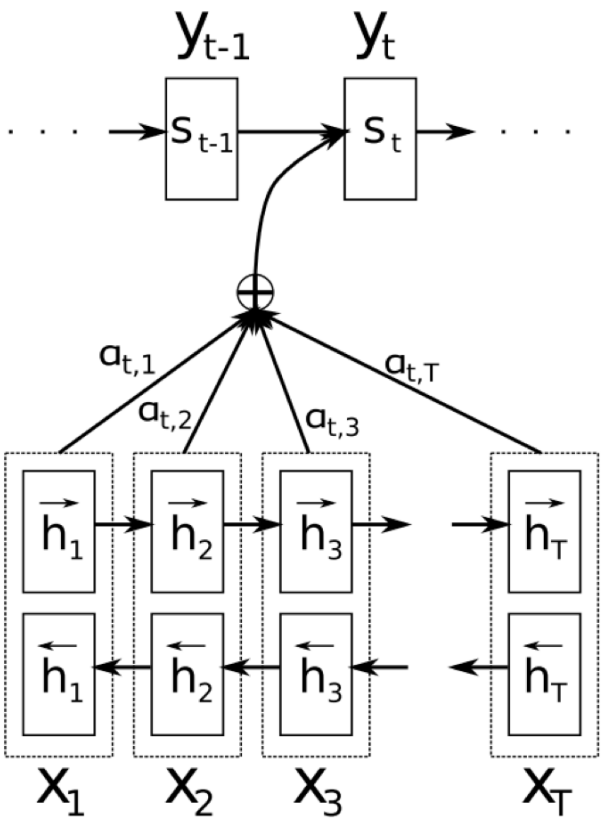


Application: machine translation



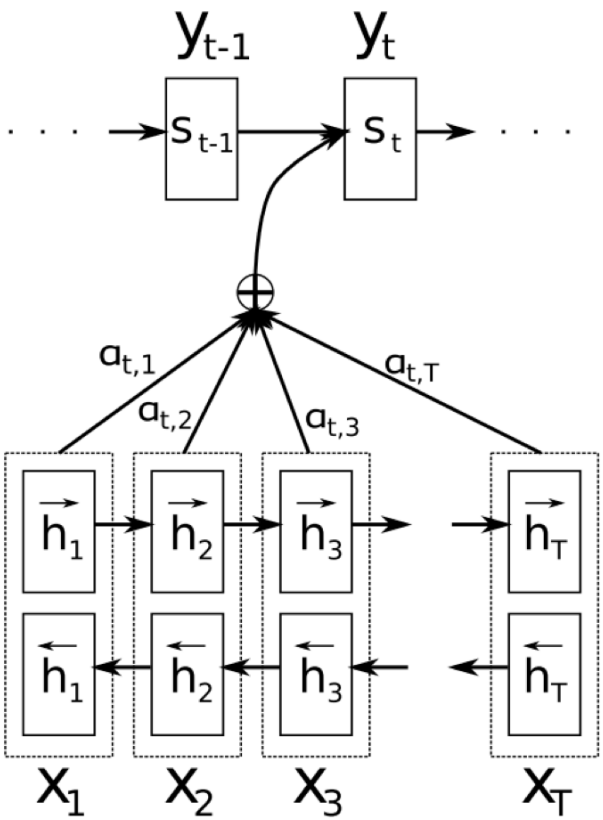
Application: machine translation

- Bahdanau et al. (2014) apply **attention** to machine translation
- **<u>Idea</u>**: when translating, *humans only need to look at certain parts of source sentence*
- Calculate scalar '*attention weights*' α for each part of the input
- Helps avoid the 'bottleneck problem'
- **Trained with backprop**



Application: machine translation

- Run a '**bidirectional LSTM**' (i.e. an LSTM both ways) over the input, concatenate hidden states
- Scalar values α calculated with a neural network
- Use a **weighted combination** of inputs to give 'context vector', used for prediction



$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \qquad \text{[Attention weights]}$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \qquad \text{[Context vector]}$$

Application: machine translation

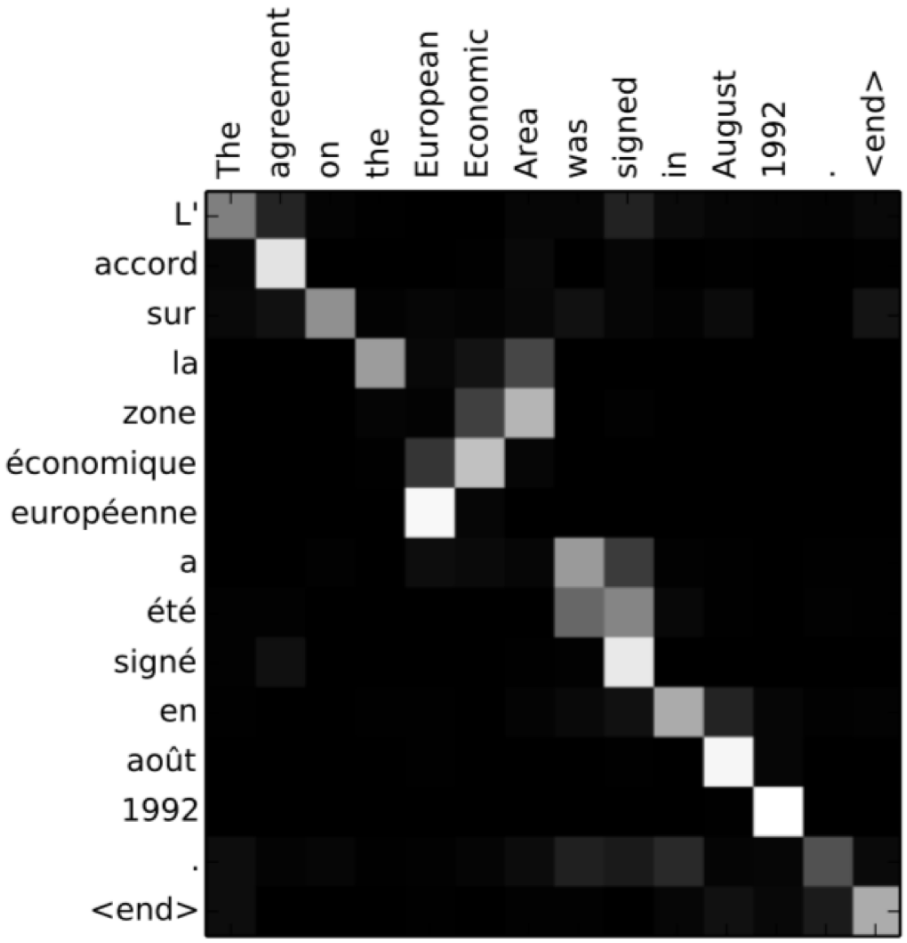


Image captioning

- Can use principle of ‘attention’ over images instead of text
- Combine with ConvNet to extract features

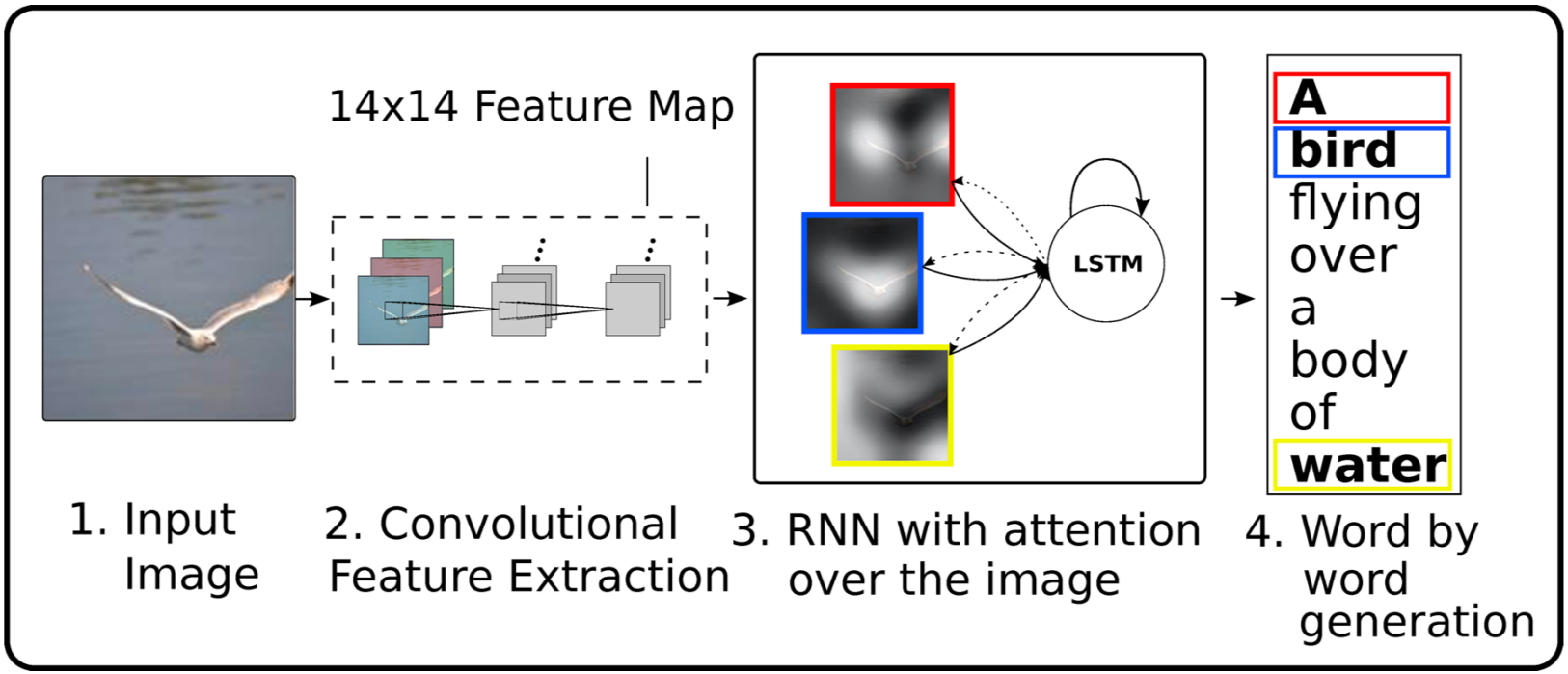


Image captioning

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

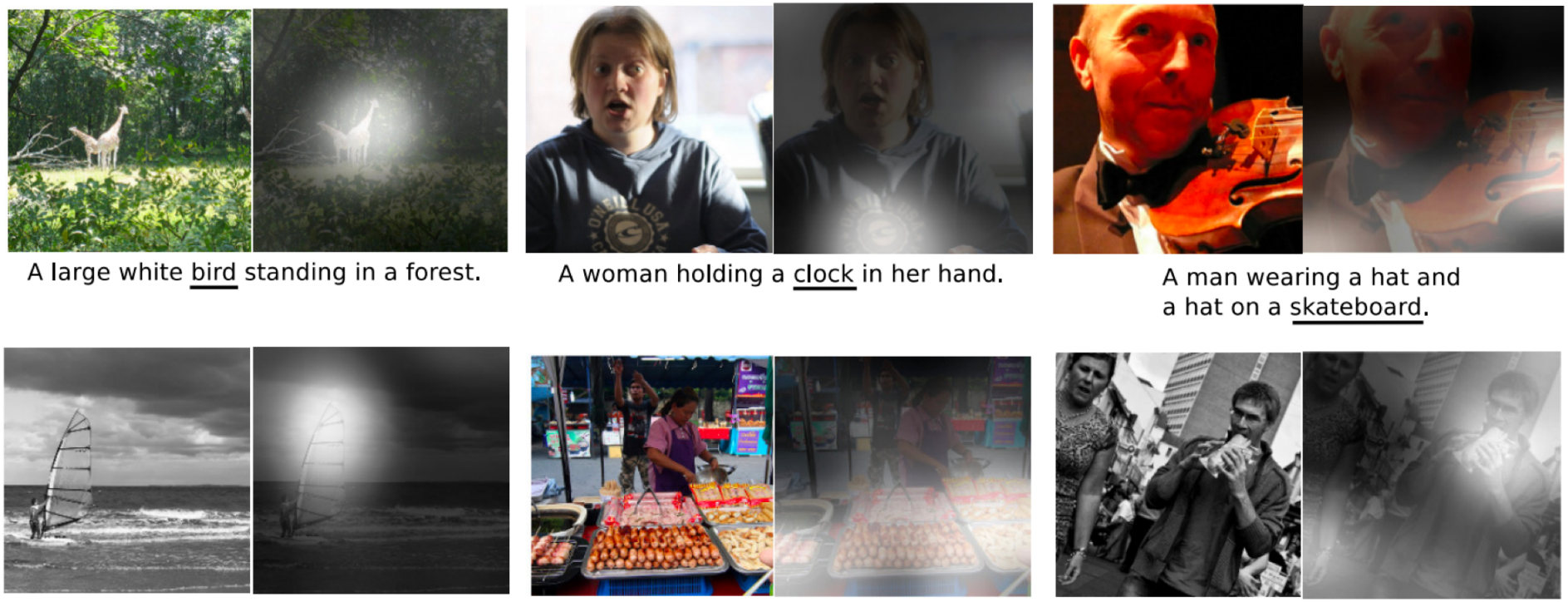
A little **girl** sitting on a bed with a teddy bear.

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

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

Xu et al., (2015)

Image captioning



Xu et al., (2015)

Generating dialogue responses

<speaker A> How are you, Tom? </s>

<speaker B> I'm good, thanks <pause> did you get my message yesterday? </s>

<speaker B nods>

<speaker B> Yes, it was interesting. </s>

<speaker C turns head around>

<speaker C> what message? </s>

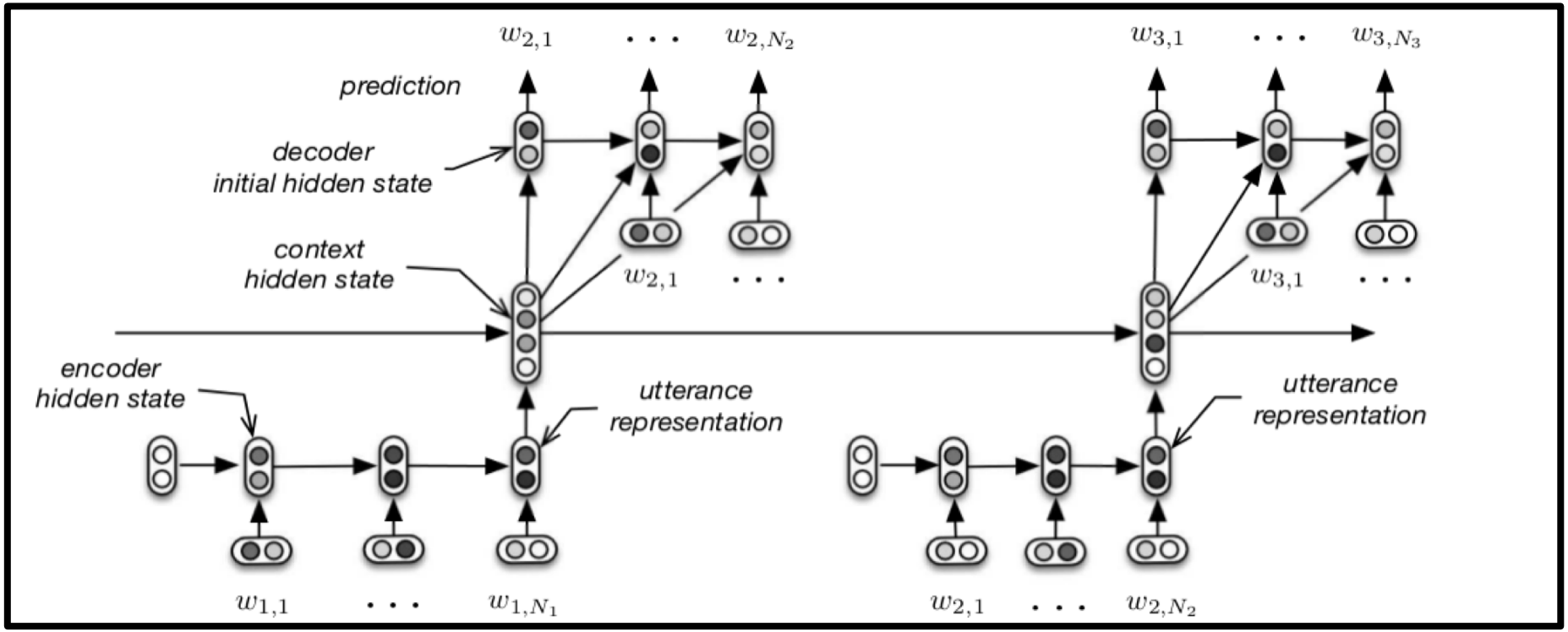
...

Generating dialogue responses

[*Serban, A. Sordoni, Y. Bengio, A. Courville, J. Pineau, AAAI 2015*]

Hierarchical Encoder-Decoder

- Two levels of encoders: encode each utterance + the conversation
- Decode response into natural language



Generating dialogue responses

Context	Response
jamaica blood!! <url> → lol. yes [name]!. [name]! → what's poppin in toronto homie	**VHRED:** nothing much. just chillin. what's good with you? **LSTM:** something!!!!!!! **HRED:** something!!!!!!
whhhhhhyyyy is it suddenly cold in tallahassee today?? this is florida! how can i dream of the beach when my fingers are icey cold?? → sittin on deck in sun looking at lake travis in austin! enjoying day b4 birthday	**VHRED:** the sun is shining!!! **LSTM:** i know!!!!!!!!! **HRED:** something!!!!!!!
reaches for [name]! → - approaches him warily - hey there buddy. - swings him into the air - how you doin? → tilts his head and your wariness and reaches for your face [name].	**VHRED:** - looks up at you and smiles - you're so cute. - kisses your cheek - **LSTM:** - smiles - i'm fine thank you. how are you? **HRED:** - grins, looking at you as i look at you -

[*Serban, Sordoni, Lowe, Charlin, Pineau, Courville, Bengio, AAAI 2017*]

- These are cherry-picked -- **dialogue generation is really hard!!**
- Generate generic things most of the time (e.g. "I don't know")
- **Very hard to evaluate progress automatically**

What you should know

- Recurrent neural networks
- Exploding/ vanishing gradient problems
- Some ways to overcome these problems (e.g. LSTM)
- Generating sequences with RNNs, encoder-decoder
- Examples of successful applications

- From more on Deep Learning, see invited talks at DLSS'16:

  https://sites.google.com/site/deeplearningsummerschool2016/speakers

- Also see ch10 of Deep Learning textbook (Goodfellow et al.)