Lecture 15: Recurrent Neural Networks

Andreas Wichert

Department of Computer Science and Engineering

Técnico Lisboa

Getting targets when modeling sequences

- When applying machine learning to sequences, we often want to turn an input sequence into an output sequence that lives in a different domain.
  - *E. g.* turn a sequence of sound pressures into a sequence of word identities.
- When there is no separate target sequence, we can get a teaching signal by trying to predict the next term in the input sequence.
- The target output sequence is the input sequence with an advance of 1 step.
  - For temporal sequences there is a natural order for the predictions.

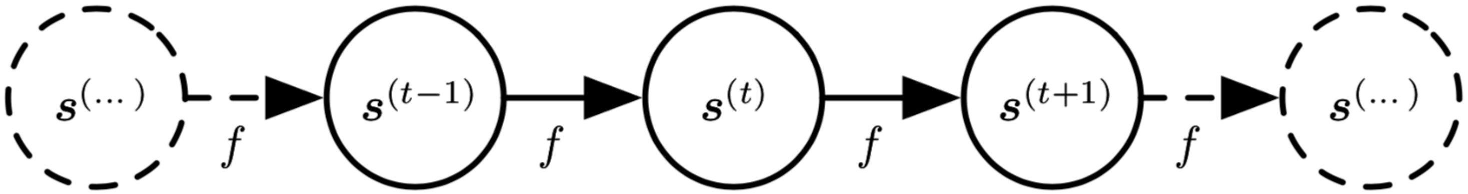
Recurrent Networks

- Recurrent neural networks or RNNs are a family of neural networks for processing sequential data
- Recurrent neural network is a neural network that is specialised for processing a sequence of values $x^{(1)}, \cdots ,x^{(\tau)}$
- Recurrent networks can scale to much longer sequences than would be practical for networks without sequence-based specialisation.
- Most recurrent networks can also process sequences of variable length.

- A traditional fully connected feedforward network would have separate parameters for each input feature
  - It would need to learn for example all of the rules of the language separately at each position in the sentence.
- By comparison, a recurrent neural network shares the same weights across several time steps.
- Each member of the output is a function of the previous members of the output.
- Each member of the output is produced using the same update rule applied to the previous outputs.

Classical Dynamical System

- Unfolding a recursive or recurrent computation into a computational graph that has a repetitive structure, typically corresponding to a chain of events.
- Unfolding this graph results in the sharing of parameters across a deep network structure.



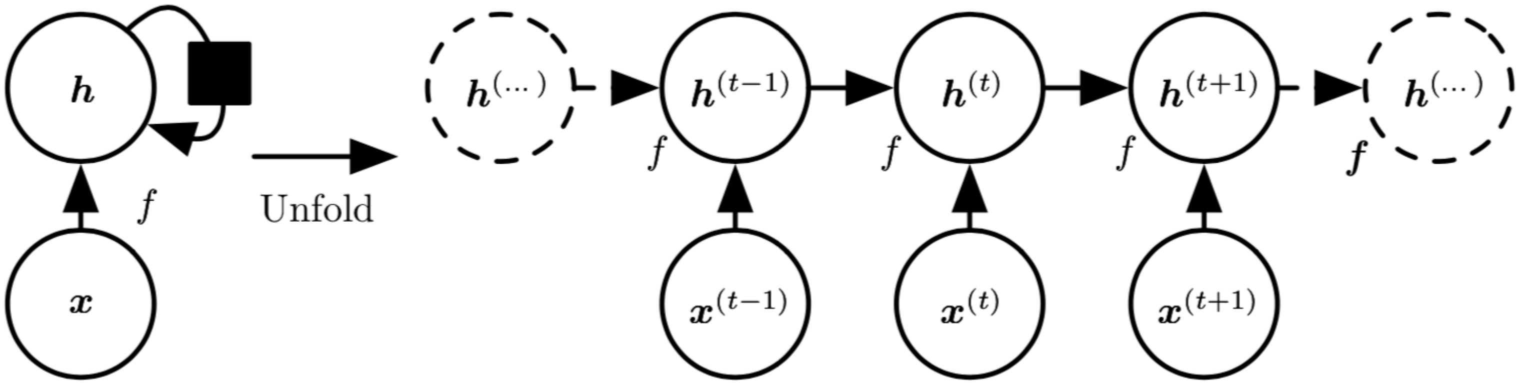
Dynamical system

$$\mathbf{s}^{(t)} = f(\mathbf{s}^{(t-1)}; \mathbf{w}),$$

with $t = 3$

$$\mathbf{s}^{(3)} = f(\mathbf{s}^{(2)}; \mathbf{w}) = f(f(\mathbf{s}^{(1)}; \mathbf{w}); \mathbf{w})$$

Unfolding Computational Graphs



Variable $\mathbf{h}$ represents the state

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}; \mathbf{w}),$$

It maps an arbitrary length sequence

$$(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-2)}, \cdots, \mathbf{x}^{(2)}, \mathbf{x}^{(1)})$$

to a fixed length vector $\mathbf{h}^{(t)}$

Statistical Language Modeling

- For example, if the RNN is used in statistical language modeling, typically to predict the next word given previous words, it may not be necessary to store all of the information in the input sequence up to time *t*,

- .. but rather only enough information to predict the rest of the sentence.

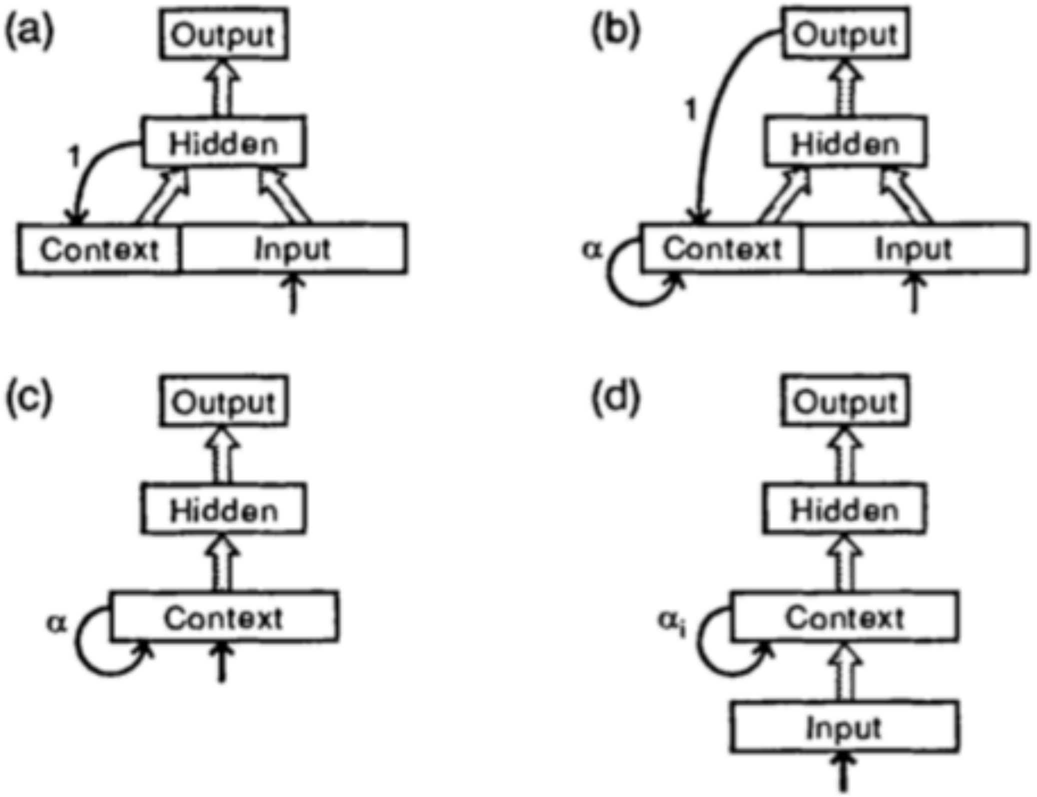
- We can represent the unfolded recurrence after *t* steps with a function g^t

$$\boldsymbol{h}^{(t)} = g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \cdot \cdot \cdot , \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{w})$$

- Regardless of the sequence length, the learned model always has the same input size, because it is specified in terms of transition from one state to another state, rather than specified in terms of a variable-length history of states.

- It is possible to use the same transition function f with the same parameters at every time step

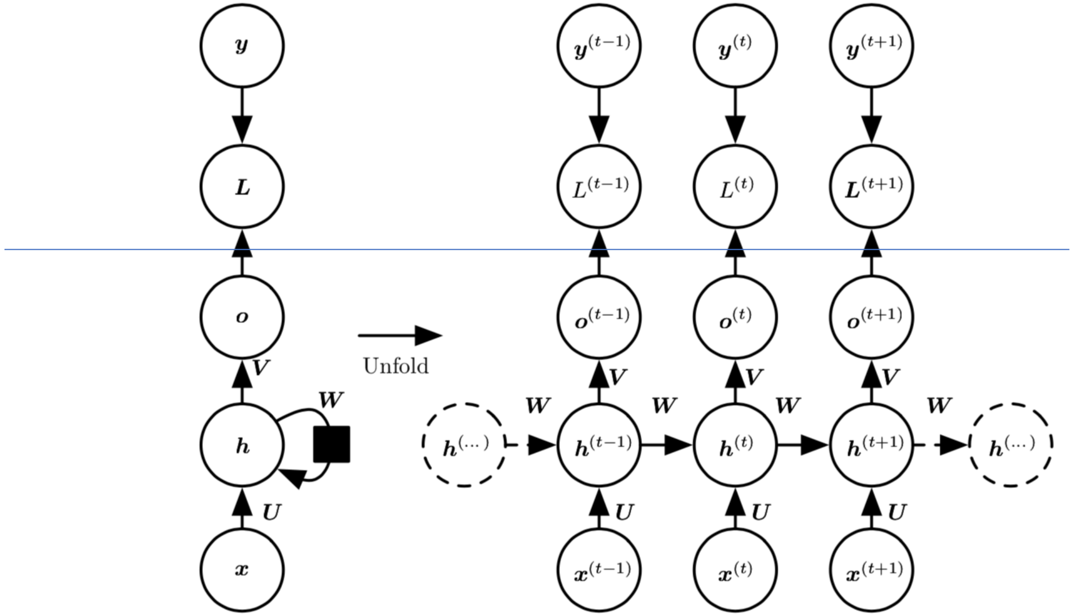
- These two factors make it possible to learn a single model *f* that operates on all time steps and all sequence lengths, rather than needing to learn a separate model $g^{(t)}$ for all possible time steps.



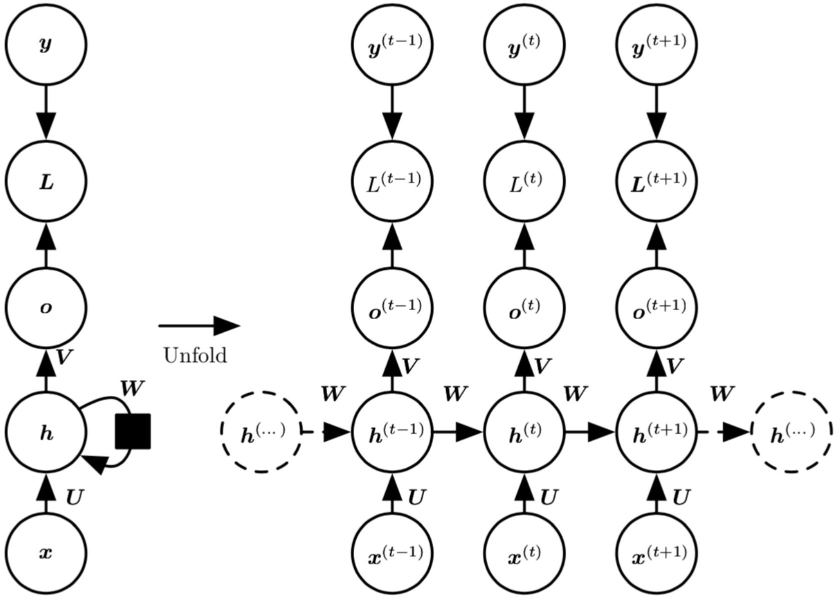
(a) Elman Network, (b) Jordan Network

Recurrent Neural Networks

- Recurrent networks that produce an output at each time step and have recurrent connections between hidden units, Vanilla RNN or Elman RNN



Recurrent Hidden Units



With specified

$$\mathbf{h}^{(0)} = initial\ state$$

For each time step $t = 1$ to $t = \tau$

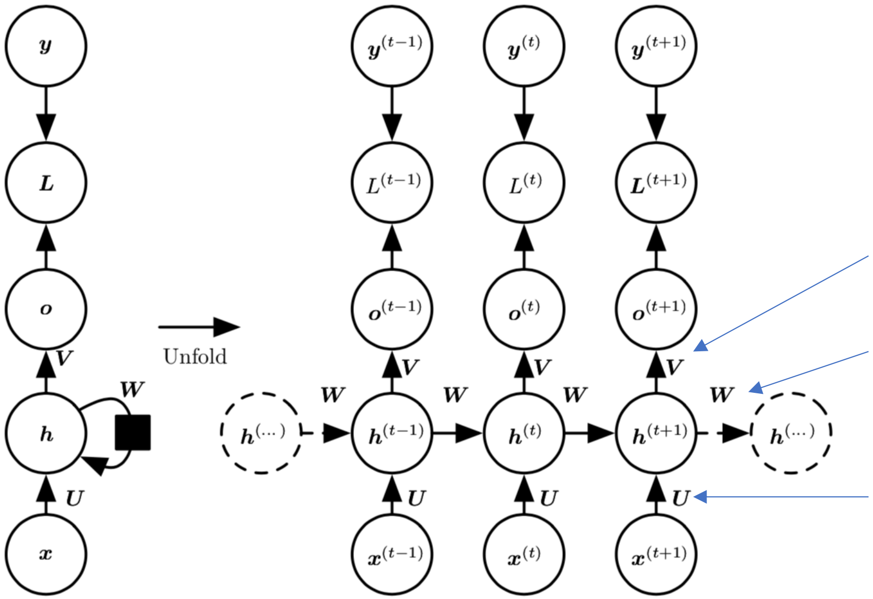
$$\mathbf{a}^{(t)} = \mathbf{b} + W \cdot \mathbf{h}^{(t-1)} + U \cdot \mathbf{x}^{(t)}$$

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

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

with $\mathbf{b}$ and $\mathbf{c}$ being the bias vectors and U, V, W the weight matrices

Recurrent Hidden Units

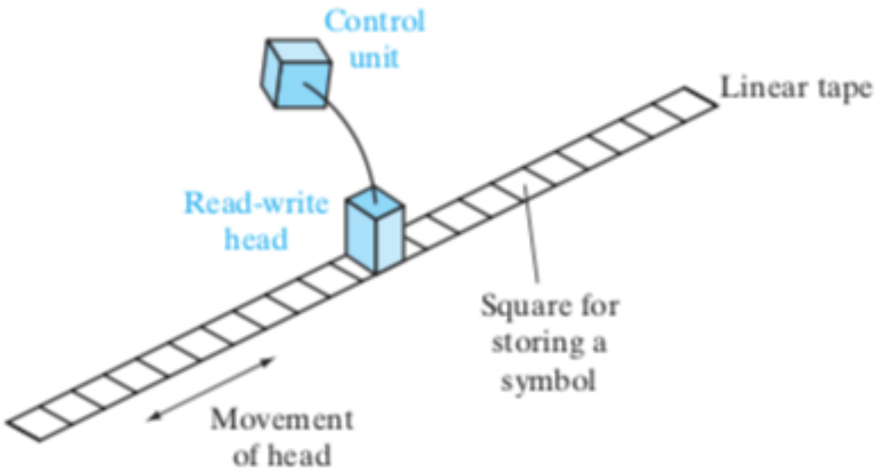


V weight matrix hidden to output layer

W weight matrix hidden to hidden layer *(new)*. Input is the preceding state.

U weight matrix input to hidden layer

Computational Power



- The recurrent neural network is universal in the sense that any function computable by a Turing machine can be computed by such a recurrent network of a finite size.

Eduardo Daniel Sontag is an American mathematician, and Distinguished University Professor at Northeastern University,

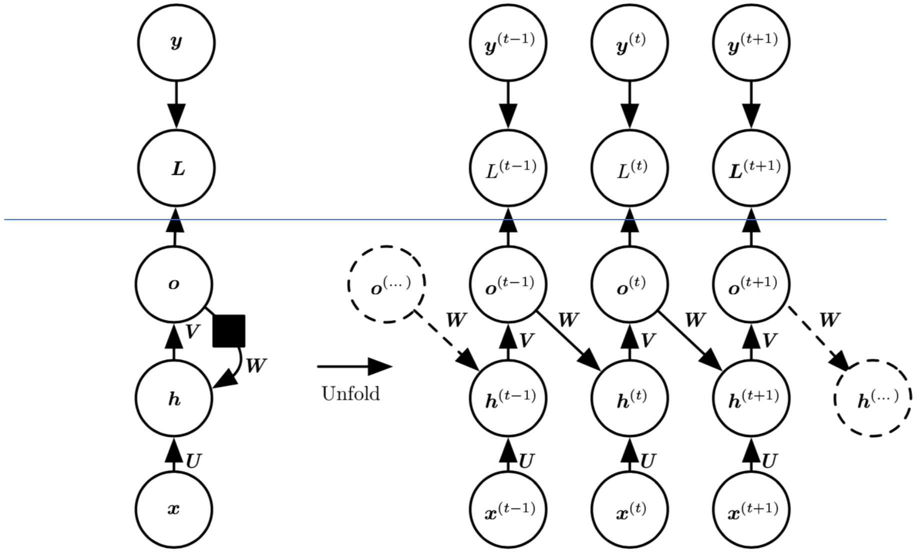
Hava Siegelmann is a professor of computer science, and a world leader in the fields of Lifelong Learning, Artificial Intelligence, Machine Learning, Neural Networks, and Computational Neuroscience.

- Siegelmann and Sontag, 1991

Theorem:

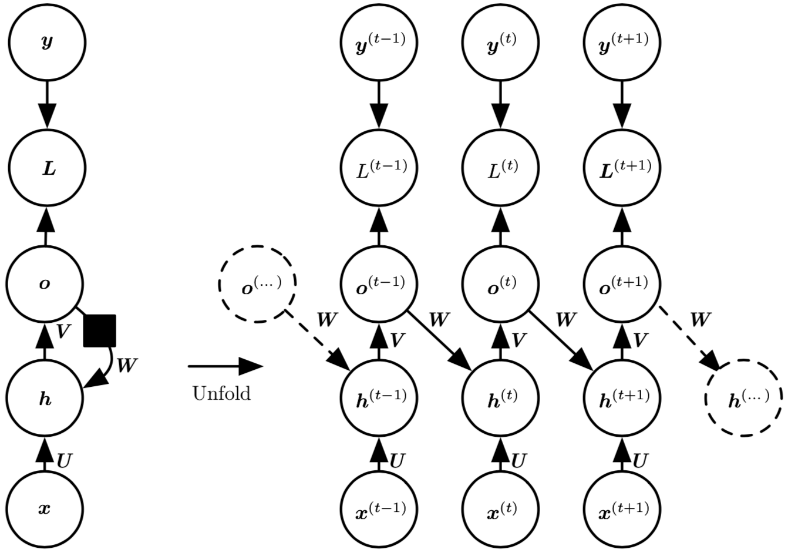
- *All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoidal activation functions.*

Recurrence through only the Output



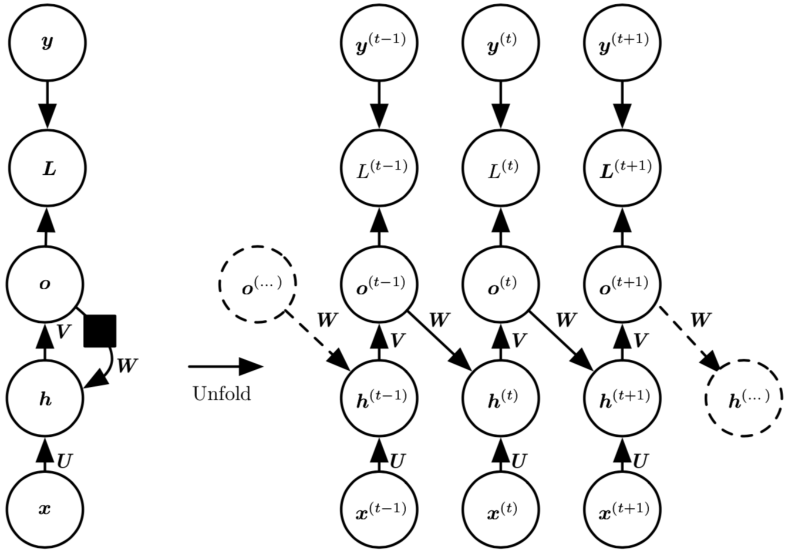
- Recurrent networks that produce an output at each time step and have recurrent connections only from the output at one time step to the hidden units at the next time step

Recurrence through only the Output



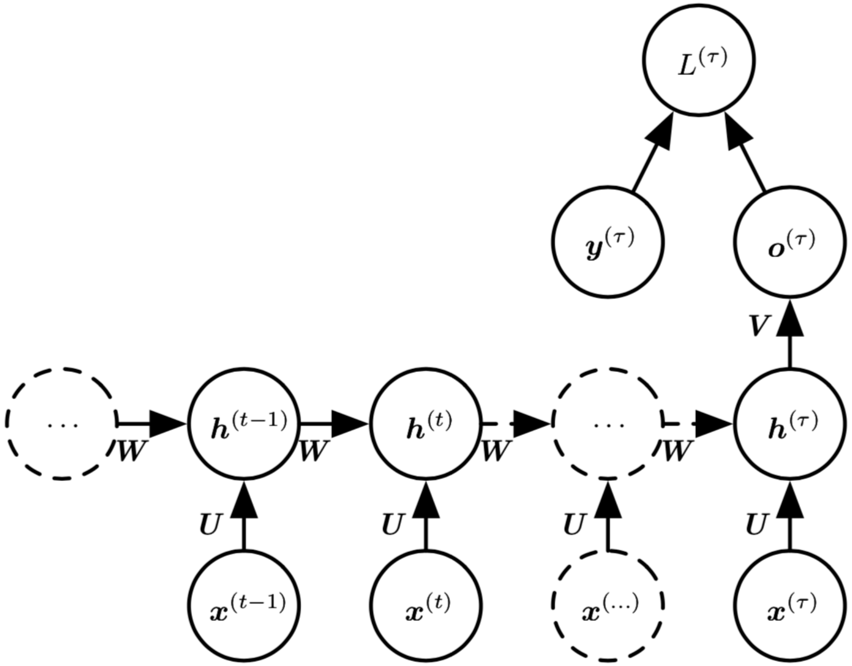
- Less powerful because it lacks hidden-to-hidden recurrent connections.
- For example, it cannot simulate a universal Turing machine.

Recurrence through only the Output



- It requires that the output units capture all of the information about the past that the network will use to predict the future.

Sequence Input, Single Output



Recurrent networks with recurrent connections between hidden units, that read an entire sequence and then produce a single output

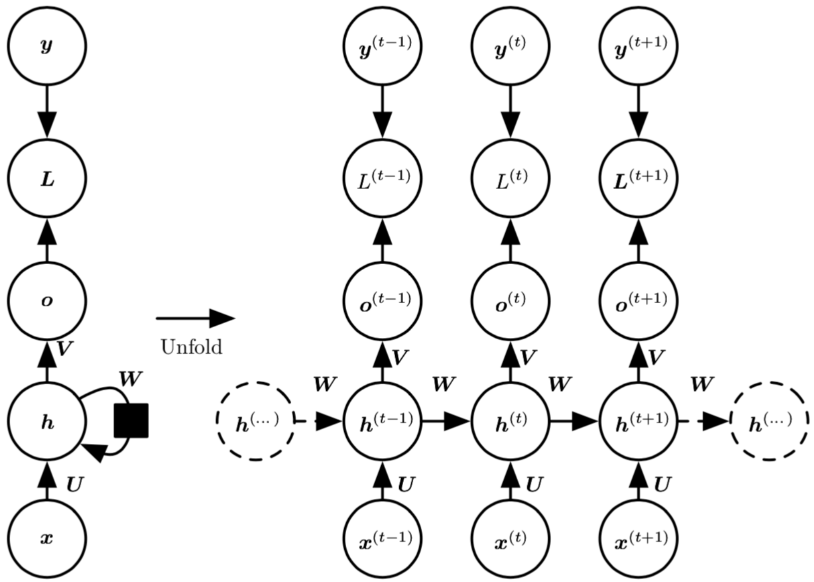
Backpropagation Trough Time

- The back-propagation algorithm applied to the unrolled graph with *O(τ)* cost is called back-propagation through time
- The training data for a recurrent neural network is an ordered sequence of τ input-output pairs
- Same idea as training over training set *D*, now the training set is *D · τ* with an additional input is the preceding state.

Backpropagation Trough Time

- The runtime is *O(τ)* and cannot be reduced by parallelization because the forward propagation graph is inherently sequential; each time step may only be computed after the previous one.
- BPTT begins by unfolding a recurrent neural network in time. The unfolded network contains *τ* inputs and outputs, but every copy of the network shares the **same** parameters.
- Then the backpropagation algorithm is used to find the gradient of the cost with respect to all the network parameters

Backpropagation Trough Time is THE usual Backpropagation!



With specified

$$\mathbf{h}^{(0)} = initial\ state$$

For each time step $t = 1$ to $t = \tau$

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

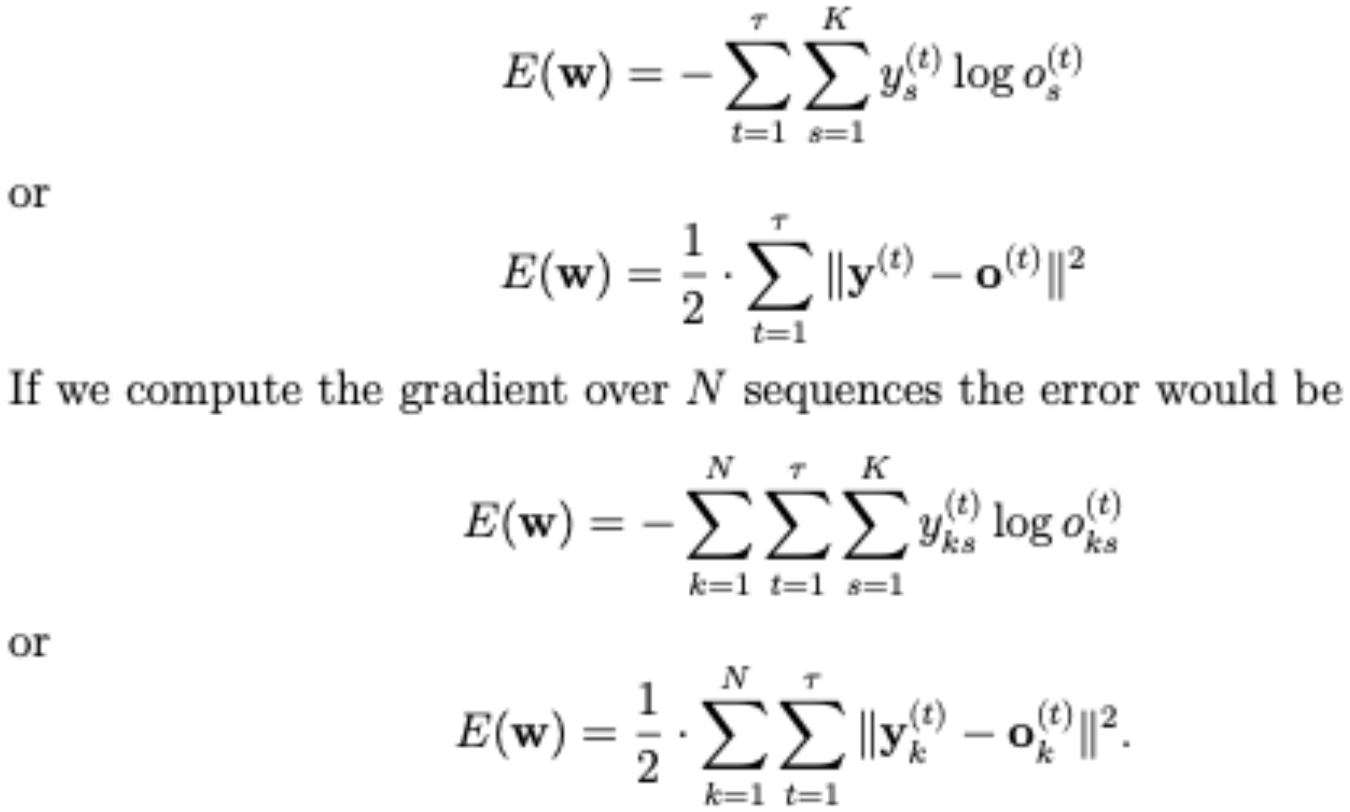
$$\mathbf{h}^{(t)} = tanh(\mathbf{a}^{(t)})$$

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

with $\mathbf{b}$ and $\mathbf{c}$ being the bias vectors and U, V, W the weight matrices

- The back-propagation algorithm applied to the unrolled graph with *O(τ)* cost is called back-propagation through time

Backpropagation Trough Time



$$E(\mathbf{w}) = -\sum_{t=1}^{\tau}\sum_{s=1}^{K} y_s^{(t)} \log o_s^{(t)}$$

or

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{t=1}^{\tau} \|\mathbf{y}^{(t)} - \mathbf{o}^{(t)}\|^2$$

If we compute the gradient over N sequences the error would be

$$E(\mathbf{w}) = -\sum_{k=1}^{N}\sum_{t=1}^{\tau}\sum_{s=1}^{K} y_{ks}^{(t)} \log o_{ks}^{(t)}$$

or

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N}\sum_{t=1}^{\tau} \|\mathbf{y}_k^{(t)} - \mathbf{o}_k^{(t)}\|^2.$$

The Challenge

- BPTT has difficulty with local optima.
- With recurrent neural networks, local optima are a much more significant problem than with feed-forward neural networks
- The recurrent feedback in such networks tends to create chaotic responses in the error surface which cause local optima to occur frequently, and in poor locations on the error surface.
- The basic problem is that gradients propagated over many stages tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization).

The Challenge

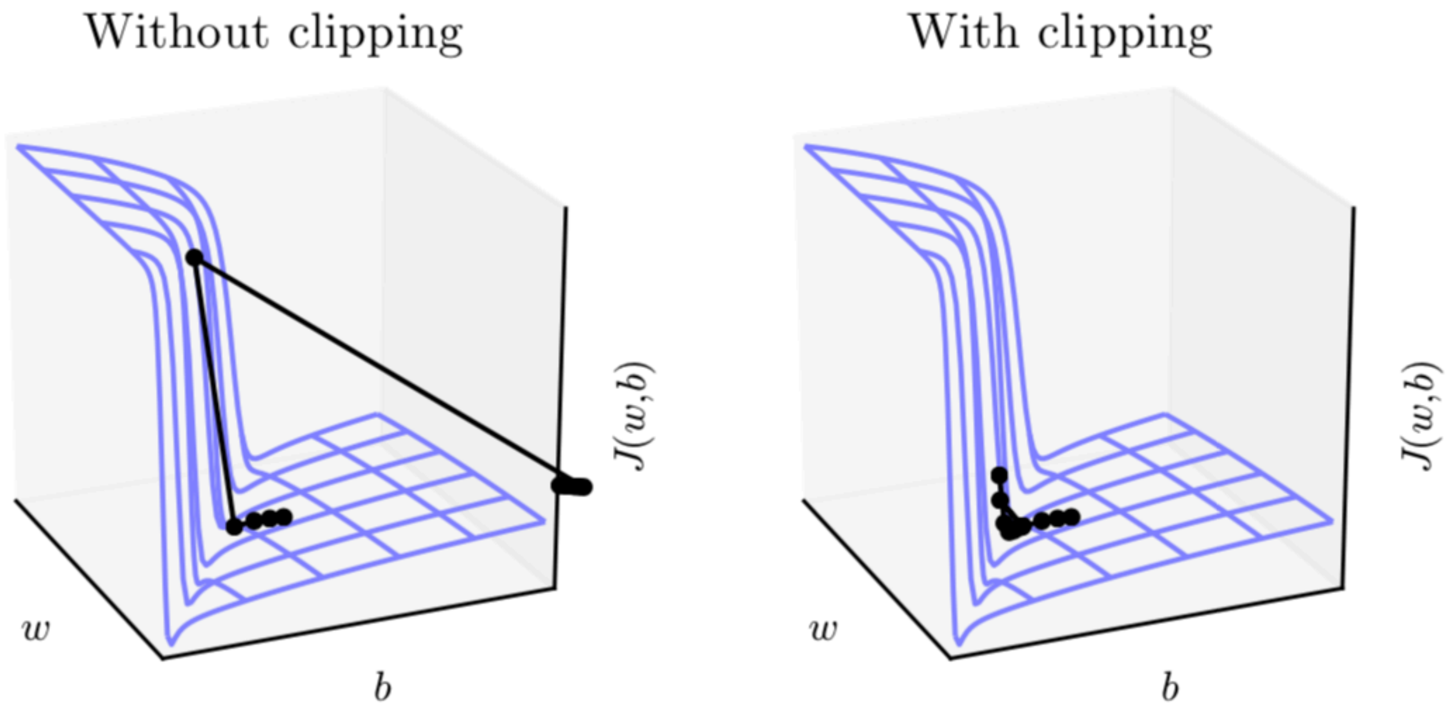
- Even if we assume that the parameters are such that the recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions

- Clip the norm ***//g//*** of the gradient ***g*** with *v* is the norm threshold. The gradient is normalised and multiplied with the scalar *v*

Gradient Clipping

$$\text{if } ||\boldsymbol{g}|| > v$$

$$\boldsymbol{g} \leftarrow \frac{\boldsymbol{g}v}{||\boldsymbol{g}||}$$

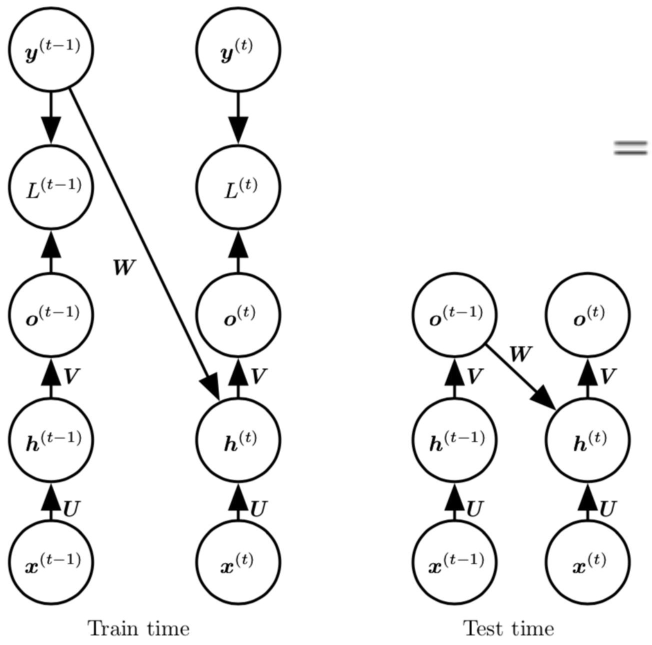


Without clipping

With clipping

Teacher Forcing and Networks with Output Recurrence

- Teacher Forcing is applicable to RNNs that have connections from their out-put to their hidden states at the next time step



$$\begin{aligned}&\log p\left(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right) \\ =&\log p\left(\boldsymbol{y}^{(2)} \mid \boldsymbol{y}^{(1)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right)+\log p\left(\boldsymbol{y}^{(1)} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right)\end{aligned}$$

Deep Recurrent Networks

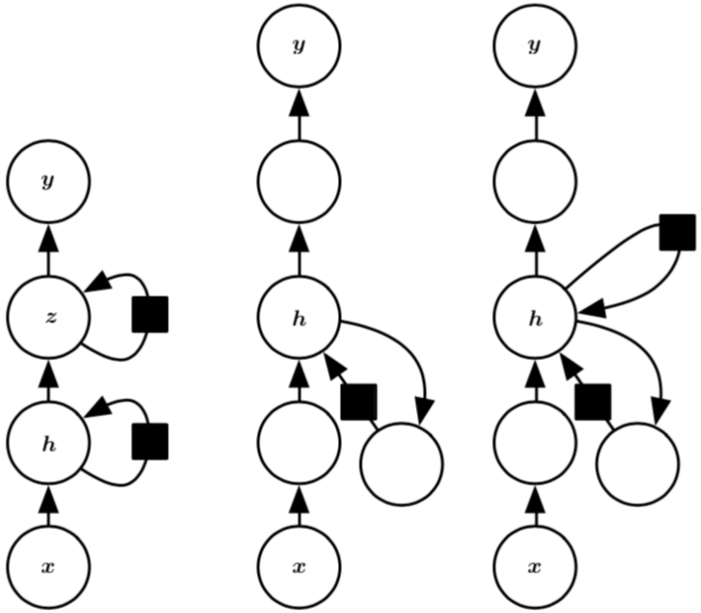
- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:
  - from the input to the hidden state,
  - from the previous hidden state to the next hidden state,
  - from the hidden state to the output.

- Would it be advantageous to introduce depth in each of these operations?
  - *The experimental evidence is in agreement with the idea that we need enough depth in order to perform the required mappings.*

Deep RNNs

A recurrent neural network can be made deep in many ways

- The hidden recurrent state can be broken down into groups organized hierarchically.
- Deeper computation (e.g., an MLP) can be introduced in the input-to-hidden, hidden-to-hidden and hidden-to-output parts
- This may lengthen the shortest path linking different time steps
  The path-lengthening effect can be mitigated by introducing skip connec- tions.



Long Short Term Memory (LSTM)

- The name should not be confused with the psychological long and short term memory models of human problem solving [Anderson (1995)].
- LSTMs are composed of several gates that allow the gradients to flow unchanged.
- They solve partially the vanishing gradient problem, however they can still suffer from the exploding gradient problem.
- There are three types of gates, forget gate, input gate and the output gate. Each gate has its own weight matrices and bias vector parameters which needs to be learned during training

Long Short Term Memory (LSTM)

- A gate is described by a sigmoid activation function and a Hadamard product (point wise multiplication operation)
- The output of a sigmoid activation function is a number between zero and one, describing how much of each component should be let through
- A value of zero means no information is let through, while a value of one indicates that all information is let through.

Long Short Term Memory (LSTM)

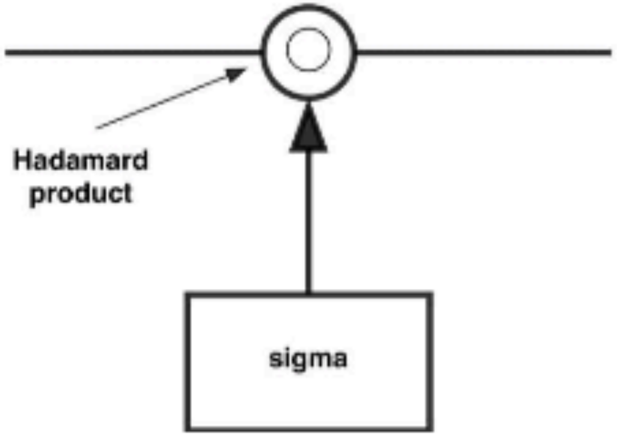


Fig. 14.10 A gate is described by a sigmoid activation function and a Hadamard product. The output of a sigmoid activation function is a number between zero and one, describing how much of each component should be let through. A value of zero means no information is let through, whereas a value of one indicates that all information is let through.

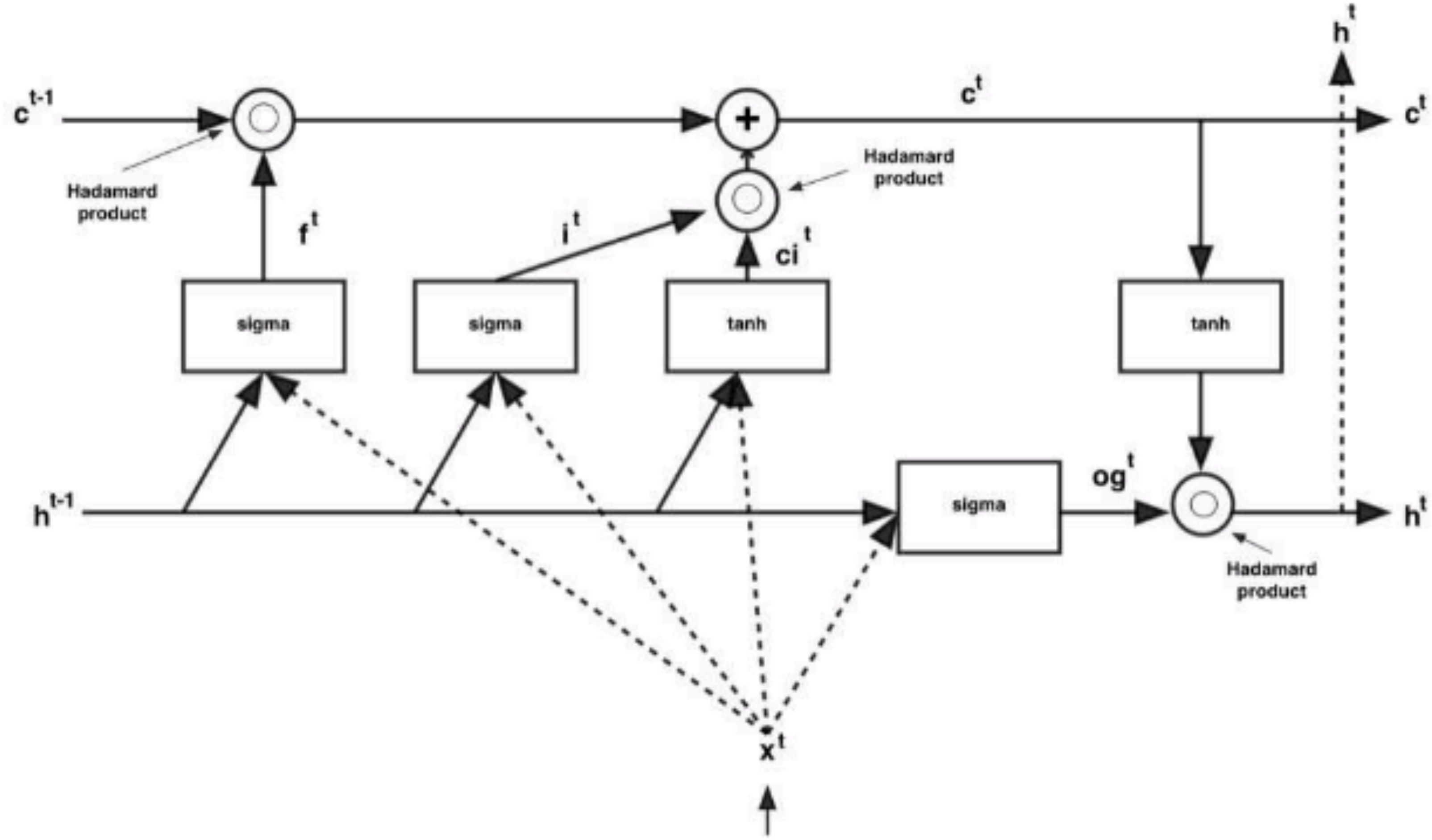
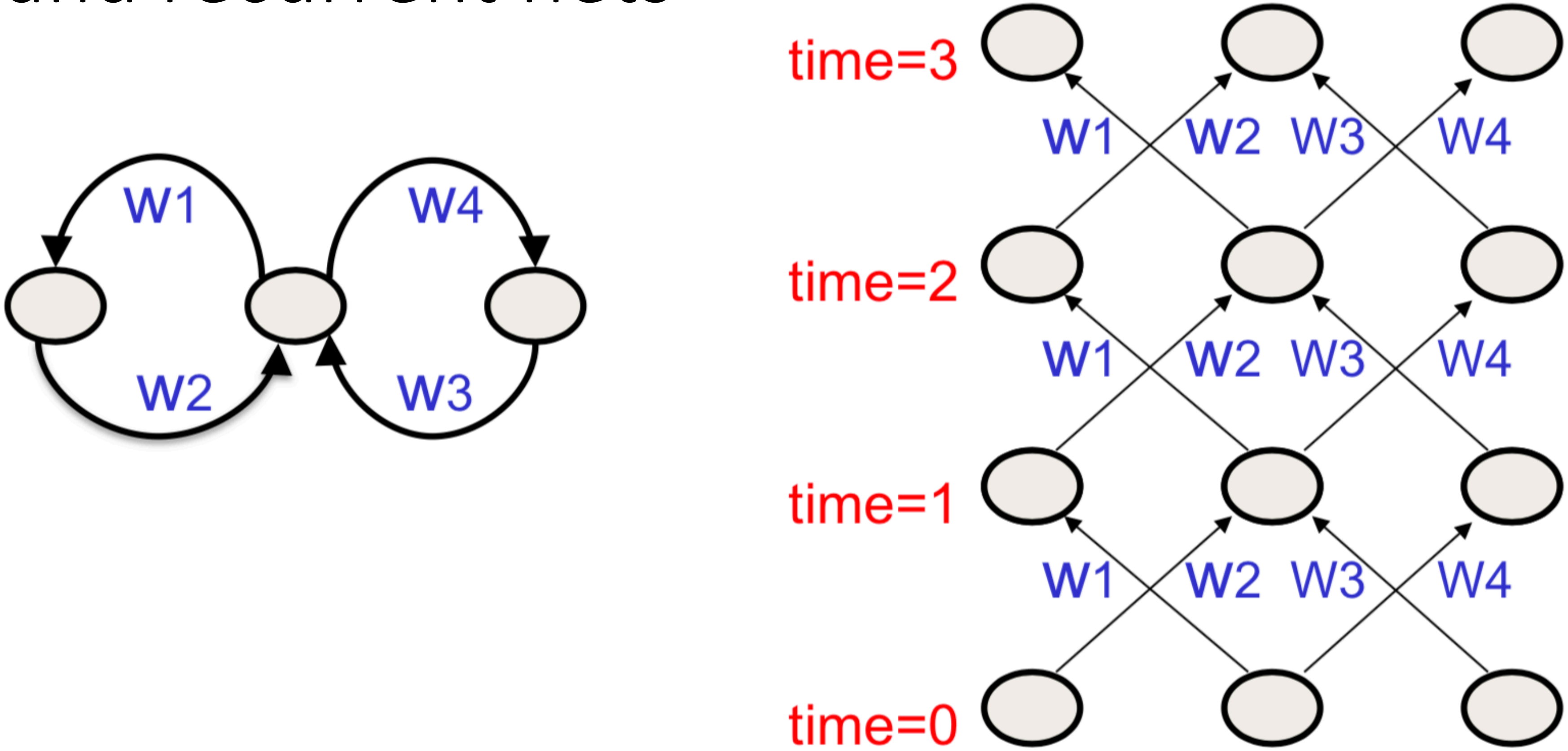


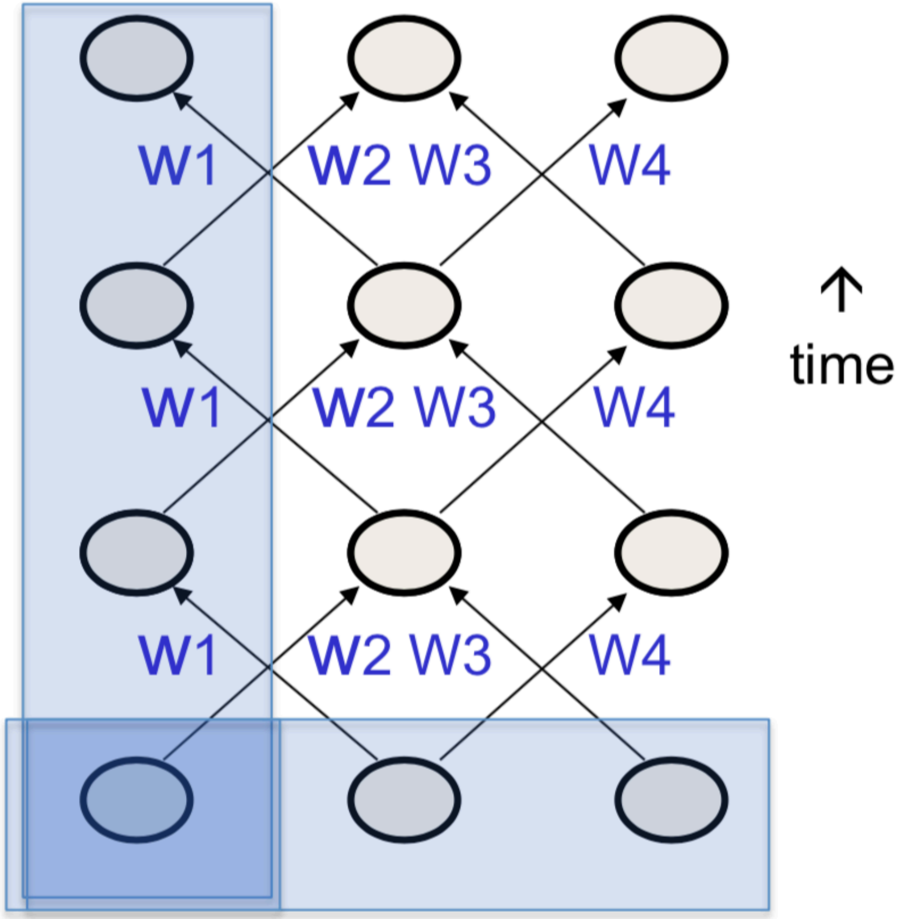
Fig. 14.11 Architecture of the Long Short-Term Memory (LSTM) and its gate that processes the data sequentially and keep its hidden state through time.

The equivalence between feedforward nets and recurrent nets



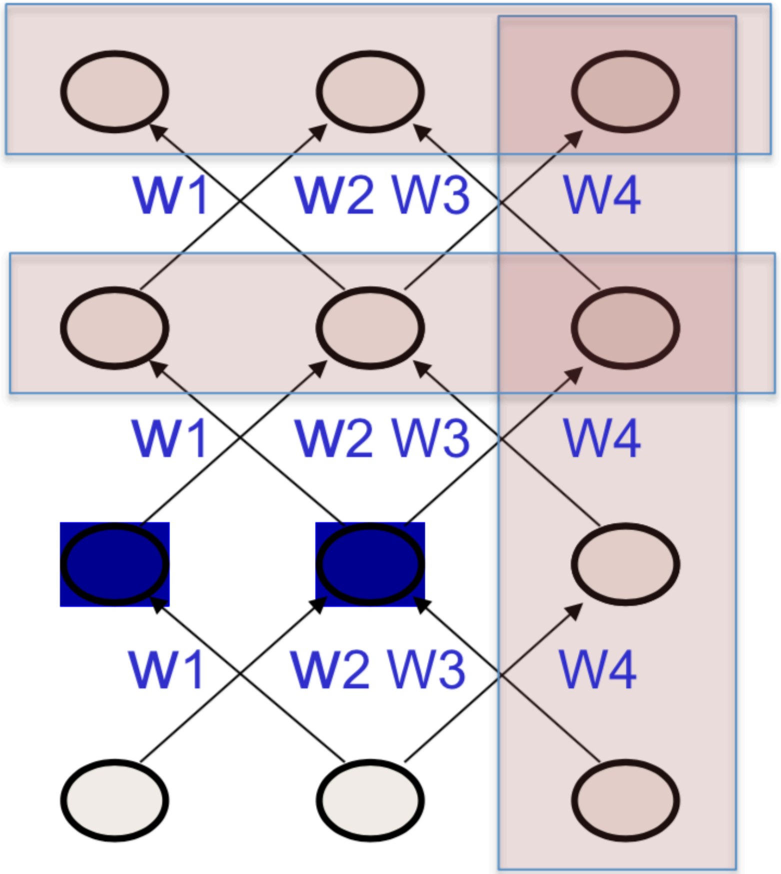
Providing input to recurrent networks

- We can specify inputs in several ways:
  - Specify the initial states of all the units.
  - Specify the initial states of a subset of the units.
  - Specify the states of the same subset of the units at every time step.
- This is the natural way to model most sequential data.

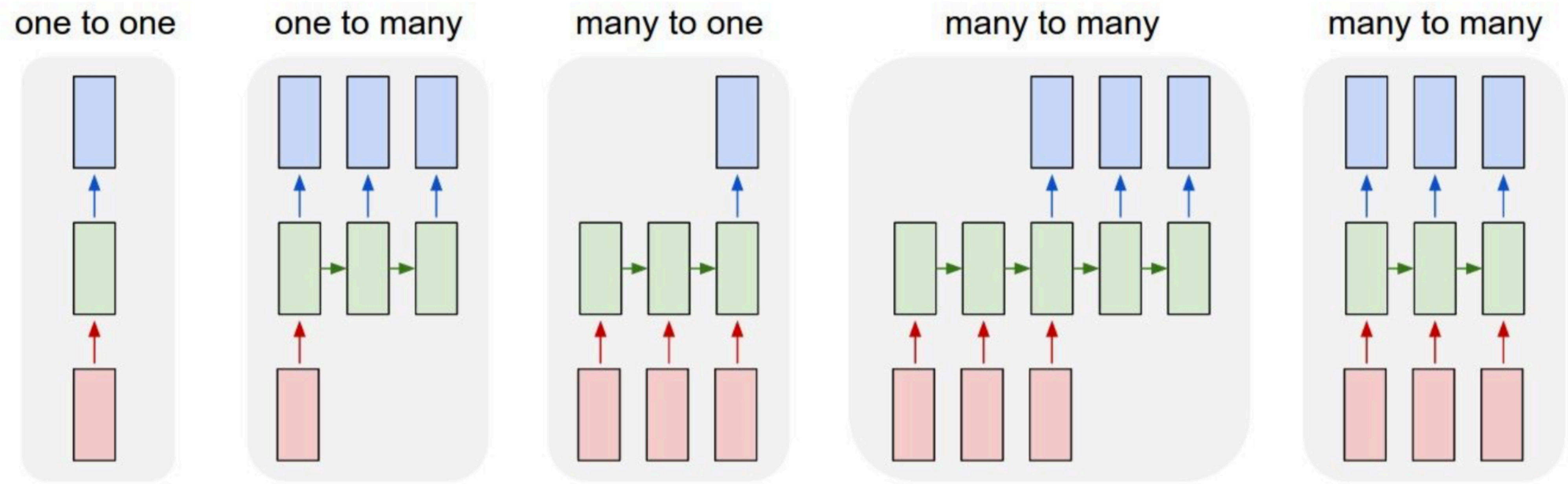


Teaching signals for recurrent networks

- We can specify targets in several ways:
  - Specify desired final activities of all the units
  - Specify desired activities of all units for the last few steps
    - Good for learning attractors
    - It is easy to add in extra error derivatives as we backpropagate.
  - Specify the desired activity of a subset of the units.
- The other units are input or hidden units.



Recurrent Neural Networks: Process Sequences



Vanilla
Neural
Networks

e.g. Image
Captioning
image ->
sequence of
words

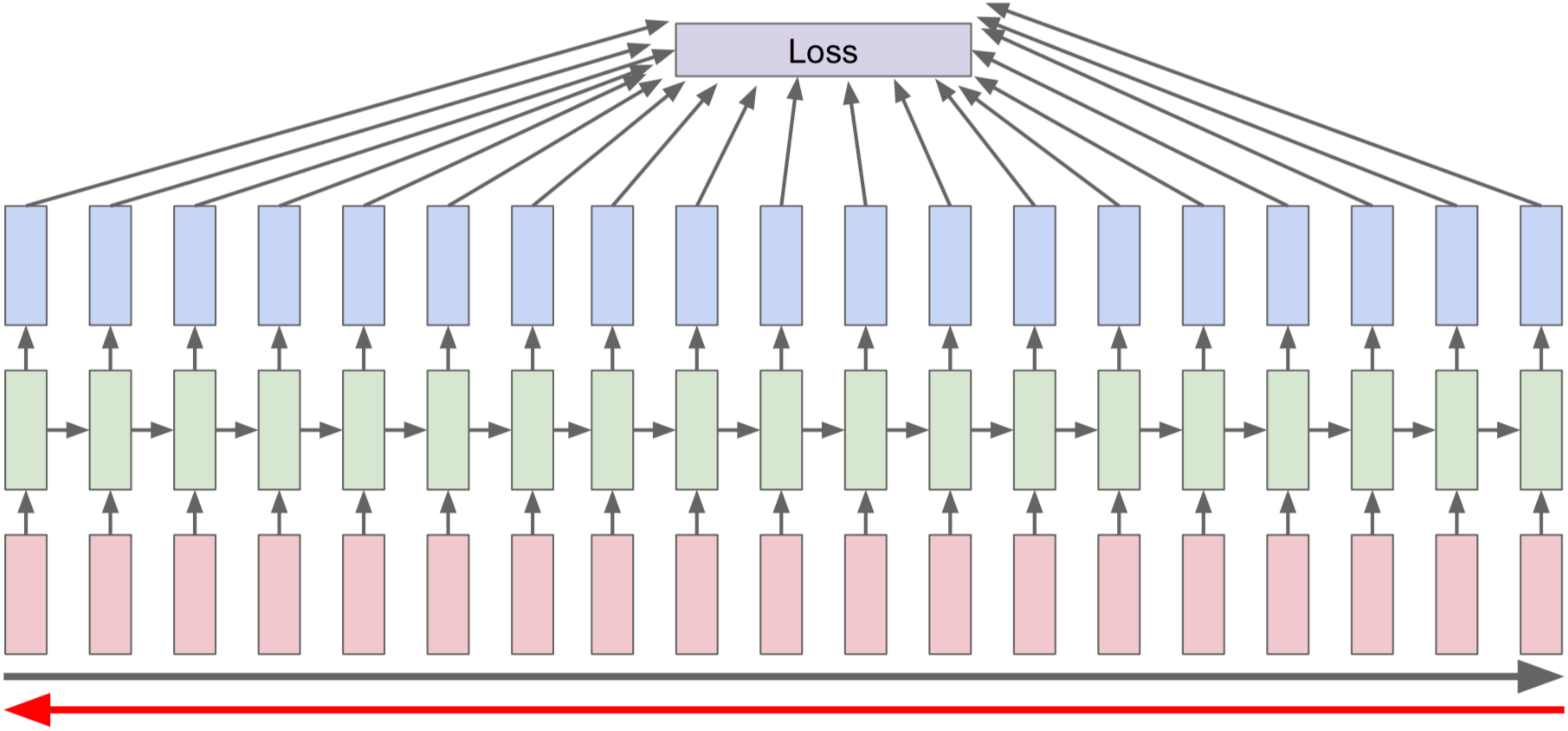
e.g. Sentiment
Classification
sequence of
words ->
sentiment

e.g. Machine
Translation
seq of words
-> seq of
words

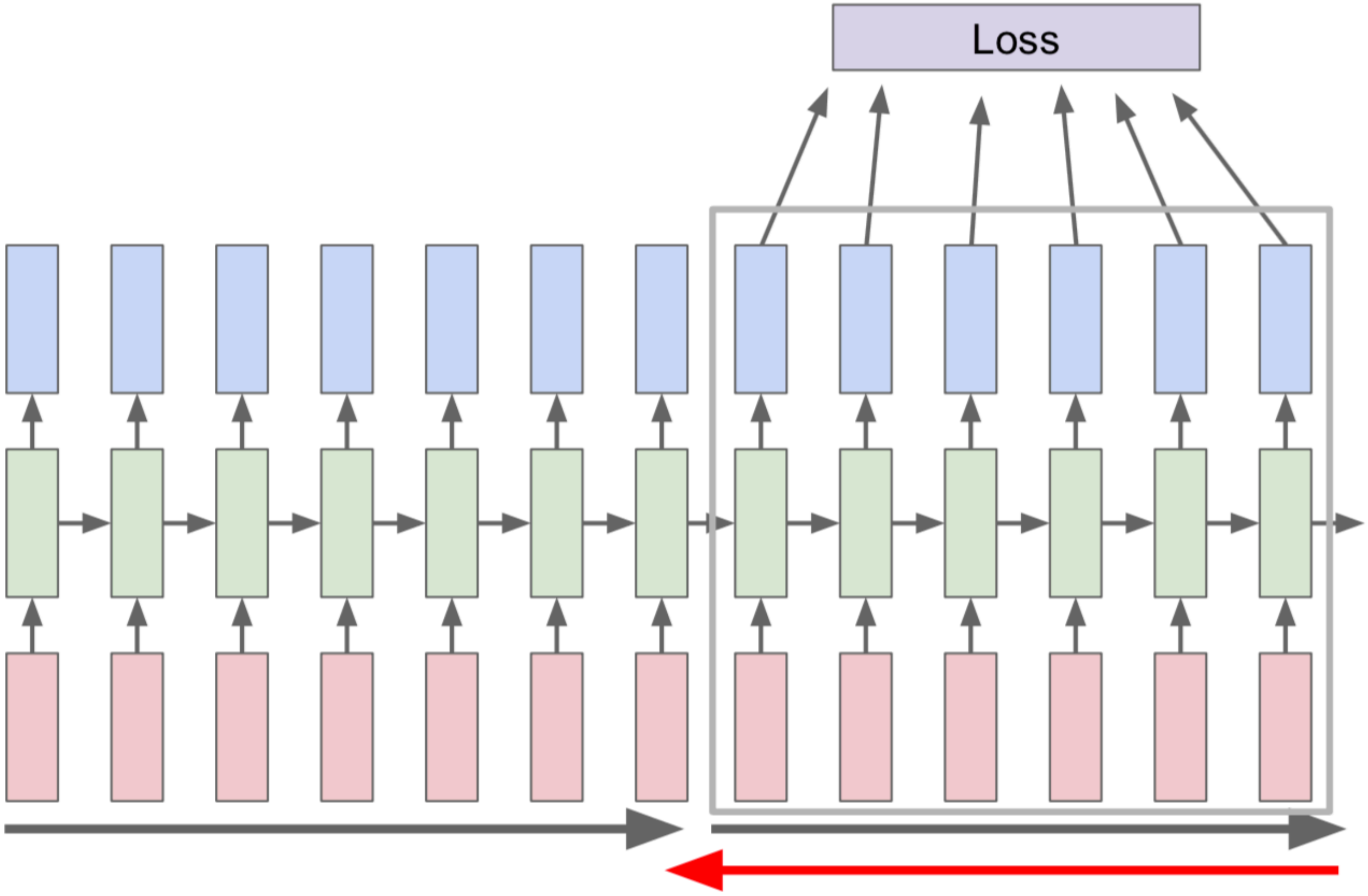
e.g. Video
classification
on frame
level

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

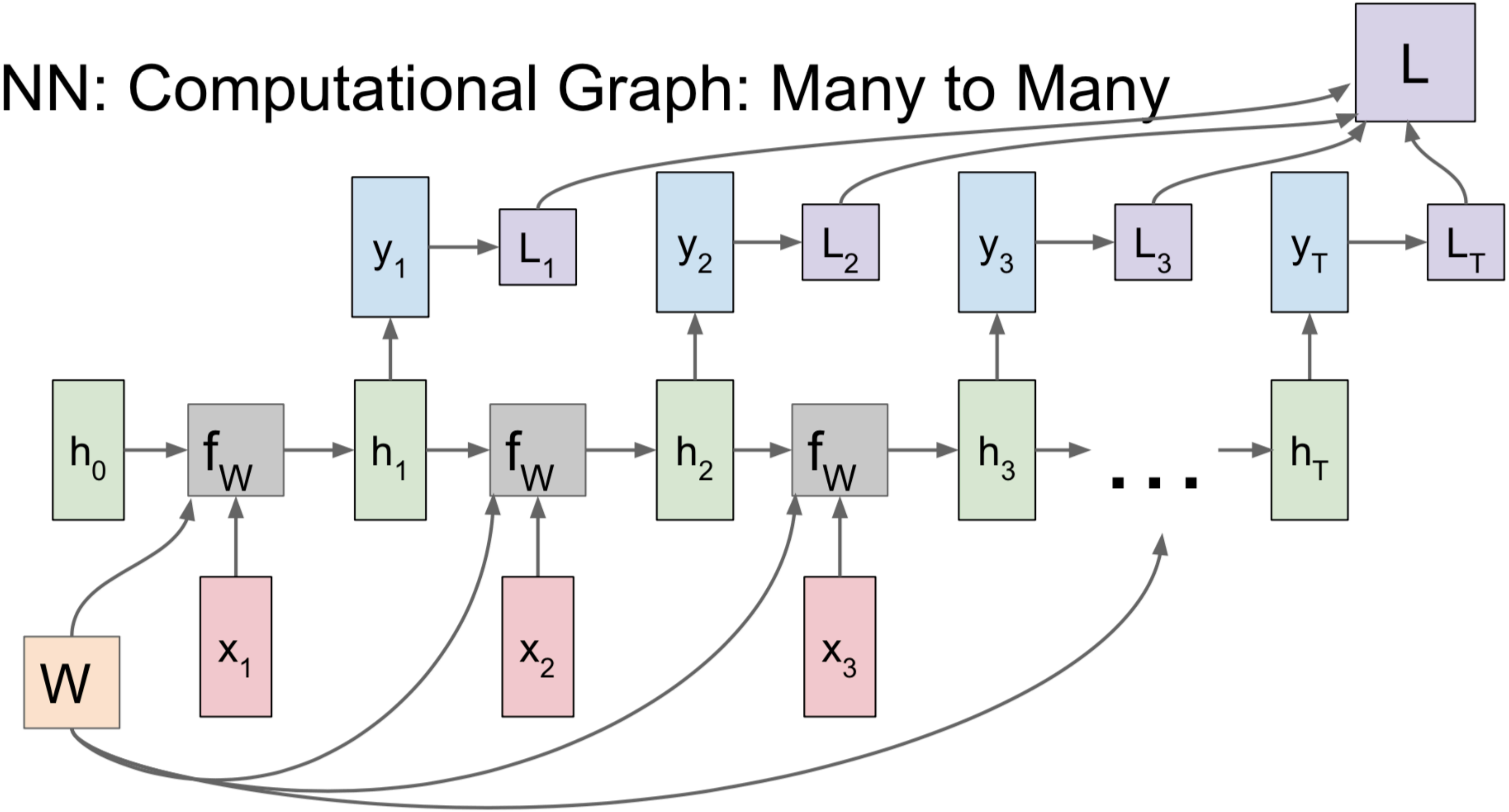


Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

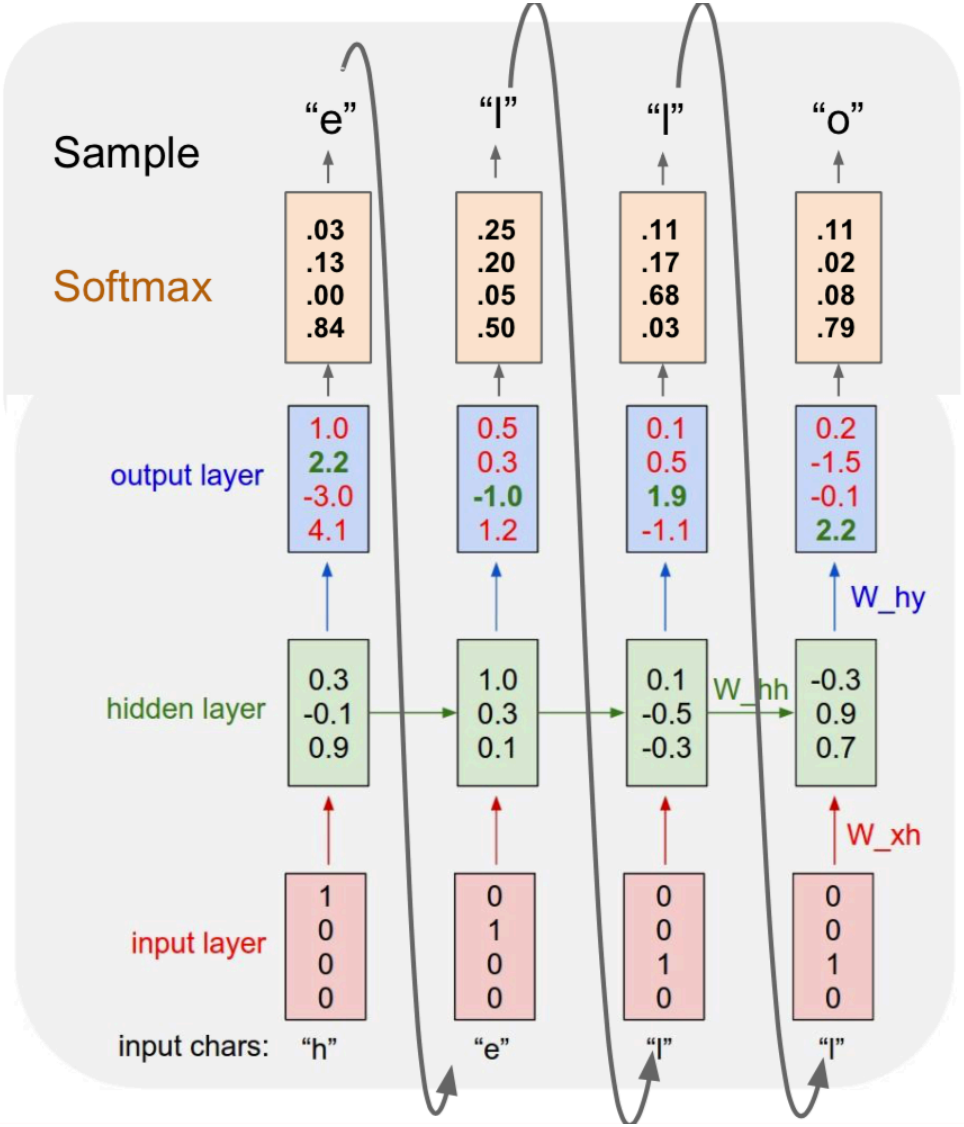
RNN: Computational Graph: Many to Many



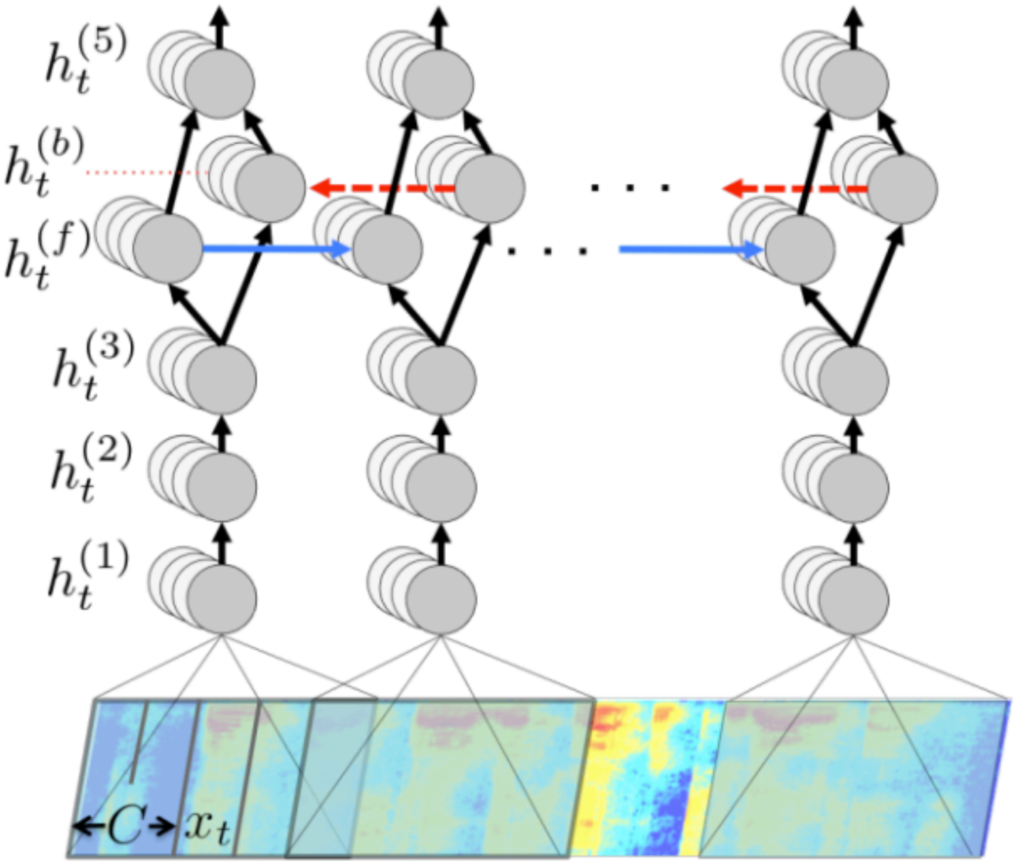
Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

At test-time sample characters one at a time, feed back to model

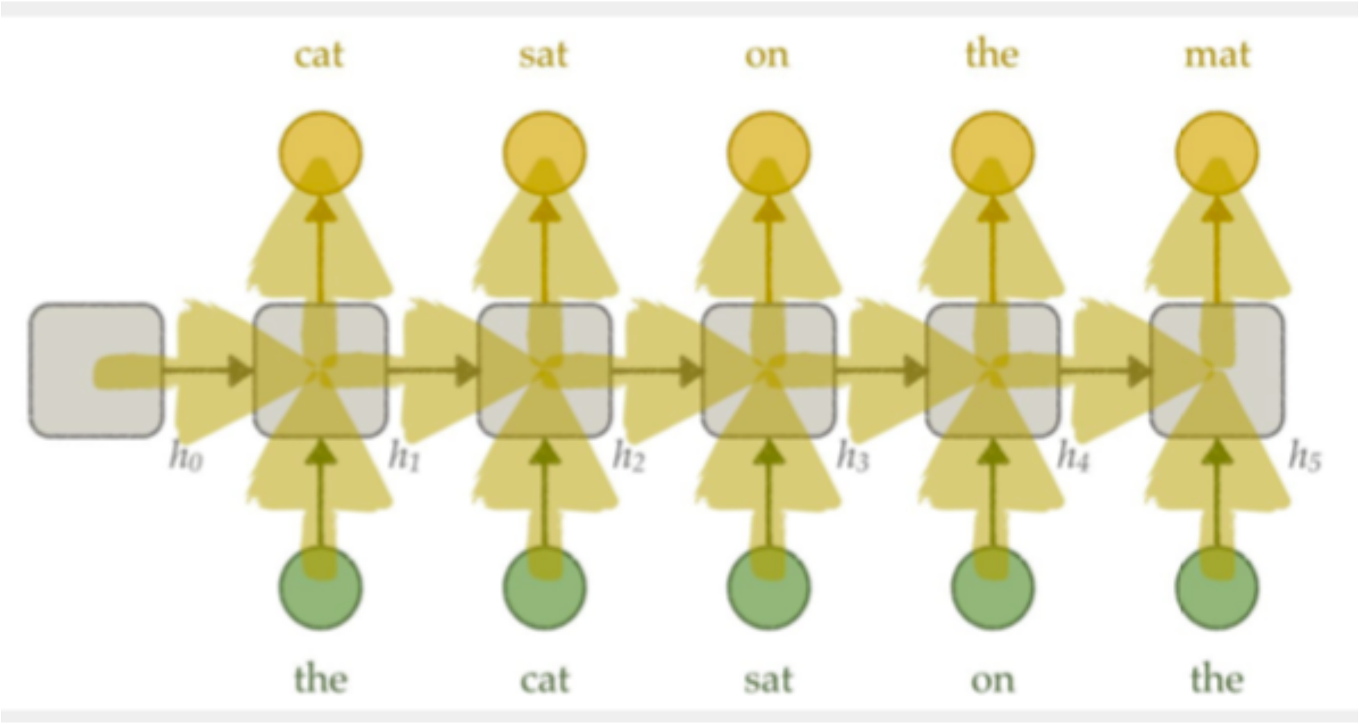


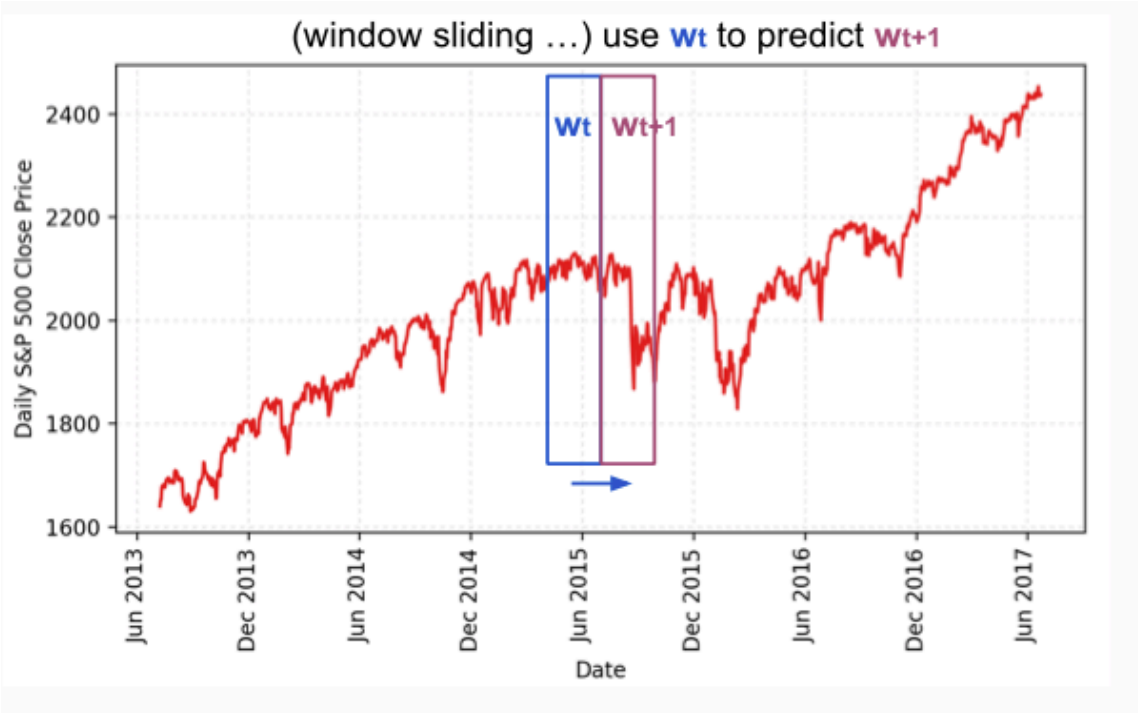
Deep Speech: Accurate Speech Recognition



Bryan Catanzaro , 2015

Neural embedding

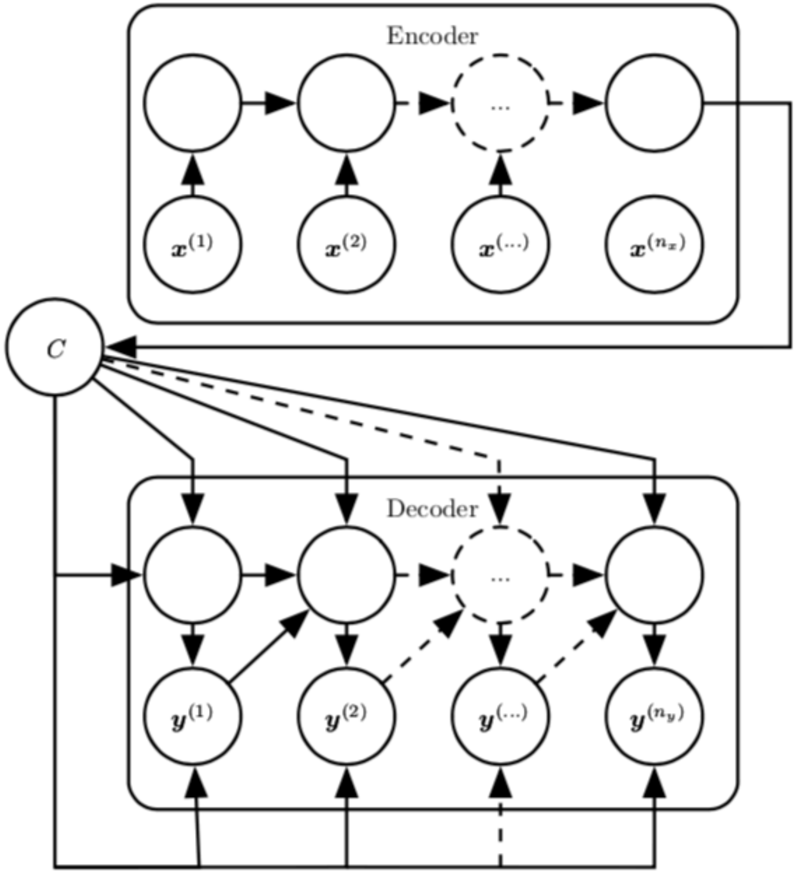




The S&P 500 prices in time

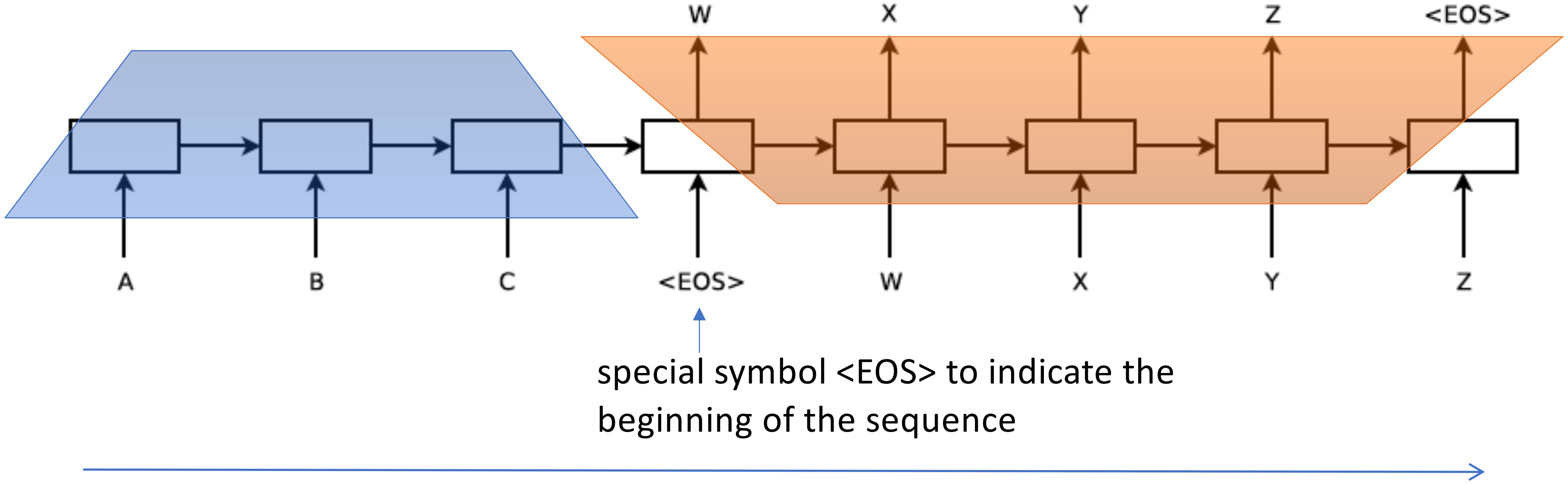
Sequence to Sequence

- For learning to generate an output sequence *$(y^{(1)},\ldots,y^{(ny)})$* given an input sequence *$(x^{(1)},x^{(2)},\ldots,x^{(nx)})$*
- It is composed of an encoder RNN that reads the input sequence decoder RNN that generates the output sequence
- The final hidden state of the encoder RNN is used to compute a generally fixed-size context variable ***C*** which represents a semantic summary of the input sequence and is given as input to the decoder RNN.



Neural Machine Translation

- Sequence to Sequence Learning with Neural Networks
  - Concatenate the source and target sequence
  - Only make prediction on the target sequence

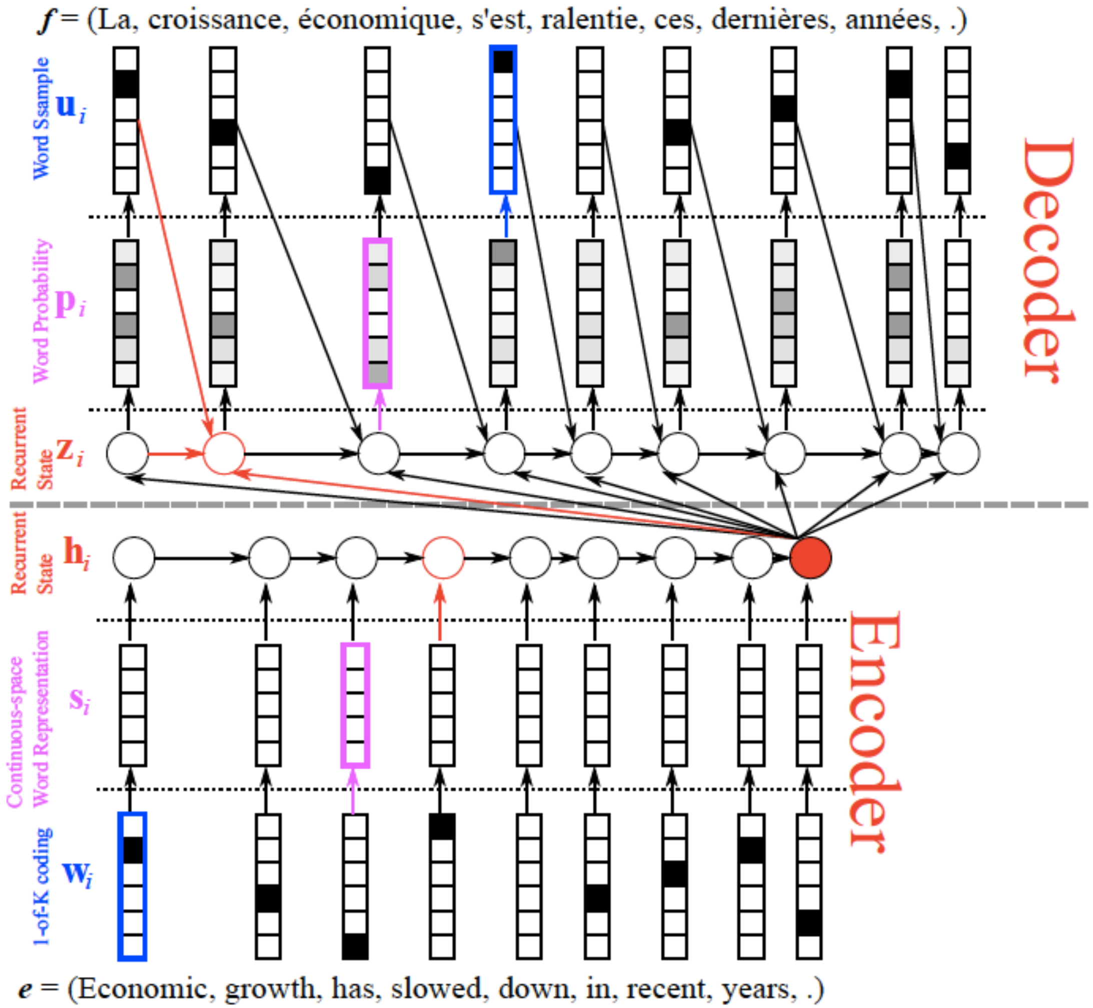


special symbol <EOS> to indicate the beginning of the sequence

Sutskever et al. 2014

- When the model was first proposed, no one would have believed that it can solve the translation problem
- But it worked quite well
- Limitations
  - Out of vocabulary (OOV) problems
  - Fixed size vocab, constrained by computational budget and GPU RAM.
  - All words not in vocab will be mapped to the same <UNK> token.
  - More of a problem for **morphologically rich languages like Polish**.

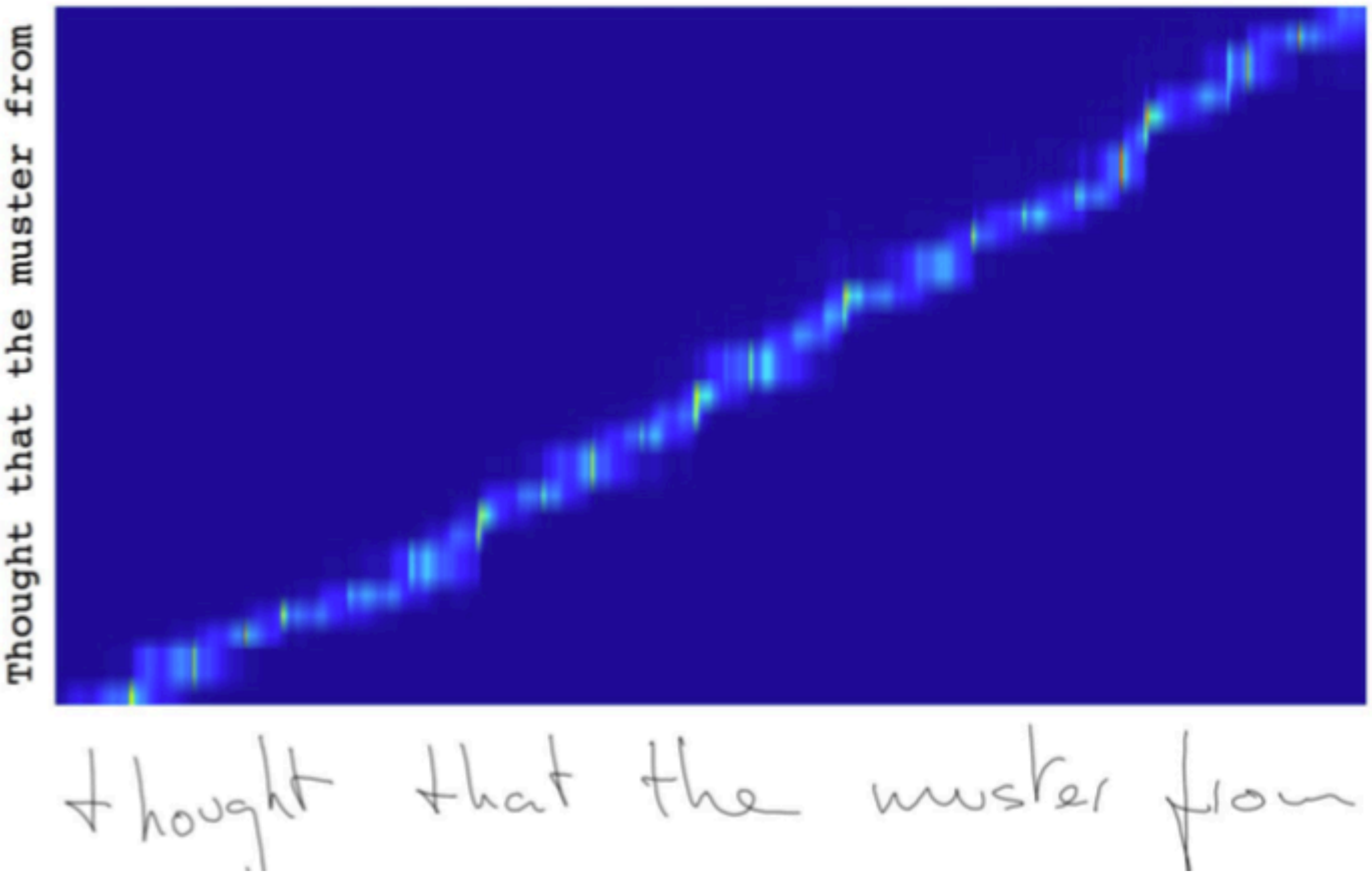
Neural Machine Translation, Model- *encoder*



f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
Recurrent State
$\mathbf{z}_i$
Decoder
Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

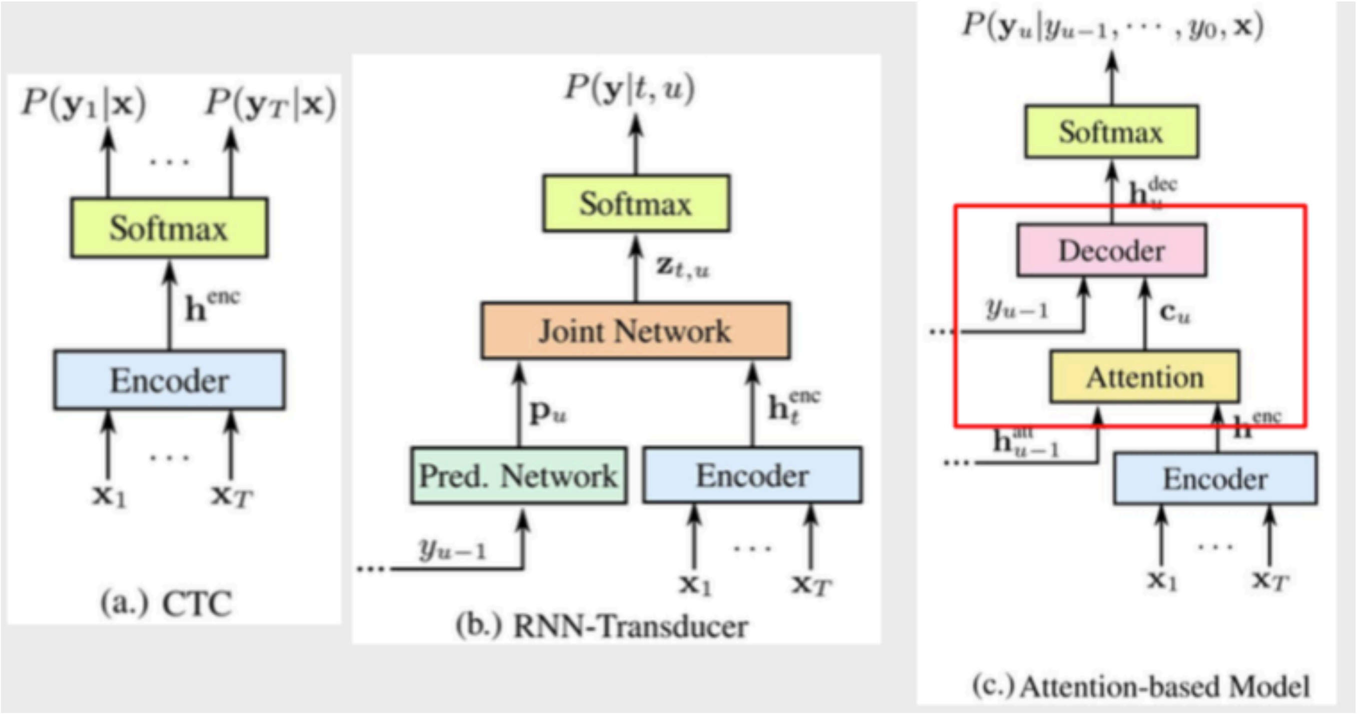
Cho: From Sequence Modeling to Translation

Handwriting synthesis



Comparing Sequence-to-Sequence Models for Speech Recognition

- Attention-based models have the encoder feeding the decoder with acoustic information.

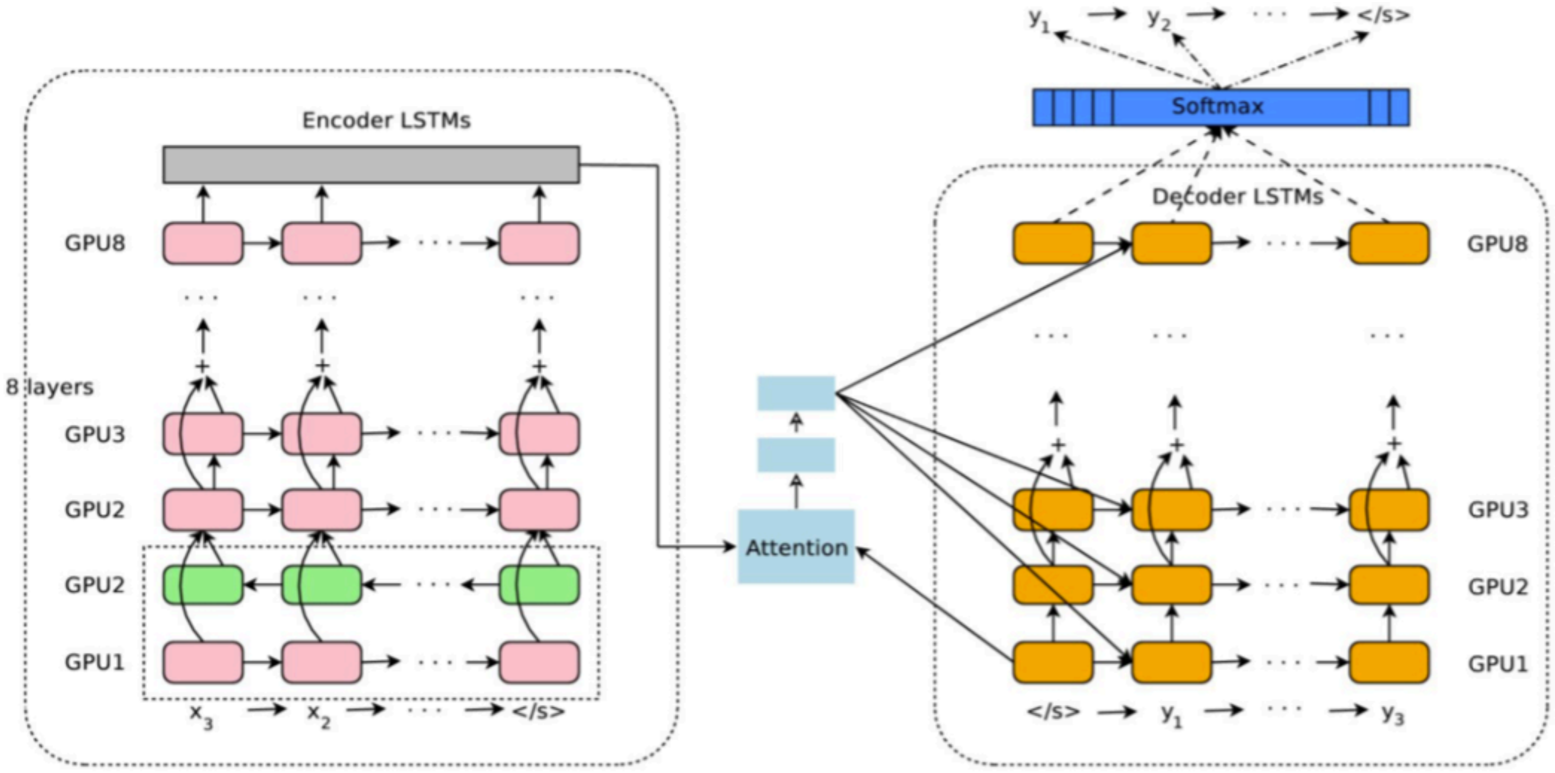


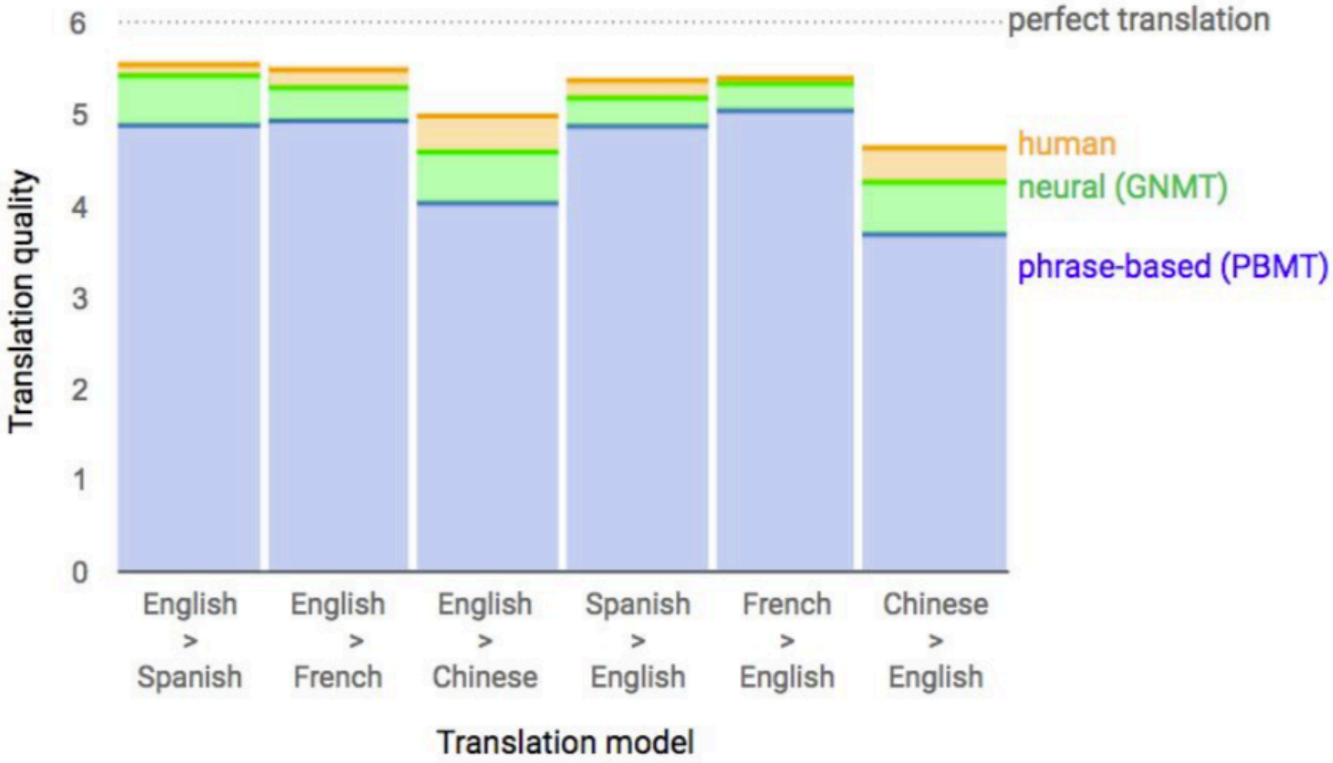
(a.) CTC

(b.) RNN-Transducer

(c.) Attention-based Model

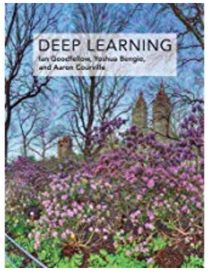
Google Neural Machine Translation



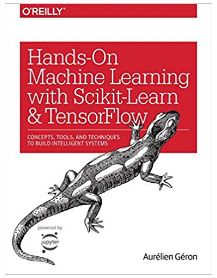


6
5
4
3
2
1
0
Translation quality
perfect translation
human
neural (GNMT)
phrase-based (PBMT)
English > Spanish
English > French
English > Chinese
Spanish > English
French > English
Chinese > English
Translation model

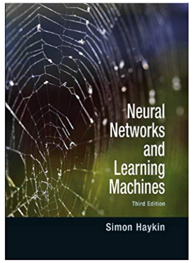
Literature



- Deep Learning, I. Goodfellow, Y. Bengio, A. Courville MIT Press 2016
  - Chapter 10

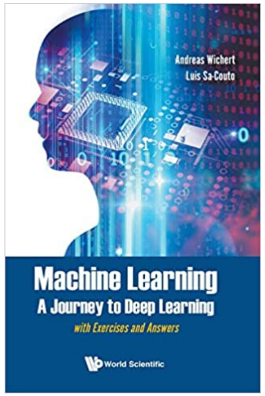

O'REILLY
Hands-On Machine Learning with Scikit-Learn & TensorFlow
CONCEPTS, TOOLS, AND TECHNIQUES TO BUILD INTELLIGENT SYSTEMS
Aurélien Géron

- Hands-On Machine Learning with Scikit-Learn and TensorFlow,   Aurélien Géron , O'Reilly Media; 1 edition, 2017
  - Chapter 14



- Simon O. Haykin, Neural Networks and Learning Machine, (3rd Edition), Pearson 2008
  - Chapter 15

Literature



- Machine Learning - A Journey to Deep Learning, A. Wichert, Luis Sa-Couto, World Scientific, 2021
  - Chapter 14