EE782 Advanced Topics in Machine Learning
RNN Architectures

Amit Sethi

Electrical Engineering, IIT Bombay

Learning outcomes for the lecture

- List issues with vanilla RNN
- Show how LSTM and GRU overcome those issues
- List design choices for LSTM
- Derive backprop for LSTM/GRU

Contents

- **Need for memory to process sequential data**
- Recurrent neural networks
- LSTM basics
- Some applications of LSTM in NLP
- Some advanced LSTM structures

What is sequential data?

- One-dimensional discrete index
  - Example: time instances, character position
- Each data point can be a scalar, vector, or a symbol from an alphabet

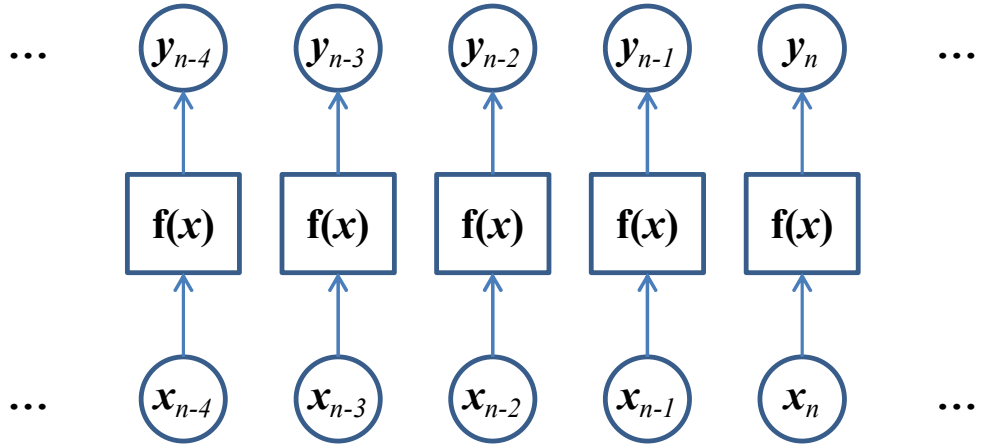
- Number of data points in a series can be ***variable***

Examples of sequential data

- Speech
- Text (NLP)
- Music
- Protein and DNA sequences
- Stock prices and other time series

Traditional ML is one-to-one

- POS tagging in NLP (input: words, output: POS tag)
- Stock trade: {Buy, NoAction, Sell}

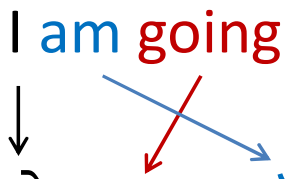


- What about taking past data into account?

Need for past data or context

- Different POS
  - It is a quick **read**
  - I like to **read**

- Translation

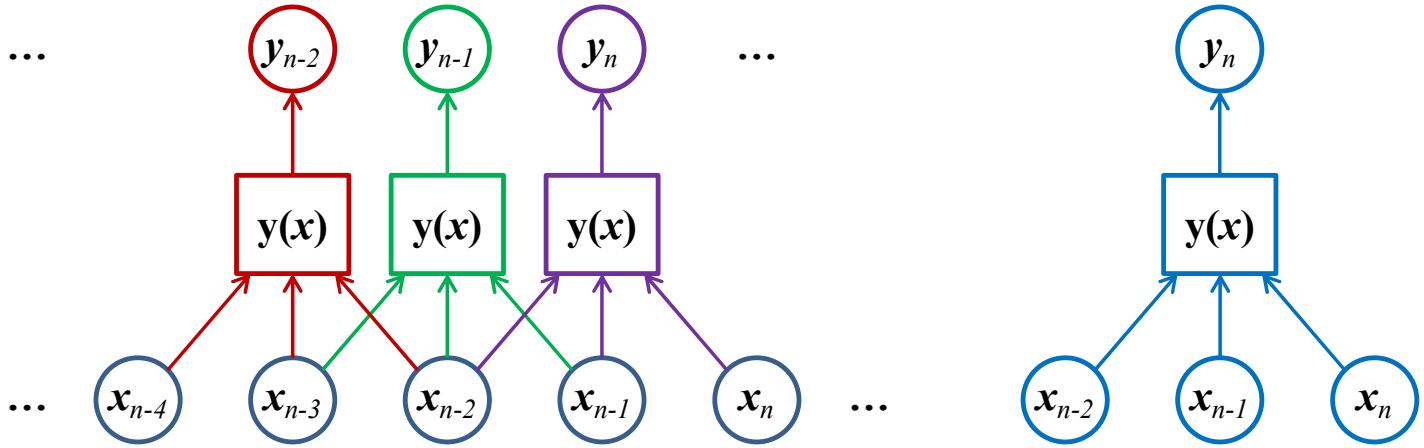


(Re-ordered, ideal)

मैं हूँ जा रहा (Word by word, less than ideal)

Using traditional ML for sequences

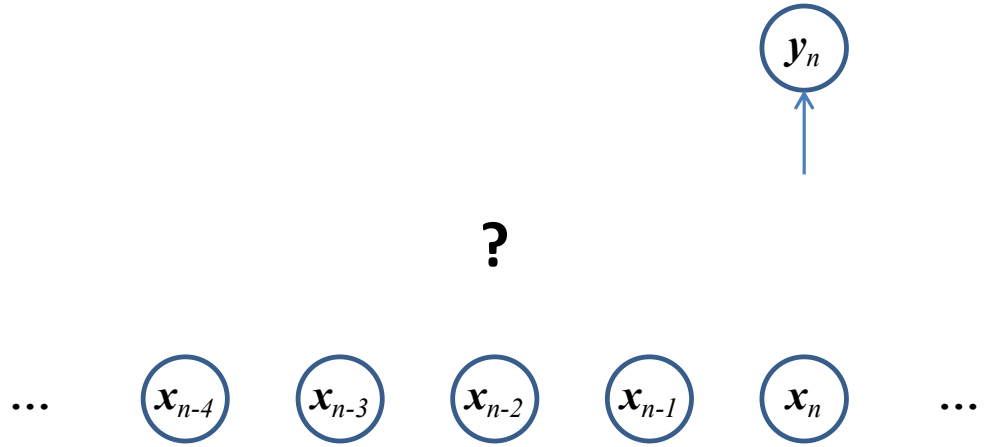
- Work with a fixed window



- What about influence of distant past?

What if we want to do many to one

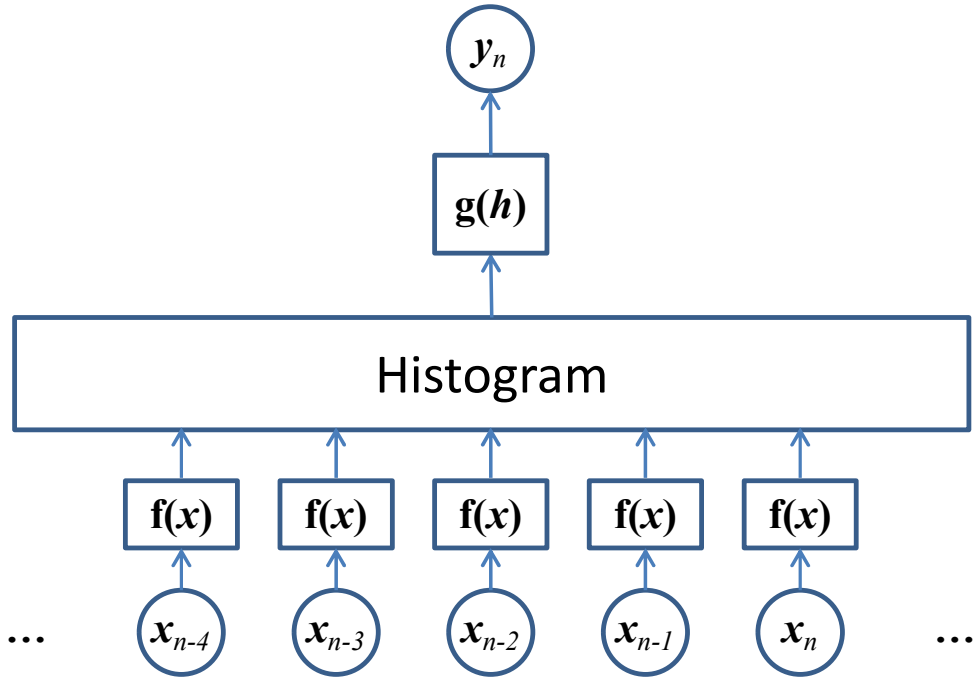
- Sentiment analysis in NLP



- What about taking past data into account?

Using traditional ML for sequences

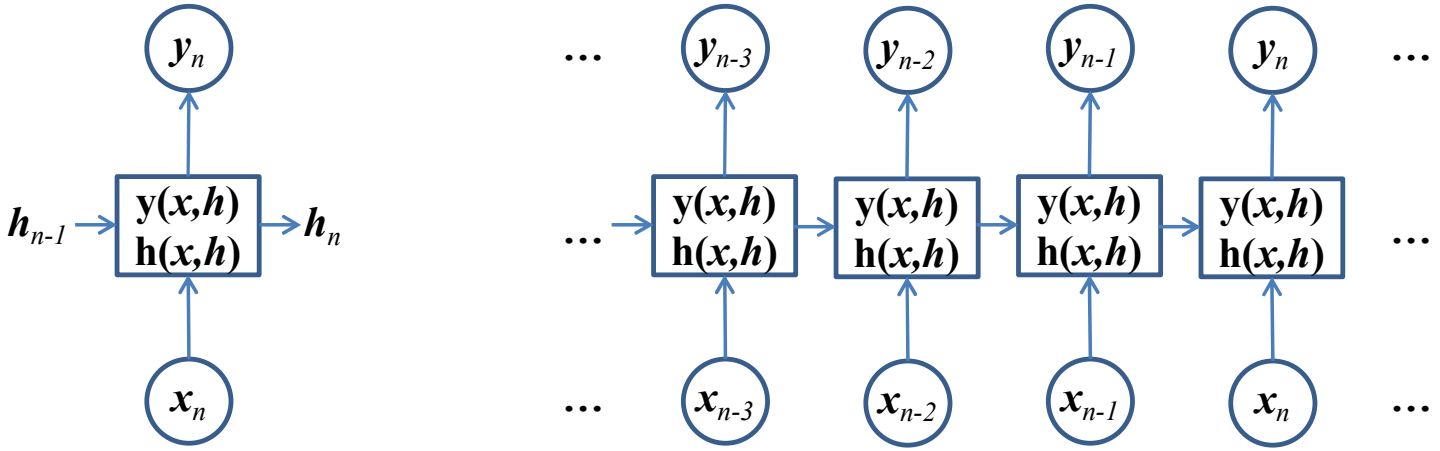
- Convert sequence into a feature vector



- What about using the order of the data?

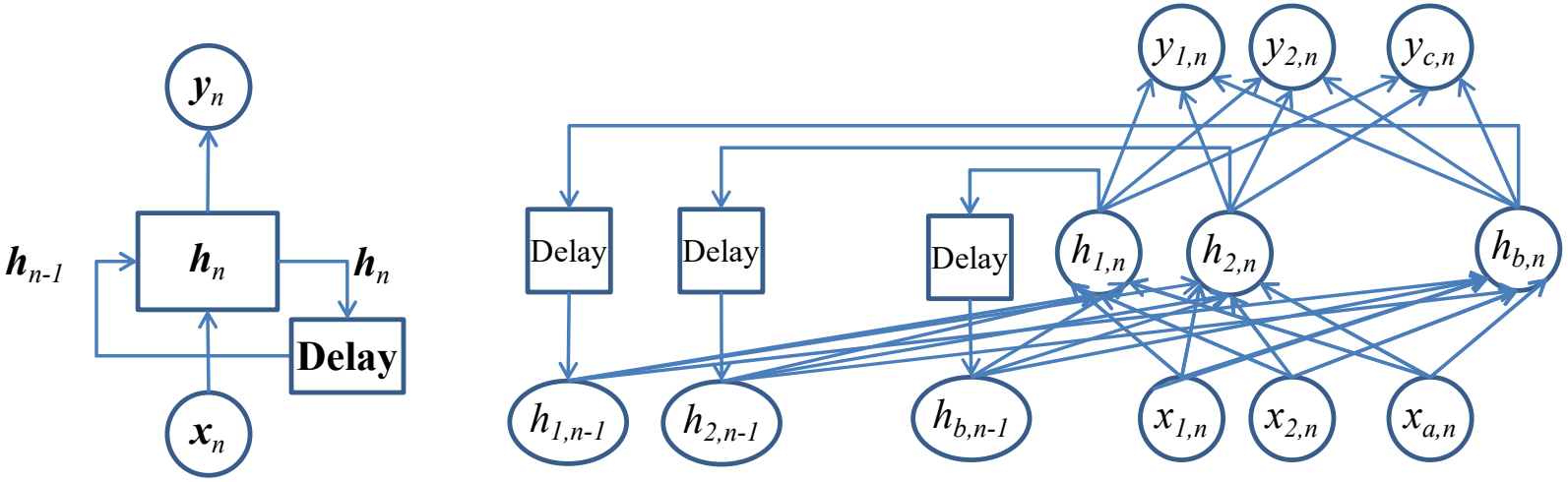
Introducing memory (recurrence or state) in neural networks

- A memory state is computed in addition to an output, which is sent to the next time instance



Another view of recurrence

- A memory state is computed in addition to an output, which is sent to the next time instance

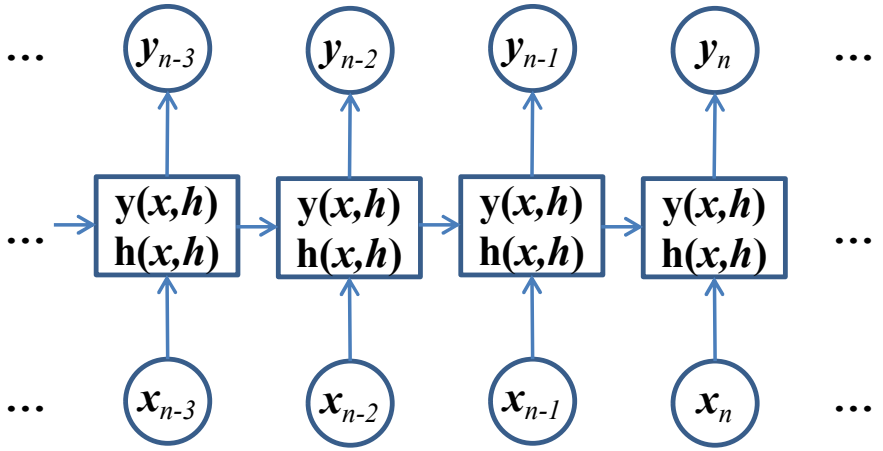


Types of analysis possible on sequential data using “recurrence”

- One to one
- One to many
- Many to one
- Many to many

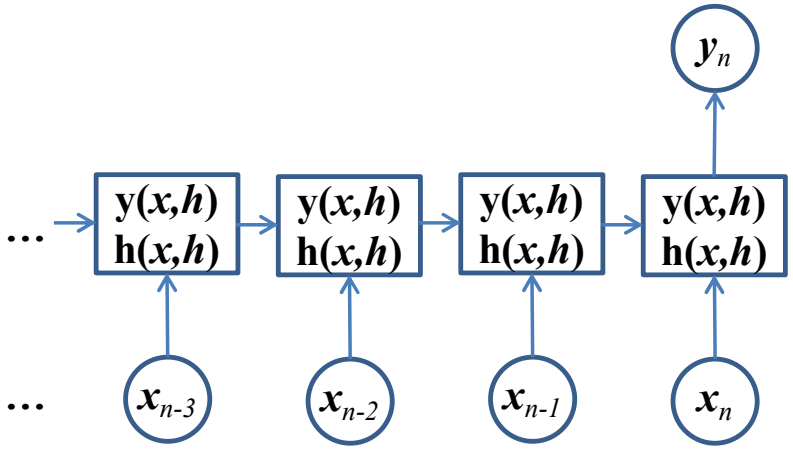
Examples: Many to many

- POS tagging in NLP
- Stock trade: {Buy, NoAction, Sell}



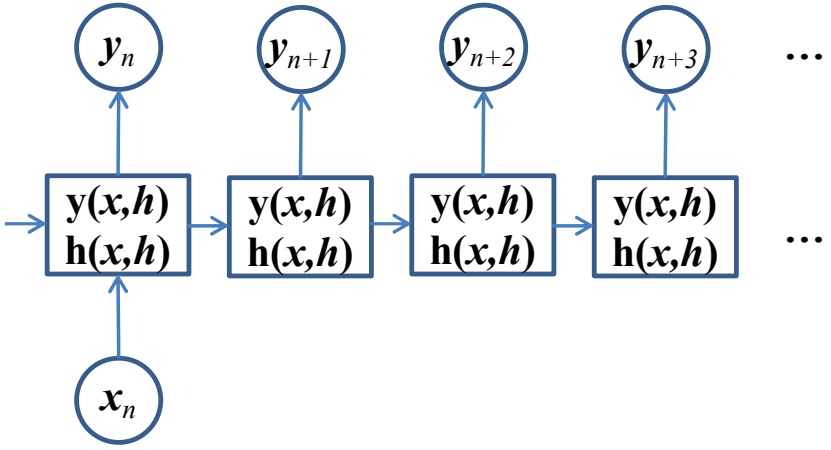
Examples: many to one

- Sentiment analysis in NLP



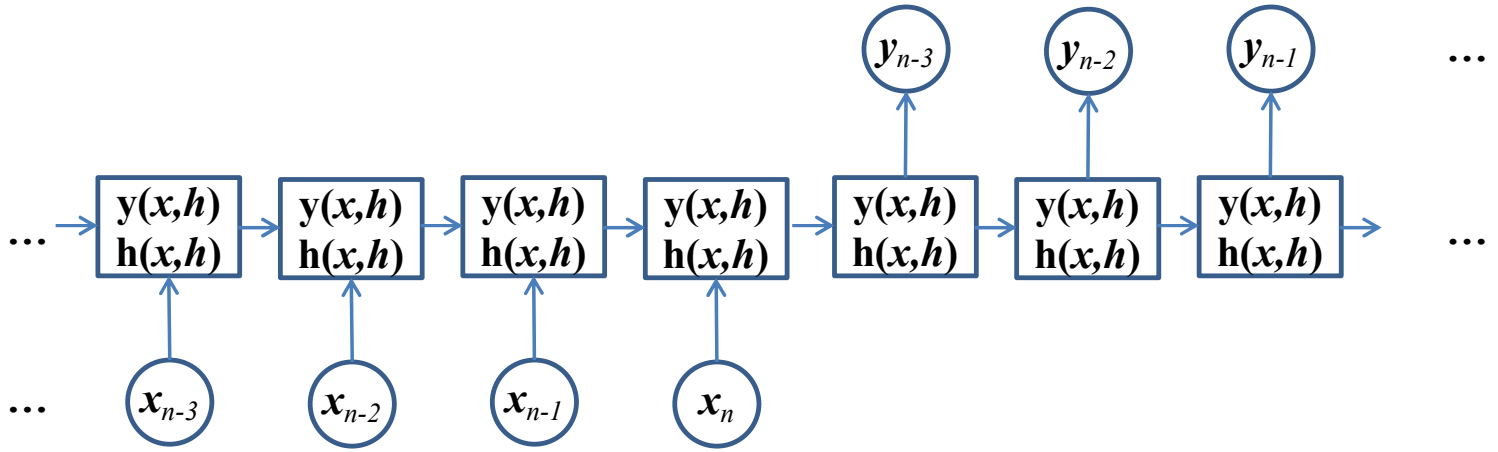
Examples: One to many

- Generate caption based on an image
- Generate text given topic



Examples: Many to many

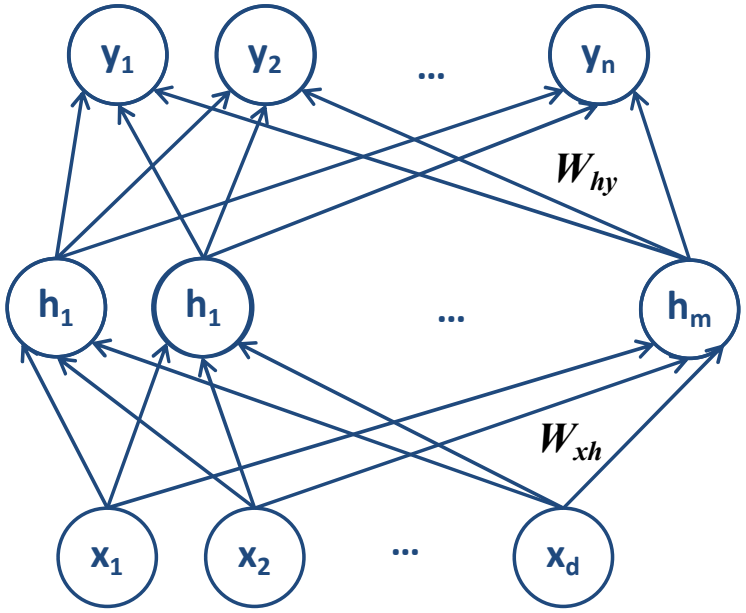
- Language translation

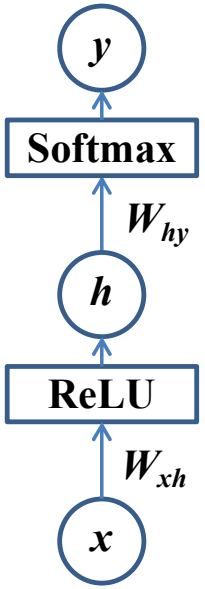


Contents

- Need for memory to process sequential data
- **Recurrent neural networks**
- LSTM basics
- Some applications of LSTM in NLP
- Some advanced LSTM structures

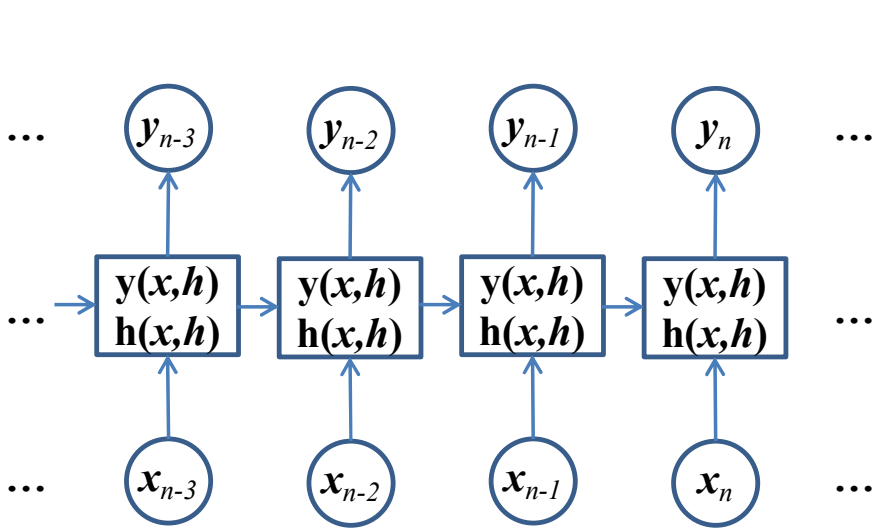
Revising feedforward neural networks

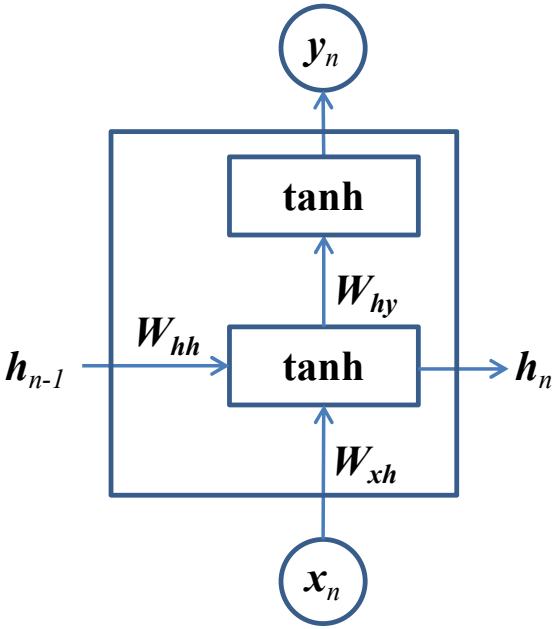




Recurrent neural networks

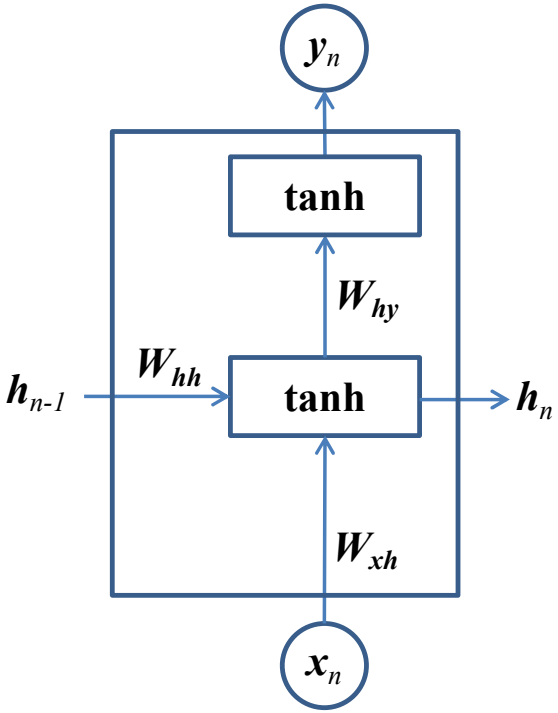
- Vanilla RNNs used the hidden layer activation as a state



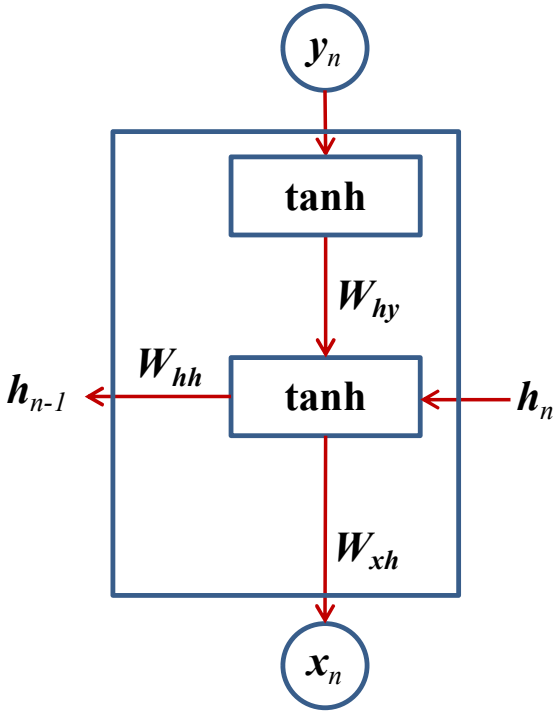


Backpropagation through time (BPTT)

- Just like how forward propagation uses previous state …

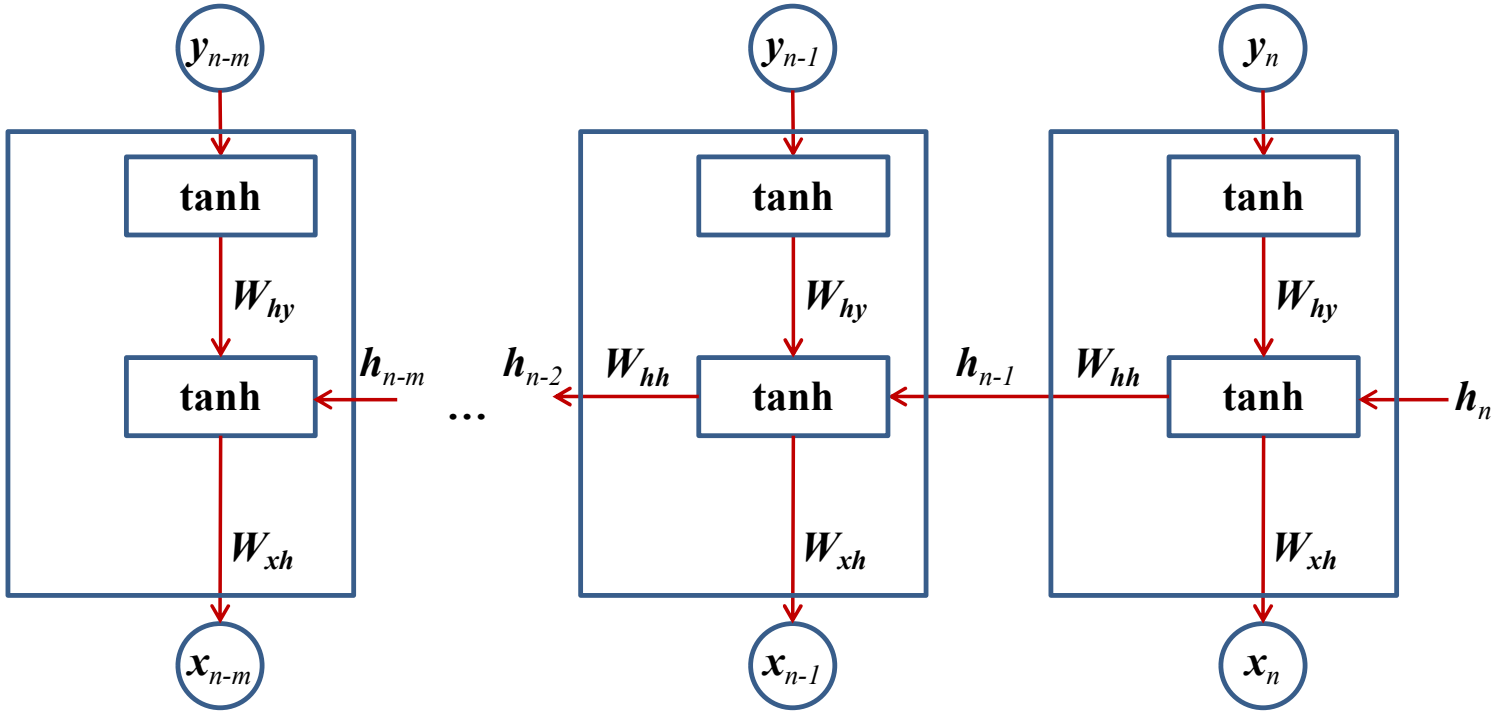


- Backpropagation uses derivative from future output



Use of a window length

- We need to put a limit on how long will the gradient travel back in time



Mathematical expression for BPTT

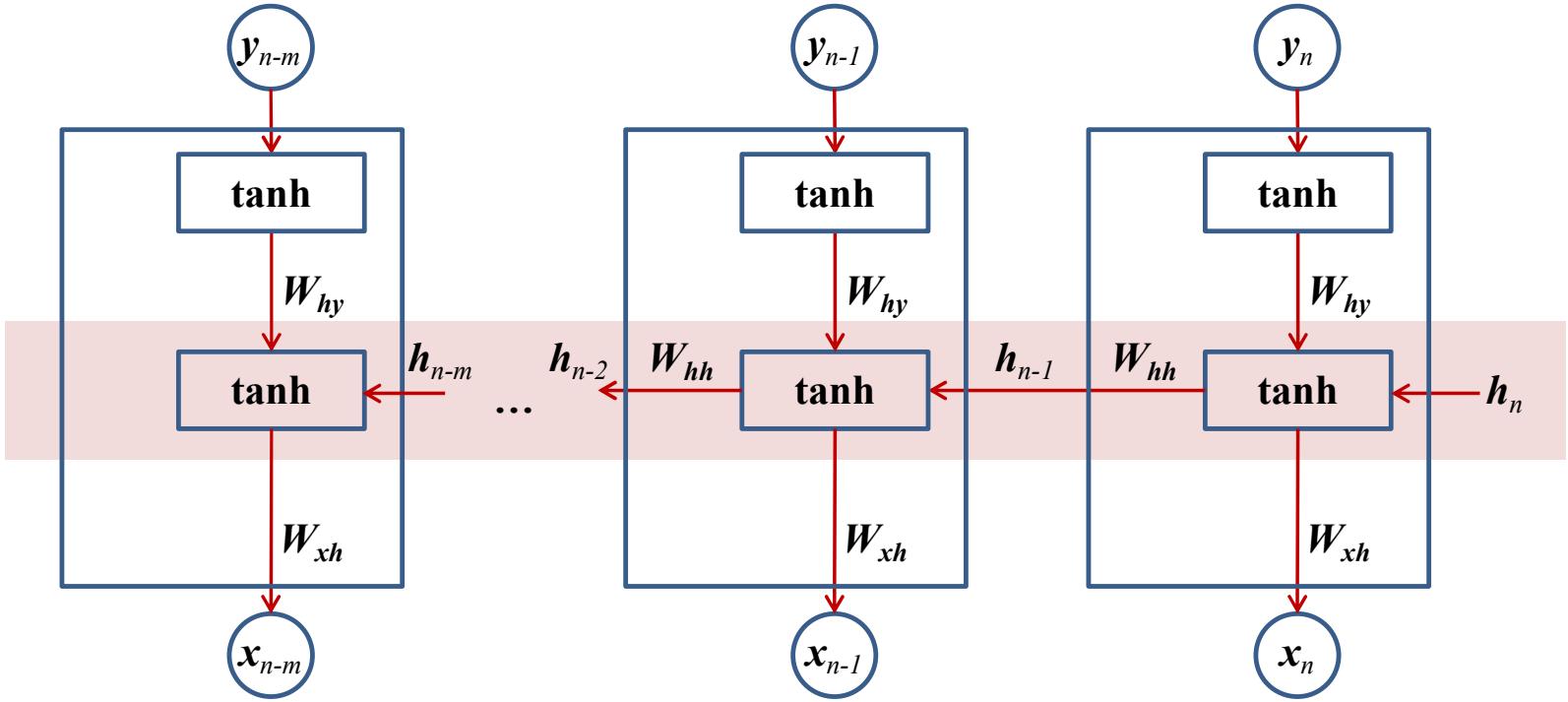
- Forward: $\boldsymbol{y}_n = \boldsymbol{g}(\boldsymbol{W}_2\boldsymbol{h}_n)$
$= \boldsymbol{g}(\boldsymbol{W}_2\boldsymbol{f}(\boldsymbol{W}_{11}\boldsymbol{h}_{n-1} + \boldsymbol{W}_{12}\boldsymbol{x}_n))$

- Backward example:

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}_{11}} = \boldsymbol{g}'\boldsymbol{W}_2\boldsymbol{h}_n{}' = \boldsymbol{g}'\boldsymbol{W}_2\boldsymbol{f}'(\boldsymbol{h}_{n-1} + \boldsymbol{W}_{11}\boldsymbol{h}_{n-1}{}')$$

Vanishing and exploding gradient

- Gradient gets repeatedly multiplied by $\boldsymbol{W}_{hh}$
- This can lead to vanishing or exploding gradient depending on the norm of $\boldsymbol{W}_{hh}$



Exploding and vanishing gradients

- Backpropagation uses chain rule
- For every layer $f_l\left(W_l{}^T f_{l-1}\left(W_{l-1}{}^T(\ldots x)\ldots\right)\right)$,
- The gradient requires multiplication of weights

$$
\begin{aligned}
&J' \\
&\times f_l'\left(W_l{}^T f\left(W_{l-1}{}^T(\ldots x)\ldots\right)\right) \\
&\times W_l \\
&\times f_{l-1}'\left(W_{l-1}{}^T(\ldots x)\ldots\right) \\
&\times W_{l-1} \\
&\times \ldots
\end{aligned}
$$

- In BPTT for RNNs, the same weight gets multiplied by itself over and over

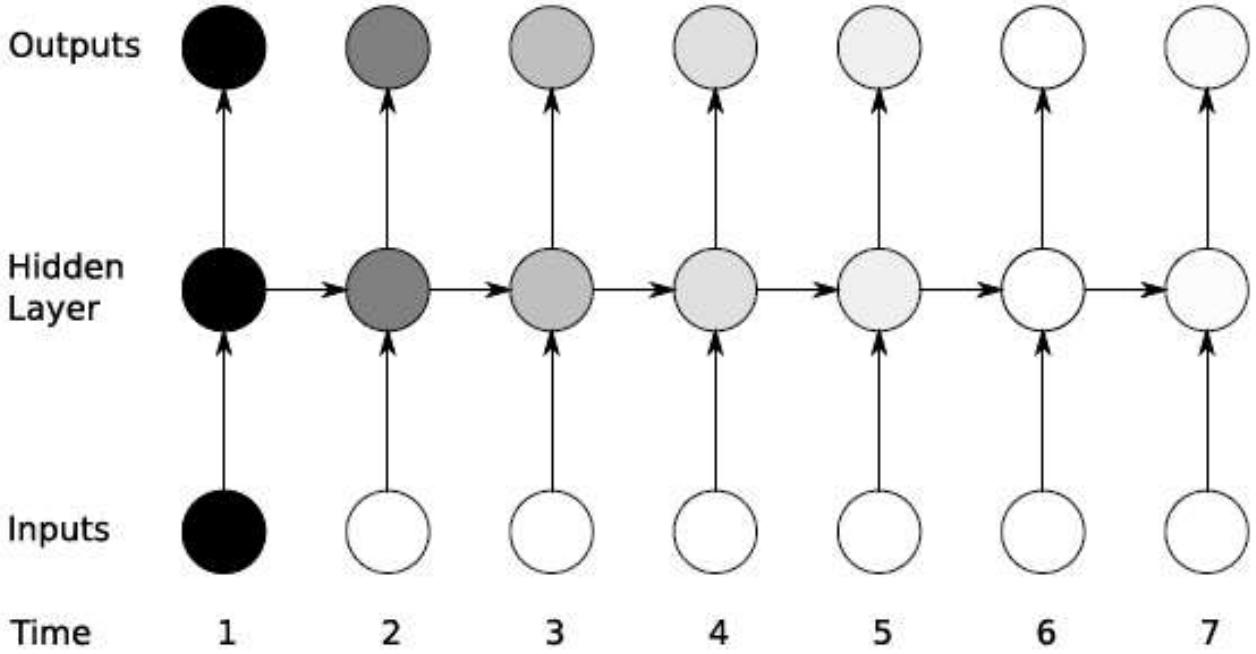


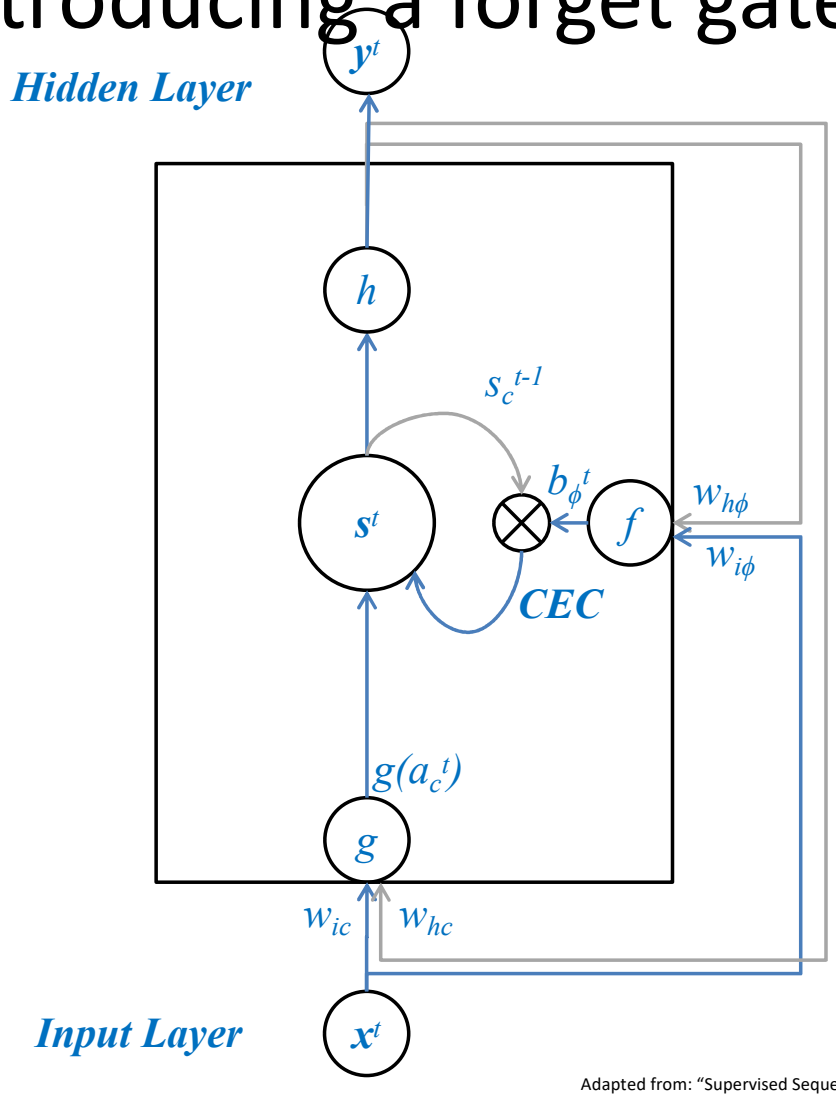
Figure 4.1: **The vanishing gradient problem for RNNs.** The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

Source: "*Supervised Sequence Labelling with Recurrent Neural Networks*." by Graves, Alex.

Contents

- Need for memory to process sequential data
- Recurrent neural networks
- **LSTM basics**
- Some applications of LSTM in NLP
- Some advanced LSTM structures

Introducing a forget gate to control the gradient

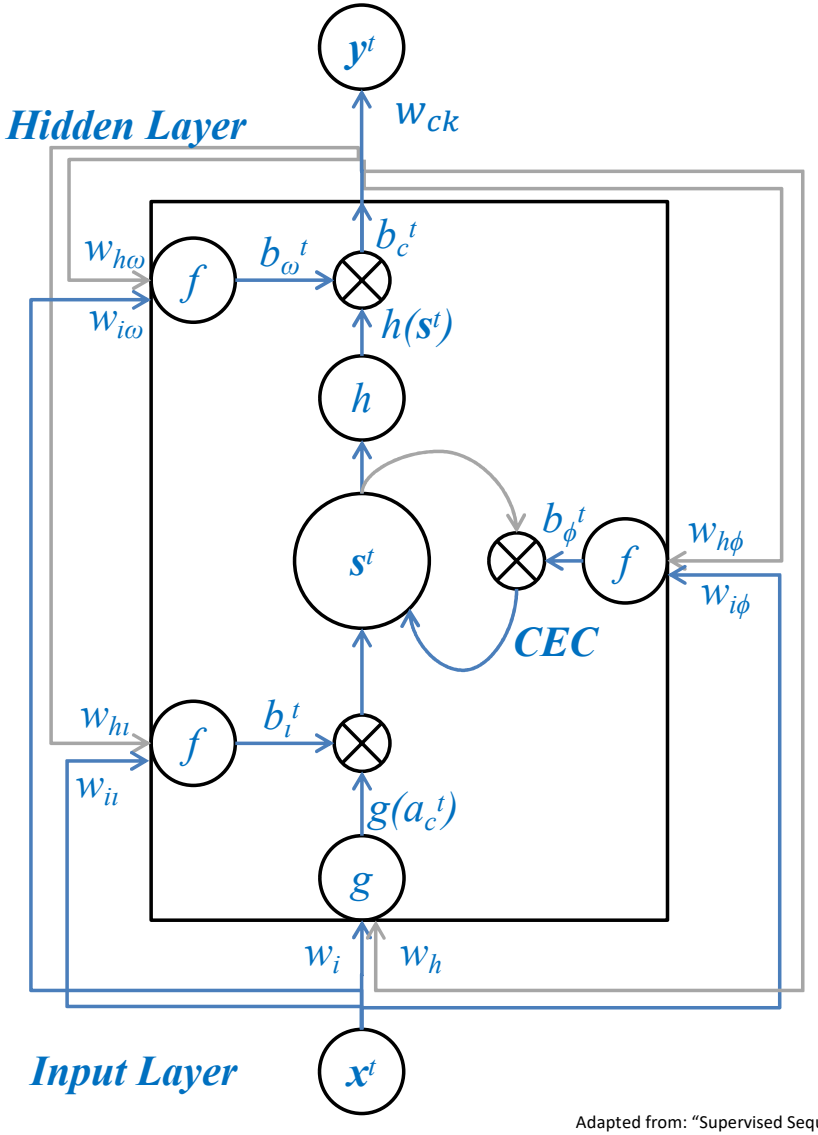


- The state doesn't multiply with a constant weight
- A gate function *f* (usually a sigmoid) represents *on* or *off*
- State is *forgotten* and replaced by the input *g* if *f* = 0
- But, what if *f* = 1? How do we control the influence of input?

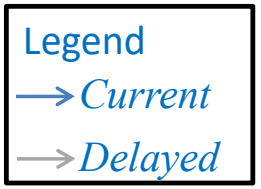
Legend
Current
Delayed

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves

Adding input and output gates



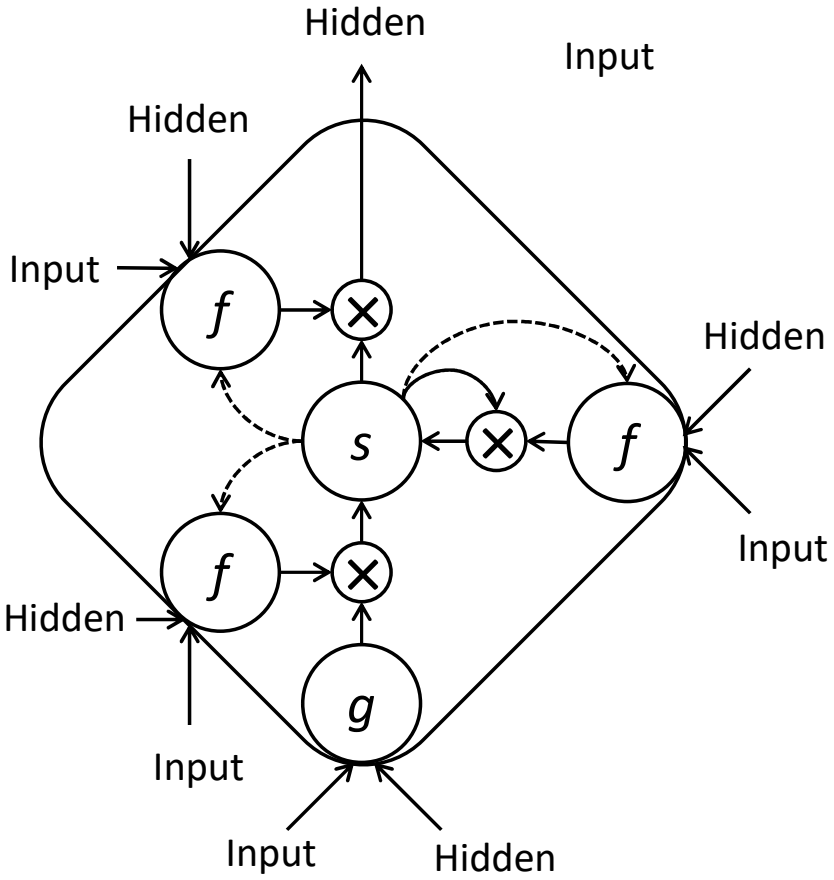
- On similar lines as the forget gate, an input gate decides whether the input will override the state or not
- Similarly, an output gate will decide whether the output will be passed out or not
- The input to all these gates are NN inputs, and NN hidden-layer output

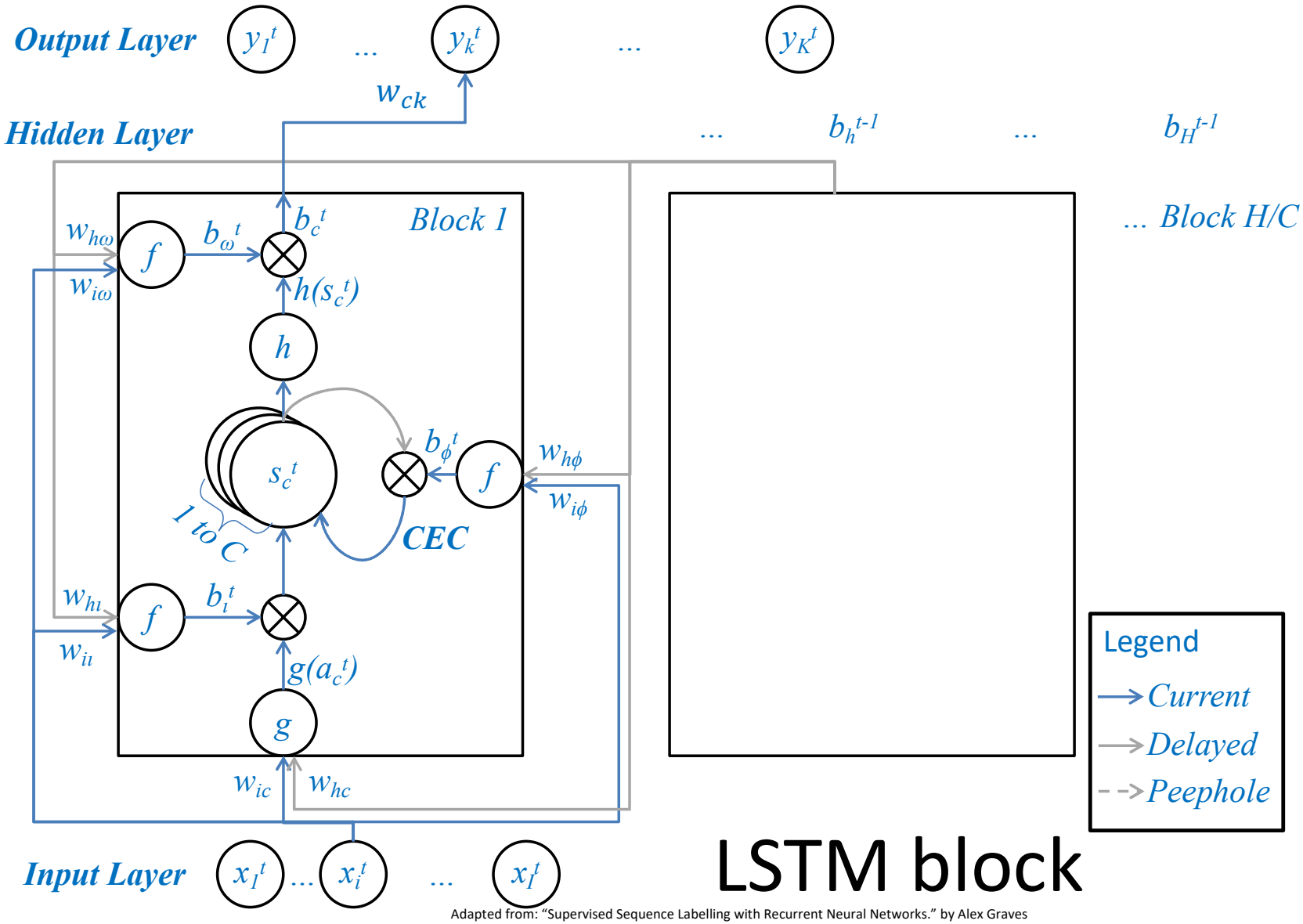


Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves

Another view

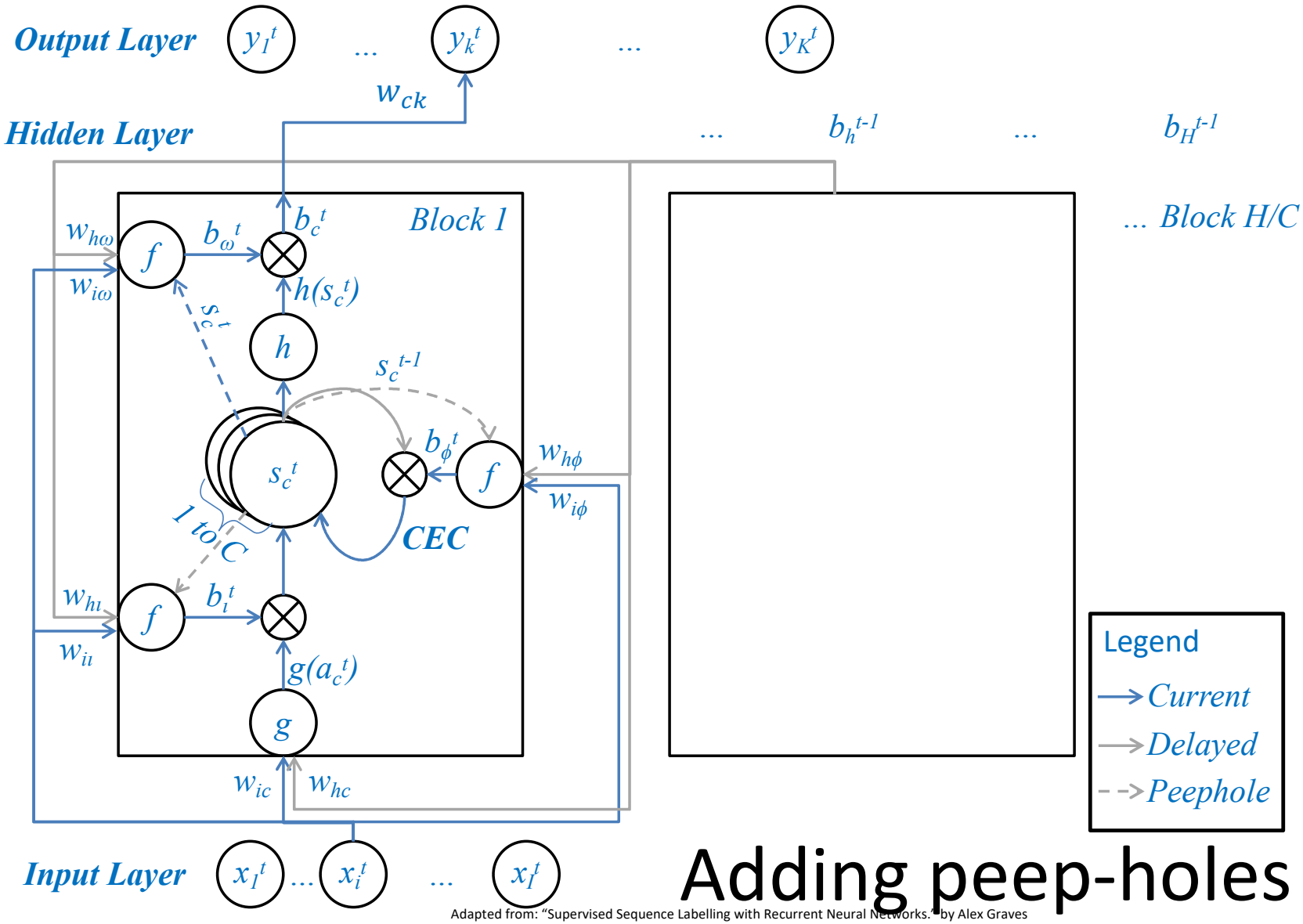
- CEC is constant error carrousel
  - No vanishing gradients
  - But, it is not always on
- Introducing gates:
  - Allow or disallow input
  - Allow or disallow output
  - Remember or forget state





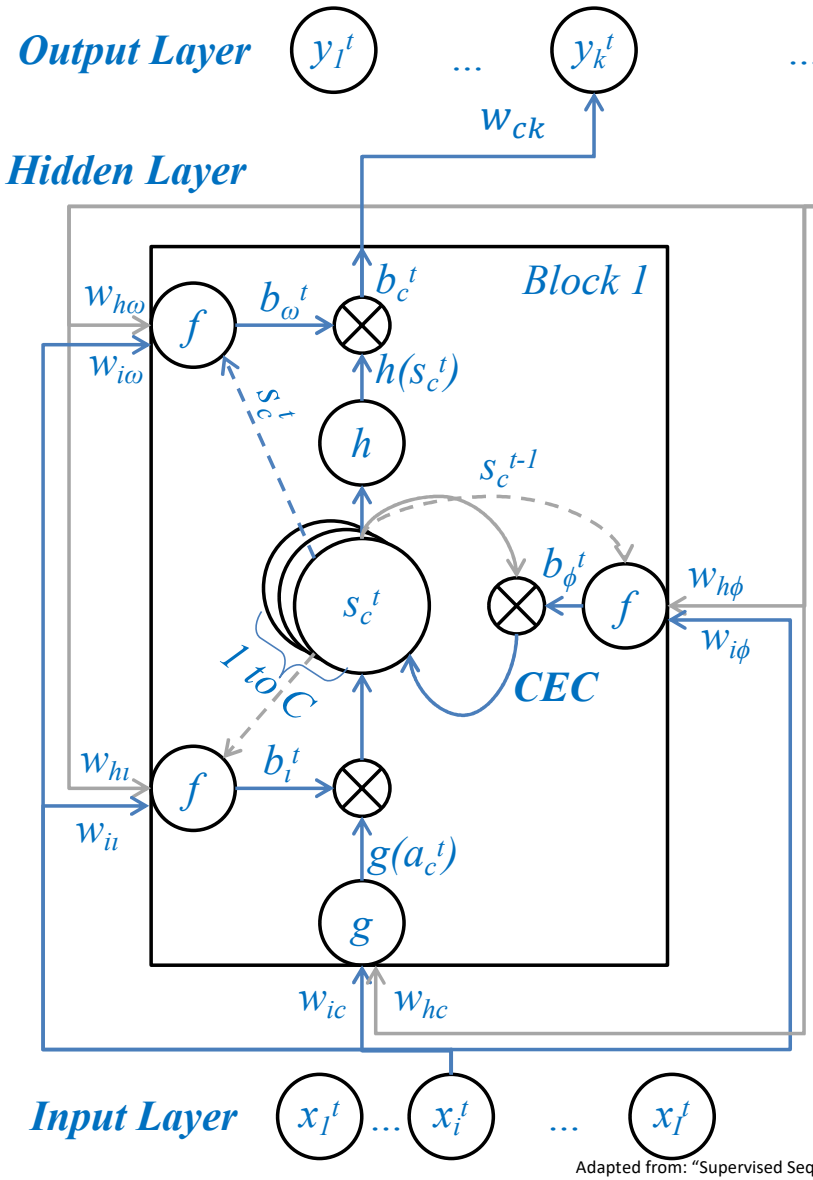
LSTM block

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves



Adding peep-holes

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves



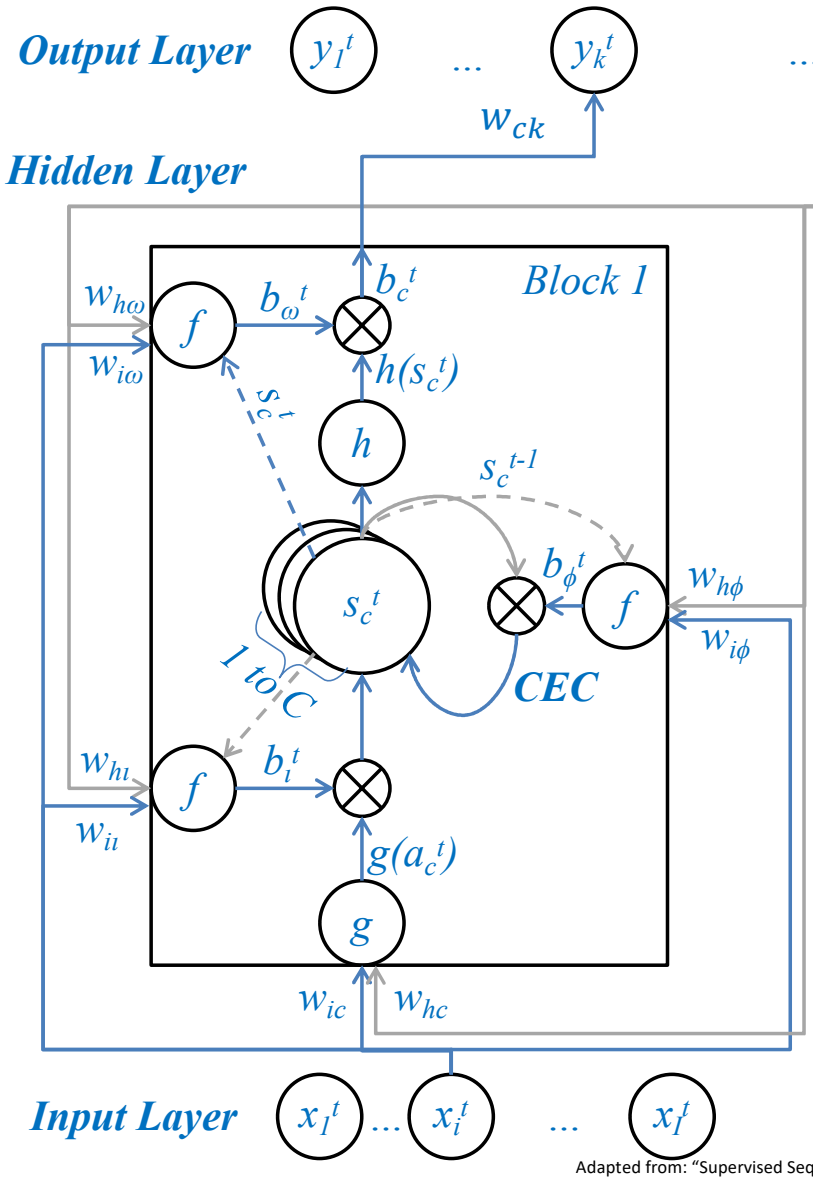
Forward Pass

Input gate:

$$a_l^t = \sum_{i=1}^{I} w_{il} x_i^t + \sum_{h=1}^{H} w_{hl} b_h^{t-1} + \sum_{c=1}^{C} w_{cl} s_c^{t-1}$$

$$b_l^t = f(a_l^t)$$

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves



Forward Pass

Forget gate:

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

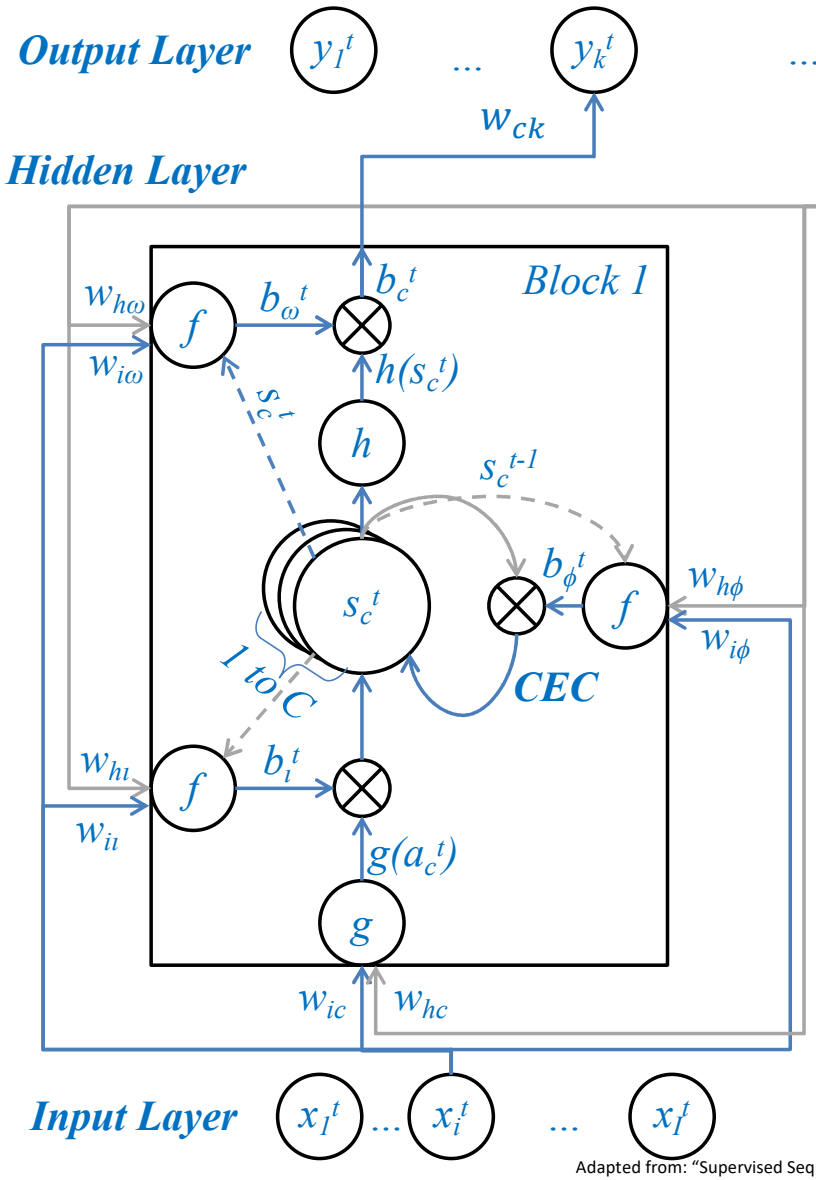
$$b_\phi^t = f(a_\phi^t)$$

Cell input:

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_l^t g(a_c^t)$$

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves



Forward Pass

Output gate:

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t$$

$$b_\omega^t = f(a_\omega^t)$$

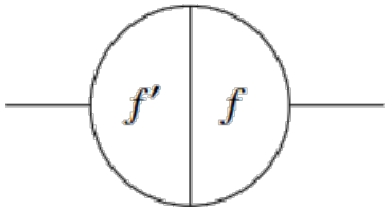
Cell output:

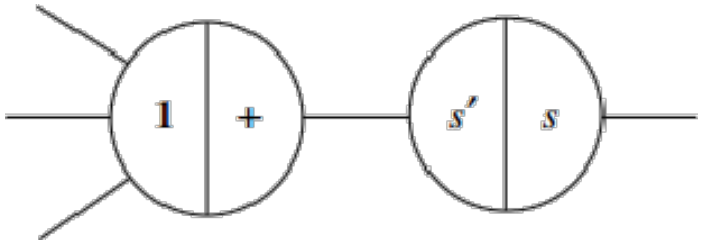
$$b_c^t = b_\omega^t h(s_c^t)$$

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves

Revisiting backpropagation through b-diagrams

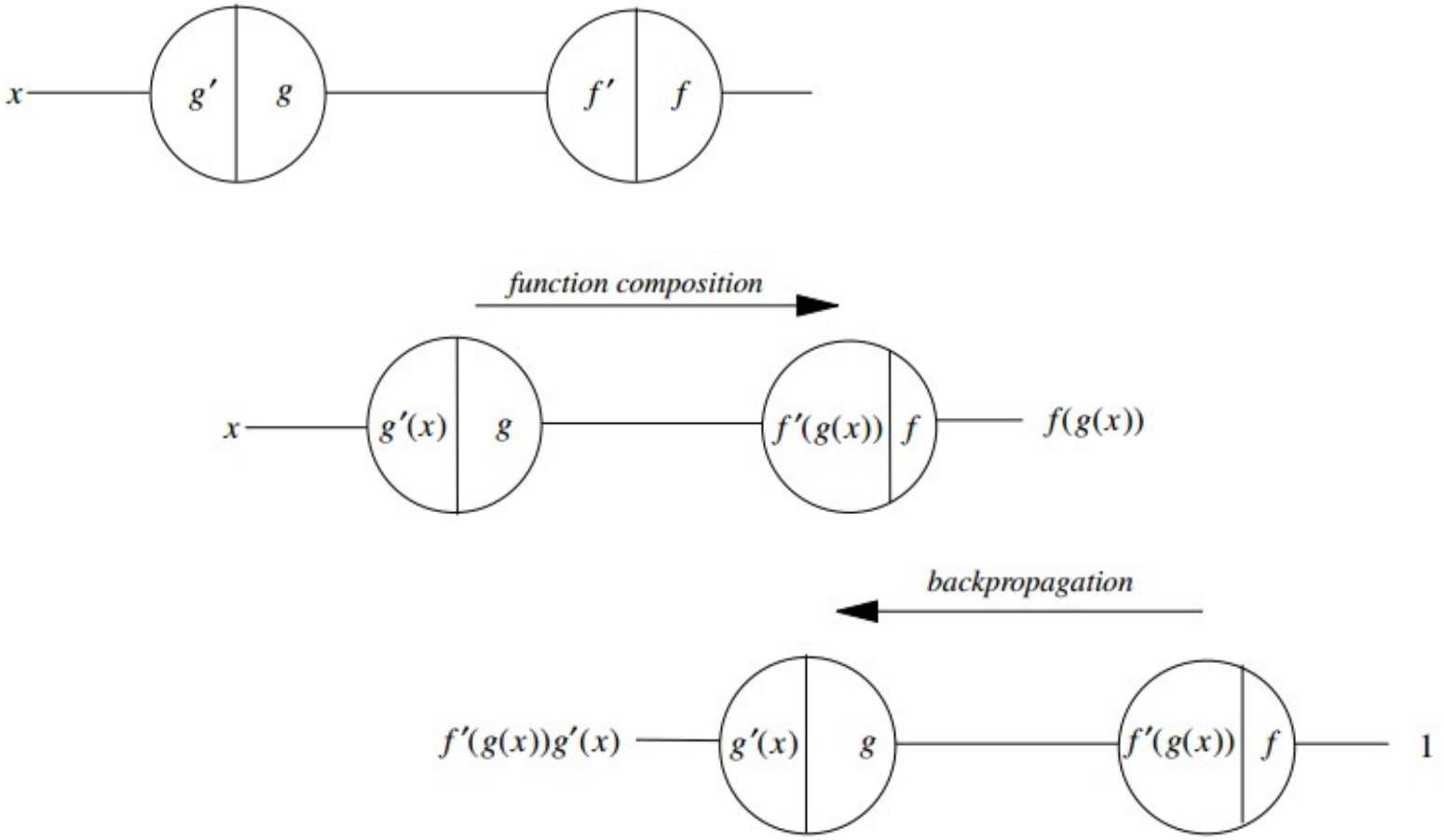
- An efficient way to perform gradient descent in NNs
- Efficiency comes from local computations
- This can be visualized using b-diagrams
  - Propagate x (actually w) forward
  - Propagate 1 backward





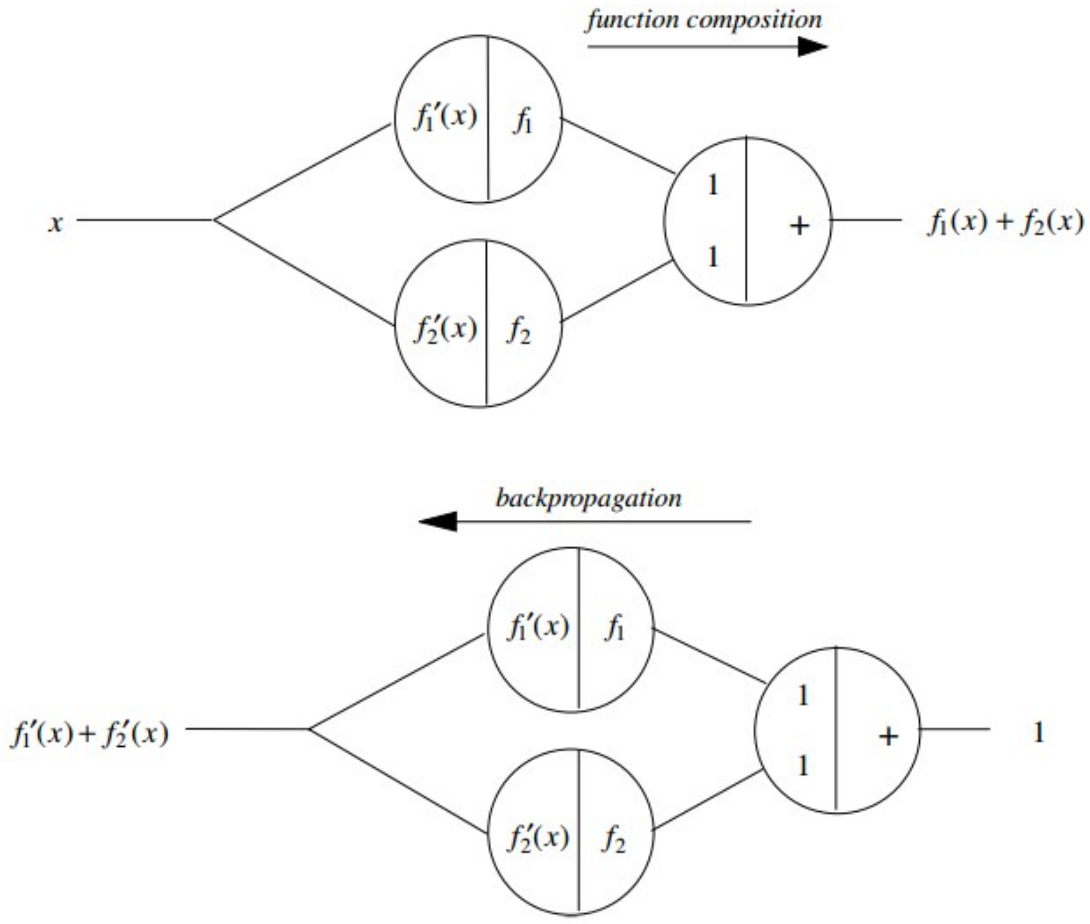
Source: "Neural Networks - A Systematic Introduction," by Raul Rojas, Springer-Verlag, Berlin, New-York, 1996.

Chain rule using b-diagram



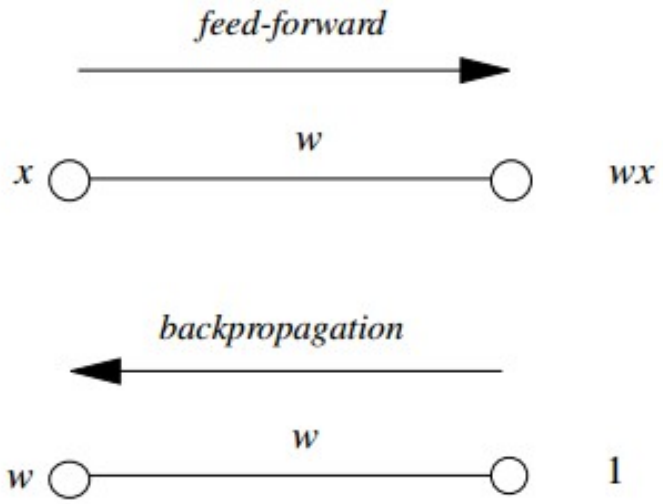
Source: "Neural Networks - A Systematic Introduction," by Raul Rojas, Springer-Verlag, Berlin, New-York, 1996.

Addition of functions using b-diagram



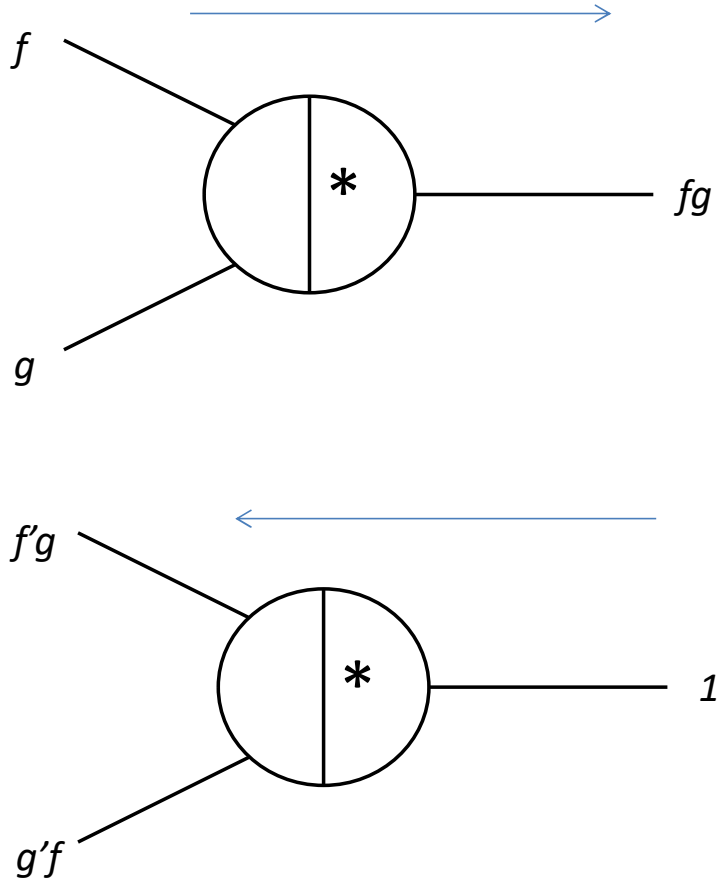
Source: "Neural Networks - A Systematic Introduction," by Raul Rojas, Springer-Verlag, Berlin, New-York, 1996.

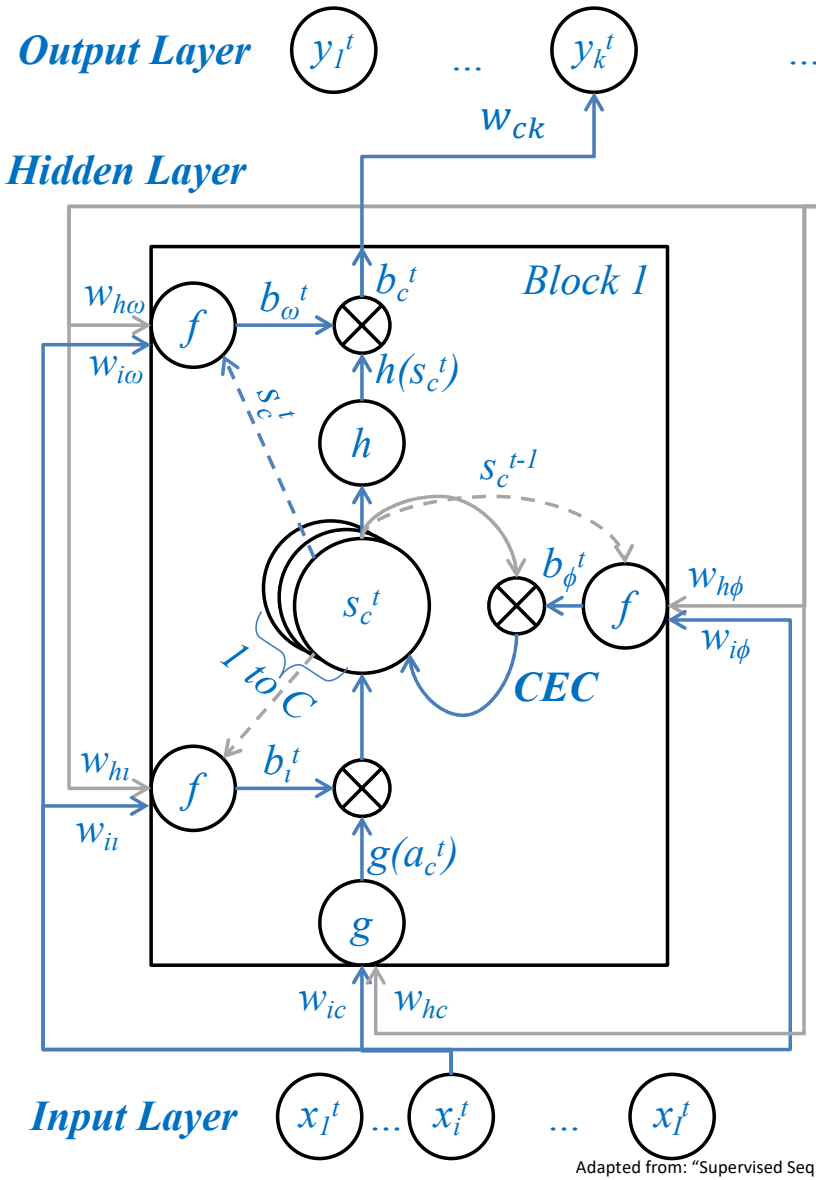
Weighted edge on a b-diagram



Source: "Neural Networks - A Systematic Introduction," by Raul Rojas, Springer-Verlag, Berlin, New-York, 1996.

Product in a b-diagram





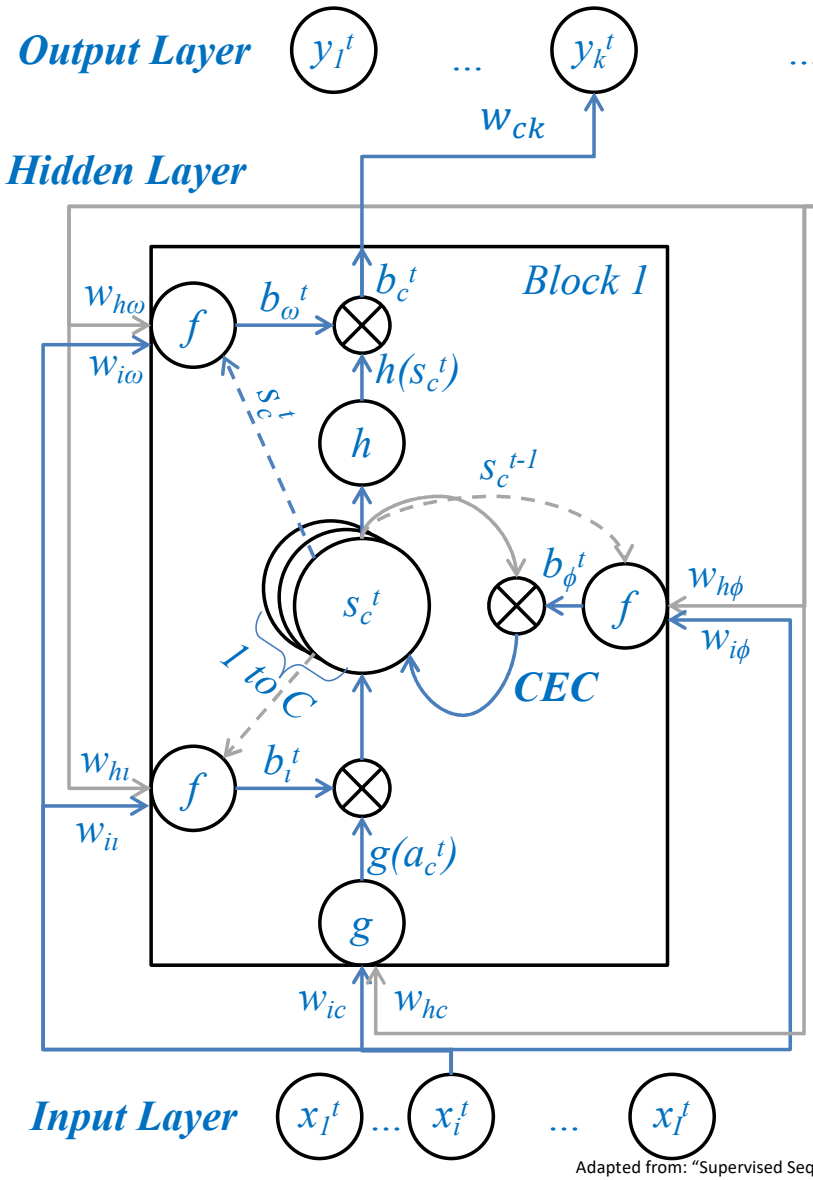
Backpropagation

$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial b_c^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial L}{\partial s_c^t}$$

Cell output:

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1}$$

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves



Backpropagation

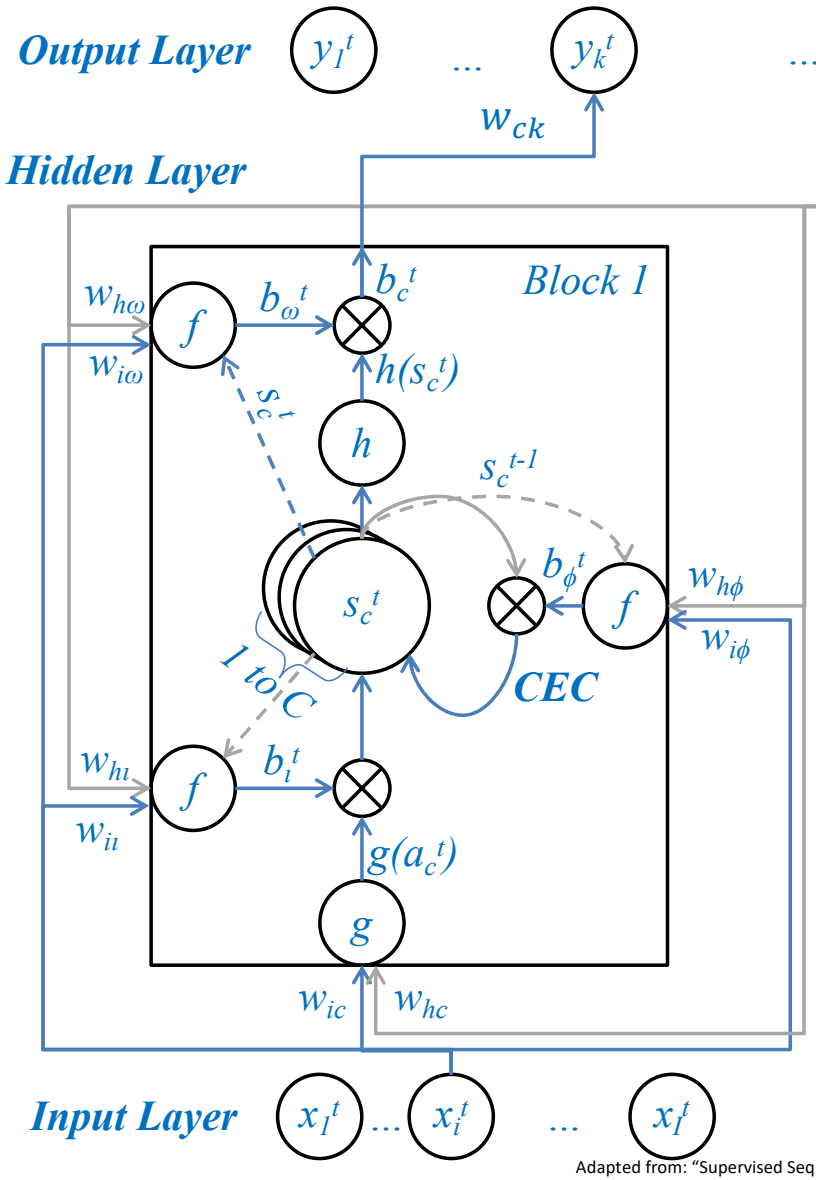
Output gate: $\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t)\epsilon_c^t$

State:

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\emptyset^{t+1}\epsilon_s^{t+1} + w_{cl}\delta_l^{t+1} + w_{c\emptyset}\delta_\emptyset^{t+1} + w_{c\omega}\delta_\omega^t$$

Cells: $\delta_c^t = b_l^t g'(a_c^t)\epsilon_s^t$

Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves



Backpropagation

Forget gate:

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^{t-1} \epsilon_s^t$$

Input gate:

$$\delta_l^t = f'(a_l^t) \sum_{c=1}^{C} g(a_c^t) \epsilon_s^t$$

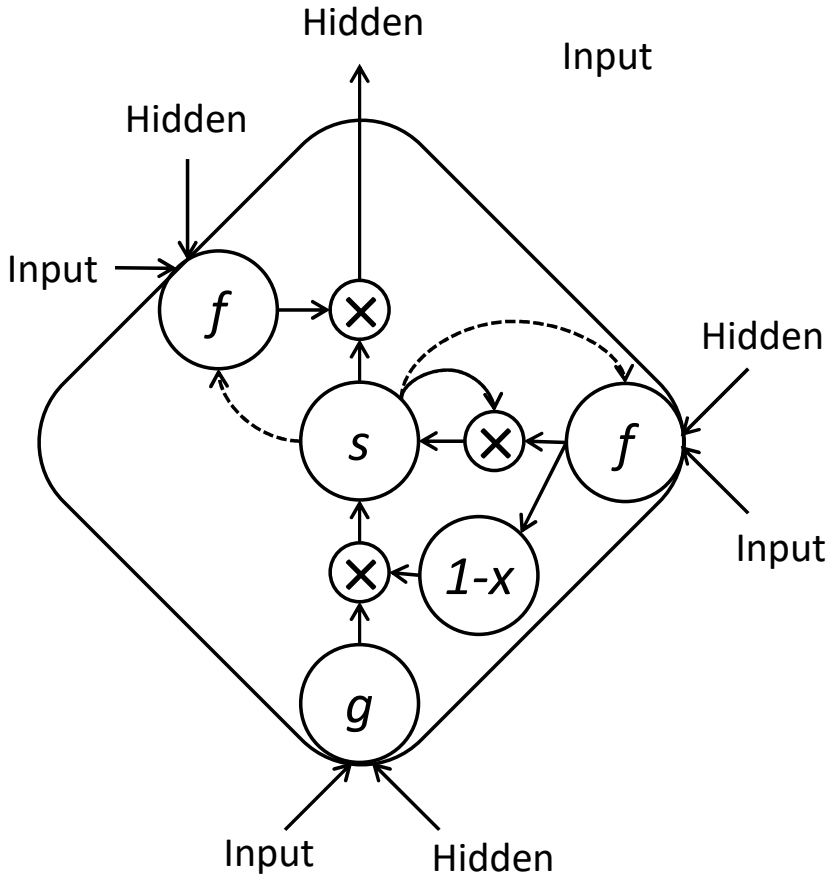
Adapted from: "Supervised Sequence Labelling with Recurrent Neural Networks." by Alex Graves

A few words about the LSTM

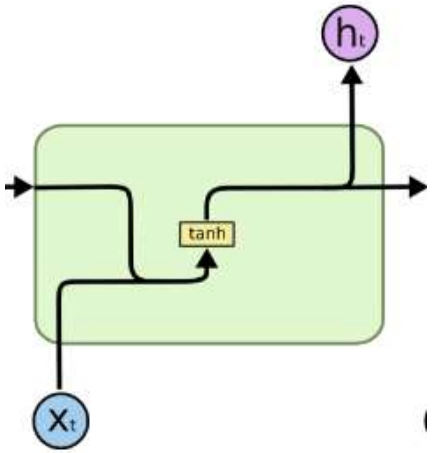
- **CEC:** With the forget gate, influence of the state forward can be modulated such that it can be remembered for a long time, until the state or the input changes to make LSTM forget it. This ability or the path to pass the past-state unaltered to the future-state (and the gradient backward) is called constant error carrousel (CEC). It gives LSTM the ability to remember long term (hence, long short term memory)
- **Blocks:** Since there are just too many weights to be learnt for a single state bit, several state bits can be combined into a single block such that the state bits in a block share gates
- **Peepholes:** The state itself can be an input for the gate using *peephole* connections
- **GRU:** In a variant of LSTM called gated recurrent unit (GRU), input gate can simply be one-minus-forget-gate. That is, if the state is being *forgotten*, then replace it by input, and if it is being *remembered*, then block the input

Gated Recurrent Unit (GRU)

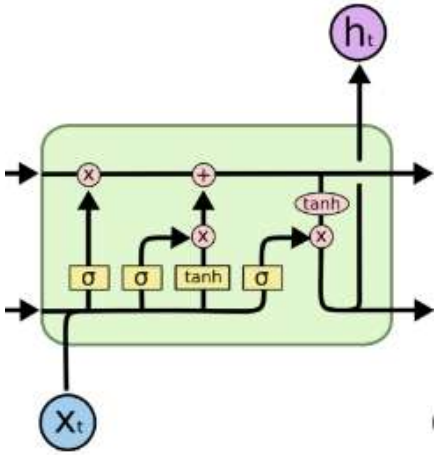
- Reduces the need for input gate by reusing the forget gate



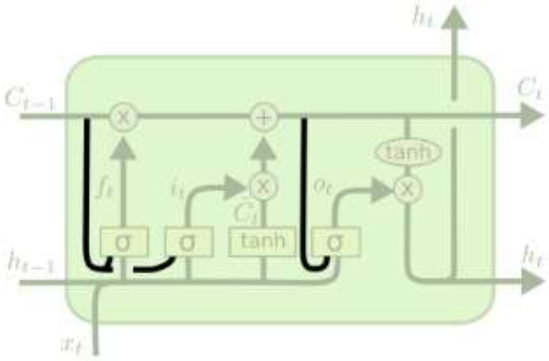
GRUs combine input and forget gates



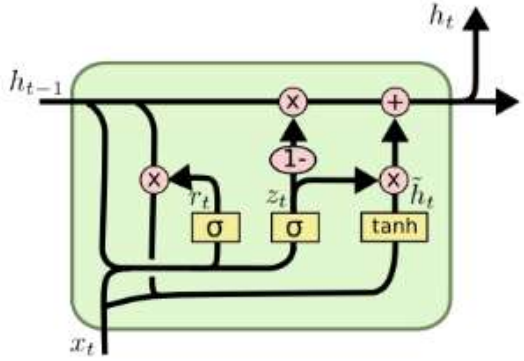
RNN unit



LSTM unit



LSTM unit with *peepholes*



Gated Recurrent Unit combines input and forget gate as *f* and *1-f*

Source: Cho, et al. (2014), and *"Understanding LSTM Networks", by C Olah, http://colah.github.io/posts/2015-08-Understanding-LSTMs/*

Contents

- Need for memory to process sequential data
- Recurrent neural networks
- LSTM basics
- **Some applications of LSTM in NLP**

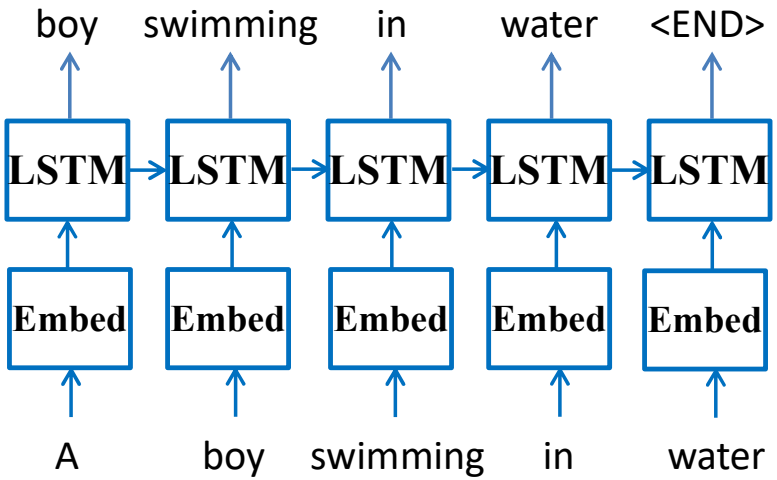
Pre-processing for NLP

- The most basic pre-processing is to convert words into an embedding using Word2Vec or GloVe

- Otherwise, a one-hot-bit input vector can be too long and sparse, and require lots on input weights

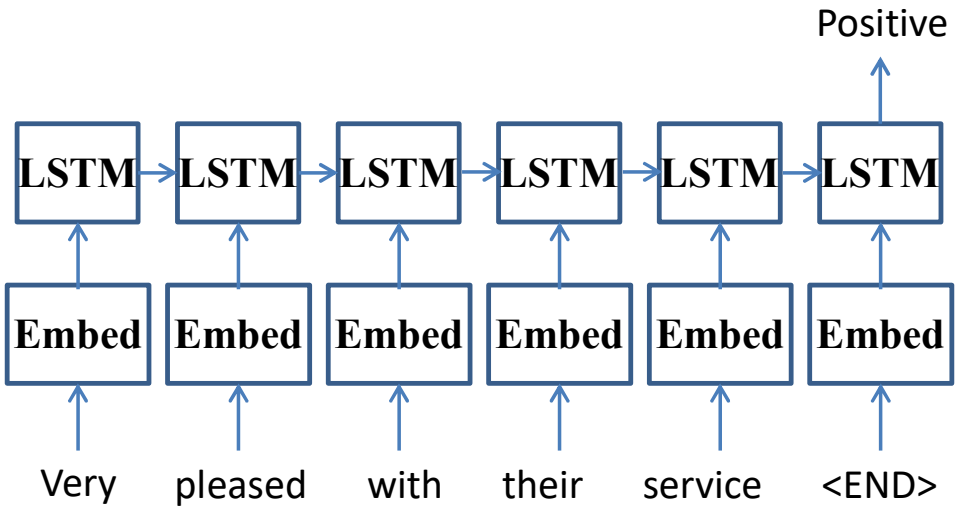
Pre-training LSTMs

- Learning to predict the next word can imprint powerful language models in LSTMs
- This captures the grammar and syntax
- Usually, LSTMs are pre-trained on corpora



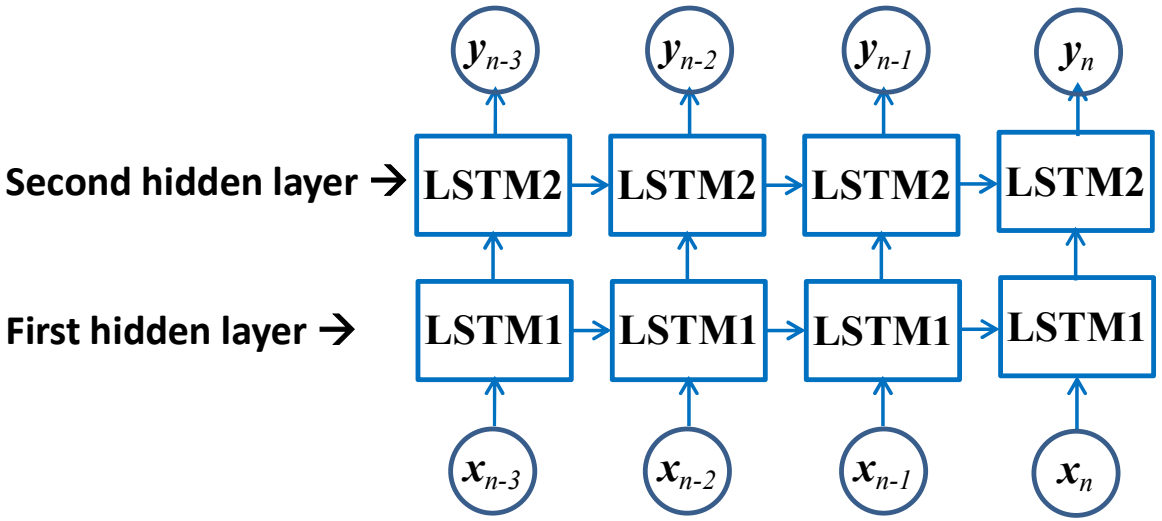
Sentiment analysis

- Very common for customer review or new article analysis
- Output before the end can be discarded (not used for backpropagation)
- This is a many-to-one task



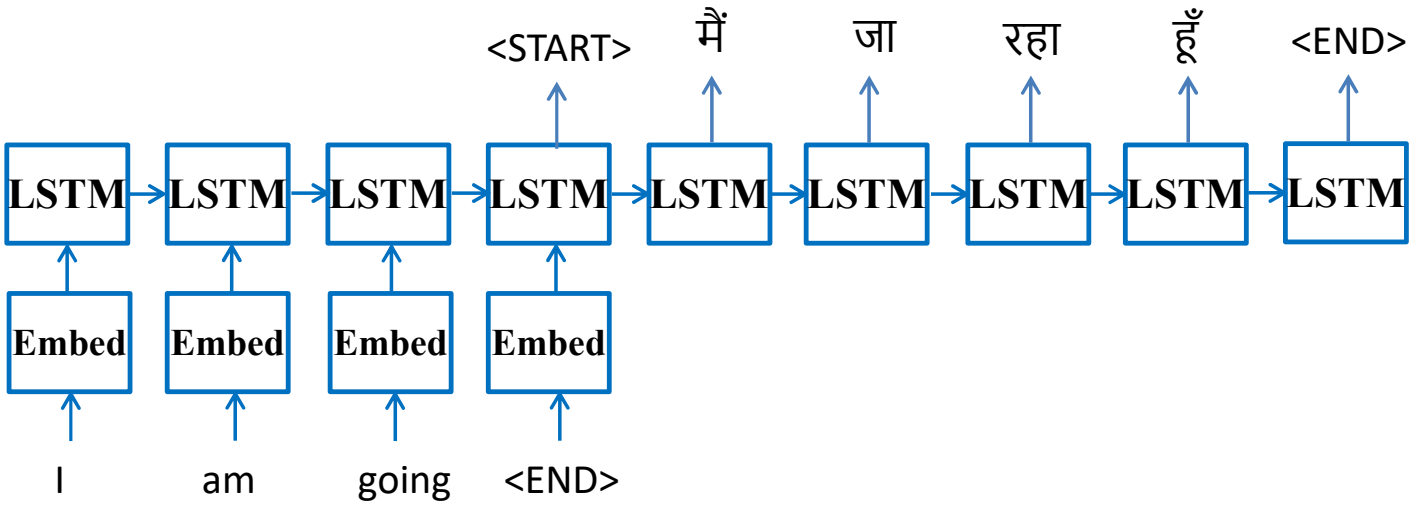
Multi-layer LSTM

- More than one hidden layer can be used



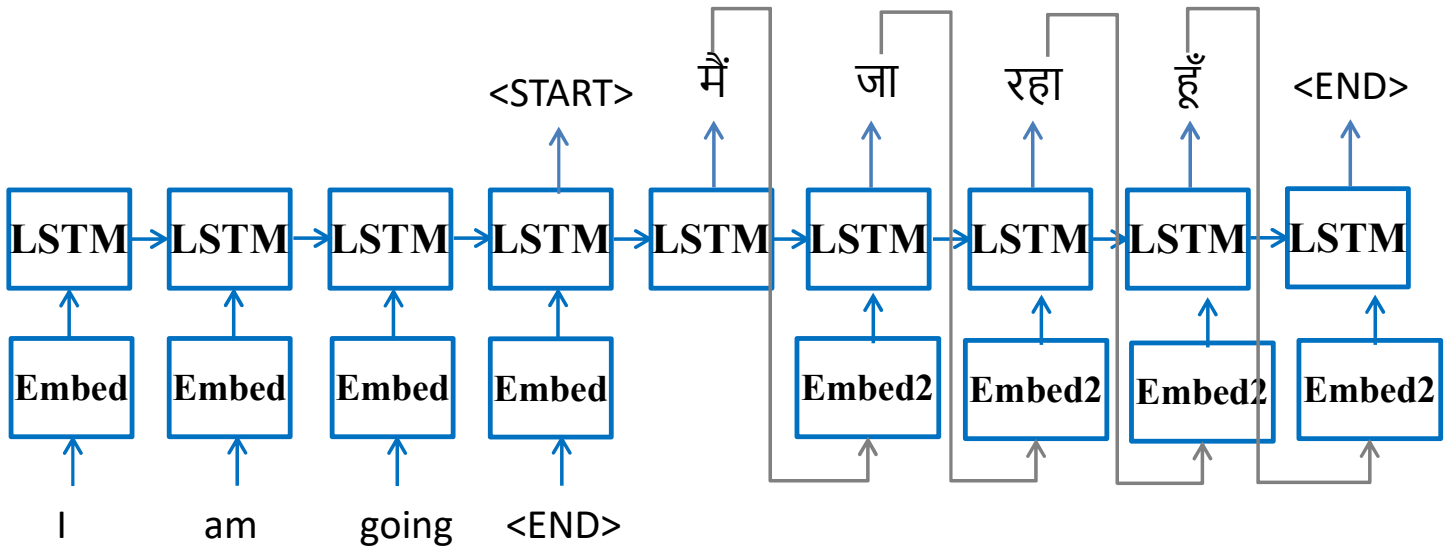
Machine translation

- A naïve model would be to use a many-to-many network and directly train it



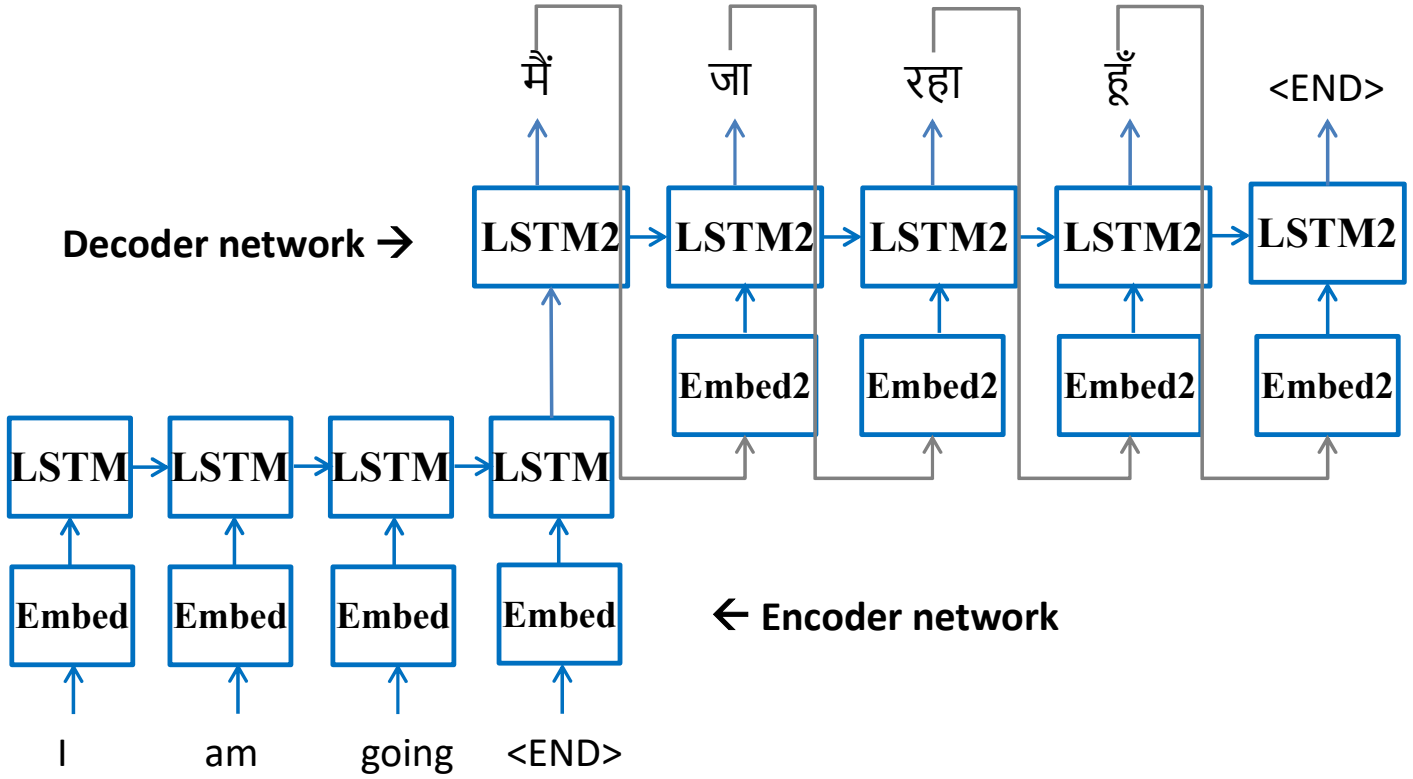
Machine translation

- One could also feed in the output to the next instance input to predict a coherent structure

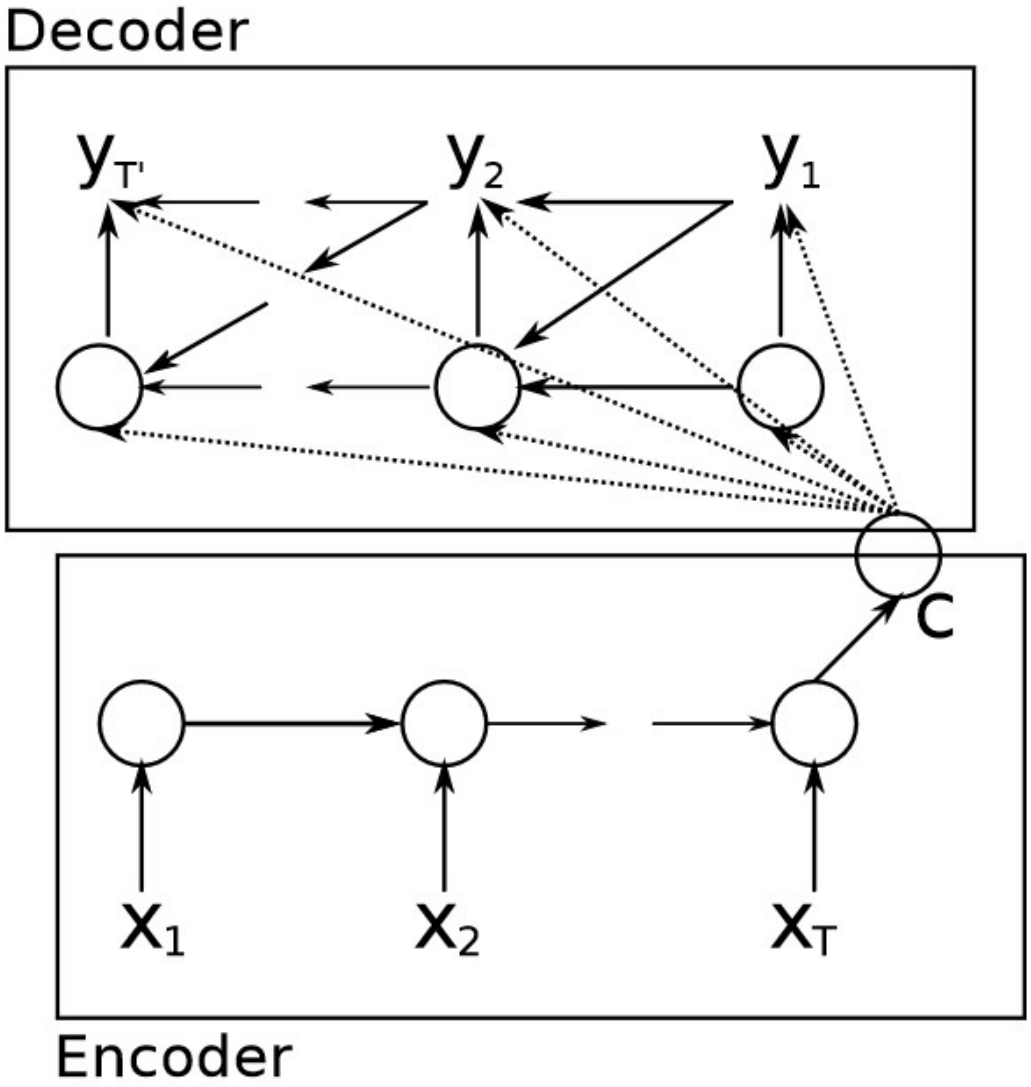


Machine translation

- In actuality, one would use separate LSTMs pre-trained on two different languages



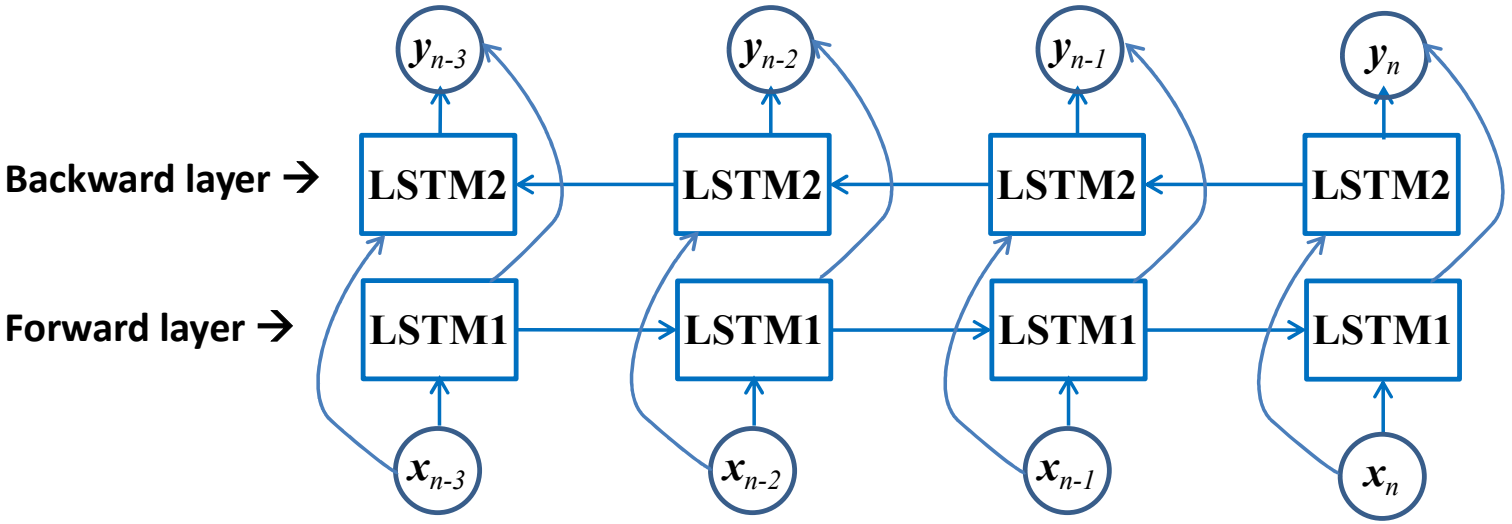
Machine translation using encoder-decoder



Source: “Learning Phrase Representations using RNN Encoder–Decoderfor Statistical Machine Translation”, Cho et al., ArXiv 2014

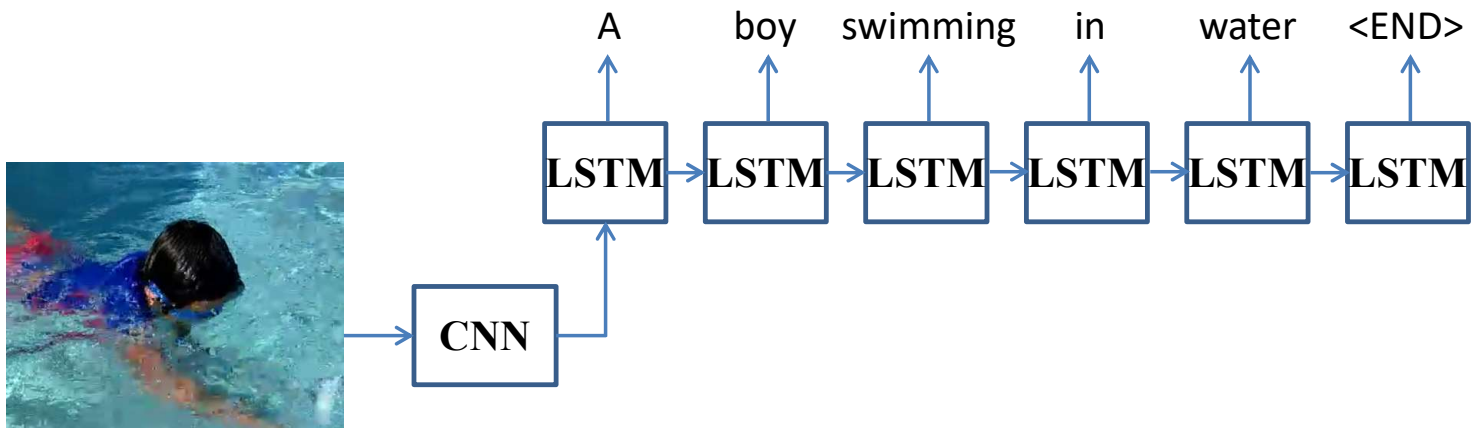
Bi-directional LSTM

- Many problems require a reverse flow of information as well
- For example, POS tagging may require context from future words



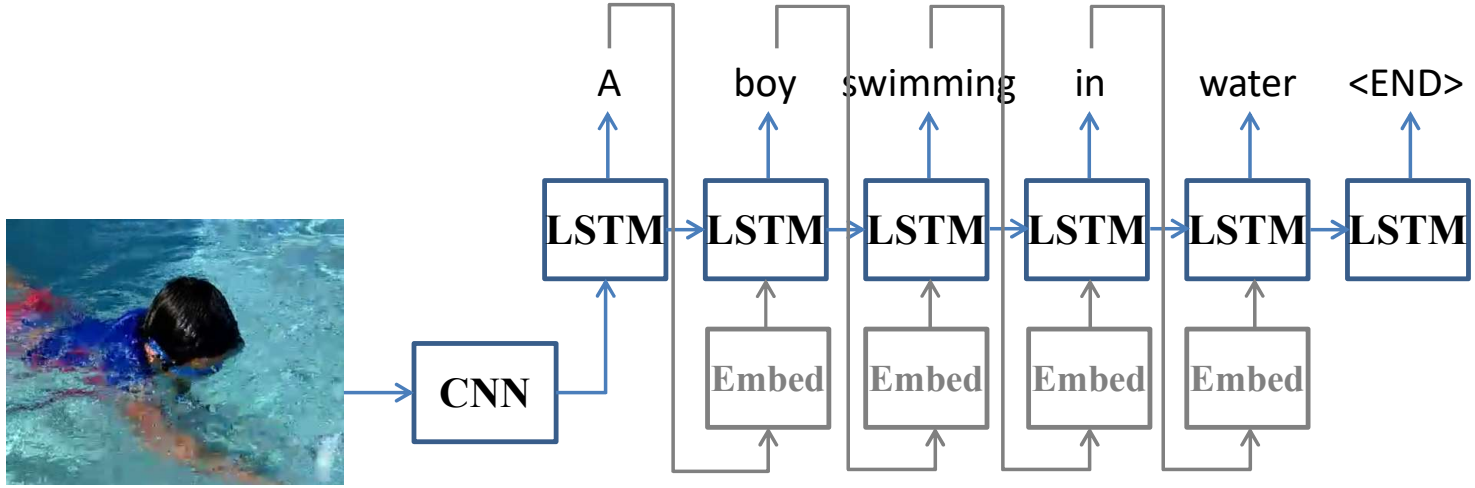
Sentence generation

- Very common for image captioning
- Input is given only in the beginning
- This is a one-to-many task

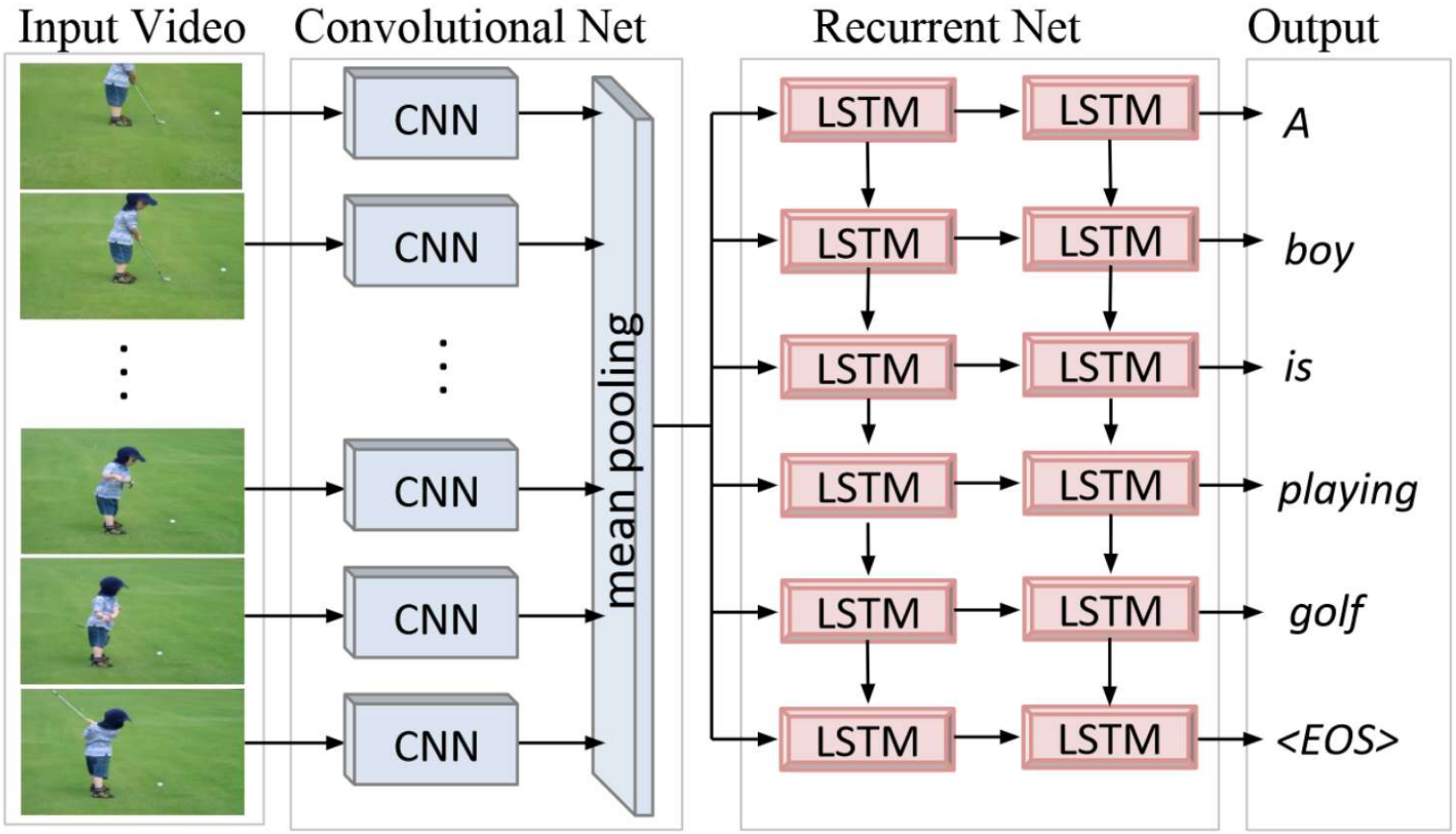


Sentence generation

- Very common for image captioning
- Input is given only in the beginning
- This is a one-to-many task



Video Caption Generation



Source: "Translating Videos to Natural Language Using Deep Recurrent Neural Networks", Venugopal et al., ArXiv 2014

Some problems in LSTM and its troubleshooting

- Inappropriate model
  - Identify the problem: One-to-many, many-to-one etc.
  - Loss only for outputs that matter
  - Separate LSTMs for separate languages
- High training loss
  - Model not expressive
    - Too few hidden nodes
    - Only one hidden layer
- Overfitting
  - Model has too much freedom
    - Too many hidden nodes
    - Too many blocks
    - Too many layers
    - Not bi-directional

Multi-dimensional RNNs

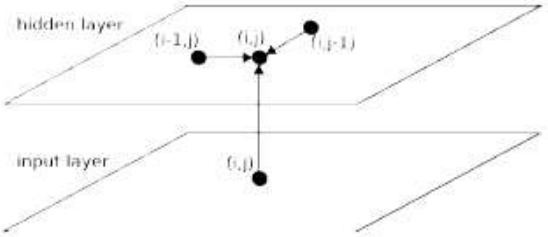


Figure 8.1: MDRNN forward pass

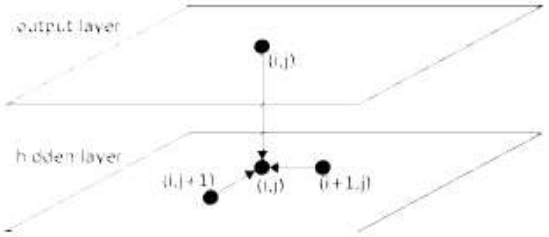


Figure 8.2: MDRNN backward pass

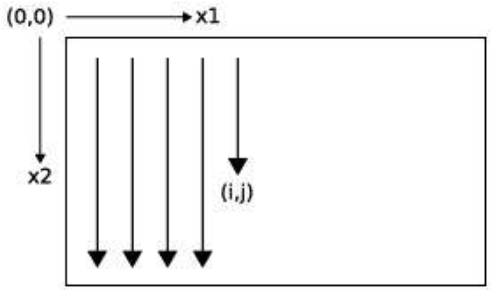


Figure 8.3: **Sequence ordering of 2D data.** The MDRNN forward pass starts at the origin and follows the direction of the arrows. The point (i,j) is never reached before both (i-1,j) and (i,j-1).

Source: "*Supervised Sequence Labelling with Recurrent Neural Networks*." by Graves, Alex.

In summary, LSTMs are powerful

- Using recurrent connections is an old idea
- It suffered from lack of gradient control over long term
- *CEC* was an important innovation for remembering states long term
- This has many applications in time series modeling
- Newer innovations are:
  - *Forget* gates
  - *Peepholes*
  - Combining input and forget gate in *GRU*
- LSTM can be generalized in direction, dimension, and number of hidden layers to produce more complex models