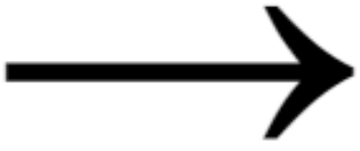
# Lecture 7: RNNs and Distributed Word Reps

# Word Representations

**Consider a simple classification task on text**

"I love this movie.
I've seen it many times
and it's still awesome."



**ML model**

"This movie is bad.
I don't like it it all.
It's terrible."

**ML model**

**How to represent text as input to ML models**

# Word Representations

for vocabulary size $D{=}10$, the one-hot vector of word ID $w{=}4$ is

$$\mathrm{e}(w) = [\,0\;0\;0\;1\;0\;0\;0\;0\;0\;0\,]$$

The major problem with the one-hot representation is that it is very high-dimensional

- the dimensionality of $\mathrm{e}(w)$ is the size of the vocabulary
- a typical vocabulary size is $\approx 100\ 000$
- a window of 10 words would correspond to an input vector of at least $1\ 000\ 000$ units!

**Slide Credit Hugo Larochelle**

# Distributed Word Representations

- A common approach is to associate each discrete object with a “dense” continuous vector

| Word | $w$ | $C(w)$ |
|---|---|---|
| “the” | 1 | [ 0.6762, -0.9607, 0.3626, -0.2410, 0.6636 ] |
| “a” | 2 | [ 0.6859, -0.9266, 0.3777, -0.2140, 0.6711 ] |
| “have” | 3 | [ 0.1656, -0.1530, 0.0310, -0.3321, -0.1342 ] |
| “be” | 4 | [ 0.1760, -0.1340, 0.0702, -0.2981, -0.1111 ] |
| “cat” | 5 | [ 0.5896, 0.9137, 0.0452, 0.7603, -0.6541 ] |
| “dog” | 6 | [ 0.5965, 0.9143, 0.0899, 0.7702, -0.6392 ] |
| “car” | 7 | [ -0.0069, 0.7995, 0.6433, 0.2898, 0.6359 ] |
| ... | ... | ... |

**Slide Credit Hugo Larochelle**

- We can now try to make meaningful distances in vector space by learning C(w)

# Distributed vs Local

- Distributed representations for symbol -> using dense vs local (sparse)

| | | |
|---|---|---|
| “ the ” | 1 | [ 0.6762, -0.9607, 0.3626, -0.2410, 0.6636 ] |

**Vs** [ 0 0 0 1 0 0 0 0 0 0 ]

- Note: the word “distributed” is sometimes used in slightly different context in deep learning literature

# Pytorch Embedding

```
import torch.nn as nn
import torch

words = 5
dim = 100

embed = nn.Embedding(words,dim)

embed(torch.LongTensor([1]))
```

```
tensor([[-0.6355, -0.2671,  0.3560,  0.2689,  0.2781, -1.3727,  0.7757,  2.0445,
          1.4461,  0.3756, -0.0162,  1.1468, -1.1845,  0.8442, -1.3146,  0.1710,
          0.5611, -0.8307,  1.1716, -1.3098, -2.9567,  0.4804,  1.4355, -0.3178,
          0.2934,  1.1778,  1.2912,  0.1803, -1.9347,  1.4485,  0.0195, -0.8828,
          1.2395,  0.2929, -1.9109, -0.6097, -1.4886, -0.5300,  0.7930,  1.2733,
         -0.4736,  0.0200, -0.0703,  0.1745, -2.8354,  1.3762, -0.7413, -0.4601,
         -0.4509, -0.7949,  0.0967, -0.6718, -0.4257,  0.0081, -0.6909, -0.2110,
         -0.3340, -0.2481,  0.1715,  0.4038, -1.9862,  1.5716,  0.8283, -0.3971,
         -0.2930,  1.4967, -0.0121, -0.3886,  0.2584, -0.1783,  0.2082,  2.0378,
         -1.5603, -0.9889, -0.7937, -0.4811,  0.5150, -0.6355,  0.0782,  0.6339,
          0.6144,  0.4604,  1.6851,  0.6491, -1.2013, -0.3850, -0.7349,  1.4328,
         -0.9501, -0.5964,  0.0910,  0.6858, -0.9850, -2.5020, -0.6703,  0.8197,
          0.9445,  0.2693, -0.5862, -0.9350]], grad_fn=<EmbeddingBackward>)
```

```
embed.weight.shape
```

```
torch.Size([5, 100])
```

- Embedding class is essentially a weight matrix where the forward indexes the rows
- Can be optimized for large vocabulary

# Language Modeling

- Language modelling is concerned with capturing the statistical properties of text

- A common objective used for learning representations of words and sentences is to predict the distribution for the next word or the missing word

**I love this movie and I've seen it many _____**

$$p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$$

$$P(w_t = k | w_{t-n+1}, \ldots w_{t-1}) = \frac{e^{a_k}}{\sum_{l=1}^{N} e^{a_l}}$$

**I love this _____ and I've seen it many times**

$$L(\theta) = \sum_t \log P(w_t | w_{t-n+1}, \ldots w_{t-1}).$$

# Simple Neural Language Model

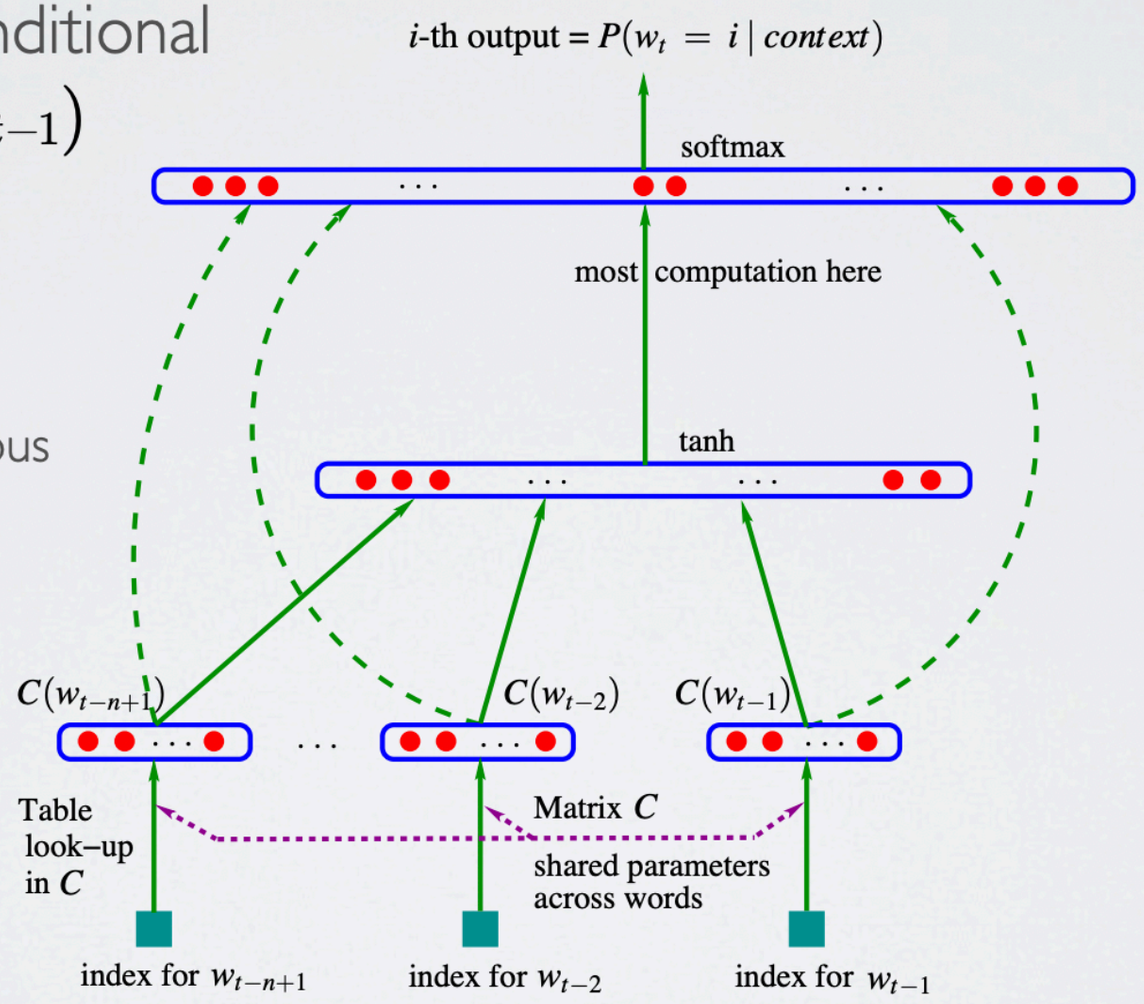
Solution: model the conditional

$p(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1})$

with a neural network

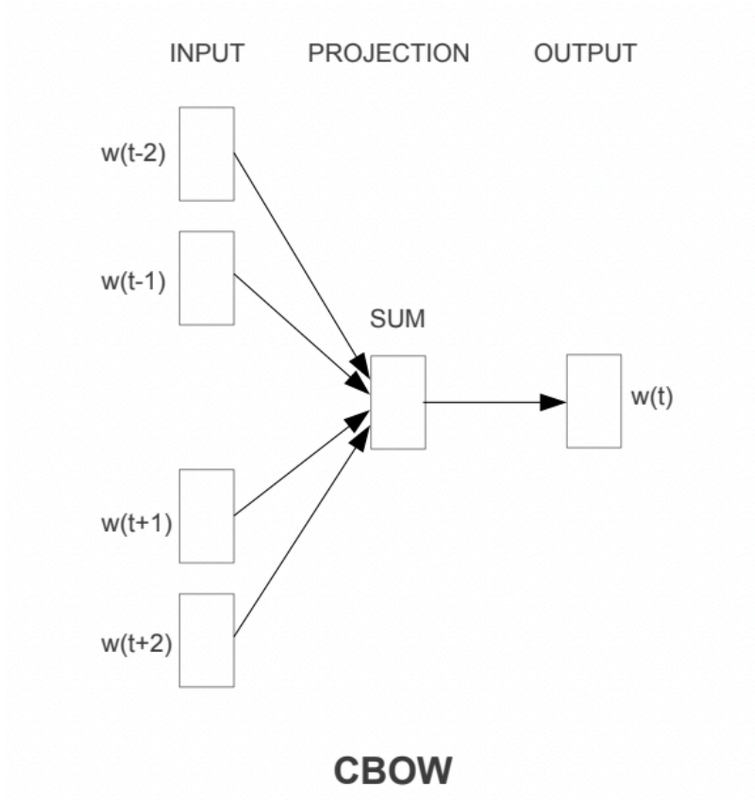
- learn word representations to allow transfer to $n$-grams not observed in training corpus

Bengio, Ducharme, Vincent and Jauvin, 2003




**Slide Credit Hugo Larochelle**

# Word2vec



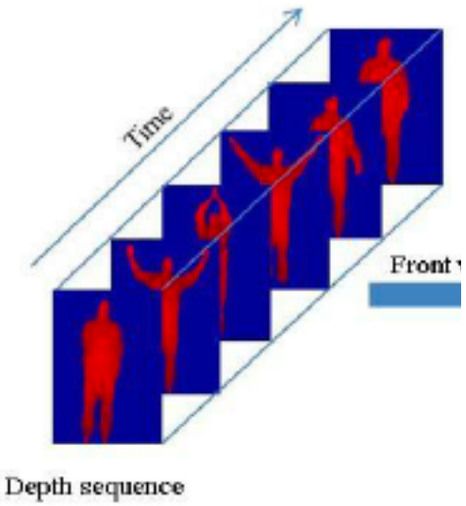

- Learns useful features using a simple linear model

**What is missing in these models (Word2vec and Bengio et al) ?**

**Mikolov 2013**

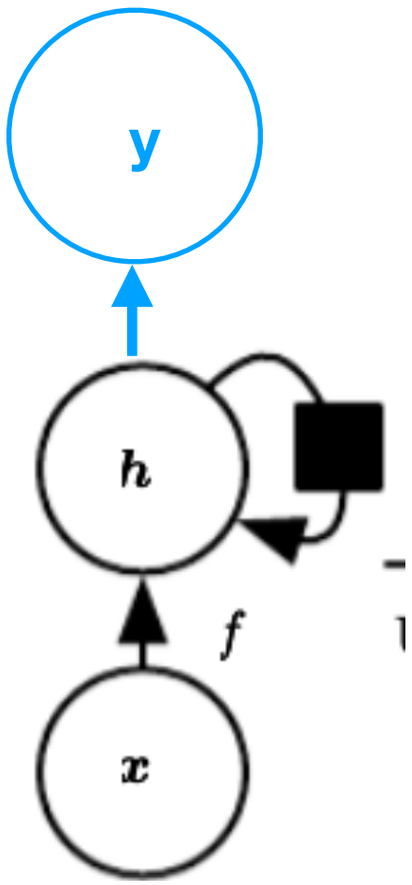
# Sequence Modeling

- Audio
- Video
- Text
- Genomes




Target Genome ATTTGCGCAGAGACCTAAGGCATTAGCTTGGCCCTAAAG

# Recurrent Neural Networks

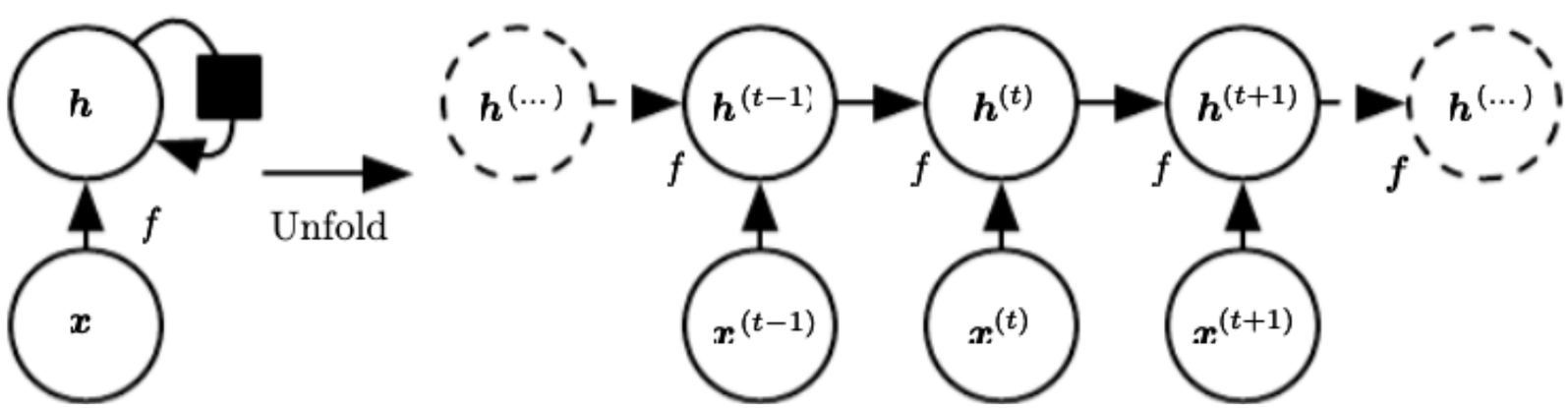



$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}),$$

# Recurrent Neural Networks

- Recurrent networks have inductive bias for sequential data
  - Have a state
  - Take input at each time-step of the sequence
  - Can be variable length - infinitely deep network
- Often helpful to think about an unrolled sequence of steps with same operations
  - Becomes analogous to feedforward network with repeated function applied

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$



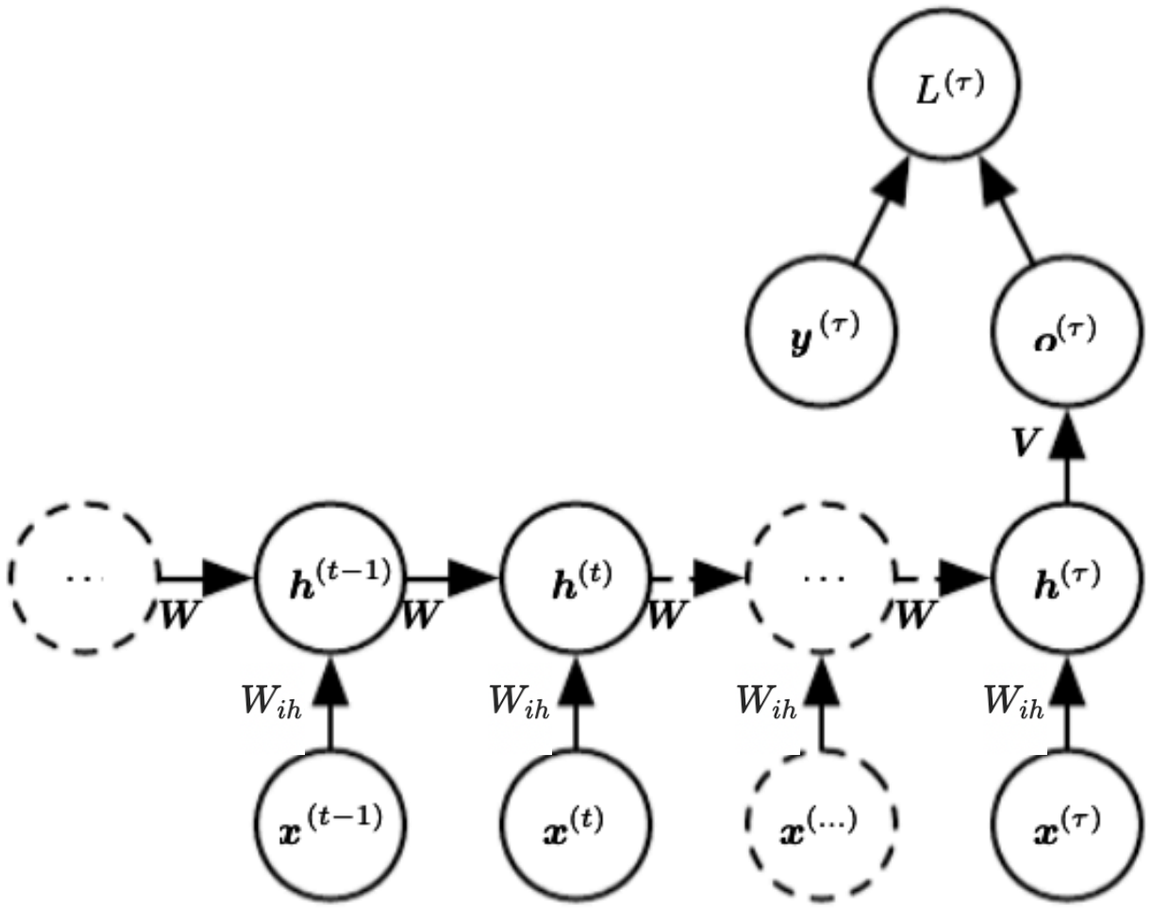
# Example Tasks

- Output at the end e.g. Sentence classification
- Output at each time step e.g. Next word prediction

# Sentence Classification Example

"I love this movie.
I've seen it many times
and it's still awesome."

"This movie is bad.
I don't like it it all.
It's terrible."



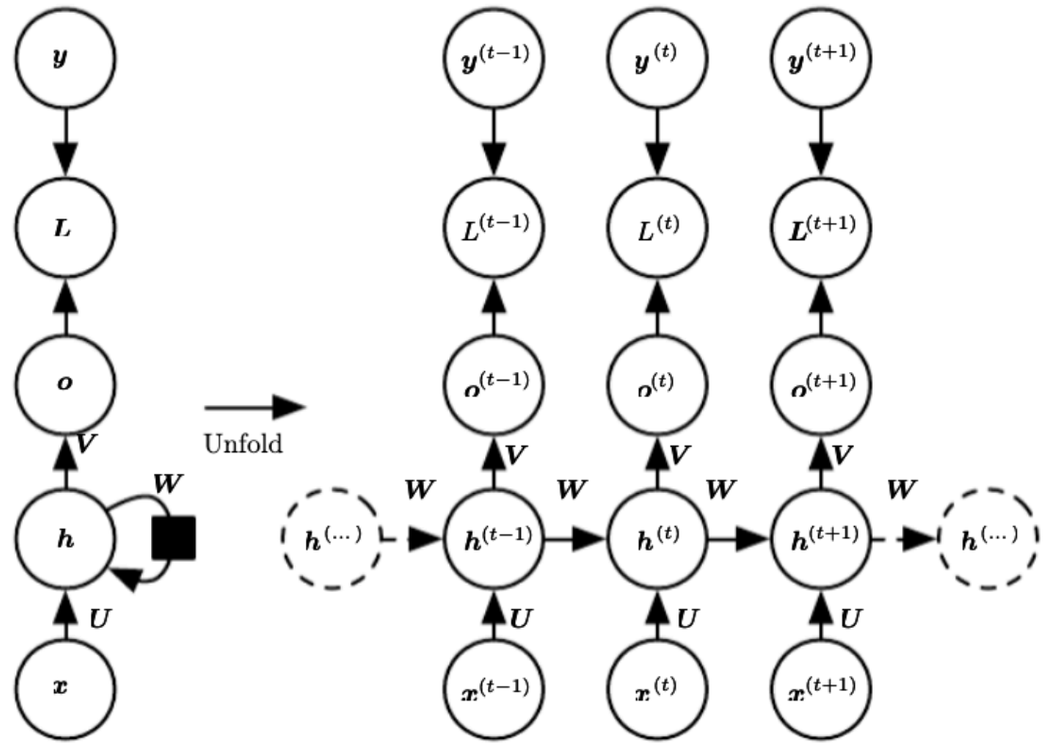

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

# RNN with Output at Each Step

- Can have output at each time step besides a state

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

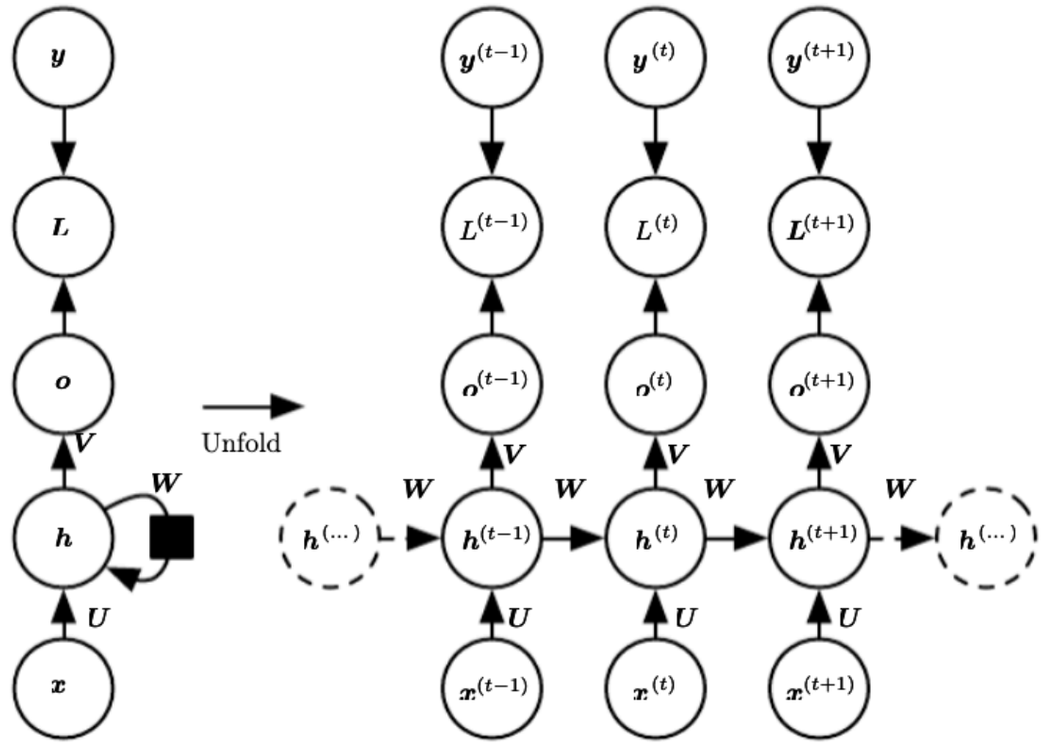


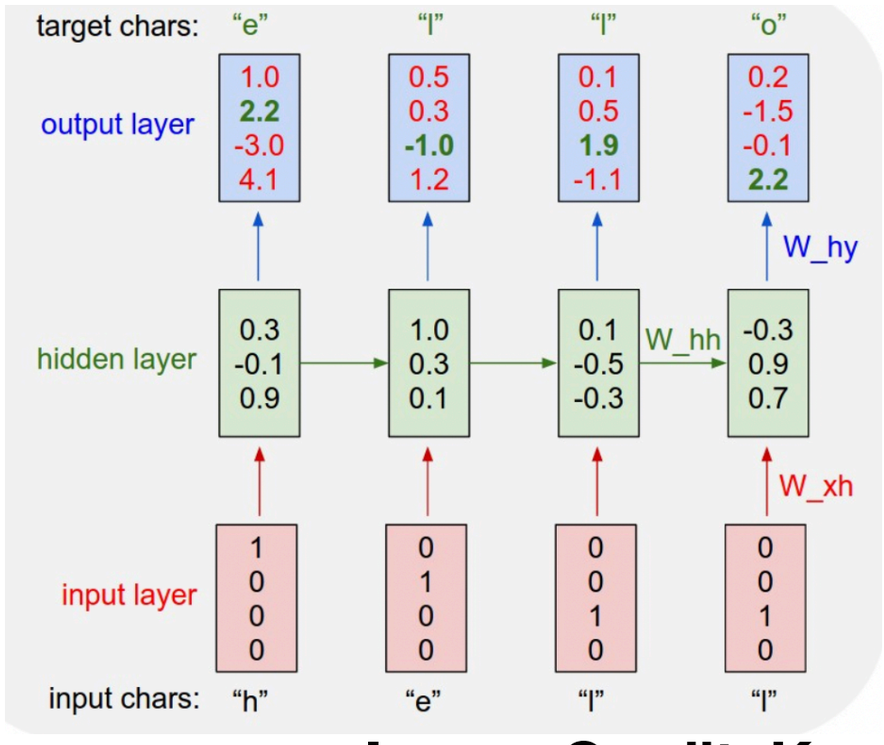

- “Deeper” RNN refers to more layers after h

# Next Char/Word Prediction

- Can have output at each time step besides a state

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$




**Loss is summed over t**

**Image Credit: Karpathy**

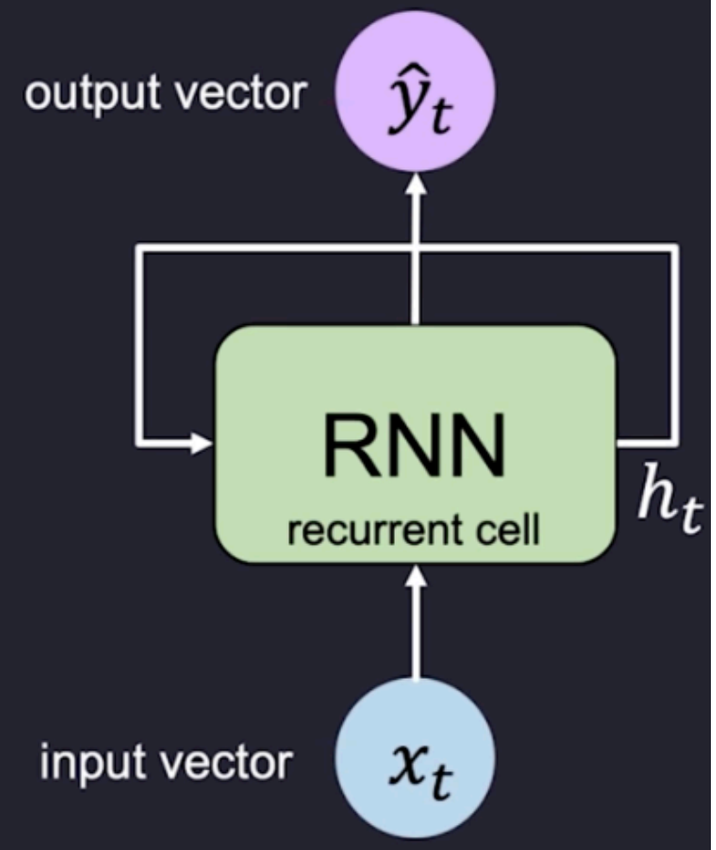
# Next Word Prediction Example

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```




**Slide Credit: Ava Soleimany**

# RNN as Generative Model

- Given a start character generate an entire sequence

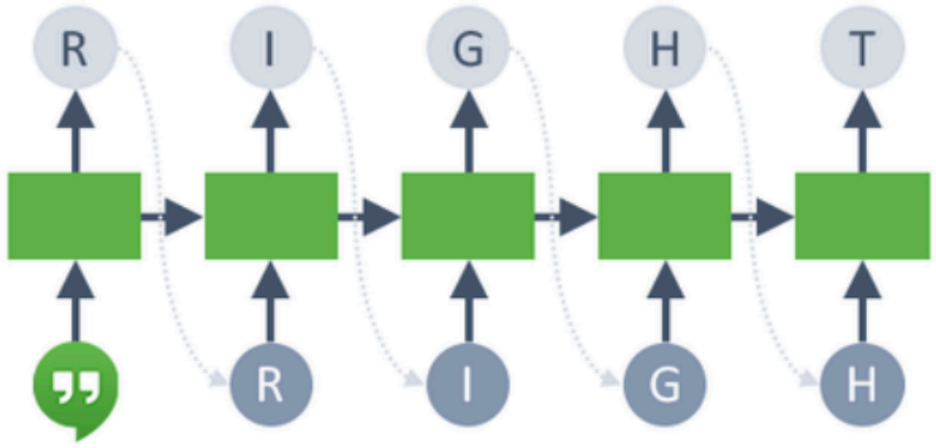
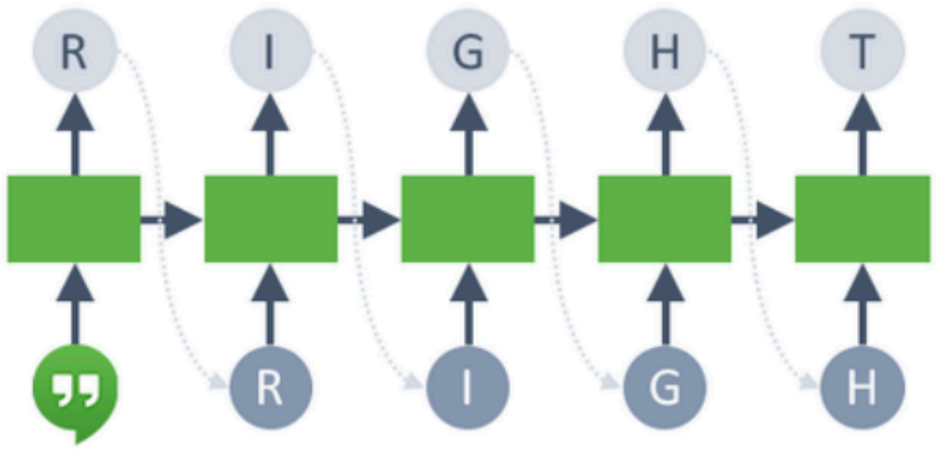



**Image credit: Lisa Zhang**

**Test time behaviour**

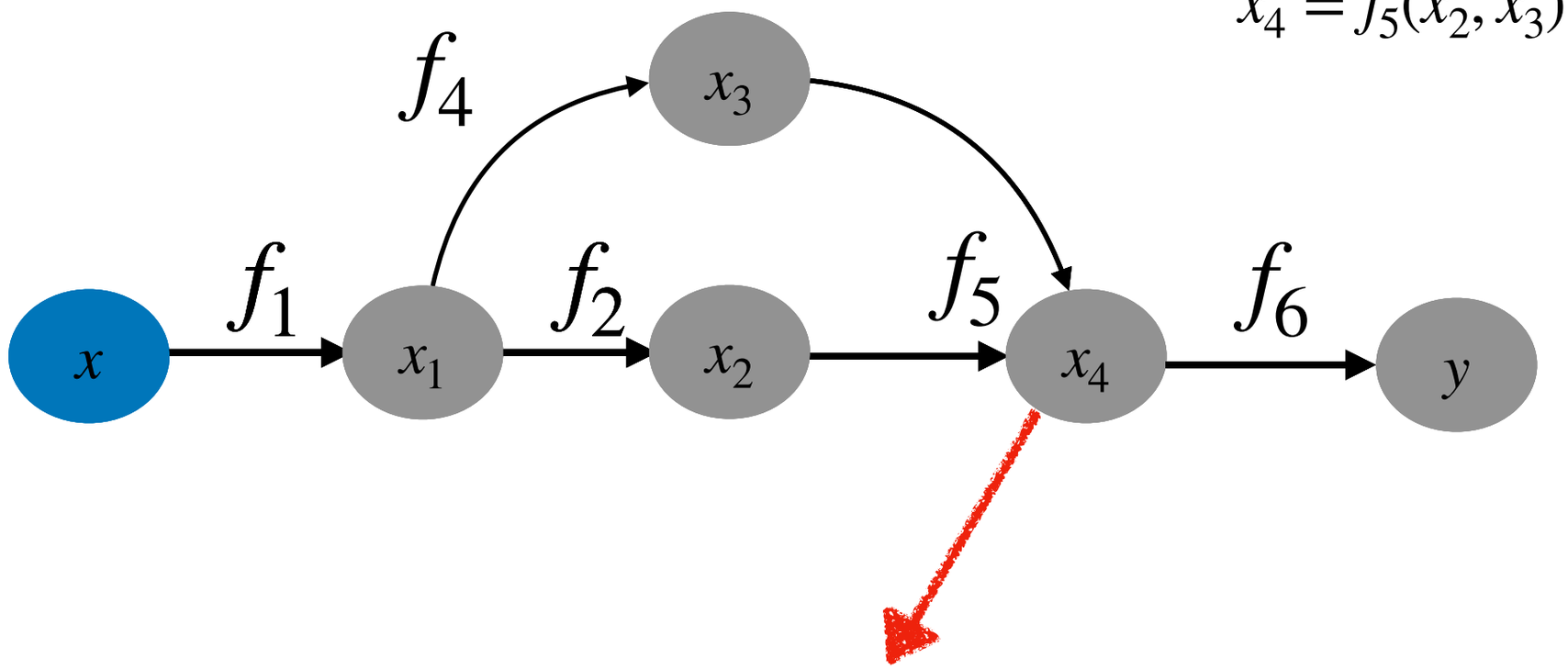
- We sample from the output distribution at each time step

# Teacher Forcing




- At training time we feed GT characters (teacher forcing)
- This can however cause an issue with training and test time distributions being different
  - MIXER, SEARN, Scheduled sampling Bengio et al
  - Similar problem arises in “imitation learning behavioral cloning” e.g. for autonomous driving

# Reminder

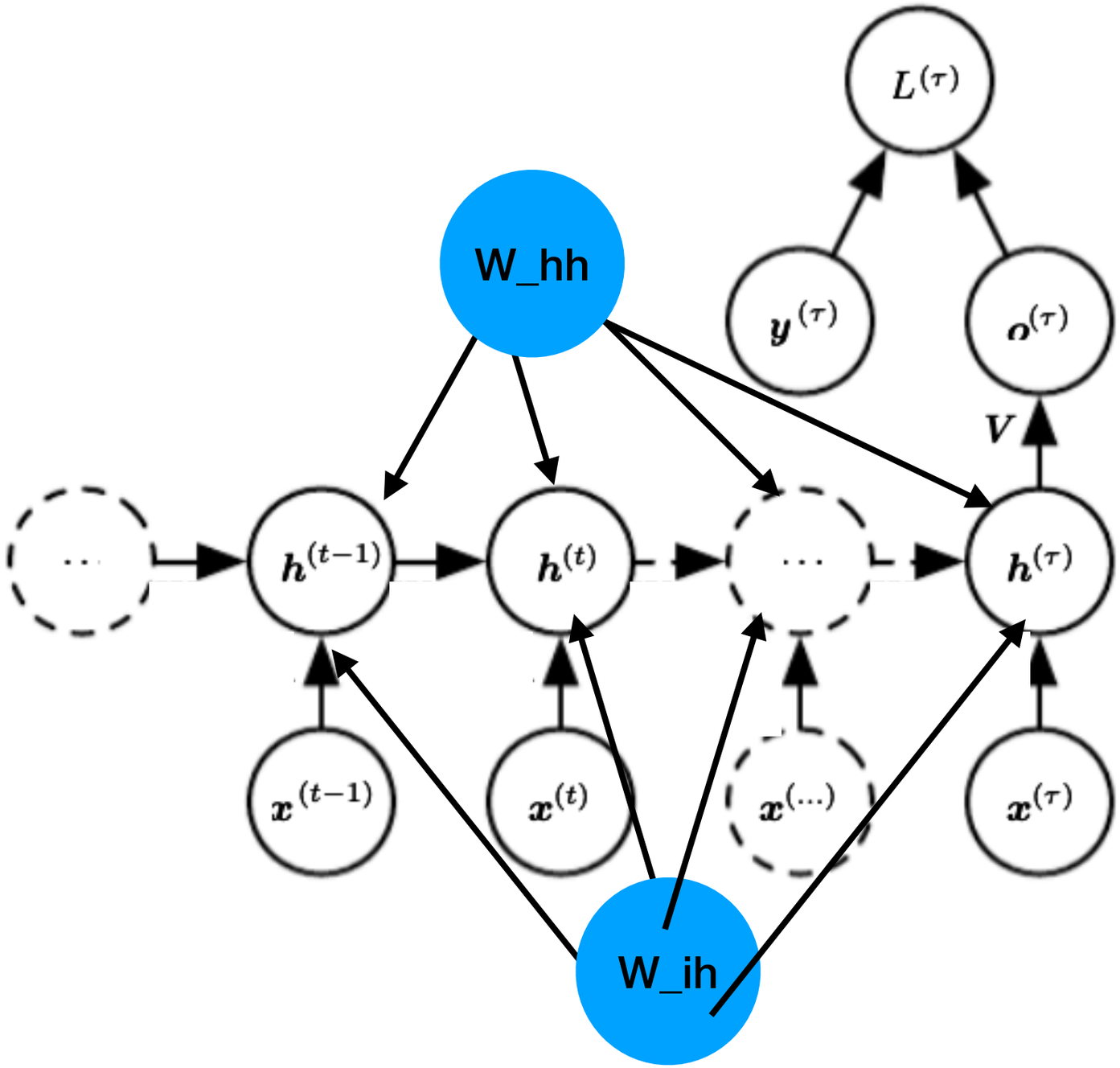



$$\frac{\partial y}{\partial \mathbf{x}_0} = \frac{\partial y}{\partial \mathbf{x}_4}\left(\frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_2}\frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} + \frac{\partial \mathbf{x}_4}{\partial \mathbf{x}_3}\frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_1}\right)\frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}$$

$$\frac{\partial y}{\partial \mathbf{x}_j} = \sum_{s \in Child(j)} \frac{\partial y}{\partial \mathbf{x}_s}\frac{\partial \mathbf{x}_s}{\partial \mathbf{x}_j}$$

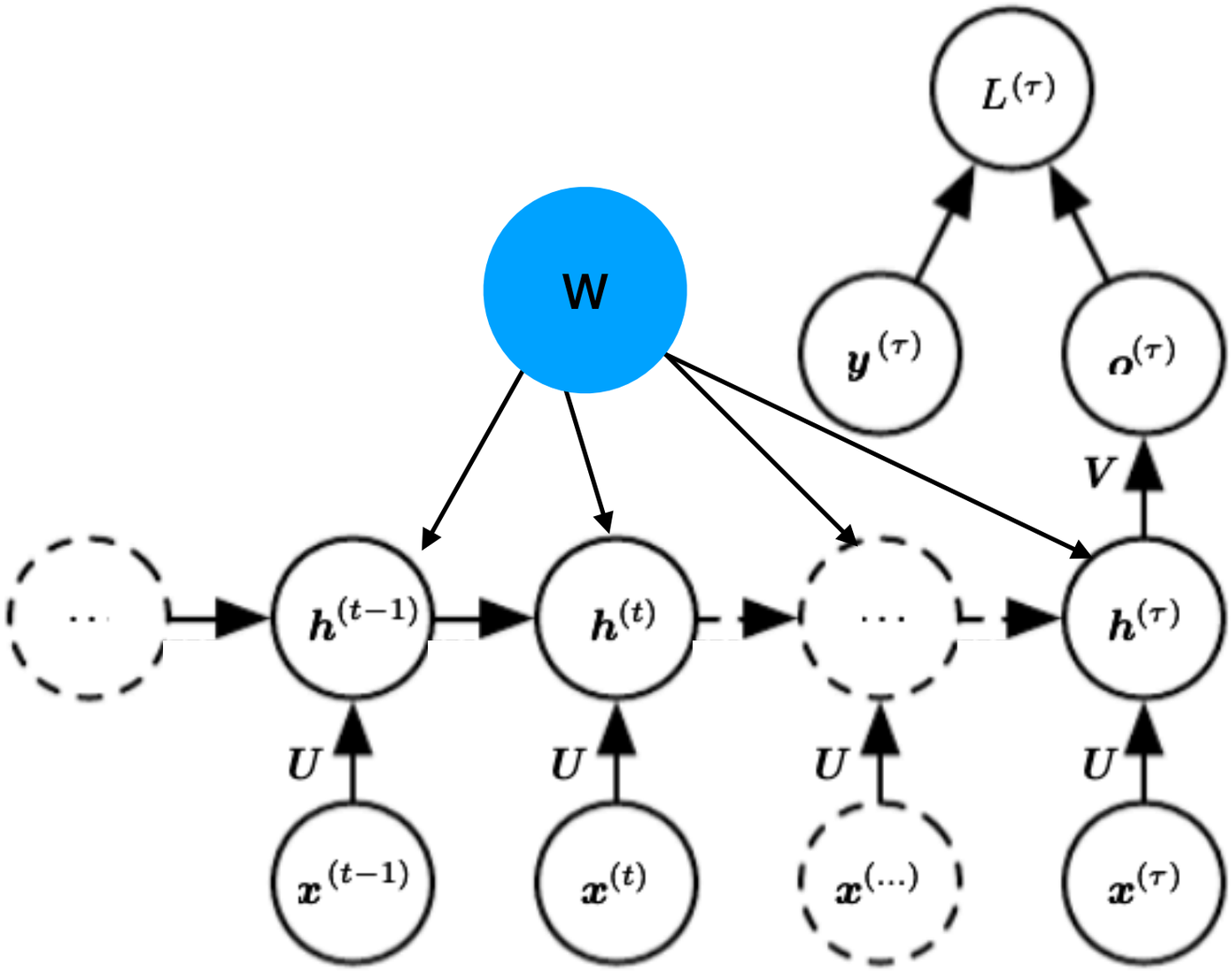
# Backpropagation Through Time

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$



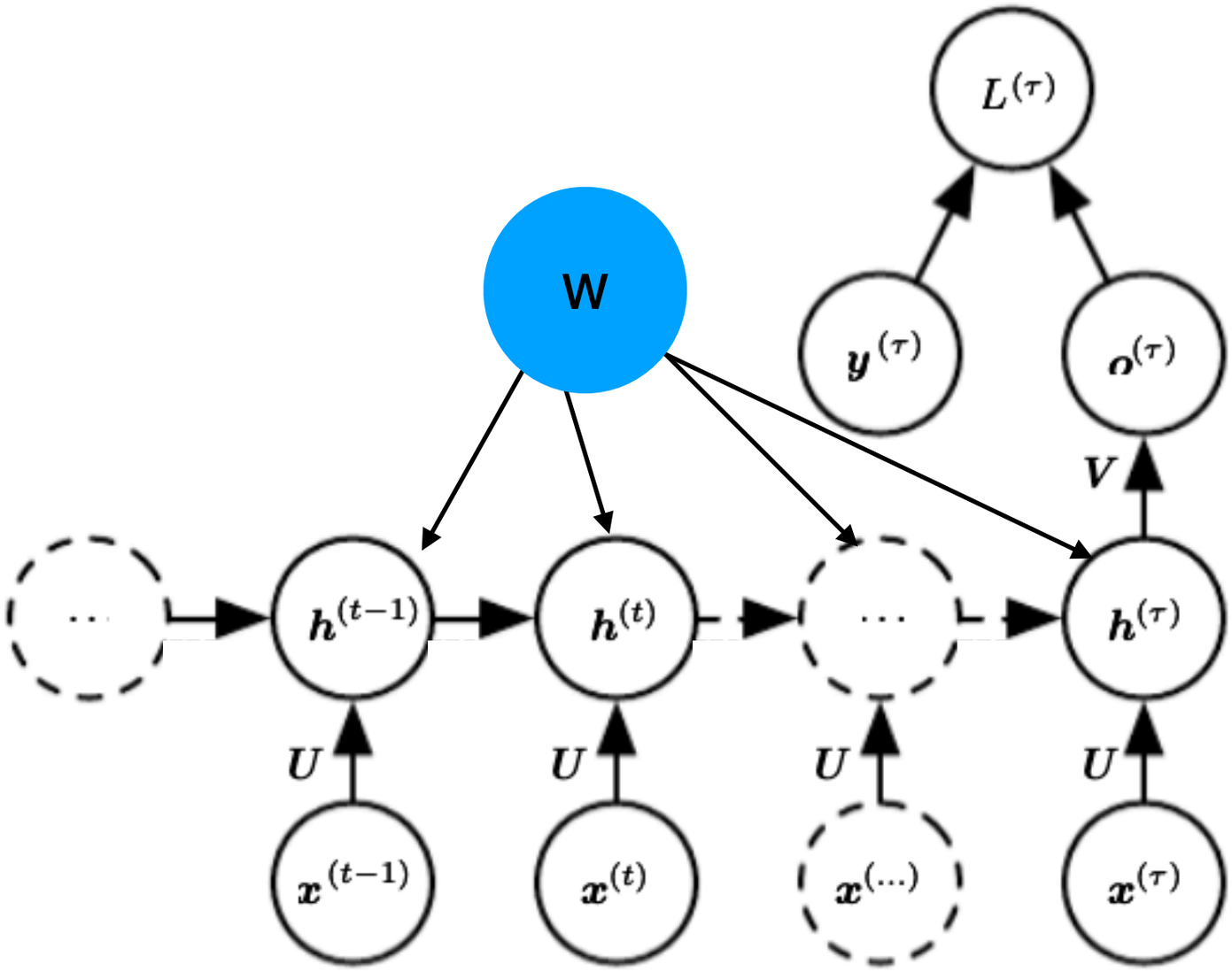

- BPTT — backprop through unrolled RNN

# Backprop and Truncated BPTT




- For long sequence memory and update cost for a single instance can be expensive
- Truncated BPTT is a technique  that simply stops backprop before reaching t=0

# Backprop and Truncated BPTT




- For long sequence memory and update cost for a single instance can be expensive
- Truncated BPTT is a technique  that simply stops backprop before reaching t=0

# Vanishing and Exploding Gradients

$$h_t = W_{hh}\rho(h_{t-1}) + W_{ih}x_t \qquad L_T = l(W_T h_T, y)$$

$$\frac{\partial L_T}{\partial \mathbf{W_{hh}}} = \frac{\partial L_T}{\partial h_T}\Pi_{i=T}\frac{\partial h_t}{\partial h_{t-1}} = \frac{\partial L_T}{\partial h_T}\Pi_{T=t..1} W_{hh}^T diag(\rho'(h_{t-1}))$$

$$\frac{\partial h_t}{\partial h_{t-1}} = W_{hh}^T diag(\rho'(h_{t-1}))$$

# Long Term Dependency

- Due to vanishing gradients it is difficult to learn long term dependencies
- Popular variants of RNNs aim to address this issue they are the LSTM and variants
- Gradient clipping also helps to avoid exploding gradients

LONG SHORT-TERM MEMORY

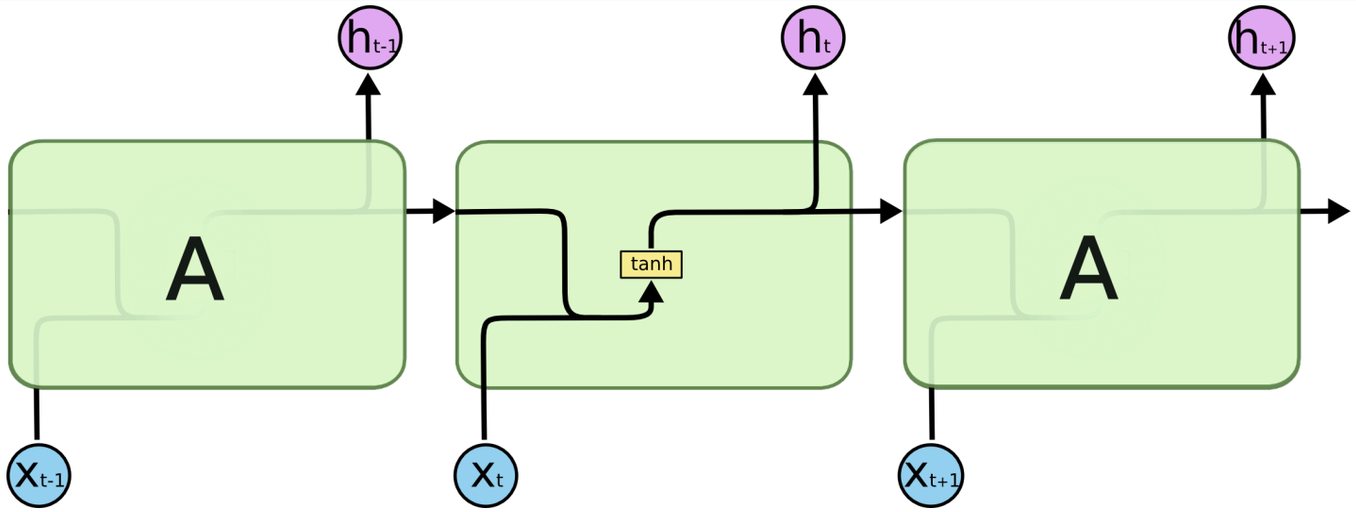
Neural Computation 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

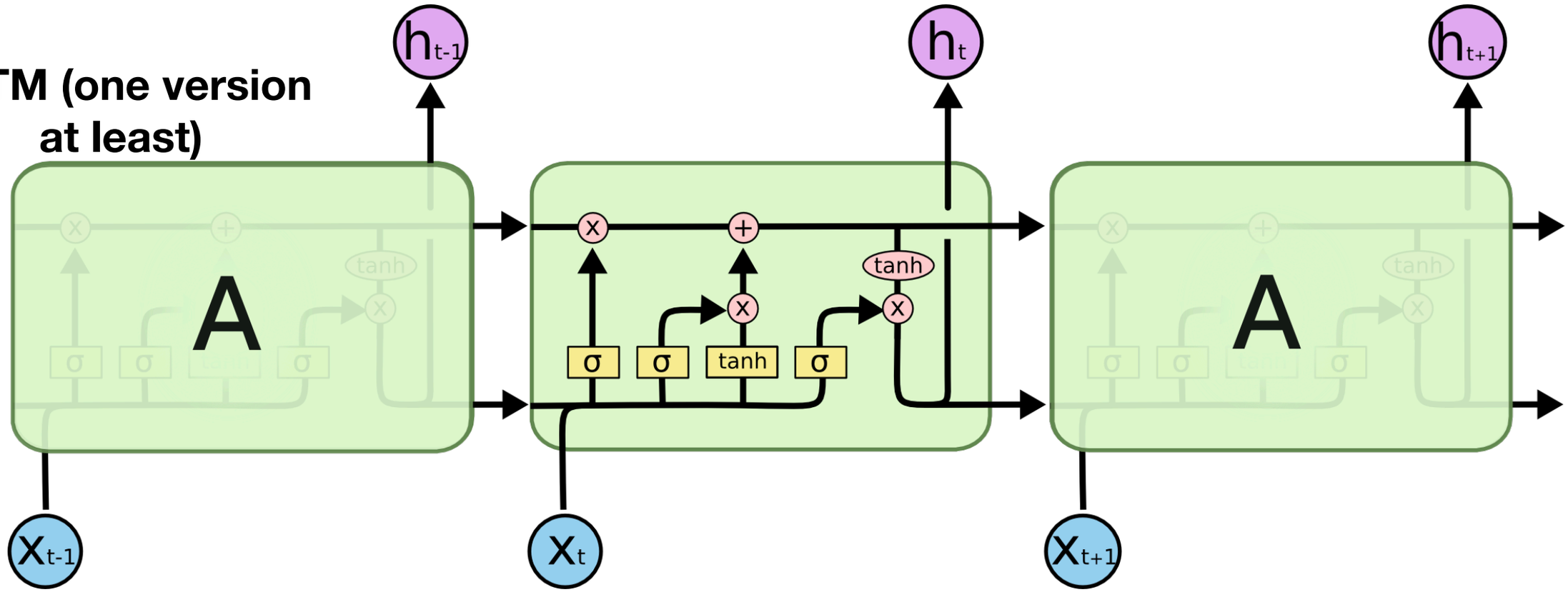
Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

# LSTMs

**Vanilla RNN**

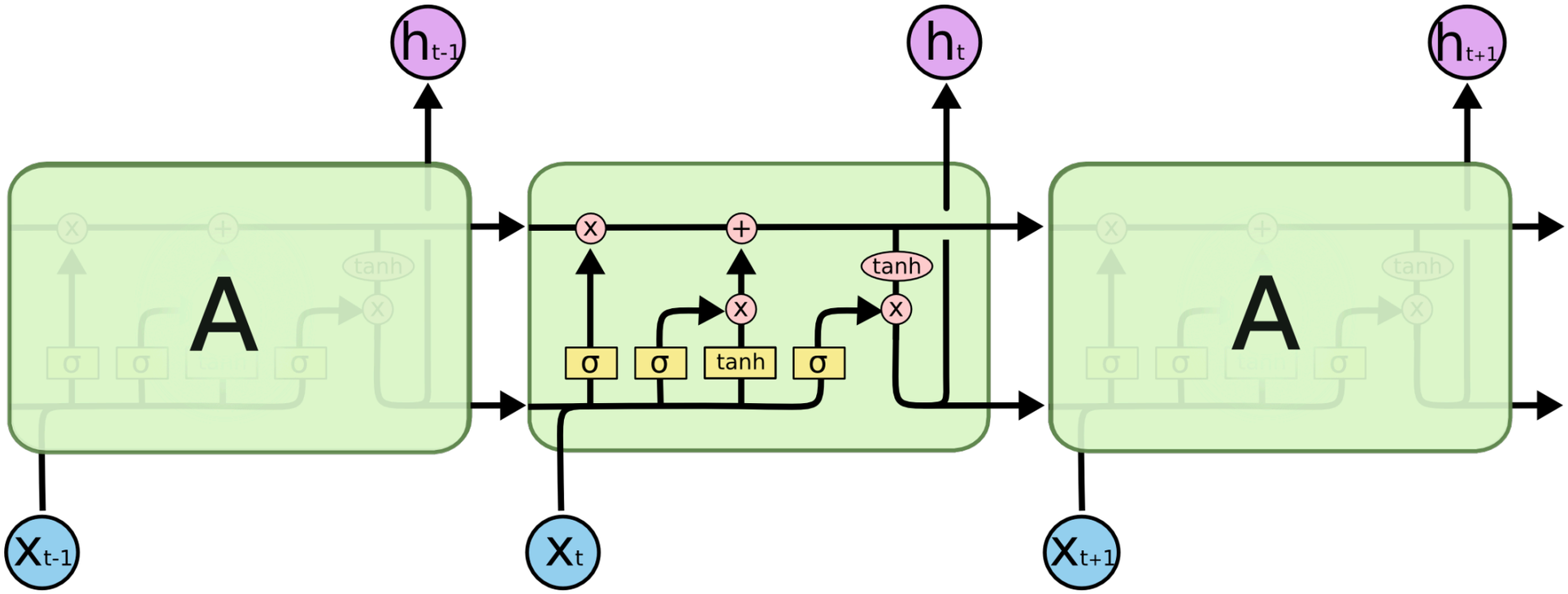



**LSTM (one version at least)**




https://colah.github.io/posts/2015-08-Understanding-LSTMs/

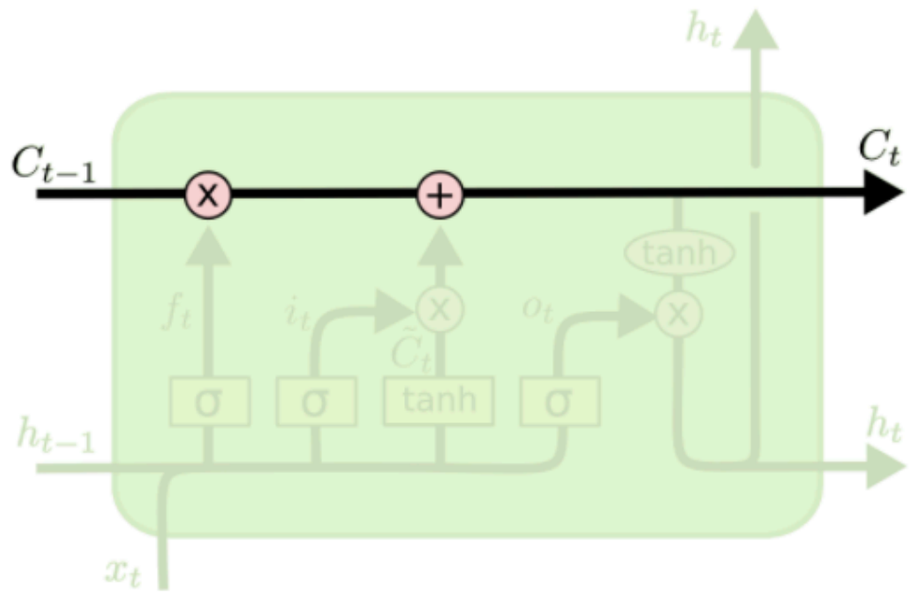
# LSTMs




- Cell state allows information to flow across many time steps modulated by gates and hidden state

- Can draw some analogies to Skip connections in ResNets

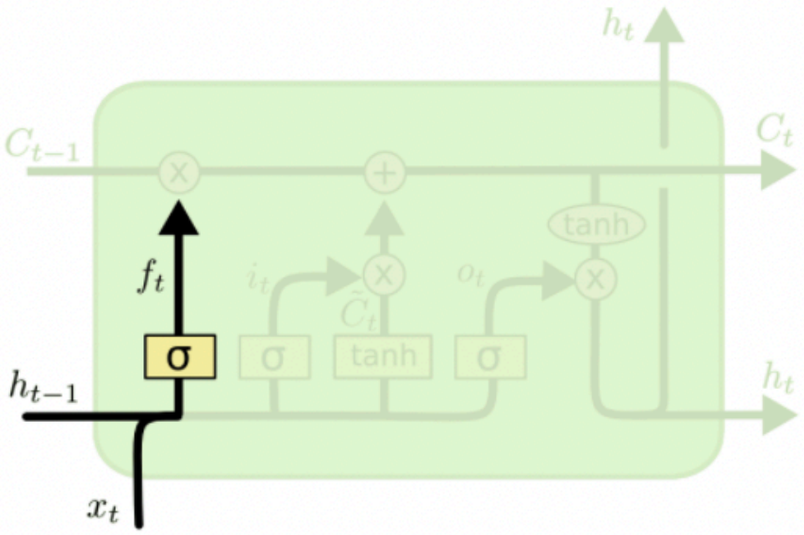
- “RNNs overwrite the hidden state, LSTM adds to it” - Ilya Sutstkever




**https://colah.github.io/posts/2015-08-Understanding-LSTMs/**

# LSTMs

**Forget gate**

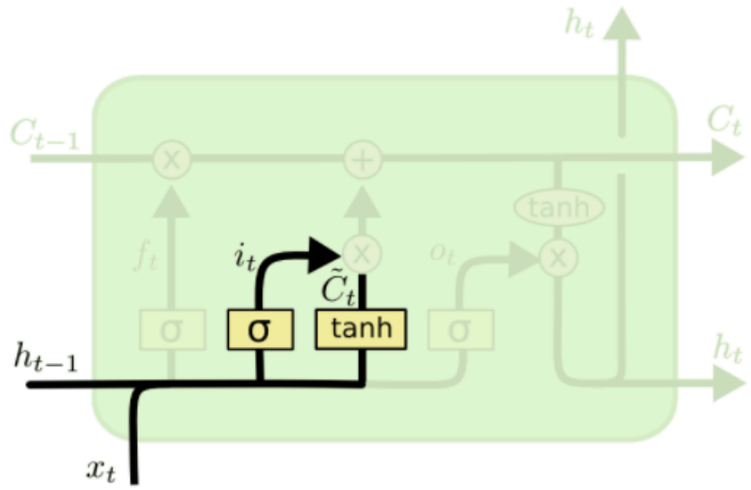



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

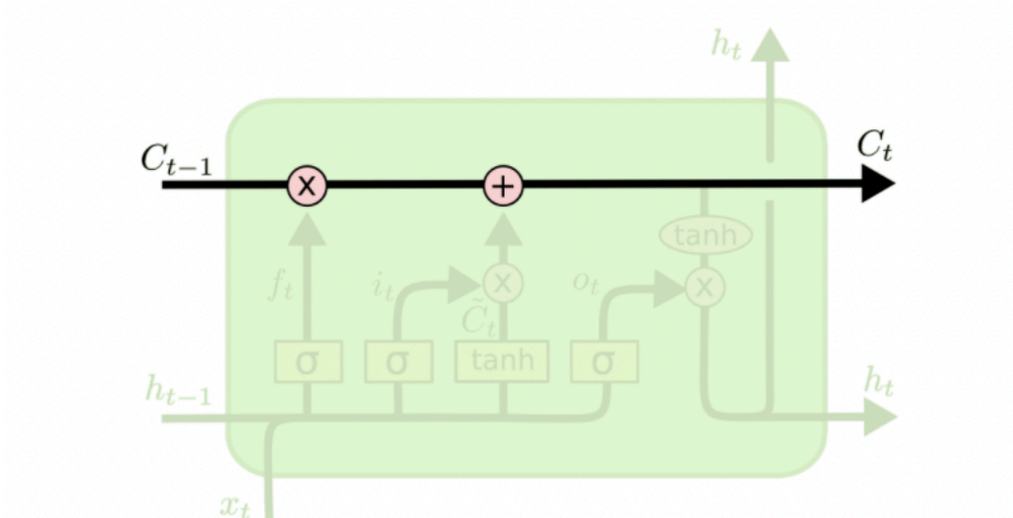
# LSTMs

## Input gate




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

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

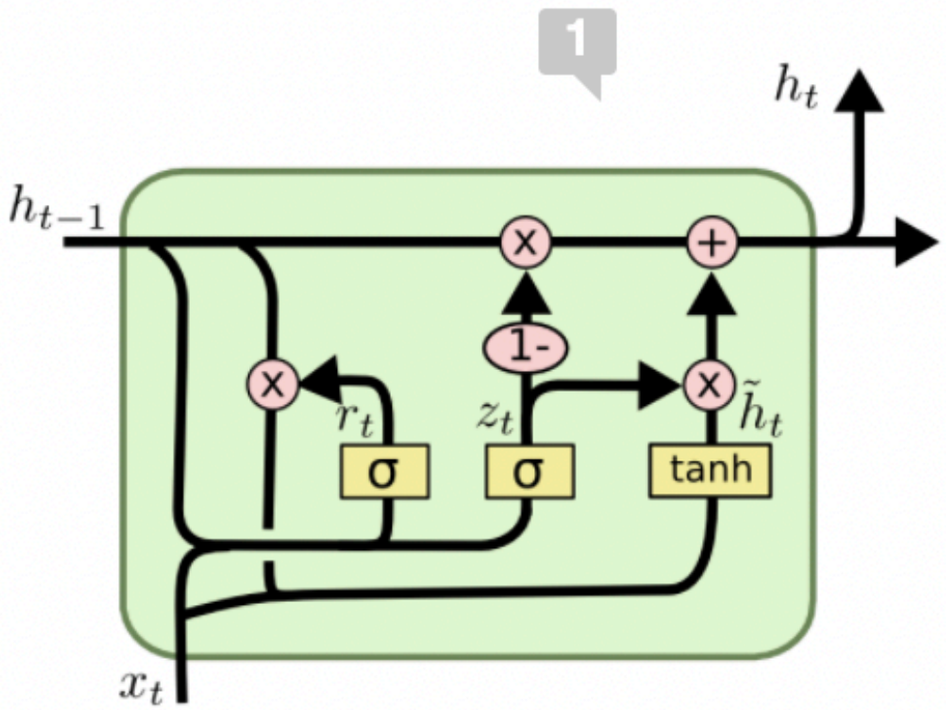



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

# GRU

**No cell state we just modulate hidden state**




$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

**https://colah.github.io/posts/2015-08-Understanding-LSTMs/**

# Issues with RNNs

- Very long term dependencies (e.g. across paragraph length) are still not well tackled by these models
  - Improved on by Memory based network and Attention based models
- Sequential processing makes them inherently slow

# RNNs are more general than FeedForward Networks

- Unlike feedforward they are Turing complete
- Can implement while and for loops
- Allow having a concept of memory
- Allow us to think about building “differentiable computers”

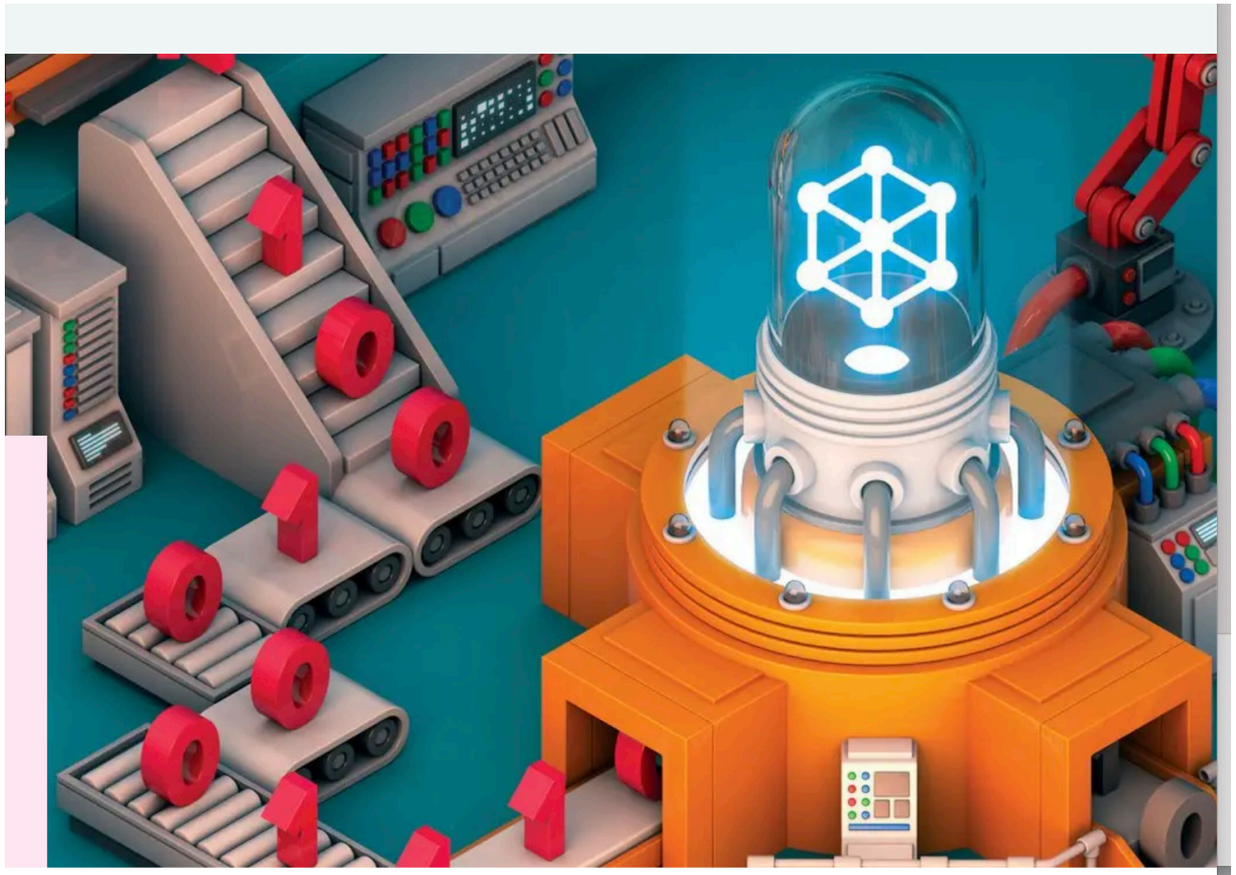


Image Credit: Deepmind

# From RNNs to Differentiable Computers

- Convnets and RNNs are just one example of how we can use network architectures to add inductive bias to our models

- A common approach in deep learning architecture design for various problems is to parametrize components of existing algorithms

- Important to make any discrete operations differentiable

- We can take the LSTM gating idea to an extreme by creating differentiable read and write operations into Memory

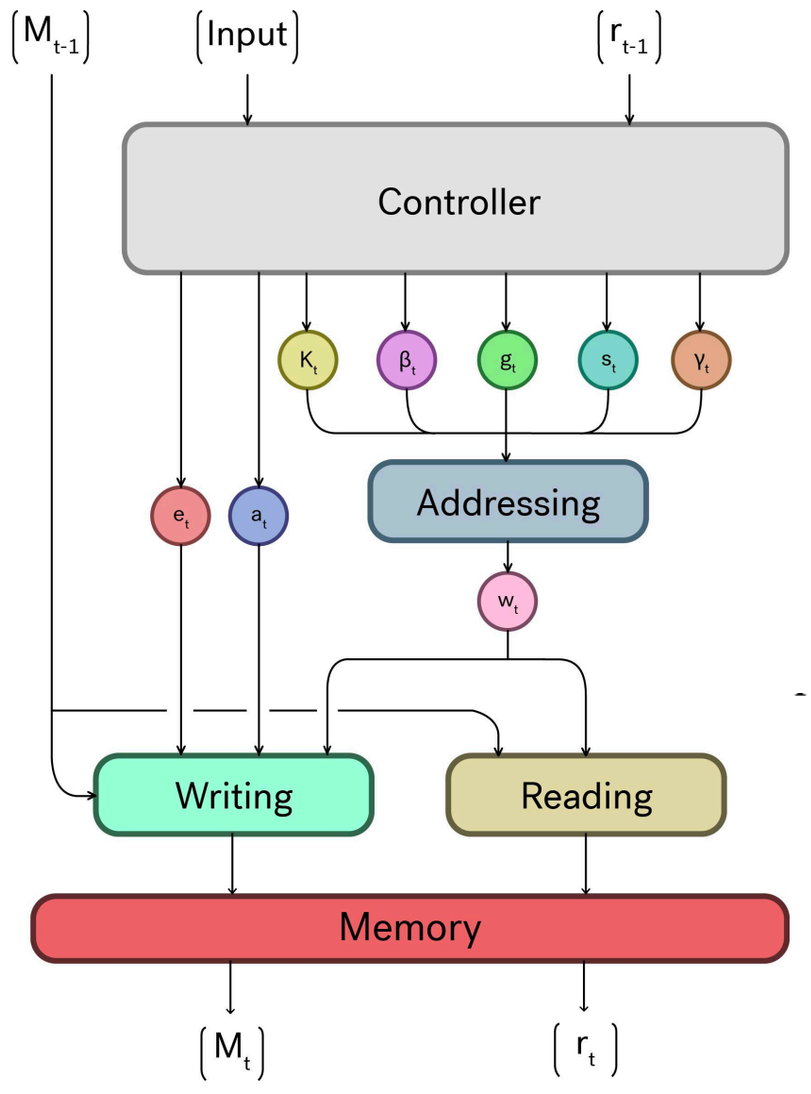
**Neural Turing Machines**


Alex Graves gravesa@google.com
Greg Wayne gregwayne@google.com
Ivo Danihelka danihelka@google.com

Google DeepMind, London, UK


MEMORY NETWORKS


**Jason Weston, Sumit Chopra & Antoine Bordes**
Facebook AI Research
770 Broadway
New York, USA
{jase,spchopra,abordes}@fb.com


$$\mathbf{r}_t \longleftarrow \sum_i w_t(i)\mathbf{M}_t(i),$$

$$\tilde{\mathbf{M}}_t(i) \longleftarrow \mathbf{M}_{t-1}(i)\left[\mathbf{1} - w_t(i)\mathbf{e}_t\right],$$

$$\mathbf{M}_t(i) \longleftarrow \tilde{\mathbf{M}}_t(i) + w_t(i)\,\mathbf{a}_t.$$

# Applications of RNNs

- Speech Recognition
- Machine Translation (before 2018)
- Video
- Many others!

# Sequence to Sequence

- Popularized in Machine Translation
- Way of thinking of encoding sequence and decoding sequence
- Efficiently trained RNNs + sequence to sequence model led to some breakthrough results in MT (circa 2014)
- Limitations on long sequences

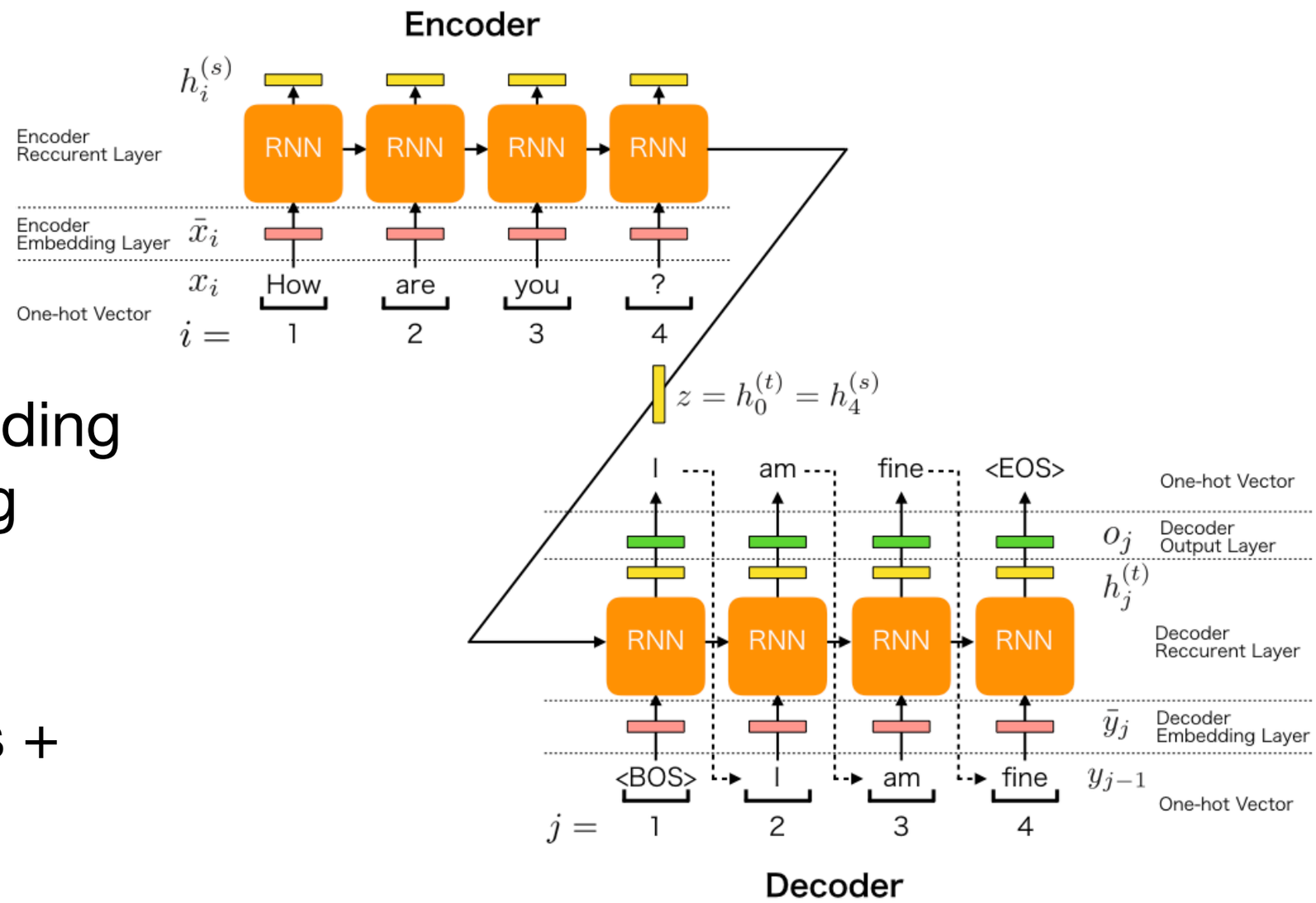



**Image source: chainer.org**

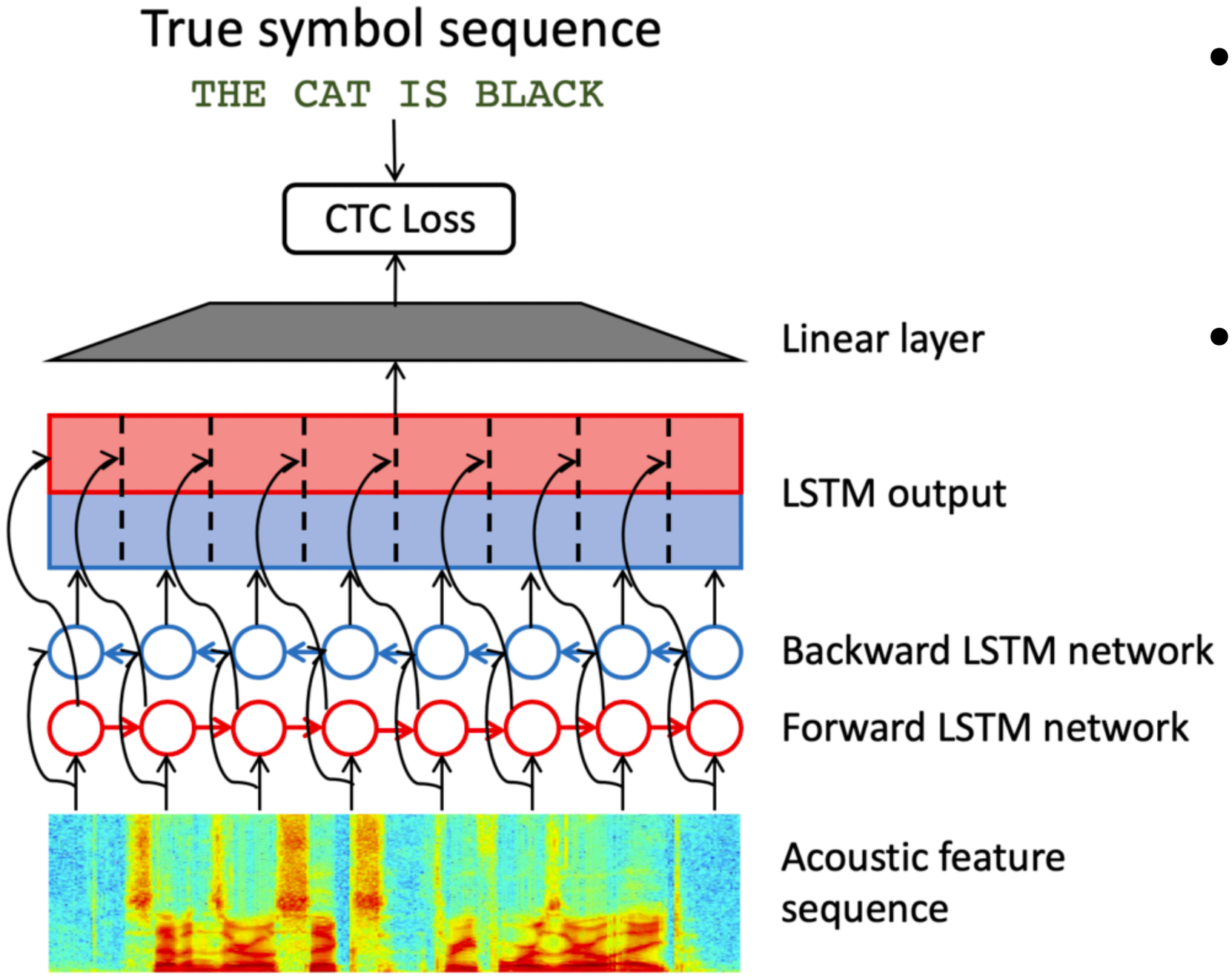
**Sequence to Sequence Learning with Neural Networks**


**Ilya Sutskever**
Google
ilyasu@google.com

**Oriol Vinyals**
Google
vinyals@google.com

**Quoc V. Le**
Google
qvl@google.com

# Speech Recognition




- Bi-directional LSTM gives extra context at each step
- Losses can go beyond sum of likelihood at each step e.g. CTC

**Slide Credit IBM Research**

# Caption Generation

## Caption Generation