Recurrent Neural Networks

Recurrent Neural Networks (RNNs)

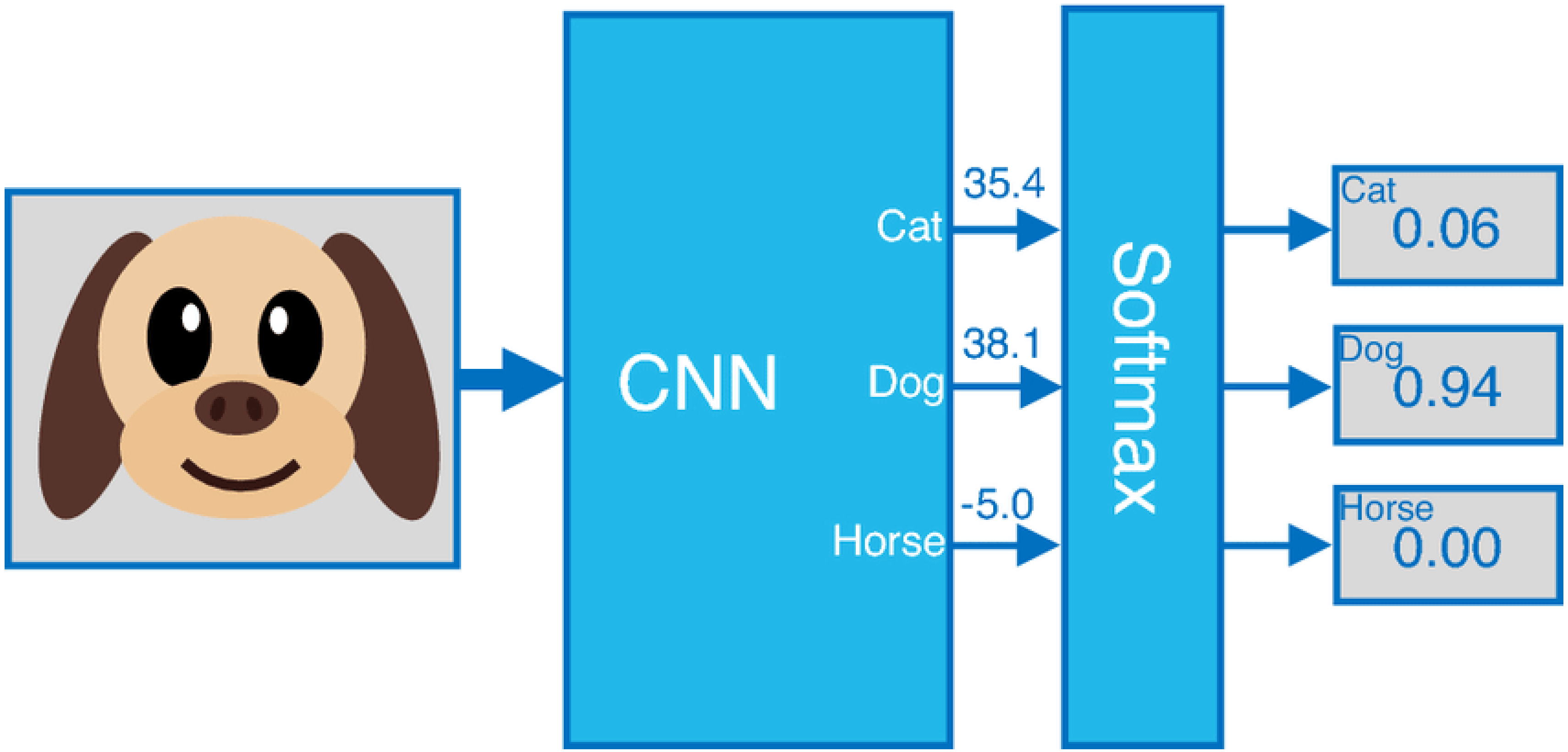
- RNN captures the sequential information present in the input date, i.e. Dependencies between the words in the text while making the prediction

- Example1: Ask the following question to ChatGPT
  - When was ICC formed?
  - Why was it formed?
  - Where are its headquarter?

Motivation of RNNs

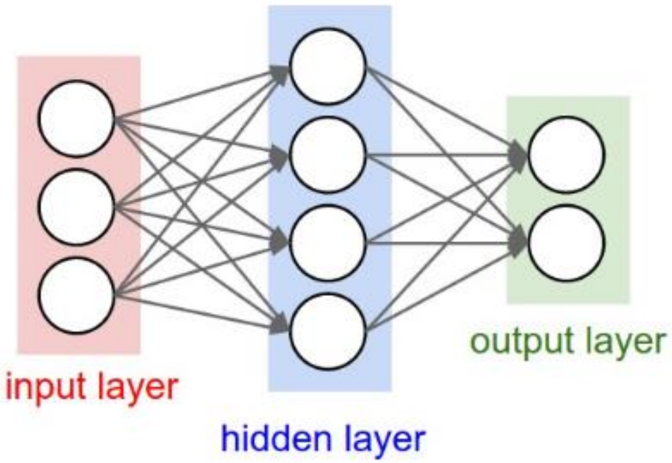
- CNN/ NN do not able to retain the long term dependency among the data

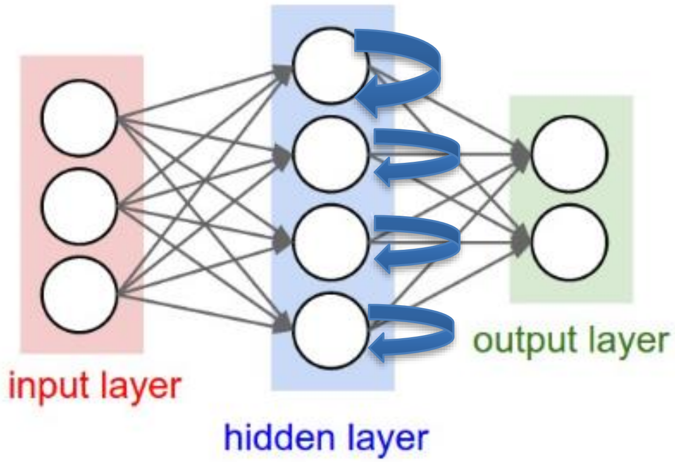
CNN Working



https://www.researchgate.net/publication/353514979/figure/fig3/AS:1050784905064449@1627538052131/Example-of-a-three-classes-CNN-Cats-Dogs-and-Horses.png

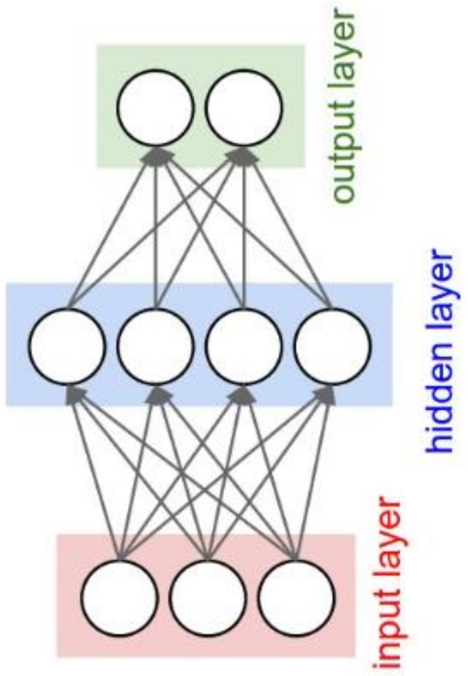
- NN architecture vs Basic RNN architecture

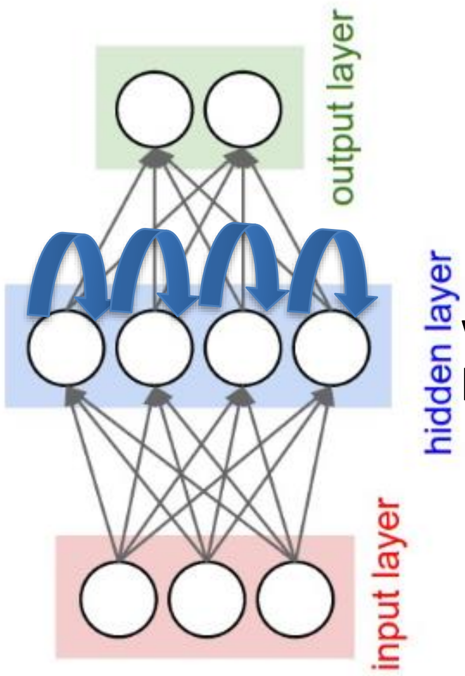




- In RNN feedback connection helps to create memory effect in network – that helps to maintain long term dependency among the data

- NN architecture vs Basic RNN architecture





W, b are the s shared weight for computing the hidden/cell state/memory

- Input sequence X = [x1, x2, x3]
- Output sequence y = [y1, y2]

RNN

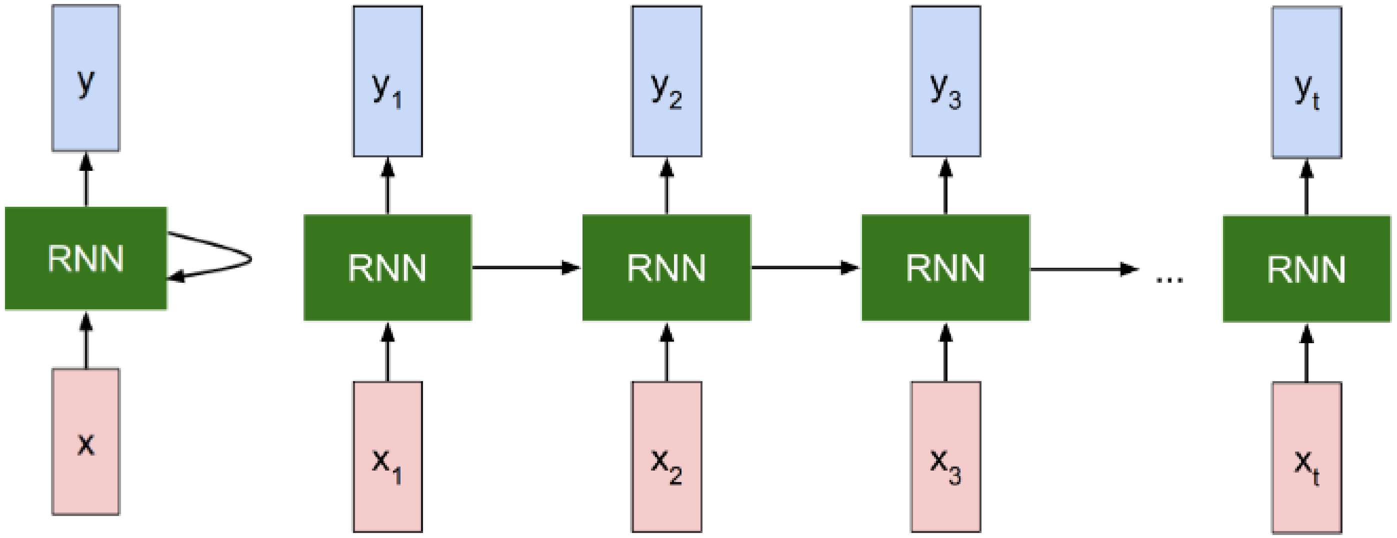
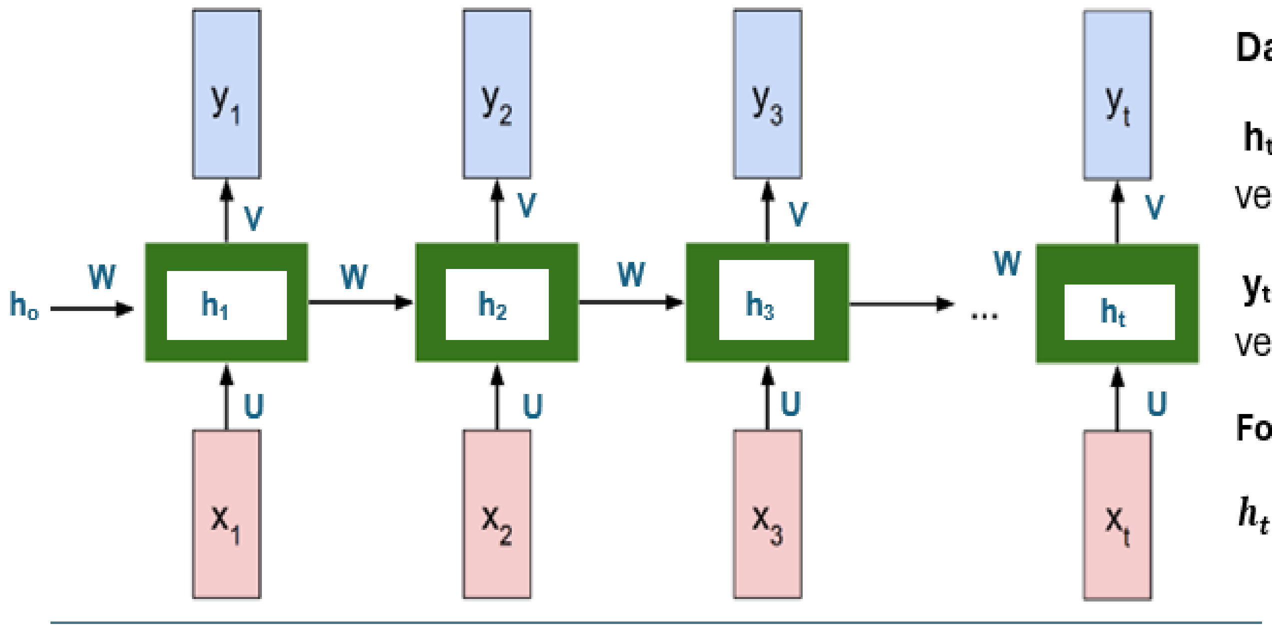


Figure 2. Simplified RNN box (Left) and Unrolled RNN (Right).

Computational graph of RNN (many to many)



Data flow:

h_t: represent flow of data in vertical direction,

y_t: represents data flow in vertical direction

Forward Pass:

$$h_t = \tanh(U\, x_{t-1} + W\, h_{t-1} + b_h)$$

$$y_t = \text{softmax}(V\, h_t\ + b_y)$$

Forward Propagation

-
- **Forward Pass:**

$$h_t = \tanh(U\, x_{t-1} + W\, h_{t-1} + b_h)$$

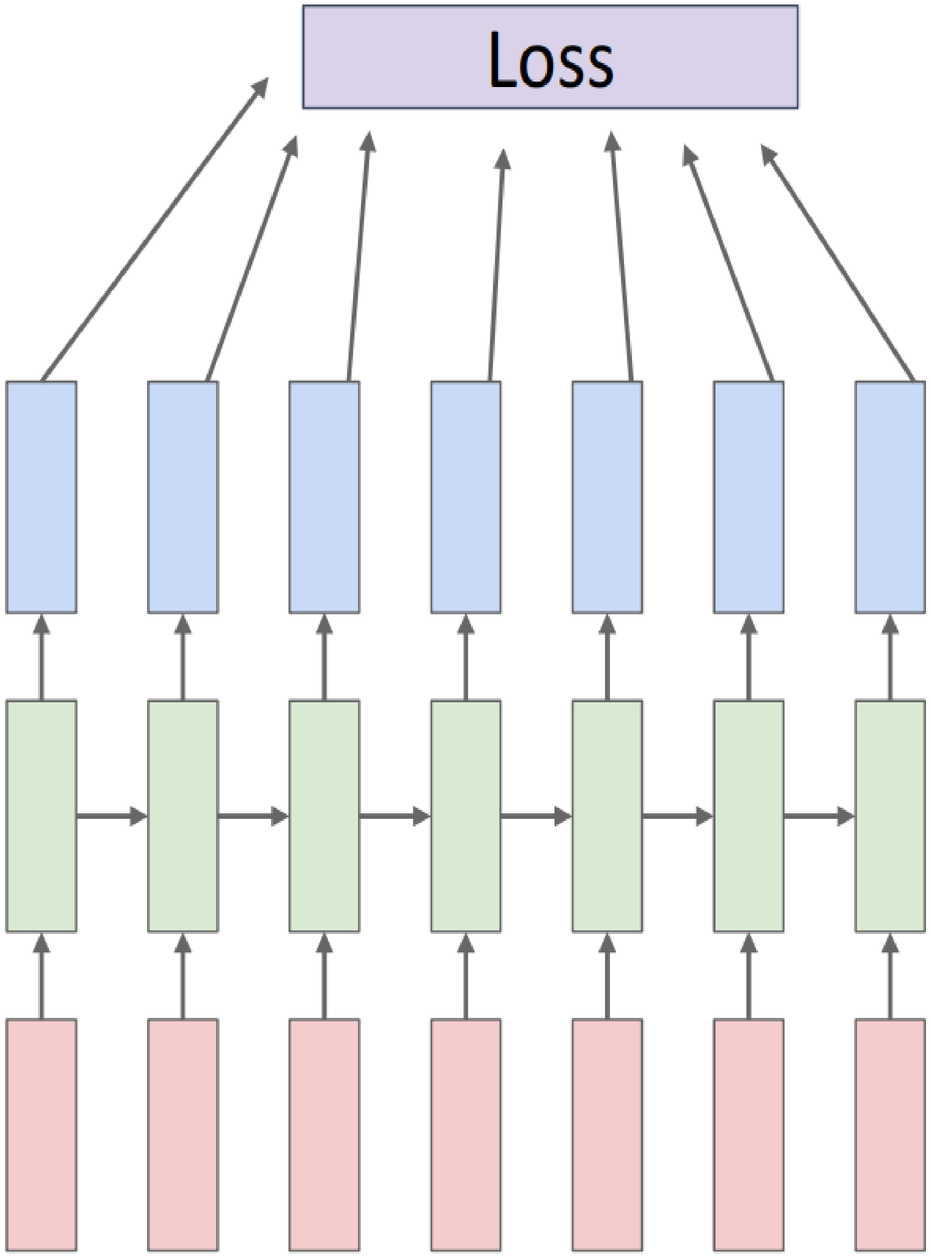
$$y_t = \mathrm{softmax}(V\, h_t + b_y)$$

- **Loss function:**

$$l_t = -\, y_t \log \hat{y_t}$$

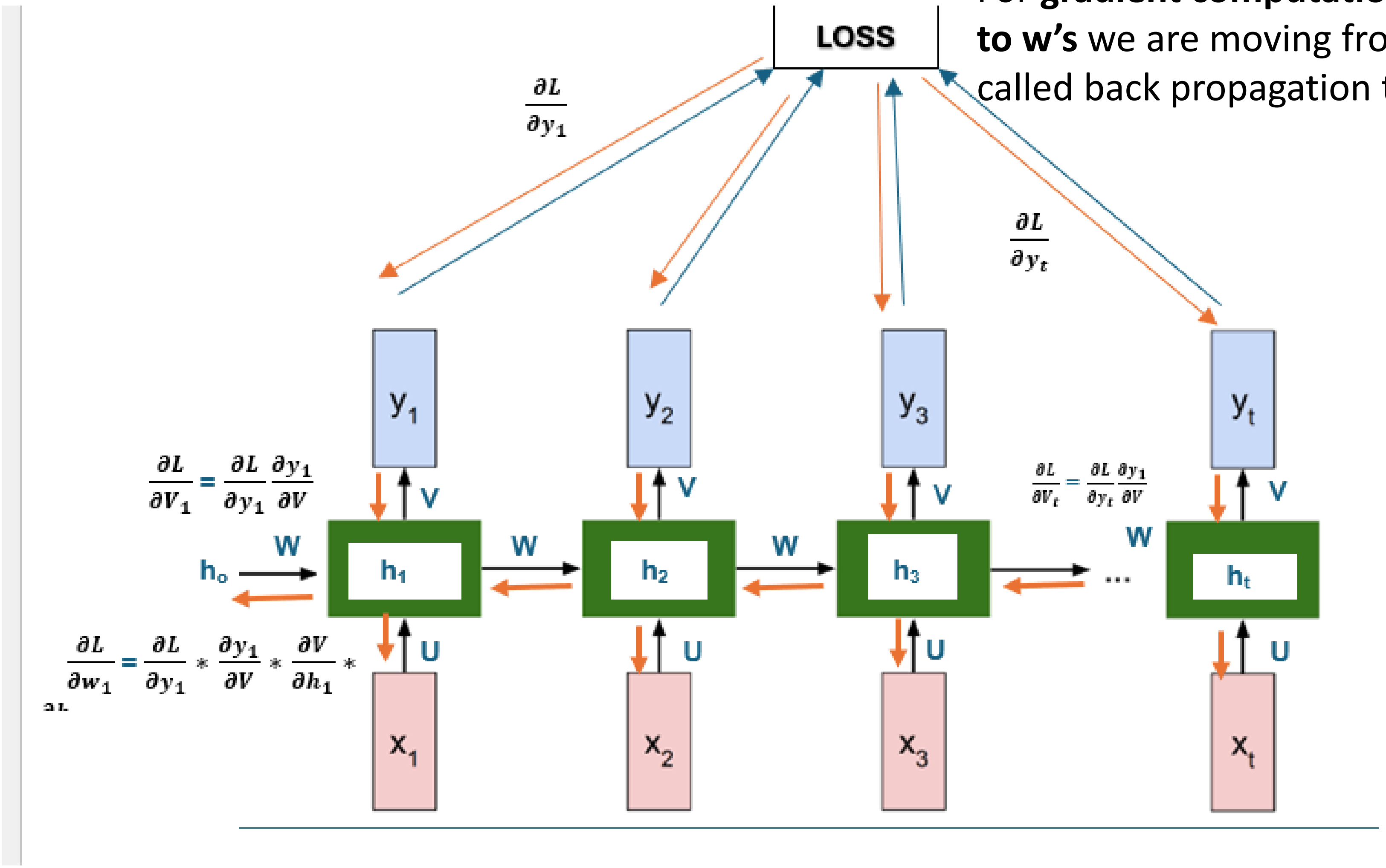
$$l_t = \sum_{k=0}^{T} l_k$$

Computational graph

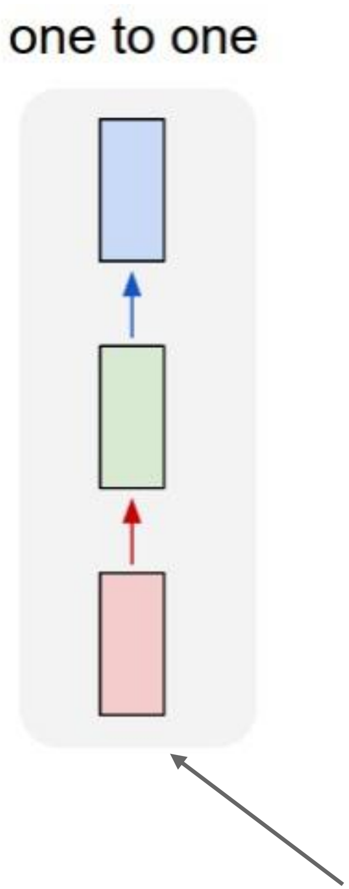


Back Propagation through time

For **gradient computation of loss with respect to w's** we are moving from time t2 to t 1, It is called back propagation through time (BPTT)

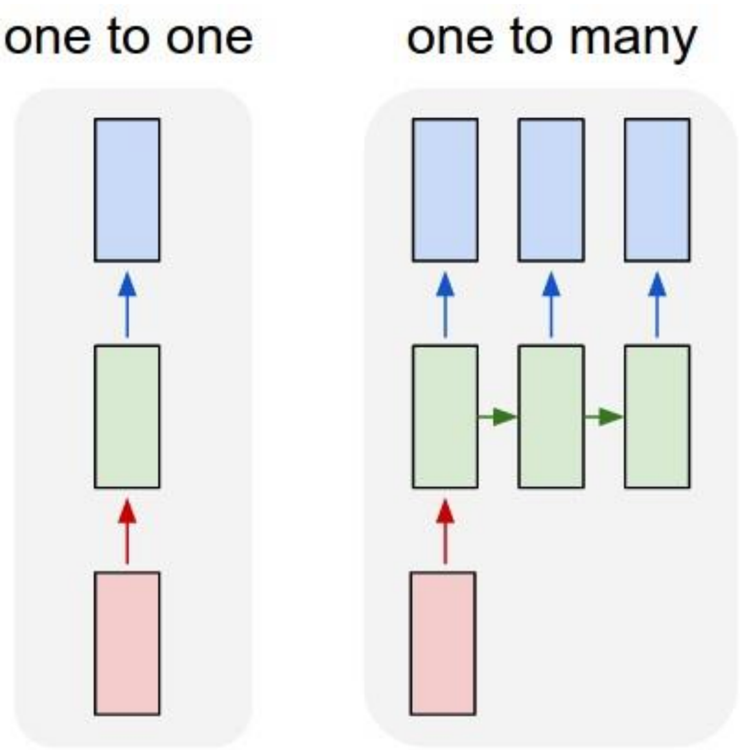


So far: "Feedforward" Neural Networks



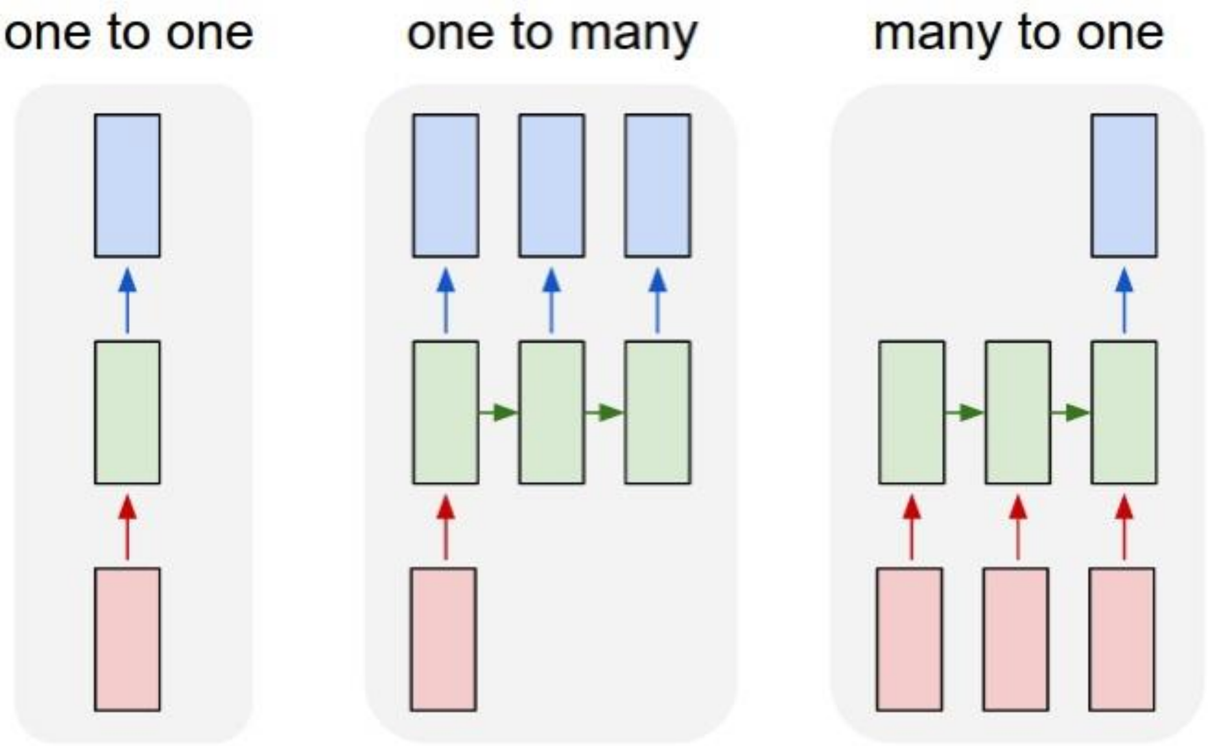
e.g. **Image classification**
Image -> Label

Recurrent Neural Networks: Process Sequences



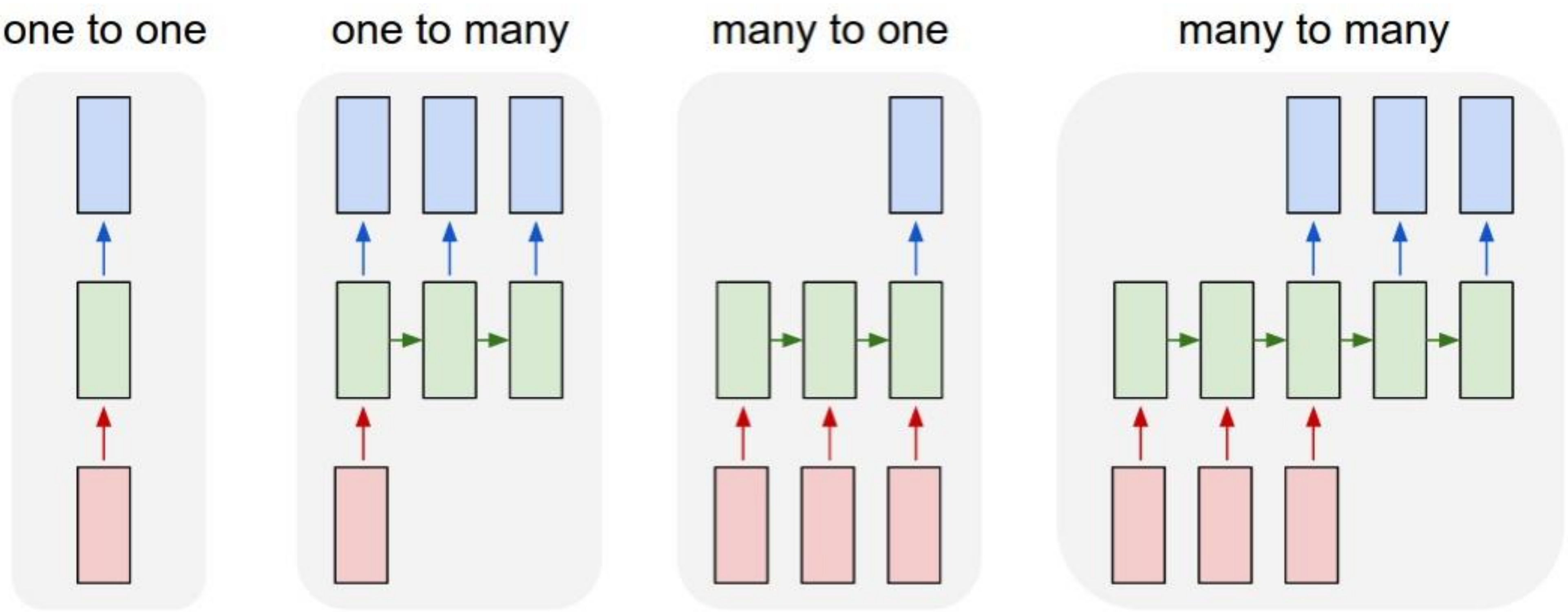
e.g. **Image Captioning**:
Image -> sequence of words

Recurrent Neural Networks: Process Sequences



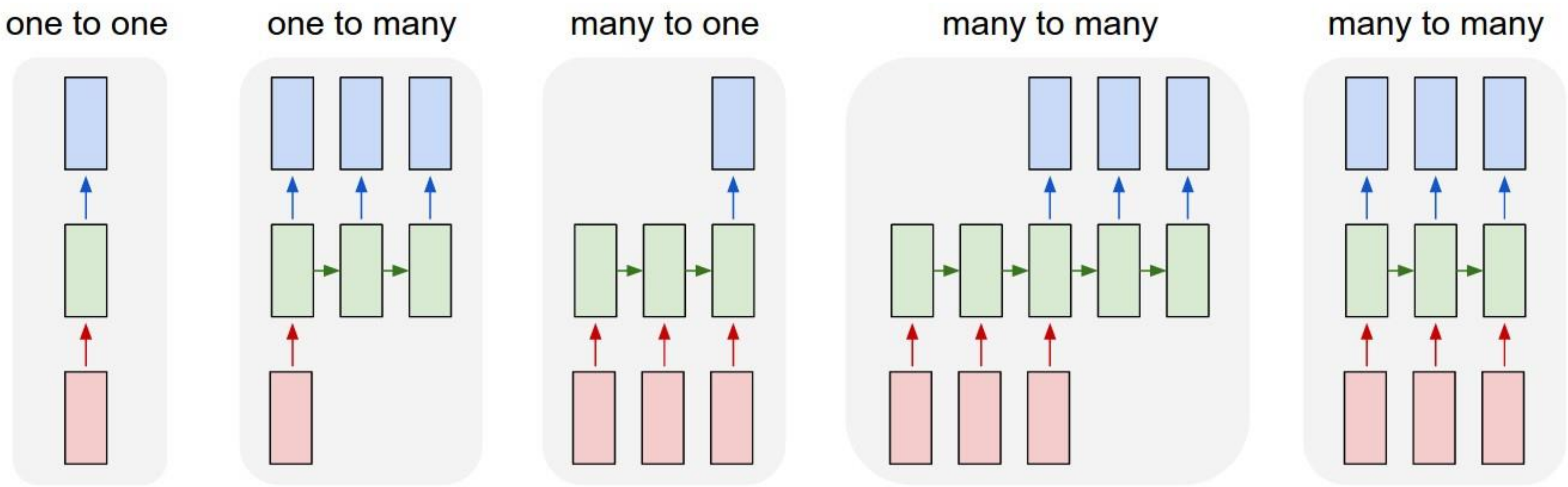
e.g. **Video classification/action recongintion :**
Sequence of images -> label

Recurrent Neural Networks: Process Sequences



e.g. **Machine Translation:**
Sequence of words -> Sequence of words

Recurrent Neural Networks: Process Sequences



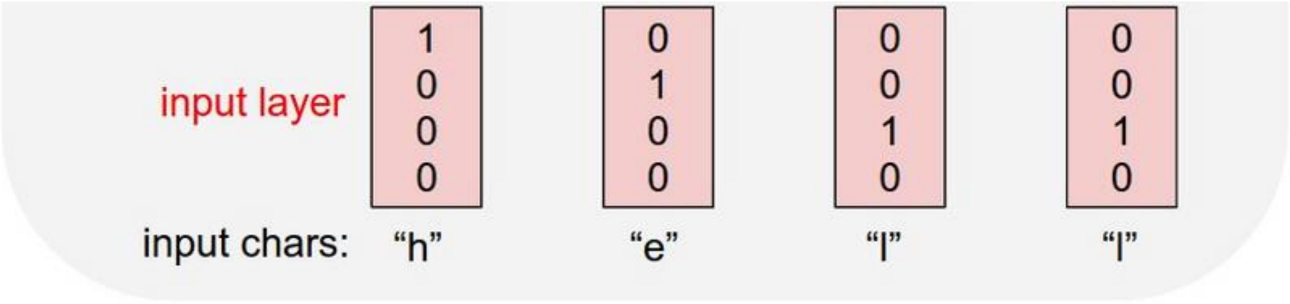
e.g. **Per-frame video classification: (action detection)**
Sequence of images -> Sequence of labels

Example: Character-level language model

Given characters 1, 2, …, t,
model predicts character t

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



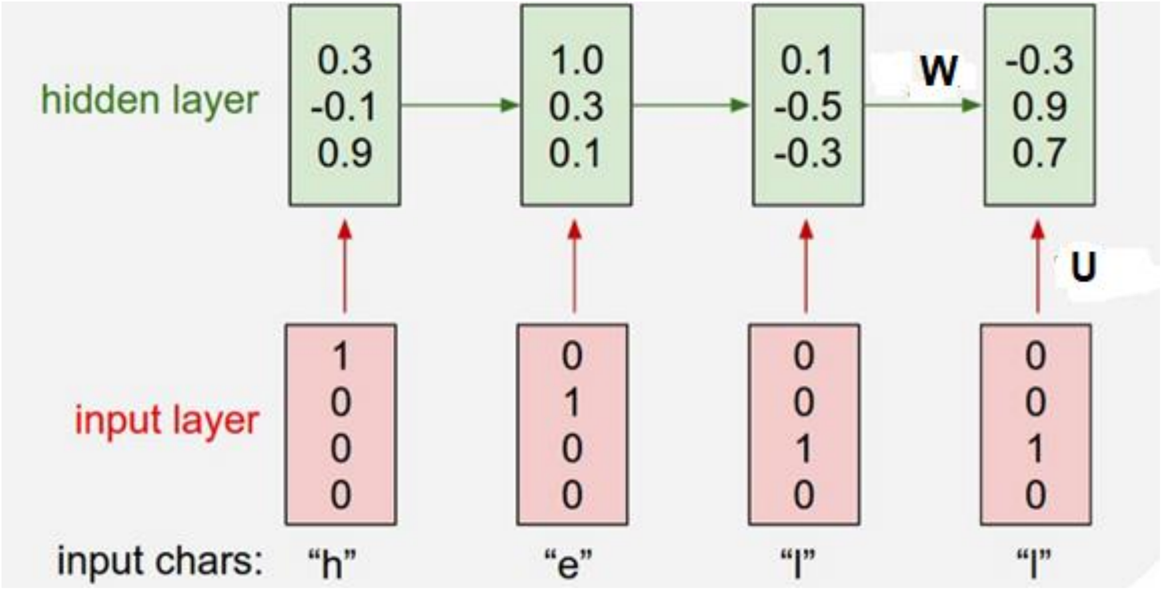
Example: Language Modeling

Given characters 1, 2, …, t,
model predicts character t

$$h_t = \tanh(W\,h_{t-1} + \mathbf{U}\,x_t)$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



Example: Language Modeling

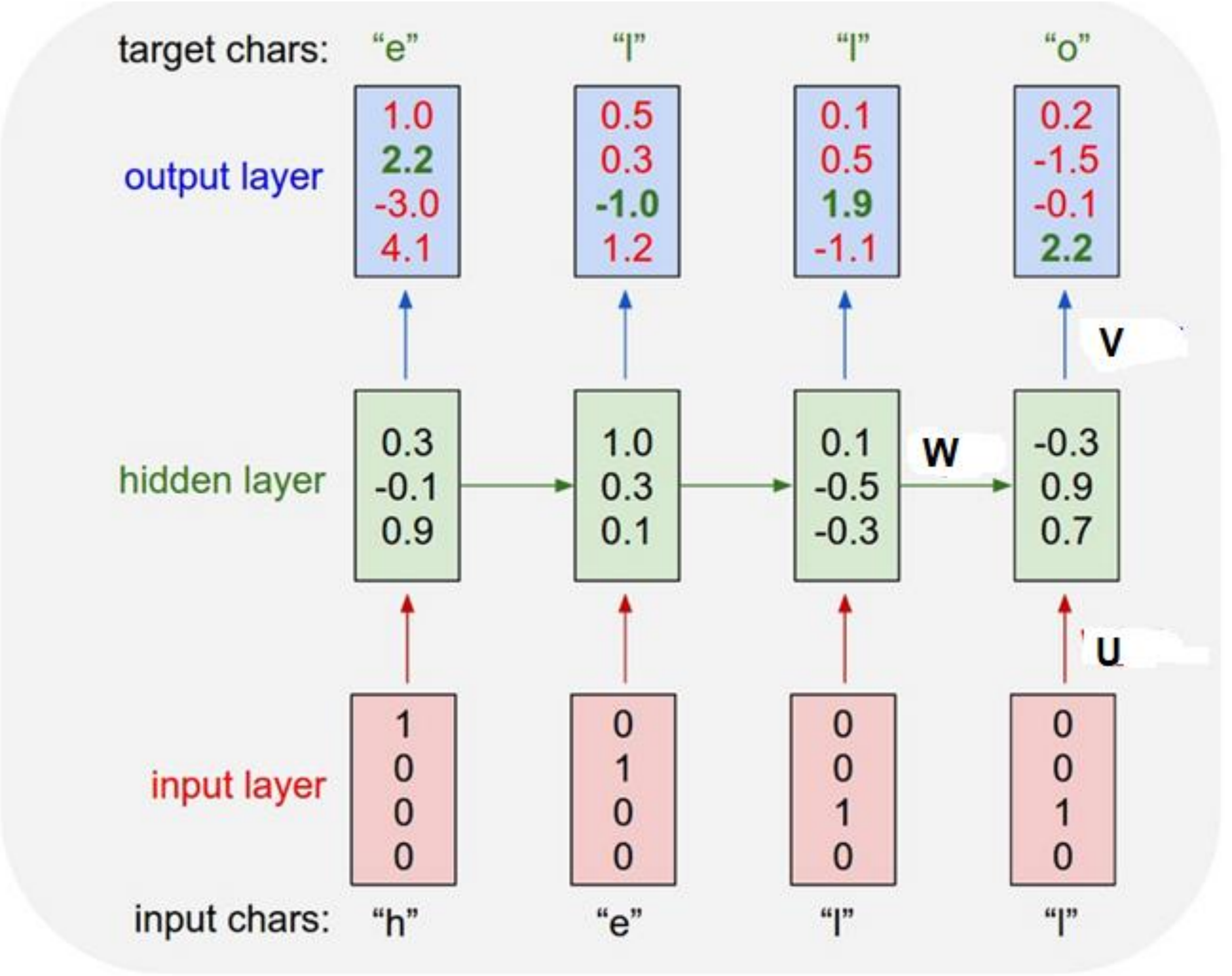
Given characters 1, 2, …, t, model predicts character t

$$h_t = \tanh(W\, h_{t-1} + \mathbf{U}\, x_t + \mathbf{bh})$$

$$\boldsymbol{y_t = \mathrm{softmax}(V\, h_t + b_y)}$$

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



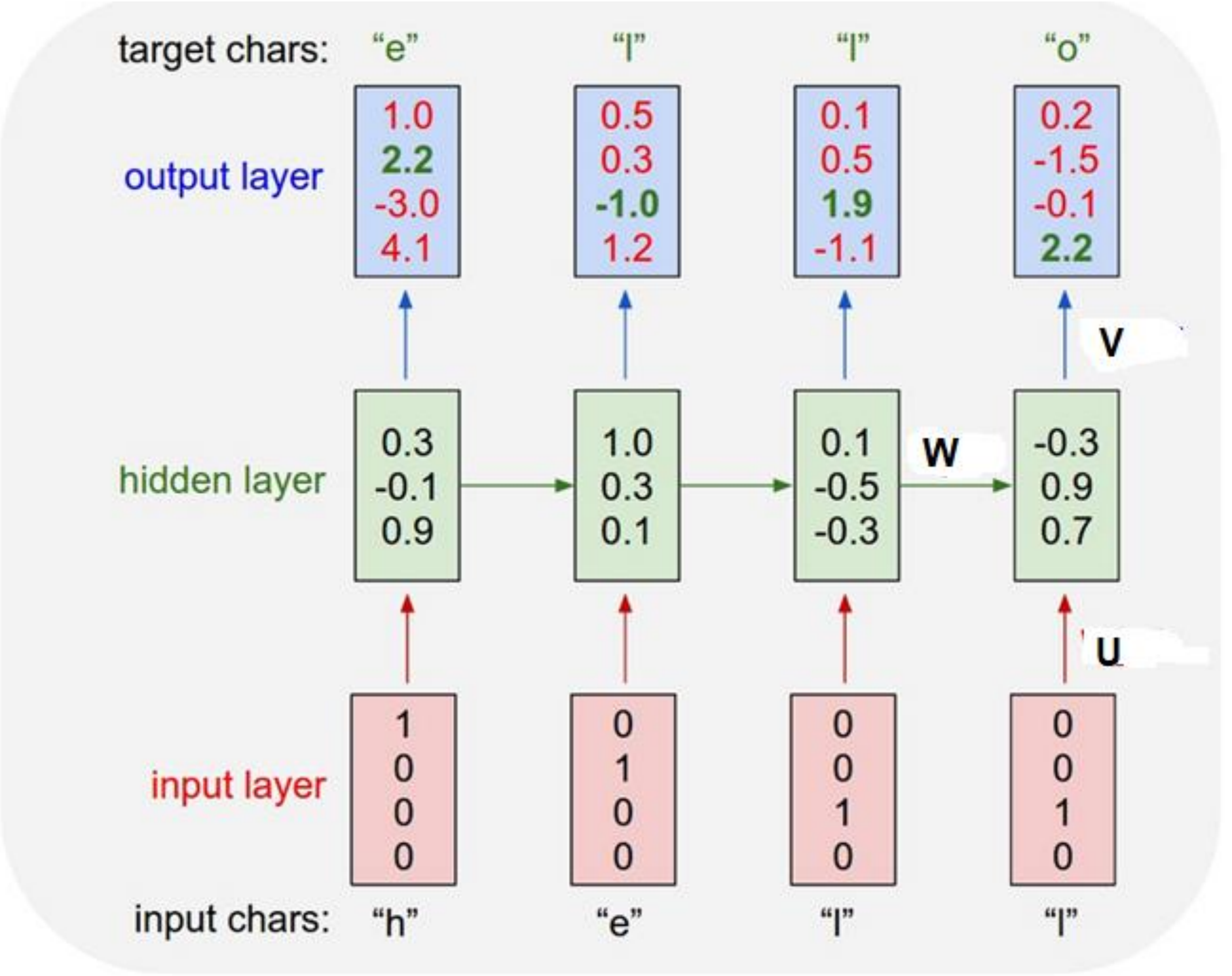
Example: Language Modeling

$$\begin{bmatrix}0.3\\-0.1\\0.9\end{bmatrix} = \tanh(\mathbf{W}\begin{bmatrix}0\\0\\0\end{bmatrix} + \mathbf{U}\begin{bmatrix}1\\0\\0\\0\end{bmatrix})$$

$$\begin{bmatrix}1.0\\0.3\\0.1\end{bmatrix} = \tanh(\mathbf{W}\begin{bmatrix}0.3\\-0.1\\0.9\end{bmatrix} + \mathbf{U}\begin{bmatrix}0\\1\\0\\0\end{bmatrix})$$

$$\begin{bmatrix}0.1\\-0.5\\-0.3\end{bmatrix} = \tanh(\mathbf{W}\begin{bmatrix}1.0\\0.3\\0.1\end{bmatrix} + \mathbf{U}\begin{bmatrix}0\\0\\1\\0\end{bmatrix})$$

$$\begin{bmatrix}-0.3\\0.9\\0.7\end{bmatrix} = \tanh(\mathbf{W}\begin{bmatrix}0.1\\-0.5\\-0.3\end{bmatrix} + \mathbf{U}\begin{bmatrix}0\\0\\1\\0\end{bmatrix})$$



Example: Language Modeling

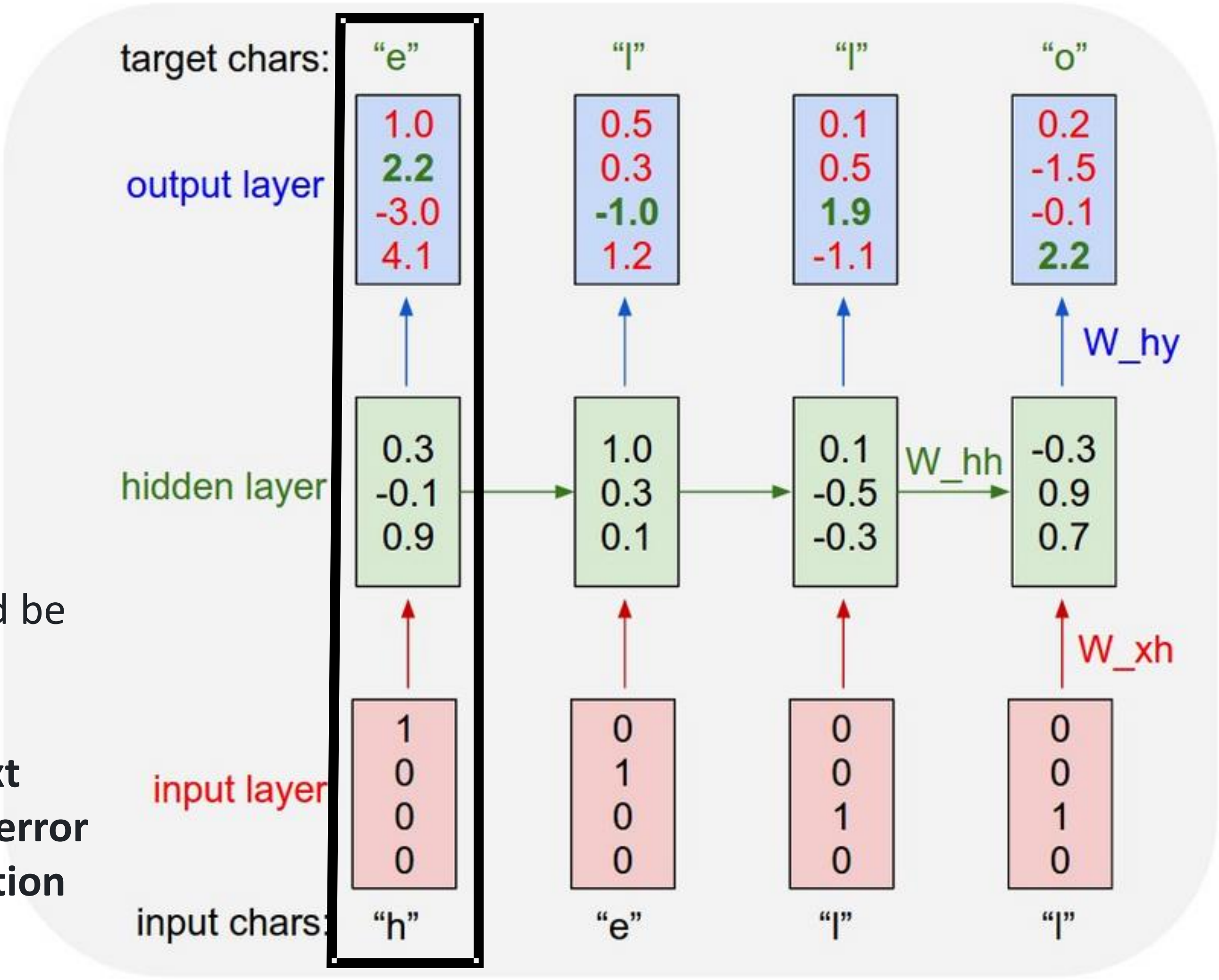
Given "h", predict "e"

- In this case, **RNN incorrectly suggests that "o" should come next**, as the score of **4.1** is the highest.

$$\begin{bmatrix} 1.0 \\ 2.2 \\ -3.0 \\ 4.1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

- Fwe want value at **second index (to predict "e")** should be high and all other scores to be low.

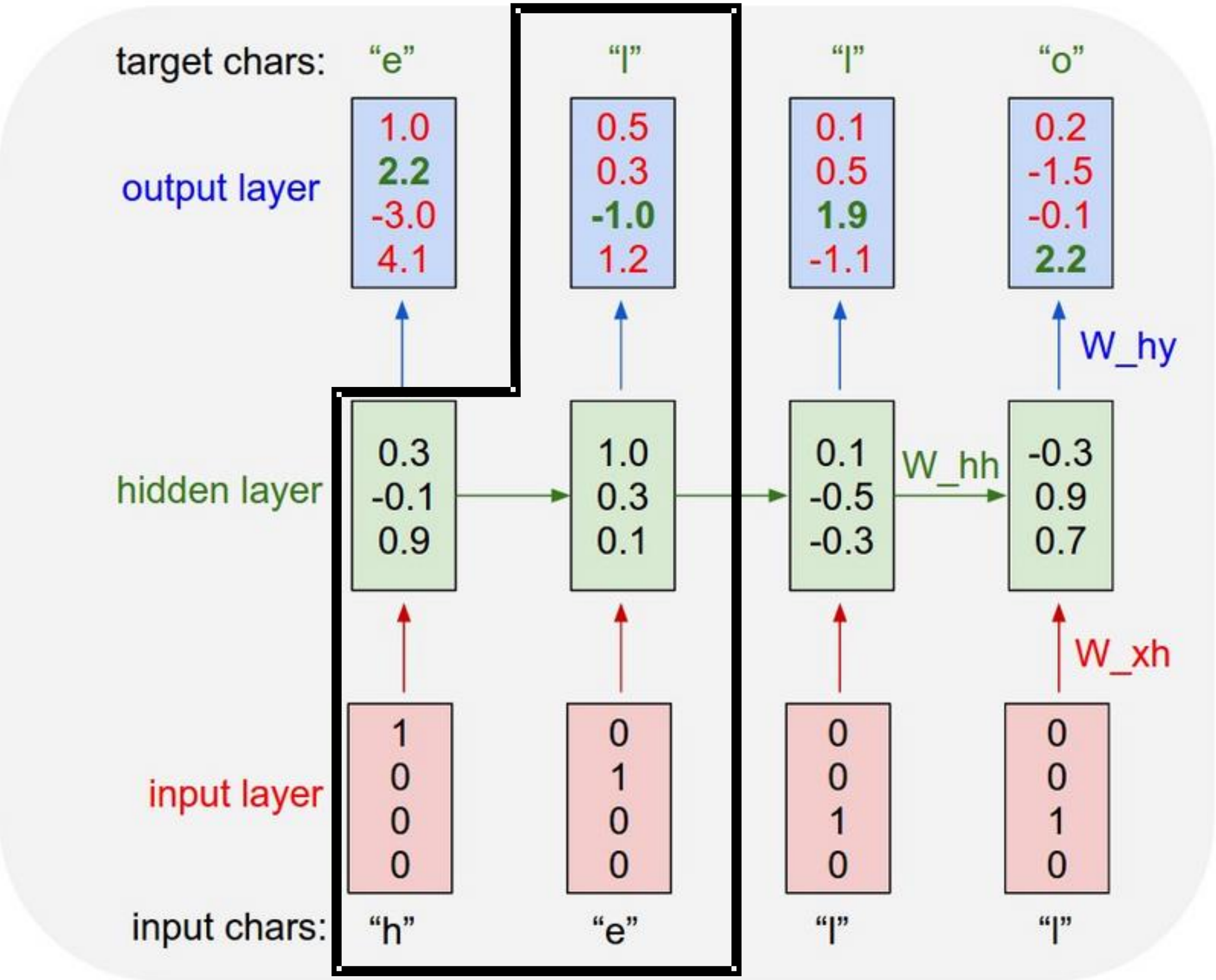
- **At every single timestep we have a target for what next character should come in the sequence**, therefore the **error signal is backpropagated as a gradient of the loss function** through the connections (computational graph).



Example: Language Modeling

Given "he", predict "l"

Given characters 1, 2, …, t, model predicts character t



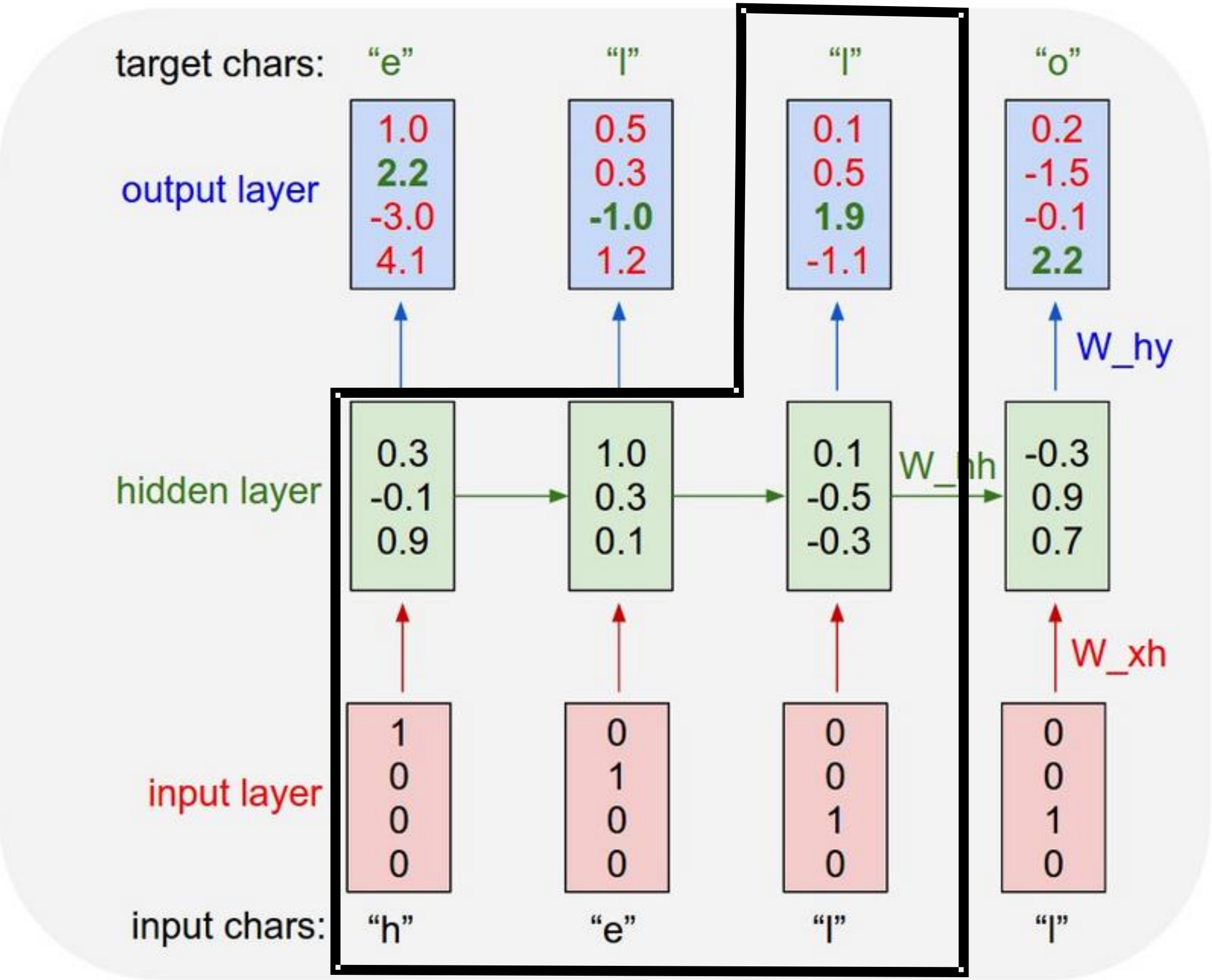
Training sequence: "hello"

Vocabulary: [h, e, l, o]

Example: Language Modeling

Given "hel", predict "l"

Given characters 1, 2, …, t, model predicts character t



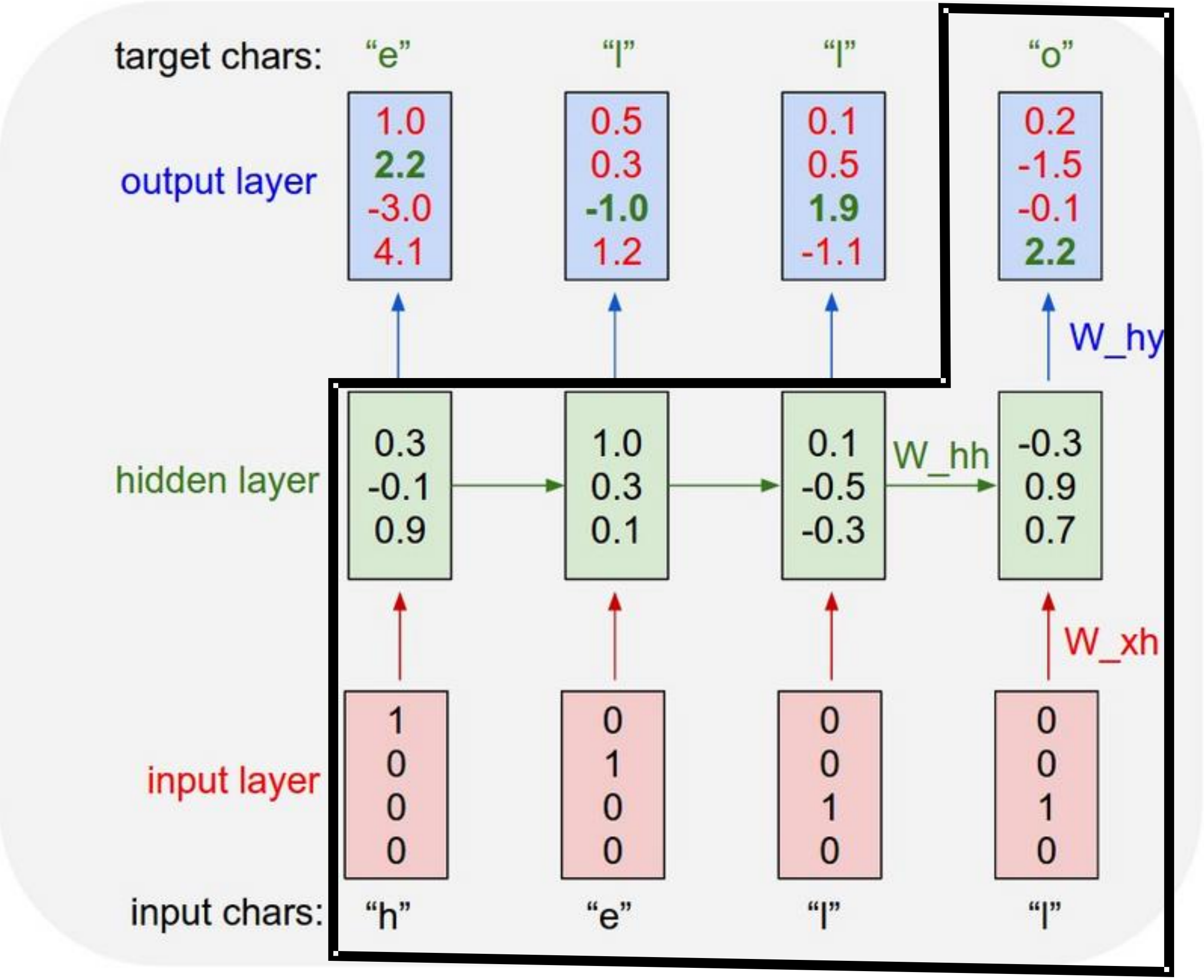
Training sequence: "hello"

Vocabulary: [h, e, l, o]

Example: Language Modeling

Given “hell”, predict “o”

Given characters 1, 2, …, t, model predicts character t

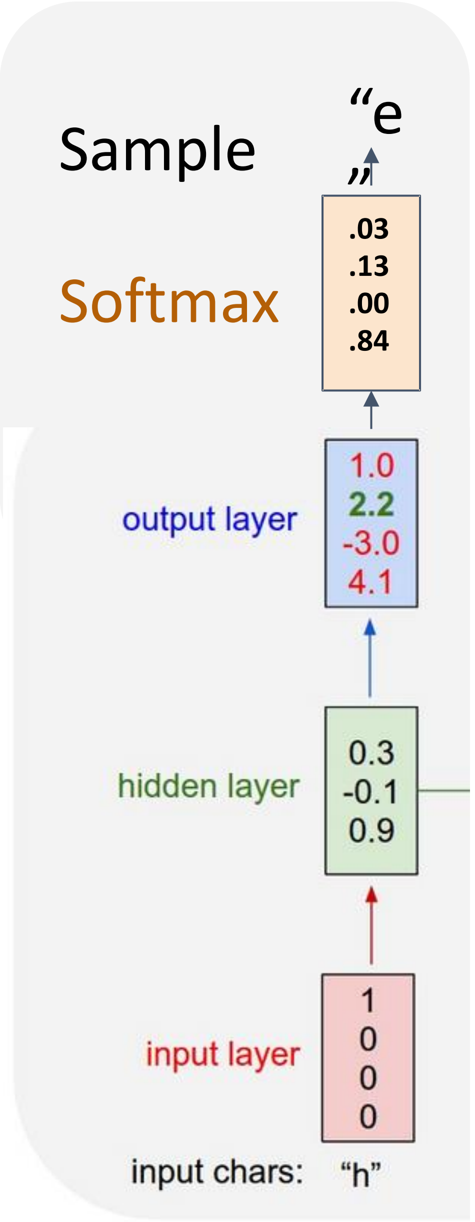


Training sequence: ”hello”

Vocabulary: [h, e, l, o]

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model



Training sequence: ”hello”

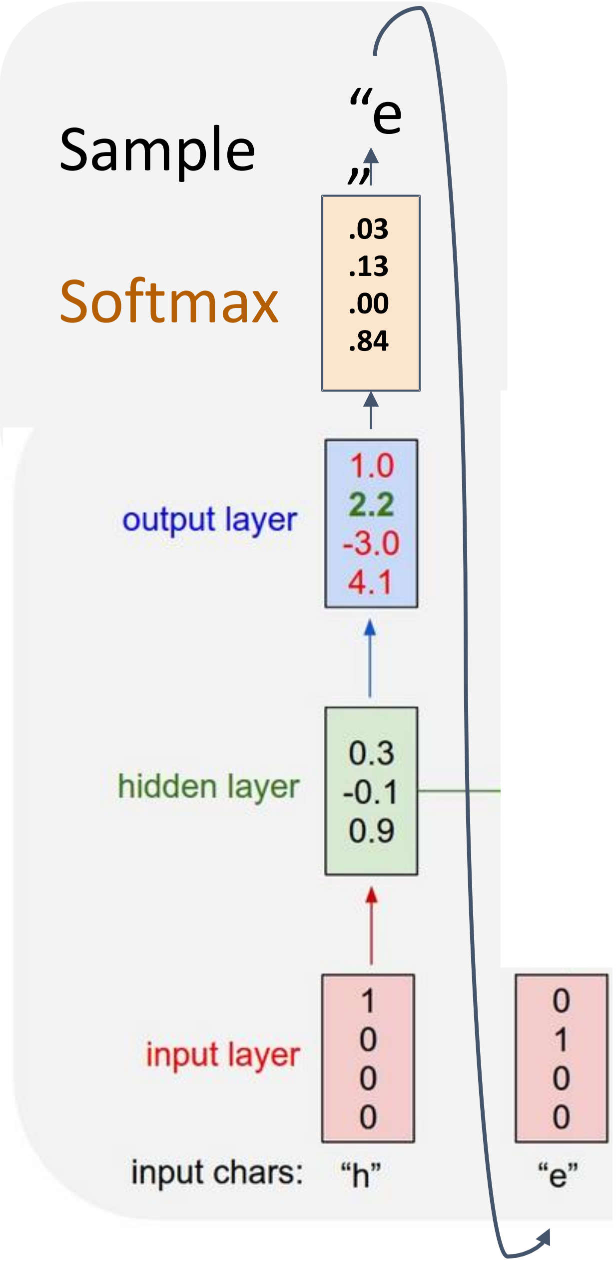
Vocabulary: [h, e, l, o]

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: "hello"

Vocabulary: [h, e, l, o]

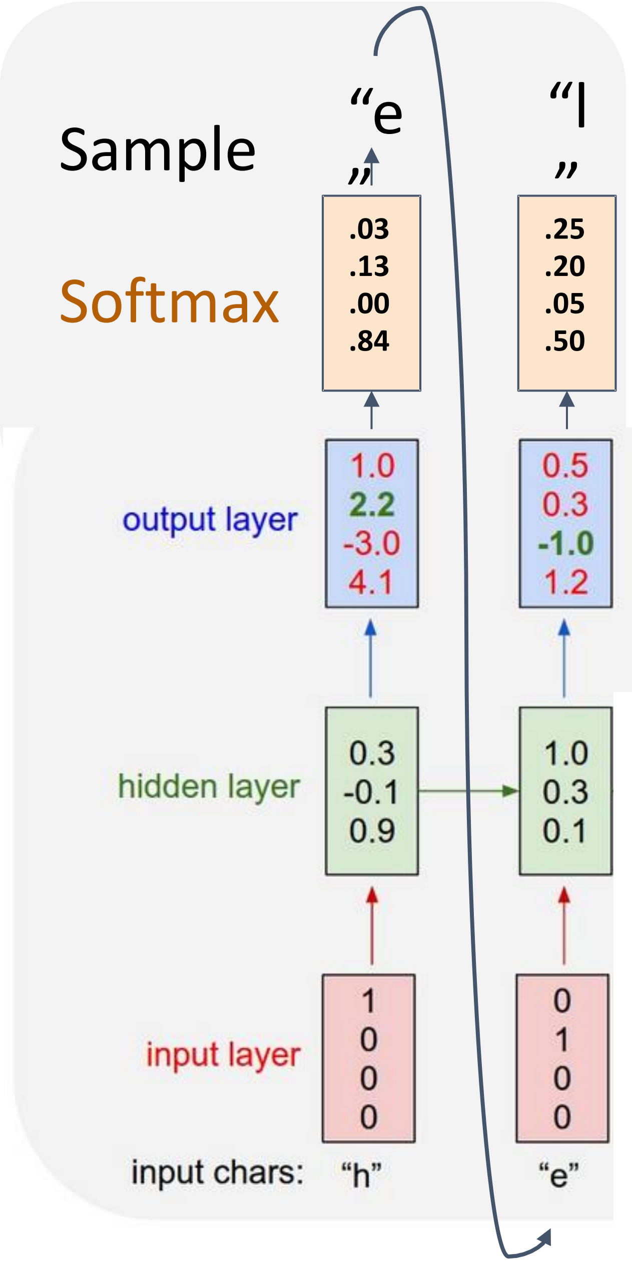


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

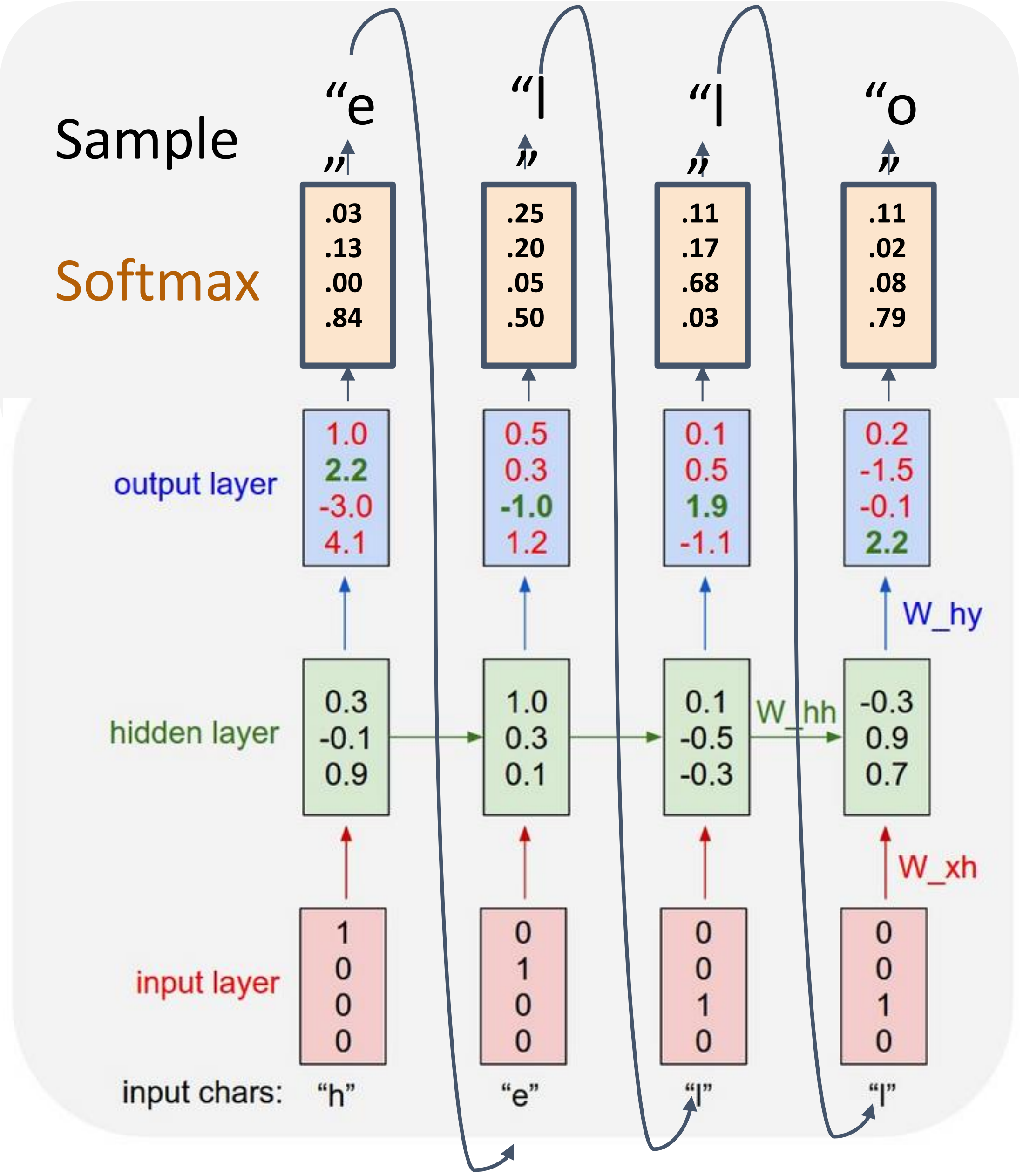
Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model



Training sequence: "hello"

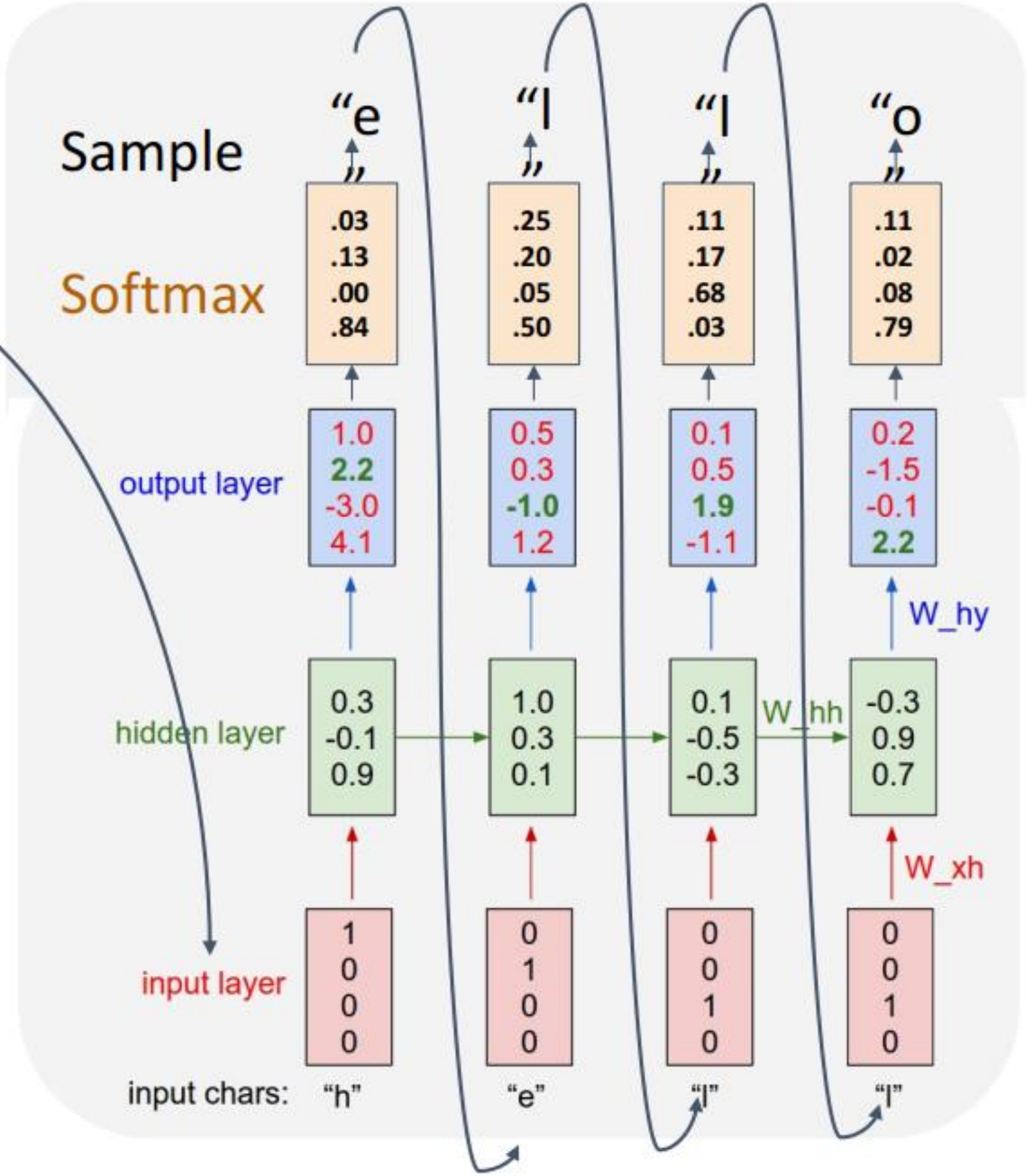
Vocabulary: [h, e, l, o]

Example: Language Modeling

So far: encode inputs as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



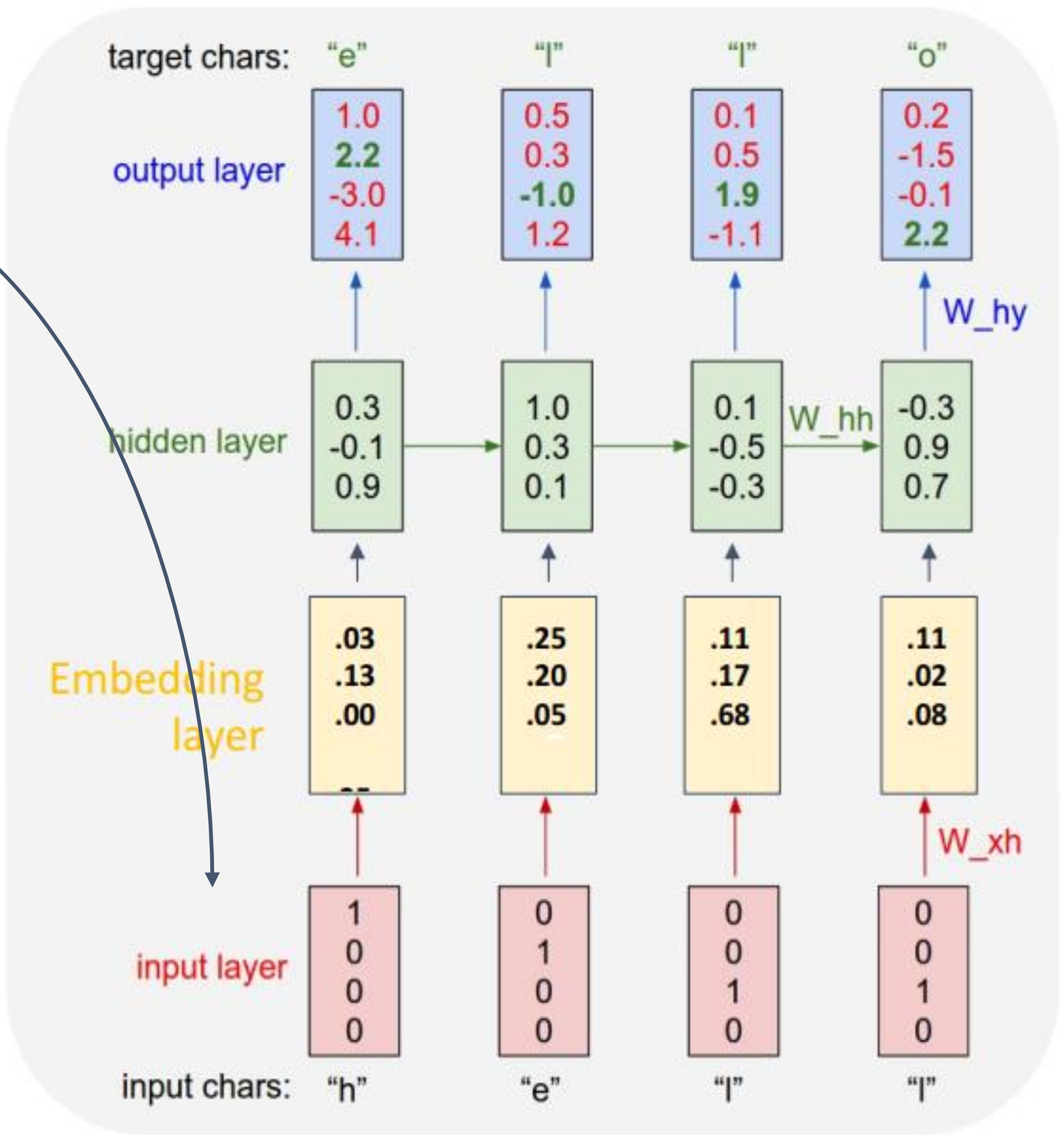
Example: Language Modeling

So far: encode inputs as **one-hot-vector**

Embedding matrix	one-hot vector		compat and semantically relevant

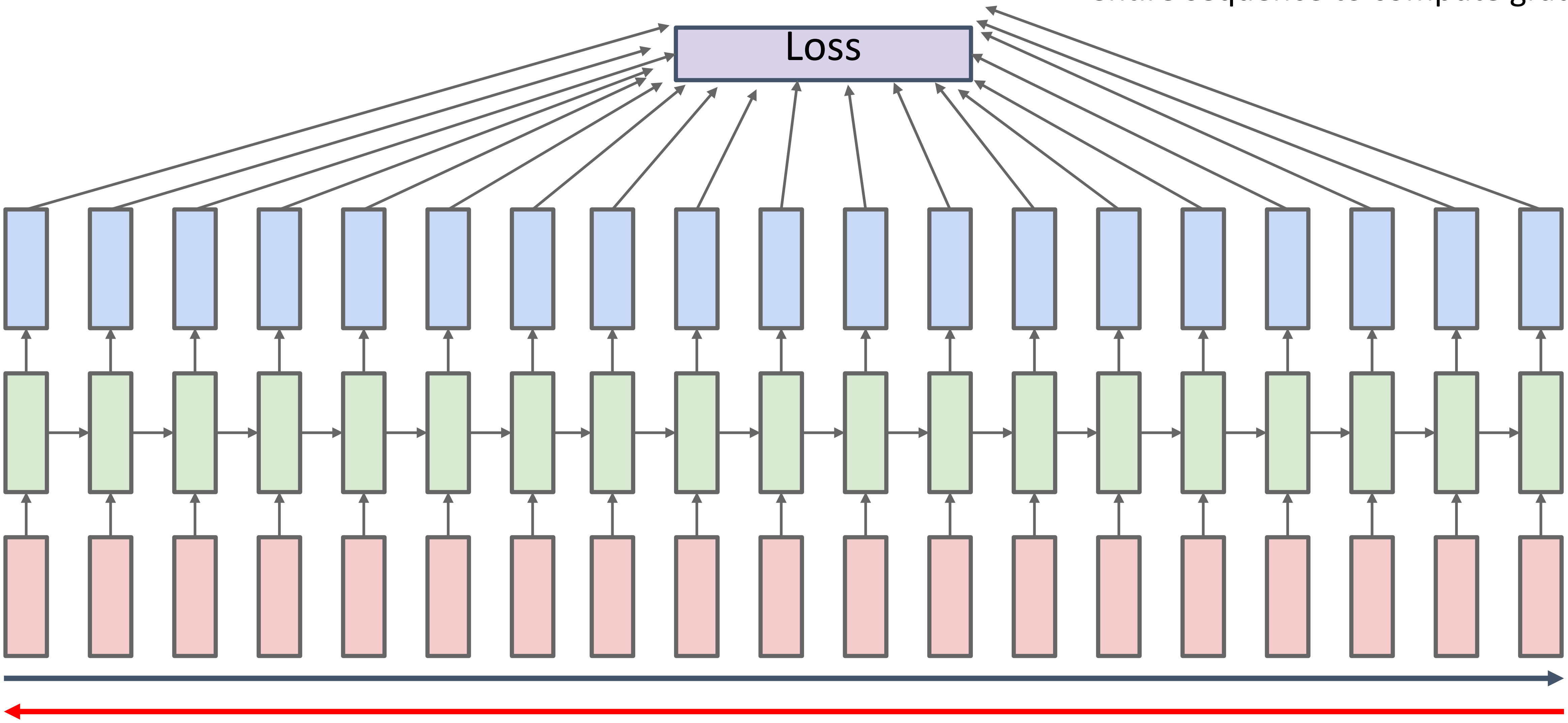
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Often extract this into a separate **embedding** layer



Backpropagation Through Time

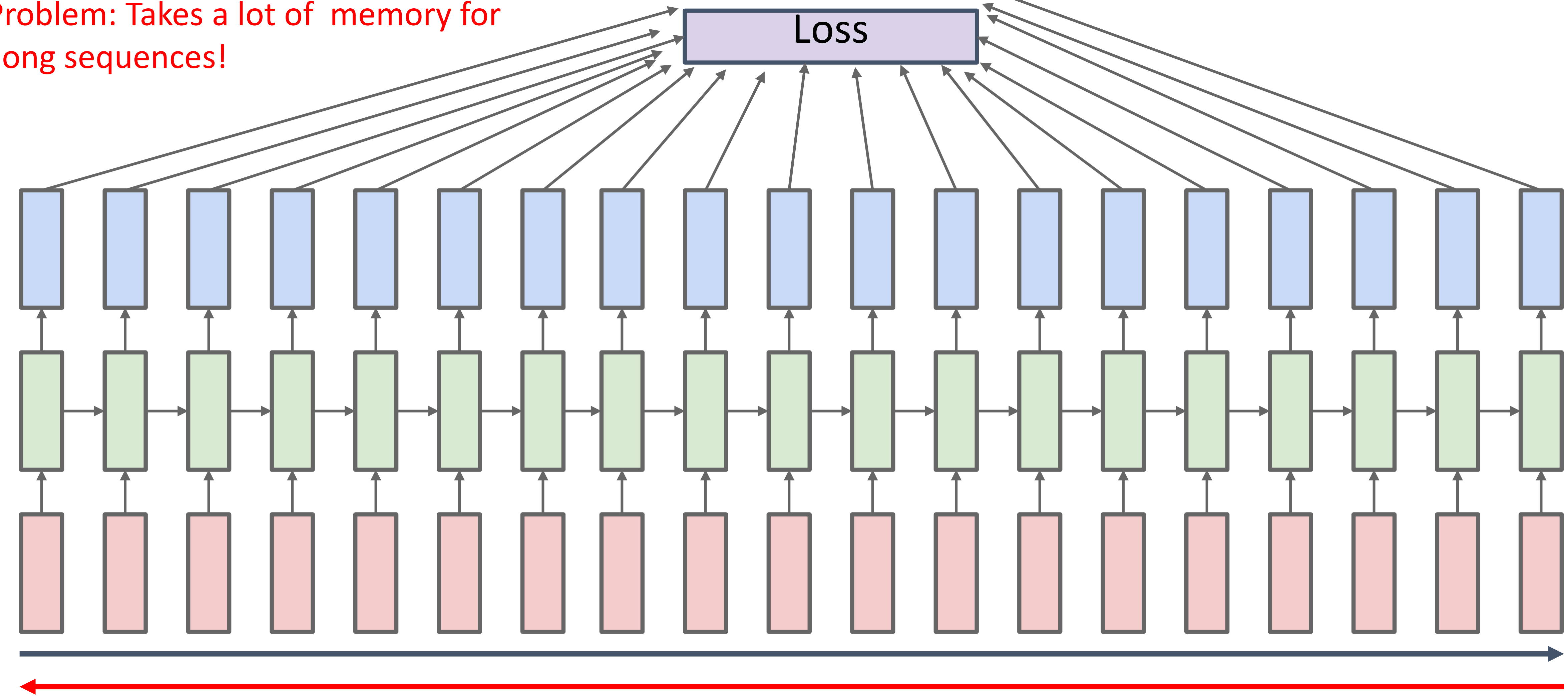
Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



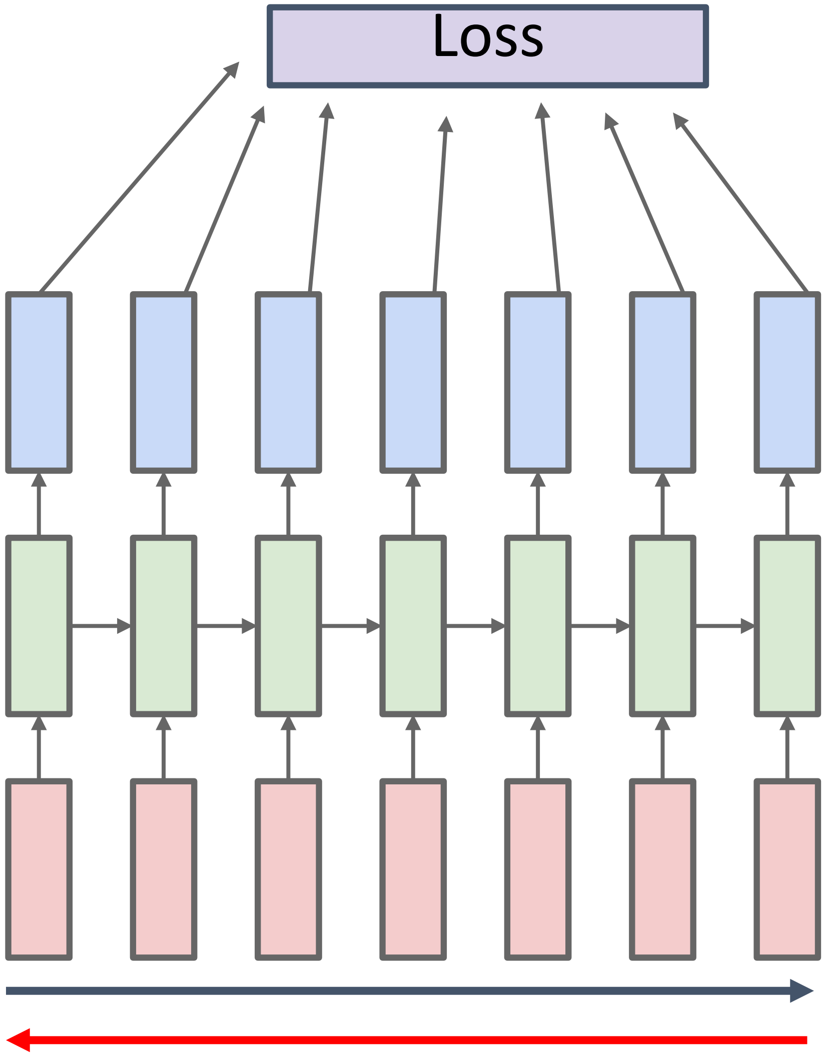
Backpropagation Through Time

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

Problem: Takes a lot of memory for long sequences!

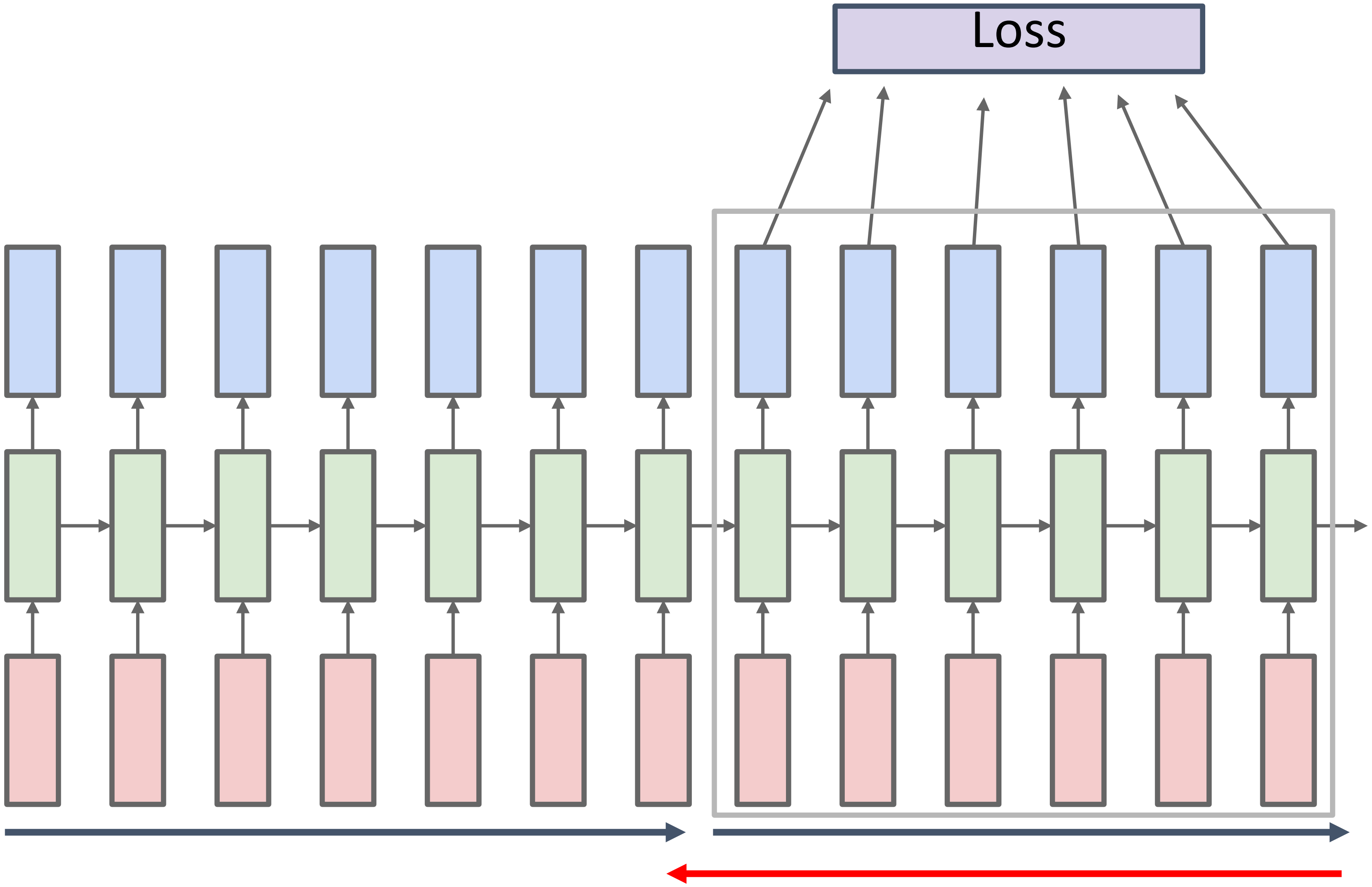


Truncated Backpropagation Through Time



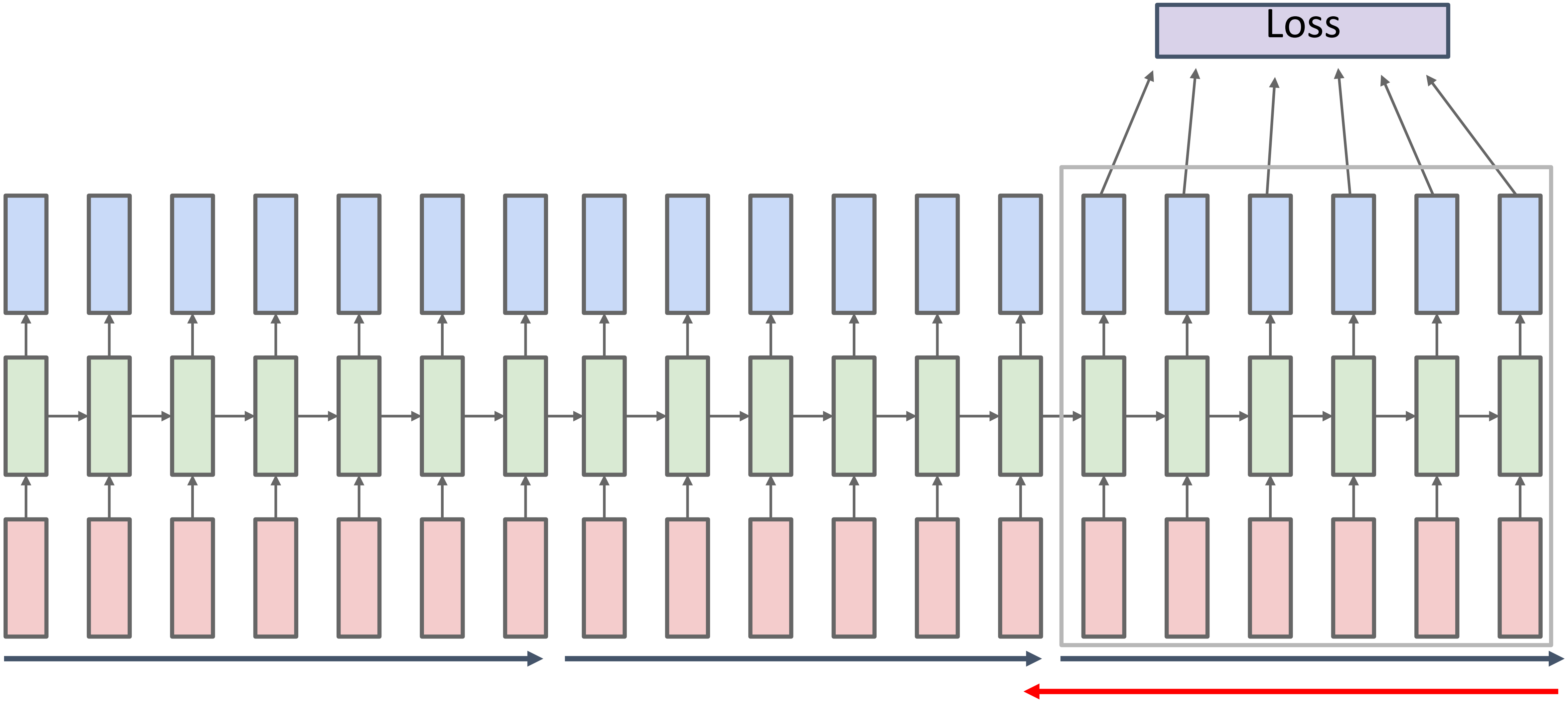
backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation Through Time

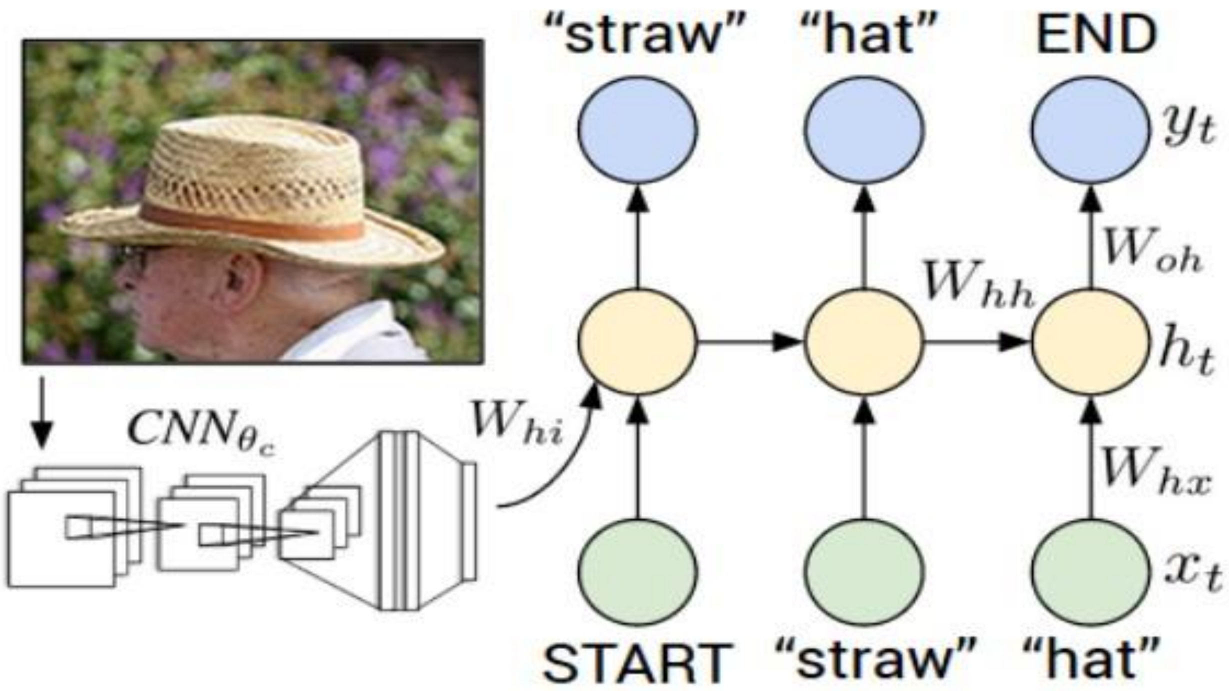


Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation Through Time



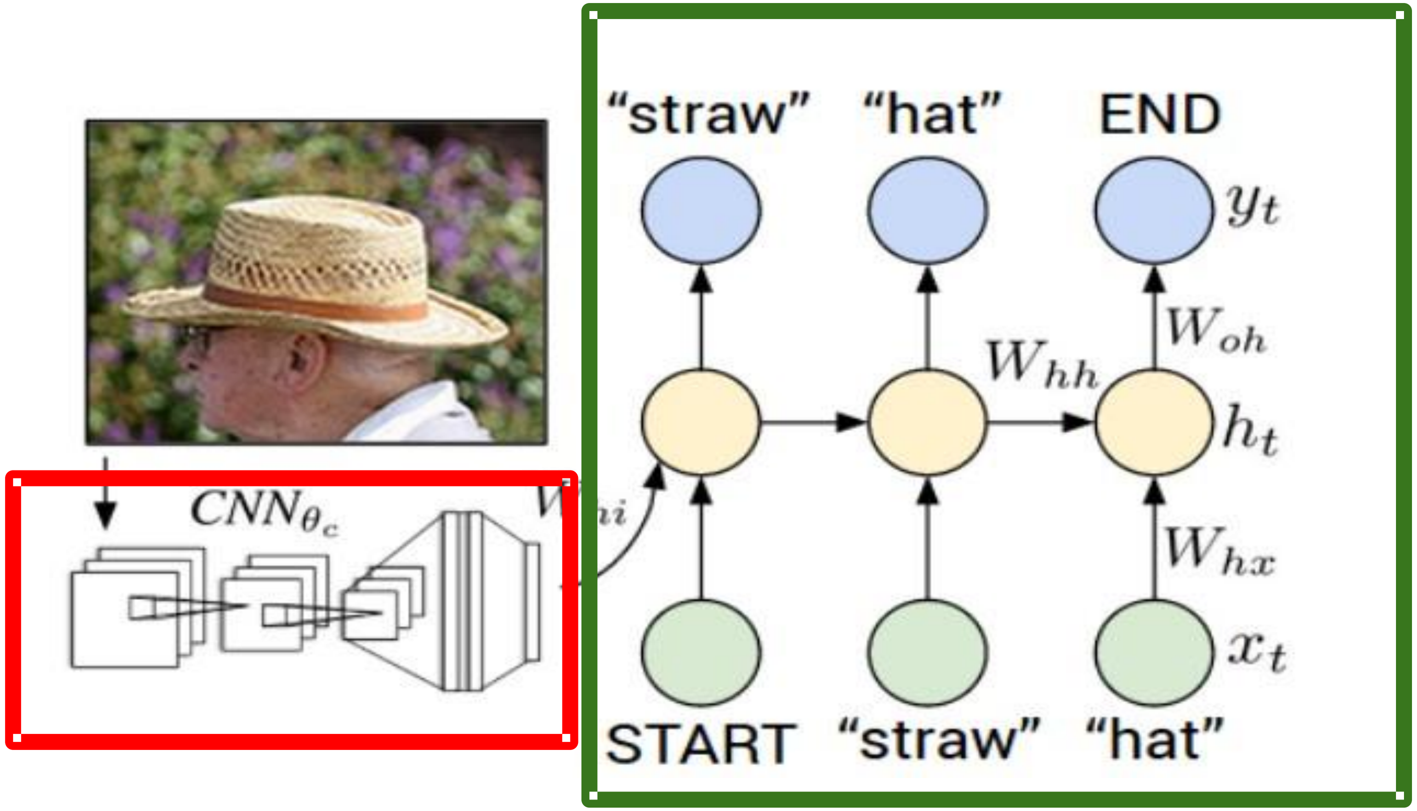
Example: Image Captioning
(one to many problem)



Mao et al, "Explain Images with Multimodal Recurrent Neural Networks", NeurIPS2014 Deep Learning and Representation Workshop
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015 Vinyals
et al, "Show and Tell: A Neural Image Caption Generator", CVPR 2015 Donahue et al, "Long-
term Recurrent Convolutional Networks for Visual Recognition and Description", CVPR 2015
Chen and Zitnick, "Learning a Recurrent Visual Representation for Image Caption Generation", CVPR 2015

Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Example: Image Captioning



Recurrent Neural Network

Convolutional Neural Network

Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Transfer learning: Take CNN trained on ImageNet, chop off last layer

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096

x0

<START>

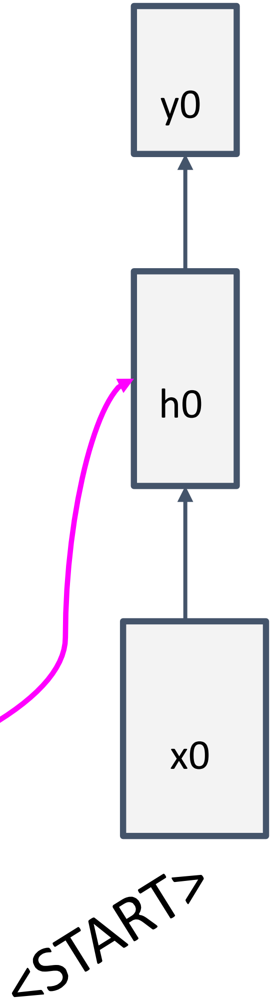
At t = 0:

$h = \tanh(U*x + W*h + \mathbf{Wi*v})$

At t > 0:

$h = \tanh(U*x + W*h)$

$\mathbf{W_i}$



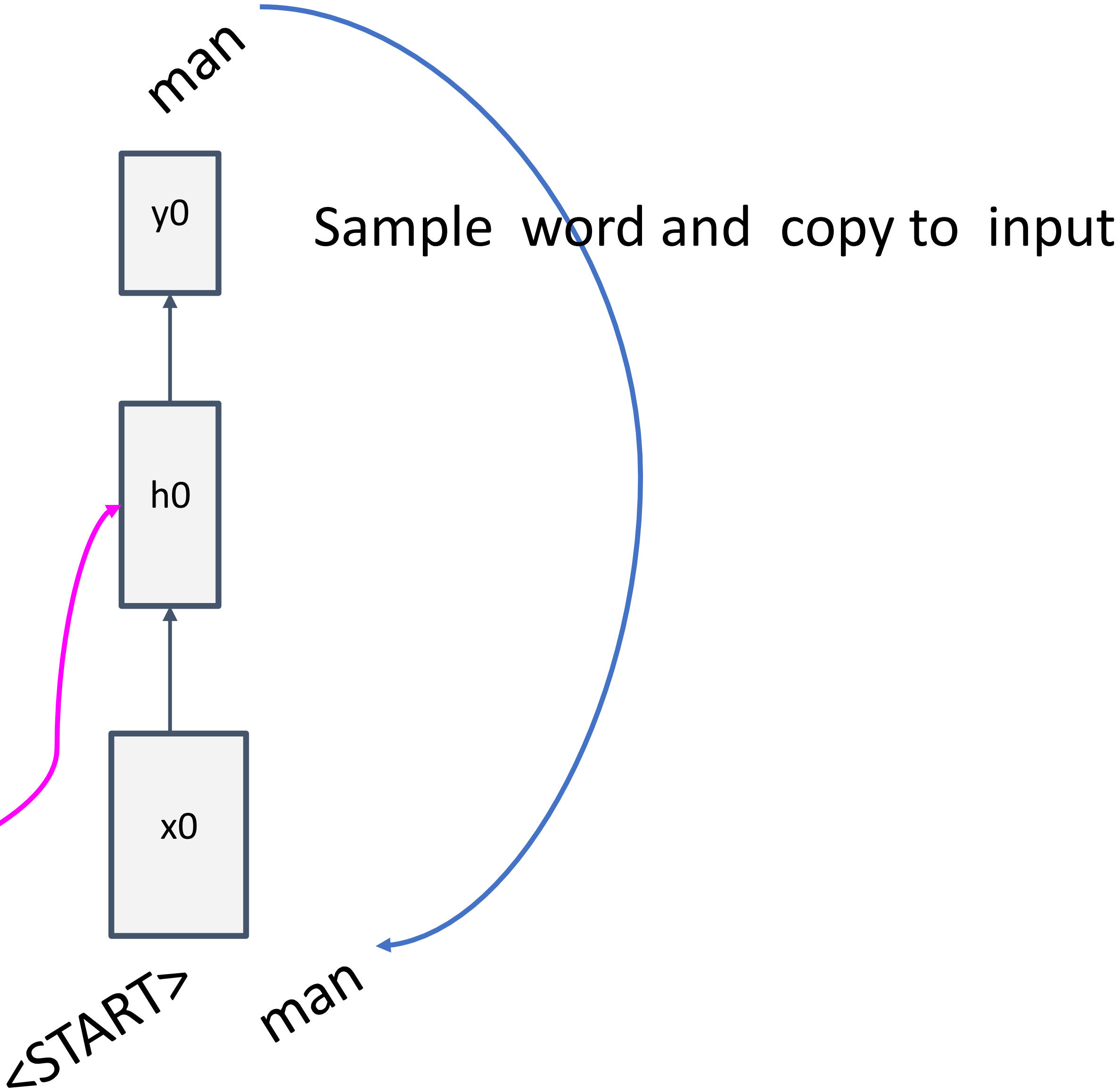
T>0:

$$h = \tanh(U*x + W*h)$$

T = 0:

$$h = \tanh(U*x + W*h + Wi*v)$$

W_i



This image is CC0 public domain

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

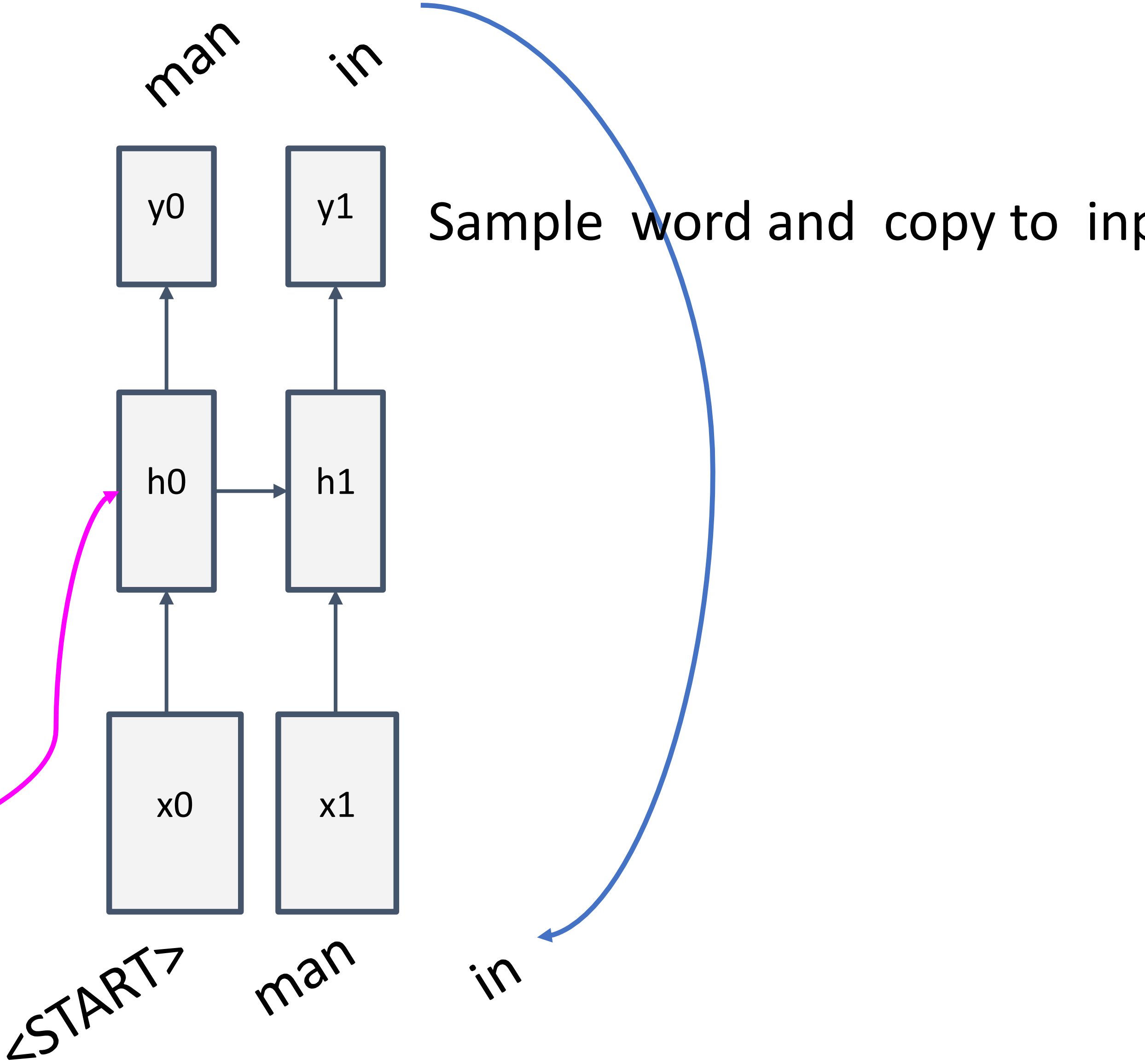
t= 0:

$h = \tanh(U*x + W*h)$

t >0:

$h = \tanh(U*x + W*h + W_i*v)$

W_i



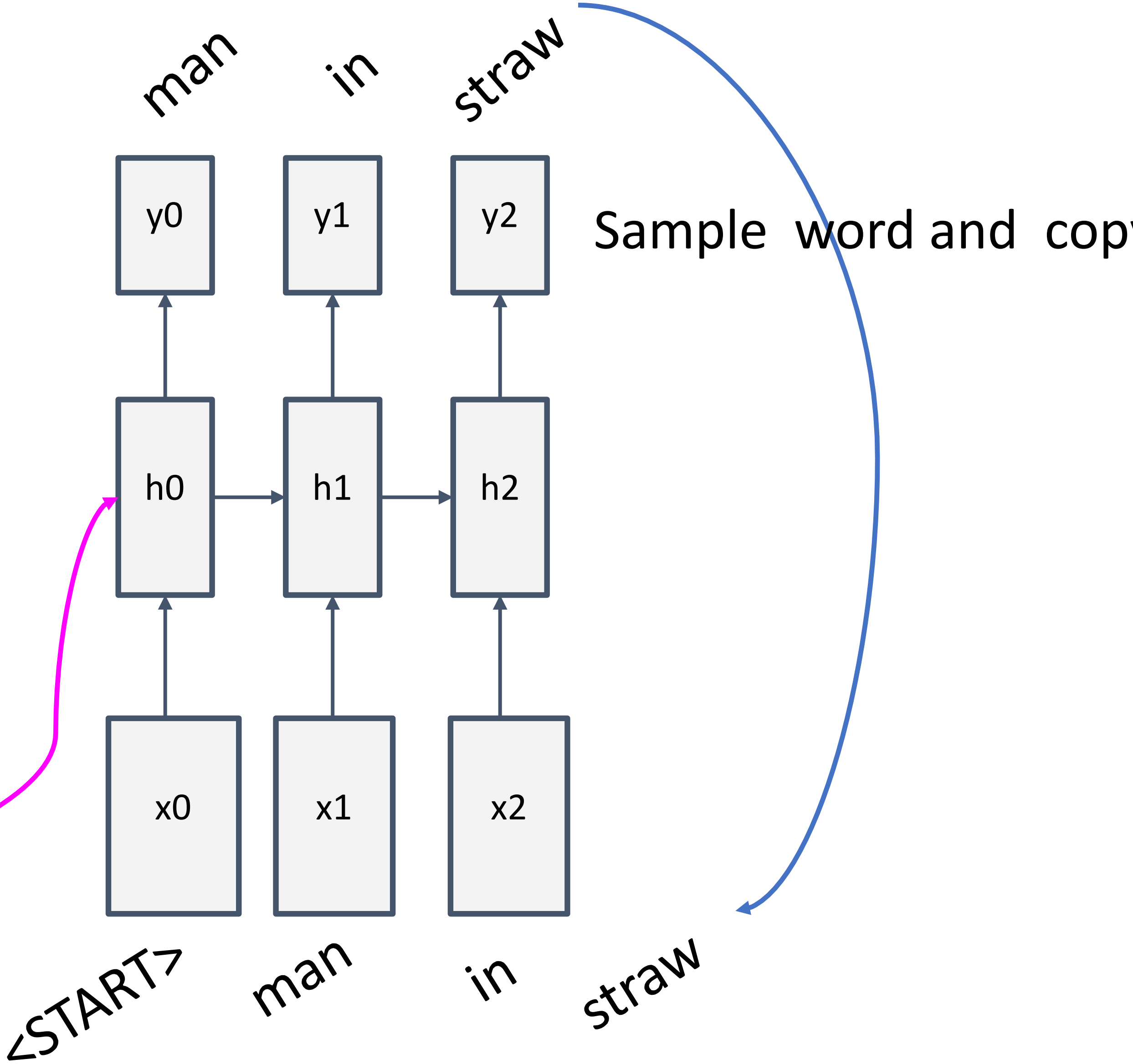
t>0:

$h = \tanh(U*x + W*h)$

t= 0:

$h = \tanh(U*x + W*h + \mathbf{Wi*v})$

W_i



This image is CC0 public domain

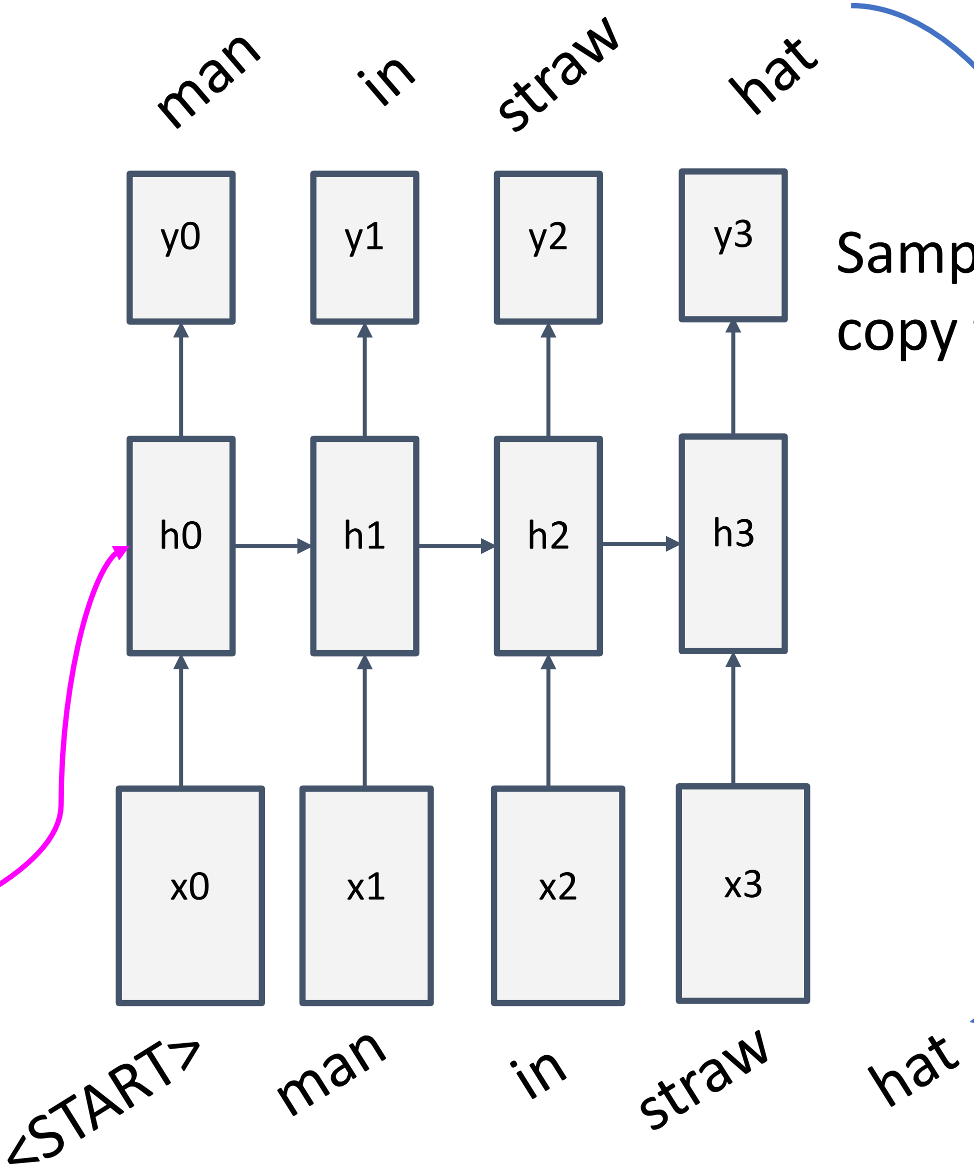
t > 0:

$h = \tanh(W*x + W*h)$

t =0:

$h = \tanh(U*x + W*h + \mathbf{Wi*v})$

$\mathbf{W_{ih}}$



Sample word ar
copy to input

Stop after sampling <END> token

t>0:

$$h = \tanh(U*x + W*h)$$

t= 0:

$$h = \tanh(U*x + W*h + Wi*v)$$

W_i

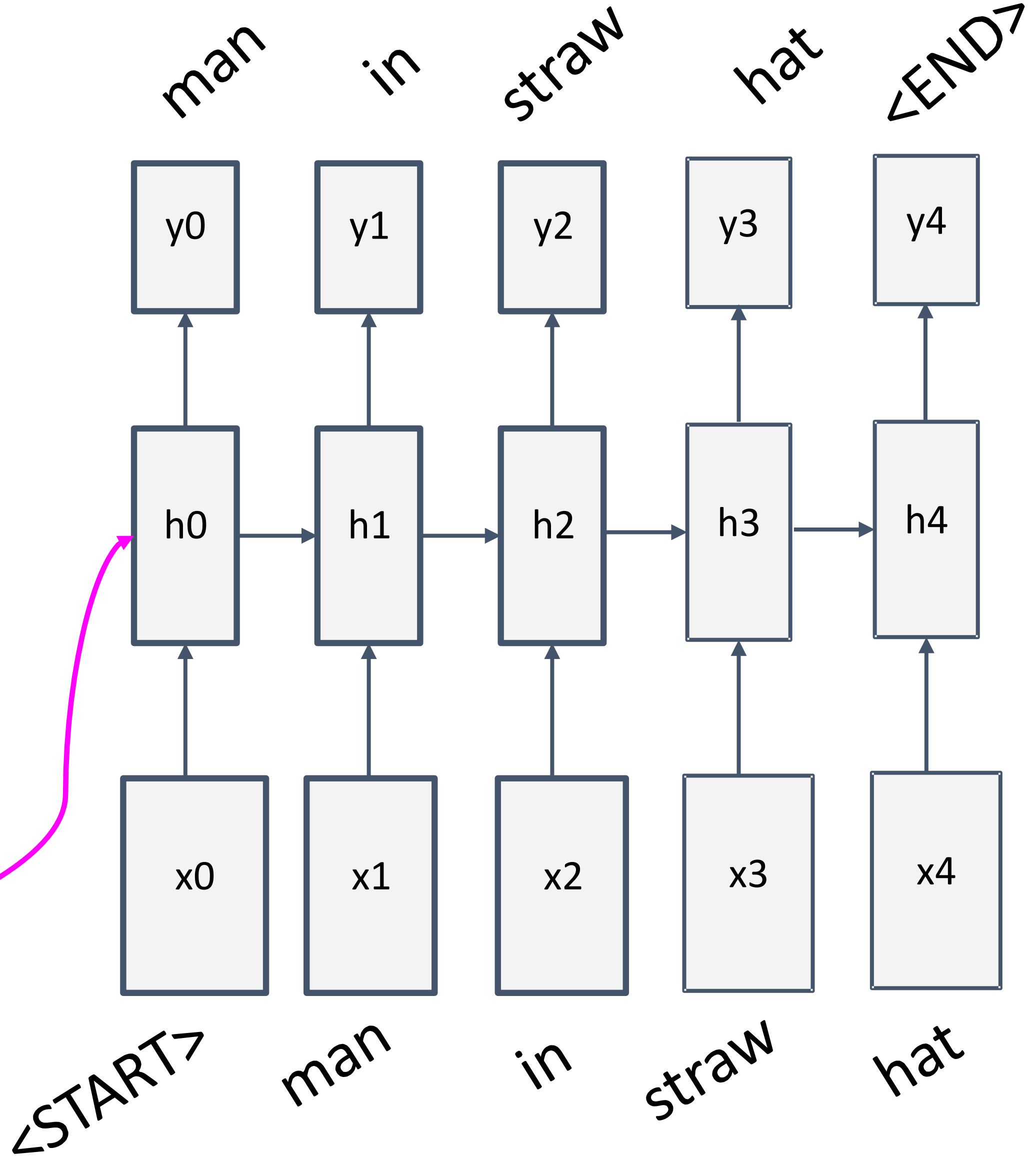


Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

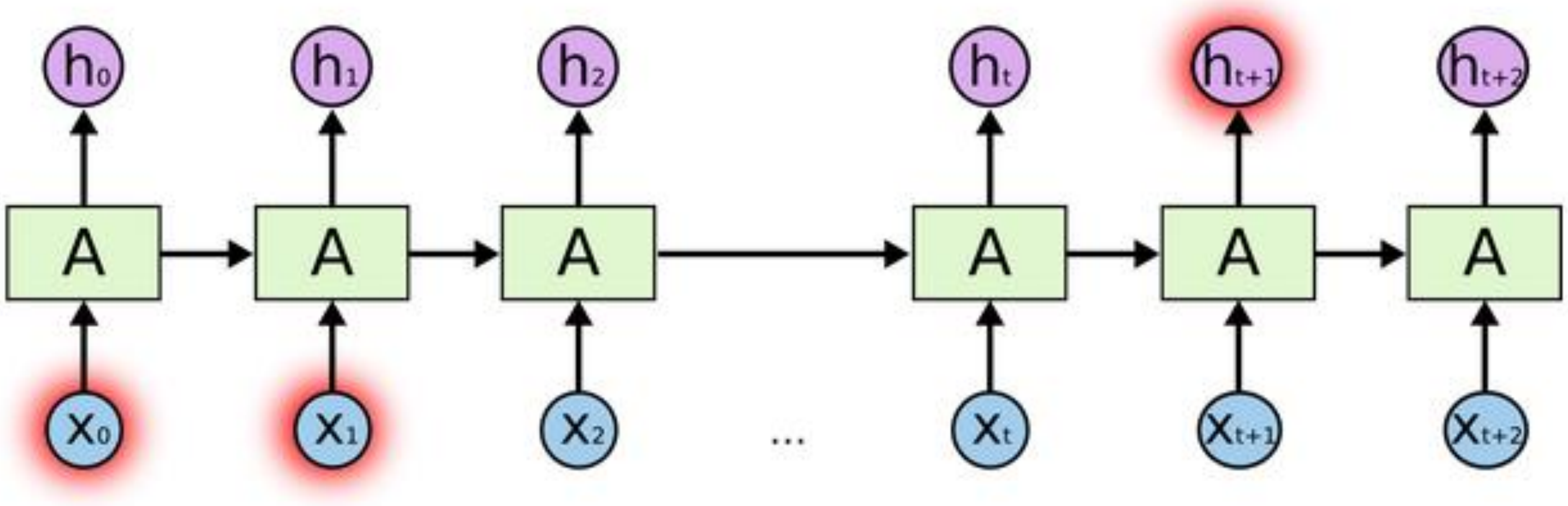
A tennis player in action on the court

Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

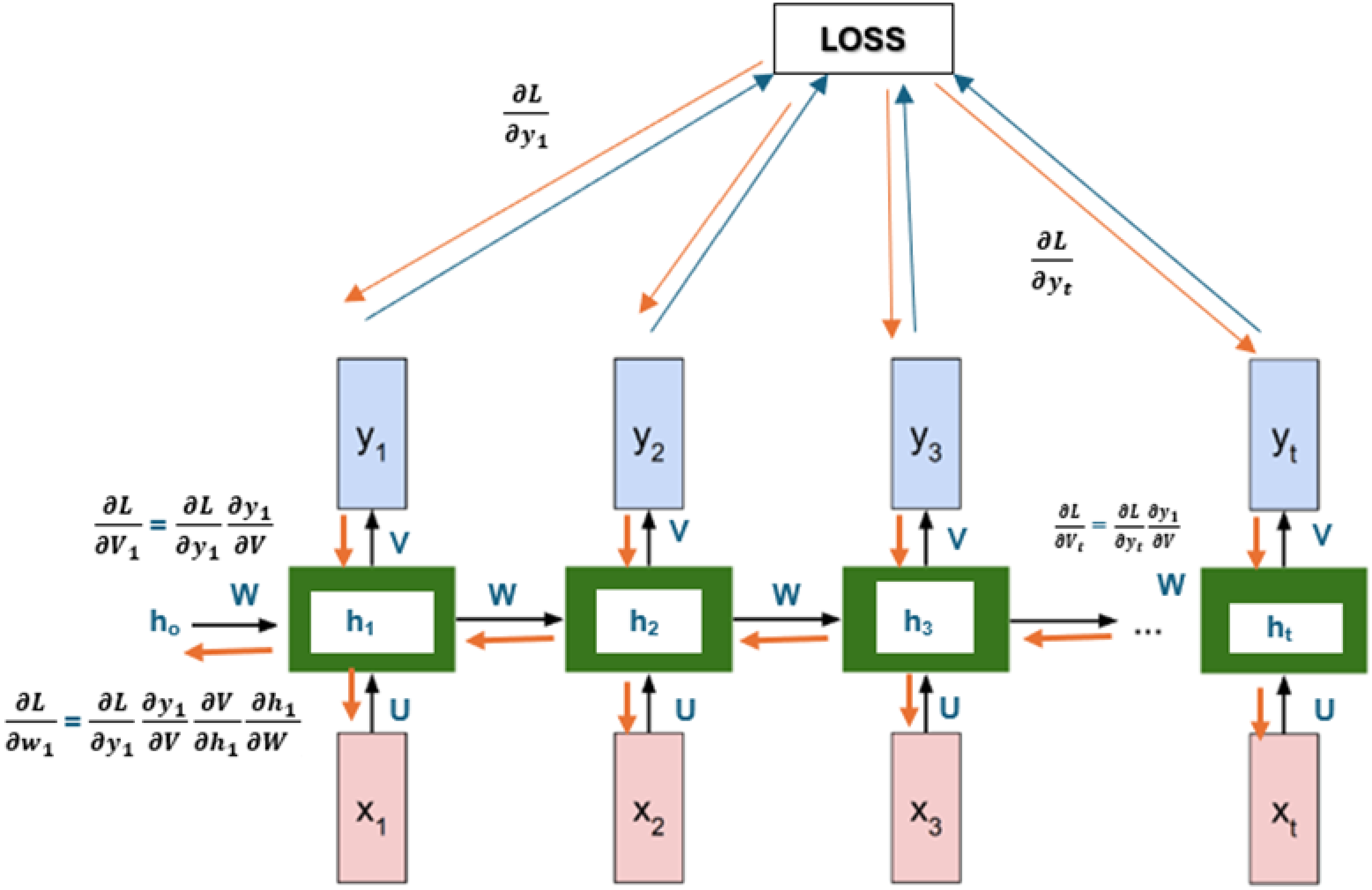
Drawback of RNN

- Thy sky is _____? blue (RNN predict easily)
- Ali live in Pakistan for 13 years. He loves playing cricket. He is fan of Imran Khan. He is fluent in _____?urdu (RNN failed to remember long term sequences)
- **As the gradients are vanish/exploding… the relationship among the data that is far way is not learnt**



h0
h1
h2
ht
ht+1
ht+2
A
A
A
A
A
A
X0
X1
X2
...
Xt
Xt+1
Xt+2

Drawback of RNN



$$\mathbf{V} = V - alpha * \frac{\partial L}{\partial V}$$

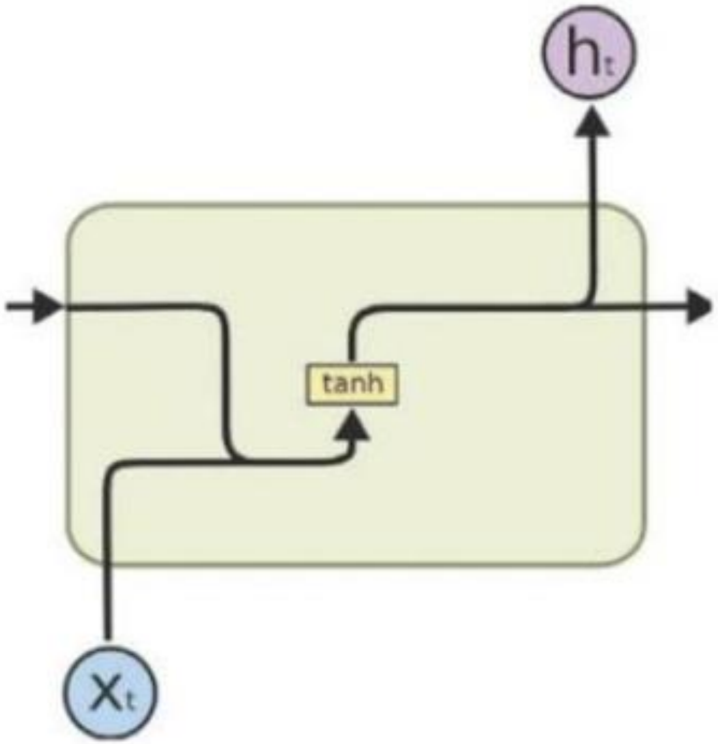
$$\mathbf{W} = W - alpha * \frac{\partial L}{\partial W}$$

$$\mathbf{U} = U - alpha * \partial L/\partial$$

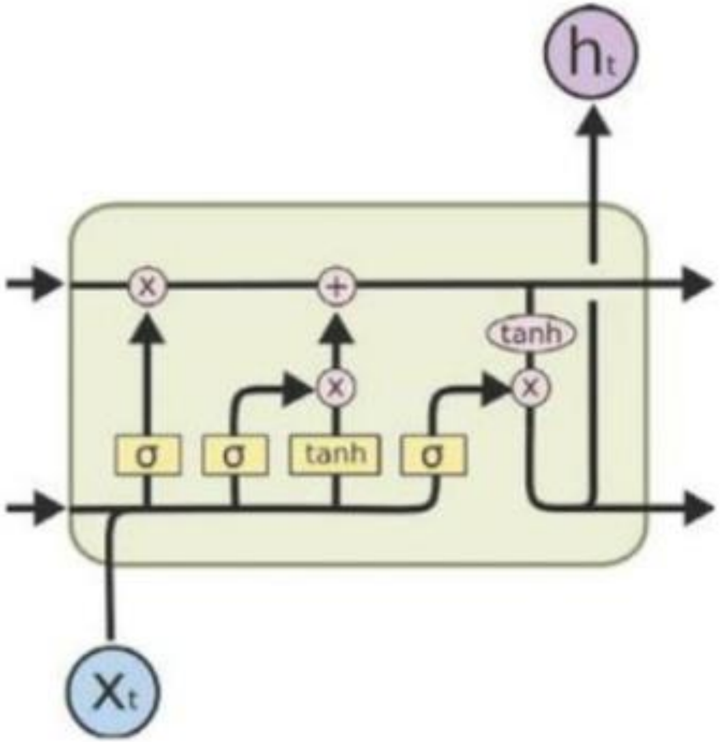
Where,

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial V_1} + \frac{\partial L}{\partial V_1} + \cdots \frac{\partial L}{\partial V_t} = \sum_{k=0}^{T} \frac{\partial L}{\partial V_t}$$

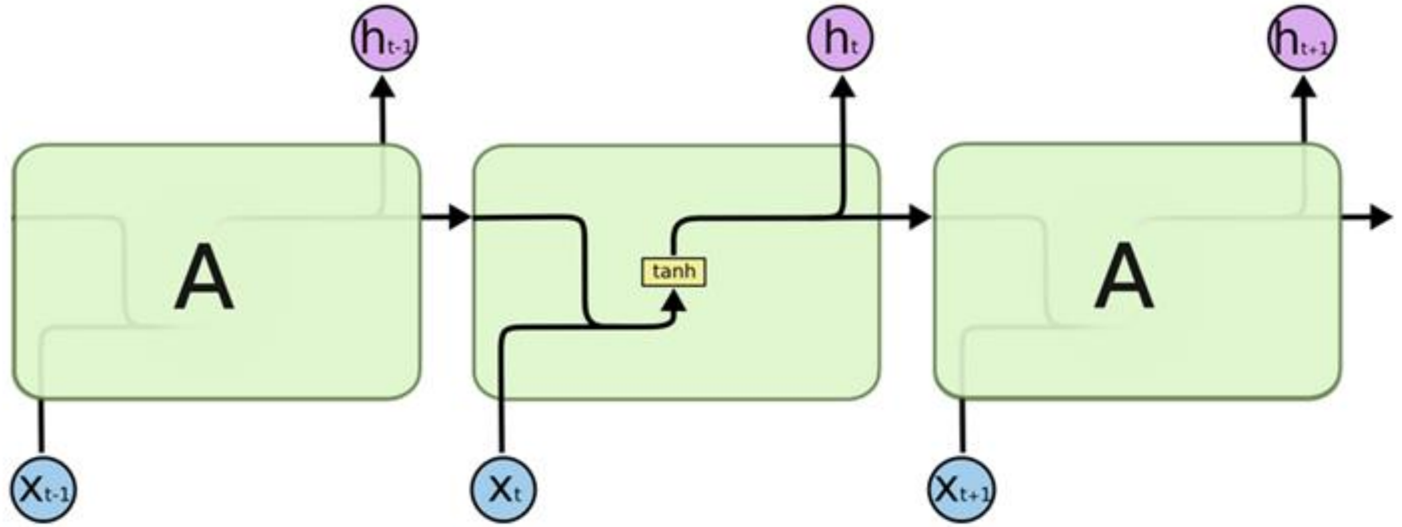
$$\frac{\partial L}{\partial U} = \frac{\partial L}{\partial U_1} + \frac{\partial L}{\partial U_1} + \cdots \frac{\partial L}{\partial U_t} = \sum_{k=0}^{T} \frac{\partial L}{\partial U_t}$$



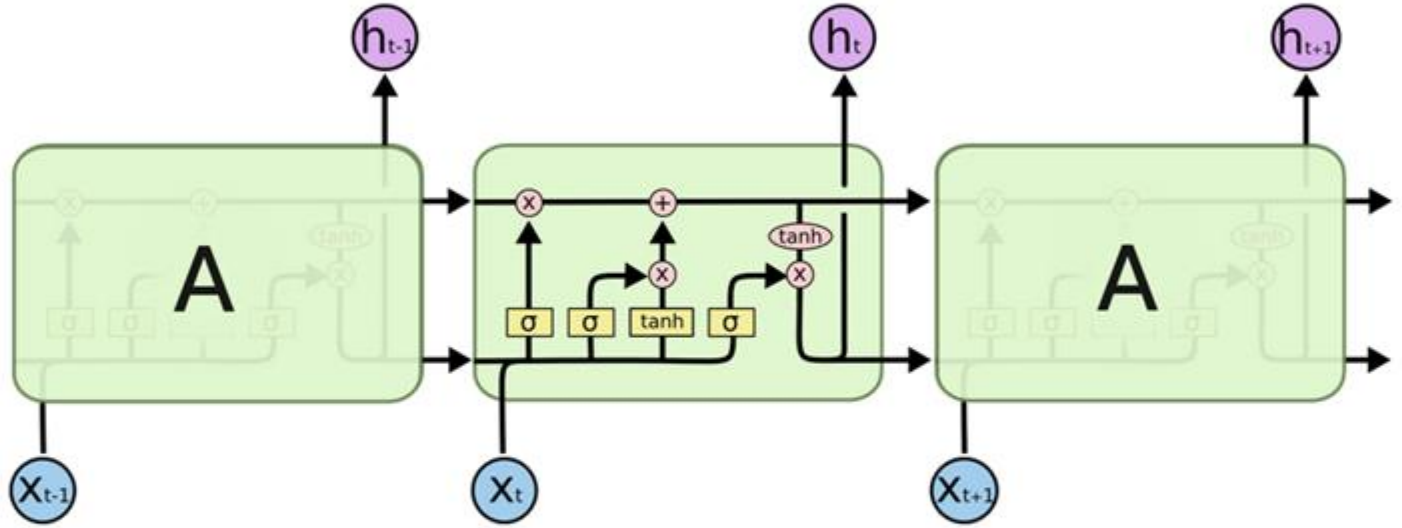
(a) RNN



(b) LSTM

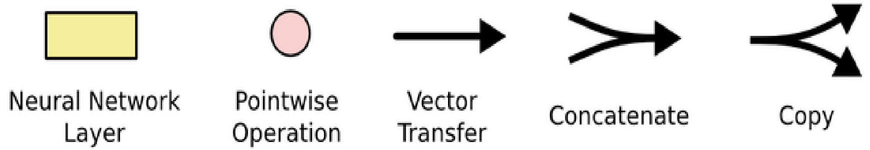


The repeating module in a standard RNN contains a single layer.



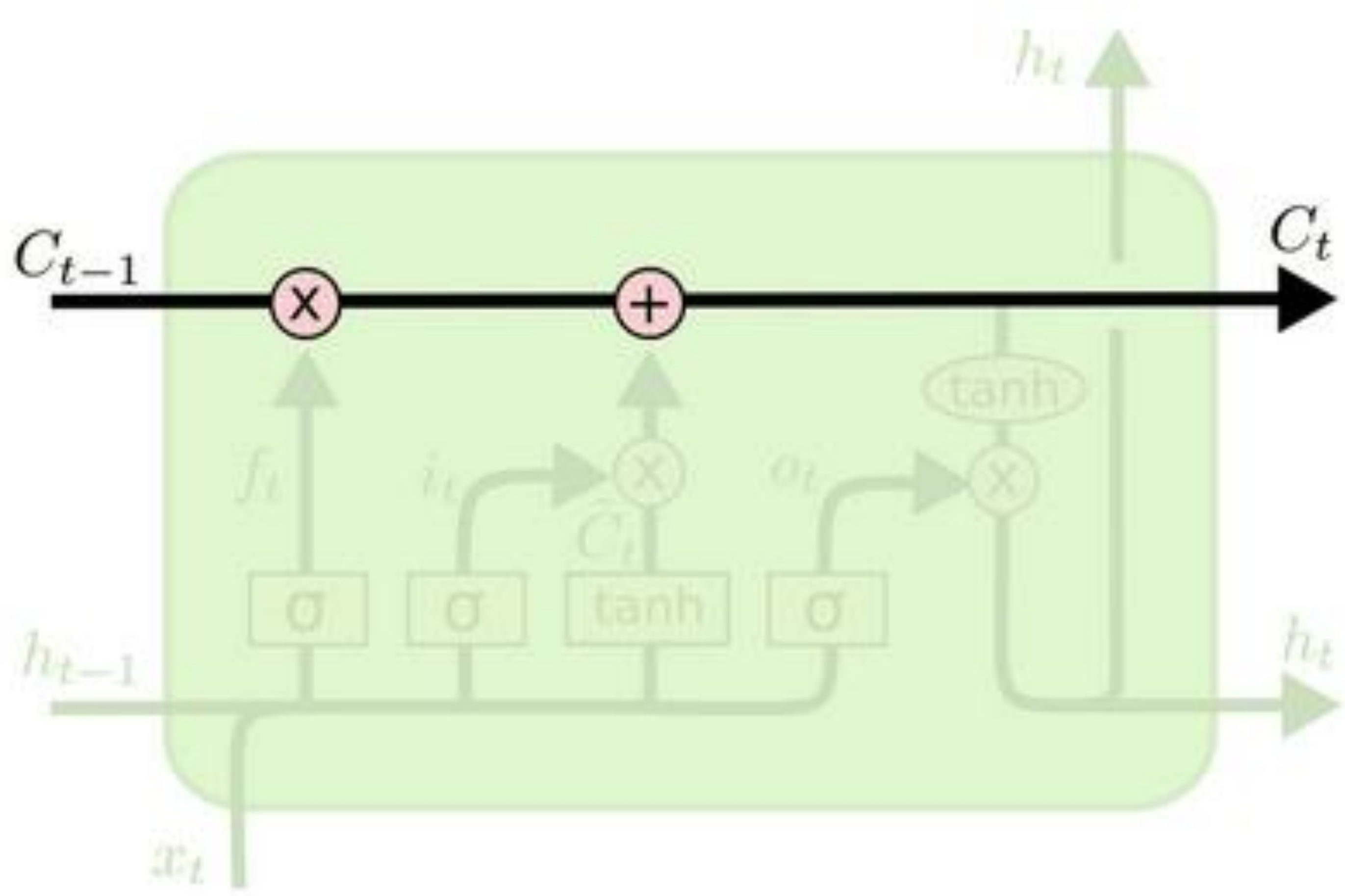
The repeating module in an LSTM contains four interacting layers.

In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others.



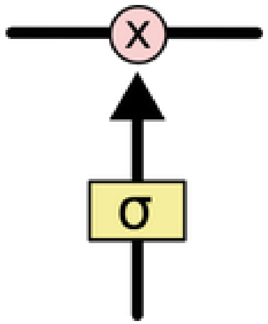
Cell state:

It's very easy for information to just flow along it unchanged.
with only **some minor linear interactions.**



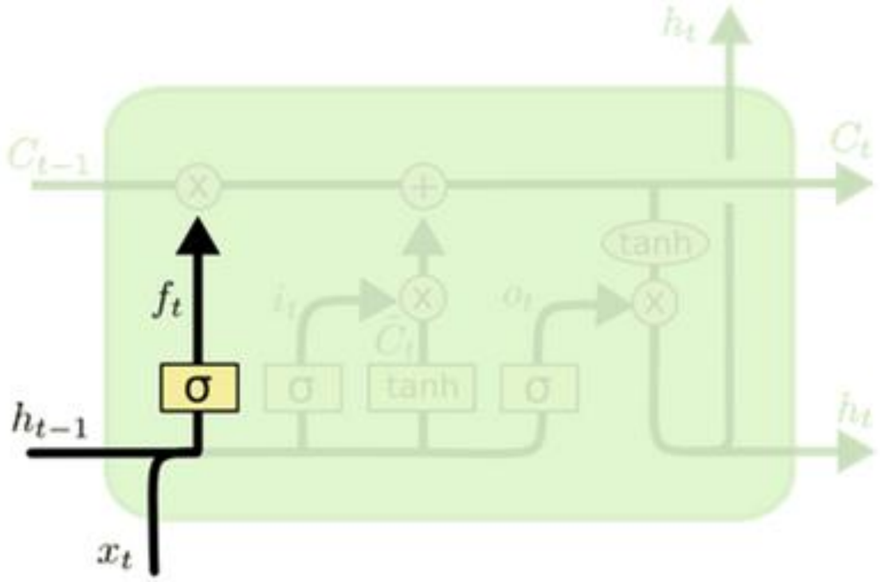
Gates

The LSTM does have the ability to **remove or add information to the cell state**, carefully regulated by structures called gates.
Gate is composed of **sigmoid NN layer** and **a pointwise multiplication operation**



The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

Forget Gate layer



Forget Gate: It is responsible for deciding what information should be **removed from the cell state**.

$$\mathbf{f_t = sigmoid\ (U_f\, x_t + W_f\, h_{(t-1)} + b_f)}$$

If $f_t = 0$, then $C_{t-1} = 0$ (forget), elseif If $f_t = 1$, then $C_{t-1} = 1$ (retain)

for example:

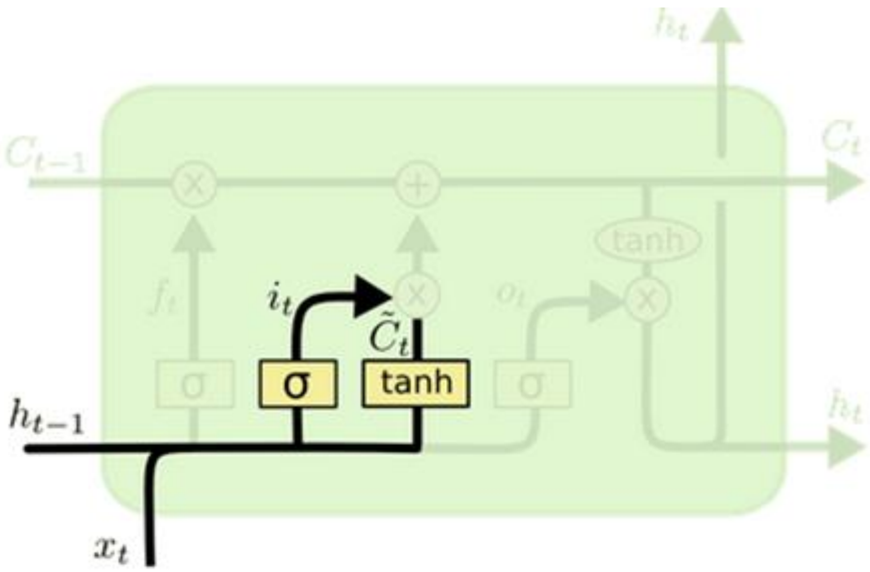
Ali is a good Singer. He lives in Lahore. **Ahmad** is also a good singer.

Note: subject is changed from **Ali to Ahmad**,

LSTM Forget gate do something wrt to change the subject.

Multiplication of ft with ct-1 helps to retain or forget the cell state

tanh layer (candidate state) and Sigmoid layer (input gate)



Candidate state role is to hold the new information. $\mathbf{i_t}$ control ***C~t*** with the help of multiplication,

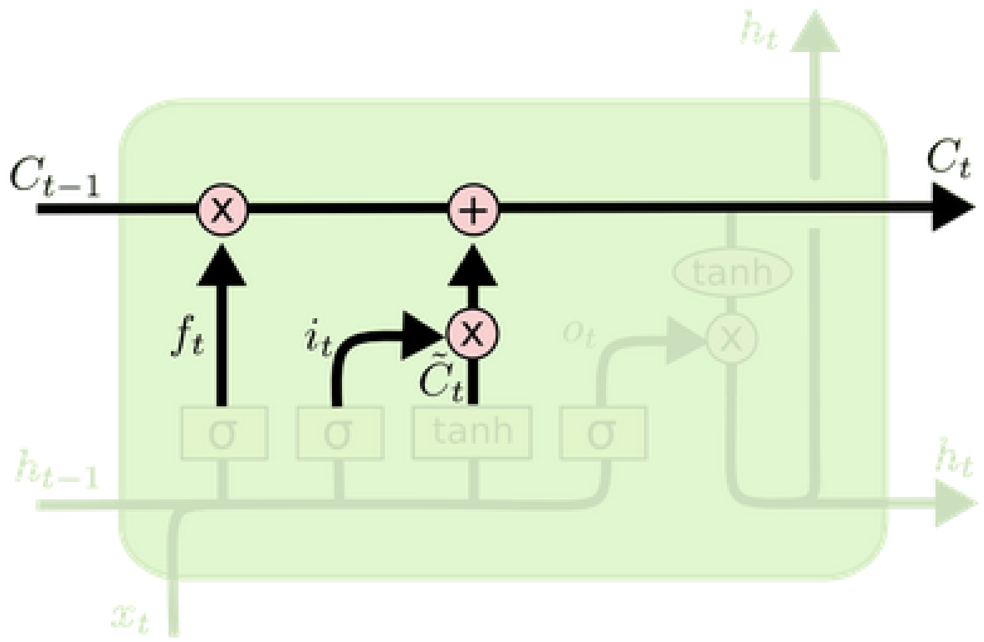
$$\boldsymbol{C\sim t} = \mathbf{tanh(U_C\, x_t + W_C\, h_{t-1} + b_c)}$$

$$\mathbf{i_t = sigmoid\ (U_i\, x_t + W_i\, h_{(t-1)} + b_i)}$$

if $\mathbf{i_t}$ =0, the ***C~t**** $\mathbf{i_t}$ = 0, and nothing is appending to ***C(t-1).*** but
if $\mathbf{i_t}$ = 1, then ***C~t**** $\mathbf{i_t}$ =1, ***C~t*** is added with $\mathbf{C_{(t-1)}}$ and update the cell state $\mathbf{C_{(t-1)}}$

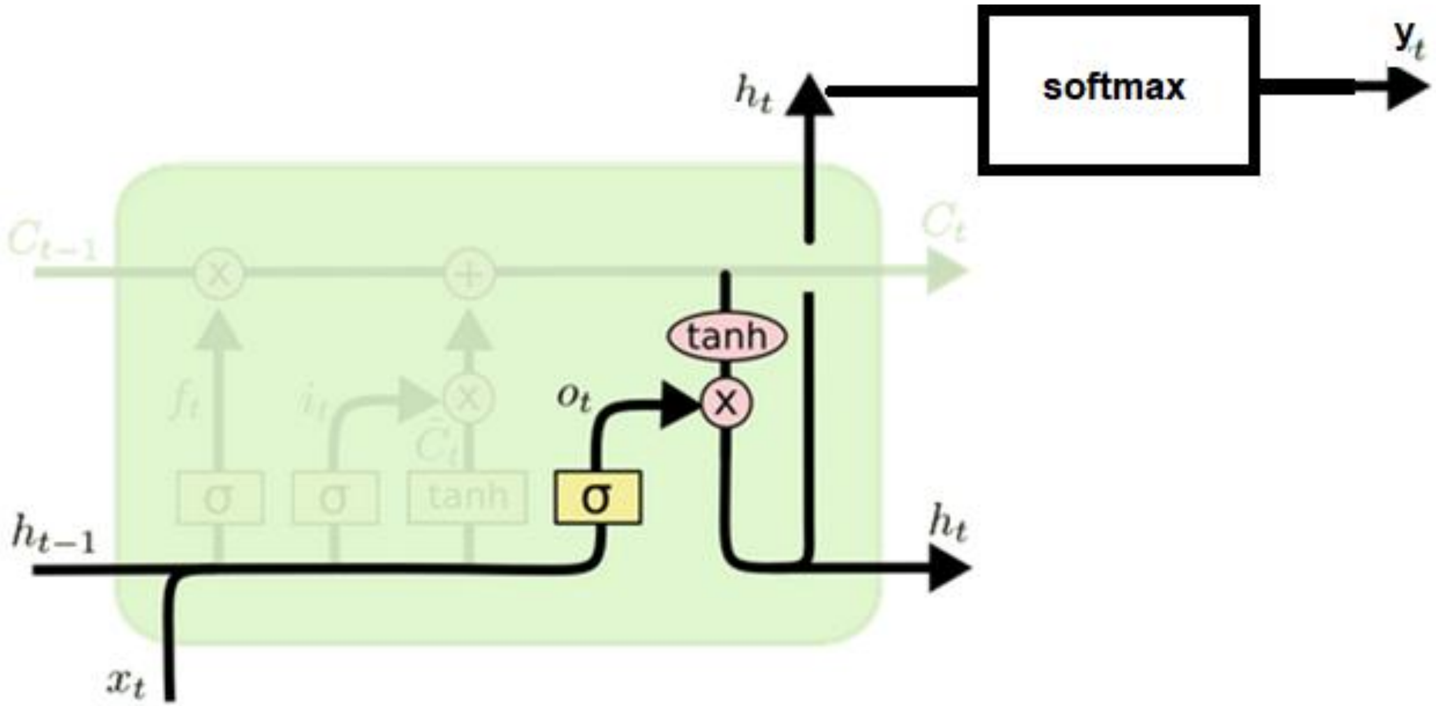
for example:
Ali is a good singer. He lives in Lahore. Ahmad is also a good singer.
Note: subject is changed from Ali to Ahmad, LSTM **Input gate change the subject from Ali to Ahmad.**

Update old cell state



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

output gate, hidden state, and output



Output Gate
A lot of information is in cell state c(t) memory.
The output gate is responsible for deciding what info
cell state (ct) to give as an output.

$$\mathbf{o_t = sigmoid\ (U_o\ x_t + W_o\ h_{(t\text{-}1)} + b_o)}$$

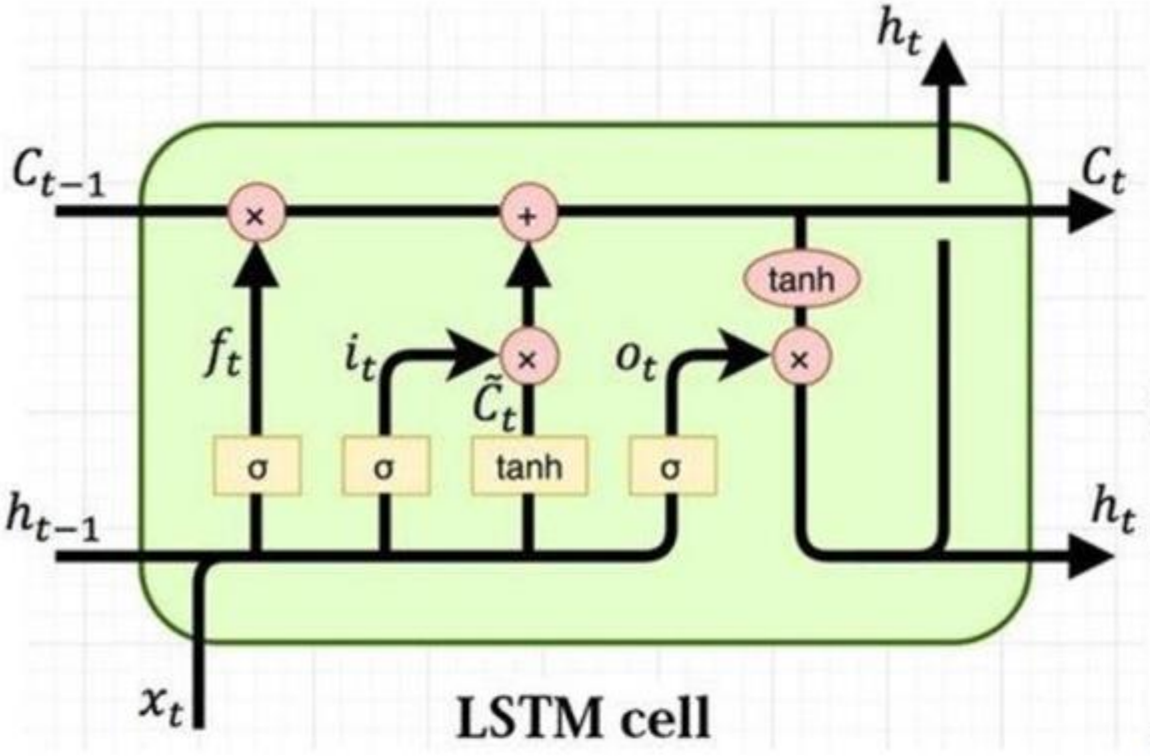
$$\mathbf{h_t = o_t *tanh\ (C_t\ )}$$

$$\mathbf{y_t = softmax * (Vh_t + b_o)}$$

Exmaple:
Ali debut album was a huge success. Congratulatio
form lot of information in cell state, we only output A

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

LSTM Cell
(Rill the missing formulations)



$i_t =$

$f_t =$

$o_t =$

$\tilde{C}_t =$

$C_t =$

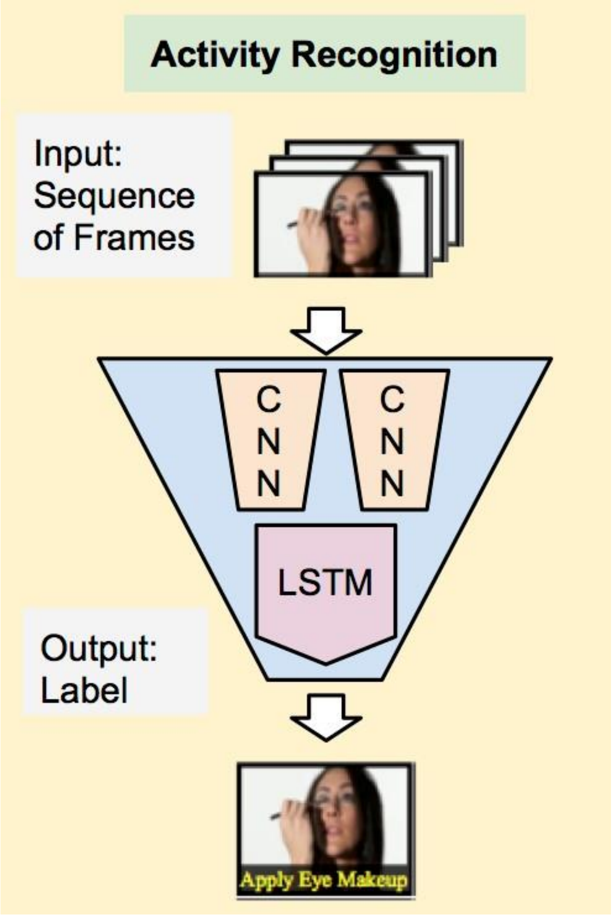
$h_t =$

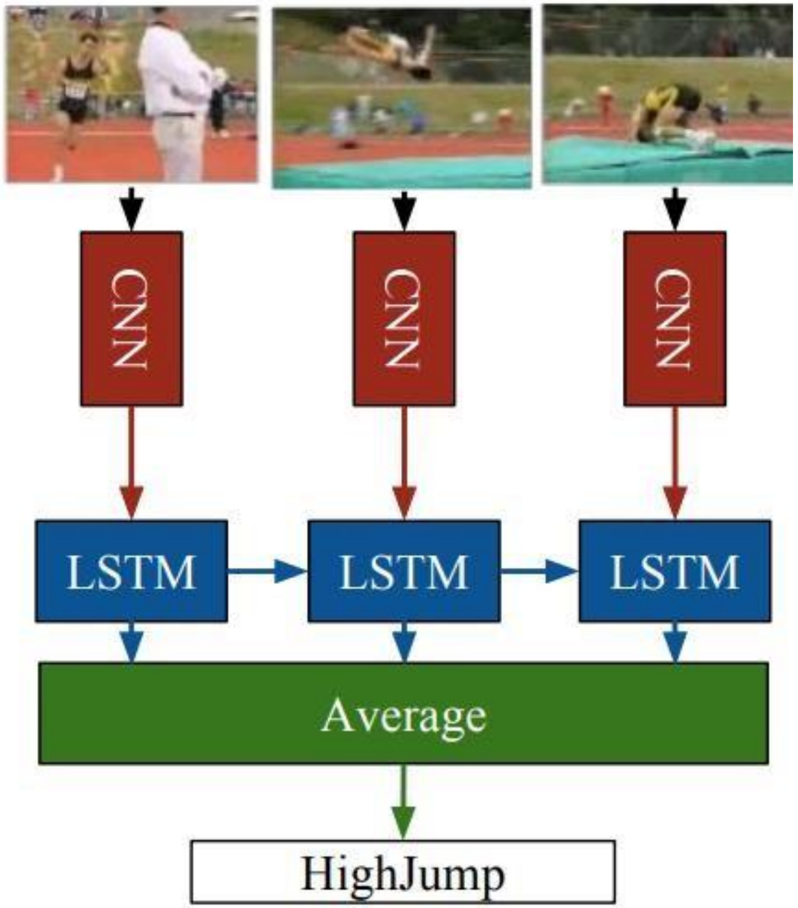
yt =

```python
import torch
import torch.nn as nn

class SimpleRNN(nn.Module):
  def __init__(self, input_size, hidden_size, output_size):
    super(SimpleRNN, self).__init__()


    self.rnn = nn.RNN(input_size, hidden_size, batch_first=True)
    self.fc = nn.Linear(hidden_size, output_size)


  def forward(self, x):

    out, _ = self.rnn(x)  # x has shape (batch_size, seq_len, input_size)


    out = self.fc(out[:, -1, :])  # Get the last hidden state
    return out
```

```python
import torch
import torch.nn as nn

class LSTMNetwork(nn.Module):
  def __init__(self, input_size, hidden_size, num_layers, output_size):
    super(LSTMNetwork, self).__init__()

    # Define the LSTM layer
    self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)

    # Define a fully connected layer
    self.fc = nn.Linear(hidden_size, output_size)

  def forward(self, x):
    # Pass the input through the LSTM layer
    lstm_out, _ = self.lstm(x)  # lstm_out contains the hidden states for all time steps

    # Extract the last hidden state
    last_hidden_state = lstm_out[:, -1, :]  # Shape: (batch_size, hidden_size)

    # Pass the last hidden state through the fully connected layer
    out = self.fc(last_hidden_state)  # Shape: (batch_size, output_size)

    return out
```

Video activity recognition





http://jeffdonahue.com/lrcn/

http://blog.qure.ai/notes/deep-learning-for-videos-action-recognition-review

Vinyals et. al. Show and Tell, 2015
Jef et. al. Long-term Recurrent Convolutional Networks 2015