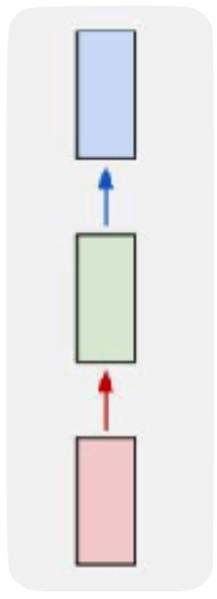
A practical guide to RNNs and Long Short-Term Memory (LSTM)

Vinay Setty

University of Stavanger

Recap: Recurrent Neural Networks

One-to-one



Vanilla Neural Networks

One-to-many

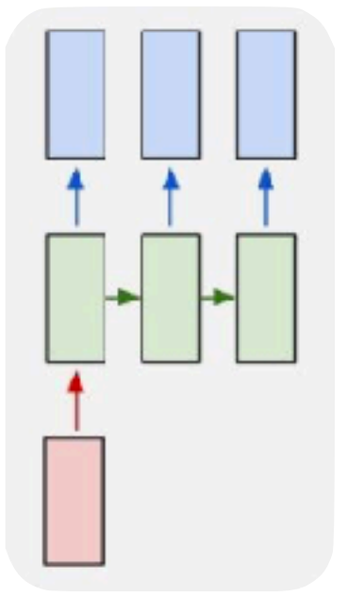
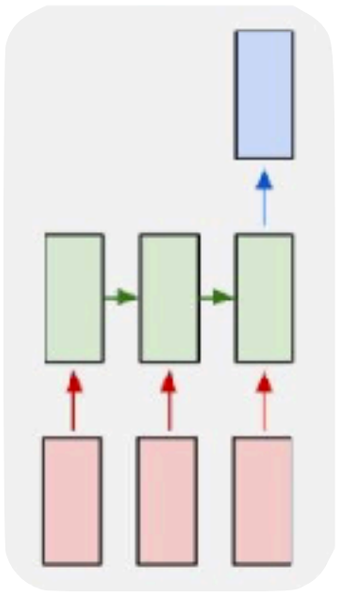


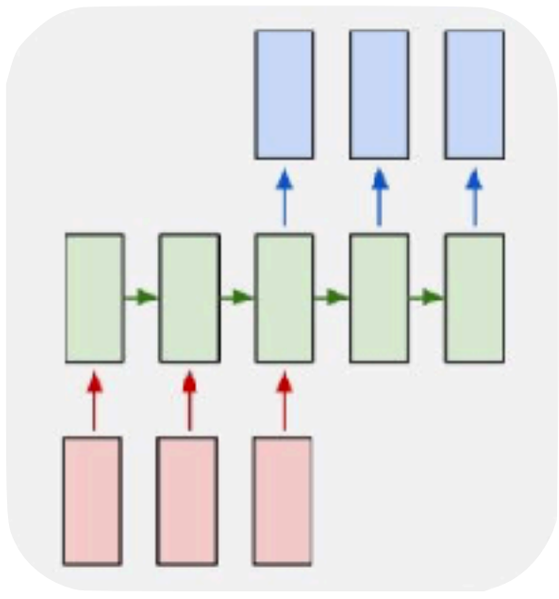
Image captioning: Image -> sequence of words

Many-to-one



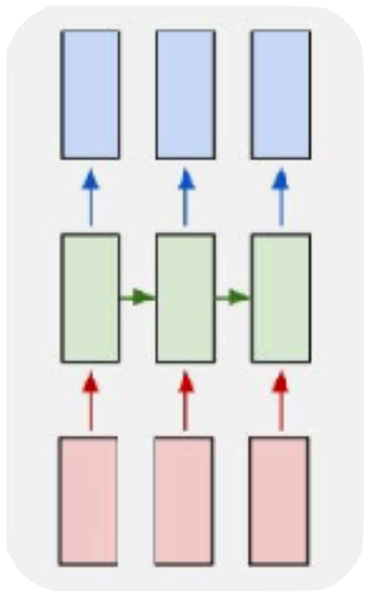
Sentiment classification: Sentence -> sentiment

Many-to-many



Machine translation: English sentence -> Norwegian sentence

Many-to-many

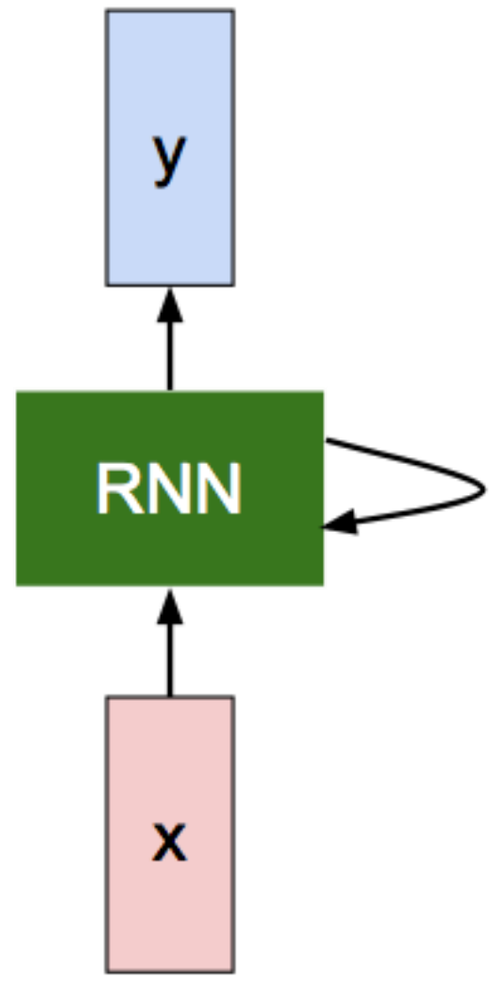


Video classification Frame by frame action

RNNs Overview

- Idea: We can process a sequence of vectors **x** by applying a recurrence formula at every time step t

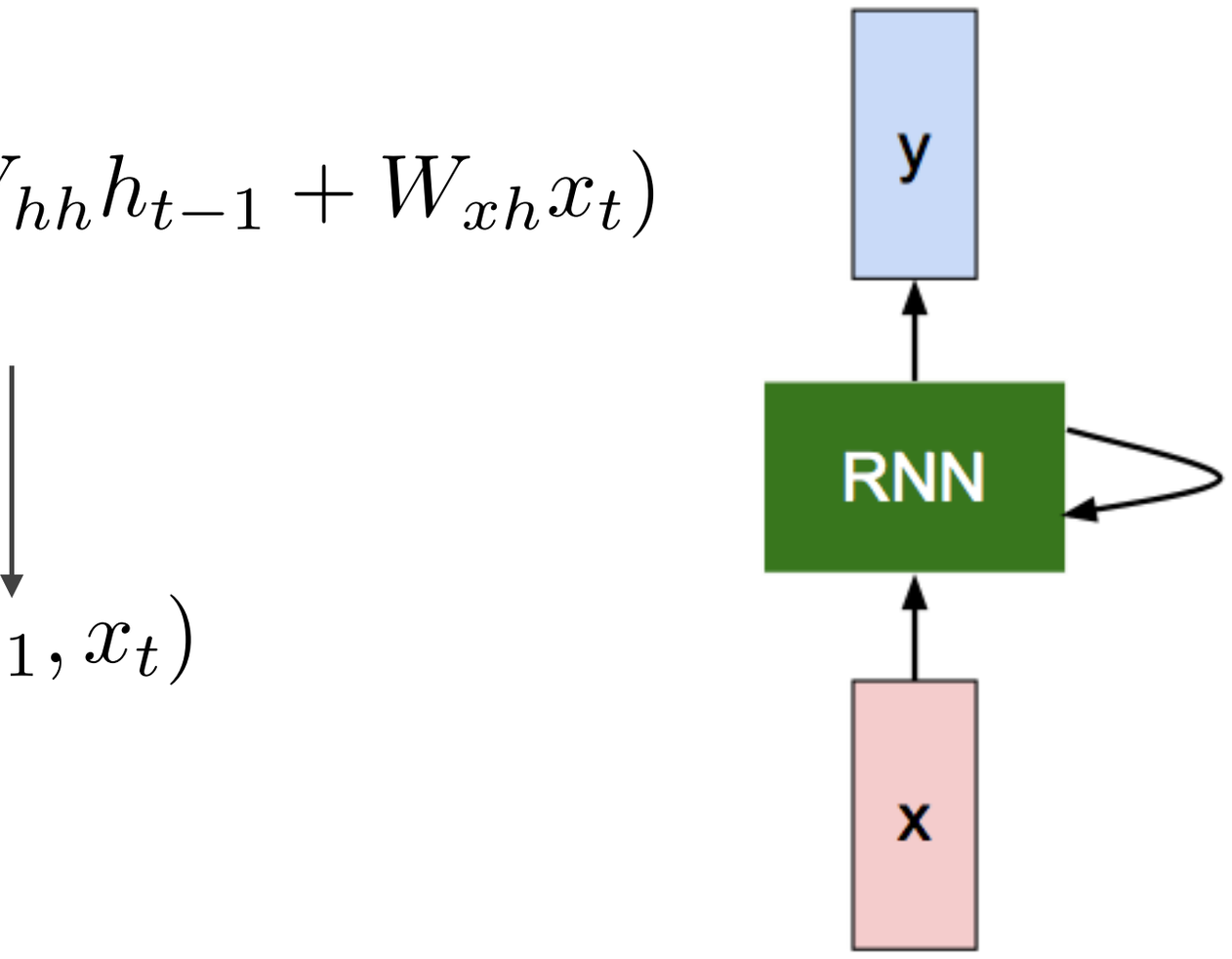
$$h_t = f_W\left(h_{t-1}, x_t\right)$$



- Note: Same parameters, weights and function used at each step.

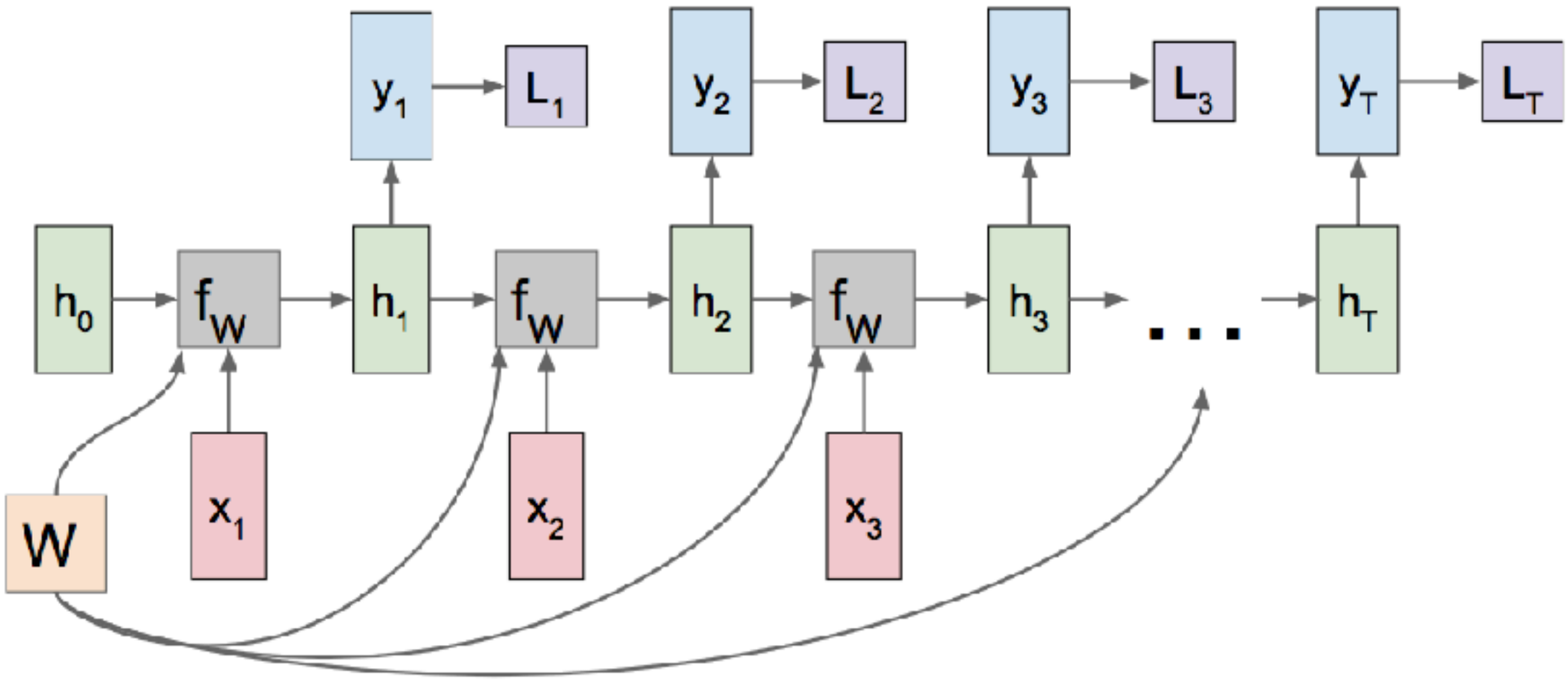
Vanilla RNN

$$h_t = \tanh\left(W_{hh} h_{t-1} + W_{xh} x_t\right)$$
$$y_t = W_{hy} h_t$$

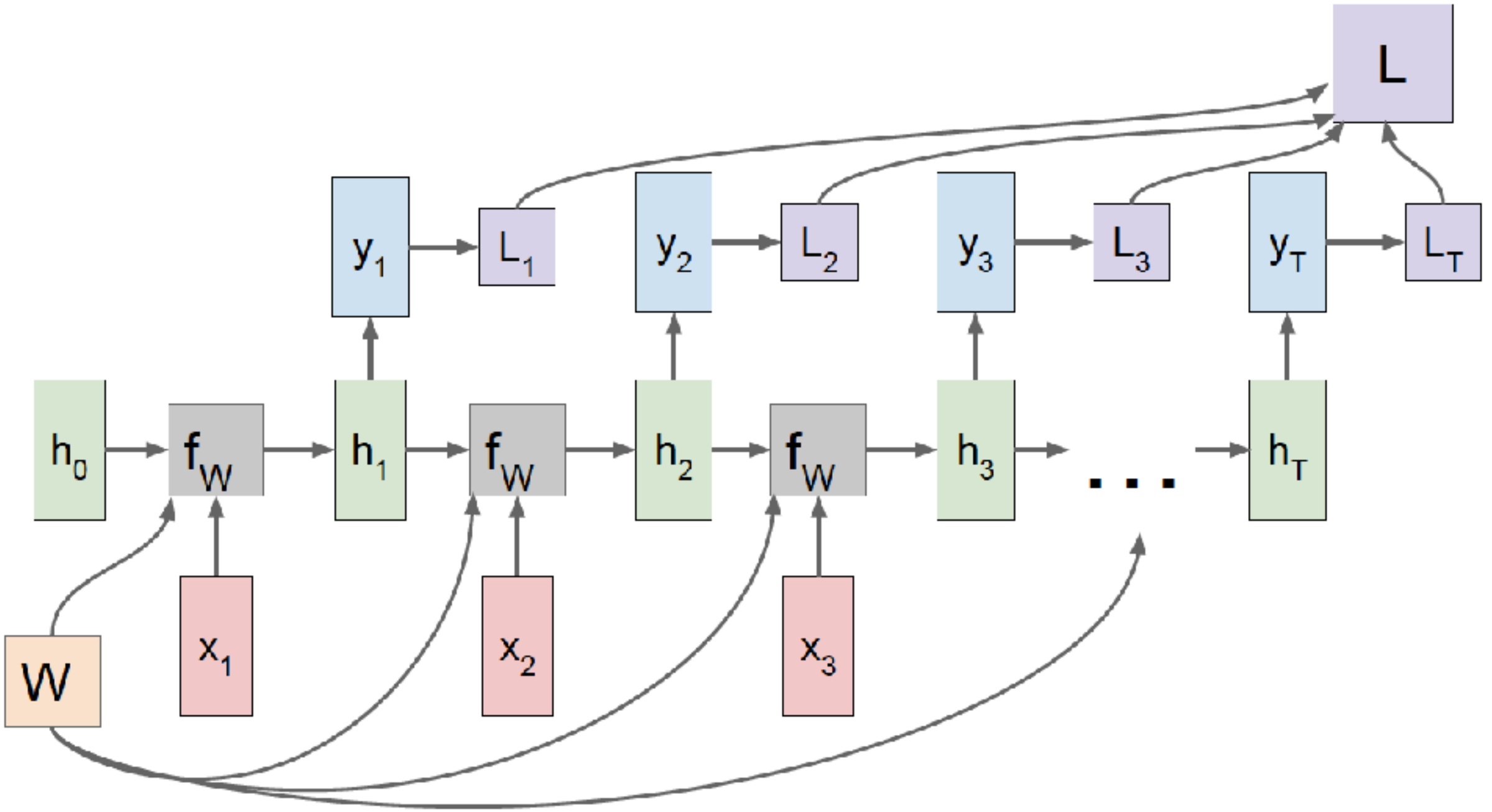


$$h_t = f_W\left(h_{t-1}, x_t\right)$$

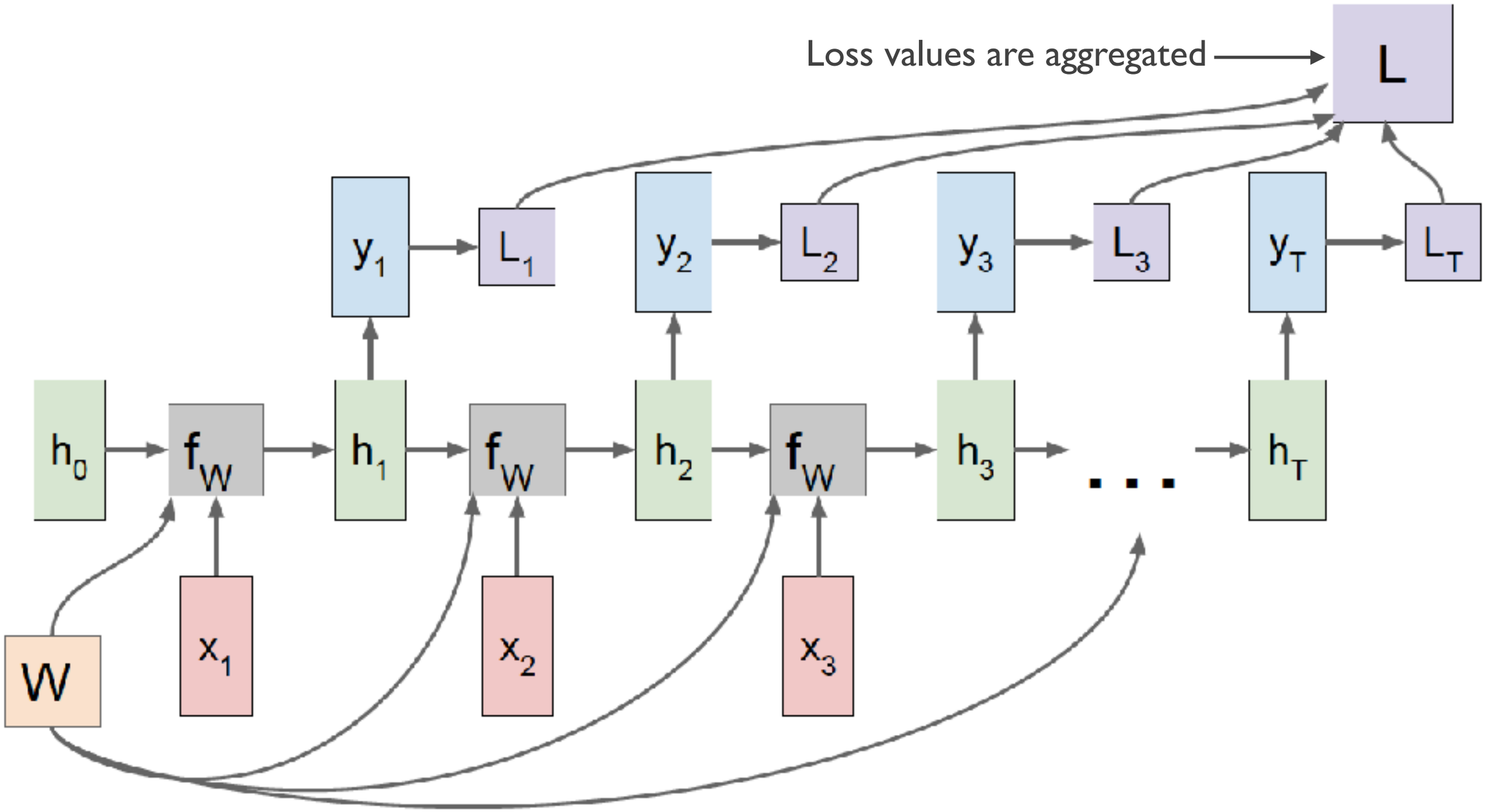
Vanilla RNN Computational Graph



Vanilla RNN Computational Graph



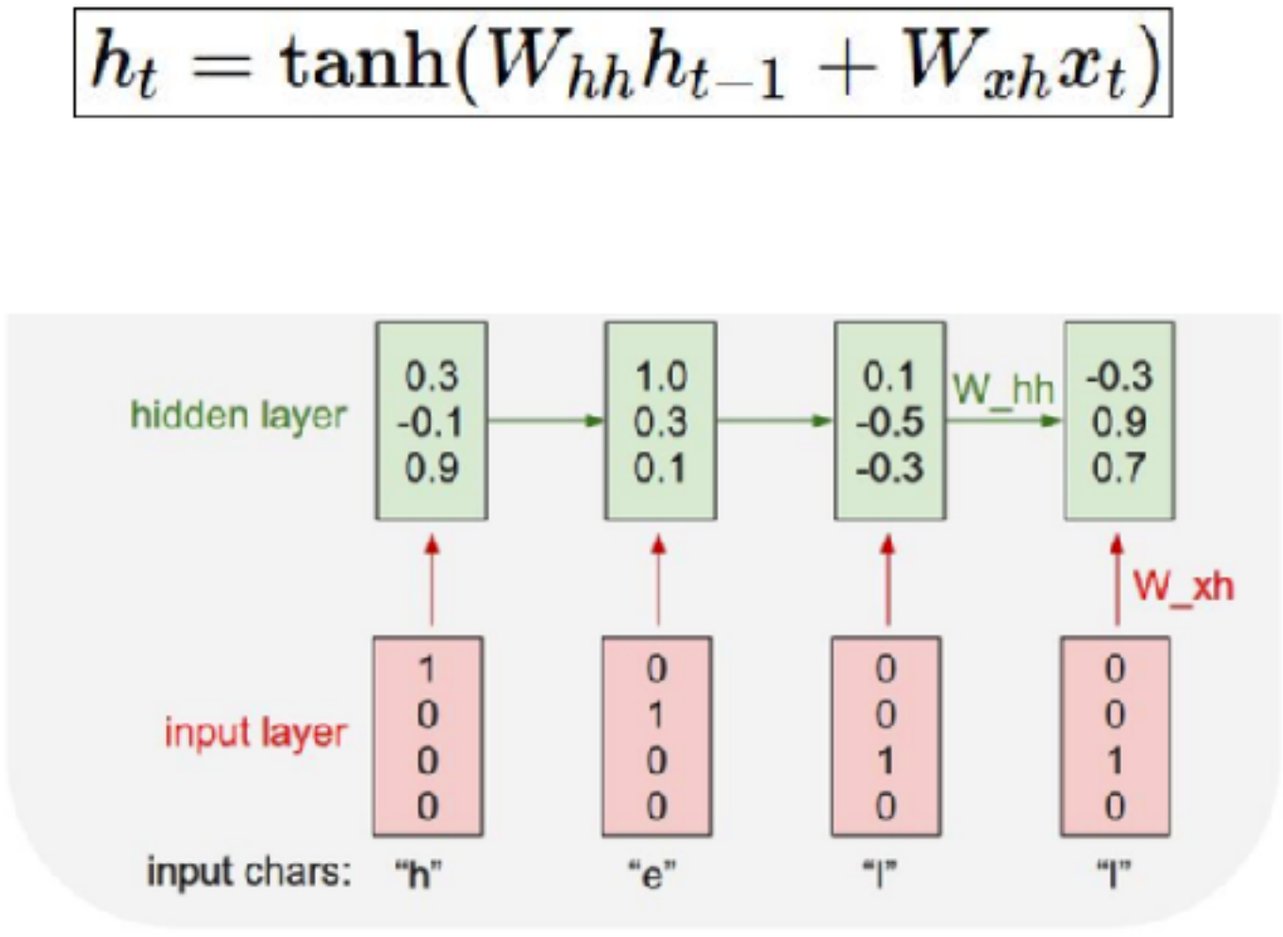
Vanilla RNN Computational Graph



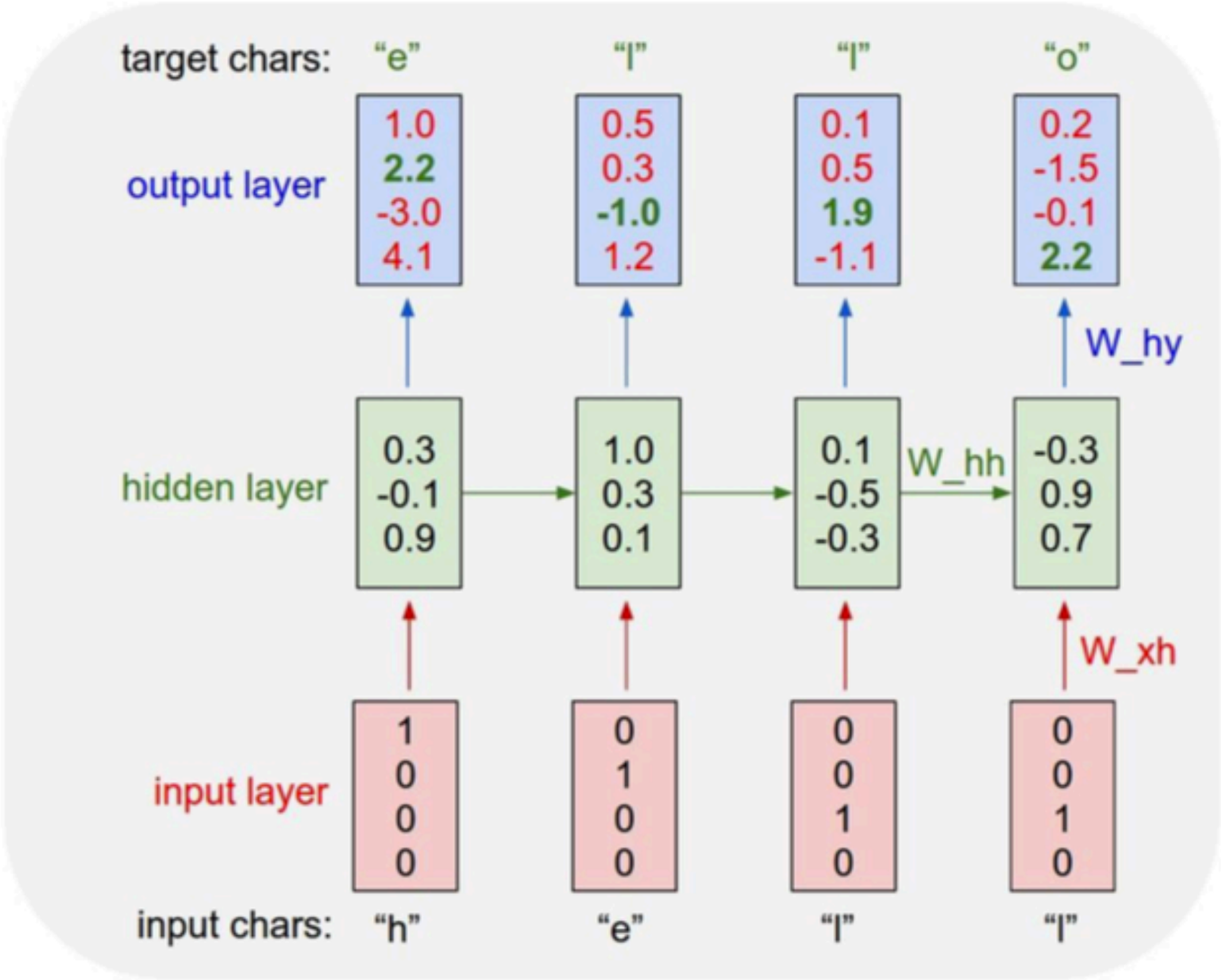
Vanilla RNN Example

- Character-level Language Model
- Given a vocabulary
- Input sequence of data
- Predict the next character
- Example:
  - Vocabulary [h,e,l,o]
  - Training sequence “hello”

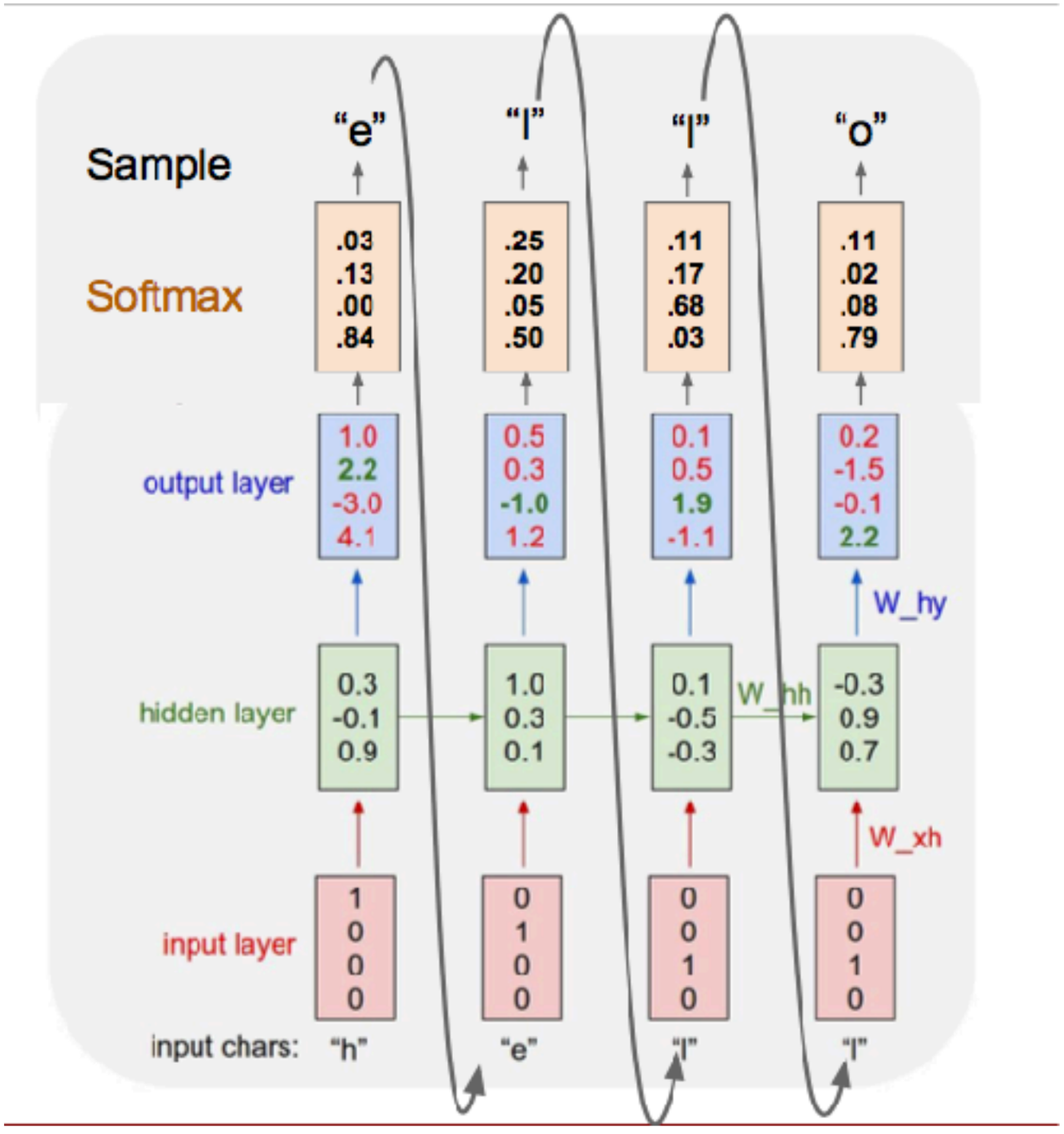
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$



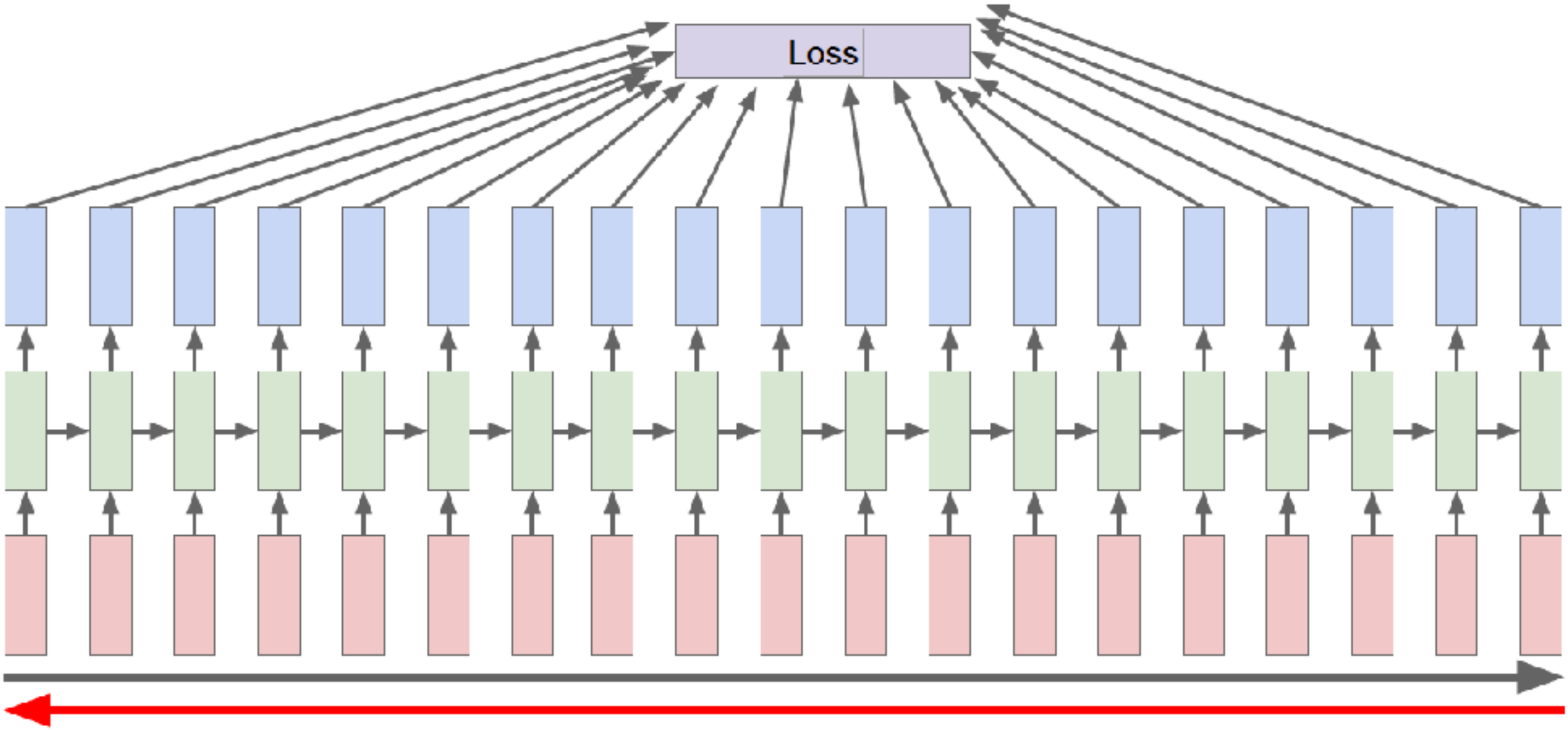
Char-RNN


target chars:
"e"
"l"
"l"
"o"
output layer
1.0
2.2
-3.0
4.1
0.5
0.3
-1.0
1.2
0.1
0.5
1.9
-1.1
0.2
-1.5
-0.1
2.2
W_hy
hidden layer
0.3
-0.1
0.9
1.0
0.3
0.1
0.1
-0.5
-0.3
W_hh
-0.3
0.9
0.7
W_xh
input layer
1
0
0
0
0
1
0
0
0
0
1
0
0
0
1
0
input chars:
"h"
"e"
"l"
"l"

Text Generation Using Char-RNN

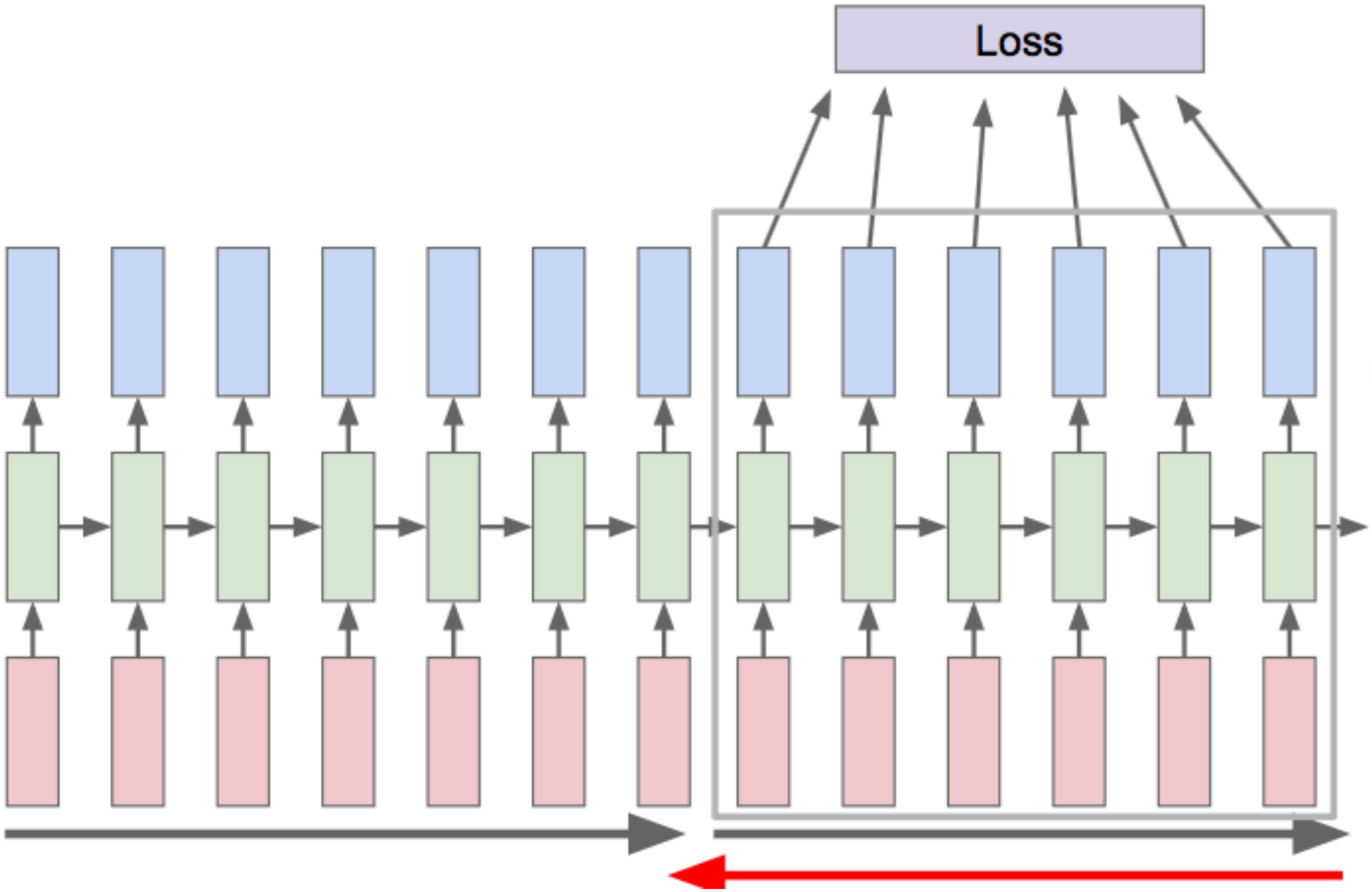


Backpropagation Through Time



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

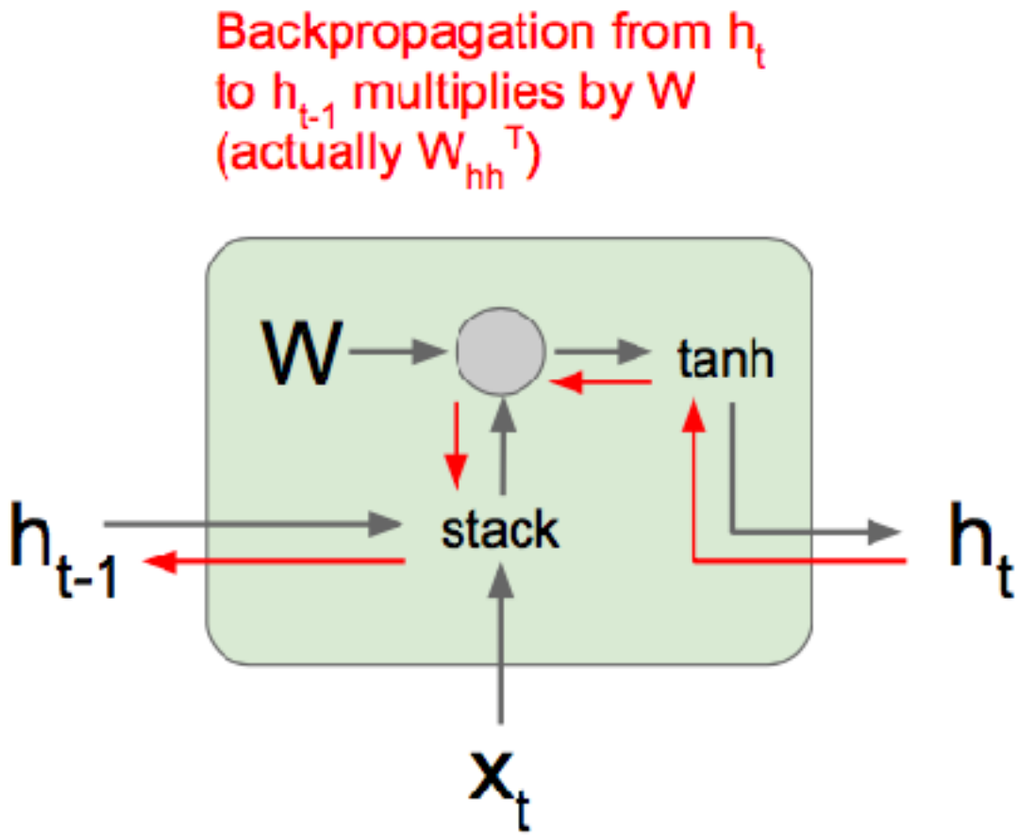
Truncated Backpropagation



Forward through entire sequence,
but only **backpropagate** for some smaller number of steps

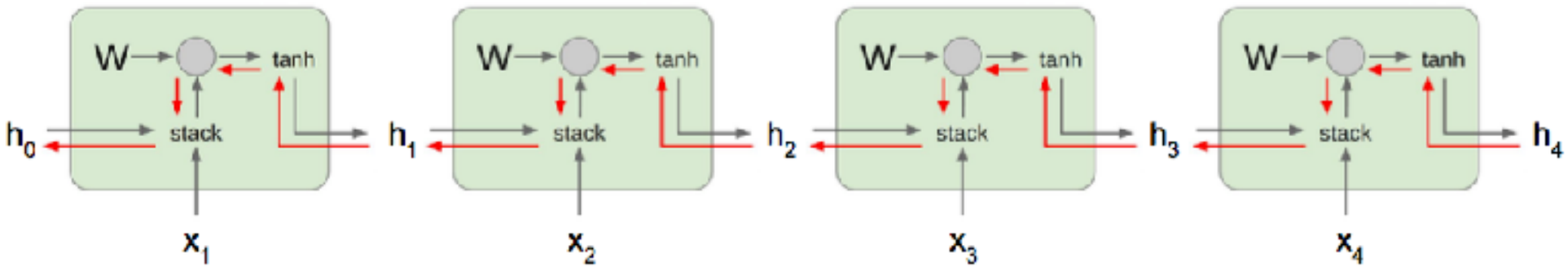
Minimal Character-level using vanilla RNN DEMO

Vanilla RNN Gradient Flow

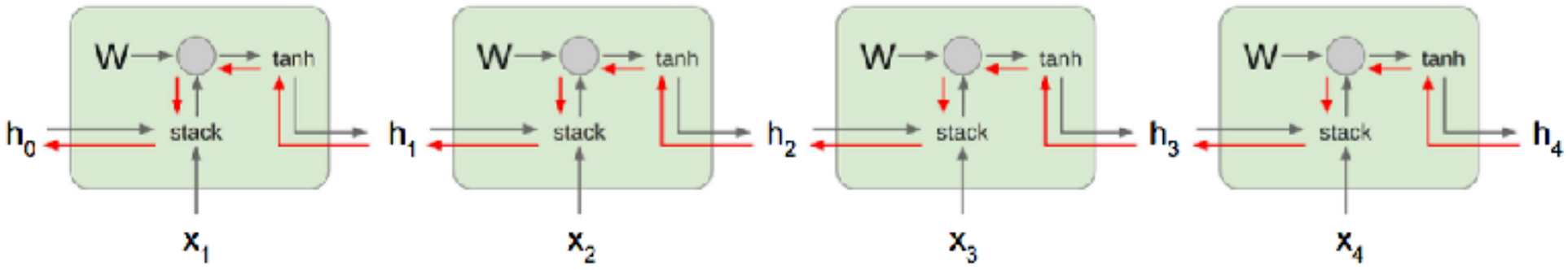


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left((W_{hh} \quad W_{hx})\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

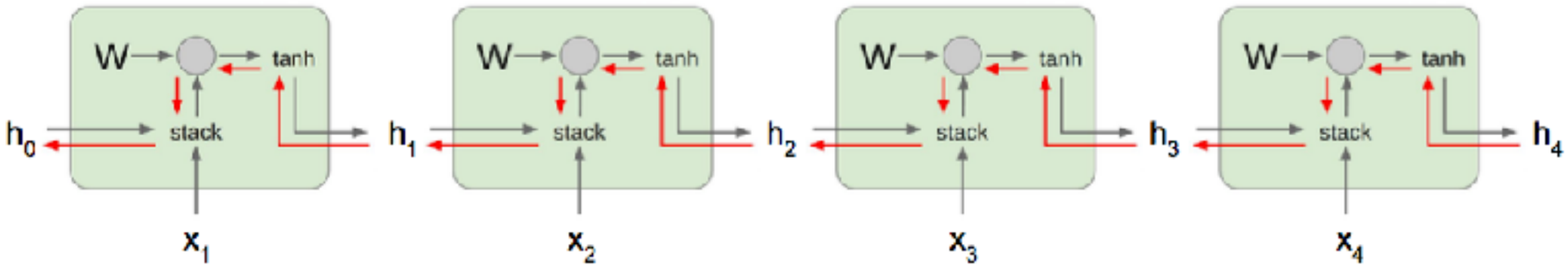


Vanilla RNN Gradient Flow



Computing the gradient for h_0 involves repeated tanh on W

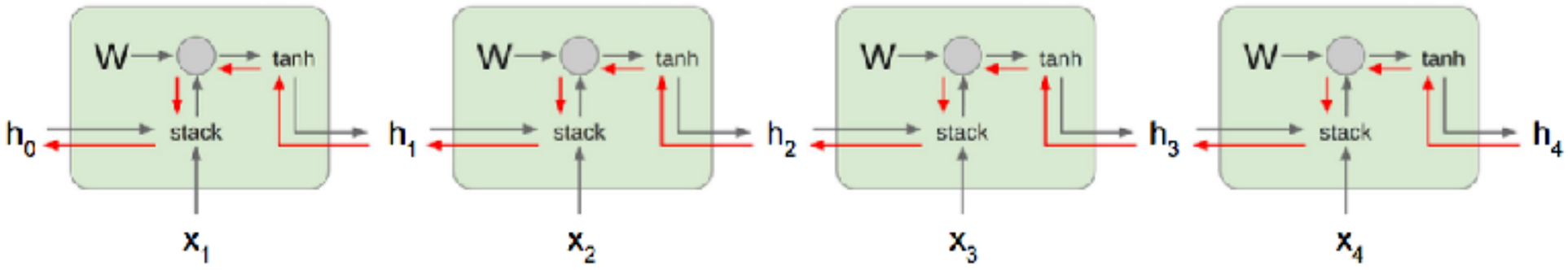
Vanilla RNN Gradient Flow



Computing the gradient for h_0 involves repeated tanh on W

Largest Weight Value > 1:
Exploding Gradient

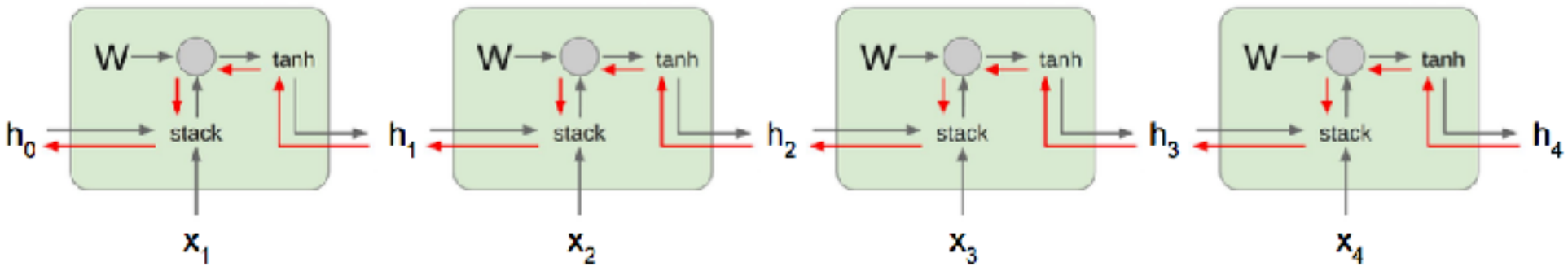
Vanilla RNN Gradient Flow



Computing the gradient for h_0 involves repeated tanh on W

Largest Weight Value > 1: **Exploding Gradient** → **Gradient Clipping:** Using some threshold

Vanilla RNN Gradient Flow

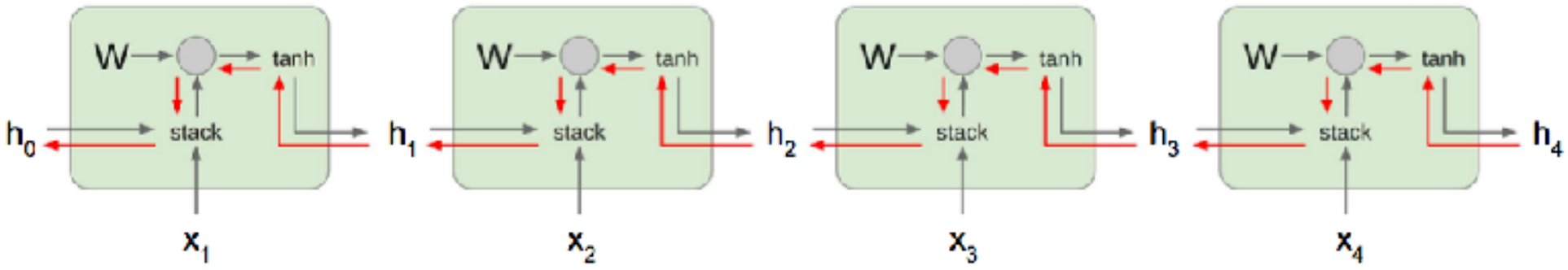


Computing the gradient for h_0 involves repeated tanh on W

Largest Weight Value > 1:
Exploding Gradient → **Gradient Clipping:** Using some threshold

Largest Weight Value < 1:
Vanishing Gradient

Vanilla RNN Gradient Flow



Computing the gradient for h_0 involves repeated tanh on W

Largest Weight Value > 1:
Exploding Gradient → **Gradient Clipping:** Using some threshold

Largest Weight Value < 1:
Vanishing Gradient → **Change the RNN Architecture!**

LSTM History

- LSTM was first proposed in 1997 by Sepp Hochreiter and **Jürgen Schmidhuber** (Neural Computation. 9 (8): 1735–1780.)
  - Submission to NIPS was rejected in 1997!
- Today used by
  - Microsoft for conversational speech recognition
  - Google for machine translation
  - Apple for siri (on iphones)
  - IBM, Baidu, Samsung and so on…

LSTM

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

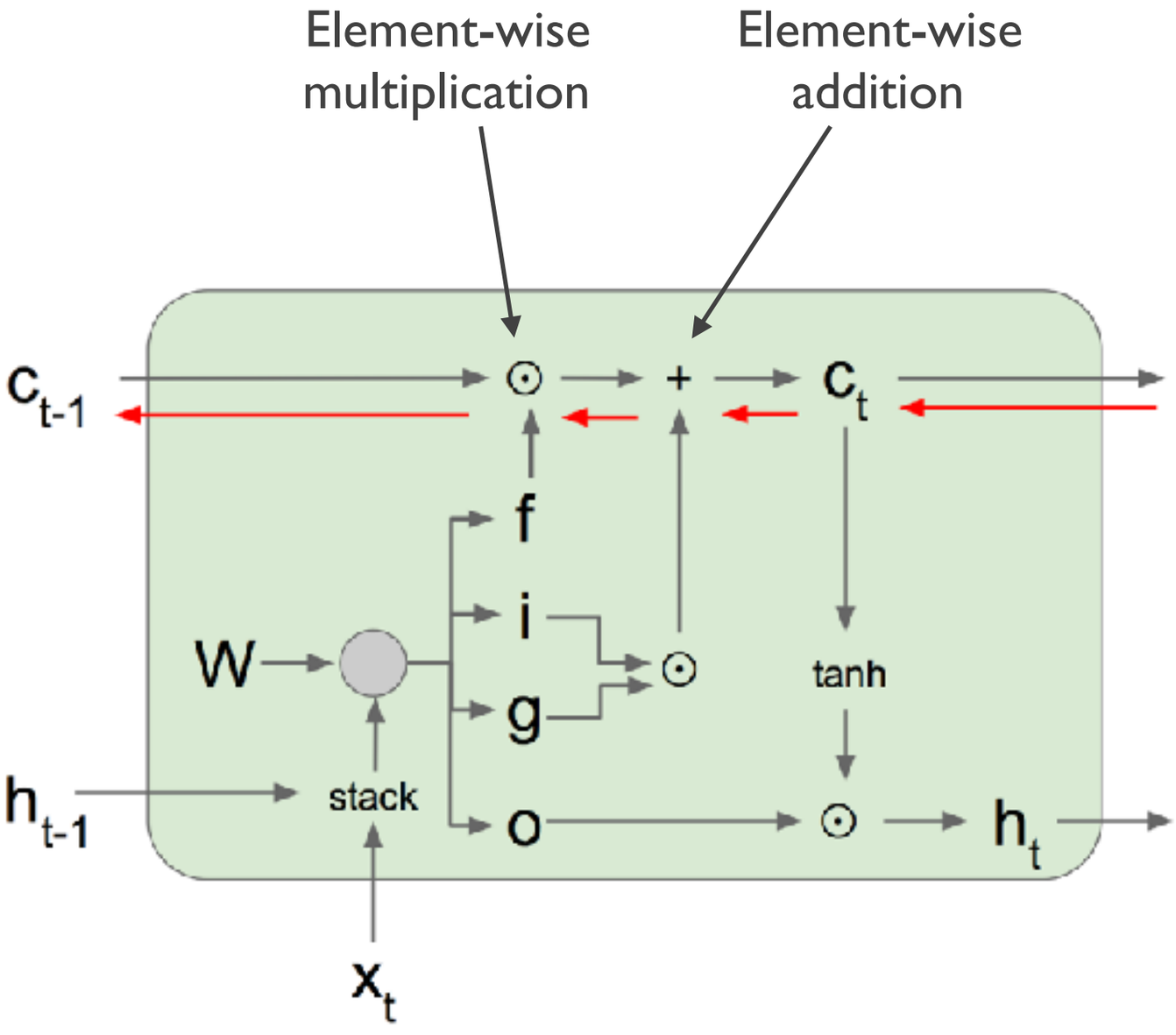
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

[Hochreiter et al., 1997]

LSTM Gradient Flow



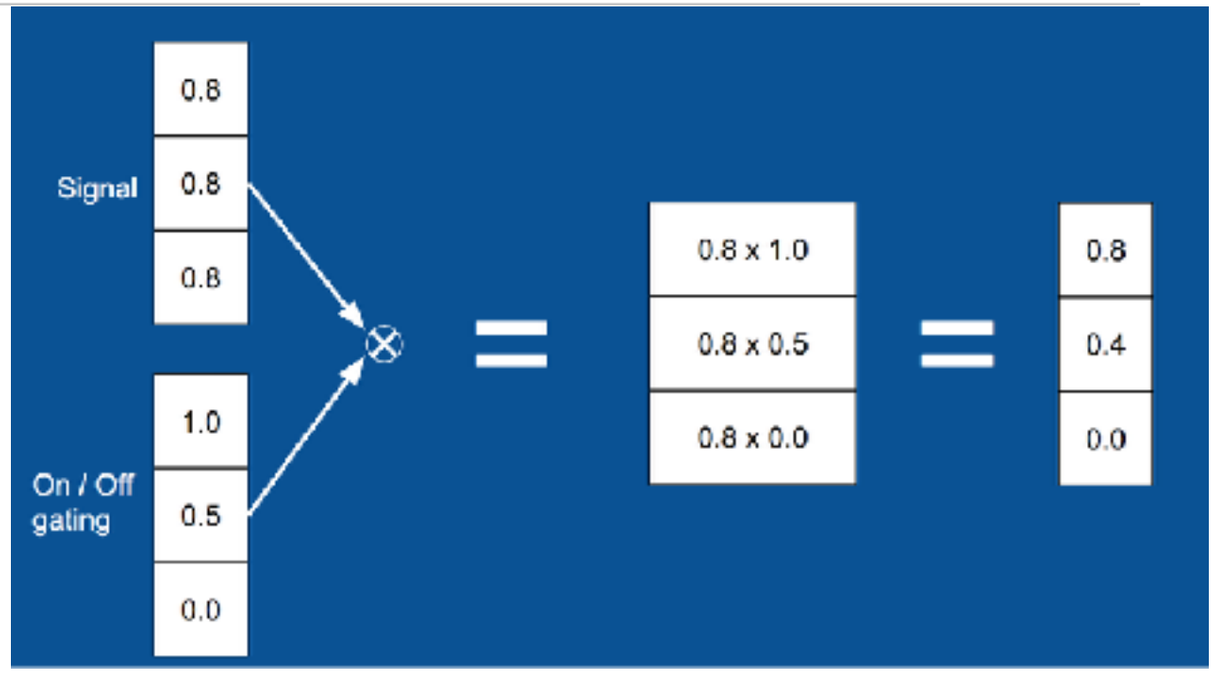
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

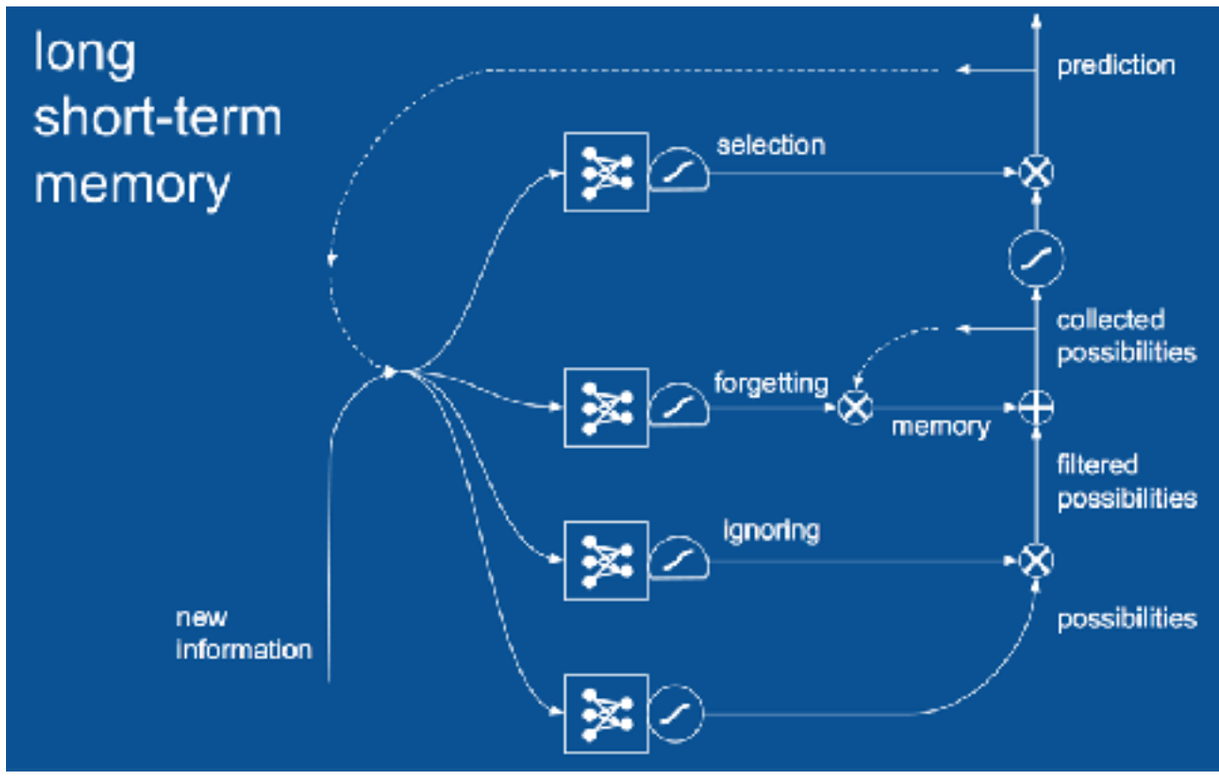
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

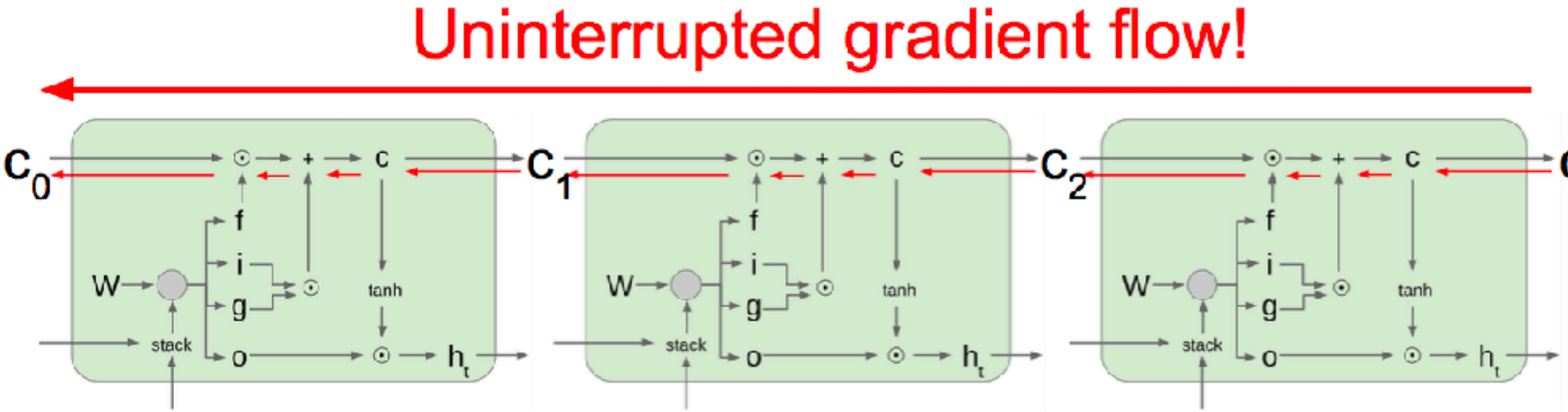
LSTM Intuition

- The problem of vanishing and exploding gradient happens due to repeated matrix multiplication in vanilla RNN
  - Recall backward propagation using the chain rule
- To avoid this we use
  - Memory cell in addition to hidden state
  - Memory cell is updated element-wise multiplication and addition
  - Gated control of the gradients to control what goes through in the forward pass and to let things pass in the backward pass
  - weights of all the gates are jointly trained





LSTM gradient flow fix



Variants of LSTM

- GRU [Learning phrase representations using RNN encoder-decoder for statistical machine translation, Cho et al. 2014]

$$
\begin{aligned}
r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\
z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\
\tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t
\end{aligned}
$$

LSTM Variants

- An Empirical Exploration of Recurrent Network Architectures, Jozefowicz et al., 2015

MUT1:

$$
\begin{aligned}
z &= \text{sigm}(W_{\text{xz}} x_t + b_{\text{z}}) \\
r &= \text{sigm}(W_{\text{xr}} x_t + W_{\text{hr}} h_t + b_{\text{r}}) \\
h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + \tanh(x_t) + b_{\text{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT2:

$$
\begin{aligned}
z &= \text{sigm}(W_{\text{xz}} x_t + W_{\text{hz}} h_t + b_{\text{z}}) \\
r &= \text{sigm}(x_t + W_{\text{hr}} h_t + b_{\text{r}}) \\
h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh} x_t + b_{\text{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT3:

$$
\begin{aligned}
z &= \text{sigm}(W_{\text{xz}} x_t + W_{\text{hz}} \tanh(h_t) + b_{\text{z}}) \\
r &= \text{sigm}(W_{\text{xr}} x_t + W_{\text{hr}} h_t + b_{\text{r}}) \\
h_{t+1} &= \tanh(W_{\text{hh}}(r \odot h_t) + W_{xh} x_t + b_{\text{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

Application: Image Captioning

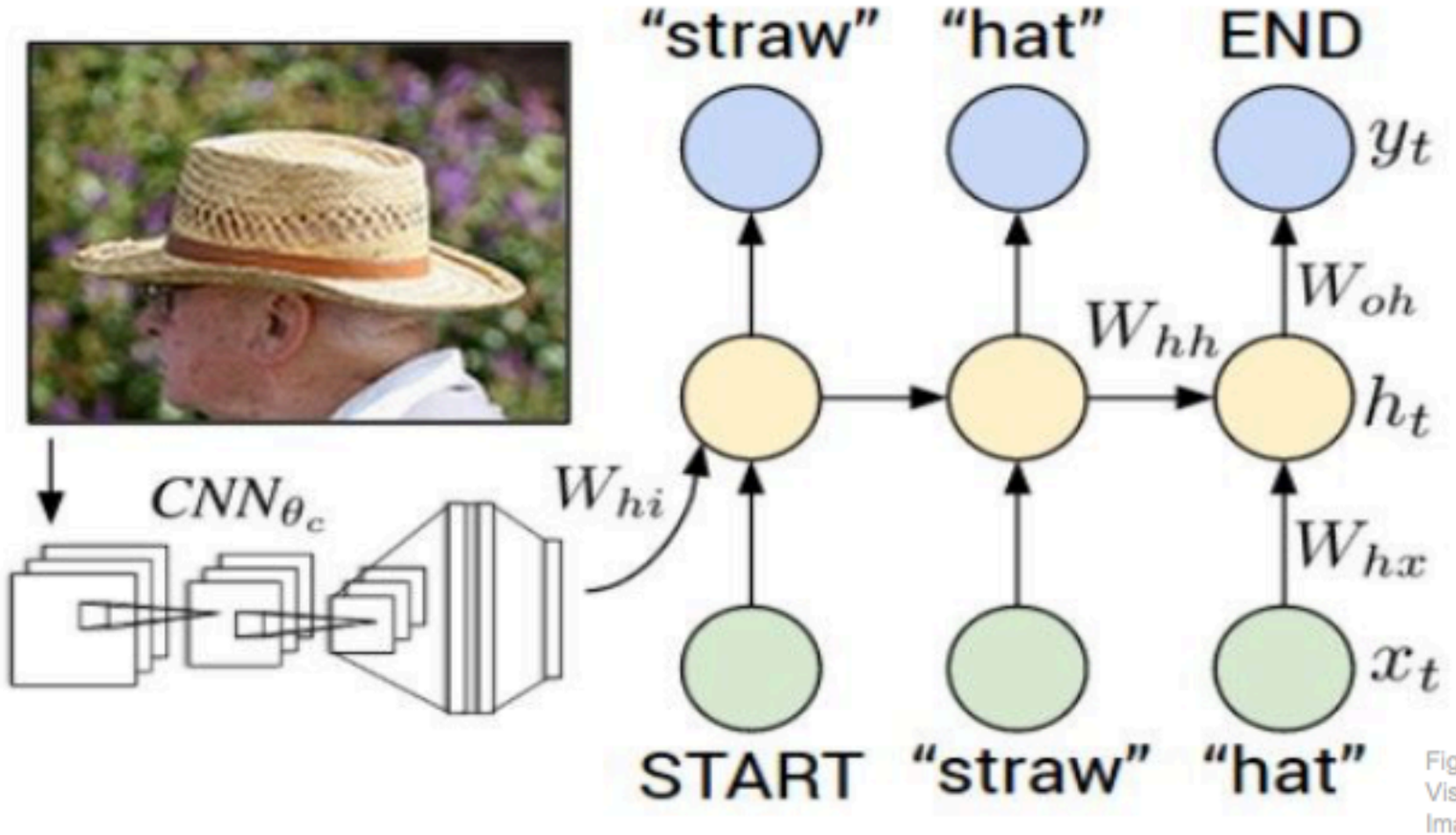


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure

Image Captioning Example

Image Captioning Example

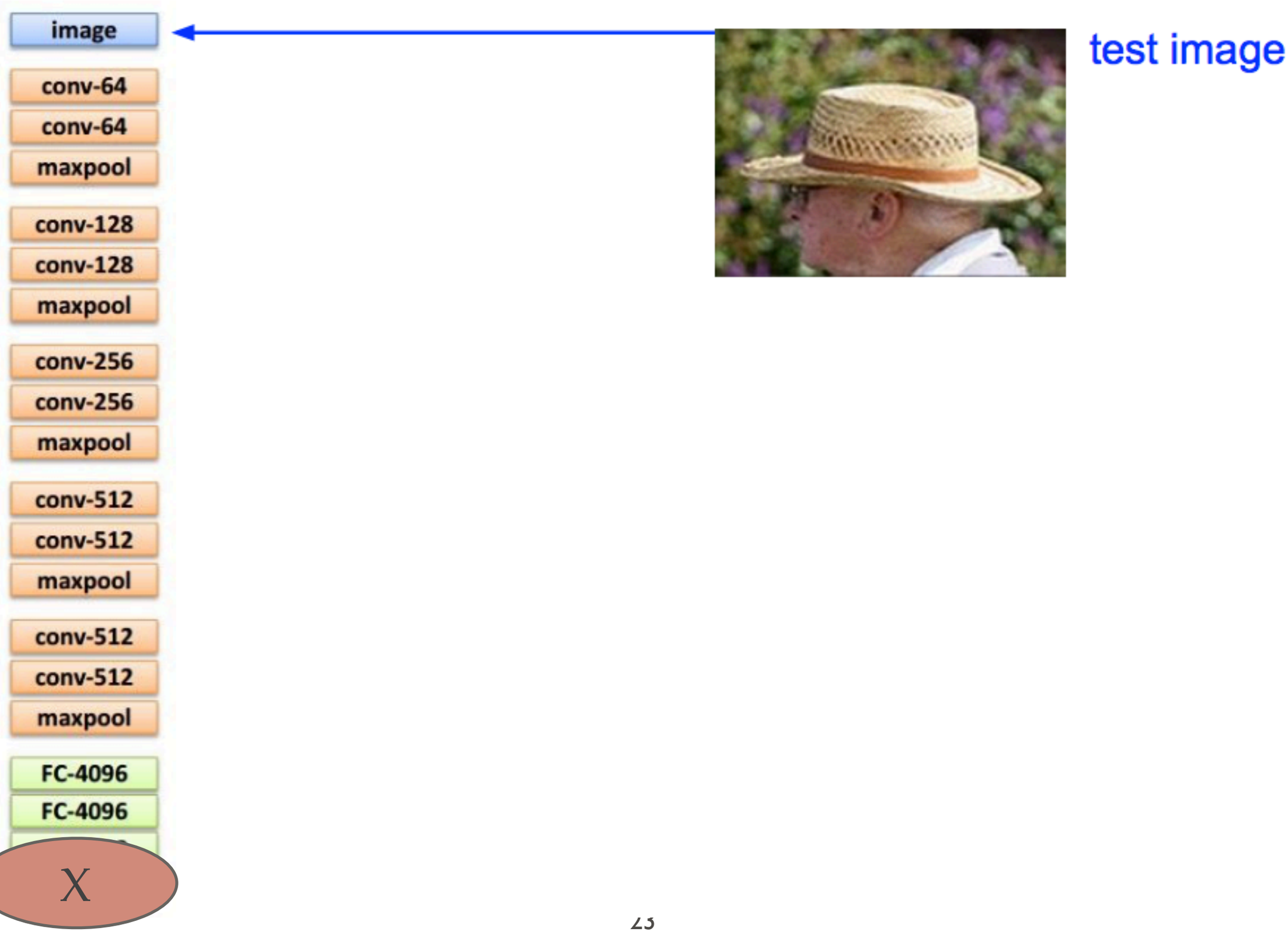
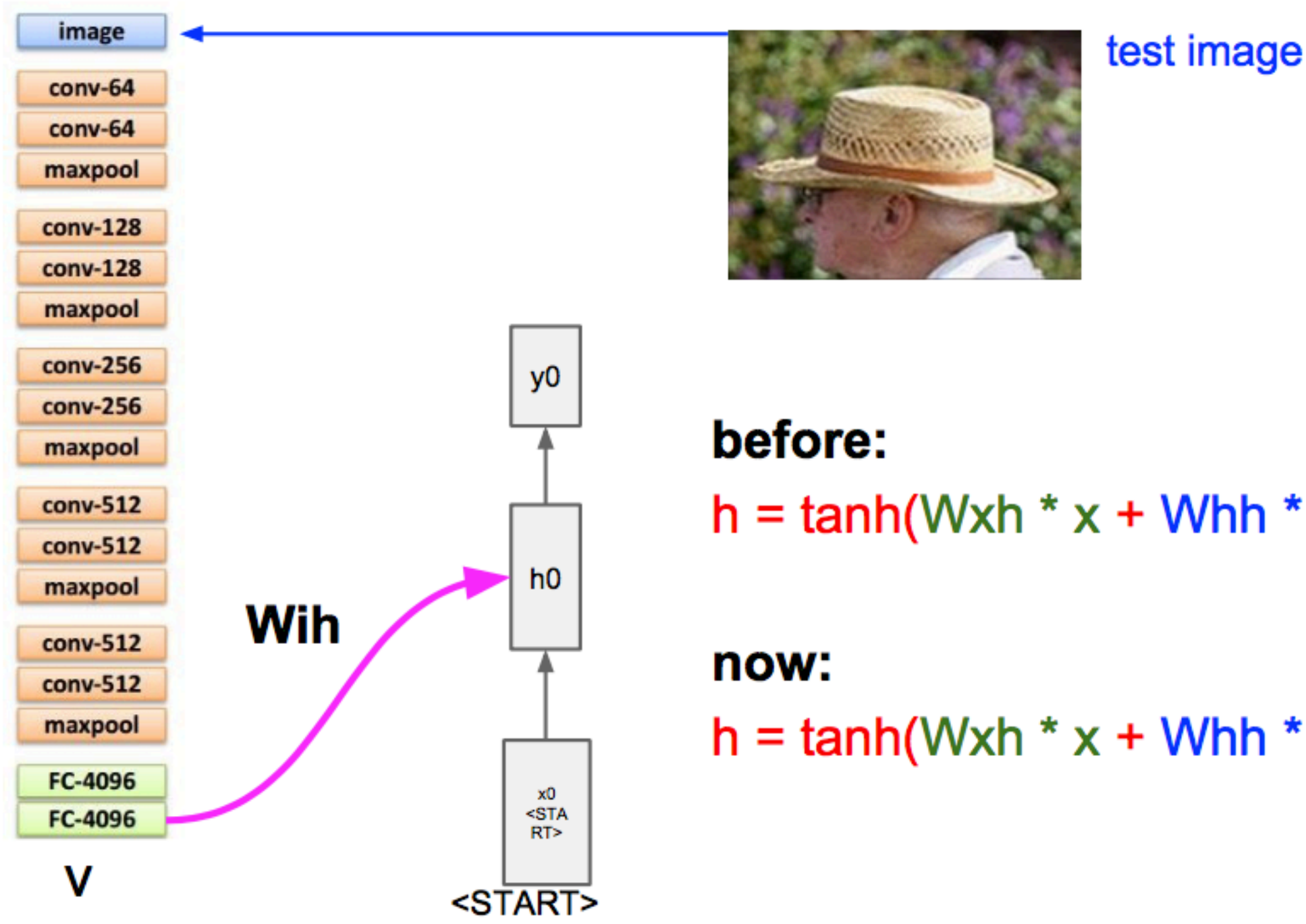


Image Captioning Example



before:

h = tanh(Wxh * x + Whh * h)

now:

h = tanh(Wxh * x + Whh * h + Wih * v)

Image Captioning Example

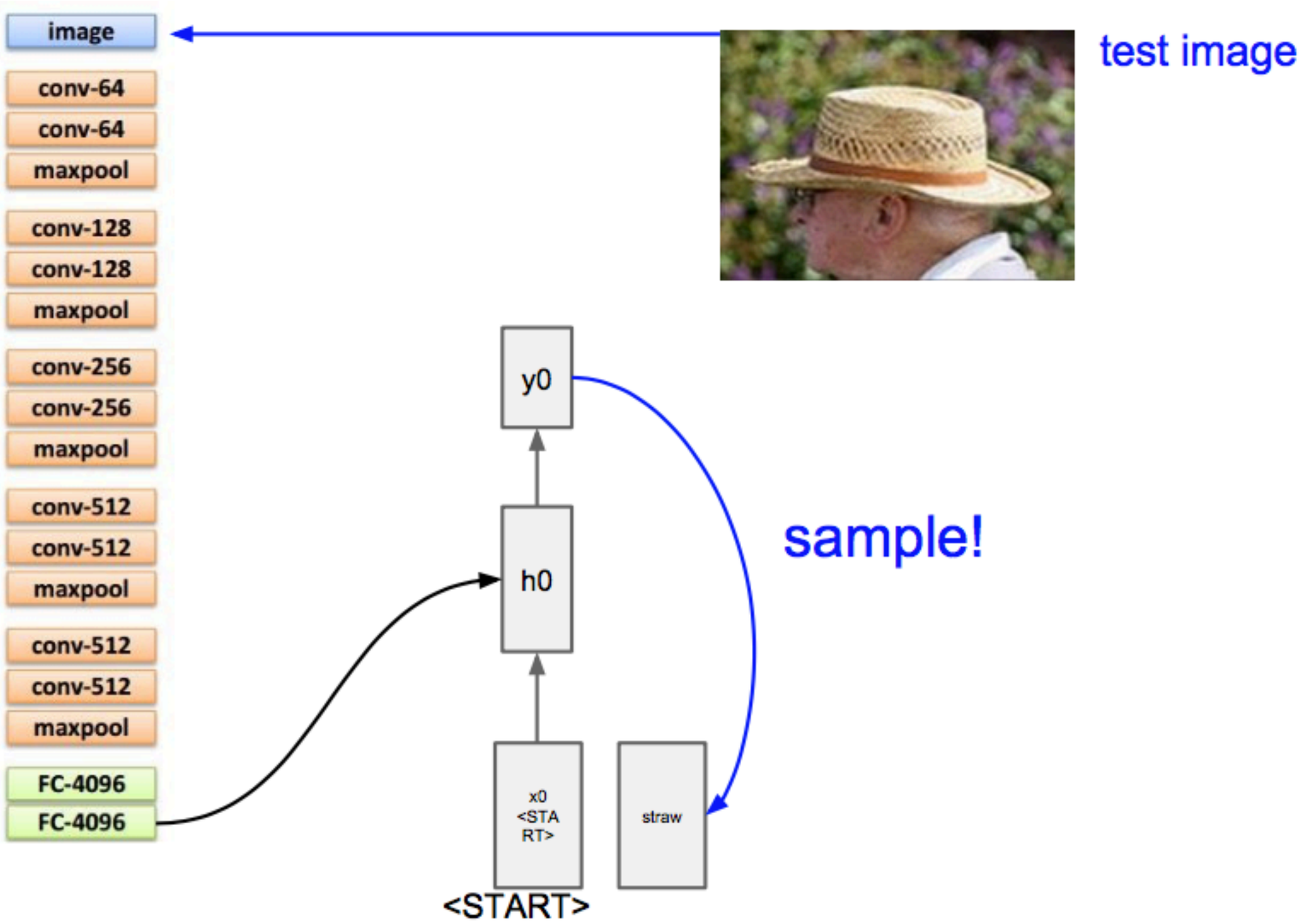


Image Captioning Dataset

Image Sentence Datasets

a man riding a bike on a dirt path through a forest.
bicyclist raises his fist as he rides on desert dirt trail.
this dirt bike rider is smiling and raising his fist in triumph.
a man riding a bicycle while pumping his fist in the air.
a mountain biker pumps his fist in celebration.

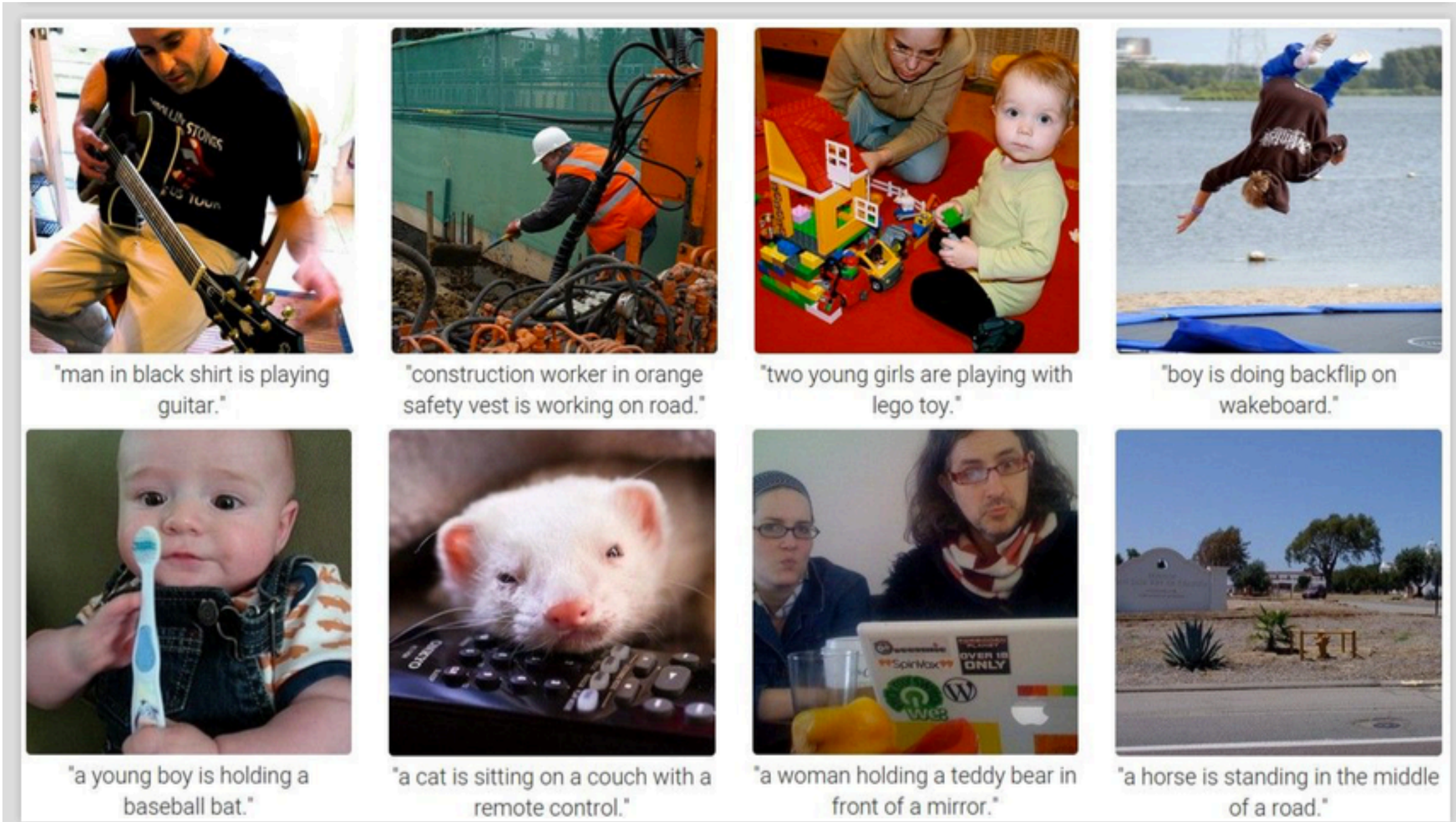
Microsoft COCO

[Tsung-Yi Lin et al. 2014]

mscoco.org

currently:
~120K images
~5 sentences each

Some Anecdotal Example



Attention Mechanism

- Attention allow for a more direct dependence between the state of the model at **different points in time**.
- For each hidden state h_t a "context vector" c_t is learned

$$e_t = a(h_t), \alpha_t = \frac{\exp(e_t)}{\sum_{k=1}^{T} \exp(e_k)}, c = \sum_{t=1}^{T} \alpha_t h_t$$

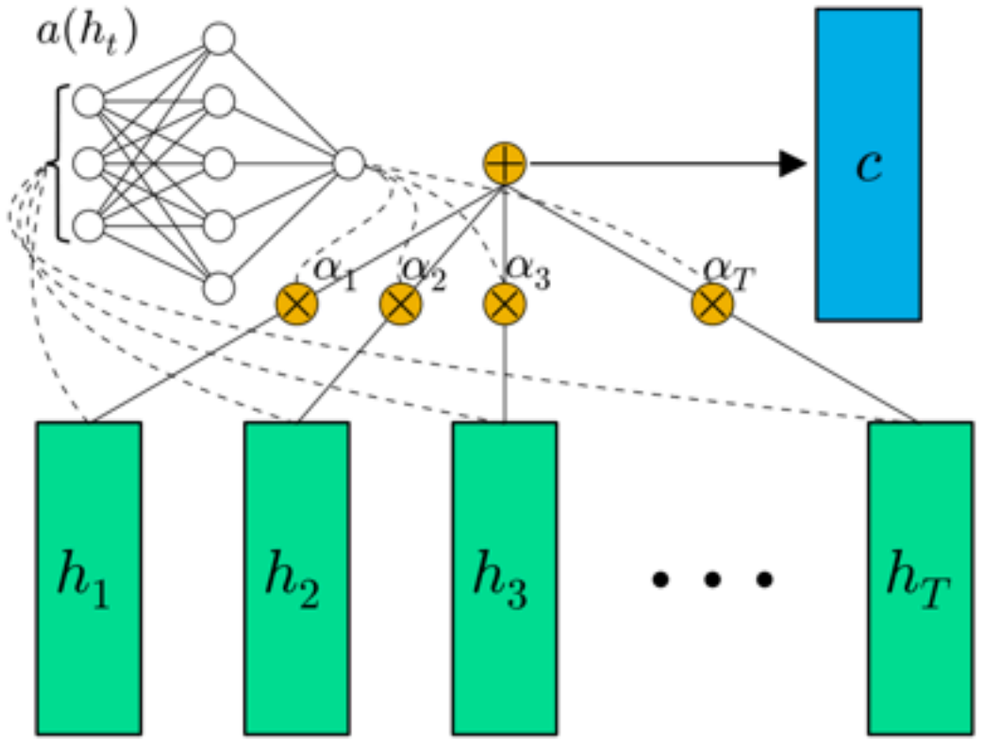
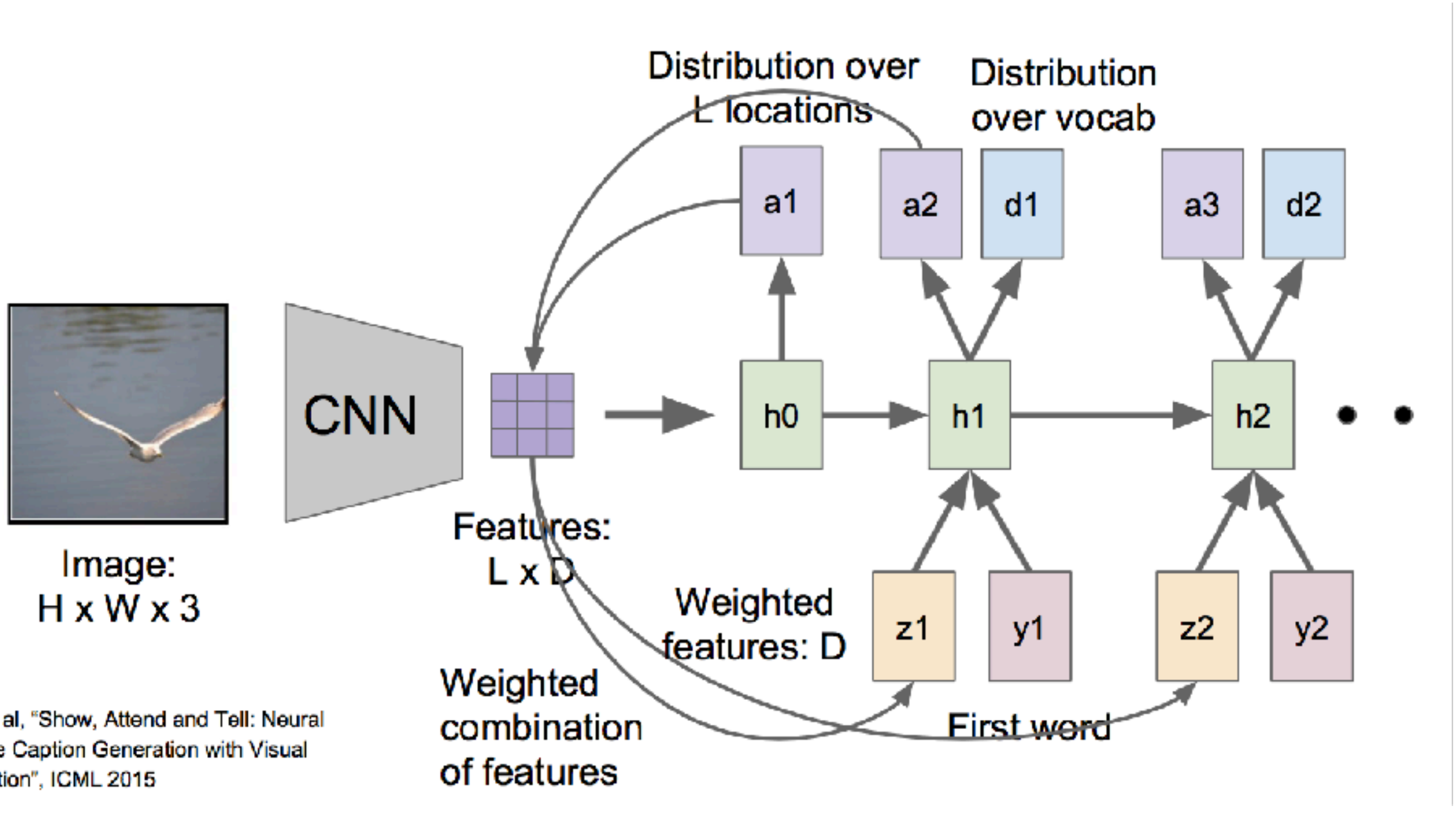


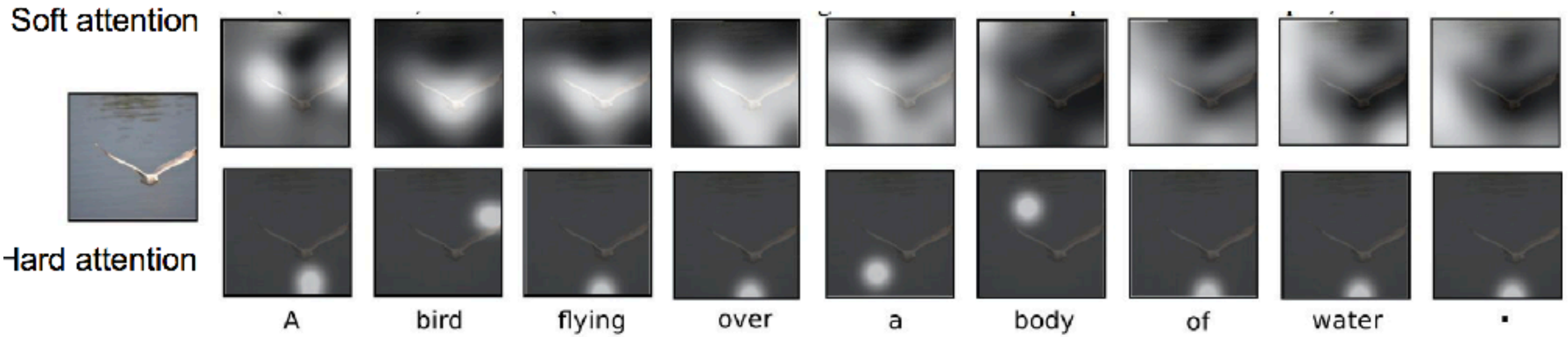
Figure 1: Schematic of our proposed "feed-forward" attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α.

Image Captioning with Attention



Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention Example



Sources

- Slides are taken from Stanford CS231n course http://cs231n.stanford.edu/syllabus.html (lecture 10)
- Keras IMDB sentiment classification example https://richliao.github.io/supervised/classification