Recurrent Neural Networks

Vishnu Lokhande

Department of Computer Science and Engineering
University at Buffalo, SUNY
vishnulo@buffalo.edu

March 26, 2025

Recurrent Neural Networks[1]

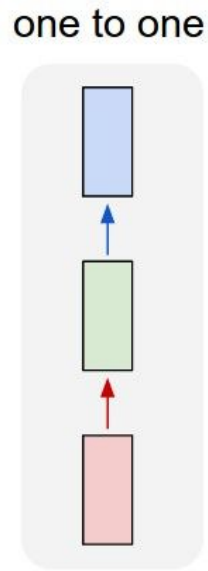
Recurrent neural networks are extensions of feed forward neural networks such that

- it can process sequential data efficiently.
- weights of the neural networks between different time are shared:
  - In CNN, weights between different regions are shared.

[1] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

“Vanilla” Neural Network

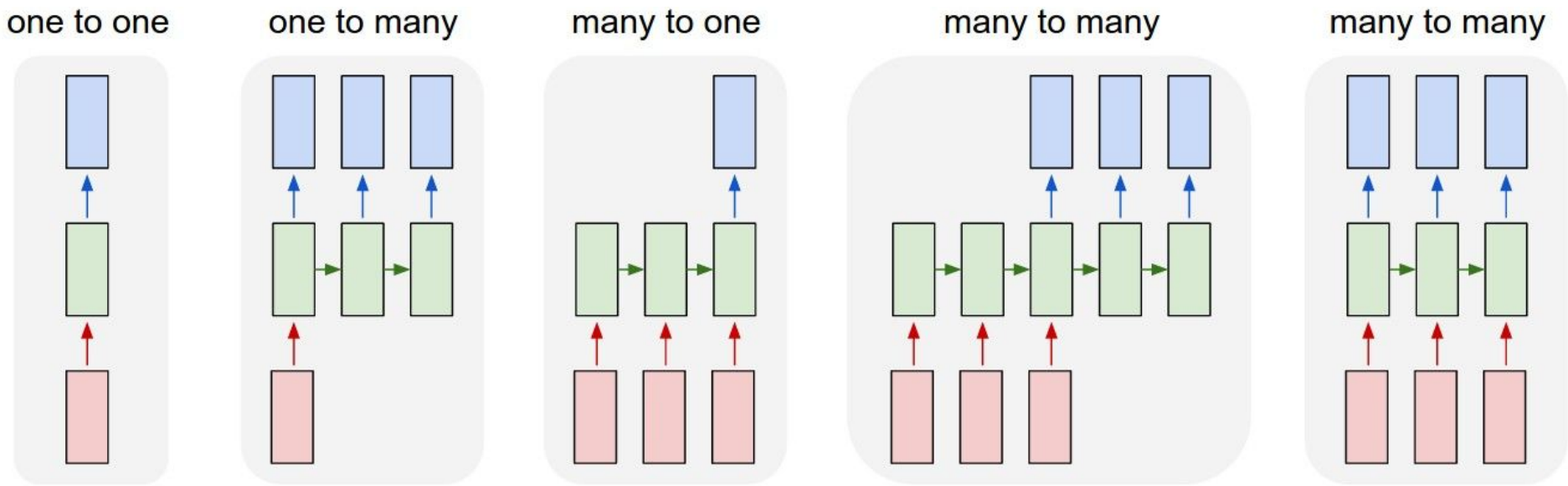
pink: input; green: hidden; blue: output



Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences

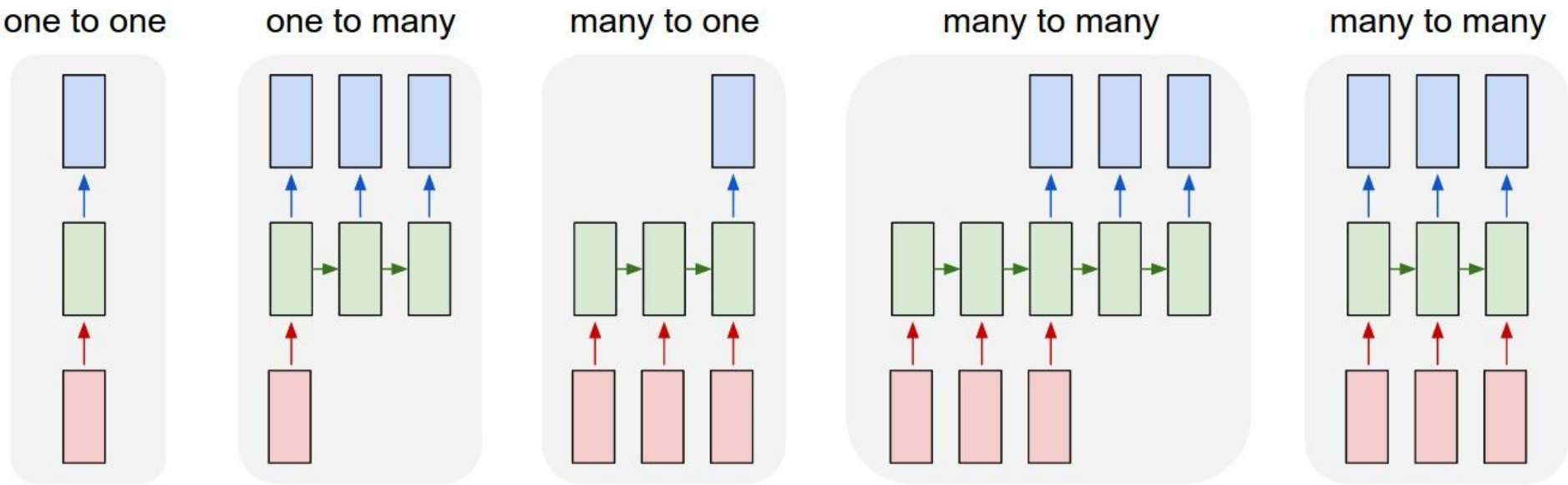
pink: input; green: hidden; blue: output



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

Recurrent Neural Networks: Process Sequences

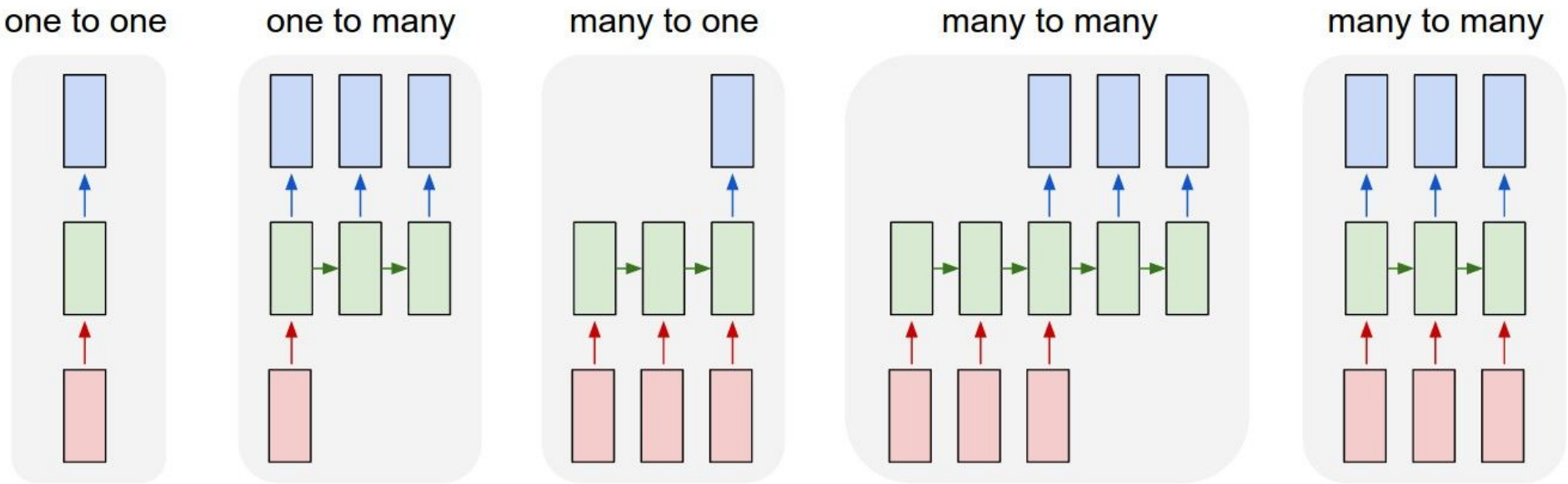
pink: input; green: hidden; blue: output



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

Recurrent Neural Networks: Process Sequences

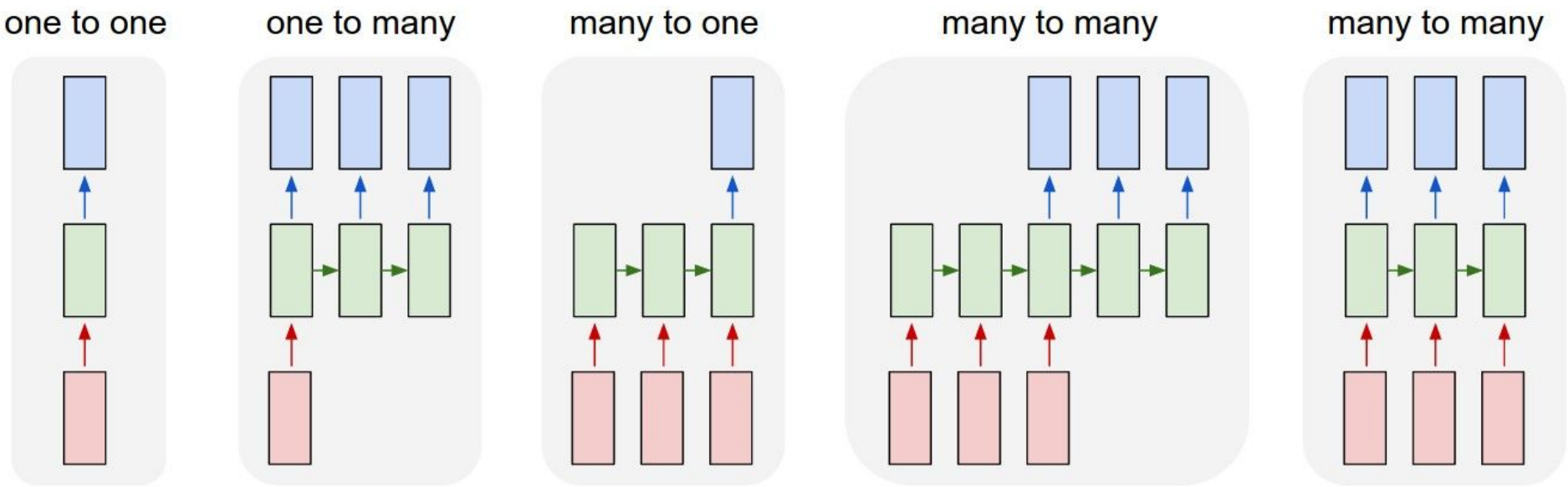
pink: input; green: hidden; blue: output



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

Recurrent Neural Networks: Process Sequences

pink: input; green: hidden; blue: output



e.g. **Video classification on frame level**

Sequential Processing of Non-Sequence Dat

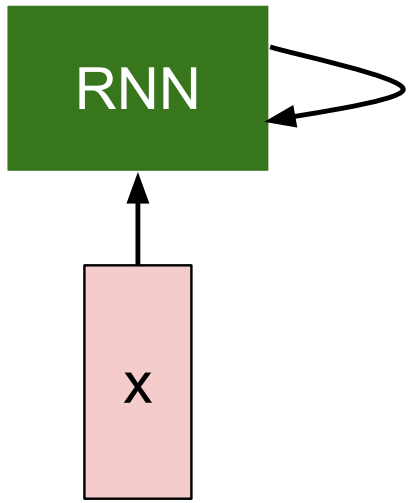
Making non-sequence data sequential

Classify images by taking a series of “glimpses”

Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

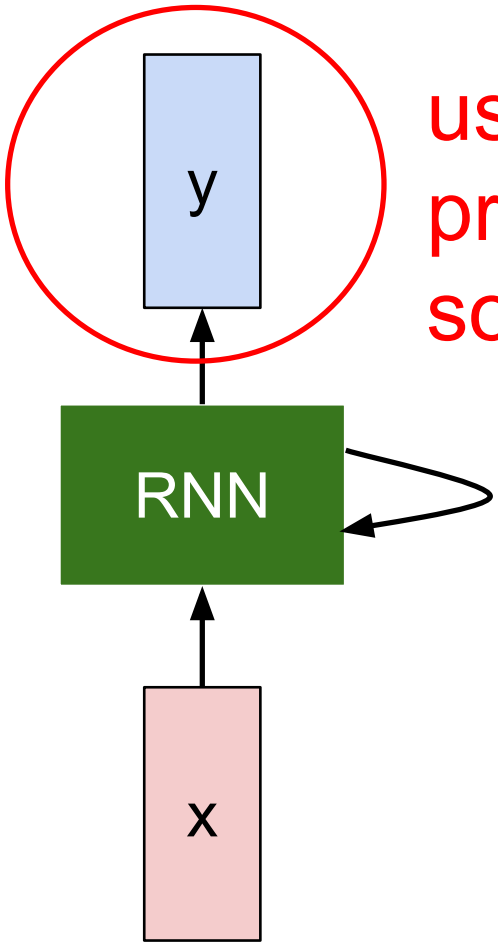
Recurrent Neural Networks

- Inputs and outputs are usually indexed by time.
- The recurrent unit generates states $\mathbf{h}_t$ for each time.



Recurrent Neural Networks

- Inputs and outputs are usually indexed by time.
- The recurrent unit generates states $\mathbf{h}_t$ for each time.



usually want to predict a vector at some time steps

Recurrent Neural Networks

Process a sequence of vectors x by applying a recurrence formula at every time step:

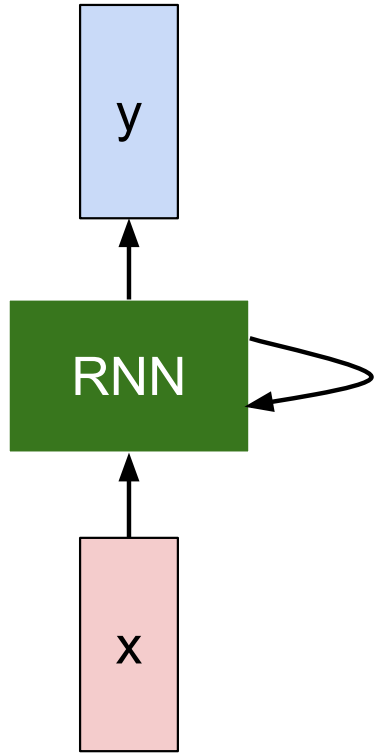
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



Recurrent Neural Networks

Process a sequence of vectors x by applying a recurrence formula at every time step:

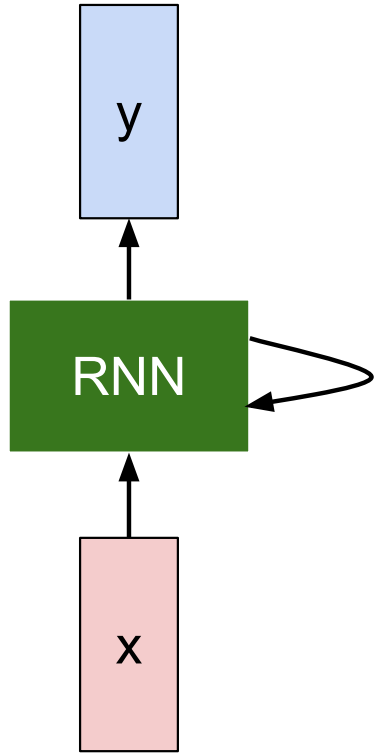
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

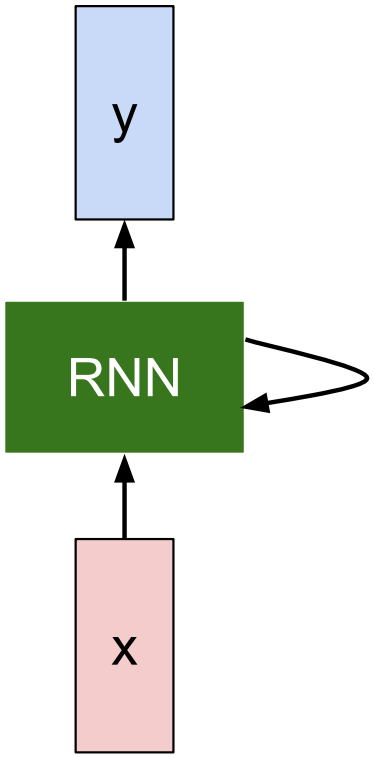
input vector at some time step



Notice: the same function and the same set of parameters are used at every time step. $\Rightarrow$ Shared!

Vanilla Recurrent Neural Network

The state consists of a single "hidden" vector h:



$$\mathbf{h}_t = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

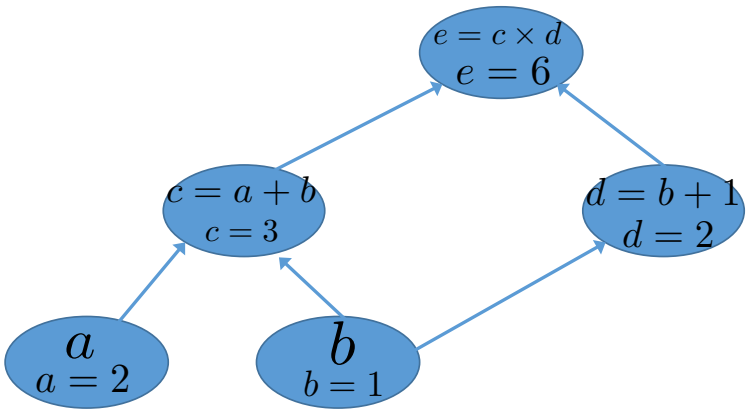
$$\Downarrow \textit{e.g.}$$

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right)$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\,\mathbf{h}_t$$

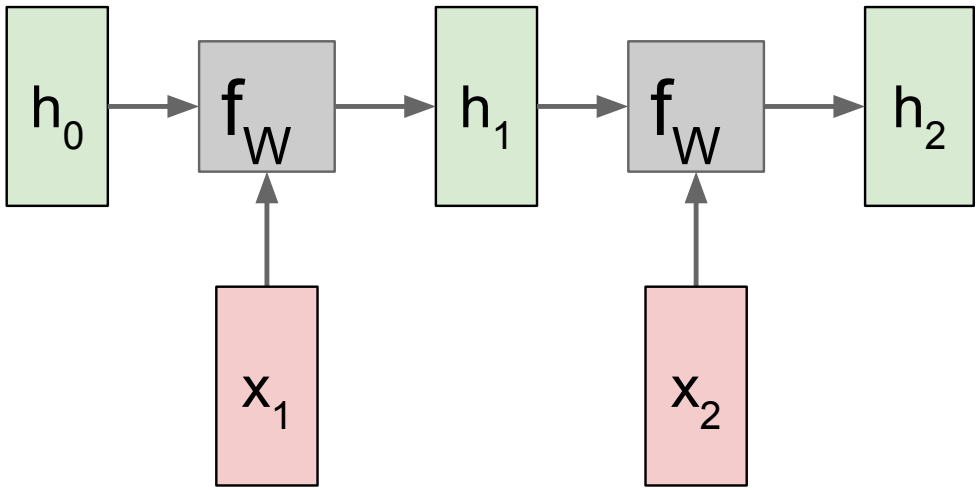
Recap: Computational Graph

- Each node is either
  - a variable: scalar, vector, matrix, tensor, or other type
  - or an operation
    - ★ simple function of one or more variables
    - ★ functions more complex than operations are obtained by composing operations
  - if variable y is computed by applying operation to variable x then draw directed edge from x to y.



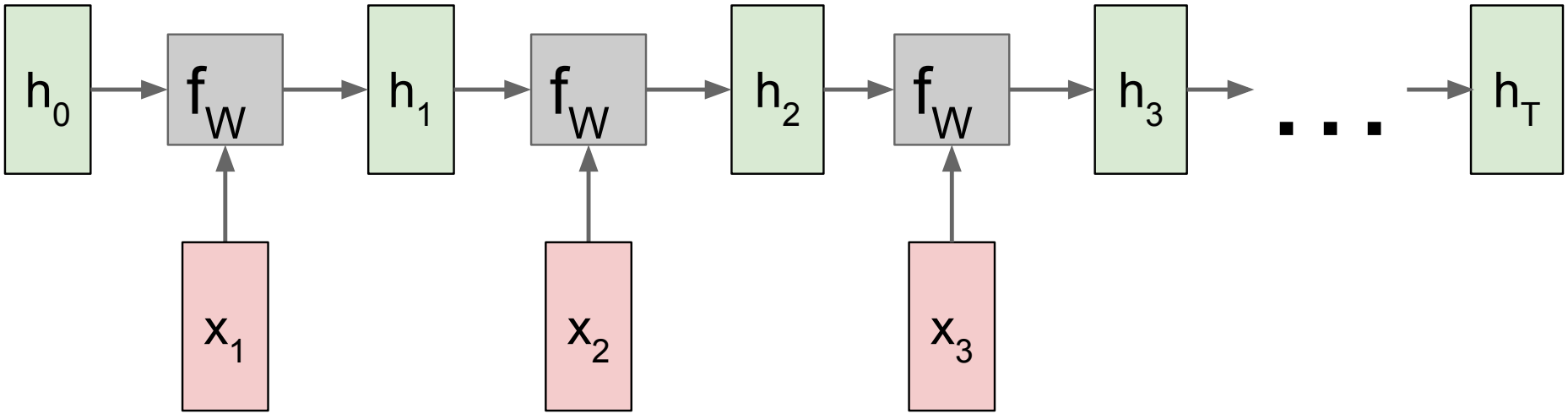
Unrolling RNN: Computational Graph

First two time steps:



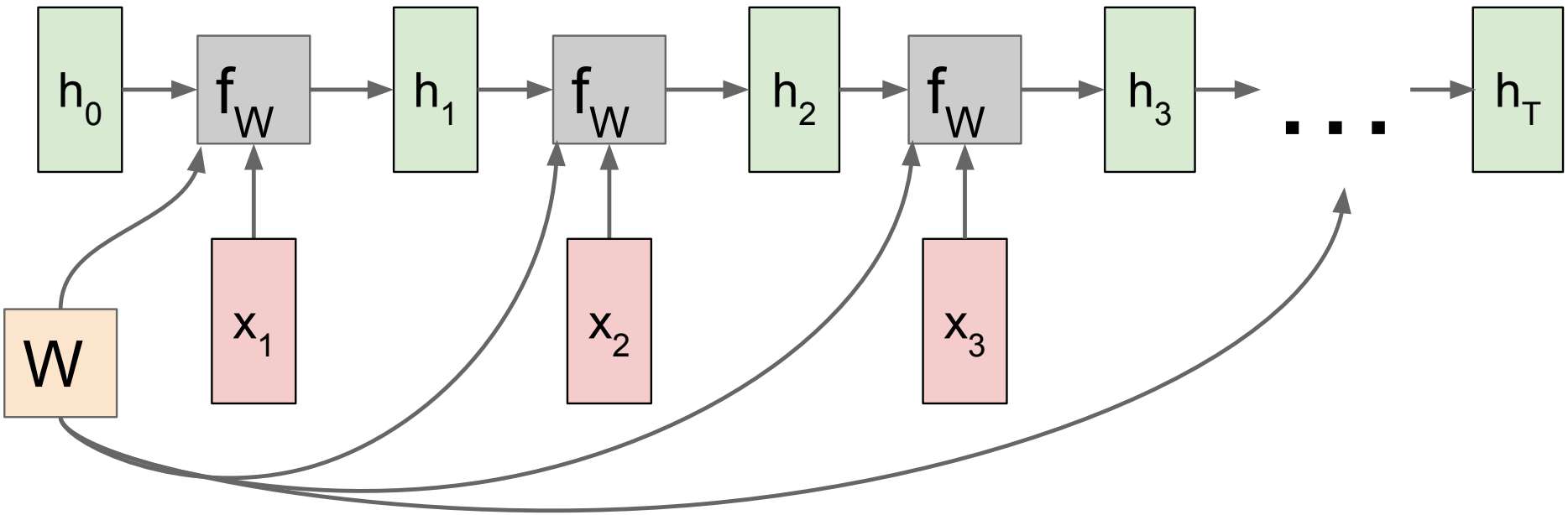
Unrolling RNN: Computational Graph

T time steps:



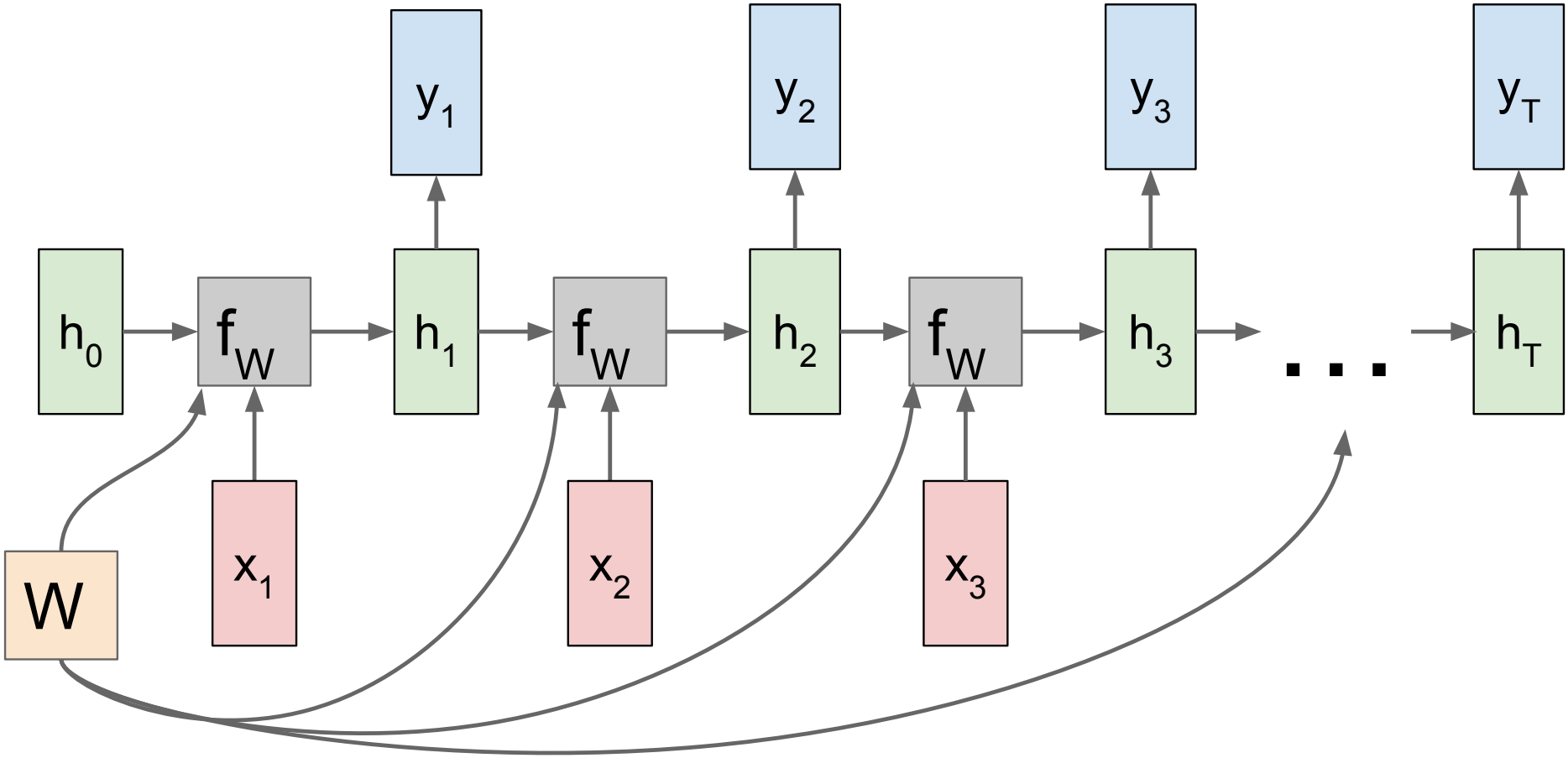
Unrolling RNN: Computational Graph

Re-use the same weight matrix at every time-step



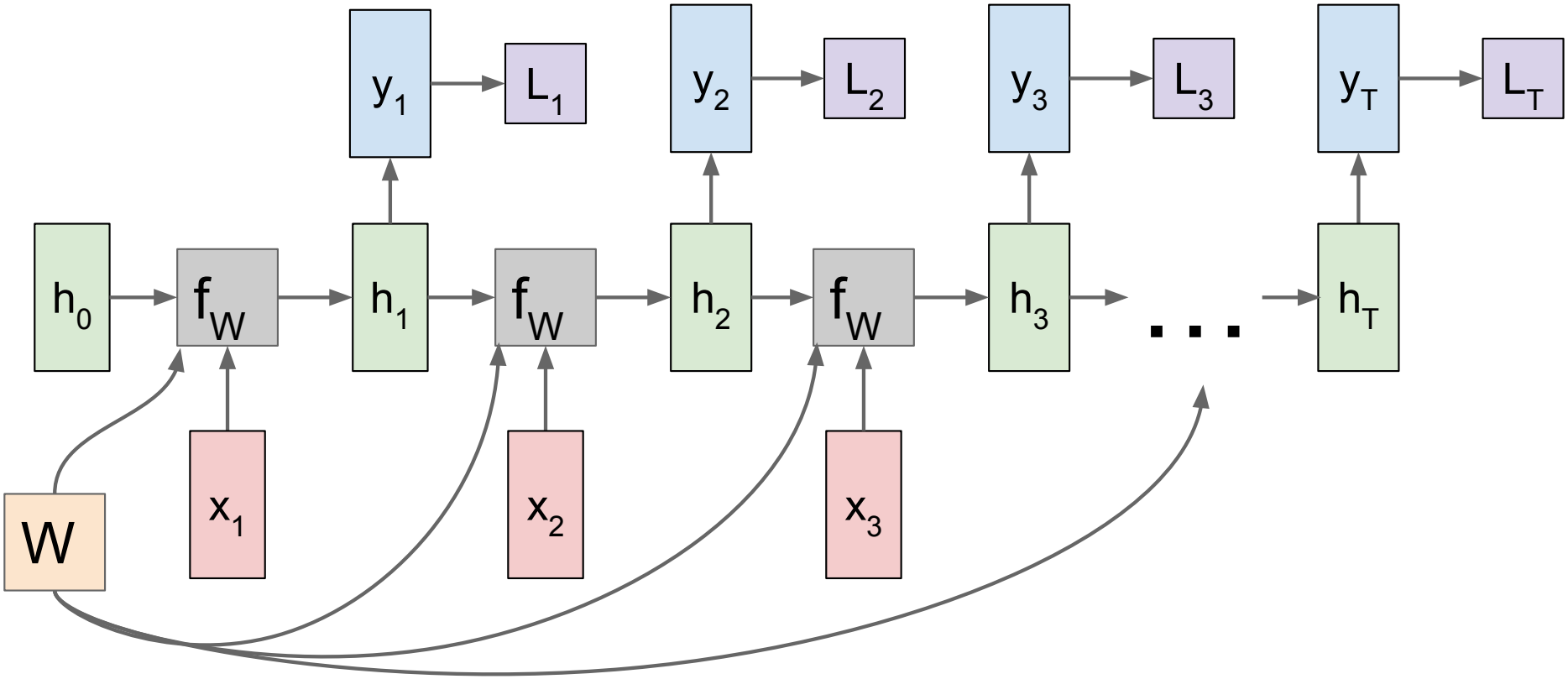
RNN: Computational Graph: Many to Many

Add outputs:



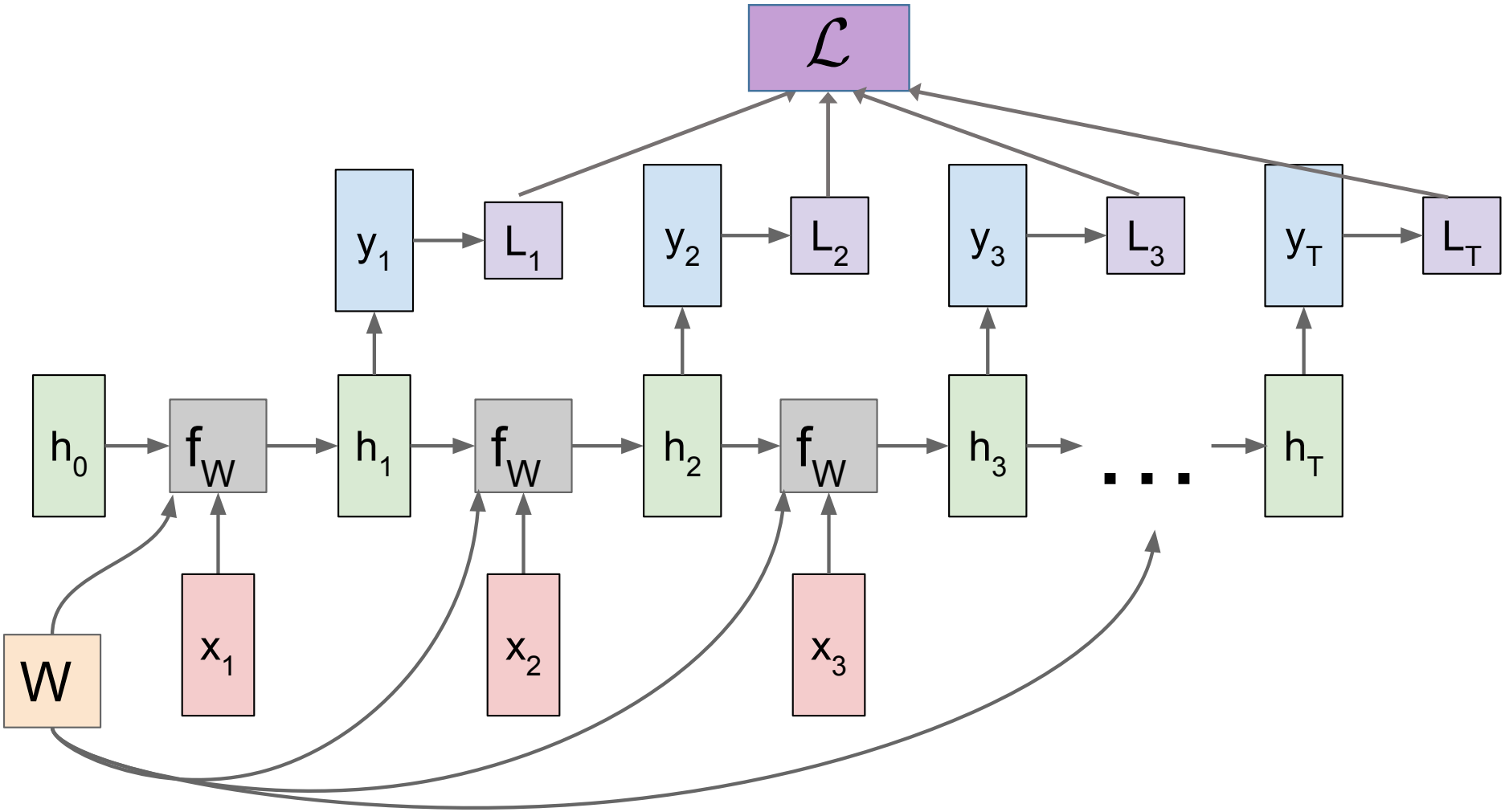
RNN: Computational Graph: Many to Many

Add loss for different time steps:



RNN: Computational Graph: Many to Many

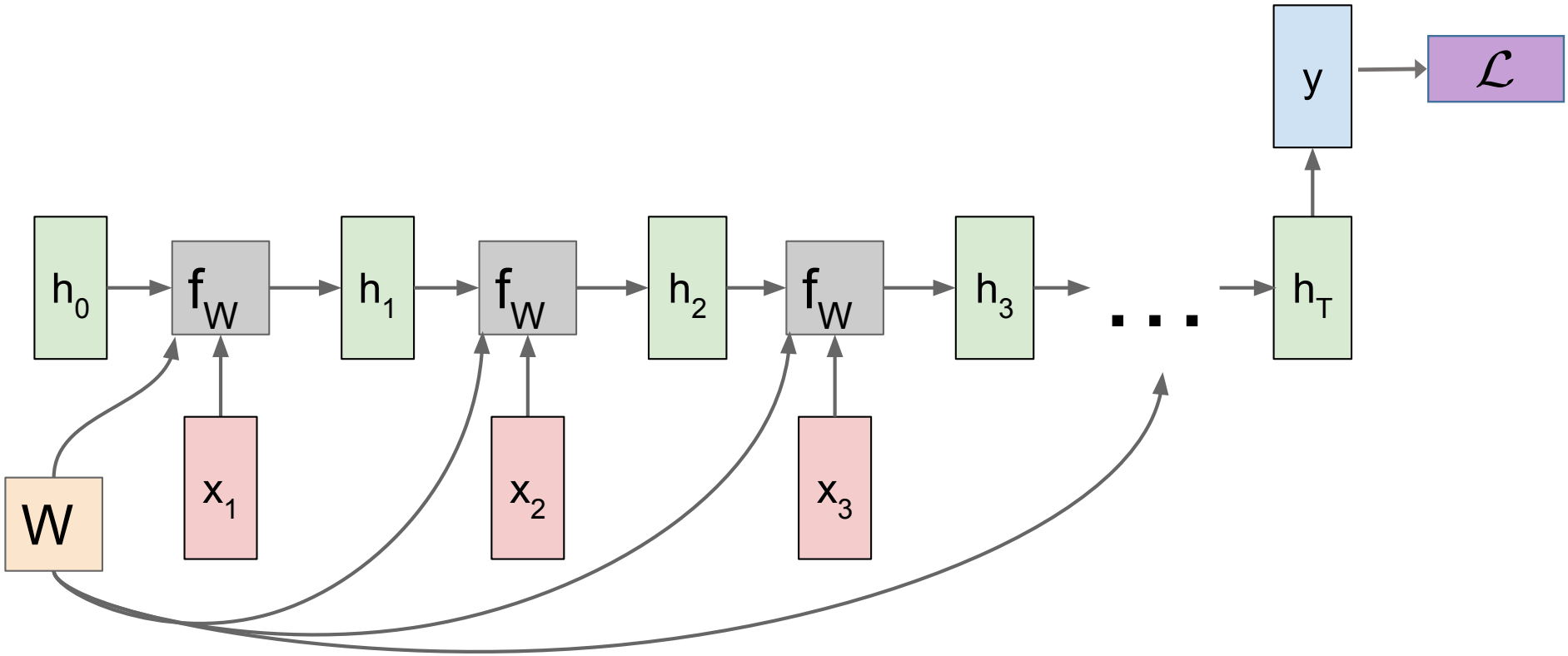
Total loss:



RNN: Computational Graph: Many to One

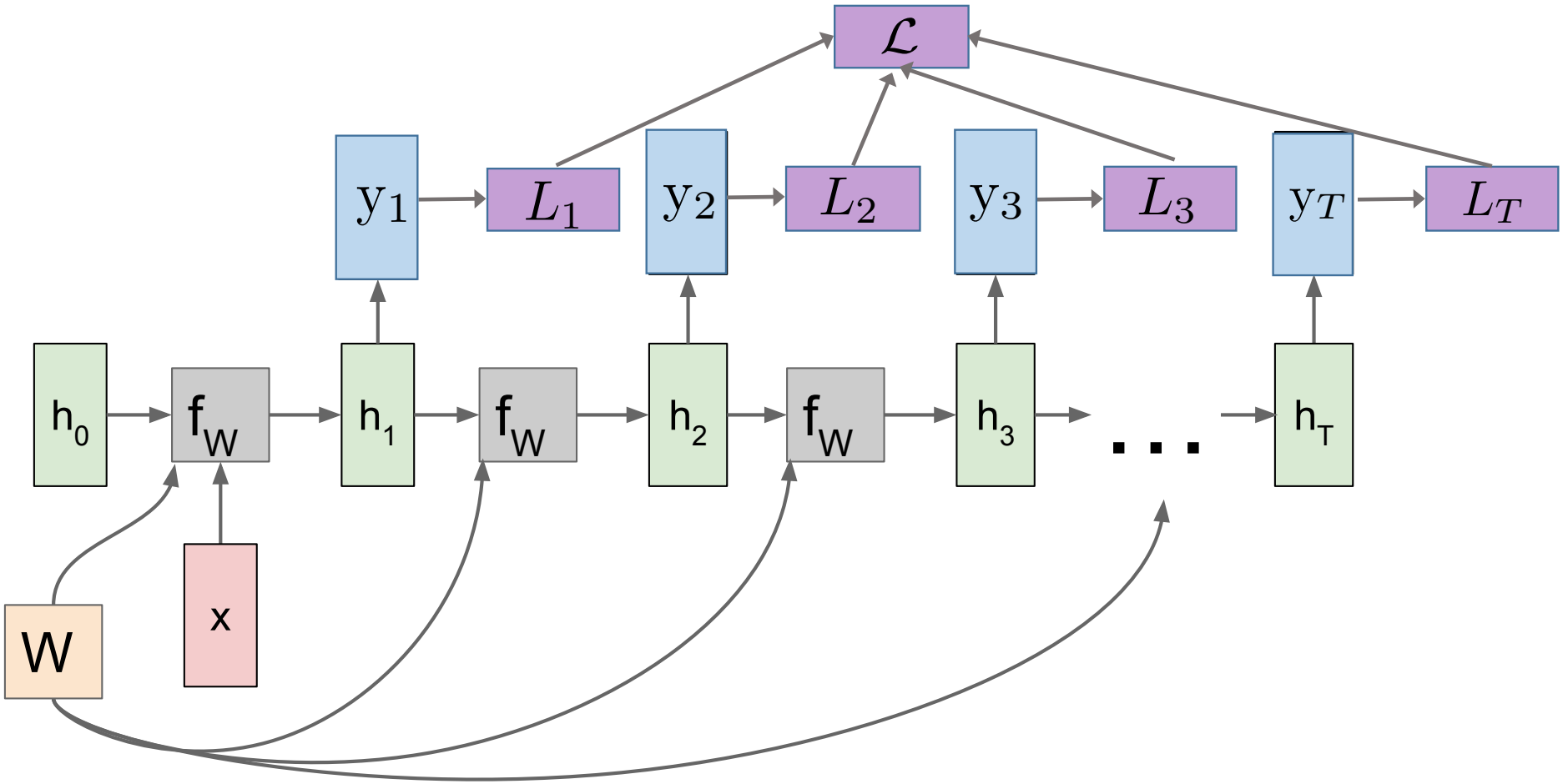
RNN: Computational Graph: Many to One

One output:



RNN: Computational Graph: One to Many

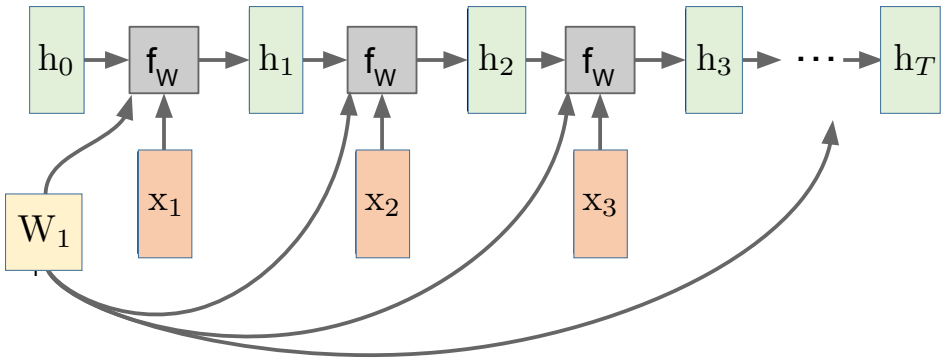
T outputs:



Sequence to Sequence: Many-to-one + one-to-many

- Encoder-decoder architecture, used for machine translation.

Many to one: Encode input sequence in a single vector

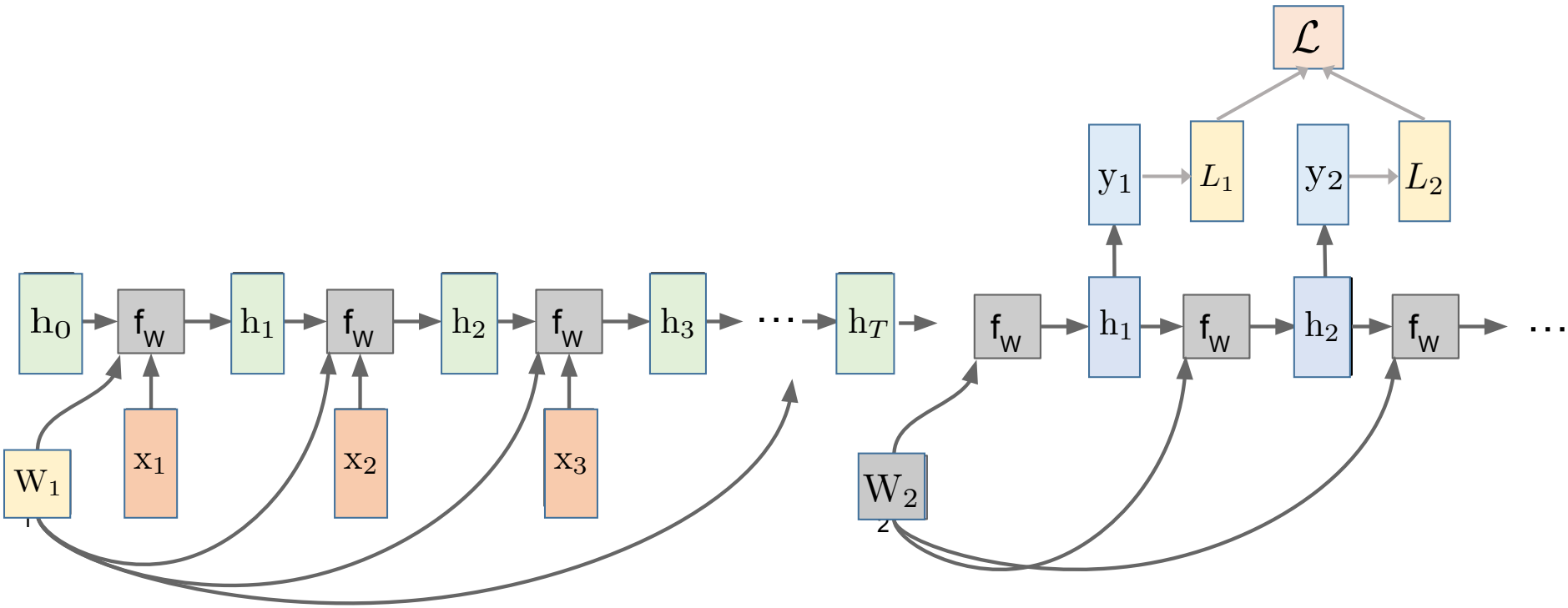


Sequence to Sequence: Many-to-one + one-to-many

- Encoder-decoder architecture, used for machine translation.

Many to one: Encode input sequence in a single vector

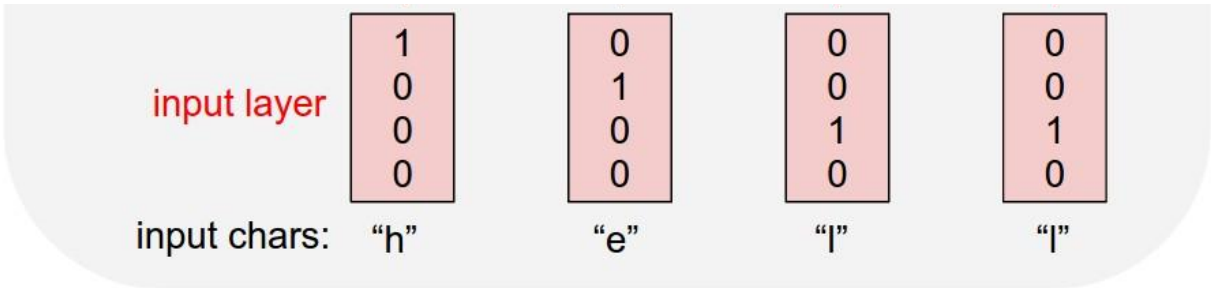
One to Many: Produce output sequence from single input vector



Example: Character-level Language Model

Many to many RNN model

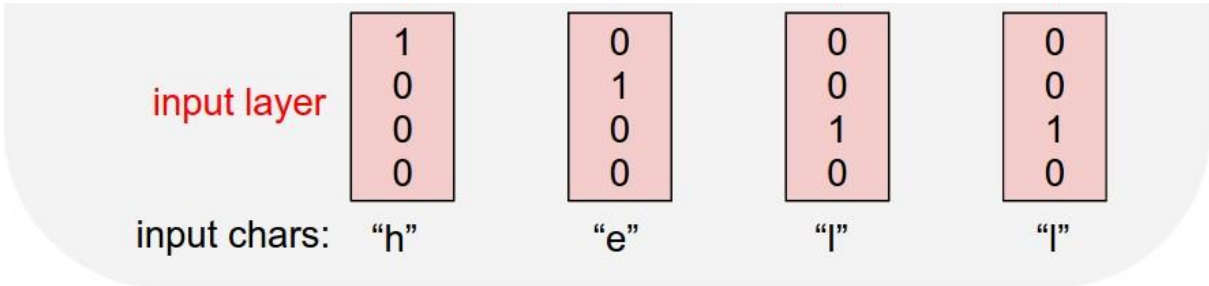
- Vocabulary:
  [h, e, l, o]
- Example training sequence:
  “hello”
  On training:



Example: Character-level Language Model

Many to many RNN model

- Vocabulary:
  [h, e, l, o]
- Example training sequence:
  “hello”
  On training:



Typically use word-embedding vector in practice.

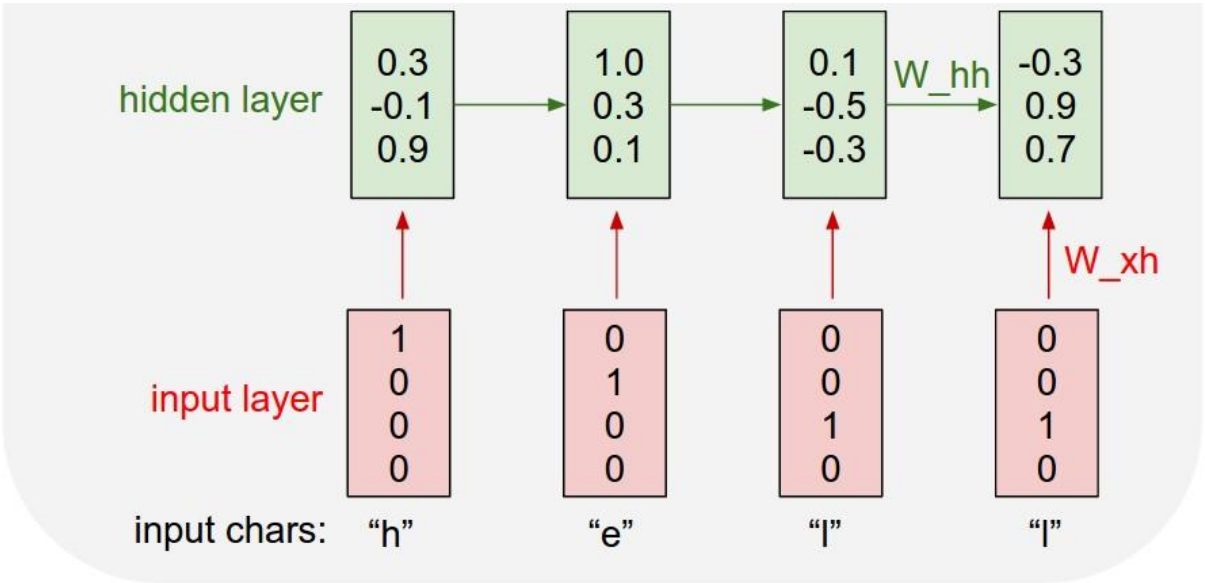
Example: Character-level Language Model

Many to many RNN model

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right)$$

- Vocabulary: [h, e, l, o]
- Example training sequence: “hello”

On training:

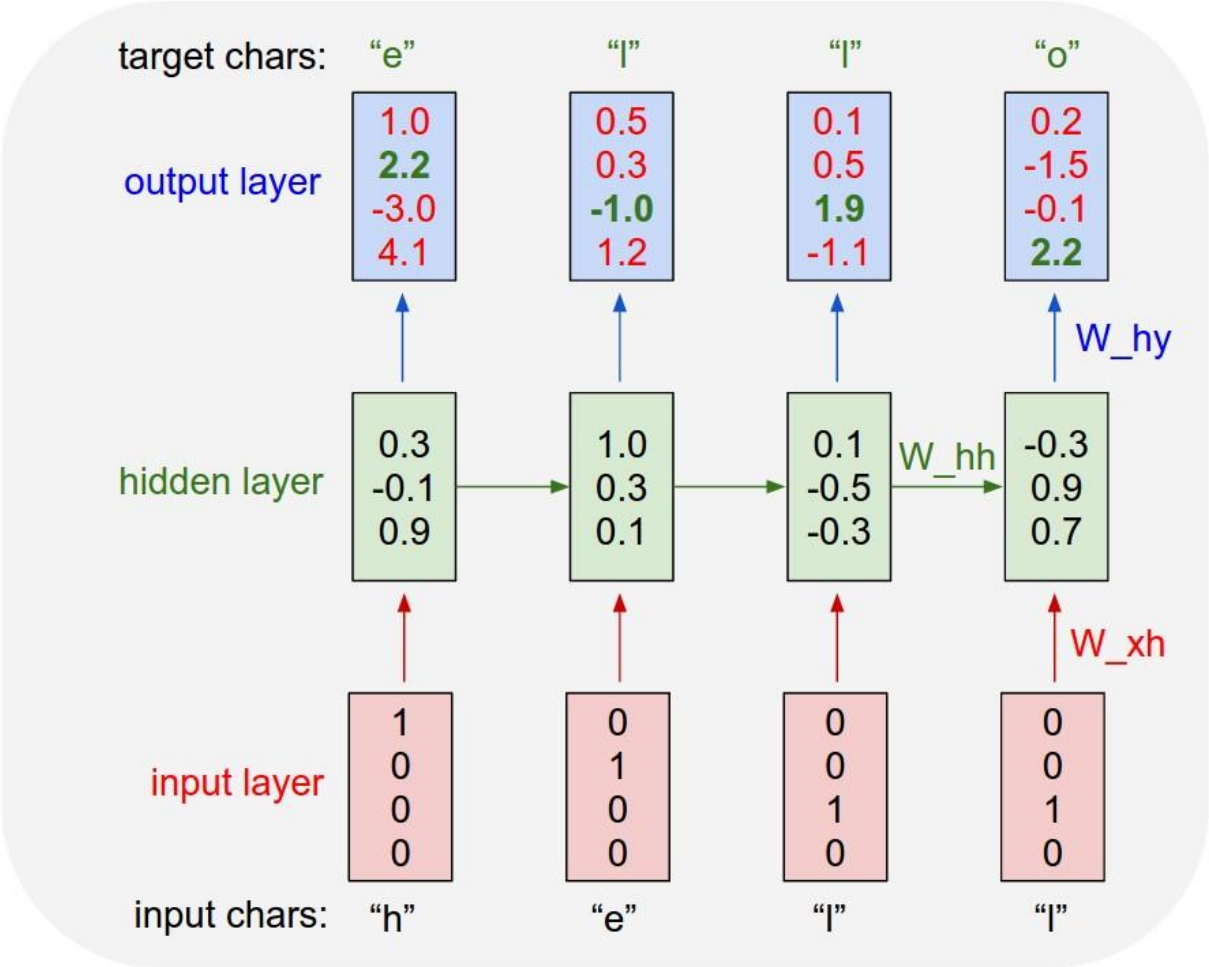


Example: Character-level Language Model

Many to many RNN model

- Vocabulary: [h, e, l, o]
- Example training sequence: “hello”

On training:

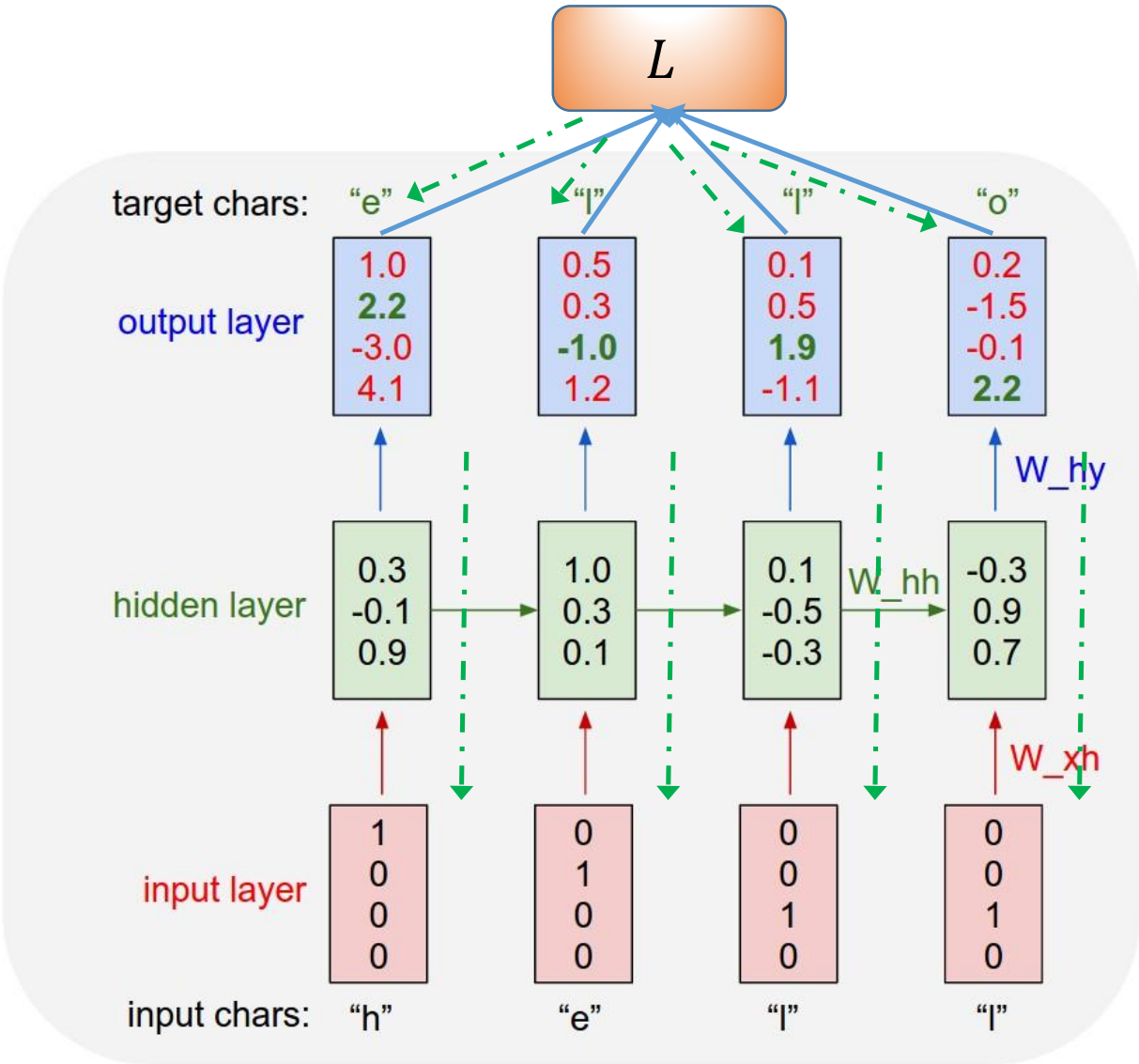


Example: Character-level Language Model

Many to many RNN model

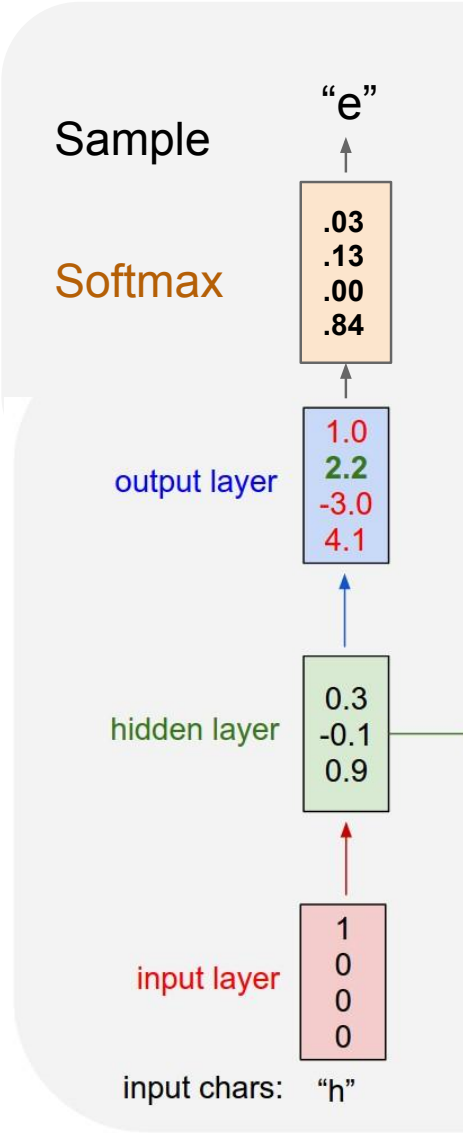
- Vocabulary: [h, e, l, o]
- Example training sequence: “hello”

On training:



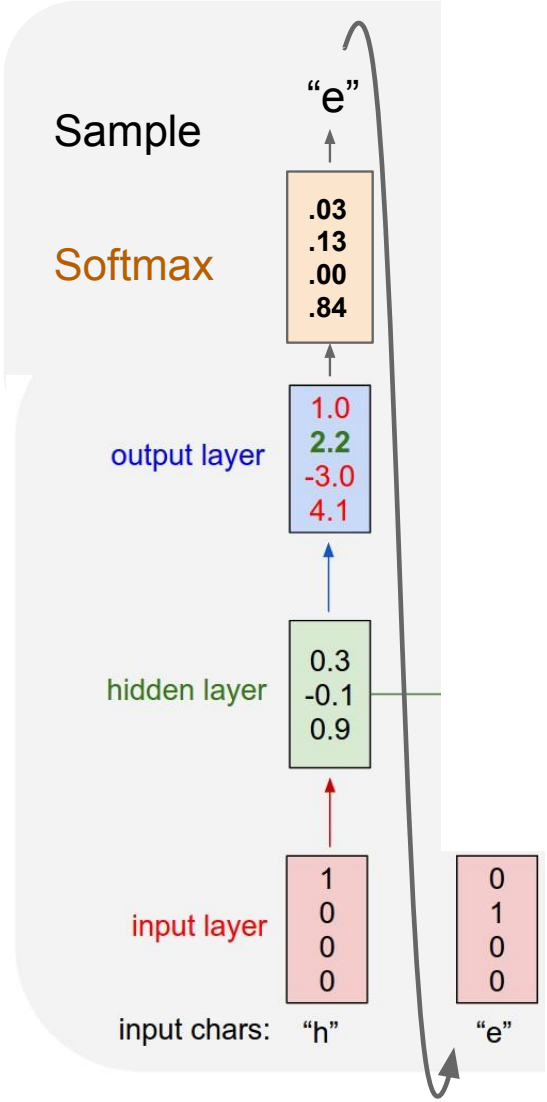
Example: Character-level Language Model

- Vocabulary: [h, e, l, o]
- At test-time sample characters one at a time, feed back to model



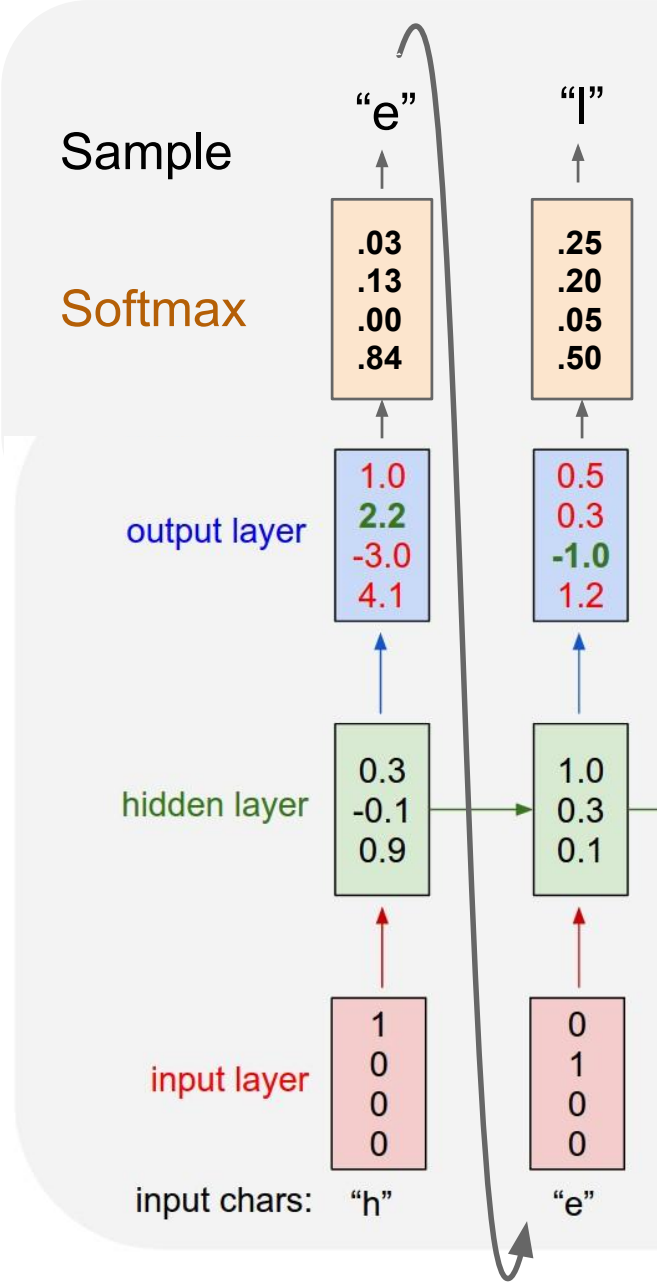
Example: Character-level Language Model

- Vocabulary: [h, e, l, o]
- At test-time sample characters one at a time, feed back to model



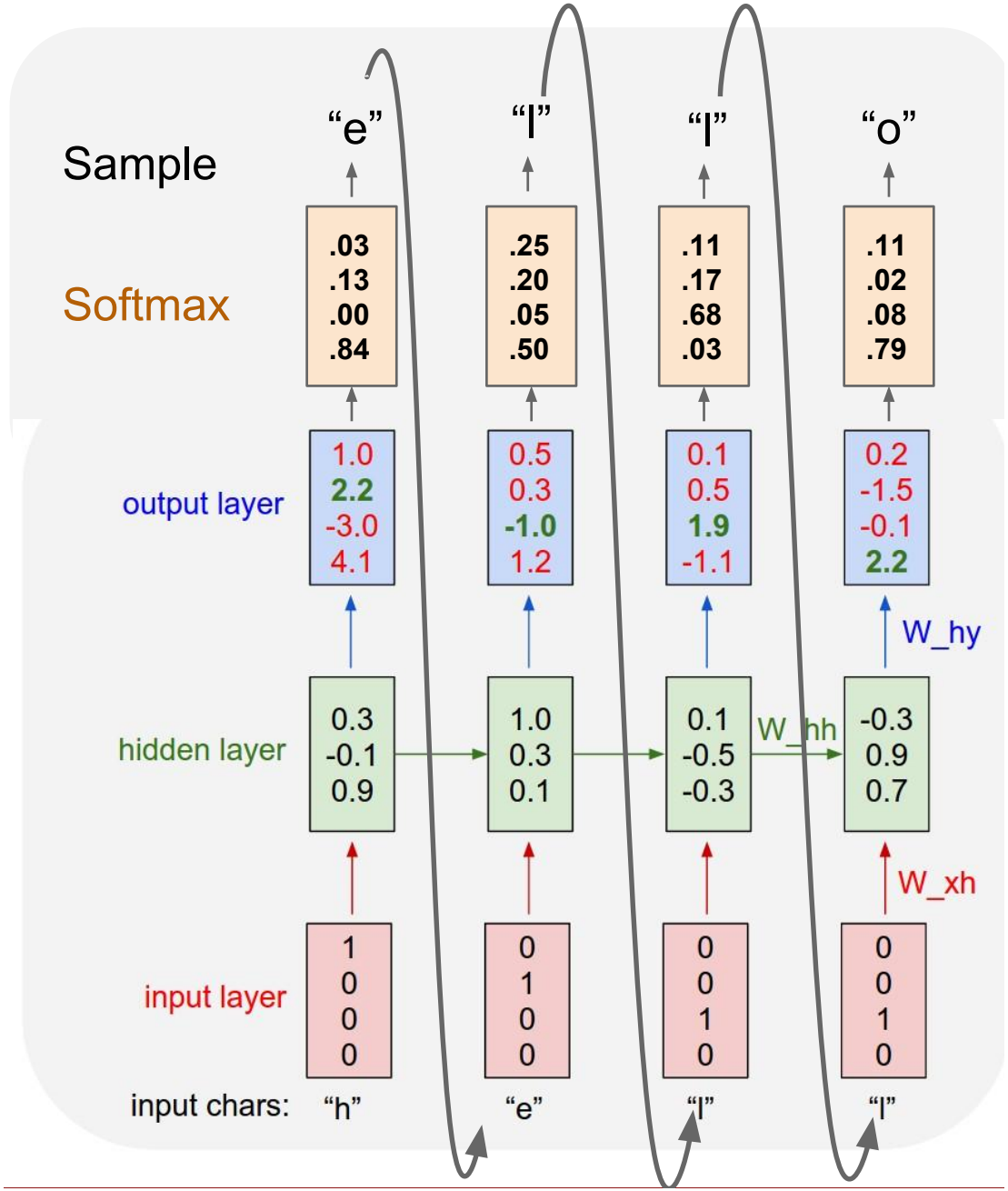
Example: Character-level Language Model

- Vocabulary: [h, e, l, o]
- At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model

- Vocabulary: [h, e, l, o]
- At test-time sample characters one at a time, feed back to model



Character-level Language Model: Trained on Shakespeare's "The Sonnets"

random generate

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```


train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```


train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```


train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Character-level Language Model: Trained on Shakespeare's "The Sonnets"

random generate

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```


train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```


train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```


train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Q: how can we generate *space* and *comma*?

Character-level Language Model: Trained on Shakespeare's "The Sonnets"

random generate

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Q: How can we generate *space* and *comma*?

A: Consider them as auxiliary characters in the vocabulary.

Character-level Language Model: Trained on Shakespeare's "The Sonnets"

random generate

at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```


train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```


train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```


train more

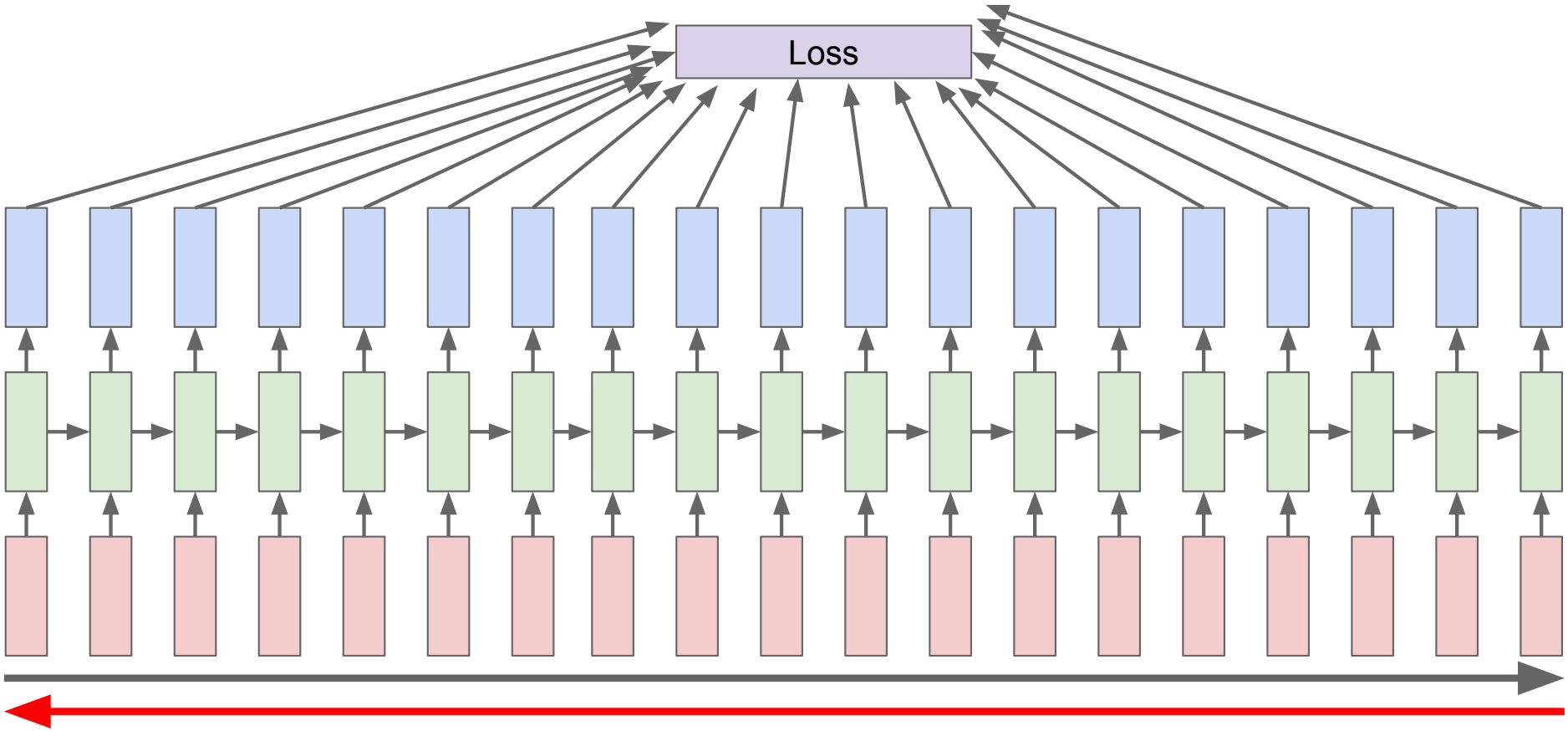
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Learning in RNN

How to do BP?

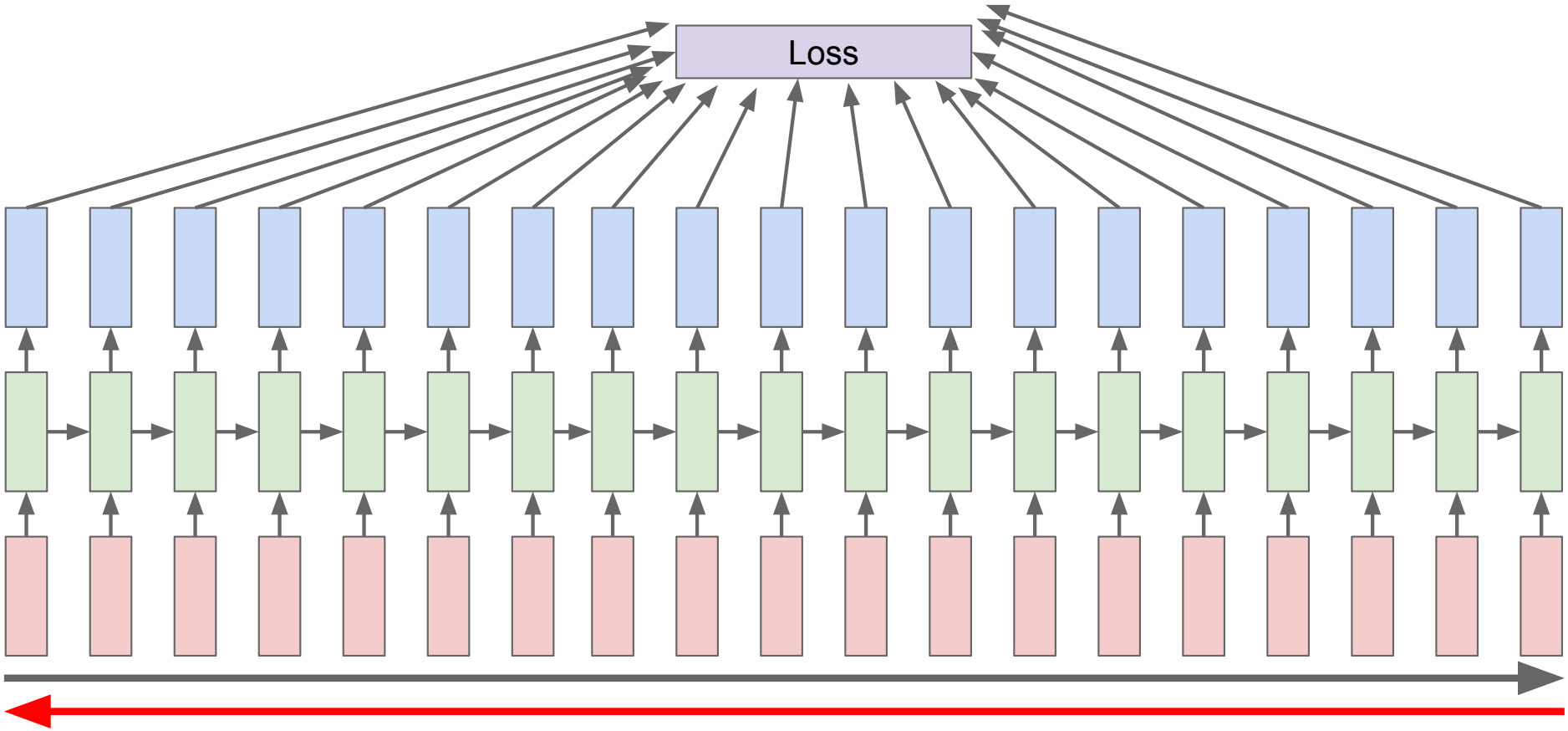
Backpropagation through Time

How is a weight correlated to local loss?



Backpropagation through Time

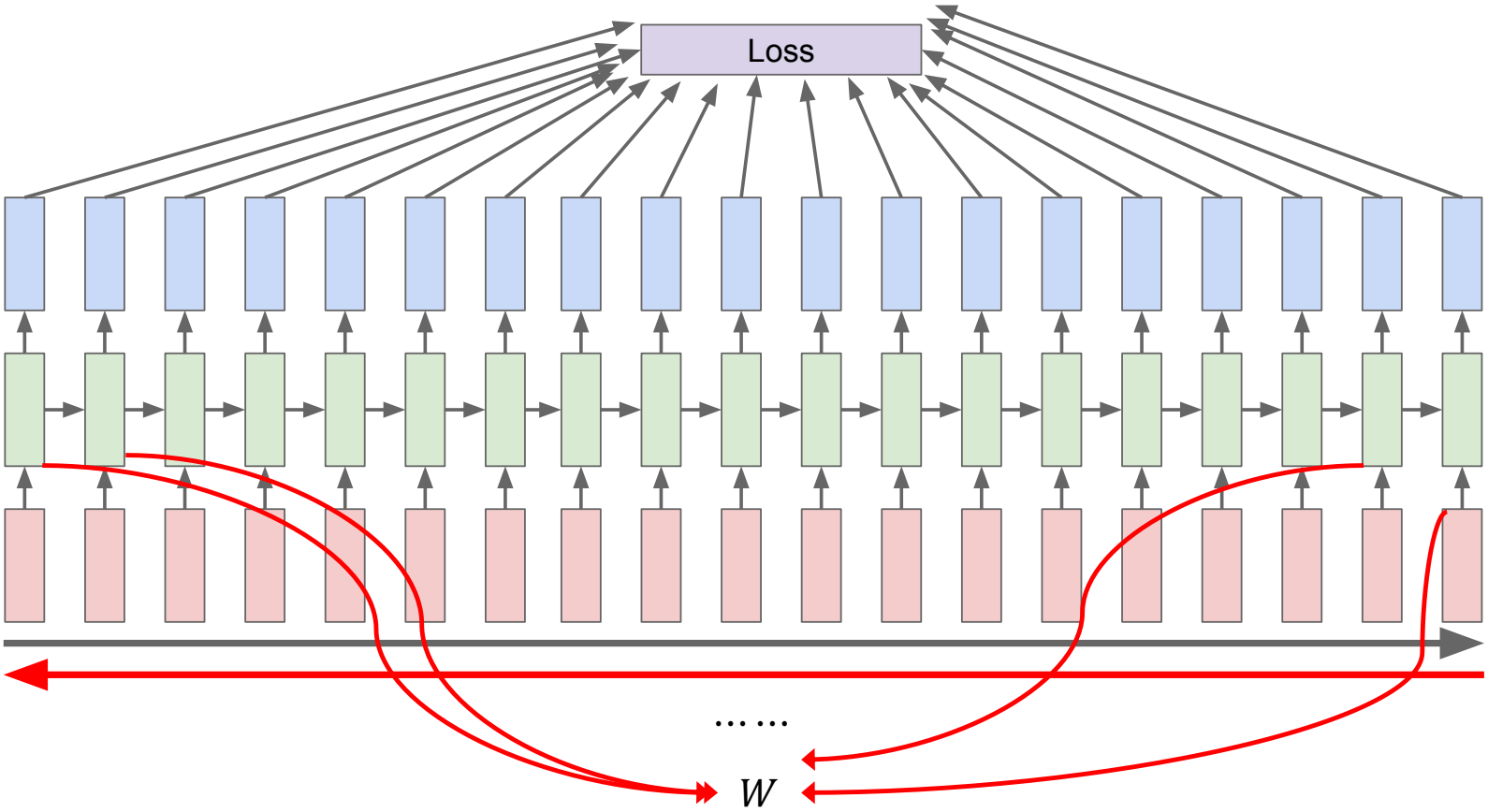
How is a weight correlated to local loss?



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

Backpropagation through Time

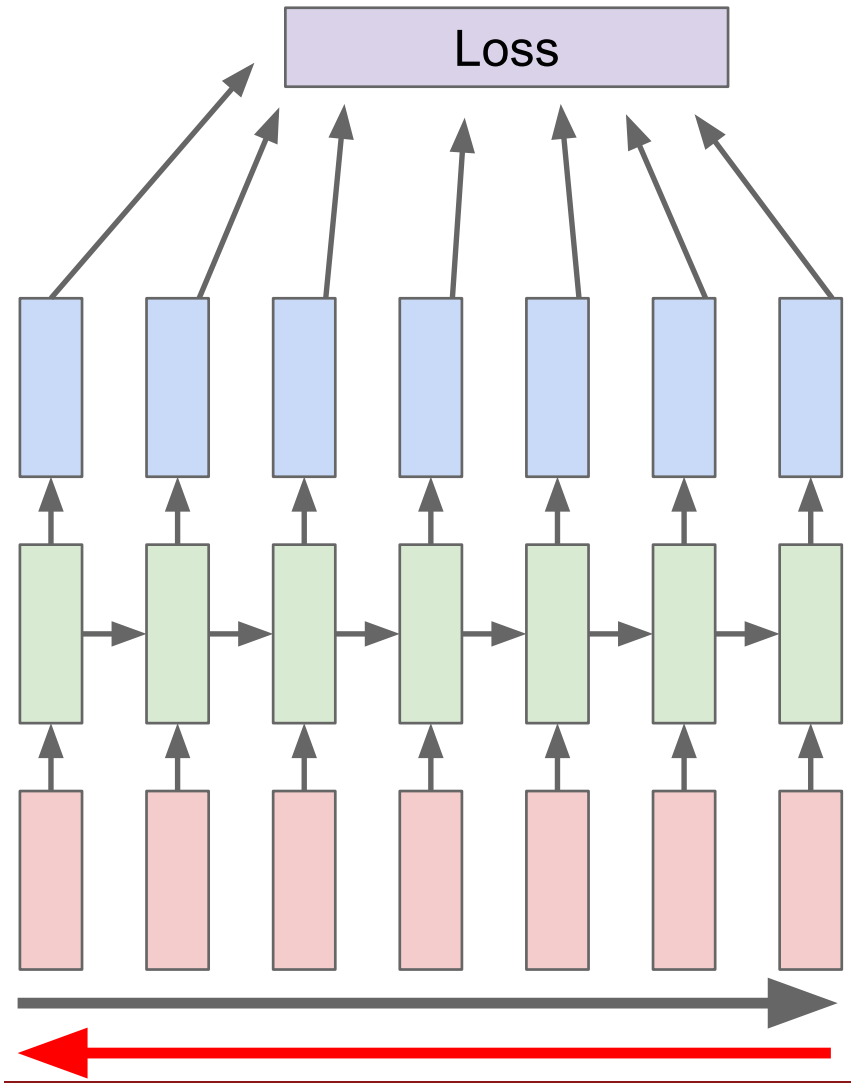
How is a weight correlated to local loss?



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

$\Rightarrow$ Too computationally expensive!

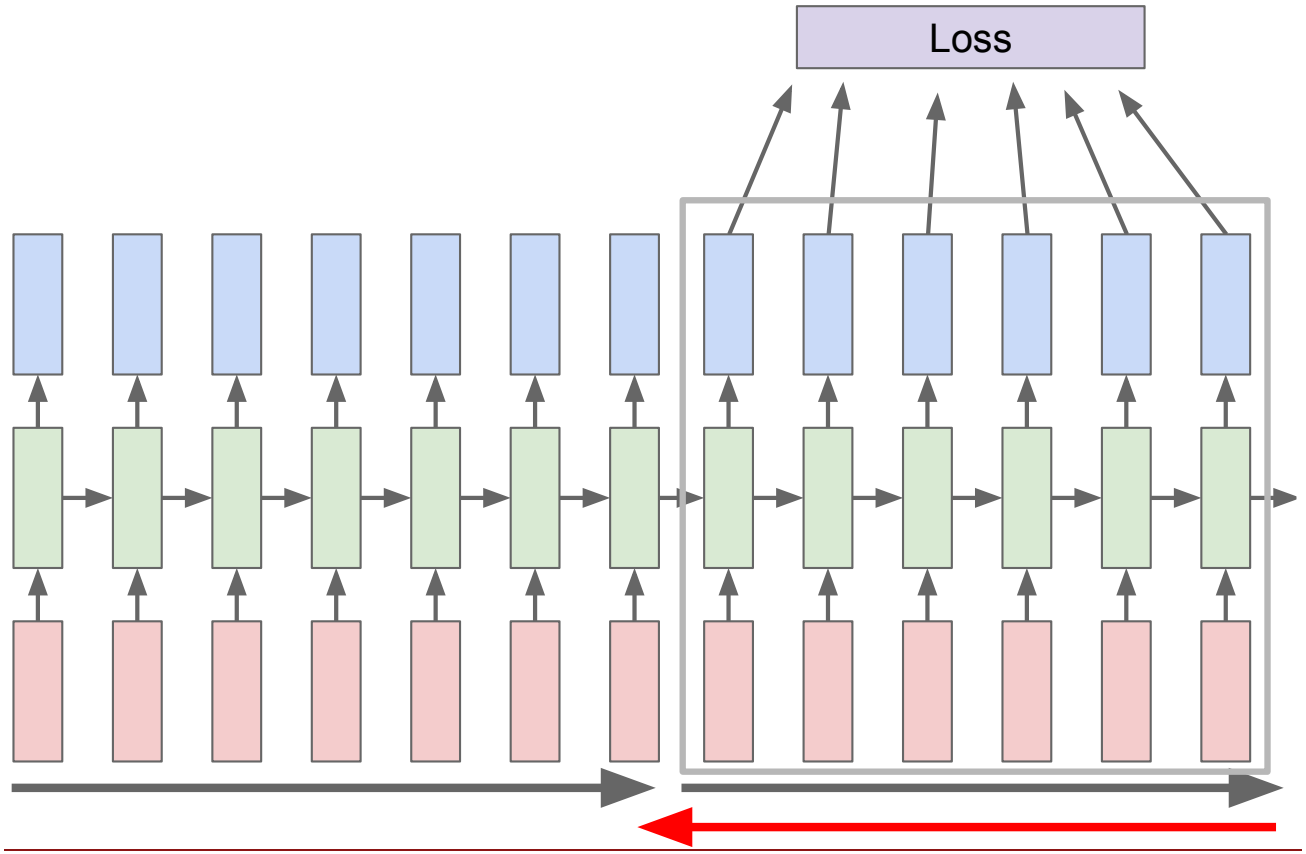
Backpropagation through Time



Run forward and backward through chunks of the sequence instead of whole sequence:

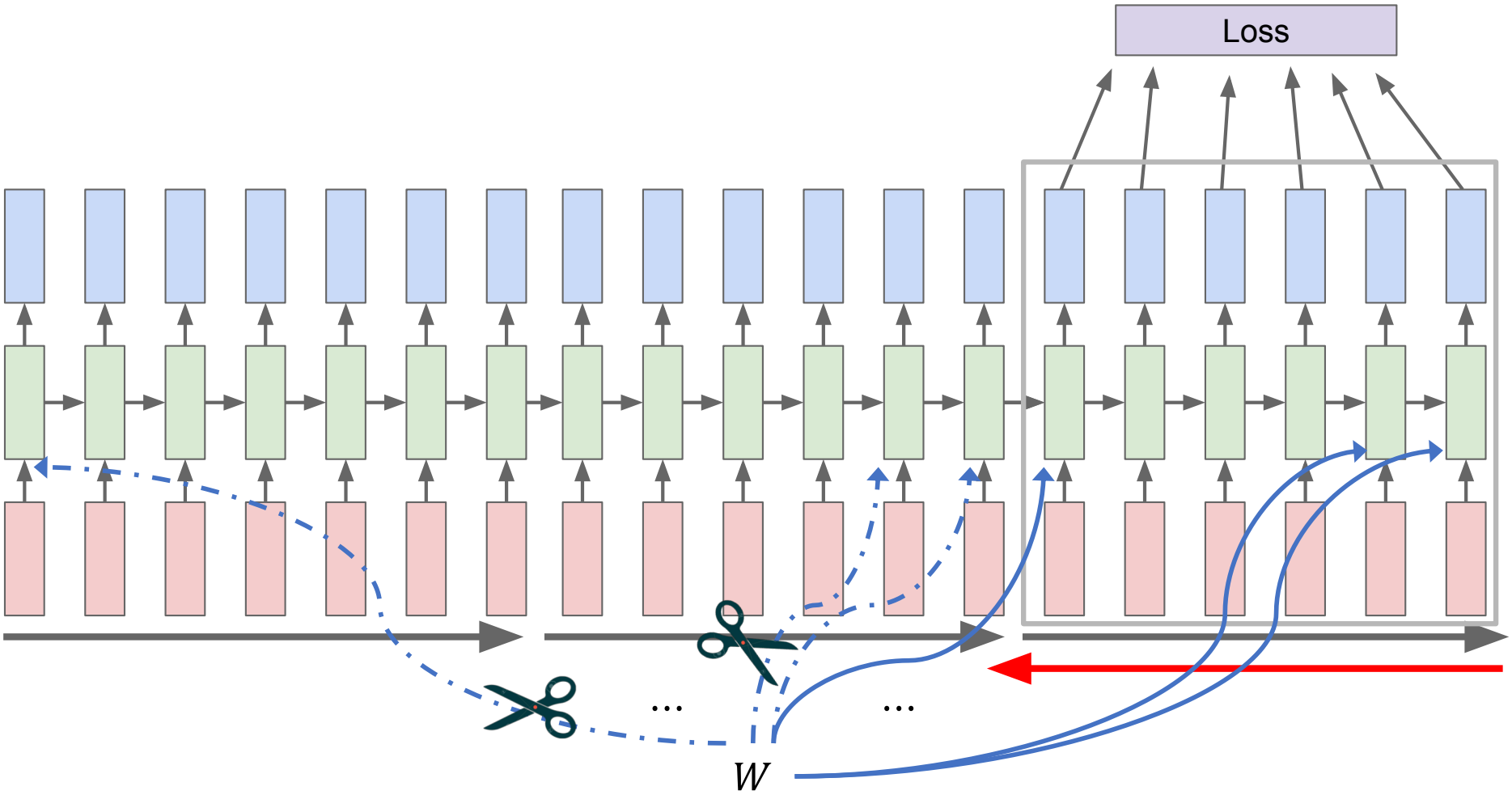
- greatly reduces computational time without too much loss in accuracy

Backpropagation through Time



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

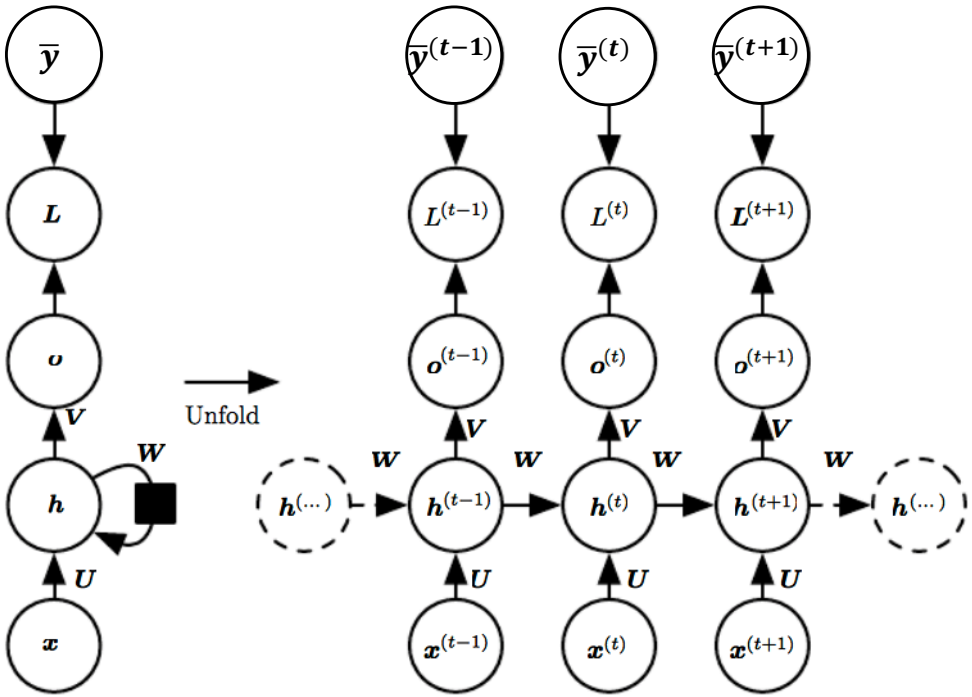
Backpropagation through Time



Example: Computing Gradients by BPTT

- Consider the following RNN model:

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}$$

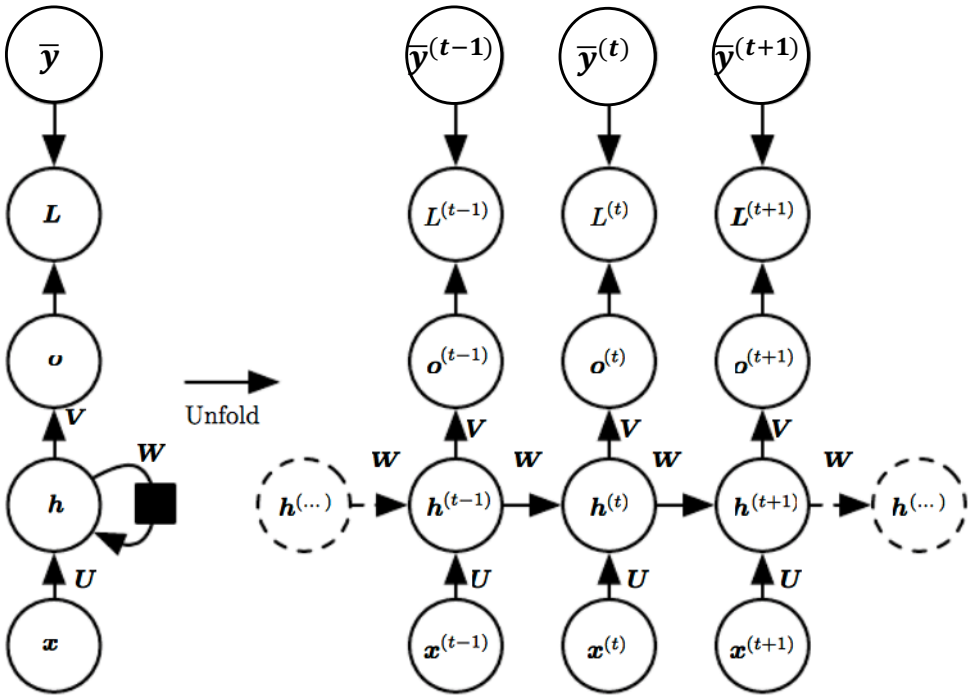


- For each node N, we need to compute the gradient $\nabla_N \mathcal{L}$ recursively, based on gradients at nodes that follow it in the graph.
- Assume the outputs $o^{(t)}$ are used as the arguments to the softmax function to obtain the vector $\hat{y}^{(t)}$ of probabilities over the output.

Example: Computing Gradients by BPTT

- Consider the following RNN model:

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}$$

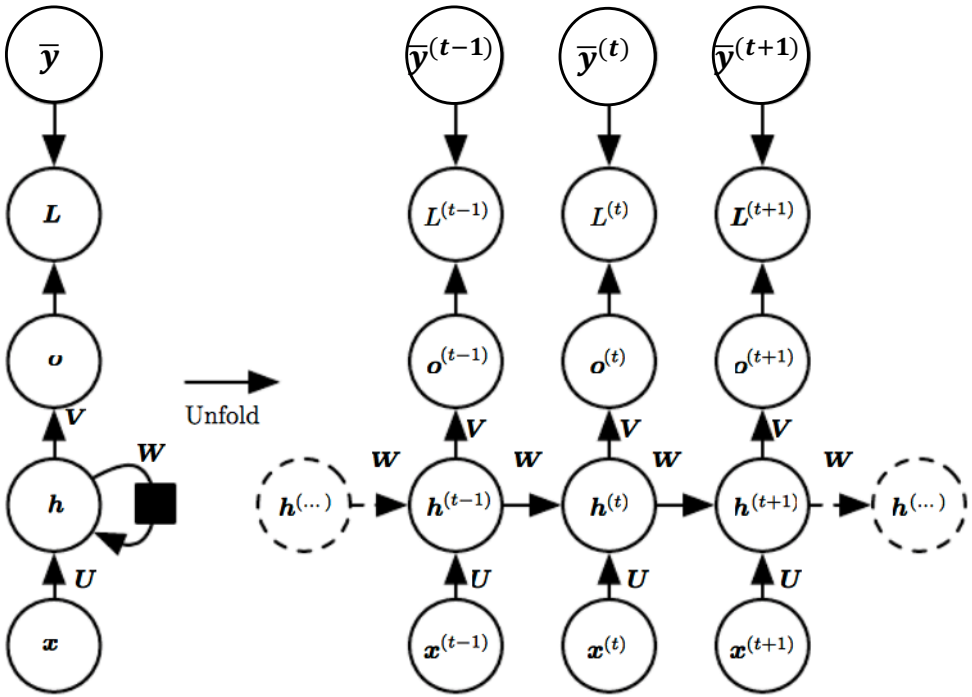


- For each node N, we need to compute the gradient $\nabla_N \mathcal{L}$ recursively, based on gradients at nodes that follow it in the graph.
- Assume the outputs $o^{(t)}$ are used as the arguments to the softmax function to obtain the vector $\hat{y}^{(t)}$ of probabilities over the output.

Example: Computing Gradients by BPTT

- Consider the following RNN model:

$$\mathbf{a}^{(t)} = \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{c}$$



- For each node N, we need to compute the gradient $\nabla_N \mathcal{L}$ recursively, based on gradients at nodes that follow it in the graph.
- Assume the outputs $o^{(t)}$ are used as the arguments to the softmax function to obtain the vector $\hat{y}^{(t)}$ of probabilities over the output.

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

$$(\nabla_{\mathbf{o}^{(t)}} \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \overset{\text{softmax property}}{=}$$

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

$$(\nabla_{\mathbf{o}^{(t)}} \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \overset{\text{softmax property}}{=} \hat{y}_i^{(t)} - \bar{y}_i^{(t)} \ .$$

Example: Computing Gradients by BPTT

1. Start the recursion with the nodes immediately preceding the final loss:

$$\mathcal{L} = \sum_t L^{(t)} \Rightarrow \frac{\partial \mathcal{L}}{\partial L^t} = 1$$

2. Use cross entropy for loss:

$$\mathcal{L}^{(t)} = -\sum_{i=1}^{K} \bar{y}_i^{(t)} \log \hat{y}_i^{(t)} \ ,$$

where $\hat{y}_i^{(t)} = \frac{e^{o_i^{(t)}}}{\sum_j e^{o_j^{(t)}}}$ is the output of softmax.

3. The gradient on the outputs at time t is:

$$(\nabla_{\mathbf{o}^{(t)}} \mathcal{L})_i = \frac{\partial \mathcal{L}}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \overset{\text{softmax property}}{=} \hat{y}_i^{(t)} - \bar{y}_i^{(t)} \ .$$

$$\Rightarrow \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \hat{\mathbf{y}}^{(t)} - \bar{\mathbf{y}}^{(t)}$$

Example: Computing Gradients by BPTT

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$

1. Iterate backwards. At the final time step t the gradient is (two paths for $\mathbf{h}^t$):

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \left(\frac{\partial\,\mathbf{h}^{(t+1)}}{\partial\,\mathbf{h}^{(t)}}\right)^T \left(\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}\right) + \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{h}^{(t)}}\right)^T \left(\nabla_{\mathbf{o}^{(t)}}\mathcal{L}\right)$$
$$= $$

Example: Computing Gradients by BPTT

$$\mathbf{a}^{(t)} = \mathbf{W}\,\mathbf{h}^{(t-1)} + \mathbf{U}\,\mathbf{x}^{(t)} + \mathbf{b}$$
$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$$
$$\mathbf{o}^{(t)} = \mathbf{V}\,\mathbf{h}^{(t)} + \mathbf{c}$$

1. Iterate backwards. At the final time step t the gradient is (two paths for $\mathbf{h}^t$):

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \left(\frac{\partial\,\mathbf{h}^{(t+1)}}{\partial\,\mathbf{h}^{(t)}}\right)^T \left(\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}\right) + \left(\frac{\partial\,\mathbf{o}^{(t)}}{\partial\,\mathbf{h}^{(t)}}\right)^T \left(\nabla_{\mathbf{o}^{(t)}}\mathcal{L}\right)$$
$$= \mathbf{W}^T \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t+1)})\right)\left(\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}\right) + \mathbf{V}^T\left(\nabla_{\mathbf{o}^{(t)}}\mathcal{L}\right)$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} =$$
$$\nabla_{\mathbf{b}} \mathcal{L} =$$
$$\nabla_{\mathbf{V}} \mathcal{L} =$$
$$\nabla_{\mathbf{W}} \mathcal{L} =$$
$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} =$$

$$\nabla_{\mathbf{V}} \mathcal{L} =$$

$$\nabla_{\mathbf{W}} \mathcal{L} =$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} =$$

$$\nabla_{\mathbf{V}} \mathcal{L} =$$

$$\nabla_{\mathbf{W}} \mathcal{L} =$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} =$$

$$\nabla_{\mathbf{W}} \mathcal{L} =$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} =$$

$$\nabla_{\mathbf{W}} \mathcal{L} =$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}} \right)^T \nabla_{\mathbf{V}} o^{(t)}$$

$$\nabla_{\mathbf{W}} \mathcal{L} =$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}} \right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t)T}$$

$$\nabla_{\mathbf{W}} \mathcal{L} =$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}} \right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t)T}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}} \right)^T \nabla_{\mathbf{W}} h^{(t)}$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}} \right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}} \right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) (\nabla_{\mathbf{h}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}} \mathcal{L} =$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial o^{(t)}} \right)^T \nabla_{\mathbf{V}} o^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}} \right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) (\nabla_{\mathbf{h}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}} \right) \nabla_{\mathbf{U}^{(t)}} h^{(t)}$$

Example: Computing Gradients by BPTT

- Once the gradients on the internal nodes of the computational graph are obtained, we can obtain gradients on the parameter nodes:

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}} \right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}} \right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(t)}} \right)^T \nabla_{\mathbf{V}} \mathbf{o}^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}} \right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) (\nabla_{\mathbf{h}^{(t)}} \mathcal{L}) \, \mathbf{h}^{(t-1)\,T}$$

$$\nabla_{\mathbf{U}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}} \right) \nabla_{\mathbf{U}^{(t)}} h^{(t)} = \sum_t \text{diag} \left(1 - \tanh^2(\mathbf{a}^{(t)}) \right) (\nabla_{\mathbf{h}^{(t)}} \mathcal{L}) \, \mathbf{x}^{(t)\,T}$$

Summary: Gradients in RNN

$$\nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \hat{\mathbf{y}}^{(t)} - \bar{\mathbf{y}}^{(t)}$$

$$\nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \mathbf{W}^T \, \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t+1)})\right) \left(\nabla_{\mathbf{h}^{(t+1)}} \mathcal{L}\right) + \mathbf{V}^T \left(\nabla_{\mathbf{o}^{(t)}} \mathcal{L}\right)$$

$$\nabla_{\mathbf{c}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}} \mathcal{L} = \sum_t \nabla_{\mathbf{o}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{b}} \mathcal{L} = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}} \mathcal{L} = \sum_t \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \nabla_{\mathbf{h}^{(t)}} \mathcal{L}$$

$$\nabla_{\mathbf{V}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial \mathbf{o}^{(t)}}\right)^T \nabla_{\mathbf{V}} \mathbf{o}^{(t)} = \sum_t \left(\nabla_{\mathbf{o}^{(t)}} \mathcal{L}\right) \mathbf{h}^{(t)\,T}$$

$$\nabla_{\mathbf{W}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}}\right)^T \nabla_{\mathbf{W}} h^{(t)} = \sum_t \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \left(\nabla_{\mathbf{h}^{(t)}} \mathcal{L}\right) \mathbf{h}^{(t-1)\,T}$$

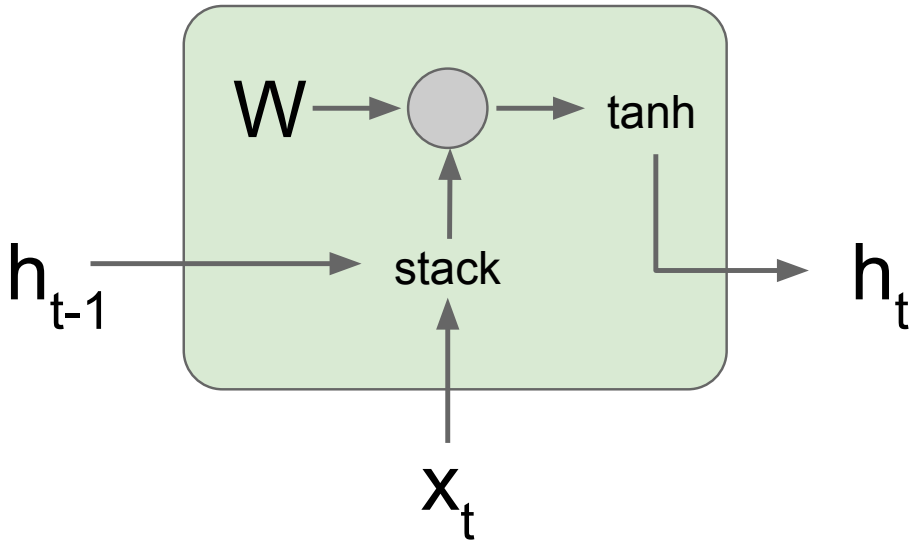
$$\nabla_{\mathbf{U}} \mathcal{L} = \sum_t \left(\frac{\partial \mathcal{L}}{\partial h^{(t)}}\right) \nabla_{\mathbf{U}^{(t)}} h^{(t)} = \sum_t \mathbf{diag}\left(1 - \tanh^2(\mathbf{a}^{(t)})\right) \left(\nabla_{\mathbf{h}^{(t)}} \mathcal{L}\right) \mathbf{x}^{(t)\,T}$$

Long Short Term Memory[2]

[2] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Vanilla RNN Gradient Flow

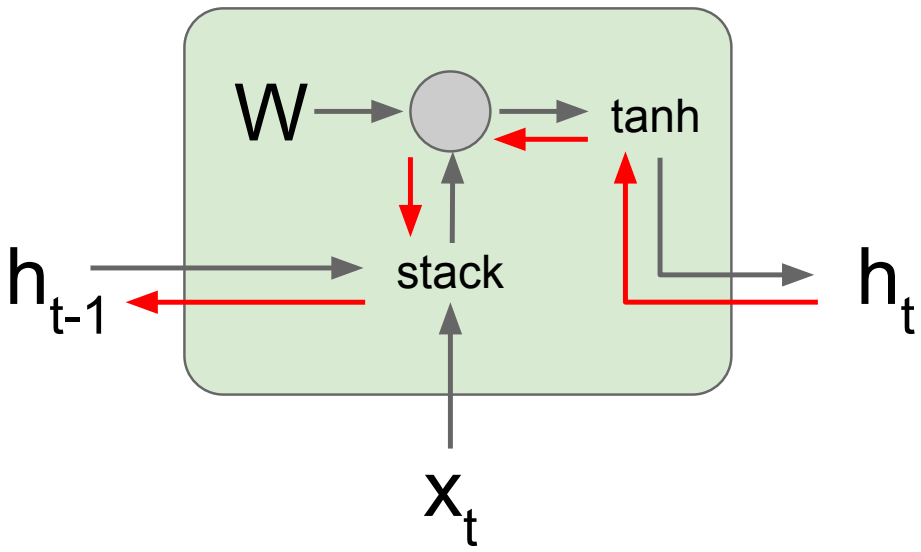
$$\begin{aligned}\mathbf{h}_t &= \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right) \\ &= \tanh\left(\begin{bmatrix}\mathbf{W}_{hh} & \mathbf{W}_{xh}\end{bmatrix}\right)\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix} \\ &= \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix}\right)\end{aligned}$$



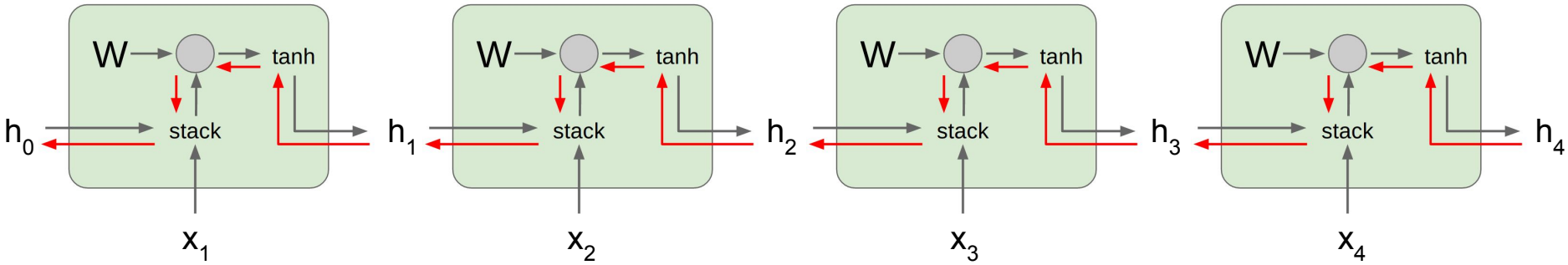
Vanilla RNN Gradient Flow

$$\begin{aligned}\mathbf{h}_t &= \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right) \\ &= \tanh\left(\begin{bmatrix}\mathbf{W}_{hh} & \mathbf{W}_{xh}\end{bmatrix}\right)\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix} \\ &= \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1} \\ \mathbf{x}_t\end{bmatrix}\right)\end{aligned}$$

Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



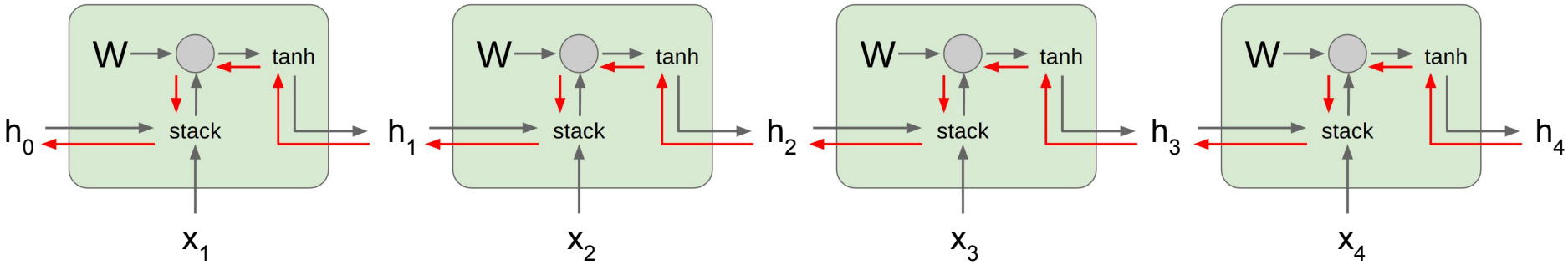
Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow

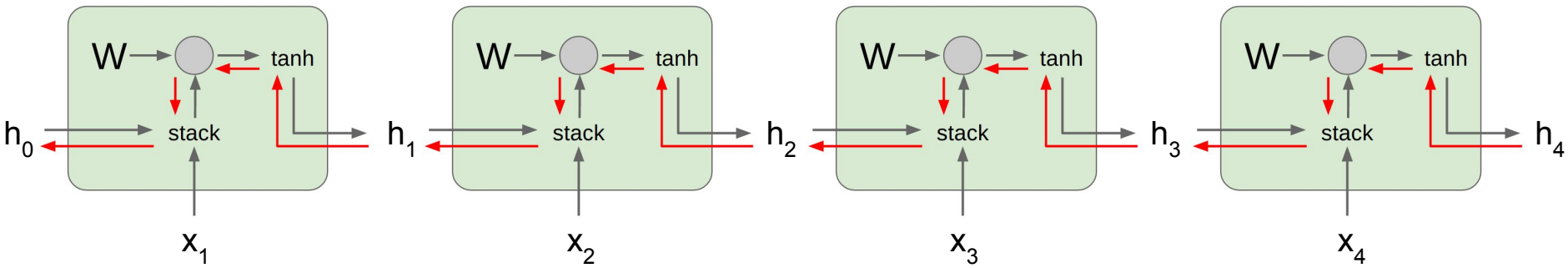


Computing gradient of h_0 involves many factors of W (and repeated tanh)

$$\nabla_{\mathbf{h}^{(t)}}\mathcal{L} = \mathbf{W}^T \operatorname{diag}\left(1 - \tanh^2(\mathbf{a}^{(t+1)})\right)\left(\nabla_{\mathbf{h}^{(t+1)}}\mathcal{L}\right) + \mathbf{V}^T\left(\nabla_{\mathbf{o}^{(t)}}\mathcal{L}\right)$$

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



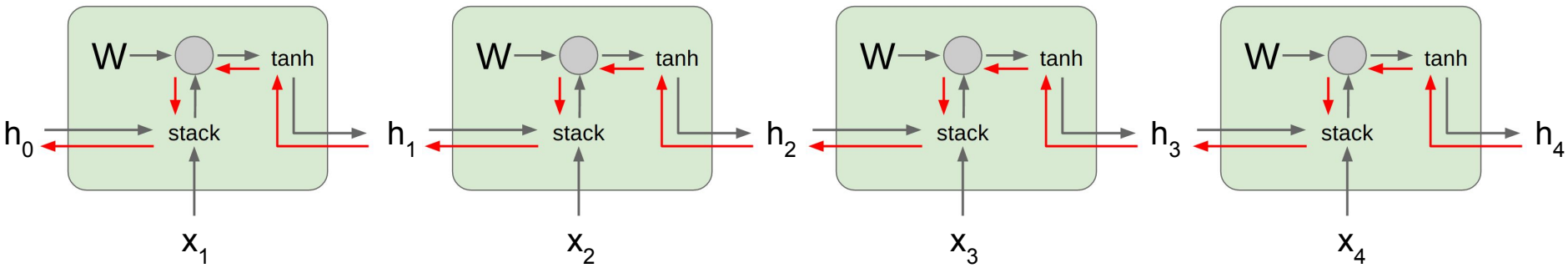
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

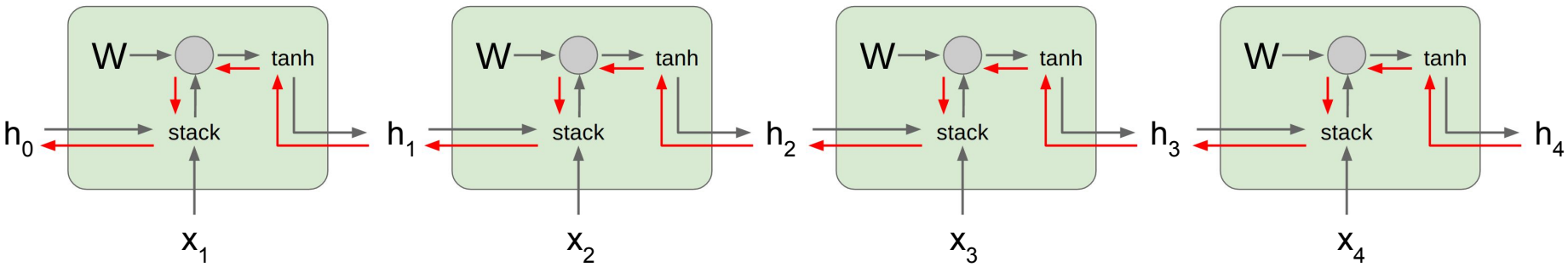
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Why?

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1: **Exploding gradients**

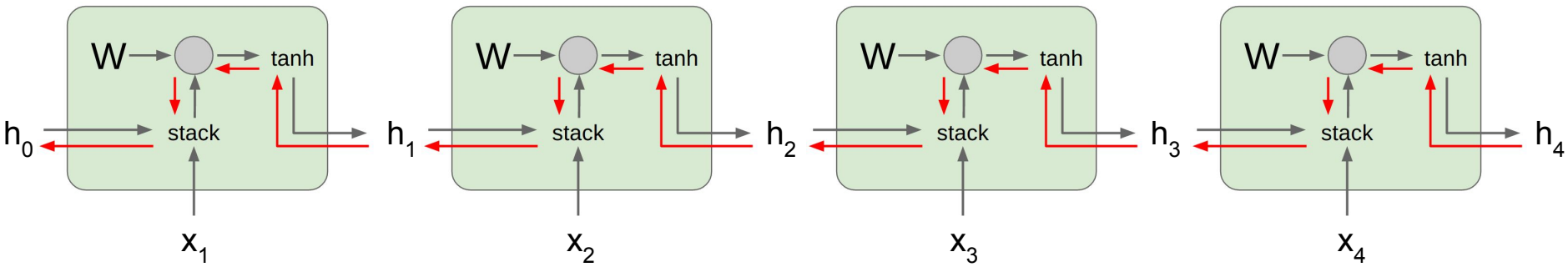
→ **Gradient clipping**: Scale gradient if its norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1: **Vanishing gradients**

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



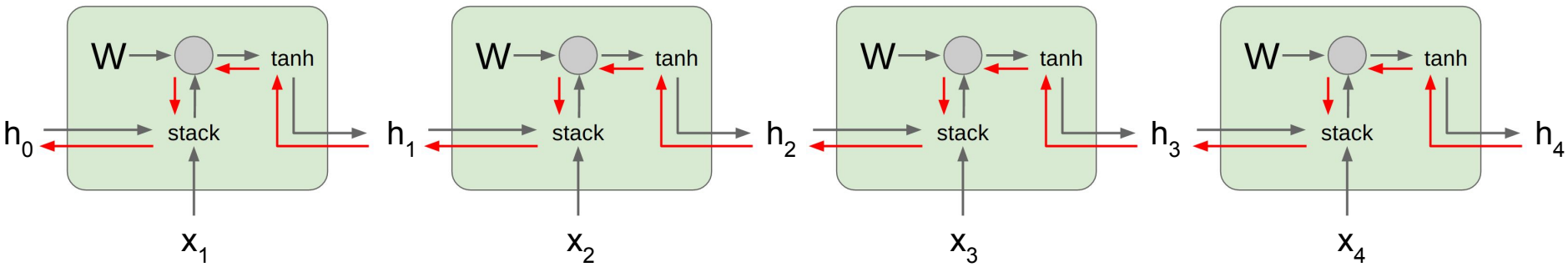
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients → Change RNN architecture

Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN Gradient Flow



Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients → Change RNN architecture

Long Short Term Memory!

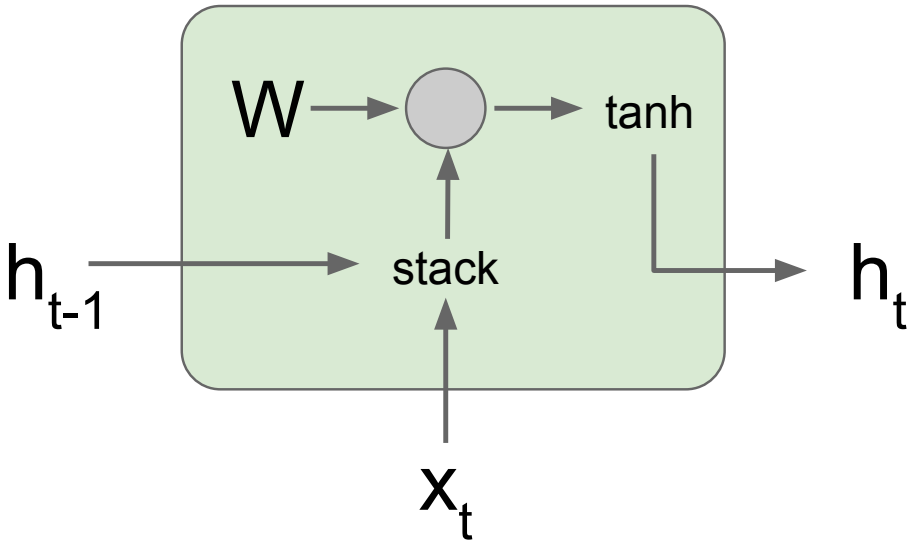
Bengio, et al., TNN 1994; Pascanu, et al., ICML 2013

Vanilla RNN

$$\mathbf{h}_t = \tanh\left(\mathbf{W}_{hh}\,\mathbf{h}_{t-1} + \mathbf{W}_{xh}\,\mathbf{x}_t\right)$$

$$= \tanh\left(\begin{bmatrix}\mathbf{W}_{hh} & \mathbf{W}_{xh}\end{bmatrix}\right)\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}$$

$$= \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_{t-1}\\ \mathbf{x}_t\end{bmatrix}\right)$$



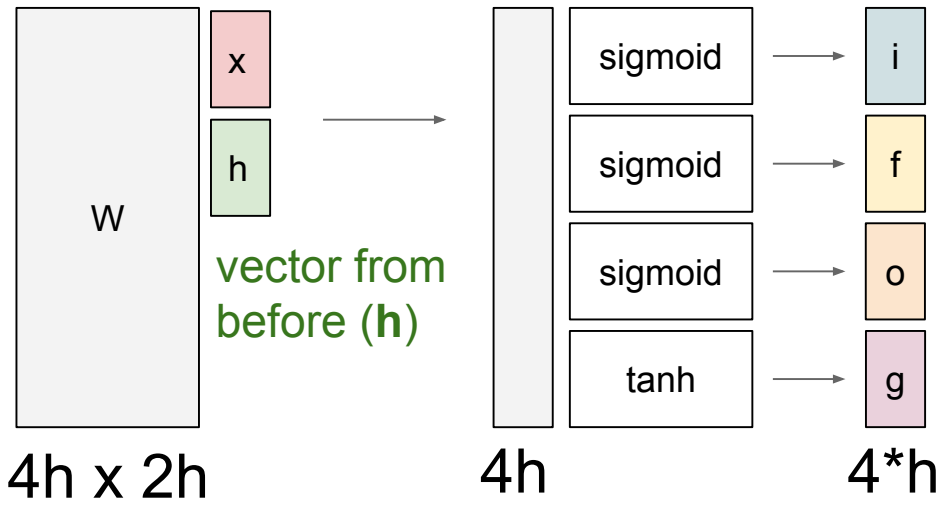
Long Short Term Memory (LSTM)

- **f**: *Forget gate*, whether to erase cell
- **i**: *Input gate*, whether to write to cell
- **g**: *New content*, how much to write to cell
- **o**: *Output gate*, how much to reveal cell

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W} \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix} \right]$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$$

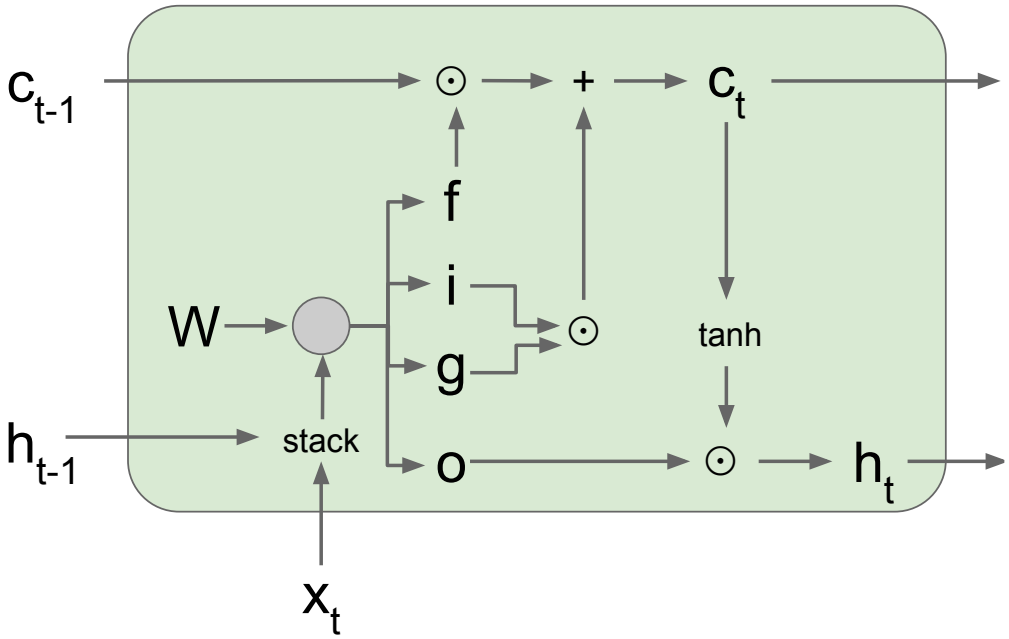
$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$



Hochreiter & Schmidhuber, Neural Computation 1997

Long Short Term Memory (LSTM)

Create an additional internal hidden state $\mathbf{c}_t$ for each time t.



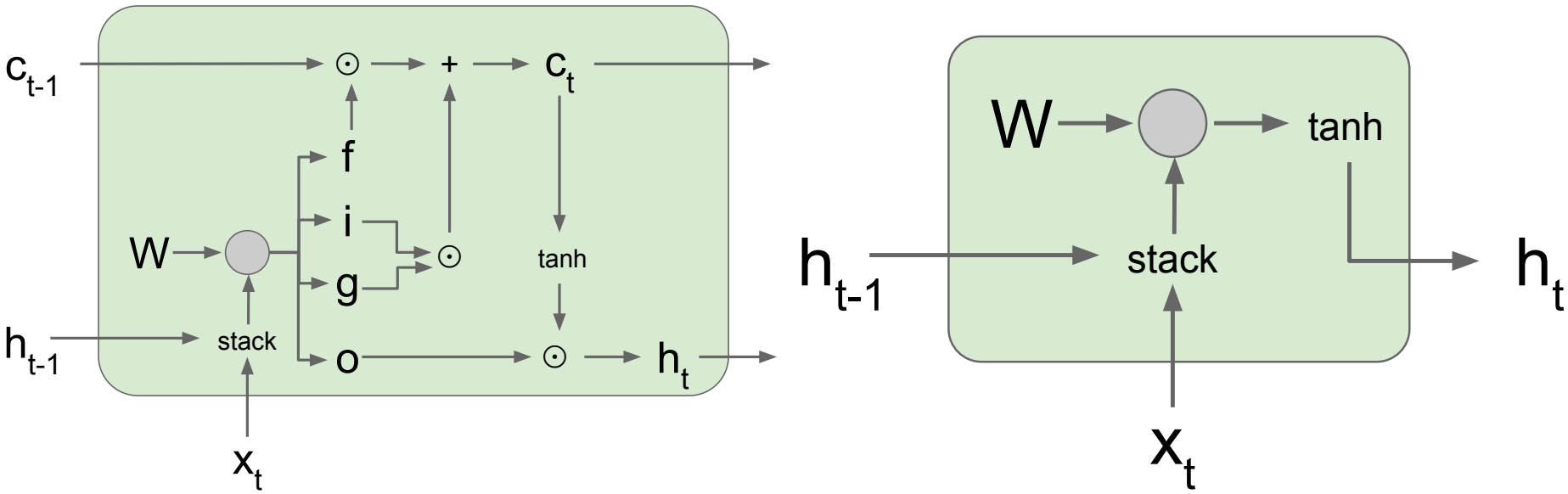
$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W} \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix} \right]$$

$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$$

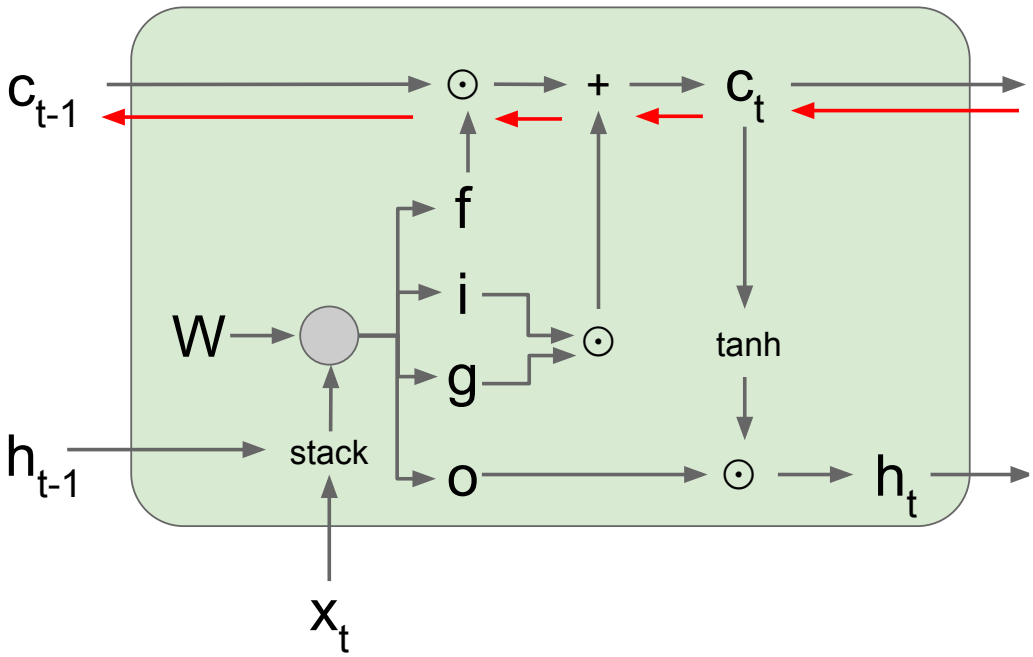
$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Hochreiter & Schmidhuber, Neural Computation 1997

Long Short Term Memory (LSTM)



Long Short Term Memory: Gradient Flow



Backpropagation from $\mathbf{c}_t$ to $\mathbf{c}_{t-1}$ only elementwise multiplication by $\mathbf{f}$, no direct matrix multiply by $\mathbf{W} \Rightarrow$ no gradient vanishing.

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W} \begin{pmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{pmatrix} \right]$$

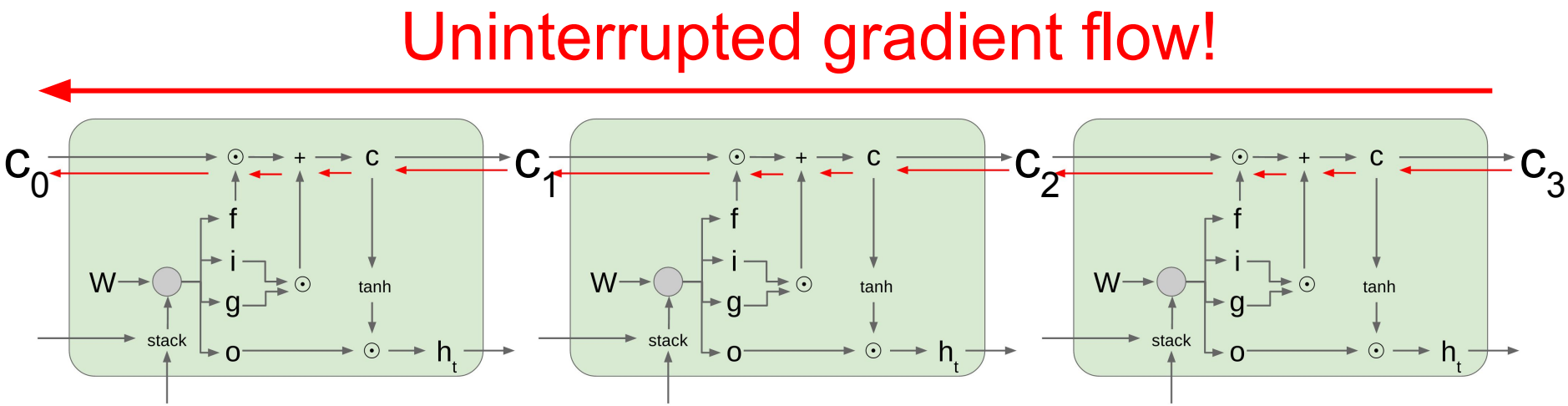
$$\mathbf{c}_t = \mathbf{f} \odot \mathbf{c}_{t-1} + \mathbf{i} \odot \mathbf{g}$$

$$\mathbf{h}_t = \mathbf{o} \odot \tanh(\mathbf{c}_t)$$

Hochreiter & Schmidhuber, Neural Computation 1997

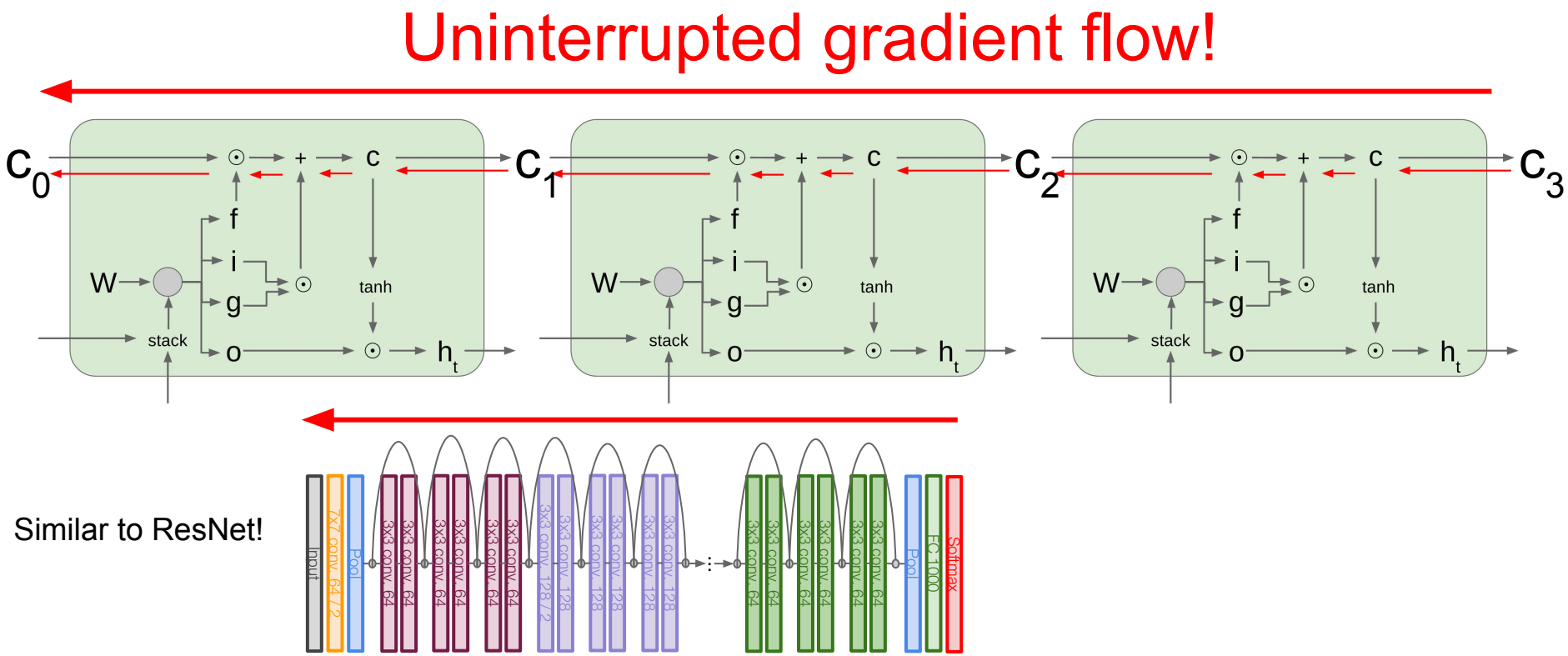
Long Short Term Memory: Gradient Flow

Gradients directly backpropagate through time.



Long Short Term Memory: Gradient Flow

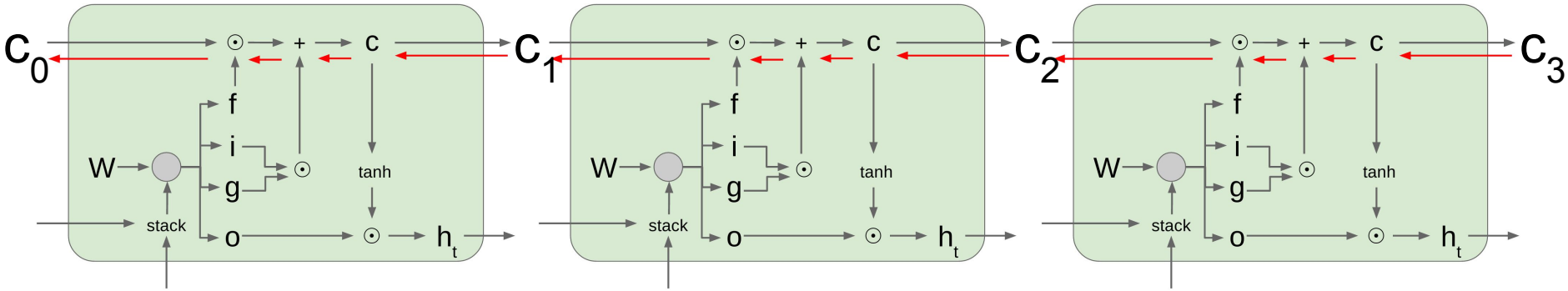
Gradients directly backpropagate through time.



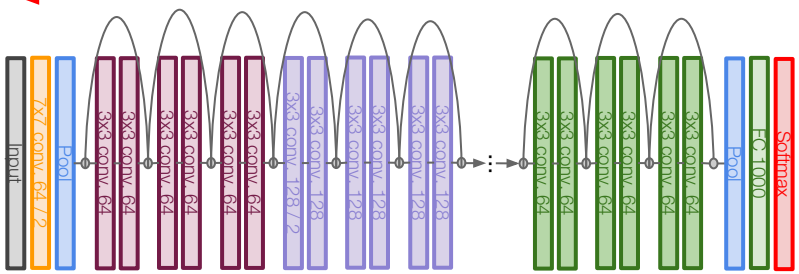
Long Short Term Memory: Gradient Flow

Gradients directly backpropagate through time.

Uninterrupted gradient flow!



Similar to ResNet!



In between:
Highway Networks

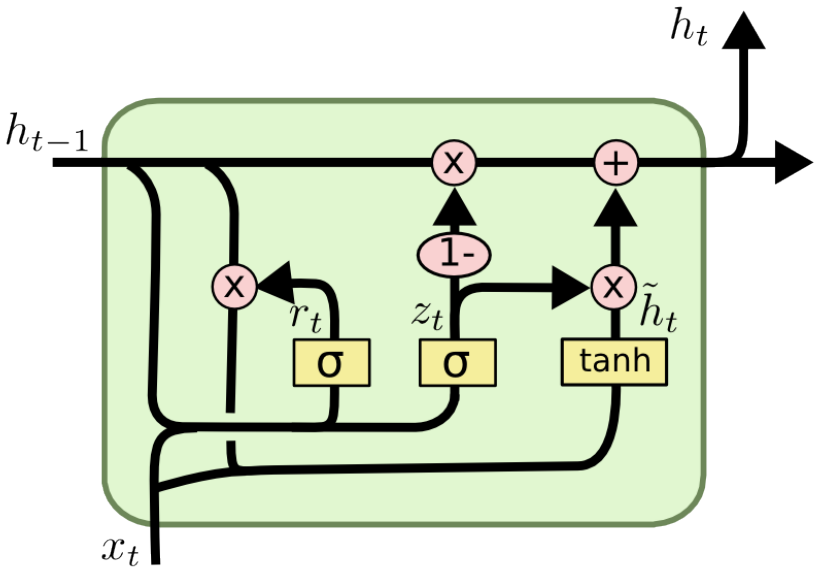
$$g = T(x, W_T)$$
$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al, "Highway Networks", ICML DL Workshop 2015

Other RNN Variants

Guideline

Need uninterrupted gradient flows to avoid gradient vanishing!



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

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

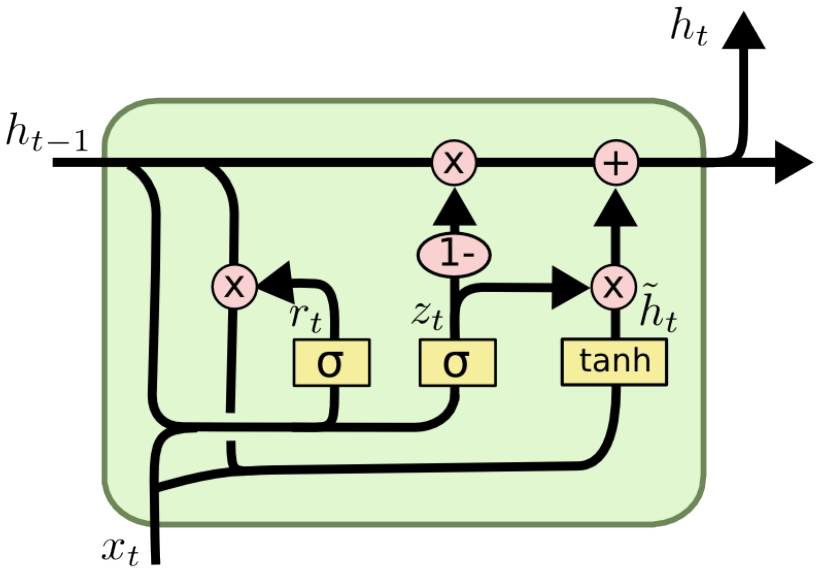
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Other RNN Variants

Gated Recurent Unit (GRU) [Cho et al., 2014]

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_{xr}\,\mathbf{x}_t + \mathbf{W}_{hr}\,\mathbf{h}_{t-1} + \mathbf{b}_r\right)$$
$$\mathbf{z}_t = \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{W}_{hz}\,\mathbf{h}_{t-1} + \mathbf{b}_z\right)$$
$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}_{xh}\,\mathbf{x}_t + \mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right) + \mathbf{b}_h\right)$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \tilde{\mathbf{h}}_{t-1} + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1}$$



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

Other RNN Variants [Jozefowicz et al., 2015]

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{b}_z\right) \\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_{xr}\,\mathbf{x}_t + \mathbf{W}_{hr}\,\mathbf{h}_t + \mathbf{b}_r\right) \\
\mathbf{h}_{t+1} &= \tanh\left(\mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_t\right) + \tanh(\mathbf{x}_t)\right. \\
&\quad \left. + \mathbf{b}_h\right)\mathbf{z}_t + \mathbf{h}_t \odot (1 - \mathbf{z}_t)
\end{aligned}$$

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{W}_{hz}\,\mathbf{h}_t + \mathbf{b}_z\right) \\
\mathbf{r}_t &= \sigma\left(\mathbf{x}_t + \mathbf{b}_r\right) \\
\mathbf{h}_{t+1} &= \tanh\left(\mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_t\right) + \mathbf{W}_{xh}\,\mathbf{x}_t\right. \\
&\quad \left. + \mathbf{b}_h\right)\mathbf{z}_t + \mathbf{h}_t \odot (1 - \mathbf{z}_t)
\end{aligned}$$

$$\begin{aligned}
\mathbf{z}_t &= \sigma\left(\mathbf{W}_{xz}\,\mathbf{x}_t + \mathbf{W}_{hz}\tanh(\mathbf{h}_t) + \mathbf{b}_z\right) \\
\mathbf{r}_t &= \sigma\left(\mathbf{W}_{xr}\,\mathbf{x}_t + \mathbf{W}_{hr}\,\mathbf{h}_t + \mathbf{b}_r\right) \\
\mathbf{h}_{t+1} &= \tanh\left(\mathbf{W}_{hh}\left(\mathbf{r}_t \odot \mathbf{h}_t\right) + \mathbf{W}_{xh}\,\mathbf{x}_t\right. \\
&\quad \left. + \mathbf{b}_h\right)\mathbf{z}_t + \mathbf{h}_t \odot (1 - \mathbf{z}_t)
\end{aligned}$$

Summary

1. RNNs allow a lot of flexibility in architecture design.
2. Vanilla RNNs are simple but don't work very well.
3. Common to use LSTM or GRU: their additive interactions improve gradient flow.
4. Backward flow of gradients in RNN can explode or vanish:
   - exploding is controlled with gradient clipping.
   - vanishing is controlled with additive interactions (LSTM).
5. Better/simpler architectures are a hot topic of current research.
6. Better understanding (both theoretical and empirical) is needed.

Applications[3]

[3] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf

Recap: Long Short Term Memory

Gradients directly backpropagate through time.

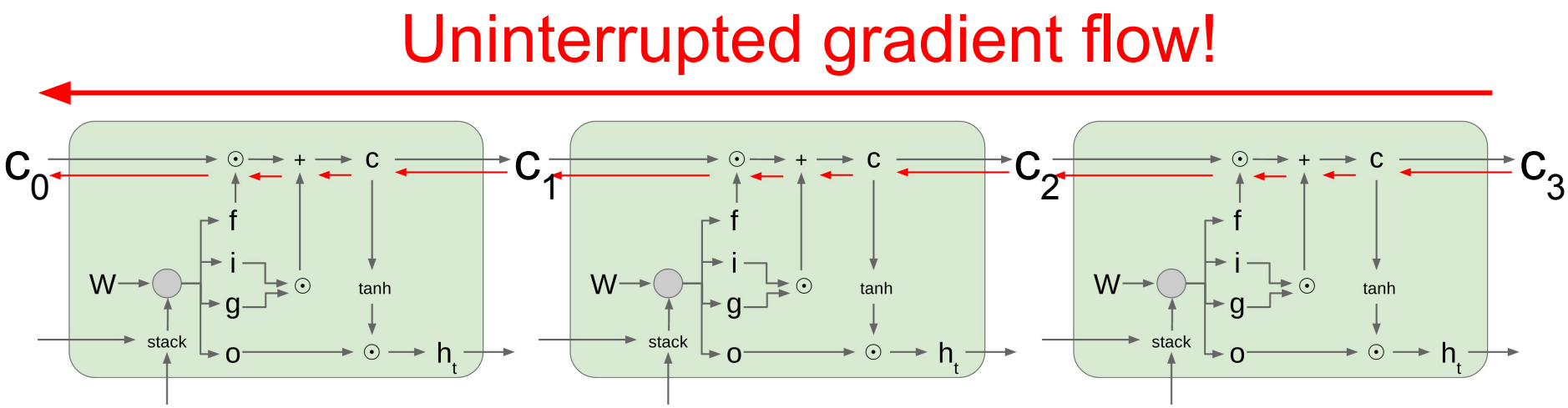


Image Captioning

How to predict a sequence of text (caption) from an image input?

Image Captioning

CNN for feature extraction; Many-to-many RNN for caption generation.

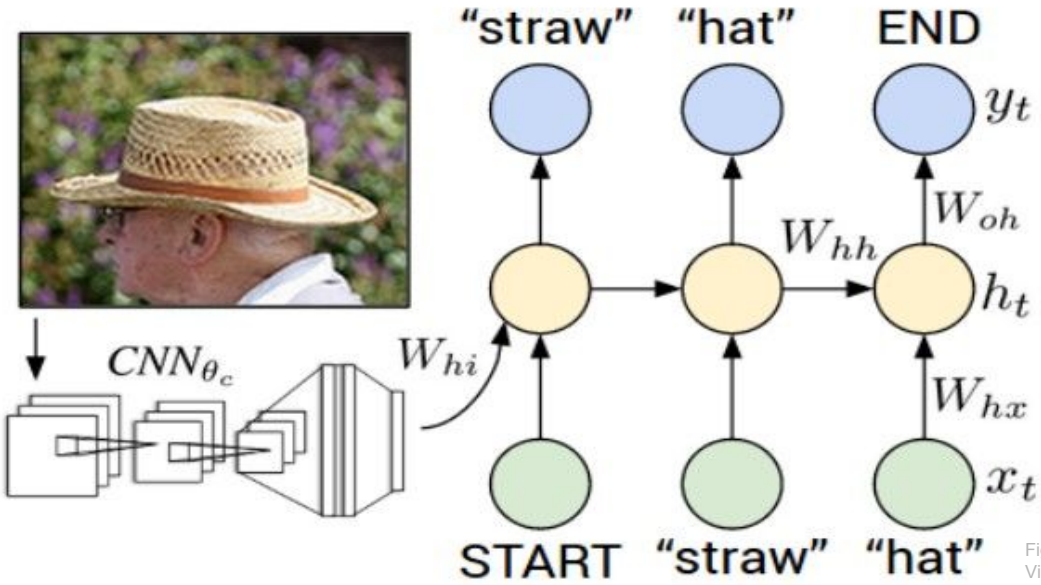


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015. Reproduced for educational purposes.

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

Image Captioning

CNN for feature extraction; Many-to-many RNN for caption generation.

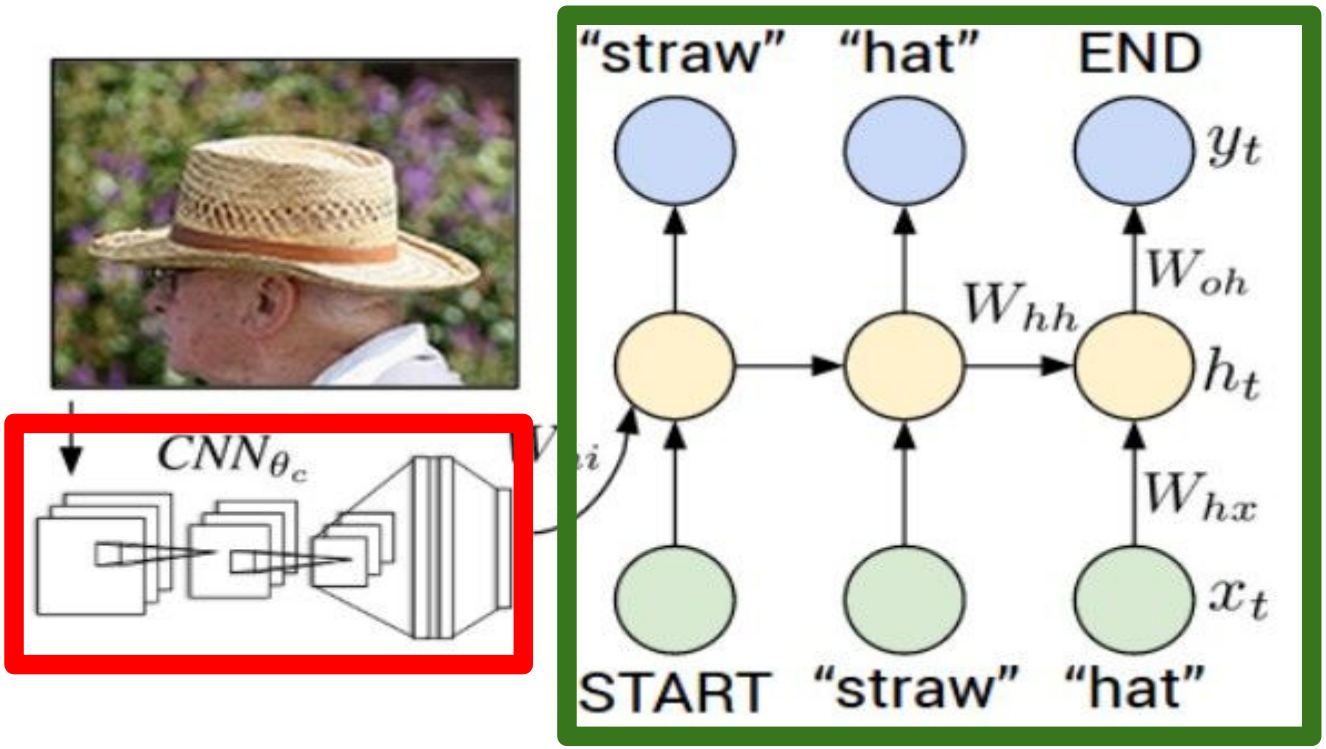


Image Captioning

This image is CC0 public domain

test image

Image Captioning

test image

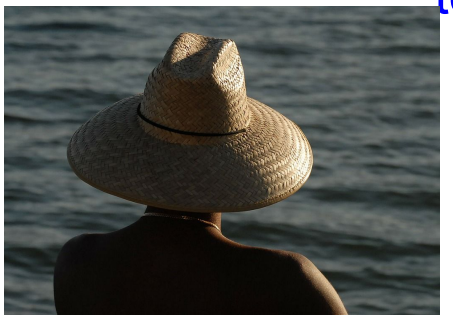
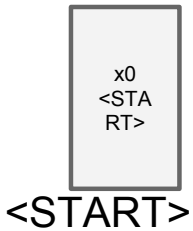


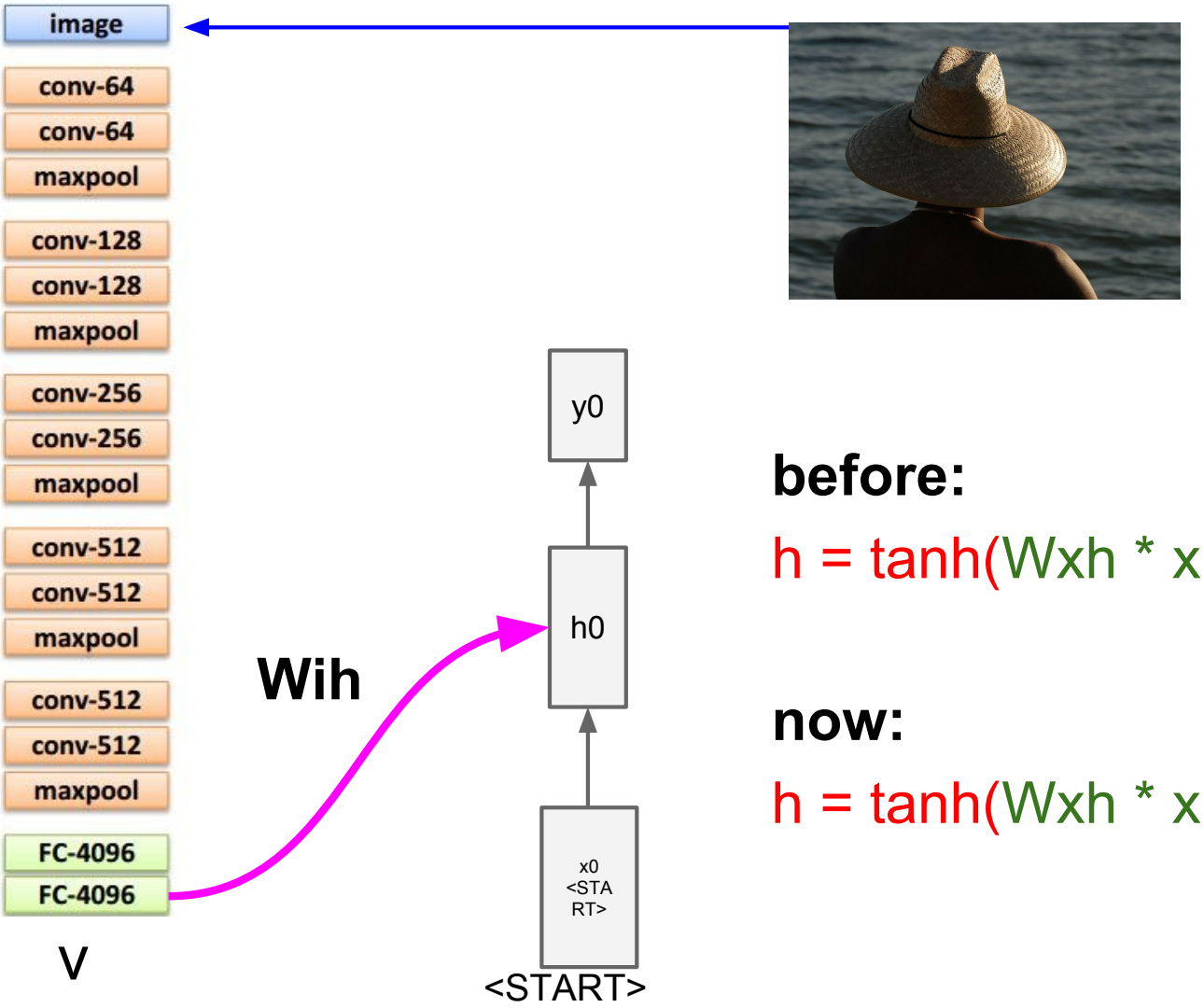
Image Captioning

test image



<START>

Image Captioning



before:

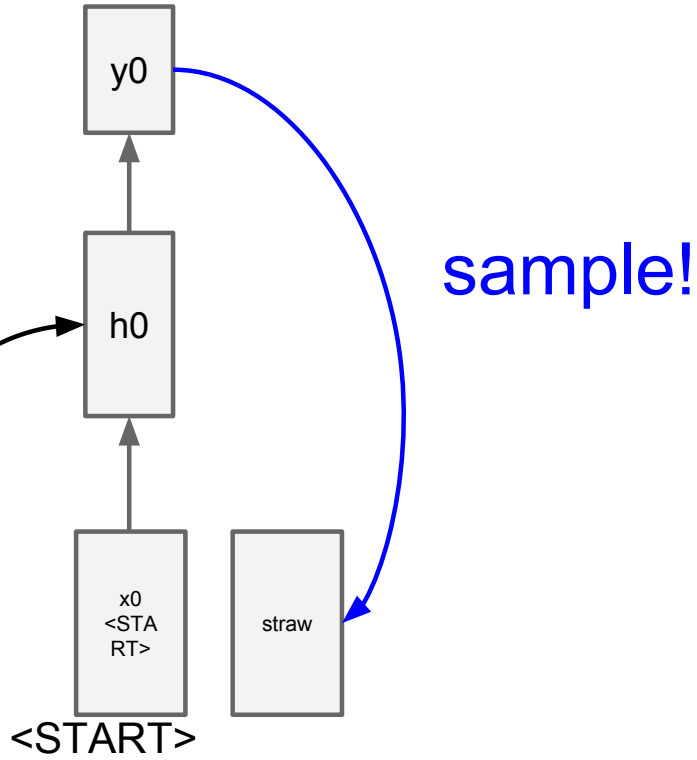
h = tanh(Wxh * x + Whh * h)

now:

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

Image Captioning

test image



sample!

Image Captioning

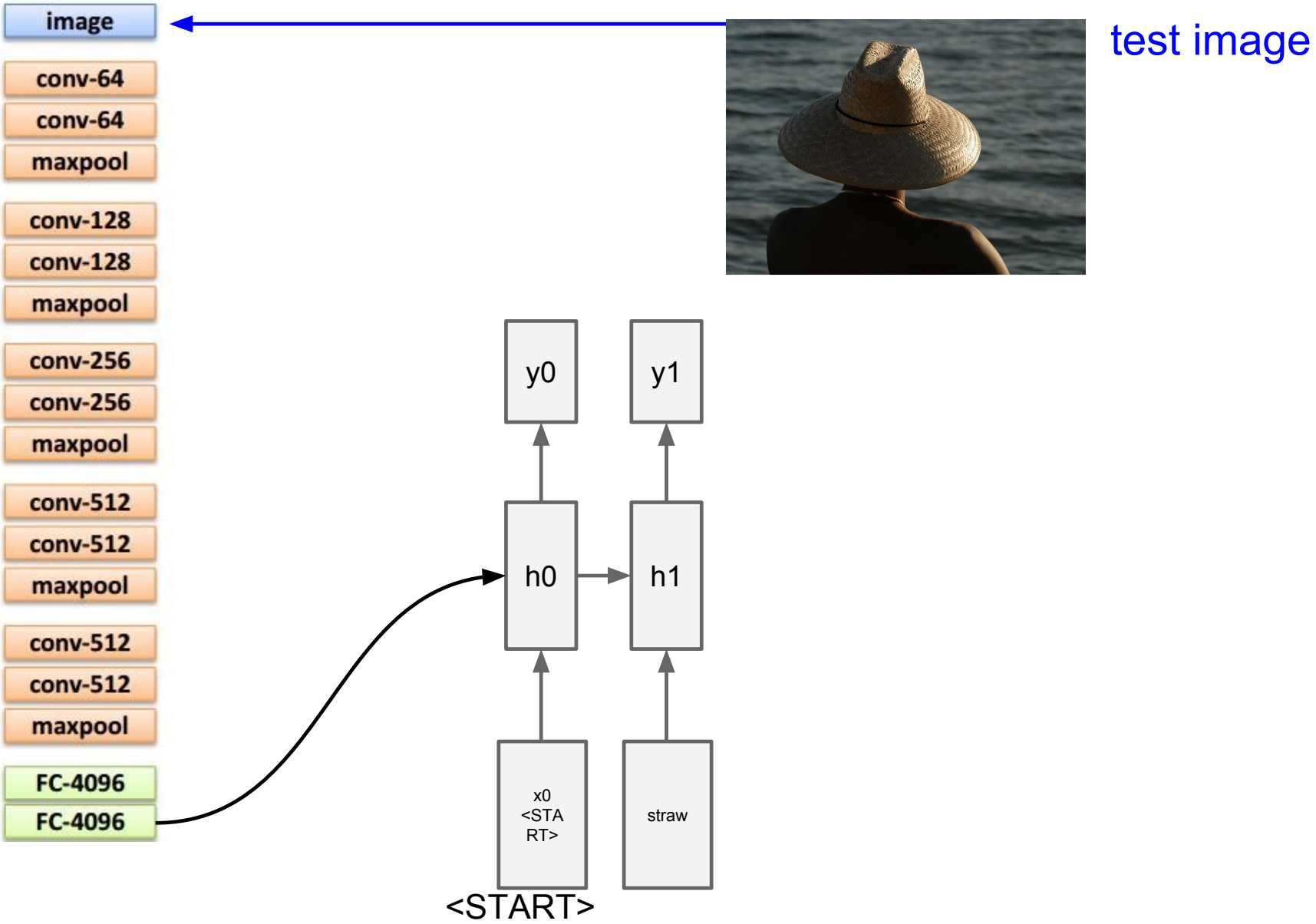


Image Captioning

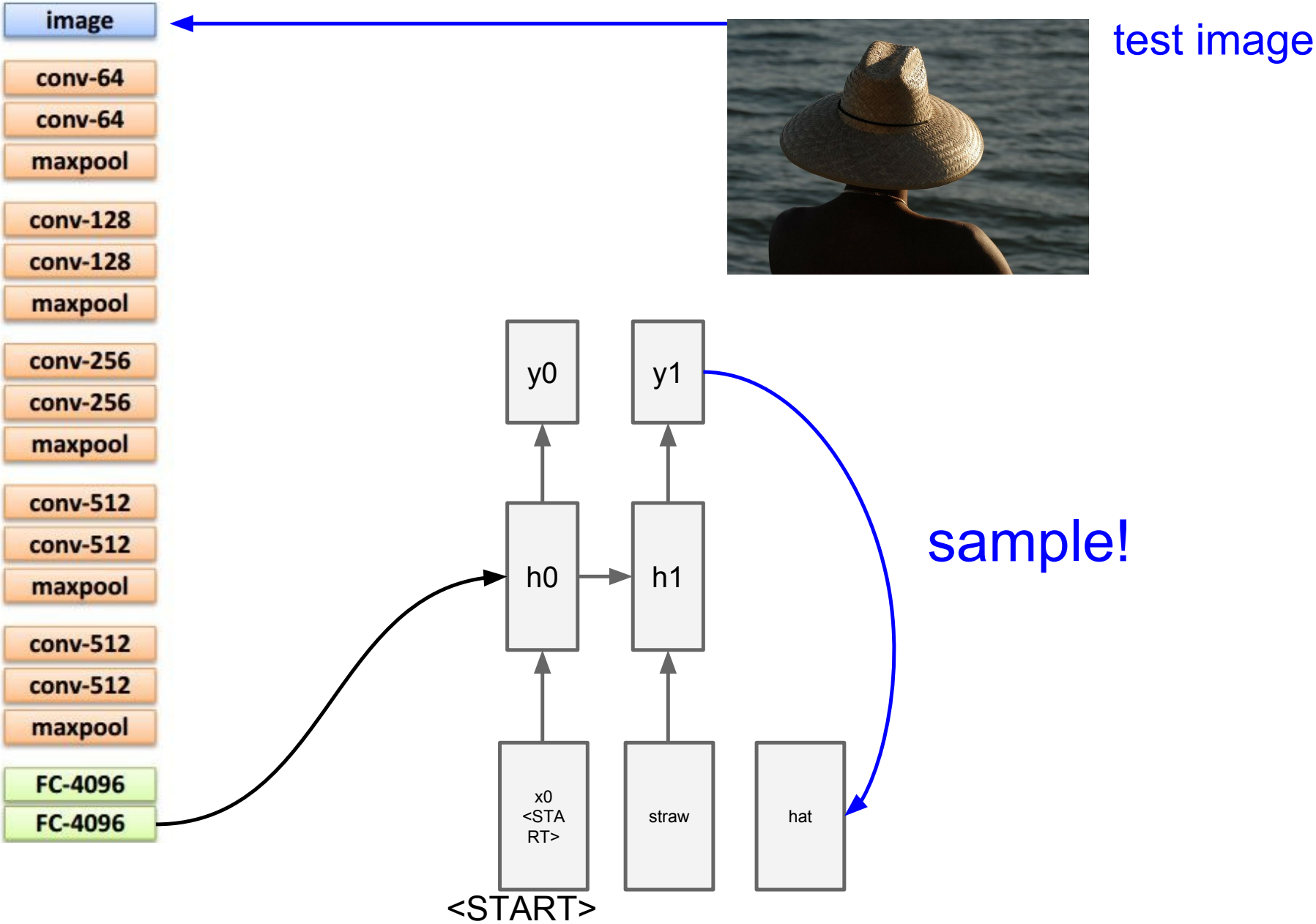


Image Captioning

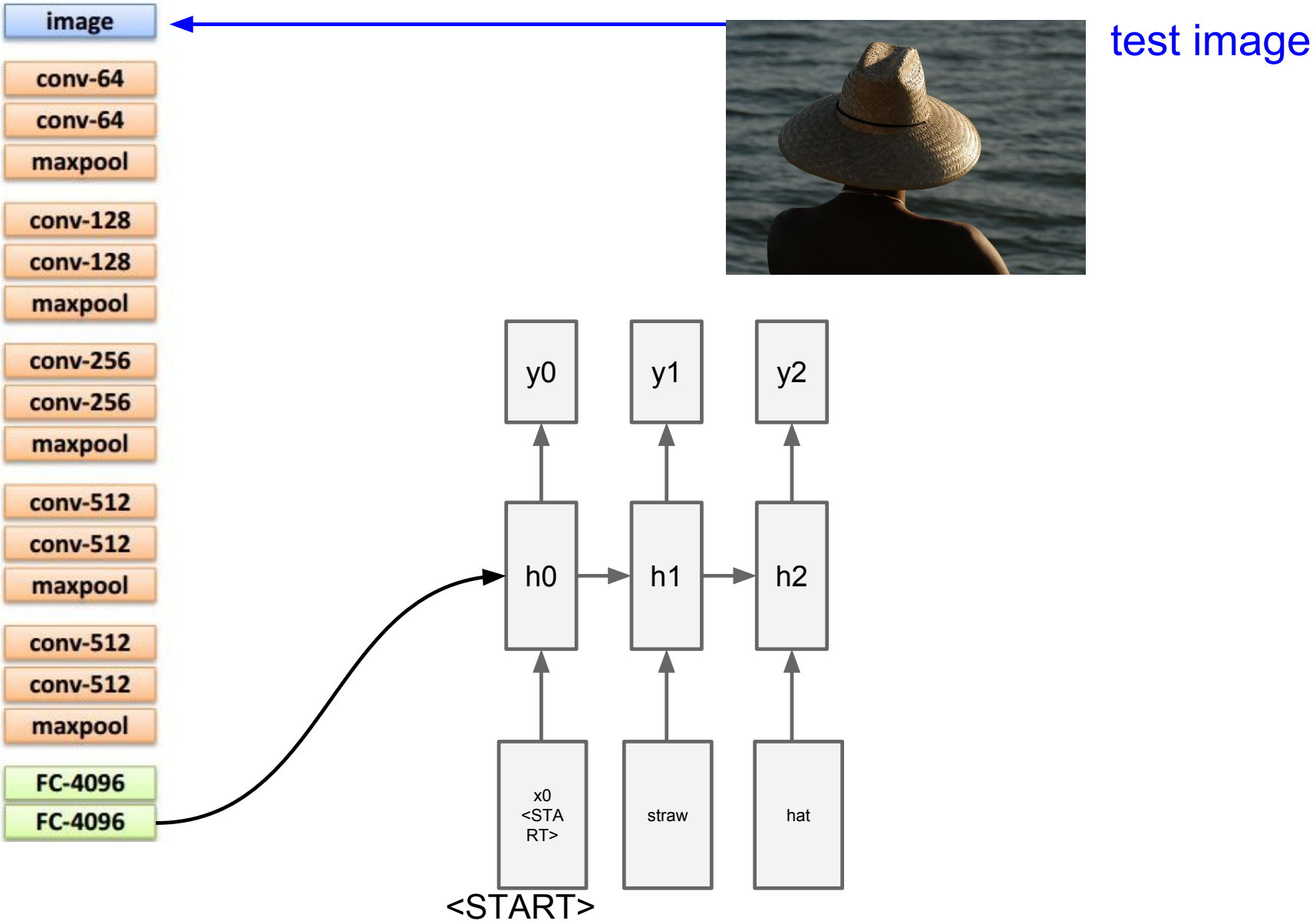


Image Captioning

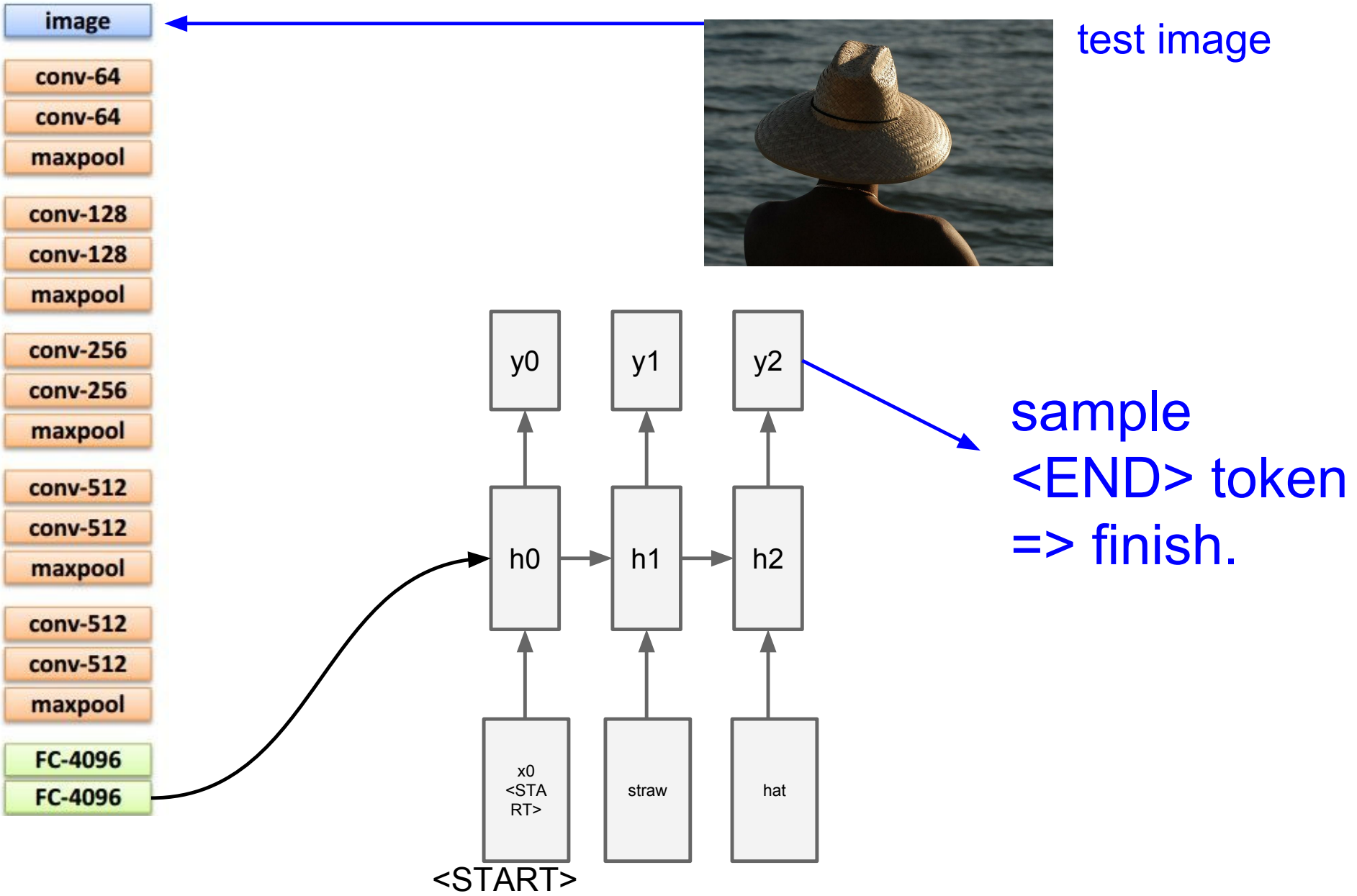
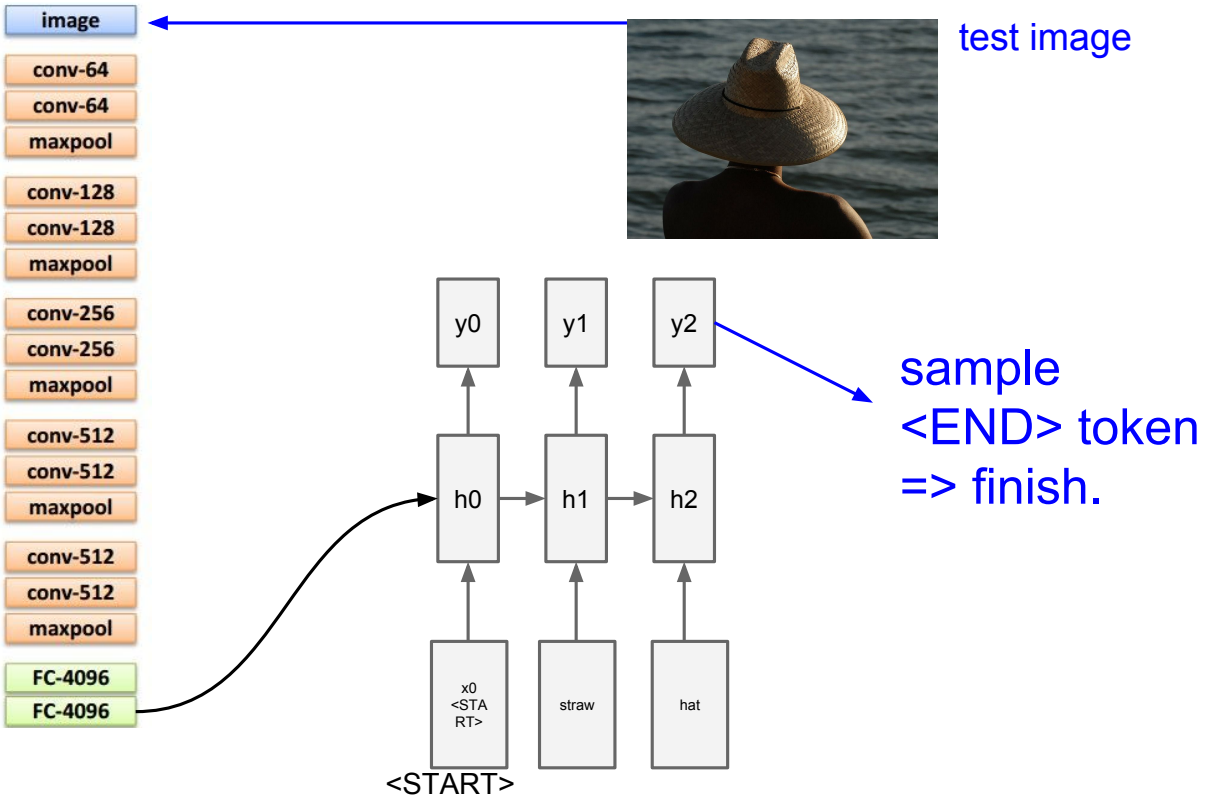


Image Captioning



Some modifications from Vinyals *et al.* 2014

- Only input image features in the first time step of an RNN.
- Use word embedding.
- Learn a feature embedding for CNN features.

- Typically fix the CNN parameters when training.

Image Captioning

Training

- Cross entropy loss for all time steps of all training image-caption pairs (I_i, S_i):

$$L(I, S) = -\sum_i \sum_t \log p_t(S_{i,t})$$

Inference

- **Sampling**: sample the first word according to p_1, then provide the corresponding embedding as input and sample p_2, continuing like this until we sample the special end-of-sentence token or some maximum length.
- **Beam Search**: iteratively consider the set of the k best sentences up to time t as candidates to generate sentences of size $t + 1$, and keep only the resulting best k of them:

  Better approximates $S = \arg\max_{S'} p(S'|I)$.

Image Captioning: Example Results

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

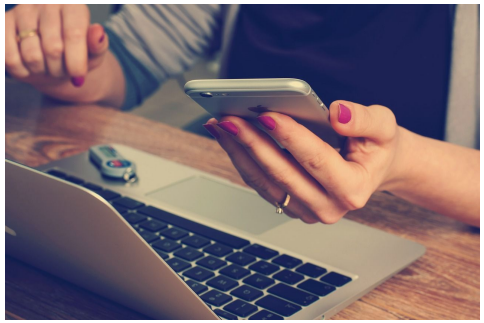
A tennis player in action on the court

Two giraffes standing in a grassy field

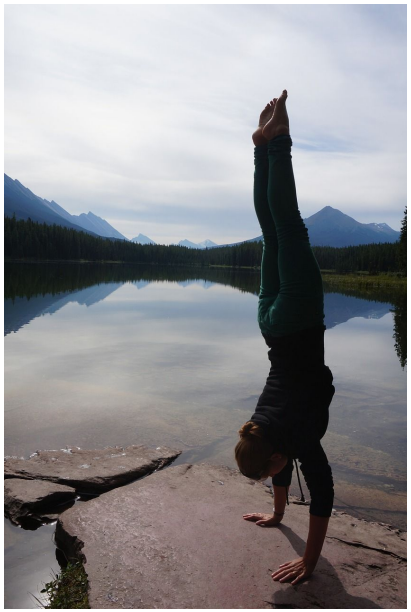
A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

A woman is holding a cat in her hand



A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

What is Attention?

- Learning to accumulate information from previous inputs.
- How to do accumulation?
  - Averaging
  - Maximum
  - Attention

What is Attention[4]?

- What is the color of the soccer ball?
- Which Georgetown player, the guys in white, is wearing the captaincy band?

[4] https://medium.com/@prakhargannu/attention-mechanism-in-deep-learning-simplified-d6a5830a079d

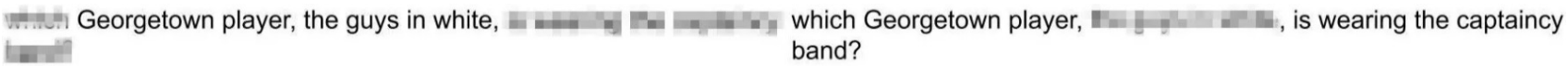
What is Attention[5]?

- What is the color of the soccer ball?
- Which Georgetown player, the guys in white, is wearing the captaincy band?

[5] https://medium.com/@prakhargannu/attention-mechanism-in-deep-learning-simplified-d6a5830a079d

What is Attention[6]?

- Also when you were reading the sentence above, did your mind start associating different words together, ignoring certain phrases at times to simplify the meaning?



[6] https://medium.com/@prakhargannu/attention-mechanism-in-deep-learning-simplified-d6a5830a079d

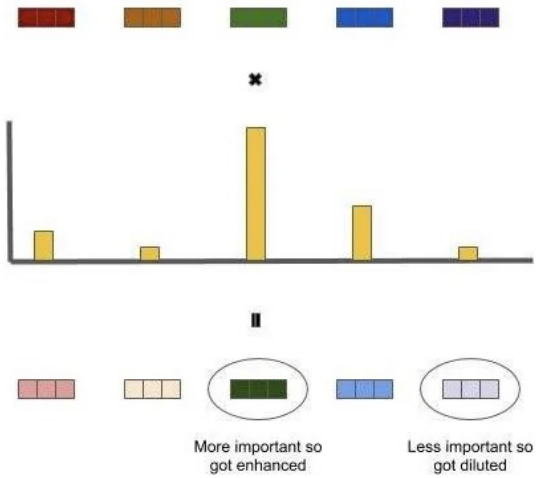
How does Attention Work[7]?

- You were 'focusing' on a smaller part of the whole thing because you knew the rest of the image/sentence was not useful to you at that particular moment.
- So when you were trying to figure out the color of the soccer ball, your mind was showing you the soccer ball in HD but the rest of the image was almost blurred.
- In deep learning, researchers attempts to borrow inspiration from how a human mind works with the attention mechanism:
  - Attention mechanism is just a way of focusing on only a smaller part of the complete input while ignoring the rest.

[7] https://medium.com/@prakhargannu/attention-mechanism-in-deep-learning-simplified-d6a5830a079d

What Happened[8]?

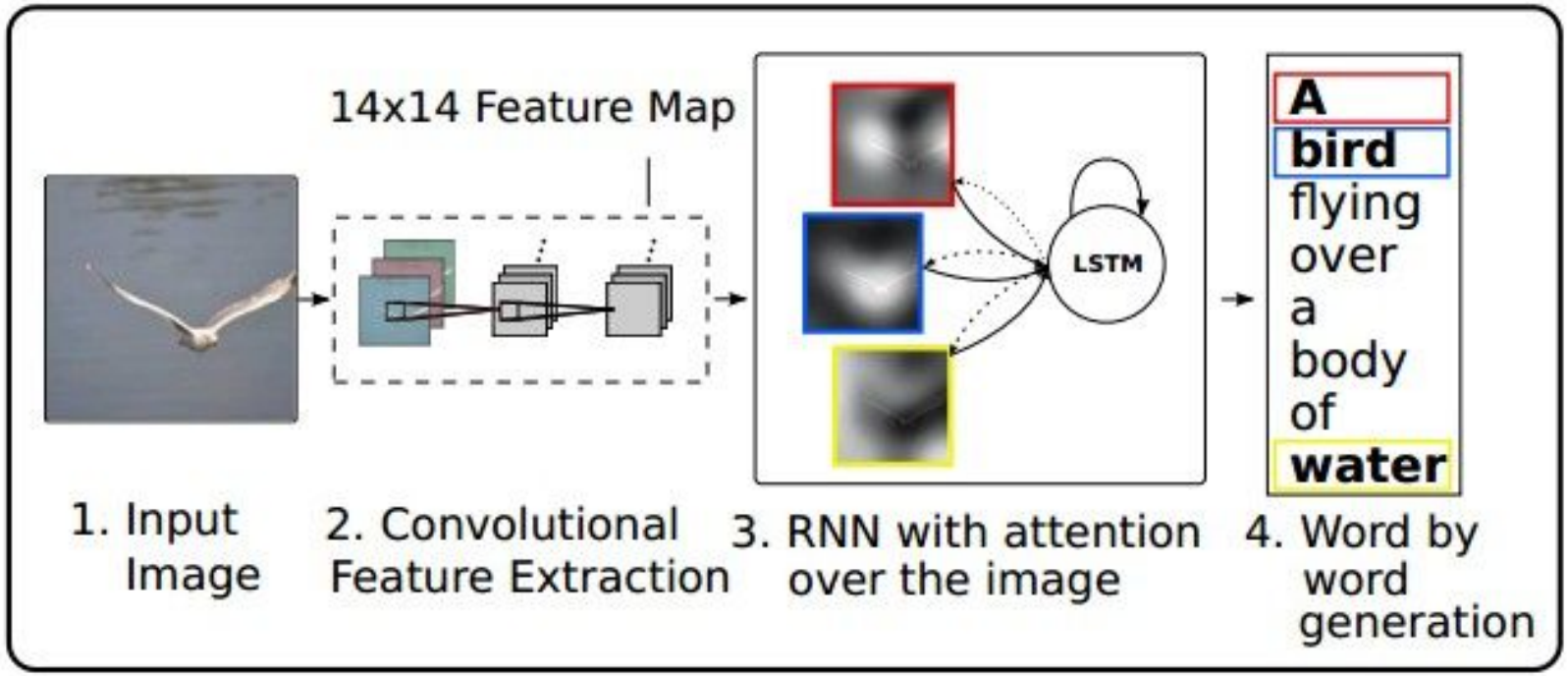
- Create a probability distribution that rates the importance of various input elements. These input representations can be words, pixels, vectors etc. This is a learnable task.
- Scale the original input using this probability distribution such that values that deserve more attention gets enhanced while others get diluted. Like blurring everything else that does not need attention.
- Now use these newly scaled inputs and do further processing to get focused outputs/results.



[8] https://medium.com/@prakhargannu/attention-mechanism-in-deep-learning-simplified-d6a5830a079d

Image Captioning with Attention

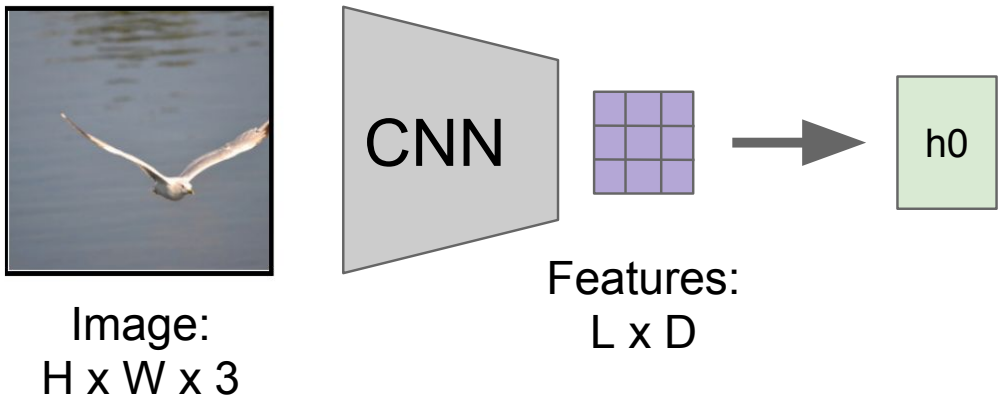
RNN focuses its attention at a different spatial location when generating each word



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

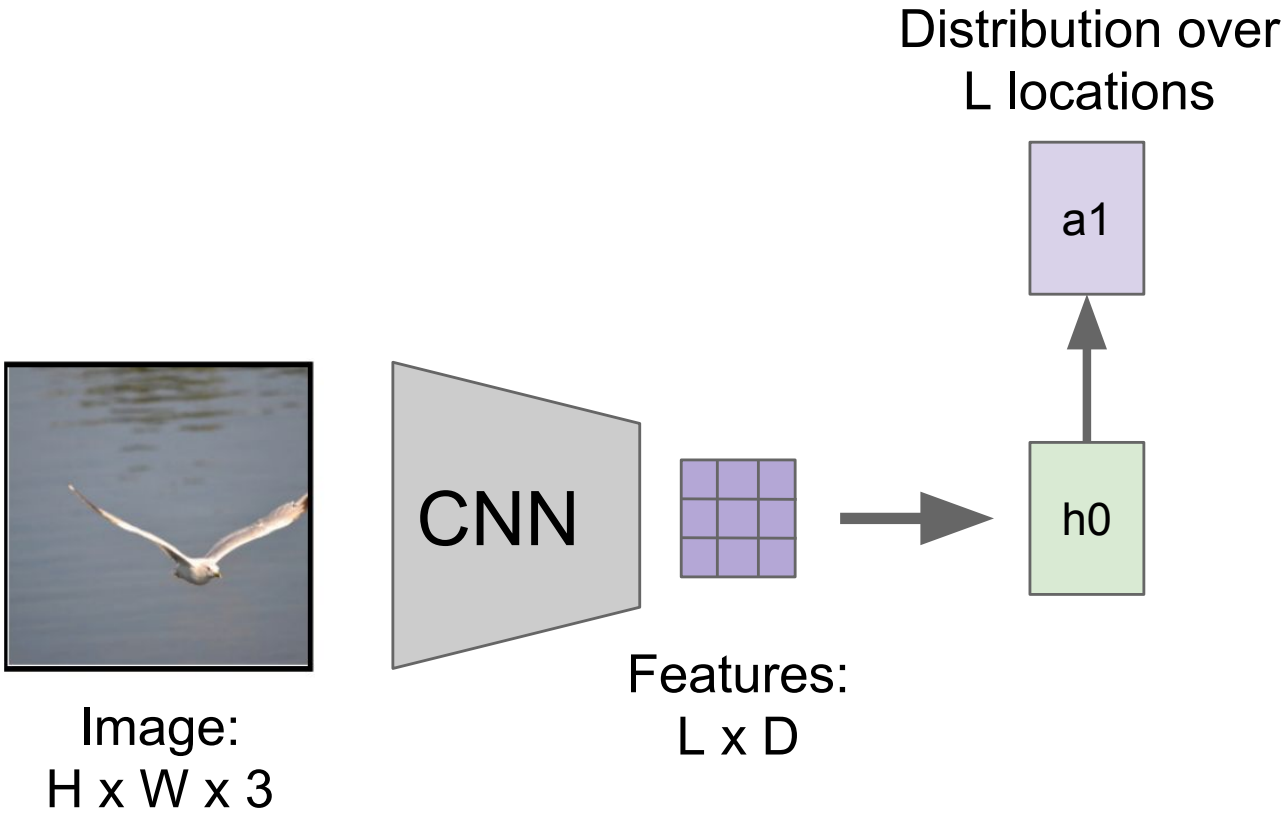
Attention: Learning to weight input elements for each output unit!

Image Captioning with Attention



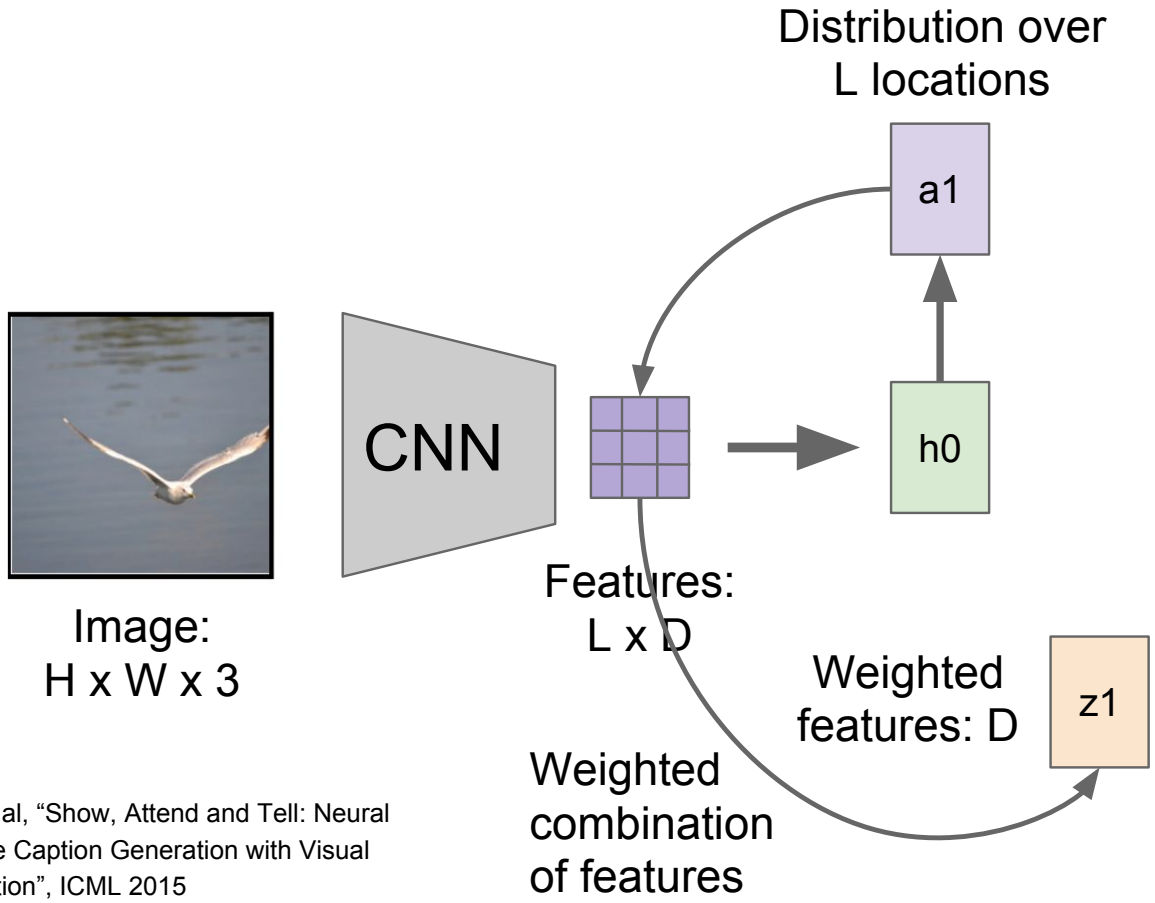
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



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

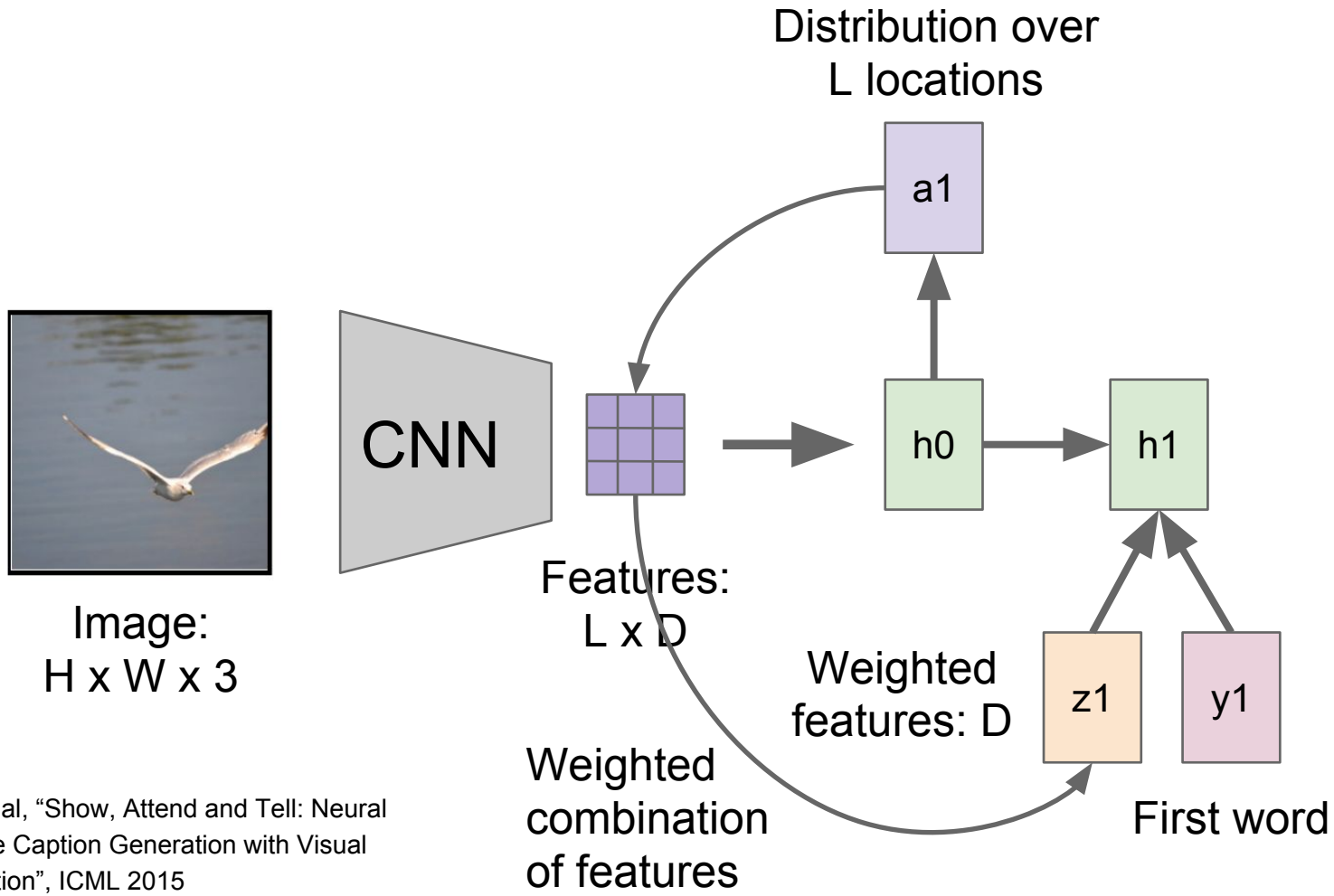
Image Captioning with Attention



$$z = \sum_{i=1}^{L} p_i v_i$$

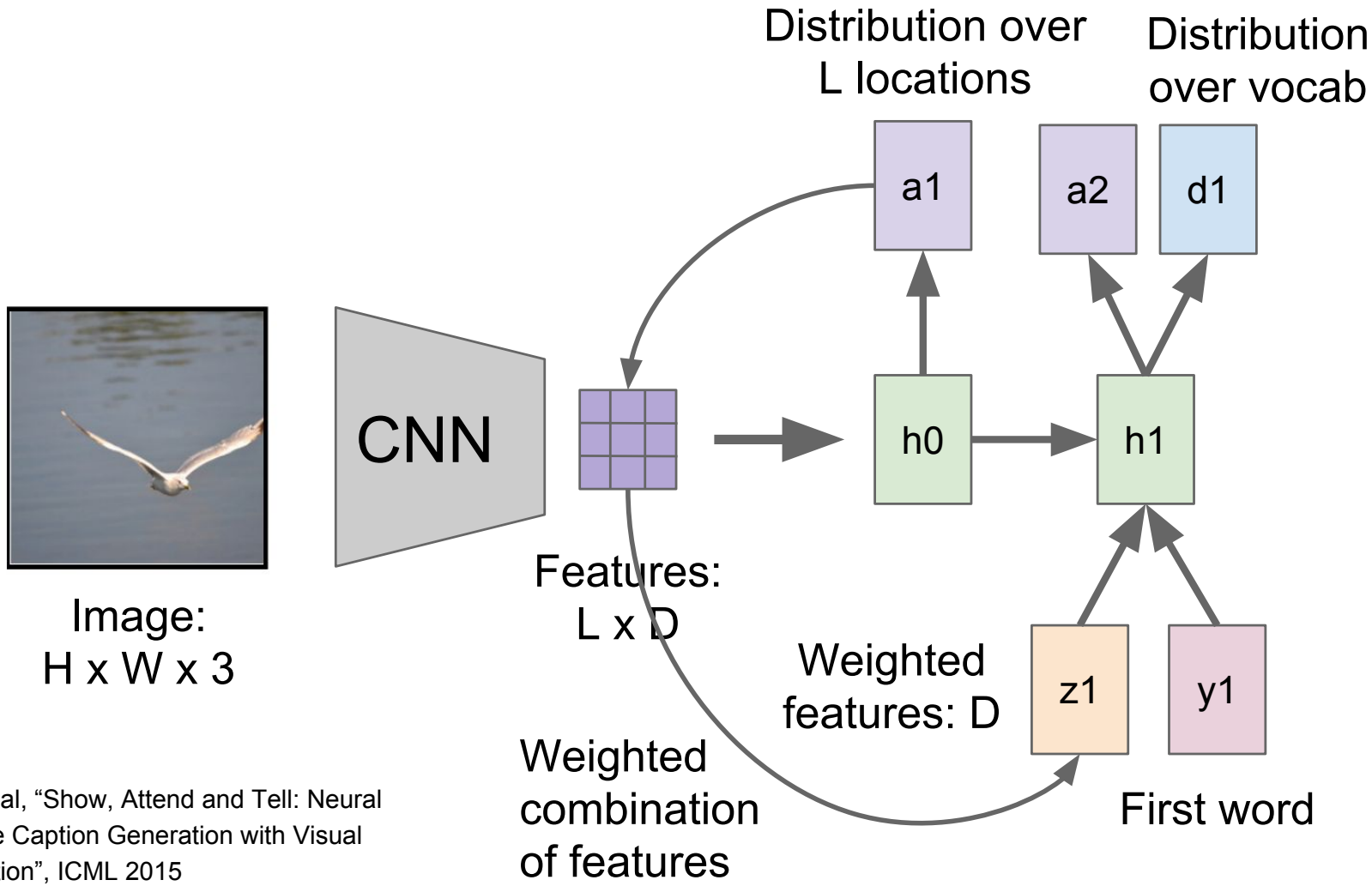
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



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

Image Captioning with Attention



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

Image Captioning with Attention

Image:
H x W x 3

CNN

Features:
L x D

Distribution over
L locations

Distribution
over vocab

a1

a2

d1

h0

h1

h2

Weighted
features: D

z1

y1

z2

y2

Weighted
combination
of features

First word

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

Image Captioning with Attention

Image:
H x W x 3

CNN

Features:
L x D

Distribution over
L locations

Distribution
over vocab

a1 a2 d1 a3 d2

h0 h1 h2

z1 y1 z2 y2

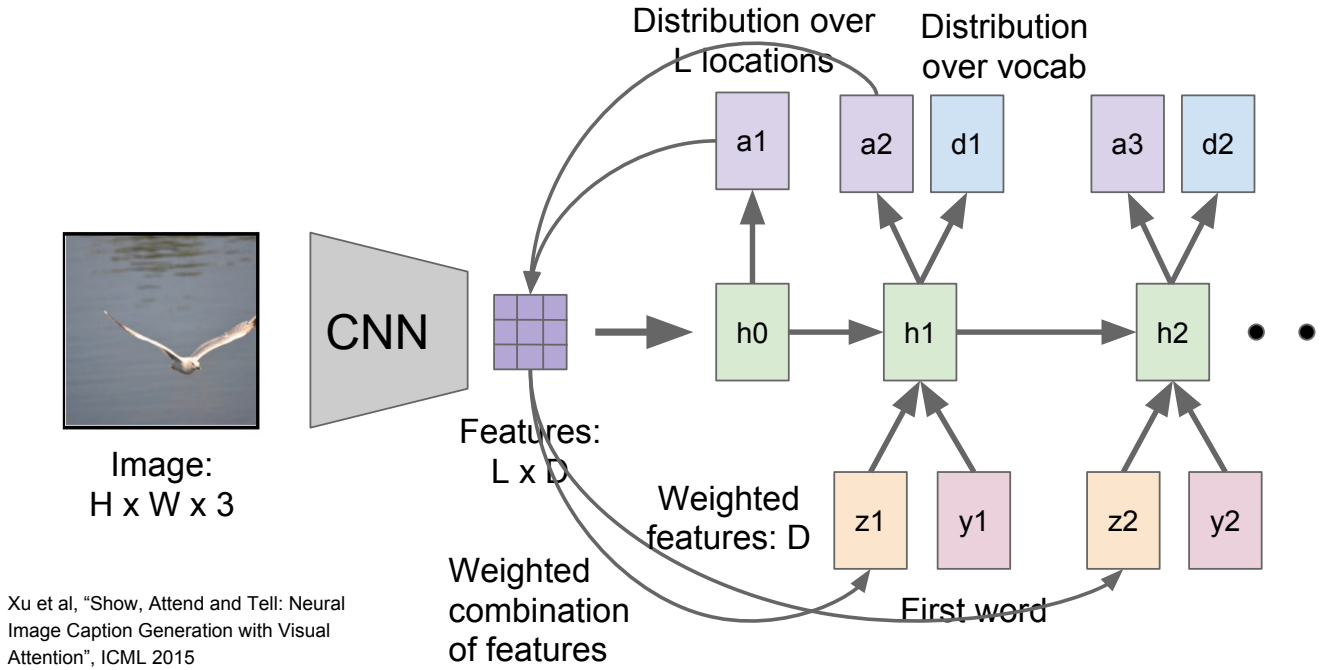
Weighted
features: D

Weighted
combination
of features

First word

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

Image Captioning with Attention



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

$$\begin{bmatrix} \mathbf{i}_t \\ \mathbf{f}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \mathbf{T} \begin{bmatrix} \mathbf{E}\,\mathbf{y}_{t-1} \\ \mathbf{h}_{t-1} \\ \mathbf{z}_t \end{bmatrix}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Image Captioning with Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Visual Question Answering

Q: What endangered animal is featured on the truck?

A: **A bald eagle.**
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?

A: **Onto 24 ¾ Rd.**
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

A: **During a wedding.**
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

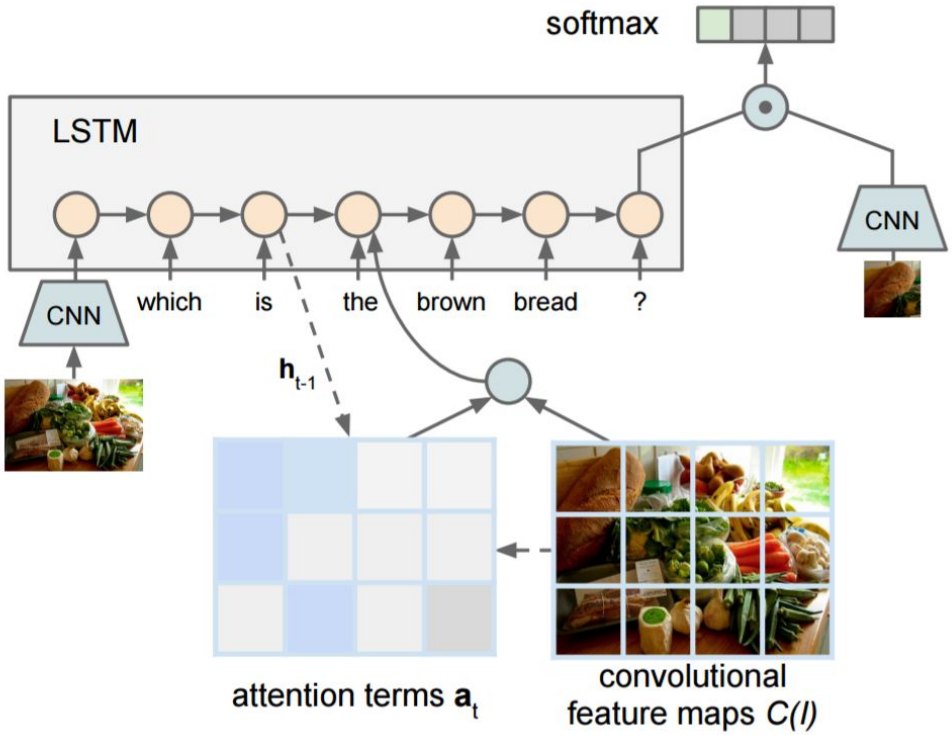
Q: Who is under the umbrella?

A: **Two women.**
A: A child.
A: An old man.
A: A husband and a wife.

Agrawal et al, “VQA: Visual Question Answering”, ICCV 2015
Zhu et al, “Visual 7W: Grounded Question Answering in Images”, CVPR 2016

Visual Question Answering: RNNs with Attention

- The model first reads the image and all the question tokens until reaching the end token of the question sequence.
- In training, continue feeding the ground-truth answer tokens.
- Compute the log-likelihood of an candidate region by a dot product between its transformed visual feature from CNN and the last LSTM hidden state.



Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

What kind of animal is in the photo?

A **cat**.

Why is the person holding a knife?

To cut the **cake** with.

Extending RNNs: Multilayer RNNs

$$\mathbf{h} \in \mathbb{R}^n$$

Vanilla RNN: $\mathbf{W}^{(\ell)} \in \mathbb{R}^{n \times 2n}$

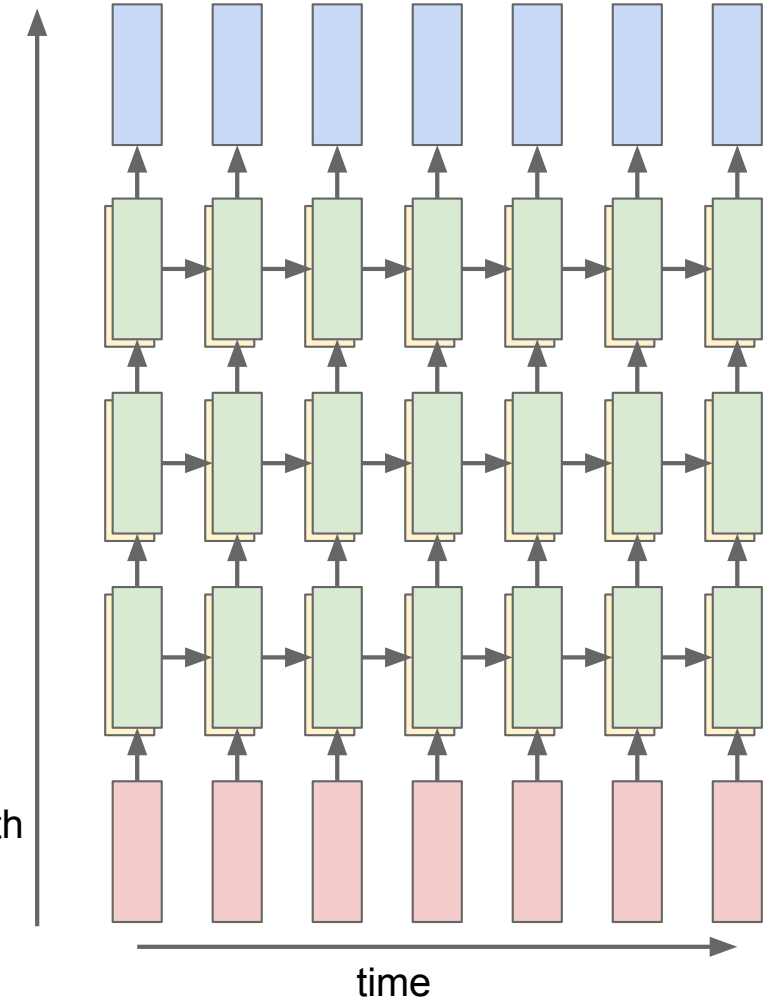
$$\mathbf{h}_t^{(\ell)} = \tanh \mathbf{W}^{(\ell)} \begin{bmatrix} \mathbf{h}_t^{(\ell-1)} \\ \mathbf{h}_{t-1}^{(\ell)} \end{bmatrix}$$

LSTM: $\mathbf{W}^{(\ell)} \in \mathbb{R}^{4n \times 2n}$

$$\begin{pmatrix} \mathbf{i}^{(\ell)} \\ \mathbf{f}^{(\ell)} \\ \mathbf{o}^{(\ell)} \\ \mathbf{g}^{(\ell)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \left[\mathbf{W}^{(\ell)} \begin{pmatrix} \mathbf{h}_t^{(} \\ \mathbf{h} \end{pmatrix} \right.$$

$$\mathbf{c}_t^{(\ell)} = \mathbf{f}^{(\ell)} \odot \mathbf{c}_{t-1}^{(\ell)} + \mathbf{i}^{(\ell)} \odot \mathbf{g}$$

$$\mathbf{h}_t^{(\ell)} = \mathbf{o}^{(\ell)} \odot \tanh(\mathbf{c}_t^{(\ell)})$$



Google Translation System