Recurrent Neural Networks (RNN), Long-Short-Term-Memory (LSTM), Attention and Transformer

Yuan YAO

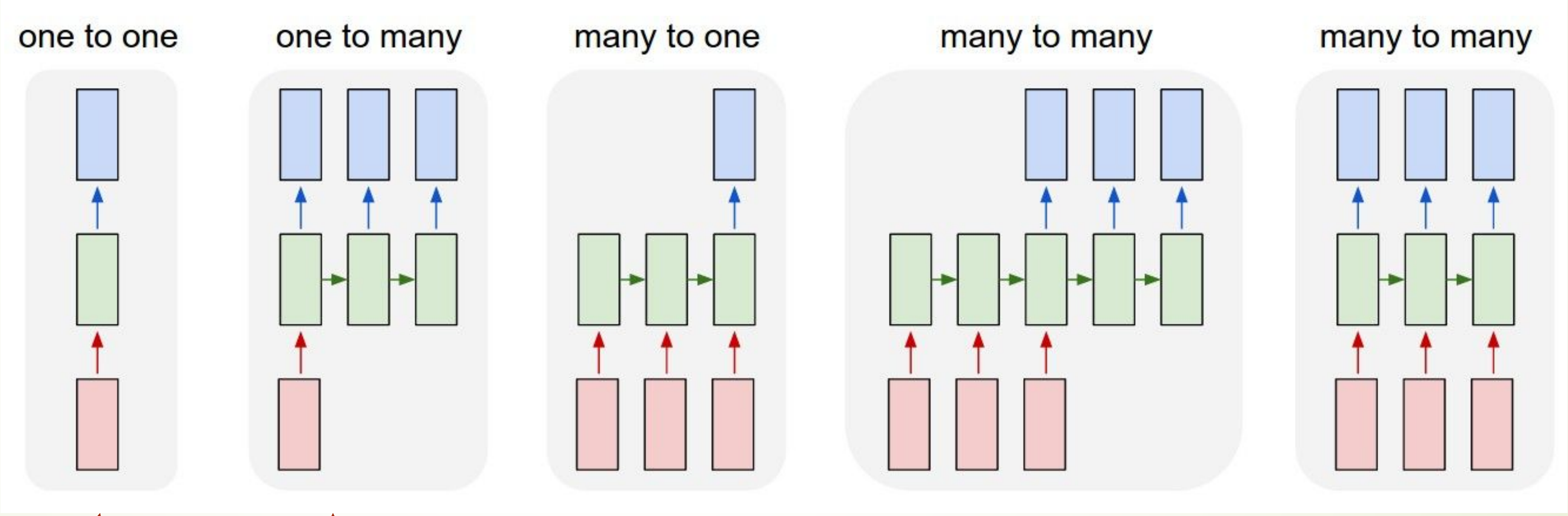
HKUST

Summary

- We have shown:
  - CNN Architectures: LeNet5, Alexnet, VGG, GoogleNet, Resnet
- Today:
  - **Recurrent Neural Networks**
  - **LSTM/GRU**
  - **Attention**
  - **Transformer**
- Reference:
  - Feifei Li, Stanford cs231n
  - Chris Manning, Stanford cs224n

Recurrent Neural Networks

Recurrent Neural Networks: Process Sequences



Vanilla Neural Network

Image Captioning

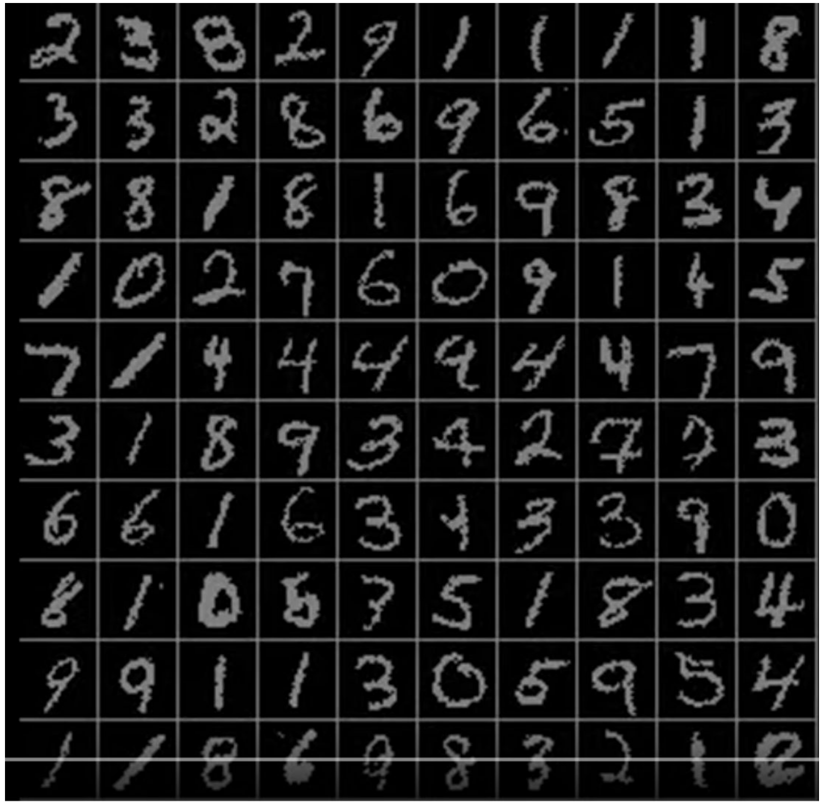
Sentiment Classification

Machine Translation, Dialogue

Video frame-based classification/annotation

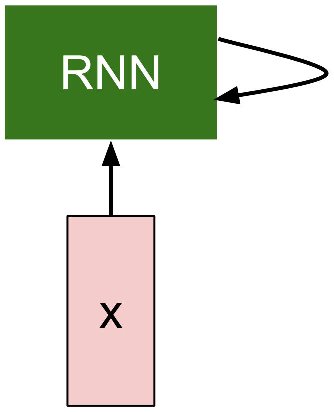
Sequential Processing of Non-Sequence Data

Classify images by taking a series of “glimpses”

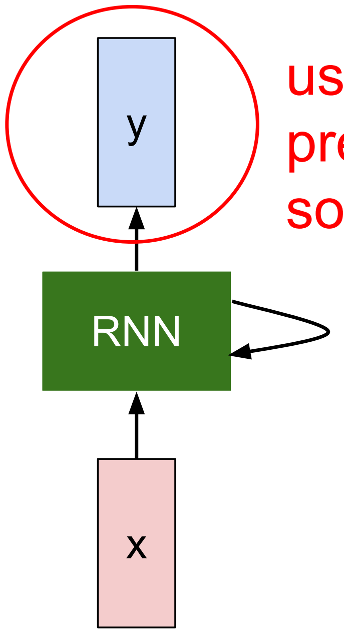


Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

Recurrent Neural Network



Recurrent Neural Network



usually want to predict a vector at some time steps

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

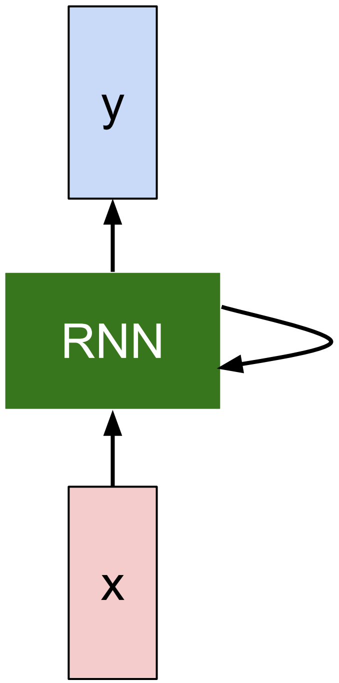
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

old state

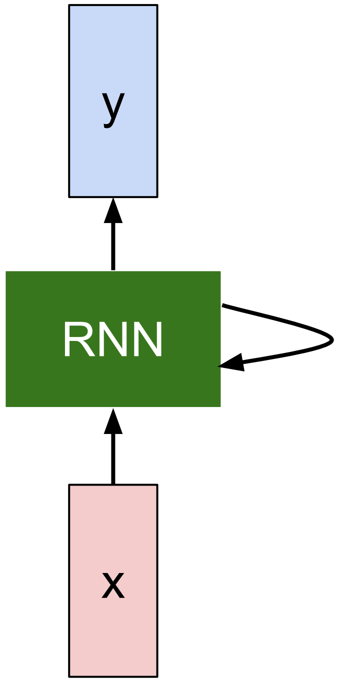
input vector at some time step

some function with parameters W



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

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

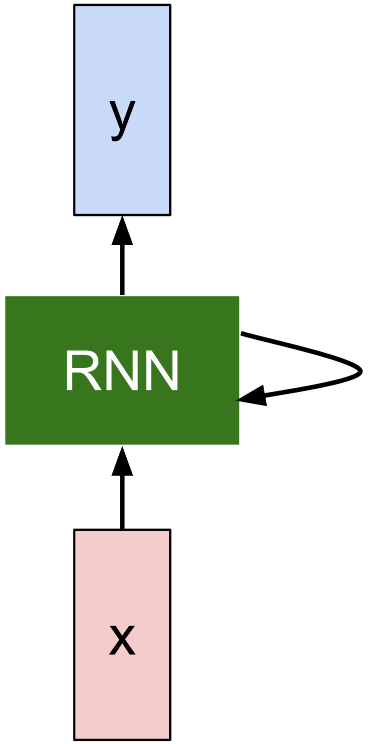


Notice: the same function and the same set of parameters are used at every time step.

Vanilla Recurrent Neural Networks

State Space equations in feedback dynamical systems

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



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

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

$$y_t = W_{hy}h_t$$

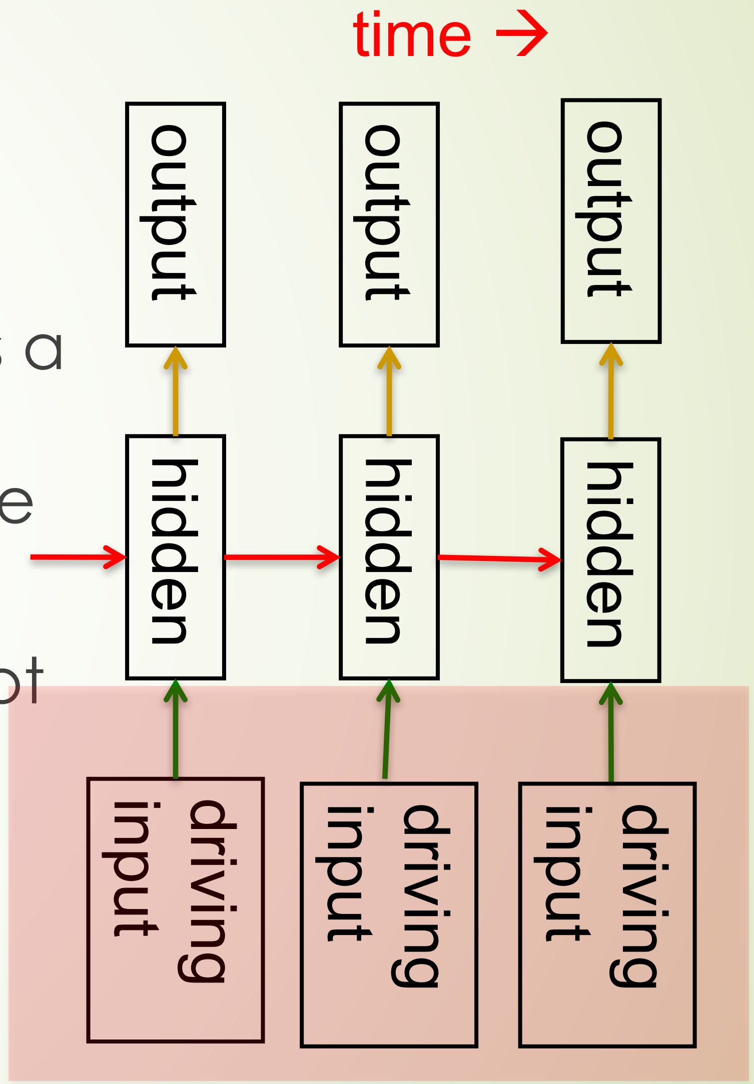
Or, $$y_t = \text{softmax}(W_{hy}h_t)$$

Linear Dynamical Systems (1940s-)

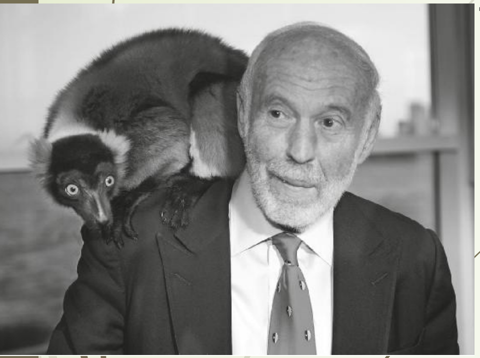
- The hidden state has linear dynamics with Gaussian noise and produces the observations using a linear model with Gaussian noise.
- Kalman Filter: A linearly transformed Gaussian is a Gaussian. So the distribution over the hidden state given the data so far is Gaussian. It can be computed using "Kalman filtering".
- To predict the next output (so that we can shoot down the missile) we need to infer the hidden state.

$$h_t = W_{hh}h_{t-1} + W_{hx}x_t + \epsilon_t^h$$

$$y_t = W_{yh}h_t + W_{yx}x_t + \epsilon_t^y$$

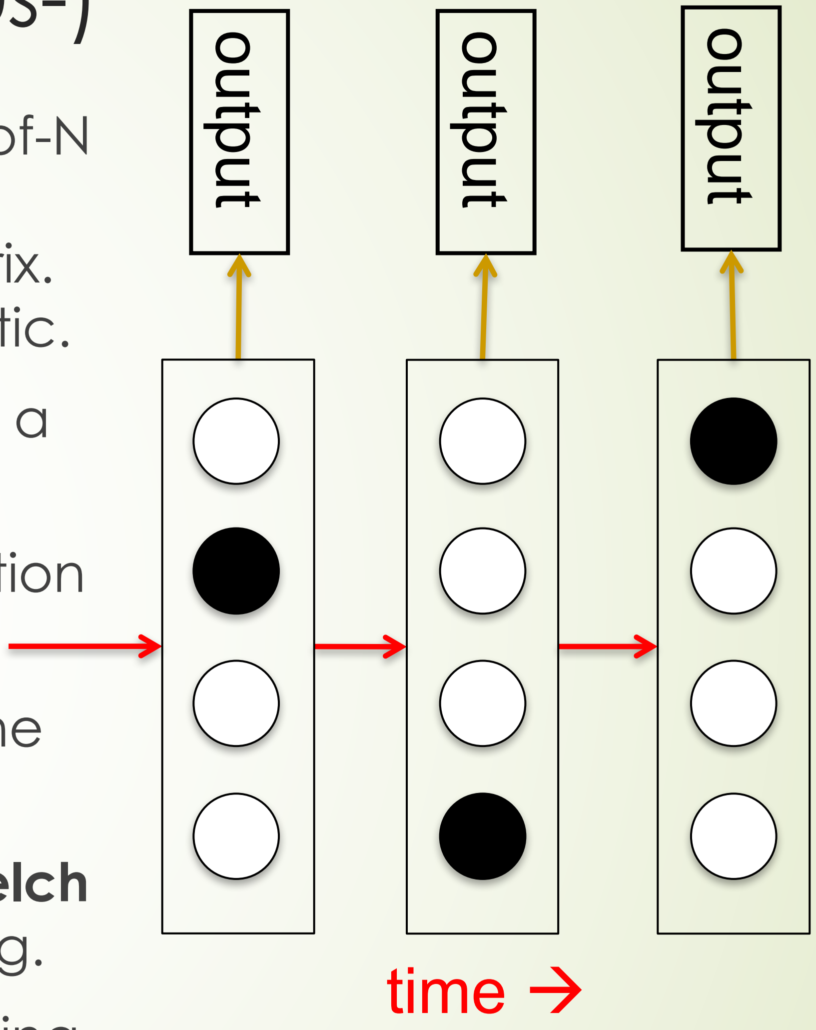


Hidden Markov Models (1970s-)



- Hidden Markov Models have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.
  - We cannot be sure which state produced a given output. So the state is "hidden".
  - It is easy to represent a probability distribution across N states with N numbers.
- To predict the next output we need to infer the probability distribution over hidden states.
  - HMMs have efficient algorithms (**Baum-Welch or EM Algorithm**) for inference and learning.
  - **Jim Simons** hires Lenny Baum as the founding member of Renaissance Technologies in 1979

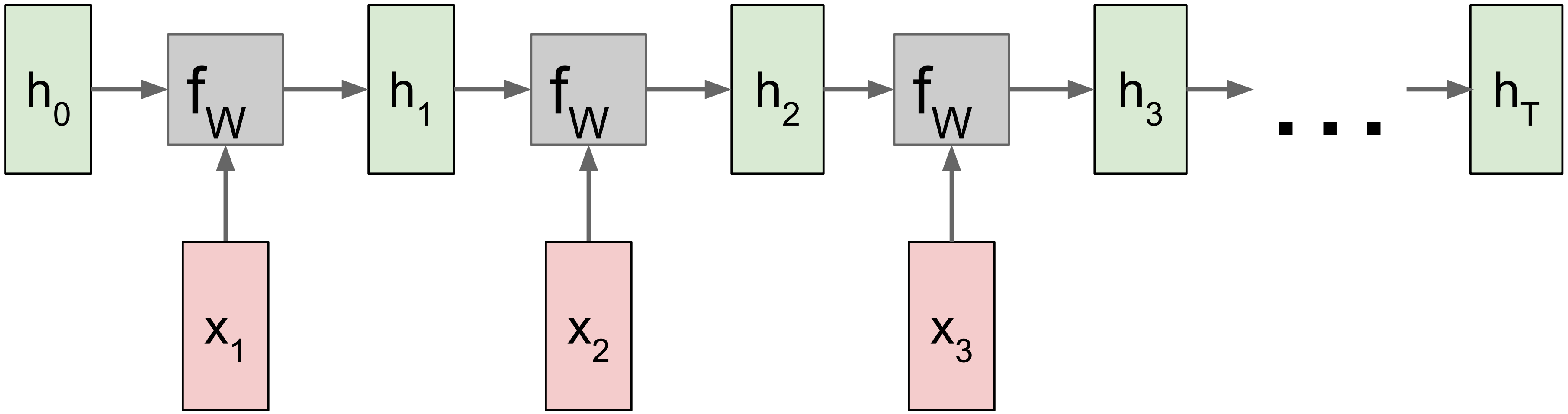
Lenny Baum became a devoted Go player despite his deteriorating eyesight.



Recurrent Neural Networks

- **The issue of a hidden Markov model (HMM)**:
  - At each time step it must select one of its hidden states. So with N hidden states it can only remember log(N) bits about what it generated so far.
- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.

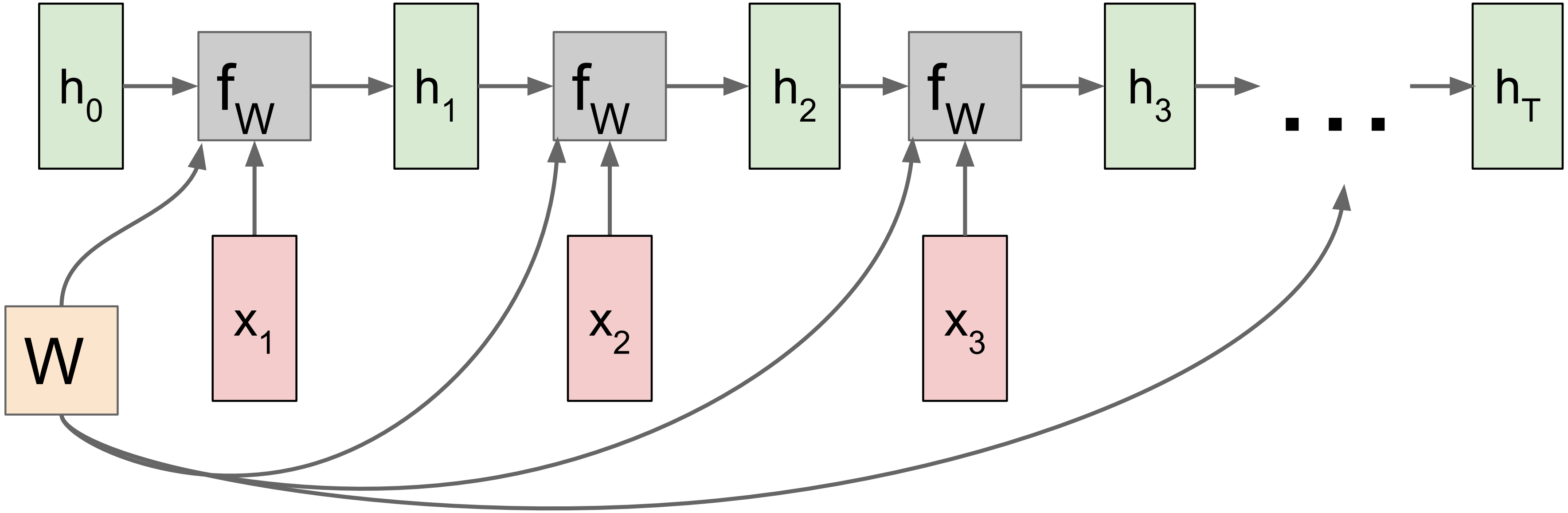
RNN: Computational Graph



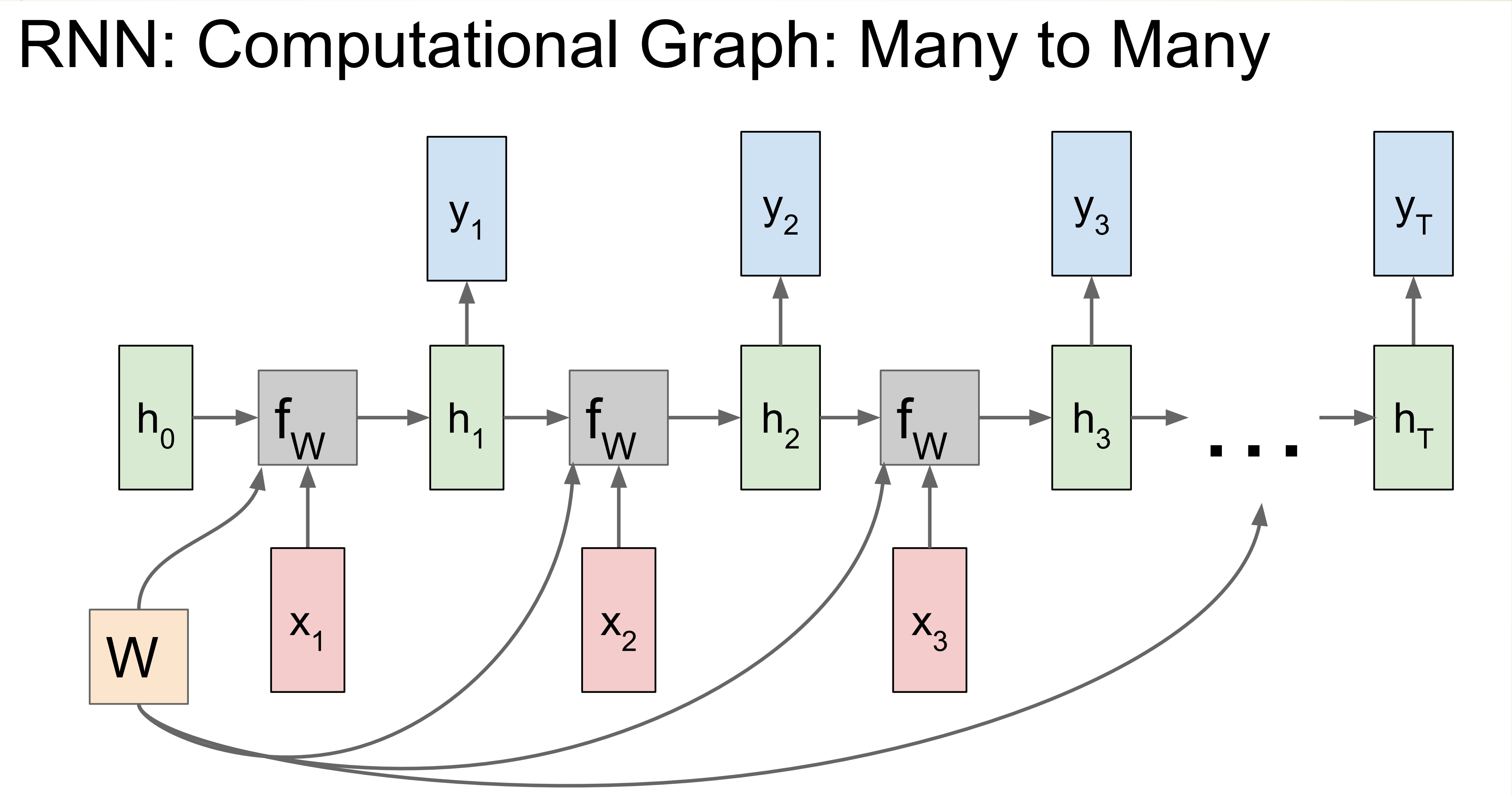
Time invariant systems

RNN: Computational Graph

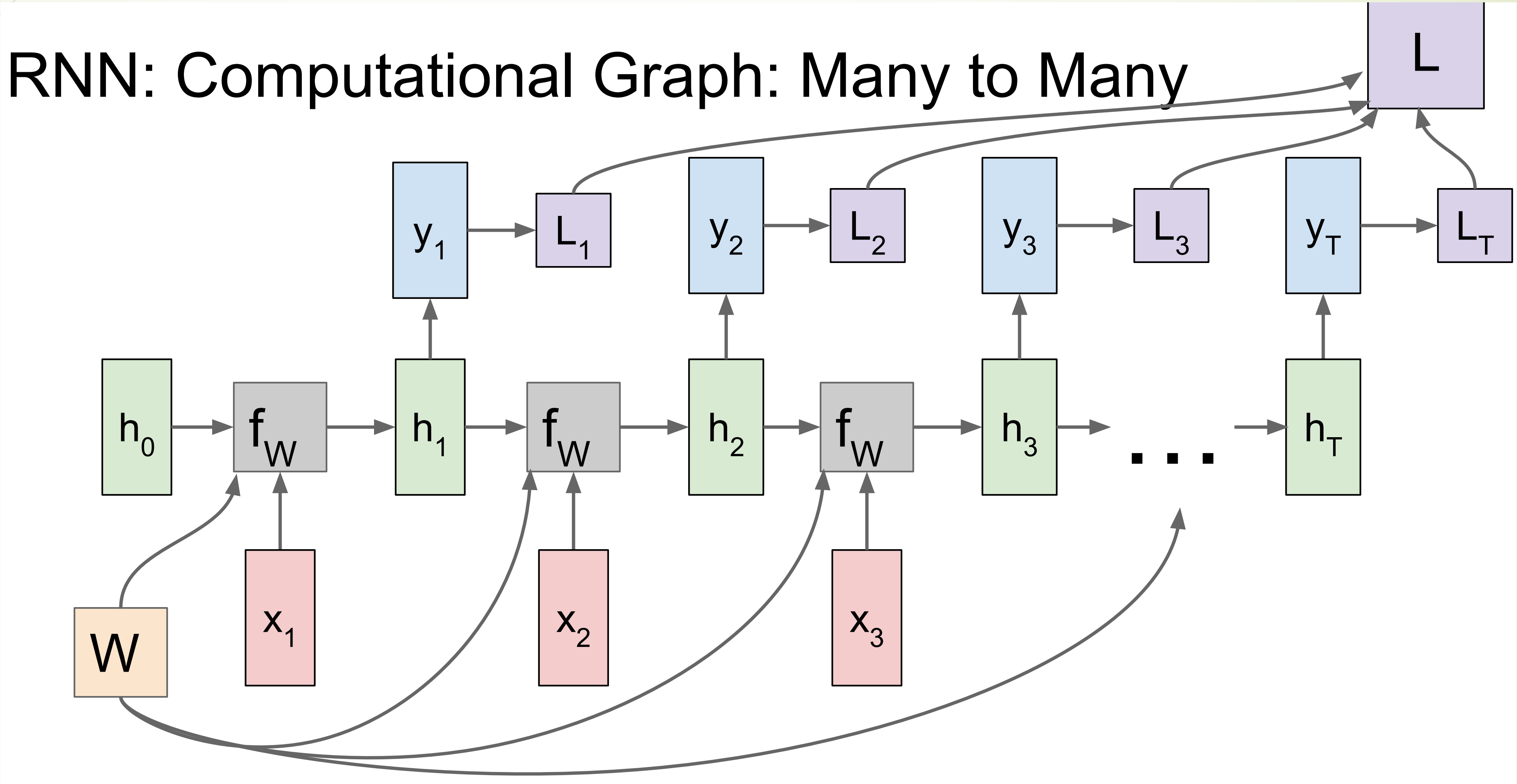
Re-use the same weight matrix at every time-step



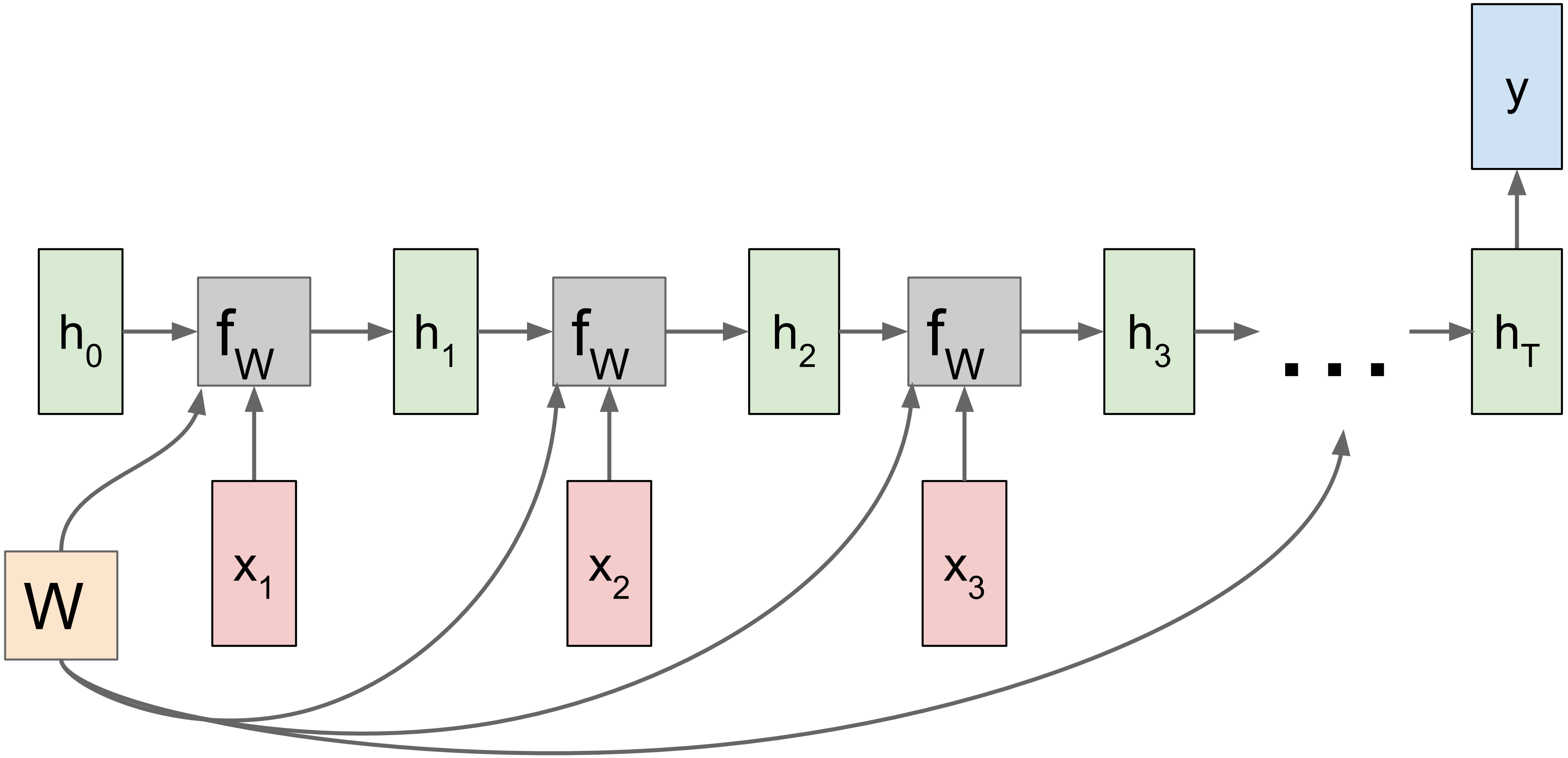
Outputs added



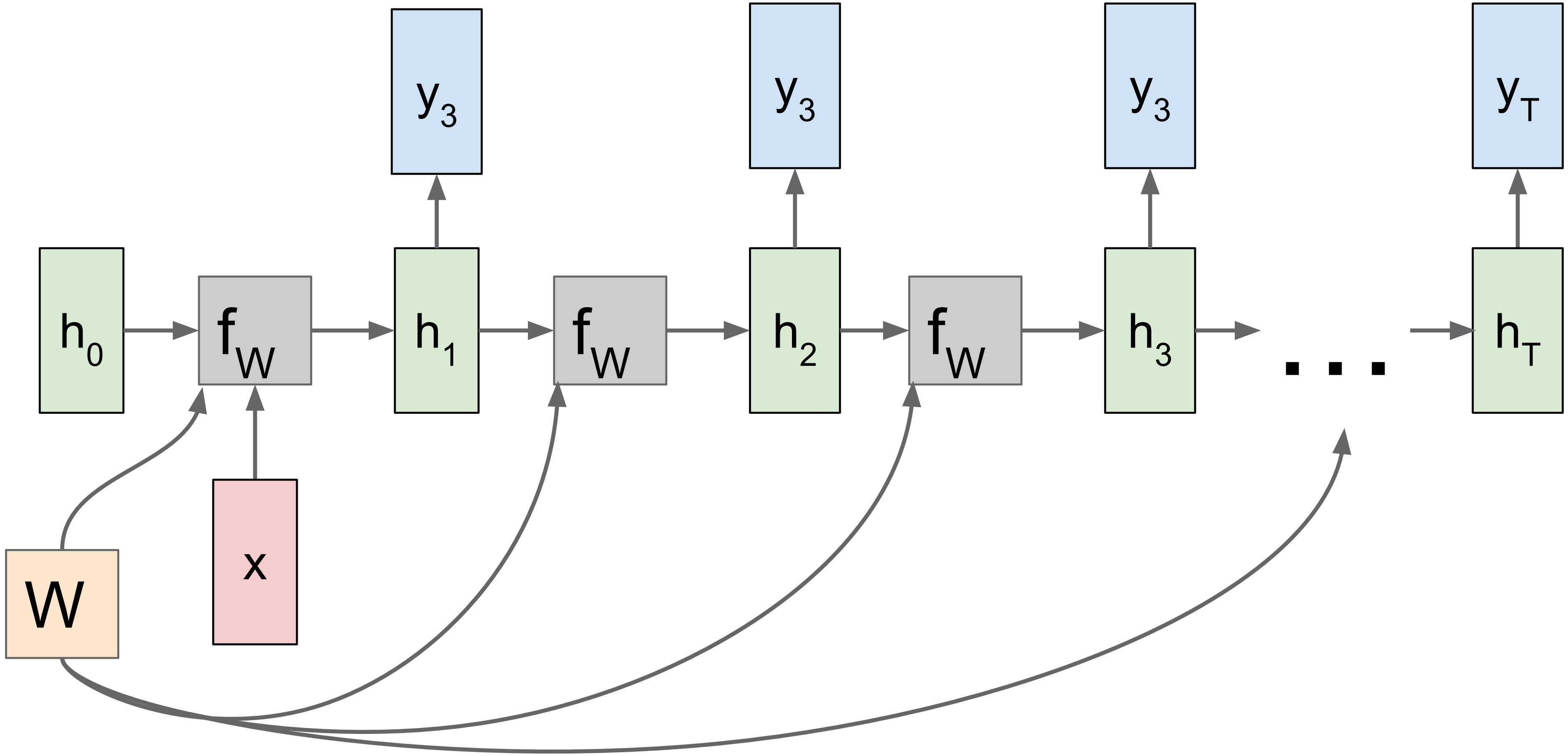
Loss modules



RNN: Computational Graph: Many to One



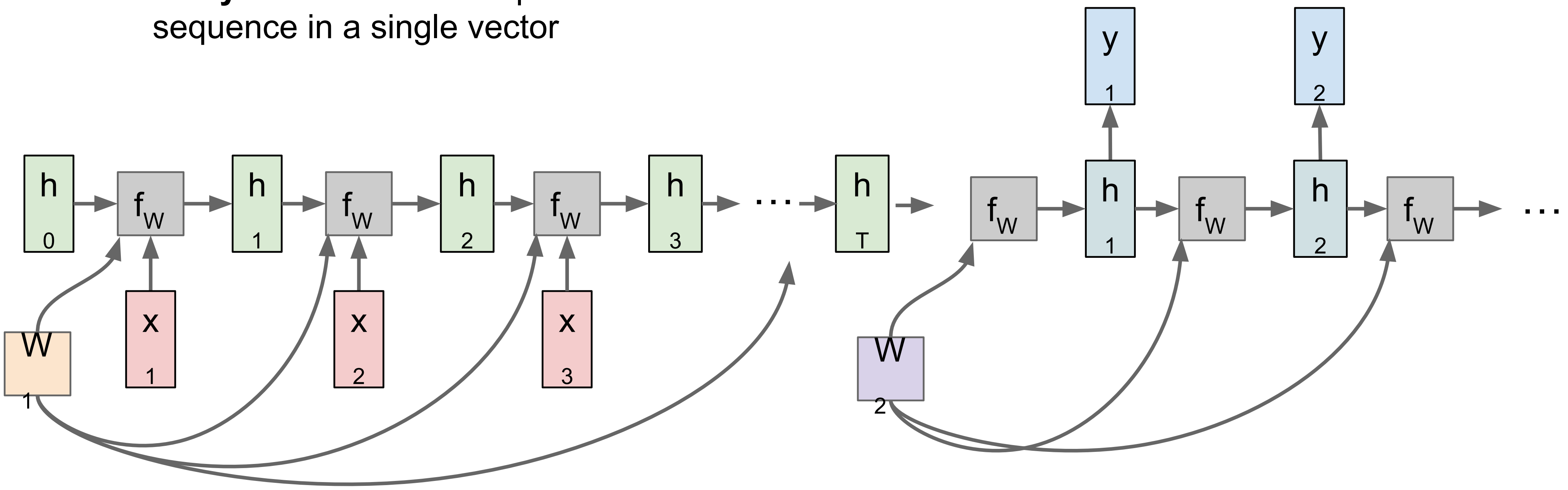
RNN: Computational Graph: One to Many



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

One to many: Produce output sequence from single input vector

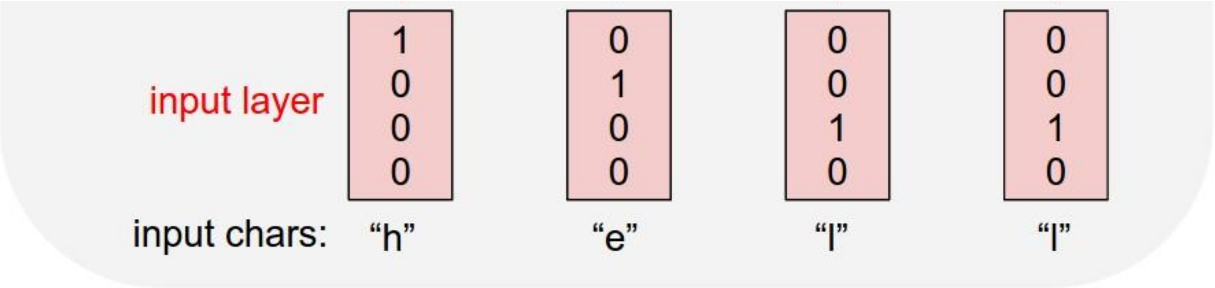
Many to one: Encode input sequence in a single vector



Example:
Character-level
Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

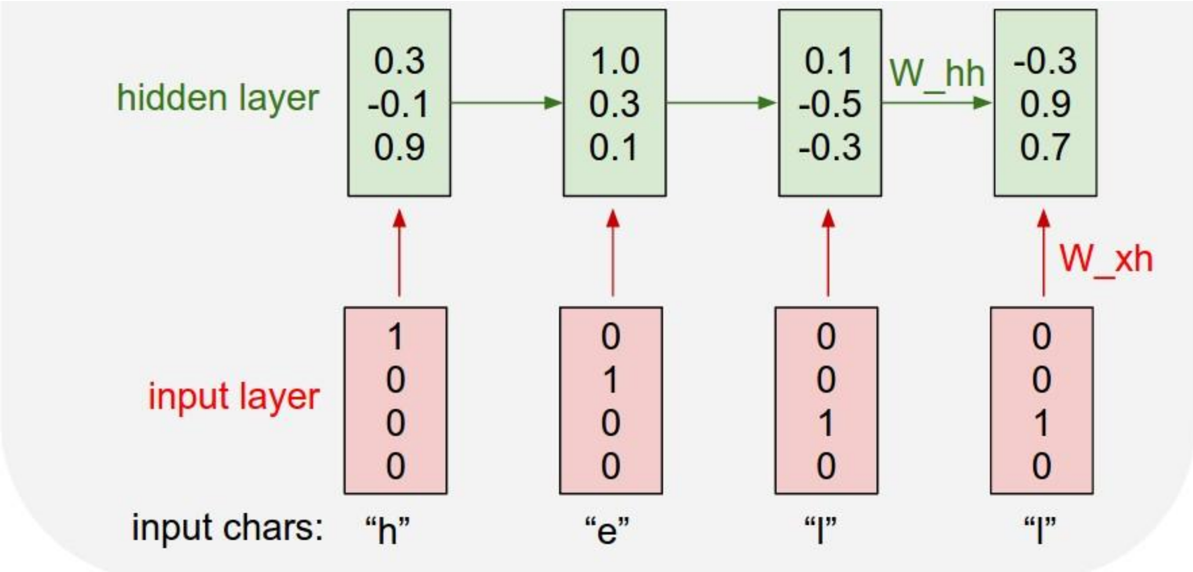


Example: Character-level Language Model

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

Vocabulary: [h,e,l,o]

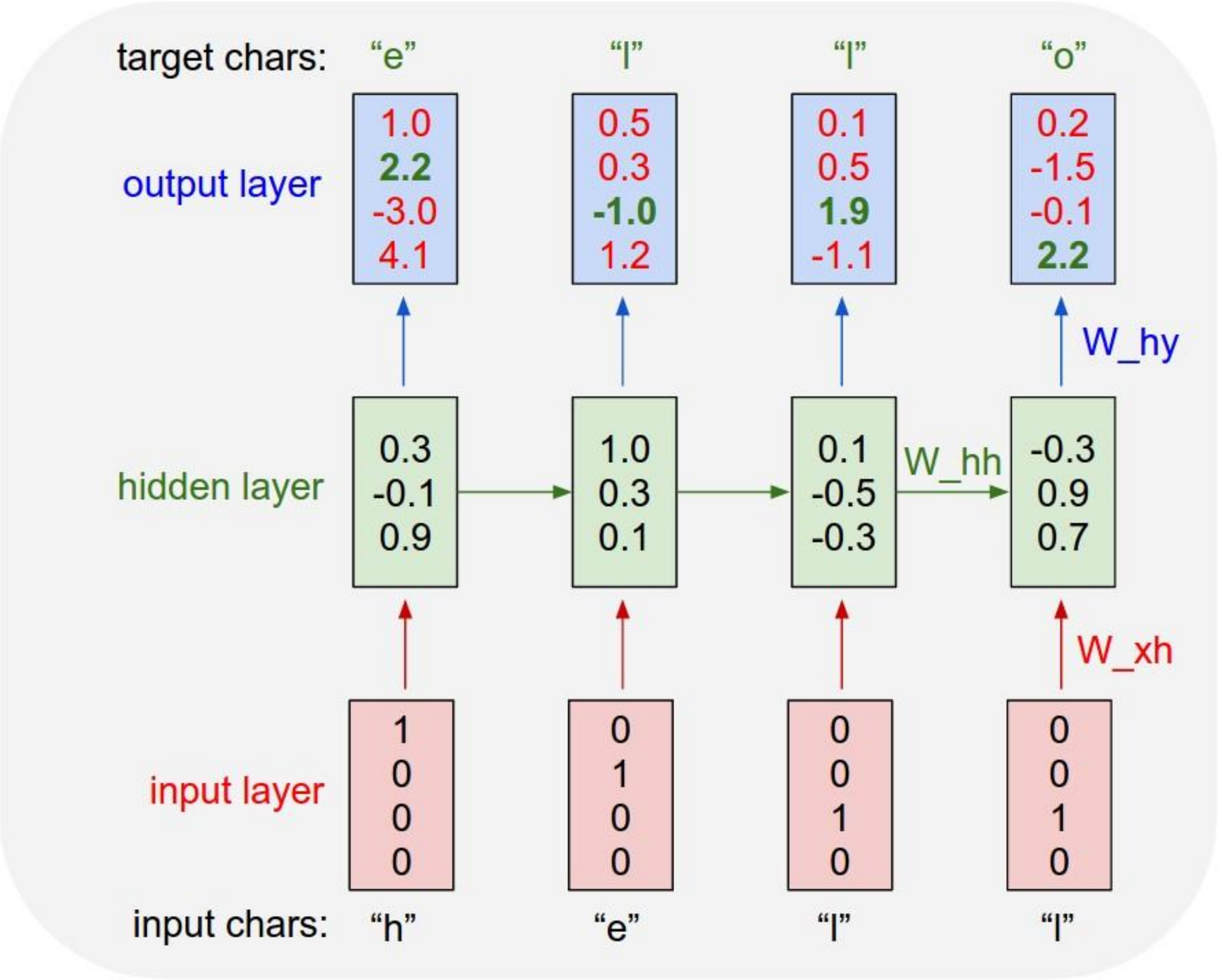
Example training sequence: **“hello”**



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

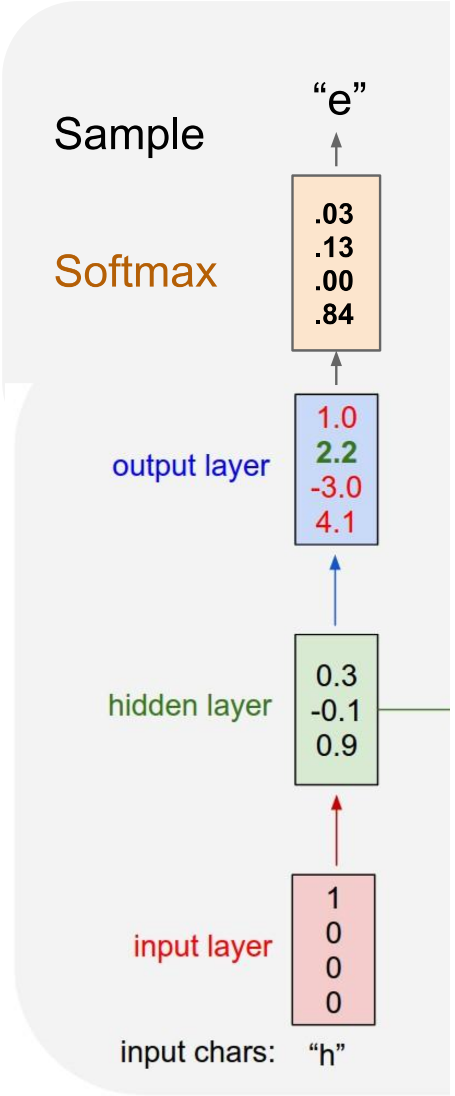
Example training sequence: **“hello”**



Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

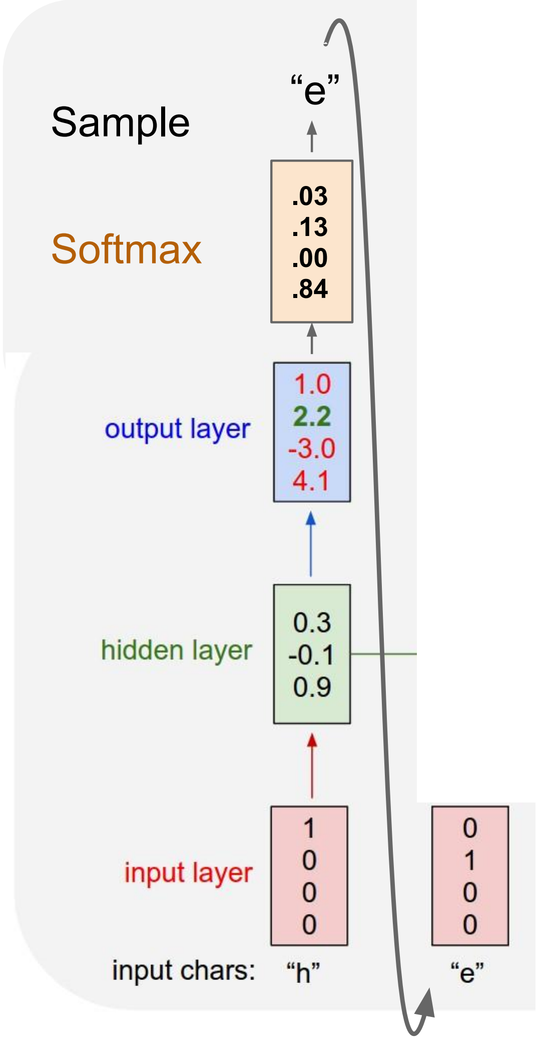
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

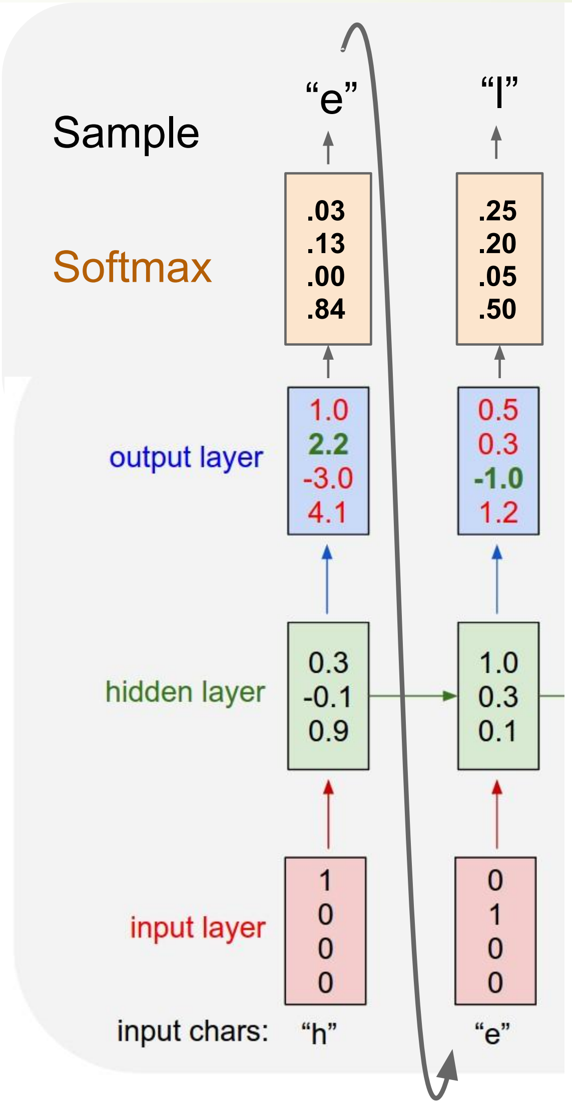
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

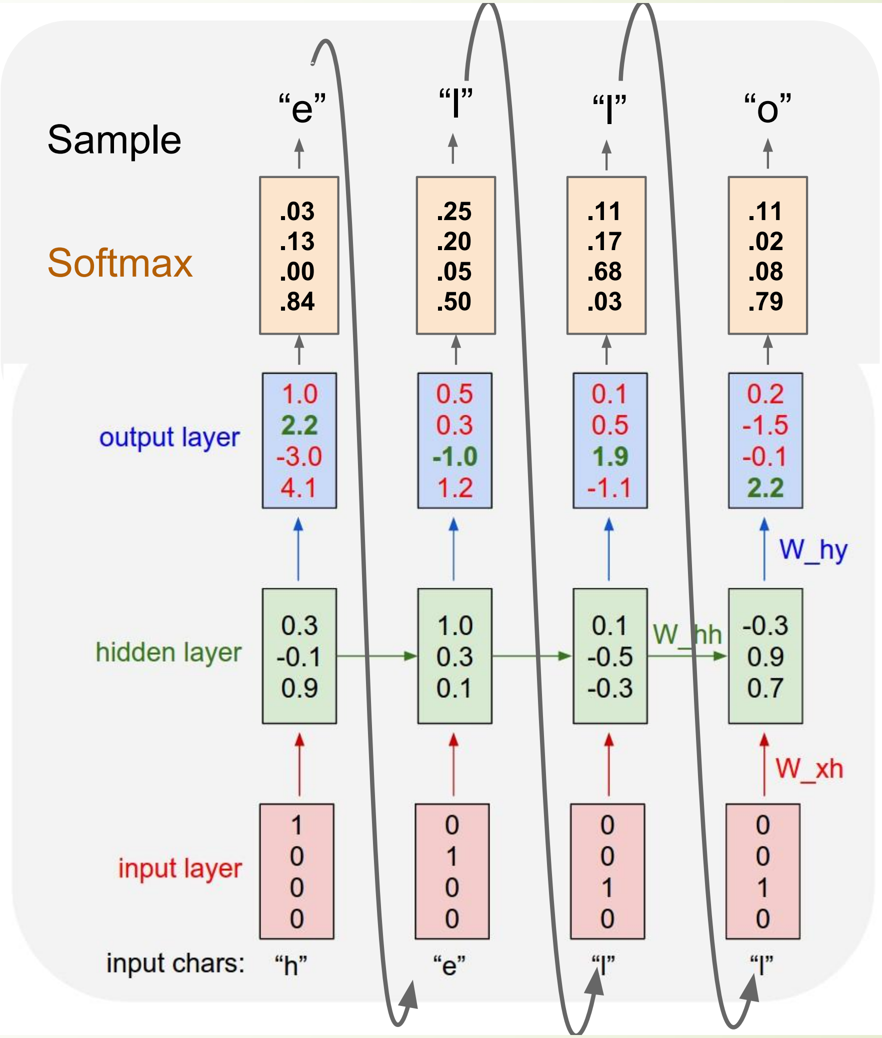
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

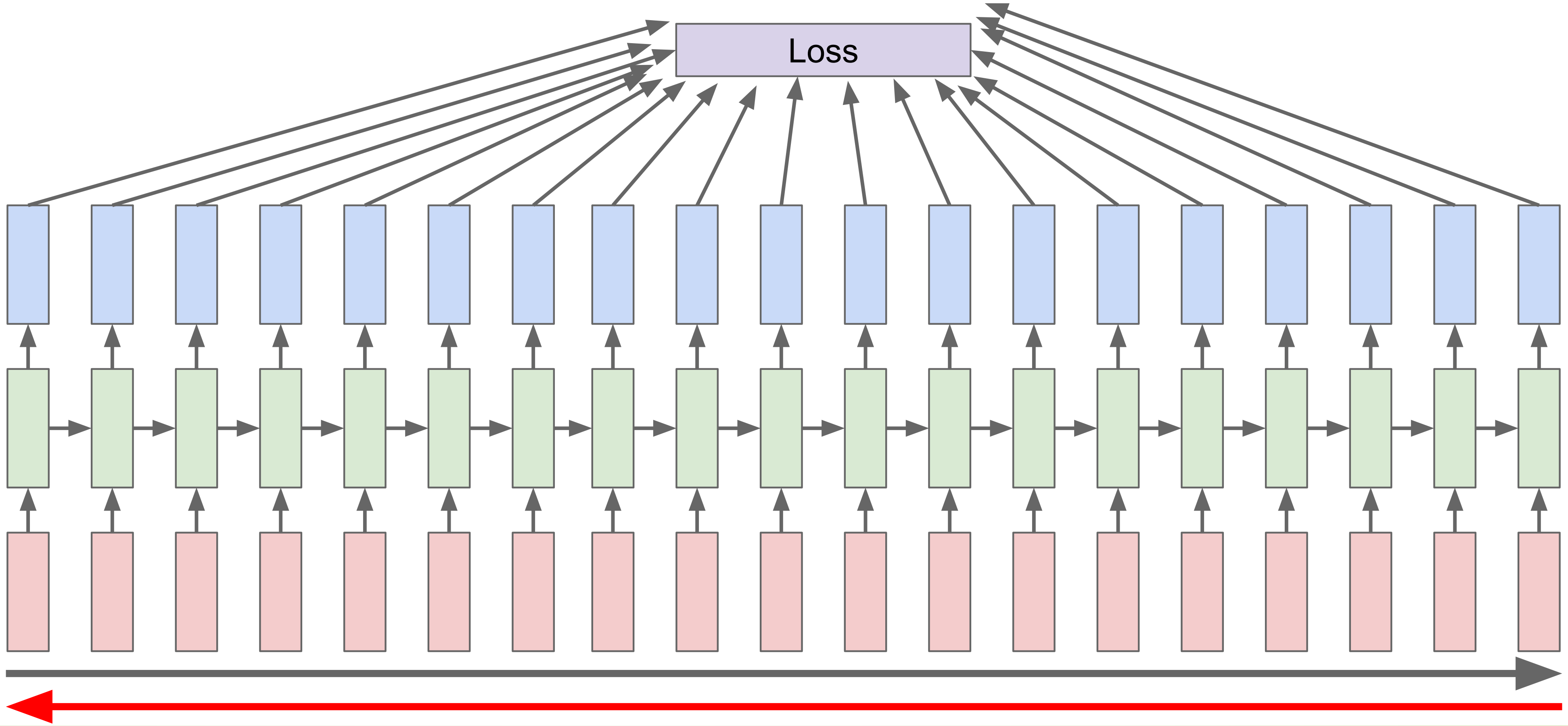
Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

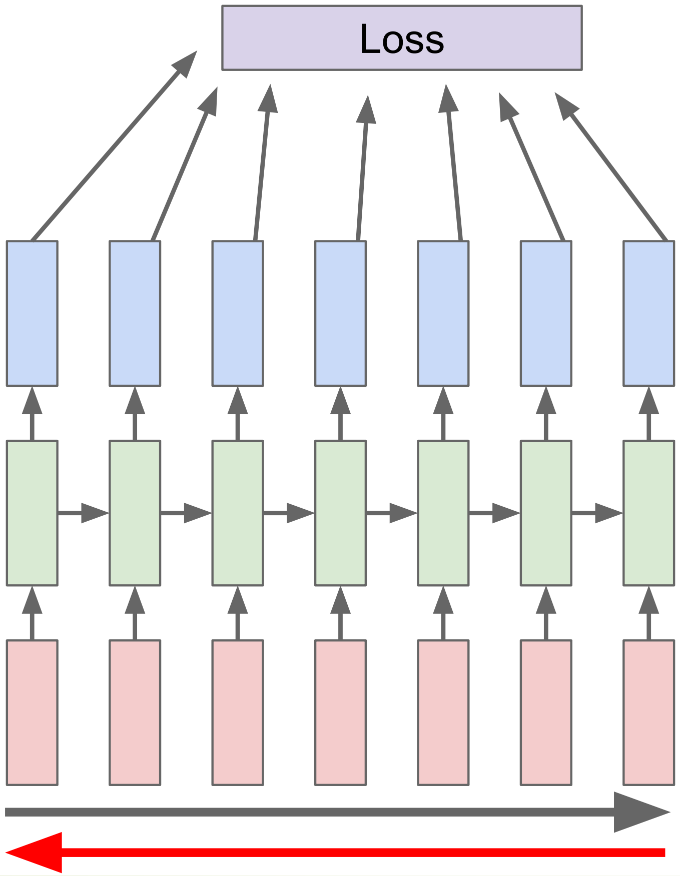


Backpropagation through time

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

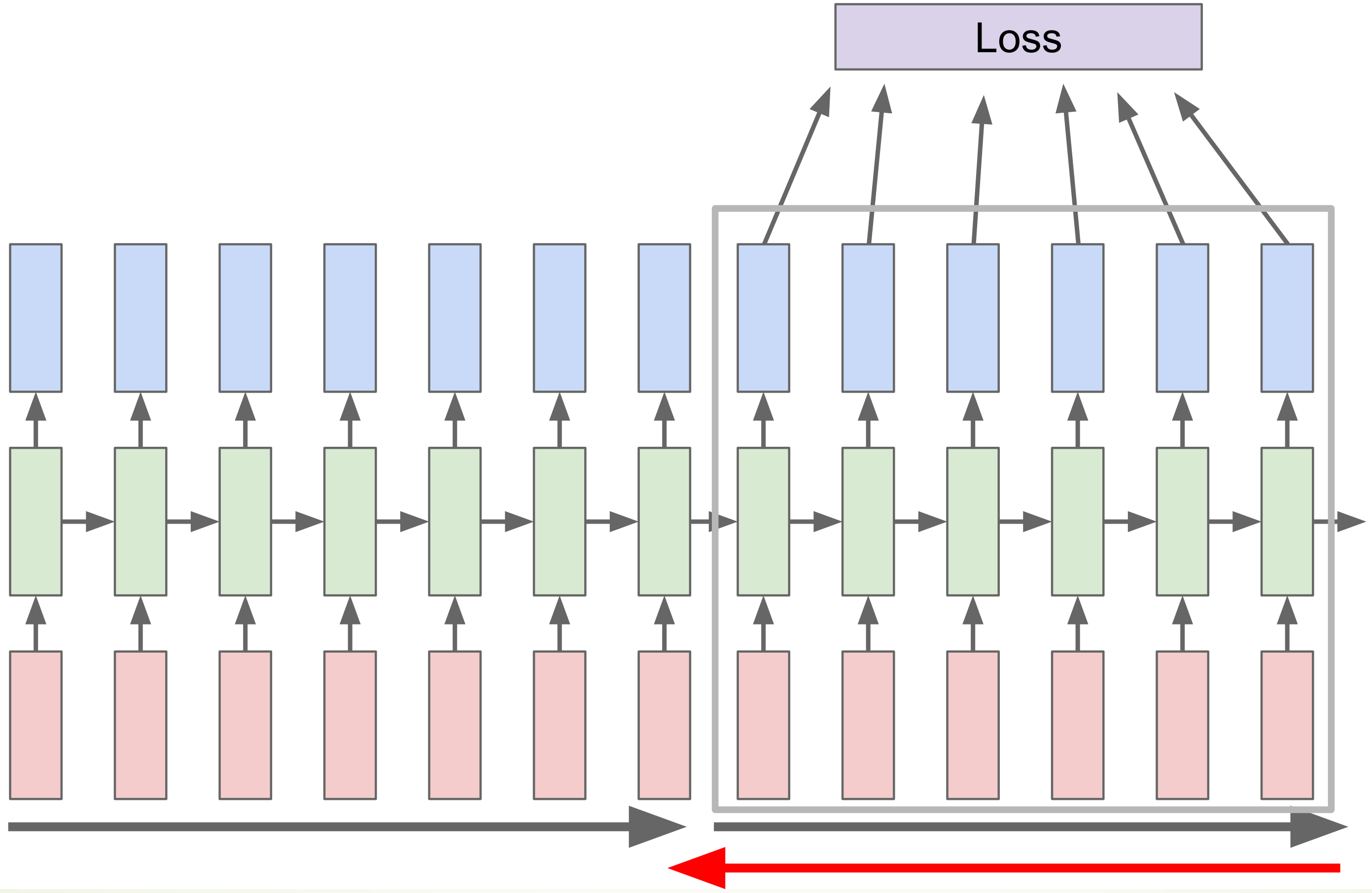


Truncated Backpropagation through time



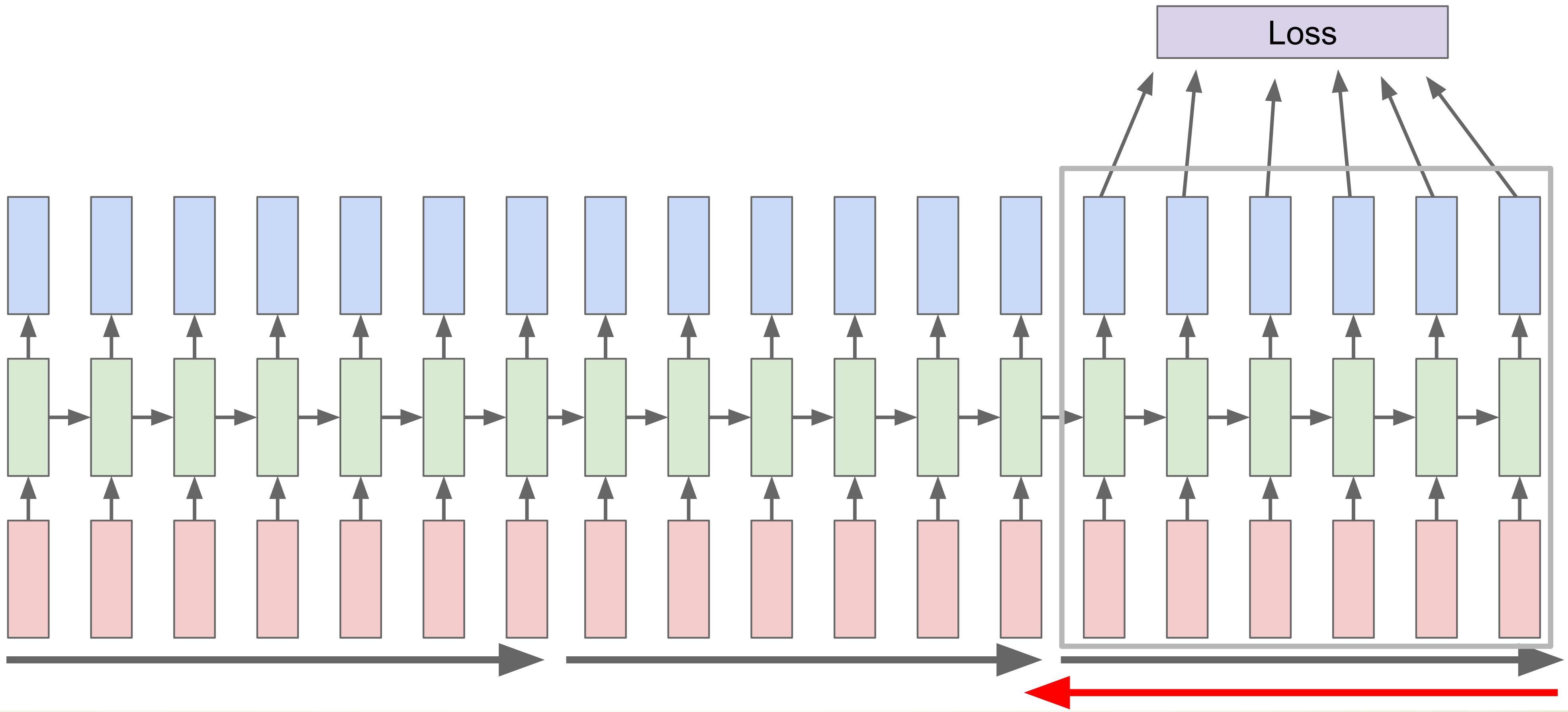
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time



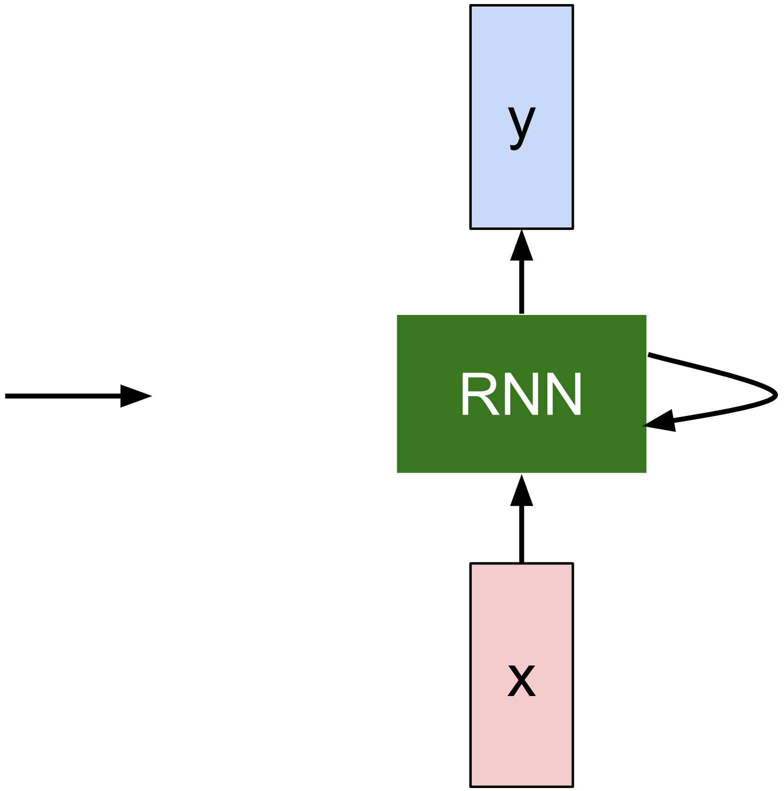
Example: Text->RNN

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
  Pity the world, or else this glutton be,
  To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
  This were to be new made when thou art old,
  And see thy blood warm when thou feel'st it cold.



https://gist.github.com/karpathy/d4dee566867f8291f086 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."
```

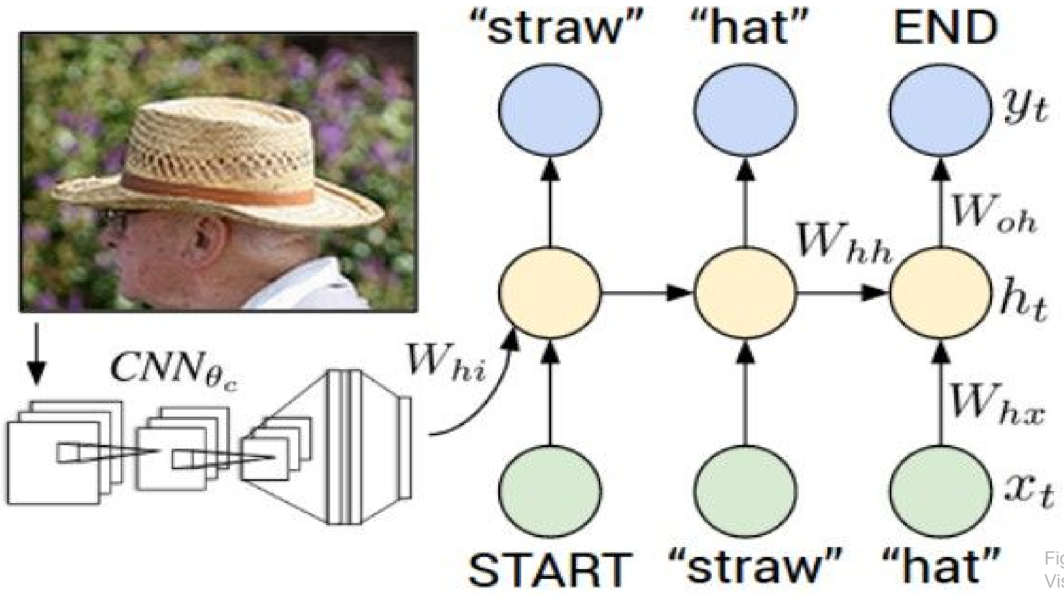
train more

```
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

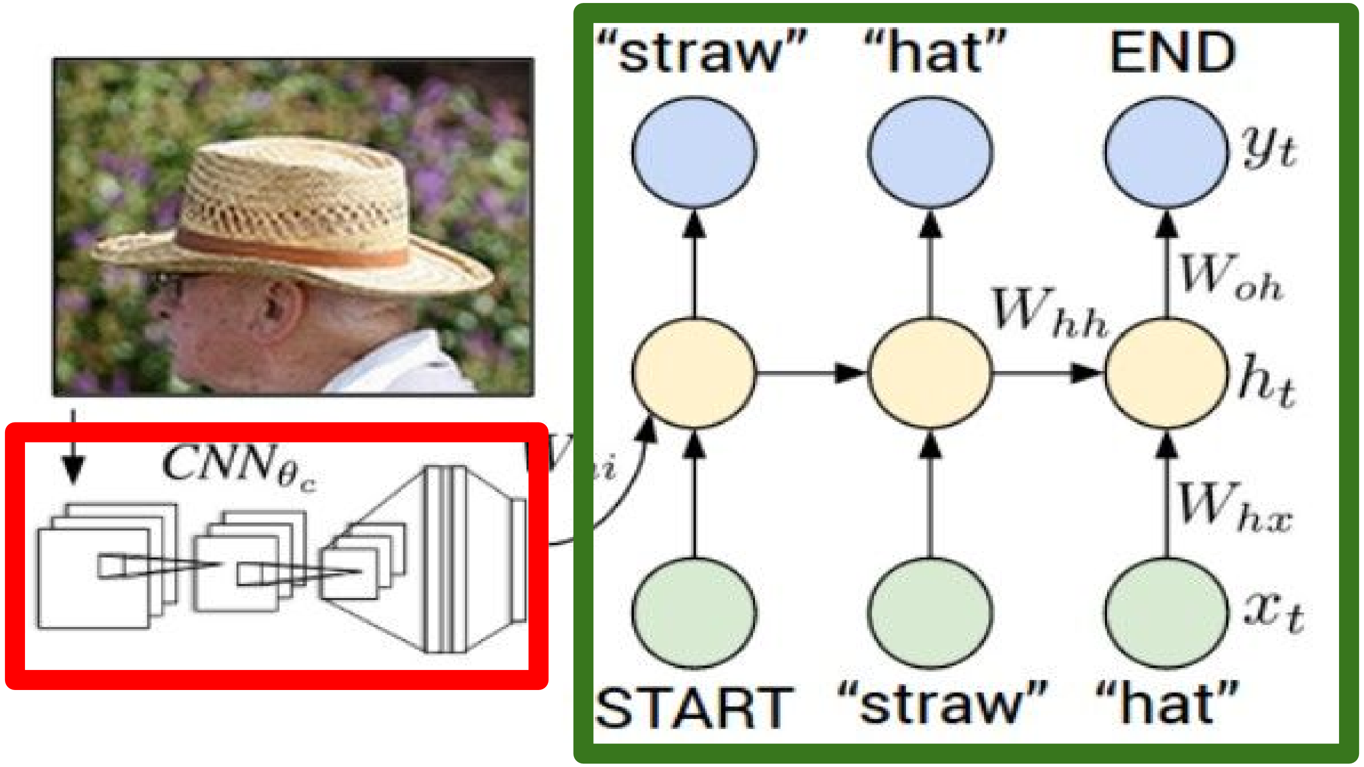
```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Image Captioning



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

Recurrent Neural Network



Convolutional Neural Network

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
FC-1000
softmax

test image

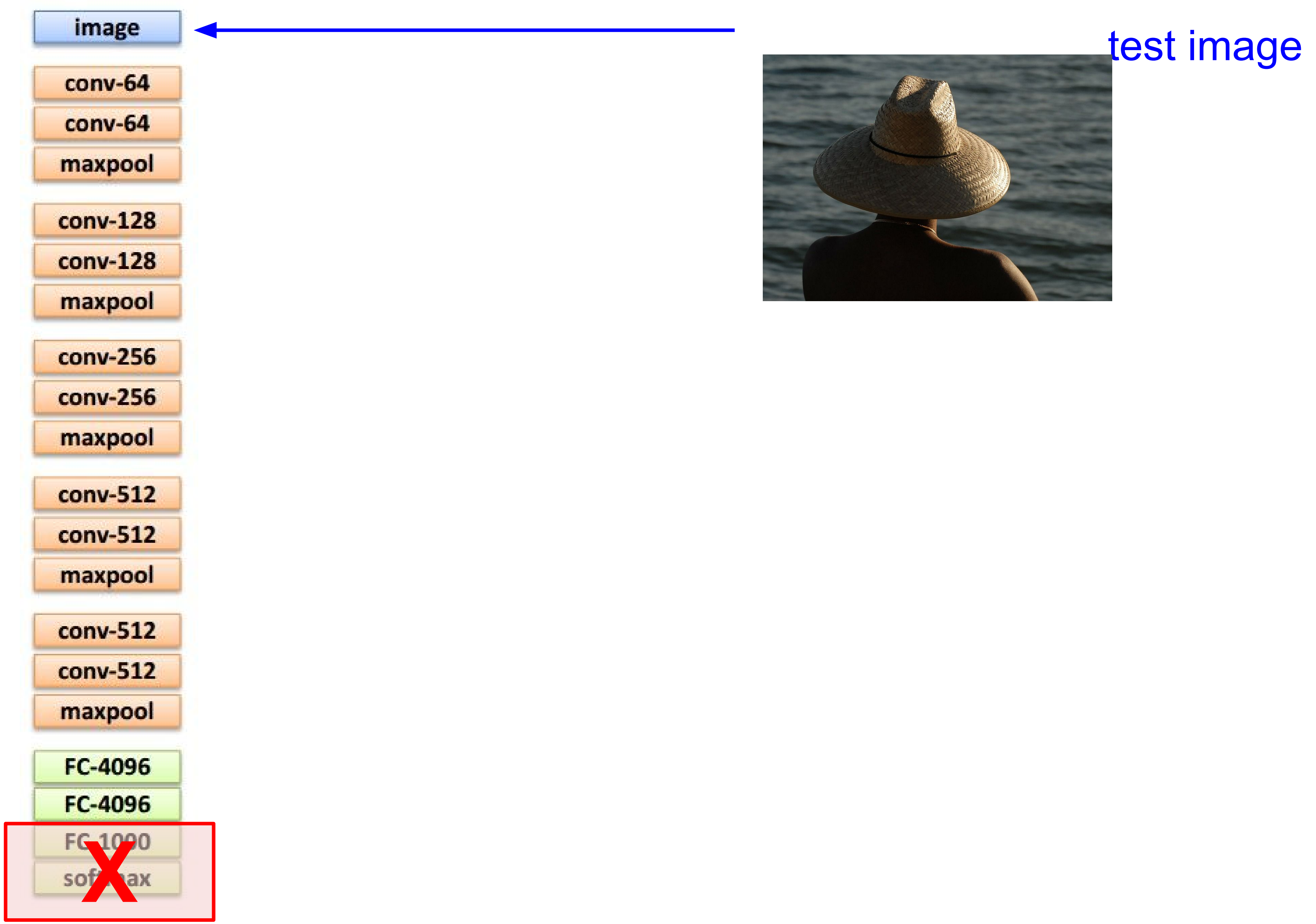
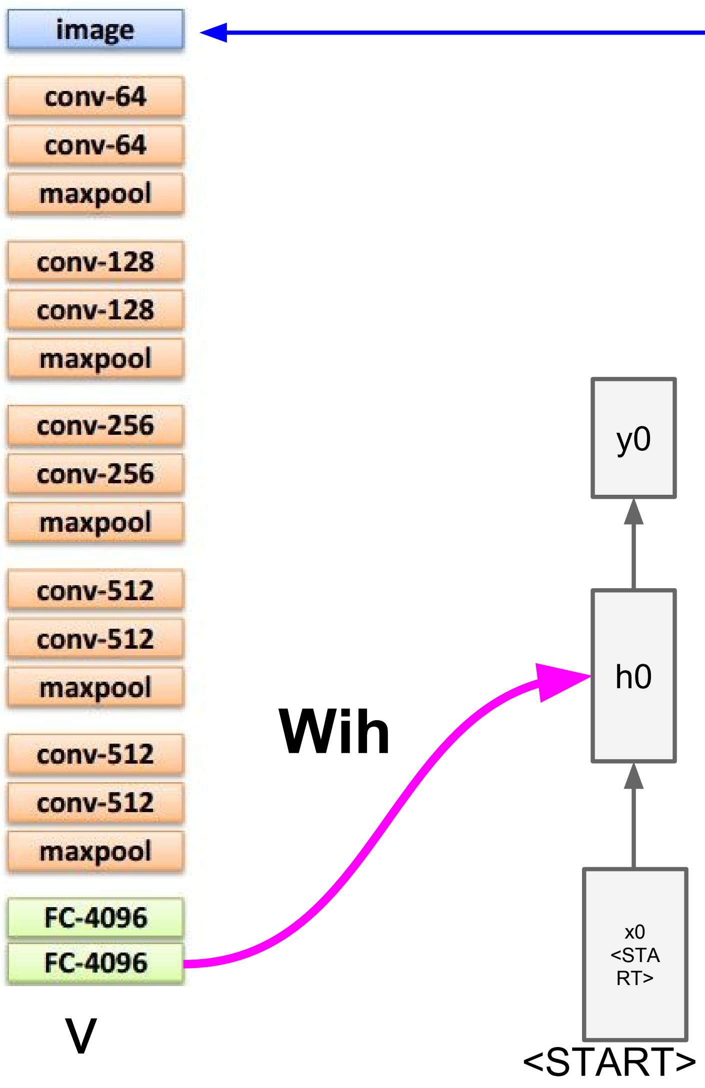


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
test image



test image

before:

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

now:

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

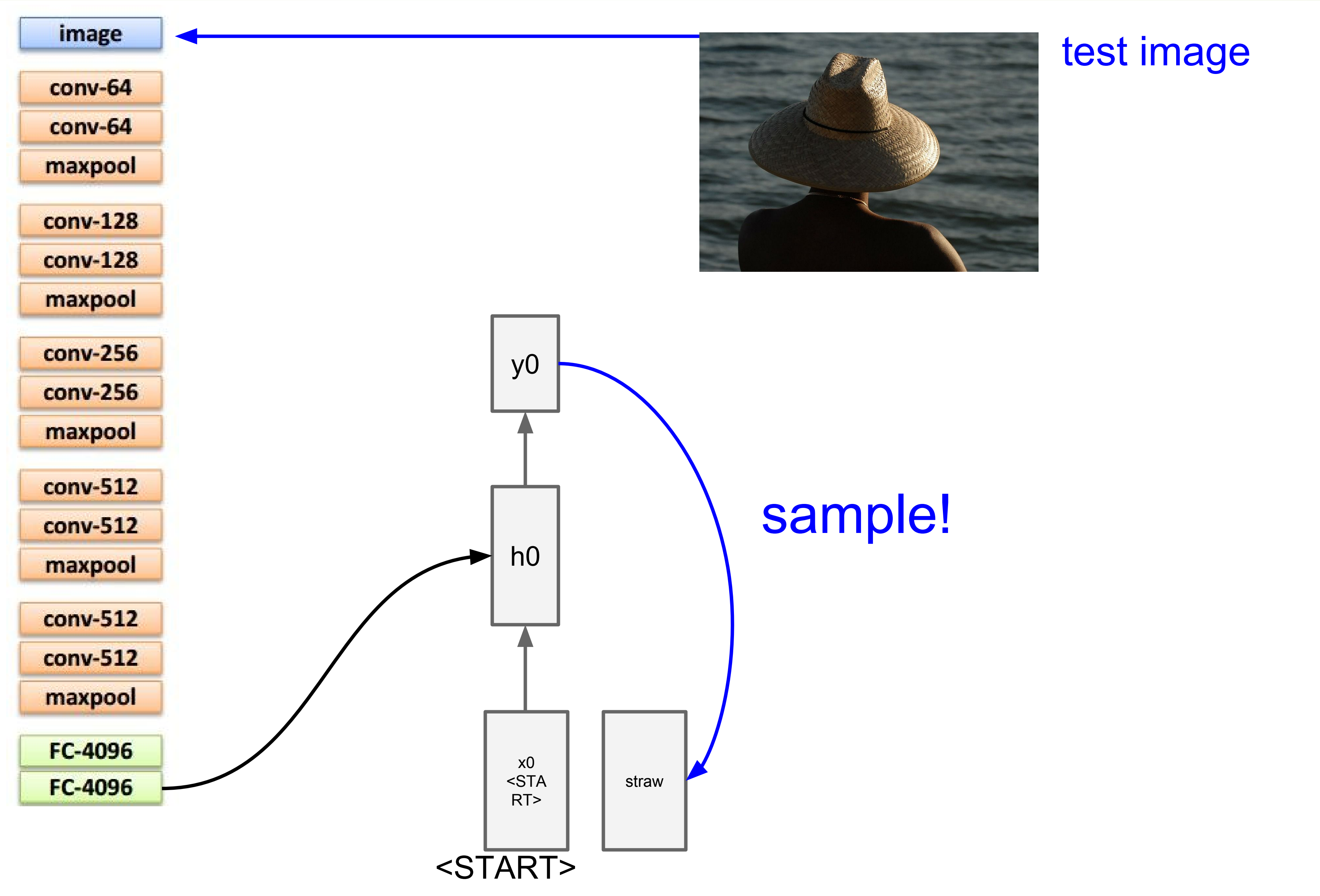


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
h0
sample!
x0
<STA
RT>
straw
<START>

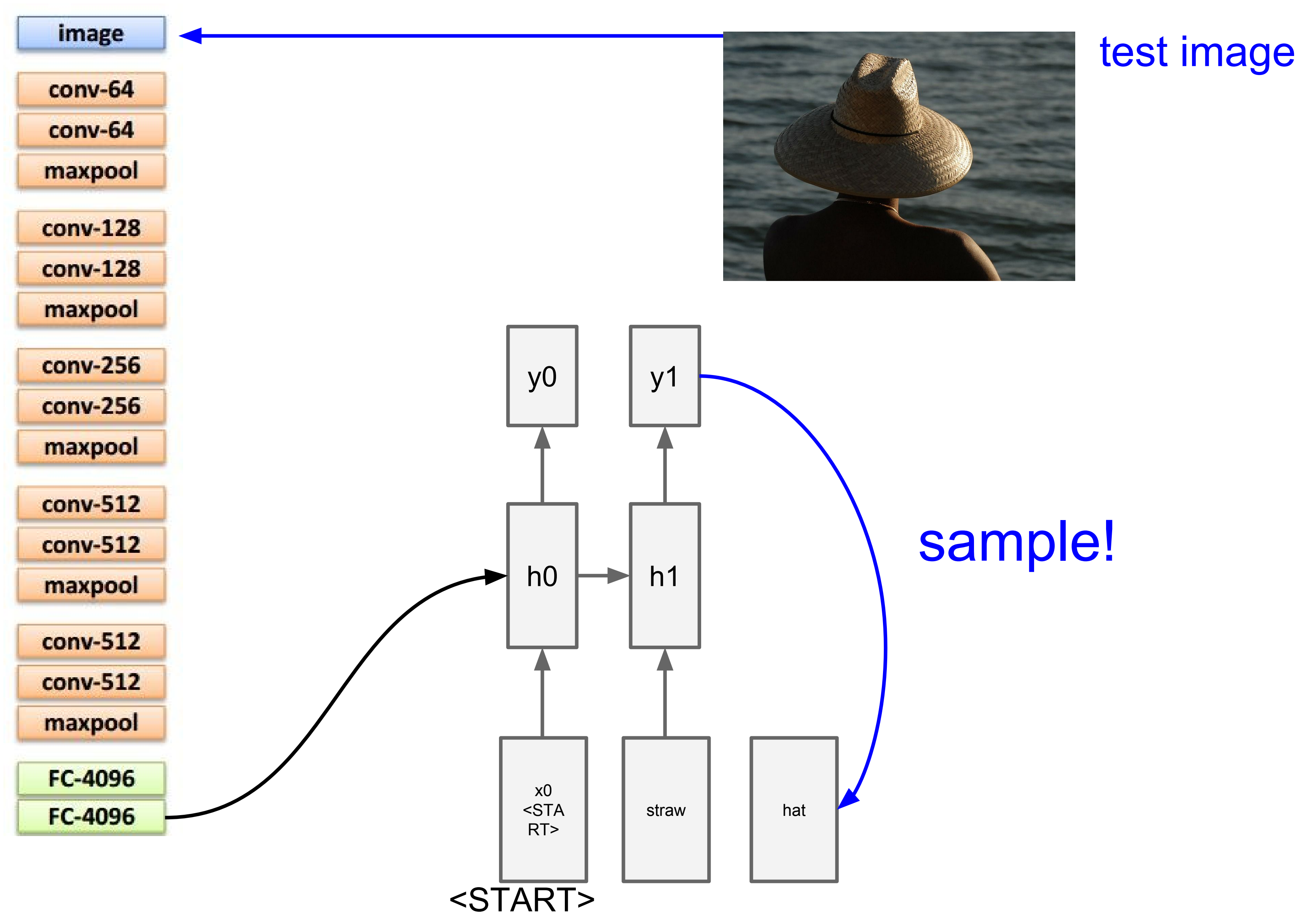


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0 <START>
straw
hat
<START>

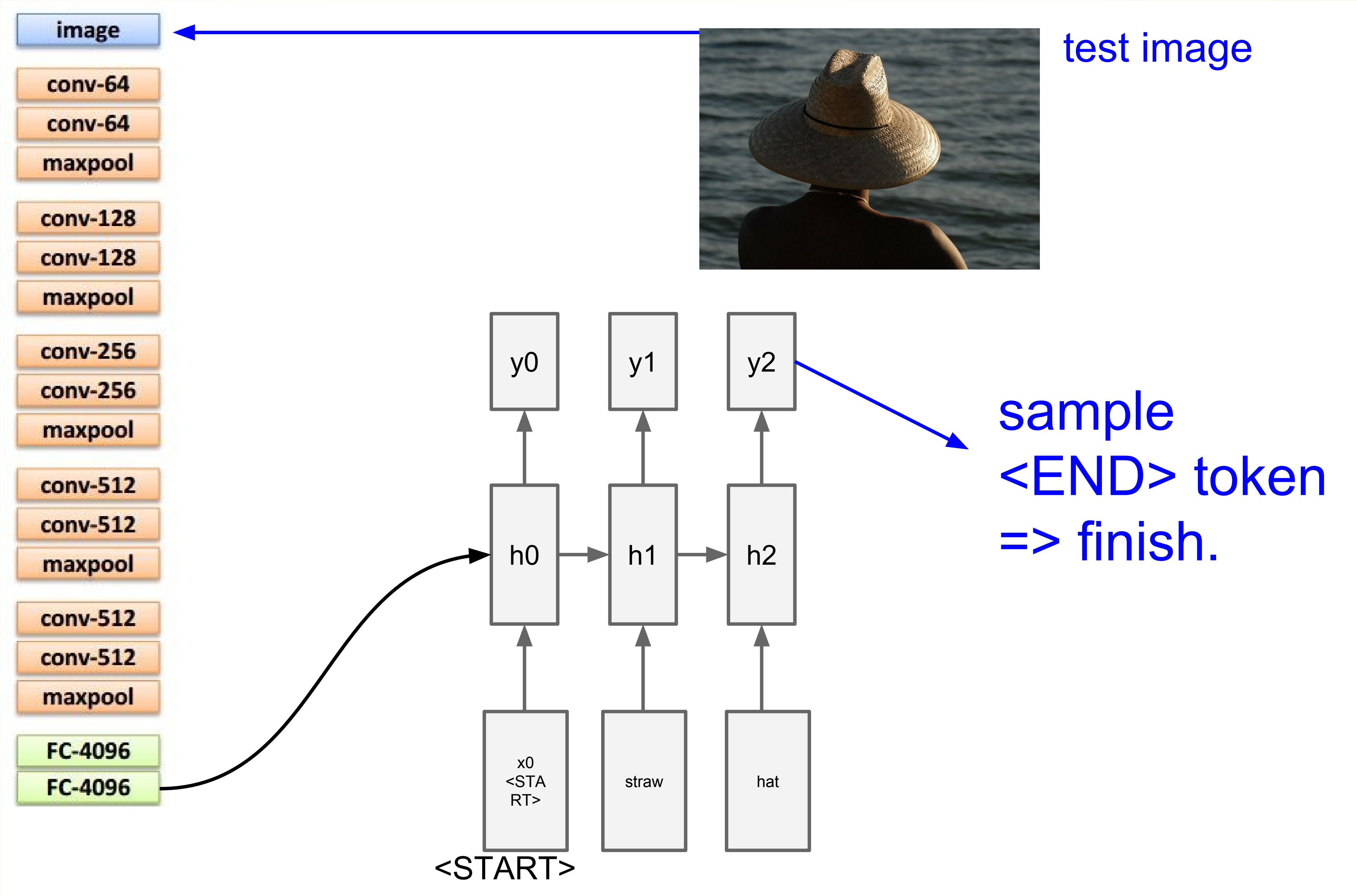


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0 <START>
straw
hat
<START>
sample <END> token => finish.

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

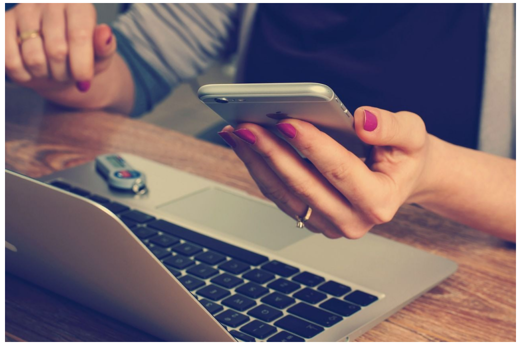
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

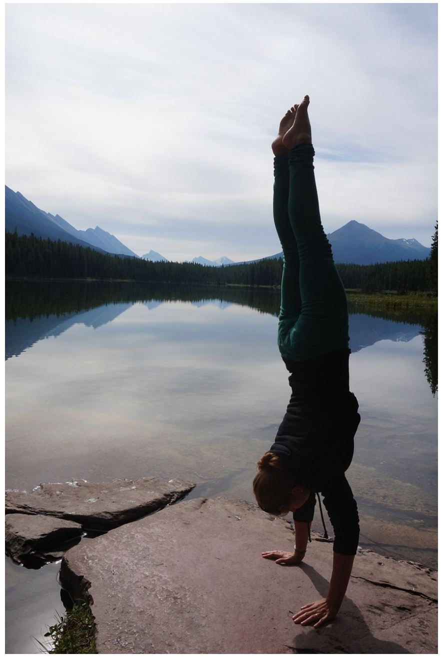
Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand



A person holding a computer mouse on a desk



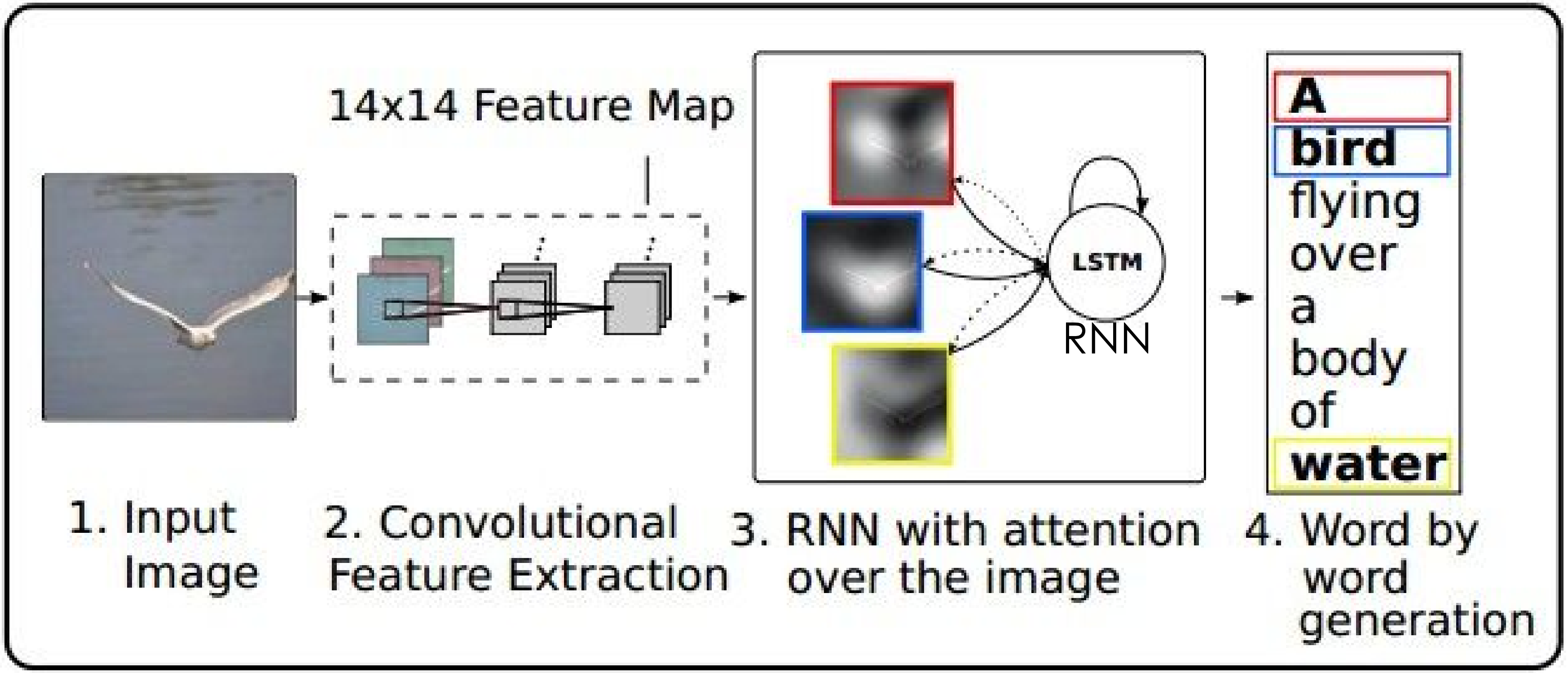
A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

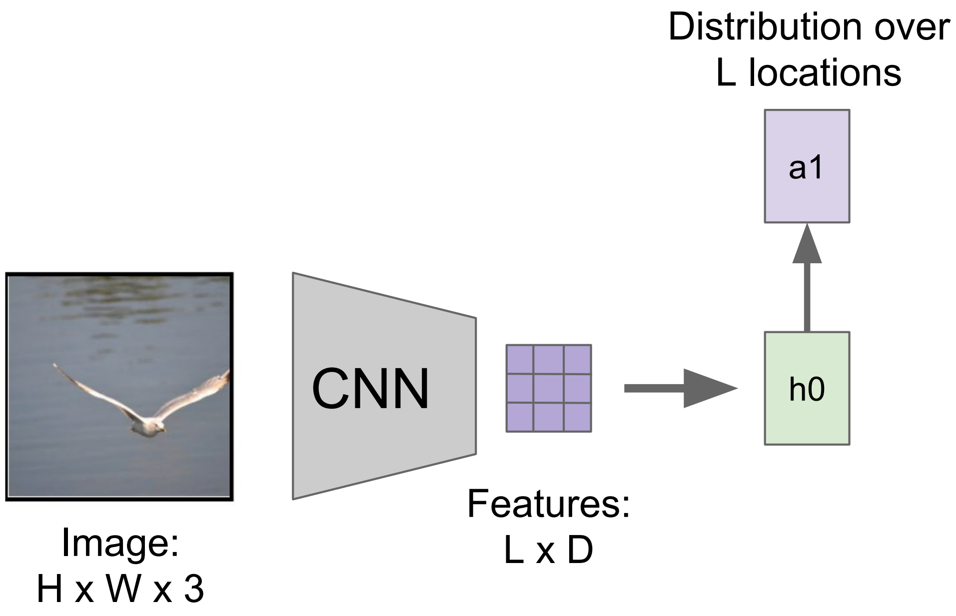
Image Captioning with Attention

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



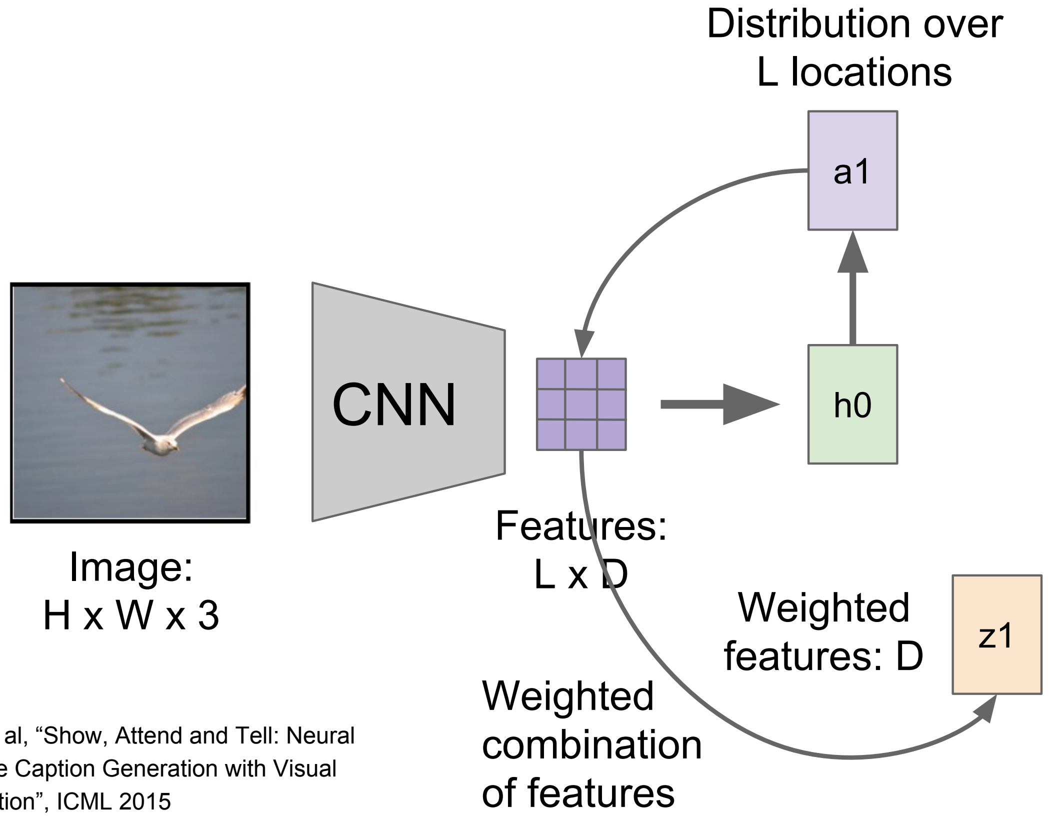
Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



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

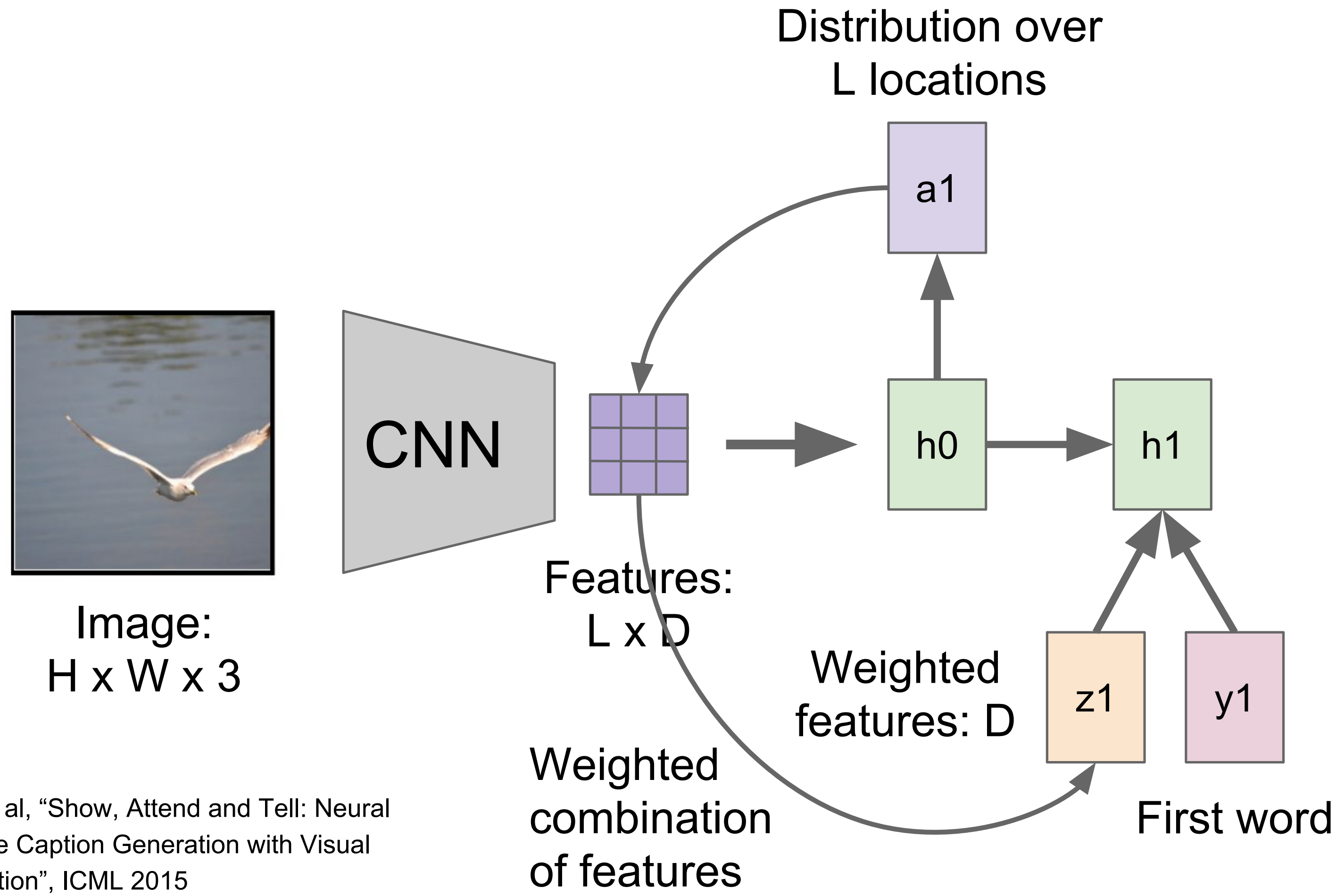
Image Captioning with Attention



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

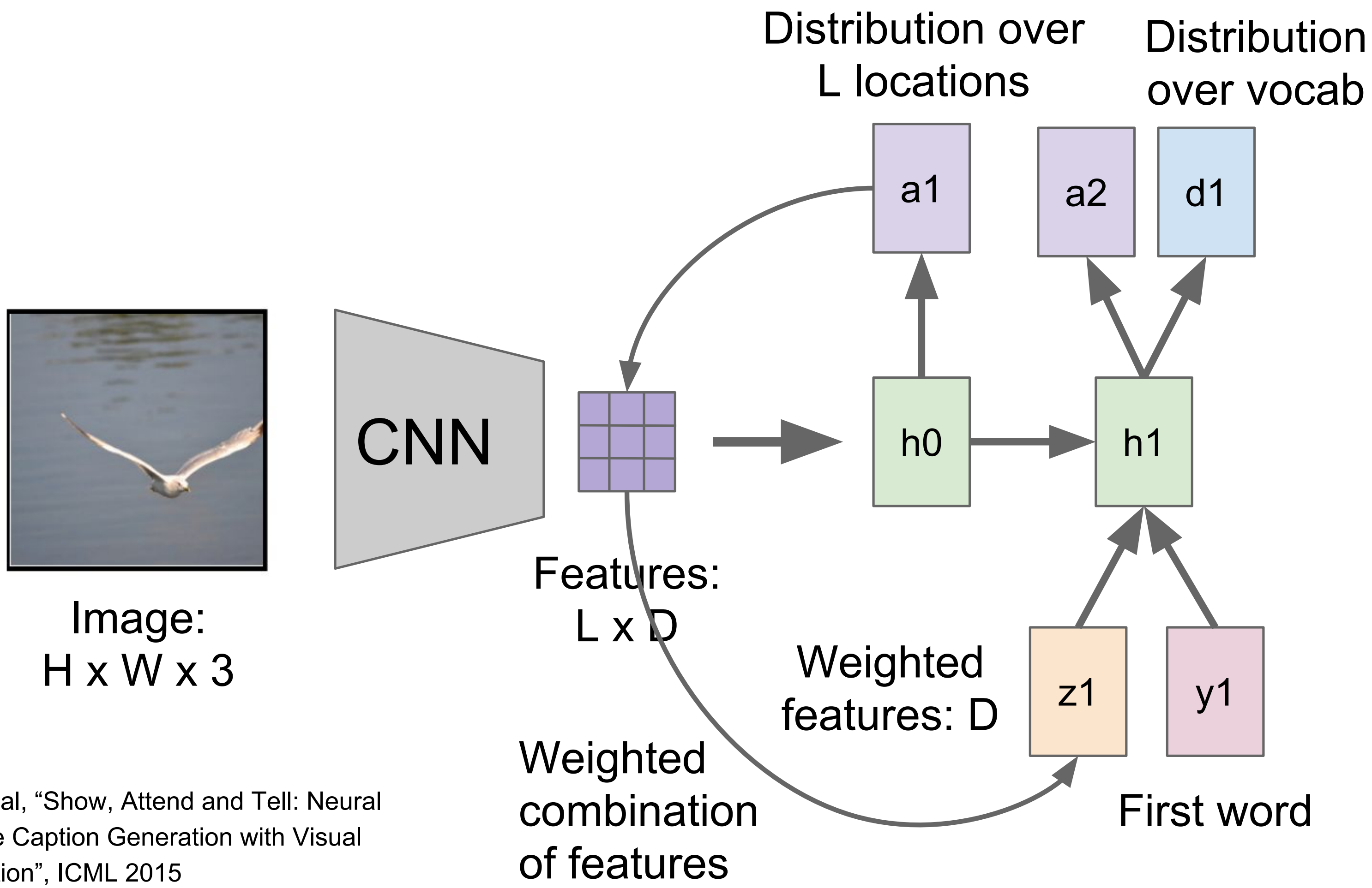
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention



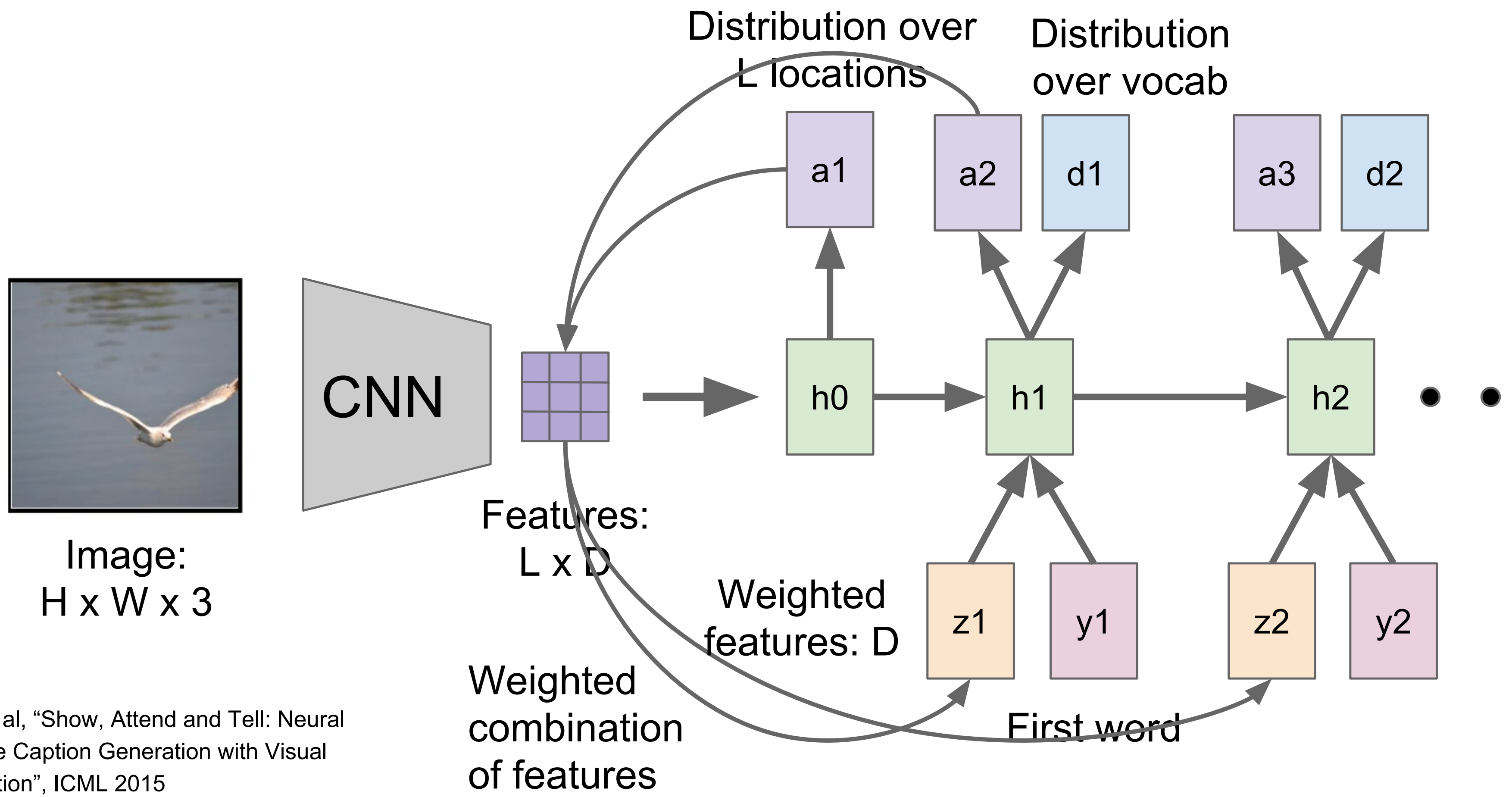
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



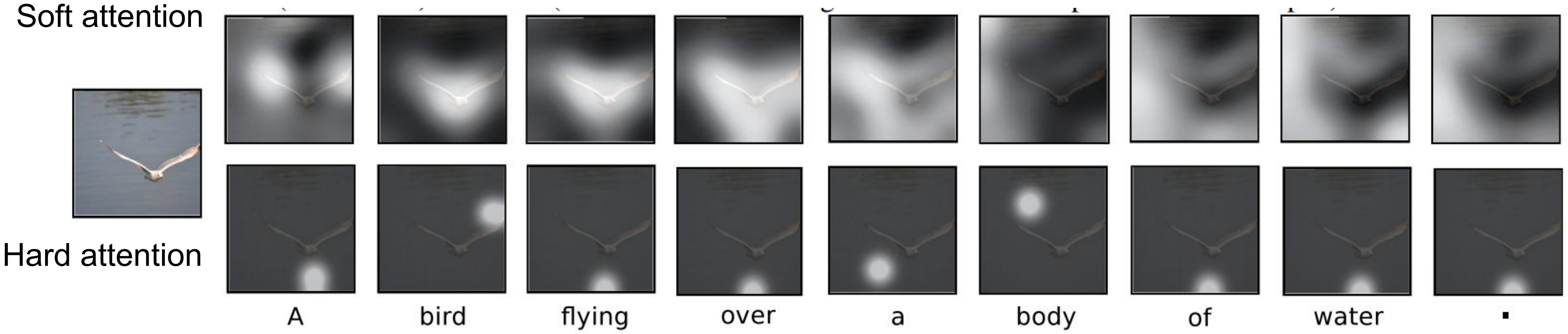
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



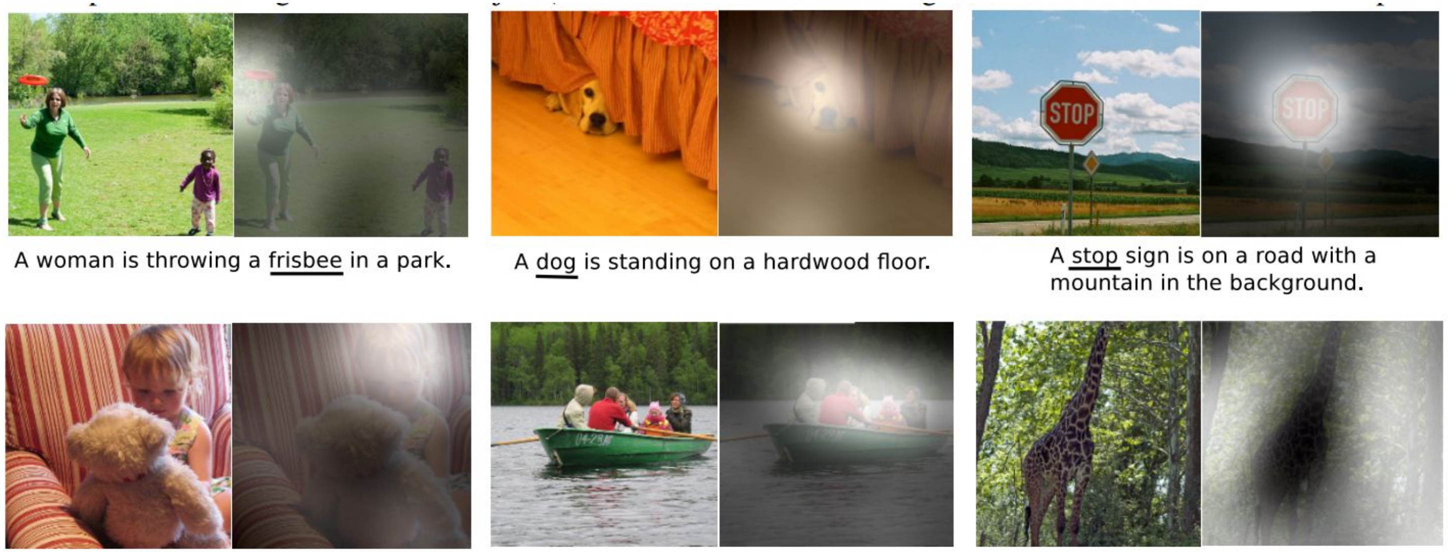
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with Attention



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

Image Captioning with Attention



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.

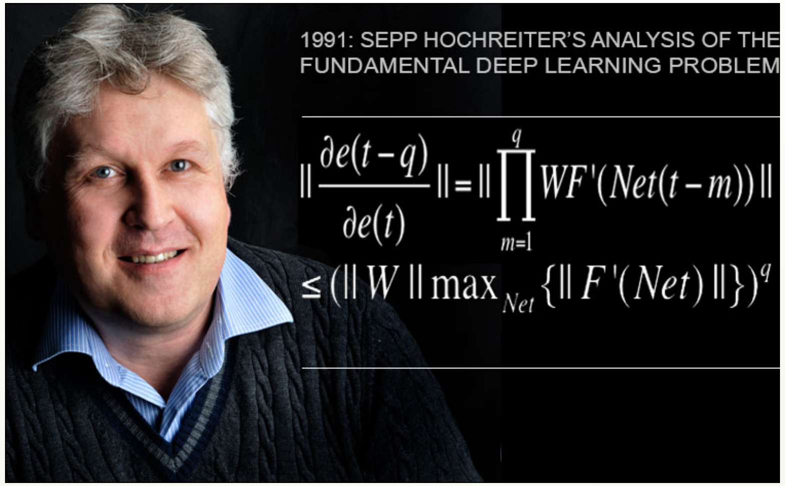
A little girl sitting on a bed with a teddy bear.

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

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

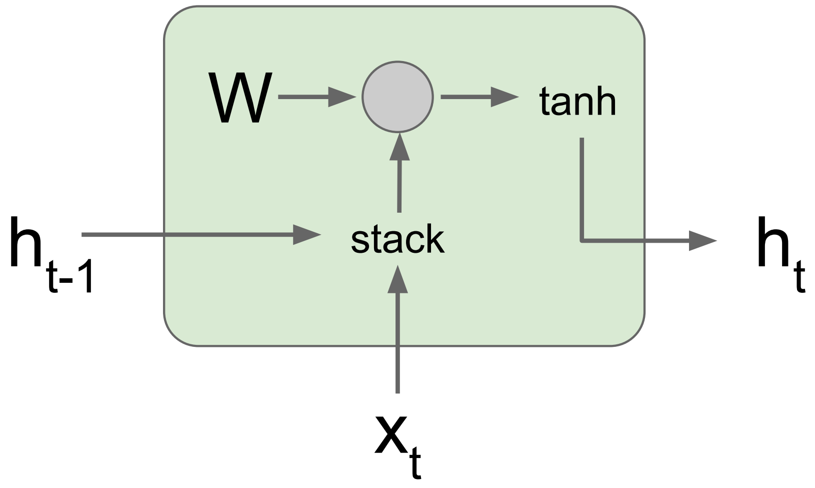
Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

The Fundamental Deep Learning Problem: **Vanishing / Exploding Gradients**



Vanilla RNN Gradient Flow

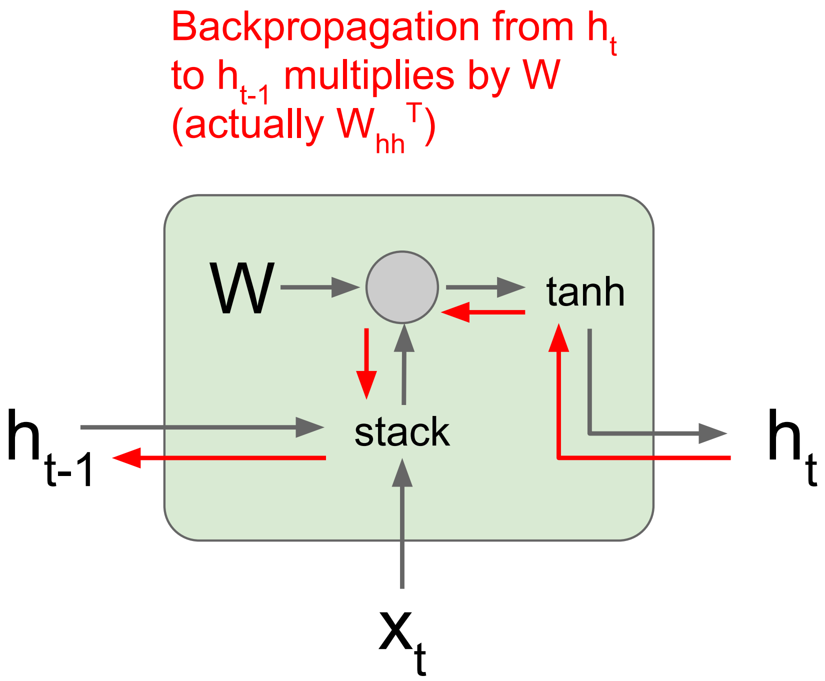
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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

Vanilla RNN Gradient Flow

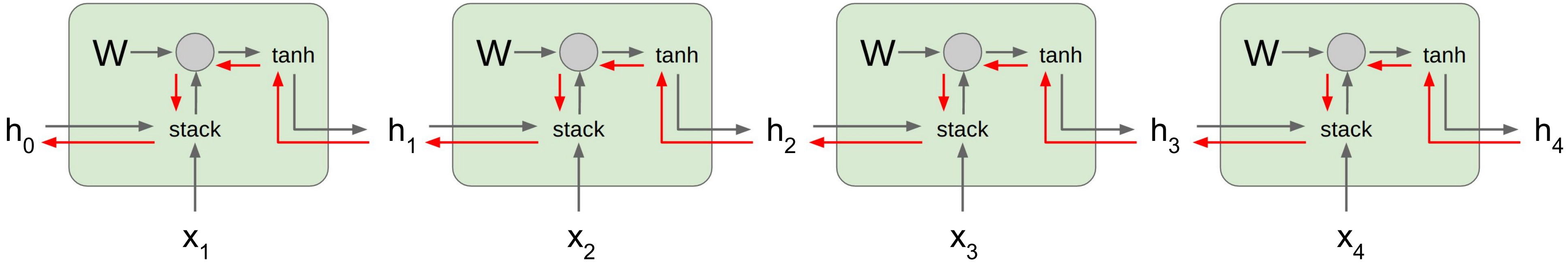
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

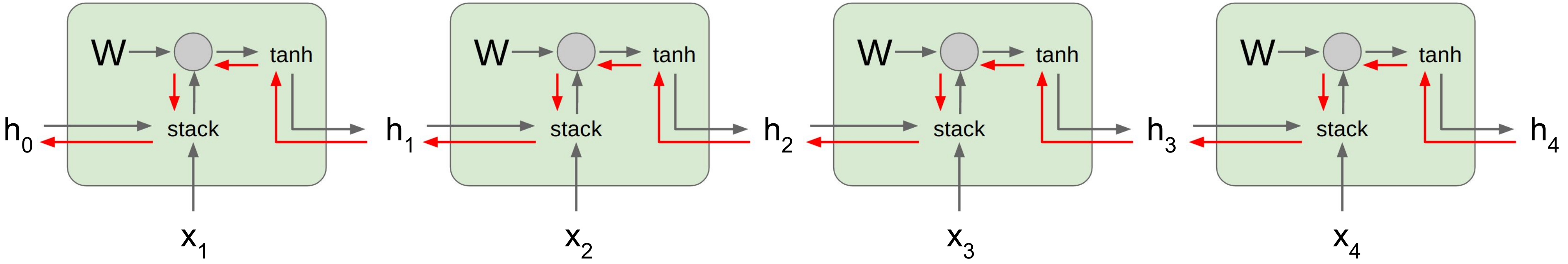
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



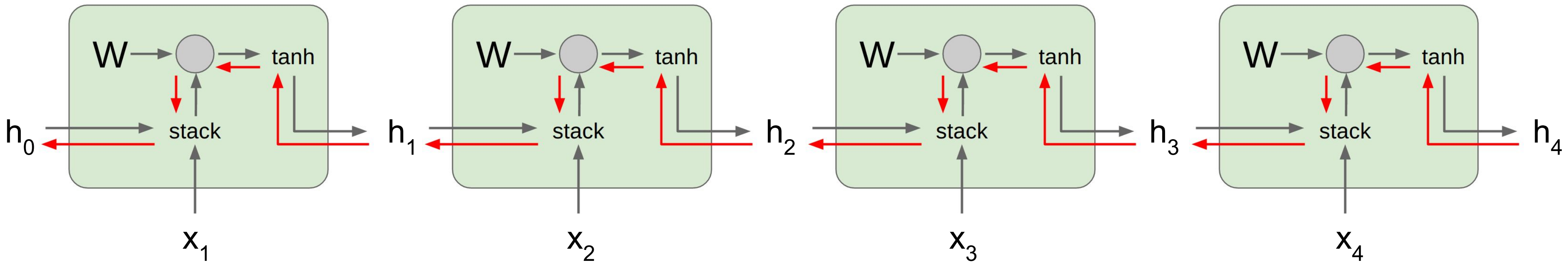
Computing gradient of h_0 involves many factors of W (and repeated tanh)

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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

Largest singular value > 1:
Exploding gradients

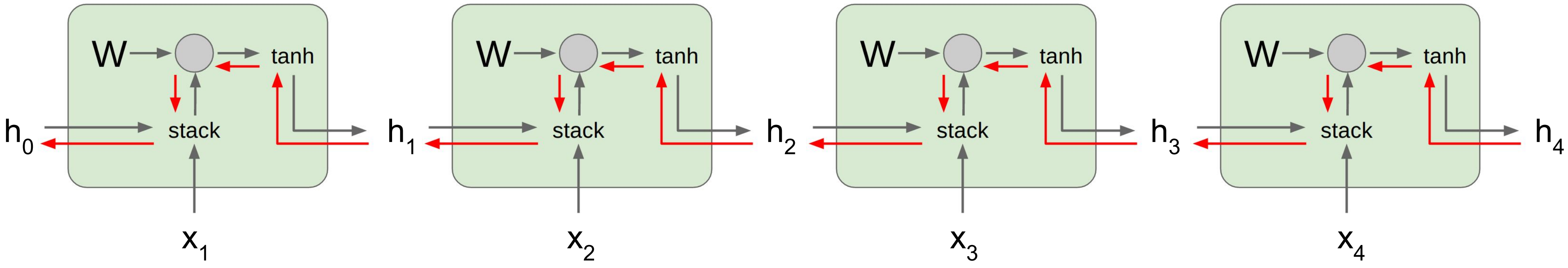
Gradient clipping: Scale gradient if its norm is too big

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

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



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 (LSTM)

Long-Short-Term-Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
  - "Long short-term memory", Hochreiter and Schmidhuber, Neural Computation, 9(8):1735-1780, 1997. Link: https://www.bioinf.jku.at/publications/older/2604.pdf
- On time step *t*, there is a hidden state h *and* a cell state c
  - Both are vectors length *n*
  - The **cell stores long-term information**
  - The LSTM can erase, write and read information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
  - The gates are also vectors length *n*
  - On each time step, each element of the gates can be open (1), closed (0),
  - or somewhere in-between.
  - The gates are **dynamic**: their value is computed based on the current context

Long-Short-Term-Memory (LSTM)

We have a sequence of inputs $\boldsymbol{x}^{(t)}$, and we will compute a sequence of hidden states $\boldsymbol{h}^{(t)}$ and cell states $\boldsymbol{c}^{(t)}$. On timestep *t*:

Sigmoid function: all gate values are between 0 and 1

Forget gate: controls what is kept vs forgotten, from previous cell state

$$\boldsymbol{f}^{(t)} = \boxed{\sigma}\left(\boldsymbol{W}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right)$$

Input gate: controls what parts of the new cell content are written to cell

$$\boldsymbol{i}^{(t)} = \sigma\left(\boldsymbol{W}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right)$$

Output gate: controls what parts of cell are output to hidden state

$$\boldsymbol{o}^{(t)} = \sigma\left(\boldsymbol{W}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right)$$

New cell content: this is the new content to be written to the cell

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh\left(\boldsymbol{W}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right)$$

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

Hidden state: read ("output") some content from the cell

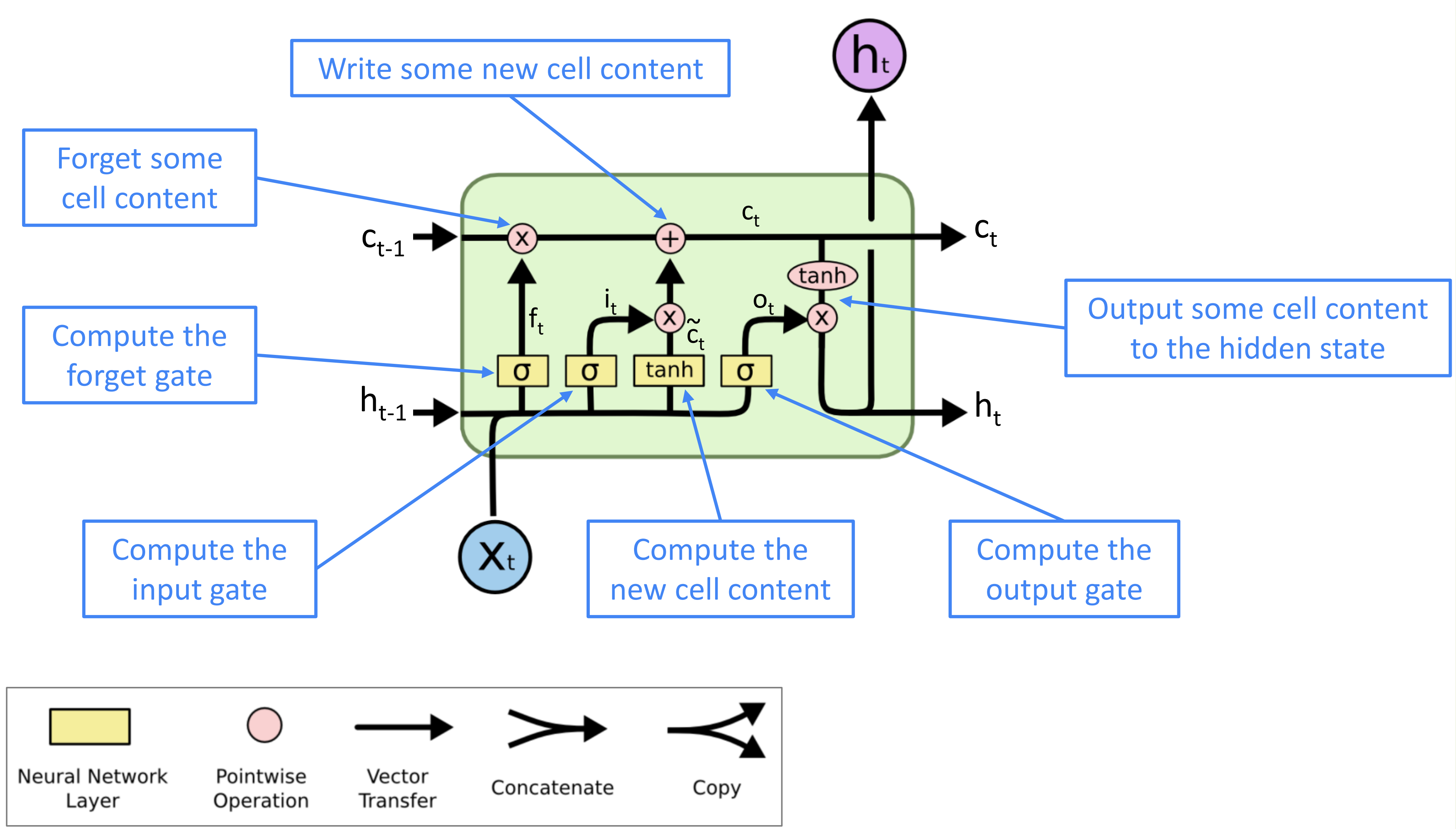
$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length *n*

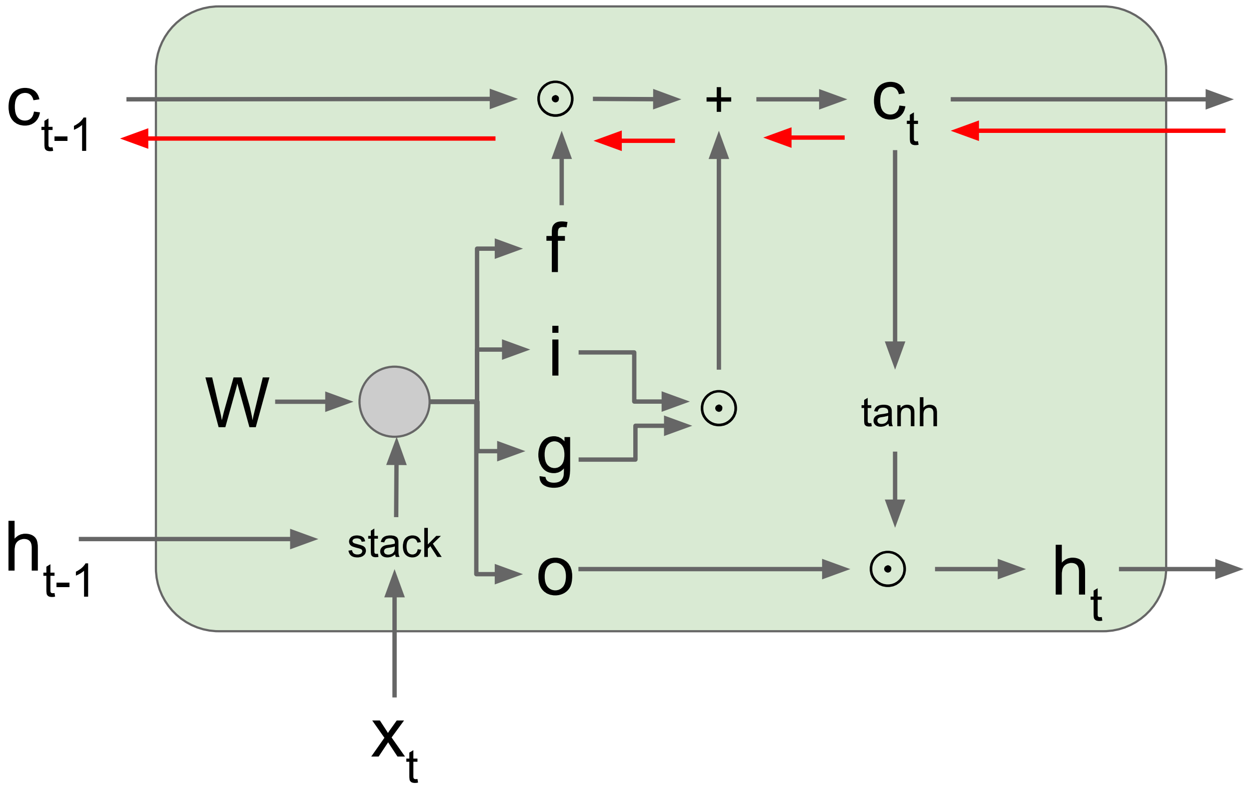
LSTM Flowchart

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



Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

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

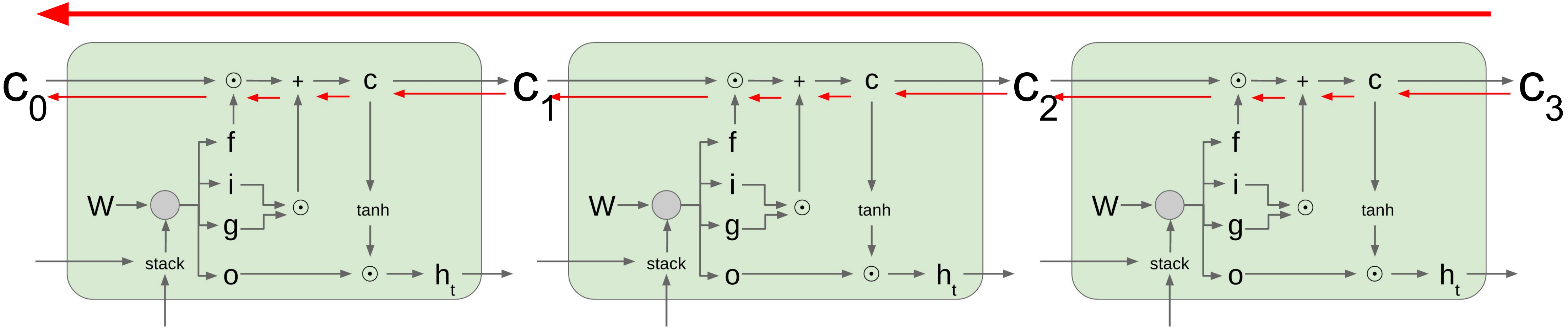
$$c_t = f \odot c_{t-1} + i \odot g$$

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

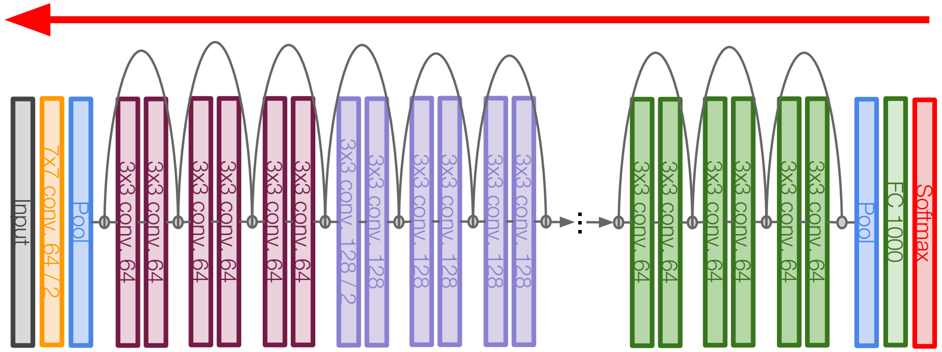
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:
Highway Networks

$$g = T(x, W_T)$$

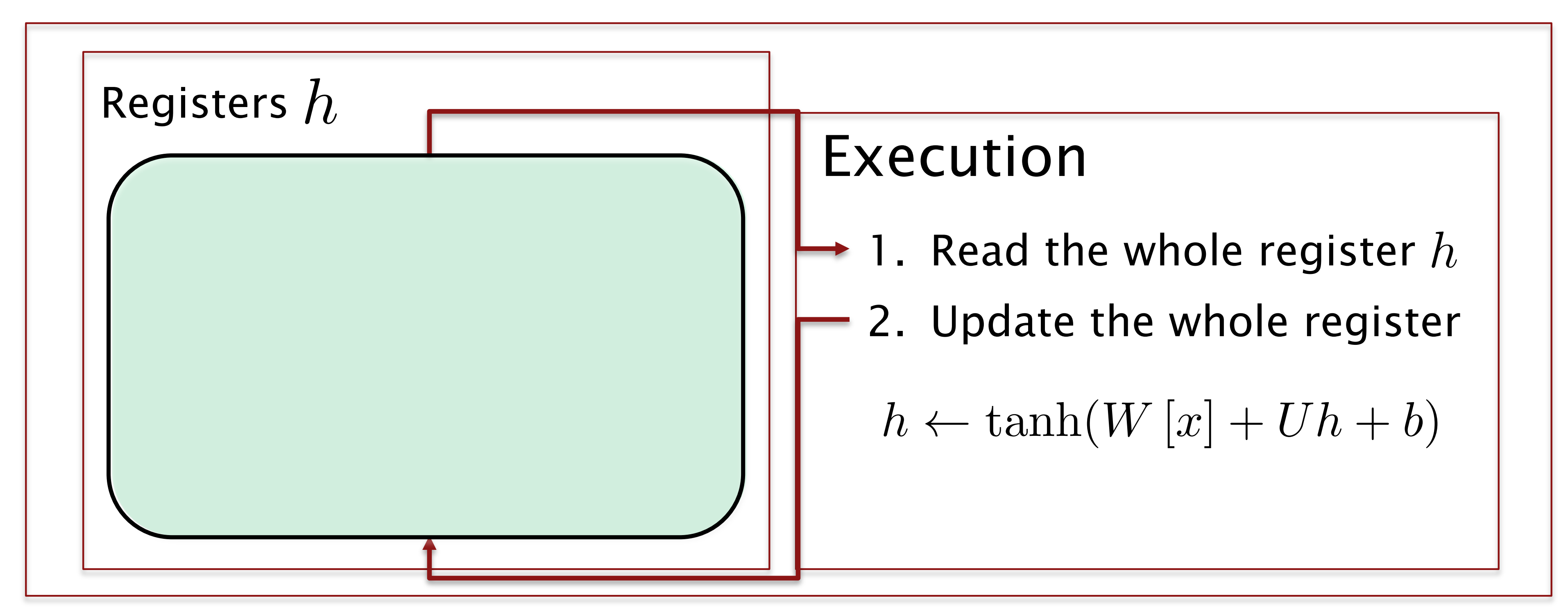
$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

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

Gated Recurrent Unit (GRU)

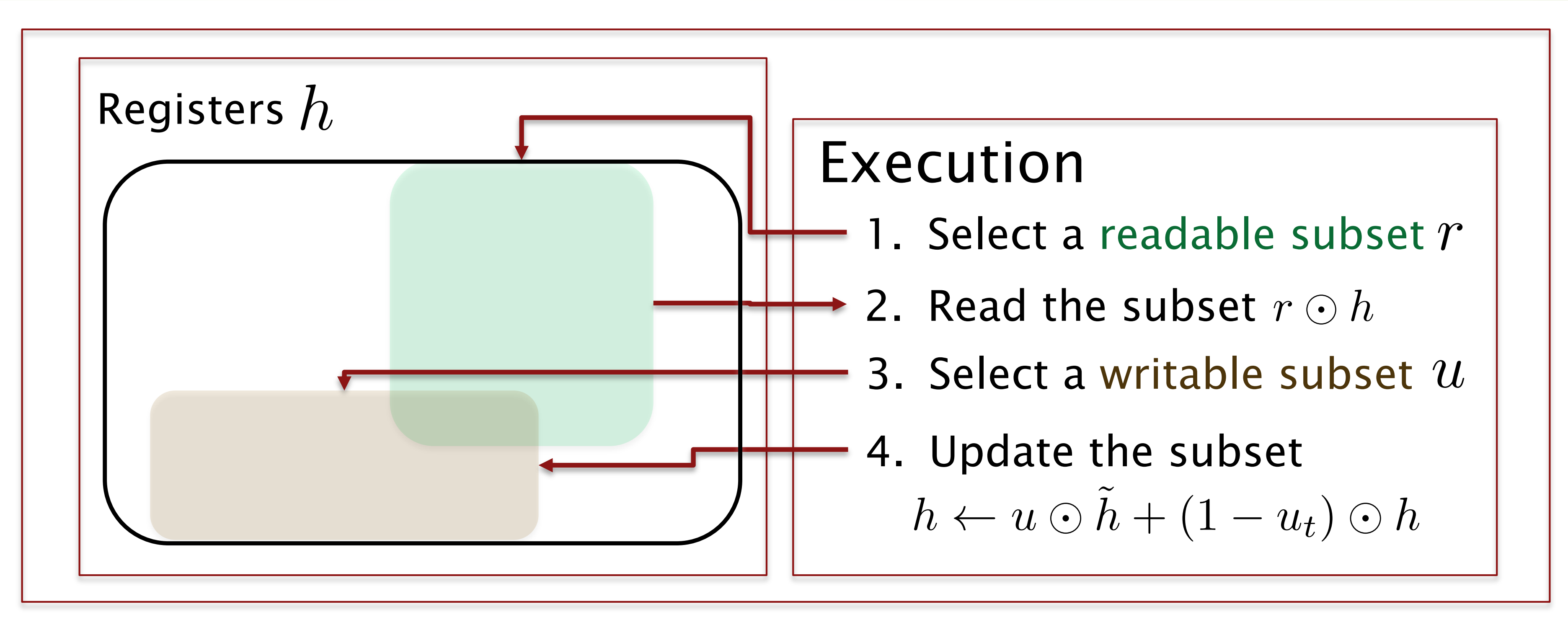
Gated Recurrent Unit: tanh RNN

- (tanh) RNN is expensive in exploiting the whole register



Gated Recurrent Unit (GRU)

- GRU is much more economic for computation!



GRU

- "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", Cho et al. 2014, https://arxiv.org/pdf/1406.1078v3.pdf

Update gate: controls what parts of hidden state are updated vs preserved

Reset gate: controls what parts of previous hidden state are used to compute new content

New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$\boldsymbol{u}^{(t)} = \sigma\left(\boldsymbol{W}_u \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_u \boldsymbol{x}^{(t)} + \boldsymbol{b}_u\right)$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right)$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h (\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right)$$

$$\boldsymbol{h}^{(t)} = (1 - \boldsymbol{u}^{(t)}) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)}$$

How does this solve vanishing gradient?
Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

GRU and LSTM

Gated Recurrent Unit

[Cho et al., EMNLP2014; Chung, Gulcehre, Cho, Bengio, DLUFL2014]

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

$$\tilde{h}_t = \tanh(W[x_t] + U(r_t \odot h_{t-1}) + b)$$

$$u_t = \sigma(W_u[x_t] + U_u h_{t-1} + b_u)$$

$$r_t = \sigma(W_r[x_t] + U_r h_{t-1} + b_r)$$

Long Short-Term Memory

[Hochreiter & Schmidhuber, NC1999; Gers, Thesis2001]

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

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$\tilde{c}_t = \tanh(W_c[x_t] + U_c h_{t-1} + b_c)$$

$$o_t = \sigma(W_o[x_t] + U_o h_{t-1} + b_o)$$

$$i_t = \sigma(W_i[x_t] + U_i h_{t-1} + b_i)$$

$$f_t = \sigma(W_f[x_t] + U_f h_{t-1} + b_f)$$

LSTM vs. GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- The biggest difference is that GRU is quicker to **compute** and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)
- **Rule of thumb**: start with LSTM, but switch to GRU if you want something more efficient

Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **deep** ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus early layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more **direct connections** (thus allowing the gradient to flow)
- For example:
  - "**HighwayNet**" with highway connections:
    - Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate
    - Inspired by LSTMs, but applied to deep feedforward/convolutional networks
  - **ResNet** with residual connections, inspired by **HighwayNet**
  - **DenseNet** directly connect everything to everything!

Summary

- RNN is flexible in architectures
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
  - Backward flow of gradients in RNN can explode or vanish.
  - Exploding is controlled with gradient clipping.
  - Vanishing is controlled with additive interactions

Some Historical Remarks on LSTM

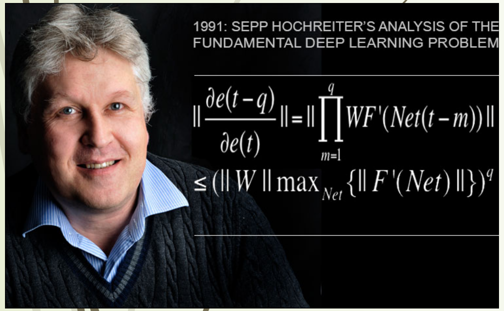
- [**LSTM0**] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. TR FKI-207-95, TUM, August 1995. Link: https://people.idsia.ch/~juergen/FKI-207-95ocr.pdf
- [**LSTM1**] S. Hochreiter, J. Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8):1735-1780, 1997. Based on [LSTM0].
- [**LSTM2**] F. A. Gers, J. Schmidhuber, F. Cummins. Learning to Forget: Continual Prediction with LSTM. Neural Computation, 12(10):2451-2471, 2000. *The "vanilla LSTM architecture" with forget gates that everybody is using today, e.g., in Google's Tensorflow.*
- [**LSTM3**] A. Graves, J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 18:5-6, pp. 602-610, 2005.

Schmidhuber: "In 2020 we celebrated the quarter-century anniversary of LSTM's first failure to pass peer review. After the main peer-reviewed publication in 1997[LSTM1] (now the most cited article in the history of *Neural Computation*), LSTM and its training procedures were further improved on my Swiss LSTM grants at IDSIA through the work of my later students Felix Gers, Alex Graves, and others. A milestone was the "*vanilla LSTM architecture*" with forget gate[LSTM2]—the LSTM variant of 1999-2000 that everybody is using today, e.g., in Google's Tensorflow. 2005 saw the first publication of LSTM with full backpropagation through time and of bi-directional LSTM[LSTM3] (now widely used)."

Some Historical Remarks on HighwayNet

- [**HW1**] R. K. Srivastava, K. Greff, J. Schmidhuber. Highway networks. Preprints arXiv:1505.00387 (May 2015) and arXiv:1507.06228 (July 2015). Also at NIPS 2015. *The first working very deep feedforward nets with over 100 layers (previous NNs had at most a few tens of layers). Let g, t, h, denote non-linear differentiable functions. Each non-input layer of a highway net computes g(x)x + t(x)h(x), where x is the data from the previous layer. (Like LSTM with forget gates*[LSTM2] *for RNNs.) Resnets*[HW2] *are a special case of this where the gates are always open: g(x)=t(x)=const=1. Highway Nets perform roughly as well as ResNets*[HW2] *on ImageNet.*[HW3] *Highway layers are also often used for natural language processing, where the simpler residual layers do not work as well.*[HW3]
- [**HW2**] He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. Preprint arXiv:1512.03385 (Dec 2015). *Residual nets are a special case of Highway Nets*[HW1] *where the gates are always open: g(x)=1 (a typical highway net initialization) and t(x)=1.*
- [**HW3**] K. Greff, R. K. Srivastava, J. Schmidhuber. Highway and Residual Networks learn Unrolled Iterative Estimation. Preprint arxiv:1612.07771 (2016). Also at ICLR 2017.

Some Historical Remarks on the "credit" debate



- [**VAN1**] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, TUM, 1991 (advisor J. Schmidhuber). Link: http://www.idsia.ch/~juergen/SeppHochreiter1991ThesisAdvisorSchmidhuber.pdf
- [**VAN2**] Y. Bengio, P. Simard, P. Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE TNN 5(2), p 157-166, 1994
- [**VAN3**] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, eds., A Field Guide to Dynamical Recurrent Neural Networks. IEEE press, 2001
- [**VAN4**] Y. Bengio. Neural net language models. Scholarpedia, 3(1):3881, 2008. Link: http://www.scholarpedia.org/article/Neural_net_language_models?CachedSimilar13
- **J. SchmidHuber** (https://people.idsia.ch/~juergen/deep-learning-miraculous-year-1990-1991.html#Sec.%203):
  - "As a part of his thesis, Sepp implemented the Neural History Compressor above (see Sec. 1) and other RNN-based systems (see Sec. 11). However, he did much more: His work formally showed that deep NNs suffer from the now famous problem of vanishing or exploding gradients: in typical deep or recurrent networks, back-propagated error signals either shrink rapidly, or grow out of bounds. In both cases, learning fails. This analysis led to basic principles of what's now called LSTM (see Sec. 4)."
  - "Interestingly, in 1994, others published results[VAN2] essentially identical to the 1991 vanishing gradient results of Sepp.[VAN1] Even after a common publication[VAN3] the first author of reference[VAN2] published papers[VAN4] that cited only their own 1994 paper but not Sepp's original work."

The Nobel Prize in Physics 2024

Ill. Niklas Elmehed © Nobel Prize Outreach

John J. Hopfield

Prize share: 1/2

Ill. Niklas Elmehed © Nobel Prize Outreach

Geoffrey E. Hinton

Prize share: 1/2

The Nobel Prize in Physics 2024 was awarded jointly to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks"

To cite this section
MLA style: The Nobel Prize in Physics 2024. NobelPrize.org. Nobel Prize Outreach AB 2024. Wed. 16 Oct 2024. <https://www.nobelprize.org/prizes/physics/2024/summary/>

The Nobel Prize in Chemistry 2024

Ill. Niklas Elmehed © Nobel Prize Outreach

David Baker

Prize share: 1/2

Ill. Niklas Elmehed © Nobel Prize Outreach

Demis Hassabis

Prize share: 1/4

Ill. Niklas Elmehed © Nobel Prize Outreach

John M. Jumper

Prize share: 1/4

The Nobel Prize in Chemistry 2024 was divided, one half awarded to David Baker "for computational protein design", the other half jointly to Demis Hassabis and John M. Jumper "for protein structure prediction"

To cite this section
MLA style: The Nobel Prize in Chemistry 2024. NobelPrize.org. Nobel Prize Outreach AB 2024. Wed. 16 Oct 2024. <https://www.nobelprize.org/prizes/chemistry/2024/summary/>

Jurgen's critics and Geoff's response

Jürgen Schmidhuber @SchmidhuberAI

The #NobelPrizeinPhysics2024 for Hopfield & Hinton rewards plagiarism and incorrect attribution in computer science. It's mostly about Amari's "Hopfield network" and the "Boltzmann Machine."

1. The Lenz-Ising recurrent architecture with neuron-like elements was published in 1925 [L20][I24][I25]. In 1972, Shun-Ichi Amari made it adaptive such that it could learn to associate input patterns with output patterns by changing its connection weights [AMH1]. However, Amari is only briefly cited in the "Scientific Background to the Nobel Prize in Physics 2024." Unfortunately, Amari's net was later called the "Hopfield network." Hopfield republished it 10 years later [AMH2], without citing Amari, not even in later papers.

2. The related Boltzmann Machine paper by Ackley, Hinton, and Sejnowski (1985) [BM] was about learning internal representations in hidden units of neural networks (NNs) [S20]. It didn't cite the first working algorithm for deep learning of internal representations by Ivakhnenko & Lapa (Ukraine, 1965)[DEEP1-2][HIN]. It didn't cite Amari's separate work (1967-68)[GD1-2] on learning internal representations in deep NNs end-to-end through stochastic gradient descent (SGD). Not even the later surveys by the authors [S20][DL3][DLP] nor the "Scientific Background to the Nobel Prize in Physics 2024" mention these origins of deep learning. ([BM] also did not cite relevant prior work by Sherrington & Kirkpatrick [SK75] & Glauber [G63].)

3. The Nobel Committee also lauds Hinton et al.'s 2006 method for layer-wise pretraining of deep NNs (2006) [UN4]. However, this work neither cited the original layer-wise training of deep NNs by Ivakhnenko & Lapa (1965)[DEEP1-2] nor the original work on unsupervised pretraining of deep NNs (1991) [UN0-1][DLP].

4. The "Popular information" says: "At the end of the 1960s, some discouraging theoretical results caused many researchers to suspect that these neural networks would never be of any real use." However, deep learning research was obviously alive and kicking in the 1960s-70s, especially outside of the Anglosphere [DEEP1-2][GD1-3][CNN1][DL1-2][DLP][DLH].

5. Many additional cases of plagiarism and incorrect attribution can be found in the following reference [DLP], which also contains the other references above. One can start with Sec. 3:

[DLP] J. Schmidhuber (2023). How 3 Turing awardees republished key methods and ideas whose creators they failed to credit. Technical Report IDSIA-23-23, Swiss AI Lab IDSIA, 14 Dec 2023. people.idsia.ch/~juergen/ai-pr...

See also the following reference [DLH] for a history of the field:

[DLH] J. Schmidhuber (2022). Annotated History of Modern AI and Deep Learning. Technical Report IDSIA-22-22, IDSIA, Lugano, Switzerland, 2022. Preprint arXiv:2212.11279. people.idsia.ch/~juergen/deep-... (This extends the 2015 award-winning survey people.idsia.ch/~juergen/deep-...)

10:30 PM · Oct 9, 2024 · **1M** Views

https://x.com/SchmidhuberAI/status/1844022724328394780

https://x.com/hardmaru/status/1253189802452647936

[–] **geoffhinton** Google Brain 46 points 4 hours ago

Having a public debate with Schmidhuber about academic credit is not advisable because it just encourages him and there is no limit to the time and effort that he is willing to put into trying to discredit his perceived rivals. He has even resorted to tricks like having multiple aliases in Wikipedia to make it look as if other people are agreeing with what he says. The page on his website about Alan Turing is a nice example of how he goes about trying to diminish other people's contributions.

Despite my own best judgement, I feel that I cannot leave his charges completely unanswered so I am going to respond once and only once. I have never claimed that I invented backpropagation. David Rumelhart invented it independently long after people in other fields had invented it. It is true that when we first published we did not know the history so there were previous inventors that we failed to cite. What I have claimed is that I was the person to clearly demonstrate that backpropagation could learn interesting internal representations and that this is what made it popular. I did this by forcing a neural net to learn vector representations for words such that it could predict the next word in a sequence from the vector representations of the previous words. It was this example that convinced the Nature referees to publish the 1986 paper.

It is true that many people in the press have said I invented backpropagation and I have spent a lot of time correcting them. Here is an excerpt from the 2018 book by Michael Ford entitled "Architects of Intelligence":

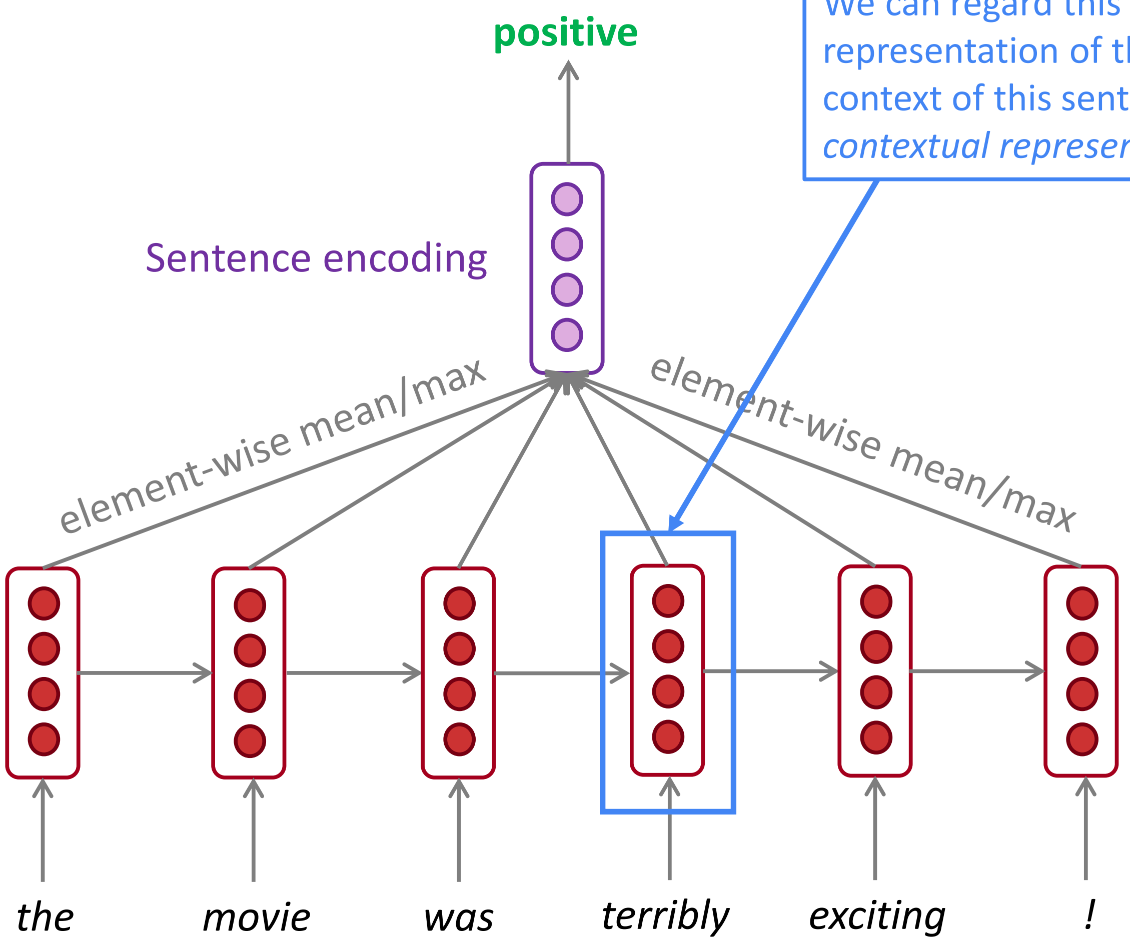
"Lots of different people invented different versions of backpropagation before David Rumelhart. They were mainly independent inventions and it's something I feel I have got too much credit for. I've seen things in the press that say that I invented backpropagation, and that is completely wrong. It's one of these rare cases where an academic feels he has got too much credit for something! My main contribution was to show how you can use it for learning distributed representations, so I'd like to set the record straight on that."

Maybe Juergen would like to set the record straight on who invented LSTMs?

Bi-Direction

Motivation of Bidirection

<u>Task: Sentiment Classification</u>



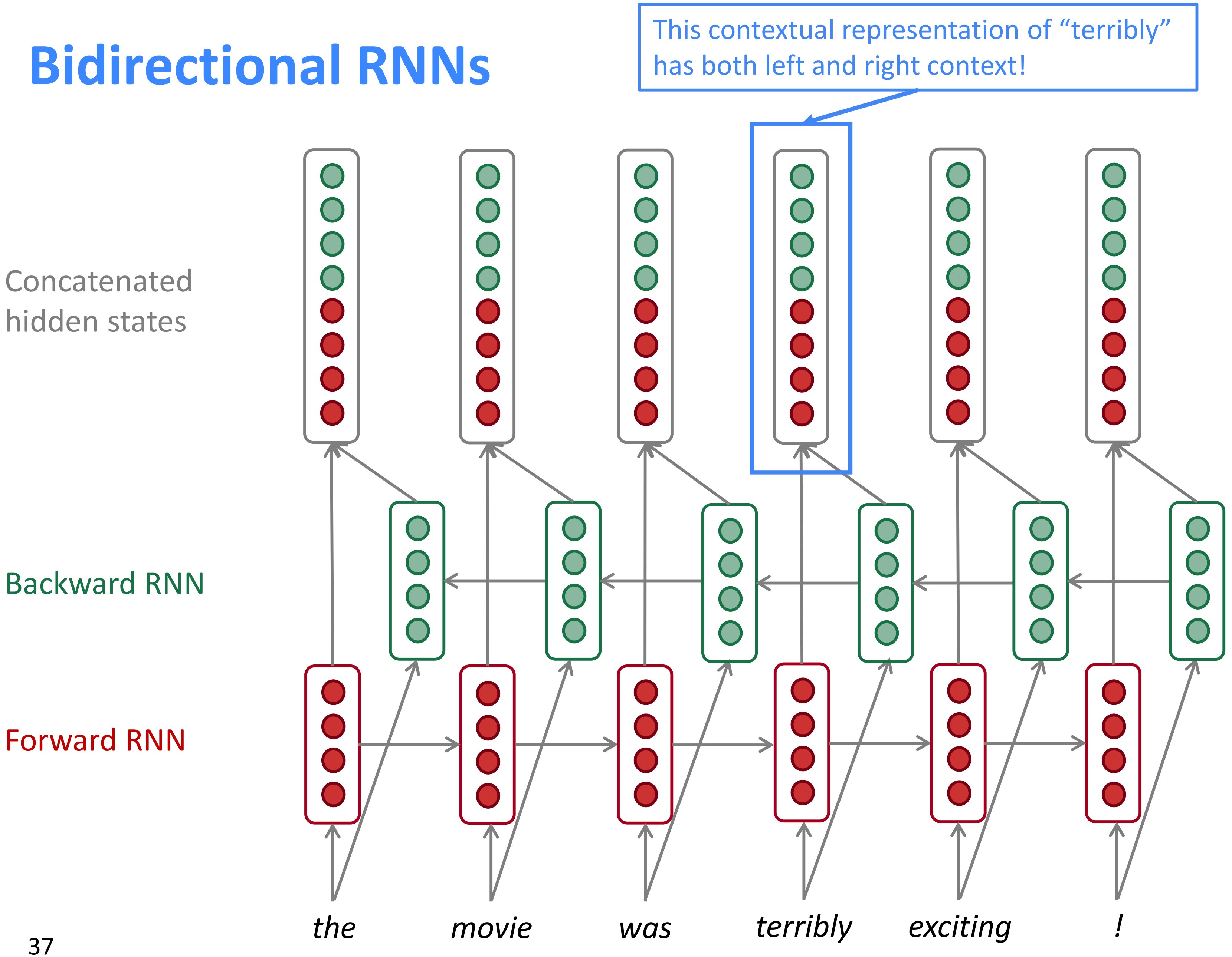
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

Bidirectional RNNs



Bidirectional RNN: simplified diagram

On timestep t:

This is a general notation to mean "compute one forward step of the RNN" – it could be a vanilla, LSTM or GRU computation.

Forward RNN $\overrightarrow{\boldsymbol{h}}^{(t)} = \boxed{\mathrm{RNN}_{\mathrm{FW}}}(\overrightarrow{\boldsymbol{h}}^{(t-1)}, \boldsymbol{x}^{(t)})$

Backward RNN $\overleftarrow{\boldsymbol{h}}^{(t)} = \mathrm{RNN}_{\mathrm{BW}}(\overleftarrow{\boldsymbol{h}}^{(t+1)}, \boldsymbol{x}^{(t)})$

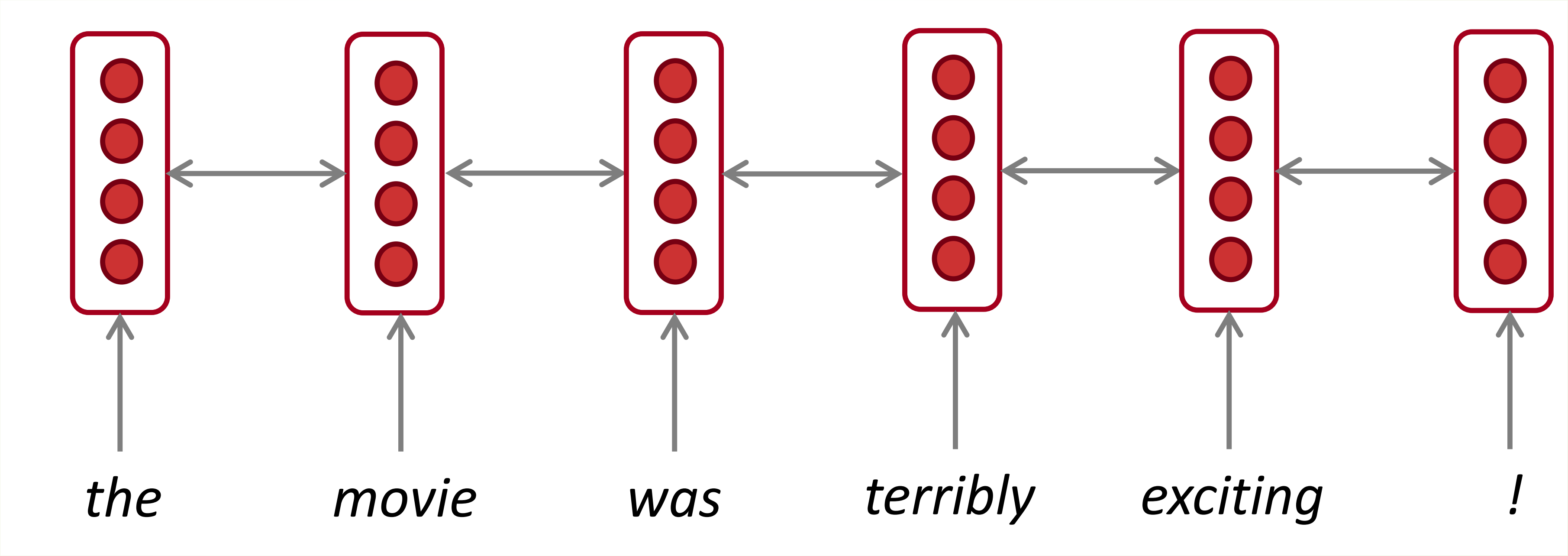
Generally, these two RNNs have separate weights

Concatenated hidden states $\boxed{\boldsymbol{h}^{(t)}} = [\overrightarrow{\boldsymbol{h}}^{(t)}; \overleftarrow{\boldsymbol{h}}^{(t)}]$

We regard this as "the hidden state" of a bidirectional RNN. This is what we pass on to the next parts of the network.

Bidirectional RNN: simplified diagram

- The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states.



Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**.
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.
  - For example, **GPT** (Generative Pre-trained Transformer) is **unidirectional** generative model
- If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).
  - For example, **BERT** (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality.

History note

- In 2013-2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include: handwriting recognition, speech
  - recognition, machine translation, parsing, image captioning
  - LSTM became the dominant approach
- Now (2019), other approaches (e.g. Transformers) have become more dominant for certain tasks.
  - For example in **WMT** (a MT conference + competition):
  - In WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2018, the report contains "RNN" 9 times and "**Transformer**" 63 times

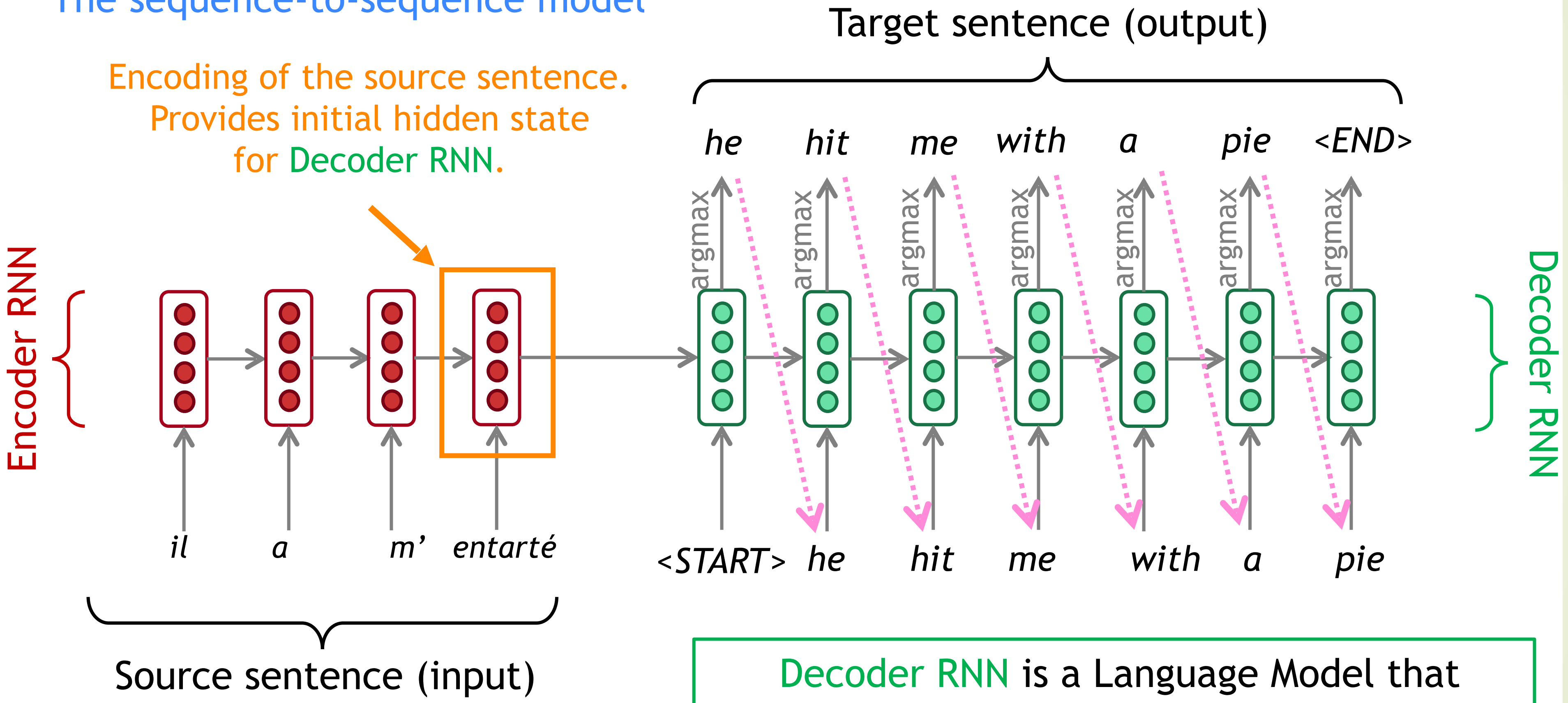
  - **Source:** "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, http://www.statmt.org/wmt16/pdf/W16-2301.pdf
  - **Source:** "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, http://www.statmt.org/wmt18/pdf/WMT028.pdf

Neural Machine Translation

Machine Translation using Neural Networks

Neural Machine Translation (NMT)

The sequence-to-sequence model



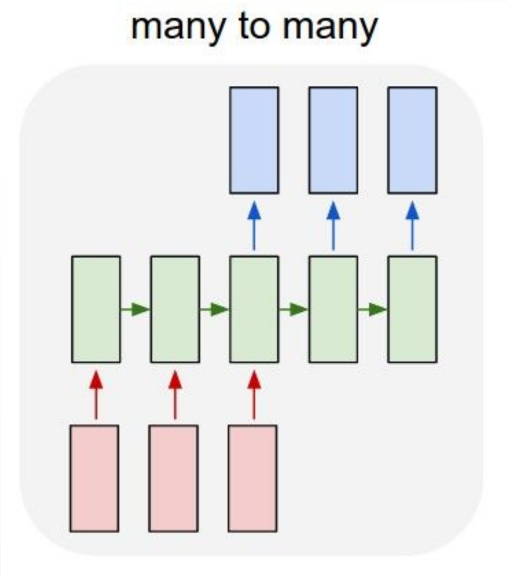
Encoder RNN produces an *encoding* of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

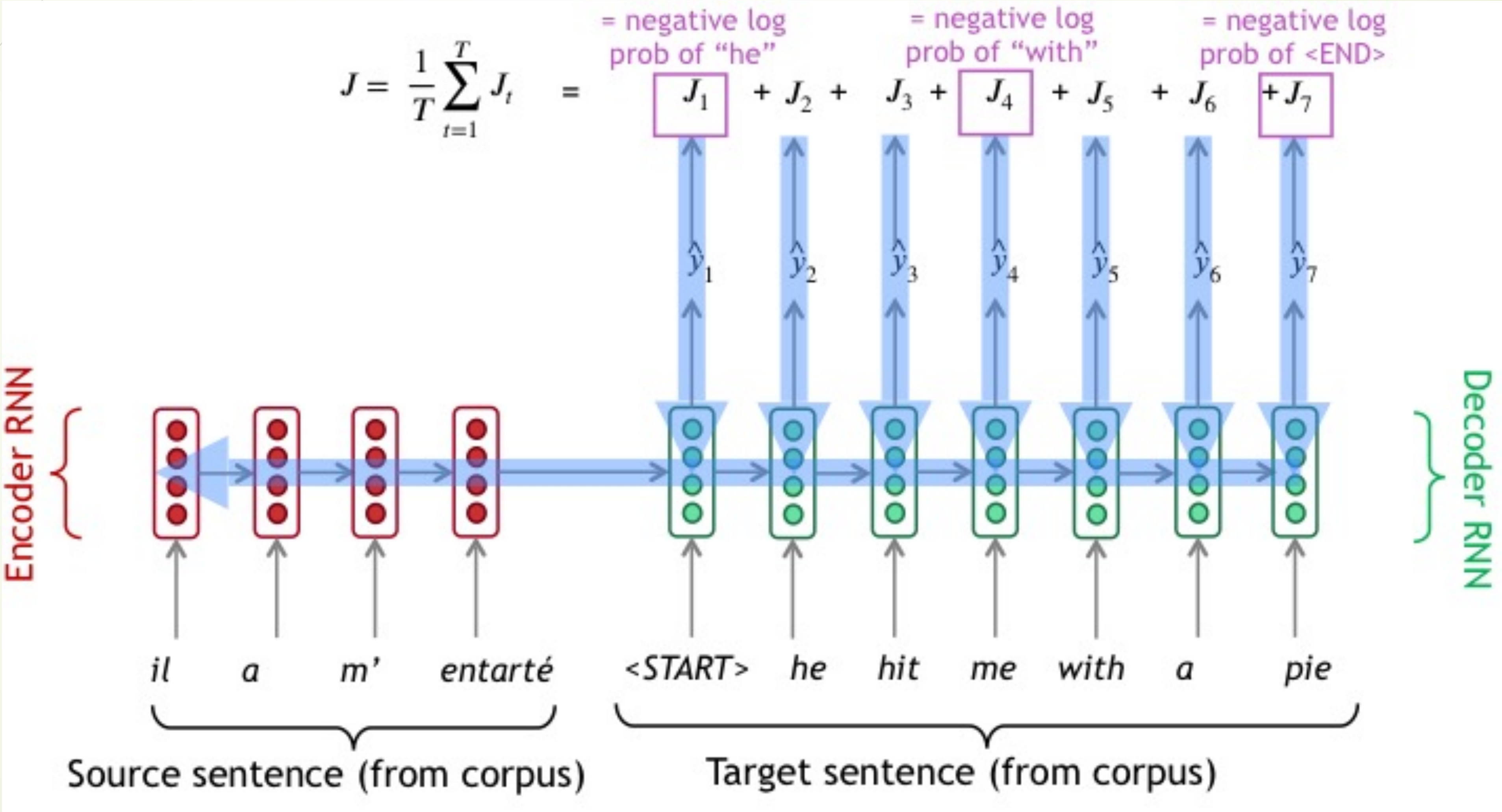
Note: This diagram shows **test time** behavior: decoder output is fed in ·····> as next step's input

Sequence-to-sequence is versatile!

- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

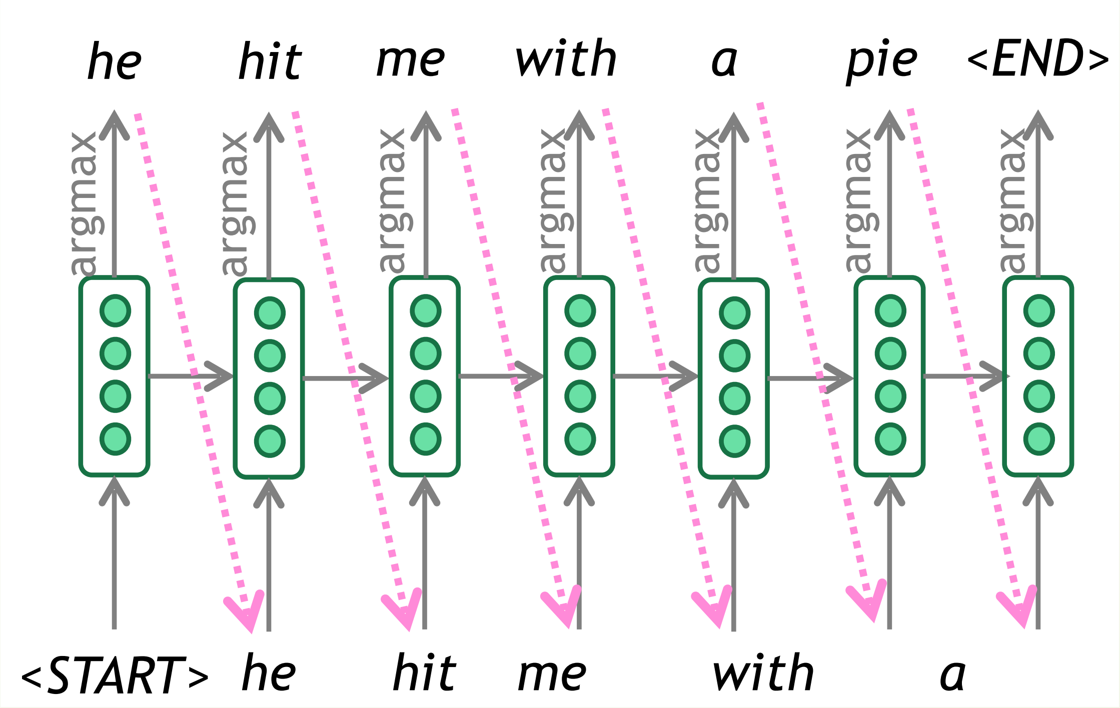


Training a NMT system by BP



Greedy Decoding

- We generate (or "decode") the target sentence by taking **argmax** on each step of the decoder, called **greedy decoding** (take most probable word on each step)
- It may not correct once wrong decisions are made



Beam Search Decoding

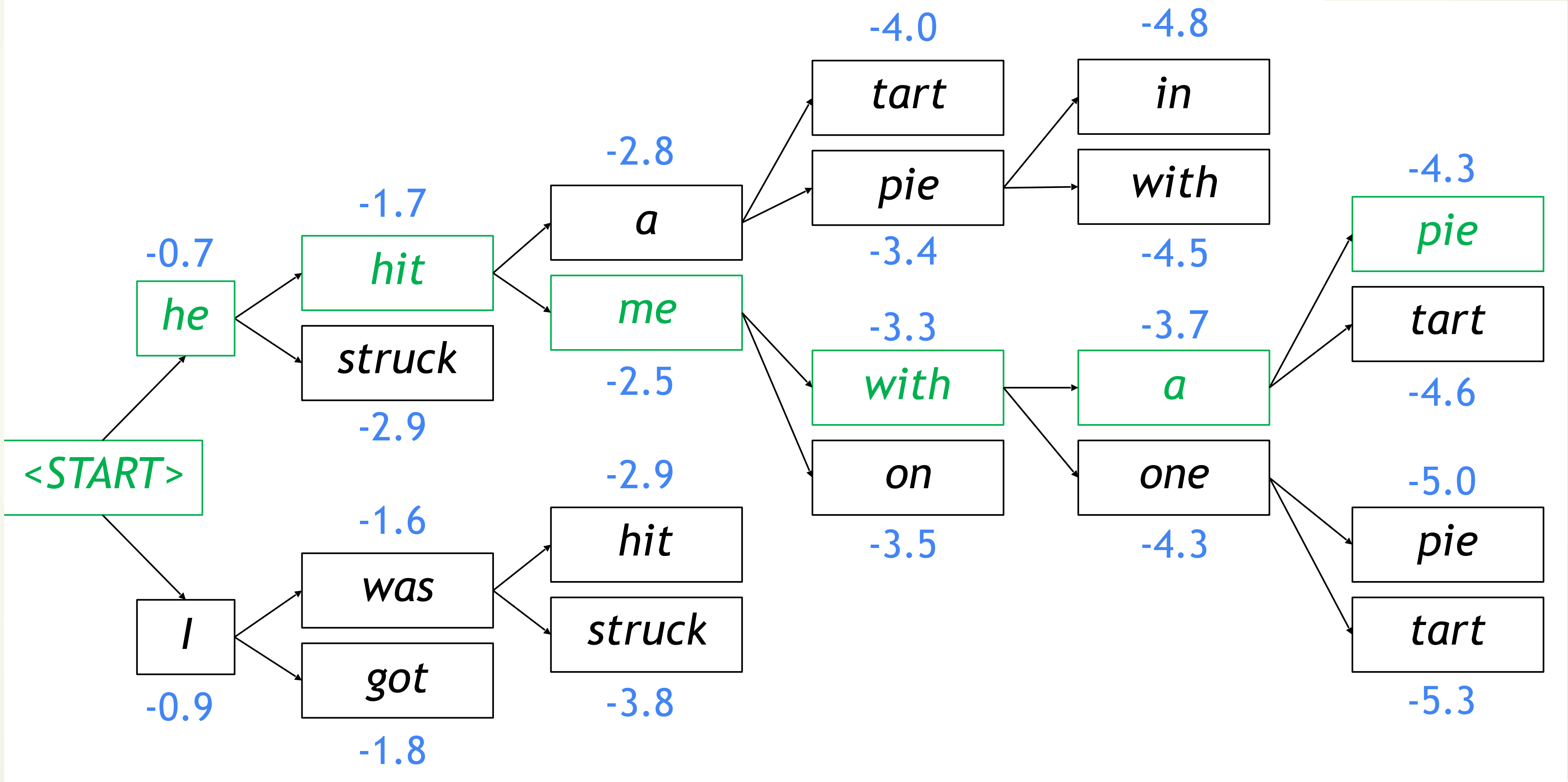
- Core idea: On each step of decoder, keep track of the ***k* most probable** partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)
- A hypothesis (y(1),…,y(t)) has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

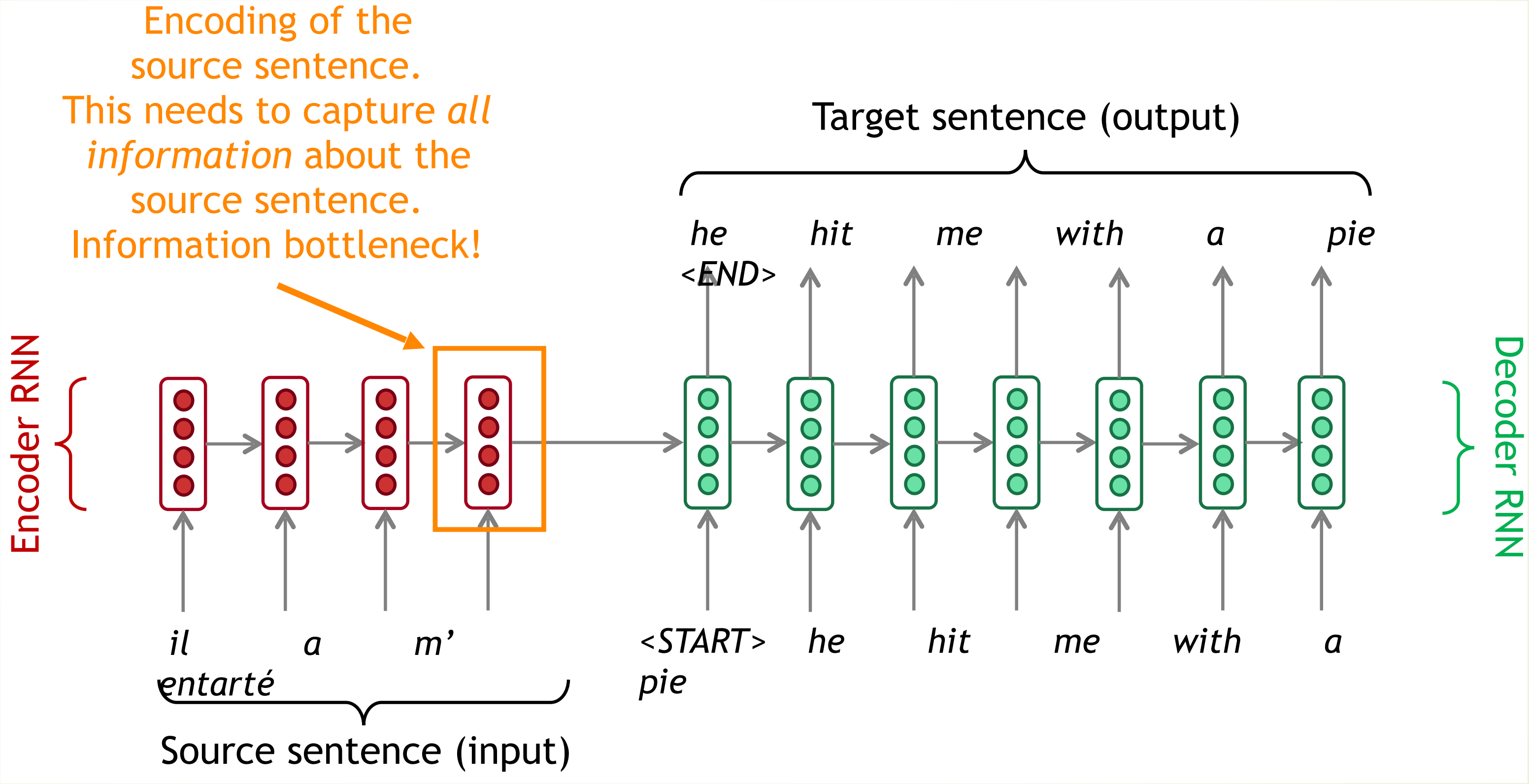
Beam search decoding example:

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$



For each of the *k* hypotheses, find top *k* next words and calculate scores

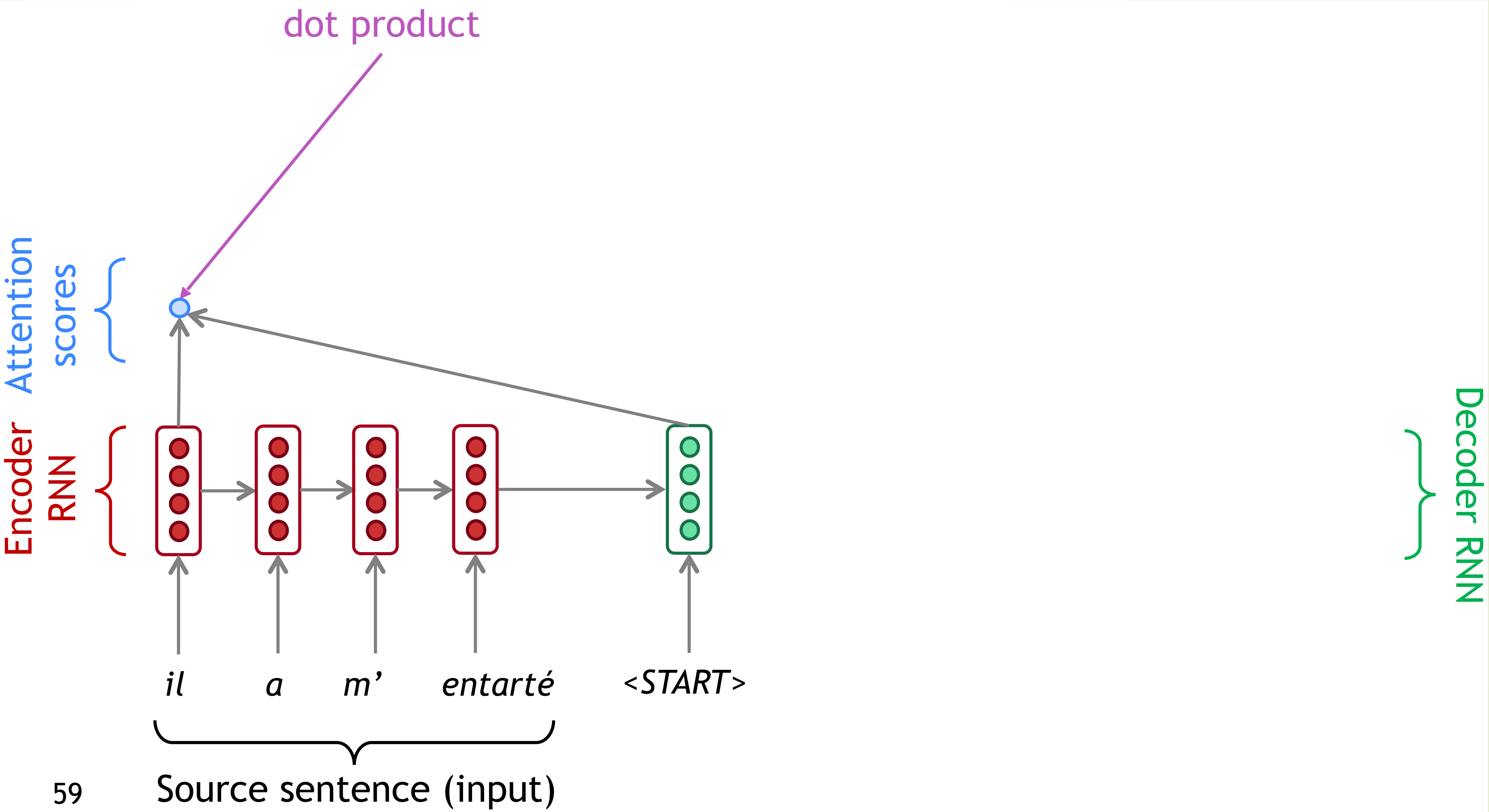
Sequence-to-sequence: the bottleneck problem

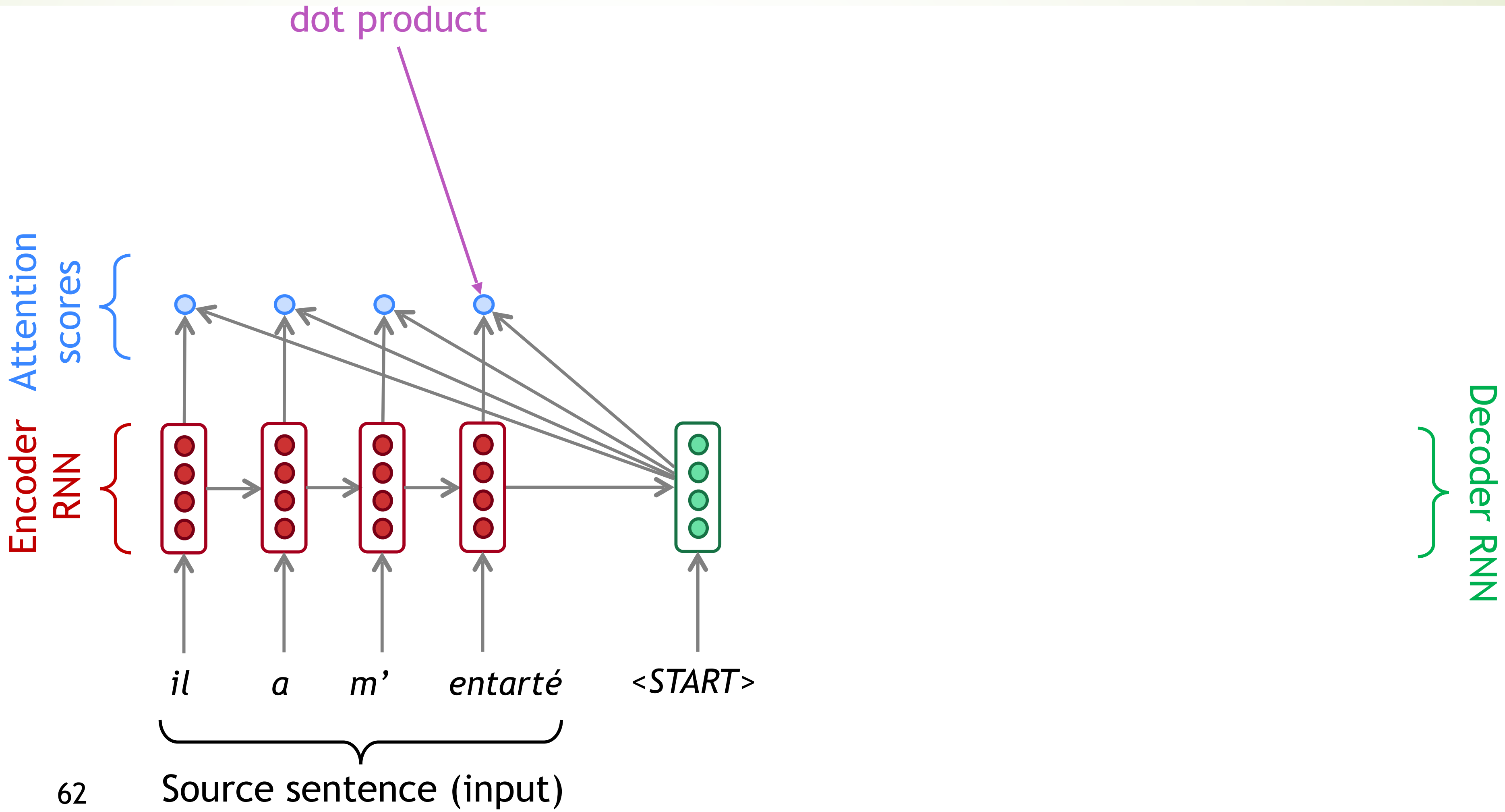


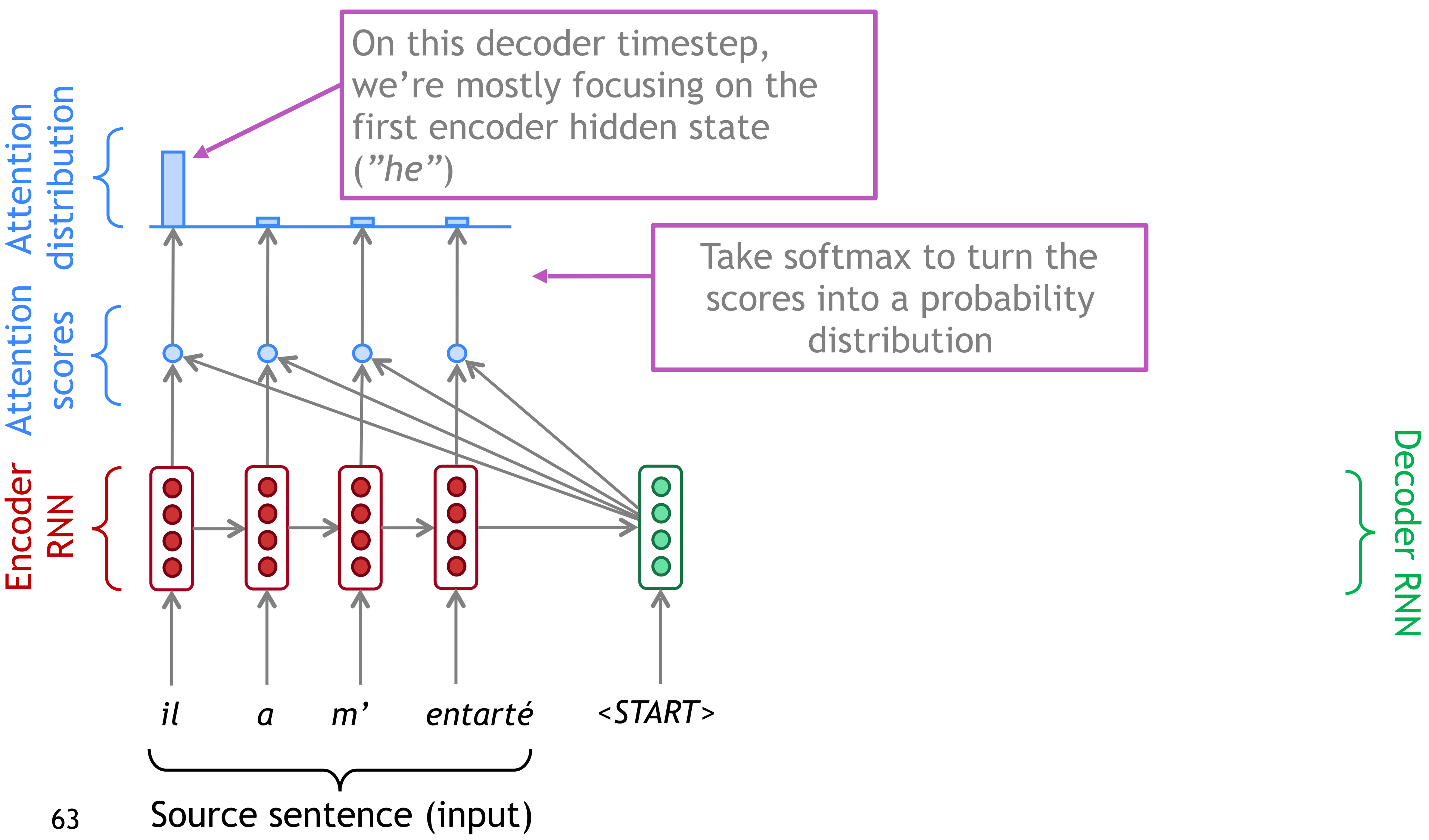
Attention Mechanism

It was firstly invented in computer vision, then to NLP.

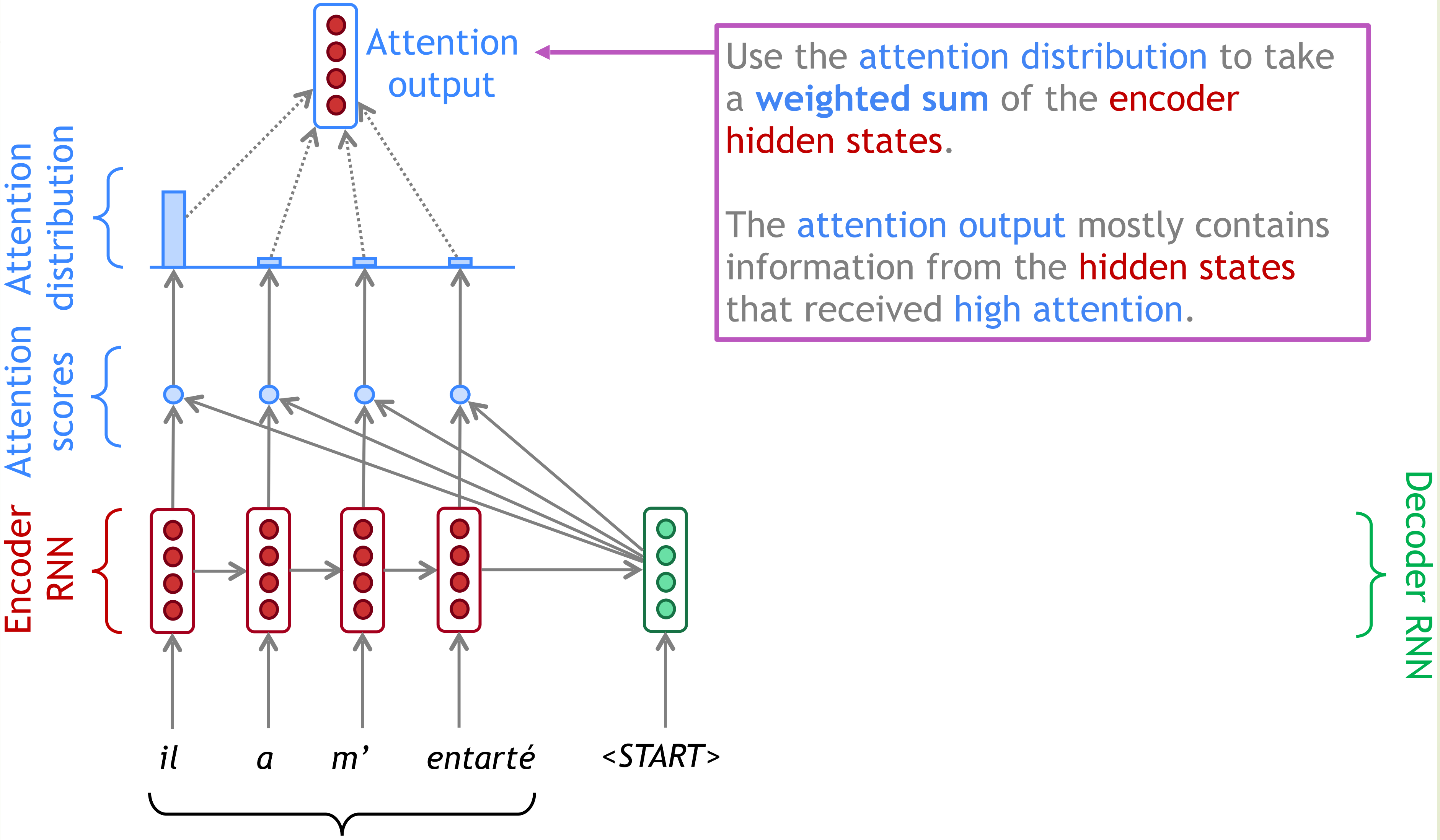
Sequence-to-sequence with attention




dot product
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source sentence (input)

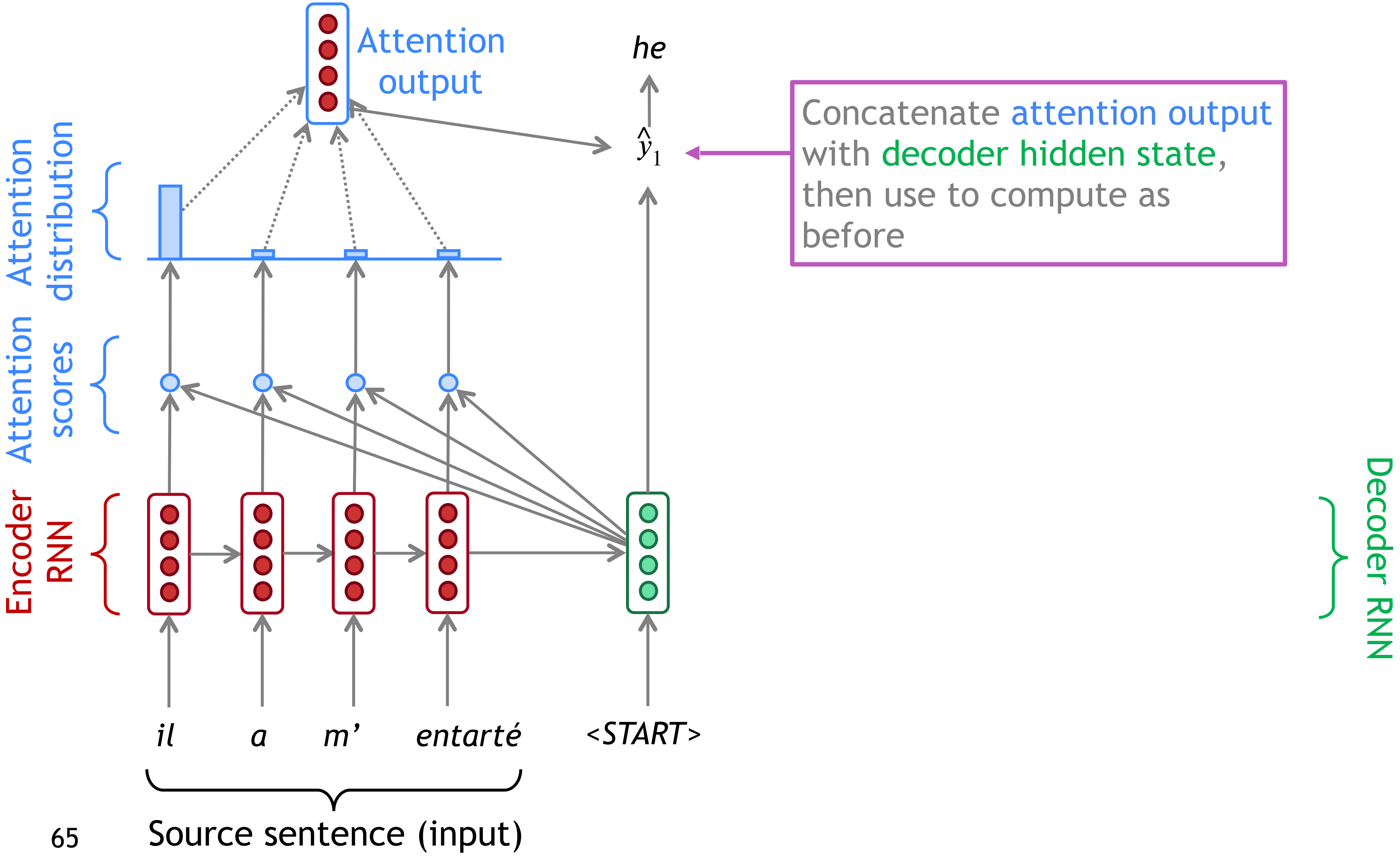


On this decoder timestep, we're mostly focusing on the first encoder hidden state ("he")
Take softmax to turn the scores into a probability distribution
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
Source sentence (input)

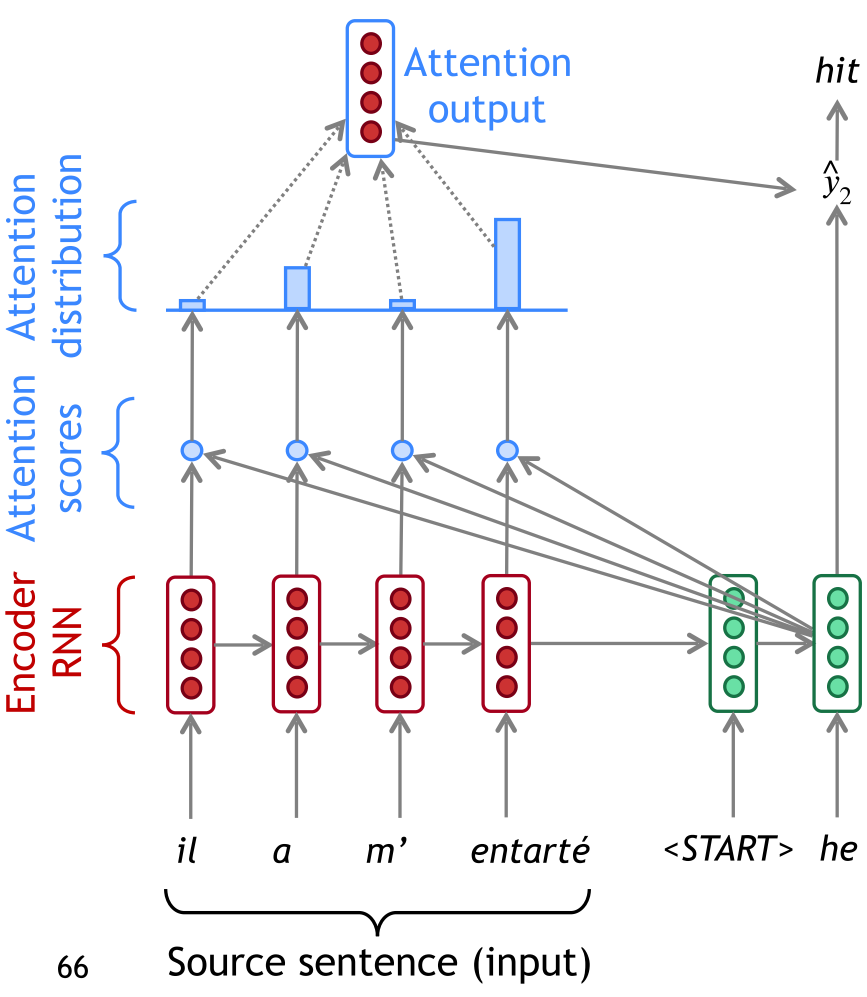


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

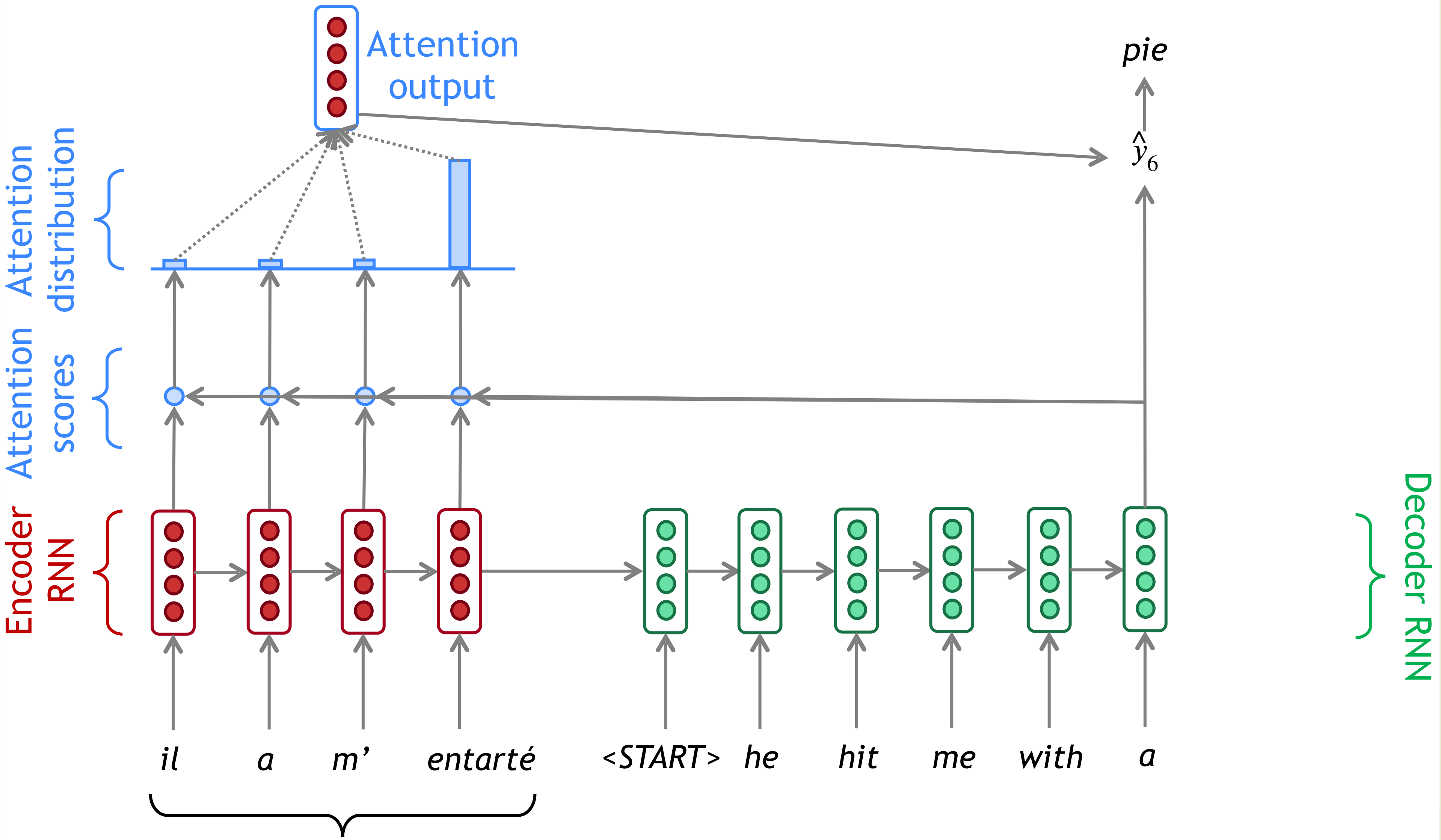
The attention output mostly contains information from the hidden states that received high attention.



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before


Attention output
hit
$\hat{y}_2$
Attention distribution
Attention scores
Encoder RNN
il
a
m'
entarté
<START>
he
Source sentence (input)

Decoder RNN


Attention output
pie
$\hat{y}_6$
Attention distribution
Attention scores
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
me
with
a
Source sentence (input)

Attention in Equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output

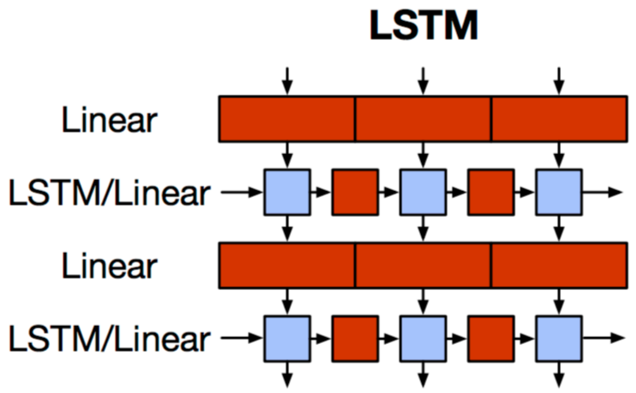
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Motivation of Transformer

- We want **parallelization** but RNNs are inherently sequential



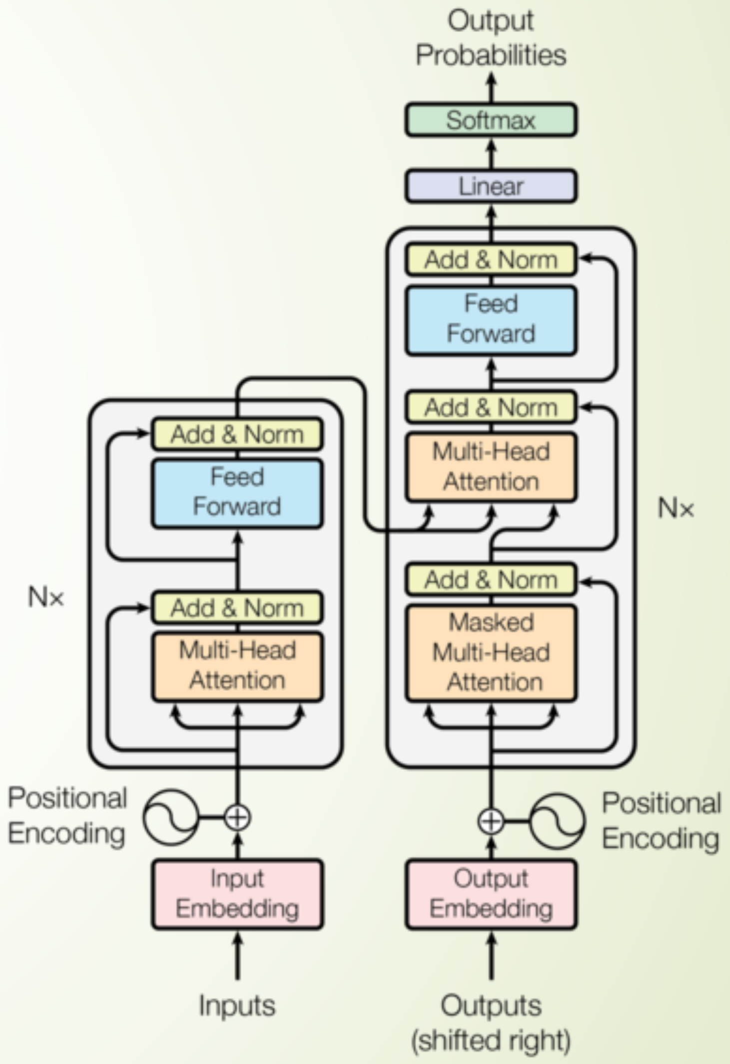
- Despite LSTMs, RNNs generally need attention mechanism to deal with long range dependencies – **path length** between states grows with distance otherwise
- But if **attention** gives us access to any state… maybe we can just use attention and don't need the RNN?
- And then NLP can have deep models … and solve our vision envy

Transformer

"Attention is all you need"

Transformer (Vaswani et al. 2017) “Attention is all you need”

- https://arxiv.org/pdf/1706.03762.pdf
- **Non-recurrent** sequence-to-sequence model
- A **deep** model with a sequence of **attention**-based transformer blocks
- Depth allows a certain amount of lateral information transfer in understanding sentences, in slightly unclear ways
- Final cost/error function is standard cross-entropy error on top of a softmax classifier
- Initially built for NMT:
  - Task: machine translation with parallel corpus
  - Predict each translated word



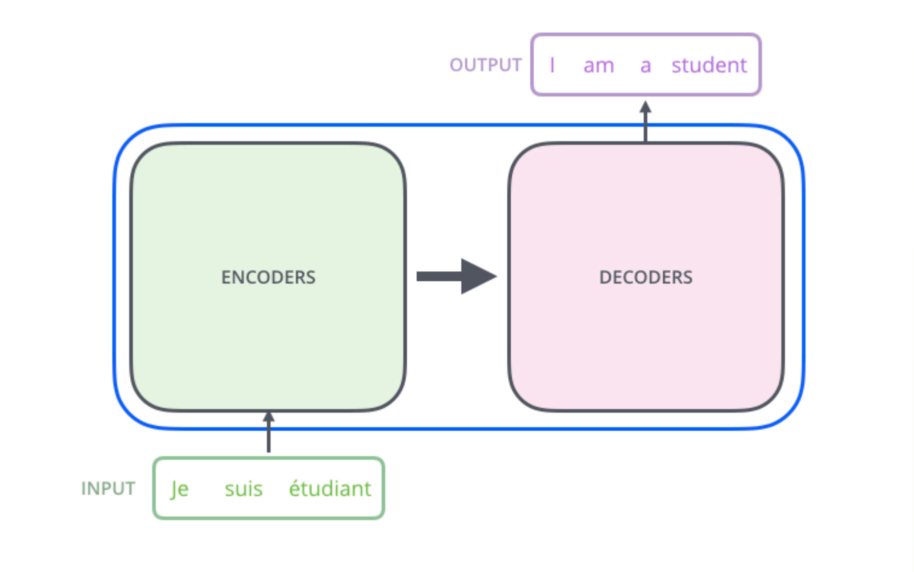
Transformer Pytorch Notebook

- Learning about transformers on your own?
- Key recommended resource:
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - The Annotated Transformer by Sasha Rush, a Jupyter Notebook using PyTorch that explains everything!

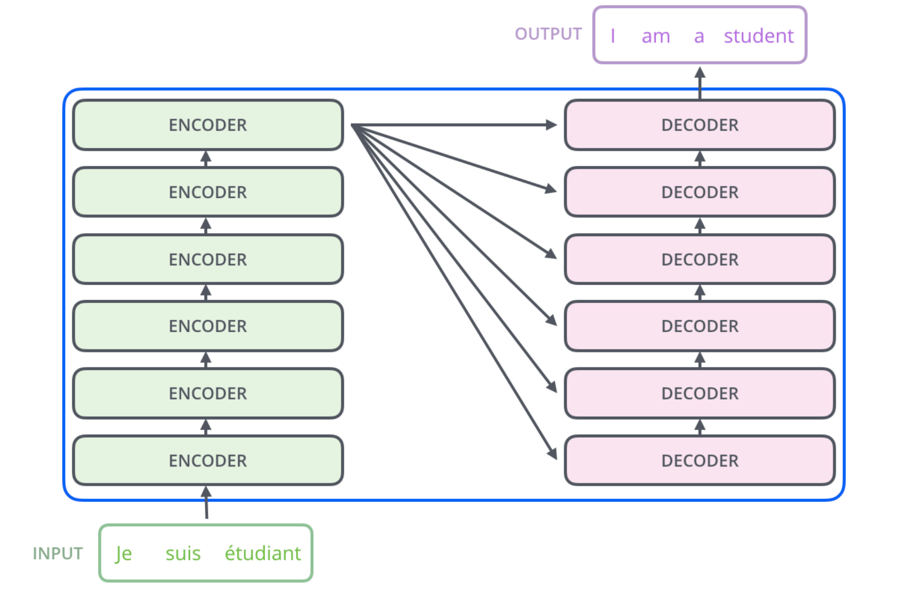
  - https://jalammar.github.io/illustrated-transformer/
  - Illustrated Transformer by Jay Alammar, a Cartoon about Transformer with attention visualization notebook based on Tensor2Tensor.

Encoder-Decoder Blocks

Encoder-Decoder

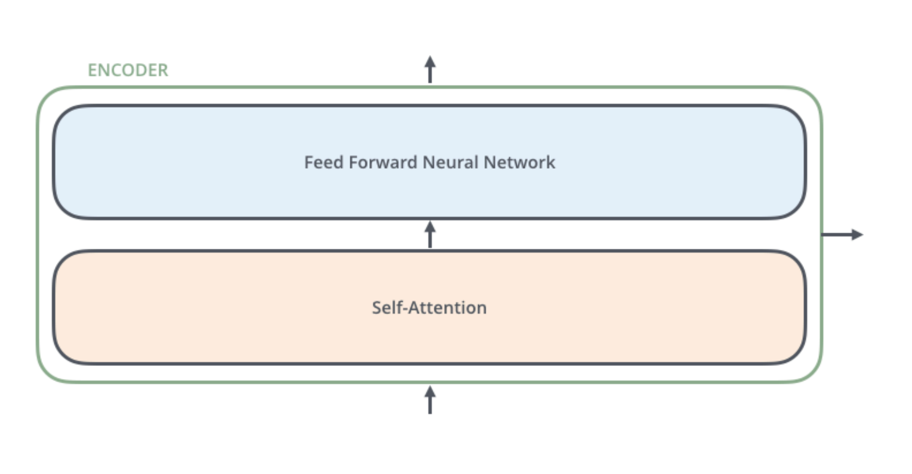


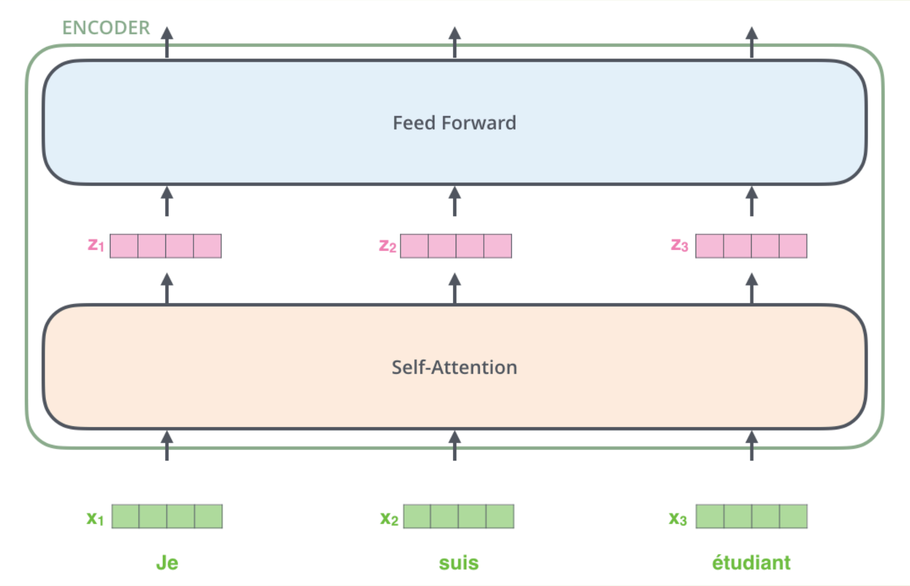
N=6 layers



Encoder has two layers

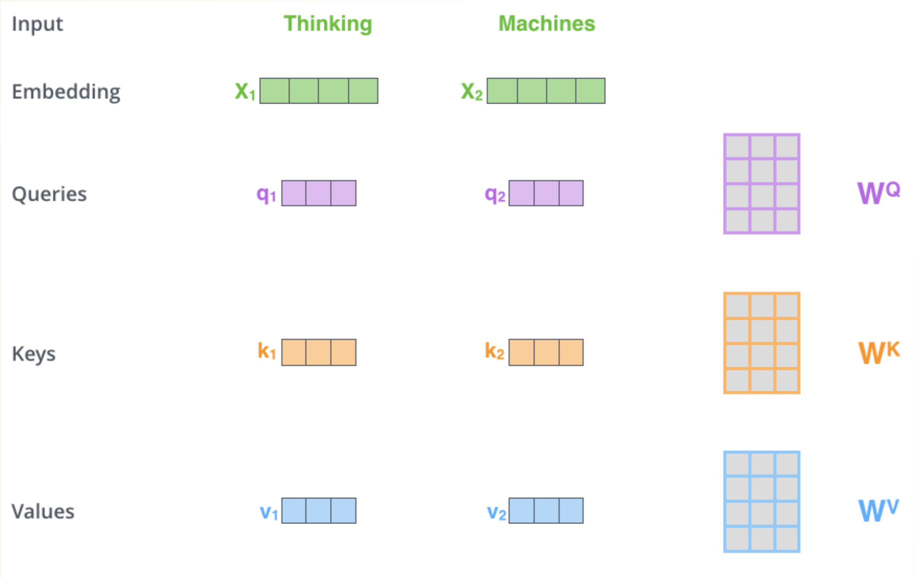
Self-Attention +
FeedForward



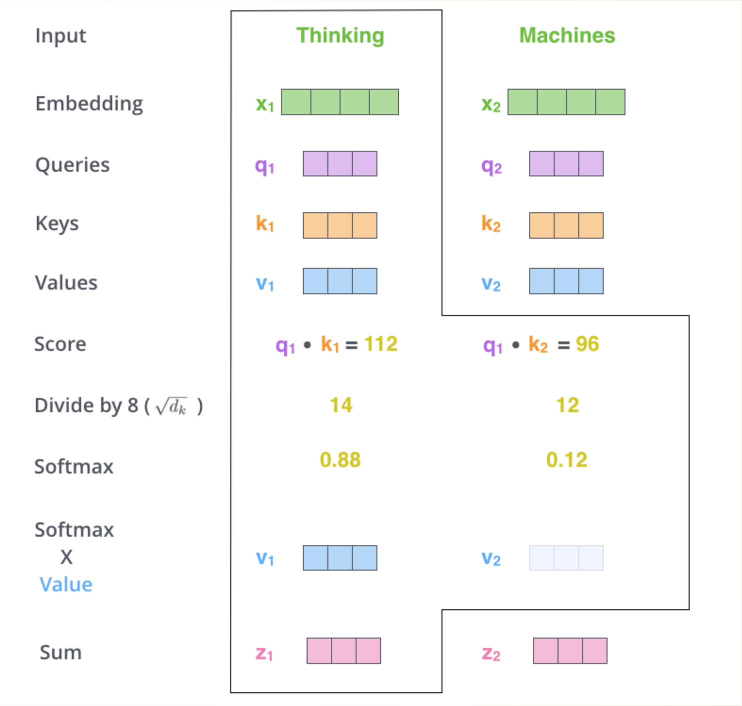


Attention Illustration

Embedding->(q,k,v)



Dot-Product Attention



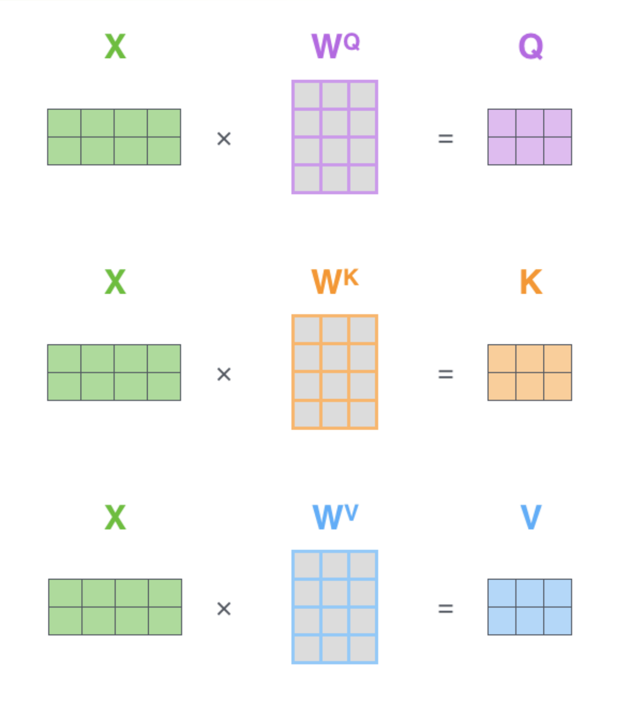
Dot-Product Self-Attention: Definition

- Inputs: a query q and a set of key-value (k-v) pairs, to an output
- Query, keys, values, and output are all vectors
- Output is weighted sum of values, where
  - Weight of each value is computed by an inner product of query and corresponding key
  - Queries and keys have same dimensionality d_k, value have d_v

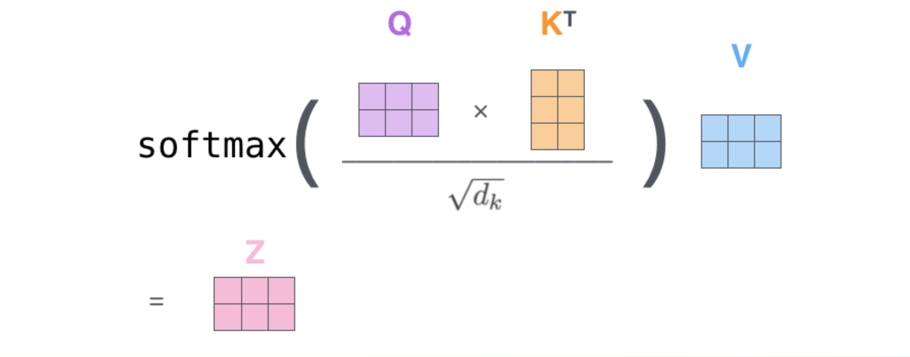
$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

Attention: Multiple Inputs

Matrix input



Scaled dot-product



Dot-Product Attention: Matrix Form

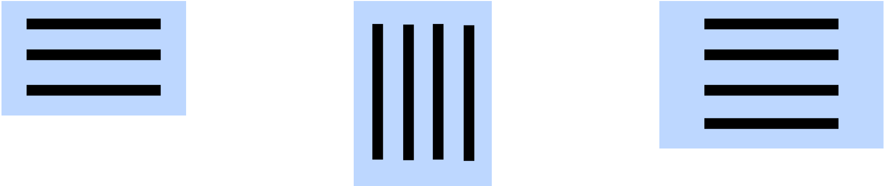
- When we have multiple queries q, we stack them in a matrix Q:

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

$$\longrightarrow \quad A(Q, K, V) = softmax(QK^T)V$$

$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$

softmax
row-wise



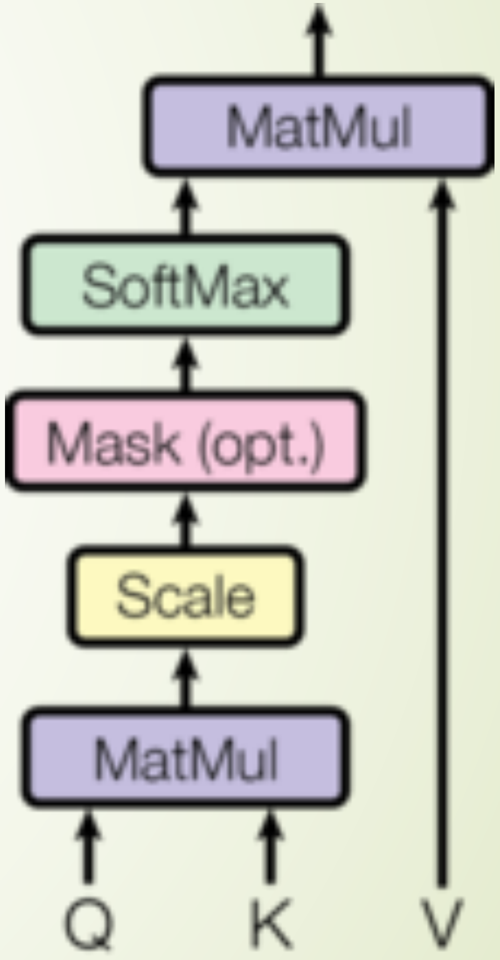
$= [|Q| \times d_v]$

Scaled Dot-Product Attention

- **Problem**: As d_k gets large, the variance of $q^T k$ increases
- some values inside the softmax get large
- the softmax gets very peaked
- hence its gradient gets smaller.

- Solution: Scale by length of query/key vectors:

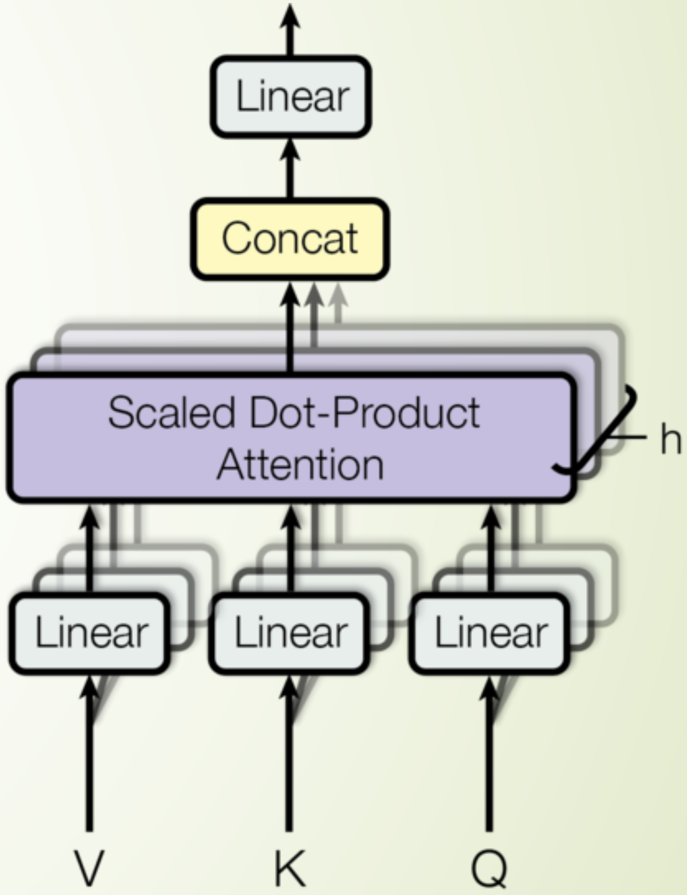
$$A(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$



Multi-head Attention

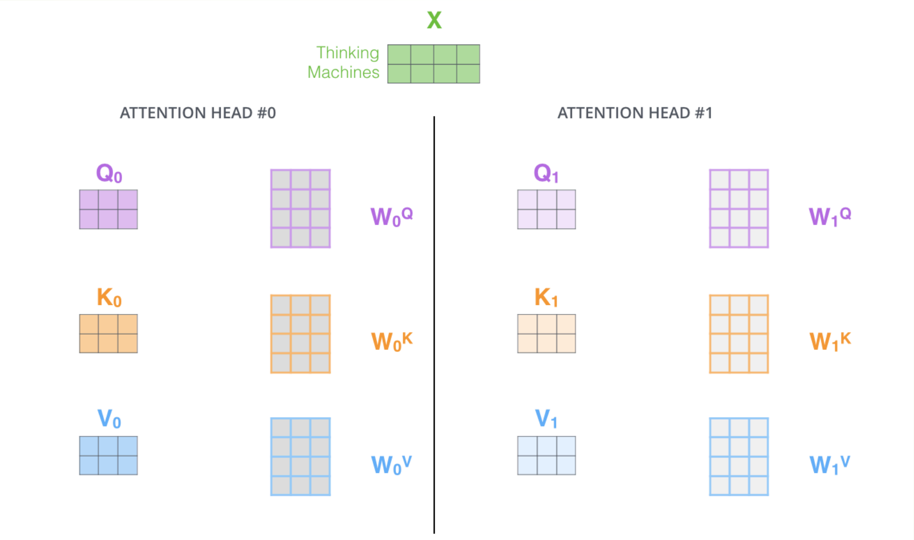
- **Problem** with simple self-attention:
  - Only one way for words to interact with one-another
- **Solution**: Multi-head attention
  - First map Q, K, V into h=8 many lower dimensional spaces via W matrices
  - Then apply attention, then concatenate outputs and pipe through linear layer
  - Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

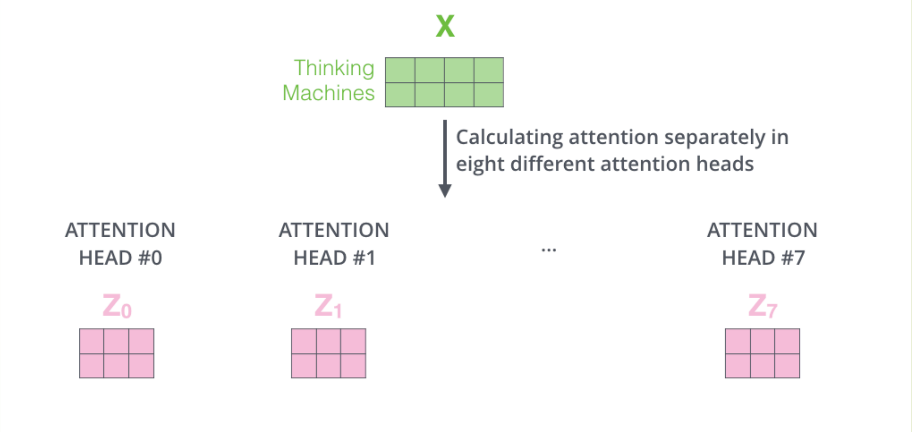


Multihead

2 heads

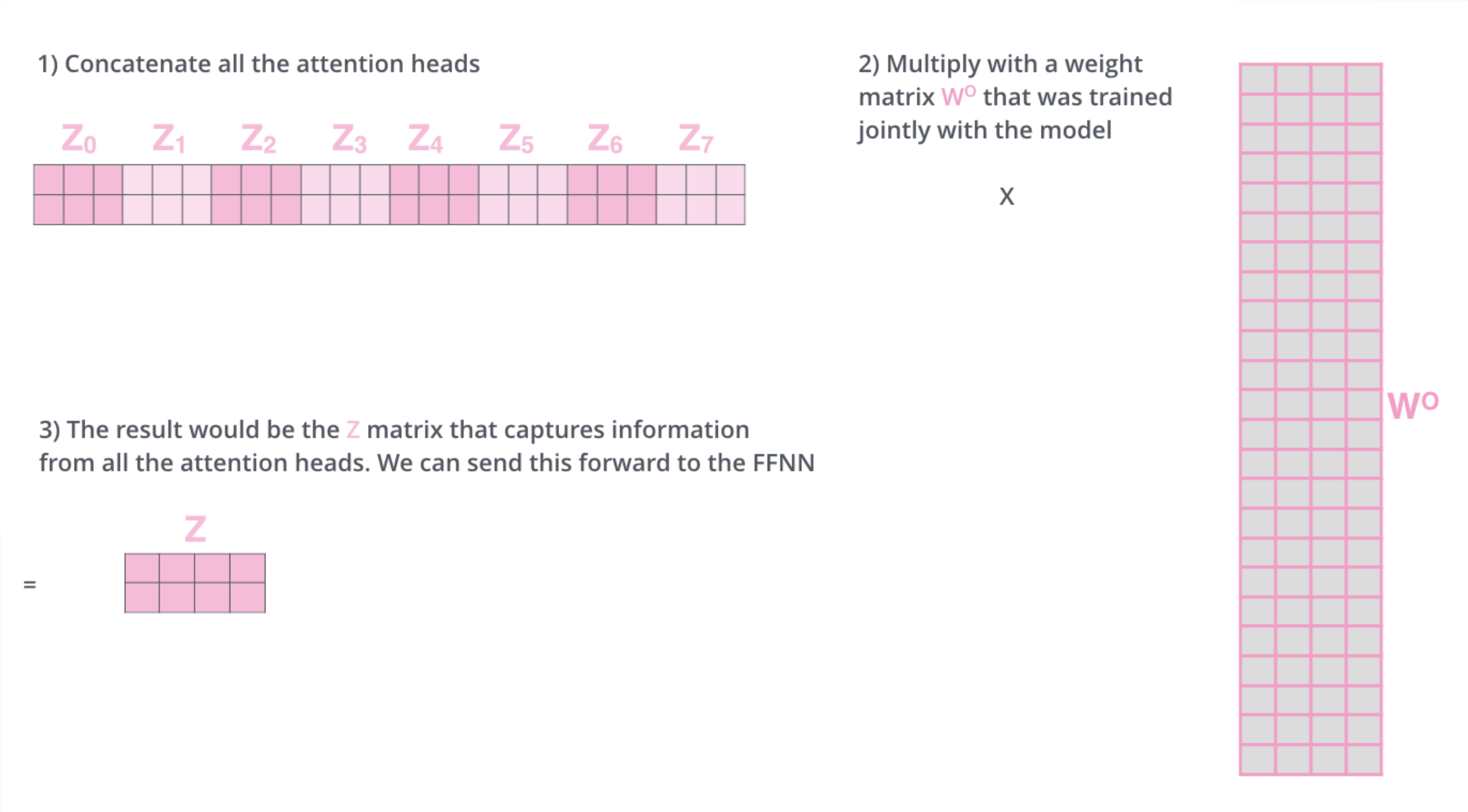


h=8 heads

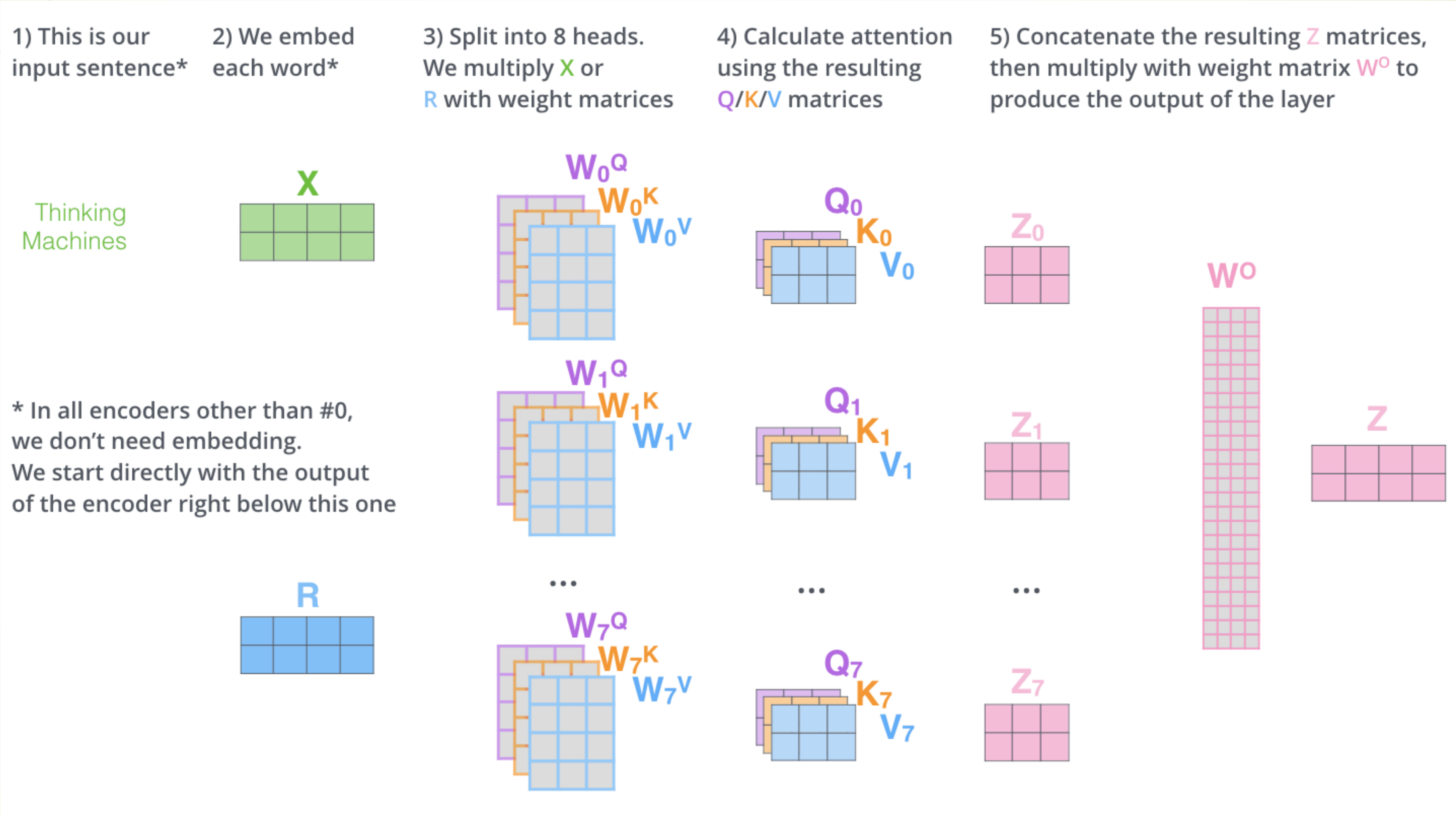


Concatenation

Linear

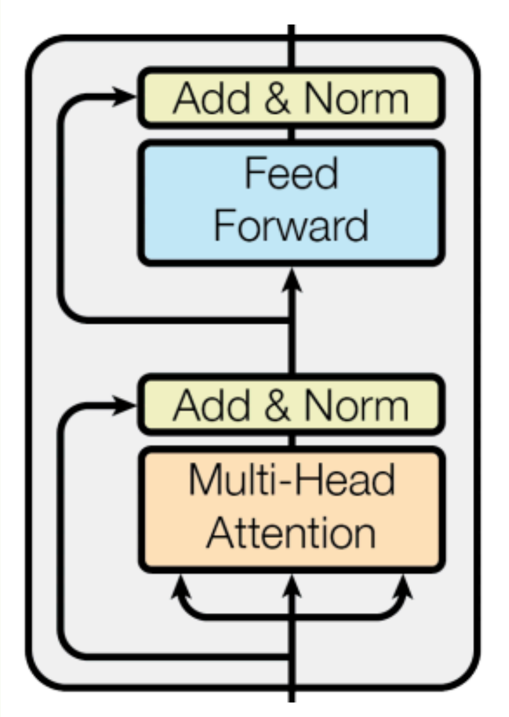


Multi-head Attention



A Transformer block

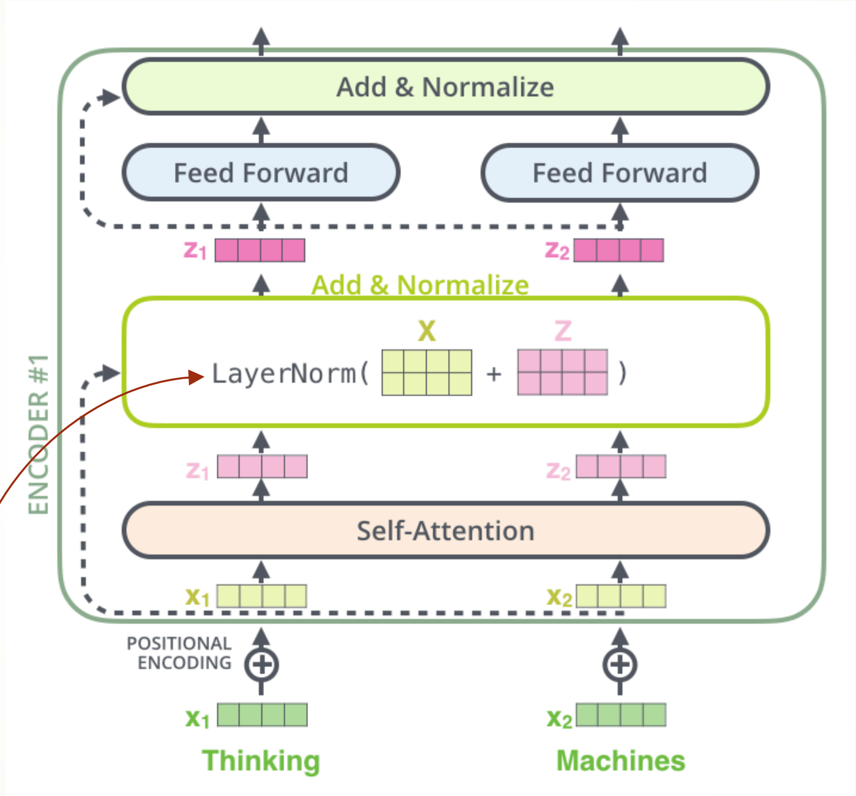
- Each block has two "sublayers"
  - Multihead attention
  - 2-layer feed-forward NNet (with ReLU)
- Each of these two steps also has:
  - Residual (short-cut) connection: x+sublayer(x)
  - LayerNorm(x+sublayer(x)) changes input features to have mean 0, variance 1, and adds two more parameters (Ba et al. 2016)



$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

$$h_i = f\left(\frac{g_i}{\sigma_i}\left(a_i - \mu_i\right) + b_i\right)$$

Residue (Shortcut)



$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \qquad h_i = f\left(\frac{g_i}{\sigma_i}\left(a_i - \mu_i\right) + b_i\right)$$

Encoder Input

- Actual word representations are word pieces: byte pair encoding
  - Start with a vocabulary of characters
  - Most frequent ngram pairs $\mapsto$ a new ngram
  - Example: "es, est" 9 times, "lo" 7 times

Dictionary

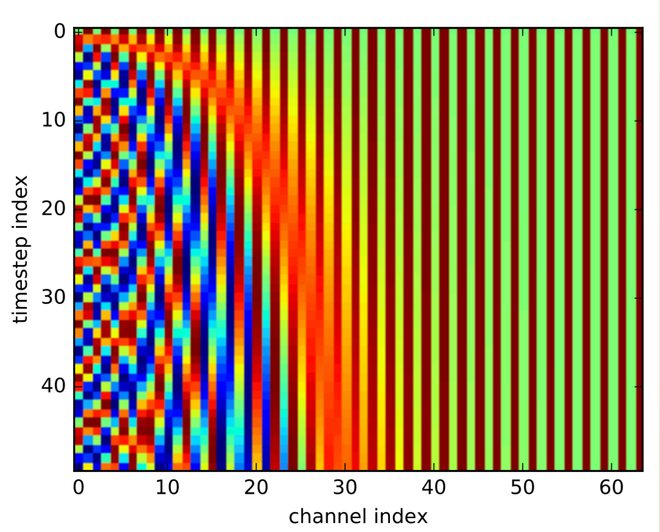
5 **lo** w
2 **lo** w e r
6 n e w est
3 w i d est

Vocabulary

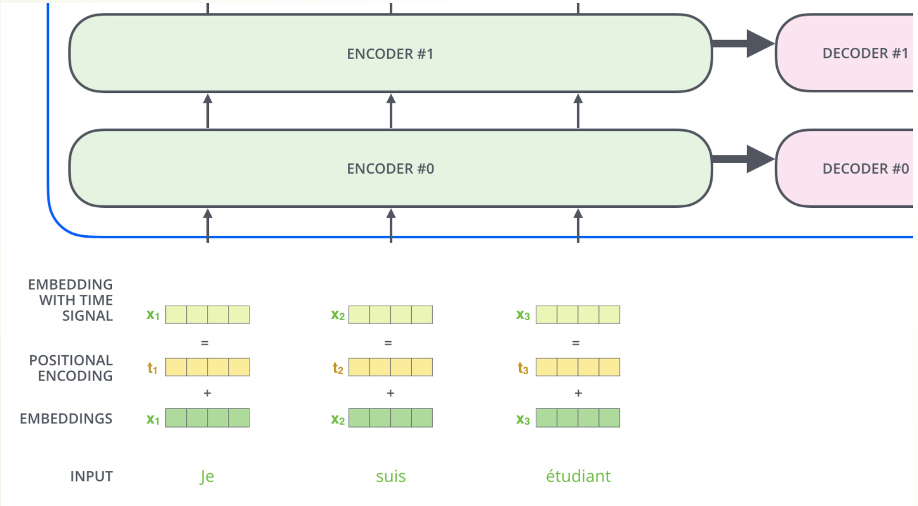
l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

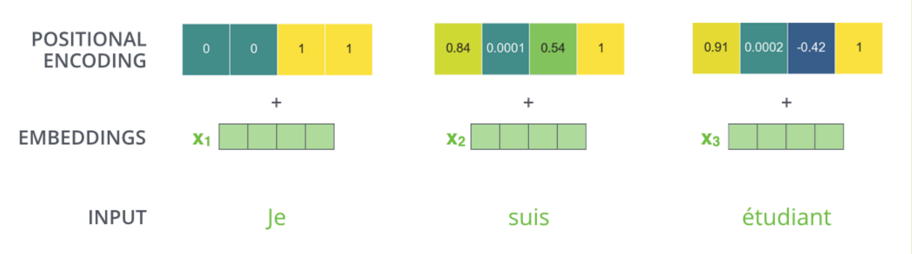
- Also added is a **positional encoding** so same words at different locations have different overall representations:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Or learned


ENCODER #1
DECODER #1
ENCODER #0
DECODER #0
EMBEDDING WITH TIME SIGNAL
POSITIONAL ENCODING
EMBEDDINGS
INPUT
Je
suis
étudiant


POSITIONAL ENCODING
0 0 1 1
0.84 0.0001 0.54 1
0.91 0.0002 -0.42 1
EMBEDDINGS
INPUT
Je
suis
étudiant

Sin/Cos Position Encoding

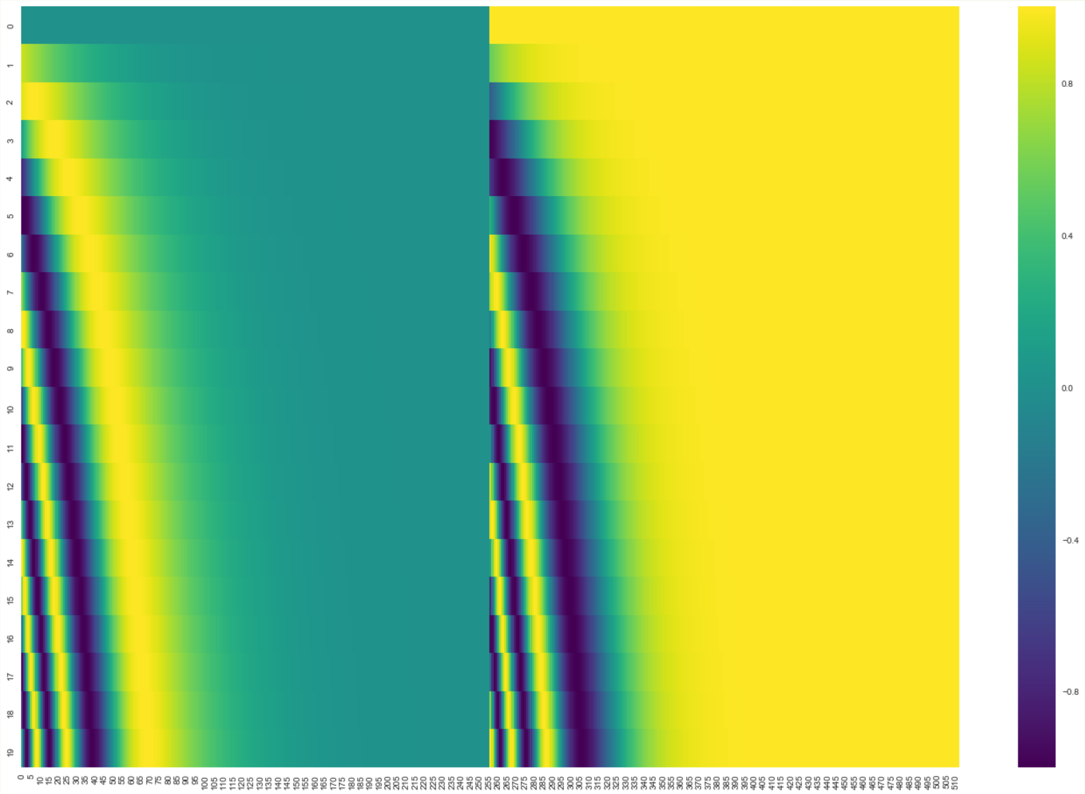
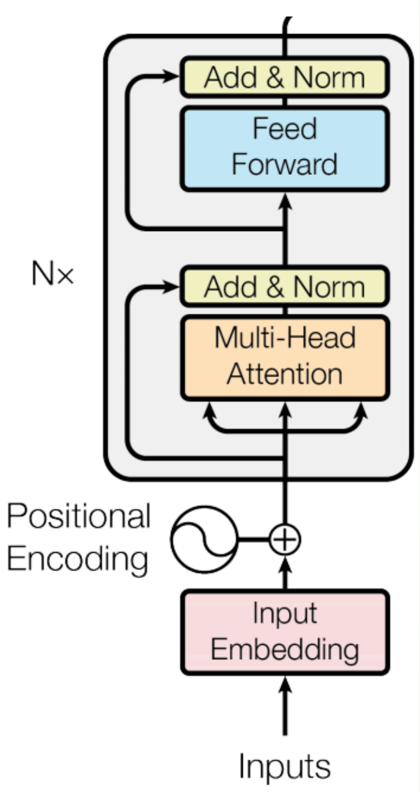
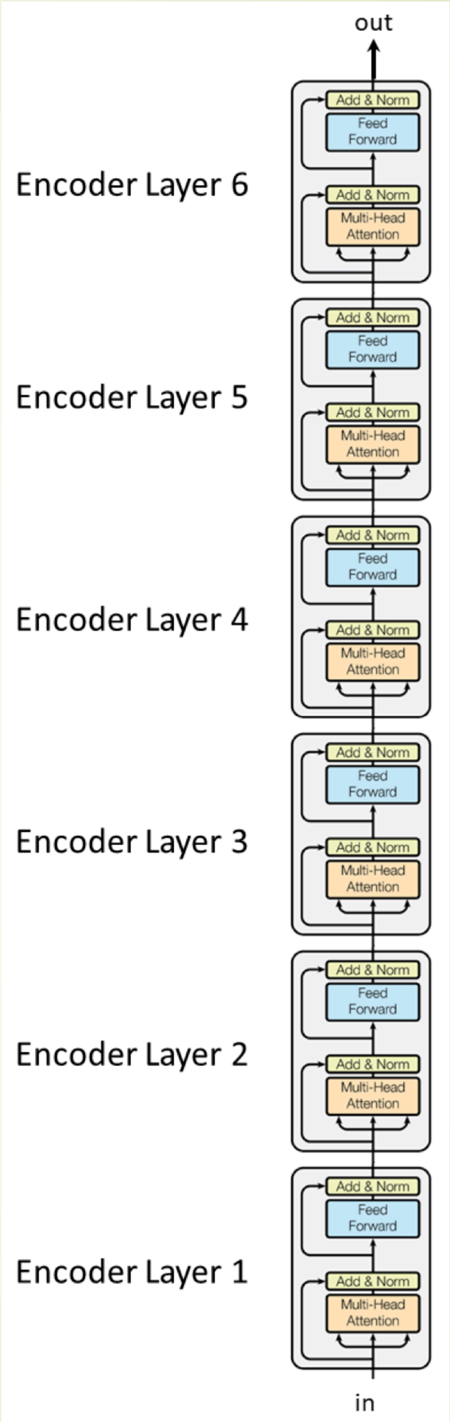


Figure. Each row corresponds the a positional encoding of a vector. So the first row would be the vector we'd add to the embedding of the first word in an input sequence. Each row contains 512 values – each with a value between 1 and -1. We've color-coded them so the pattern is visible.

Transformer Encoder

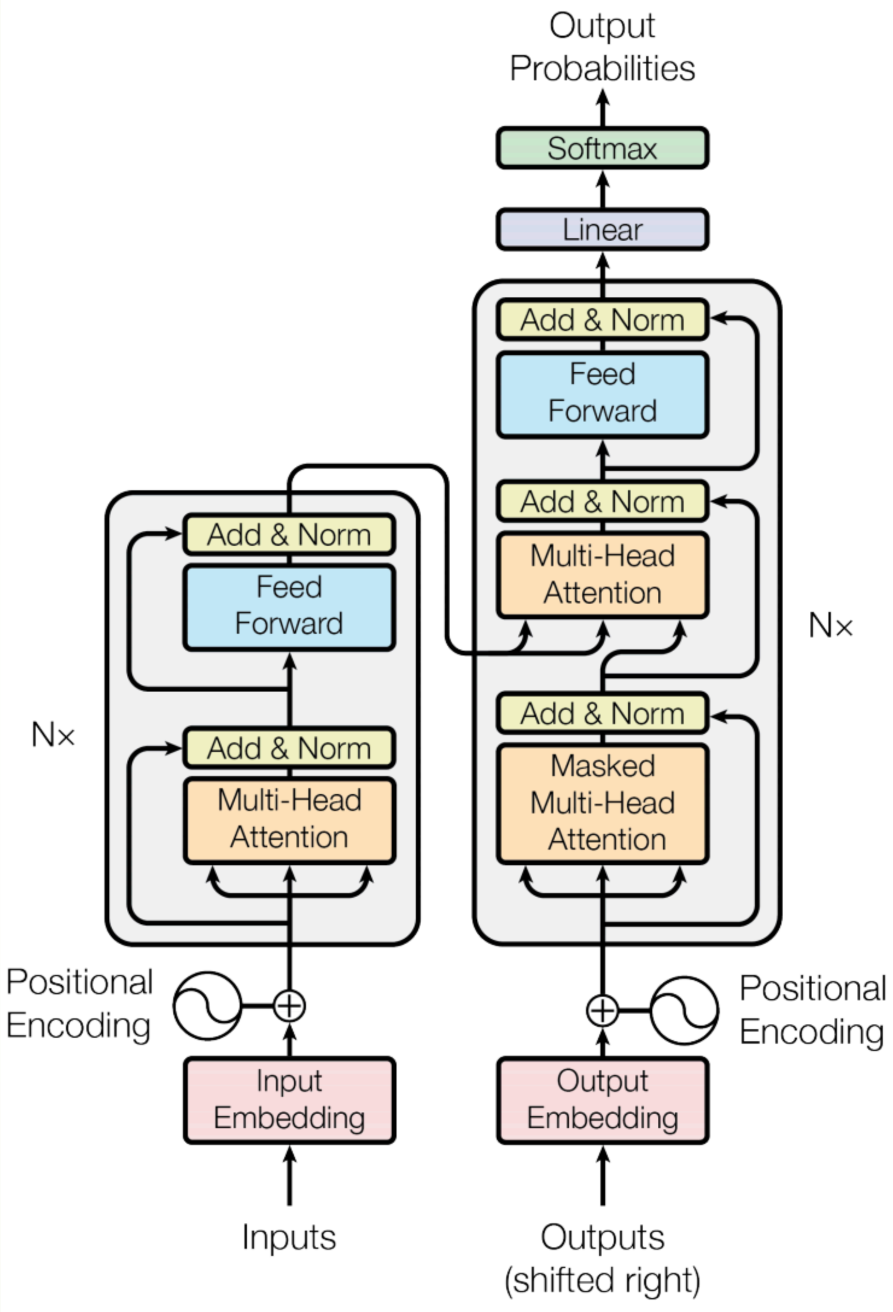
- Blocks are repeated N=6 or more times



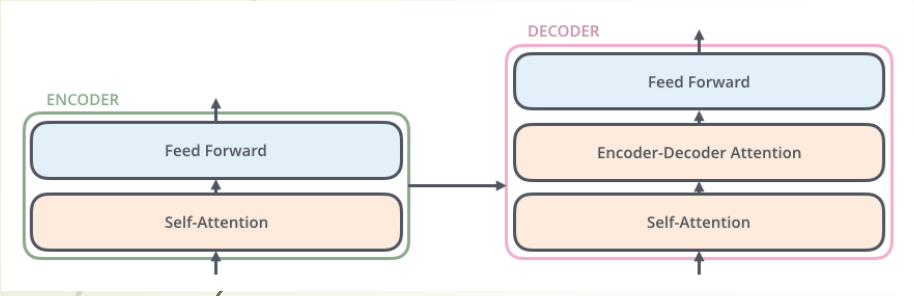


Transformer Decoder

- 2 sublayer changes in decoder
  - Masked decoder self-attention on previously generated outputs
  - Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder
- Blocks repeated N=6 times also



Encoder-Decoder



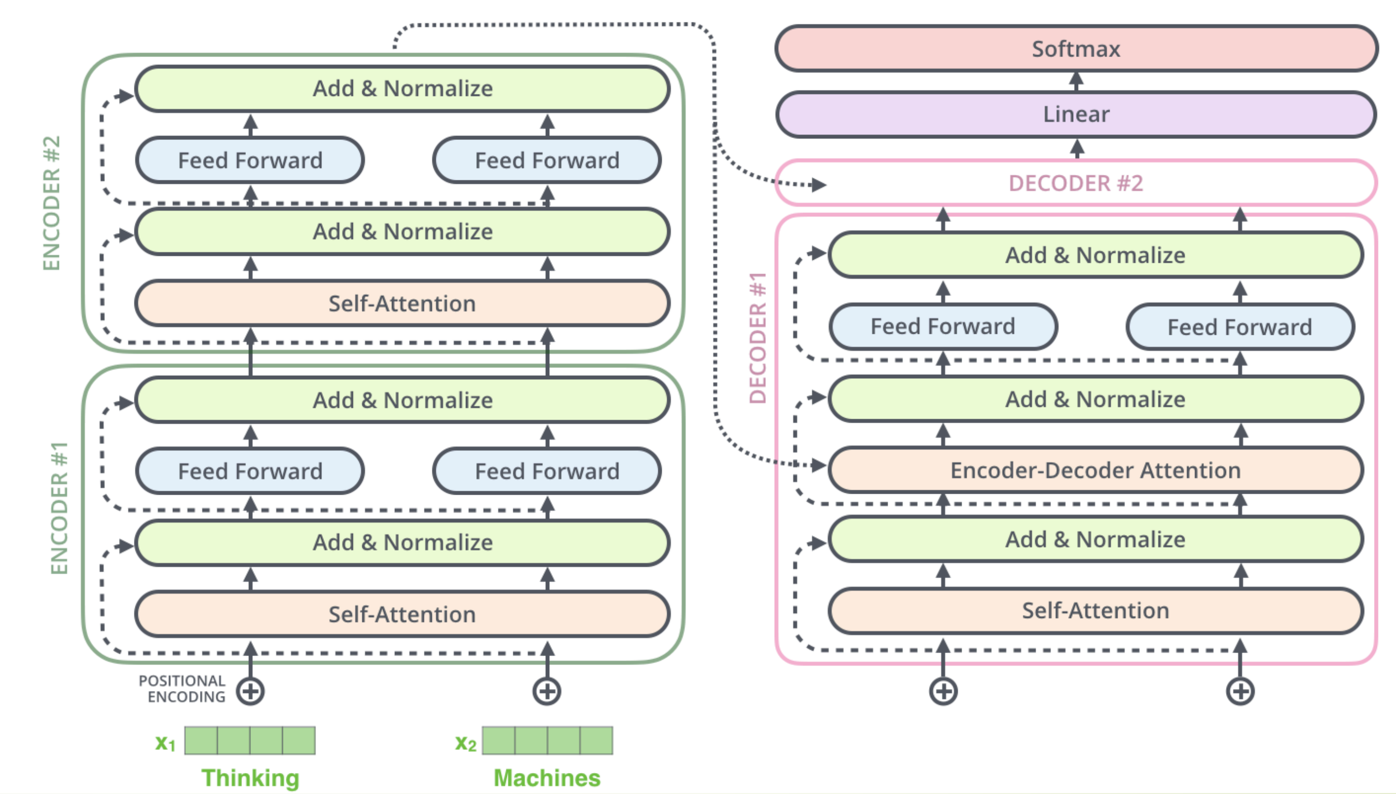


Illustration of Encoder-Decoder

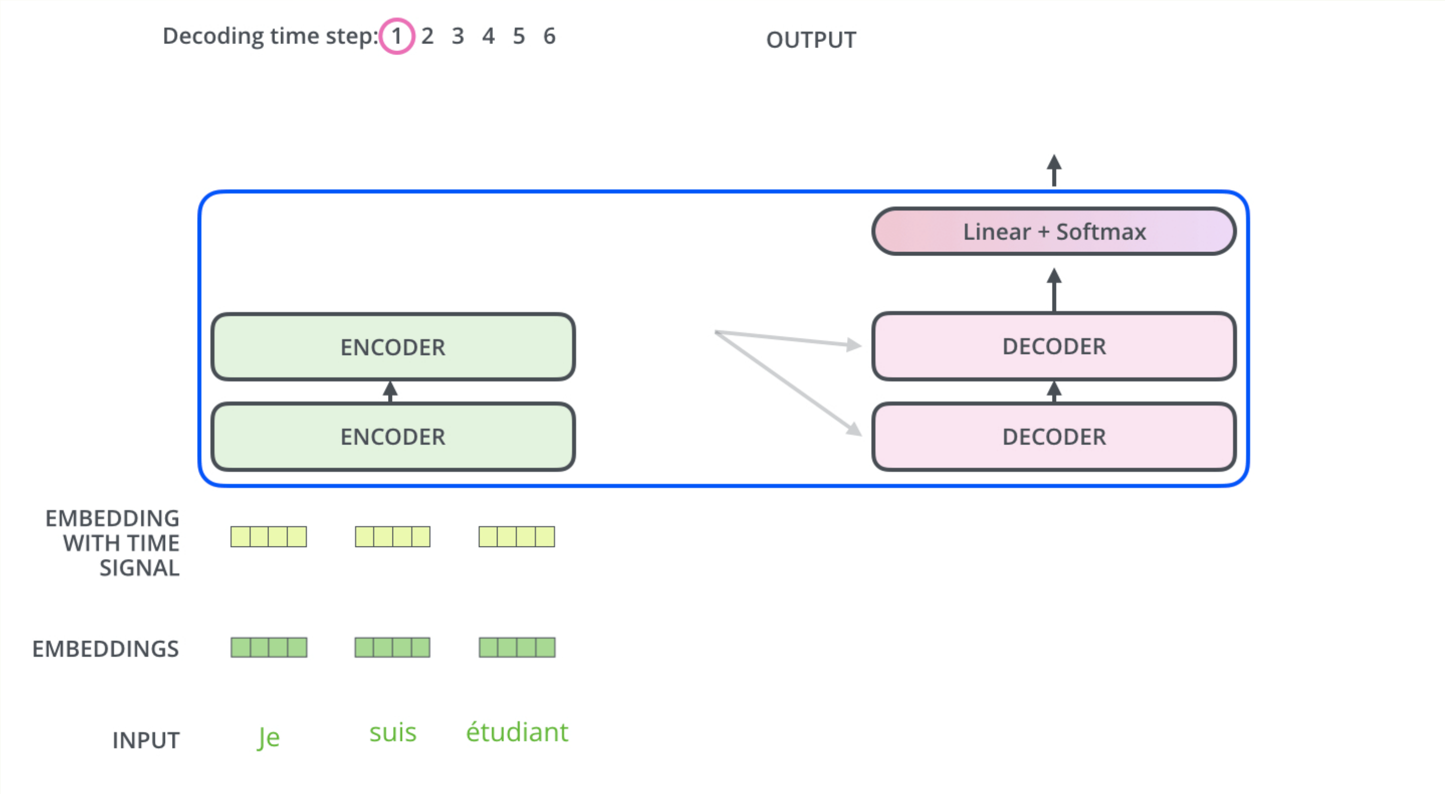
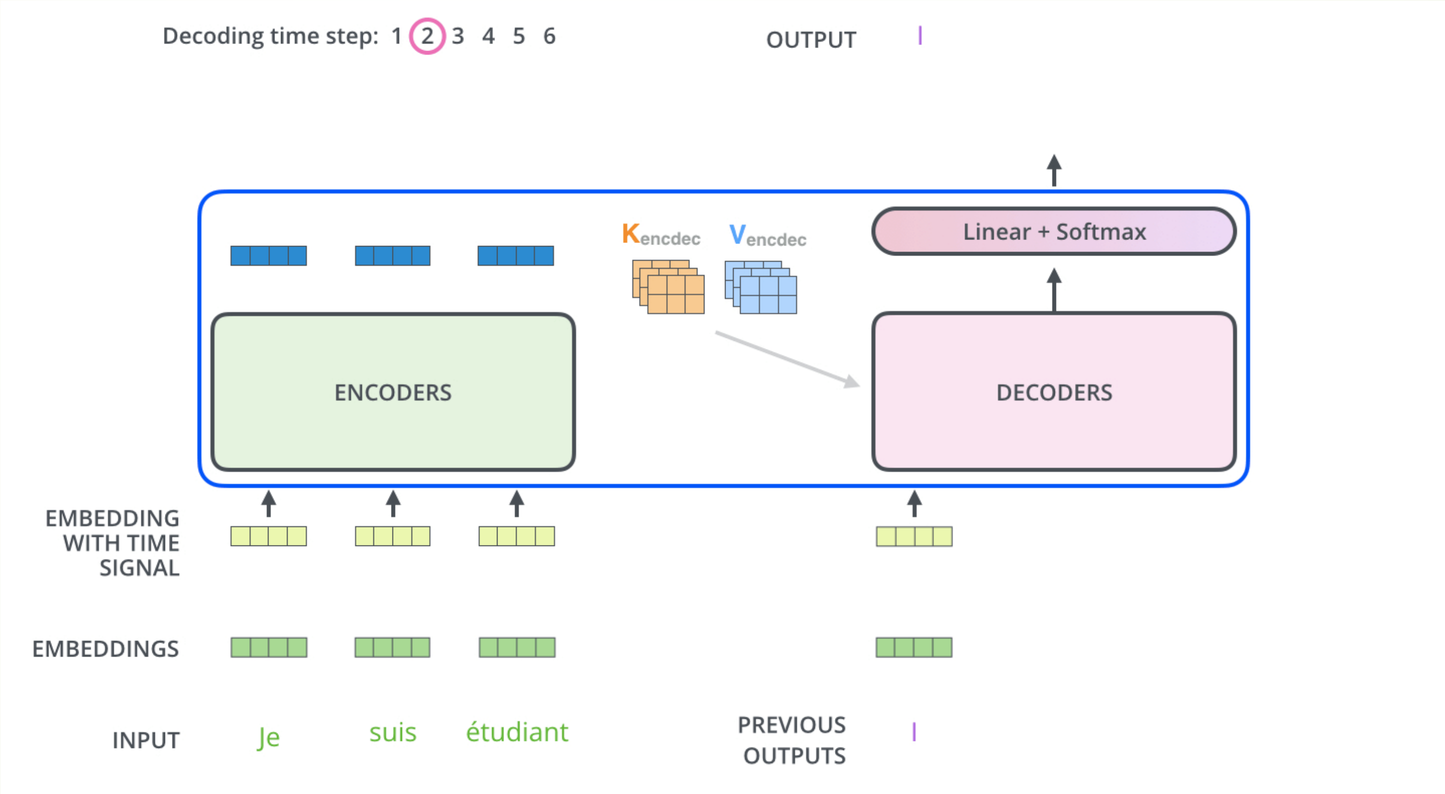


Illustration of Encoder-Decoder



Thank you!