Carnegie Mellon University

Introduction to Deep Learning for Engineers

Spring 2024, Deep Learning for Engineers
Feb 20, 2024, Eleventh Session

Amir Barati Farimani
Associate Professor of Mechanical Engineering and Bio-Engineering
Carnegie Mellon University

Story so far…

Challenge: Learning Sequences both on the inputs and outputs

1. What kind of sequences do we need to learn? what type of relations?

We need to learn complex relationships in sequence data, some with different functional relationships, long and short dependencies

"The concert was boring for the first 15 minutes while the band warmed up but then was terribly exciting."

2. Why simple, feed forward neural network can not model the sequences?

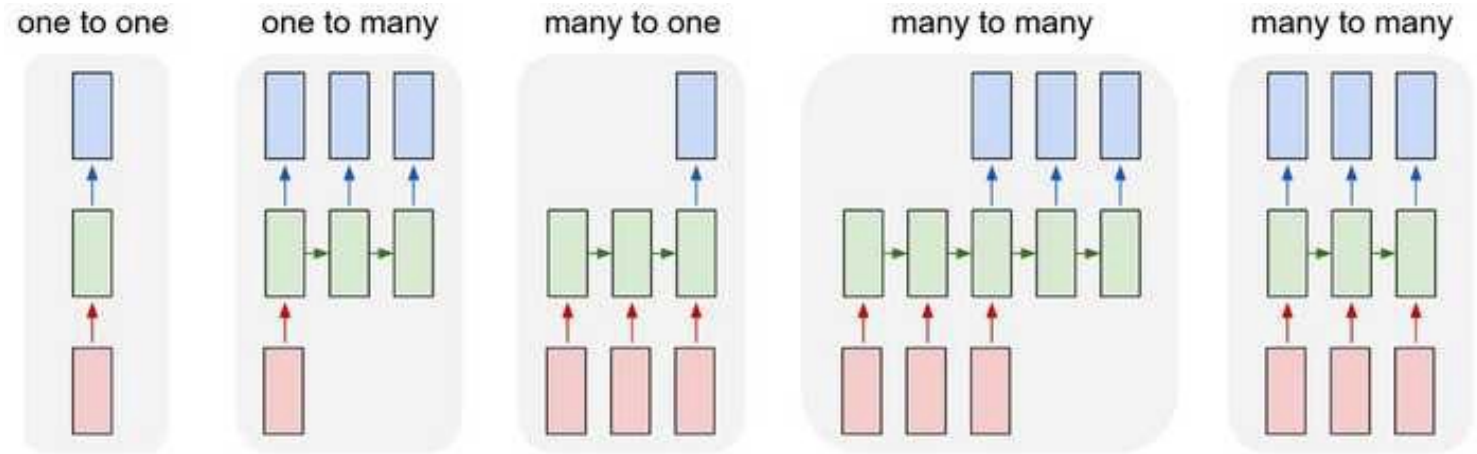
Simple NN can only model relationships in sequences between the input features and output, not at the same time learn the dependencies between input features

I am a student→ Je Suis en etudiante

Story so far…

Challenge: Learning Sequences both on the inputs and outputs

3. What are the types of tasks we are looking at in sequence learning?



- **LM task:** *When she tried to print her _____, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her tickets.*

RNN Sequence Types

RNNs work by iteratively updating a hidden state h, which is a vector that can also have arbitrary dimension. At any given step t,

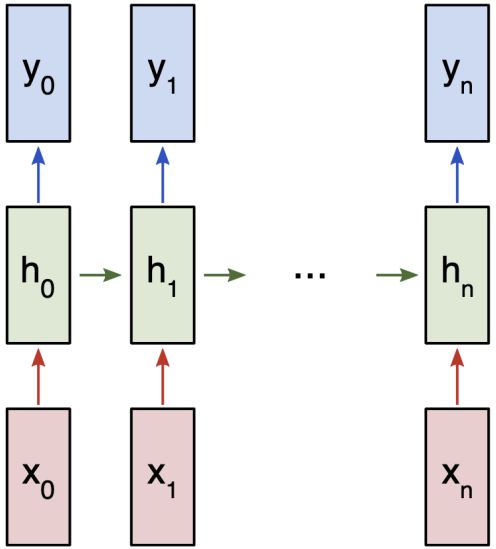
1.The next hidden state h_t is calculated using the previous hidden state h_{t-1} and the next input x_t.

2.The next output y_t is calculated using h_t.

- *Wxh*, used for all x_t → h_t links.
- *Whh*, used for all h_{t-1} → h_t links.
- *Why*, used for all h_t → y_t links.

We'll also use two biases for our RNN:

- b_h, added when calculating h_t.
- b_y, added when calculating y_t.

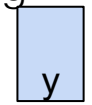


Story so far…

Challenge: Learning Sequences both on the inputs and outputs

4. How can we design a general learning framework that can learn the sequential dependencies between input features, output labels and the relationship between inputs and outputs?

By creating a hidden state connecting the input and outputs and by learning the weights connecting the hidden states to each others, we will learn the sequential dependencies between input sequence and output sequence



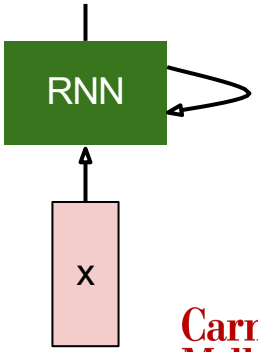
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



Story so far…

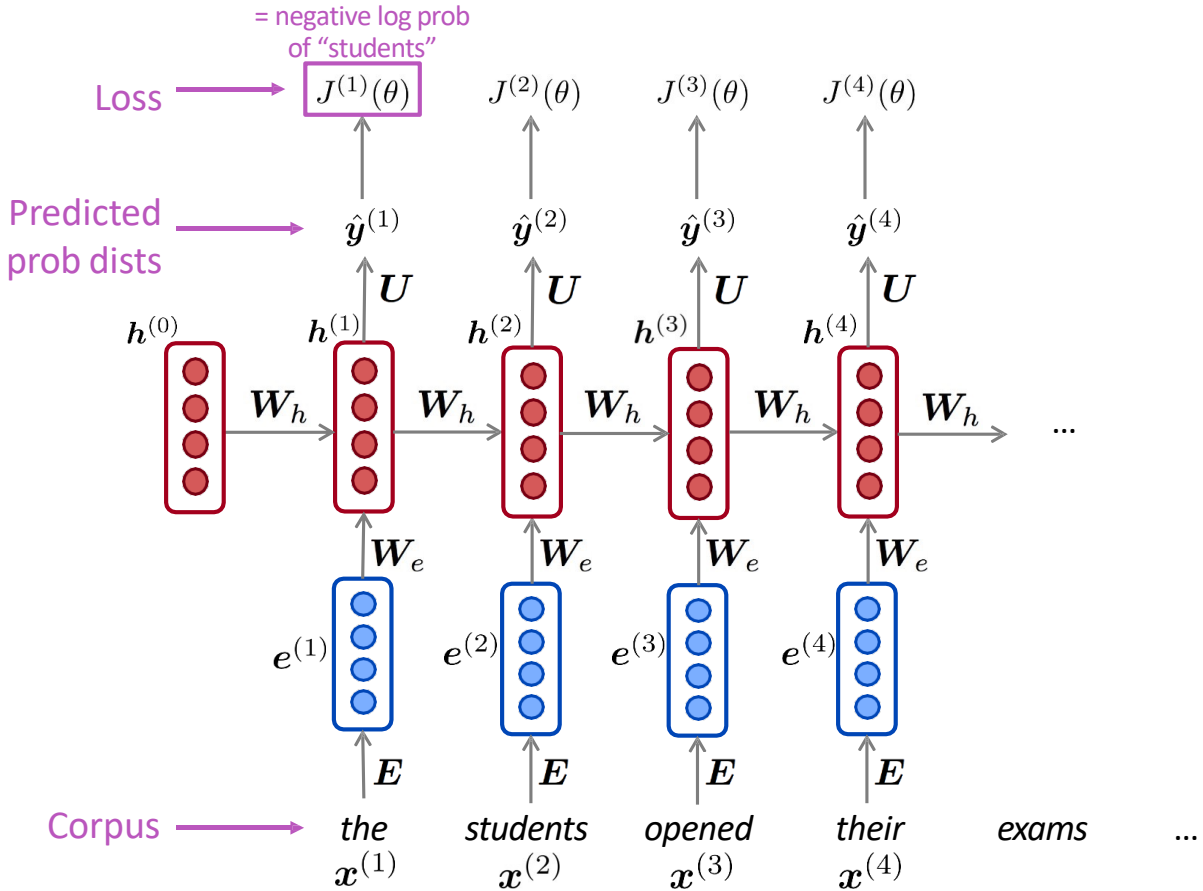
Challenge: Learning Sequences both on the inputs and outputs

5. How we solve the challenge of the inputs and outputs length to be dramatically different?

We will open up sufficient depth in the recurrent unit (by the largest sequence) and the RNNs are flexible during backpropagation.

5. How is backpropagation done in RNNs?

Training a RNN Language Model

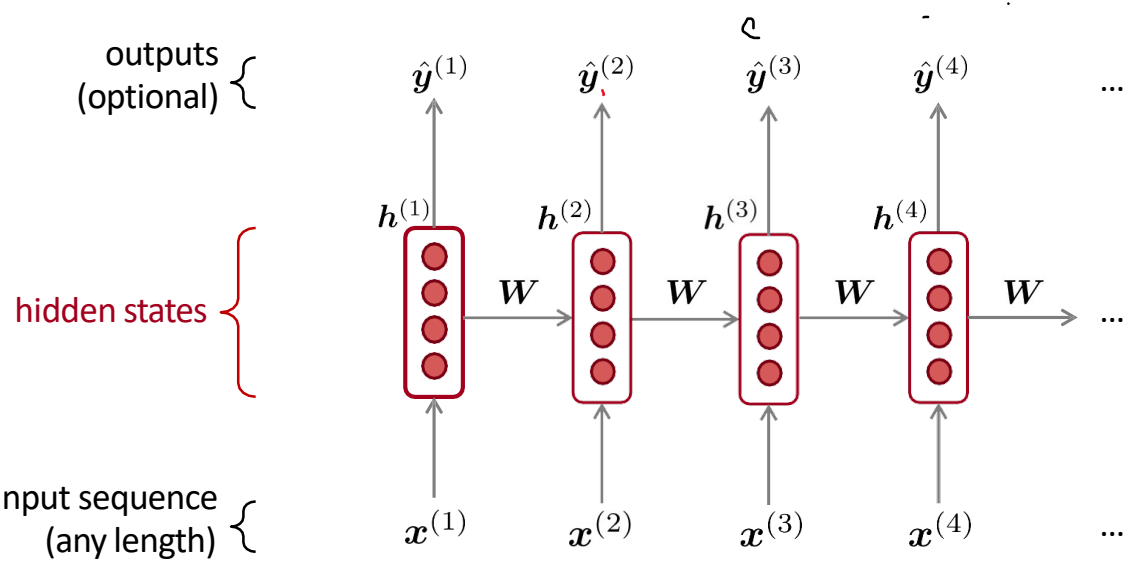


Carnegie Mellon University

Recurrent Neural Networks (RNN)

A family of neural architectures

Core idea: Apply the same weights $\boldsymbol{W}$ *repeatedly*



Carnegie Mellon University

A RNN Language Model

$\hat{\boldsymbol{y}}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

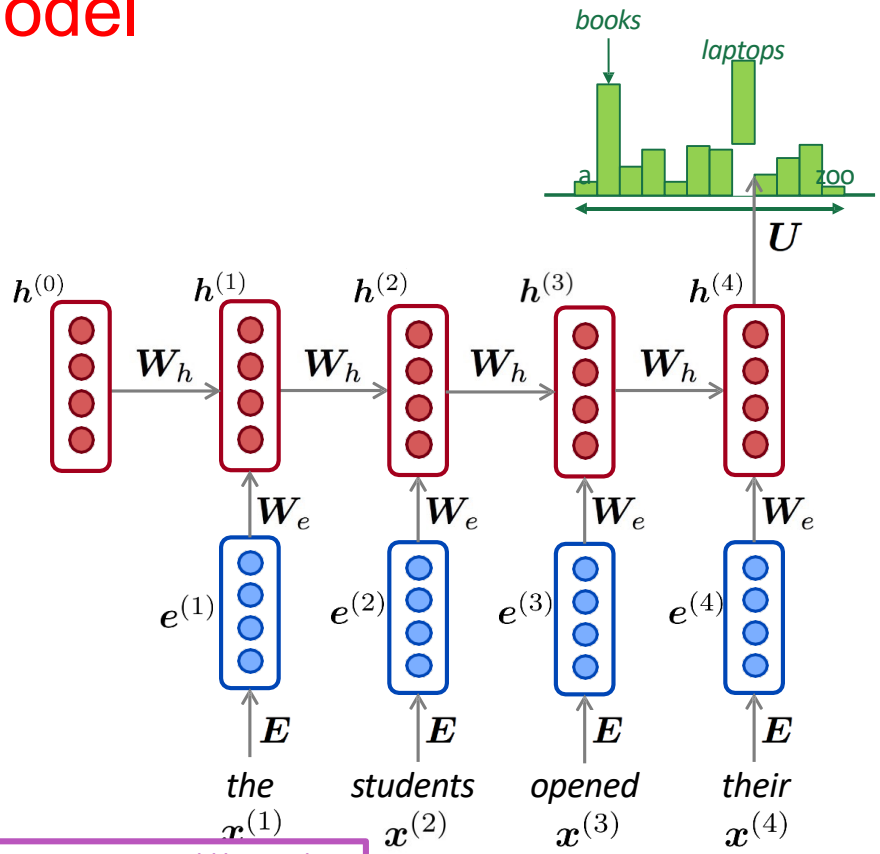
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Note: *this input sequence could be much longer, but this slide doesn't have space!*

Carnegie Mellon University

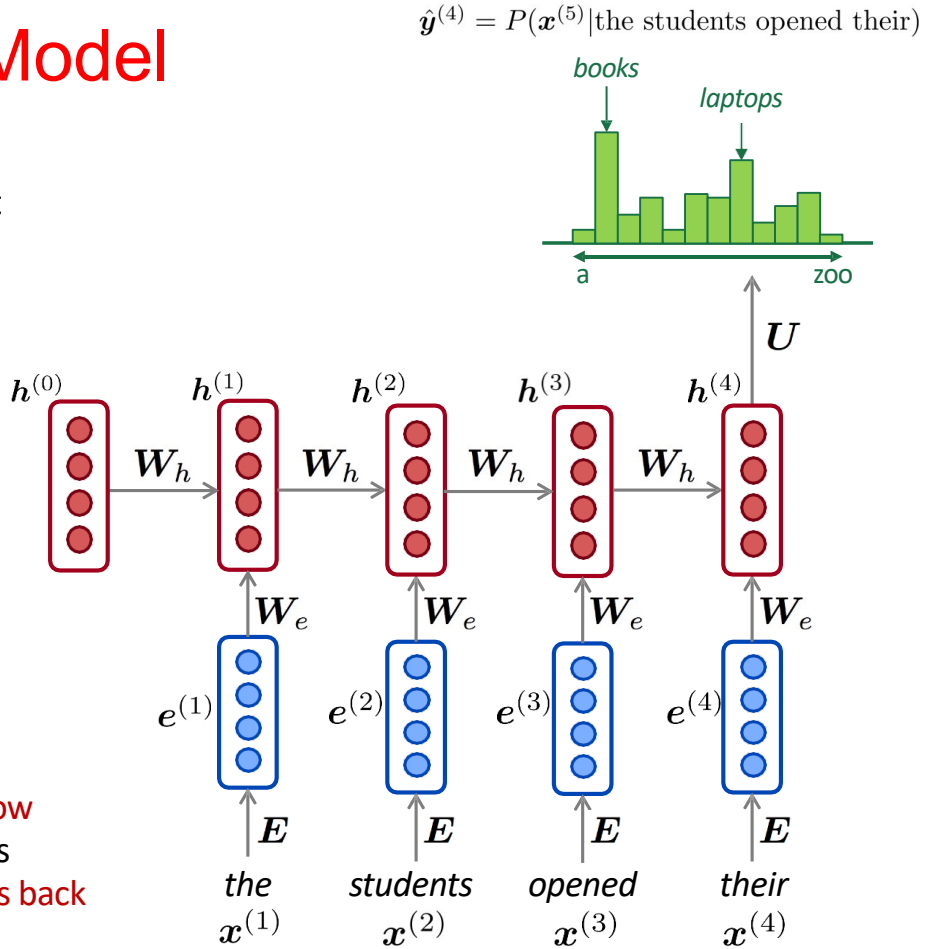
A RNN Language Model

RNN **Advantages**:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed

RNN **Disadvantages**:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

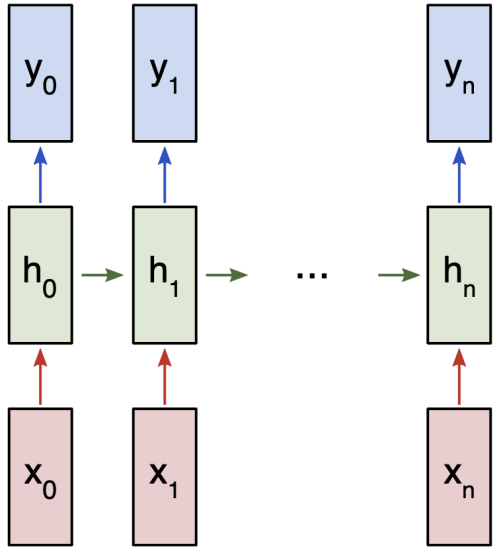


Carnegie Mellon University

Weights

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

$$y_t = W_{hy}h_t + b_y$$



Training a RNN Language Model

- Get a big corpus of text which is a sequence of words $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$
- Feed into RNN-LM; compute output distribution $\hat{\boldsymbol{y}}^{(t)}$ for *every step t*.
  - i.e. predict probability distribution of *every word*, given words so far

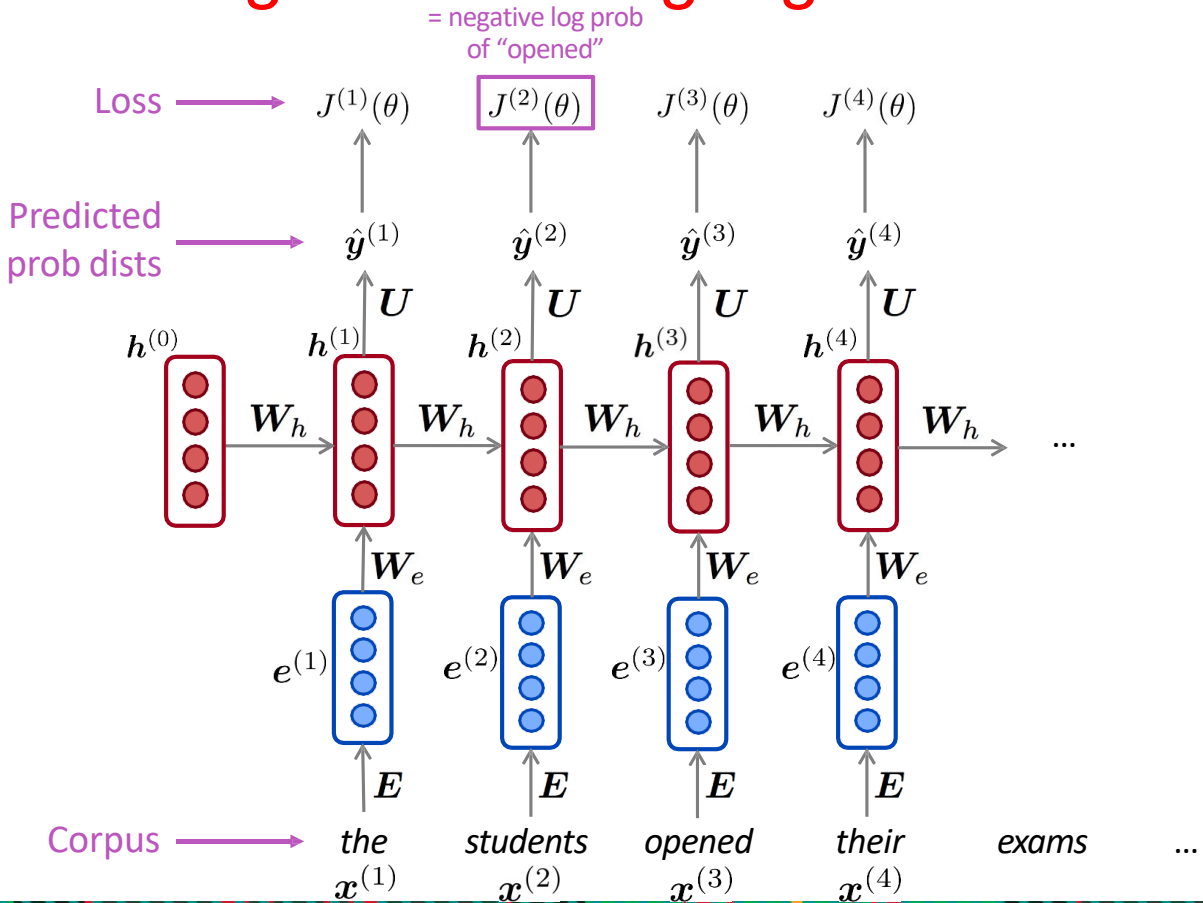
- Loss function on step *t* is cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$, and the true next word $\boldsymbol{y}^{(t)}$ (one-hot for $\boldsymbol{x}^{(t+1)}$):

$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

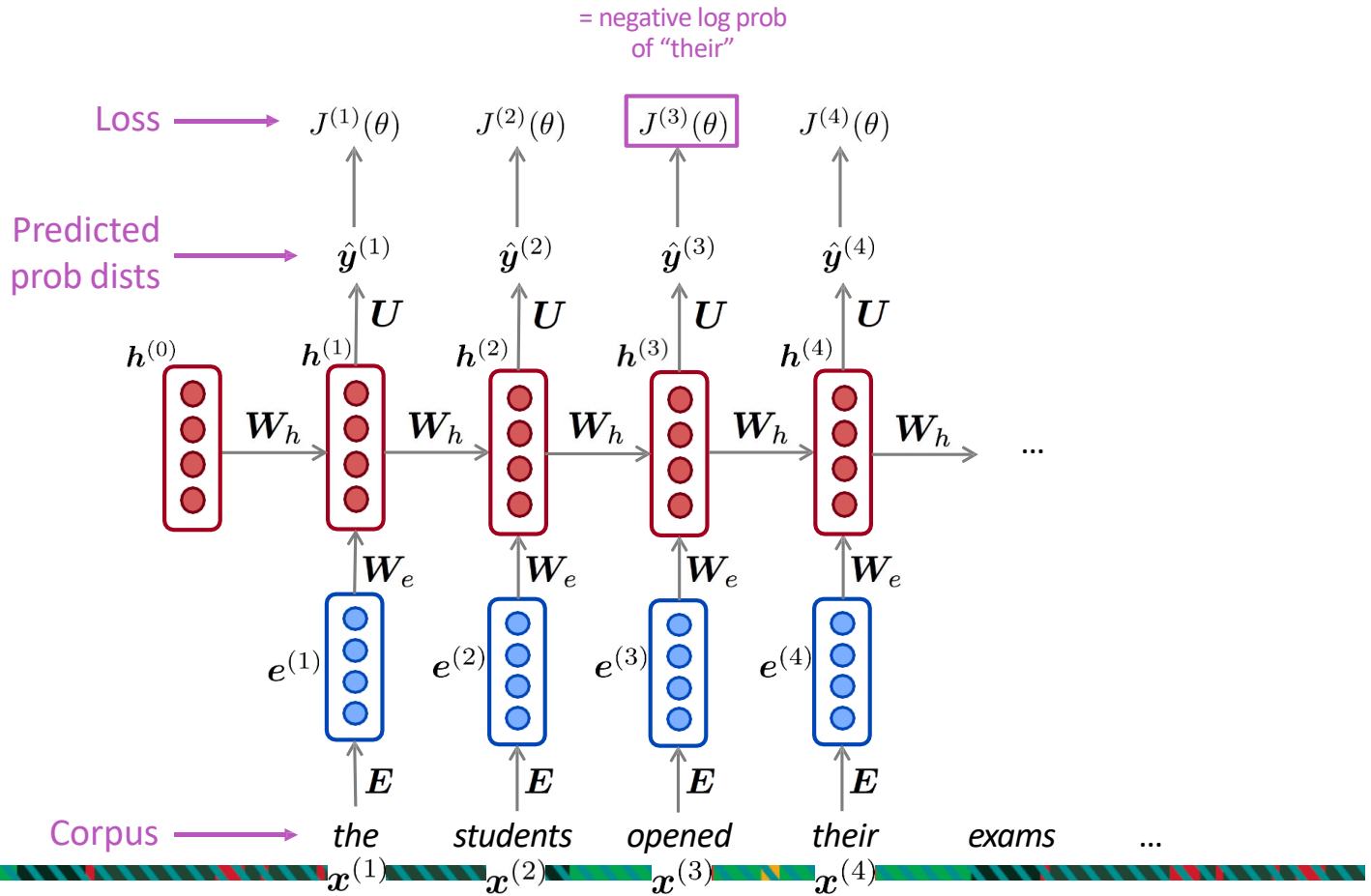
$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

Training a RNN Language Model



Carnegie Mellon University

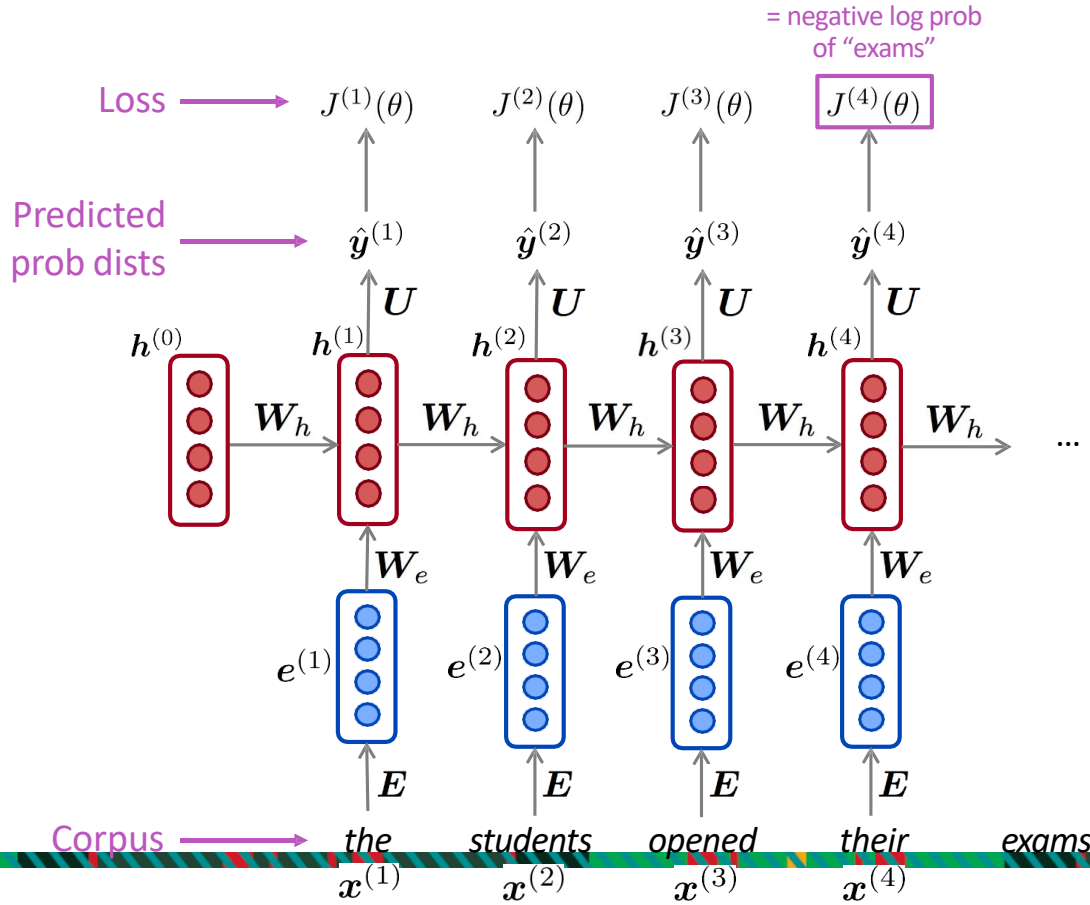
Training a RNN Language Model



Abigail See

Carnegie Mellon University

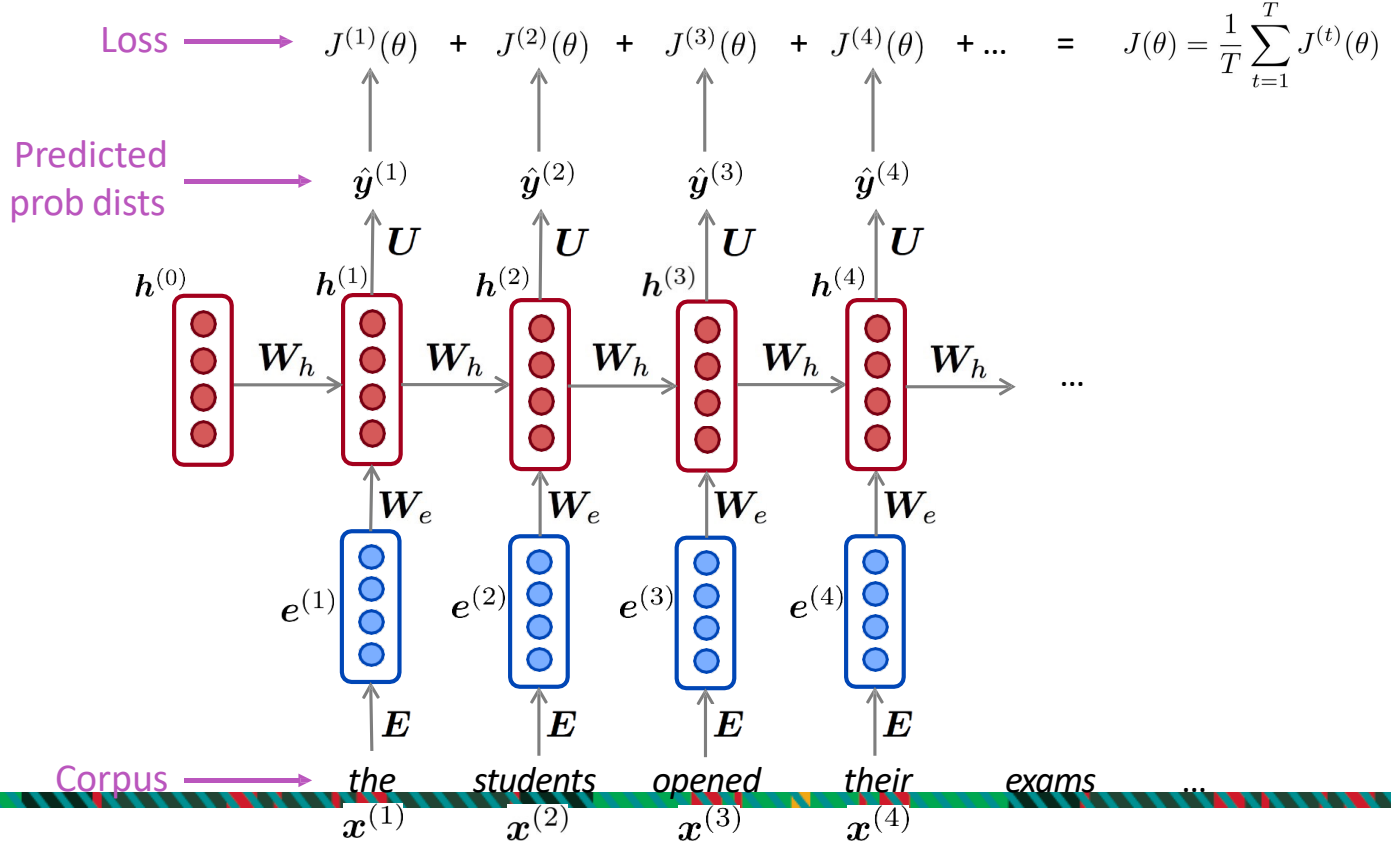
Training a RNN Language Model



Abigail See

Carnegie Mellon University

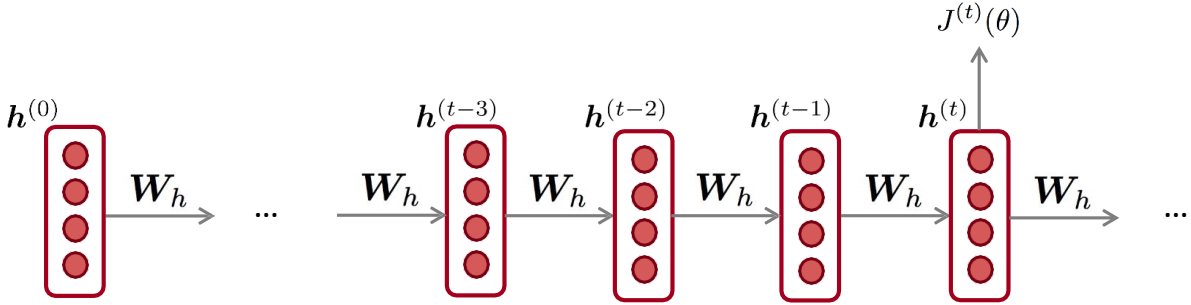
Training a RNN Language Model



Abigail See

Carnegie Mellon University

Backpropagation for RNNs

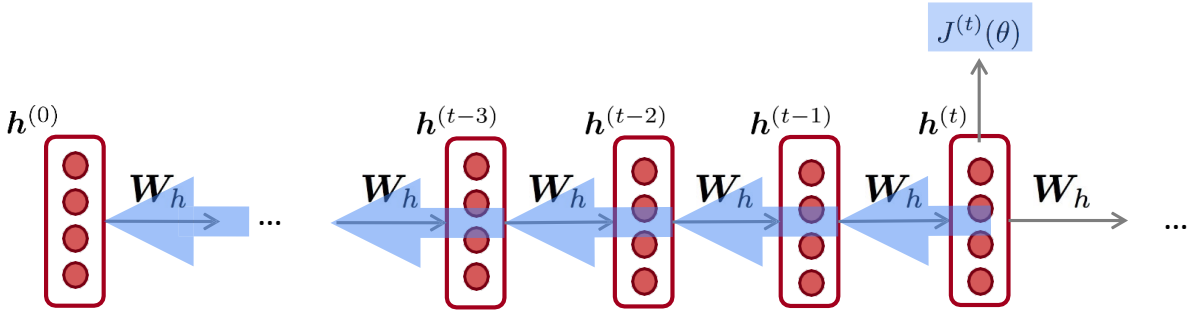


Question: What's the derivative of $J^{(t)}(\theta)$ w.r.t. the repeated weight matrix $\boldsymbol{W_h}$?

Answer: $$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\bigg|_{(i)}$$

"The gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears"

Backpropagation for RNNs



$$\frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}} = \sum_{i=1}^{t} \frac{\partial J^{(t)}}{\partial \boldsymbol{W_h}}\Big|_{(i)}$$

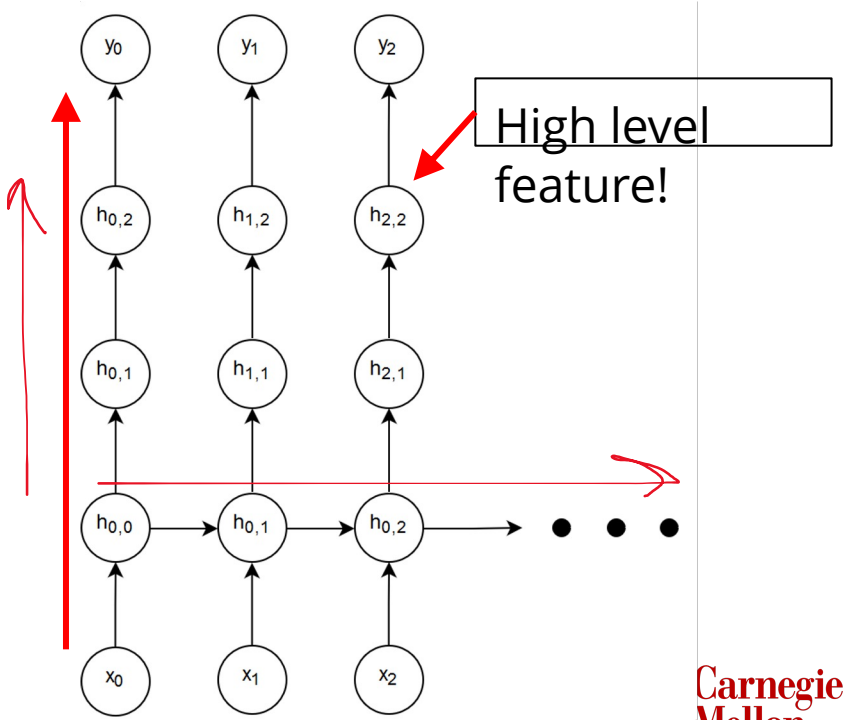
Question: How do we calculate this?

Answer: Backpropagate over timesteps *i*=*t*,…,0, summing gradients as you go.
This algorithm is called **"backpropagation through time"**

Option 1: Feedforward Depth (d_f)

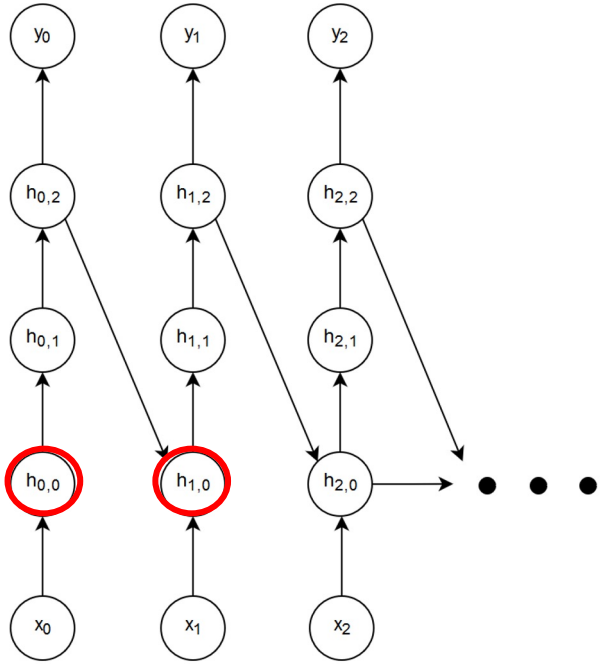
Notation: $h_{0,1}$ ⇒ time step 0, neuron #1

FEEDFORWARD DEPTH: **LONGEST** PATH BETWEEN AN INPUT AND OUTPUT AT THE **SAME TIMESTEP**



Feedforward depth = 4

Option 2: Recurrent Depth (d_r)

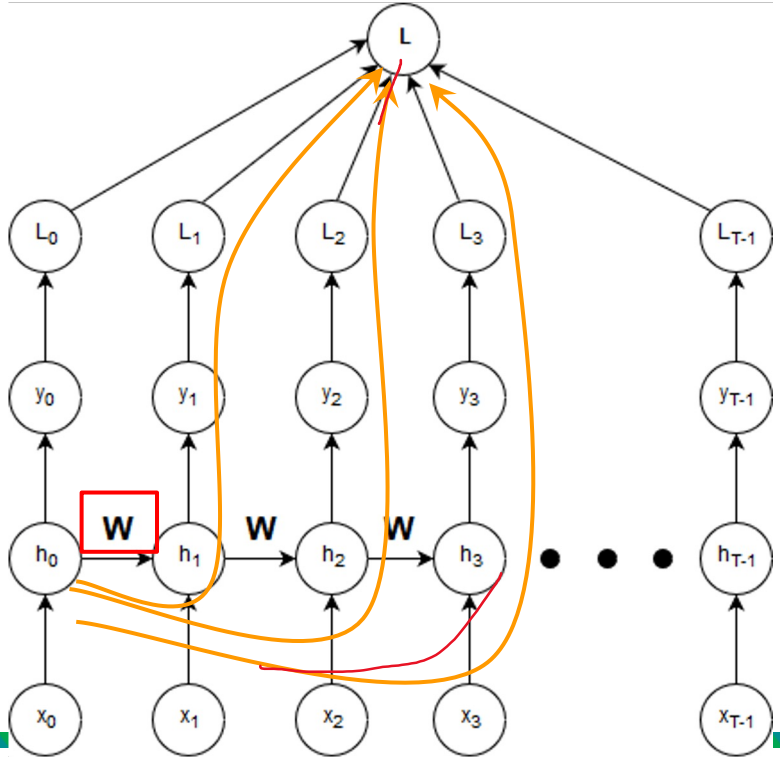


- **Recurrent depth**: Longest path between **same hidden state** in **successive timesteps**

Recurrent depth = 3

Carnegie Mellon University

Backpropagation Through Time (BPTT)



OBJECTIVE IS TO UPDATE THE WEIGHT MATRIX:

$$\mathbf{W} \rightarrow \mathbf{W} - \alpha \frac{\partial L}{\partial \mathbf{W}}$$

ISSUE: **W** OCCURS EACH TIMESTEP

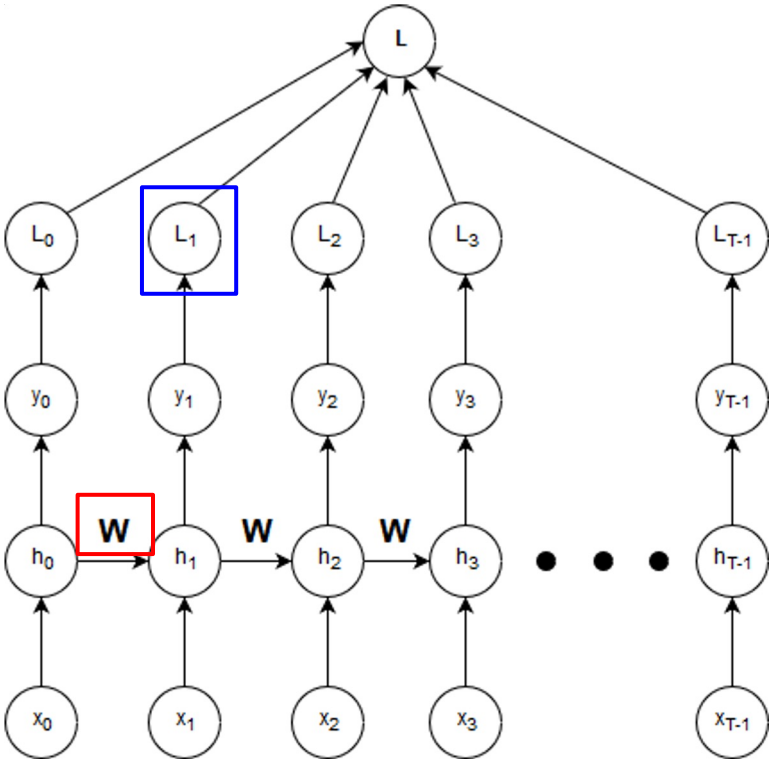
EVERY PATH FROM **W** TO L IS ONE DEPENDENCY

FIND ALL PATHS FROM **W** TO L!

(note: dropping subscript h from $\mathbf{W_h}$ for brevity)

Carnegie Mellon University

Systematically Finding All Paths

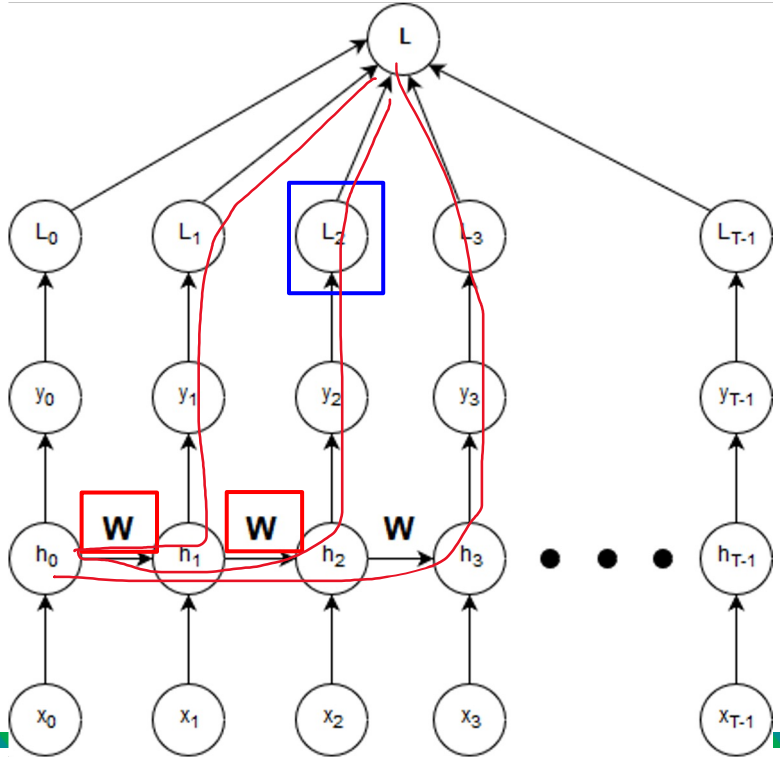


How many paths exist from W to L *through* L_1?

Just 1. Originating at h_0.

Carnegie Mellon University

Systematically Finding All Paths

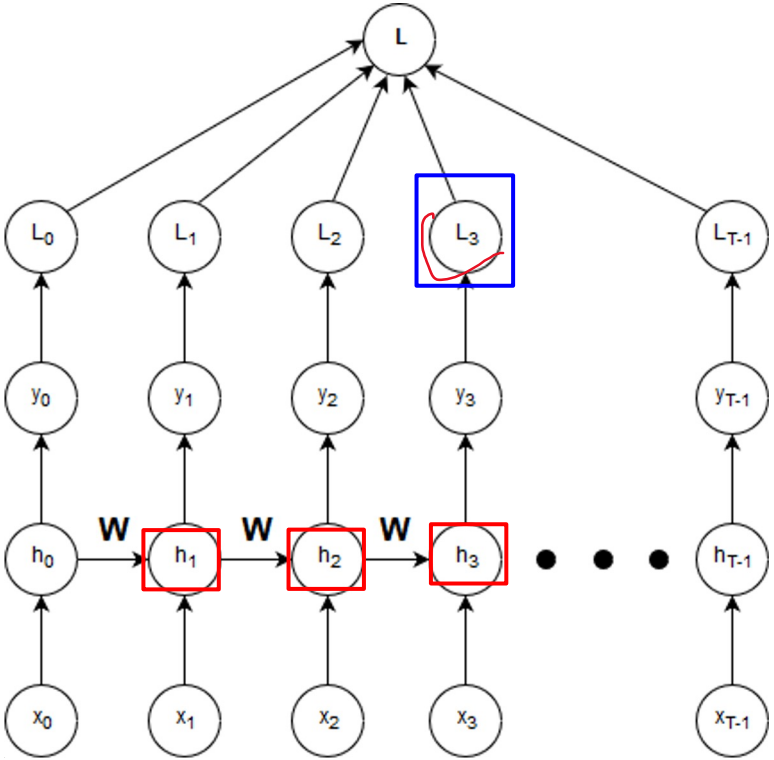


How many paths from **W** to L *through* L_2?

2. Originating at h_0 and h_1.

Carnegie Mellon University

Systematically Finding All Paths



And 3 in this case.

Origin of path = basis for Σ

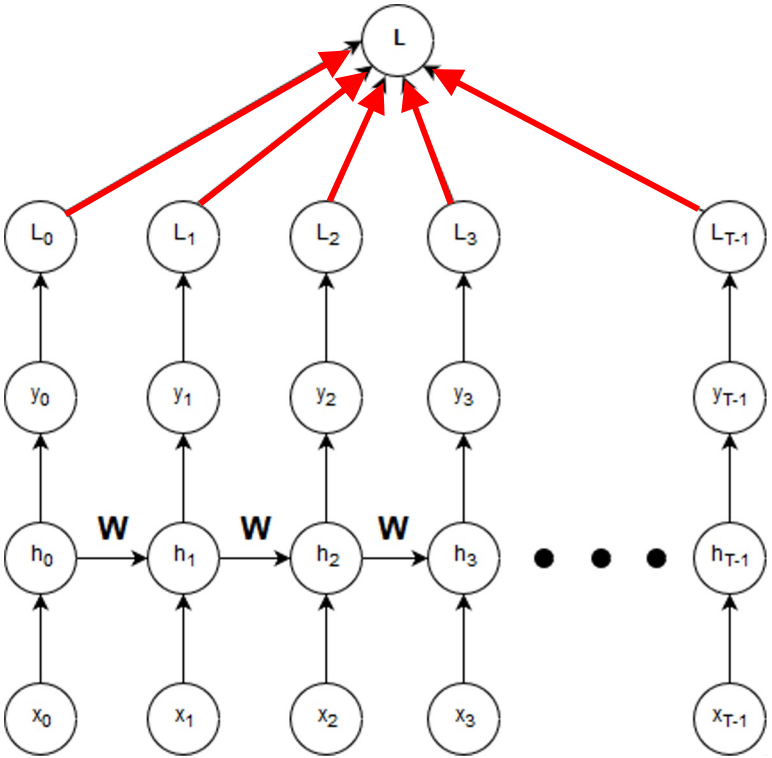
$$\frac{\partial L}{\partial \mathbf{W}}$$

The gradient has two summations:
1: Over L_j
2: Over h_k

To skip proof, click here.

Carnegie Mellon University

Backpropagation as two summations

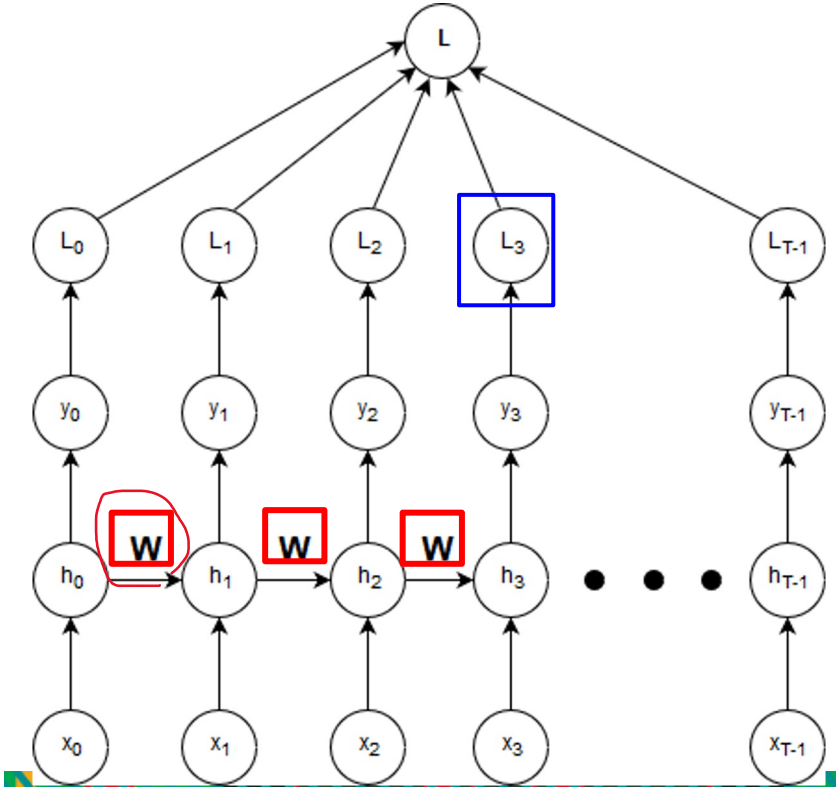


First summation over L

$$\frac{\partial L}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial L_j}{\partial \mathbf{W}}$$

Carnegie Mellon University

Backpropagation as two summations



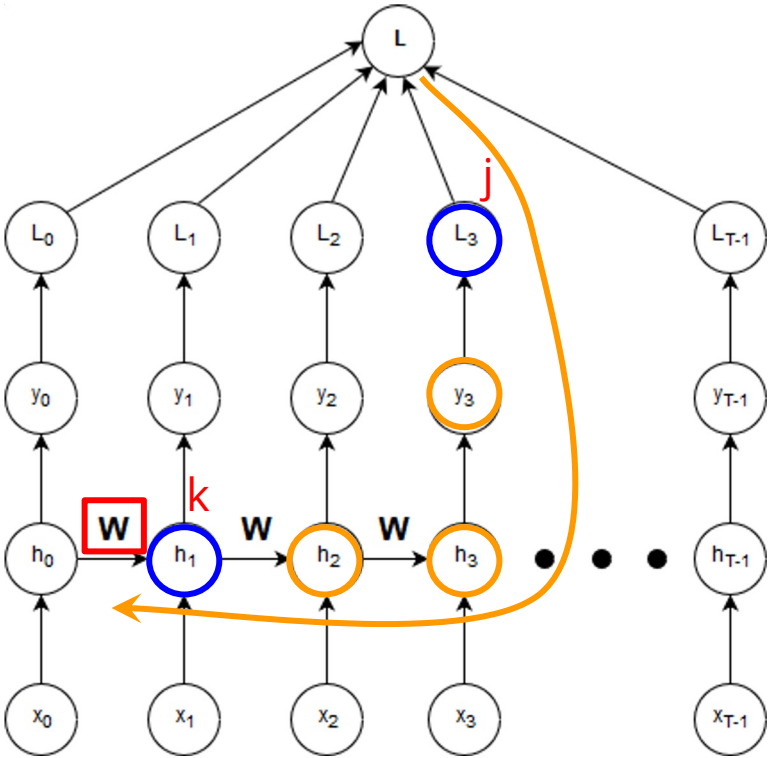
- **Second summation over h:**
  Each $\mathbf{L_j}$ depends on the weight matrices *before it*

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

L_j depends on all h_k before it.

Carnegie Mellon University

Backpropagation as two summations

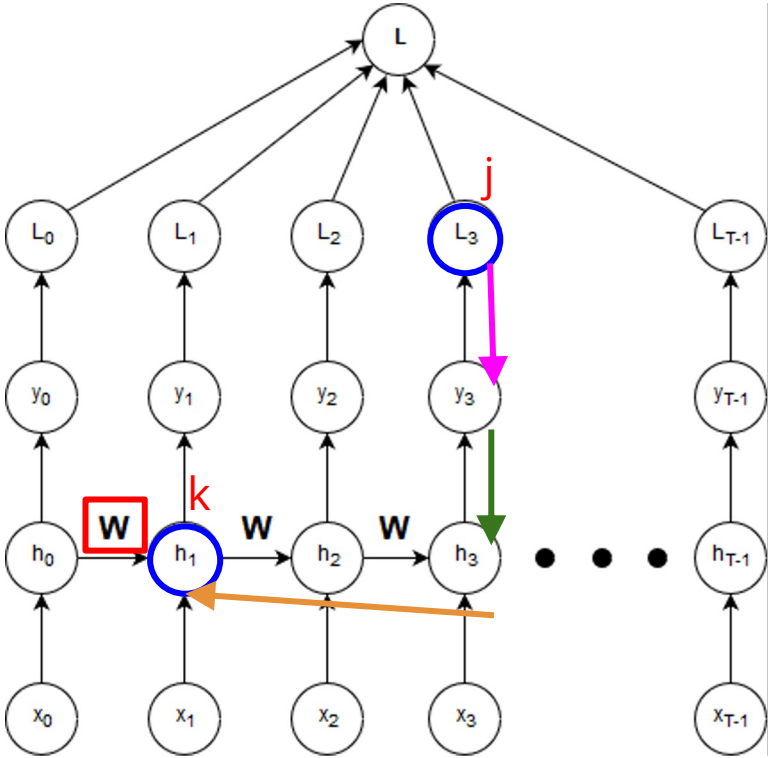


$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

- No explicit of L_j on h_k
- Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

Backpropagation as two summations

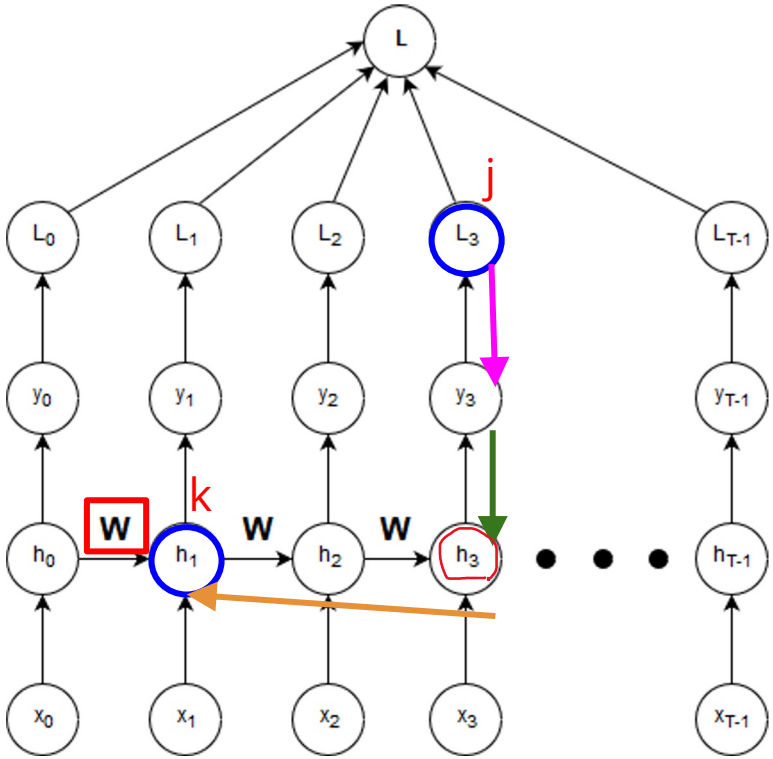


$$\frac{\partial L_j}{\partial W} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

- No explicit of L_j on h_k
- Use chain rule to fill missing steps

$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

The Jacobian



$$\frac{\partial L_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$

Indirect dependency. One final use of the chain rule gives:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$

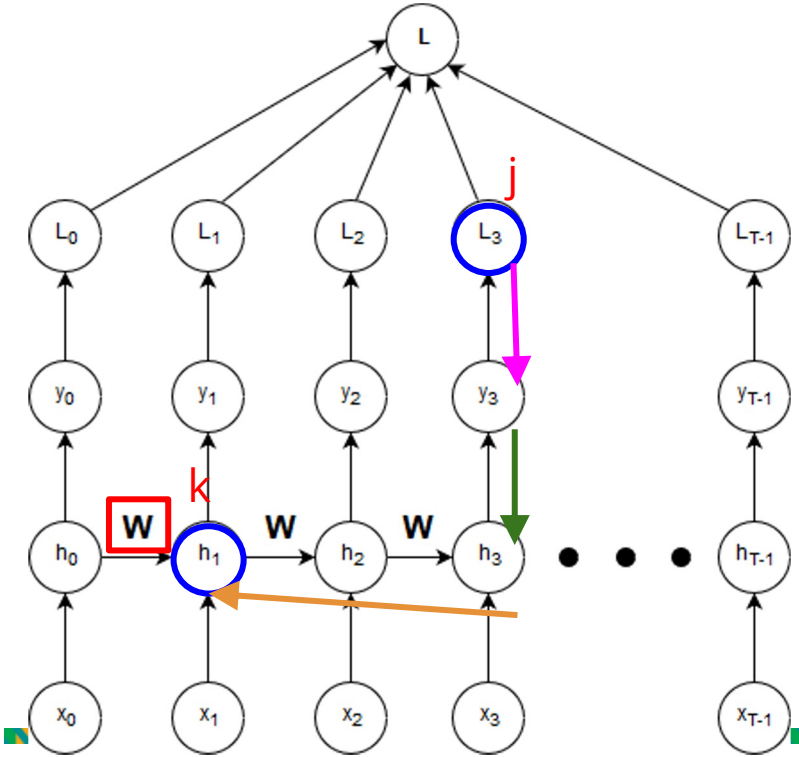
"The Jacobian"

Carnegie Mellon University

The Final Backpropagation Equation

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

Backpropagation as two summations



$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$

- Often, to reduce memory requirement, we truncate the network
- Inner summation runs from *j-p* to *j* for some *p* ==> truncated BPTT

Carnegie Mellon University

Expanding the Jacobian

$$\frac{\partial L}{\partial W} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

The Issue with the Jacobian

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Weight Matrix

Derivative of activation function

REPEATED MATRIX MULTIPLICATIONS LEADS TO **VANISHING AND EXPLODING GRADIENTS**.
HOW? LET'S TAKE A SLIGHT DETOUR.

Eigenvalues and Stability

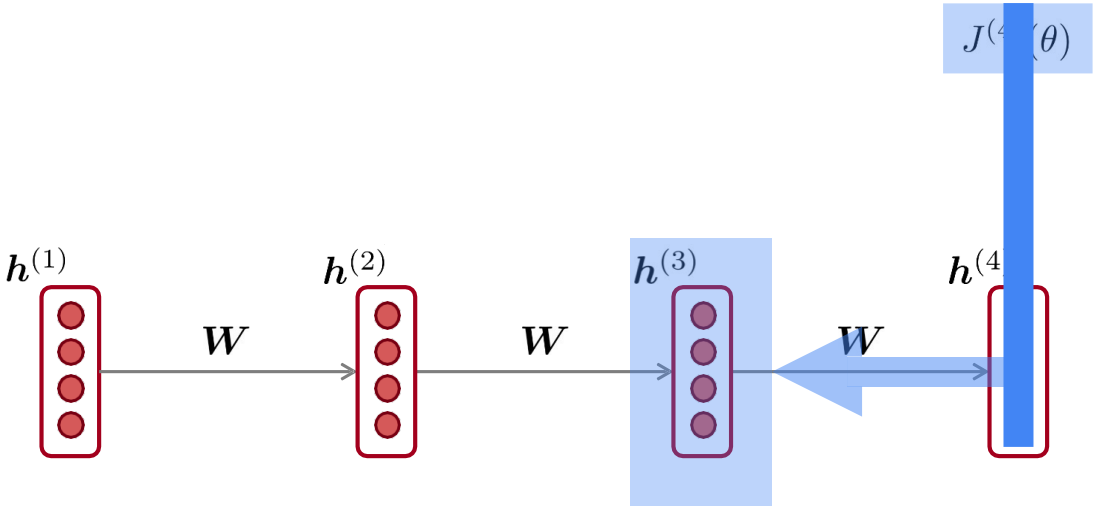
$$\Lambda = \begin{bmatrix} -0.6180 & 0 \\ 0 & 1.6180 \end{bmatrix} \rightarrow \Lambda^{10} = \begin{bmatrix} 0.0081 & 0 \\ 0 & 122.9919 \end{bmatrix}$$

Vanishing gradients

Exploding gradients

$$W_h^n = Q^{-1} * \Lambda^n * Q$$

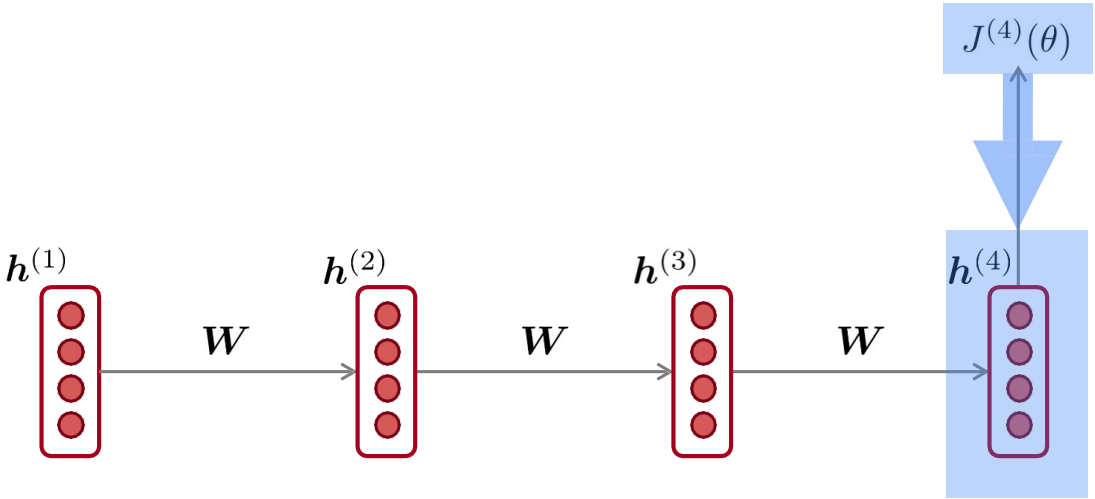
Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(3)}}$$

chain rule!

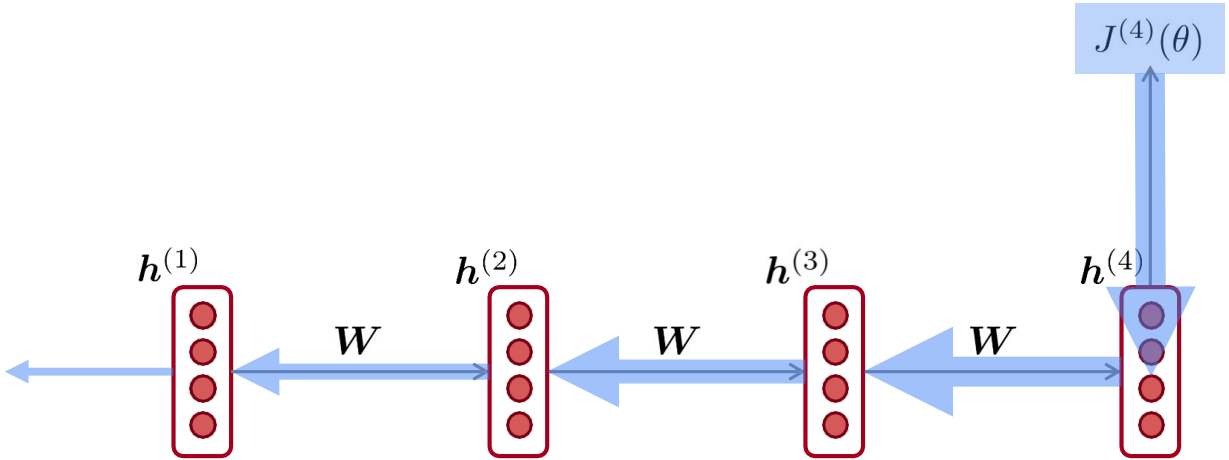
Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

chain rule!

Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Vanishing gradient proof sketch

- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$

- Therefore: $$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h$$ (chain rule)

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step *i*, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step *j*.

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$ (chain rule)

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{(i-j)} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)$$ (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

RNN Training Problem

Training recurrent neural networks can be very difficult. Two common issues with training recurrent neural networks are vanishing gradients and exploding gradients. Exploding gradients can occur when the gradient becomes too large and error gradients accumulate, resulting in an unstable network. Vanishing gradients can happen when optimization gets stuck at a certain point because the gradient is too small to progress. Gradient clipping can prevent these issues in the gradients that mess up the parameters during training.

Effect of vanishing gradient on RNN-LM

- **LM task:** *When she tried to print her _____, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her tickets.*

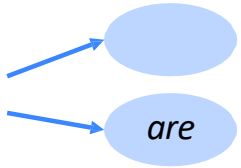
- To learn from this training example, the RNN-LM needs to model the dependency between *"tickets"* on the 7th step and the target word *"tickets"* at the end.

- But if gradient is small, the model can't learn this dependency
  - So the model is unable to predict similar long-distance dependencies at test time

Adapted from Abigail See

Effect of vanishing gradient on RNN-LM

- **LM task:** *The writer of the books ____*



- **Correct answer**: *The writer of the books <u>is</u> planning a sequel*
- **Syntactic recency:** *The <u>writer</u> of the books <u>is</u>* (correct)
- **Sequential recency:** *The writer of the <u>books</u> <u>are</u>* (incorrect)
- Due to vanishing gradient, RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we'd like [Linzen et al 2016]

"Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies", Linzen et al, 2016. https://arxiv.org/pdf/1611.01368.pdf

Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

learning rate

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

gradient

- This can cause bad updates: we take too large a step and reach a bad parameter configuration (with large loss)

- In the worst case, this will result in Inf or NaN in your network (then you have to restart training from an earlier checkpoint)

Gradient clipping: solution for exploding gradient

- Gradient clipping: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update

Algorithm 1 Pseudo-code for norm clipping

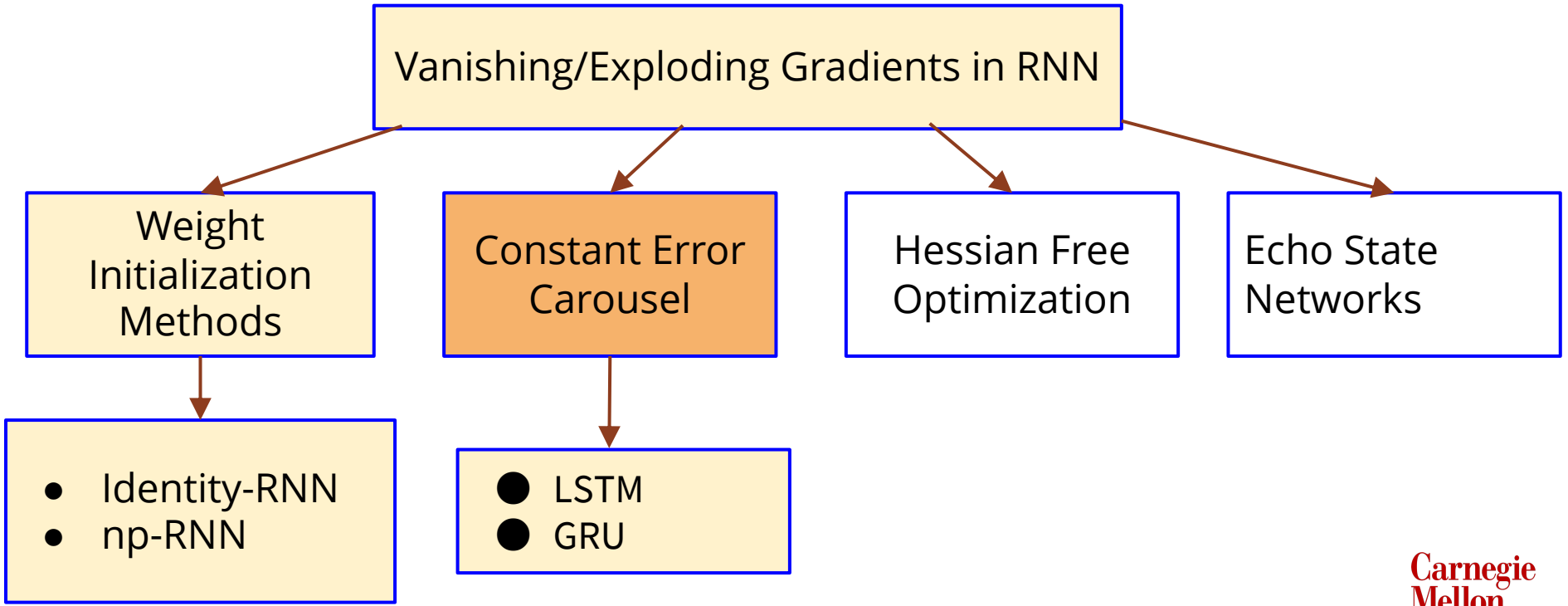
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

- Intuition: take a step in the same direction, but a smaller step

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Learning Long Term Dependencies

Outline


Vanishing/Exploding Gradients in RNN
Weight Initialization Methods
Constant Error Carousel
Hessian Free Optimization
Echo State Networks
Identity-RNN
np-RNN
LSTM
GRU

Weight Initialization Method

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \operatorname{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Activation function : ReLU

$$\frac{\partial h_j}{\partial h_k} = (\mathbf{W}_h^T)^n = Q^{-1} * \Lambda^n * Q$$

Weight Initialization Method

Random W_H initialization of RNN has no constraint on eigenvalues

$\Rightarrow$ vanishing or exploding gradients in the initial epoch

Weight Initialization Trick #1: IRNN

- W_H initialized to Identity
- Activation function: ReLU

Geoffrey et al, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units"

Weight Initialization Trick #2: np-RNN

- W_h positive definite (+ve real eigenvalues)
- At least one eigenvalue is 1, others all less than equal to one
- Activation function: ReLU

Geoffrey et al, “Improving Performance of Recurrent Neural Network with ReLU nonlinearity””

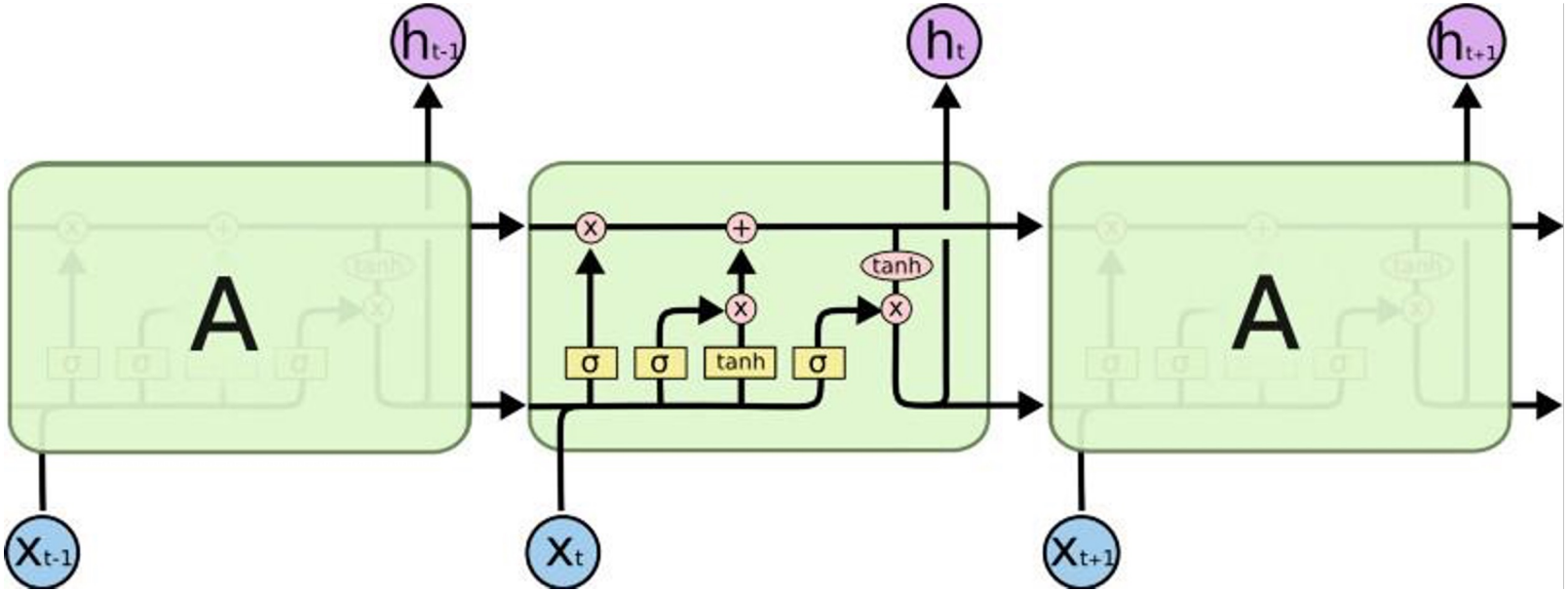
Weight Initialization Method

Careful initialization of W_H with suitable eigenvalues

$\Rightarrow$ allows the RNN to learn in the initial epochs

$\Rightarrow$ hence can generalize well for further iterations

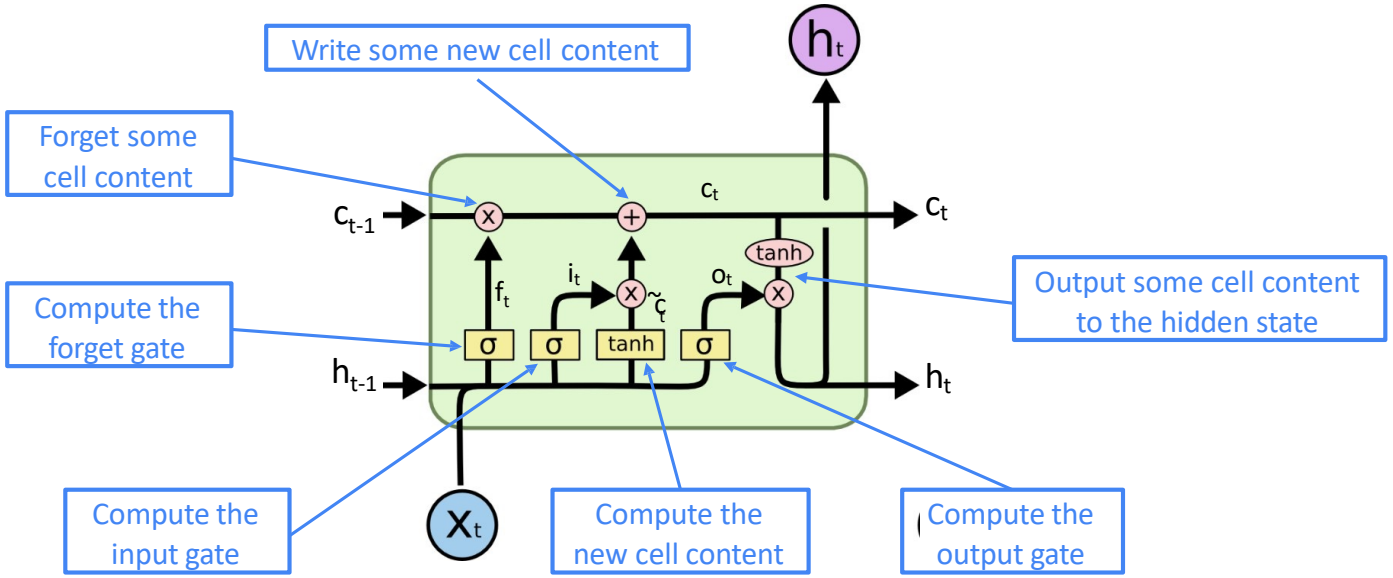
The LSTM Network

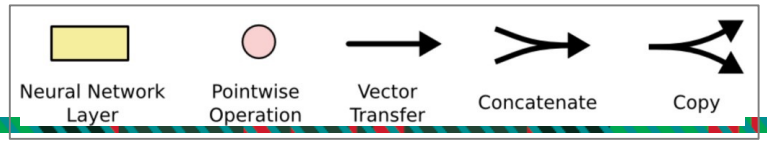


Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Review on your own: Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:





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

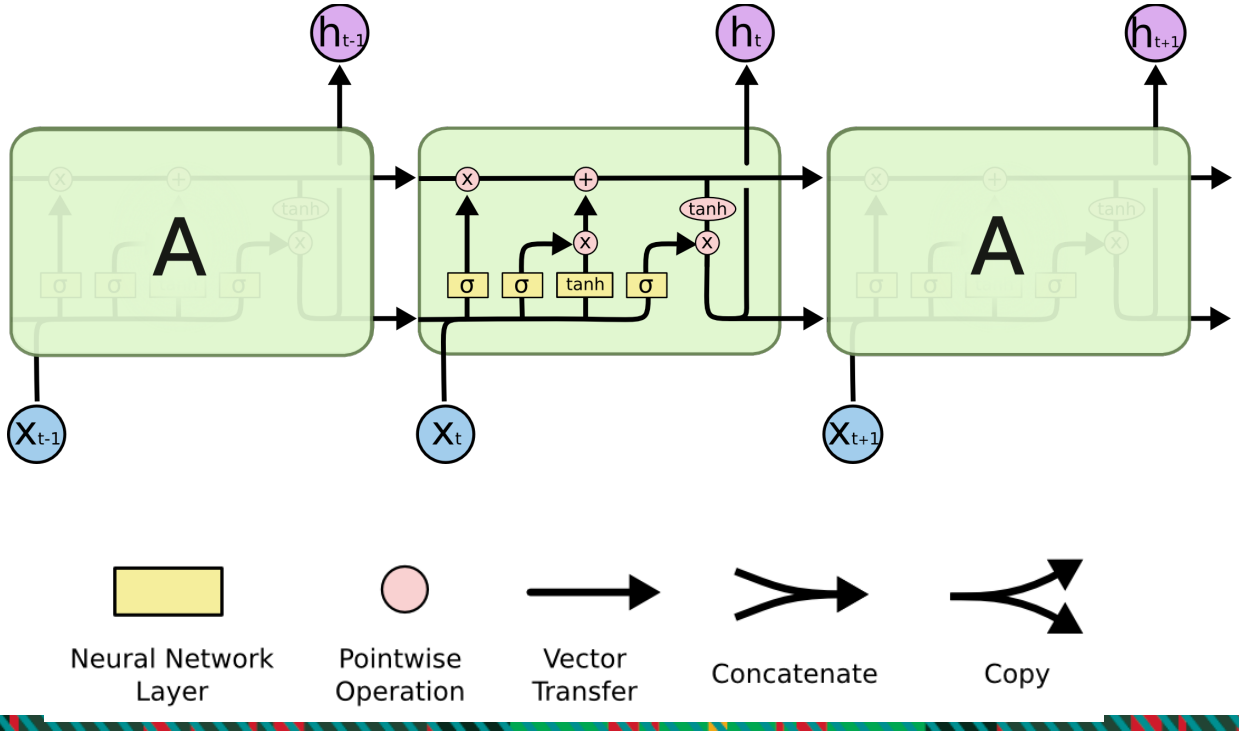
Long Short Term Memory

LSTM networks, add additional gating units in each memory cell.

- Forget gate
- Input gate
- Output gate

Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.

LSTM Network Architecture

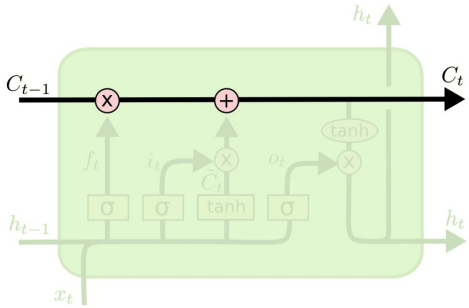


Carnegie Mellon University

Cell State

Maintains a vector C_T that is the same dimensionality as the hidden state, H_T

Information can be added or deleted from this state vector via the forget and input gates.



Cell State Example

Want to remember person & number of a subject noun so that it can be checked to agree with the person & number of verb when it is eventually encountered.

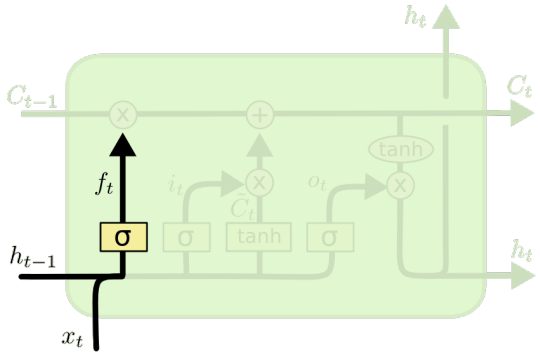
Forget gate will remove existing information of a prior subject when a new one is encountered.

Input gate "adds" in the information for the new subject.

Forget Gate

Forget gate computes a 0-1 value using a logistic sigmoid output function from the input, X_T, and the current hidden state, H_T:

Multiplicatively combined with cell state, "forgetting" information where the gate outputs something close to 0.

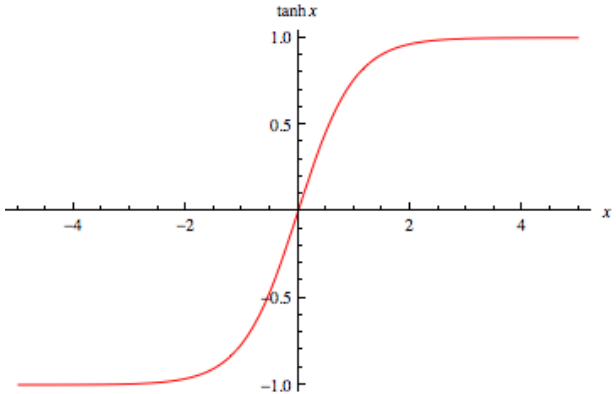


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Hyperbolic Tangent Units

Tanh can be used as an alternative nonlinear function to the sigmoid logistic (0-1) output function.

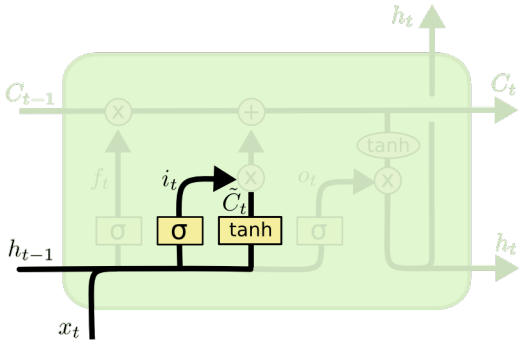
Used to produce thresholded output between –1 and 1.



Input Gate

First, determine which entries in the cell state to update by computing 0-1 sigmoid output.

Then determine what amount to add/subtract from these entries by computing a tanh output (valued -1 to 1) function of the input and hidden state.

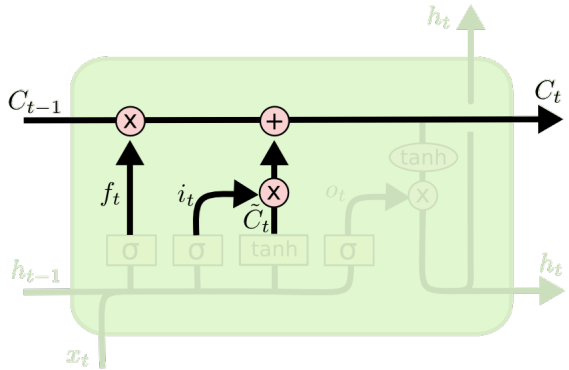


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

Updating the Cell State

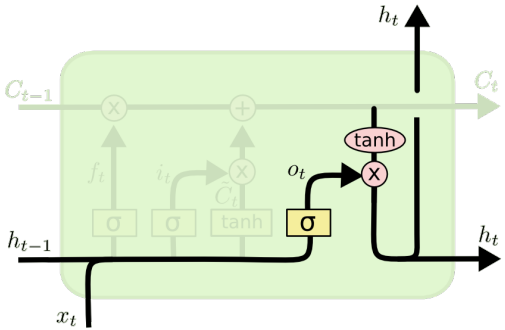
Cell state is updated by using component-wise vector multiply to "forget" and vector addition to "input" new information.



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

Output Gate

- Hidden state is updated based on a "filtered" version of the cell state, scaled to –1 to 1 using tanh.
- Output gate computes a sigmoid function of the input and current hidden state to determine which elements of the cell state to "output".

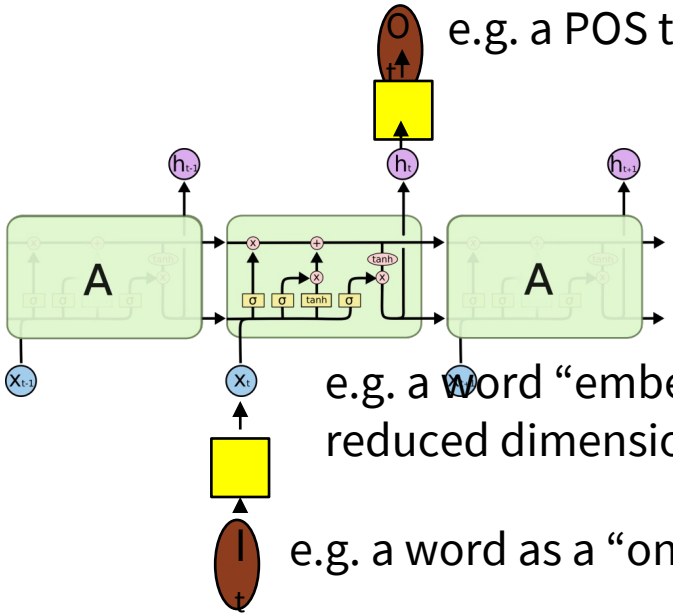


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Overall Network Architecture

SINGLE OR MULTILAYER NETWORKS CAN COMPUTE LSTM INPUTS FROM PROBLEM INPUTS AND PROBLEM OUTPUTS FROM LSTM OUTPUTS.



Carnegie Mellon University

LSTM Training

TRAINABLE WITH BACKPROP DERIVATIVES SUCH AS:

- Stochastic gradient descent (randomize order of examples in each epoch) with momentum (bias weight changes to continue in same direction as last update).
- ADAM optimizer (Kingma & Ma, 2015)

EACH CELL HAS MANY PARAMETERS (W_F, W_I, W_C, W_O)

- Generally requires lots of training data.
- Requires lots of compute time that exploits GPU clusters.

Long-short-term-memories (LSTMs)

- Proposed by Hochreiter and Schmidhuber in 1997
- We can make the units even more complex
- Allow each time step to modify
  - Input gate (current cell matters) $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$
  - Forget (gate 0, forget past) $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$
  - Output (how much cell is exposed) $o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$
  - New memory cell $\tilde{c}_t = \tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$
- Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- Final hidden state: $h_t = o_t \circ \tanh(c_t)$

Carnegie Mellon University

Adapted from Richard Socher

Why should we care about Language Modeling?

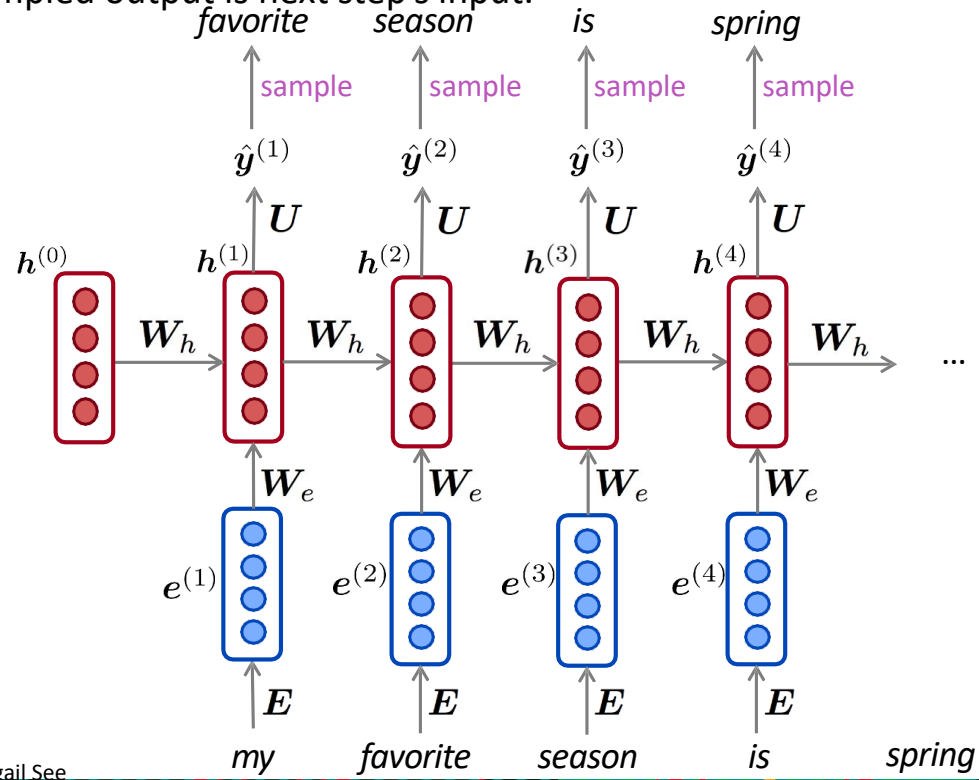
- Language Modeling is a benchmark task that helps us measure our progress on understanding language

- Language Modeling is a subcomponent of many NLP tasks, especially those involving generating text or estimating the probability of text:

    - Predictive typing
    - Speech recognition
    - Handwriting recognition
    - Spelling/grammar correction
    - Authorship identification
    - Machine translation
    - Summarization
    - Dialogue
    - etc.

Generating text with a RNN Language Model

You can use a RNN Language Model to generate text by repeated sampling.
Sampled output is next step's input.



Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on Obama speeches:

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.

Source: https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Generating text with a RNN Language Model

- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:

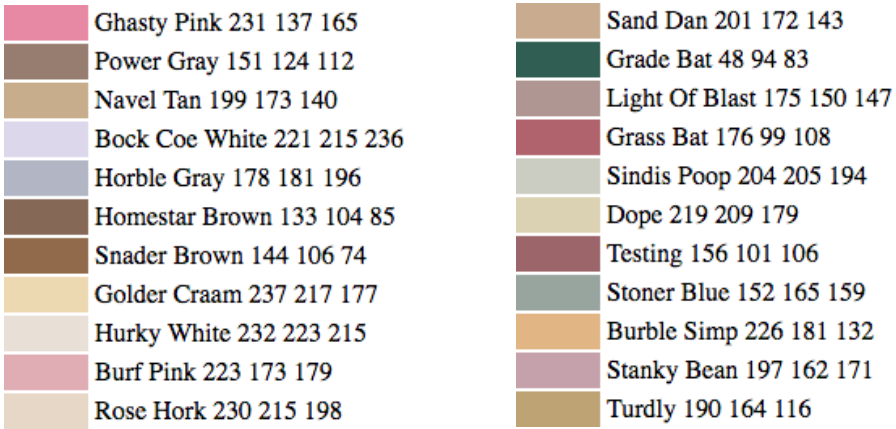
"Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"

"No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

Source: https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Generating text with a RNN Language Model

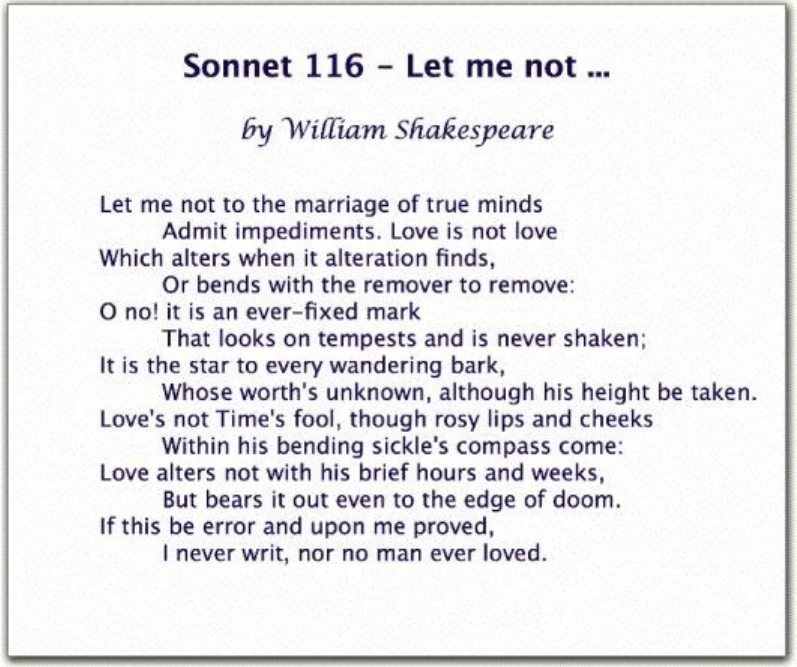
- Let's have some fun!
- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on paint color names:



This is an example of a character-level RNN-LM (predicts what character comes next)

Source: http://aiweirdness.com/post/160776374467/new-paint-colors-invented-by-neural-network

Generating poetry with RNNs



Sonnet 116 - Let me not ...

by William Shakespeare

Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no! it is an ever-fixed mark
That looks on tempests and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come:
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.

Generating poetry with RNNs

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.
```

More info: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Generating poetry with RNNs

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

```
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.
```

Generating textbooks with RNNs

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciely in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S,} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

*When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition **??** (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) *f is locally of finite type. Since $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation **??**. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

Generating code with RNNs

```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

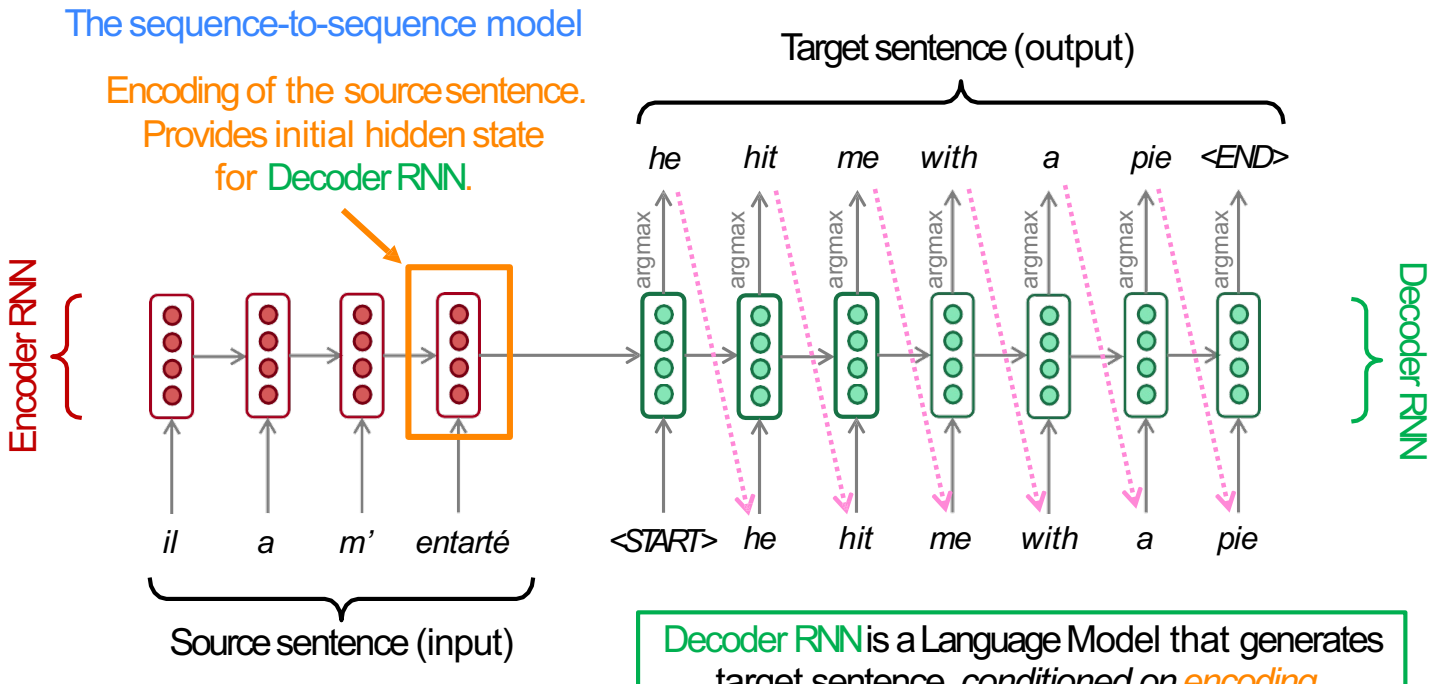
Generated
C code

Neural Machine Translation

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*

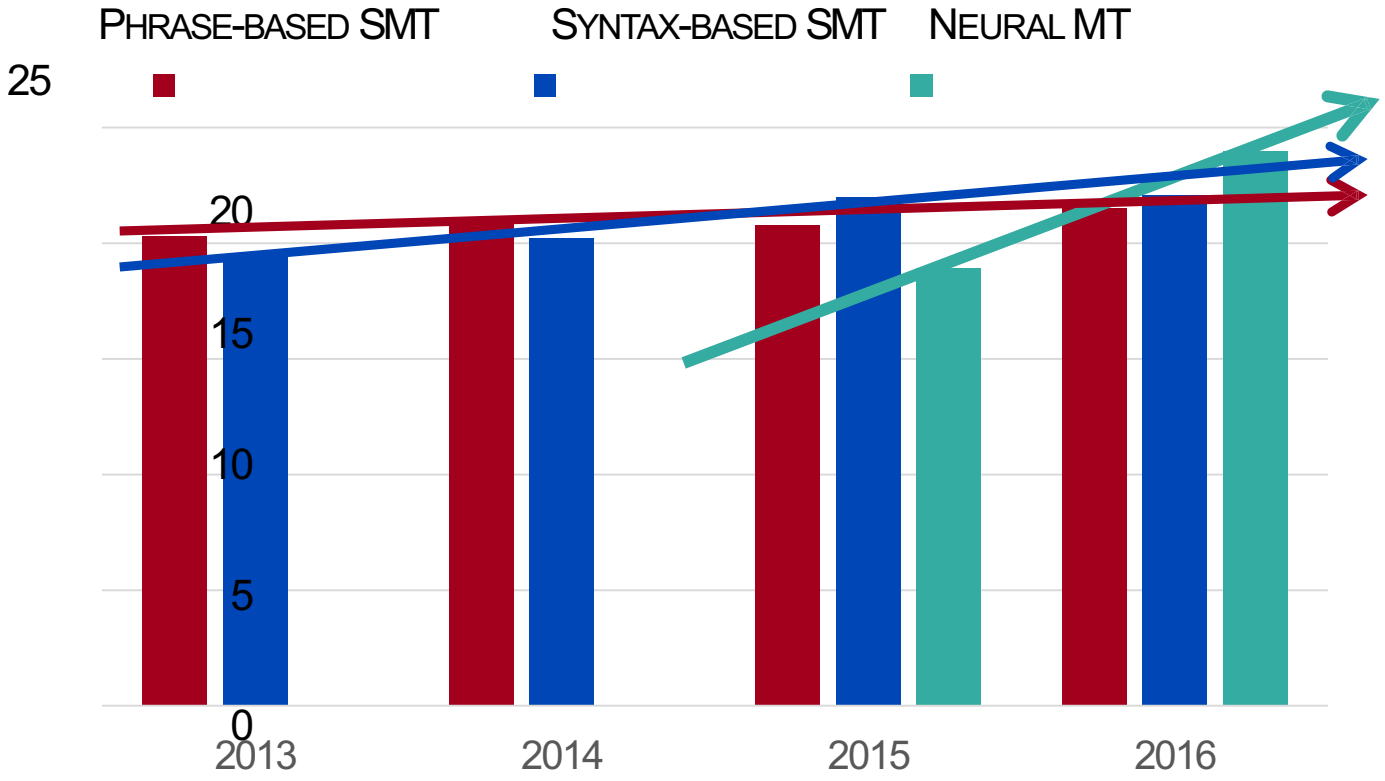
- The neural network architecture is called sequence-to-sequence (aka seq2seq) and it involves *two* RNNs.

Neural Machine Translation (NMT)



MT progress overtime

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal]



Source: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf

NMT: the biggest success story of NLP Deep Learning

Neural Machine Translation went from a fringe research activity in **2014** to the leading standard method in **2016**

- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT
- This is amazing!
  - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months

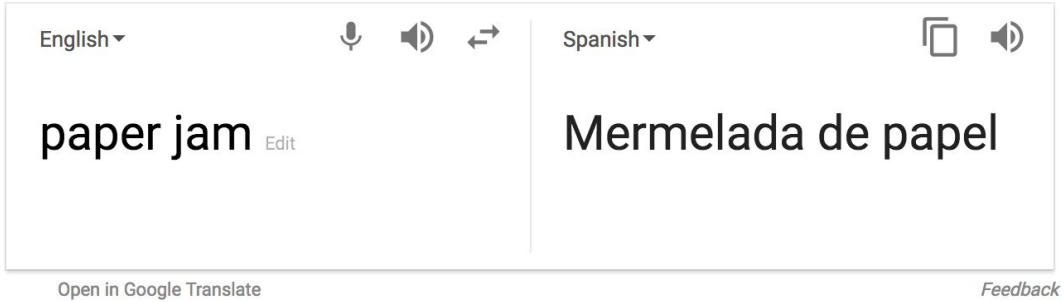
So is Machine Translation solved?

- **Nope!**
- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs

Further reading: "*Has AI surpassed humans at translation? Not even close!*"
https://www.skynettoday.com/editorials/state_of_nmt

So is Machine Translation solved?

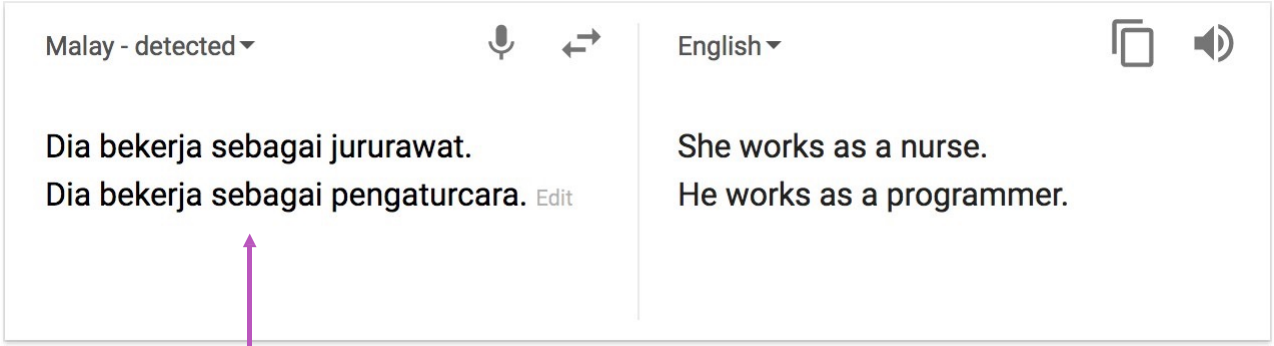
- **Nope!**
- Using common sense is still hard



?

So is Machine Translation solved?

- **Nope!**
- NMT picks up biases in training data



Source: https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c

NMT research continues

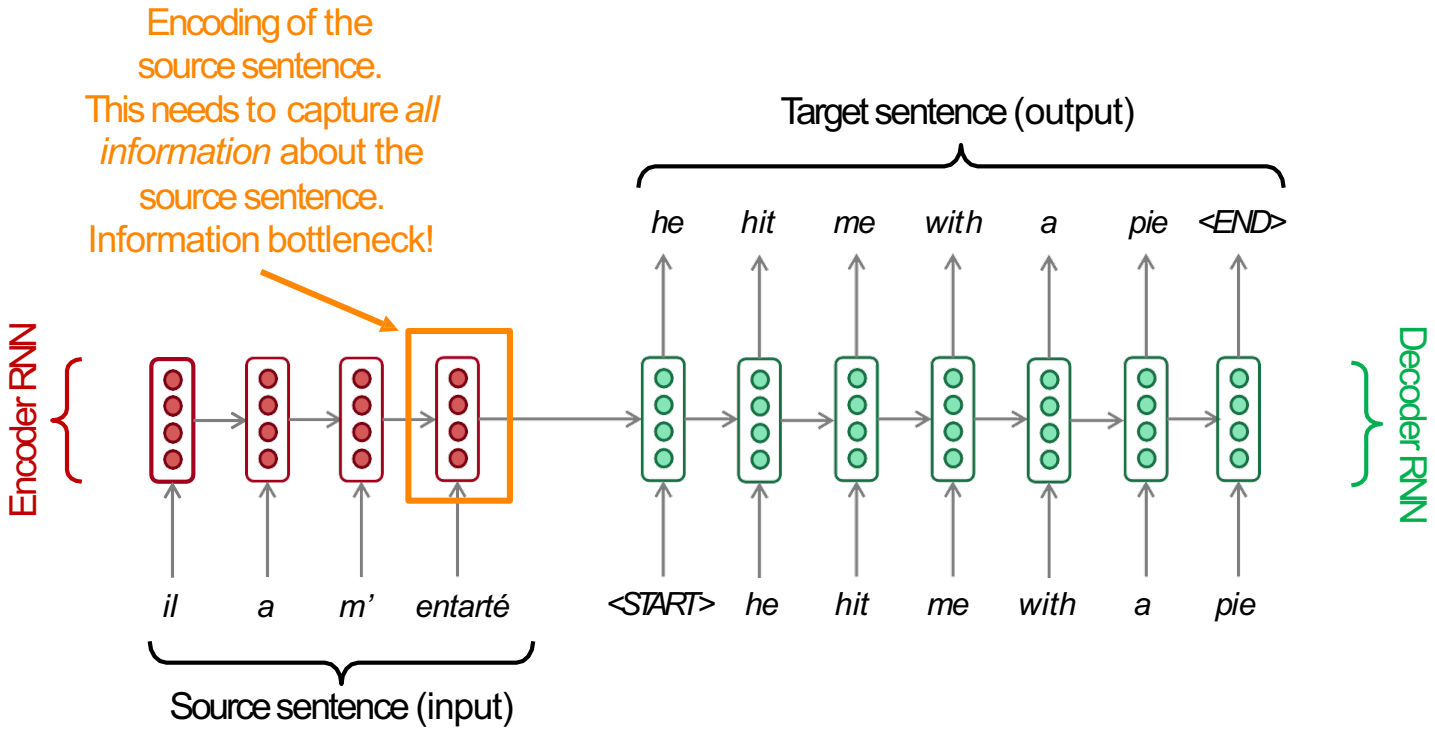
NMT is the **flagship task** for NLP Deep Learning

- NMT research has pioneered many of the recent innovations of NLP Deep Learning

- In **2019**: NMT research continues to thrive
  - Researchers have found *many, many* improvements to the "vanilla" seq2seq NMT system
  - But one improvement is so integral that it is the new vanilla…

ATTENTION

Sequence-to-sequence: the bottleneck problem



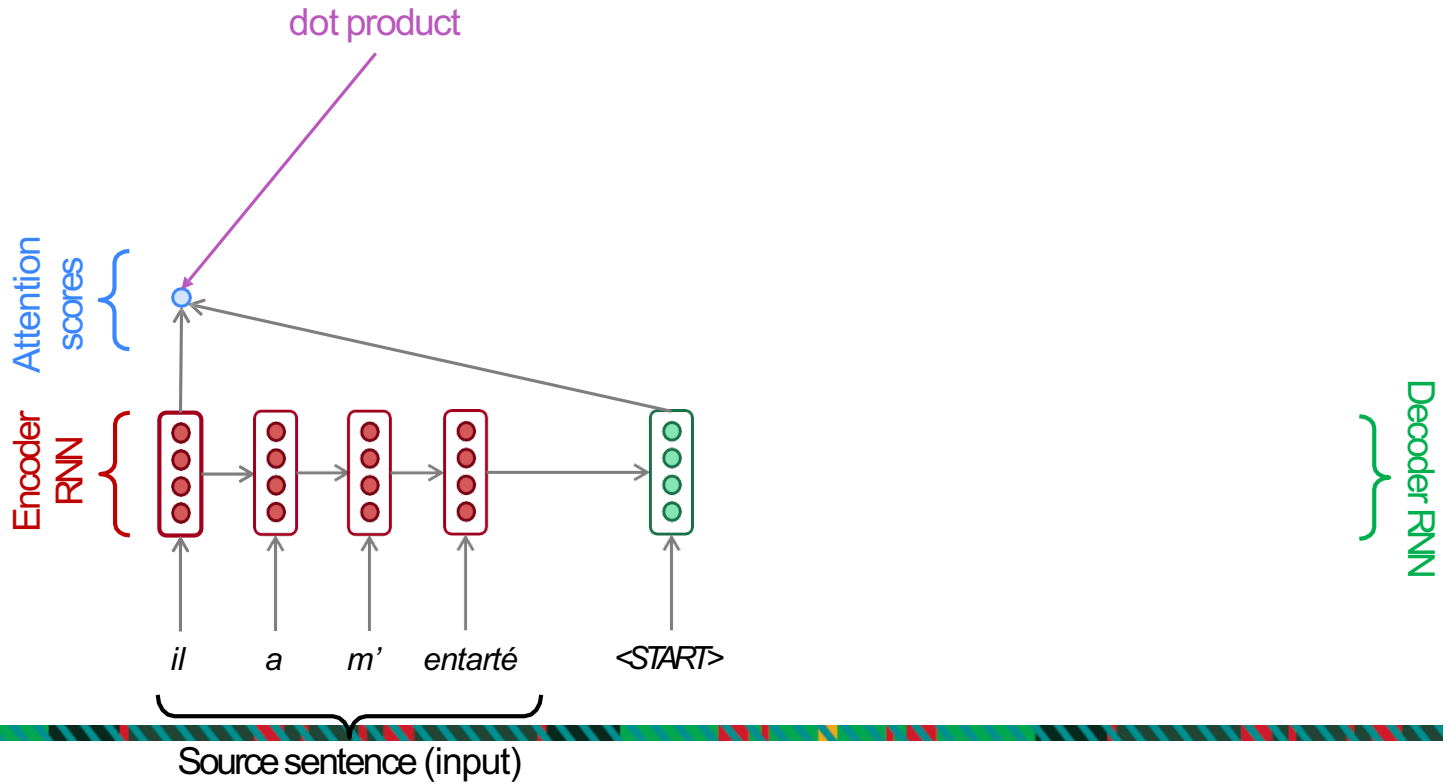
Attention

- **Attention** provides a solution to the bottleneck problem.

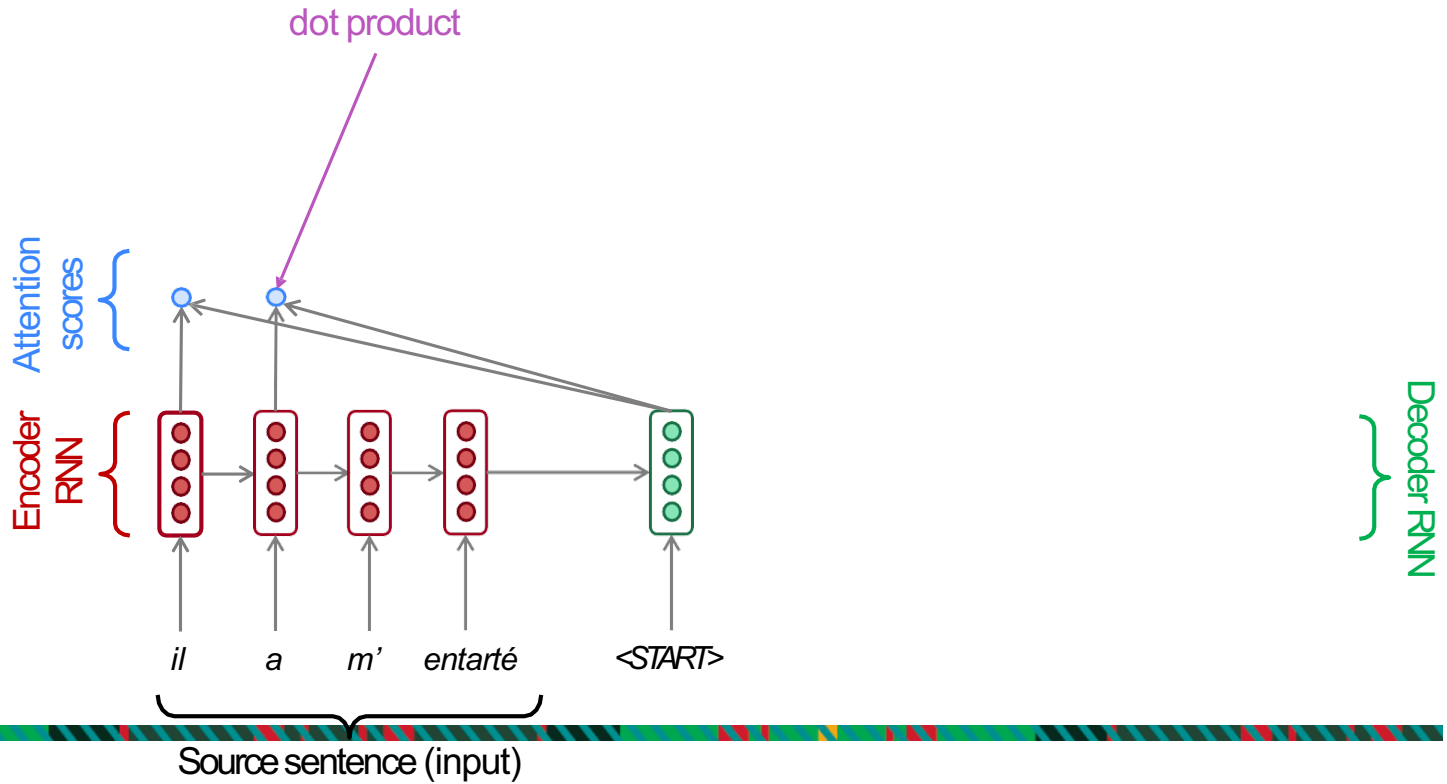
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First we will show via diagram (no equations), then we will show with equations

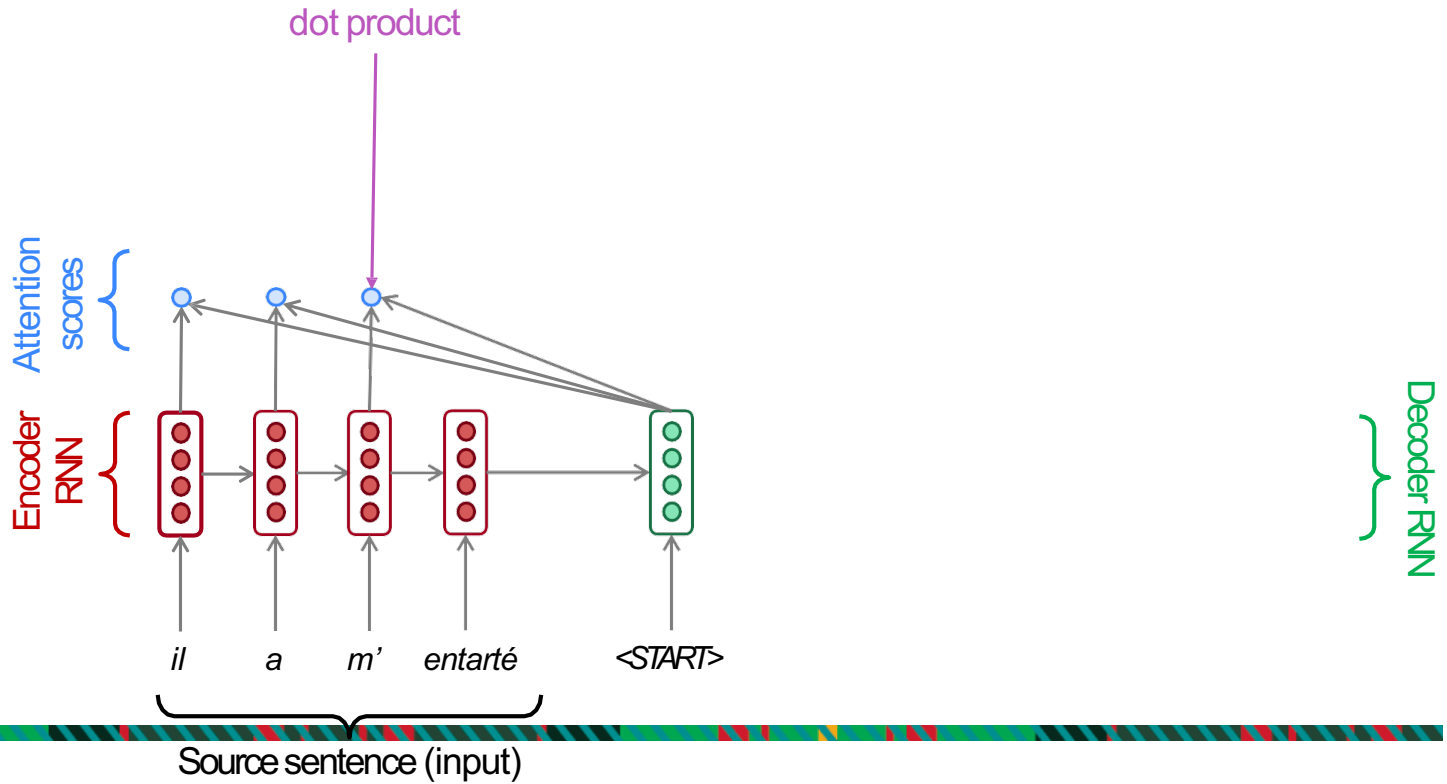
Sequence-to-sequence with attention



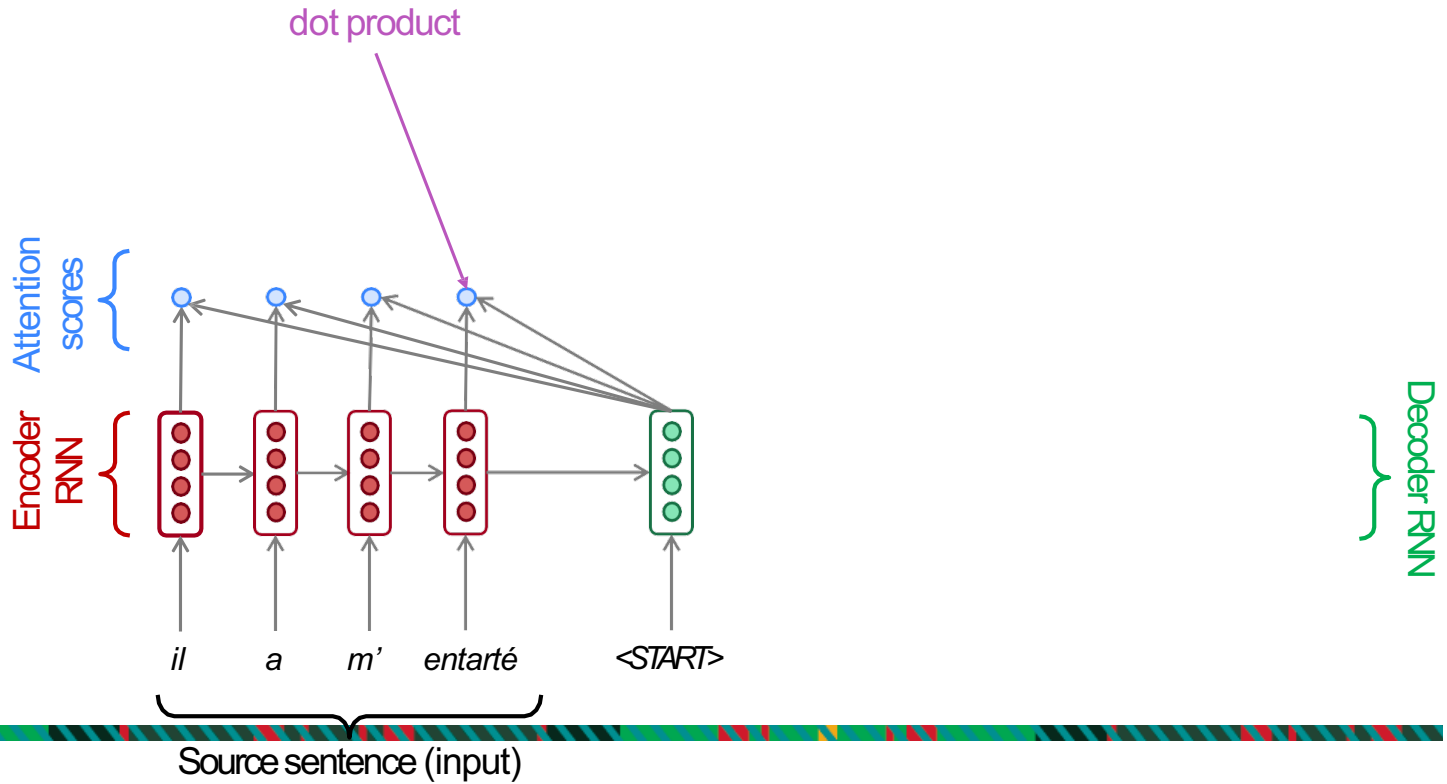
Sequence-to-sequence with attention



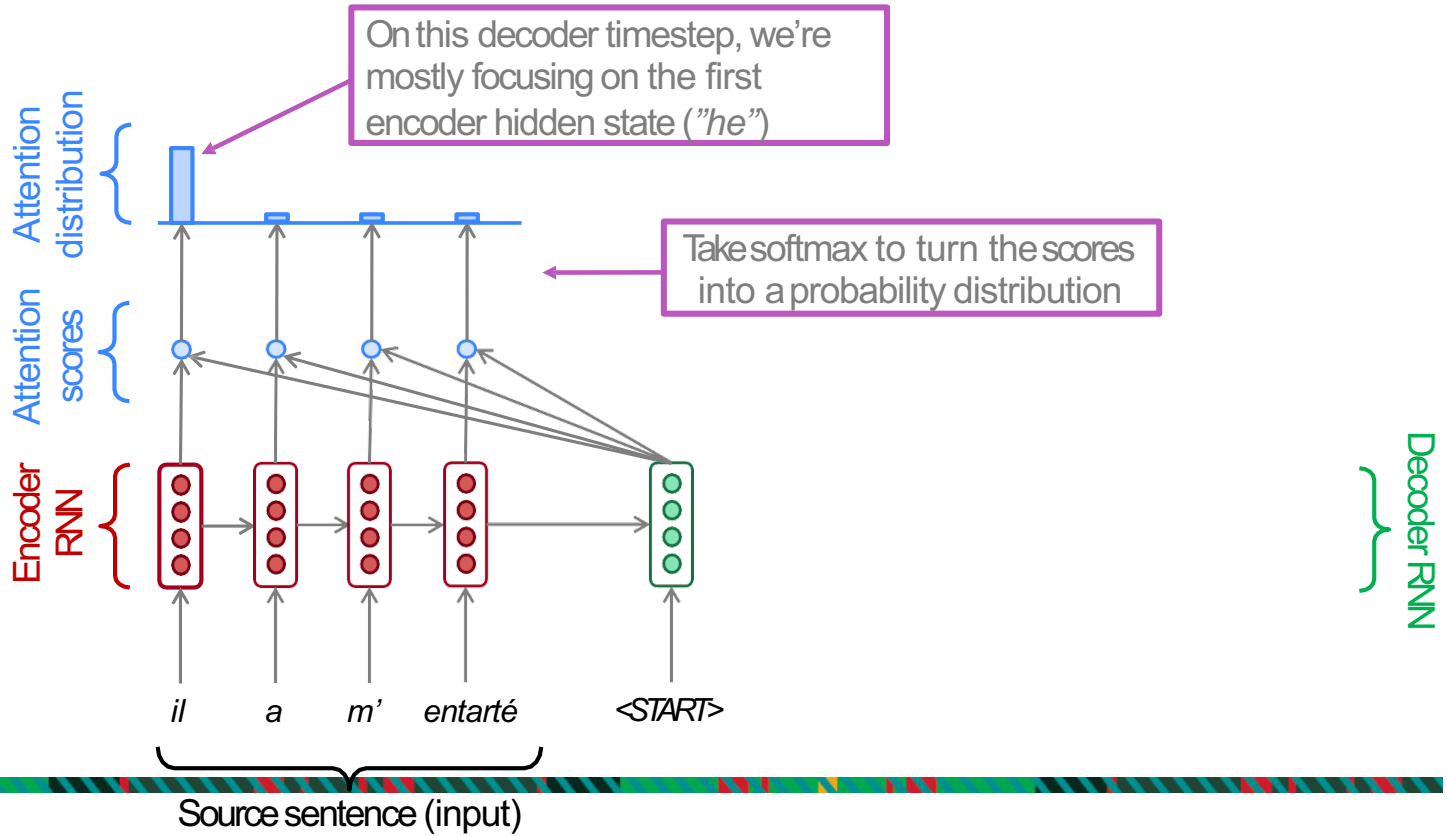
Sequence-to-sequence with attention



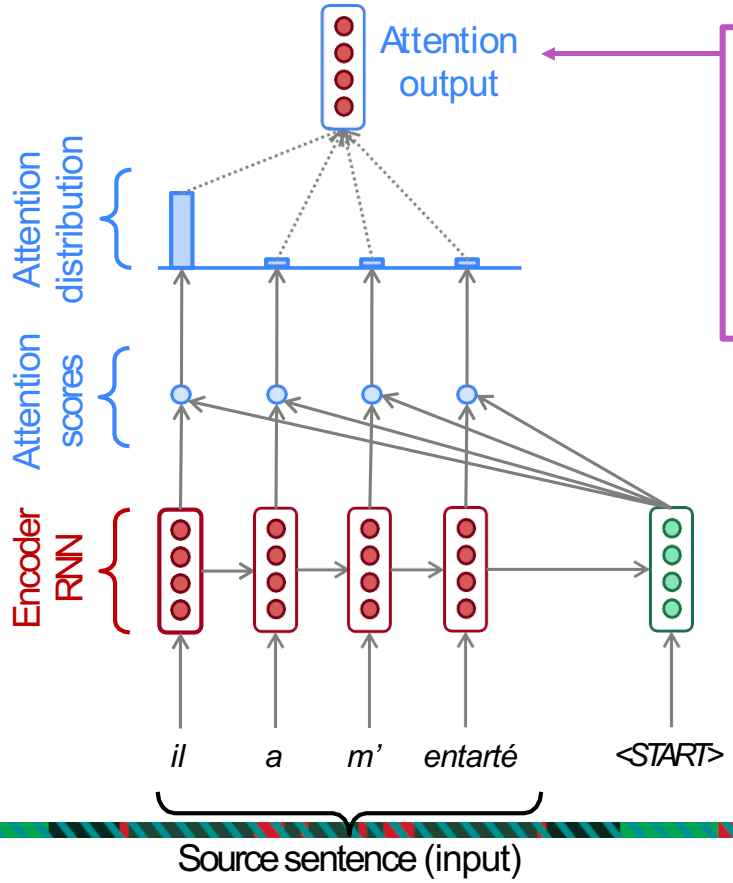
Sequence-to-sequence with attention



Sequence-to-sequence with attention



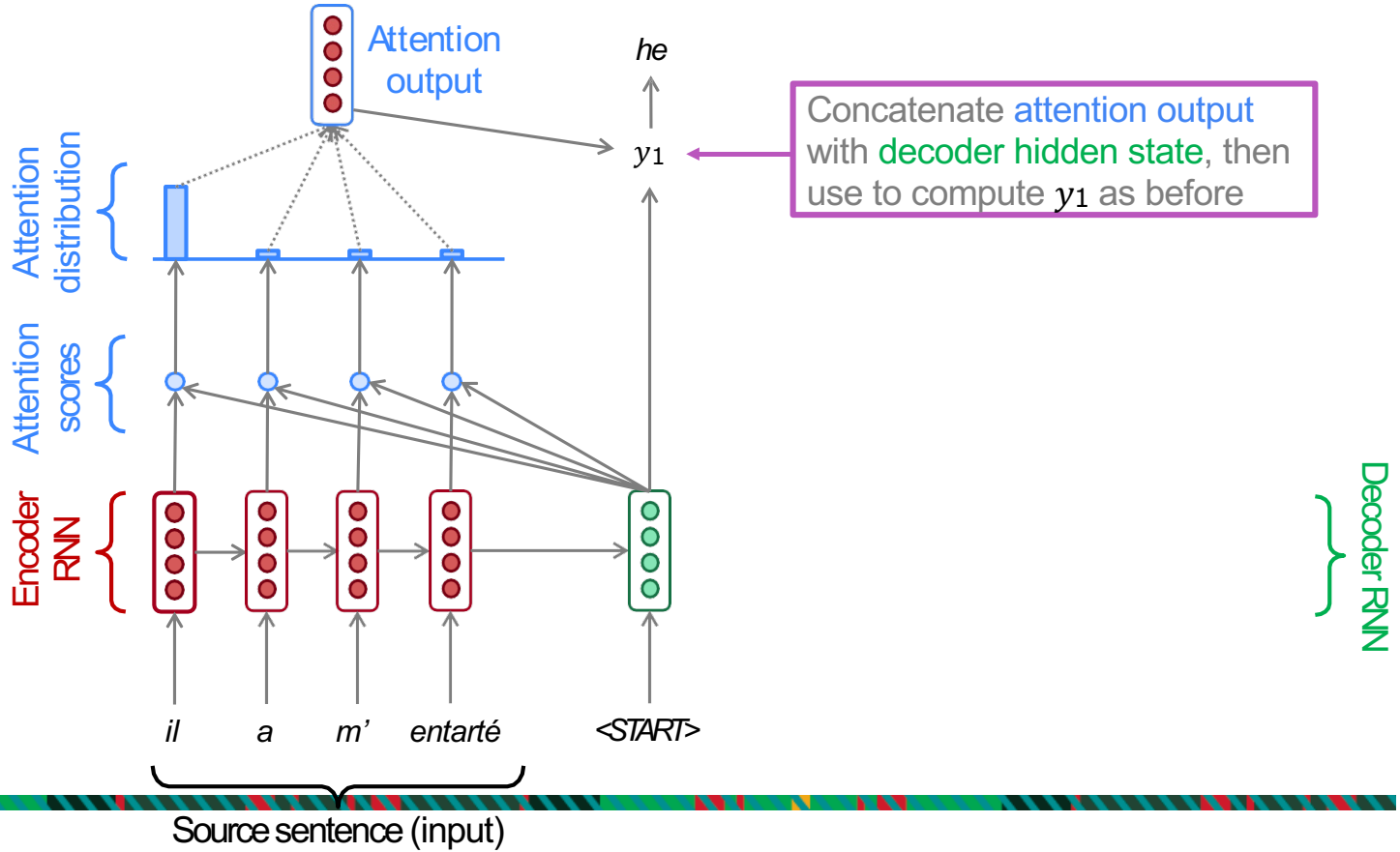
Sequence-to-sequence with attention



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.

Sequence-to-sequence with attention



Sequence-to-sequence with attention

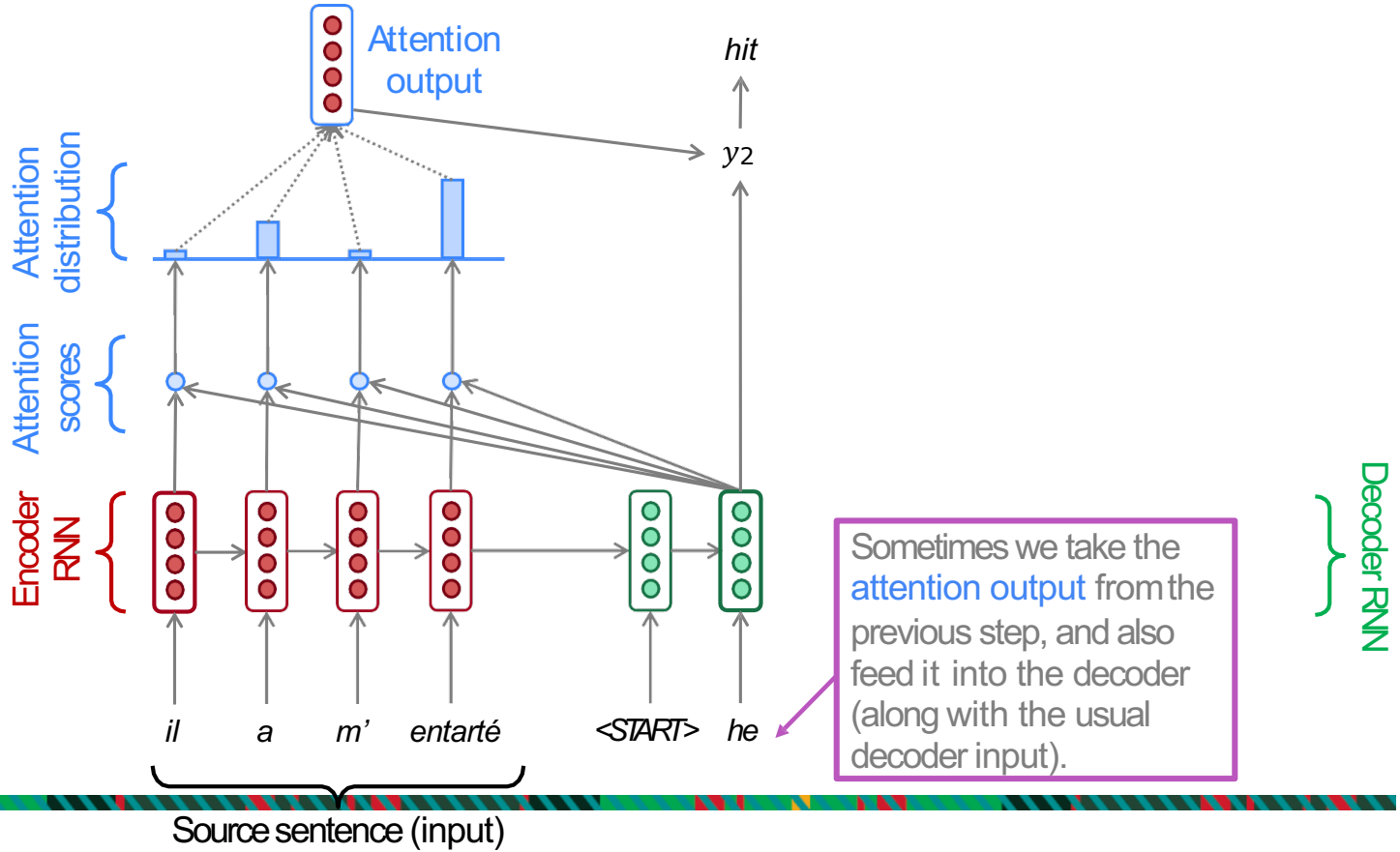
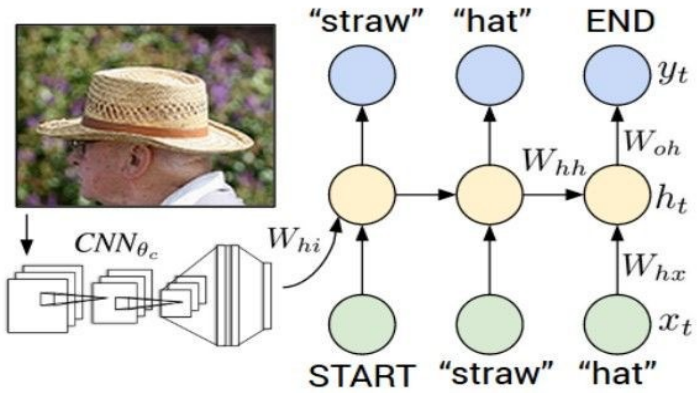


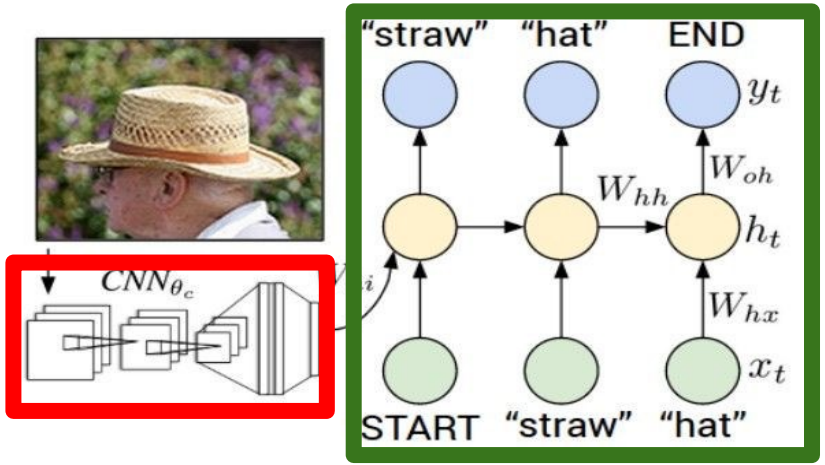
Image Captioning



CVPR 2015:
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

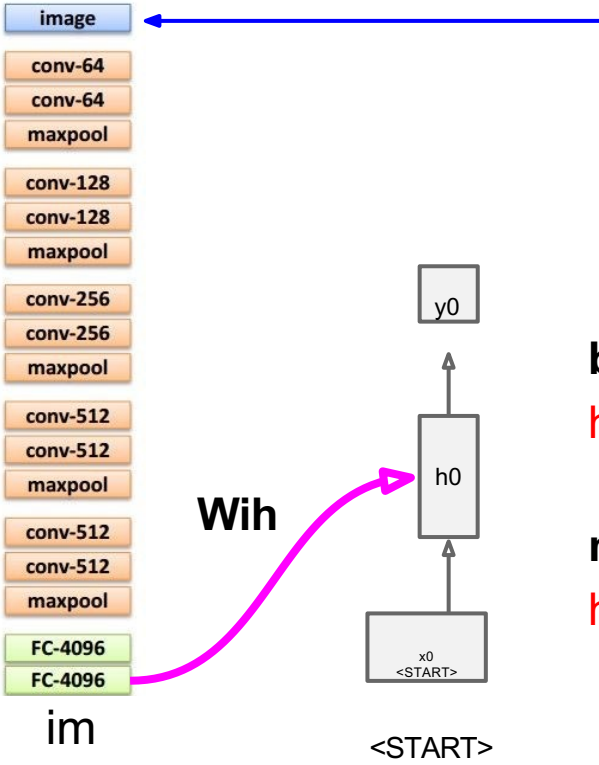
Recurrent Neural Network



Convolutional Neural Network

Image Captioning

Image Captioning



test image

before:

$h = \tanh(W_{xh} * x + W_{hh} * h)$

now:

$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * im)$

Image Captioning

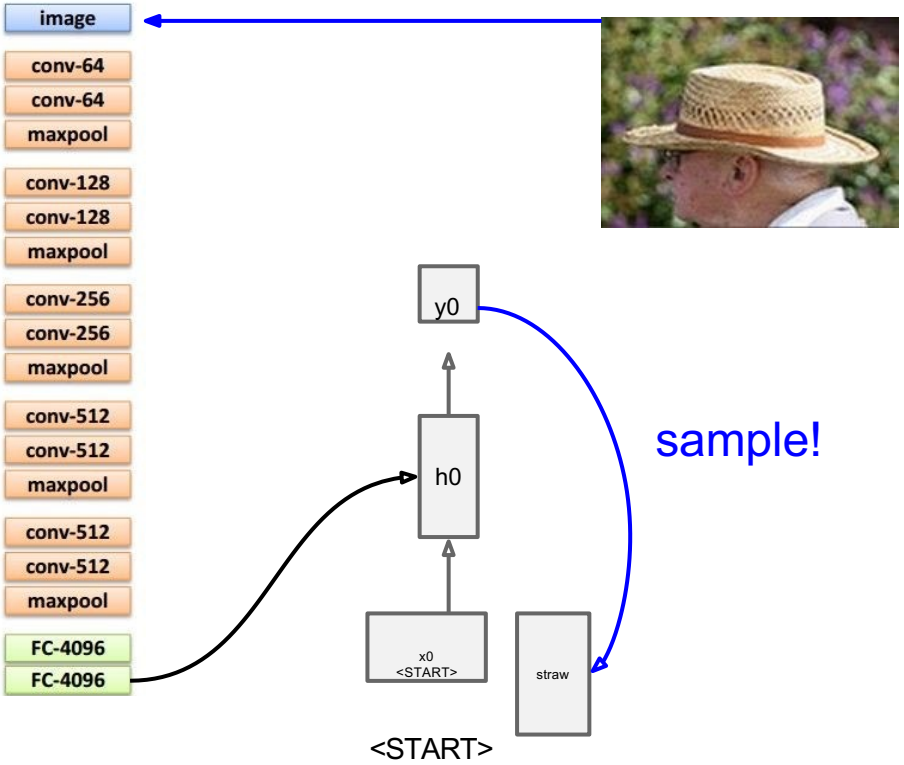


Image Captioning

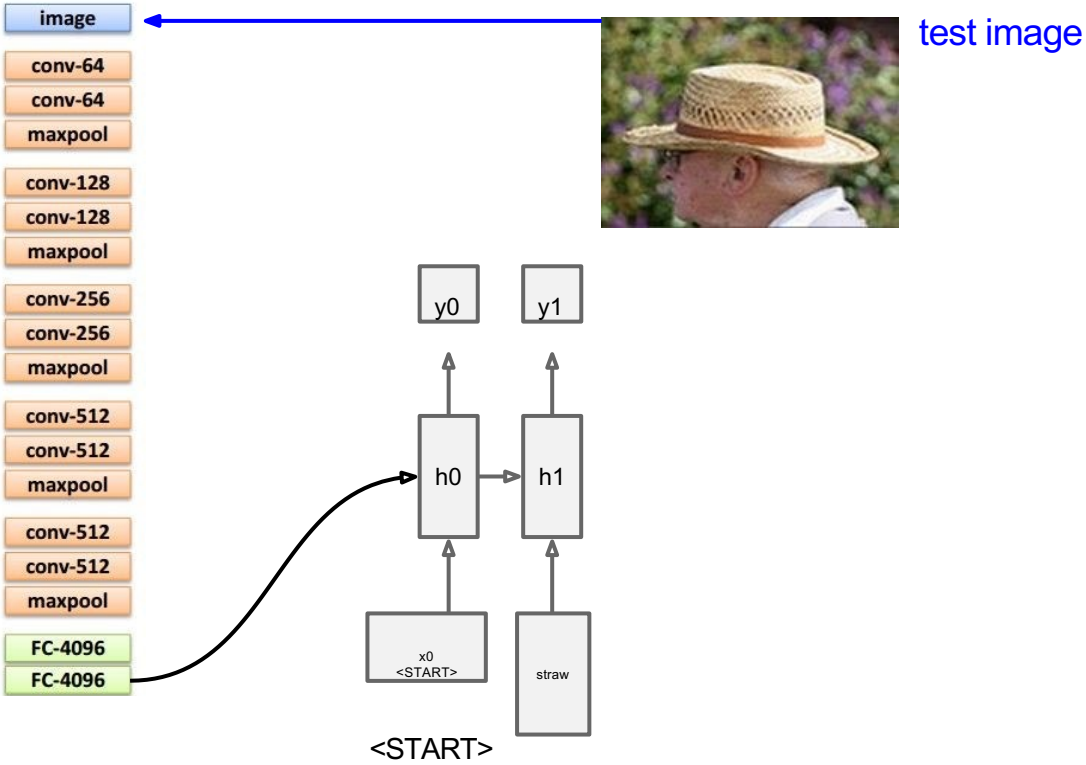


Image Captioning

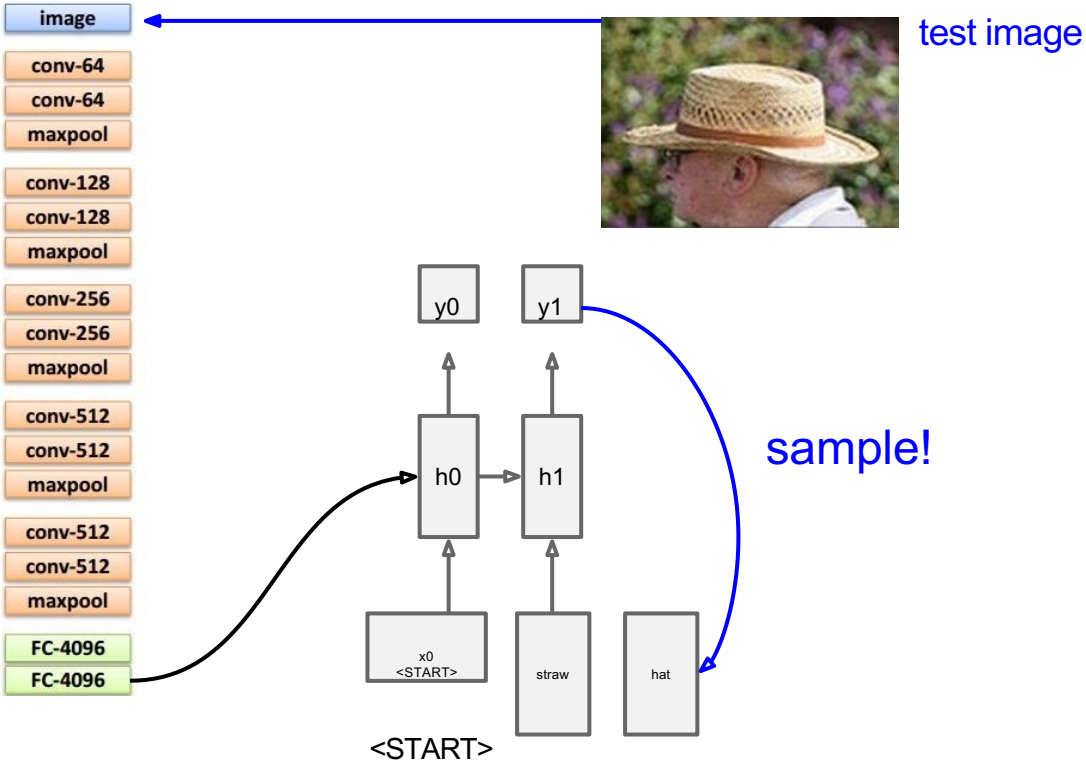


Image Captioning

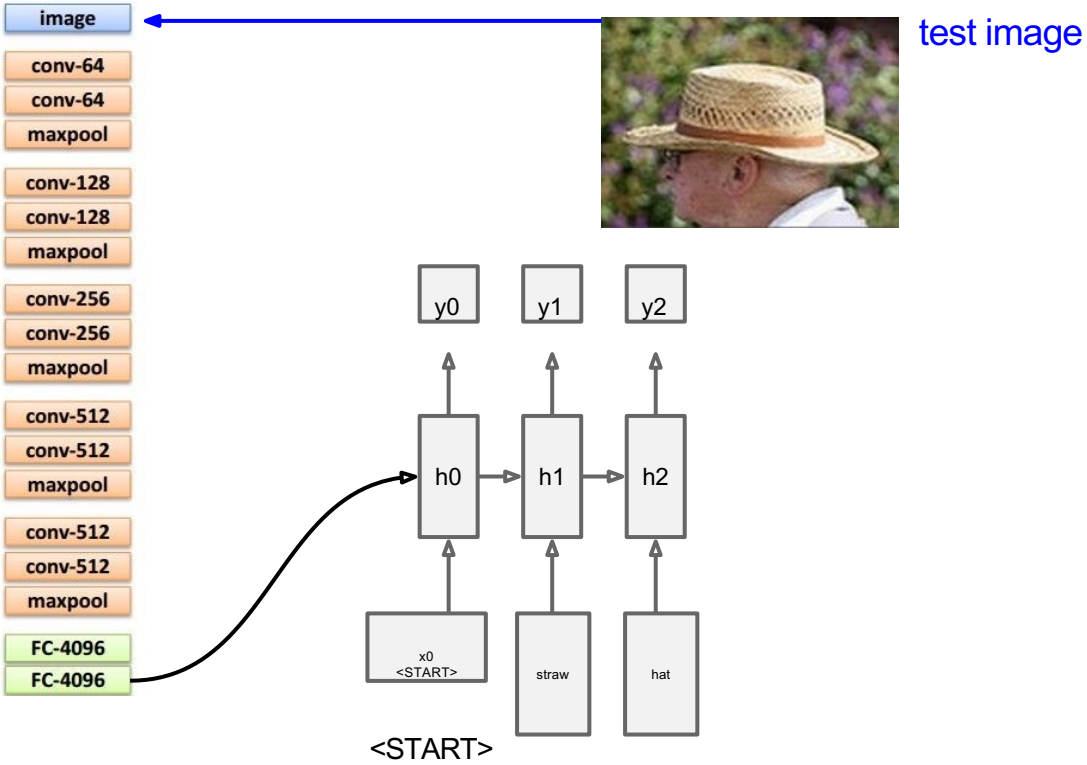


Image Captioning

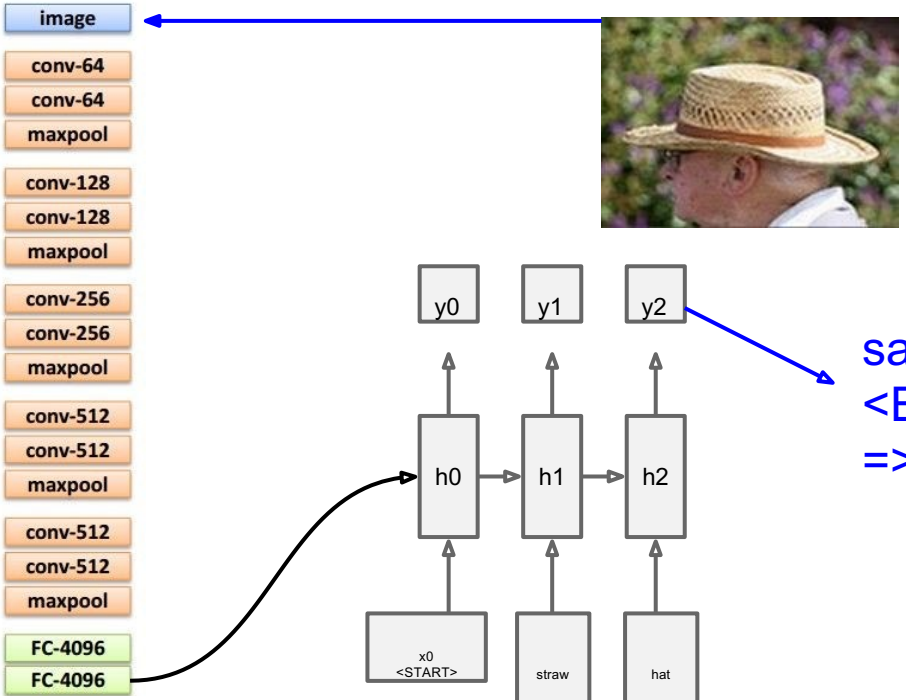


Image Captioning

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

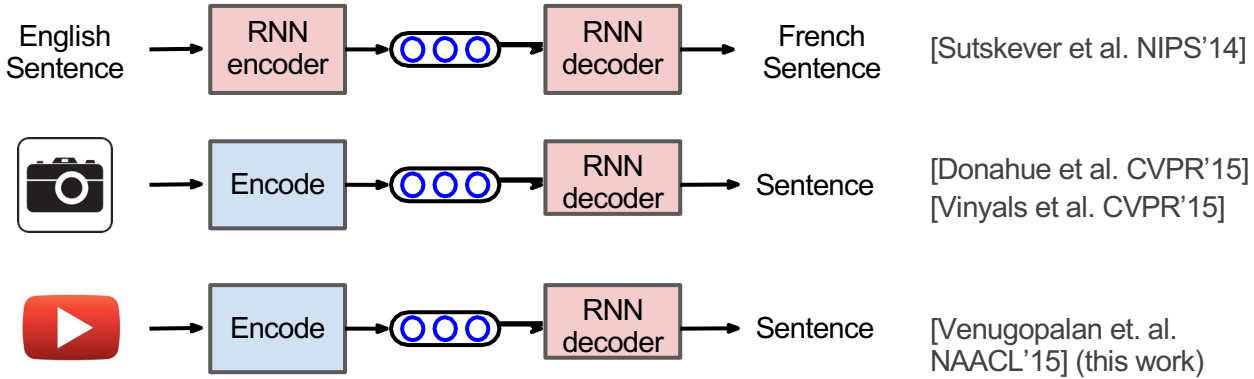
"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

Video Captioning

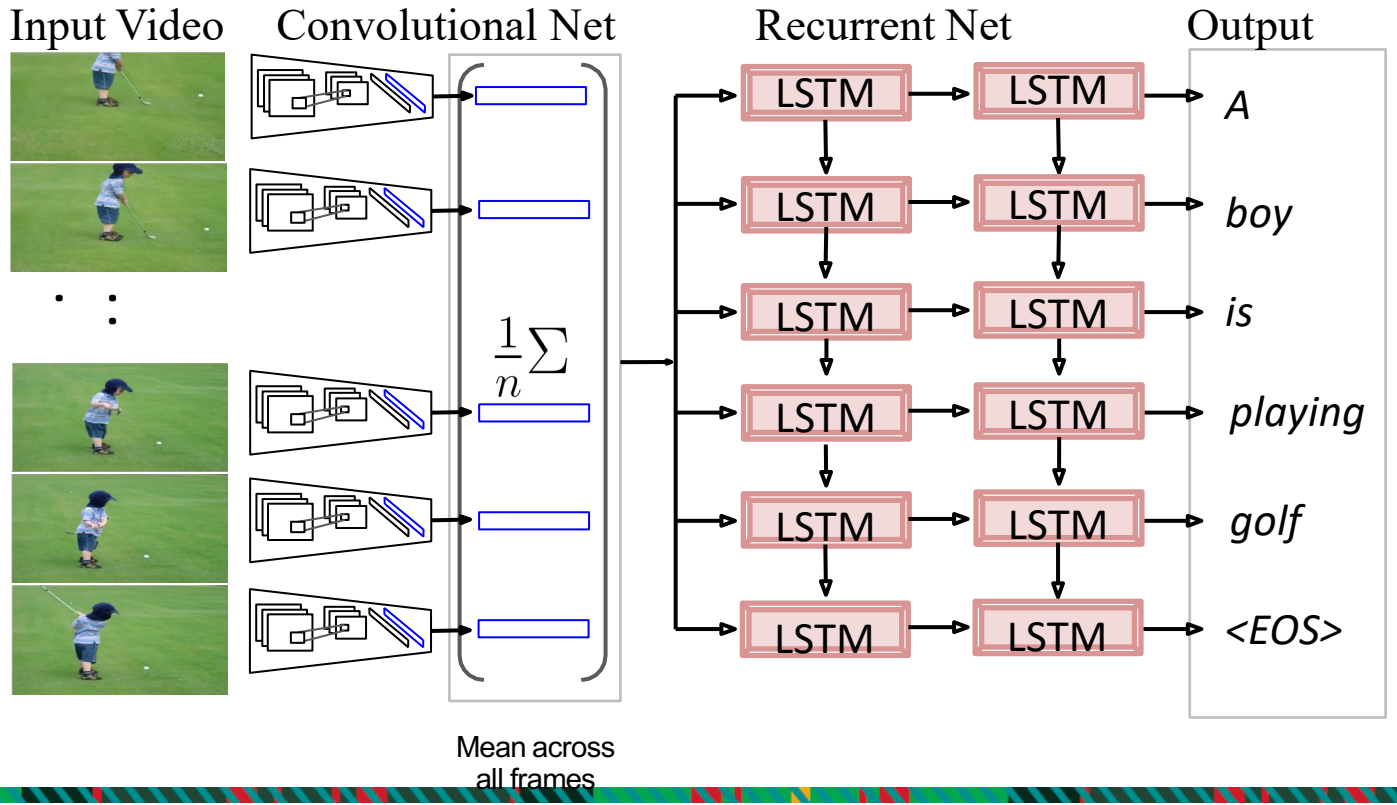


Key Insight:

Generate feature representation of the video and "decode" it to a sentence

Carnegie Mellon University

Video Captioning

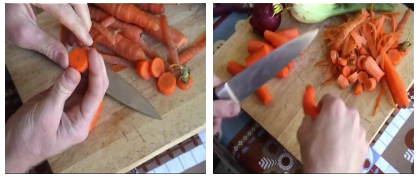


Venugopalan et al., "Translating Videos to Natural Language using Deep Recurrent Neural Networks", NAACL-HTL 2015

Carnegie Mellon University

Video Captioning

FGM: A person is dancing with the person on the stage.
YT: A group of men are riding the forest.
I+V: **A group of people are dancing.**
GT: Many men and women are dancing in the street.



FGM: A person is cutting a potato in the kitchen.
YT: A man is slicing a tomato.
I+V: **A man is slicing a carrot.**
GT: A man is slicing carrots.

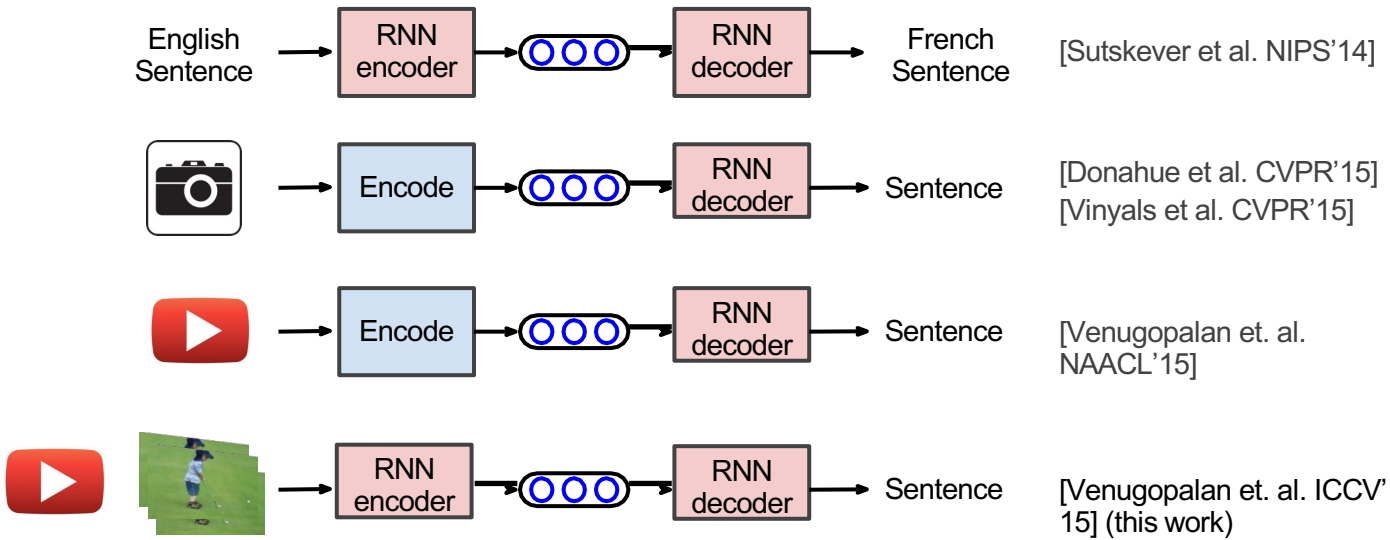
FGM: A person is walking with a person in the forest.
YT: A monkey is walking.
I+V: **A bear is eating a tree.**
GT: Two bear cubs are digging into dirt and plant matter at the base of a tree.

FGM: A person is riding a horse on the stage.
YT: A group of playing are playing in the ball.
I+V: **A basketball player is playing**.
GT: Dwayne wade does a fancy layup in an allstar game.

Carnegie Mellon University

Venugopalan et al., "Translating Videos to Natural Language using Deep Recurrent Neural Networks", NAACL-HTL 2015

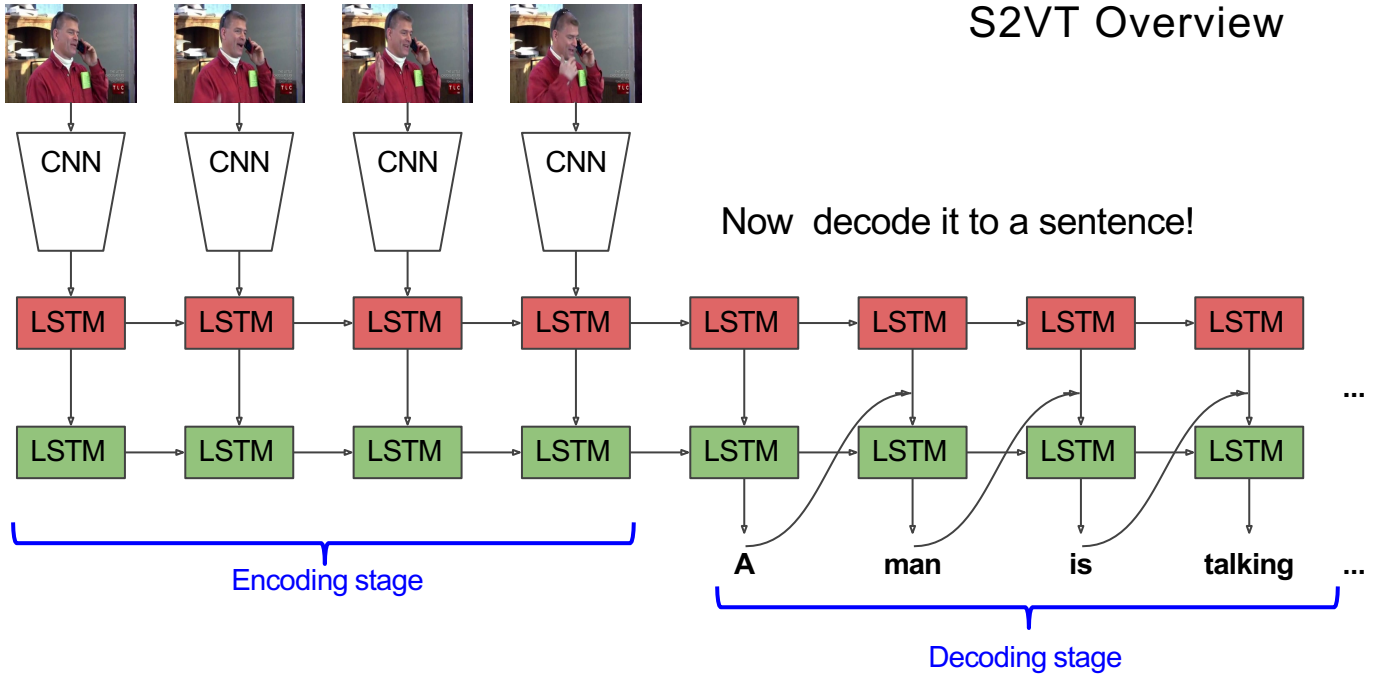
Video Captioning



Carnegie Mellon University

Venugopalan et al., "Sequence to Sequence - Video to Text", ICCV 2015

Video Captioning



Venugopalan et al., "Sequence to Sequence - Video to Text", ICCV 2015

Carnegie Mellon University