

---

# Deep Learning and Applications

# Today's Outline

---

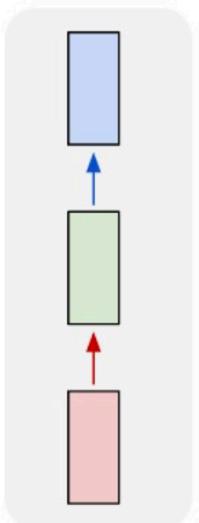
- Recurrent Neural Networks
- Long-Short Term Memory based RNNs
- Sequence to Sequence Learning and other RNN Applications

# Recurrent Neural Network

# RNN Application Categories

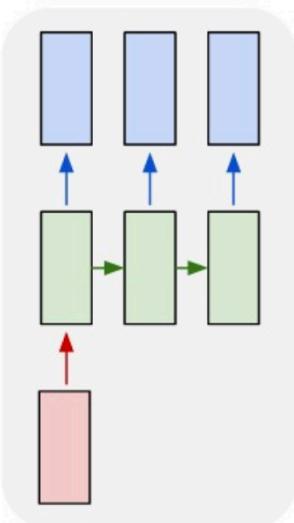
- Input: Red, Output: Blue, RNN's state: Green

one to one



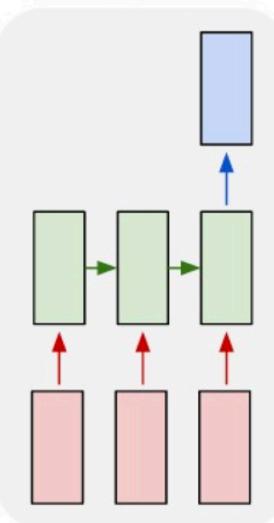
Classifier  
Fixed input  
Fixed output

one to many



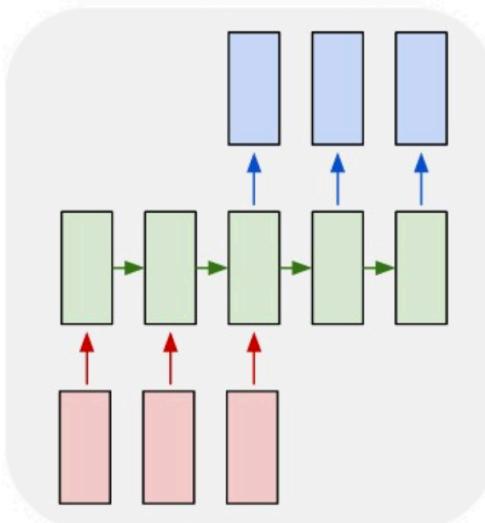
Sequence output  
E.g.: Image captioning

many to one



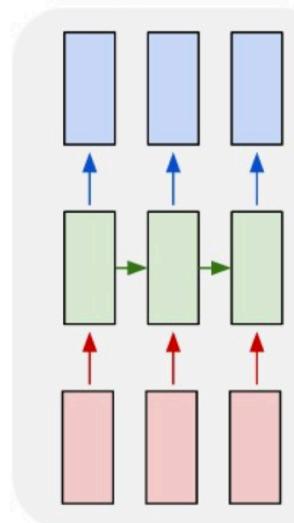
Sequence input  
E.g.: Sentiment analysis

many to many



Sequence input  
Sequence output  
E.g.: Machine translation

many to many



Sequence input  
Sequence output  
E.g.: Video classification

<sup>1</sup>Figure: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# The Idea of Persistence (I)

---

- Our thoughts have persistence
- We understand the present given what we have seen in the past
- Feedforward neural networks and CNNs don't explicitly model persistence
  - Example:
    - classify every scene in a movie
      - Output size (number of classes) is fixed
      - Number of layers is fixed
    - Unclear how a CNN can use information from previous scenes

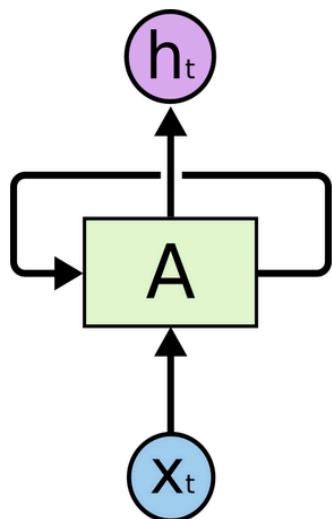
# The Idea of Persistence (II)

---

- Architectures called Recurrent Neural Networks address the idea of persistence explicitly

# Unrolled Diagrams (I)

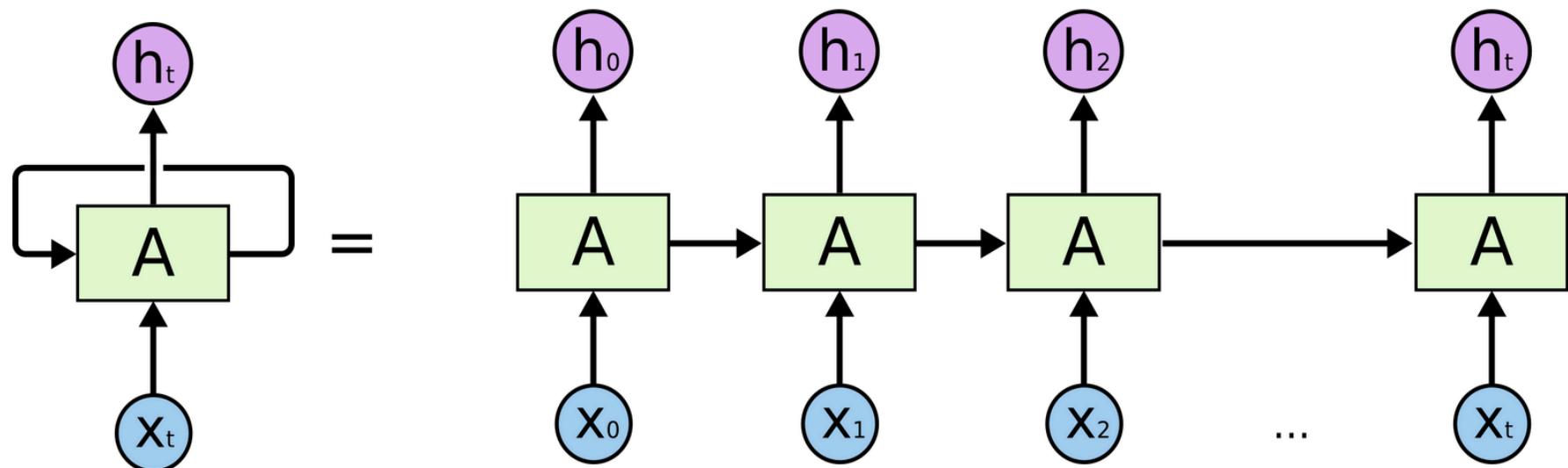
- Let  $A$  represent a base **network** with two inputs and two outputs
- A **loop** based drawing of the architecture is as follows:



<sup>1</sup>Figure: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

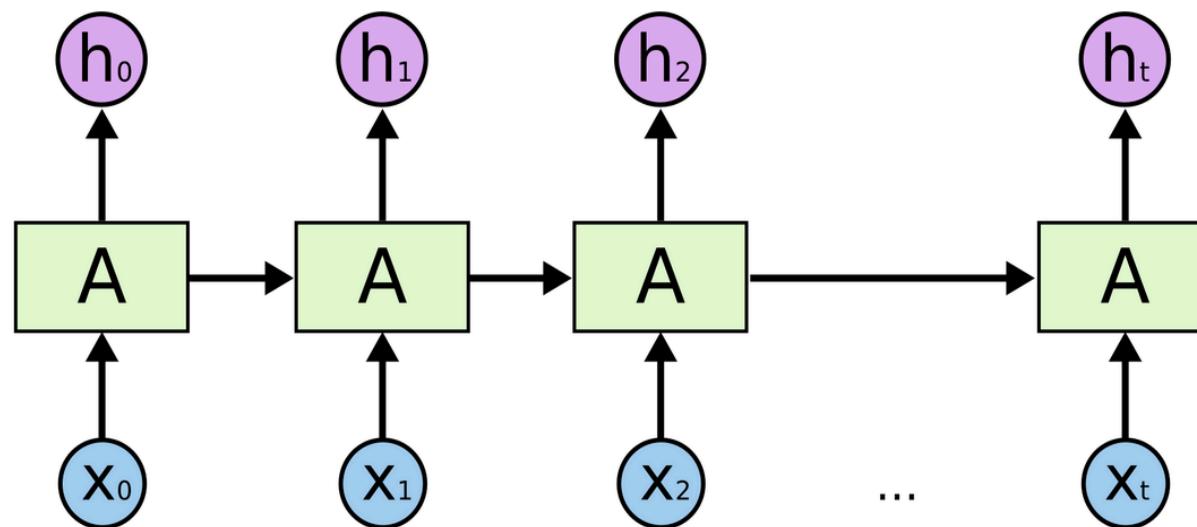
# Unrolled Diagrams (II)

- Here is the unrolled representation



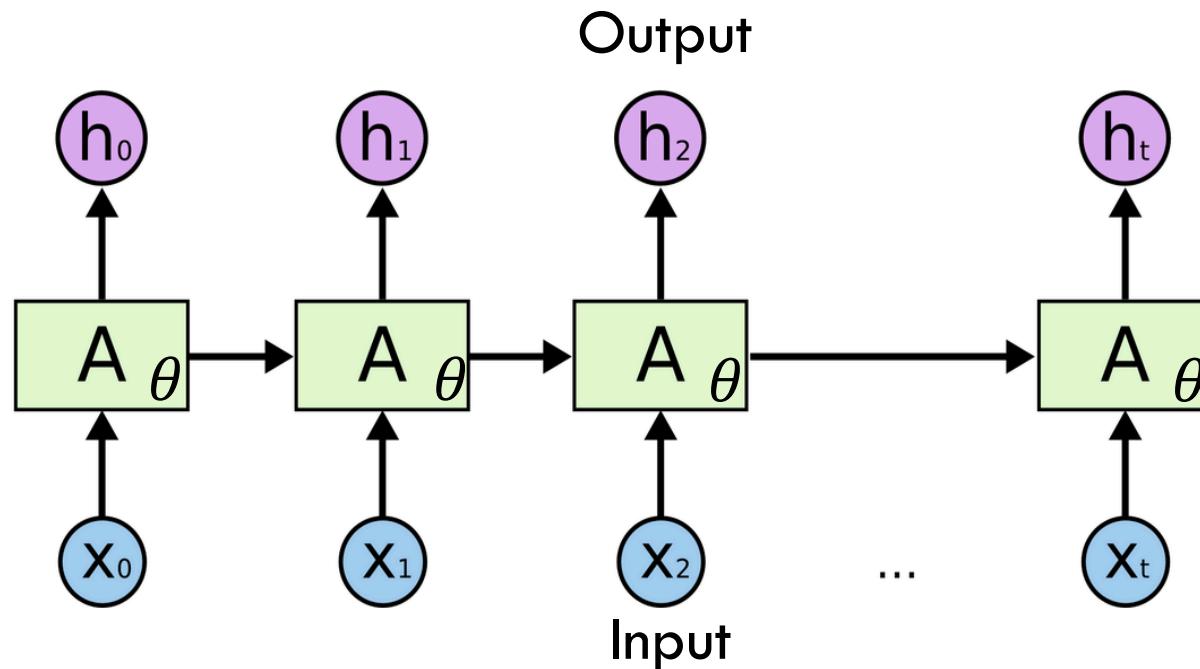
# Unrolled Diagrams (III)

- This sequential or repetitive structure is useful for working with sequences
  - Of images
  - Of words



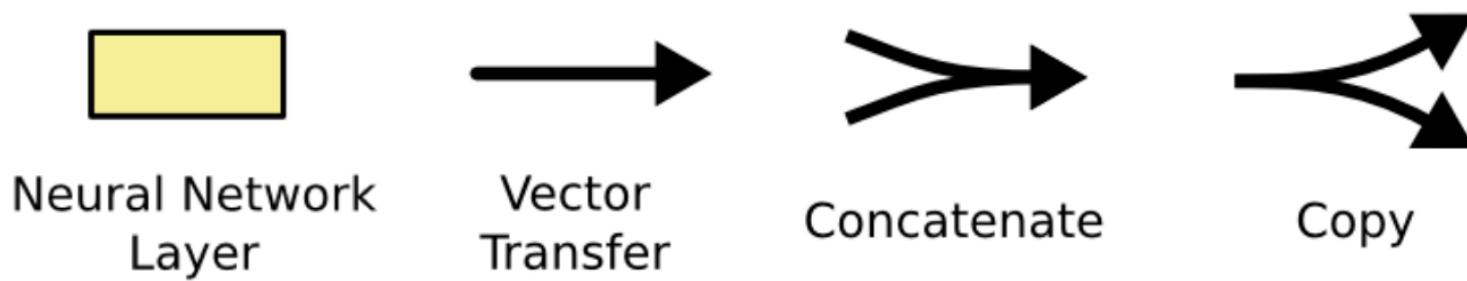
# Unrolled Diagrams (V)

- At a *stage*, they accept an input and give an output, which are parts of sequences



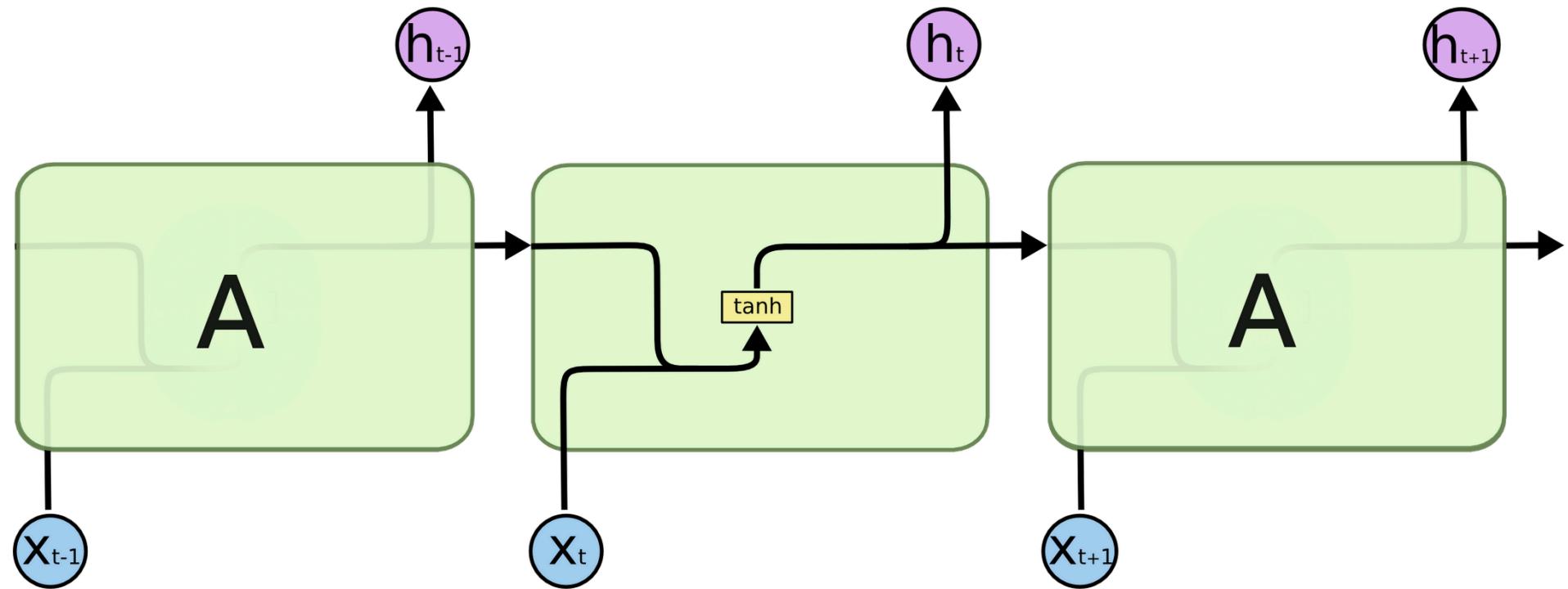
# Vanilla RNN (I)

- Some quick notation
  - Dark arrow represents a vector
  - Box represents a (fully connected hidden) layer



# Vanilla RNN (II)

- Unrolled representation is key to understanding
  - For vanilla RNN it is:



- Assuming a single hidden layer with tanh nonlinearity

# Vanilla RNN using Numpy

- Training an RNN means finding  $\theta$  (e.g.,  $W$  and  $b$ ) that give rise to a desired behavior quantified by a loss function

```
import numpy as np

class RNN:
    #...
    def __init__(self, len_h, len_x):
        self.h = np.zeros(len_h)
        self.W = np.random.randn(len_h, len_h+len_x)
        self.bias = np.random.randn(len_h)
        #...
    def step(self, x_t):
        activation = np.dot(self.W, np.hstack((self.h, x_t))) + self.bias
        self.h = np.tanh(activation)
        return self.h #could have returned g(self.h) for some function g

rnn = RNN(3, 4)
for _ in range(5):
    x_t = np.random.randn(4)
    h_t = rnn.step(x_t)
    print h_t
```

# Language Model (LM) Example

---

- Build a character-level language model
  - Give RNN a large text dataset
  - Model the probability of the next character given a sequence of previous characters
- Application: allows us to generate new text, can be used as a prior for classification tasks
- Note: This is a toy example

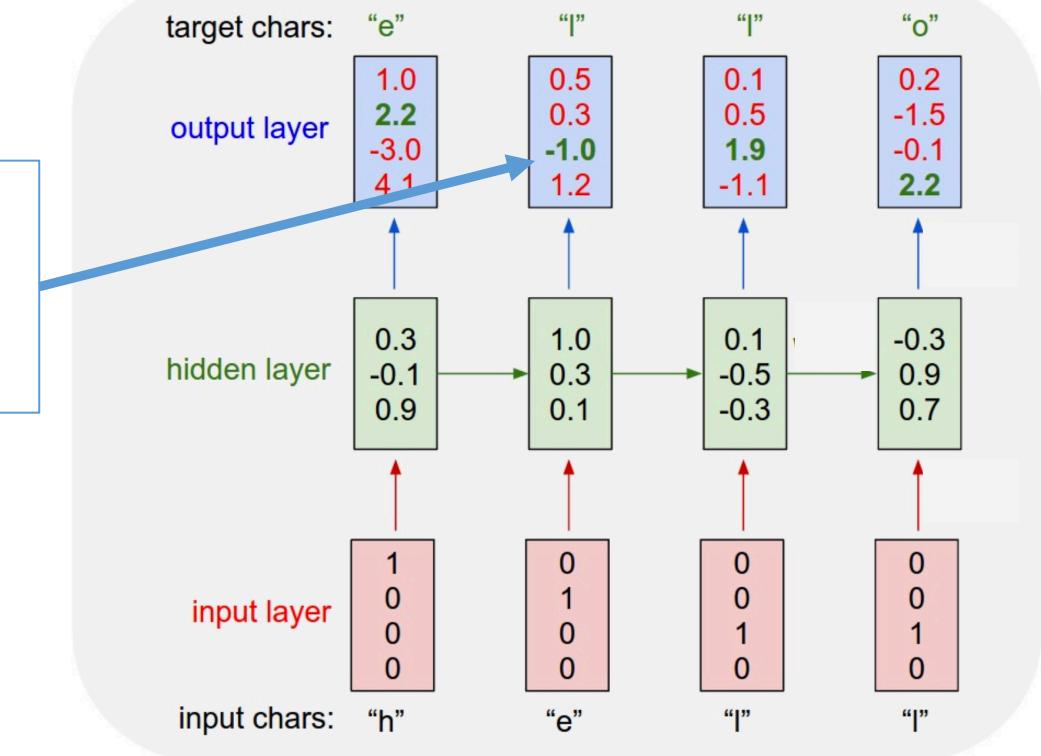
# LM Example: Data and Embedding

---

- Vocabulary: {h,e,l,o}
- Training sequence: {h,e,l,l,o}
  - Four training examples:
    - $P(e|h)$  should be high
    - $P(l|he)$  should be high
    - $P(l|hel)$  should be high
    - $P(o|hell)$  should be high
- Embedding:
  - Encode each character as a 4-dimensional vector

# LM Example: RNN

We want green numbers  
to be high and red  
numbers to be low



- Feed each vector into the RNN
- Output is a sequence of vectors
  - Let dimension be 4
  - Interpret as the confidence that the corresponding character is the next in sequence

<sup>1</sup>Figure: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

# LM Example: RNN

---

- Define loss as the cross entropy loss (i.e., multiclass logistic) on every output vector simultaneously
- When first time  $\{l\}$  is input, the next character should be  $\{l\}$
- When the second time  $\{l\}$  is input, the next character should be  $\{o\}$
- Hence, we need state/persistence, which the RNN hopefully captures

---

---

# Questions?

# Today's Outline

---

- Recurrent Neural Networks
- Long-Short Term Memory based RNNs
- Sequence to Sequence Learning and other RNN Applications

---

# Long-Short Term Memory RNNs

# Long Term vs Short Term (I)

---

- Why are we looking at RNN?
  - Hypothesis: enable the network to connect past information to the current
  - Can they persist both long and short range information?
    - It depends...

# Long Term vs Short Term (II)

---

- Consider a model predicting next word based on previous words
- Case A:
  - $R(\dots \text{ advanced prediction}) = \text{"models"}$
  - Here, the **immediate preceding words** are helpful

# Long Term vs Short Term (II)

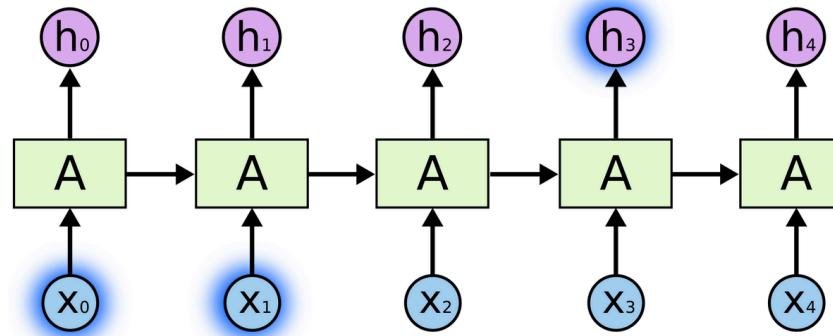
---

- Consider a model predicting next word based on previous words
- Case A:
  - $R(\dots \text{ advanced prediction}) = \text{"models"}$
  - Here, the **immediate preceding words** are helpful
- Case B:
  - $R(\text{"I went to UIC... I lived in [?]"}) = \text{"Chicago"}$
  - Here, more context is needed
    - Recent info suggests [?] is a place.
    - Need the context of UIC from further back

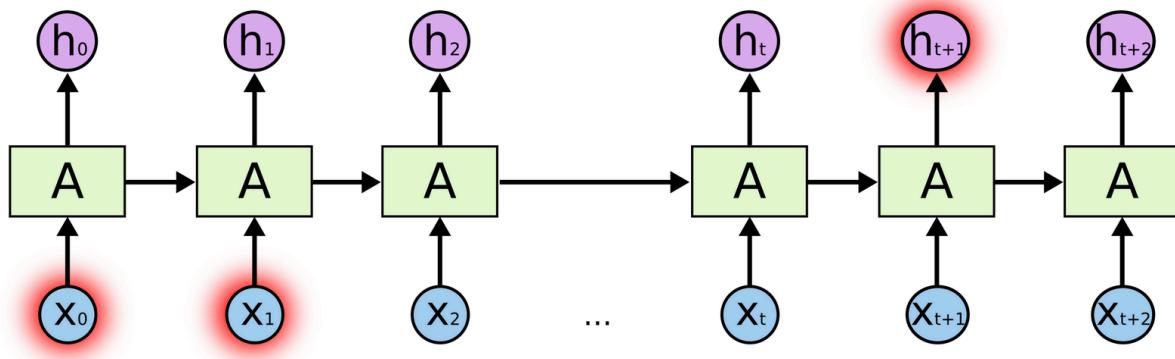
# Long Term vs Short Term (III)

- Consider a model predicting next word based on previous words

- Case A:



- Case B:



# A Special RNN: LSTM

---

- The gap between the relevant information and the point where it is needed can become unbounded
- **Empirical observation:** Vanilla RNNs seem unable to learn to connect long range information.
- This is a reason why we are looking at LSTMs (Long Short Term Memory Cells)

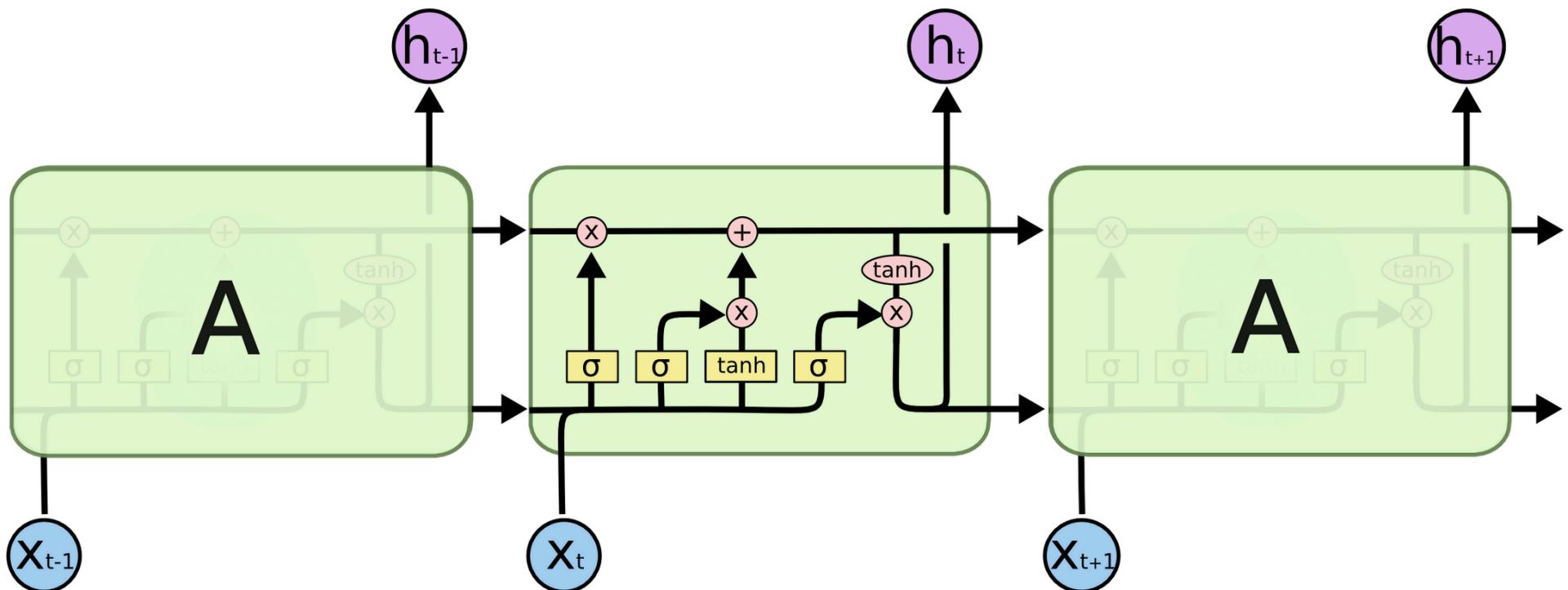
# LSTM: Long Short Term Memory based RNN

---

- Potentially capable of learning long-term dependencies
- Designed to avoid the long range issue that a **vanilla RNN** faces
  - How do they do that? We will address that now

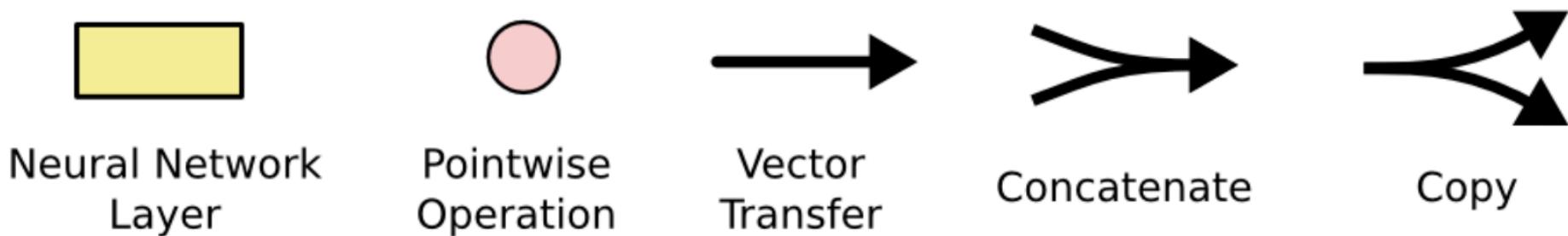
# LSTM: Block Level

- LSTM RNN have a similar structure to vanilla RNNs
- Only the repeating module is different
- Instead of a single neural layer, they have four



<sup>1</sup>Figure: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

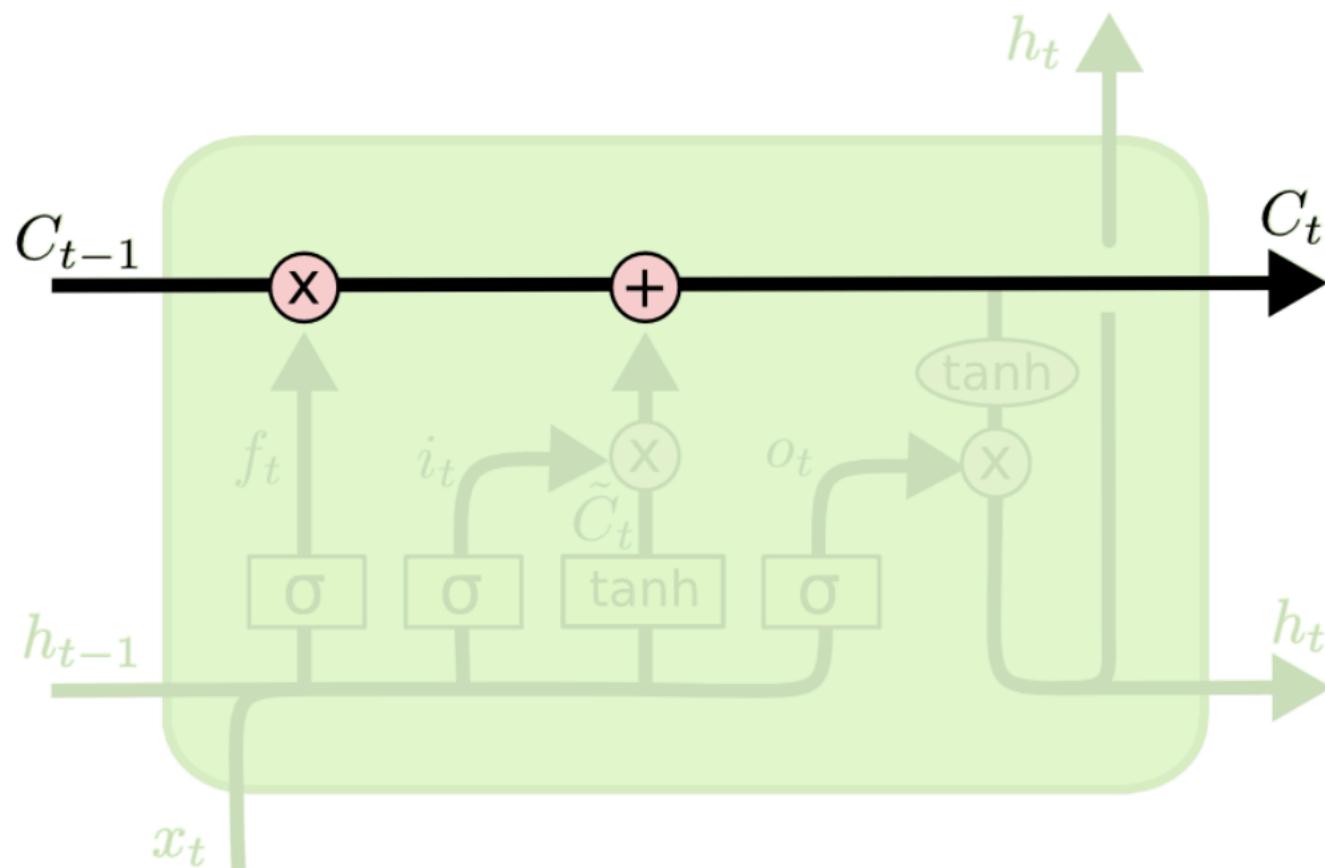
# LSTM: Recall Notation



- Dark arrow represents a vector, output from one layer and input to another
- Circle represents element-wise operations
  - Example: sum of two vectors
- Box represents a (fully connected) hidden layer

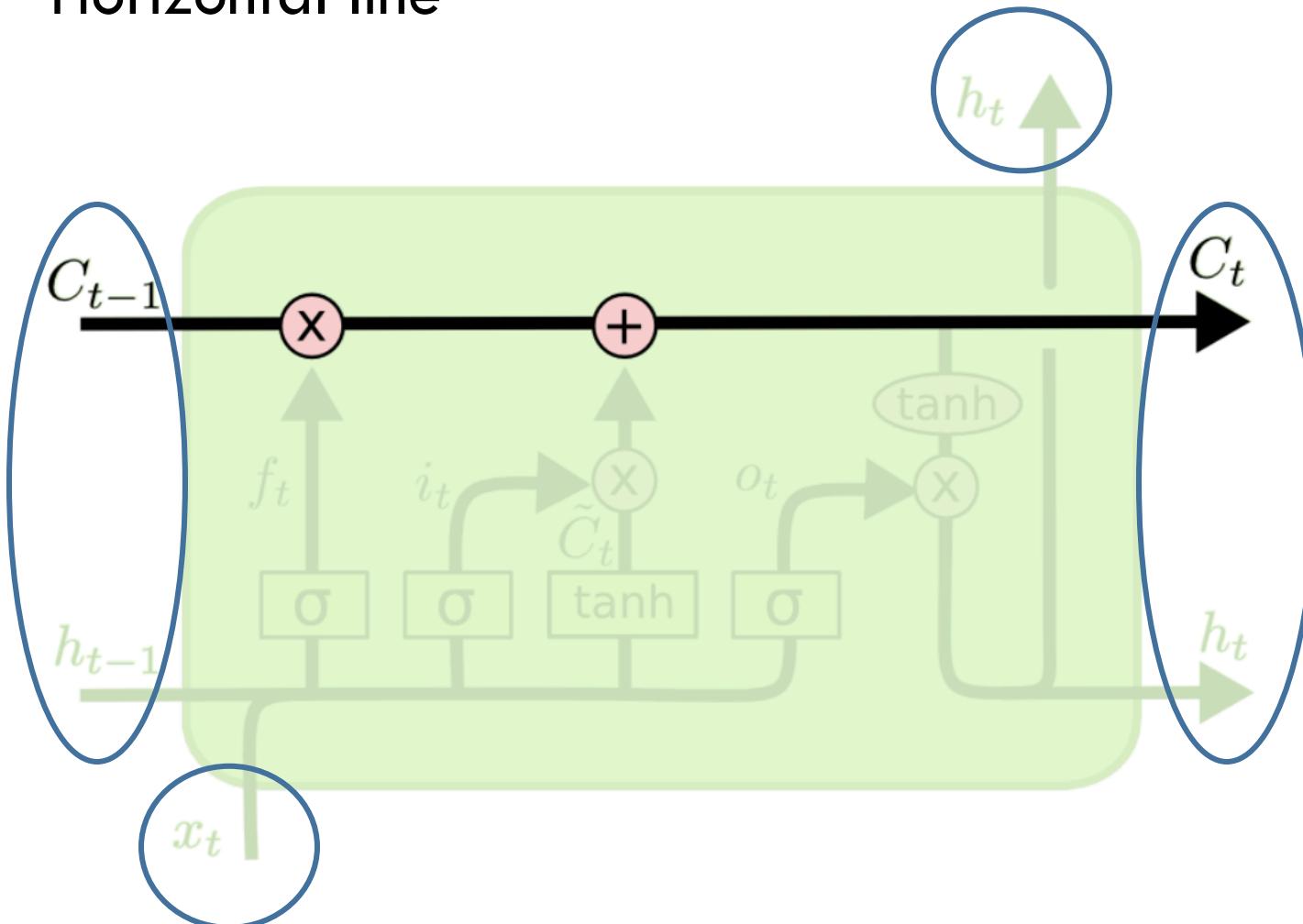
# LSTM: Cell State (I)

- There is a notion of **cell state**
  - Horizontal line



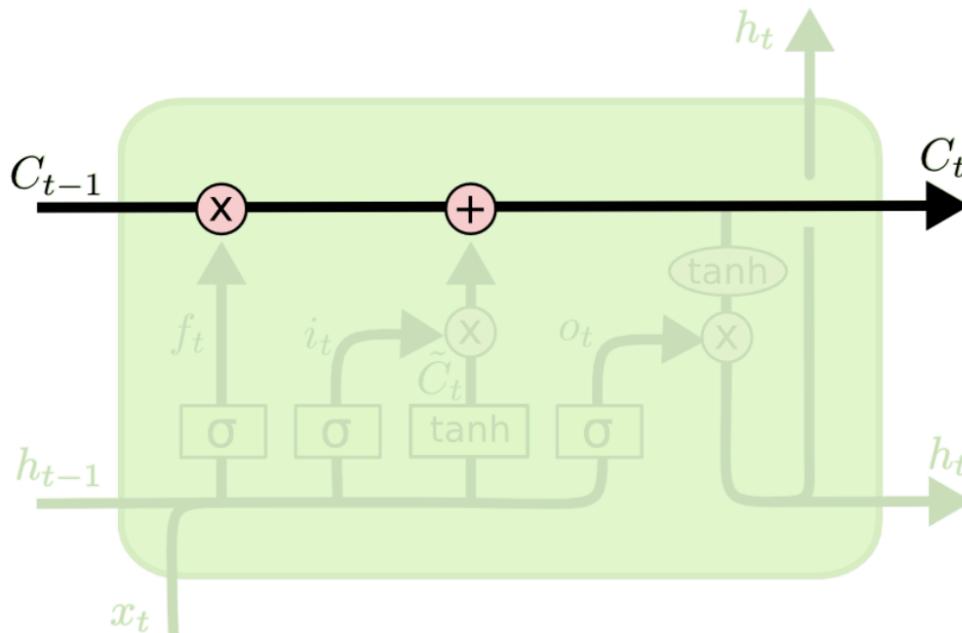
# LSTM: Cell State (I)

- There is a notion of **cell state**
  - Horizontal line



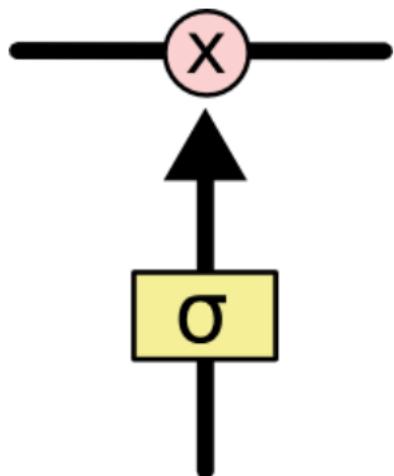
# LSTM: Cell State (II)

- **Cell state:**
  - Runs straight down the unrolled network
  - Minor interactions
  - Information could **flow** along it unchanged



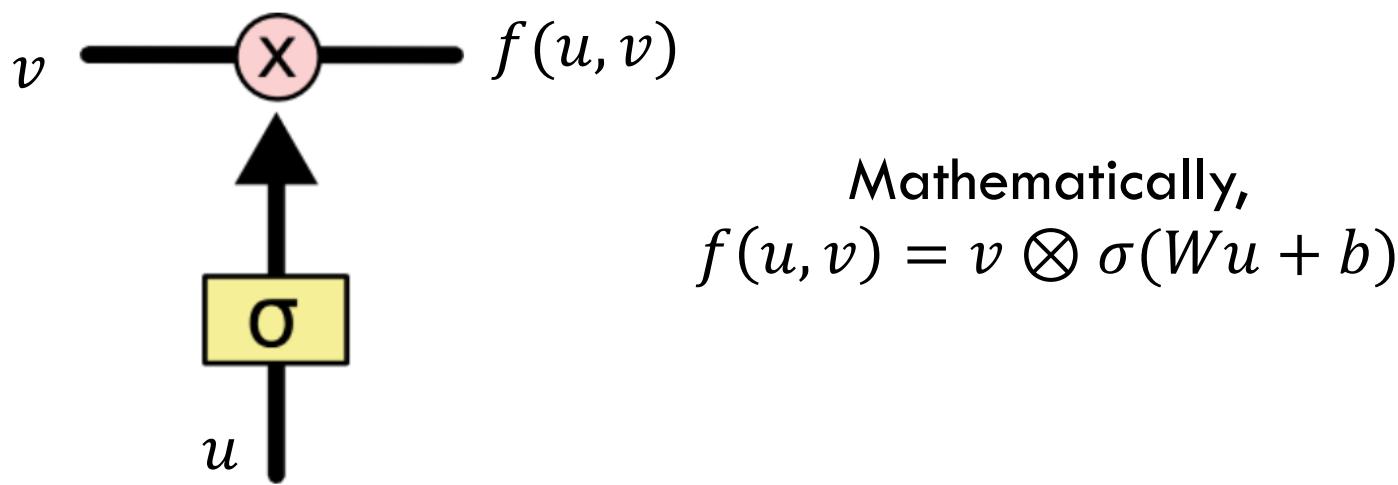
# LSTM: Gates (I)

- The LSTM can add or remove information to the cell state by regulating gates
- Gates optionally let information through
  - Made of a sigmoid NN layer and a pointwise multiplication



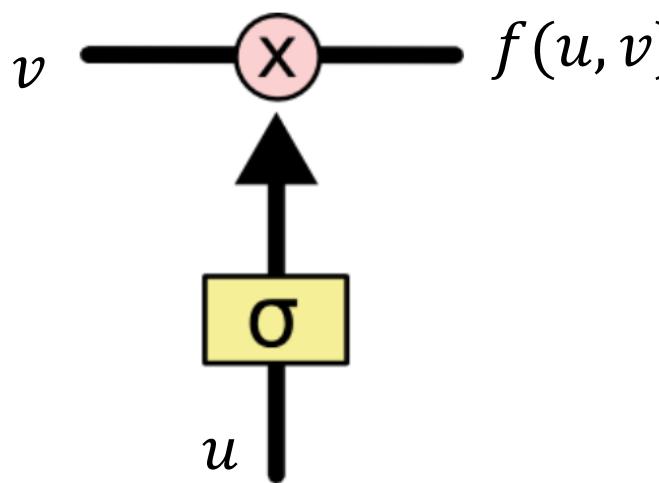
# LSTM: Gates (I)

- The LSTM can add or remove information to the cell state by regulating gates
- Gates optionally let information through
  - Made of a sigmoid NN layer and a pointwise multiplication



# LSTM: Gates (II)

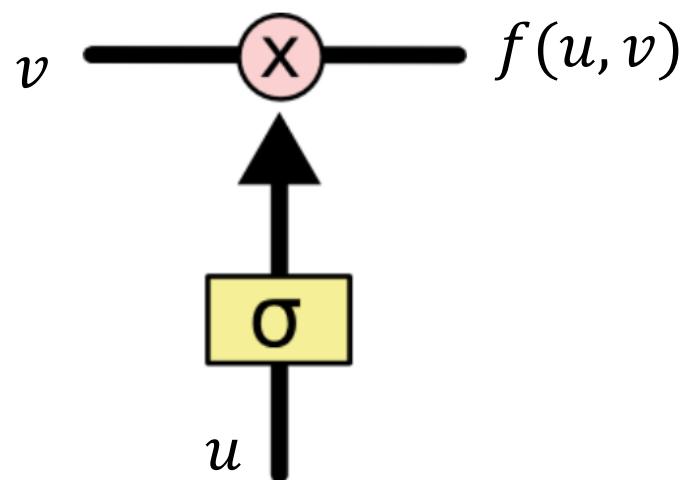
- Gate:
  - The sigmoid layer outputs numbers in (0,1)
  - Determines how much of each component to let through
    - 0 means ‘do not let input through’
    - 1 means ‘let input through’



Mathematically,  
$$f(u, v) = v \otimes \sigma(Wu + b)$$

# LSTM: Gates (III)

- LSTM has three gates to control the cell state



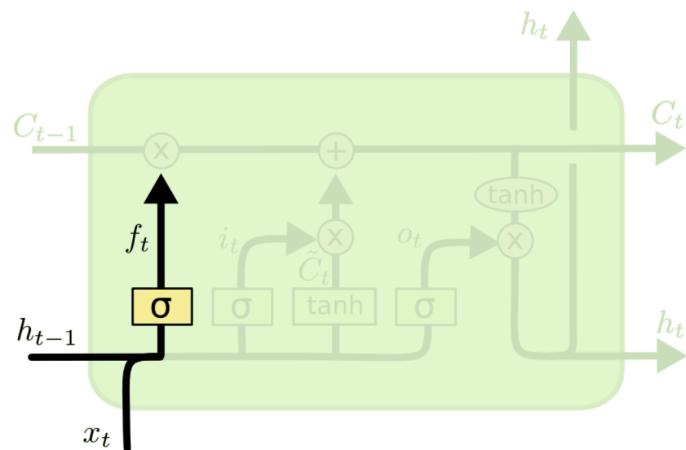
# LSTM: Forget Old Information

---

- First Step: what information to throw away from cell state
- Decided by forget gate layer
  - Input:  $h_{t-1}$  and  $x_t$
  - Output: a vector with entries in  $(0,1)$  corresponding to entries in  $C_{t-1}$ 
    - 1 corresponds to keep the input
    - 0 corresponds to get rid of the input

# LSTM: Forget Old Information

- Example: In the task of predicting the next word based on all previous ones
  - Cell state **may** include gender of current subject
    - This will be useful to predict/use correct pronouns (male: he, female: she)
  - When a new subject is observed
    - Need to forget the gender of old subject



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

<sup>1</sup>Figure: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

# LSTM: Remember New Information

---

- Next step: decide what new information we will store in cell state
- Two ingredients
  - Input gate layer
  - Tanh layer
- Input gate layer
  - Decides which values to update

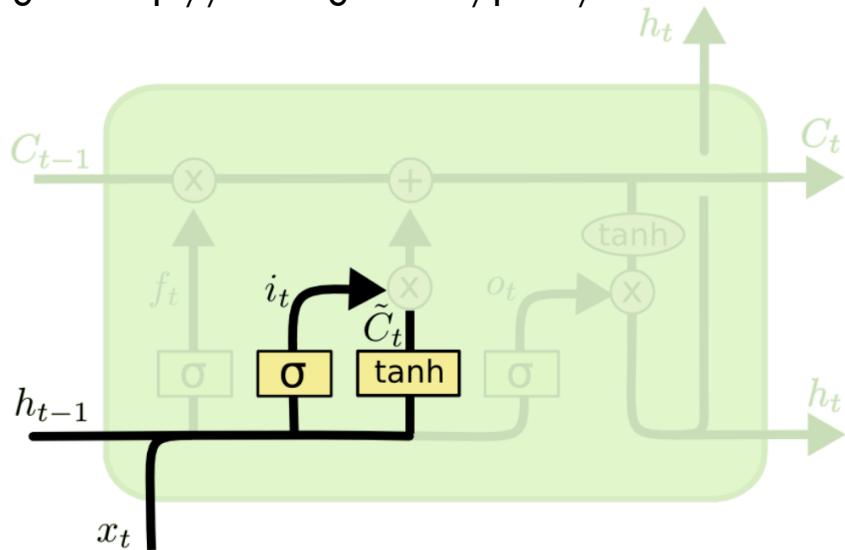
# LSTM: Remember New Information

---

- Next step: decide what new information we will store in cell state
- Two ingredients
  - Input gate layer
  - Tanh layer
- Input gate layer
  - Decides which values to update
- Tanh layer
  - Creates a vector of new candidate values  $\tilde{C}_t$  that can be added to the cell state

# LSTM: Remember New Information

<sup>1</sup>Figure: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

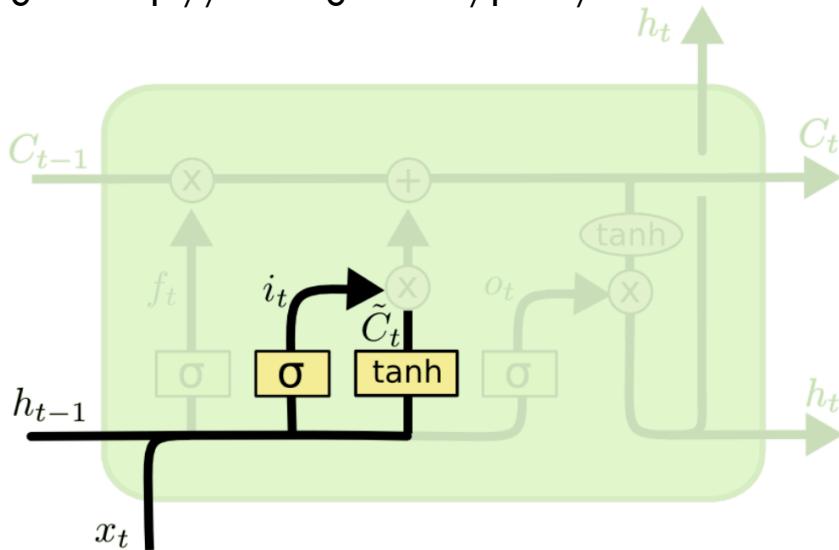


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- Input gate layer
  - Decides which values to update
- Tanh layer
  - Creates a vector of new candidate values  $\tilde{C}_t$  that can be added to the cell state

# LSTM: Remember New Information

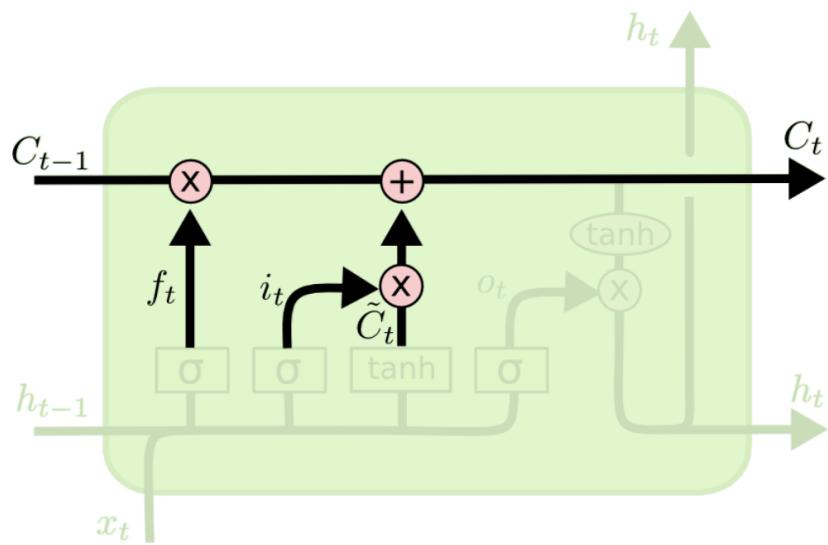
<sup>1</sup>Figure: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>



- Combine  $\tilde{C}_t$  with the output  $i_t$  of the input gate layer to get  $i_t \otimes \tilde{C}_t$
- In the language model example
  - Add the gender of the new subject to the cell state (this replaces the old one we are forgetting)

# LSTM: Forget and Remember

- Last step:
  - Modify the cell state



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

- $i_t \otimes \tilde{C}_t$  are the new values, scaled by how much we want to update each coordinate of cell state

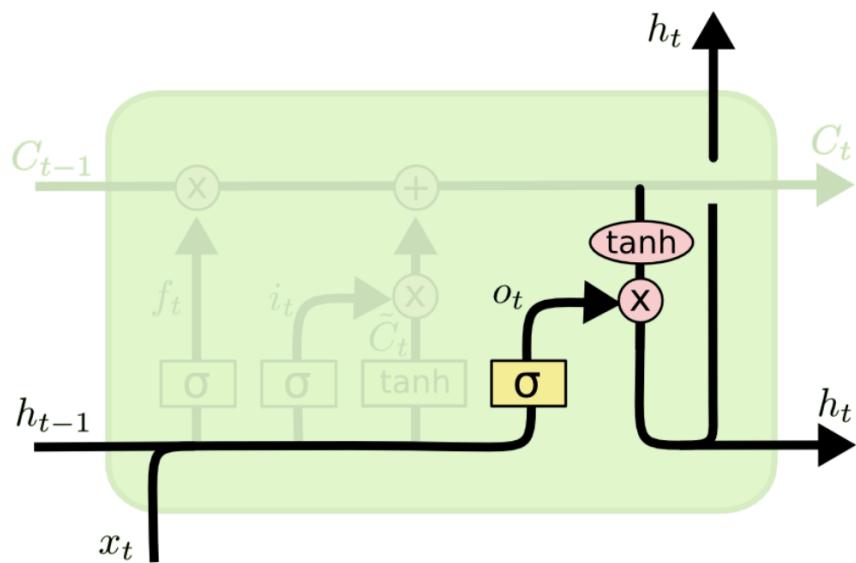
# LSTM: Output

---

- Output a filtered or transformed version of cell state
- Two stages:
  - Pass the cell state through a tanh layer
  - Scale it with a sigmoid layer output
    - The sigmoid layer decides what parts of the cell state we will output

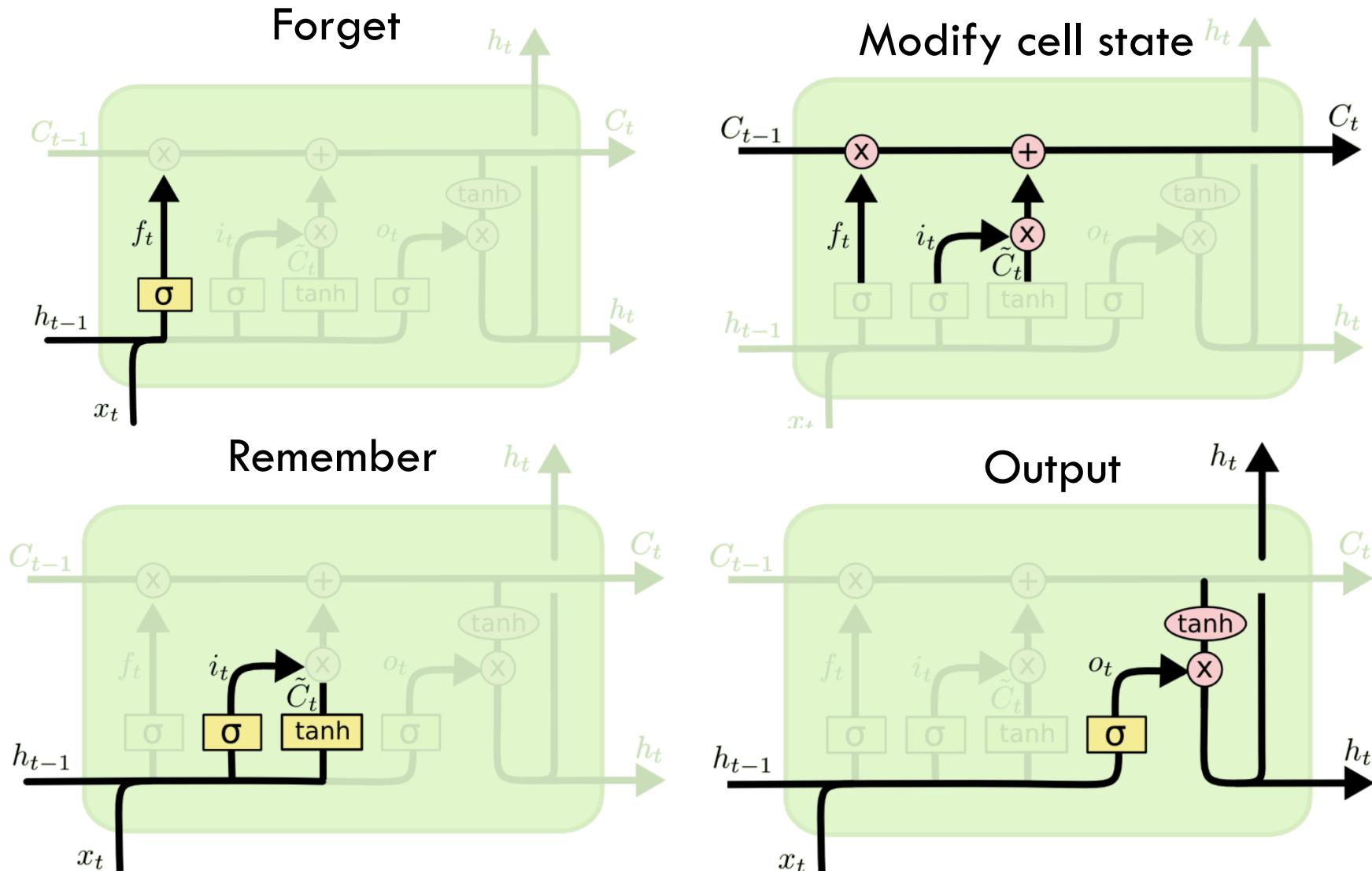
# LSTM: Output

- In the language model example
  - Since it just saw a new subject, it may output information related to actions (verbs)
    - Output whether the subject is singular or plural so verb can be modified appropriately



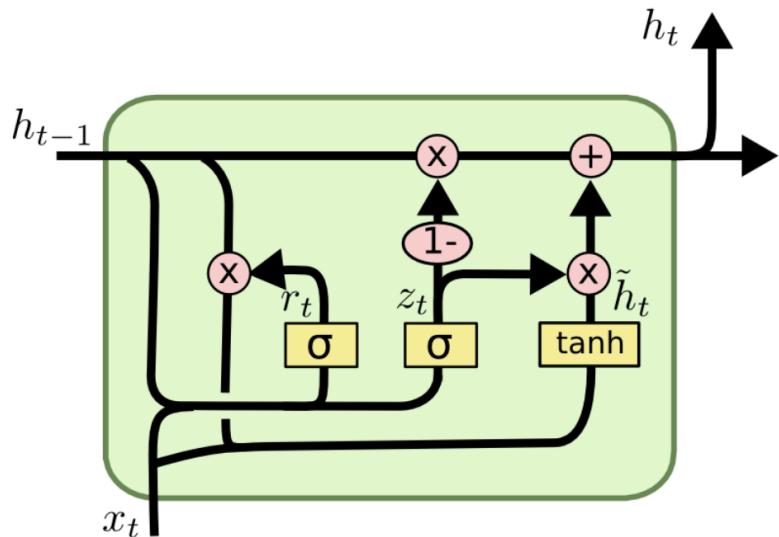
$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$
$$h_t = o_t \otimes \tanh (C_t)$$

# LSTM: Architecture Summary



# Other Variations in the Family of RNNs (I)

- The vanilla RNN and the LSTM we saw are just one of many variations
- Example: Gated Recurrent Unit (GRU)
  - Combines the forget and input gates
  - Merges the cell state and hidden state
  - ...



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

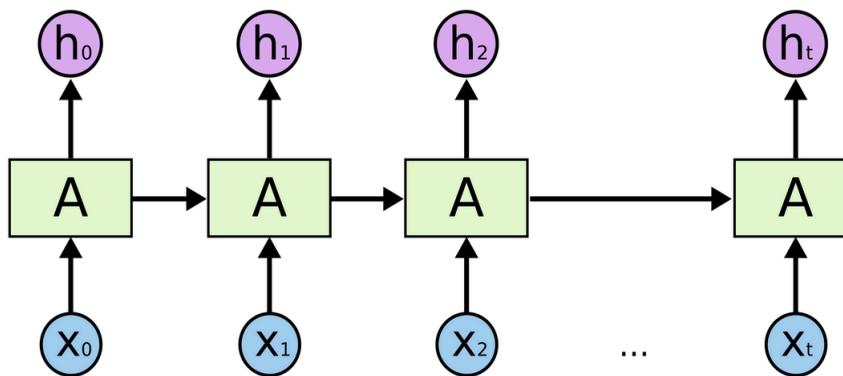
$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

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

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

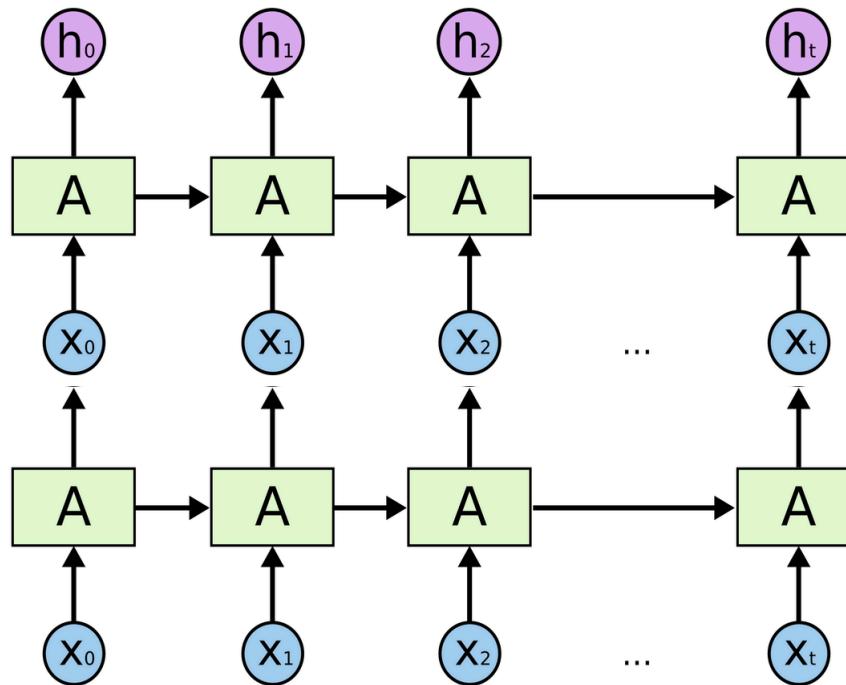
# Other Variations in the Family of RNNs (II)

- One can also go deep by stacking RNNs on top of each other



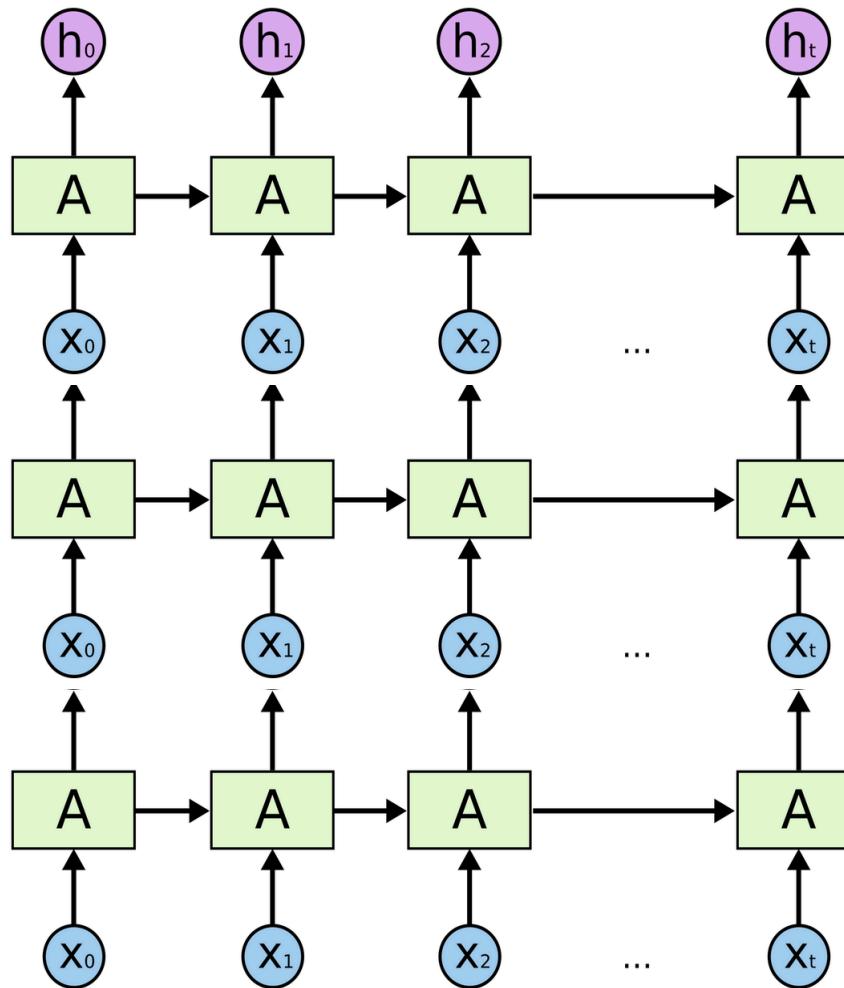
# Other Variations in the Family of RNNs (II)

- One can also go deep by stacking RNNs on top of each other



# Other Variations in the Family of RNNs (II)

- One can also go deep by stacking RNNs on top of each other



# Other Variations in the Family of RNNs (III)

---

- Extensive investigation has been done to see which variations are the best<sup>1,2</sup>
- As a practitioner, use popular architectures as starting points
- To recap, we are studying RNNs because we:
  - Want a notion of state/persistence to capture long term dependence
  - Want to process variable length sequences

<sup>1</sup>Reference: <http://arxiv.org/pdf/1503.04069.pdf>

<sup>2</sup>Reference: <http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf>

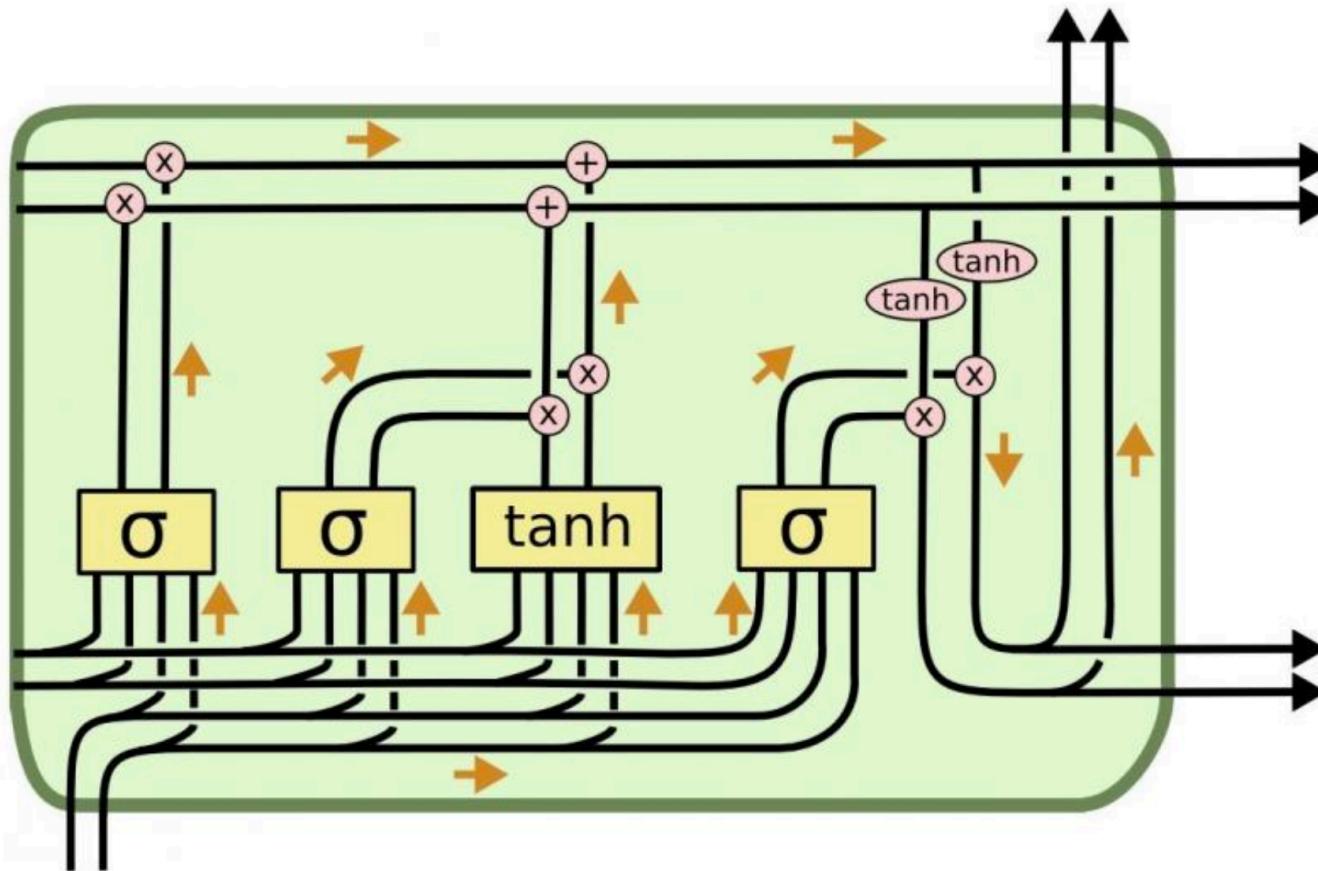
# Training RNNs

---

- These networks consist of differentiable operations
- Suitably define loss
- Run backpropagation to find best parameters

# LSTM Recap: Accounting for Dimensions

- Think of  $h_t$  as 2 dimensional and cell state as 2 dimensional



---

---

# Questions?

# Today's Outline

---

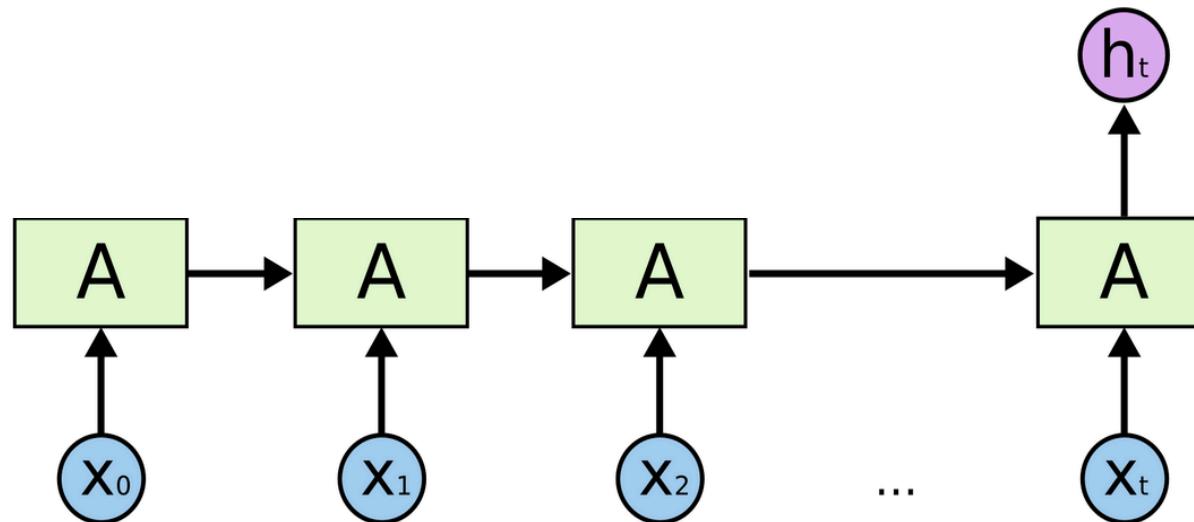
- Recurrent Neural Networks
- Long-Short Term Memory based RNNs
- Sequence to Sequence Learning and other RNN Applications

---

# Sequence to Sequence Learning and other RNN Applications

# Example I: Sentence Classification

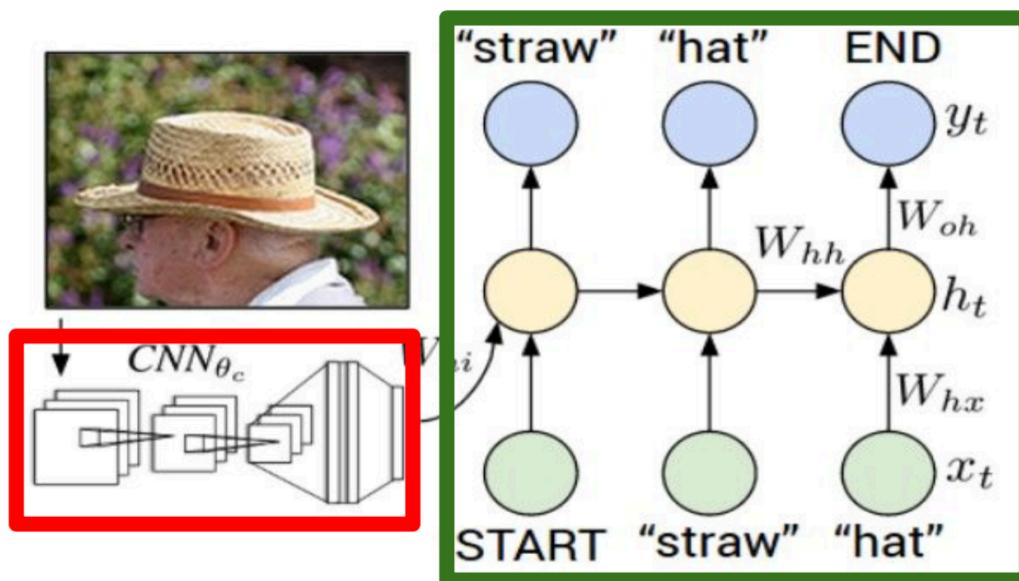
- We saw how to use a CNN for this task.
- Now, we can use an RNN as well:



# Example II: Image Captioning

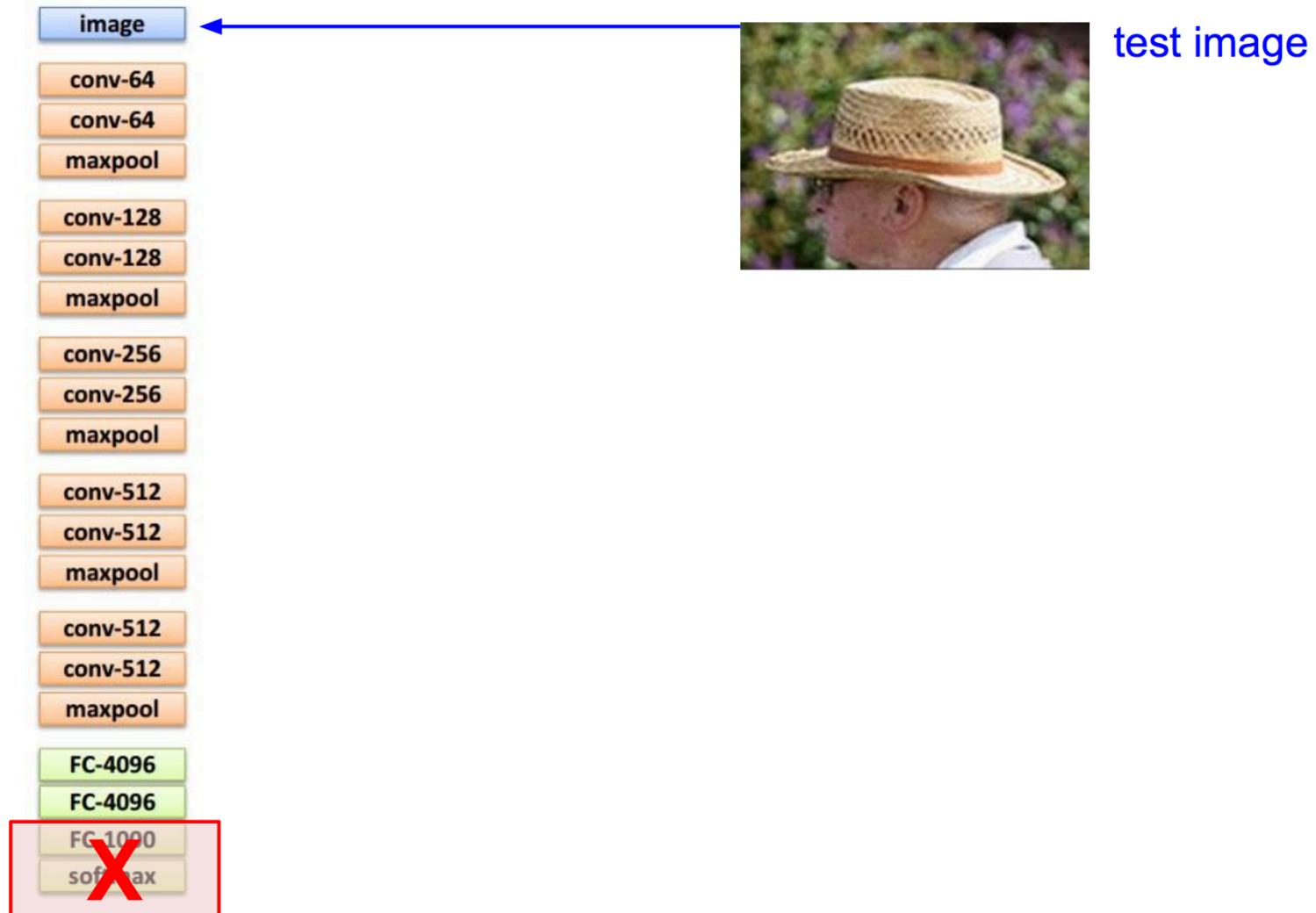
- Use CNNs and RNNs together to go from one data type to another

## Recurrent Neural Network

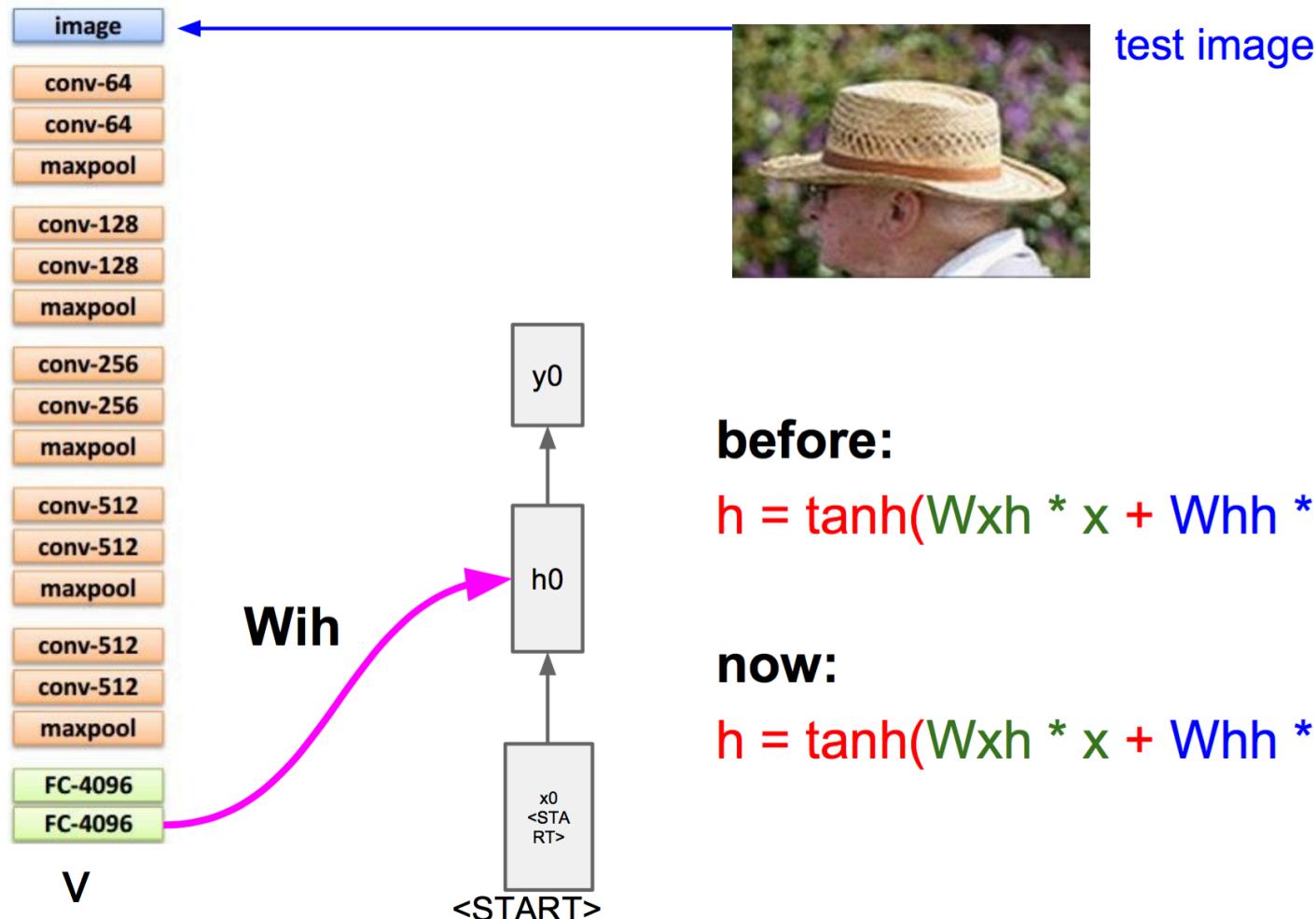


## Convolutional Neural Network

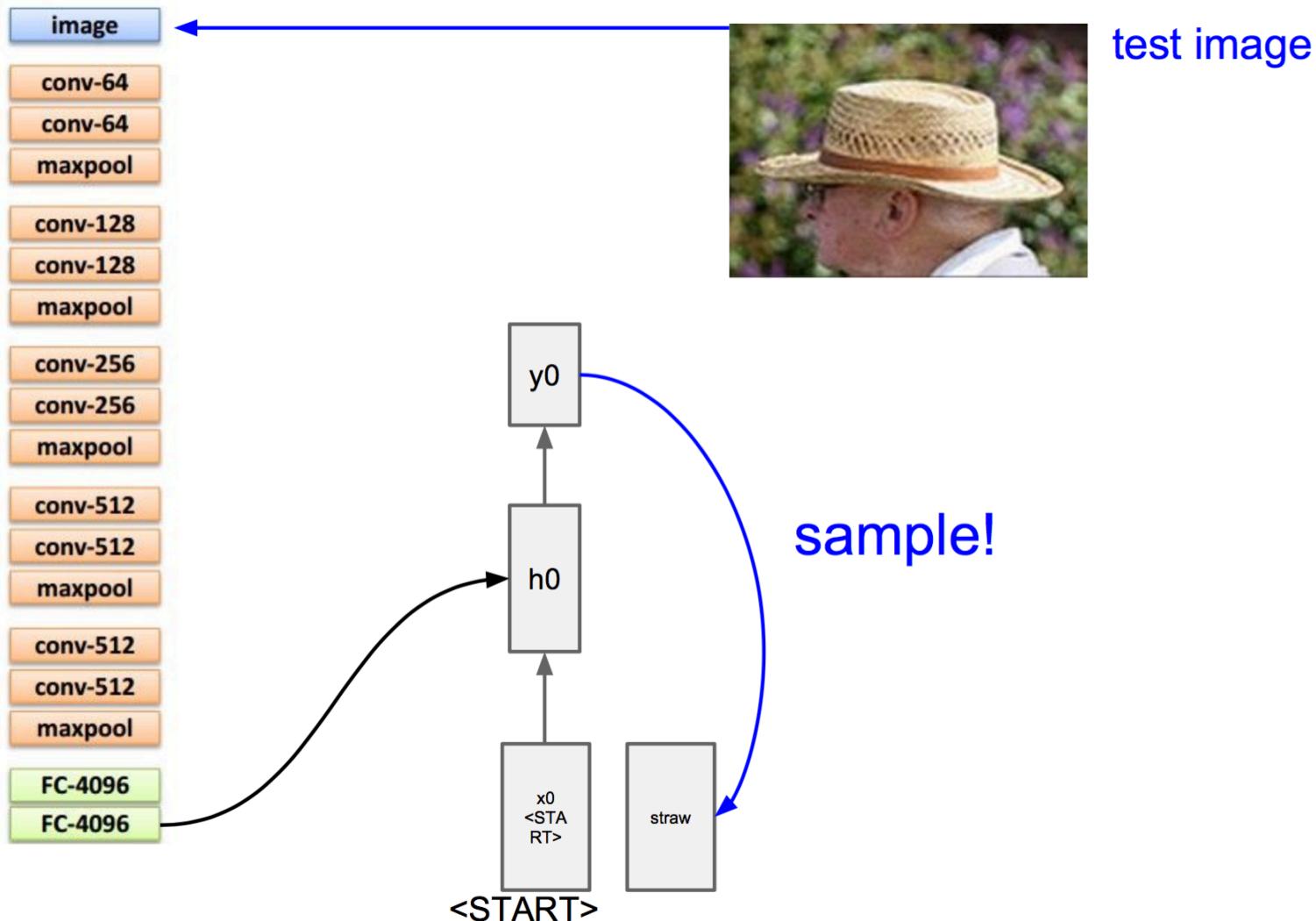
# Example II: Image Captioning



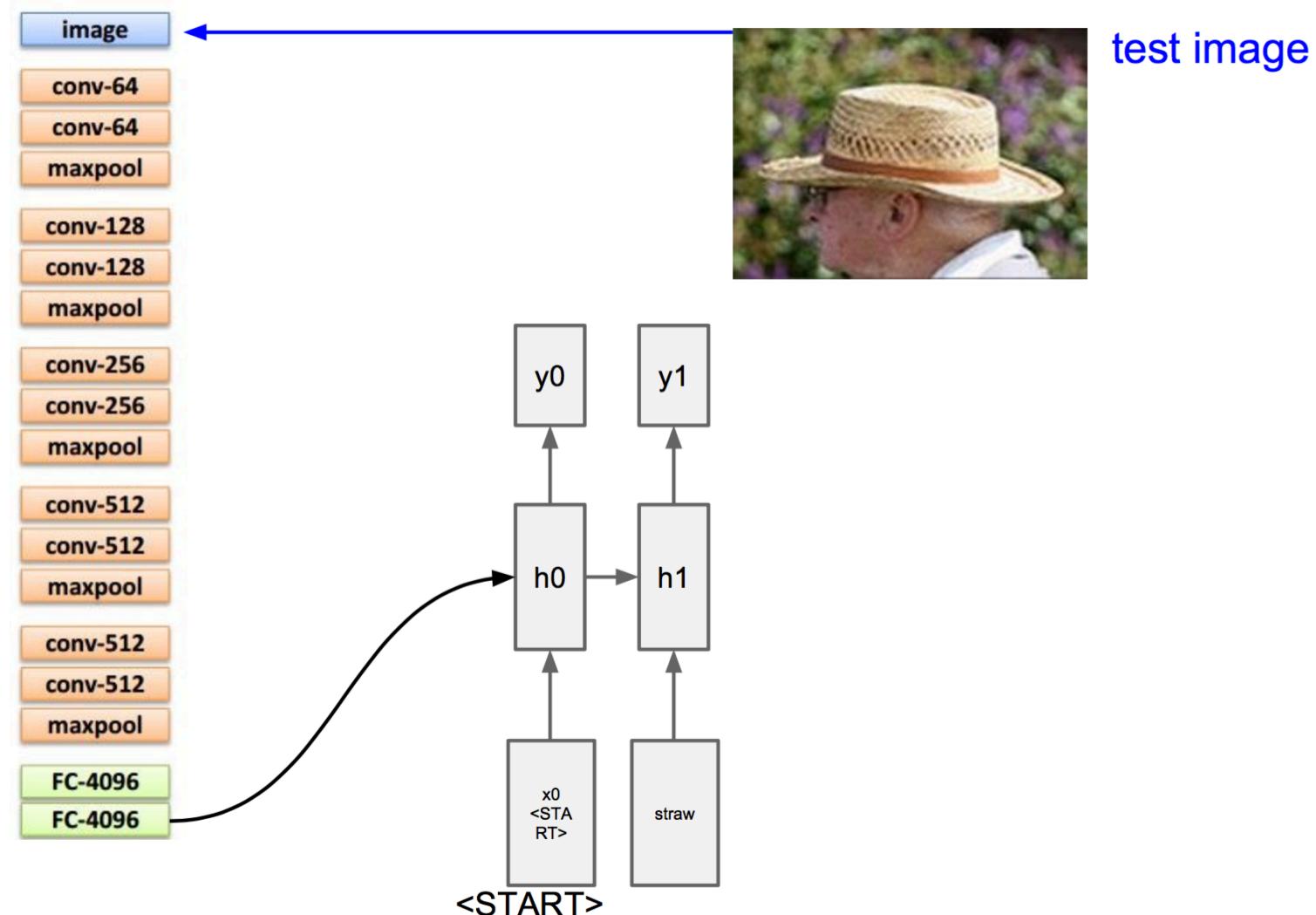
# Example II: Image Captioning



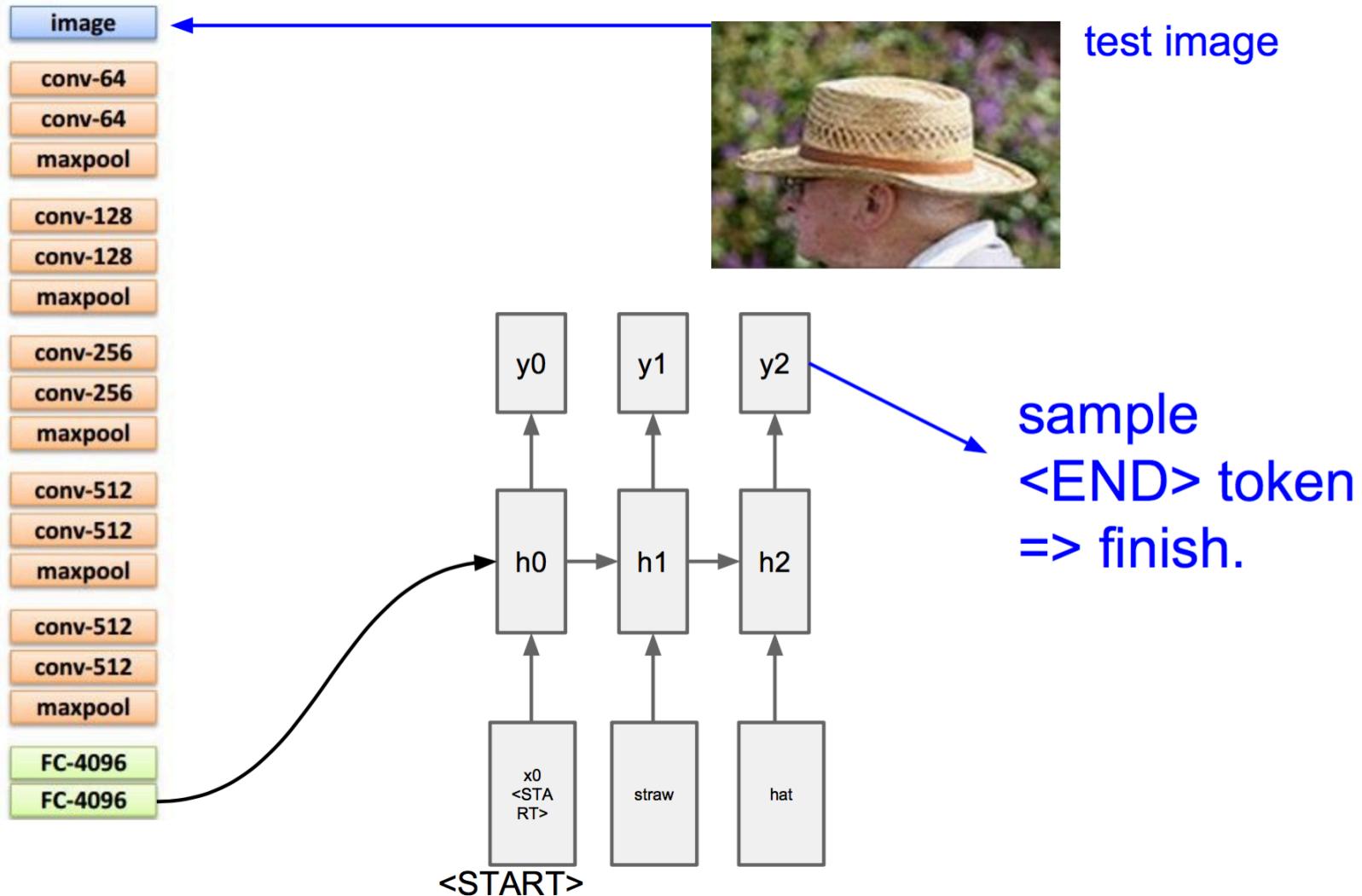
# Example II: Image Captioning



# Example II: Image Captioning



# Example II: Image Captioning



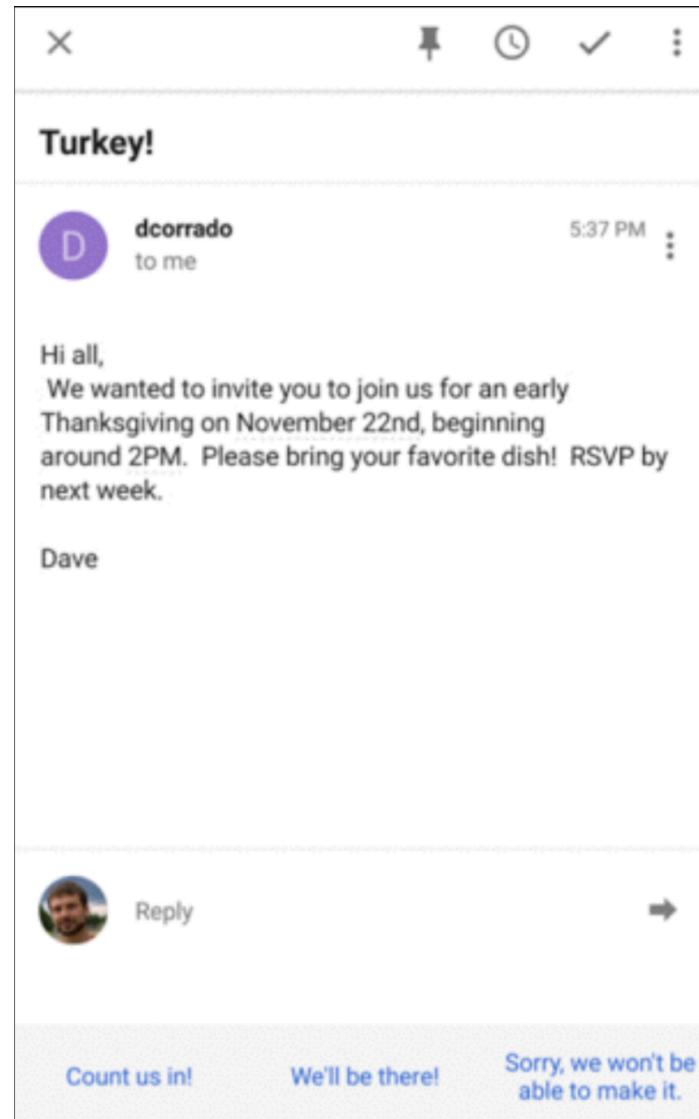
# Example III: Auto-Reply

---

- In this family of applications, we want mapping between variable length inputs to variable length outputs
- Other applications:
  - Translation
  - Summarizing
  - Speech transcription
  - Question answering

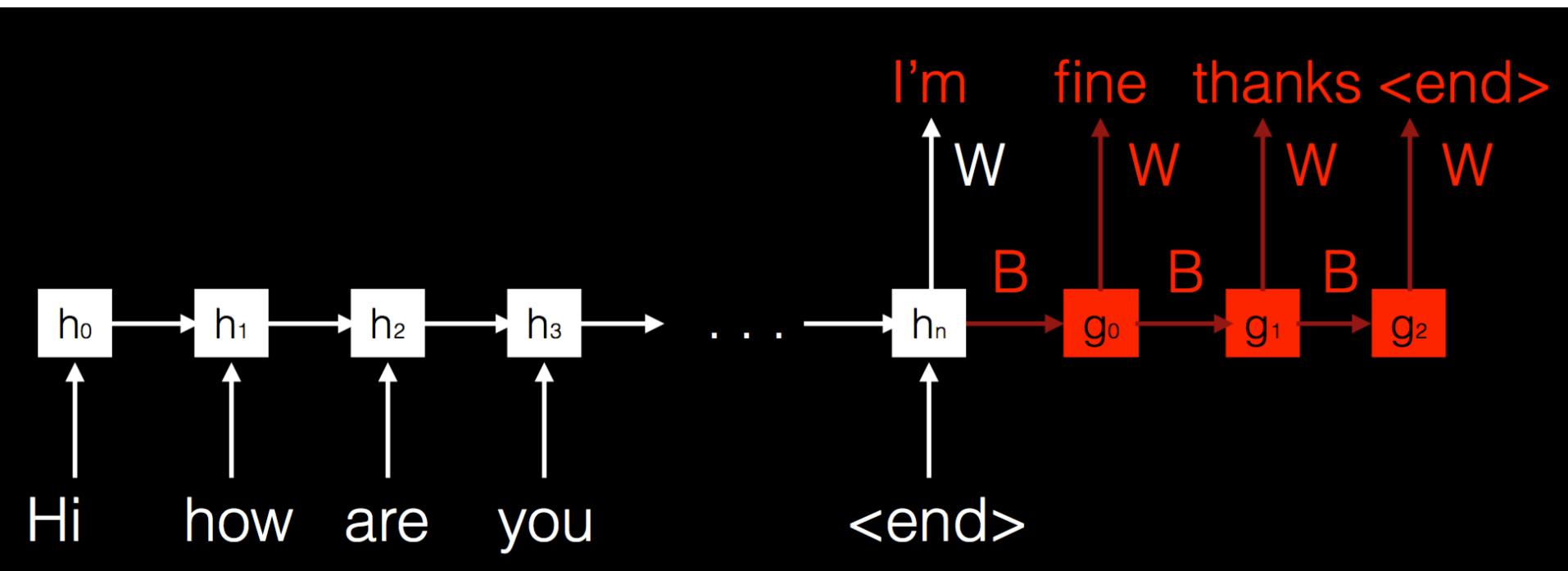
# Example III: Auto-Reply

- Auto-reply is a feature where the computer reads your email and responds appropriately



# Example III: Auto-Reply

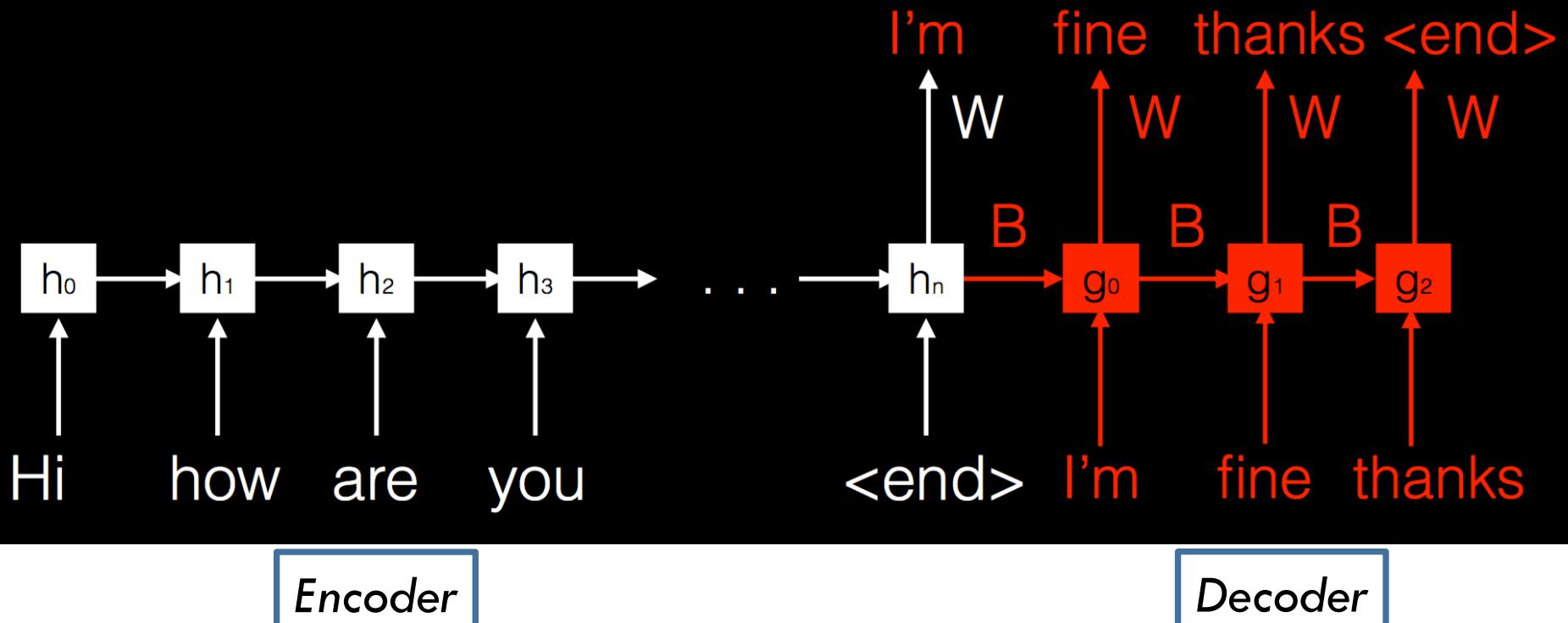
- First version



- Note that the number of classes in output is the number of words in the vocab!

# Example III: Auto-Reply

- Second version



- Feed back the true output at each stage during initial training

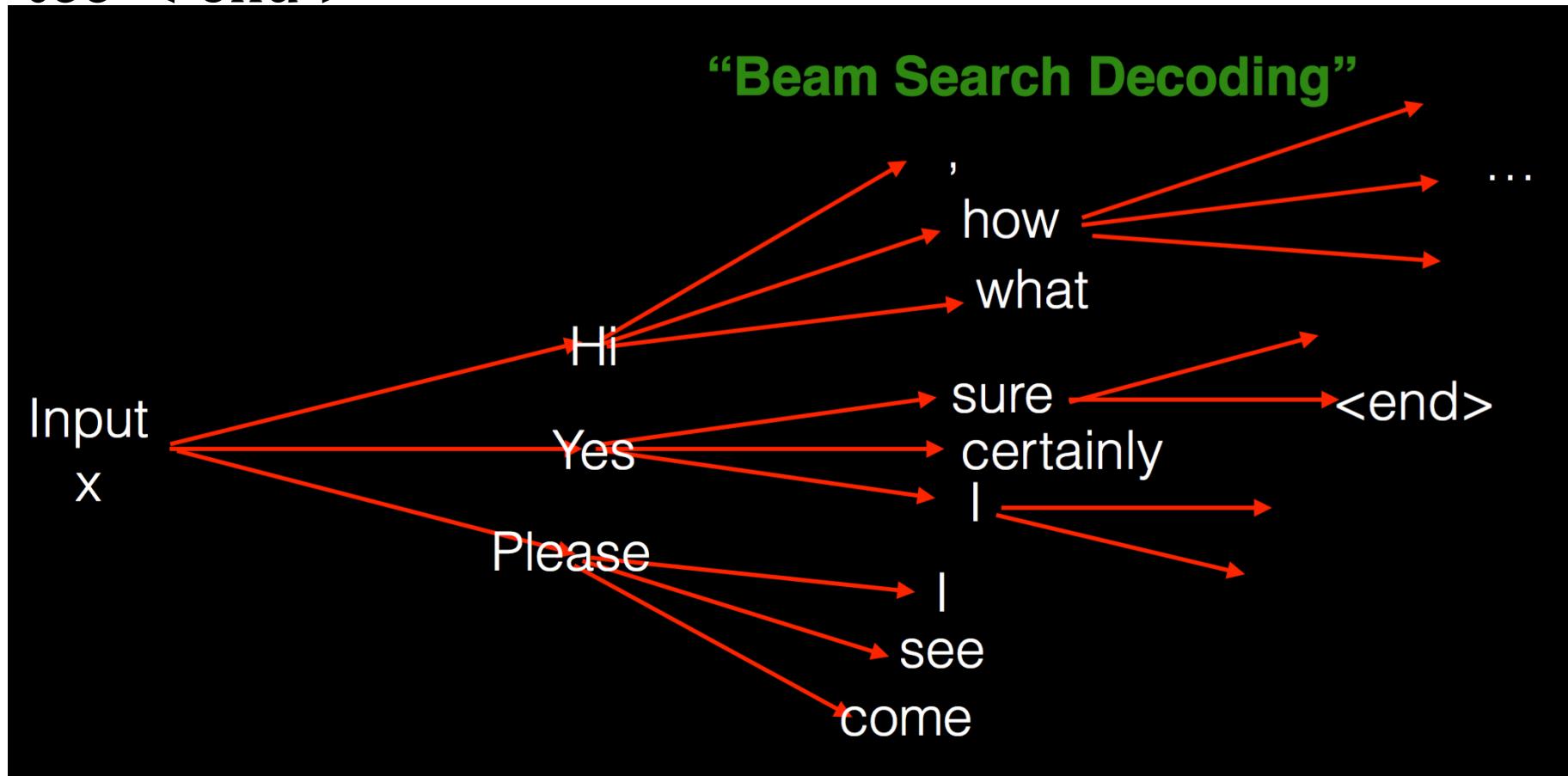
# Example III: Auto-Reply

---

- As we saw with image captioning example,
  - Given input sequence  $x$ , we first output  $y_0$  which has the highest probability
  - Given  $x$  and  $y_0$ , we output  $y_1$ , which has the highest probability
- 
- This is greedy
    - Does not correct for mistakes

# Example III: Auto-Reply

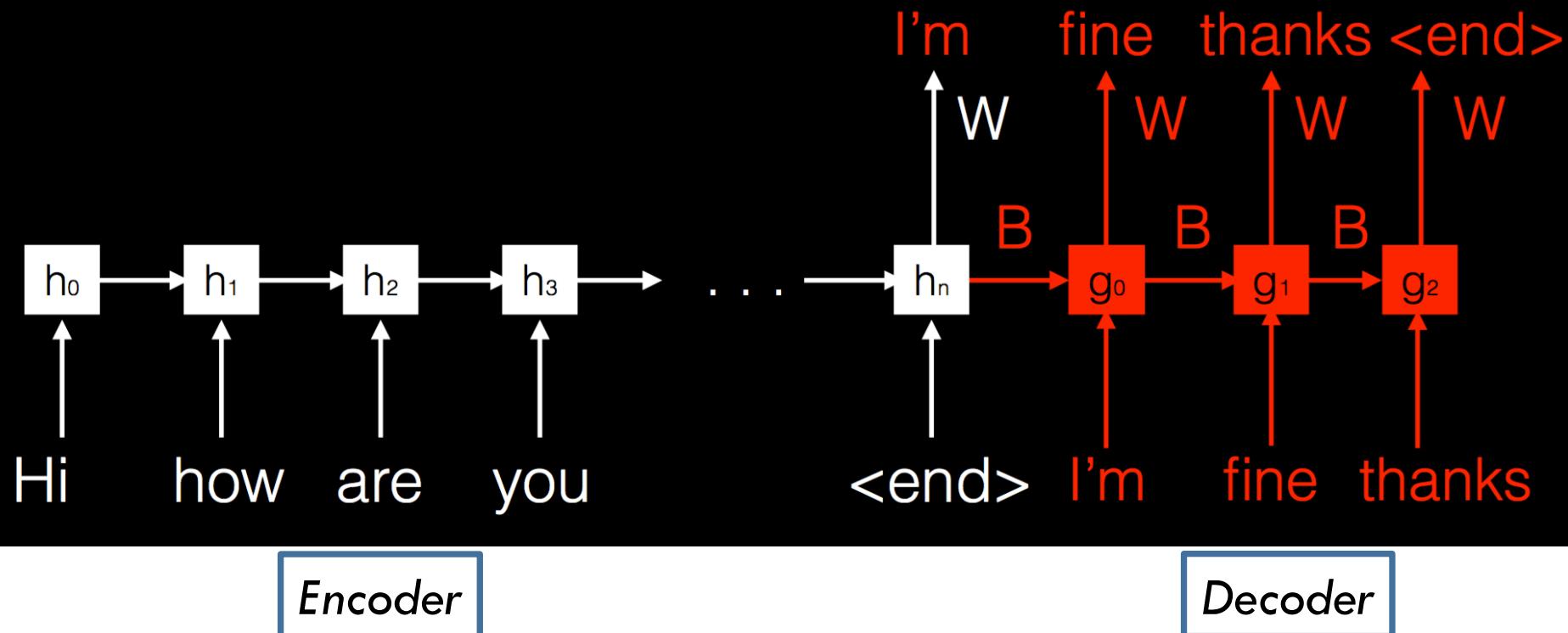
- Beam Search Decoding
- Retain  $k$  best candidate output sequences up to the time we see  $\langle \text{end} \rangle$



<sup>1</sup>Figure: Quoc Le, Google Brain

# Example III: Auto-Reply

- Issue with second version:  $h_n$  is the only link
  - In fact, it is a fixed length vector. Whereas input is variable length
- Can be fixed with an ‘attention’ layer



<sup>1</sup>Figure: Quoc Le, Google Brain

# Example IV: Speech Transcription

---

- Traditional pipeline has
  - Acoustic model  $P(\text{output}|\text{word})$
  - Language model  $P(\text{word})$
  - Feature engineering
  - ...
- Sequence to sequence learning can do ‘end-to-end’ without much feature engineering or blockwise modeling

# Example IV: Speech Transcription

- What we want is the following



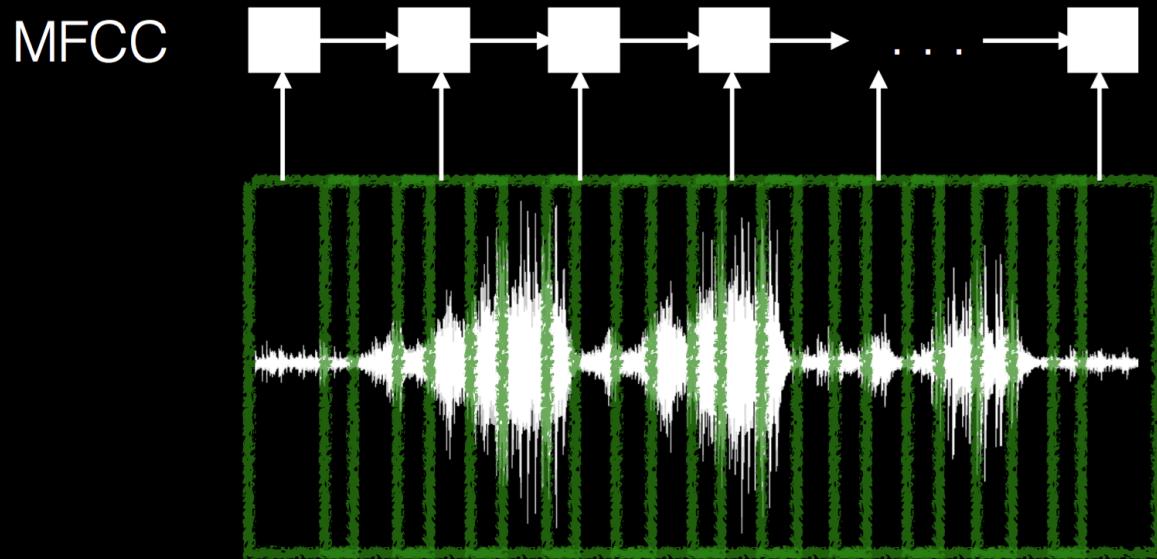
# Example IV: Speech Transcription

- Step 1: Get some fixed length vectors



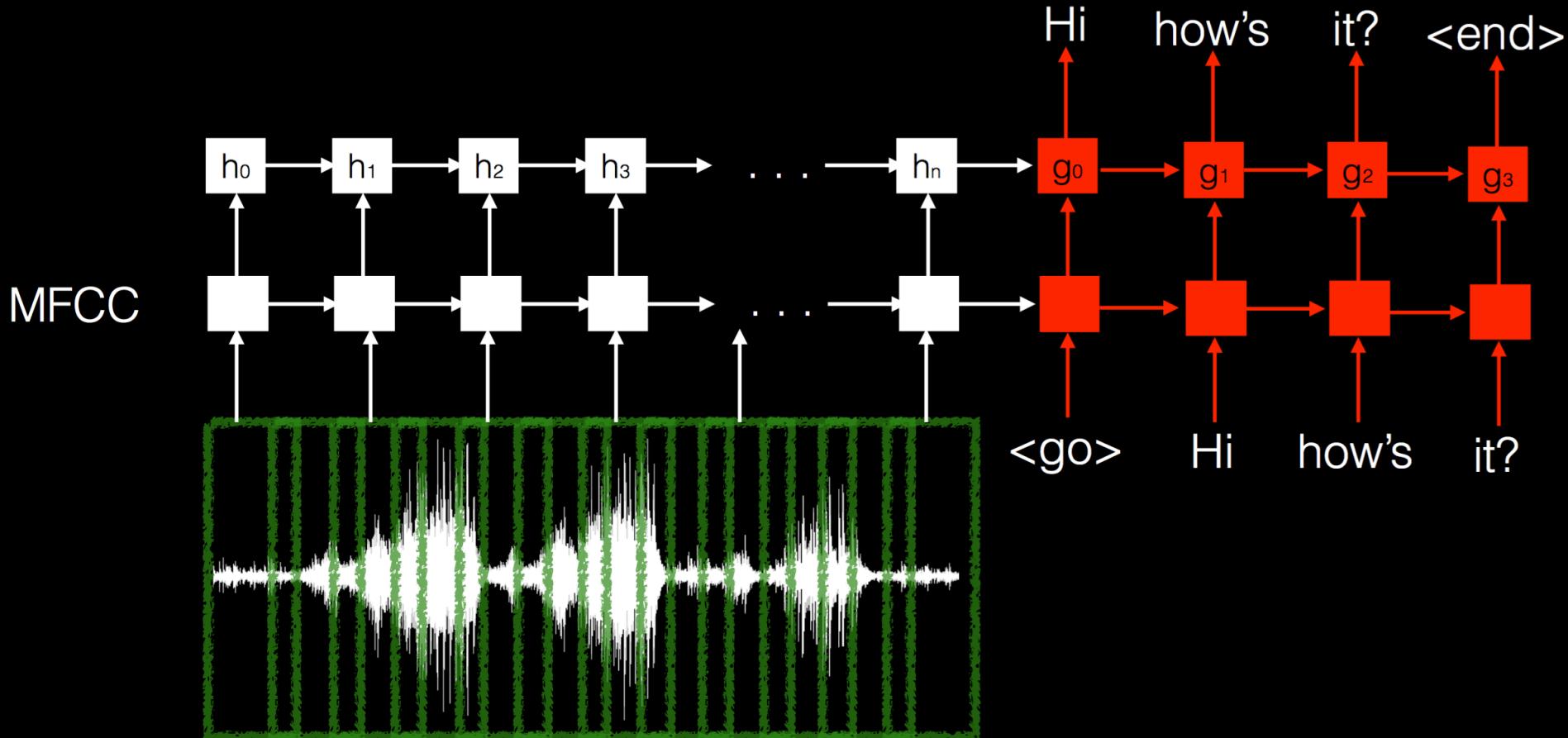
# Example IV: Speech Transcription

- Step 2: Pass through an encoder



# Example IV: Speech Transcription

- Step 3: Decode
- This is only a high level idea. Many many challenges.



---

---

# Questions?

# Summary

---

- We motivated when RNNs can be used
- Understood the internal working of RNNs (incl. LSTMs)
- Looked at some details for of ‘sequence to sequence’ applications.
  - These significantly extend beyond classification

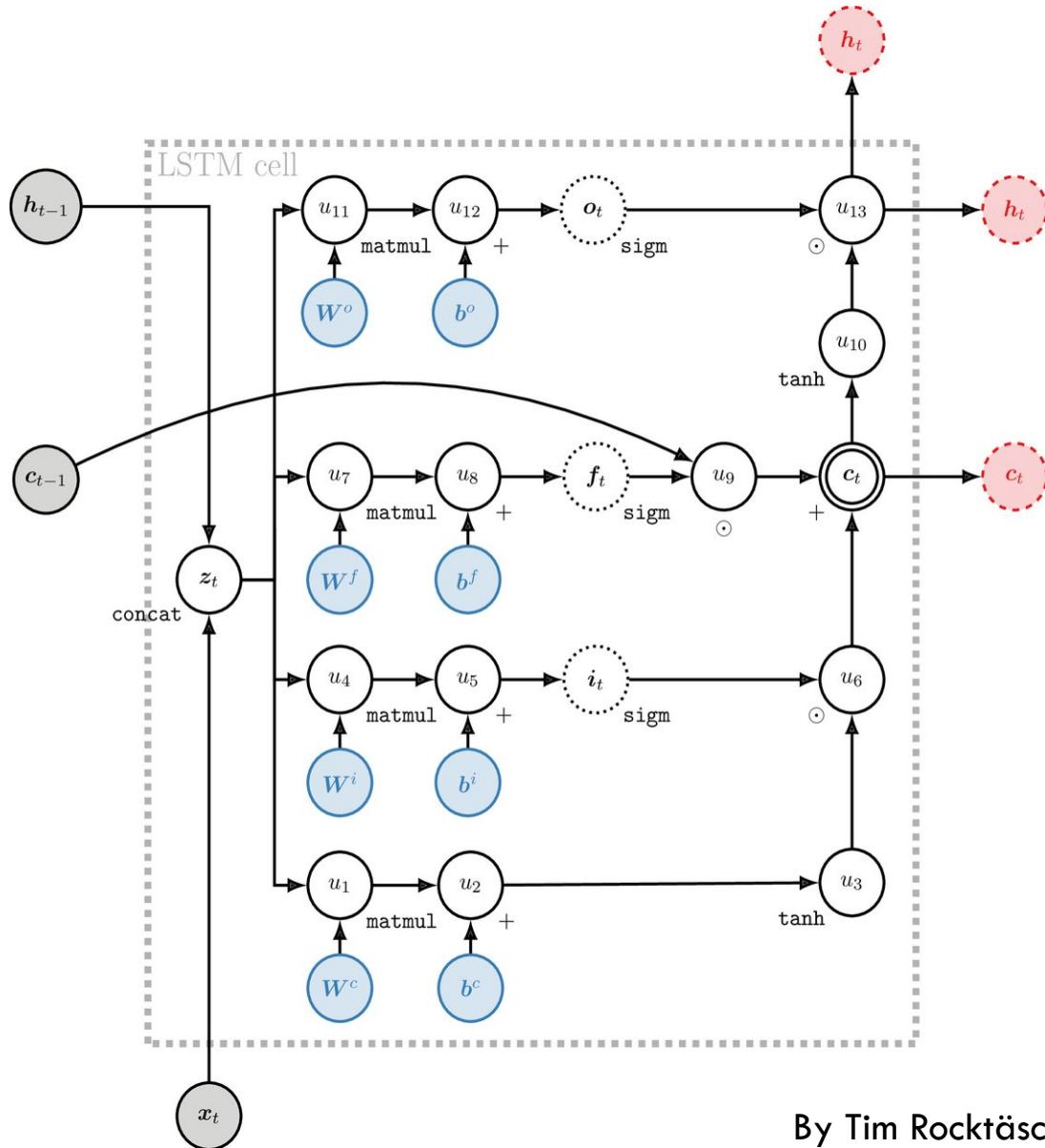
# Appendix

# Sample Exam Questions

---

- What is the need for an RNN architecture?
- What shortcoming of vanilla RNNs does an LSTM RNN attempt to fix?
- Describe how sentence classification can be done with both an RNN and a CNN.

# Yet Another Diagram of LSTM



# Understanding LSTM: LSTMVis

- A visual tool to see which cell states do what



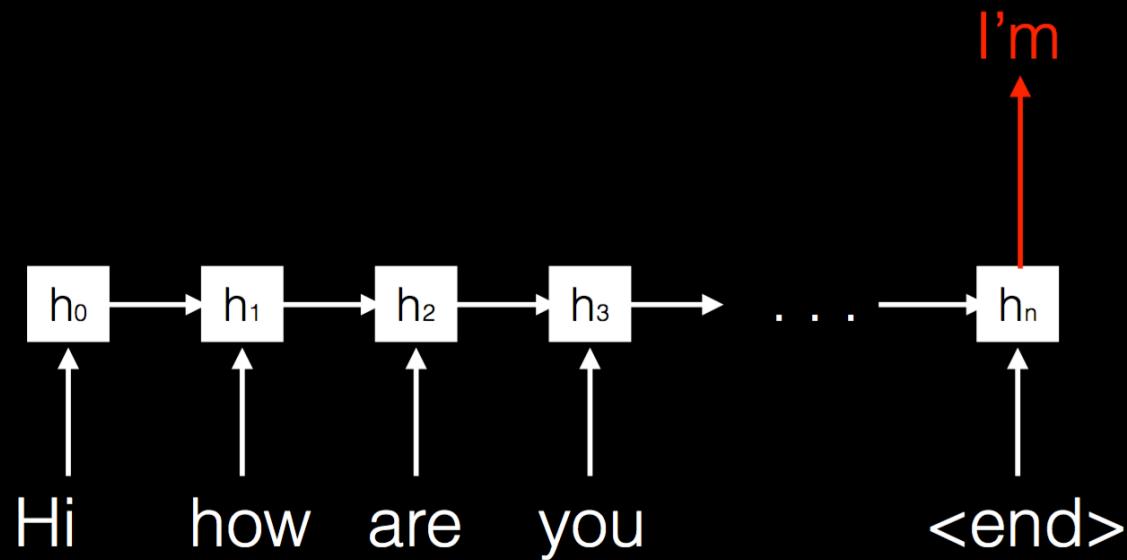
# Tensorflow Seq2Seq/RNN Models

---

- For sequence to sequence modeling nuances, especially about how to deal with variable length training input and output data, see  
<https://www.tensorflow.org/tutorials/seq2seq/>

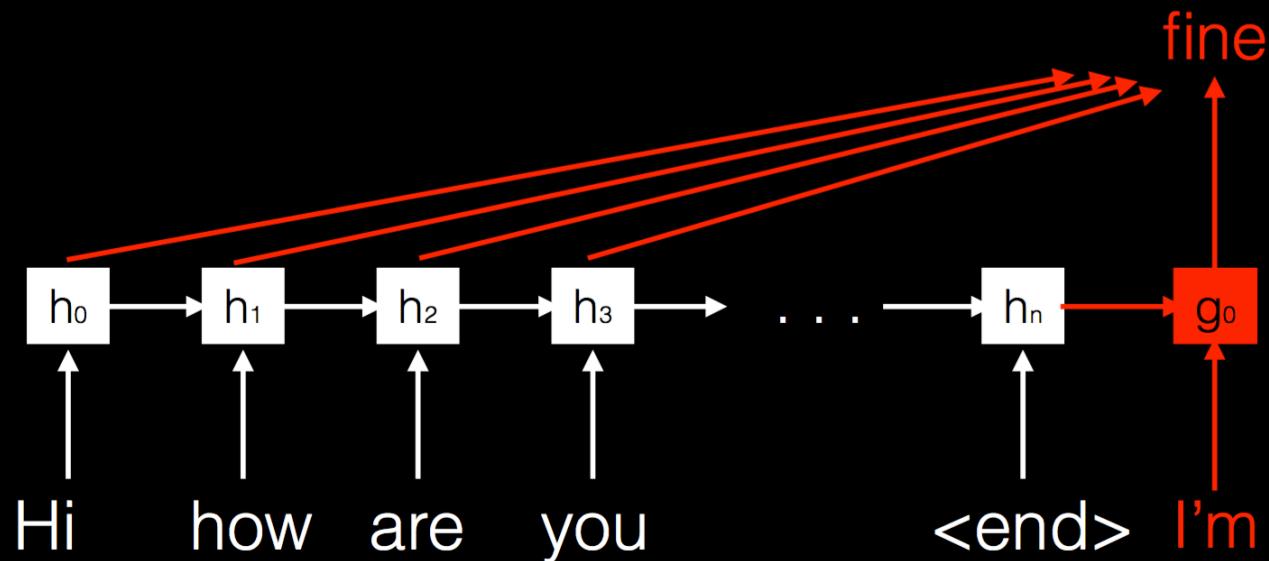
# Example III (Extension): Auto-Reply

- Third version: Attention Mechanism
- Ideally output could consider ‘attention’ to parts of history



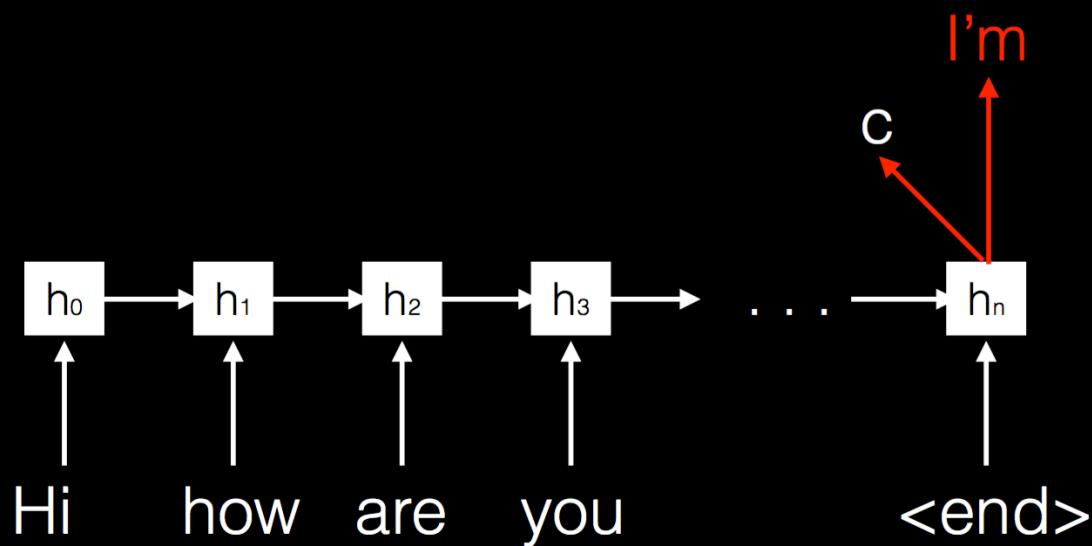
# Example III (Extension): Auto-Reply

- Could look at every state in the past



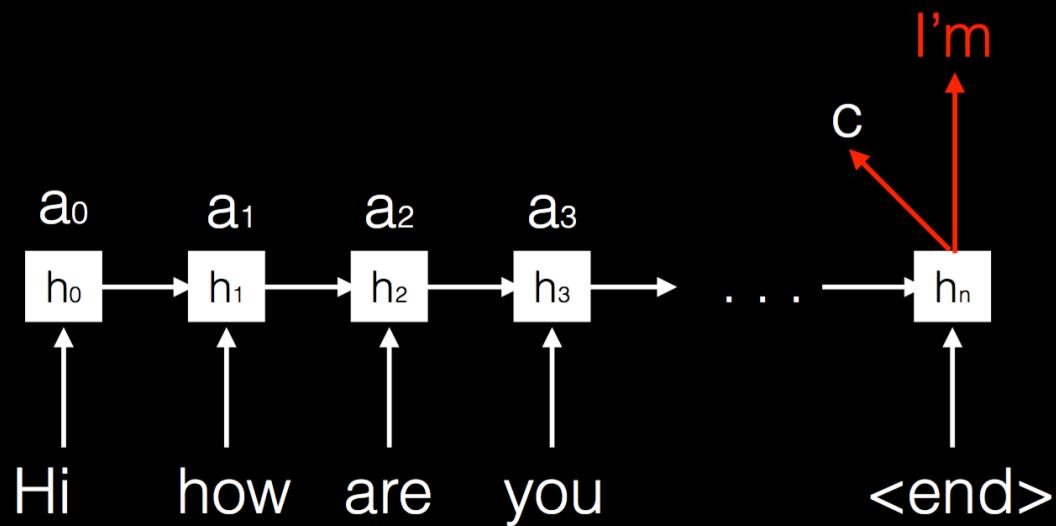
# Example III (Extension): Auto-Reply

- So instead of returning a word, output the current state



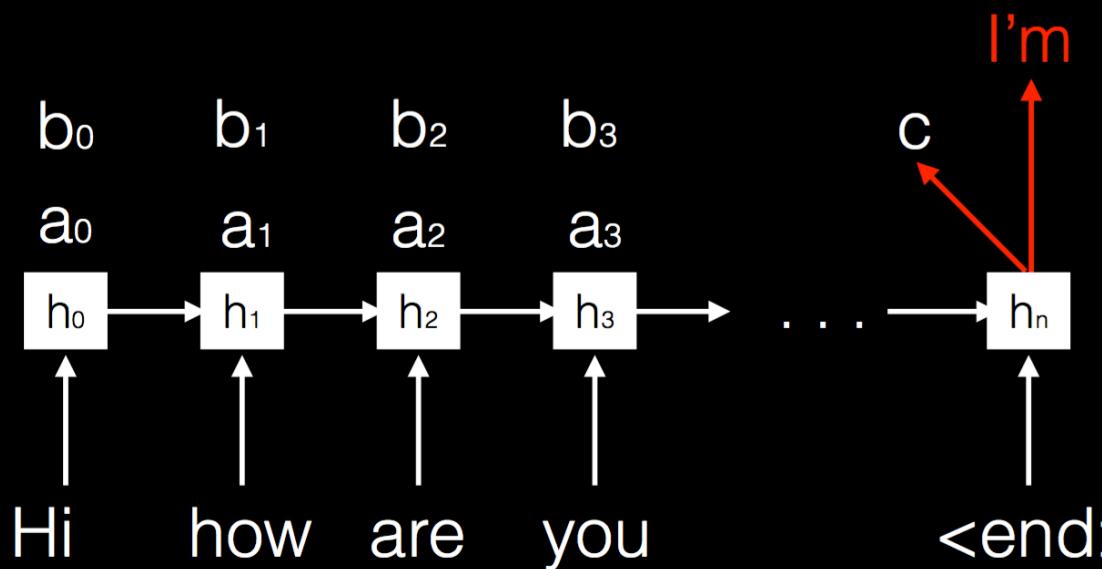
# Example III (Extension): Auto-Reply

- Take inner products with previous states



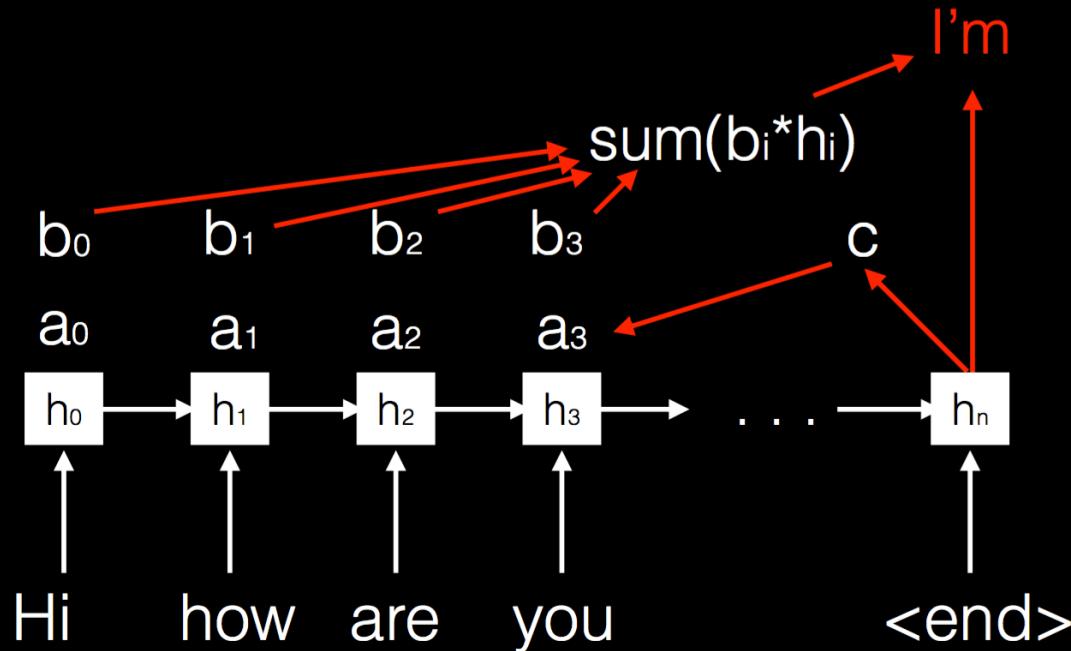
# Example III (Extension): Auto-Reply

- Take inner products with previous states



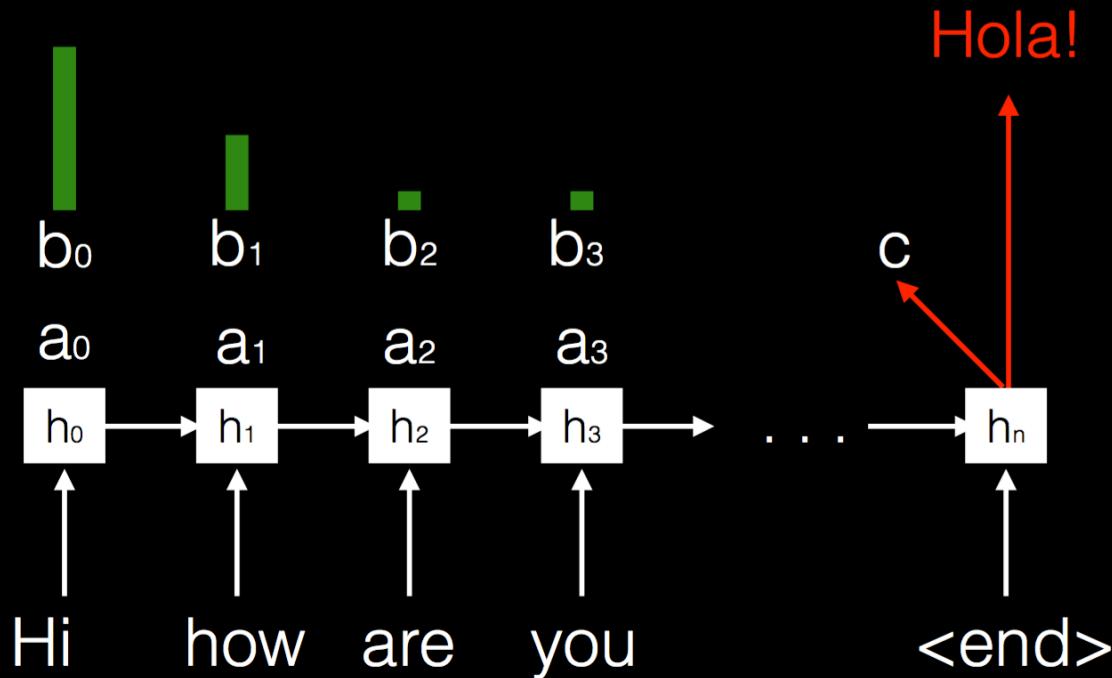
# Example III (Extension): Auto-Reply

- Pass through a neural net layer to predict final word



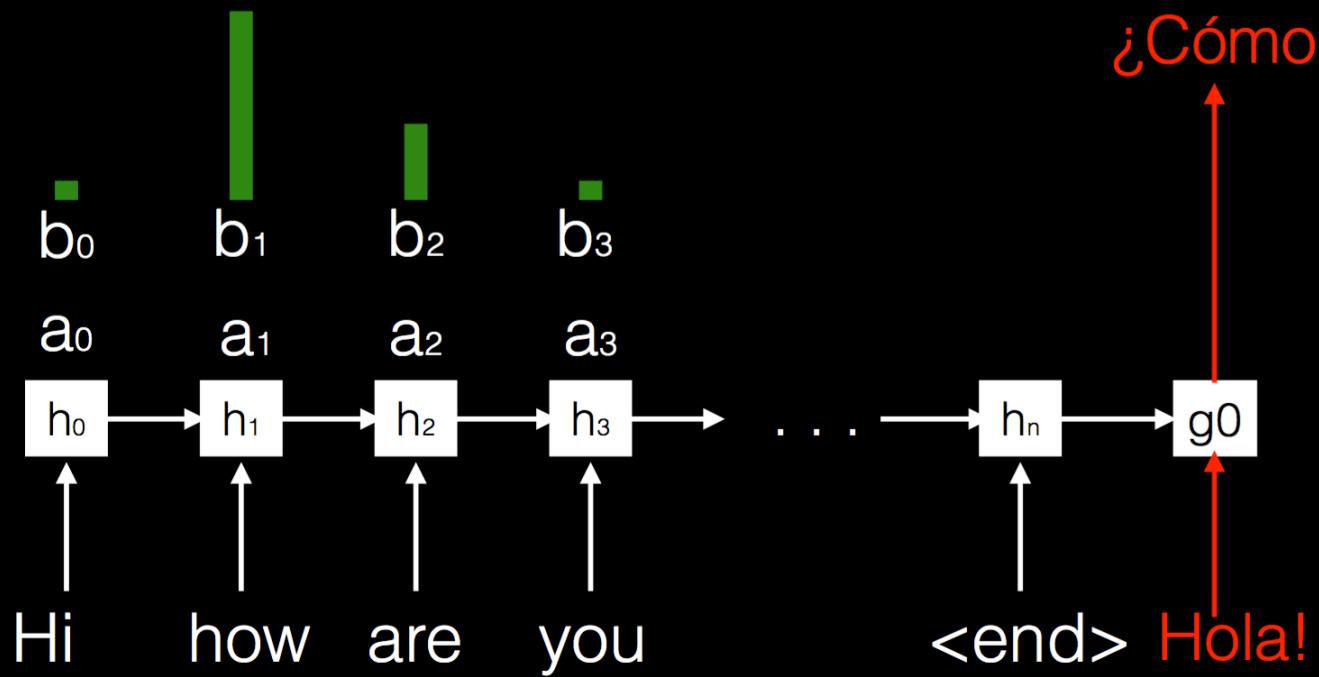
# Example III (Extension): Same with Translation!

- Same principle also applies for translation. The first prediction learns to focus on certain part of the input



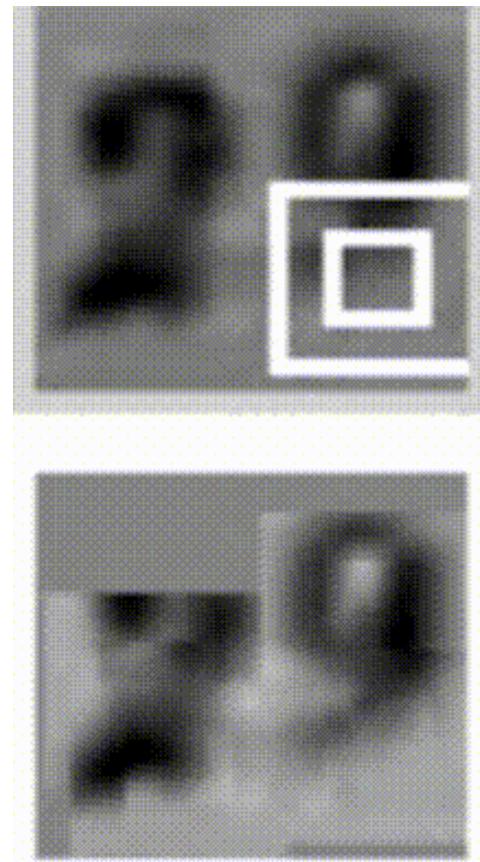
# Example III (Extension): Auto-Reply

- The second prediction learns to focus on certain part of the input



# Example V: Object Recognition with Visual Attention

- Even if we do not have sequences, we can still use RNNs to process the single fixed input in a sequence



<sup>1</sup>Figure: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

<sup>2</sup>Reference: <http://arxiv.org/abs/1412.7755>