Advanced Prediction Models

Deep Learning, Graphical Models and Reinforcement Learning

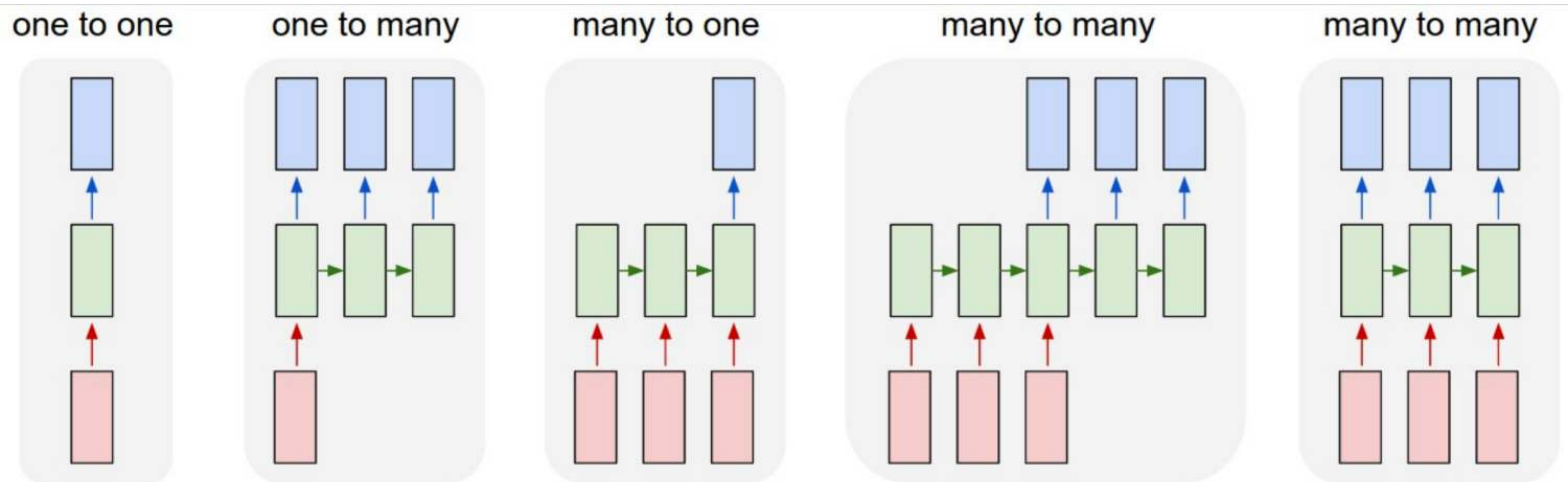
Today's Outline

- 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



Classifier
Fixed input
Fixed output

Sequence output
E.g.: Image captioning

Sequence input
E.g.: Sentiment analysis

Sequence input
Sequence output
E.g.: Machine translation

Sequence input
Sequence output
E.g.: Video classification

[1]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

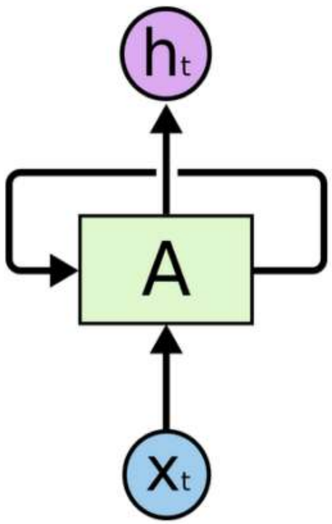
The Idea of Persistence (II)

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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolled Diagrams (I)

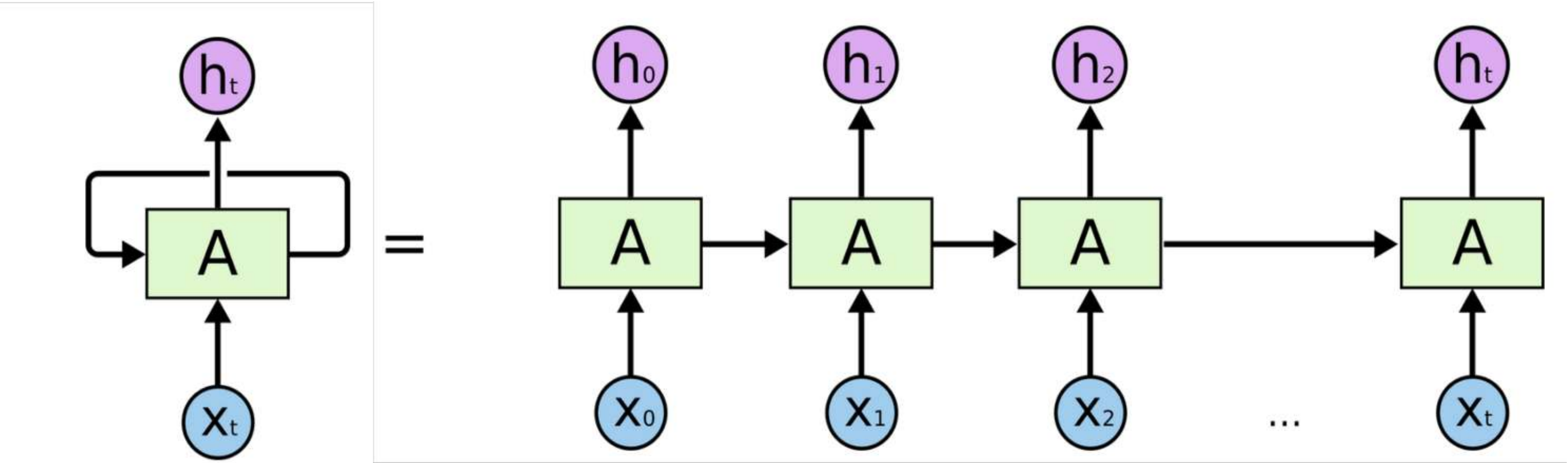
- Let A repersent a base network with two inputs and two outputs
- A loop based drawing of the architecture is as follows:



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolled Diagrams (II)

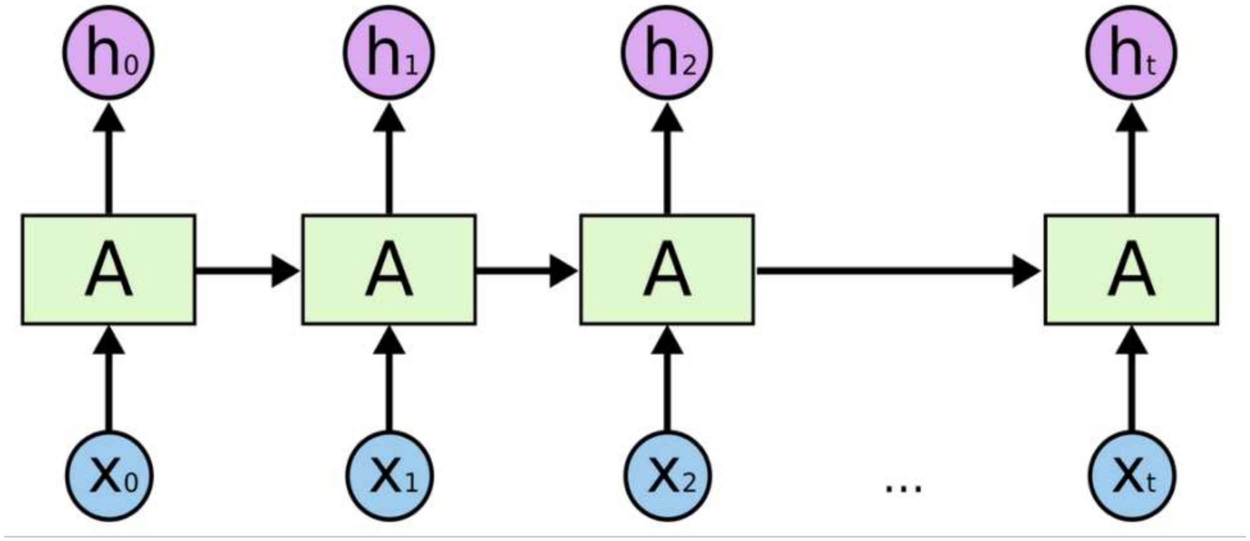
- Here is the unrolled representation



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolled Diagrams (III)

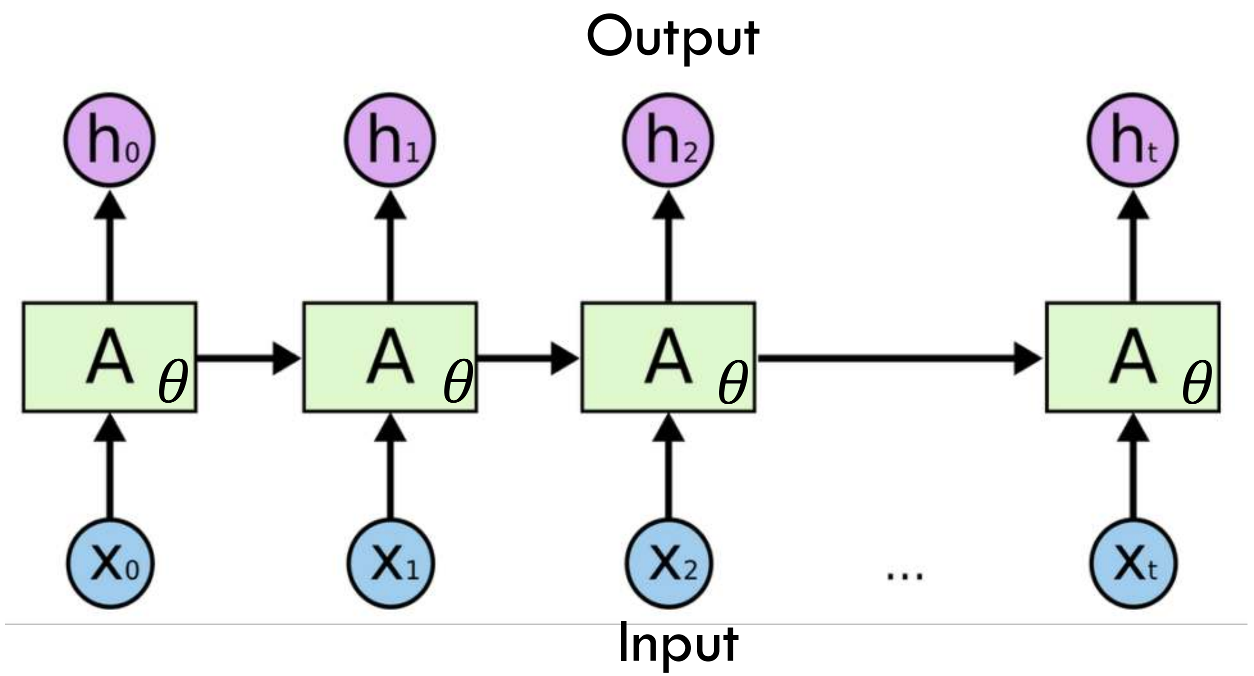
- This sequential or repetitive structure is useful for working with sequences
  - Of images
  - Of words



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolled Diagrams (V)

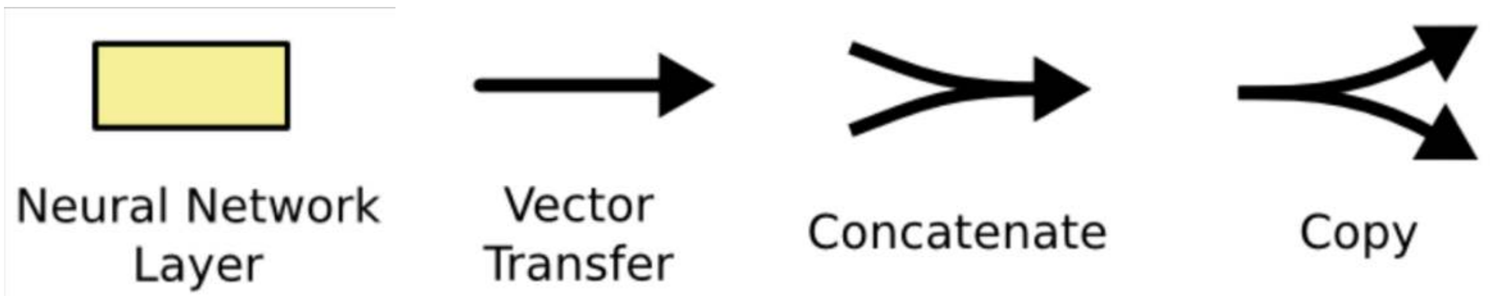
- At a stage, they accept an input and give an output, which are parts of sequences



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanilla RNN (I)

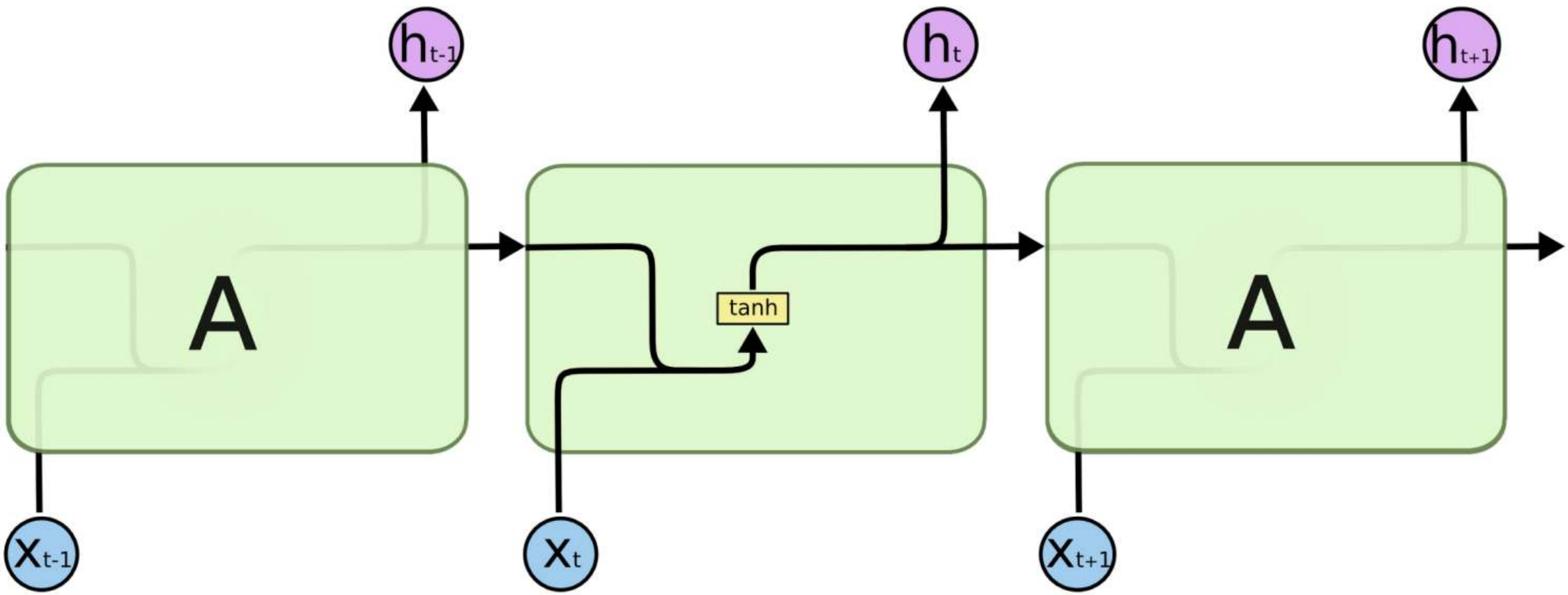
- Some quick notation
  - Dark arrow represents a vector
  - Box represents a (fully connected hidden) layer



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanilla RNN (II)

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



  - Assuming a single hidden layer with tanh nonlinearity

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Vanilla RNN using Numpy

- Training an RNN means finding θ (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

[1]Reference: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LM Example: Data and Embedding

- Vocabulary: {h,e,l,o}

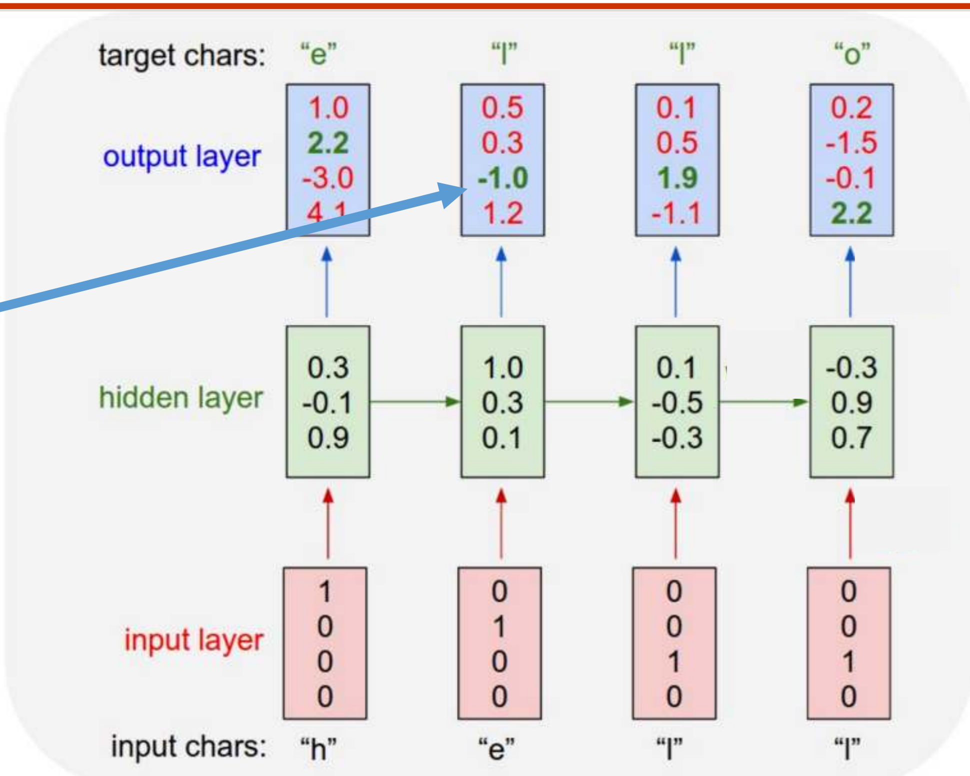
- Training sequence: {h,e,l,l,o}
  - Four training examples:
    - P(e|h) should be high
    - P(l|he) should be high
    - P(l|hel) should be high
    - P(o|hell) should be high

- Embedding:
  - Encode each character as a 4-dimensional vector

[1]Reference: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LM Example: RNN

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



[1]Figure: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- 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

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("… advanced prediction") = "models"
  - Here, the immediate preceding words are helpful

Long Term vs Short Term (II)

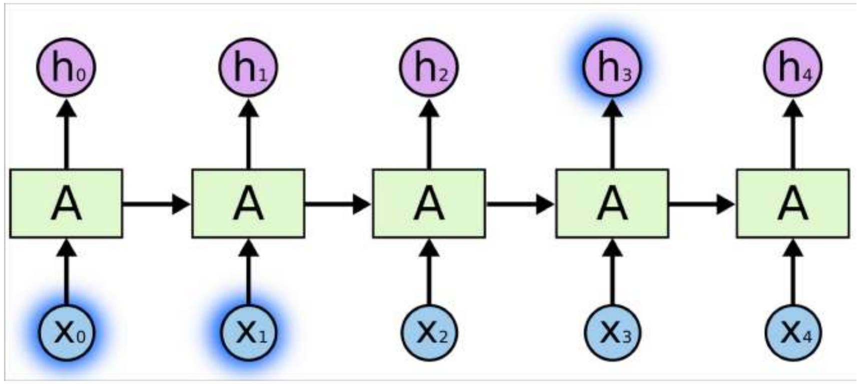
- Consider a model predicting next word based on previous words

- Case A:
  - R("… advanced prediction") = "models"
  - Here, the immediate preceding words are helpful

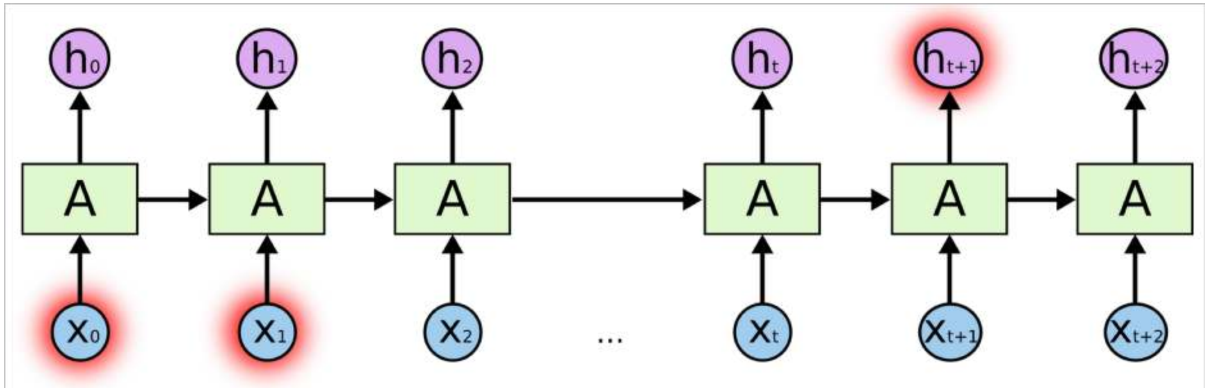
- Case B:
  - R("I went to UIC… I lived in [?]") = "Chicago"
  - Here, more context is needed
    - Recent info suggests [?] is a place.
    - Need the context of UIC from further back

Long Term vs Short Term (III)

- Consider a model predicting next word based on previous words
- Case A:



- Case B:



[1]Figures: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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)

[1]Reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

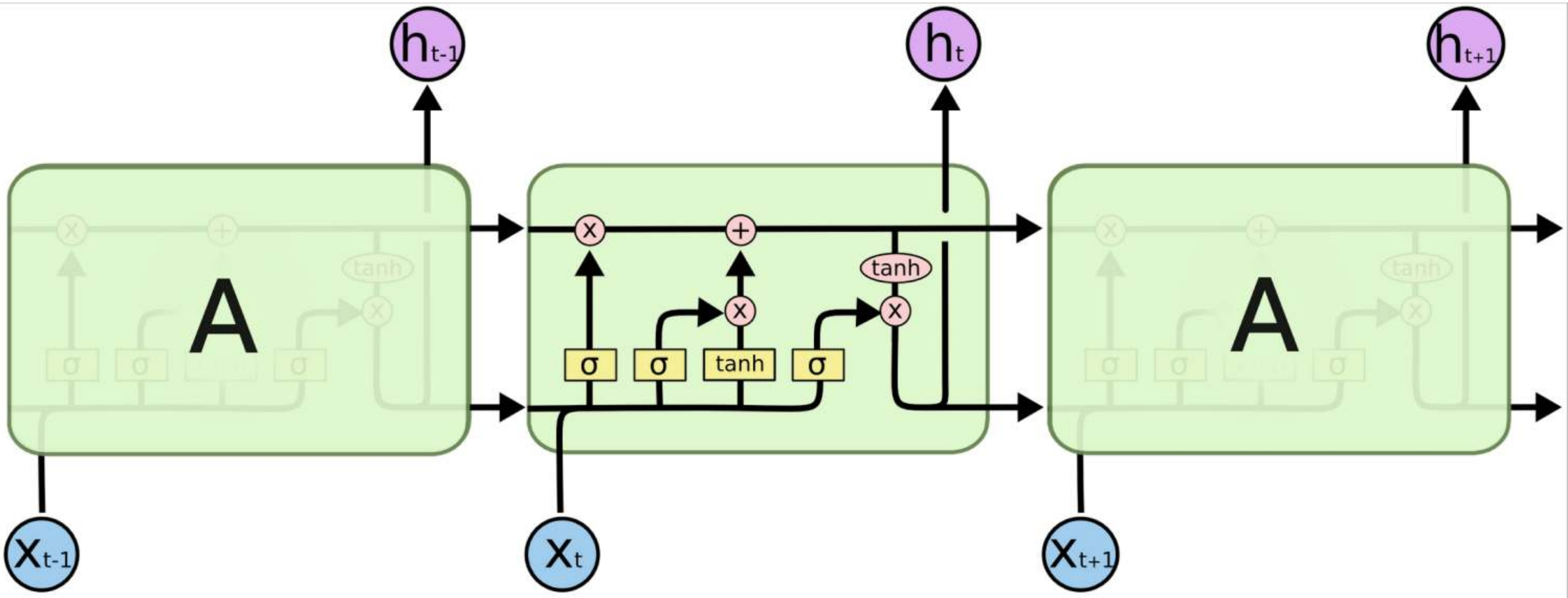
LSTM: Long Short Term Memory based RNN

- 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

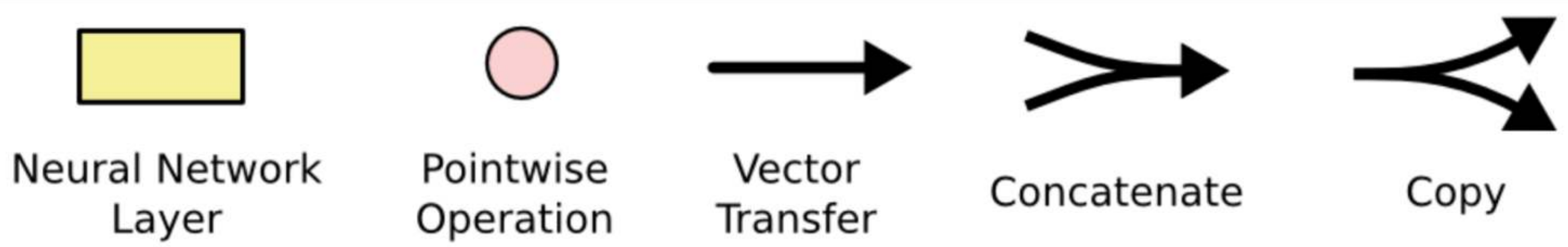
LSTM: Block Level

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



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: Recall Notation

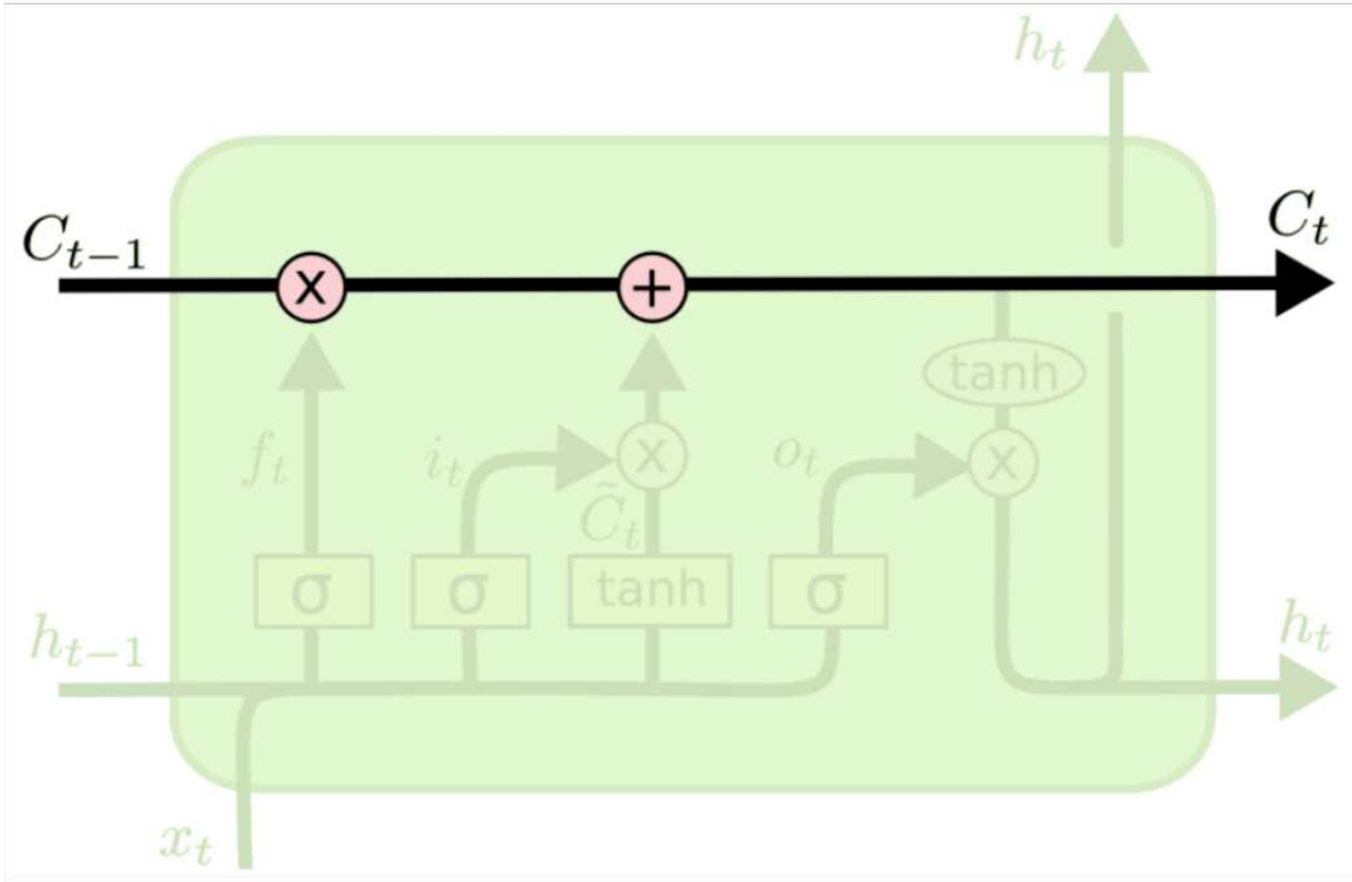


- Dark arrow represents a vector, output from one layer and input to another
- Circle represents element-wise operations
  - Example: sum of two vectors
- Box represents a (fully connected) hidden layer

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: Cell State (I)

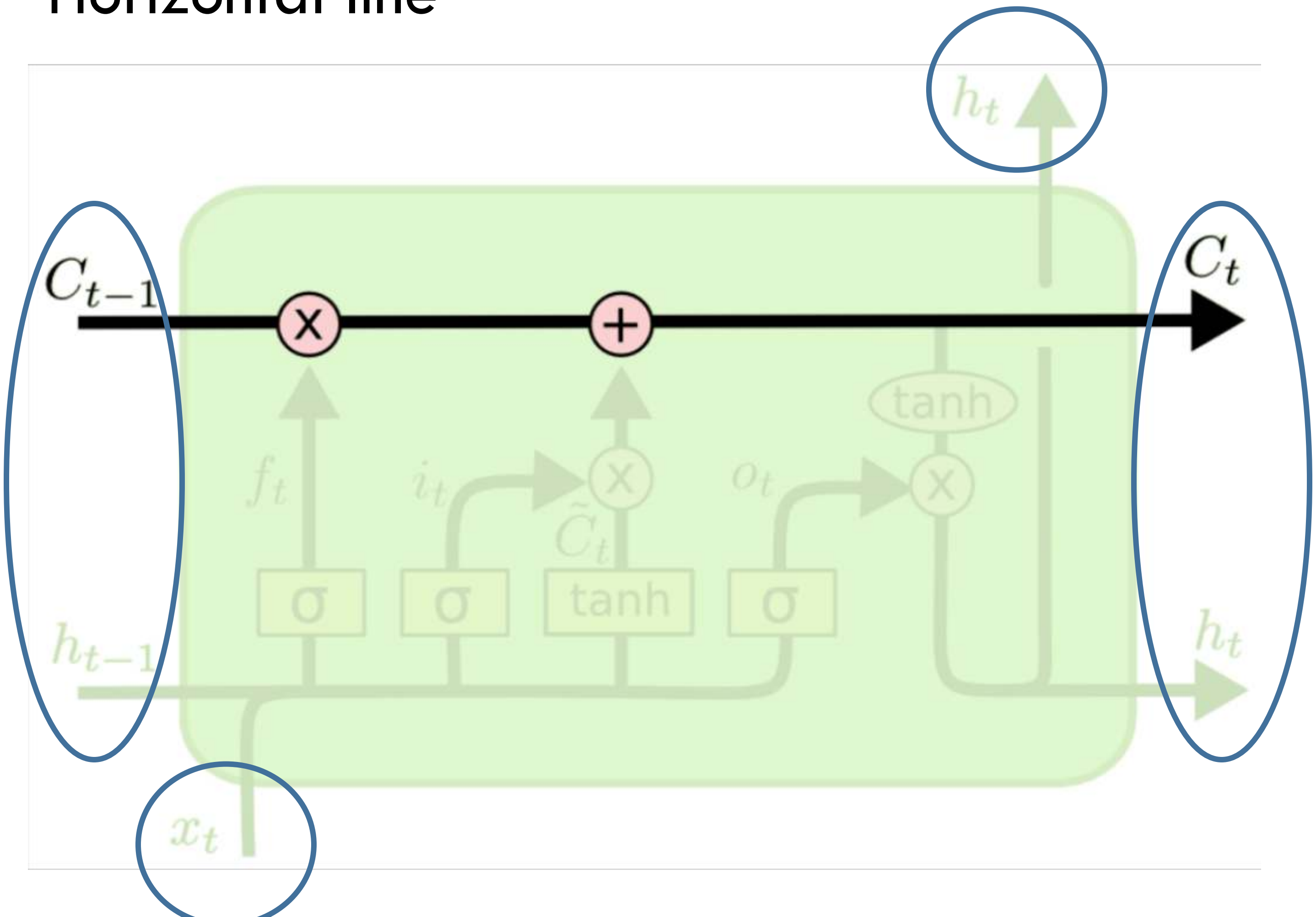
- There is a notion of cell state
  - Horizontal line



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

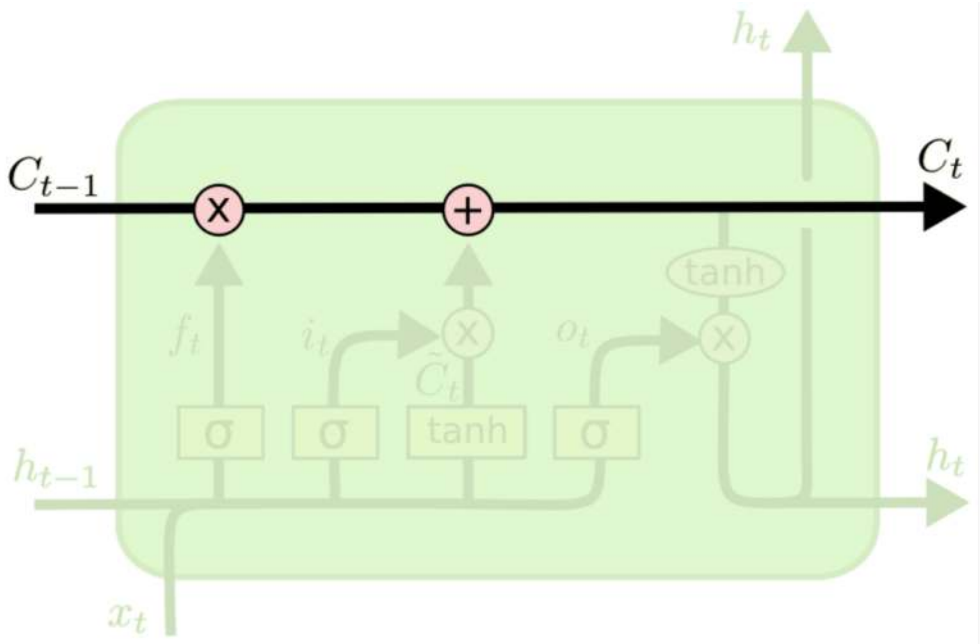
LSTM: Cell State (I)

- 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

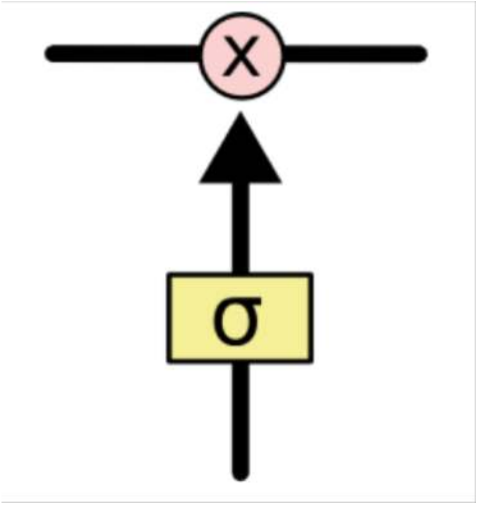


[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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

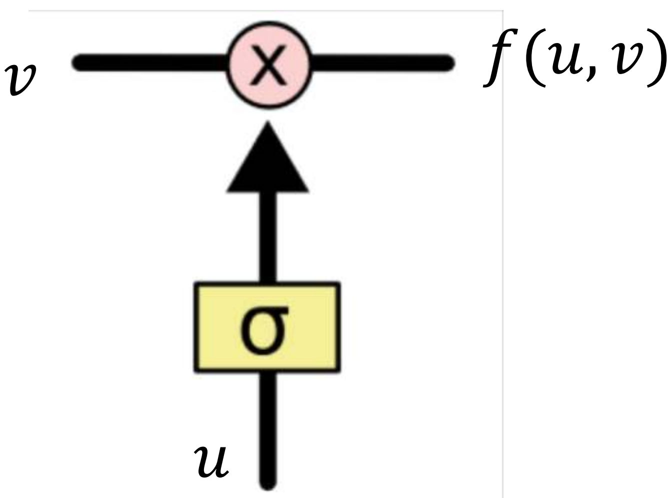


[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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



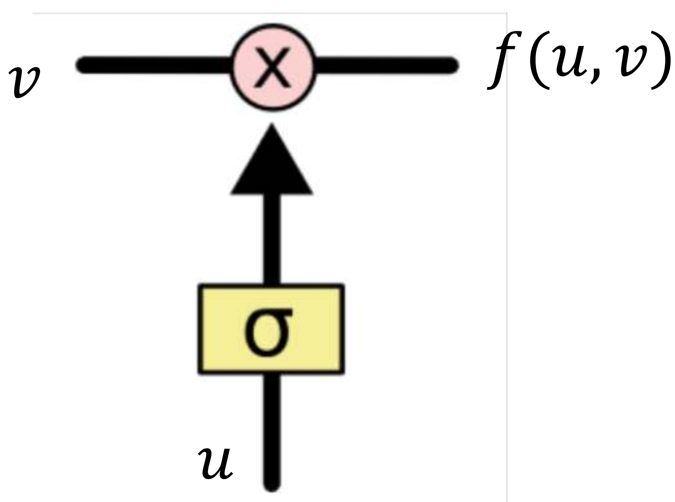
Mathematically,

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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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’



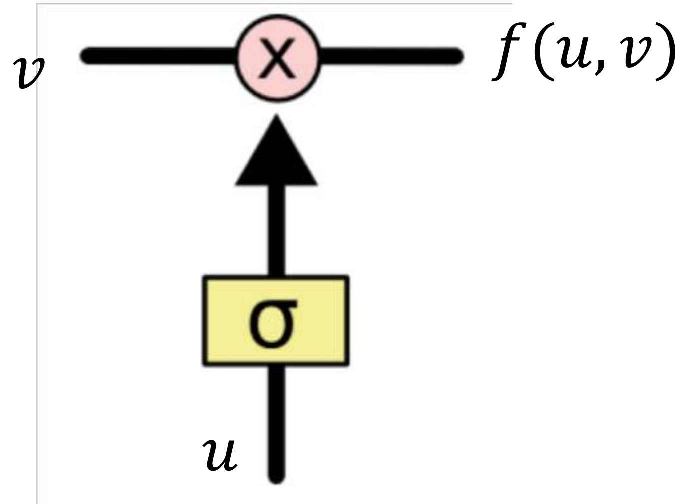
Mathematically,

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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM: Gates (III)

- LSTM has three gates to control the cell state



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

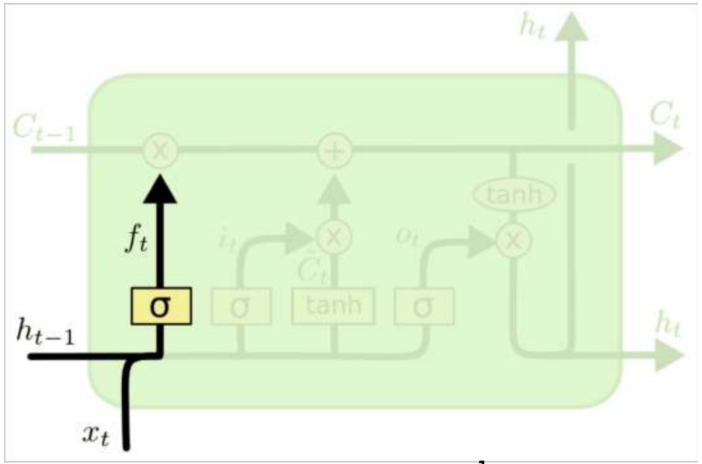
LSTM: Forget Old Information

- 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



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

[1]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

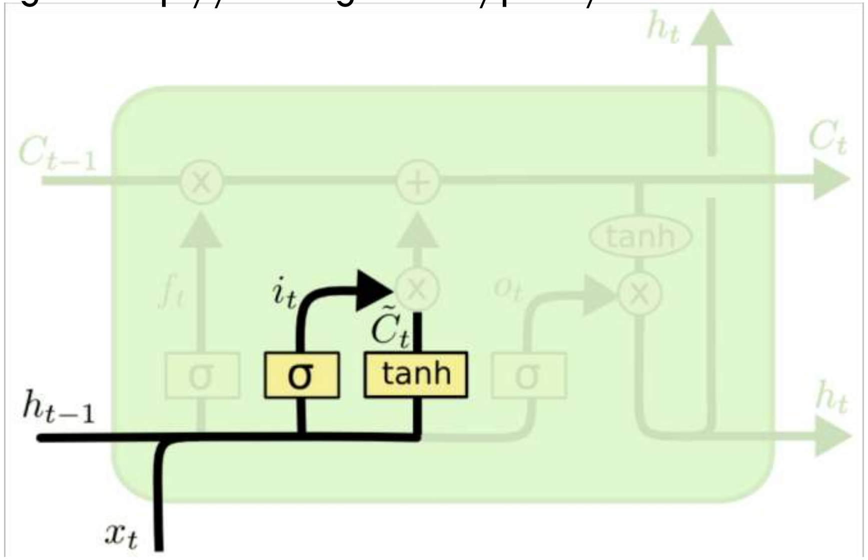
- Next step: decide what new information we will store in cell state
- Two ingredients
  - Input gate layer
  - Tanh layer

- 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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/



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

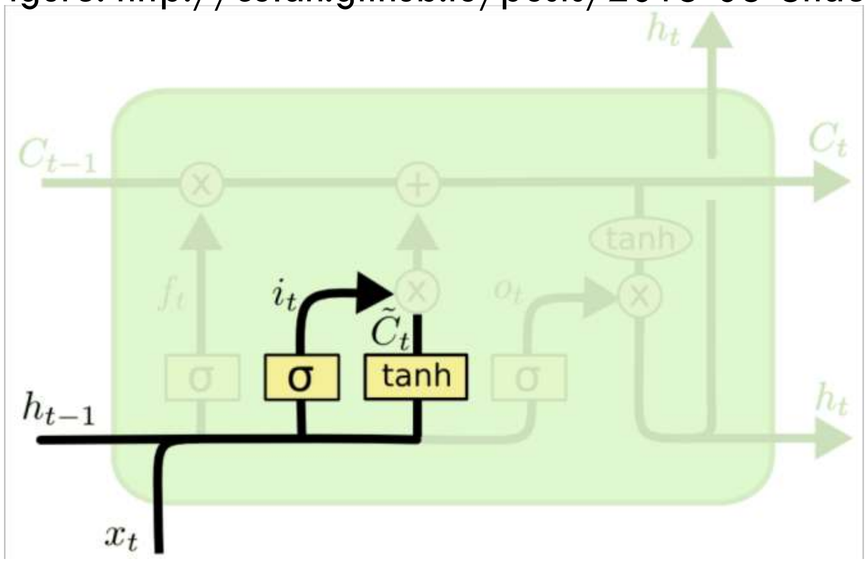
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- 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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

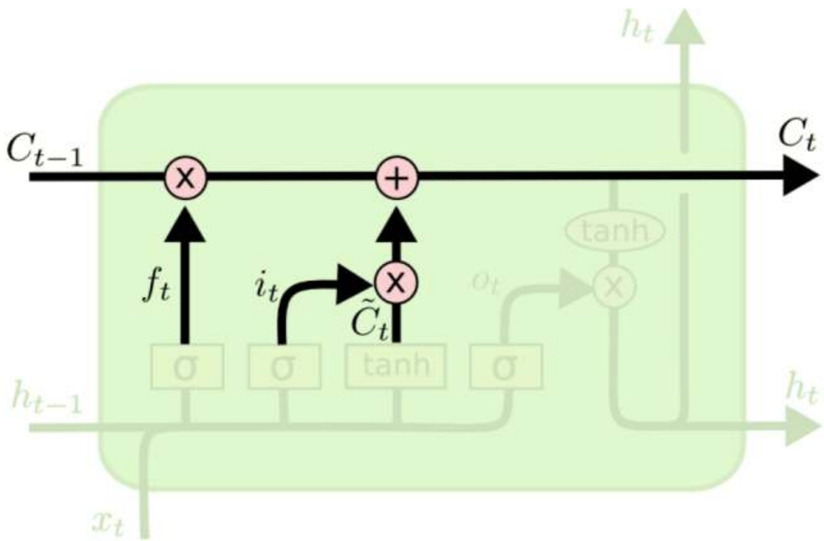


- Combine $\tilde{C}_t$ with the output i_t of the input gate layer to get $i_t \otimes \tilde{C}_t$

- 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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

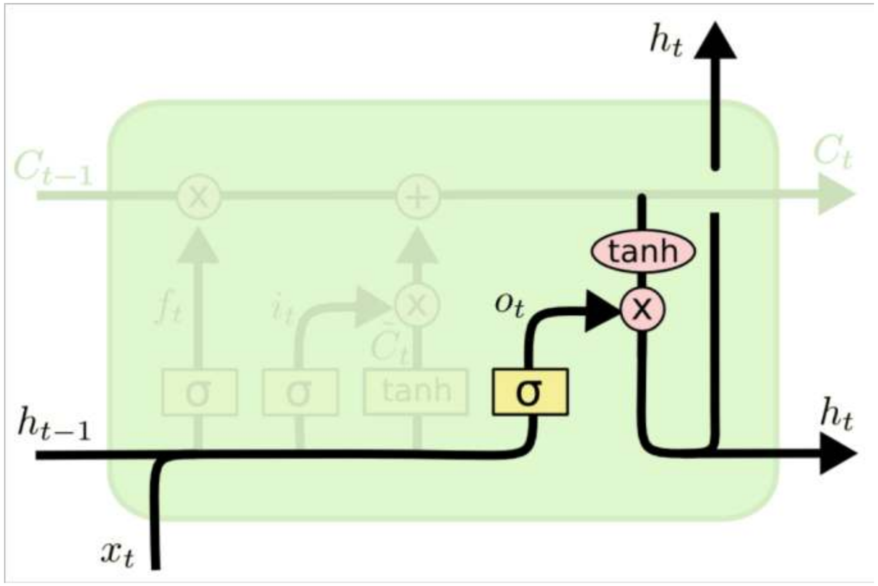
LSTM: Output

- 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

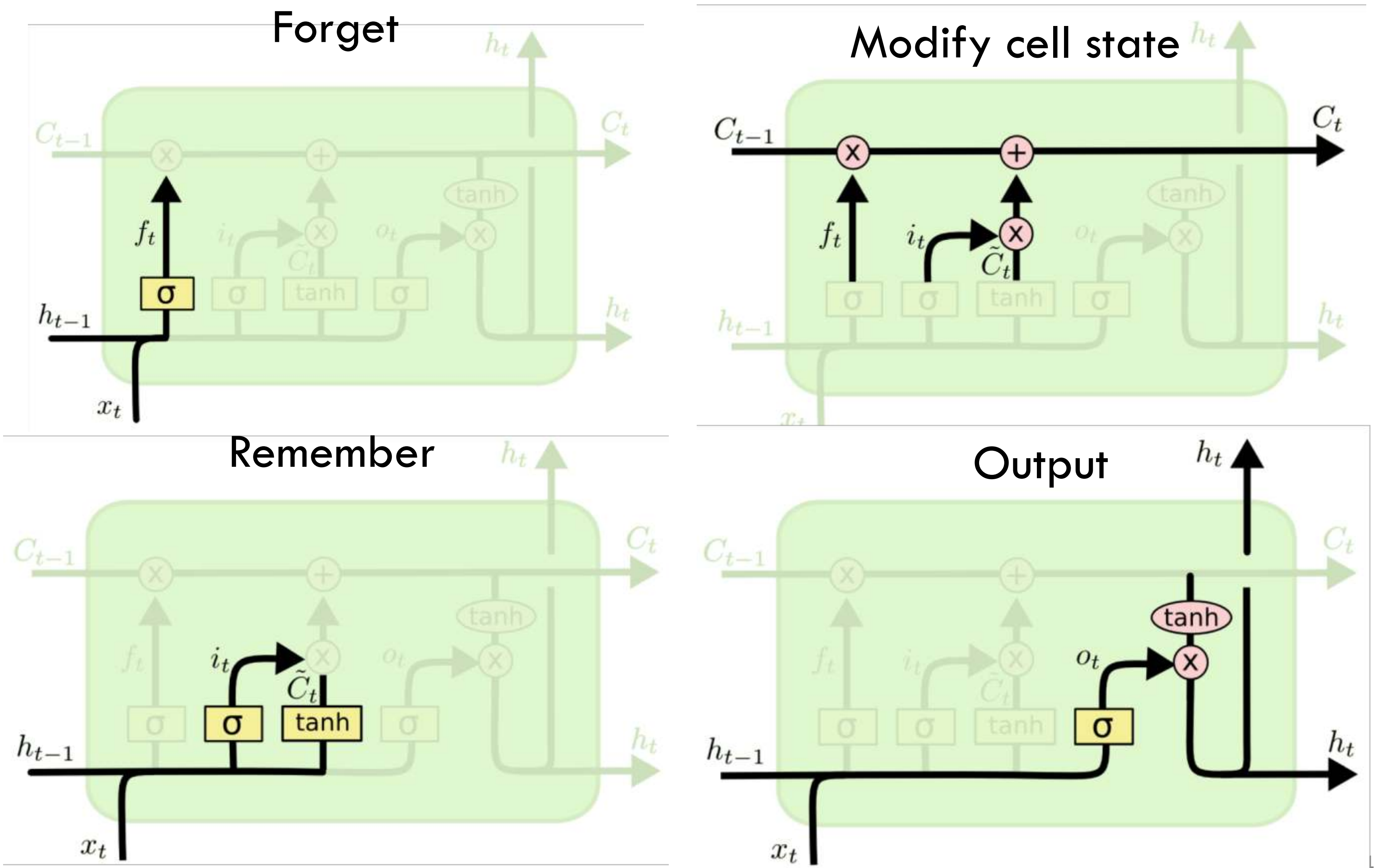


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

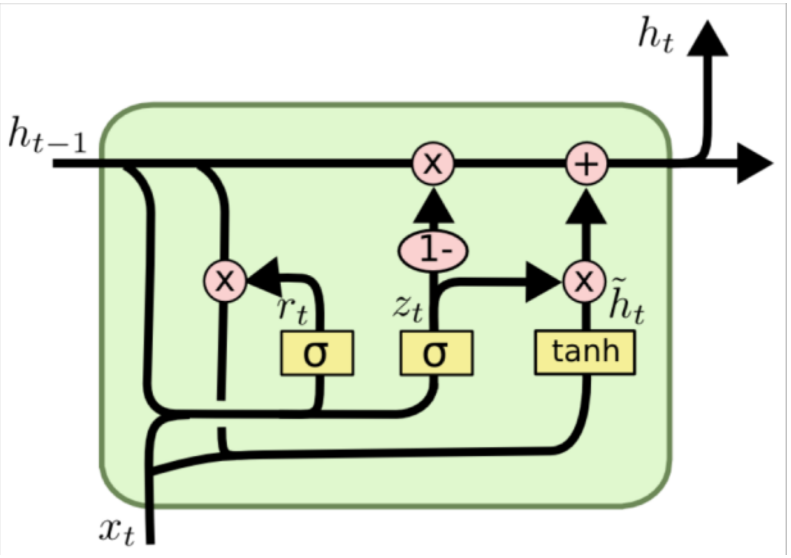
LSTM: Architecture Summary



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

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
  - …



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

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

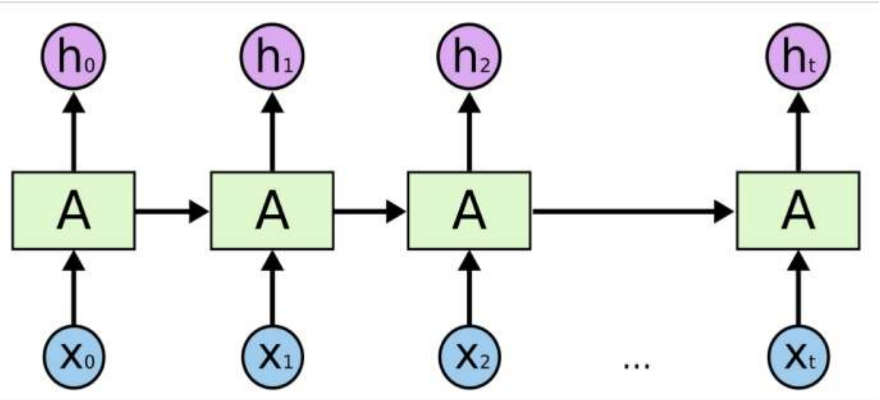
$$\tilde{h}_t = \tanh\left(W \cdot [r_t \otimes h_{t-1}, x_t]\right)$$

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

[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Other Variations in the Family of RNNs (II)

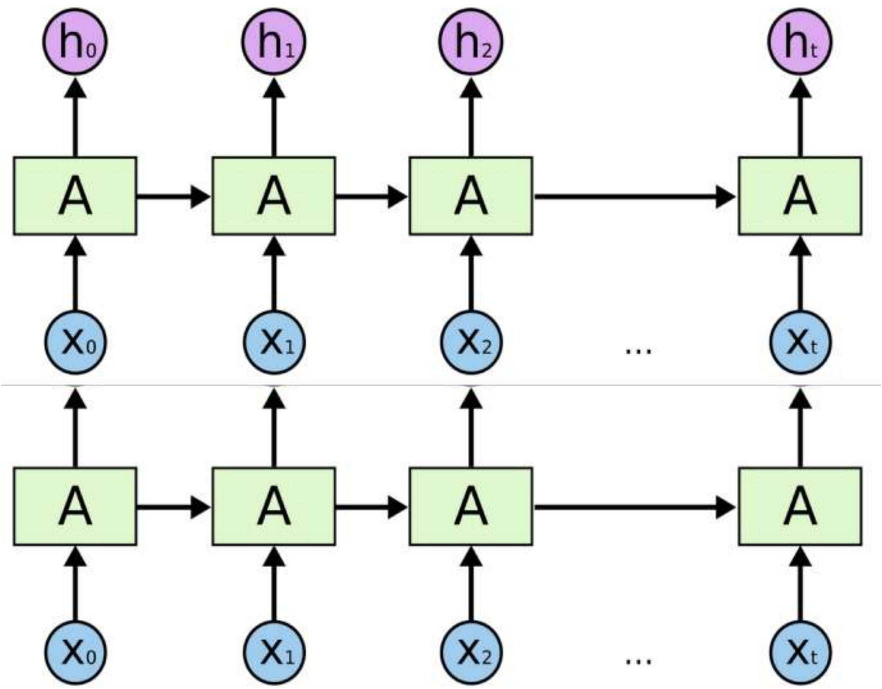
- One can also go deep by stacking RNNs on top of each other



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Other Variations in the Family of RNNs (II)

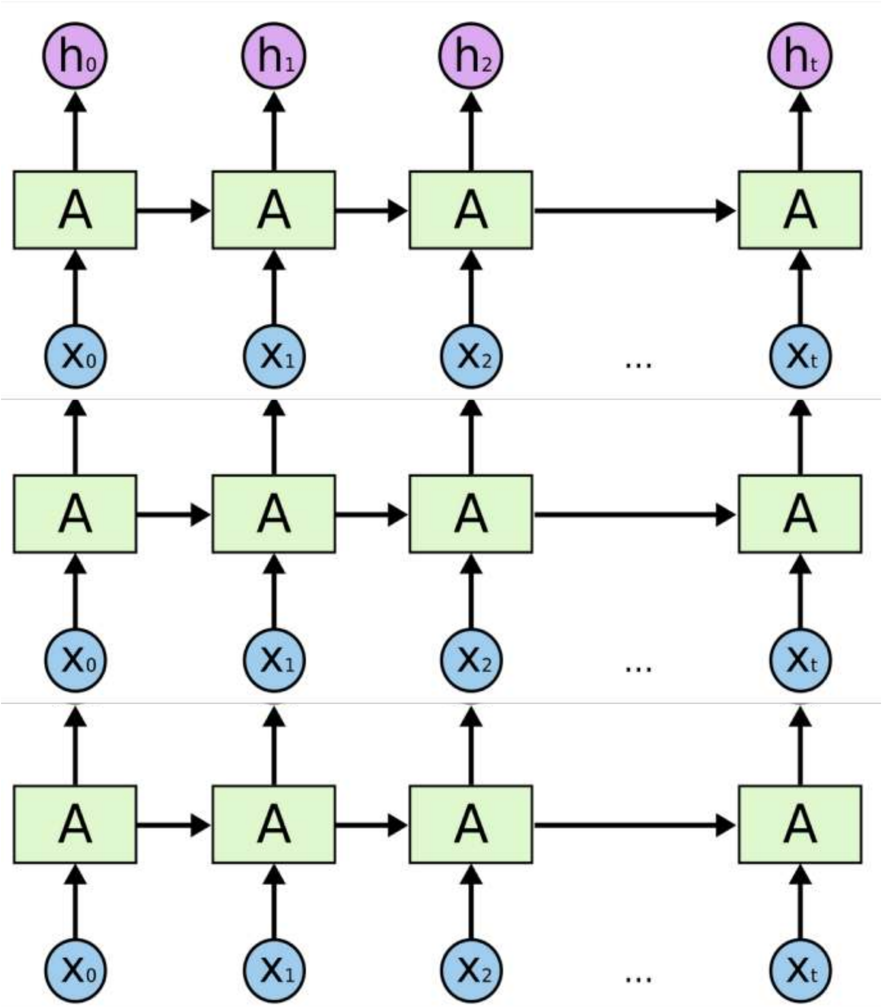
- One can also go deep by stacking RNNs on top of each other



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Other Variations in the Family of RNNs (II)

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



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Other Variations in the Family of RNNs (III)

- Extensive investigation has been done to see which variations are the best[1,2]

- 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

[1]Reference: http://arxiv.org/pdf/1503.04069.pdf

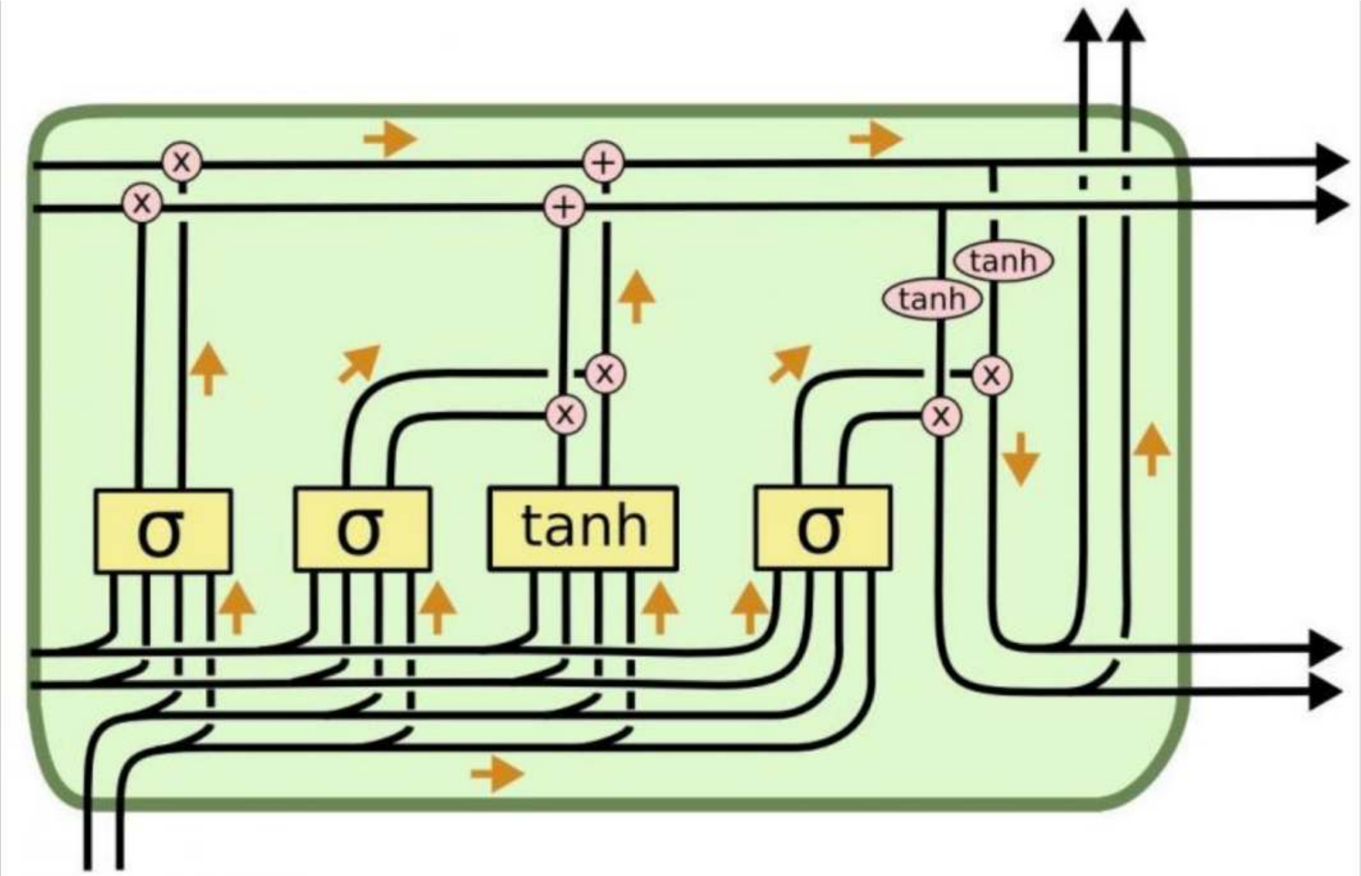
[2]Reference: http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf

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



[1]Figure: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Questions?

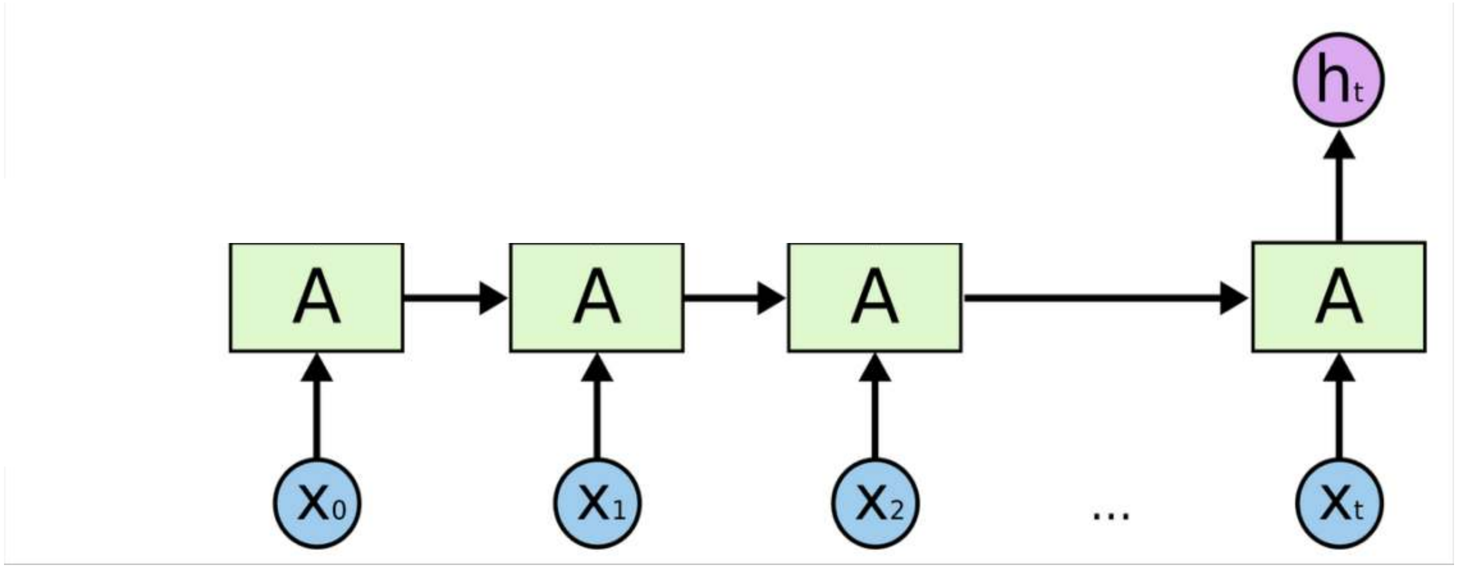
Today's Outline

- 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

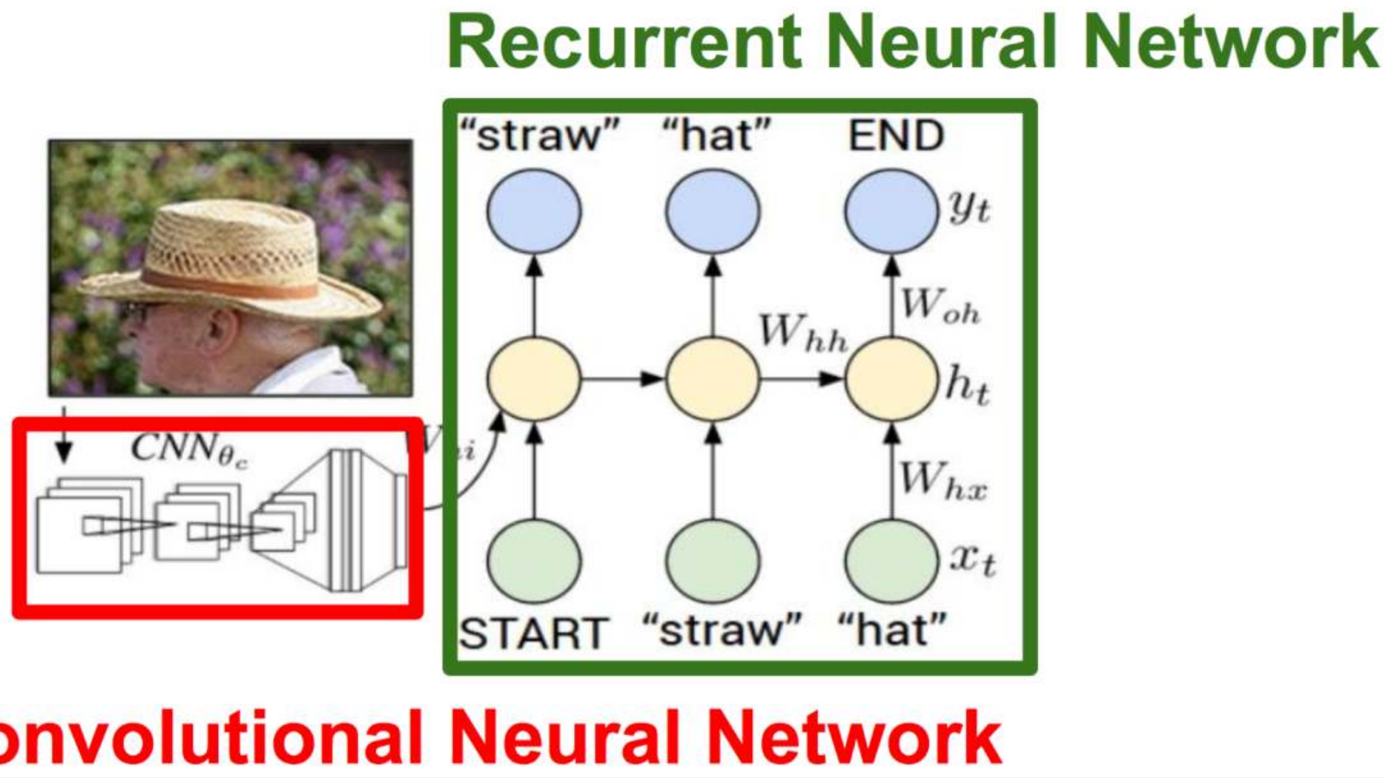
- We saw how to use a CNN for this task.
- Now, we can use an RNN as well:



[1]Additional Info: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

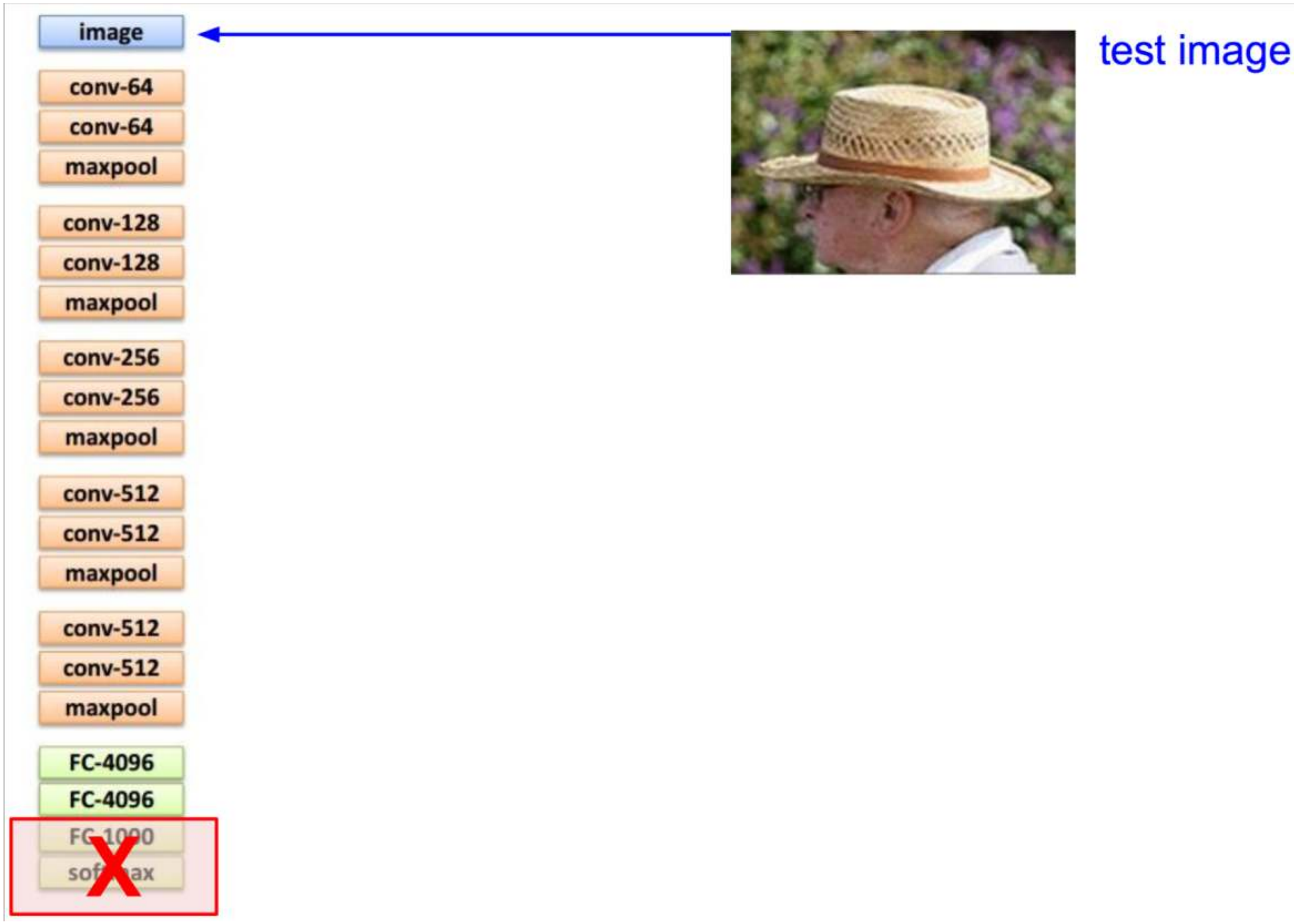
Example II: Image Captioning

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



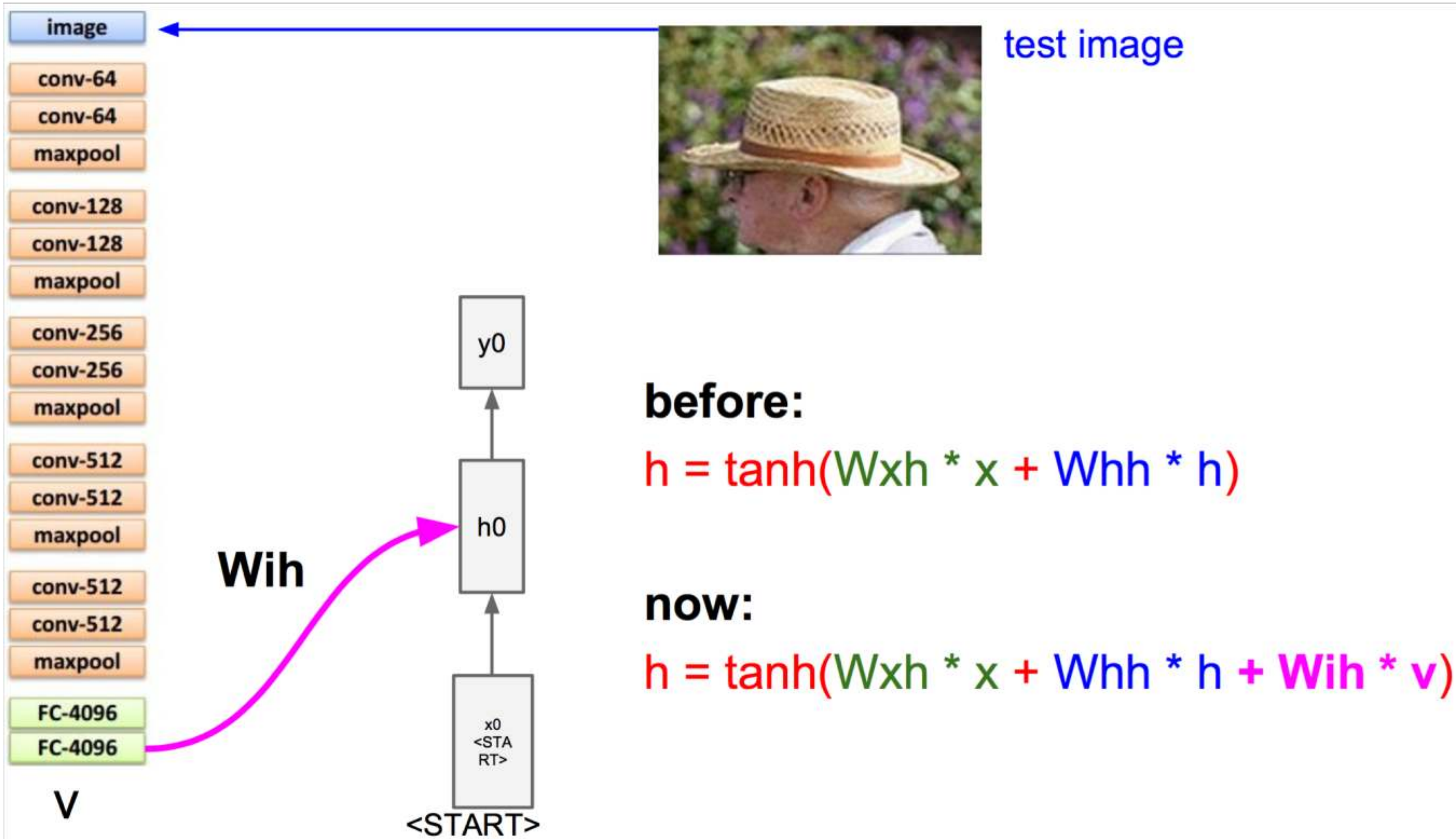
[1]Figure: http://cs231n.stanford.edu/ Lecture 10

Example II: Image Captioning



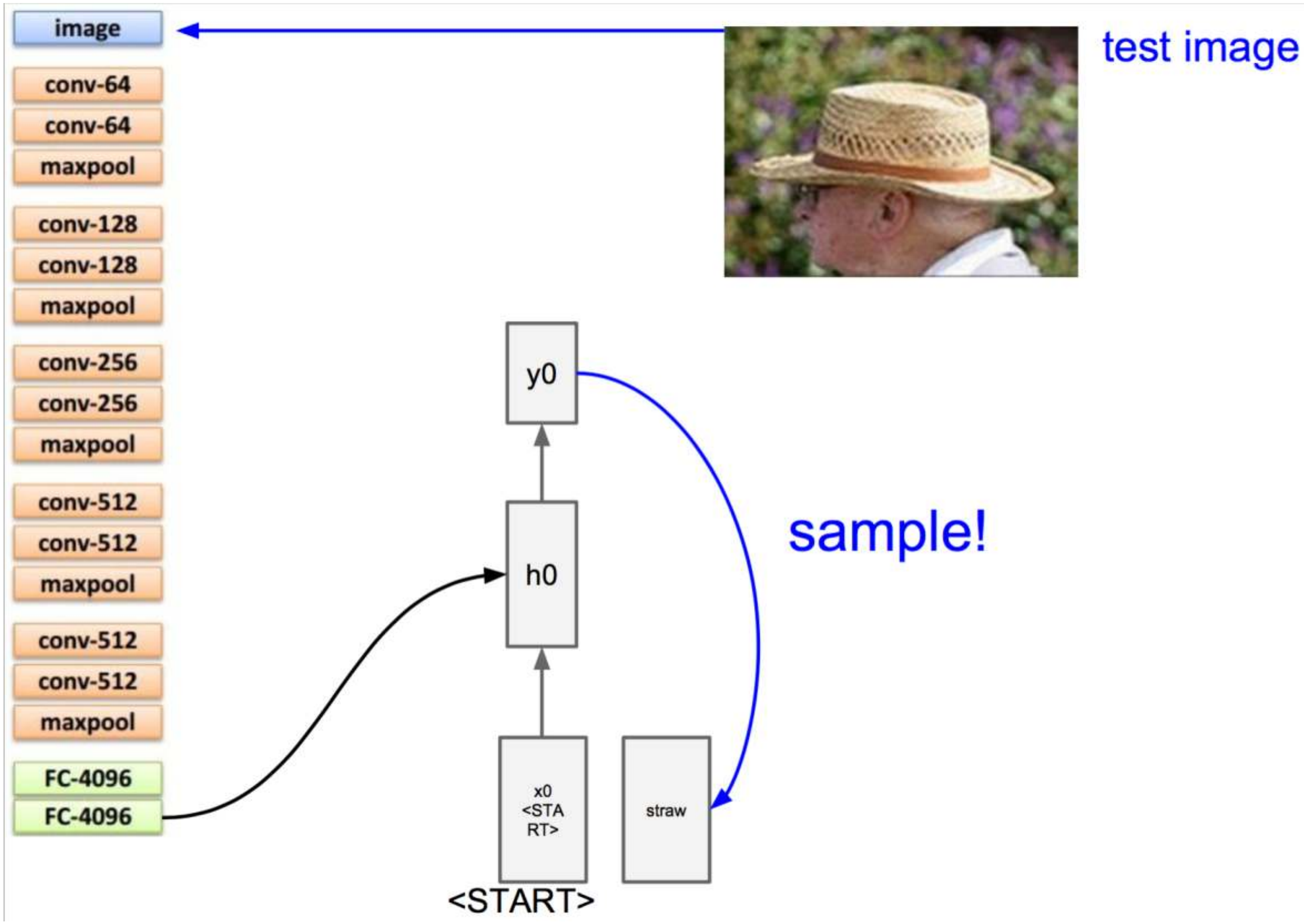
[1]Figure: http://cs231n.stanford.edu/ Lecture 10

Example II: Image Captioning



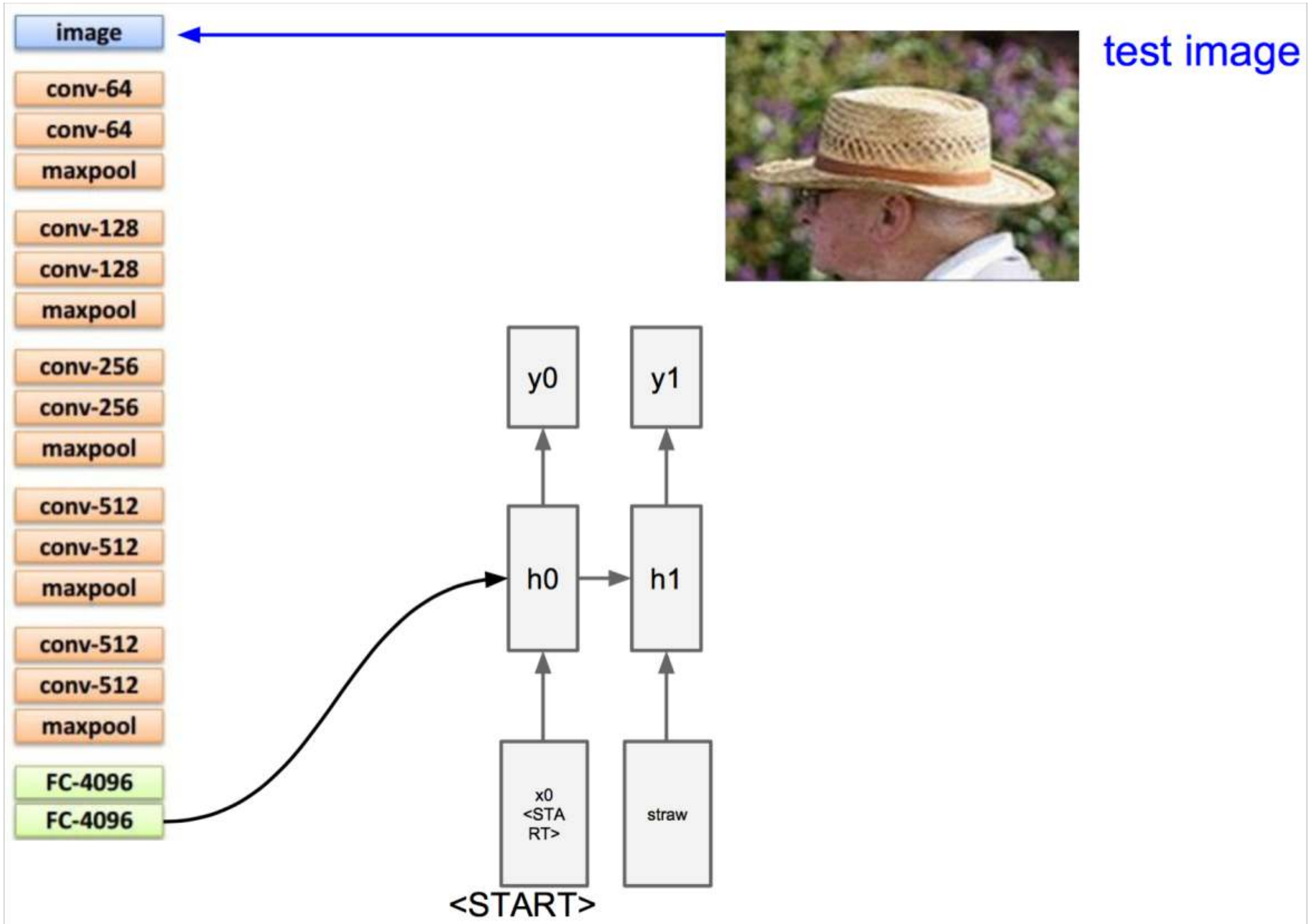
[1]Figure: http://cs231n.stanford.edu/ Lecture 10

Example II: Image Captioning



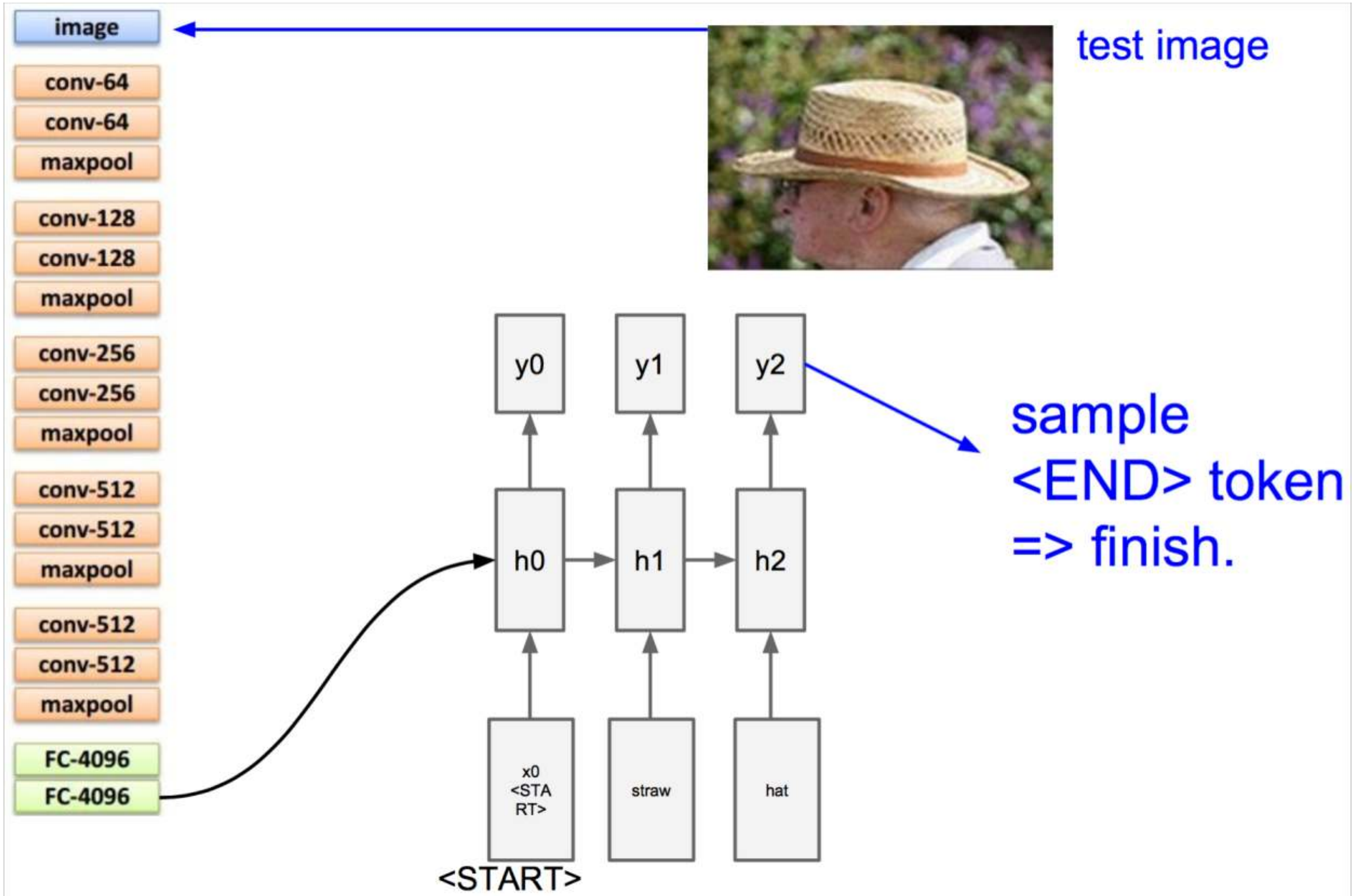
[1]Figure: http://cs231n.stanford.edu/ Lecture 10

Example II: Image Captioning



[1]Figure: http://cs231n.stanford.edu/ Lecture 10

Example II: Image Captioning



[1]Figure: http://cs231n.stanford.edu/ Lecture 10

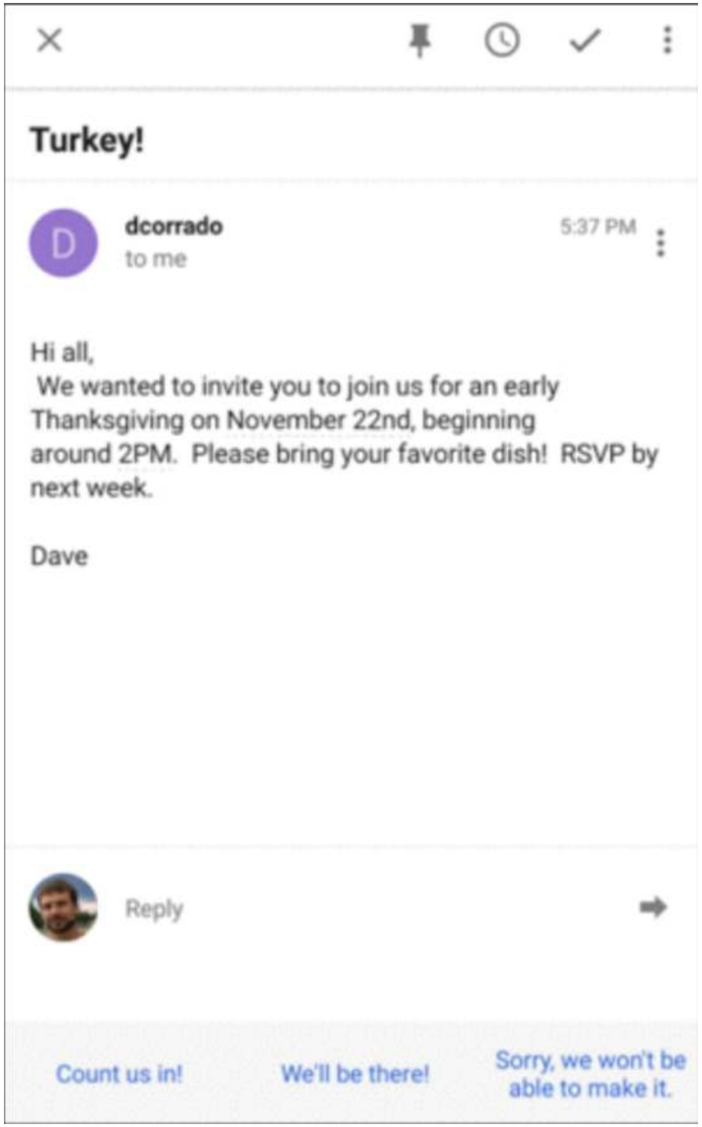
Example III: Auto-Reply

- 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

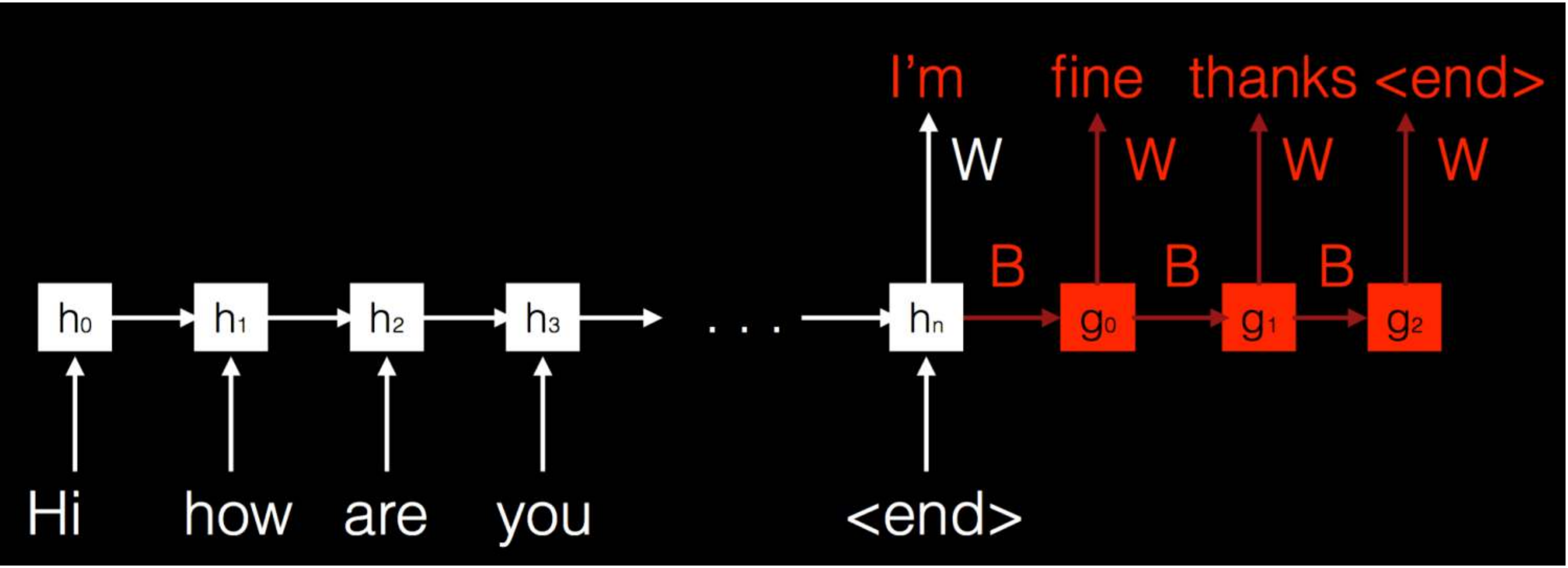
- Auto-reply is a feature where the computer reads your email and responds appropriately



[1]Figure: Quoc Le, Google Brain

Example III: Auto-Reply

- First version

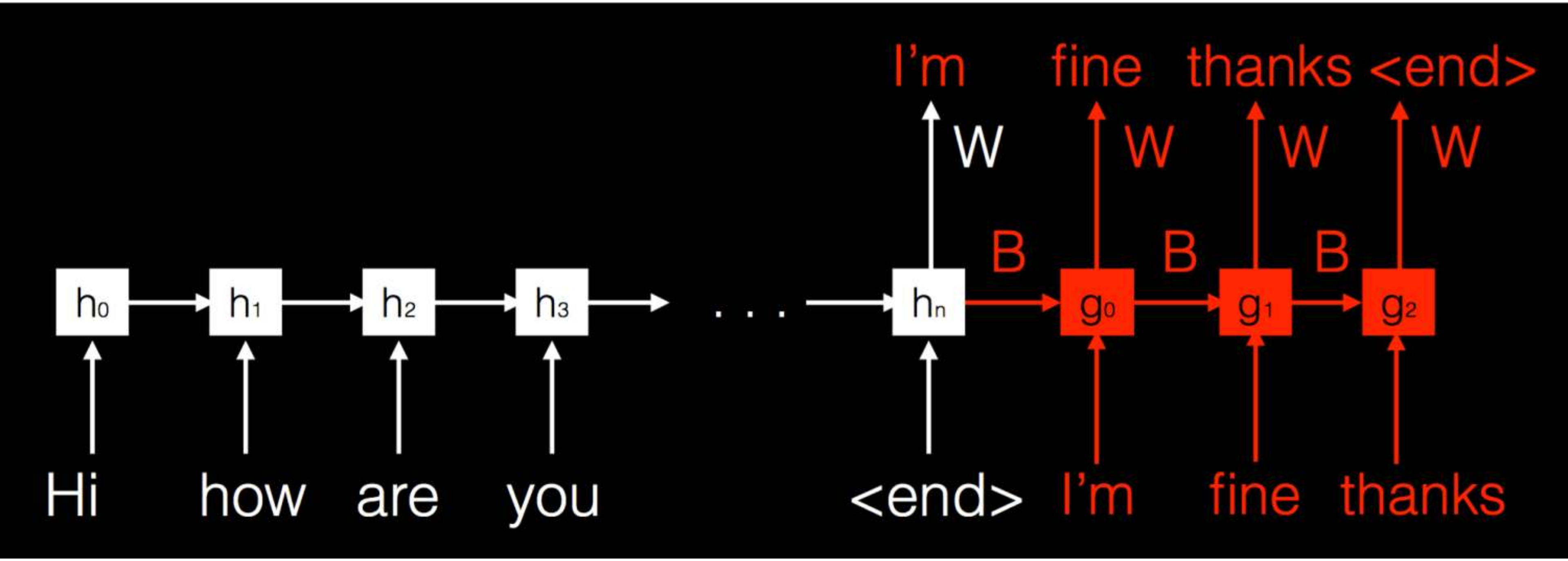


- Note that the number of classes in output is the number of words in the vocab!

[1]Figure: Quoc Le, Google Brain

Example III: Auto-Reply

- Second version



Encoder *Decoder*

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

[1]Figure: Quoc Le, Google Brain

Example III: Auto-Reply

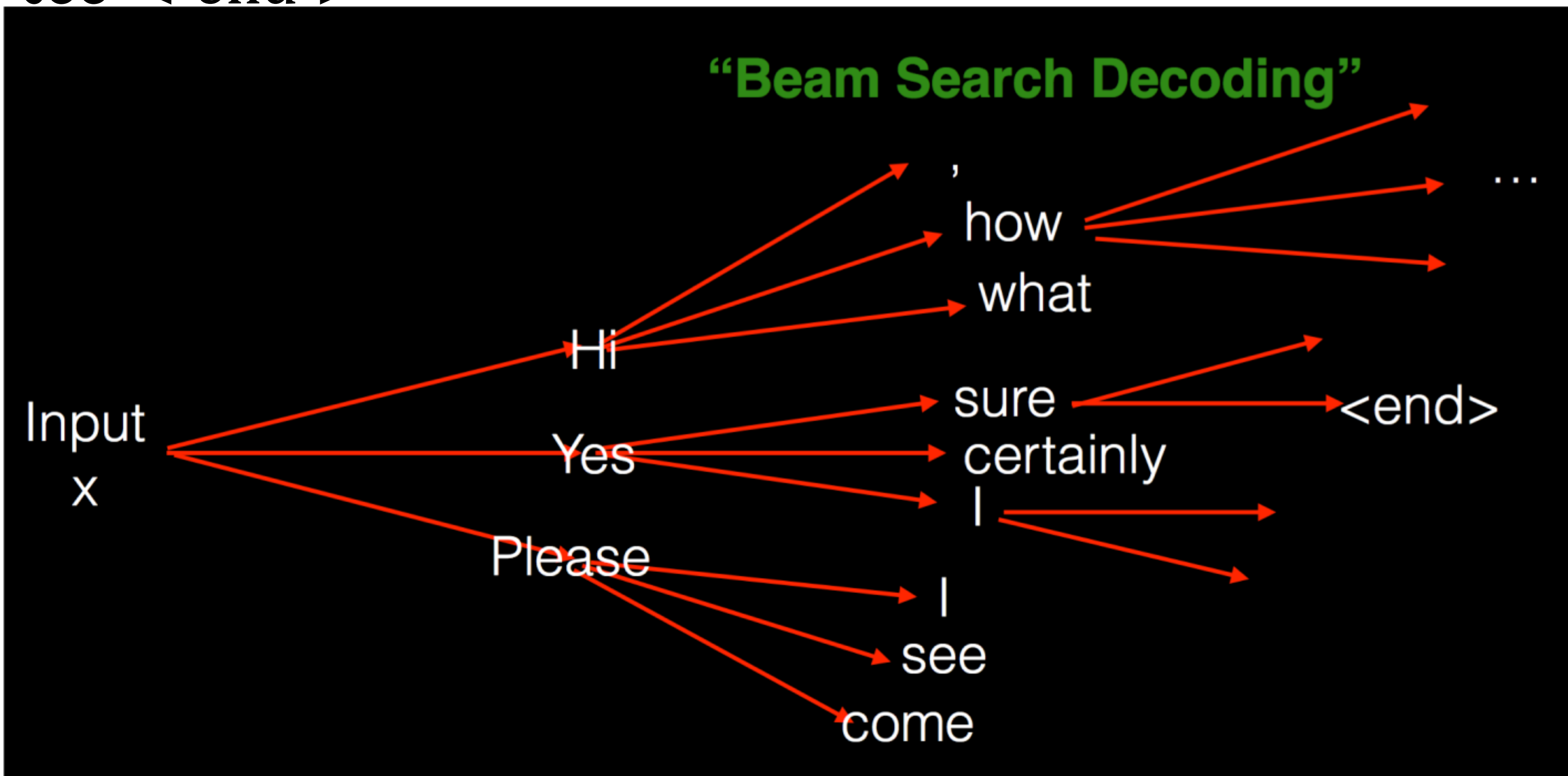
- As we saw with image captioning example,
- Given input sequence x, we first output y_0 which has the highest probability
- Given x and y_0, we output y_1, which has the highest probability

- This is greedy
  - Does not correct for mistakes

[1]Figure: Quoc Le, Google Brain

Example III: Auto-Reply

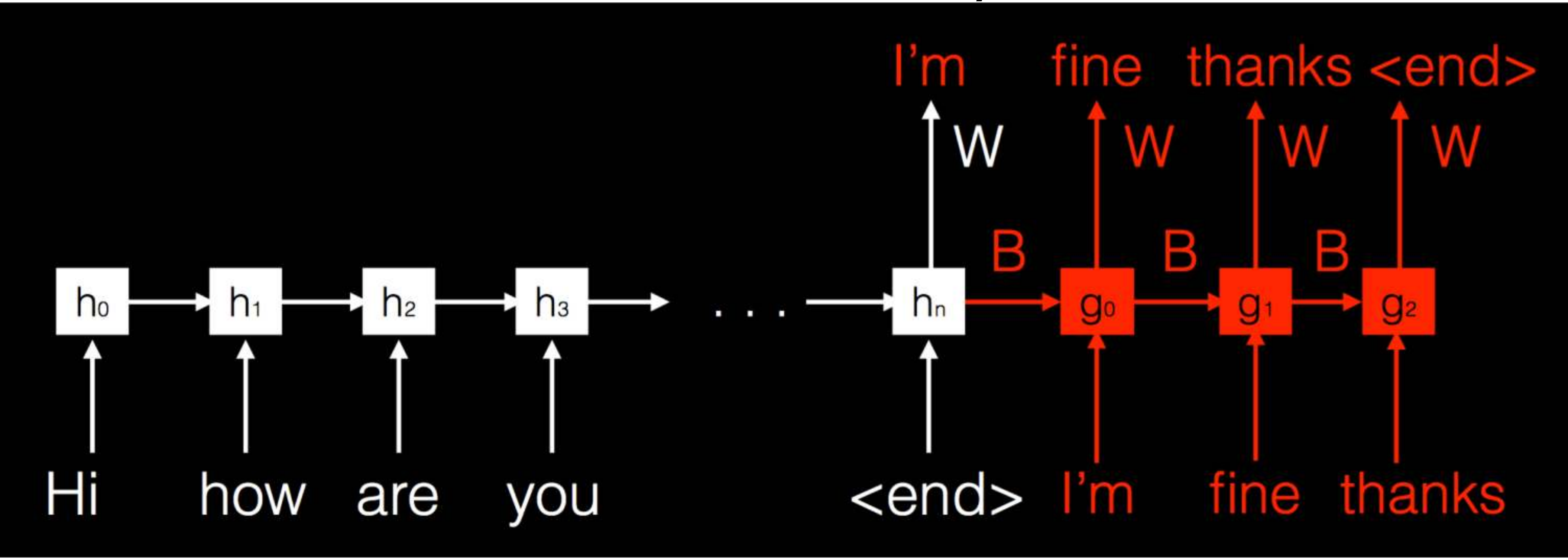
- Beam Search Decoding
- Retain k best candidate output sequences up to the time we see $< \text{end} >$



[1]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



Encoder

Decoder

[1]Figure: Quoc Le, Google Brain

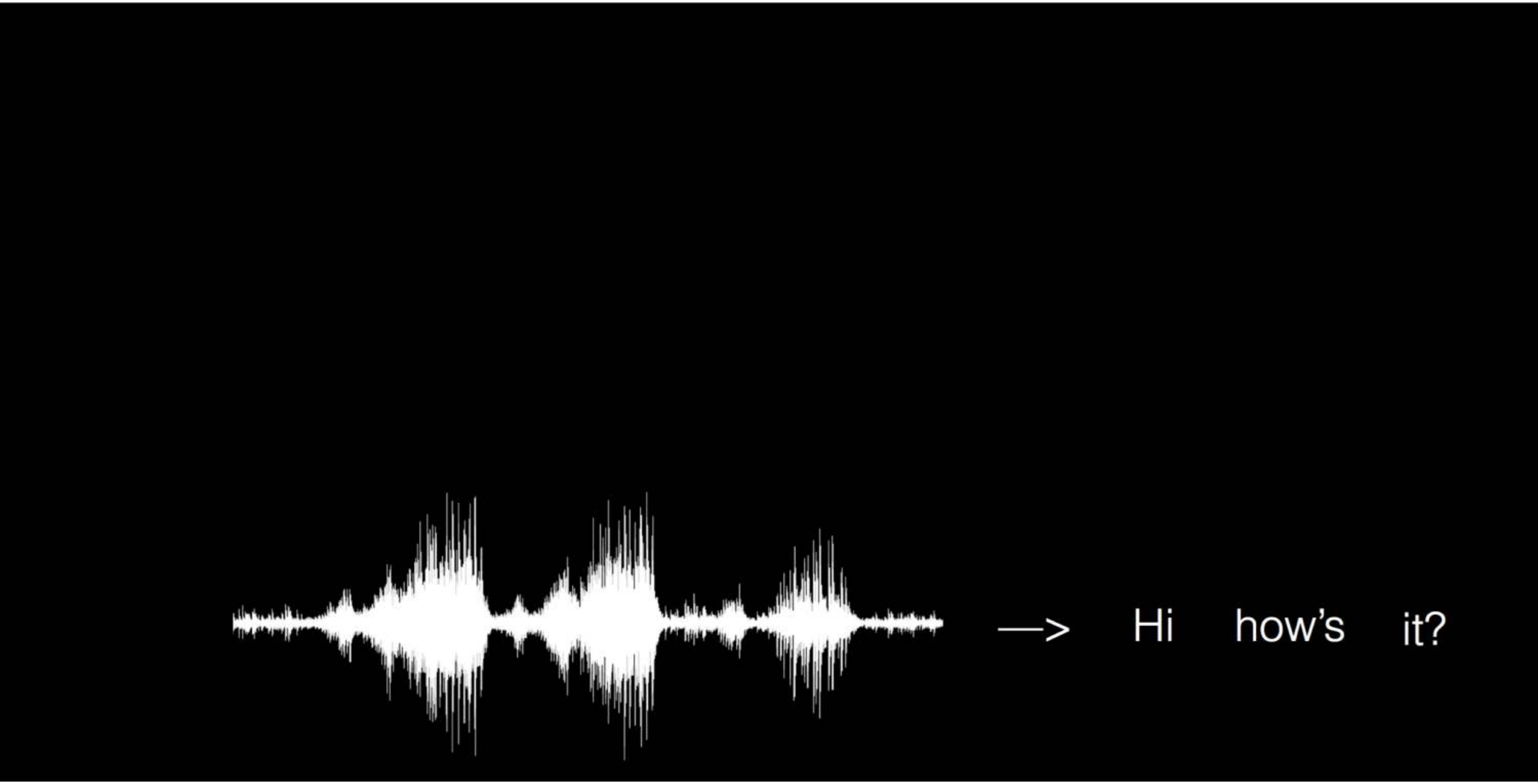
Example IV: Speech Transcription

- Traditional pipeline has
  - Acoustic model $P(output|word)$
  - Language model $P(word)$
  - 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



[1]Figure: Quoc Le, Google Brain

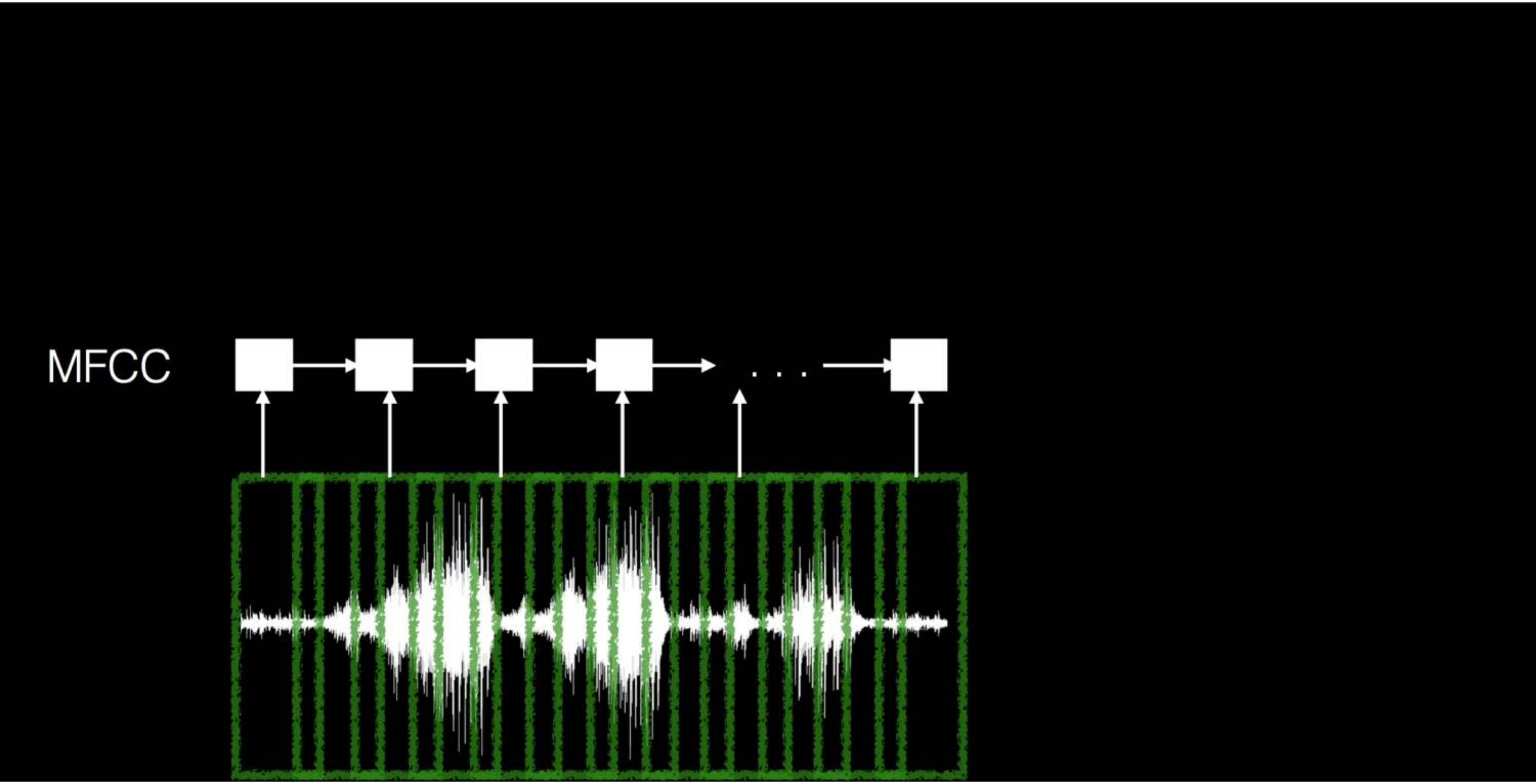
Example IV: Speech Transcription

- Step 1: Get some fixed length vectors

[1]Figure: Quoc Le, Google Brain

Example IV: Speech Transcription

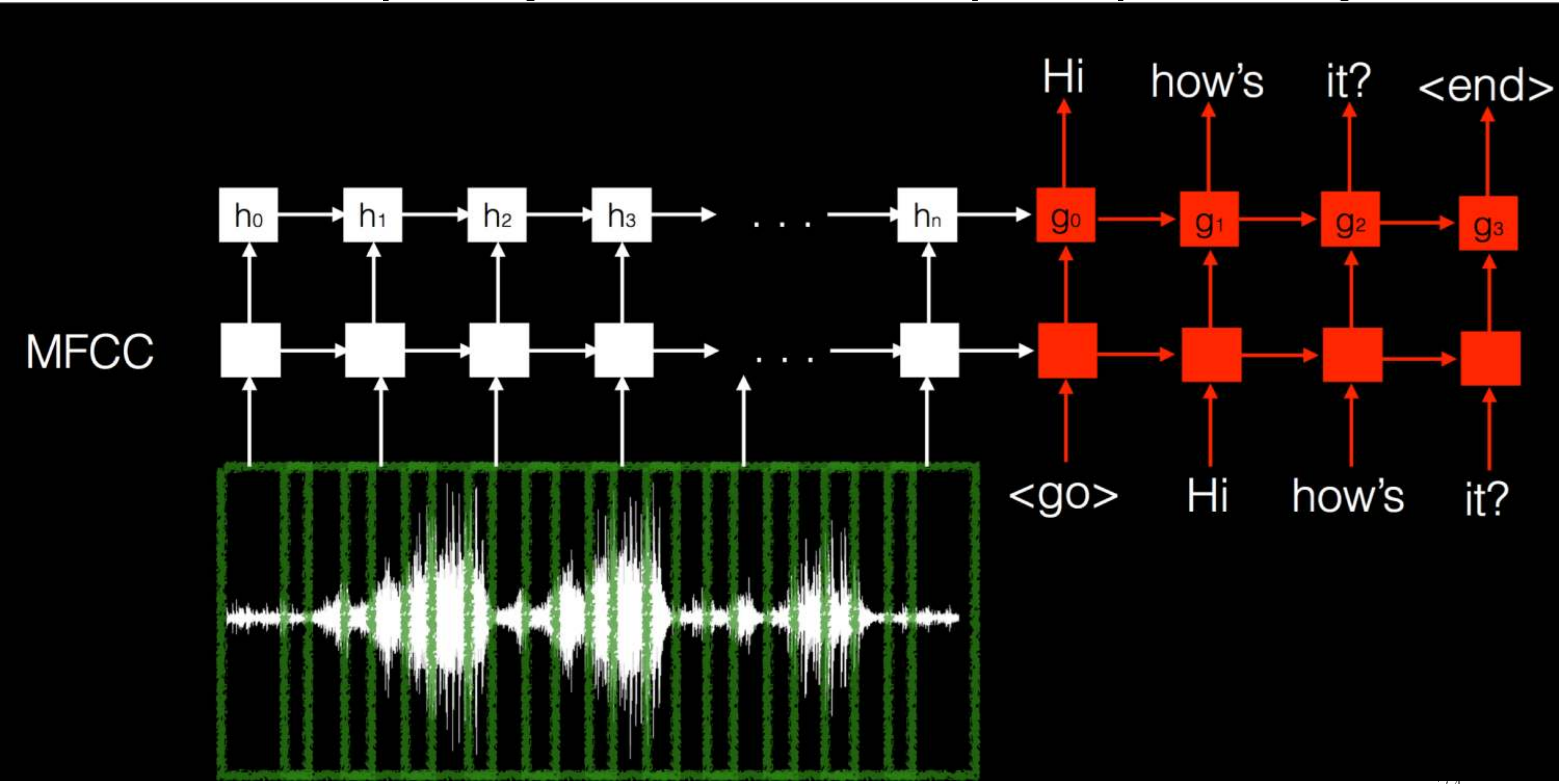
- Step 2: Pass through an encoder



[1]Figure: Quoc Le, Google Brain

Example IV: Speech Transcription

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



[1]Figure: Quoc Le, Google Brain

Questions?

Summary

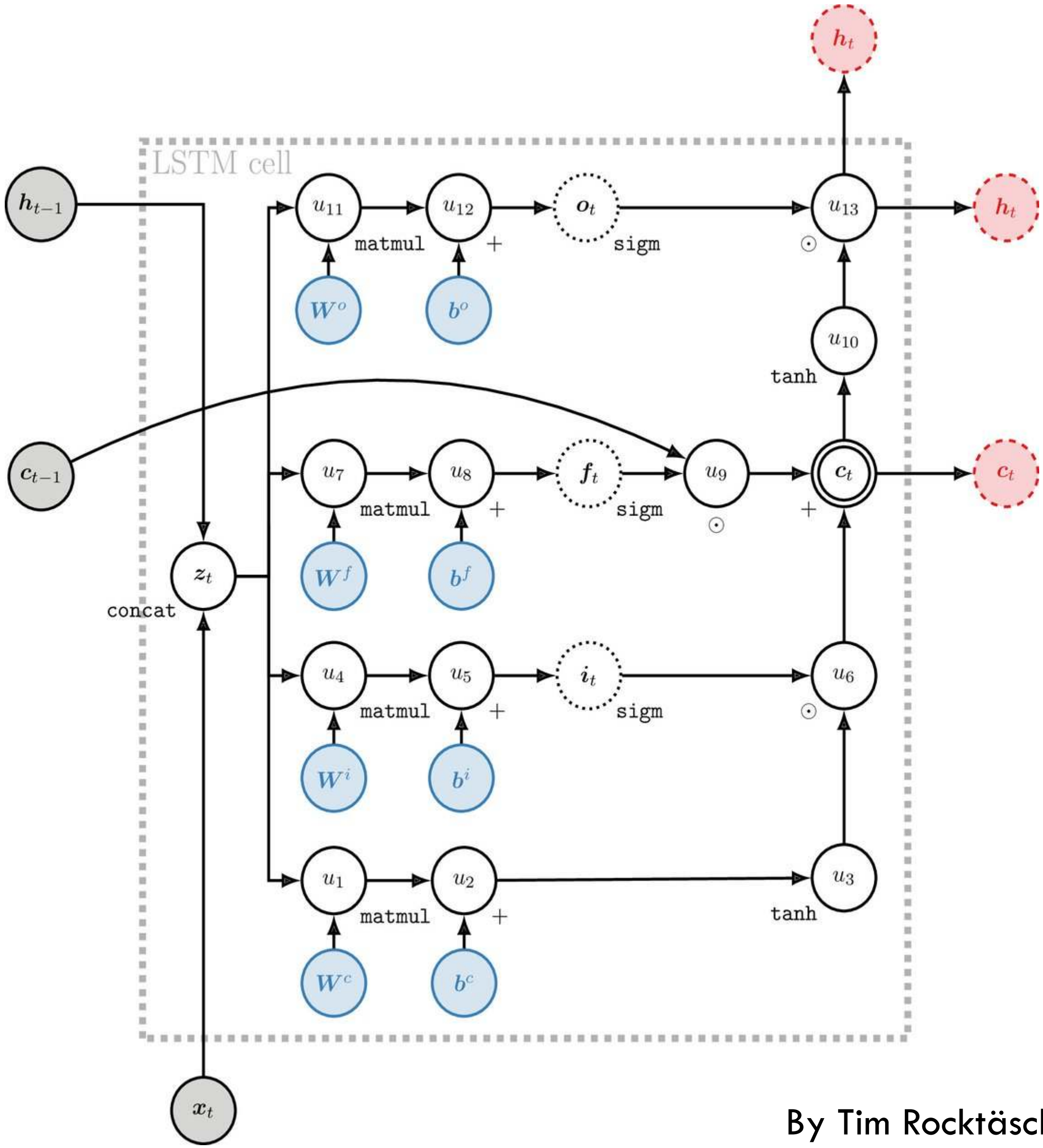
- We motivated when RNNs can be used
- Understood the internal working of RNNs (incl. LSTMs)
- Looked at some details for of ‘sequence to sequence’ applications.
  - These significantly extend beyond classification

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.

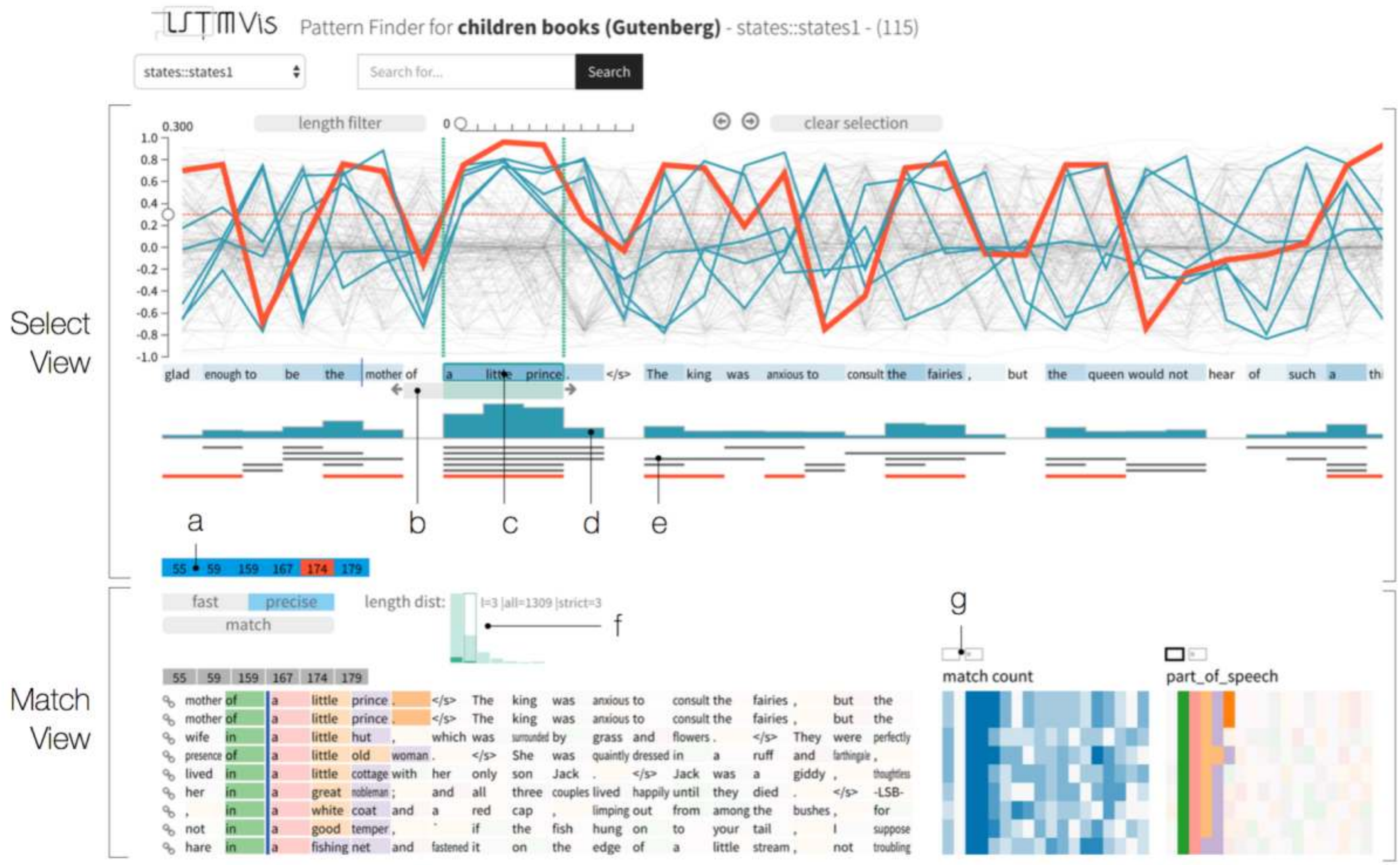
Yet Another Diagram of LSTM



By Tim Rocktäschel

Understanding LSTM: LSTMVis

- A visual tool to see which cell states do what



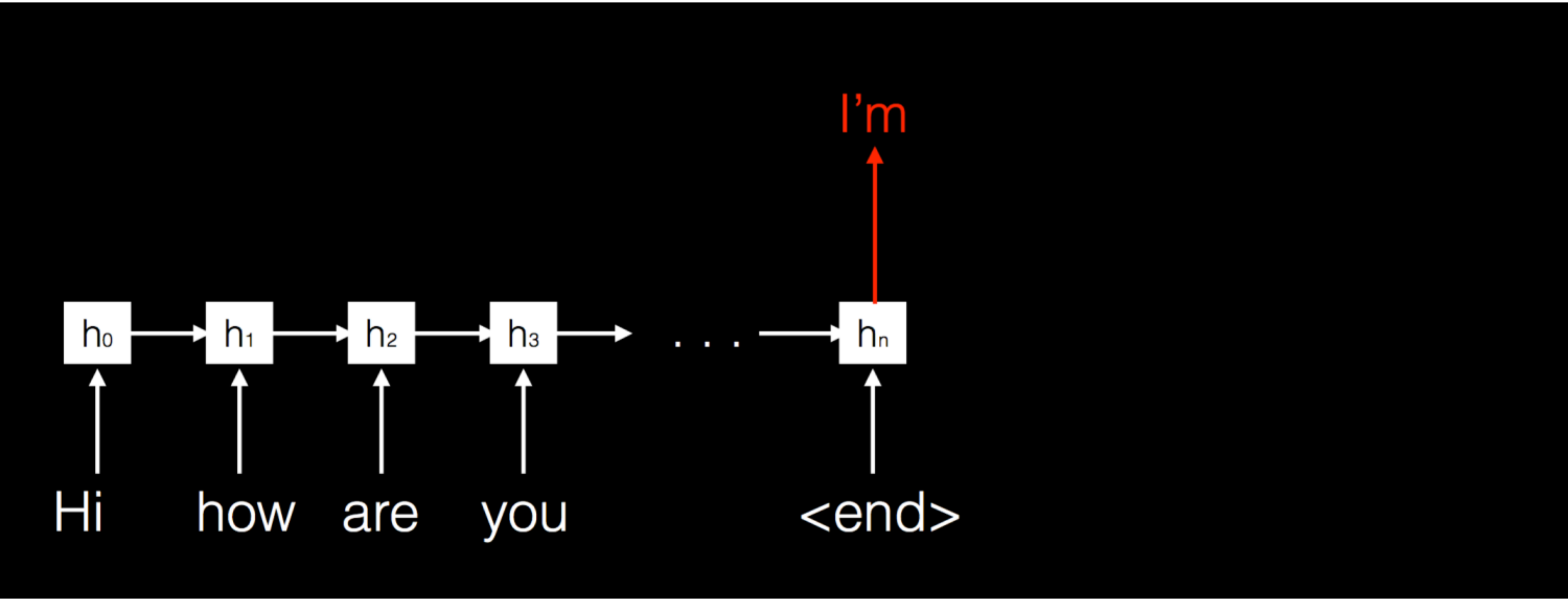
[1]Reference: https://github.com/HendrikStrobelt/LSTMVis

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

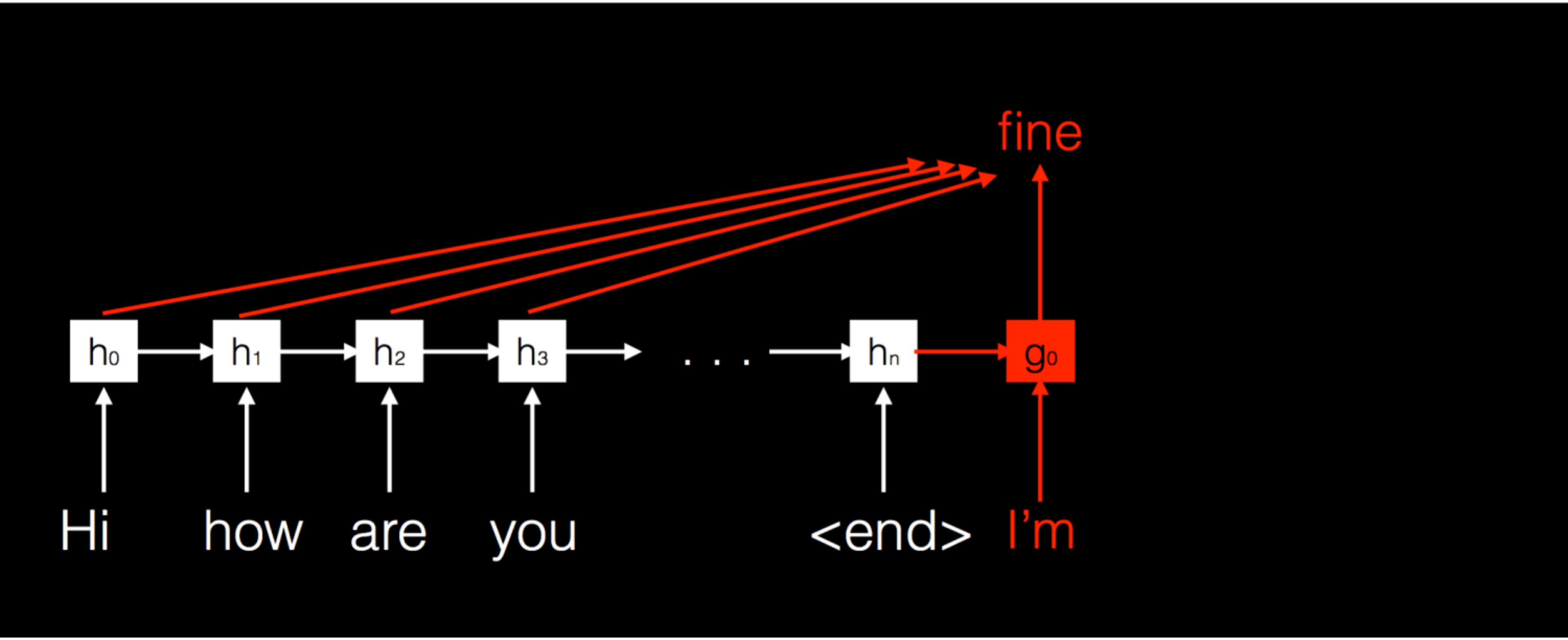
- Third version: Attention Mechanism
- Ideally output could consider 'attention' to parts of history



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

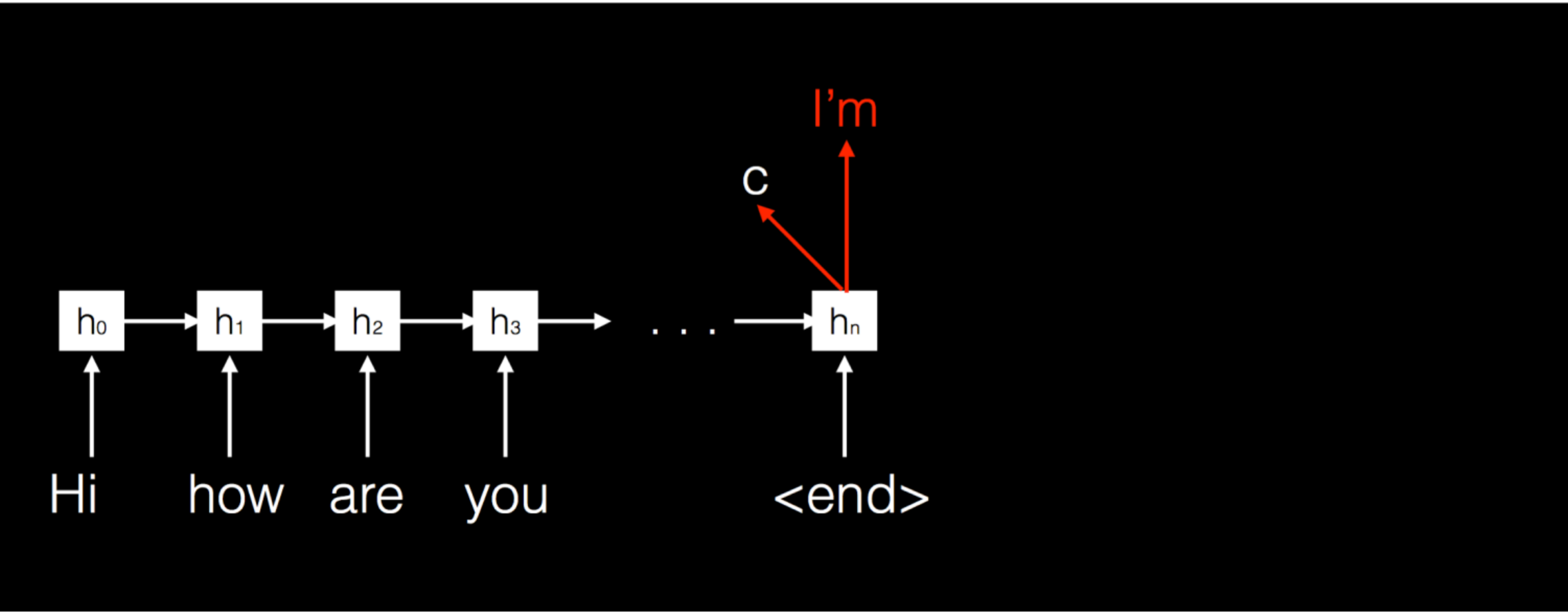
- Could look at every state in the past



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

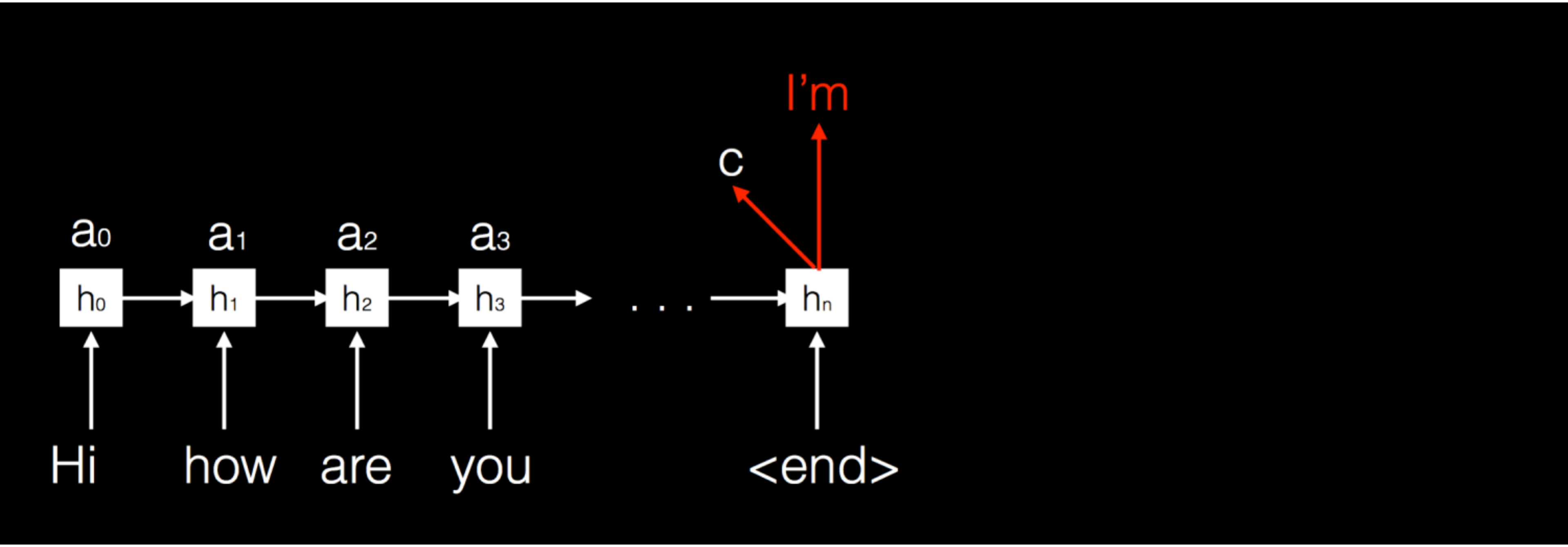
- So instead of returning a word, output the current state



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

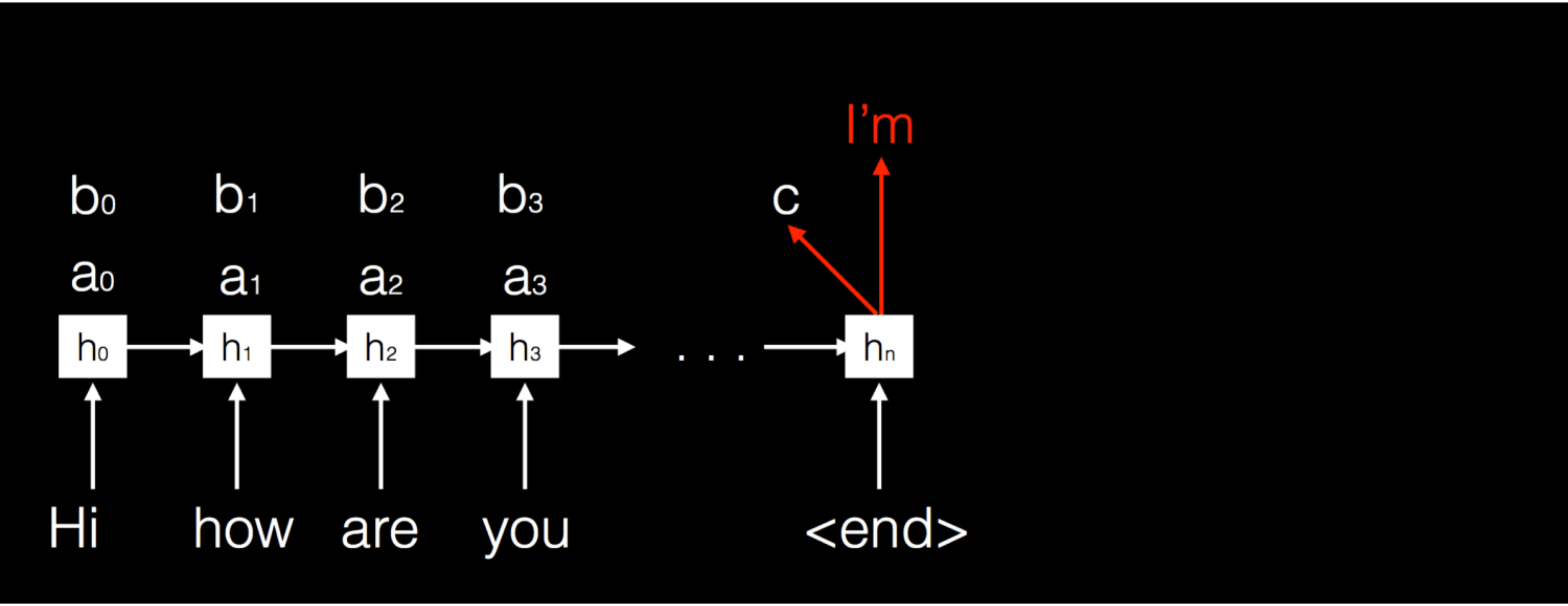
- Take inner products with previous states



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

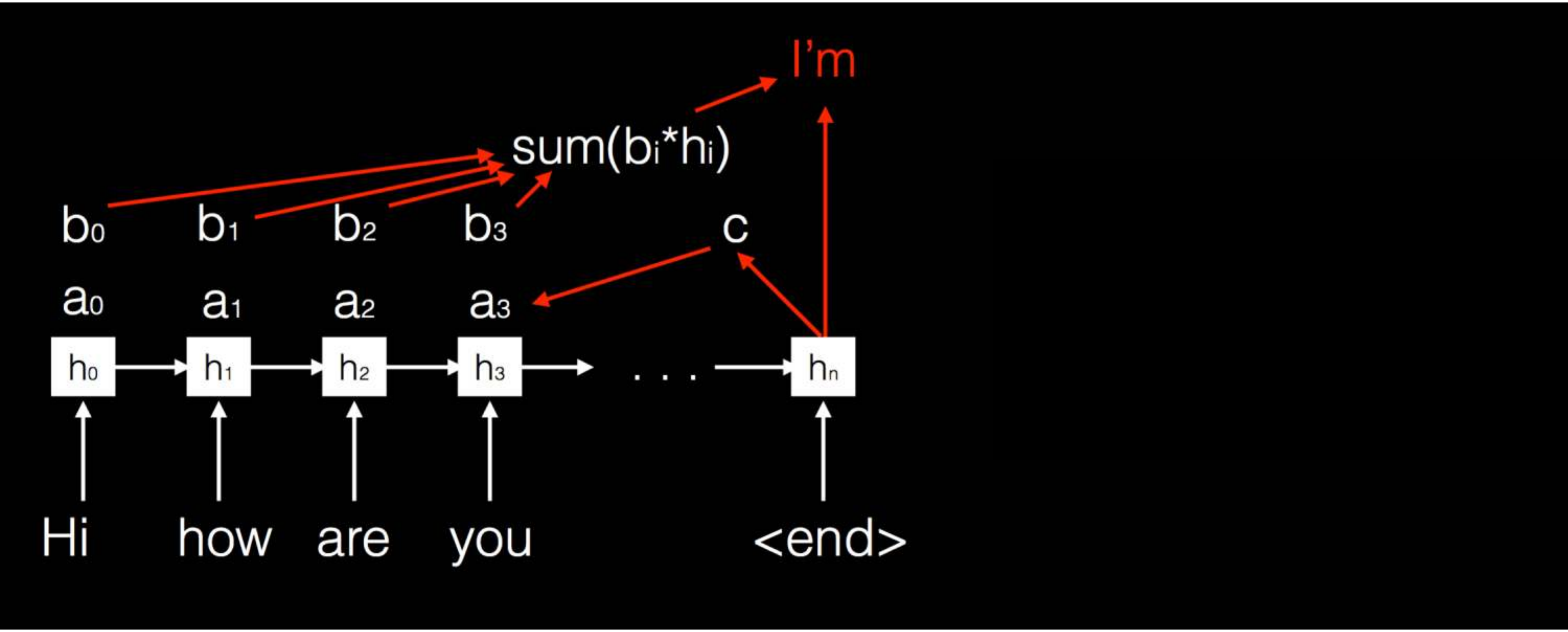
- Take inner products with previous states



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

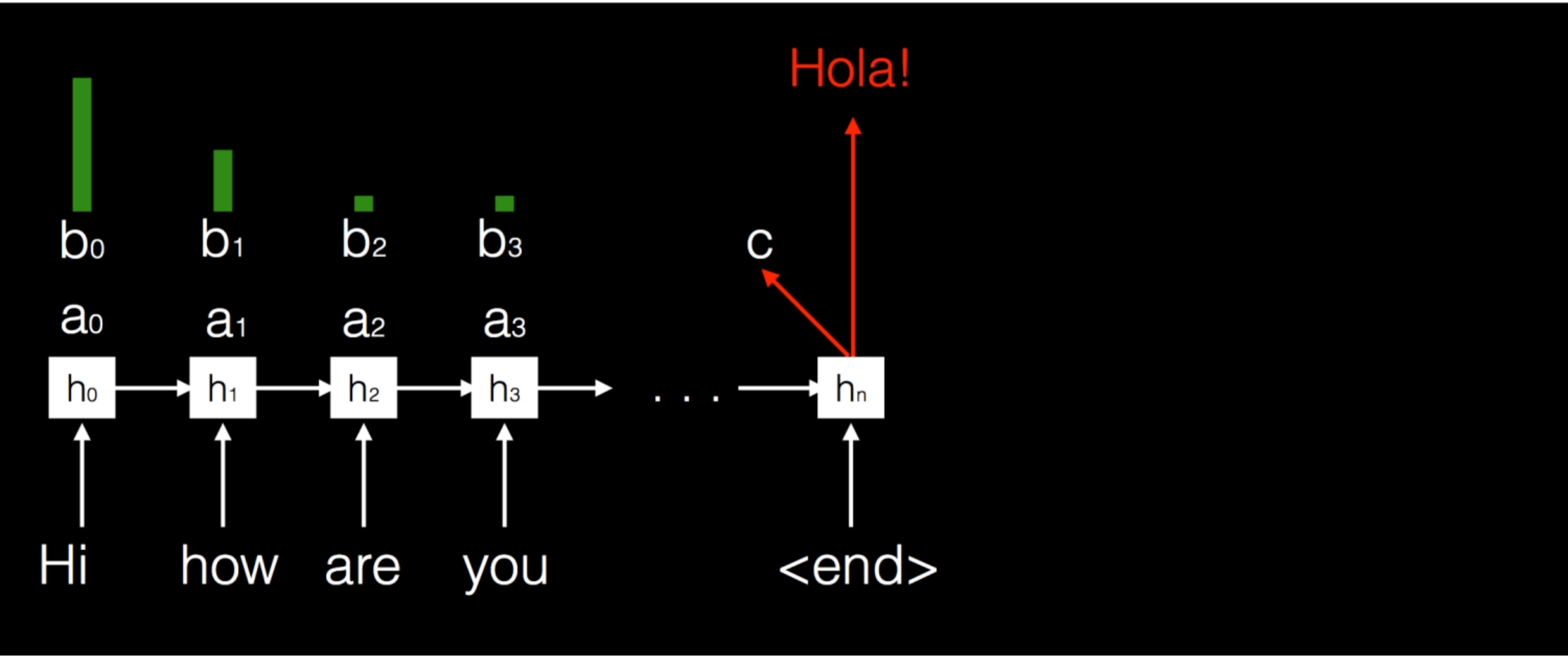
- Pass through a neural net layer to predict final word



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Same with Translation!

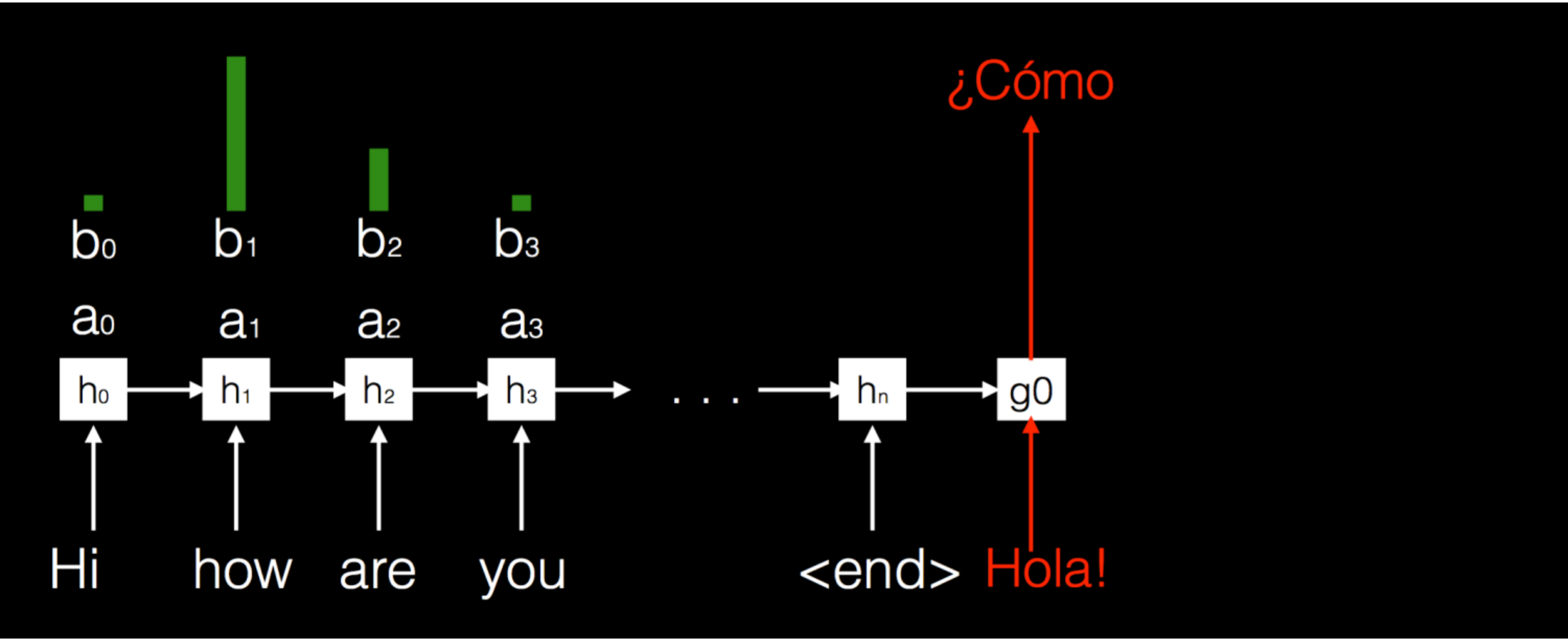
- Same principle also applies for translation. The first prediction learns to focus on certain part of the input



[1]Figure: Quoc Le, Google Brain

Example III (Extension): Auto-Reply

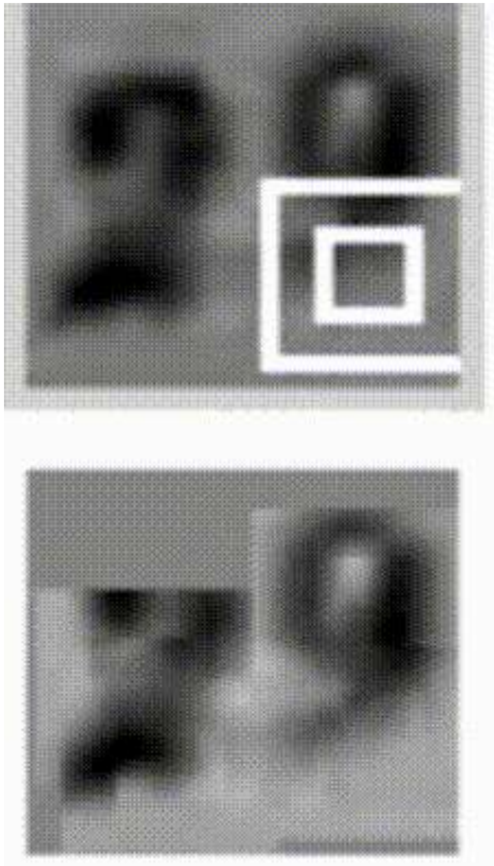
- The second prediction learns to focus on certain part of the input



[1]Figure: Quoc Le, Google Brain

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



[1]Figure: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[2]Reference: http://arxiv.org/abs/1412.7755