SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

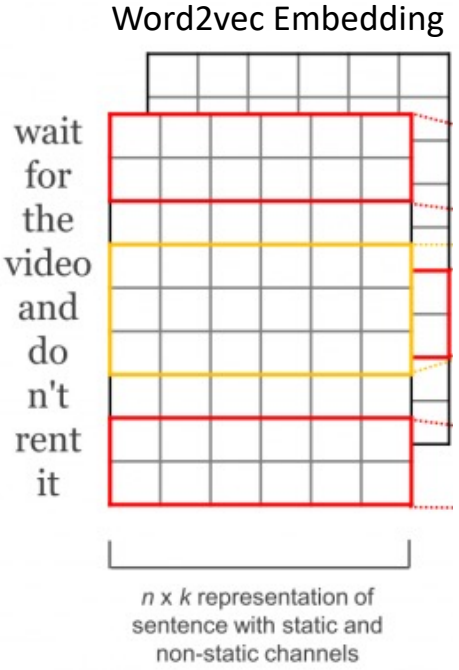
Recurrent Neural Networks (Part I)

Soujanya Poria

50.038 Computational data science

CNN for Text

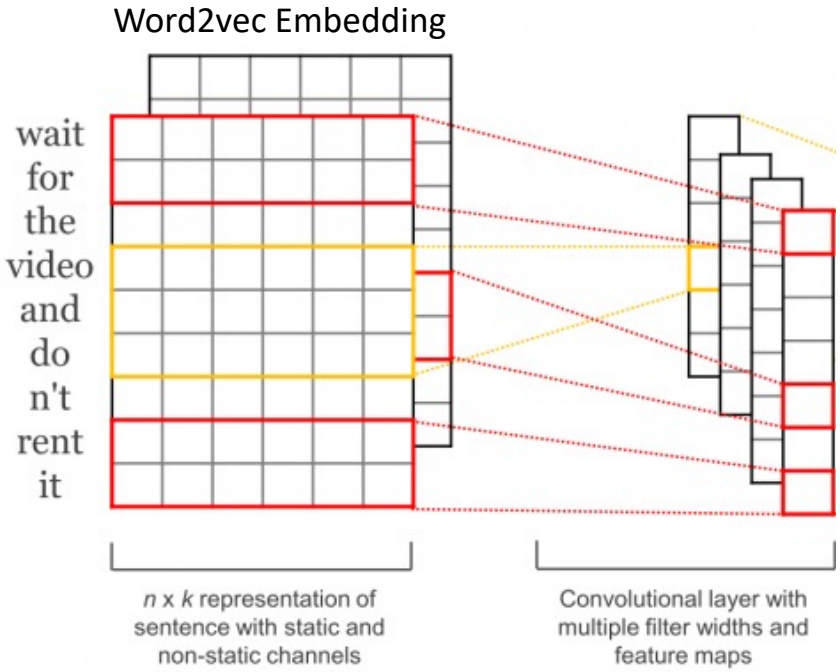
- "Shallow" CNN on sentences represented by word embeddings
  - Sentence represented as *n x k* matrix (*n* words and embedding dimension *k*)



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*(pp. 1746-1751).

CNN for Text

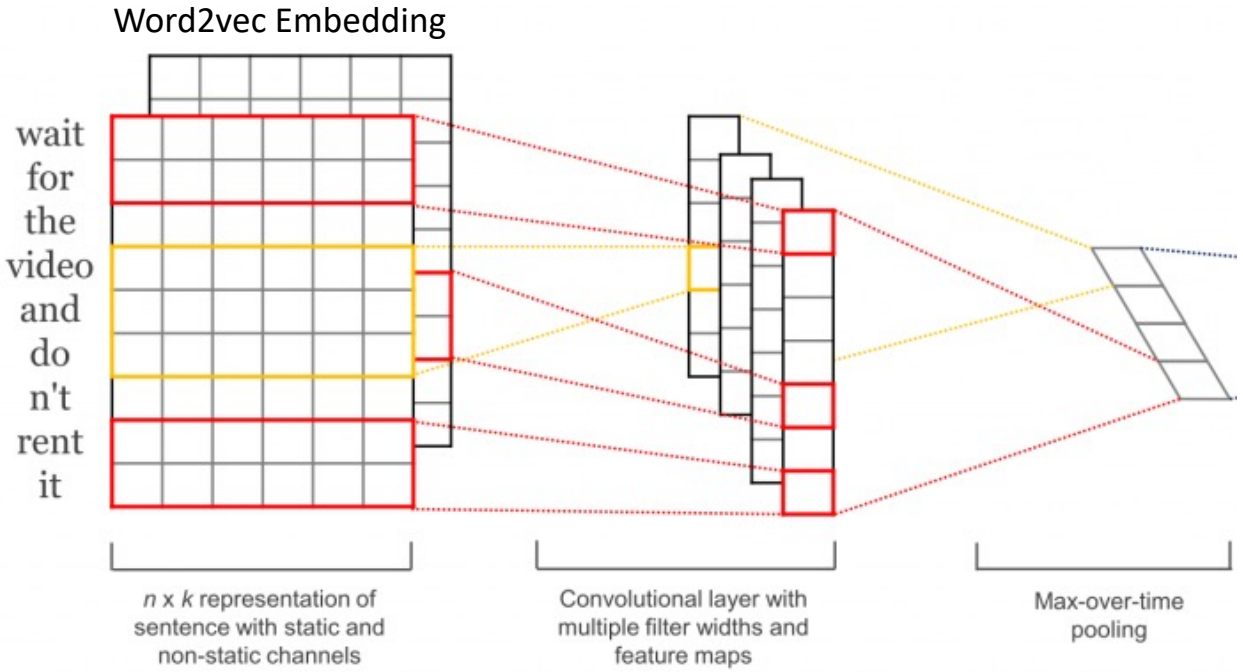
- "Shallow" CNN on sentences represented by word embeddings
  - Apply convolutional filters that cover 2,3,n words at a time (with width *k*)



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*(pp. 1746-1751).

CNN for Text

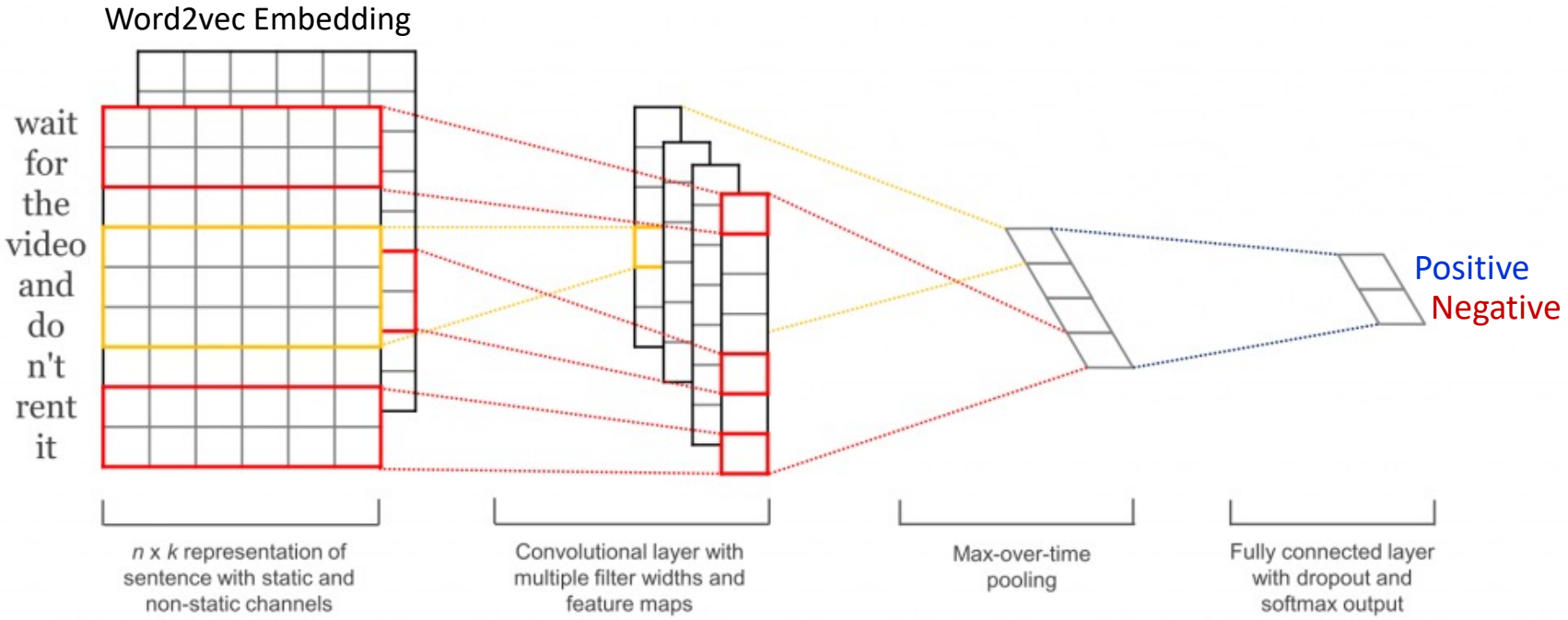
- "Shallow" CNN on sentences represented by word embeddings
  - Apply max pooling to select highest value, and flatten layer



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*(pp. 1746-1751).

CNN for Text

- "Shallow" CNN on sentences represented by word embeddings
  - Final softmax layer to determine most likely class



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*(pp. 1746-1751).

Convolutional neural network for text classification

- The task: Given a textual training data, train a CNN for classification/regression.
- Do you find any similarity with the CNN applied on images?

Convolutional neural network for text classification

- Convert text to sequences

vocabulary - all unique words in a source of text
token - an integer value assigned to each word in the vocabulary

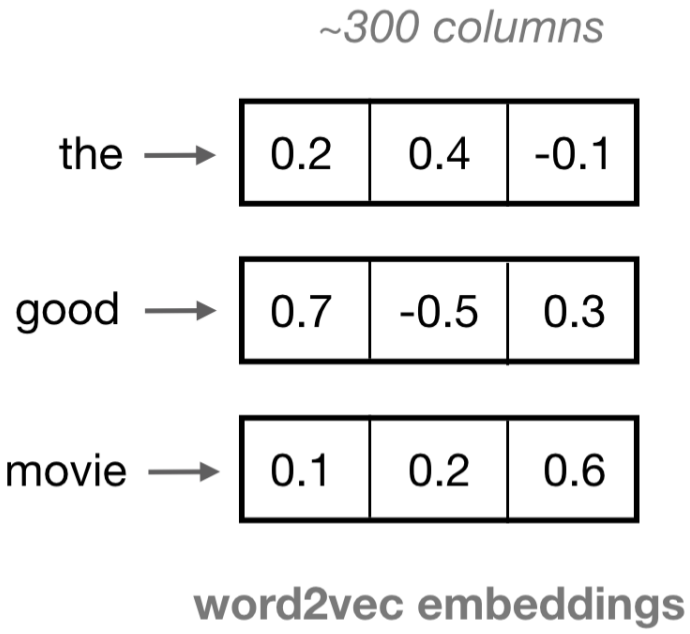
token dictionary

{'the': **0**, 'of': **1**, 'so': **2**, 'then': **3**, 'you': **4**, … 'learn': **3191**, … 'artificial': **30297**… }

sample text		**tokenized text**
"the pettiness of the whole situation"	⟶	`[0, 121241, 1, 0, 988, 25910]`

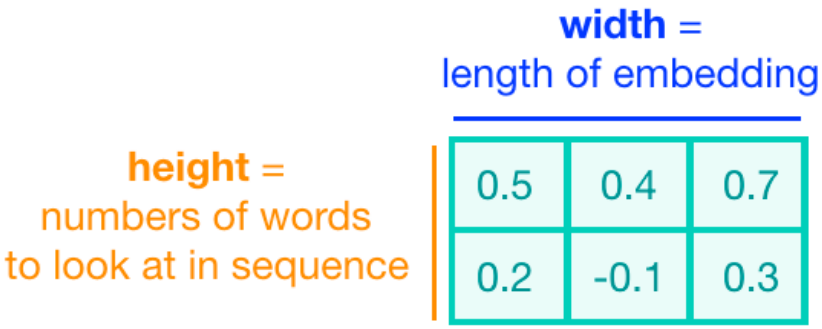
Convolutional neural network for text classification

- Use word embeddings



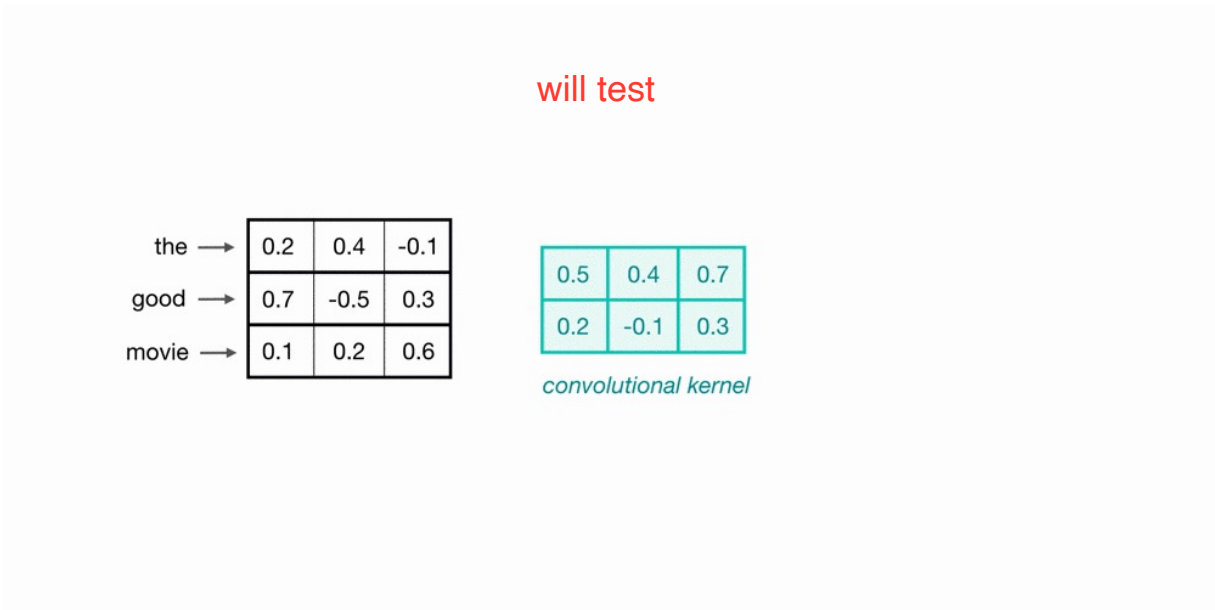
Convolutional neural network for text classification

- Convolutional kernels



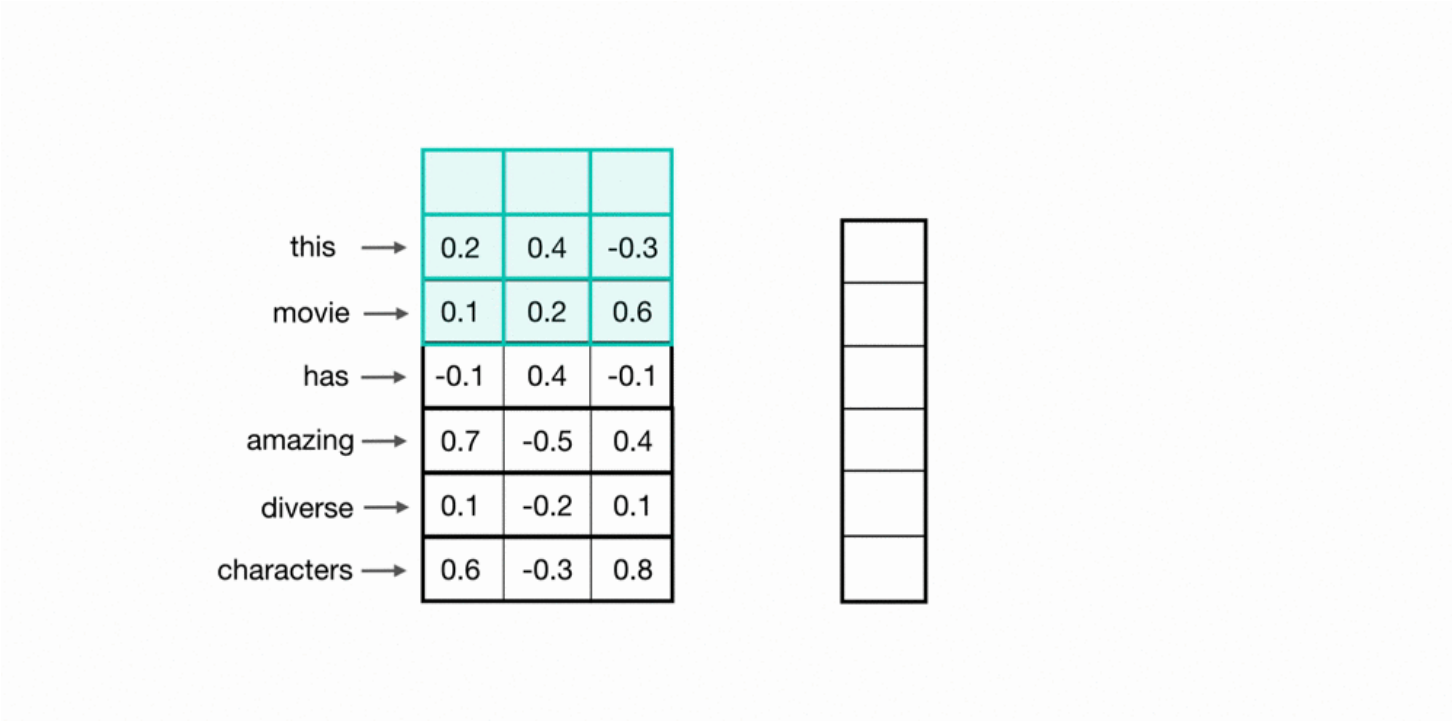
Convolutional neural network for text classification

- Convolution over Word Sequences
  - Example – convolution over Bigrams.



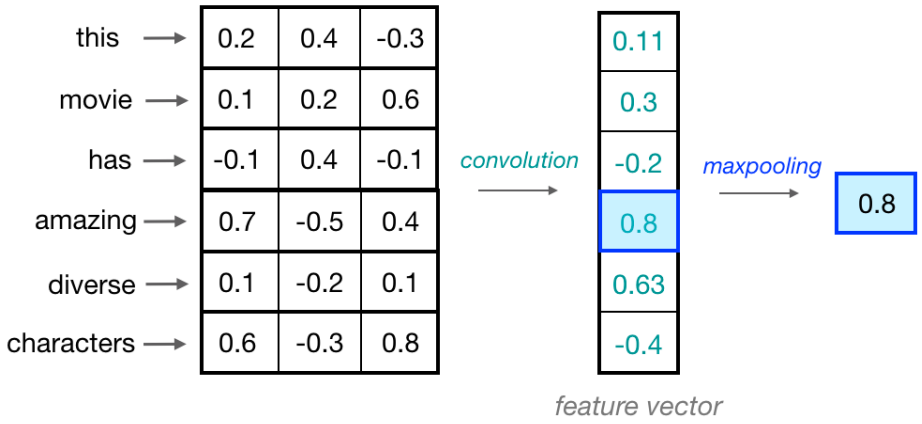
Convolutional neural network for text classification

- Convolution over Word Sequences
  - Example – convolution over trigrams.



Convolutional neural network for text classification

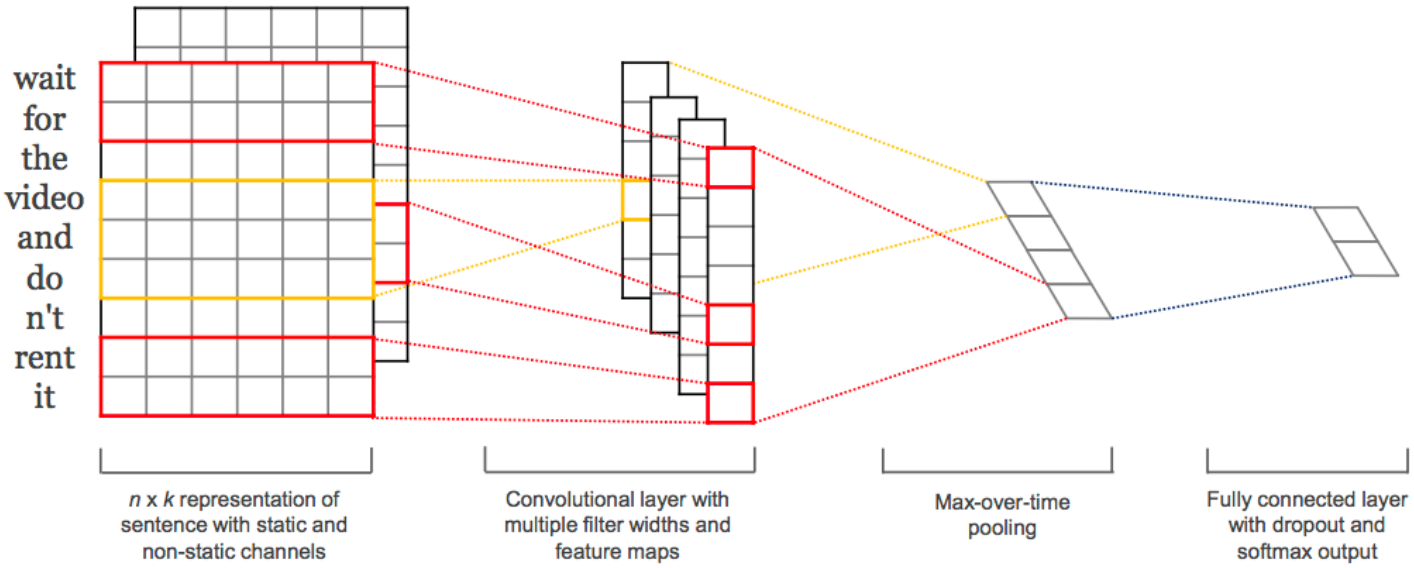
- Maxpool non-linear



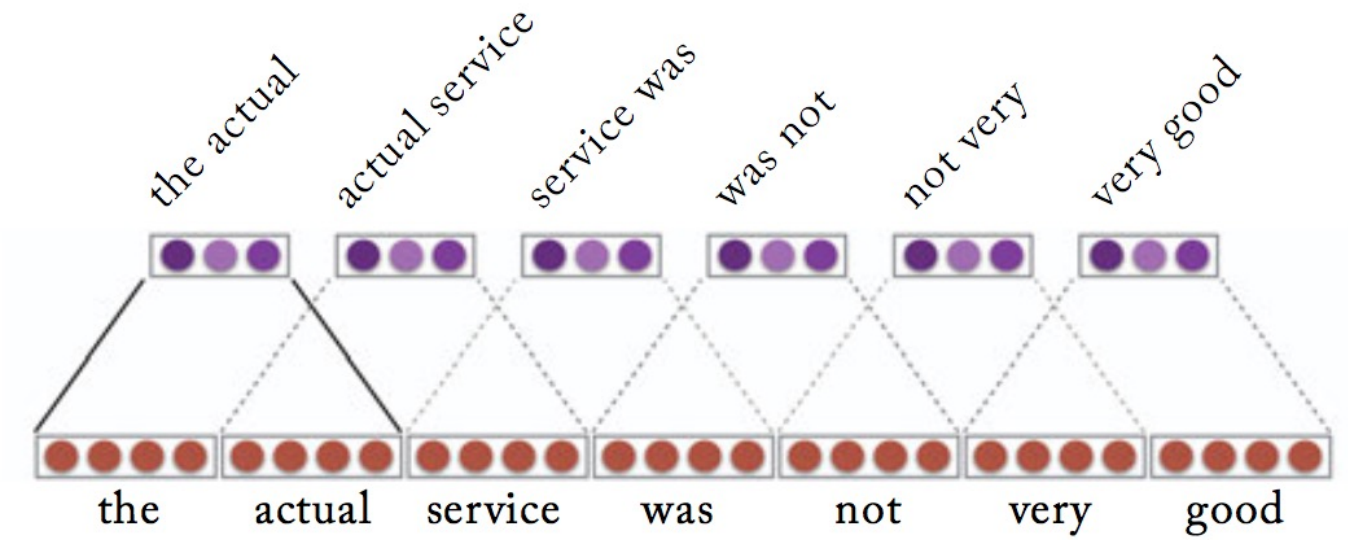
Convolutional neural network for text classification

- The overall network

is a feature extractor, not classification or regression

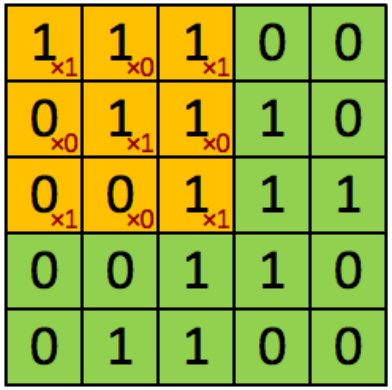


Convolutional neural network as N-gram feature extractor

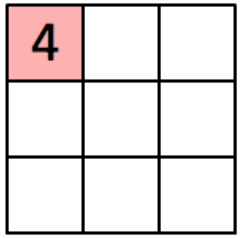


CNN on images vs text

2D convolution



Image



Convolved Feature

1D convolution

the →	0.2	0.4	-0.1
good →	0.7	-0.5	0.3
movie →	0.1	0.2	0.6

0.5	0.4	0.7
0.2	-0.1	0.3

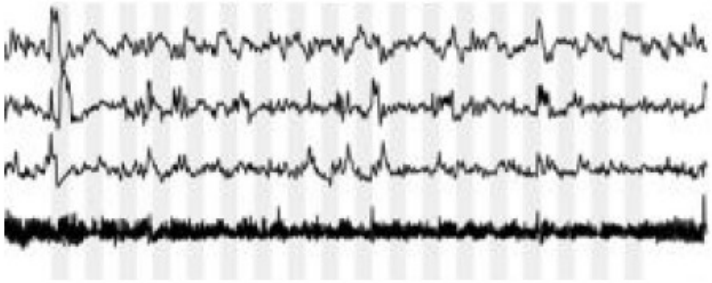
convolutional kernel

Exercise 1: Standard Neural Networks

- So far, we have covered topics on the single perceptron, MLP, and CNN.
- What is the common characteristic and limitation of these neural networks?

Temporal Sequences

"This morning I took the dog for a walk." *Sentences*

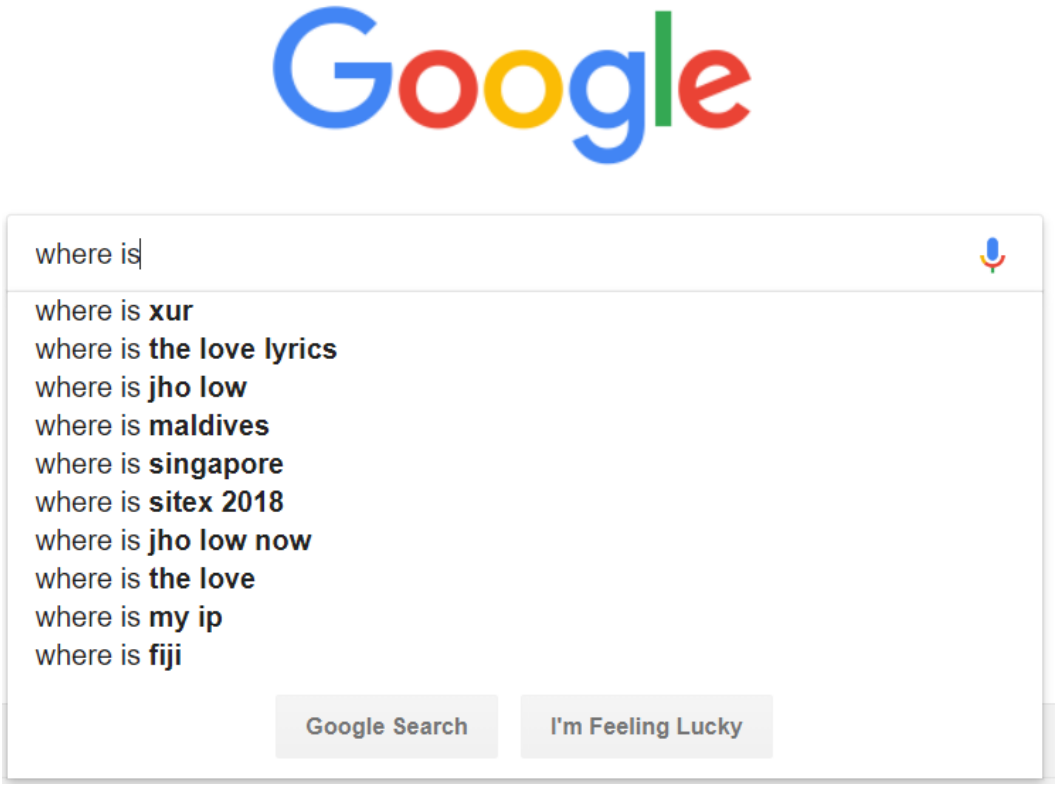


Heart-rates

Audio Recordings

Sequence Modelling Problem

- E.g., Sentence completion



Sequence Modelling Problem

- E.g., Sentence completion

"This morning I took the dog for a walk."

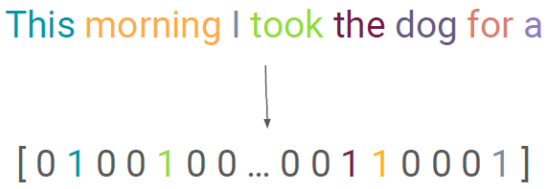
given these words

predict what comes next?

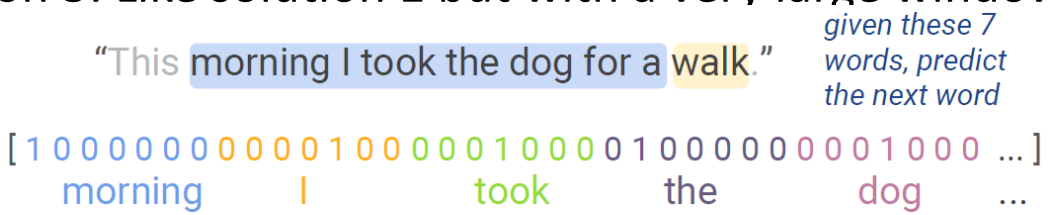
Exercise 2: Approaches to Sequence Modelling Problem

- Using a Sentence Completion example, what are the possible problems?
  - Solution 1

  - Solution 2: Model entire sentence as Bag-of-words



  - Solution 3: Like solution 1 but with a very large window

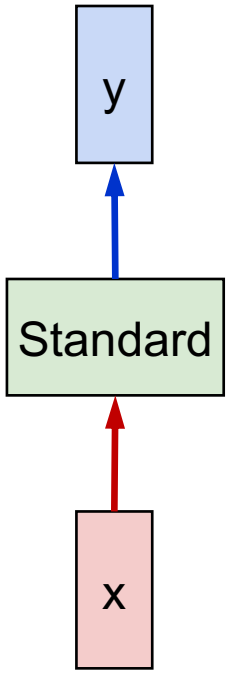


Motivations behind RNN

- Inputs and output may not be of a fixed length
  - E.g., An input paragraph of variable word count
- Want to model temporal aspect (sequence order) as context
  - E.g., language translation or stock prediction
- Hard to determine appropriate window size for context
  - E.g., past 3 days VS 6 months, previous word VS last 8^{th} word
- Want to share parameters across sequence
  - E.g, patterns that appear in different parts of the temporal sequence

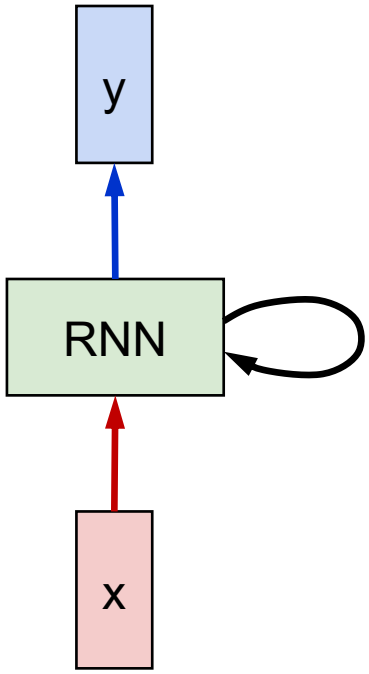
Recurrent Neural Networks

- Standard Neural Networks generate a single output



Recurrent Neural Networks

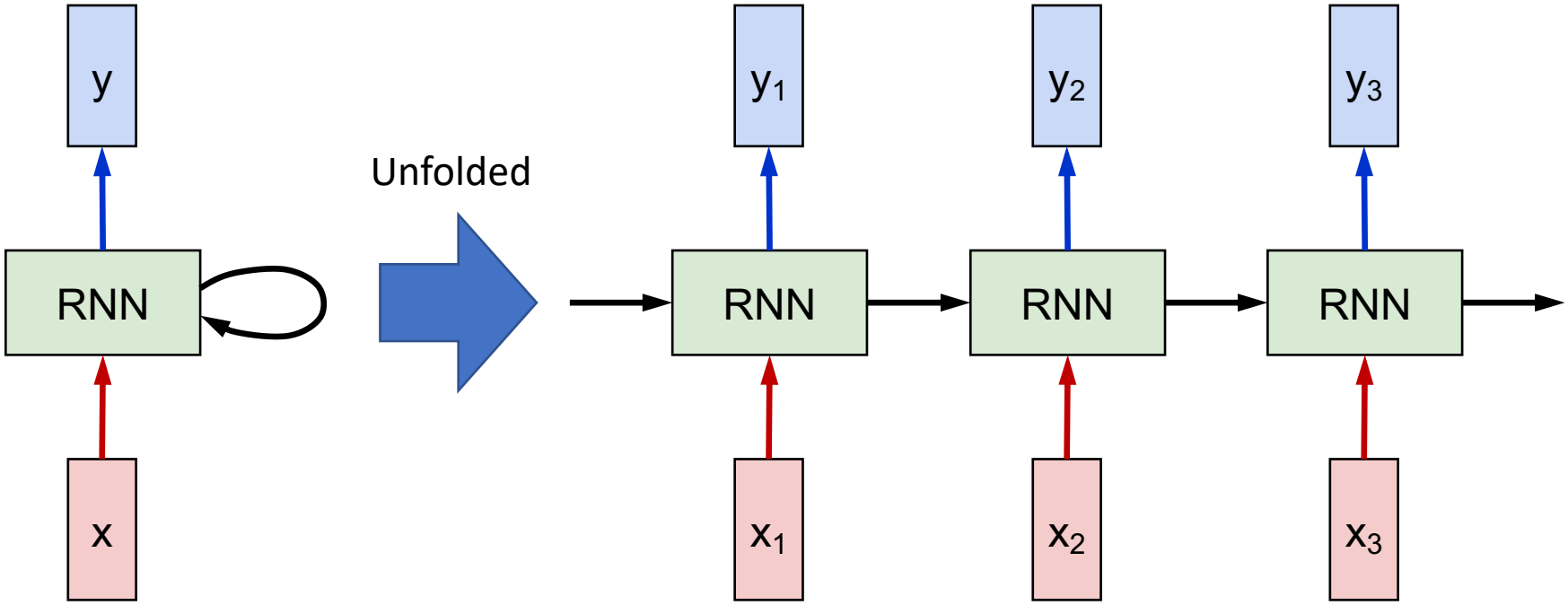
- Recurrent Neural Networks aim to process a sequence of inputs to generate a sequence of outputs at different time steps



Recurrent Neural Networks

memory, pass to next time step

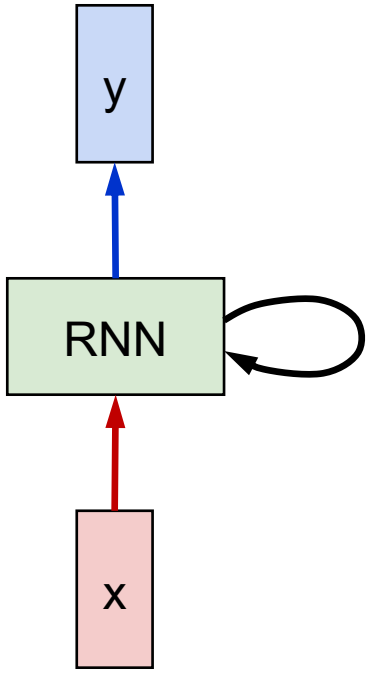
- Recurrent Neural Networks aim to process a sequence of inputs to generate a sequence of outputs at different time steps



Recurrent Neural Networks

- RNNs process a sequence of inputs *X* using a recurrence formula at each time-step *t*

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



Recurrent Neural Networks

- RNNs process a sequence of inputs *X* using a recurrence formula at each time-step *t*
  - Each time-step *t* depends on its previous time-step *t-1*

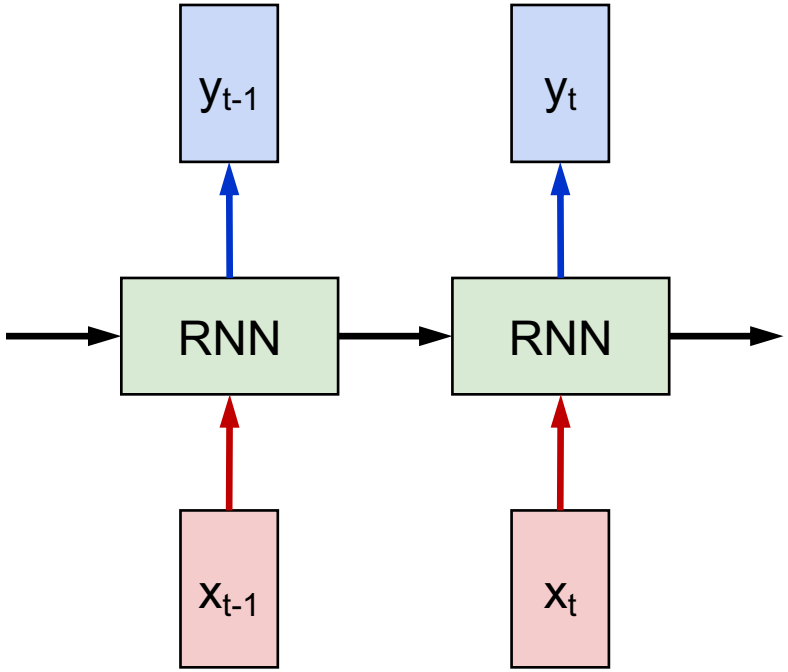
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



Recurrent Neural Networks

- RNNs process a sequence of inputs *X* using a recurrence formula at each time-step *t*
  - The same function f_W and parameters *W* are shared across time-steps

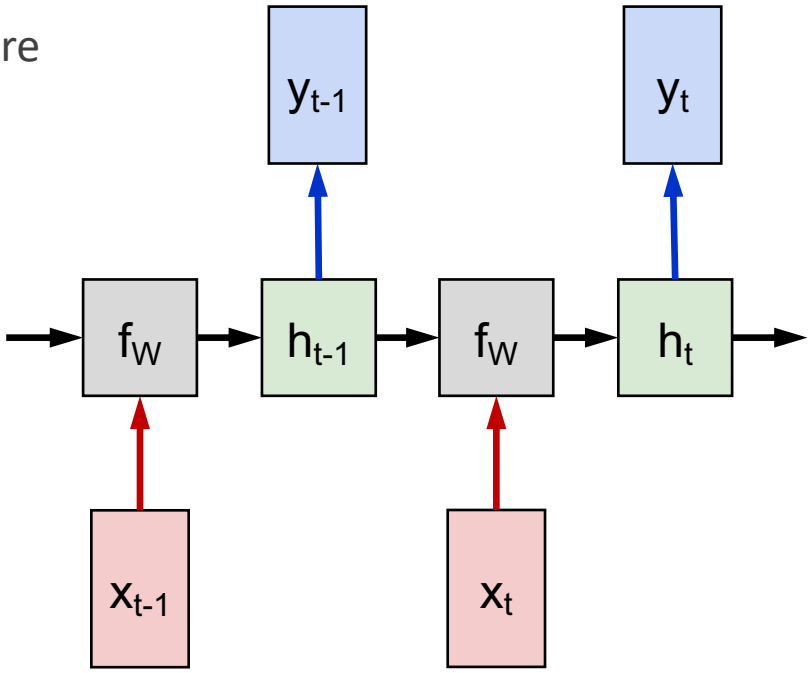
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



Recurrent Neural Networks

- RNNs process a sequence of inputs *X* using a recurrence formula at each time-step *t*
  - The same function f_W and parameters *W* are shared across time-steps

makes the model linear w constraints

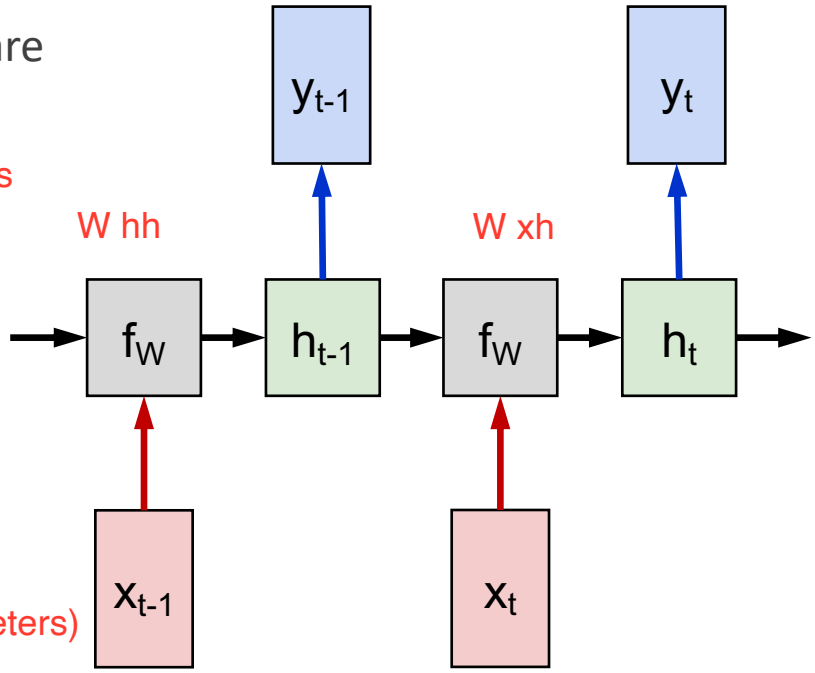
$$h_t = f_W(h_{t-1}, x_t)$$

non-linear function tahn

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

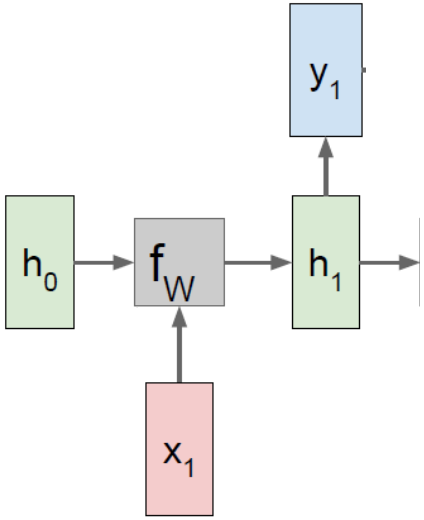
$$y_t = W_{hy}h_t$$

can take any dimensional input
+ hidden state size (hyperparameters)



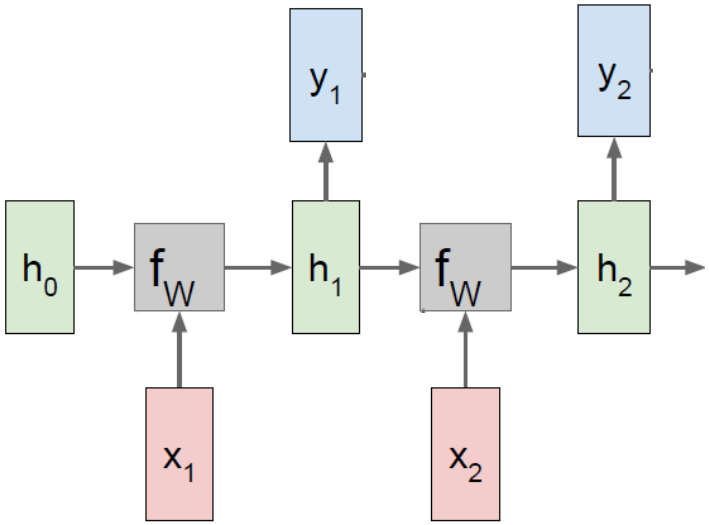
RNN Example

- Start from time-step 1
  - Output y_1,



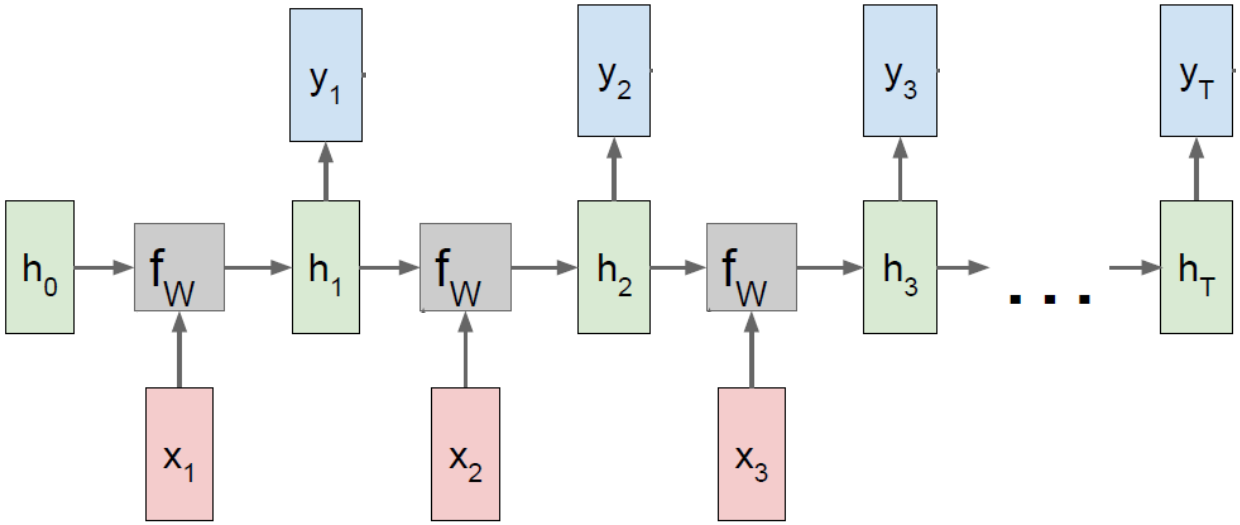
RNN Example

- Start from time-step 1, then time-step 2,
  - Output y_1, y_2 generated



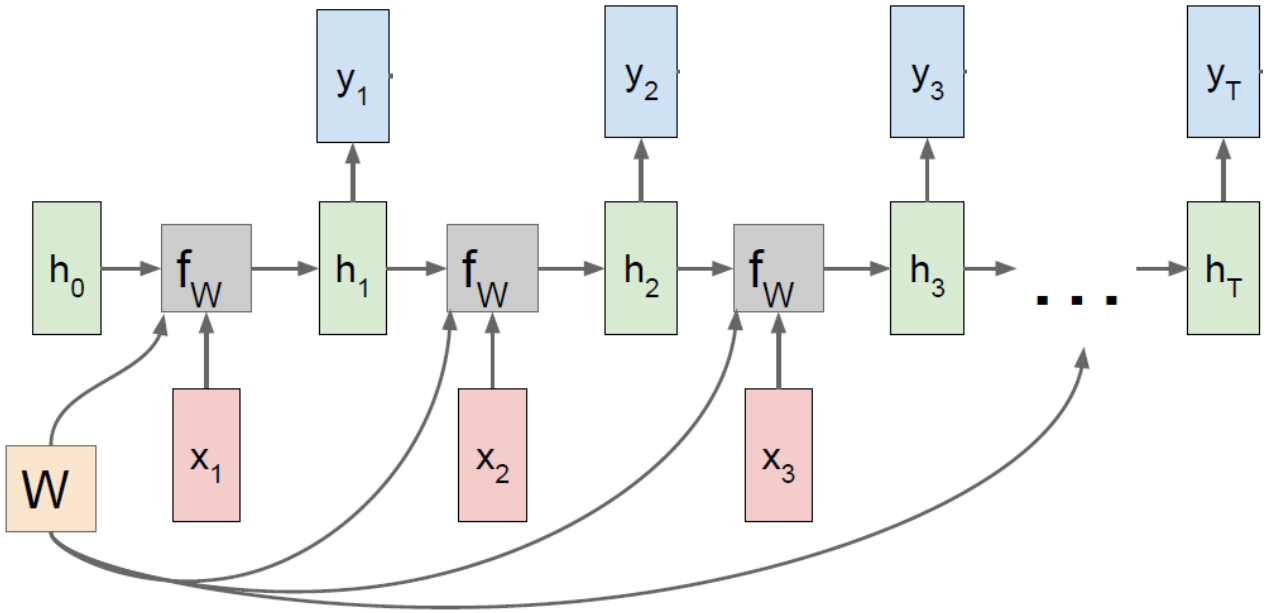
RNN Example

- Start from time-step 1, then time-step 2, until time-step *T*
  - Output y_1, y_2, …, y_T generated



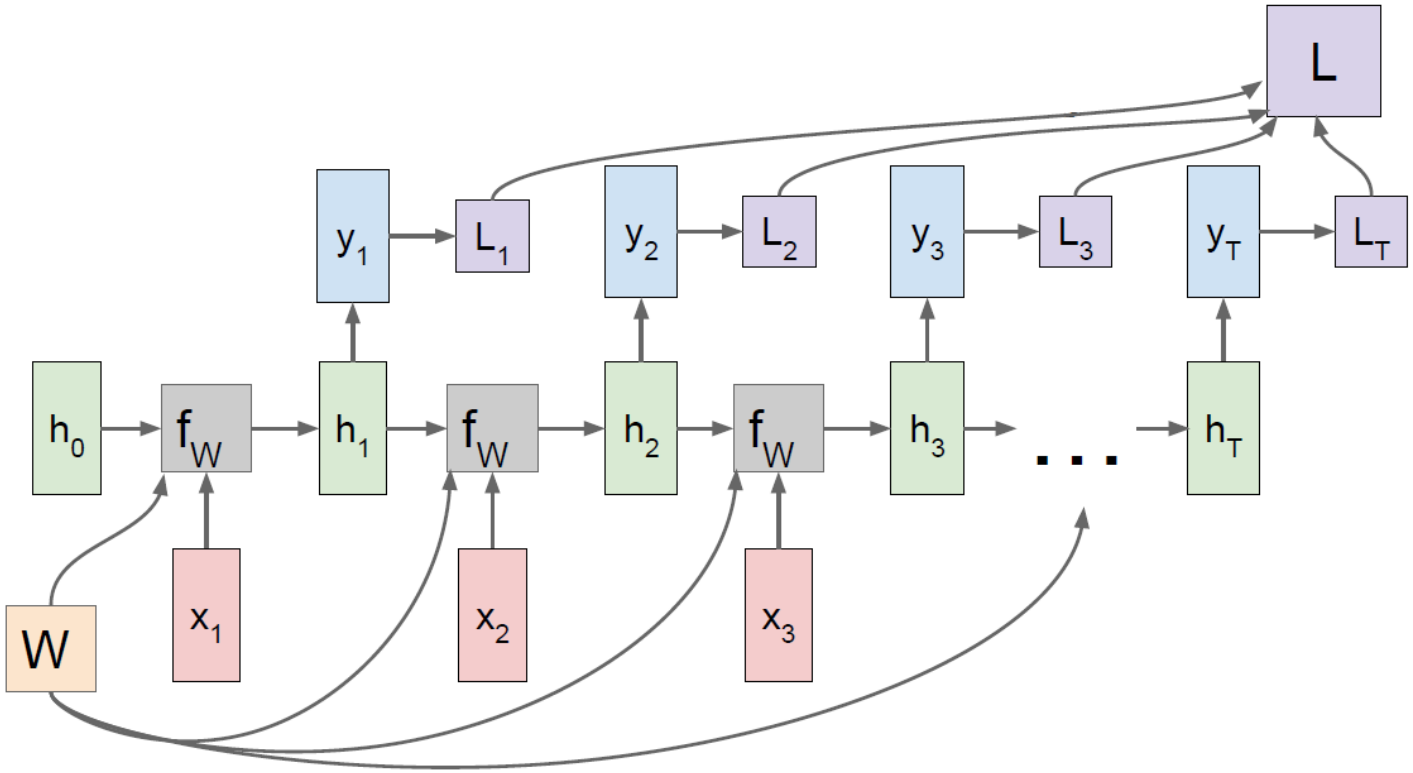
RNN Example

- Same weights *W* used throughout all time-steps



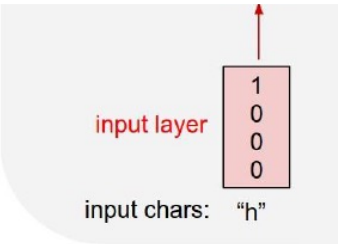
RNN Example

- Same weights *W* used throughout all time-steps
  - Learn weights by minimizing overall loss *L* from all time-steps



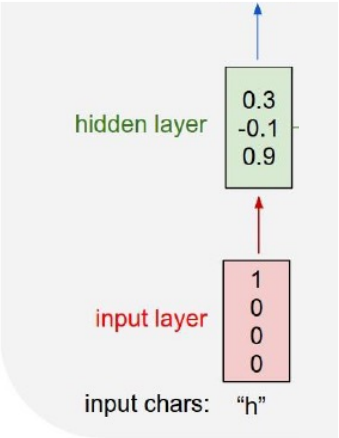
RNN Example: Character-level

- Problem: Trying to predict a sequence of characters
  - Assuming a vocabulary of [h,e,l,o]



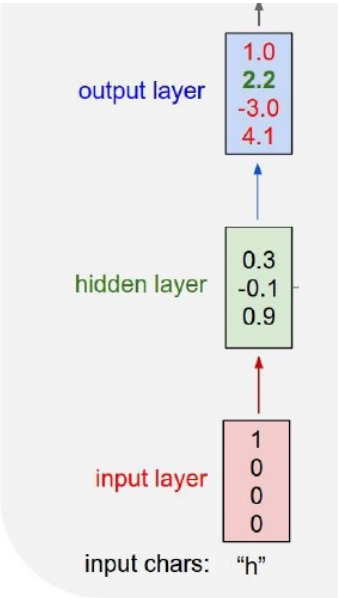
RNN Example: Character-level

- Problem: Trying to predict a sequence of characters
  - Assuming a vocabulary of [h,e,l,o]



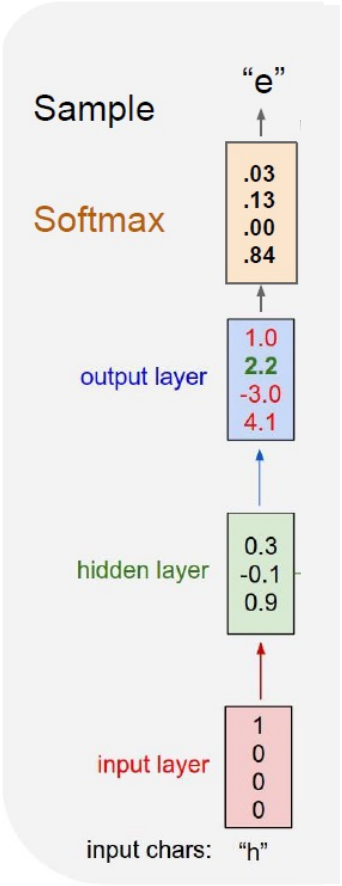
RNN Example: Character-level

- Problem: Trying to predict a sequence of characters
  - Assuming a vocabulary of [h,e,l,o]



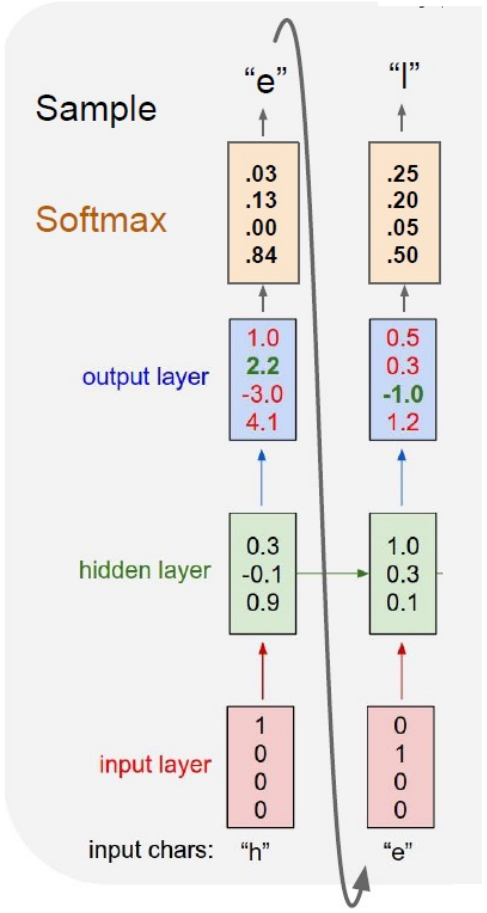
RNN Example: Character-level

- Problem: Trying to predict a sequence of characters
  - Assuming a vocabulary of [h,e,l,o]



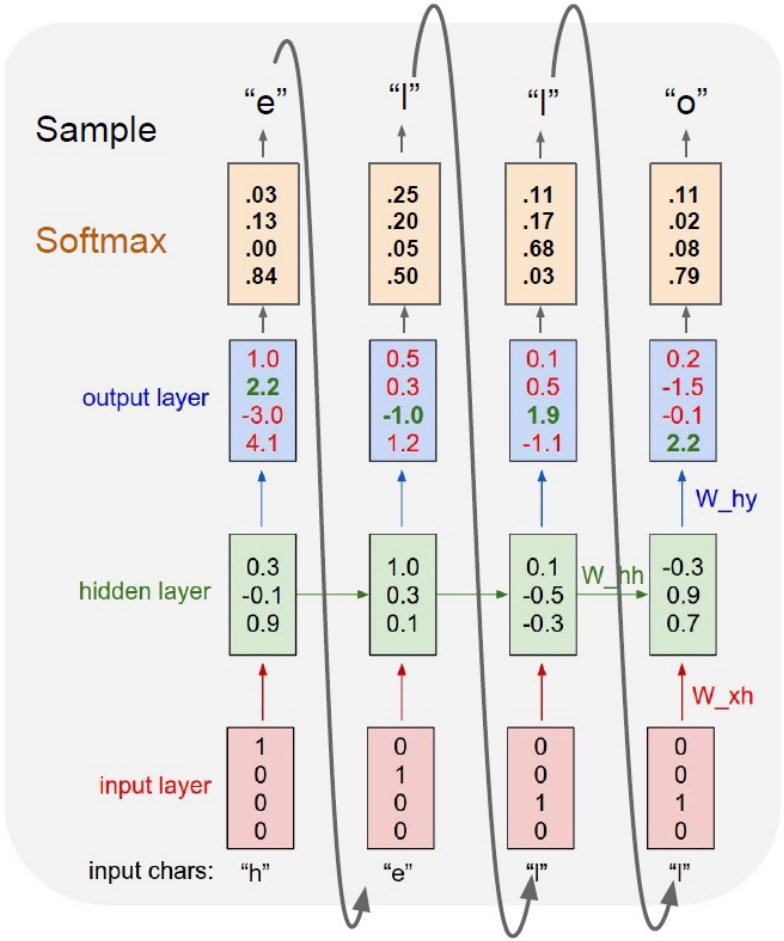
RNN Example: Character-level

- Problem: Trying to predict a sequence of characters
  - Assuming a vocabulary of [h,e,l,o]



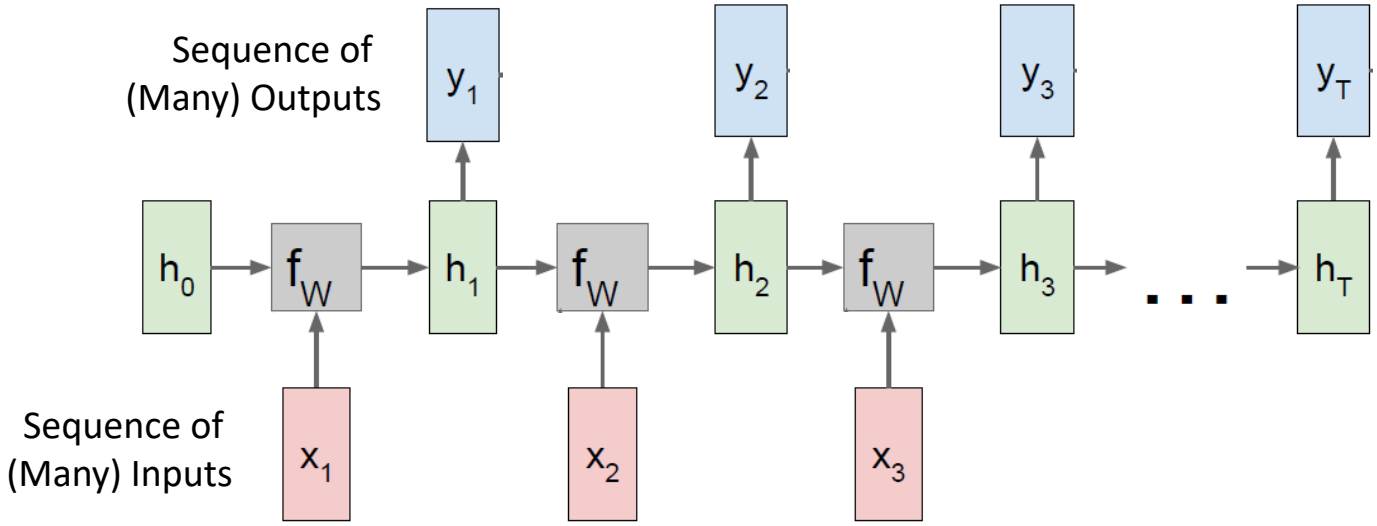
RNN Example: Character-level

- Problem: Trying to predict a sequence of characters
  - Assuming a vocabulary of [h,e,l,o]



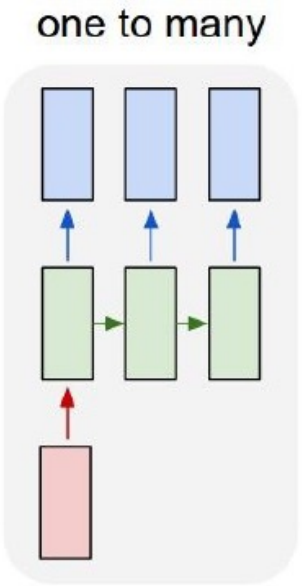
Exercise 3: Types of RNN

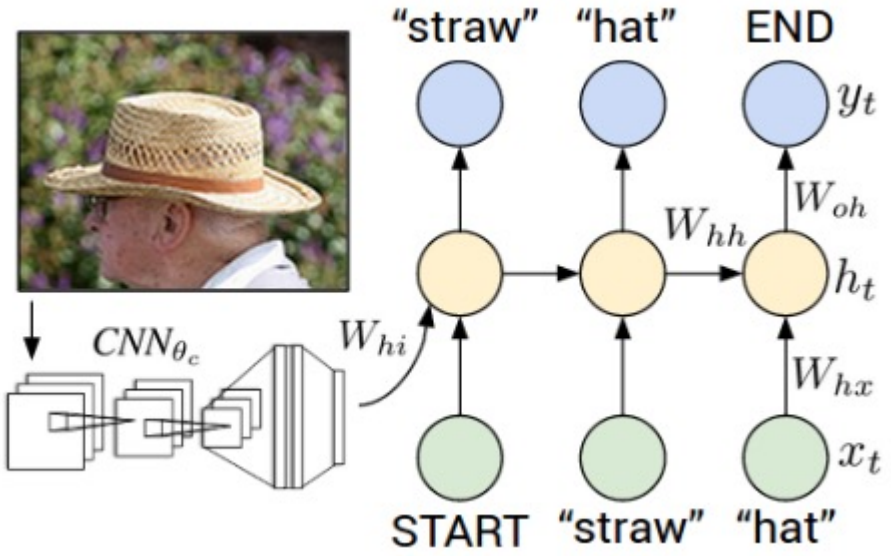
- Based on the input/output sequence, we have previously examined a many-to-many type of RNN. What other types can you think of?



RNN Applications

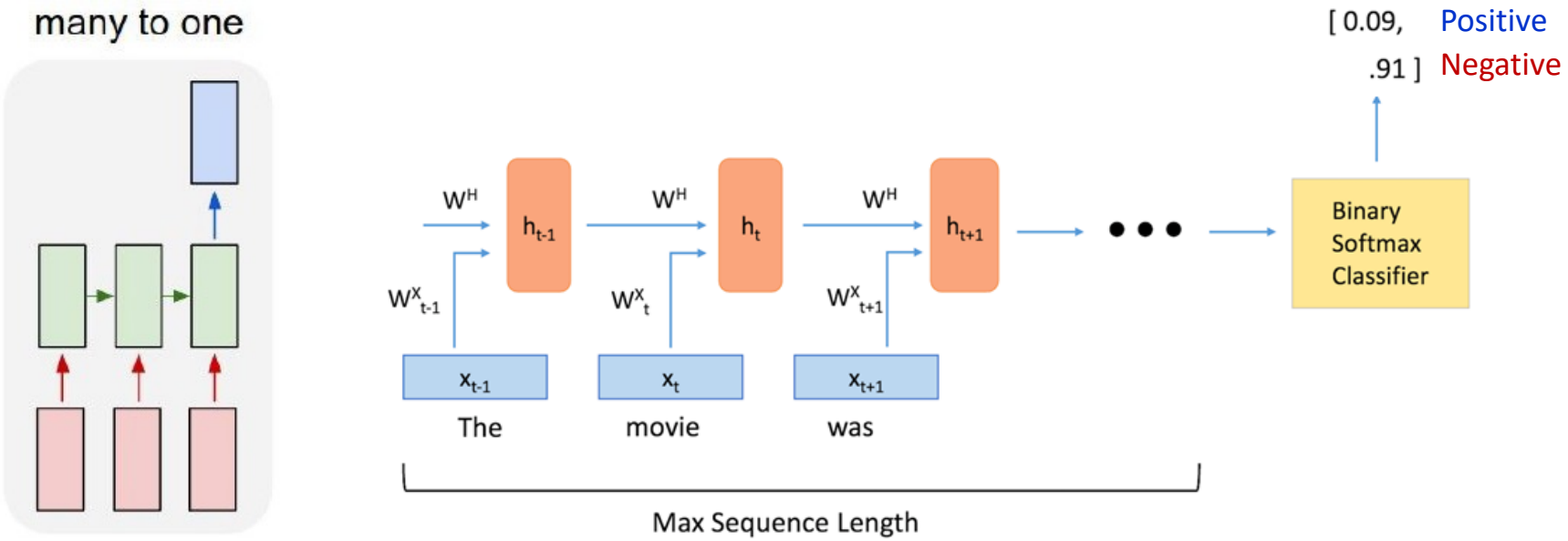
- One-to-many: Image Captioning





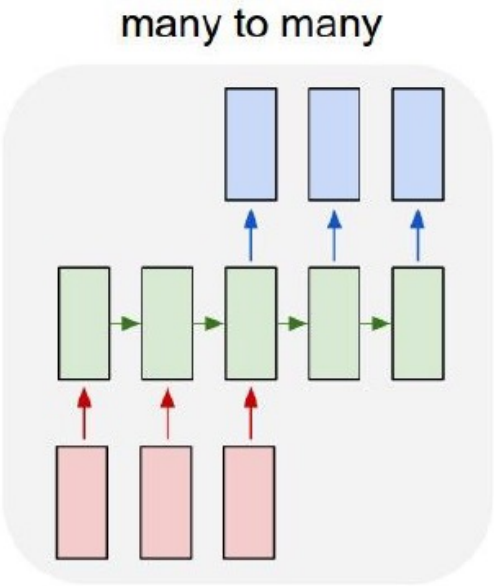
RNN Applications

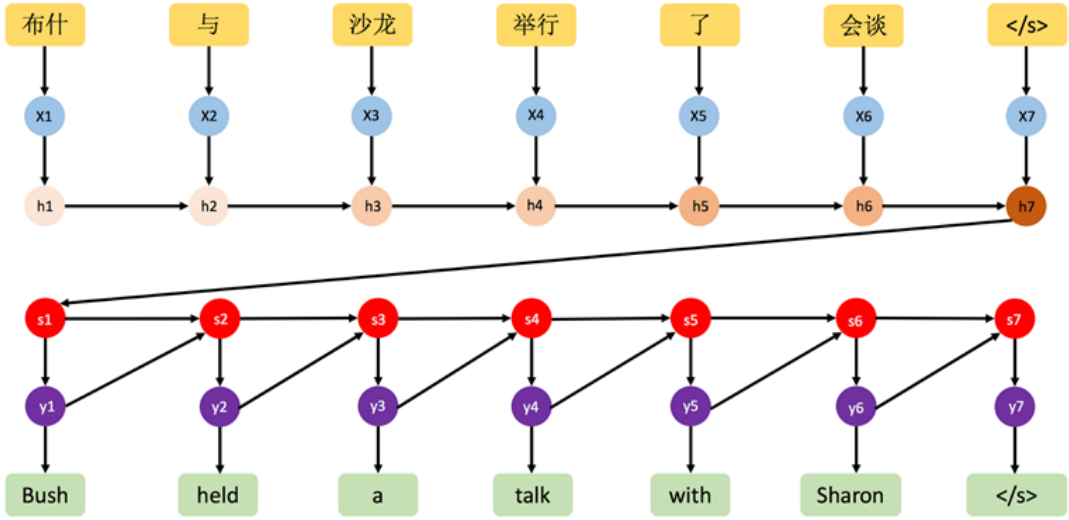
- Many-to-one: Sentiment Classification



RNN Applications

- Many-to-many: Language Translation

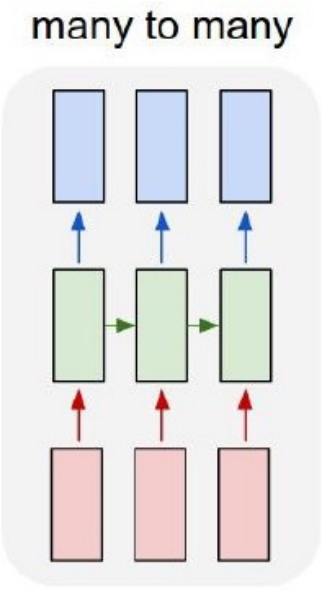


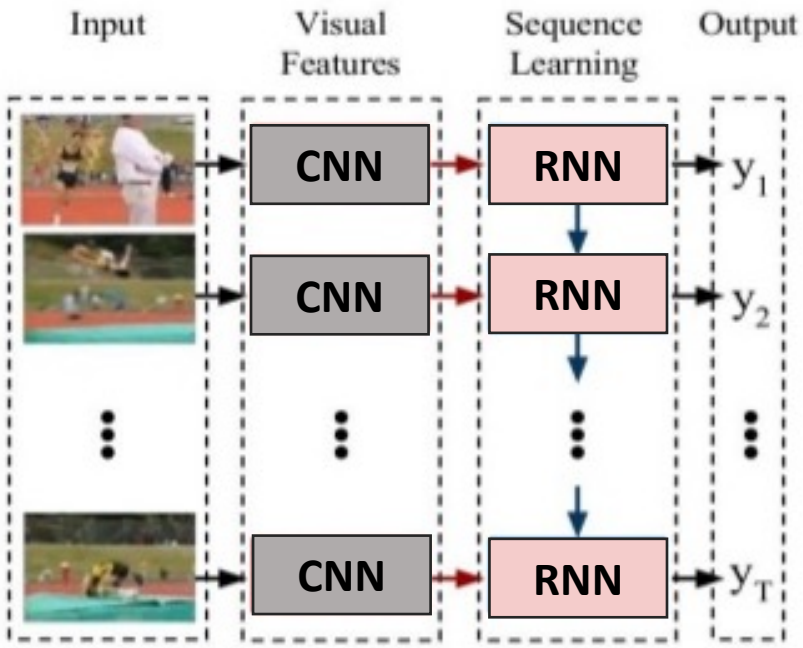


(Sutskever et al., 2014)

RNN Applications

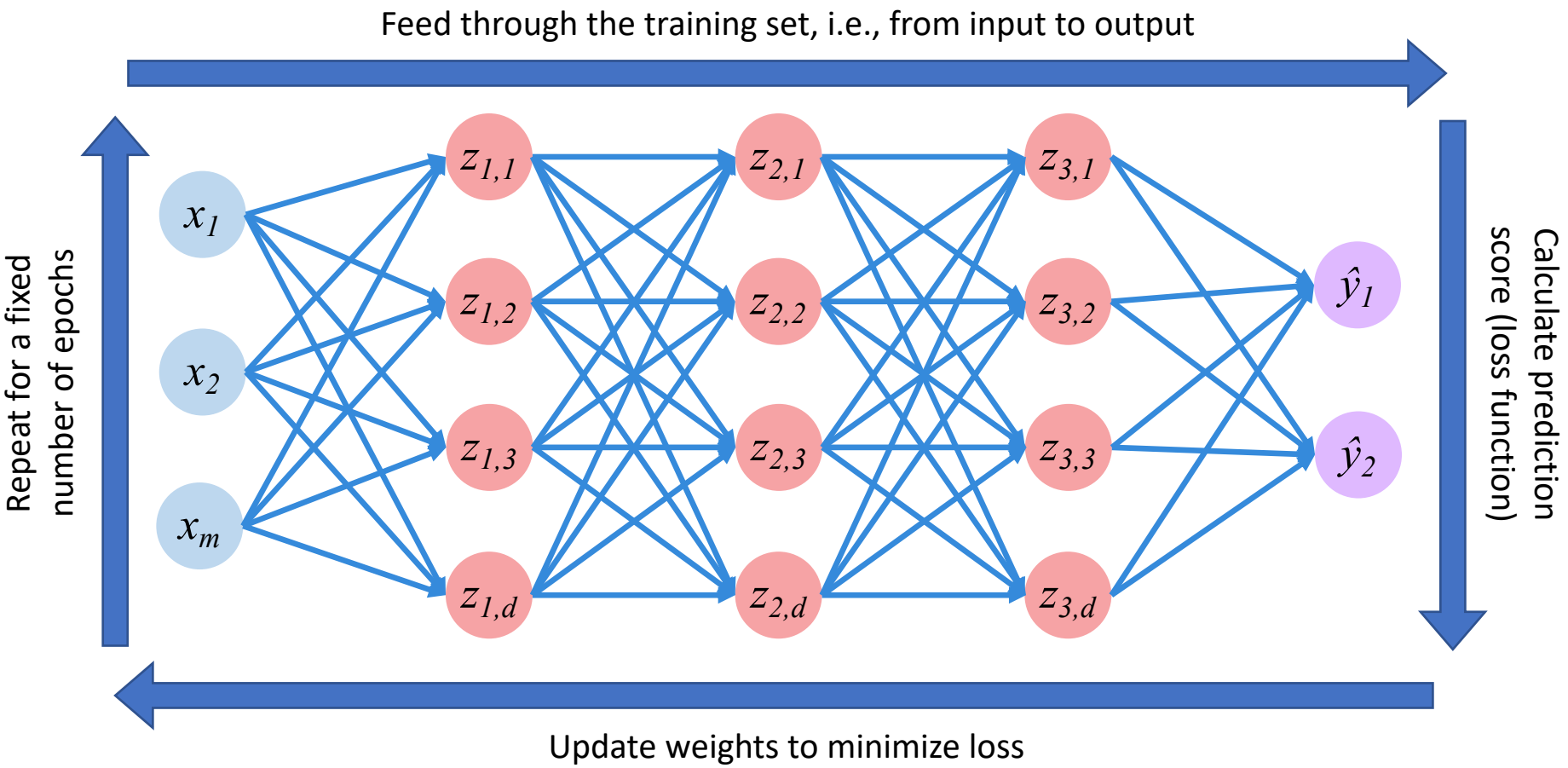
- Many-to-many: Video Classification





multilayer perceptron

Recap on Training MLP



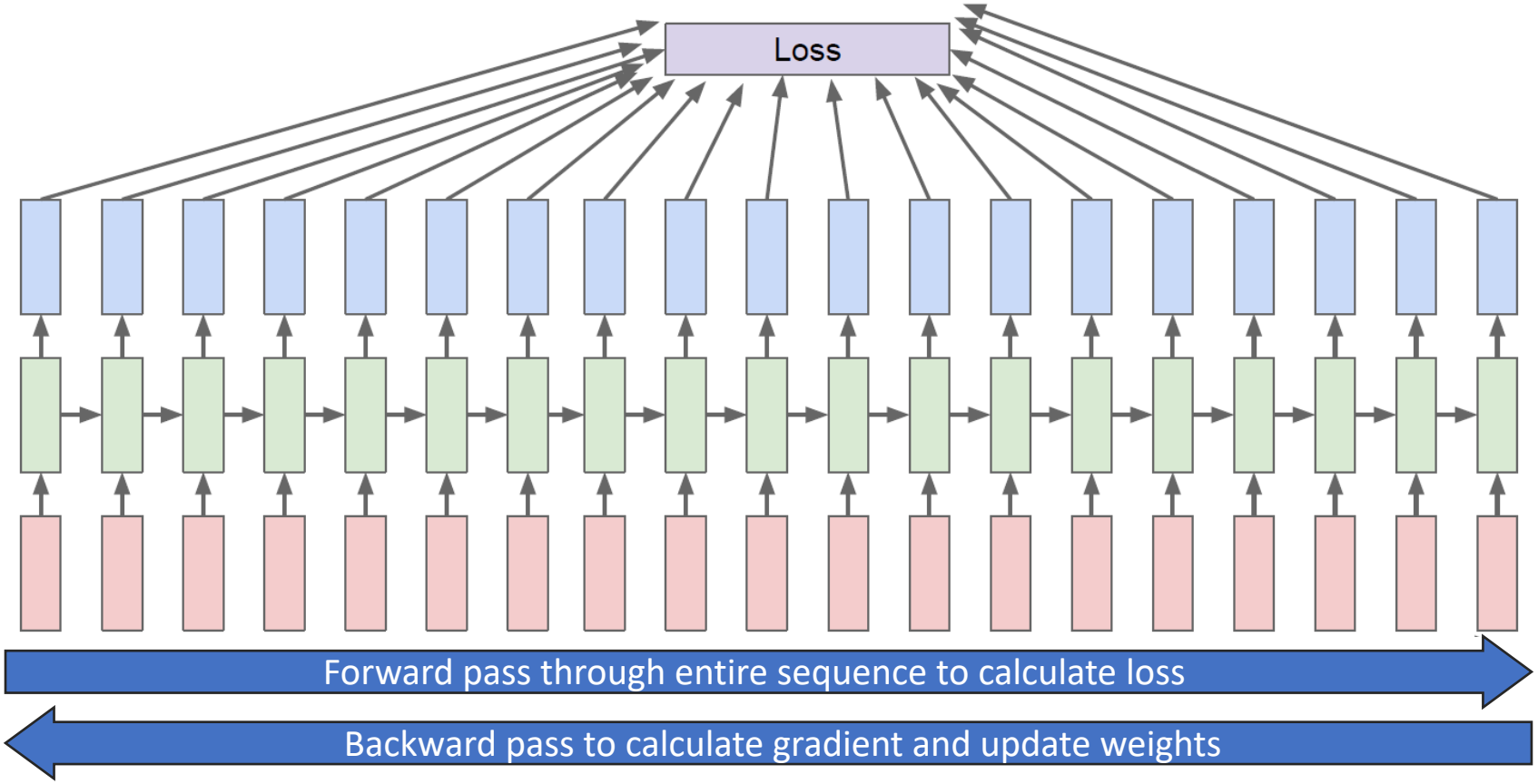
RNN Training

predicted is o t = g(h t, w o)

y t is original

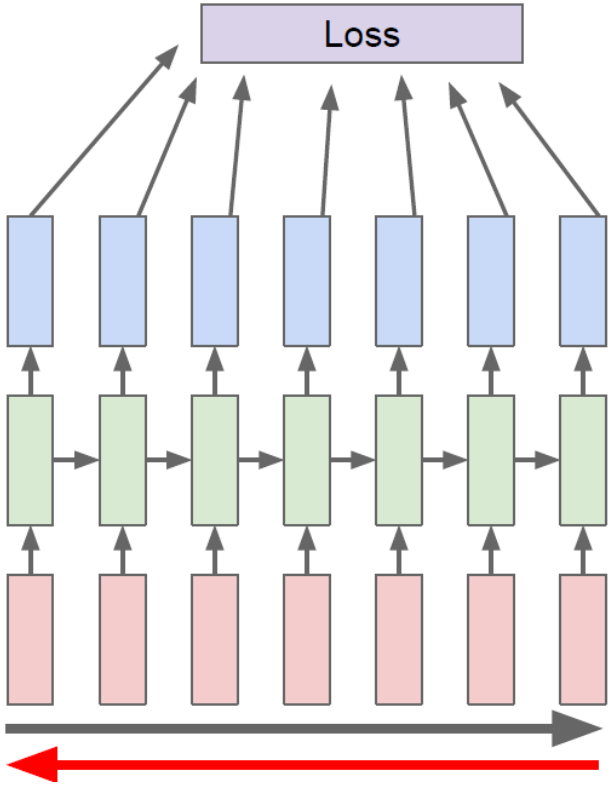
- Backpropagation through time

inputs to loss function (y t, o t)



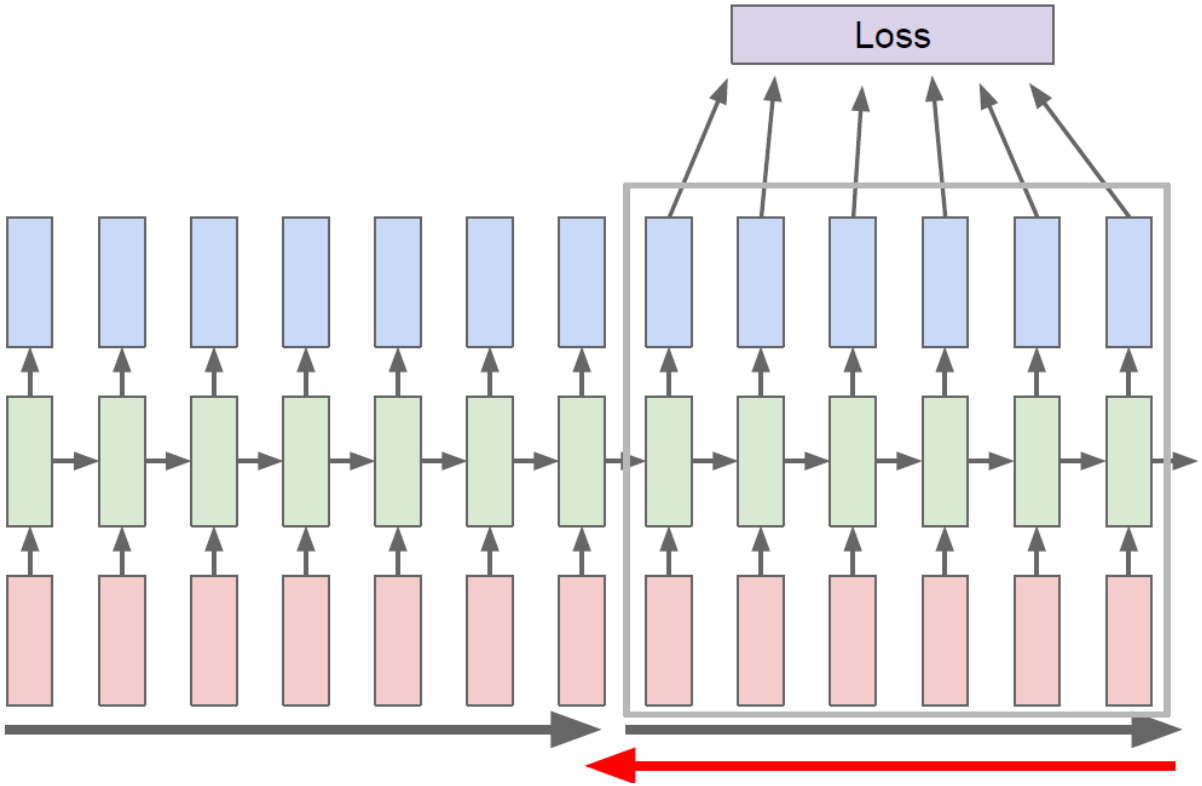
RNN Training

- Truncated Backpropagation through time
  - Instead of the entire sequence, break it up into smaller sub-sequences



RNN Training

- Truncated Backpropagation through time
  - Perform the forward/backward pass for each sub-sequence



RNN Training

related to conceptual questions:

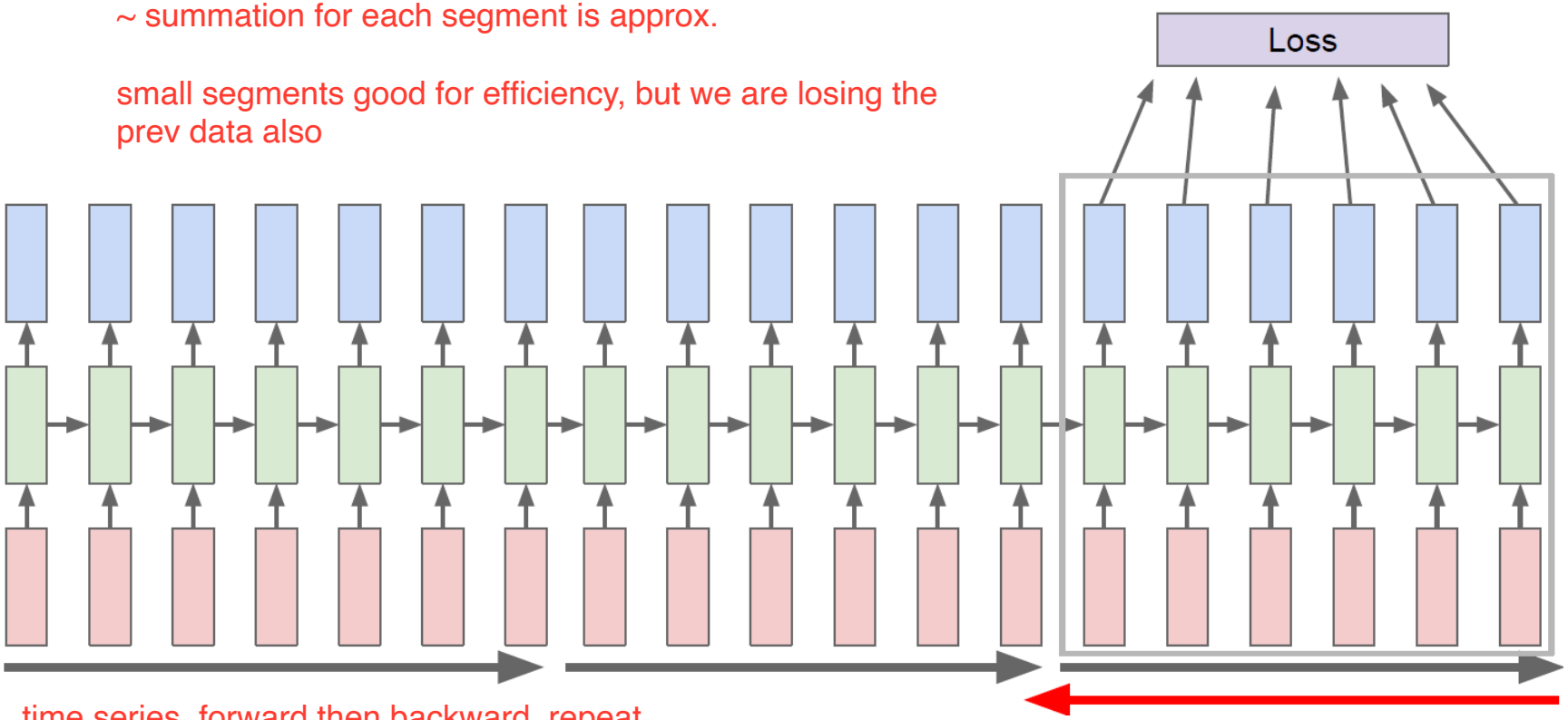
why do we need truncated backpropagation
- recurrence, if seq is long, computing this is difficult, takes a long time
- break it into small segments, accumulate and compute

- Truncated Backpropagation through time

gradient h t / w r t w h

gradients are accumulated and is updated parameters
~ summation for each segment is approx.

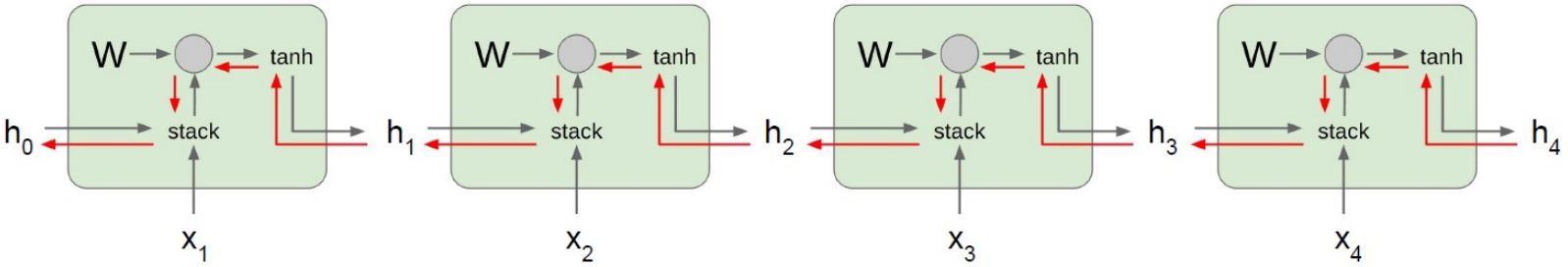
small segments good for efficiency, but we are losing the prev data also



time series, forward then backward, repeat
Wh is parameter for hidden state, shared across time steps

Problems with Vanilla RNNs

- Vanishing and Exploding Gradients



- Unable to remember inputs from long ago

*I live in **France** … … I speak fluent **French**.*

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

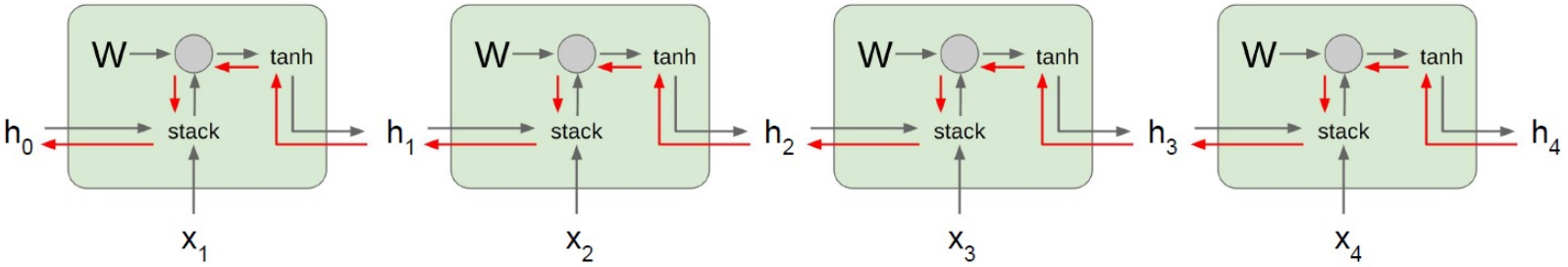
Recurrent Neural Networks (Part II)

Soujanya Poria

50.038 Computational data science

Problems with Vanilla RNNs

- Vanishing and Exploding Gradients



- Unable to remember inputs from long ago

*I live in **France** … … I speak fluent **French**.*

Recap: A Vanilla RNN Cell

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

new state

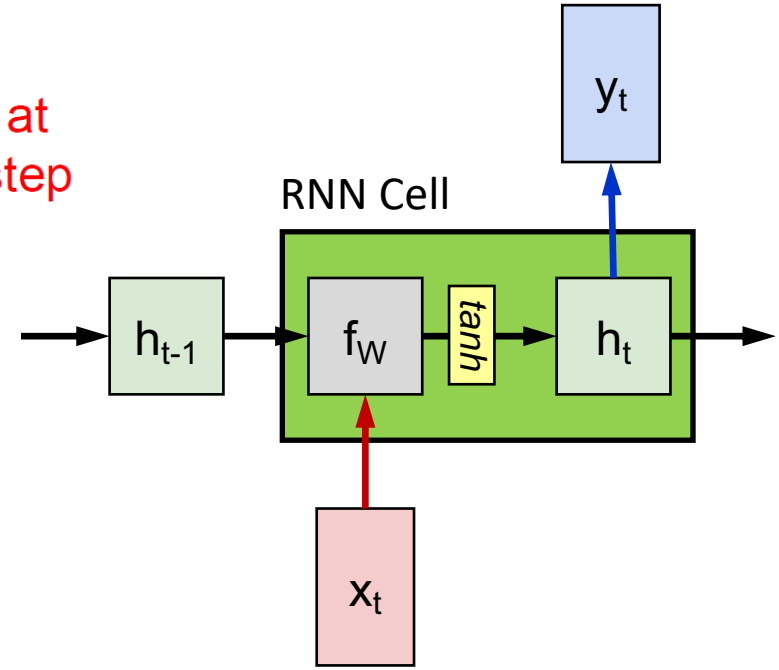
old state

input vector at some time step

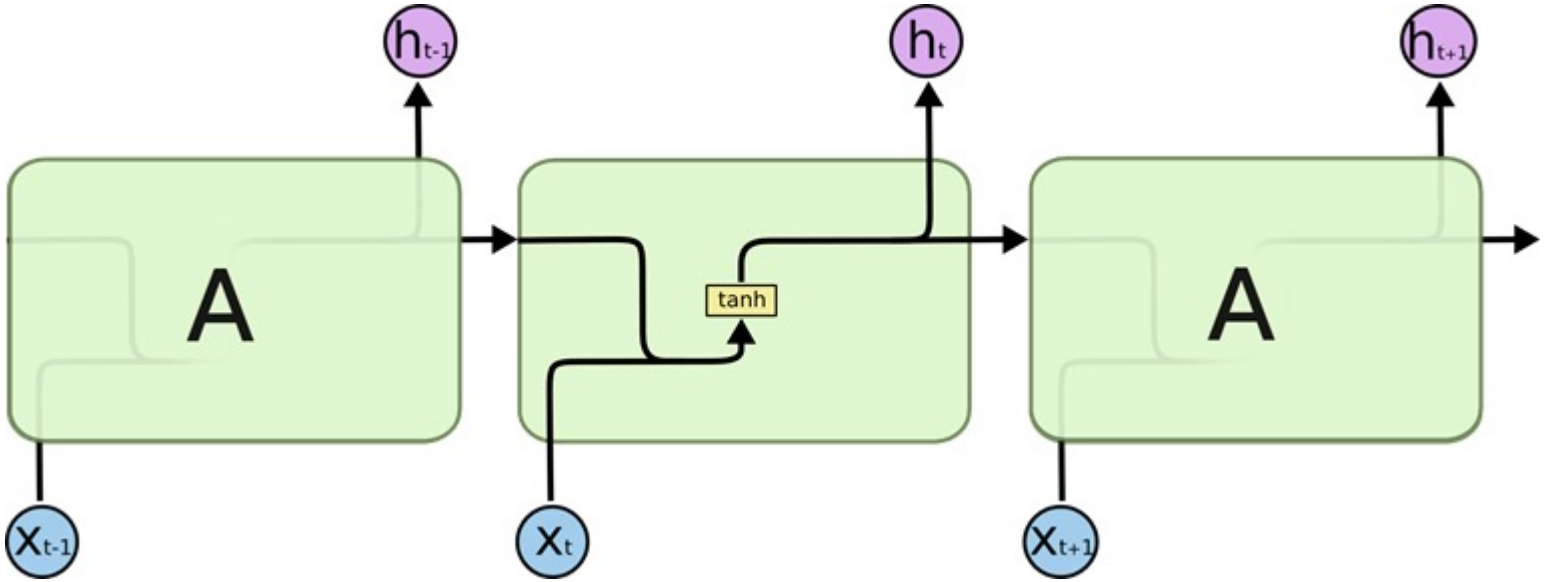
some function with parameters W

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

$$y_t = W_{hy} h_t$$



Recap: A Vanilla RNN Unit



Long Short-Term Memory

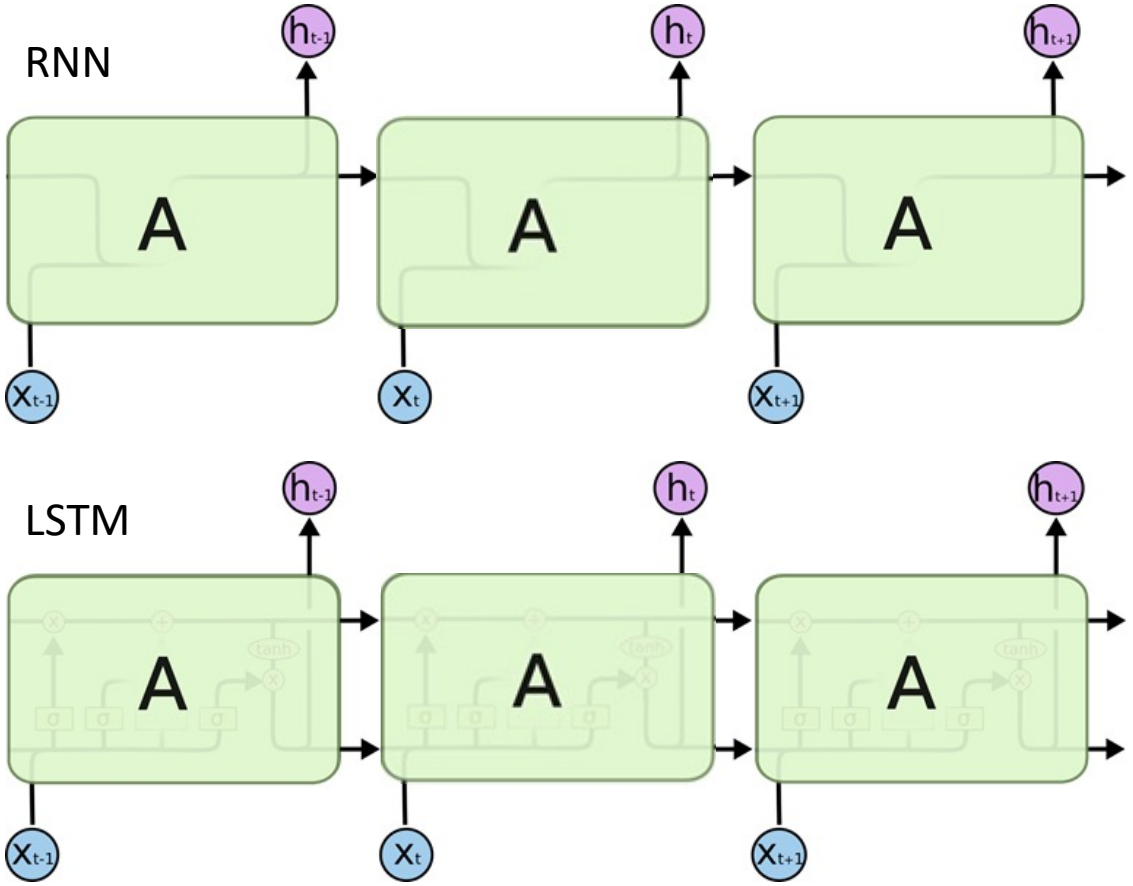
- Long Short-Term Memory (LSTM) were designed to overcome problems of vanilla RNNs
  - i.e., vanishing/exploding gradients, long-term dependencies

- A LSTM cell/unit comprises the following:
  - Cell State
  - Forget Gate
  - Input Gate
  - Output Gate

- Contrast this to the vanilla RNN cell

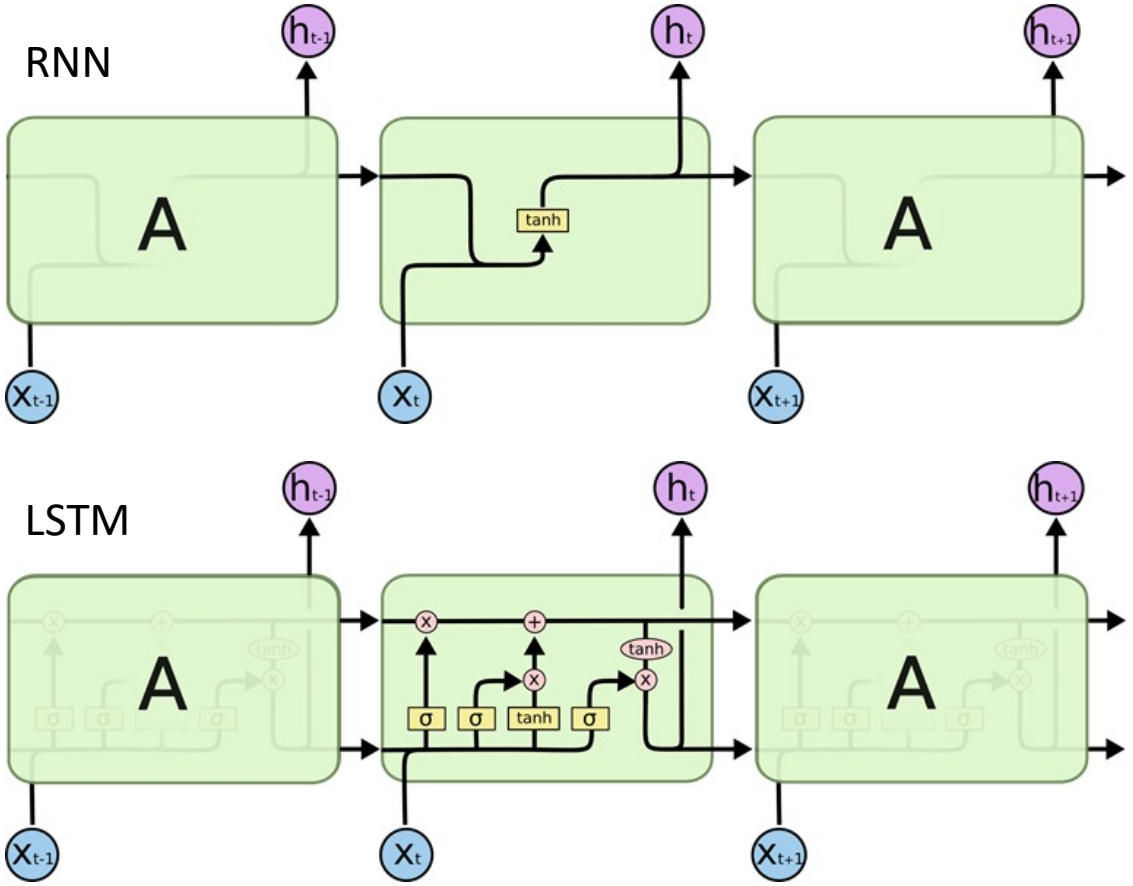
Exercise 4: Vanilla RNN vs LSTM

- Based on an overview of both architectures, what similarities do you observe?



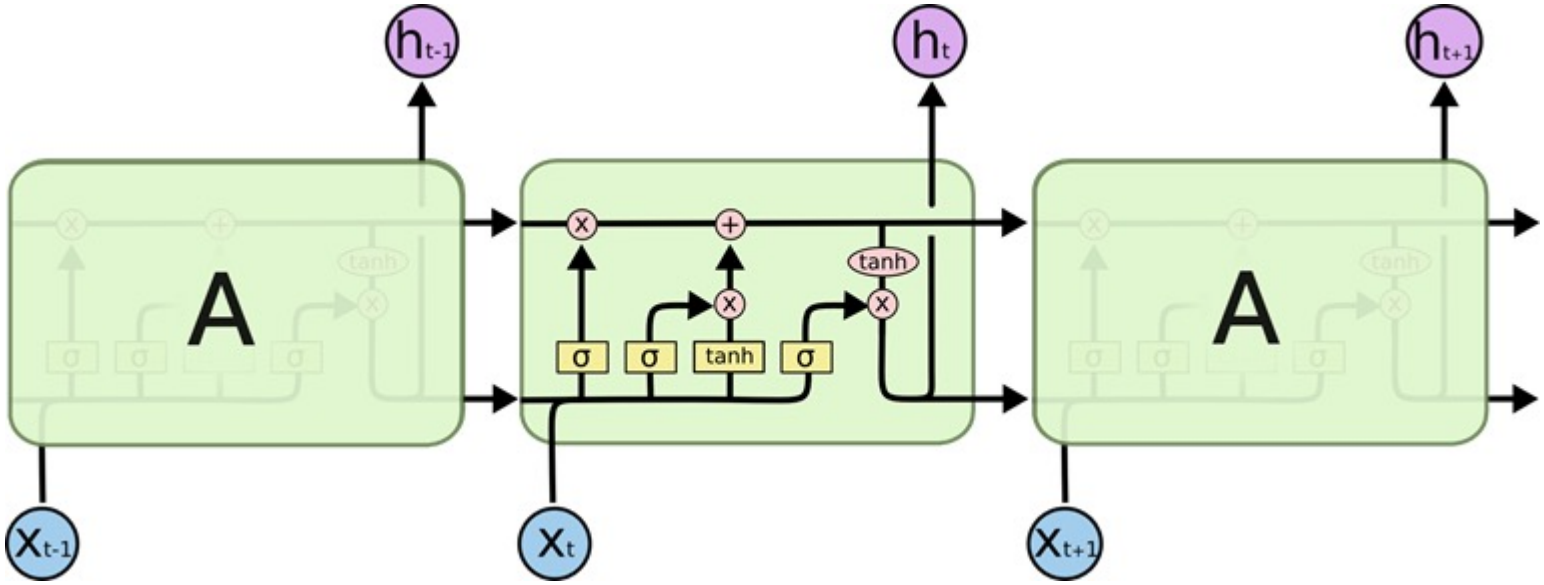
Vanilla RNN vs LSTM

- Differences in terms of internal cell operations
  - Vanilla RNN only considers the previous state h_{t-1} and current input x_t
  - LSTM has the three gates and cell state



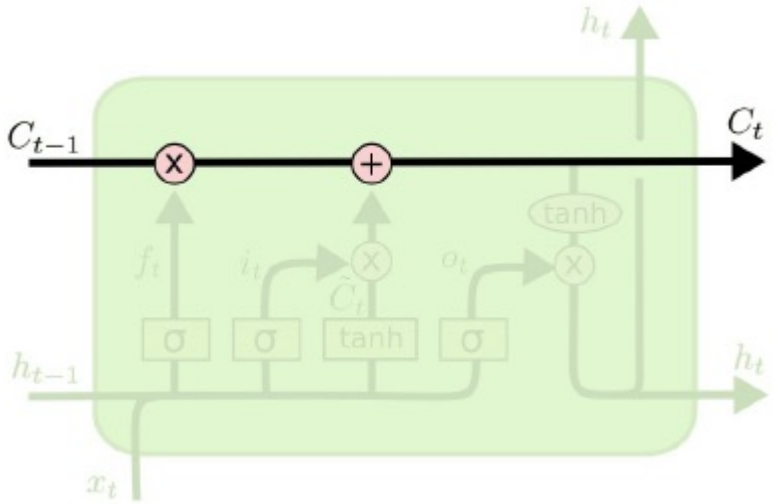
LSTM Components

- Cell state, Forget gate, Input gate, Output gate



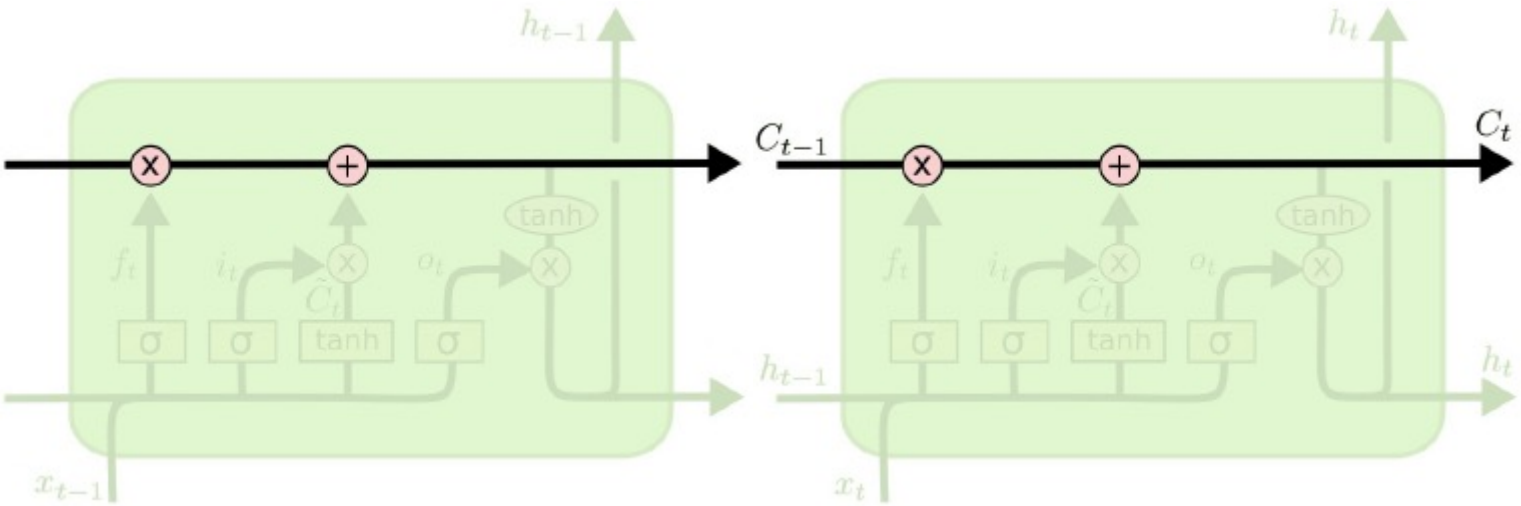
LSTM Components

- Cell State
  - One of the key difference from vanilla RNNs



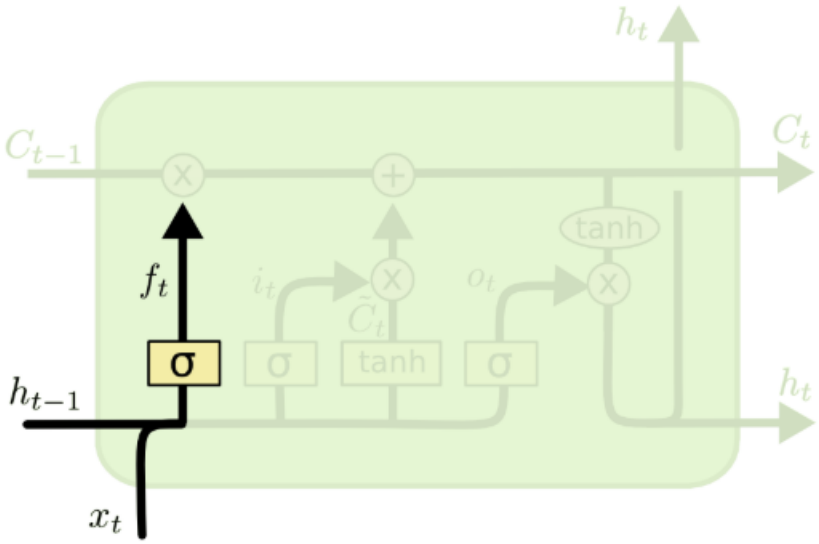
LSTM Components

- Cell State
  - One of the key difference from vanilla RNNs
  - Serves like an information highway through all LSTM cells
    - How is information added to this cell state?



LSTM Components

- Forget Gate
  - Determines which part of the previous cell state to remember/forget
  - The sigmoid function allows us to completely forget (0) or remember (1)

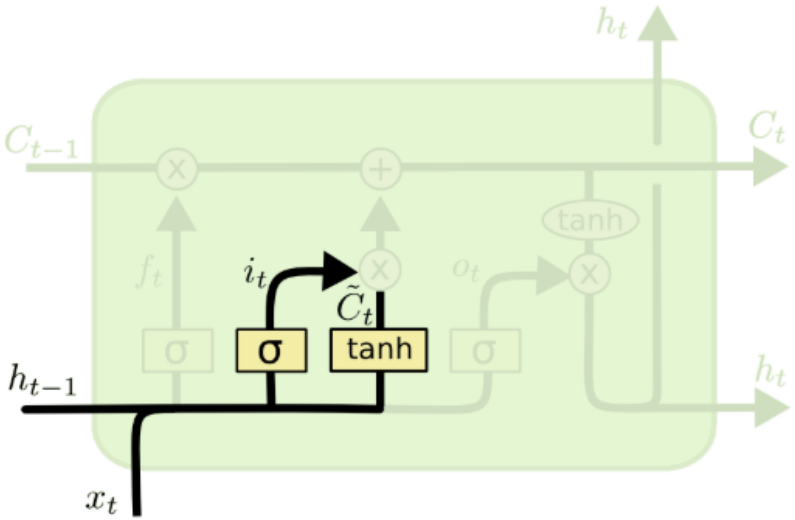


contextualised w hidden state + input

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

LSTM Components

- Input Gate
  - Determine what new information to write to cell state
  - Sigmoid to choose what to write, tanh to generate the candidate values

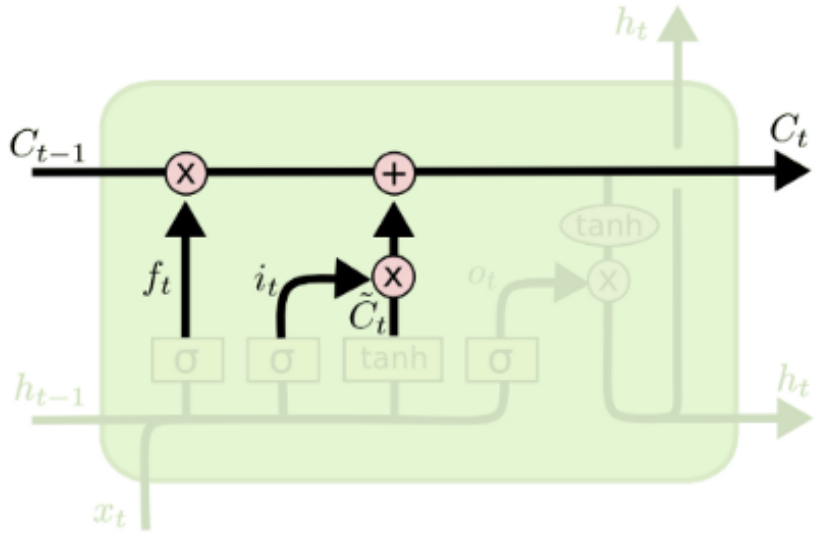


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

LSTM Components

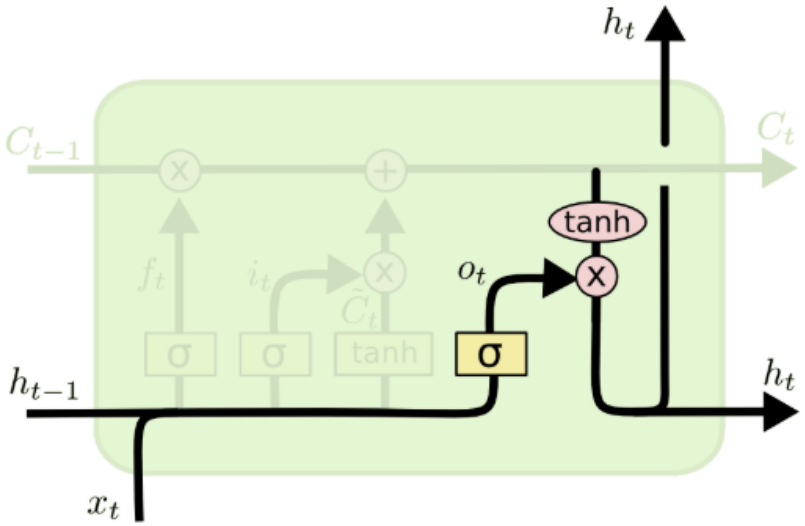
- Cell State – Forget and Input
  - Operations to update the cell state from C_{t-1} to C_t
    - Based on the previous Forget gate and Input gate



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

LSTM Components

- Output Gate
  - Determines which part of the new cell state to output
  - The output h_t can then be passed to the next LSTM cell and/or used to generate a prediction at timestep *t+1*

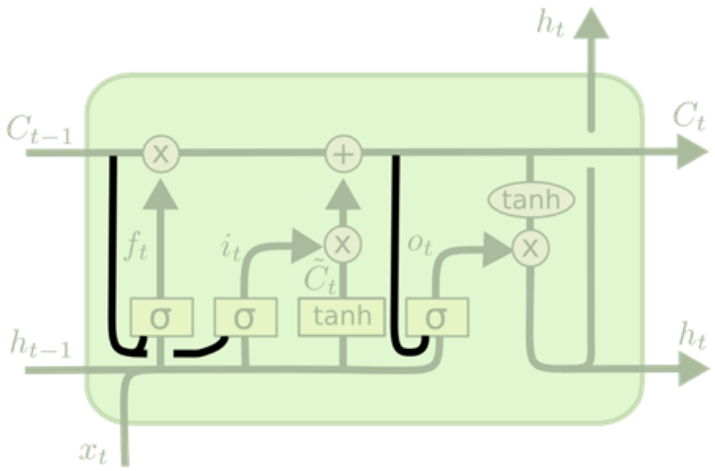


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

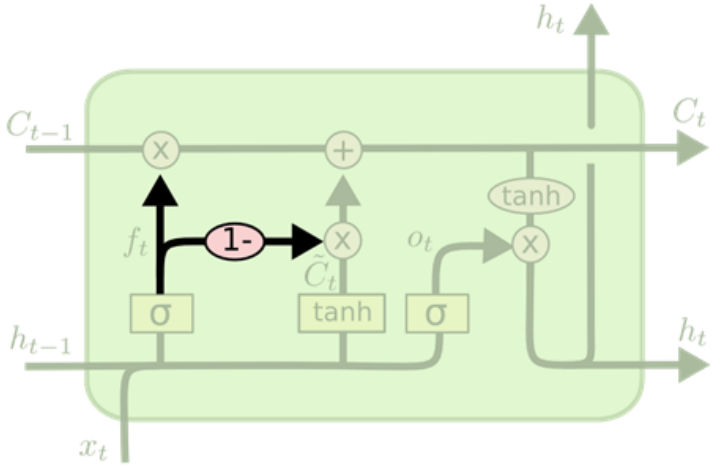
$$h_t = o_t * \tanh\left(C_t\right)$$

Other LSTM Variants

- LSTM with Peepholes
  - Allow the three gates to "peep" into the cell state



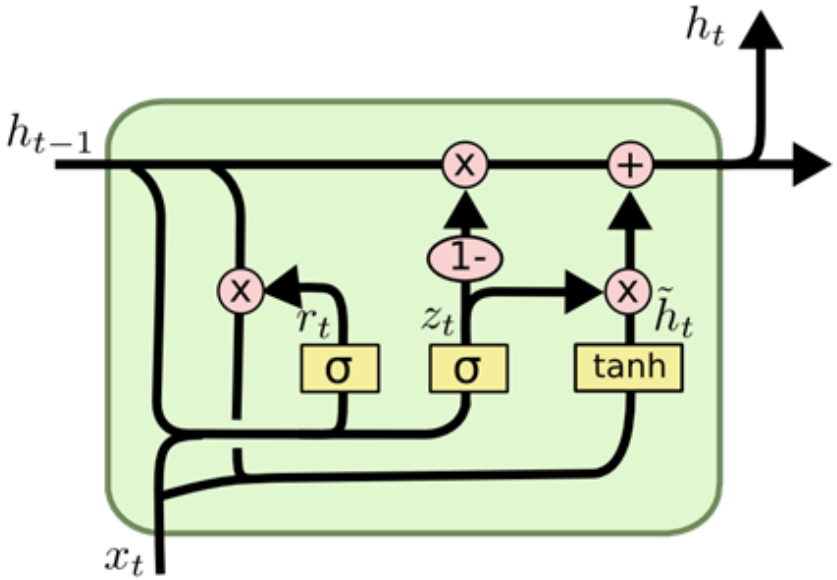
- LSTM with combined Forget/Input gates



Other Variants

less parameters
perform similarly, GRU is faster

- Gated Recurrent Units
  - A simplified version of LSTM
  - Has no independent cell state
  - Has a Reset gate (r_t)
    - Determines how to combine the new input with the previous memory
  - Has a Update gate (z_t)
    - Determines how much of the previous memory to keep around
    - Combines the Forget and Input gates of LSTM



Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

LSTM in Keras

- Initialize a sequential model (layers added one after another)

```
model = Sequential()
```

- For a task like text classification, add an embedding layer to convert words to a dense vector (e.g., of dimension 256)

```
model.add(Embedding(max_features, output_dim=256))
```

- Add in a LSTM layer that outputs a dense vector of dimension 128

```
model.add(LSTM(128))
```

- Add regularization, if necessary

```
model.add(Dropout(0.5))
```

LSTM in Keras

- Add in a final fully-connected layer to predict labels (change number of nodes and activation function, as required)

```
model.add(Dense(1, activation='sigmoid'))
```

- Define parameters for training the defined model

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

LSTM in Keras

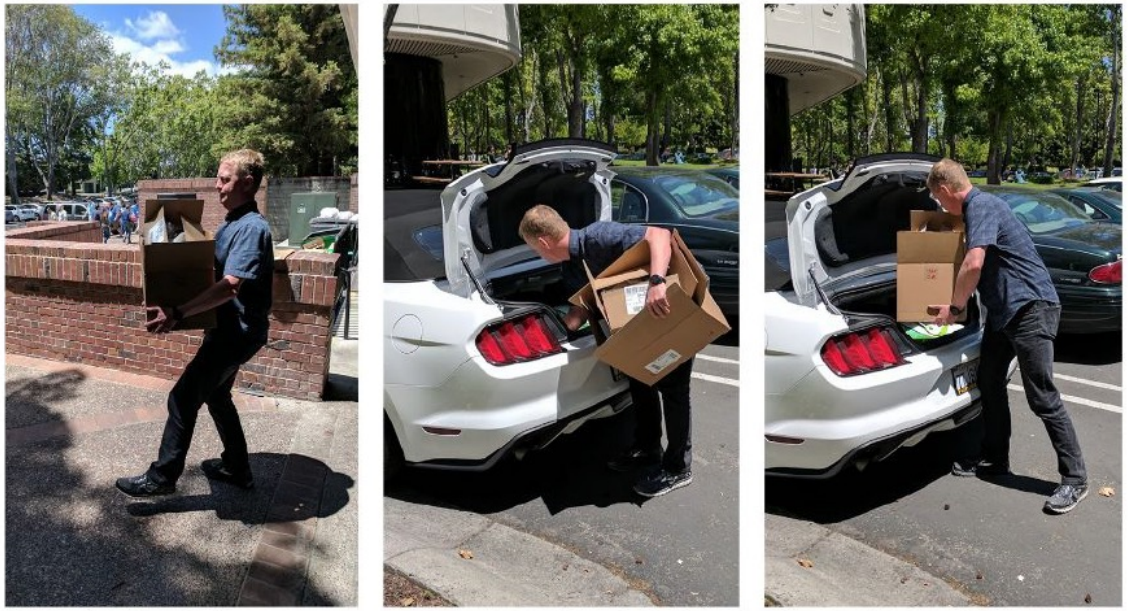
- Training the model

```
model.fit(x_train, y_train, batch_size=16, epochs=10)
```

- Testing the model

```
score = model.evaluate(x_test, y_test, batch_size=16)
```

Application: Video Question Answering

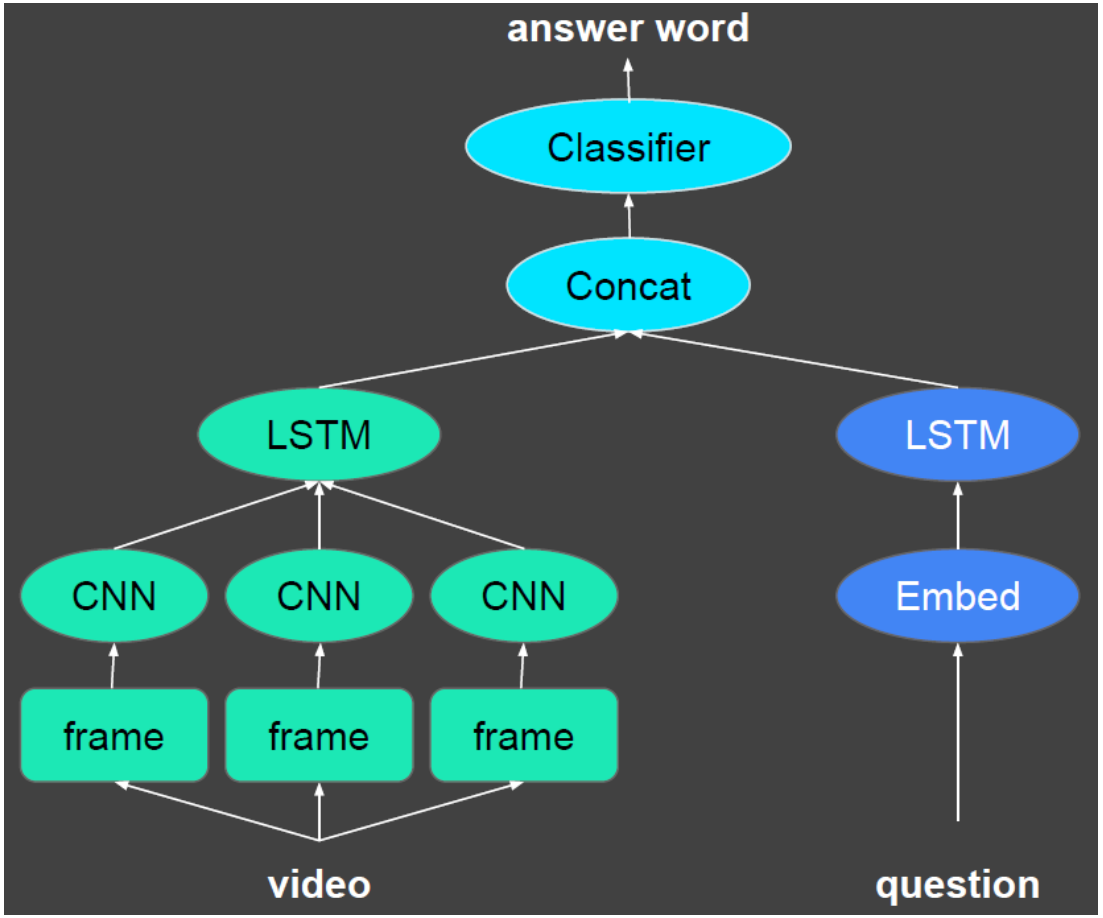


> ***"What is the man doing?"***
> packing

> ***"What color is his shirt?"***
> blue

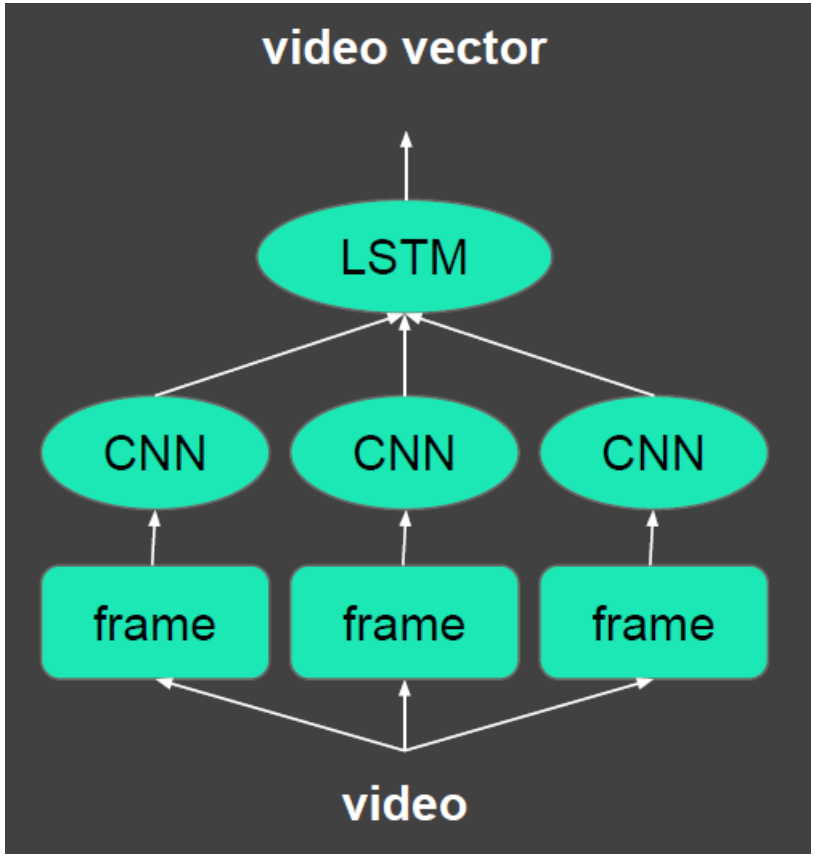
Application: Video Question Answering

- Utilizing MLP, CNN, LSTM and embeddings
  - One component for video
  - One component for question



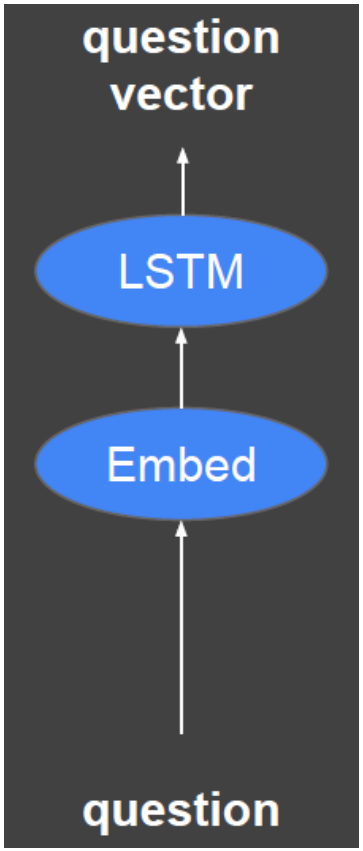
Application: Video Question Answering

- Video component
  - Main idea is to represent video as a dense vector
  - Video split into frames, each frame represented as a dense vector
  - Each “frame” then provided as input to LSTM as a temporal sequence



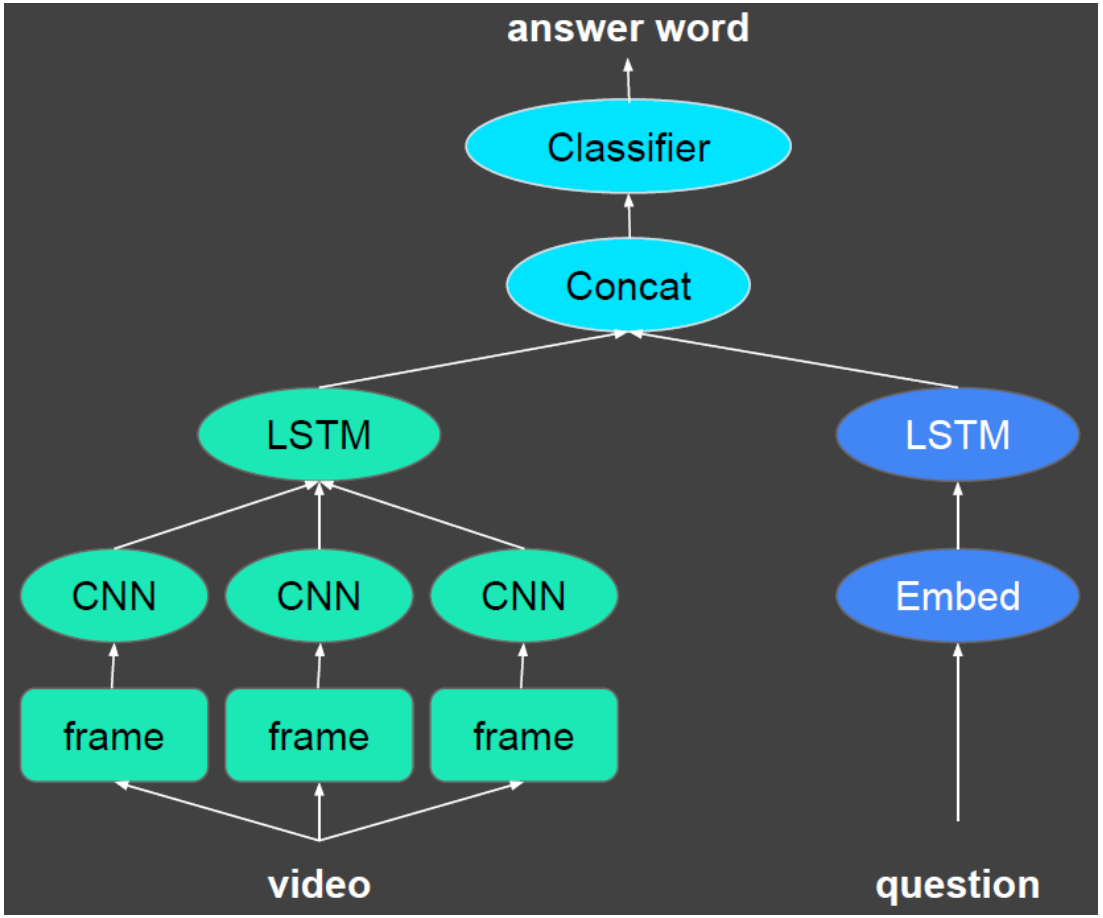
Application: Video Question Answering

- Question component
  - Main idea is to represent question as a dense vector
  - Question is split into words, each represented by a word embedding
  - Each word embedding then provided as input to LSTM as a temporal sequence



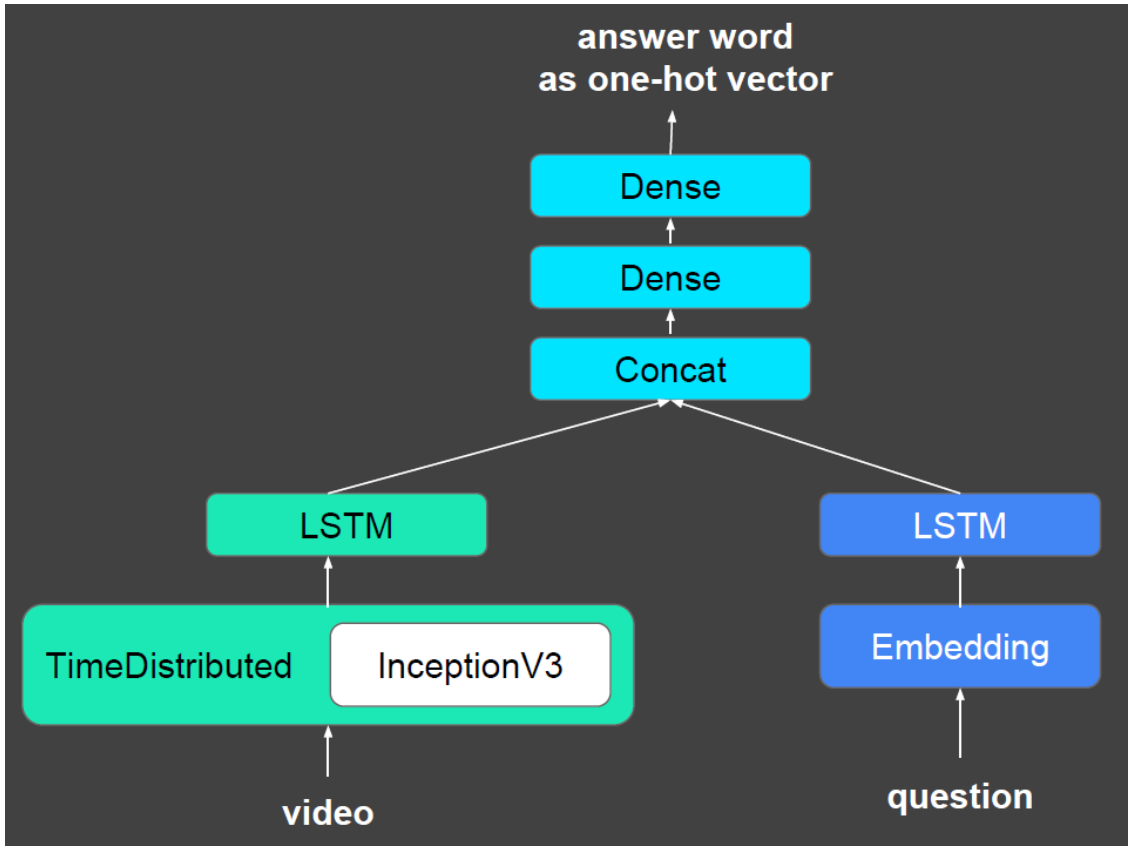
Application: Video Question Answering

- Utilizing MLP, CNN, LSTM and embeddings
  - Dense vectors for video and question components are concatenated
  - Concatenated vector then provided as input to a MLP classifier



Application: Video Question Answering

- Overview of implementation in Keras



Application: Video Question Answering

- Video component
  - Utilizing pre-trained weights to convert video to dense vectors

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

Application: Video Question Answering

- Question component
  - Converting question to a sequence of word (embeddings)

```
question = keras.Input(shape=(None,), dtype='int32', name='question')
embedded_words = layers.Embedding(input_voc_size, 256)(question)
question_vector = layers.LSTM(128)(embedded_words)
```

Application: Video Question Answering

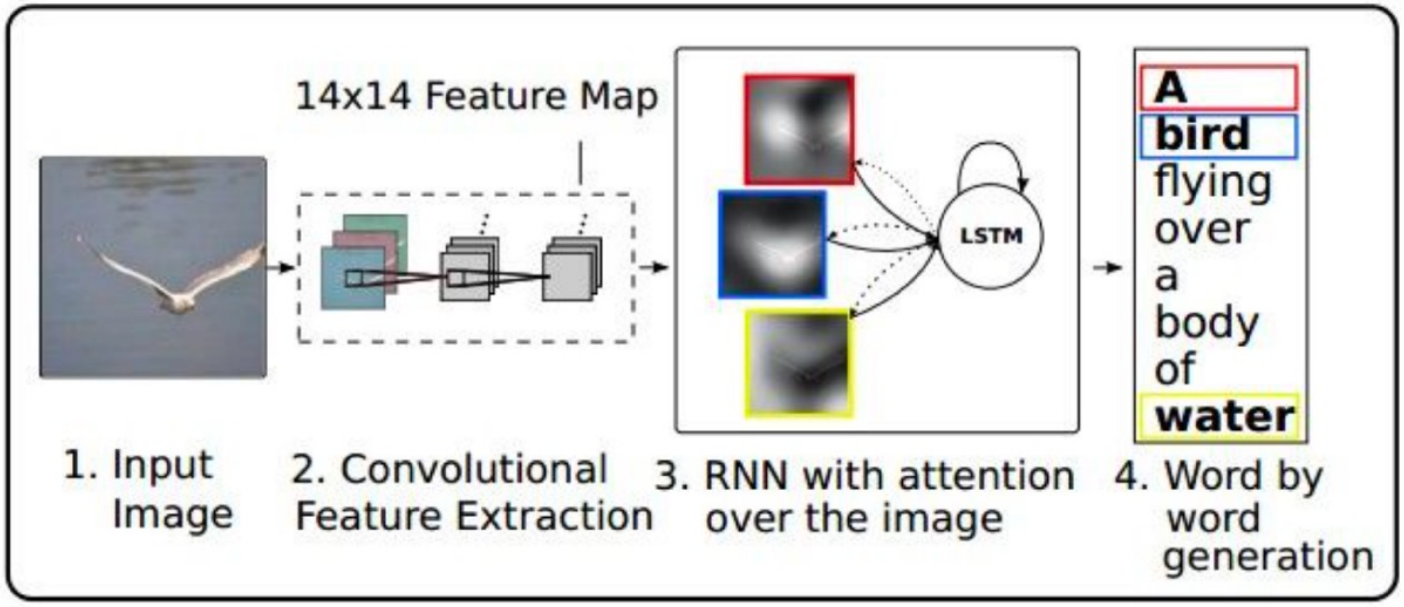
- Answer prediction
  - Concatenating video and question vector
  - Predicting answer using a MLP

```
x = layers.concatenate([video_vector, question_vector])
x = layers.Dense(128, activation=tf.nn.relu)(x)
predictions = layers.Dense(output_voc_size,
                           activation='softmax',
                           name='predictions')(x)
```

Other RNN Applications

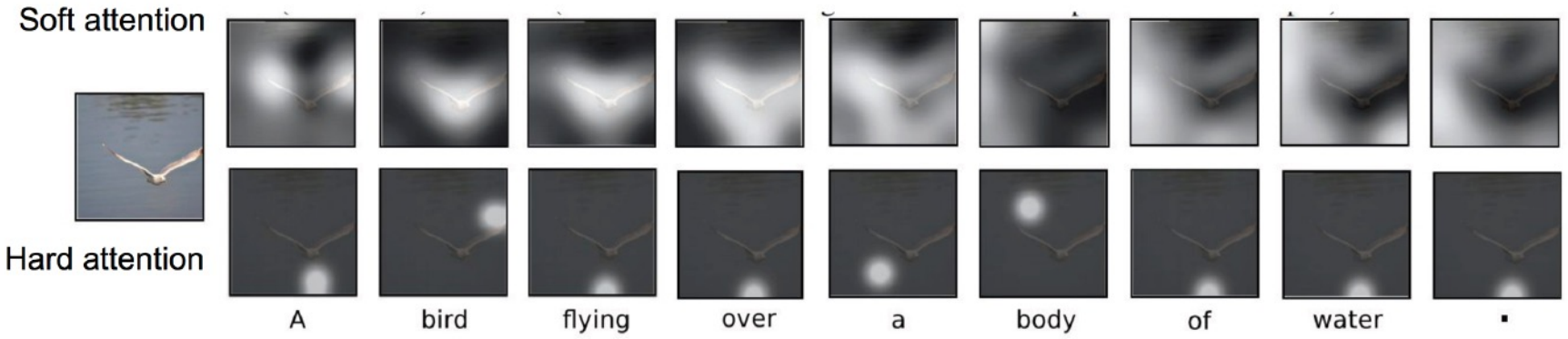
Applications: Image Captioning with Attention [Xu et al., ICML 2015]

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



Other RNN Applications

Applications: Image Captioning with Attention [Xu et al., ICML 2015]

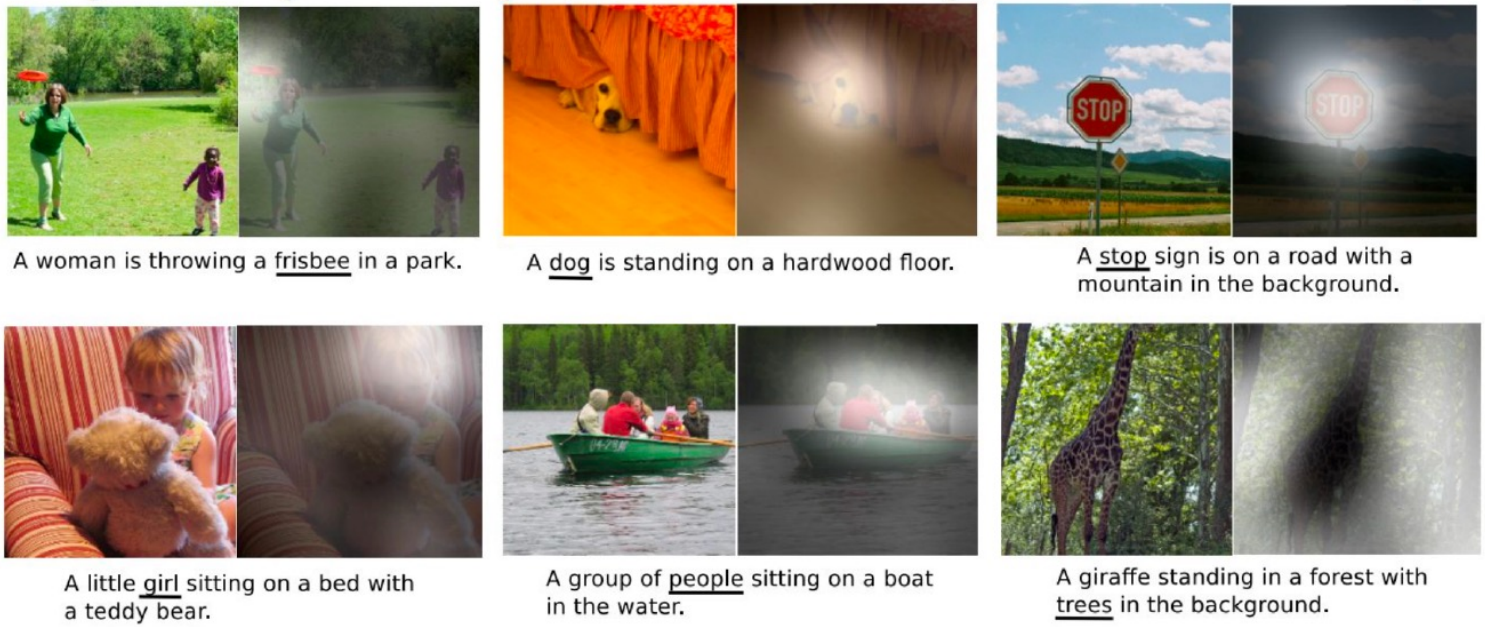


Other RNN Applications

Applications: Image Captioning with Attention

[Xu et al., ICML 2015]

Good results

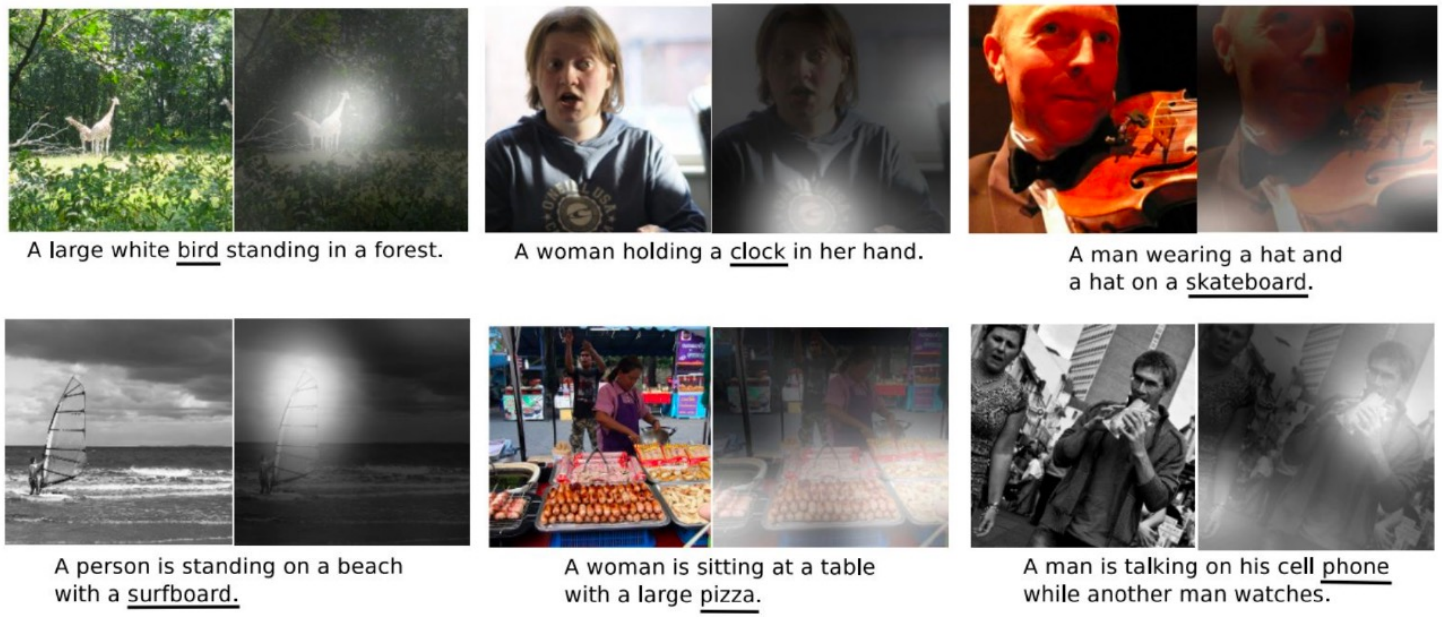


Other RNN Applications

Applications: Image Captioning with Attention

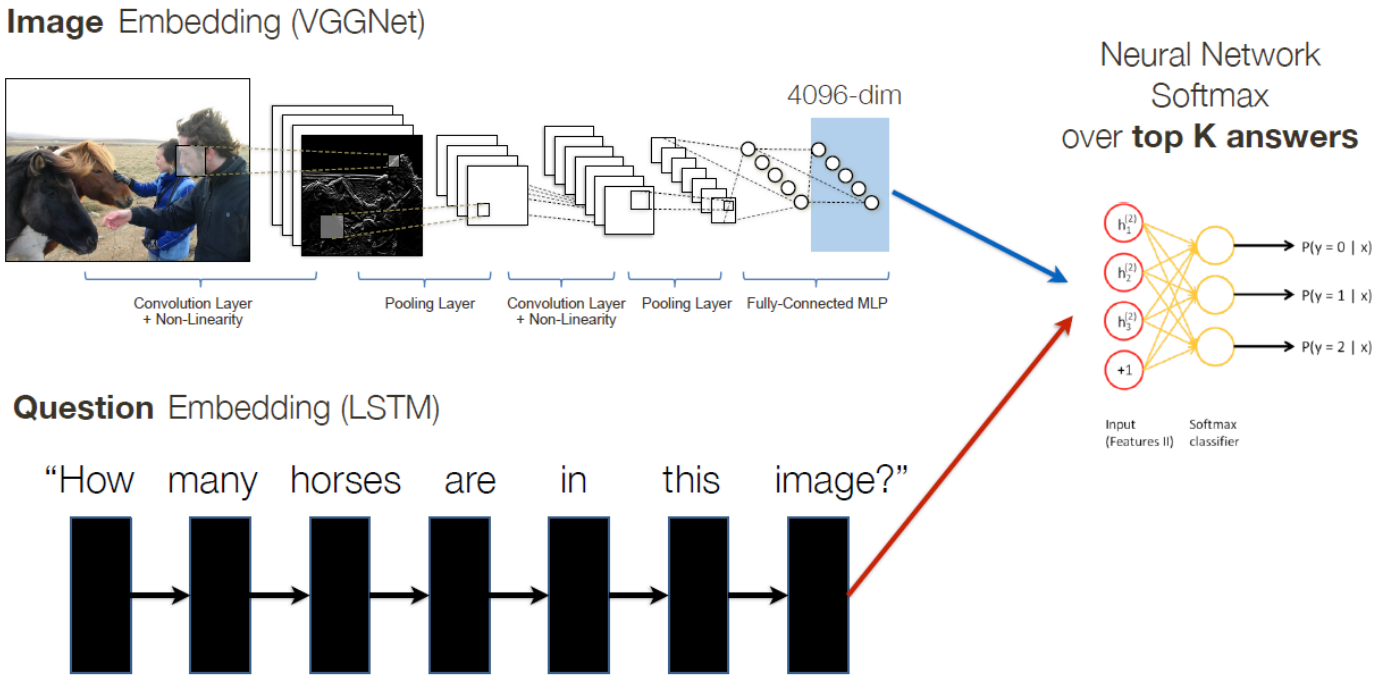
[Xu et al., ICML 2015]

Failure results



Other RNN Applications

Applications: Typical Visual Question Answering (VQA)



Other RNN Applications

Applications: Activity Detection

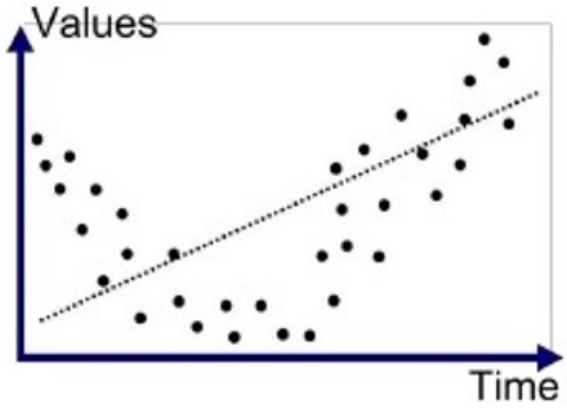
[Ma et al., 2014]

Activity: A collection of human/object movements with a particular semantic meaning

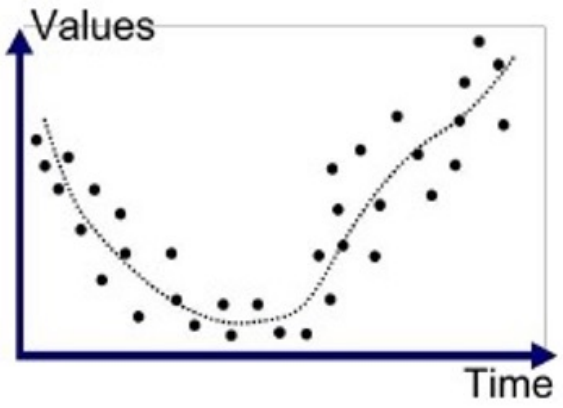
Action Recognition: Finding if a video segment contains such a movement

Action Detection: Finding a segment (beginning and start) and recognize the action in it

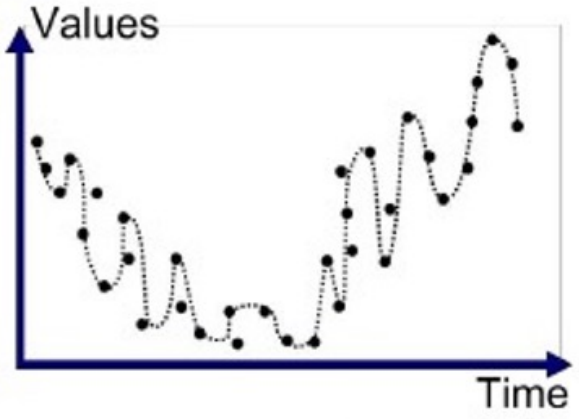
Under/Over-fitting Problem



Underfitted



Good Fit/Robust



Overfitted

Source: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

Regularization

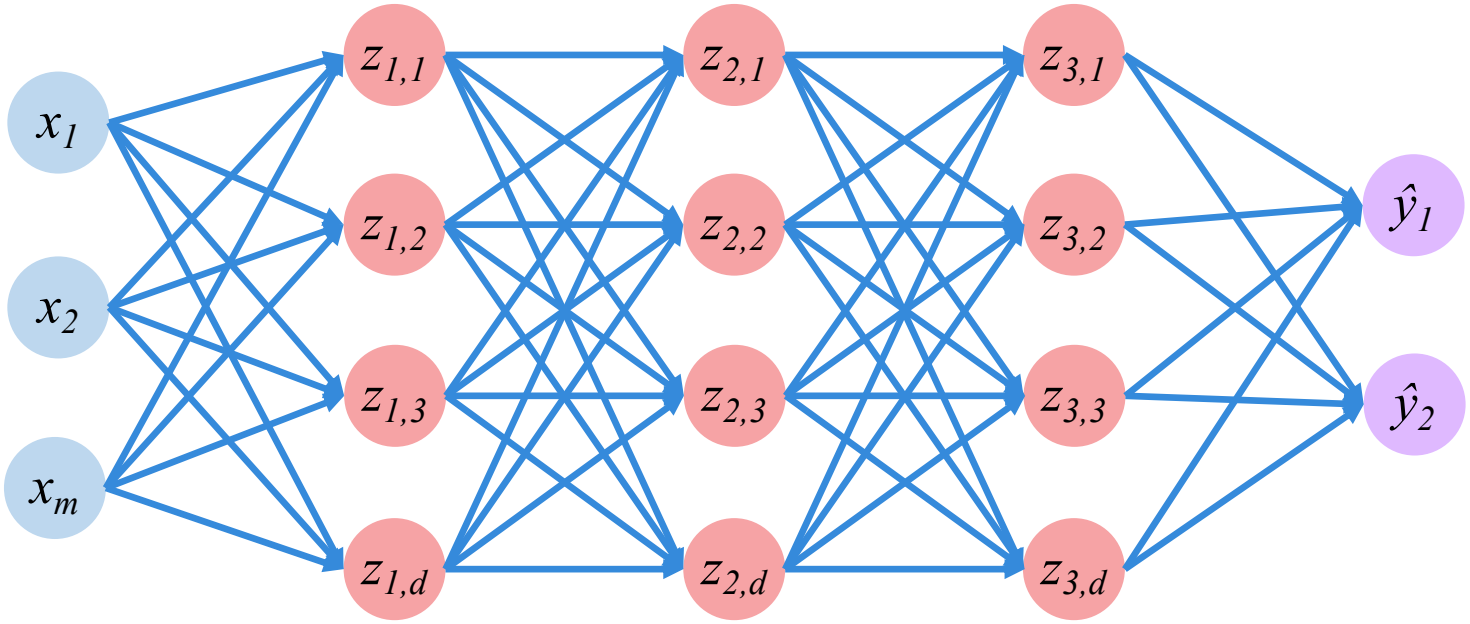
- Technique to help prevent overfitting on training data
  - Helps to generalize our model on unseen (testing) data

- Two main types
  - Dropout
  - Early Stopping

Dropout

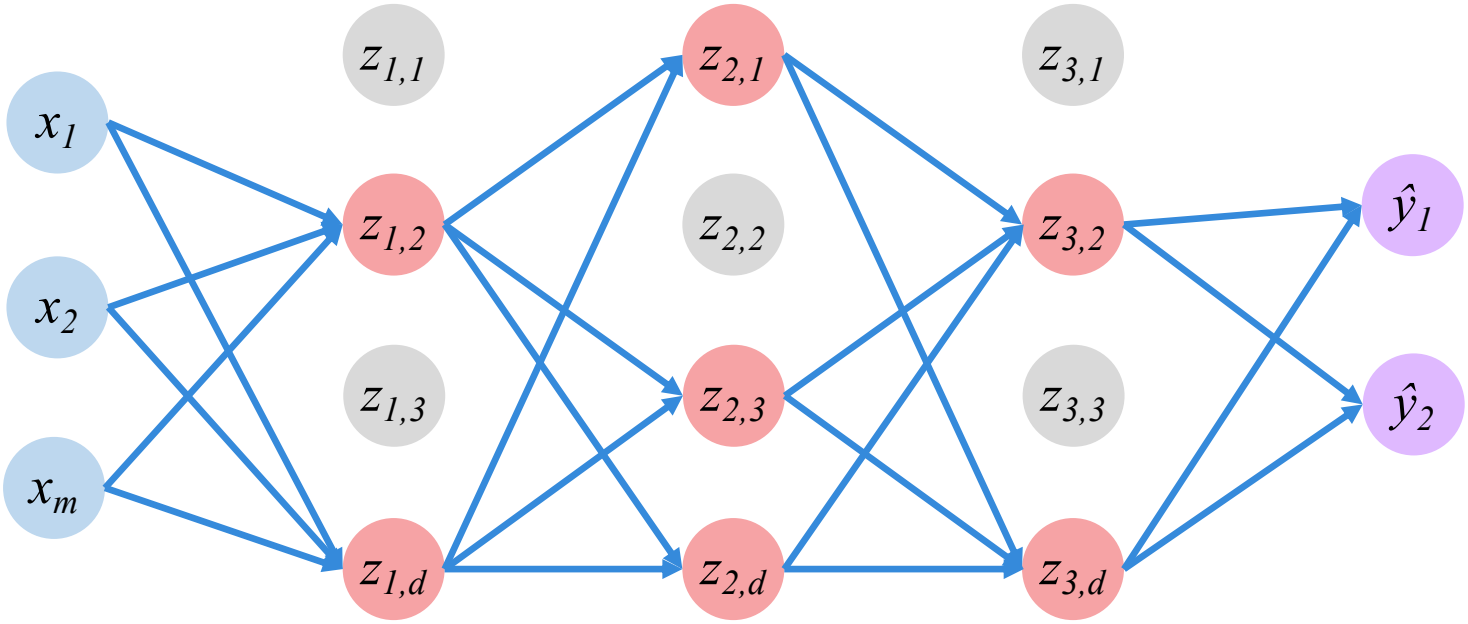
training nn monral don't use dropout

- Regularization by randomly ignoring certain neuron
  - I.e., selected activations are set to 0
  - Prevents over-reliance on a single neuron



Dropout

- Regularization by randomly ignoring certain neuron
  - I.e., selected activations are set to 0
  - Prevents over-reliance on a single neuron



Early Stopping

- Stop training of model before overfitting

Summary

- Provided an overview of the types of RNNs
- Discussed about vanilla RNNs and its limitation
- Discussed about how LSTM works
- Have an overview of the different variants of LSTMs
- Studied various applications of RNNs/LSTMs

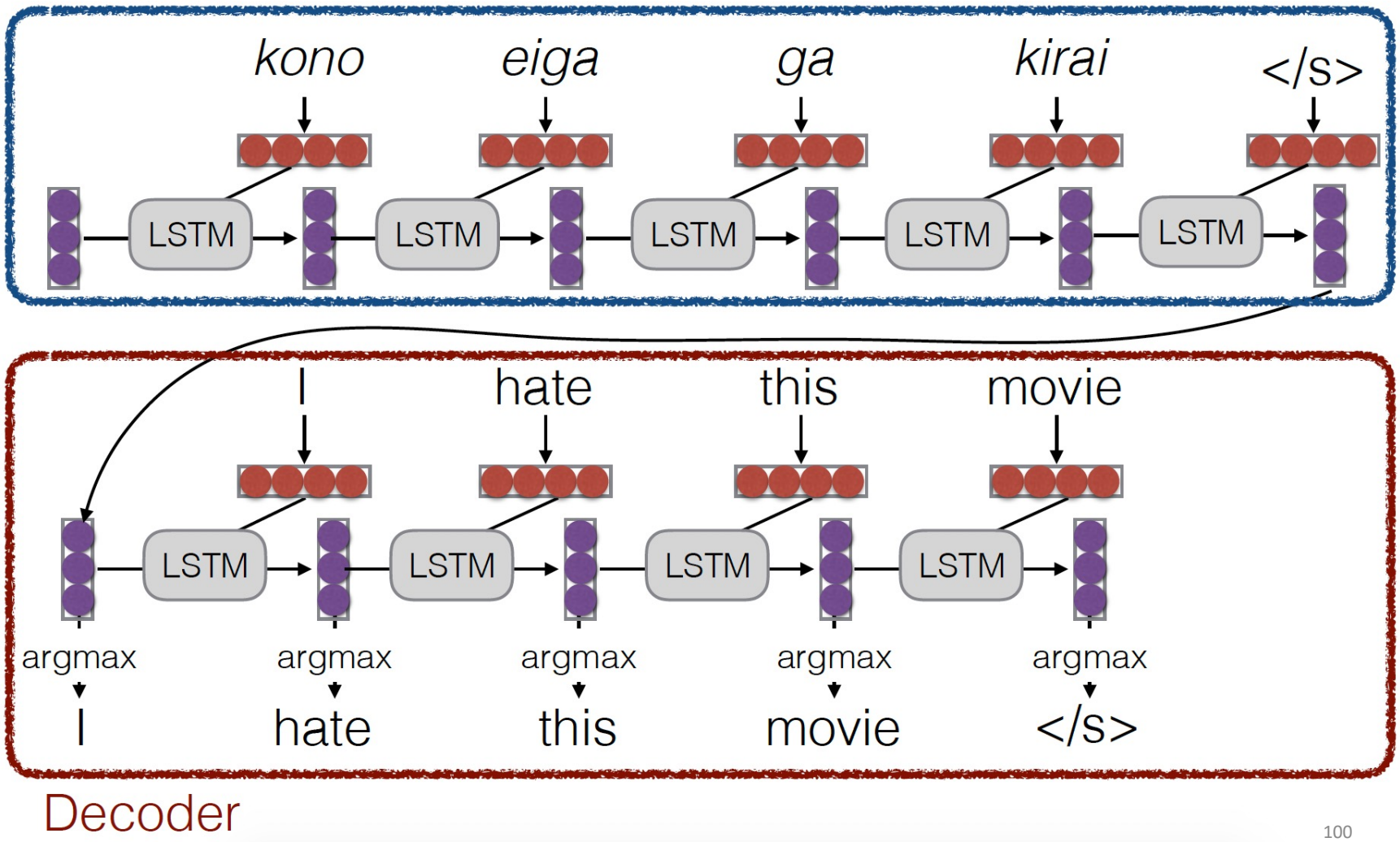
References

- http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture10.pdf
- http://introtodeeplearning.com/materials/2018_6S191_Lecture2.pdf
- https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf
- https://www.cs.ubc.ca/~lsigal/532L/Lecture8.pdf

Attention mechanism, 1D convolution and pre-trained feature extraction

Encoder-decoder Models

(Sutskever et al. 2014)

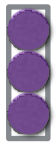


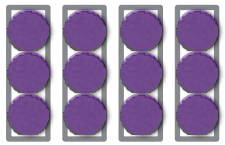
Sentence Representations

Problem!

"You can't cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!"
— Ray Mooney

- But what if we could use multiple vectors, based on the length of the sentence.

this is an example ⟶ 

this is an example ⟶ 

Attention

Why attention?

- Look into distant features
- Combine all the features in the sequence to produce a better feature representation

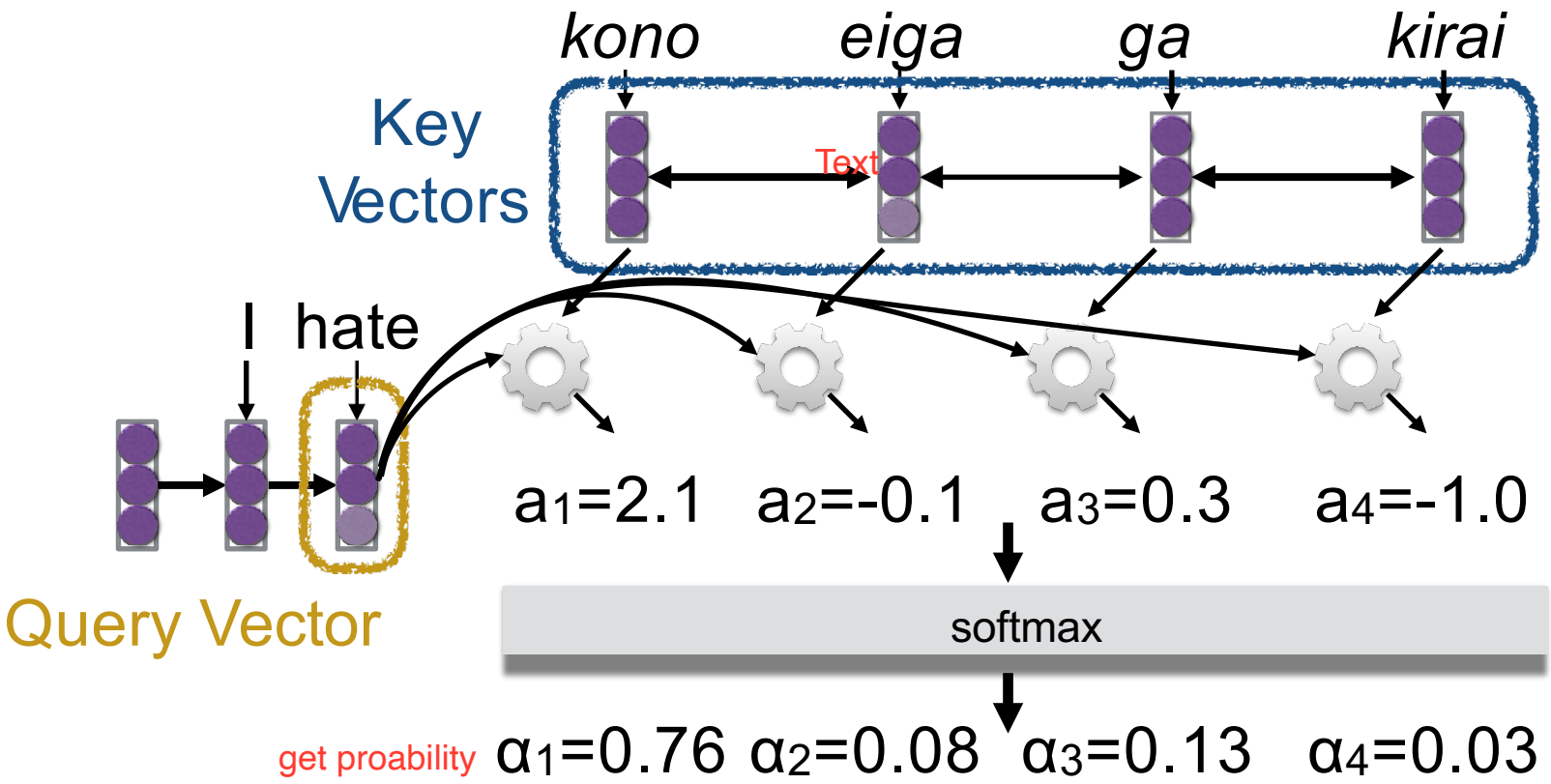
Basic Idea

(Bahdanau et al. 2015)

- Encode each word in the sentence into a vector
- When decoding, perform a linear combination of these vectors, weighted by “attention weights”
- Use this combination in picking the next word

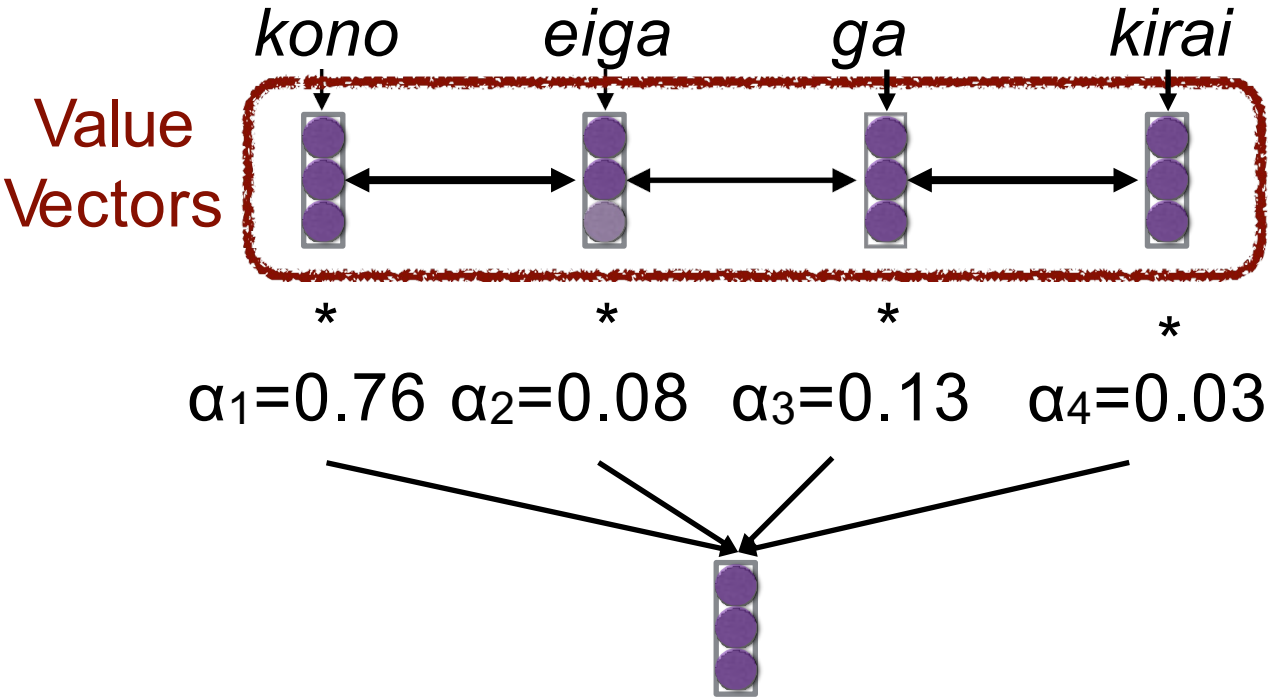
Calculating Attention (1)

- Use “query” vector (decoder state) and “key” vectors (all encoder states)
- For each query-key pair, calculate weight
- Normalize to add to one using softmax



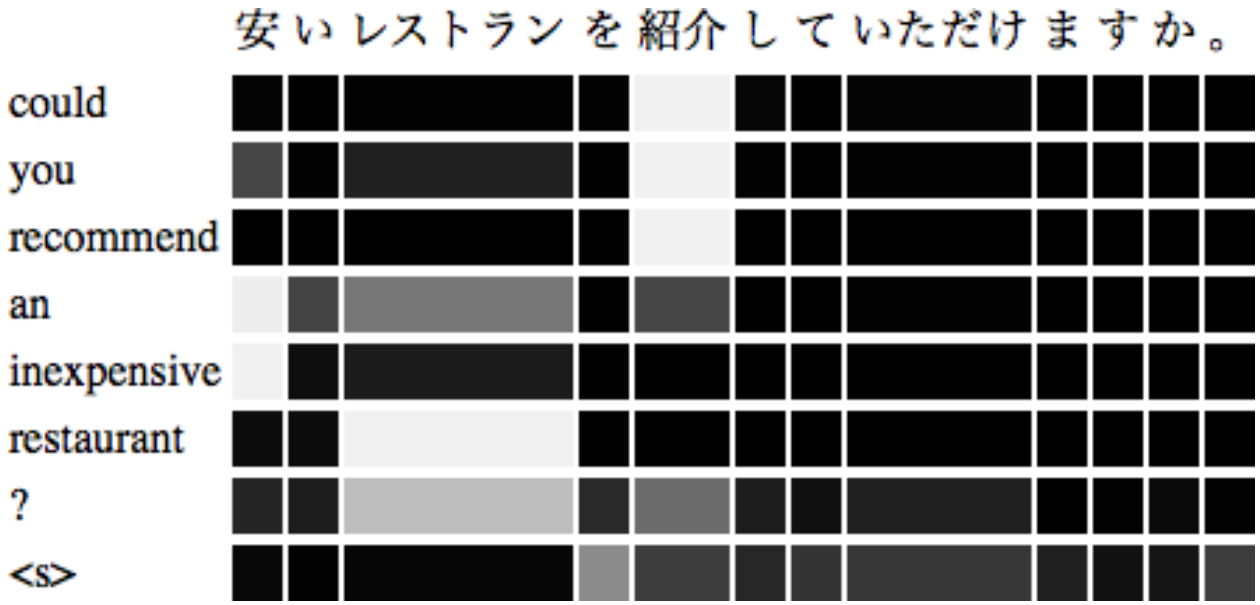
Calculating Attention (2)

- Combine together value vectors (usually encoder states, like key vectors) by taking the weighted sum



- Use this in any part of the model you like

A Graphical Example



Attention Score Functions (1)

- ***q*** is the query and ***k*** is the key
- **Multi-layer Perceptron** (Bahdanau et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^\intercal \tanh(W_1[\boldsymbol{q}; \boldsymbol{k}])$$

  - Flexible, often very good with large data

- **Bilinear** (Luong et al. 2015)

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal W \boldsymbol{k}$$

Attention Score Functions (2)

- **Dot Product** (Luong et al. 2015)
  - No parameters! But requires sizes to be the same.

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\intercal \boldsymbol{k}$$

- **Scaled Dot Product** (Vaswani et al. 2017)
  - Problem: scale of dot product increases as dimensions get larger

$$a(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^\intercal \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$$

  - Fix: scale by size of the vector