Context Sensitive Learning

Sequential Deep Learning Models for NLP

Concepts & Techniques

Dipanjan (DJ) Sarkar

Session Agenda

Classical Language Models

Recurrent Neural Networks

Bidirectional LSTMs

Popular Sequential Deep Learning Model Architectures

Long Short Term Memory Networks

Contextual Embeddings - ELMo

Embedding Layer Representation

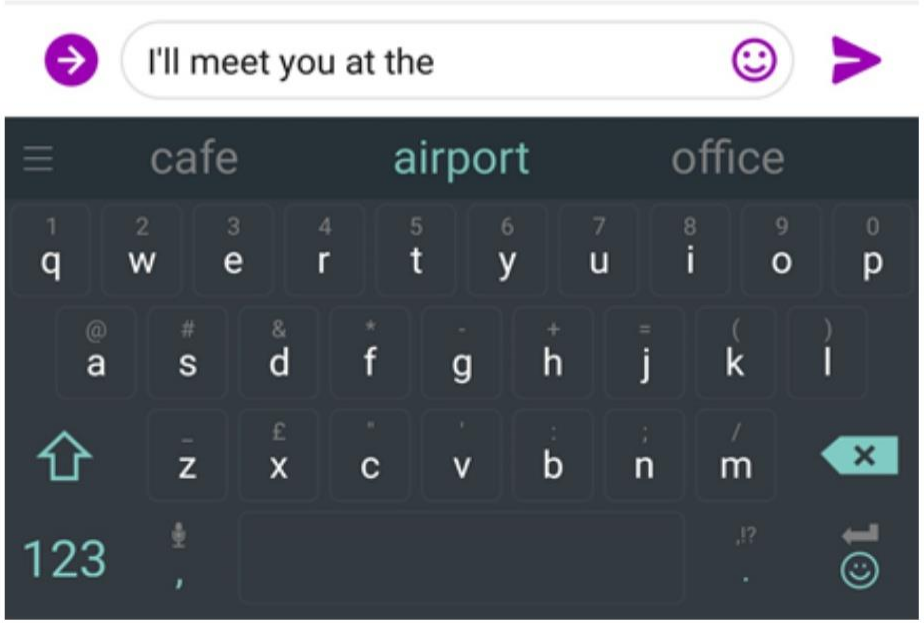
Gated Recurrent Units

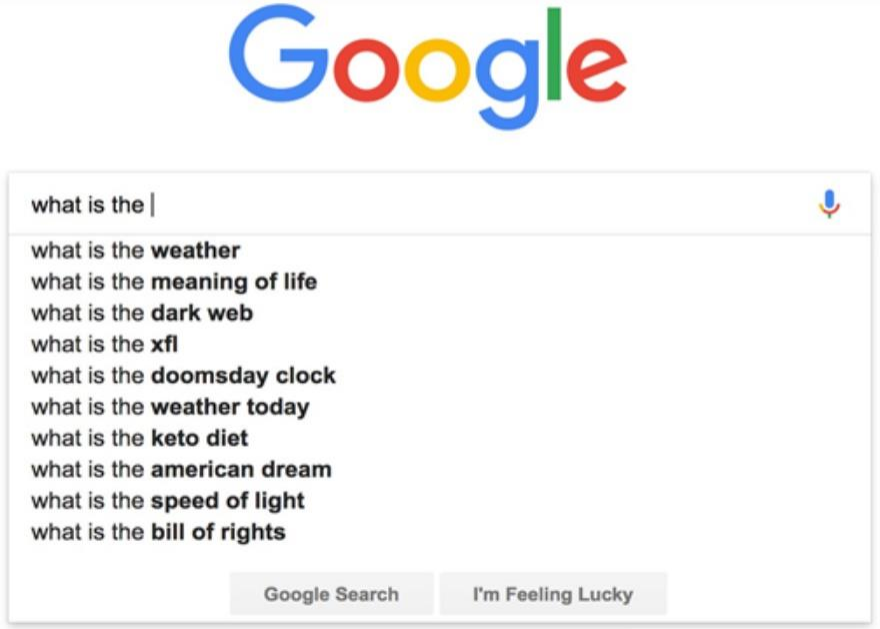
Sequence to Sequence Models

Convolutional Neural Networks


Classical Language
Models

Language models are everywhere!





Classical Language Models

the students opened their _____???

- **Question**: How to learn a Language Model?
- **Answer**: Learn a *n*-gram Language Model!
  (pre- Deep Learning)

Classical Language Models

- Definition: A *n*-gram is a chunk of *n* consecutive words.
  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - 4-grams: “the students opened their”

- Idea: Collect statistics about how frequent different n-grams are, and use these to predict the next word.

N-gram Language Models

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ *students opened their*

discard

condition on this

N-gram Language Models

$$P(\boldsymbol{w}_j|\text{ students opened their }) = \frac{\text{count (students opened their } \boldsymbol{w}_j)}{\text{count (students opened their)}}$$

In the corpus:

- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - P(exams | students opened their) = 0.1

Should we have discarded the "proctor" context?

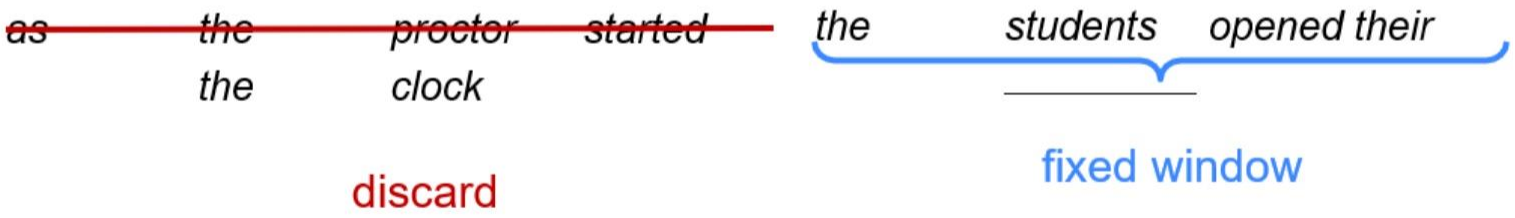
N-gram Language Models - Challenges

$$P(\boldsymbol{w}_j|\text{ students opened their }) = \frac{\text{count (students opened their } \boldsymbol{w}_j)}{\text{count (students opened their)}}$$

Storage:
Need to store count for all possible *n*-grams. So model size is O(exp(*n*)).

Increasing *n* makes model size huge!

N-gram Language Models - Challenges



Fixed-width Neural Language Model

output distribution

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

hidden layer

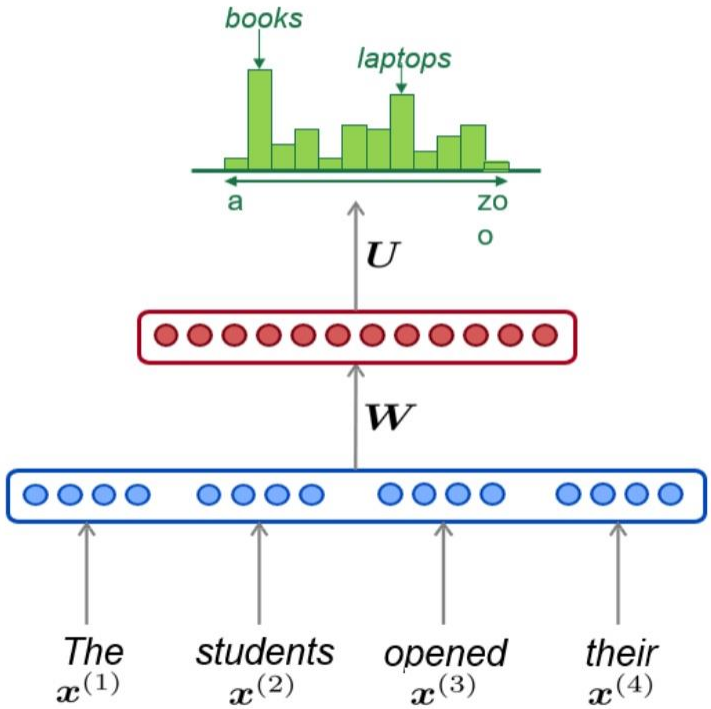
$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

concatenated word embeddings

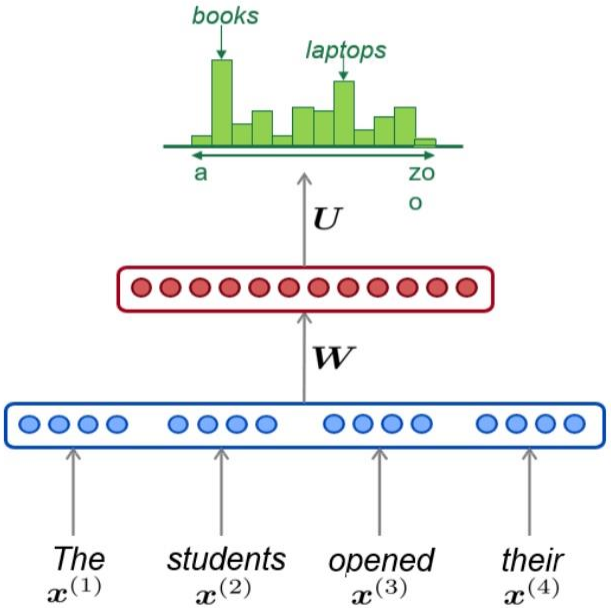
$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Fixed-width Neural Language Model



- **Improvements over n-gram LM**
  - Dense Embeddings vs. Sparse Vectors
  - Model size is O(n) vs. O(exp(n)
- **Challenges**
  - Fixed window is still small
  - Can't process or learn context from longer sequences
  - Enlarging window of words with increase embedding size (W)
  - Weights are not shared


Popular Sequential Deep
Learning Model Architectures

Sequential Deep Learning Model Architectures

1. **Convolutional Neural Networks (CNNs)**

   Used extensively in computer vision problems with image, video. Can also be used for audio and text

2. **Recurrent Neural Networks**

   Good for sequential data, used for time series forecasting and NLP problems

3. **Long Short Term Memory Networks (LSTMs)**

   Can remember longer sequences of data and better than RNNs

4. **Gated Recurrent Units (GRUs)**

   Can remember longer sequences of data and faster than LSTMs

5. **Bi-directional Models**

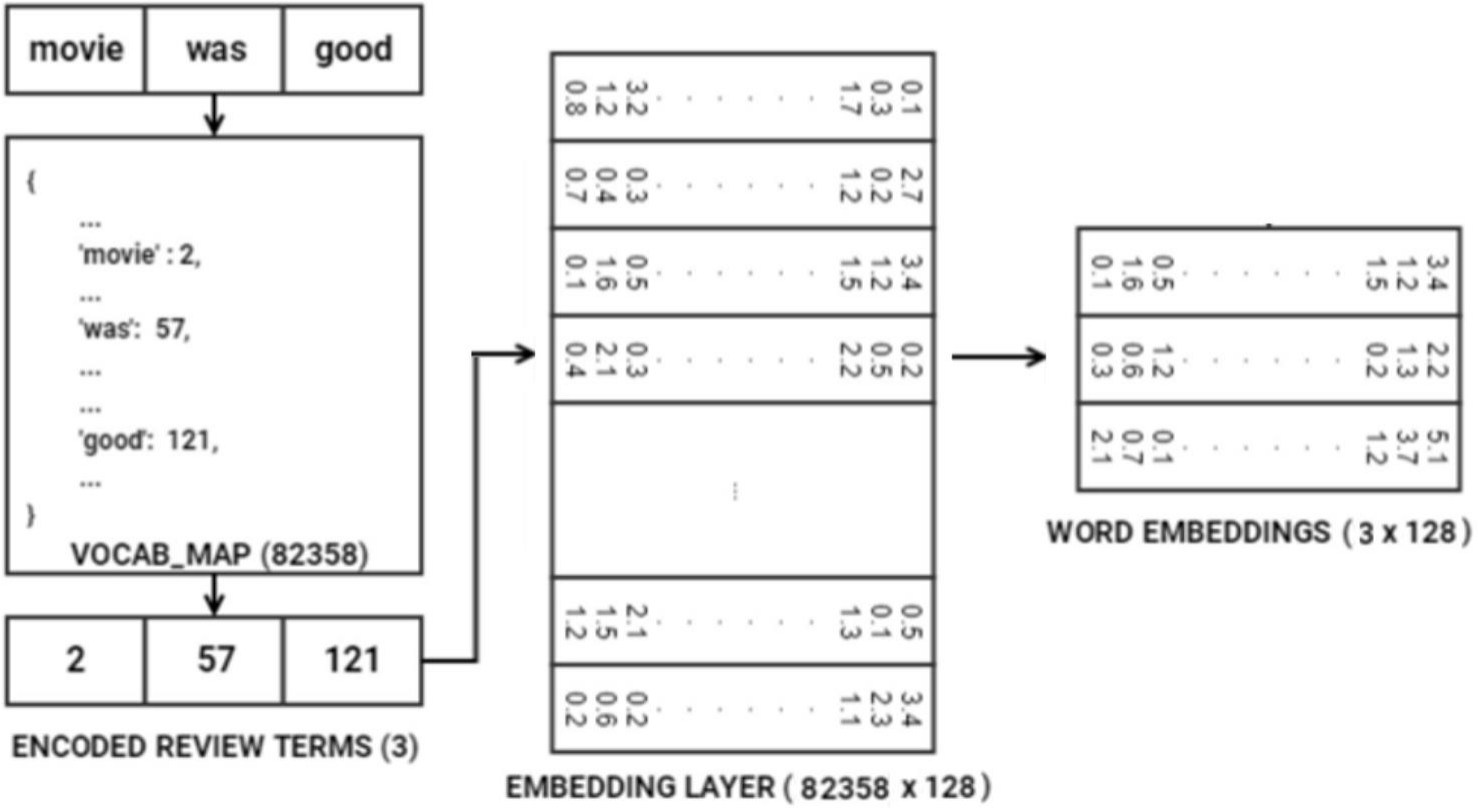
   Processes sequences of data in both directions for capturing better contextual information

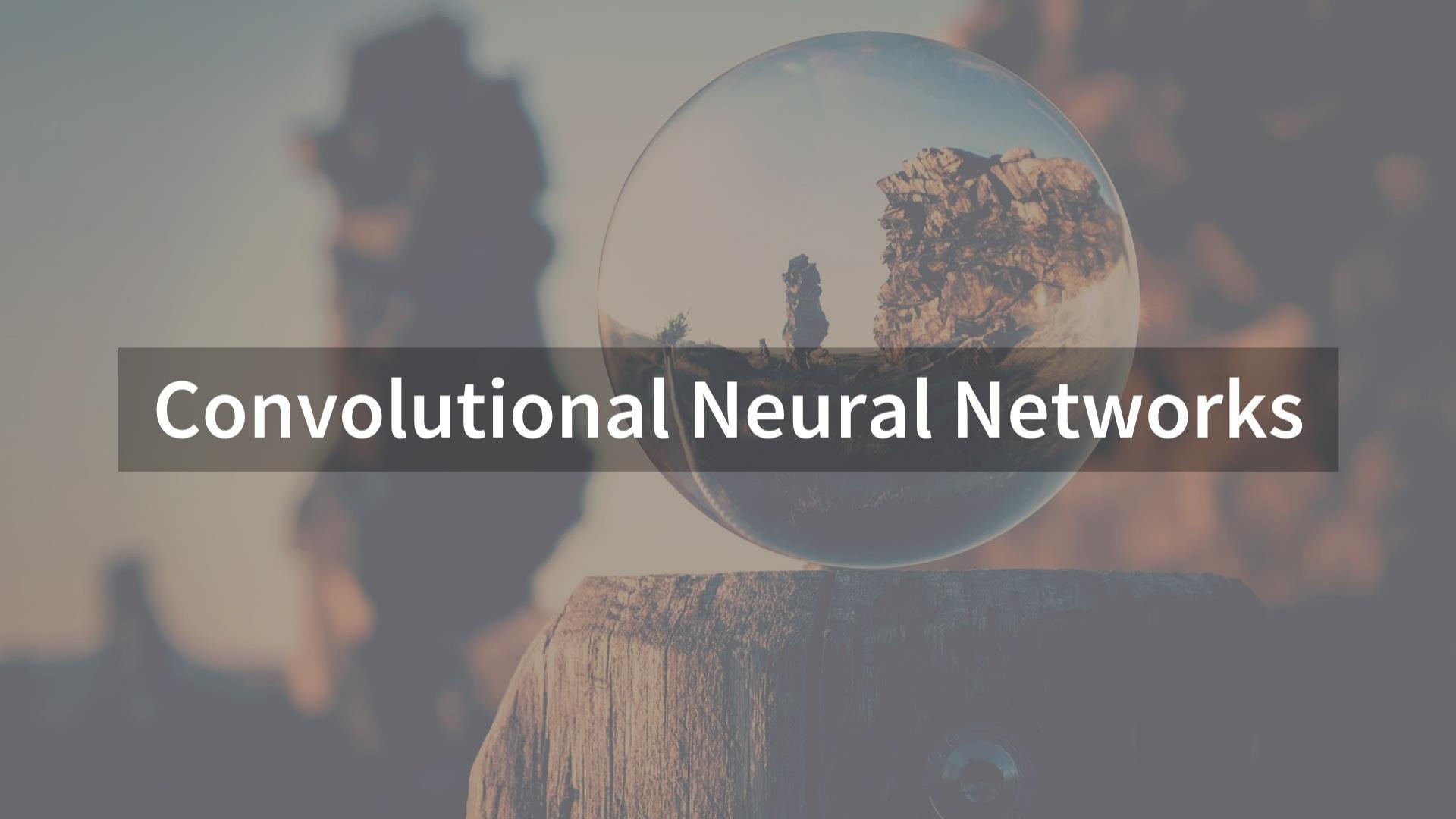
6. **Encoder-Decoder Models**

   Takes in a sequence of data and generates a sequence of data as output

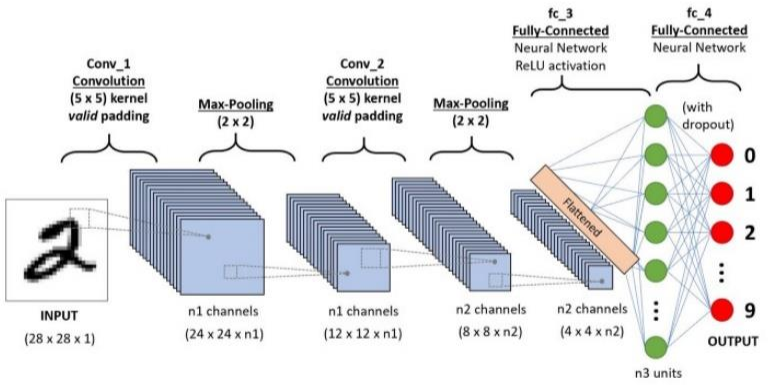
Embedding Layer Representation

Embedding Layer



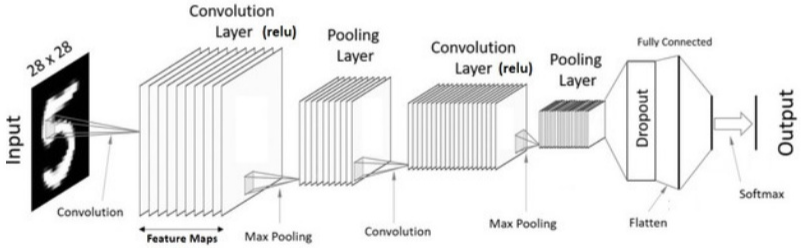

Convolutional Neural Networks

Convolutional Neural Networks (CNNs)



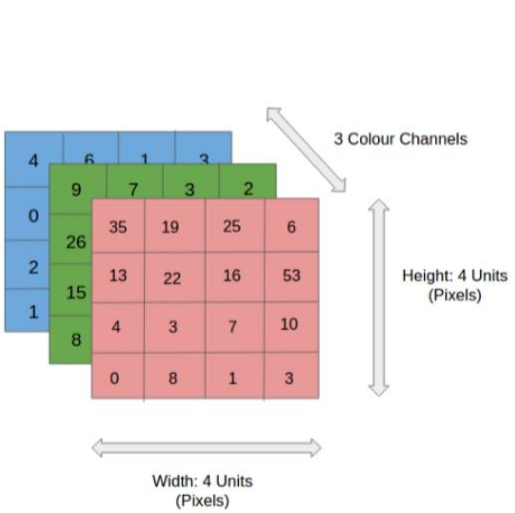
- CNNs have a layered architecture of several layers to learn hierarchical spatial features
- Convolution Layers use convolution filters to build feature maps (feature extraction)
- Pooling Layers help in reducing dimensionality after convolutions (compression)
- Non-linear activation functions are applied in the network as usual
- Dropout or BatchNormalization Layers may be used to prevent model overfitting
- FC Layers in final stages help with flattening and prediction

CNN - Main Layer Components

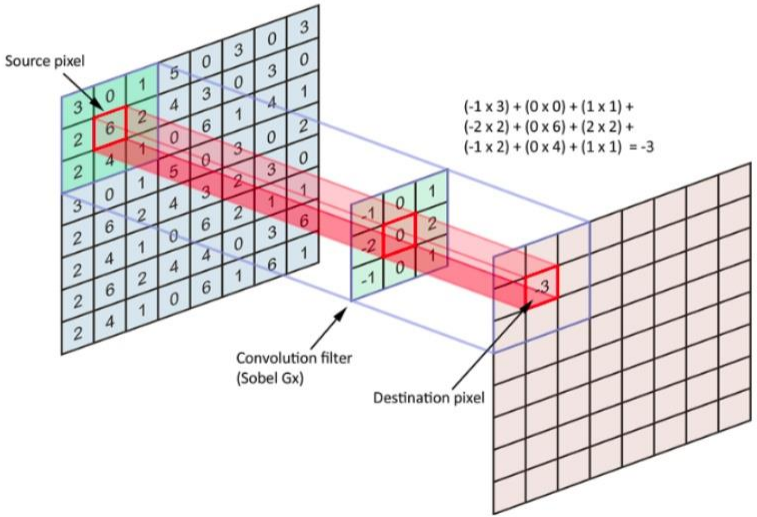


- **CNNs have a stacked layered architecture of several convolution and pooling layers**
- **Convolution layer**
  - Consists of several filters or kernels
  - Passed over the entire image in patches and computes a dot product
  - Result is summed up into one number per operation (dot product)
- **Pooling layer**
  - Downsamples feature maps from conv layers
  - Typically max-pooling is used which selects the max-pixel value out of a patch of pixels
- **Activation layer**
  - Feature maps \ pooled outputs are sent through non-linear activations
  - Introduces non-linearity and helps train via. backpropagation

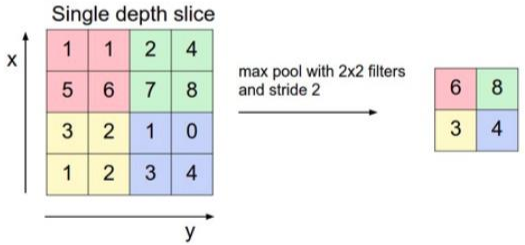
CNN - Conv - Pool Operations



Source Image



Convolution Layer



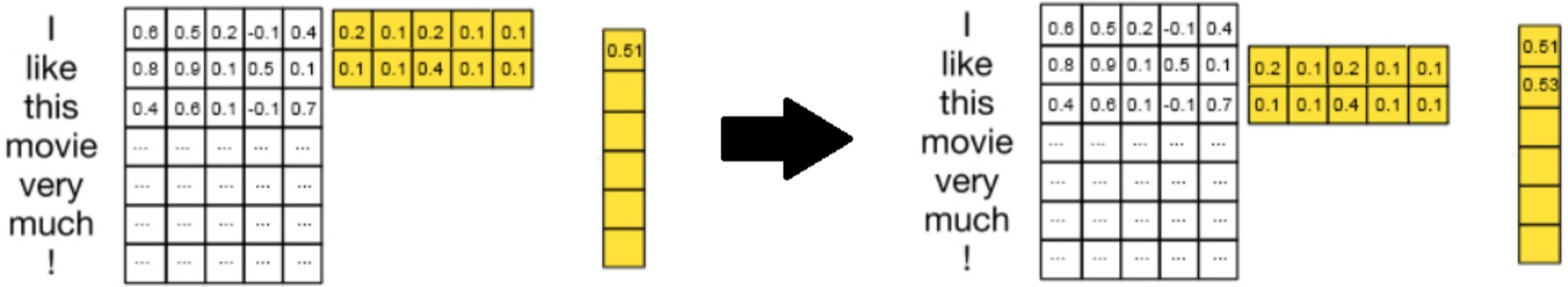
Pooling Layer

1D - Convolutions

```
>>> # The inputs are 128-length vectors with 10 timesteps, and the batch size
>>> # is 4.
>>> input_shape = (4, 10, 128)
>>> x = tf.random.normal(input_shape)
>>> y = tf.keras.layers.Conv1D(
... 32, 3, activation='relu',input_shape=input_shape[1:])(x)
>>> print(y.shape)
(4, 8, 32)
```

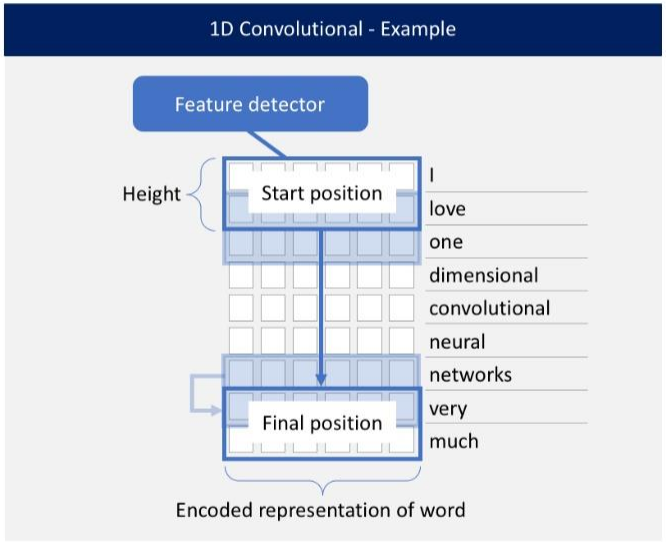
Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv1D

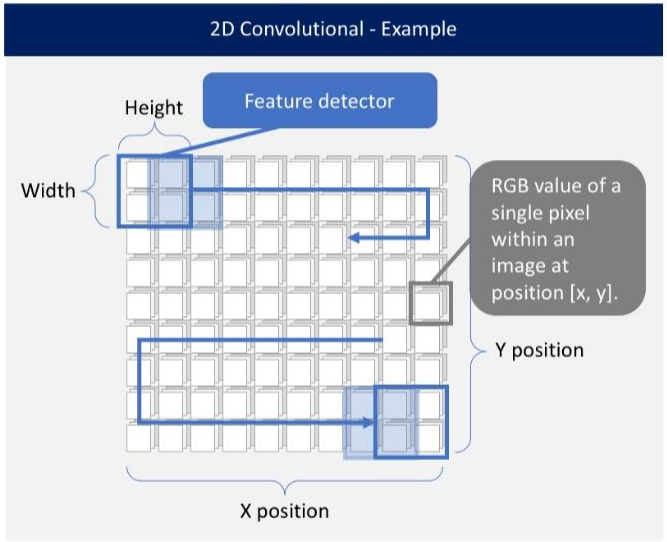
1D - Convolutions



Source: http://www.joshuakim.io/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-embeddings/

1D vs. 2D - Convolutions



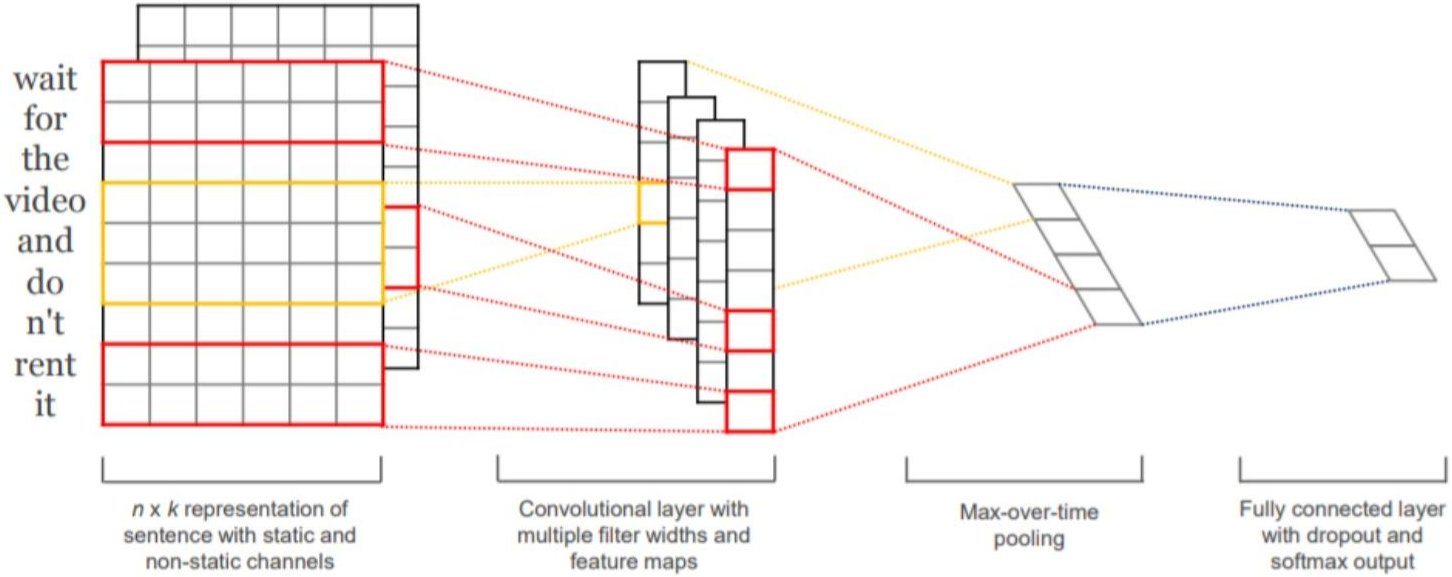


In this example for natural language processing, a sentence is made up of 9 words. Each word is a vector that represents a word as a low dimensional representation. The feature detector will always cover the whole word. The height determines how many words are considered when training the feature detector. In our example, the height is two. In this example the feature detector will iterate through the data 8 times.

In this example for computer vision, each pixel within the image is represented by its x- and y position as well as three values (RGB). The feature detector has a dimension of 2 x 2 in our example. The feature detector will now slide both horizontally and vertically across the image.

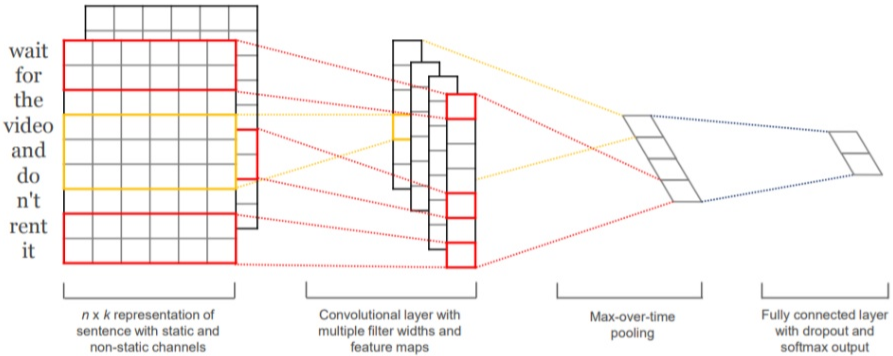
"1D versus 2D CNN" by Nils Ackermann

CNNs for Text Classification



Source: Convolutional Neural Networks for Sentence Classification

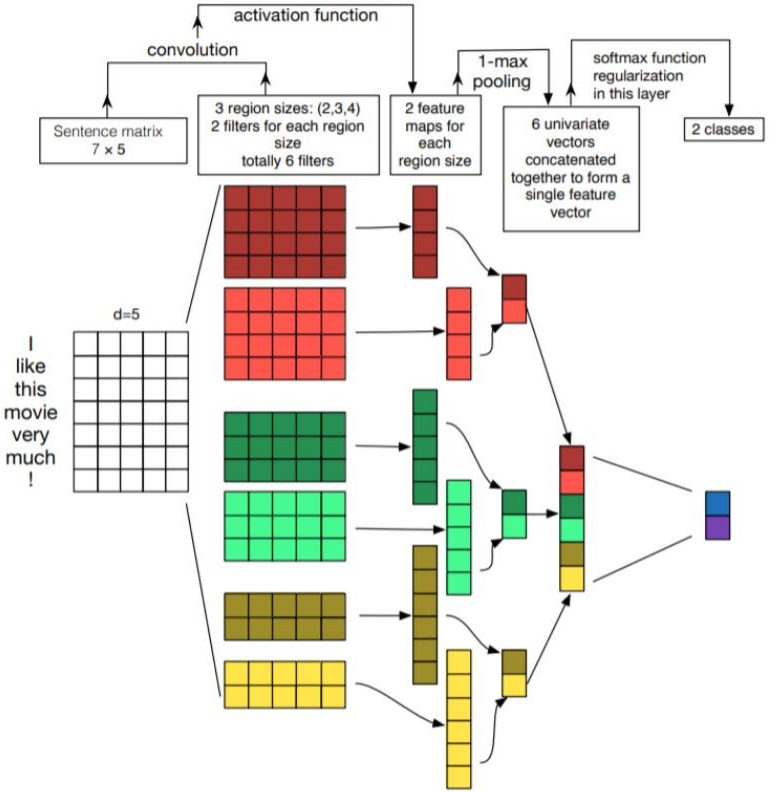
CNNs for Text Classification

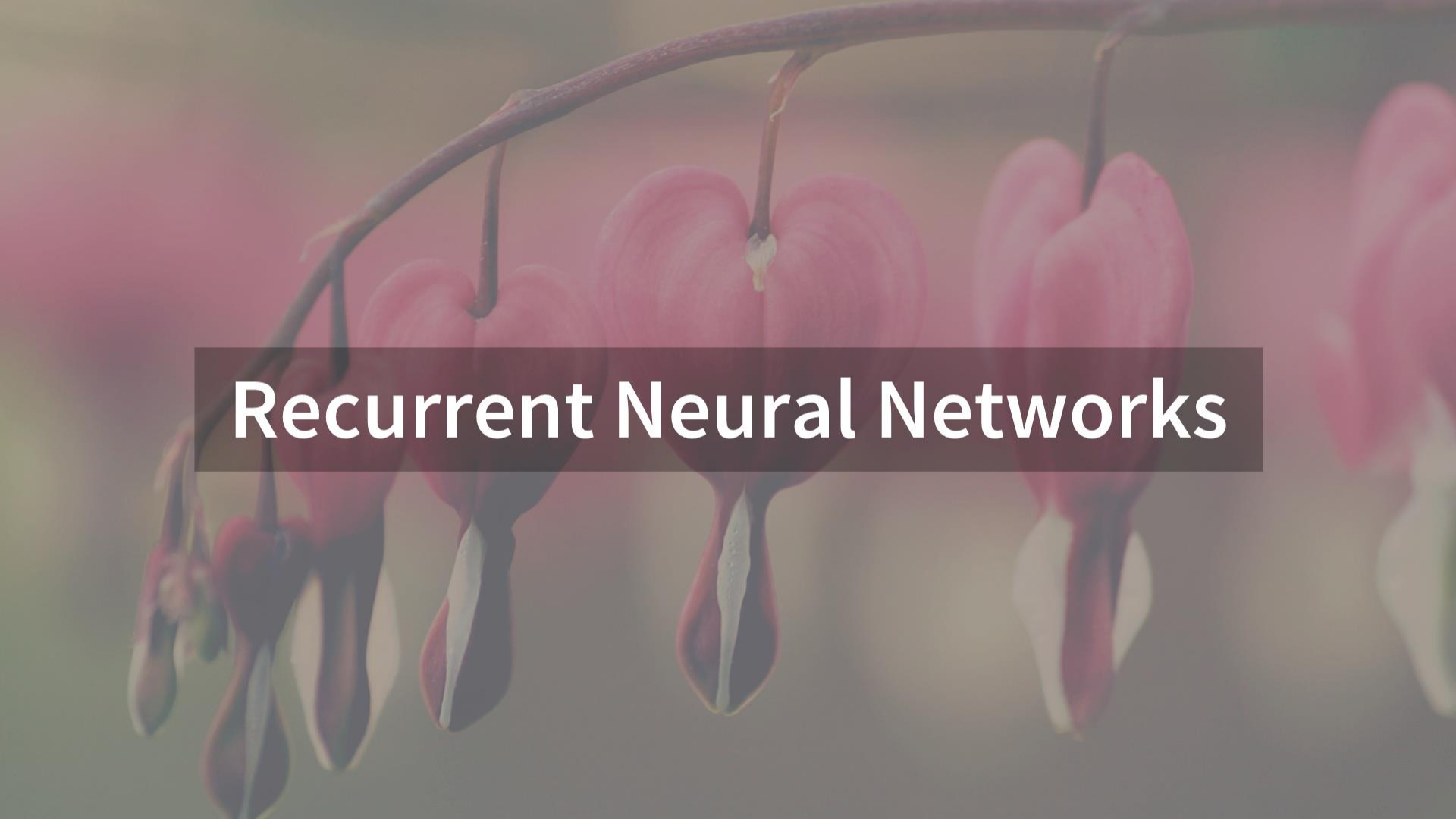


- Embedding layer embeds words into dense vectors
- 1D Conv layer performs convolutions over the embedded word vectors

  Acts as a sliding window over words based on size of the filter
- Max-pool 1D helps in downsampling feature maps (1D)
- Combined flat feature vector passes through a softmax layer for classification

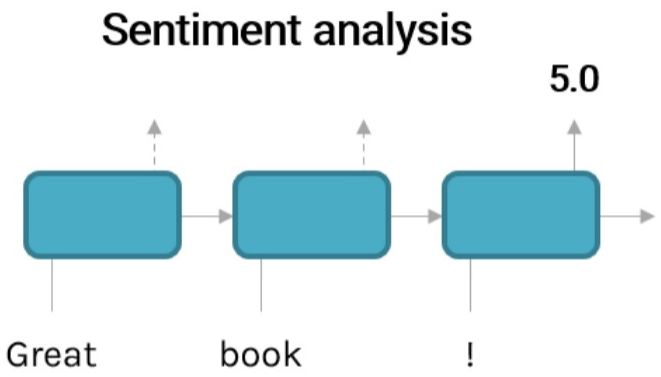
CNNs for Text Classification

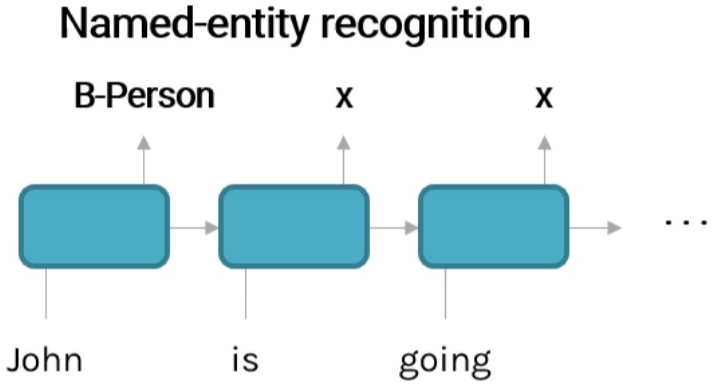


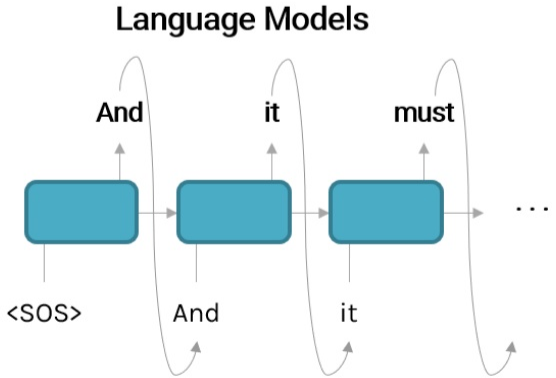


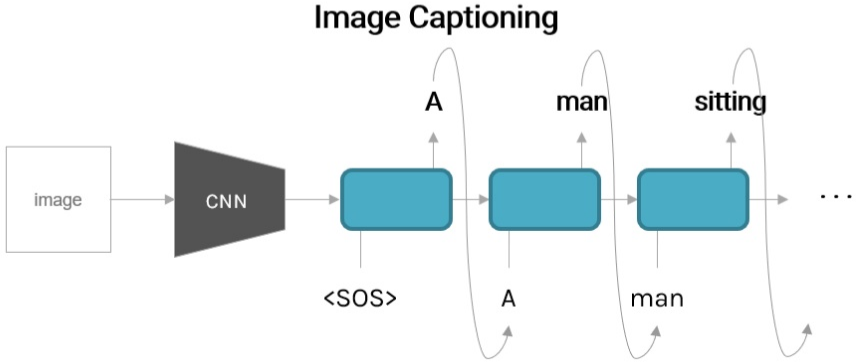
Recurrent Neural Networks

Recurrent Neural Networks



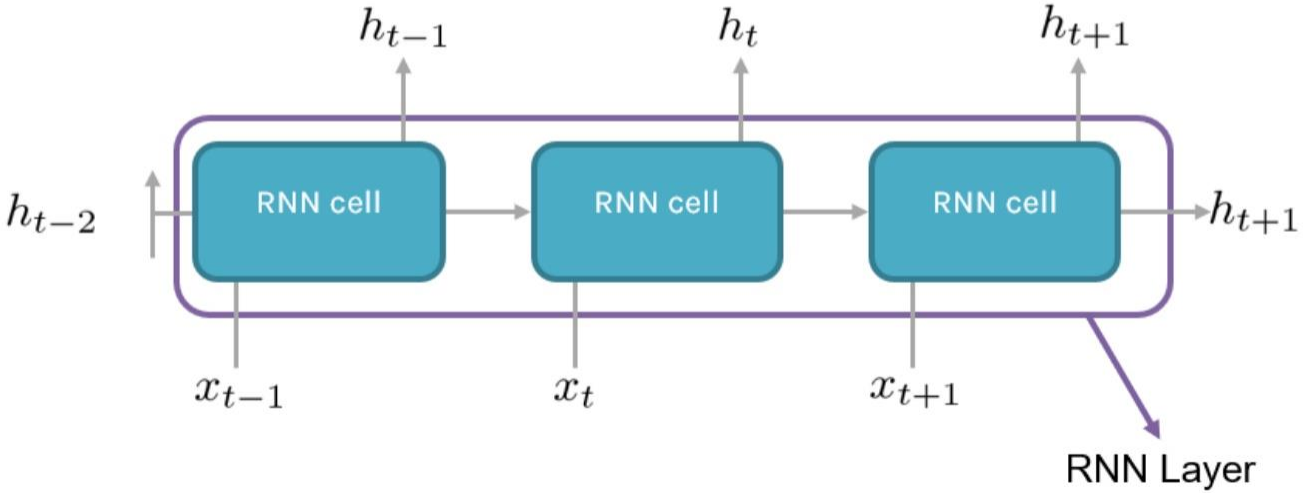






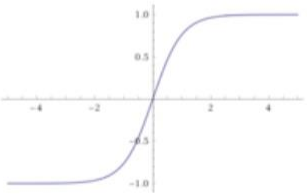
RNN Layer Architecture

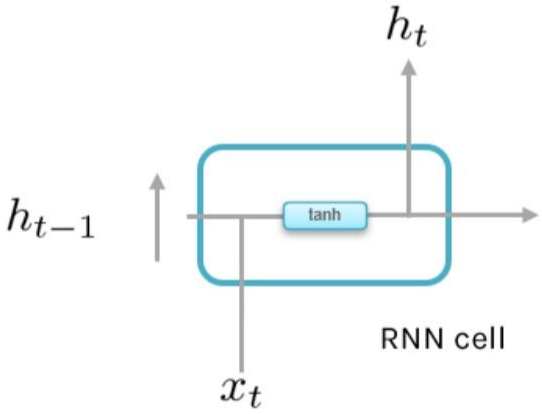
Can process longer sequences of fixed or variable length



RNN Layer Architecture

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$





RNN Language Models

output distribution

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

hidden states

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

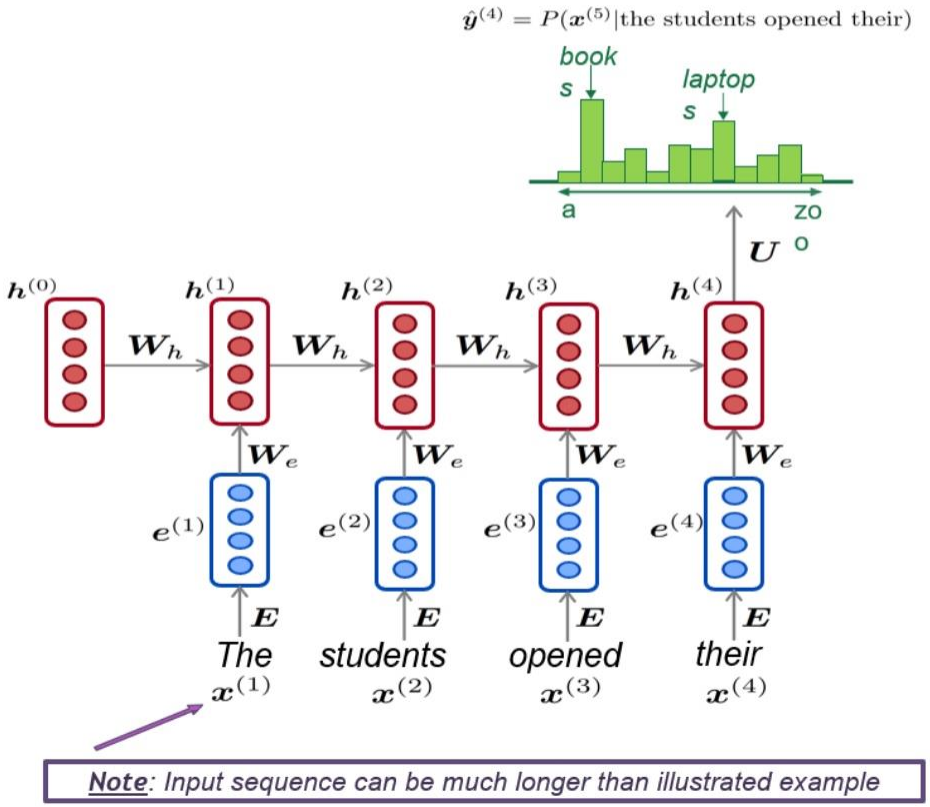
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

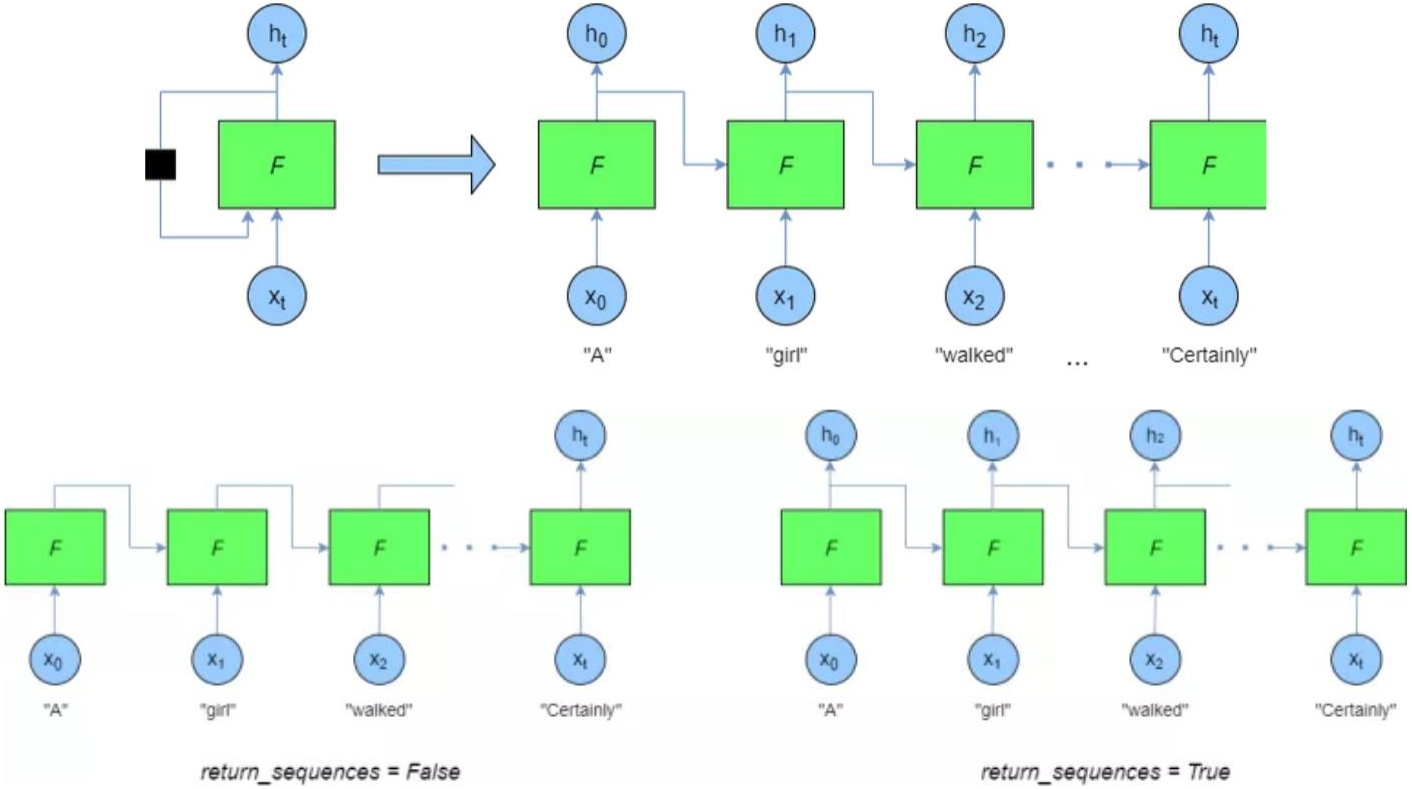
$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$

words / one-hot vectors

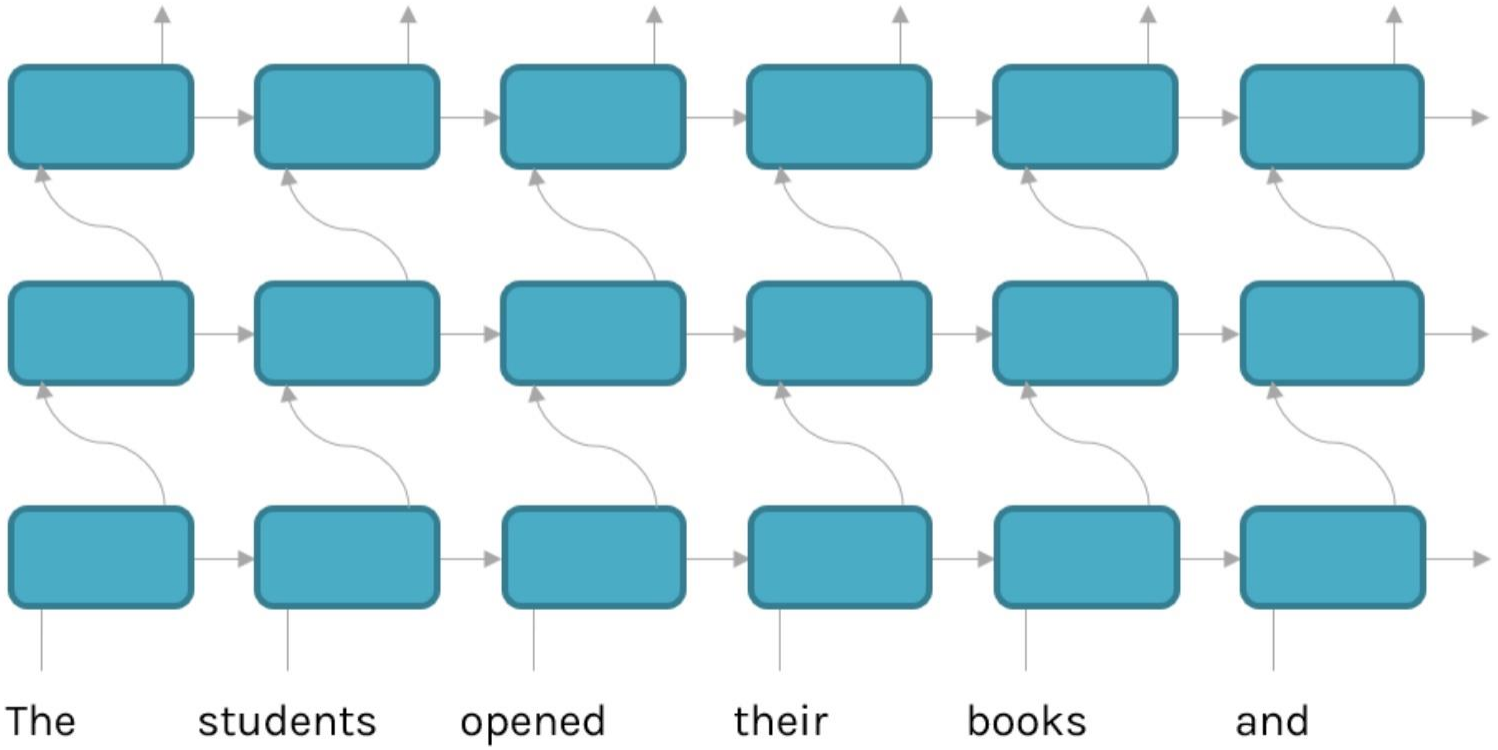
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$

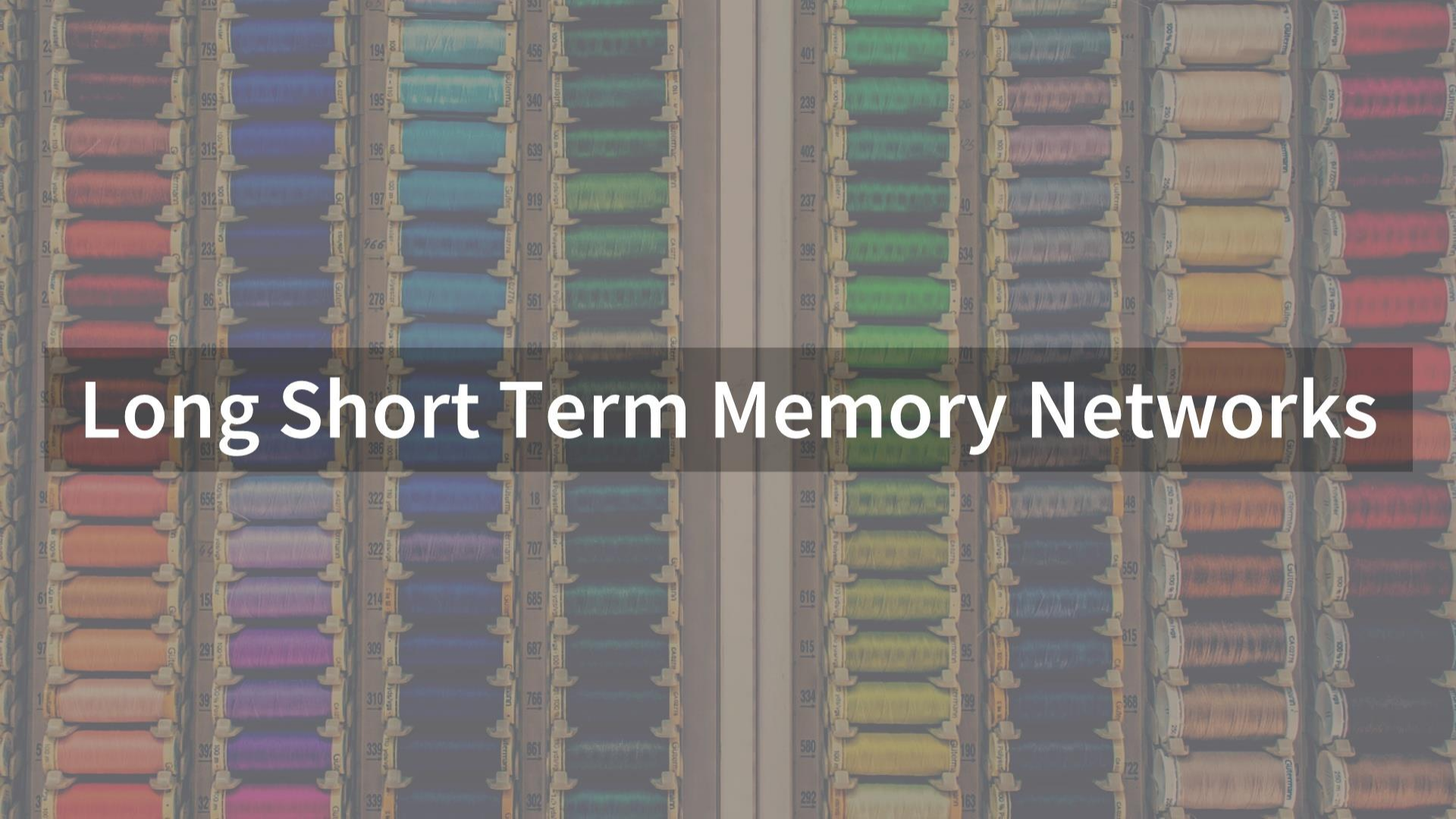


Sequential Models (RNN \ LSTM \ GRU)

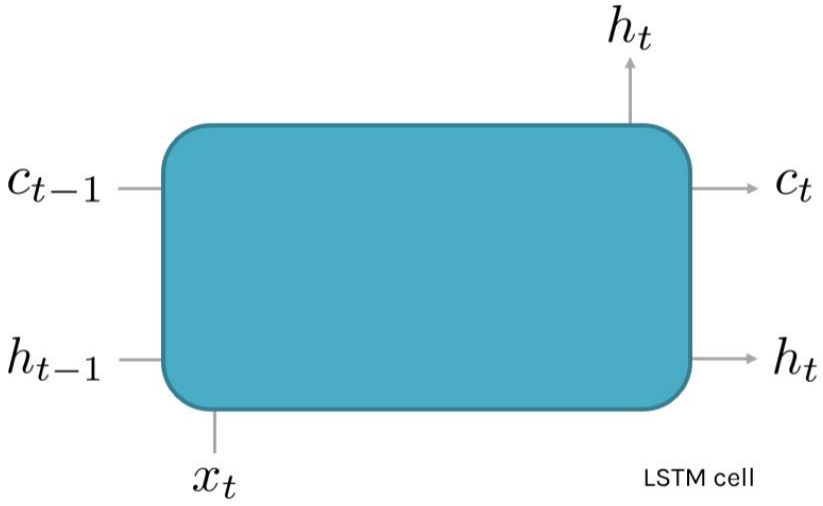


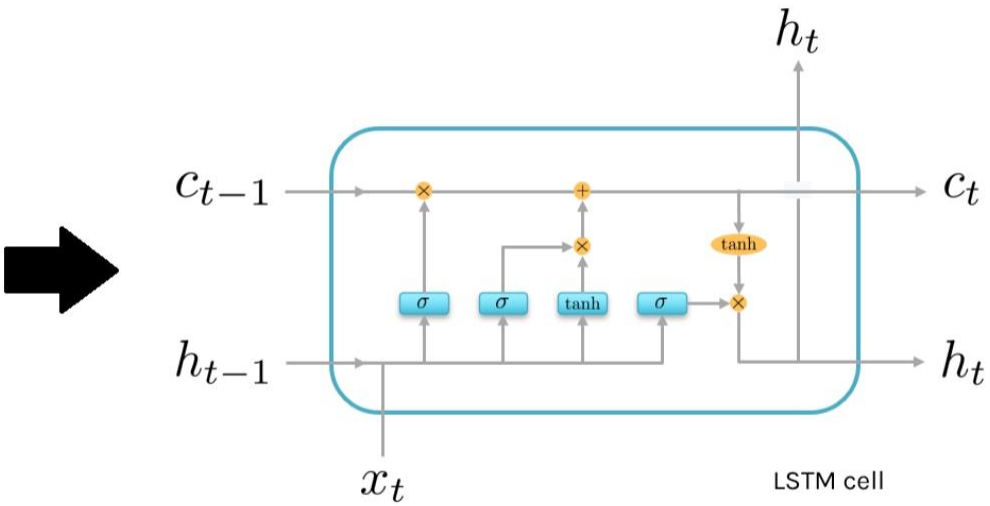
Stacked RNNs




Long Short Term Memory Networks

Unboxing the LSTM cell

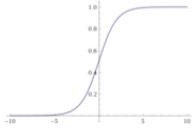




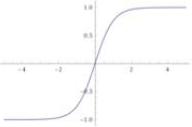
= dense layer

= pointwise operation

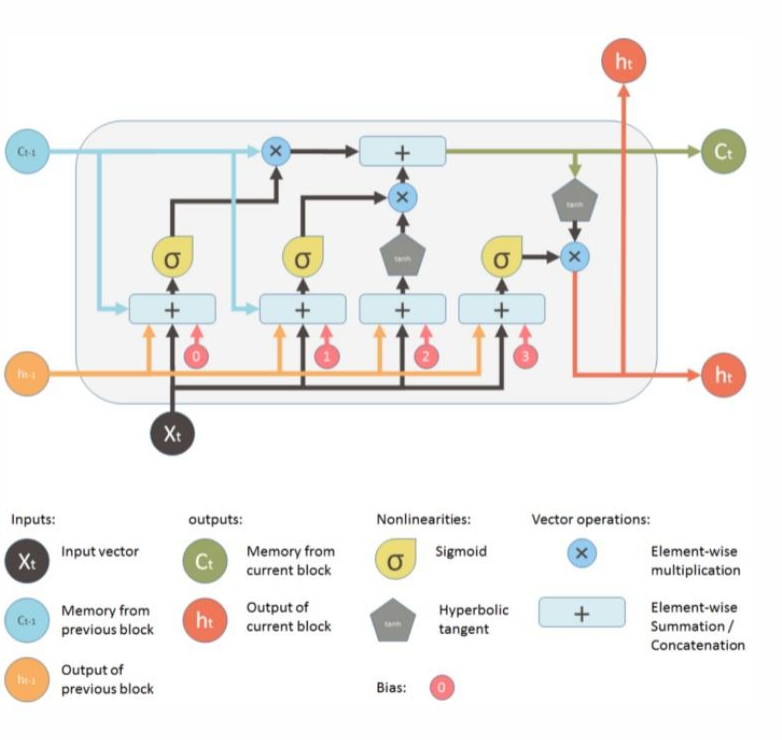
$$\sigma(x) = \frac{e^x}{e^x + 1}$$



$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



Sequential Models - LSTM



- LSTMs can remember longer sequences of data than RNNs
- The network takes three inputs
  - X_t is the input of the current time step
  - h_{t-1} is the output from the previous LSTM unit
  - C_{t-1} is the "memory" or cell state of the previous unit
- The forget gate decides what is relevant to retain from previous steps
- The input gate decides what information can be added from the current step
- The output gate decides what the next hidden state should be

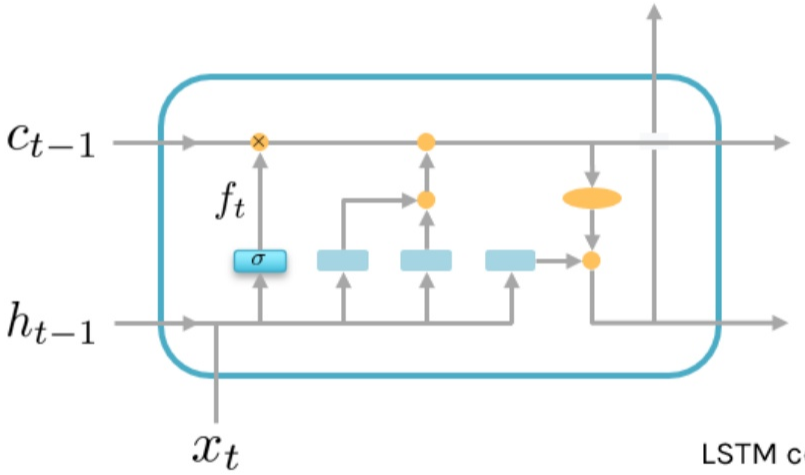
LSTM cell components

The **forget** gate

= dense layer

= pointwise operation

- Sigmoid function crushes the input between 0 and 1
- Decides what information can be thrown away from the cell state



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

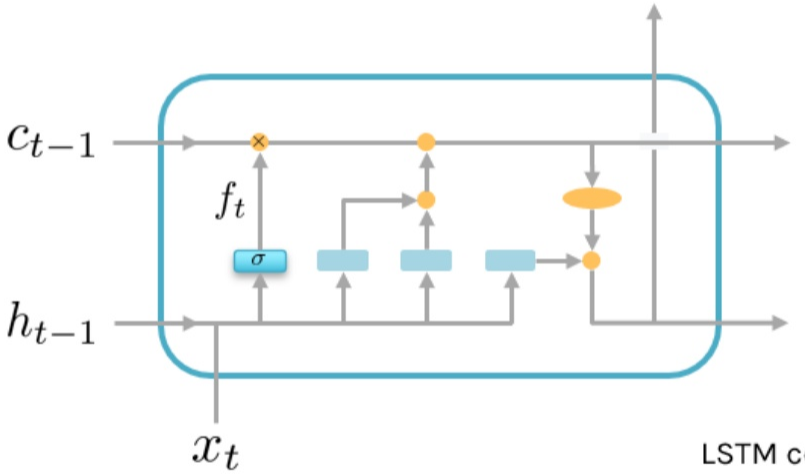
LSTM cell components

The **forget** gate

= dense layer

= pointwise operation

- Sigmoid function crushes the input between 0 and 1
- Decides what information can be thrown away from the cell state



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

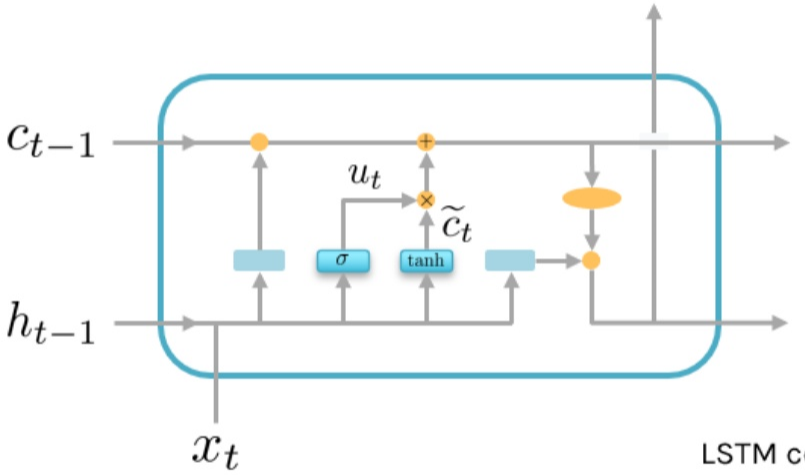
LSTM cell components

The **input \ update** gates

= dense layer

= pointwise operation

- **Sigmoid layer** decides which values to update
- **Tanh layer** creates a vector of new candidates, that could be added to the state



LSTM cell

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

$$\widetilde{c}_t = \tanh(W_C[h_{t-1}, x_t] + b_c)$$

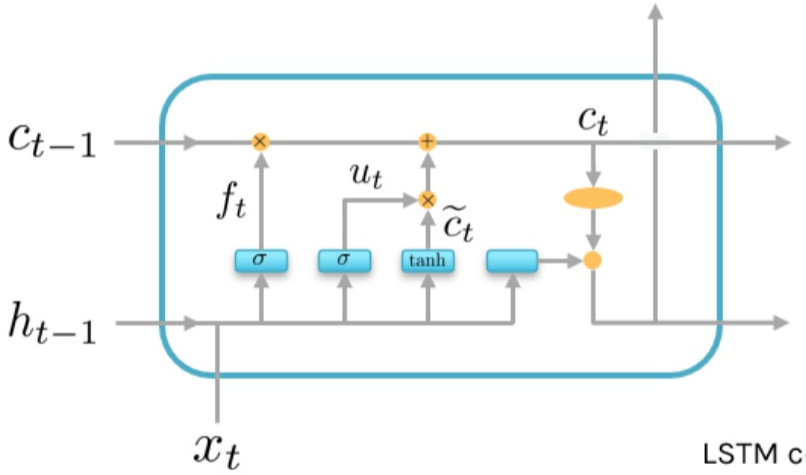
LSTM cell components

Updating c(t)

= dense layer

= pointwise operation

- Multiply the old state by ft to forget the things we decided to forget earlier
- Add ut∗Ct to the cell state, this adds new information



LSTM cell

$$c_t = f_t * c_{t-1} + u_t * \tilde{c}_t$$

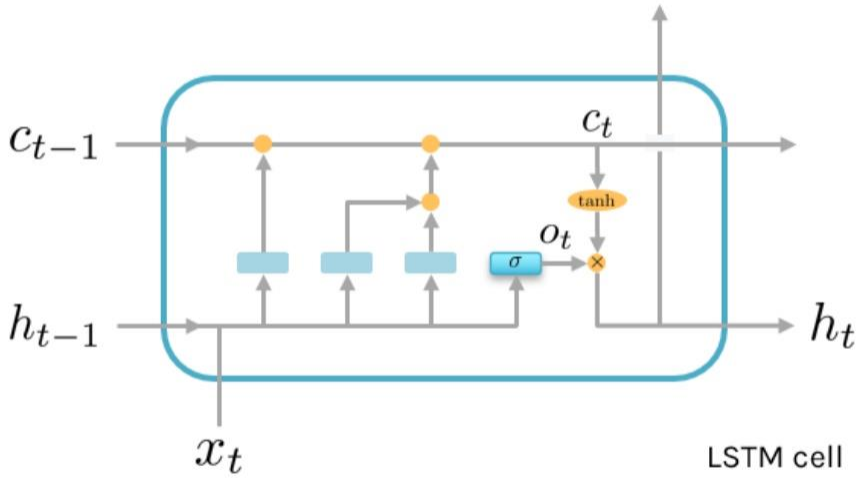
LSTM cell components

The **output** gate

Updating h(t)

= dense layer

= pointwise operation



LSTM cell

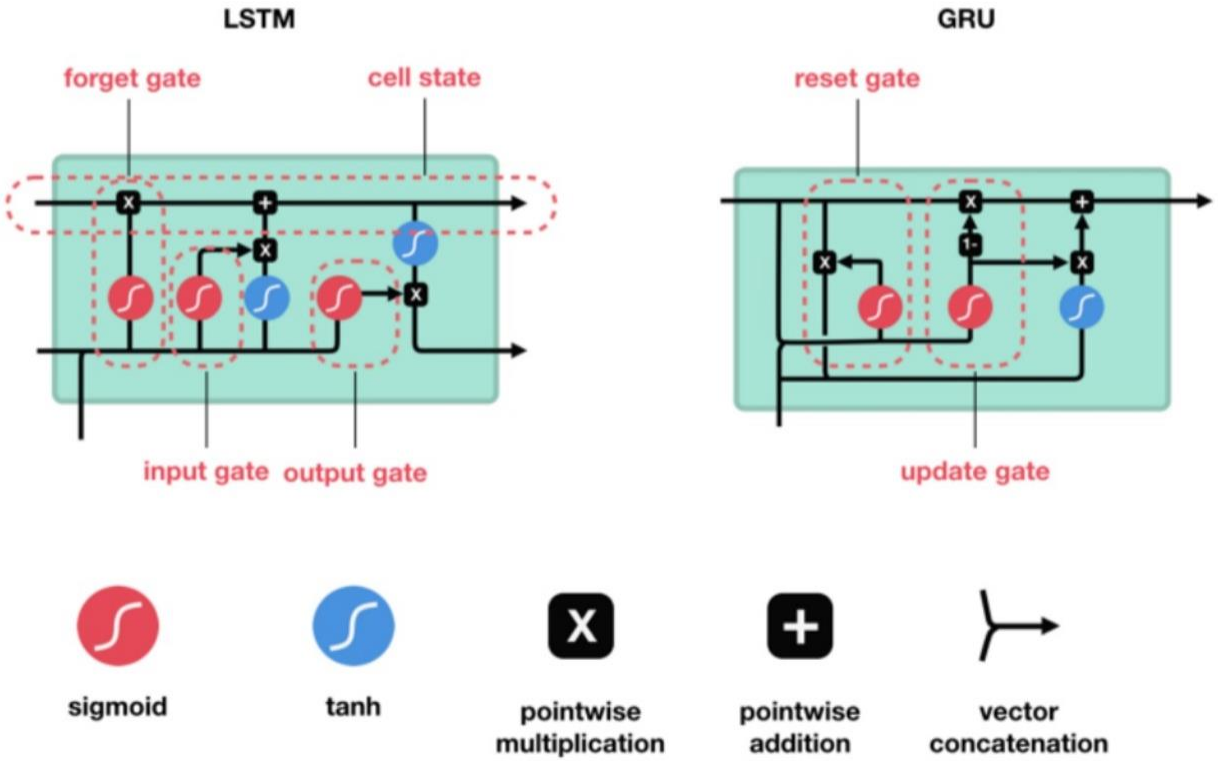
- Selects particular output for next step
- Tanh crushes values between (-1, 1)
- sigmoid helps to select which information is relevant

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

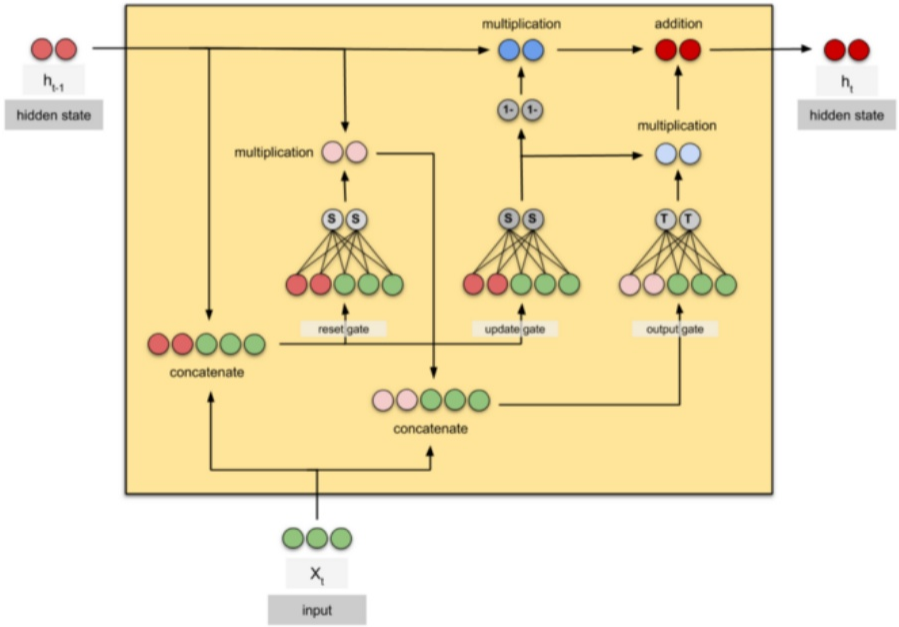
$$h_t = o_t * \tanh(c_t)$$


Gated Recurrent Units

LSTMs vs. GRUs

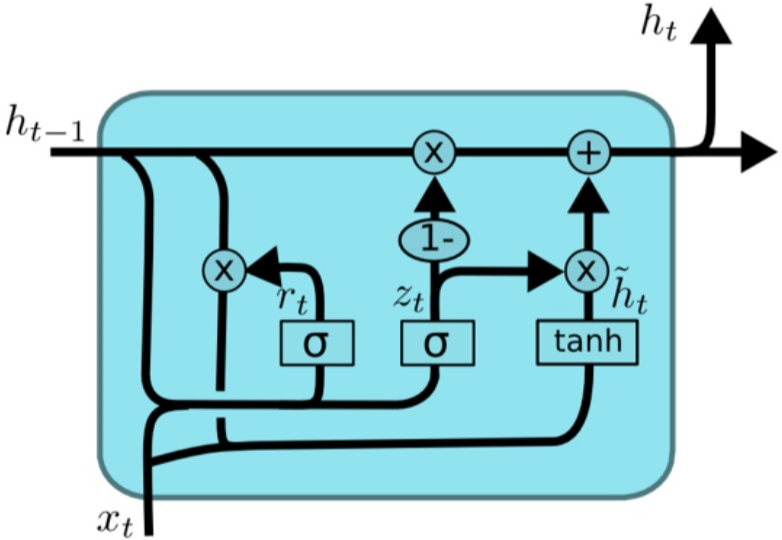


GRU - The subtle differences



- The update gate acts similar to the forget and input gate of an LSTM
- The update gate decides what information to throw away and what new information to add
- The reset gate decides how much past information should be retained
- Fewer tensor ops and speedier to train than LSTMs

GRU cell components



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

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

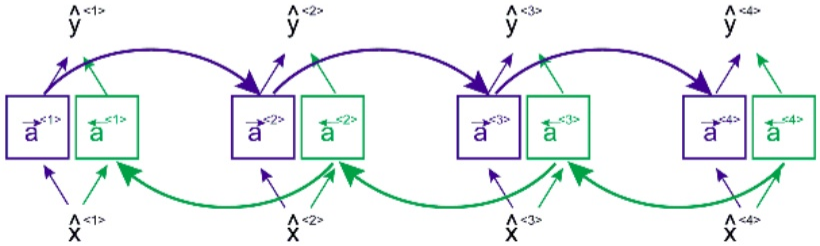
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

- Combines the forget and input gates into a single update gate
- Merges the cell state and hidden state
- Simpler than standard LSTM models, but can give better or worse performance than LSTMs


Bi-directional LSTMs

The need for Bi-directional LSTMs



- **Bi-directional LSTMs are just putting two independent LSTMs together**

  One Bi-directional LSTM layer consists of *two LSTM layers* inside it

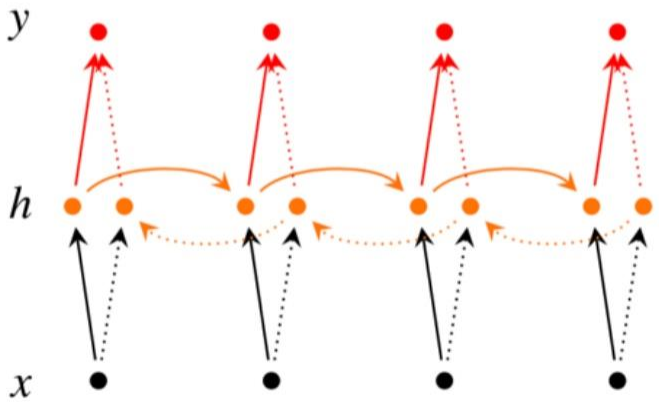
- **The input sequence is fed in forward order for one LSTM layer, and in reverse order for the other LSTM layer**

- **The outputs of the two networks are usually concatenated at each time step**

- **Preserving information from both past and future helps understand context better**

Bi-directional LSTMs

Problem: Incorporate information from words both preceding and following in a sequence of words



$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$

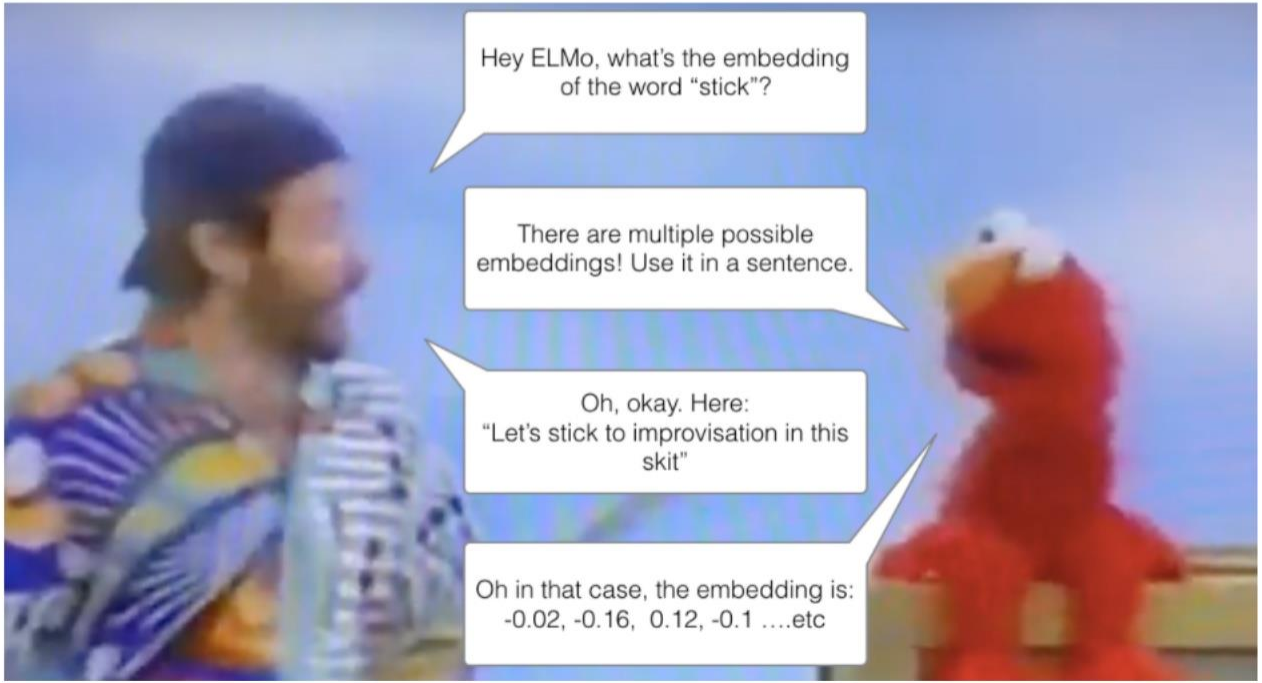
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t] + c)$$

$h = [\vec{h}; \overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

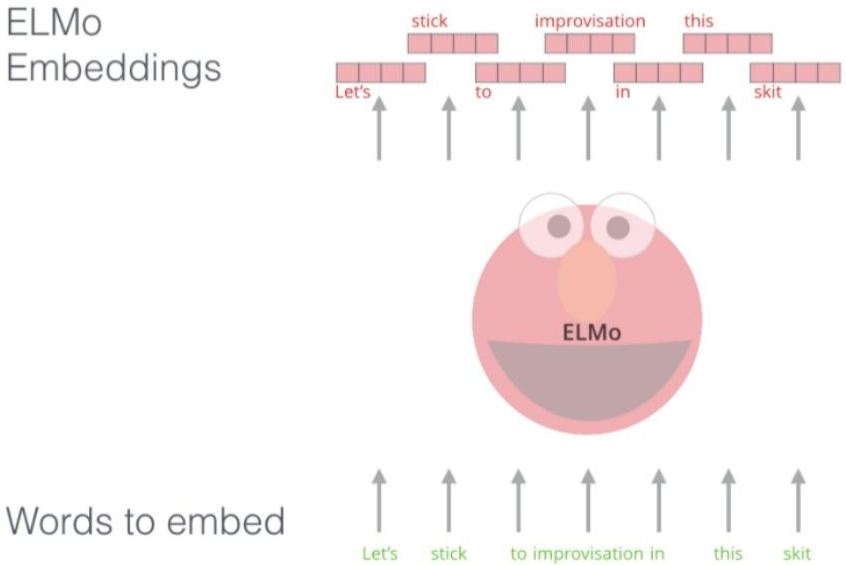

Contextual Embeddings - ELMo

Need for contextualized word embeddings



Source: http://jalammar.github.io/illustrated-bert

Contextual Embeddings with ELMo



Source: http://jalammar.github.io/illustrated-bert/

- **Word embedding models assign a fixed-width embedding representation for each word**

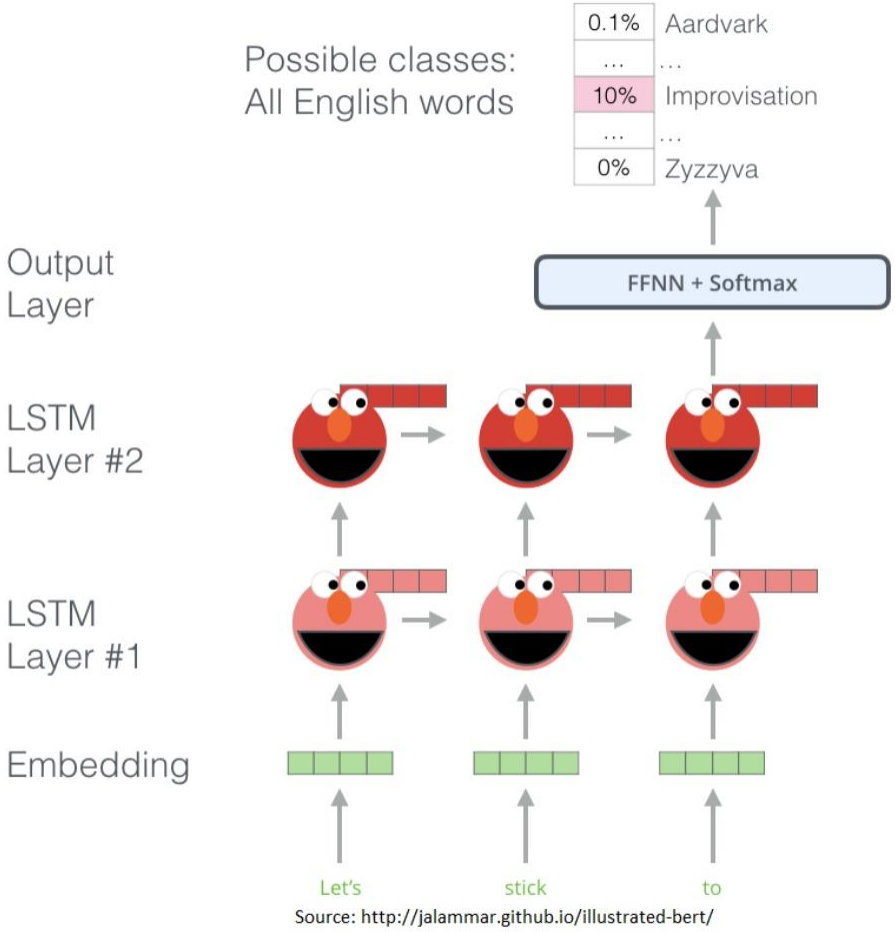
  Embedding value remains the same regardless of the context in which the word is used e.g., ***river bank vs. financial bank***

- **ELMo looks at the entire sentence before assigning each word a specfic contextual embedding**

- **ELMo stands for Embeddings from Language Models**

  ELMo is trained as a language model on a huge corpora of text data

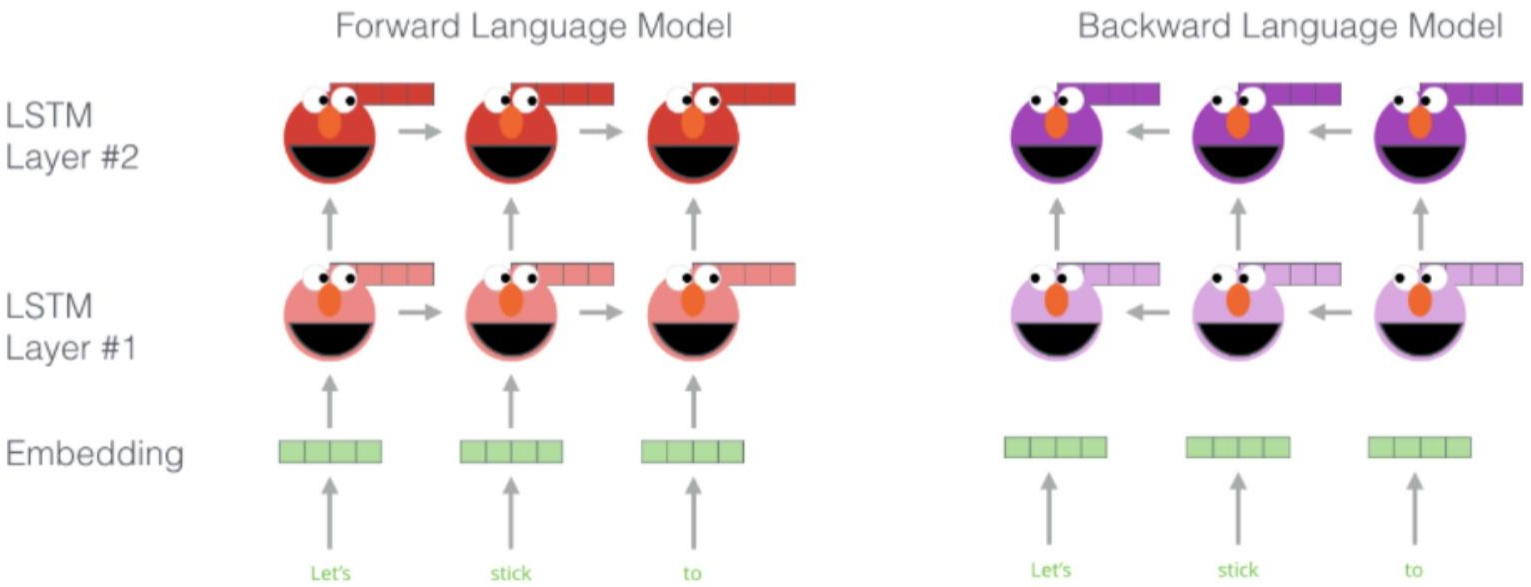
Contextual Embeddings with ELMo



Source: http://jalammar.github.io/illustrated-bert/

- **ELMo consists of two layer bi-directional LSTMs trained on a classical Language Modeling task**
- **ELMo gains the contextual understanding from being trained on a classic LM task - "predict the next word from a sequence of words"**
  - In the pre-training phase, we feed a large amount of sequential text data
  - ELMo can learn relevant representations without needing traditional labeled data

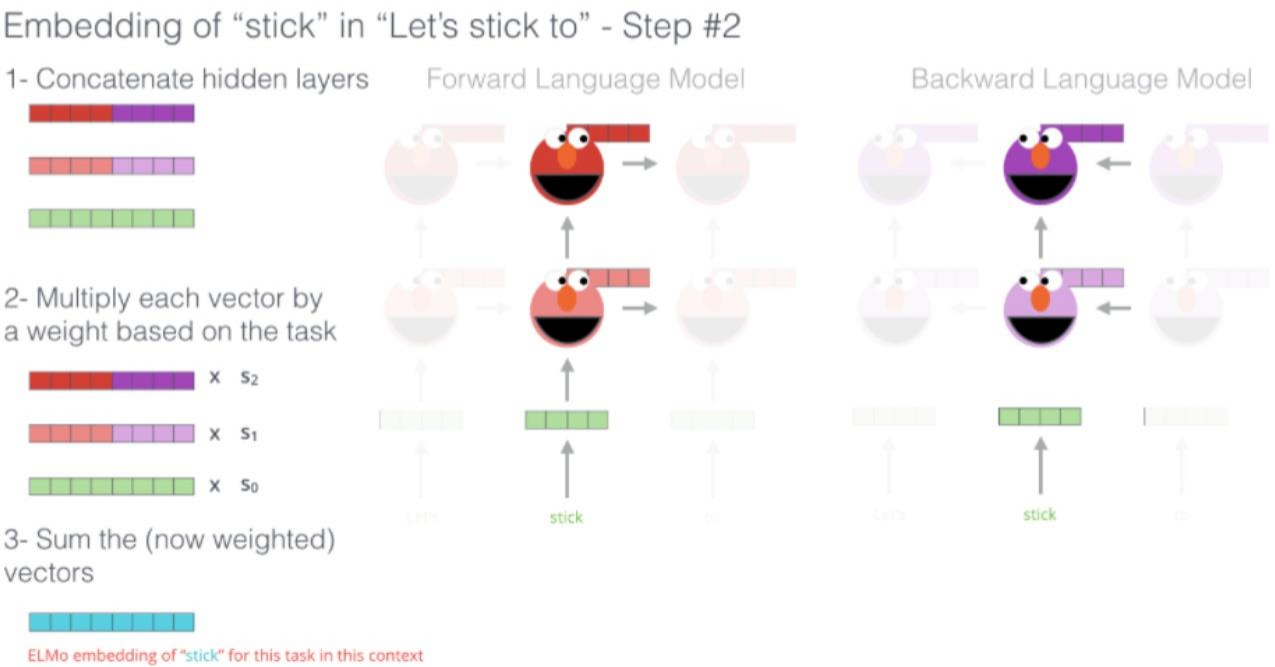
ELMo uses Bi-directional LSTMs



Source: http://jalammar.github.io/illustrated-bert

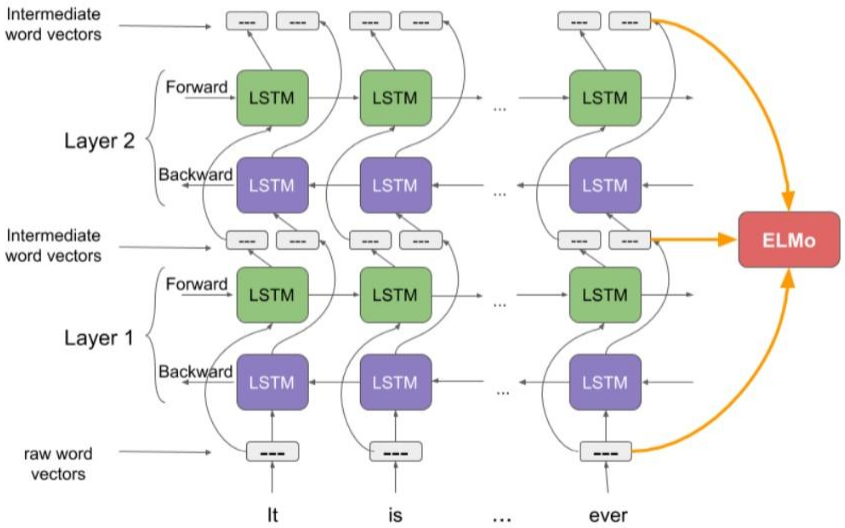
ELMo uses Bi-directional LSTMs

ELMo comes up with the contextualized embedding through grouping together the hidden states (and initial embedding) in a certain way (concatenation followed by weighted summation).

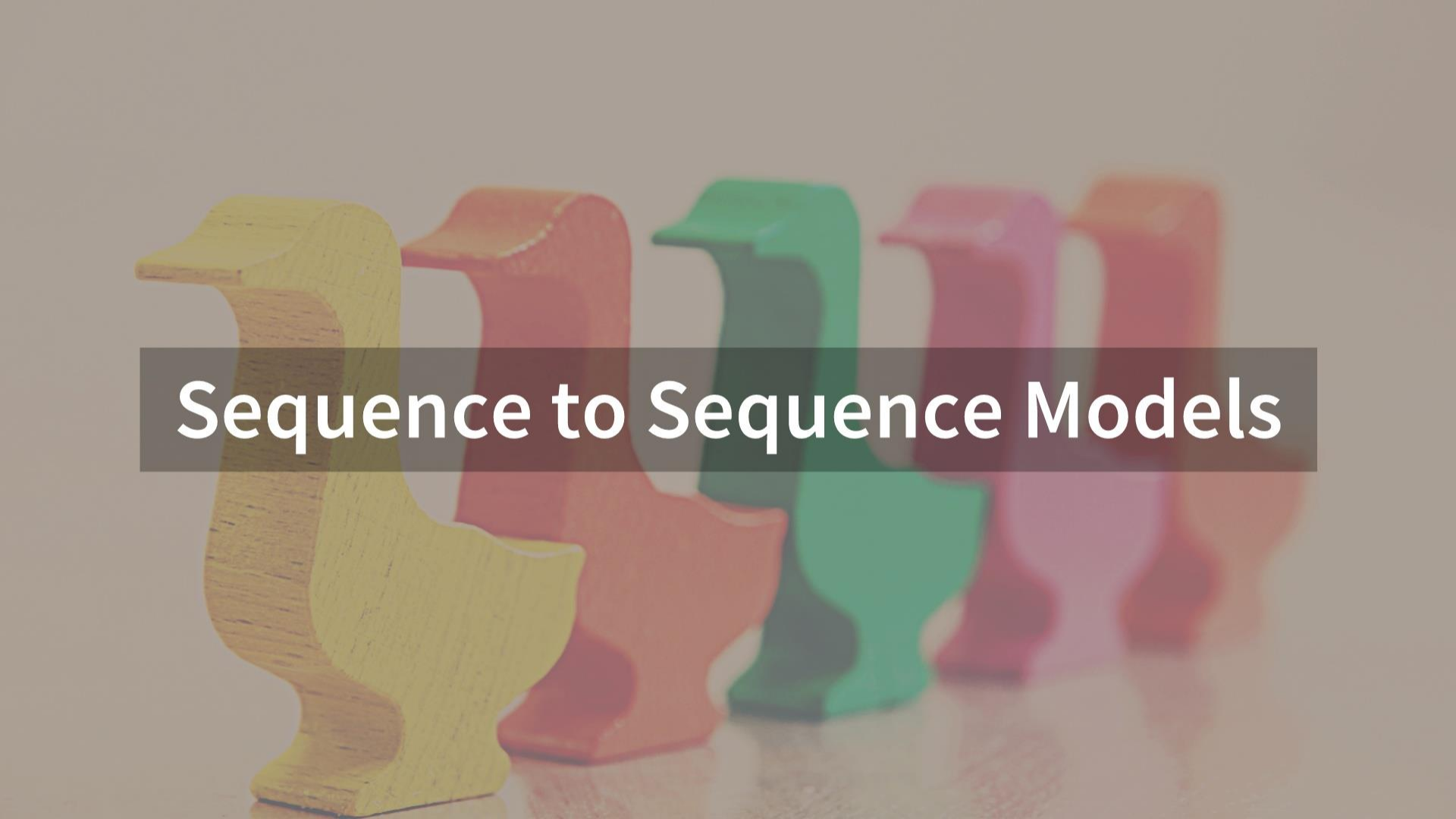


Source: http://jalammar.github.io/illustrated-bert/

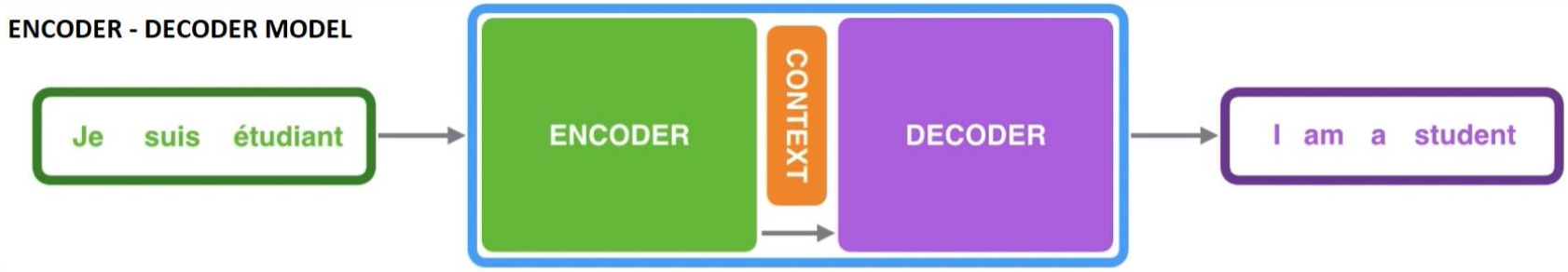
Contextual Embeddings with ELMo

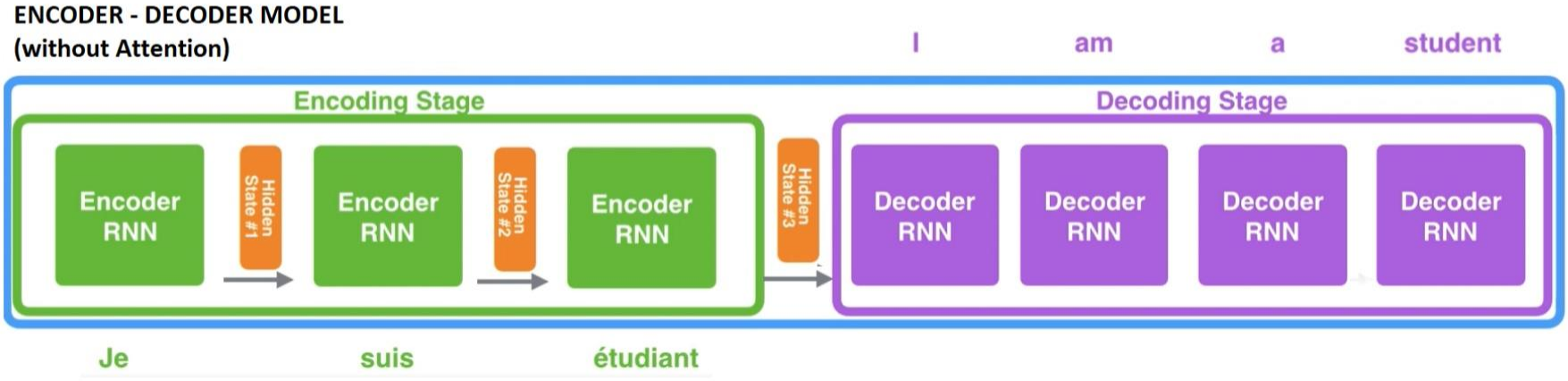


- ELMo uses a character-level convolutional neural network (CNN) to represent words as raw word vectors (embeddings)
  - **word_emb** in ELMo from TensorFlow Hub
- These raw word vectors act as inputs to the first bi-directional LSTM layer
  - Hidden states from BiLSTM layer 1 generate embeddings
  - **lstm_outputs1** in ELMo from TensorFlow Hub
- These intermediate embeddings are fed as inputs to the second bi-directional LSTM layer
  - Hidden states from BiLSTM layer 2 generate embeddings
  - **lstm_outputs2** in ELMo from TensorFlow Hub
- The final embedding (elmo) is the weighted sum of the raw word embeddings and the 2 intermediate embedding representations

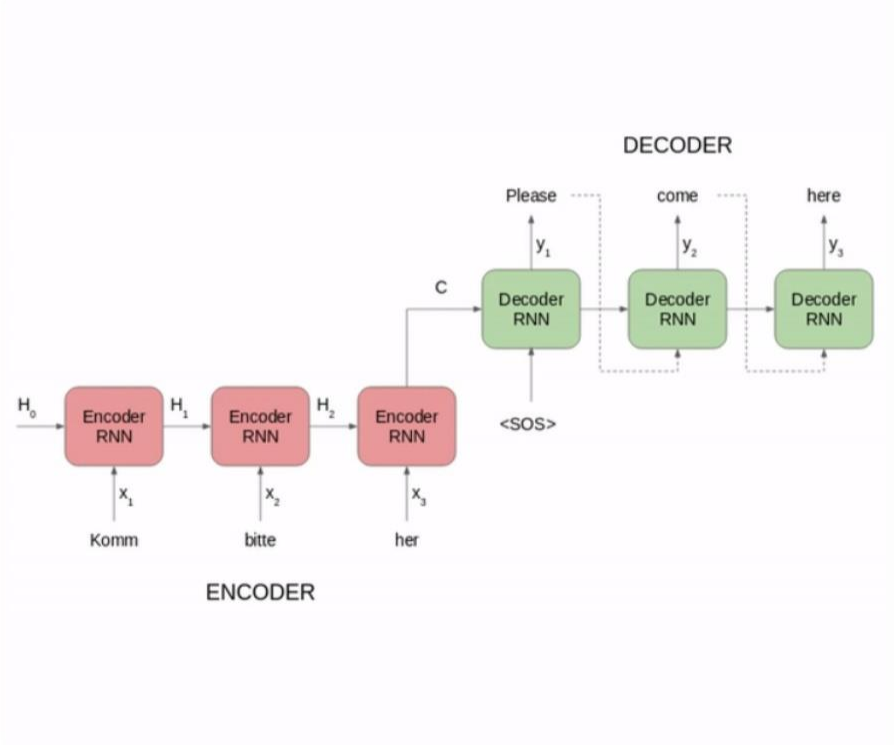

Sequence to Sequence Models

Sequence to Sequence (Encoder-Decoder) Model





Sequence to Sequence Models - Summary



- Encoder & Decoder blocks consist of multiple LSTM cells
- Hidden States are updated at each time step
- Context or Thought Vector from last step encodes information of the complete sequence
- Encoder helps in encoding representations of the input sequence
- Decoder leverages hidden state representations from the encoder to output target sequence