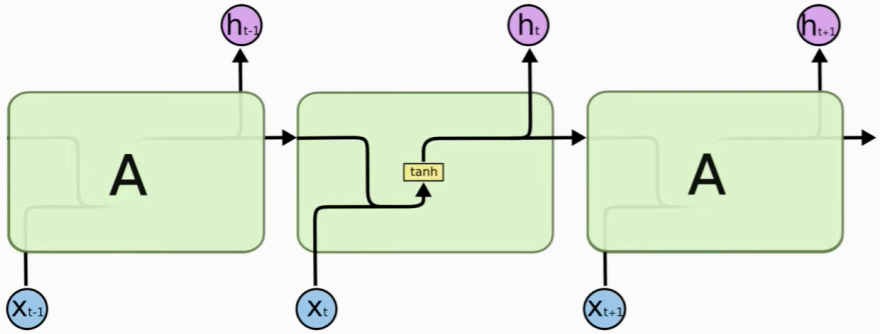
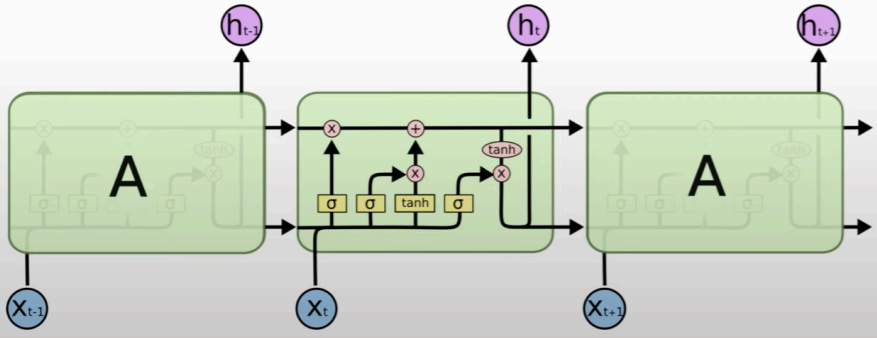
LSTM Networks



Simple RNN

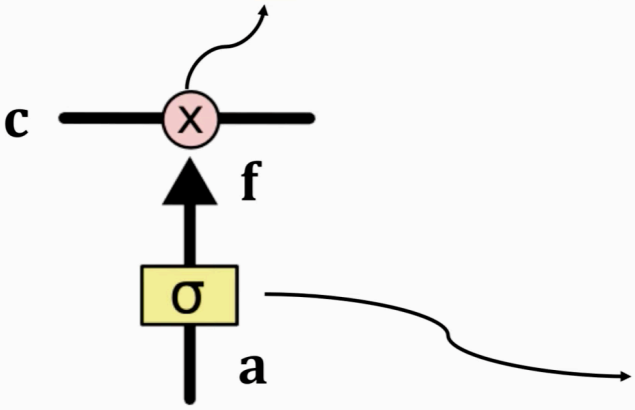


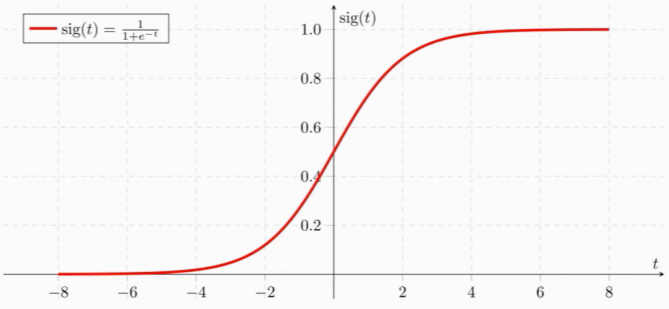
LSTM

Figures are from Christopher Olah's blog: Understanding LSTM Networks.

LSTM: Forget Gate

Elementwise multiplication of 2 vectors.

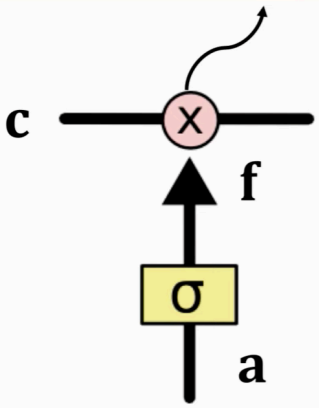




Sigmoid function: between 0 and 1.

LSTM: Forget Gate

Elementwise multiplication of 2 vectors.

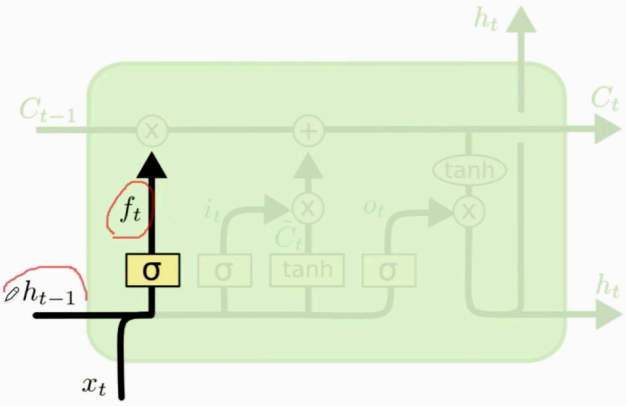


$$\begin{bmatrix} 0.9 \\ 0.2 \\ -0.5 \\ -0.1 \end{bmatrix} \circ \begin{bmatrix} 0.5 \\ 0 \\ 1 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0 \\ -0.5 \\ -0.08 \end{bmatrix}$$

c **f** **output**

LSTM: Forget Gate

- Forget gate (**f**): a vector (the same shape as **c** and **h**).
  - A value of zero means "let nothing through".
  - A value of one means "let everything through!"

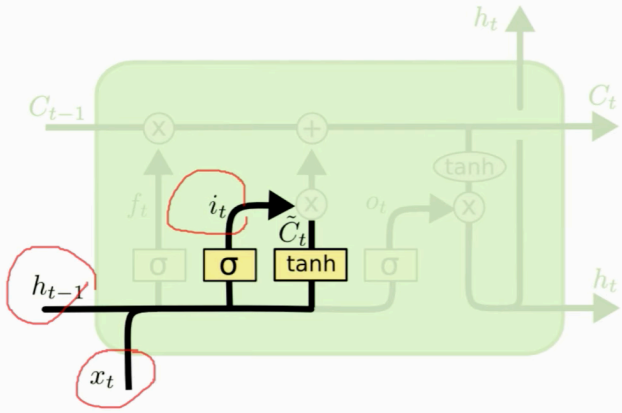


$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f \cdot \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix}\right)$$

left figure is from Christopher Olah's blog: Understanding LSTM Networks

LSTM: Input Gate

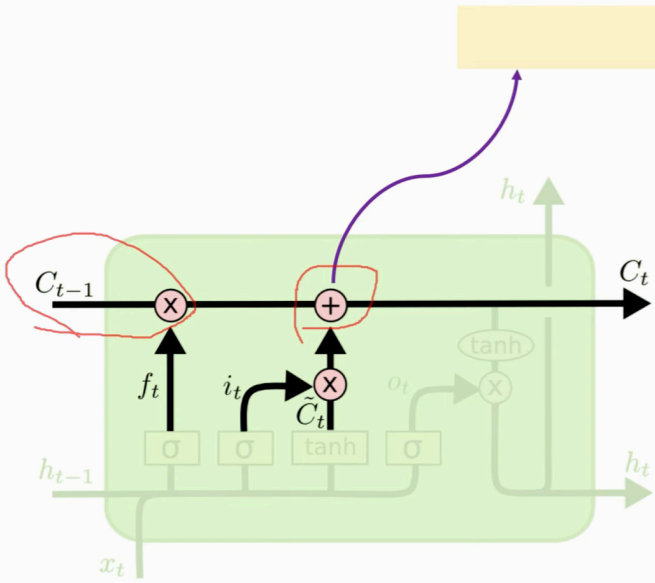
- Input gate ($\mathbf{i}_t$): decides which values of the conveyor belt we'll update.



$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i \cdot \begin{bmatrix} \mathbf{h}_{t-1} \\ \mathbf{x}_t \end{bmatrix}\right)$$

The left figure is from Christopher Olah's blog: Understanding LSTM Networks.

LSTM: Update the Conveyor Belt

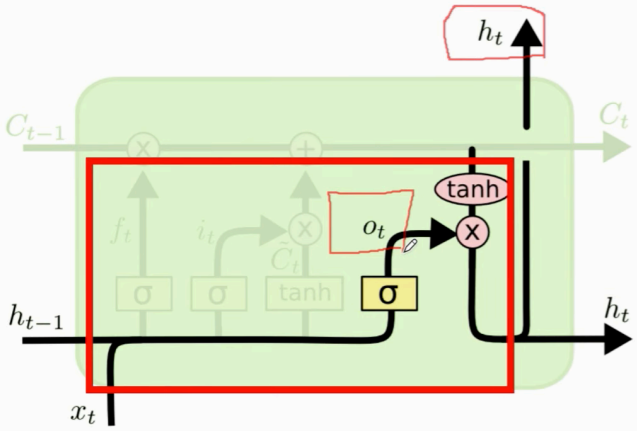


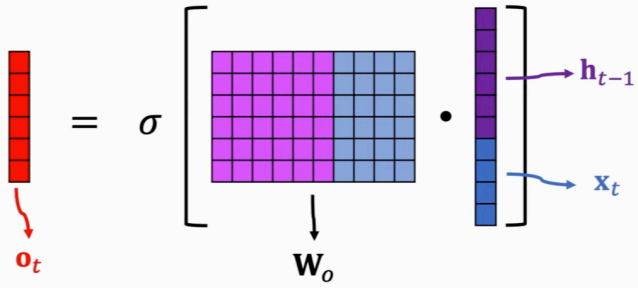
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$

The left figure is from Christopher Olah's blog: Understanding LSTM Networks.

LSTM: Output Gate

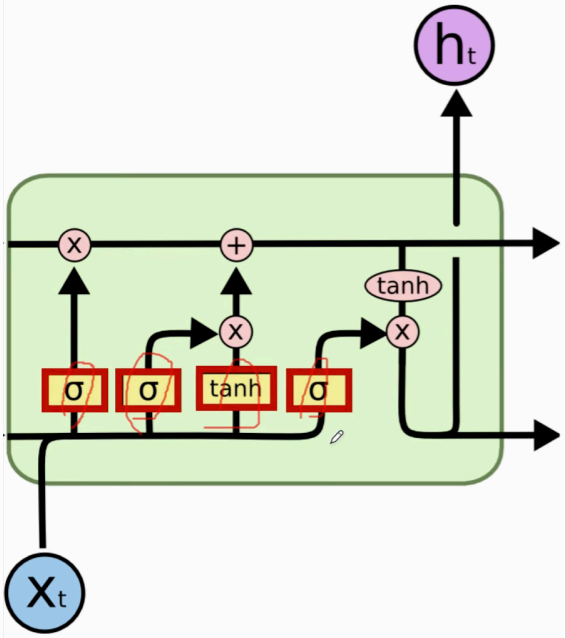
- Output gate ($\mathbf{o}_t$): decide what flows from the conveyor belt C_{t-1} to the state h_t.





The left figure is from Christopher Olah's blog: Understanding LSTM Networks.

LSTM: Number of Parameters



#parameters: 4 times as many as SimpleRNN

- 4 parameter matrices, each of which has
  - `#rows: shape(h)`
  - `#cols: shape(h)+shape(x)`
- #parameter (do not count intercept):

`4 × shape(h) × [shape(h)+shape(x)]`

The left figure is from Christopher Olah's blog: Understanding LSTM Networks.

LSTM for IMDB Review

```
from keras.models import Sequential
from keras.layers import LSTM, Embedding, Dense, Flatten

vocabulary = 10000
embedding_dim = 32
word_num = 500
state_dim = 32

model = Sequential()
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
model.add(LSTM(state_dim, return_sequences=False))
model.add(Dense(1, activation='sigmoid'))
```

Replace "SimpleRNN" by "LSTM".

```
model.summary()
```

LSTM for IMDB Review

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 32)	320000
lstm_1 (LSTM)	(None, 32)	8320
dense_1 (Dense)	(None, 1)	33

Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0

#parameters in LSTM:

- $8320 = 2080 \times 4$
- $2080 = 32 \times (32 + 32) + 32$

$\text{shape}(\mathbf{h}) = 32$ $\text{shape}(\mathbf{x})$

LSTM Dropout

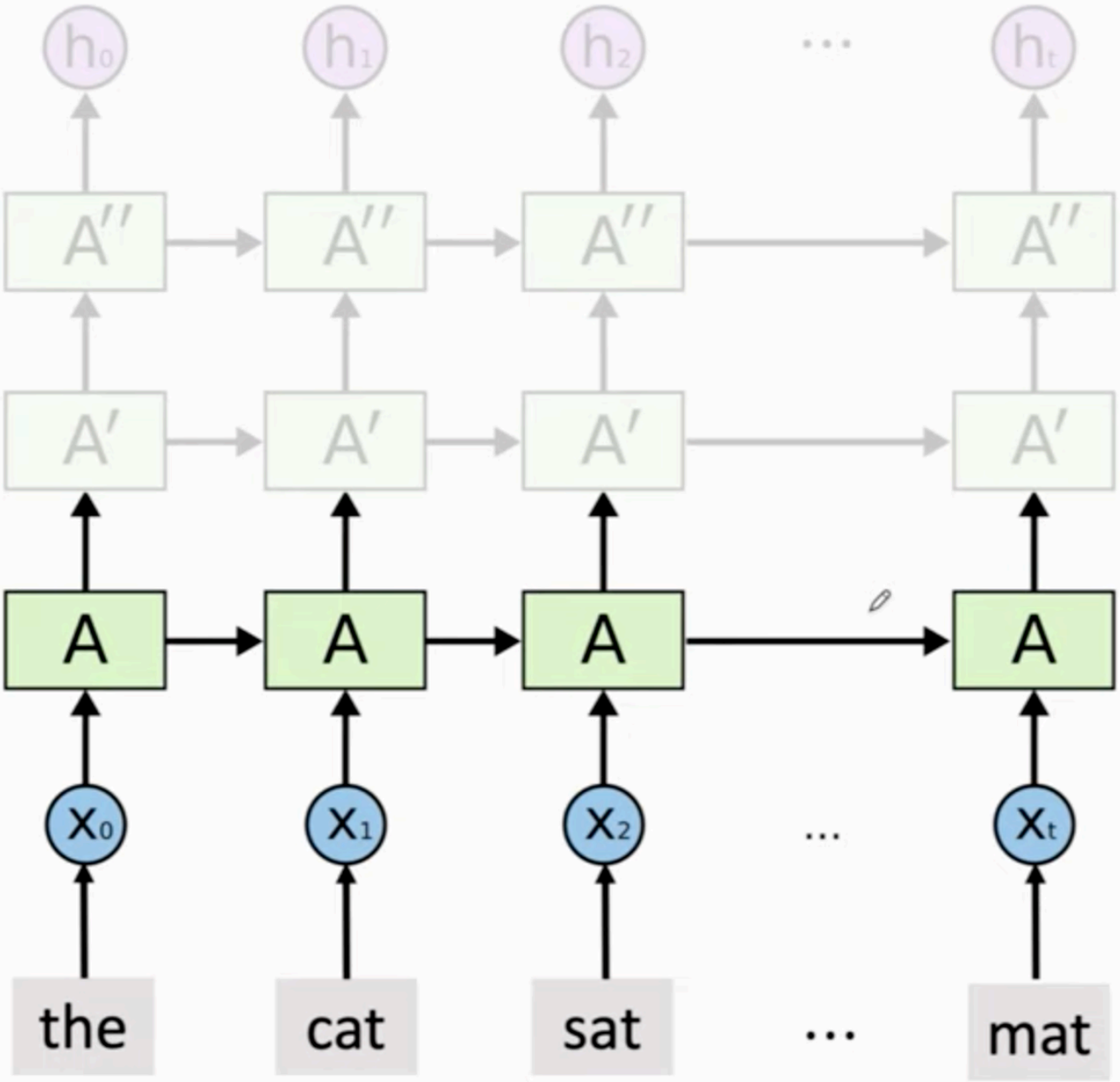
```
from keras.models import Sequential
from keras.layers import LSTM, Embedding, Dense, Flatten

vocabulary = 10000
embedding_dim = 32
word_num = 500
state_dim = 32

model = Sequential()
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
model.add(LSTM(state_dim, return_sequences=False, dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Summary

- LSTM uses a "conveyor belt" to get longer memory than SimpleRNN.
- Each of the following blocks has a parameter matrix:
  - Forget gate.
  - Input gate.
  - New values.
  - Output gate.
- Number of parameters:

$$4 \times \text{shape}(\mathbf{h}) \times [\text{shape}(\mathbf{h}) + \text{shape}(\mathbf{x})].$$

Stacked RNN

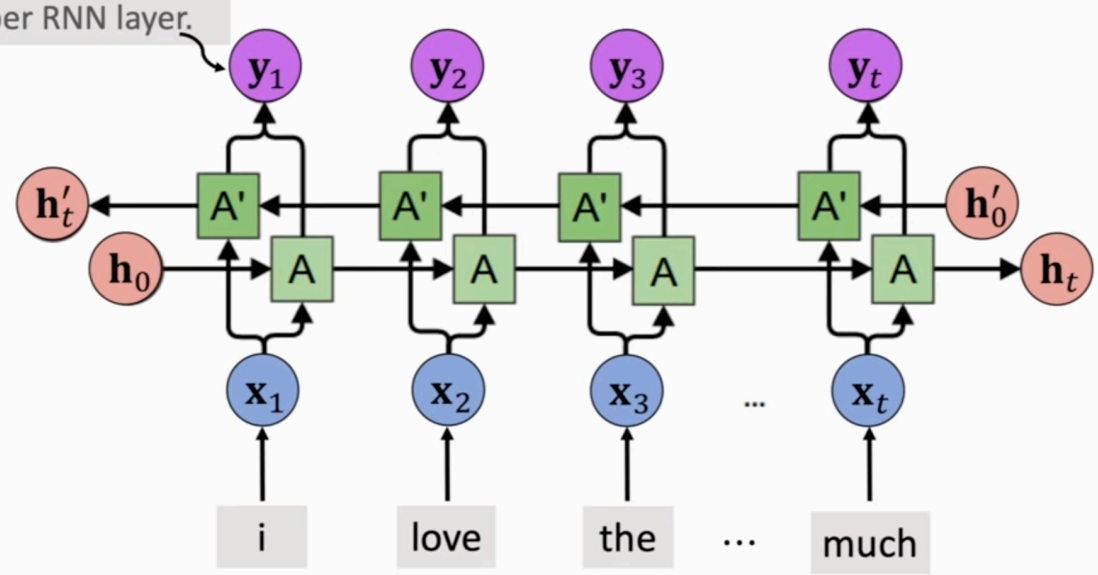


Stacked LSTM

```
from keras.models import Sequential
from keras.layers import LSTM, Embedding, Dense

vocabulary = 10000
embedding_dim = 32
word_num = 500
state_dim = 32

model = Sequential()
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
model.add(LSTM(state_dim, return_sequences=True, dropout=0.2))
model.add(LSTM(state_dim, return_sequences=True, dropout=0.2))
model.add(LSTM(state_dim, return_sequences=False, dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
```

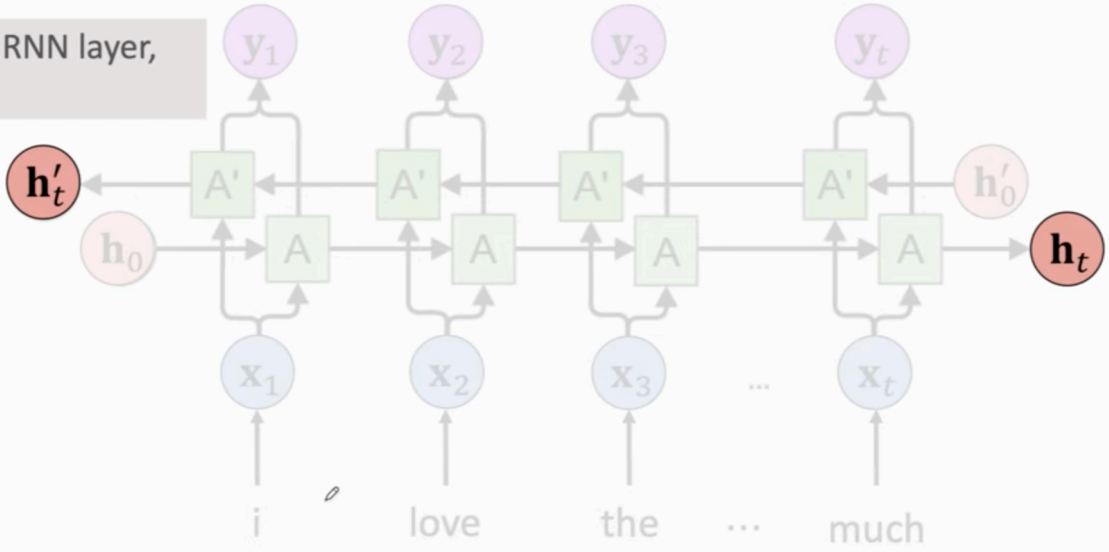
Bidirectional RNN

- Stack of 2 states.
- Passed to the upper RNN layer.



Bidirectional RNN

If there is no upper RNN layer, then return $[\mathbf{h}_t, \mathbf{h}'_t]$



Bi-LSTM

```
from keras.models import Sequential
from keras.layers import LSTM, Embedding, Dense, Bidirectional

vocabulary = 10000
embedding_dim = 32
word_num = 500
state_dim = 32

model = Sequential()
model.add(Embedding(vocabulary, embedding_dim, input_length=word_num))
model.add(Bidirectional(LSTM(state_dim, return_sequences=False, dropout=0.2)))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Bi-LSTM

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 32)	320000
bidirectional_1 (Bidirection	(None, 64)	16640
dense_1 (Dense)	(None, 1)	65

Total params: 336,705
Trainable params: 336,705
Non-trainable params: 0

Why pretraining?

Observation: The embedding layer contributes most of the parameters!

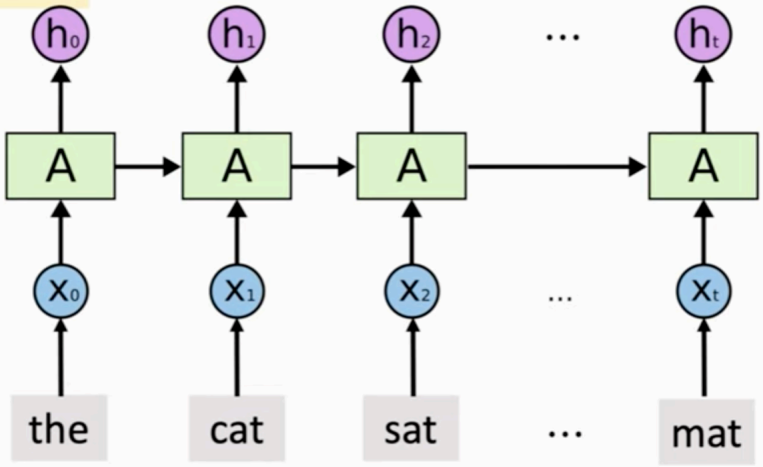
Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 32)	320000
bidirectional_1 (Bidirection	(None, 64)	16640
dense_1 (Dense)	(None, 1)	65

Total params: 336,705
Trainable params: 336,705
Non-trainable params: 0

Pretrain the Embedding Layer

Step 1: Train a model on large dataset.

- Perhaps different problem.
- Perhaps different model.

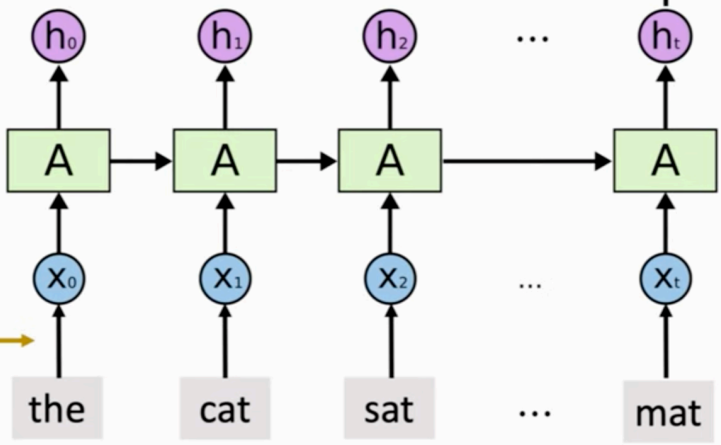


Pretrain the Embedding Layer

Step 3: Train new LSTM and output layers.

$\text{sigmoid}(\mathbf{v}^T \mathbf{h}_t)$

Set the embedding layer non-trainable.



Summary

- SimpleRNN and LSTM are two kinds of RNNs; always use LSTM instead of SimpleRNN.
- Use Bi-RNN instead of RNN whenever possible.
- Stacked RNN may be better than a single RNN layer (if n is big).
- Pretrain the embedding layer (if n is small).