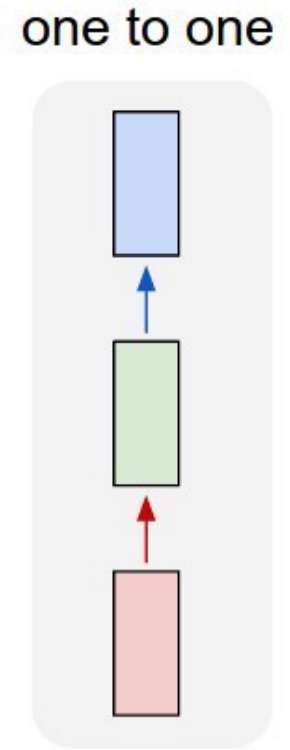
Recurrent Neural Networks (RNNs)

Shusen Wang

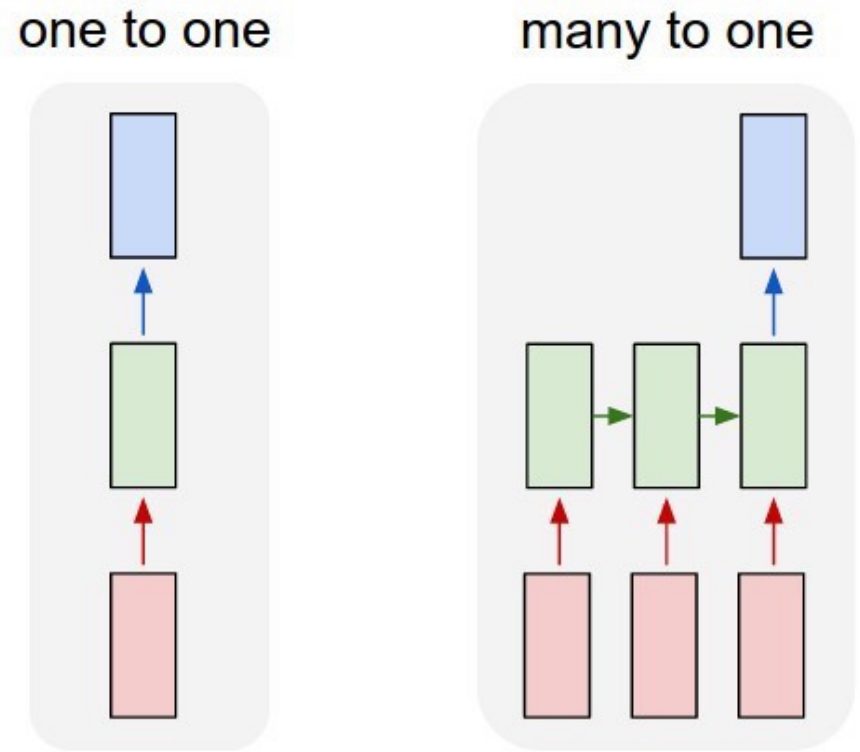
Basics of RNNs

How to model sequential data?



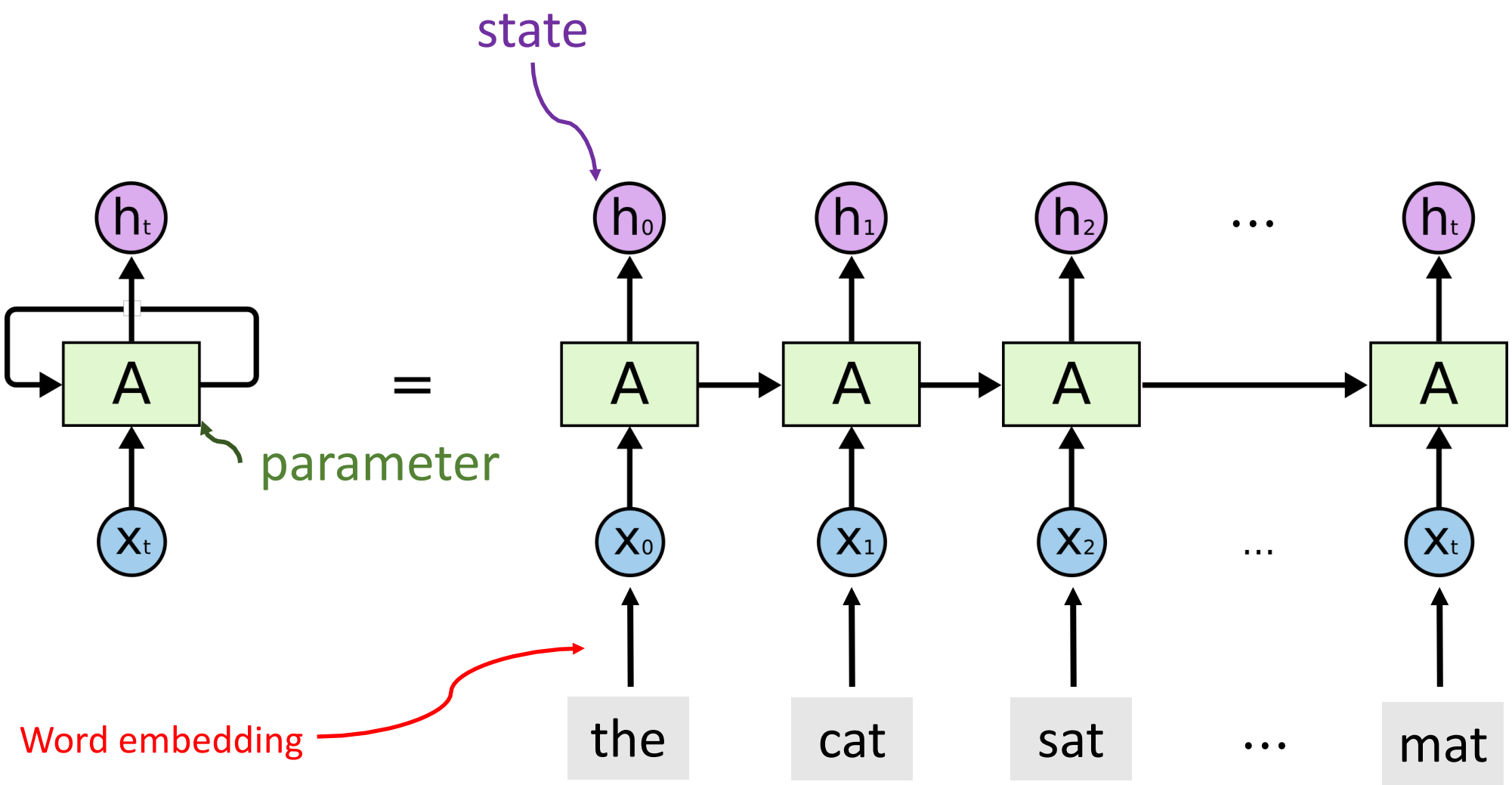
- Limitations of FC Nets and ConvNets:
  - Process a paragraph as a whole.
  - Fixed-size input (e.g., image).
  - Fixed-size output (e.g., the one-hot encode of labels).

How to model sequential data?

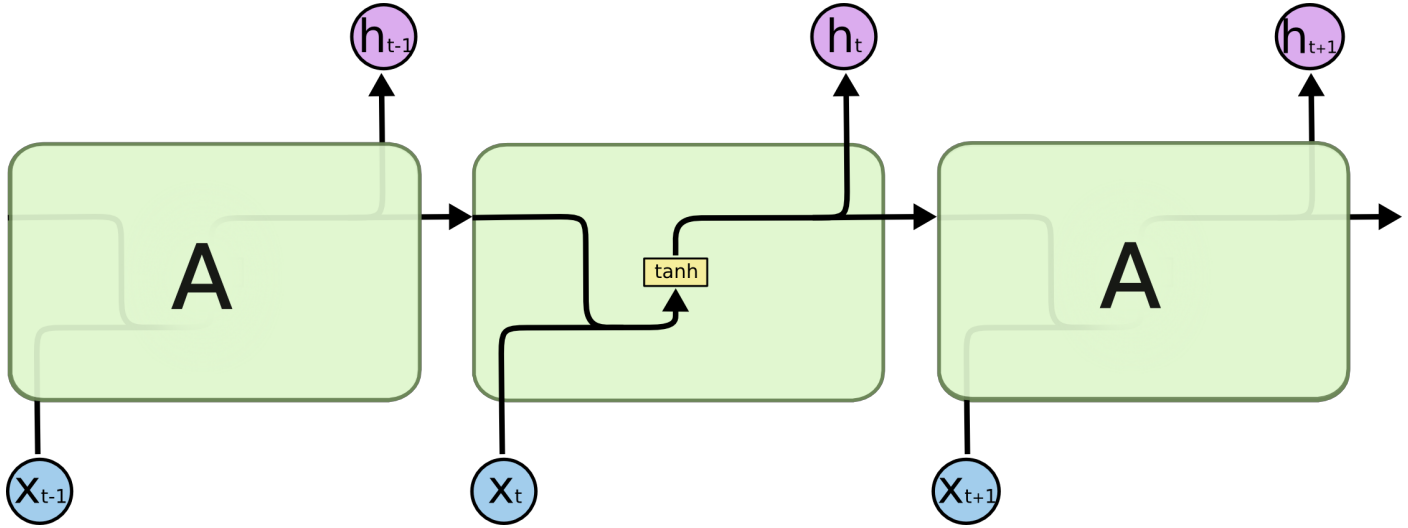


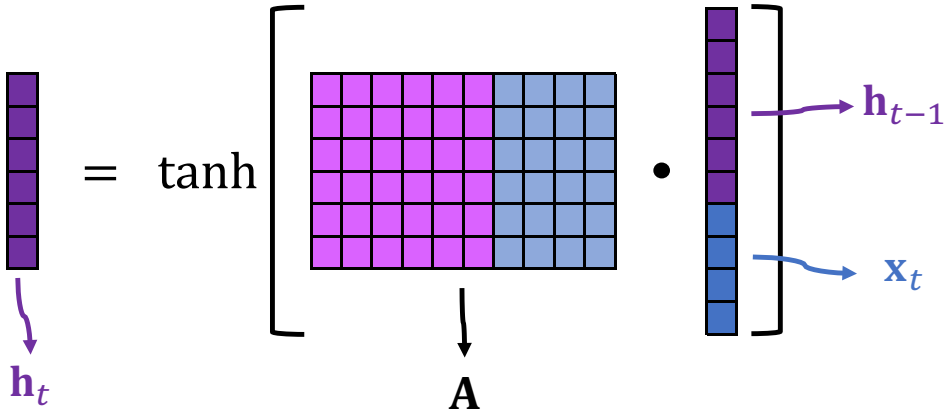
- Limitations of FC Nets and ConvNets:
  - Process a paragraph as a whole.
  - Fixed-size input (e.g., image).
  - Fixed-size output (e.g., the one-hot encode of labels).
- There are better ways to model the sequential data (e.g., text, speech, and time series): RNNs.

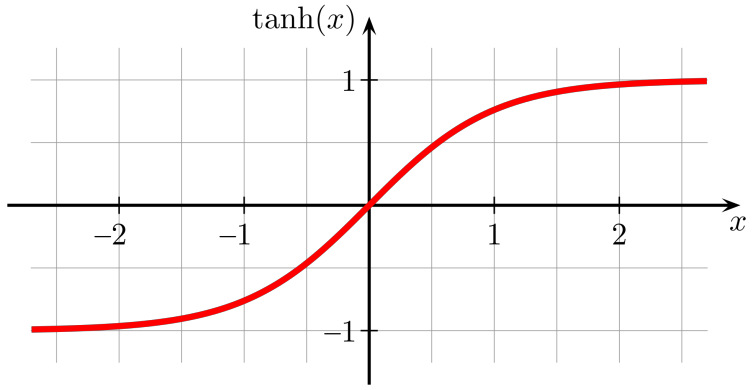
Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)







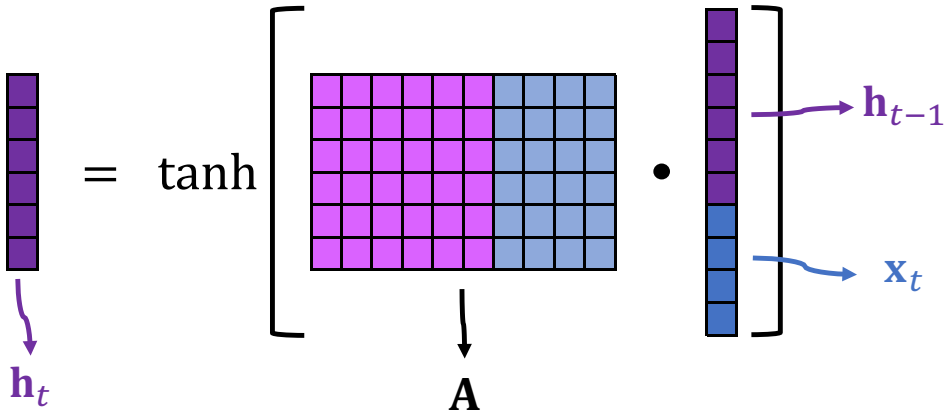
hyperbolic tangent function

Recurrent Neural Networks (RNNs)

Trainable parameters: matrix **A**

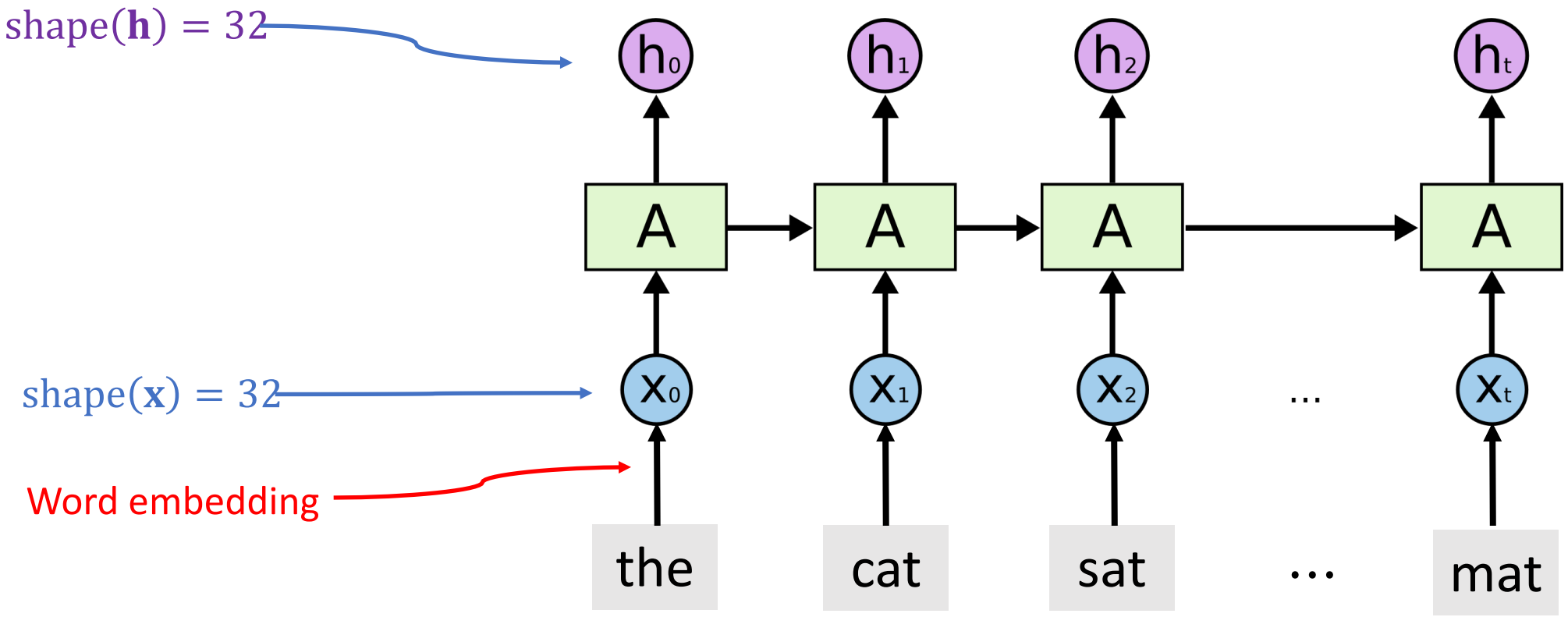
- #rows of **A**: shape(**h**)
- #cols of **A**: shape(**h**)+shape(**x**)

- Total #parameter: shape(**h**)× [shape(**h**)+shape(**x**)]

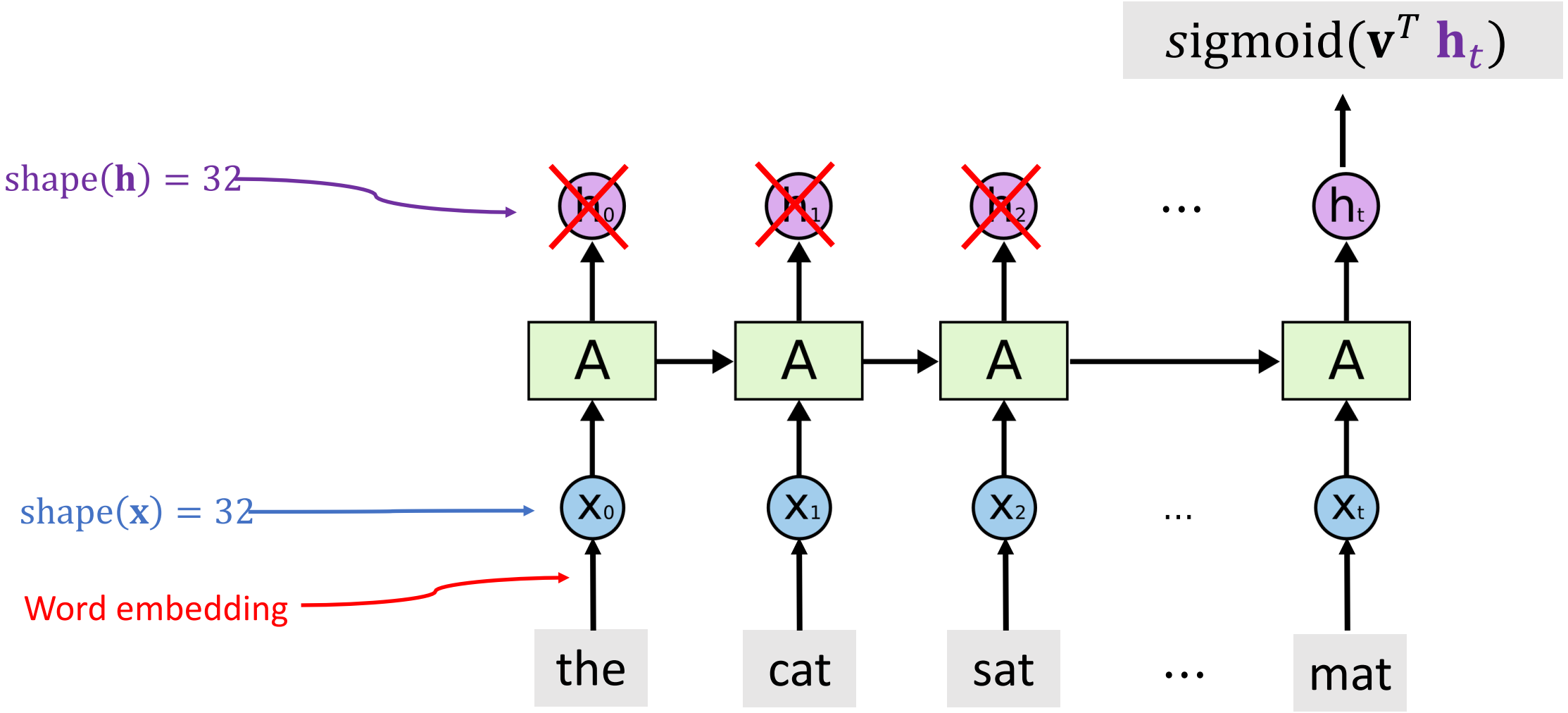


Simple RNN for Movie Review Analysis

Simple RNN for IMDB Review



Simple RNN for IMDB Review



Simple RNN for IMDB Review

```
from keras.models import Sequential
from keras.layers import SimpleRNN, 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(SimpleRNN(state_dim, return_sequences=False))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Only return the last hidden state h_t.

Simple RNN for IMDB Review

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 32)	320000
simple_rnn_1 (SimpleRNN)	(None, 32)	2080
dense_1 (Dense)	(None, 1)	33

Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0

#parameters in RNN:

$$2080 = 32 \times (32 + 32) + 32$$

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

Simple RNN for IMDB Review

```
from keras import optimizers

epochs = 3

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=epochs,
                    batch_size=32, validation_data=(x_valid, y_valid))
```

Early stopping alleviates overfitting

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/3
20000/20000 [==============================] - 65s 3ms/step - loss: 0.5514 - acc: 0.6959 - val_loss: 0.4095 - val_acc: 0.8176
Epoch 2/3
20000/20000 [==============================] - 66s 3ms/step - loss: 0.3336 - acc: 0.8620 - val_loss: 0.3296 - val_acc: 0.8658
Epoch 3/3
20000/20000 [==============================] - 65s 3ms/step - loss: 0.2774 - acc: 0.8918 - val_loss: 0.3569 - val_acc: 0.8428
```

Simple RNN for IMDB Review

```
loss_and_acc = model.evaluate(x_test, labels_test)
print('loss = ' + str(loss_and_acc[0]))
print('acc = ' + str(loss_and_acc[1]))
```

```
25000/25000 [==============================] - 21s 829us/step
loss = 0.36507524153709414
acc = 0.84364
```

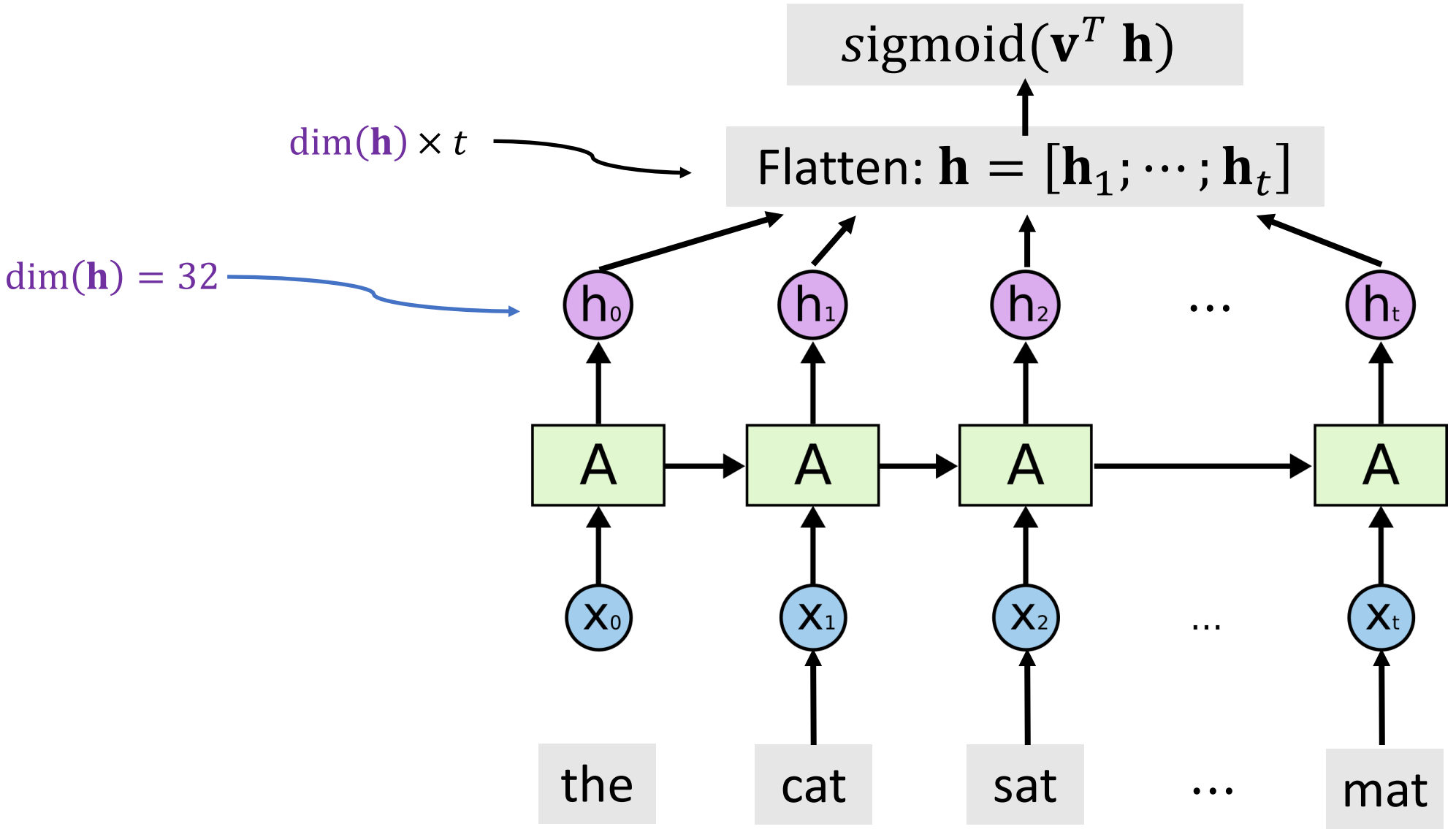
Higher than a naïve shallow model (whose test accuracy is about 75%).

Simple RNN for IMDB Review

- Training Accuracy: 89.2%
- Validation Accuracy: 84.3%
- Test Accuracy: 84.4%

Higher than a naïve shallow model (whose test accuracy is about 75%).

Simple RNN for IMDB Review



Simple RNN for IMDB Review

```
from keras.models import Sequential
from keras.layers import SimpleRNN, 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(SimpleRNN(state_dim, return_sequences=True))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

Return all the last hidden states $h_1, \cdots, h_t$.

Simple RNN for IMDB Review

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           320000
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 500, 32)           2080
_________________________________________________________________
flatten_2 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 16001
=================================================================
Total params: 338,081
Trainable params: 338,081
Non-trainable params: 0
```

Simple RNN for IMDB Review

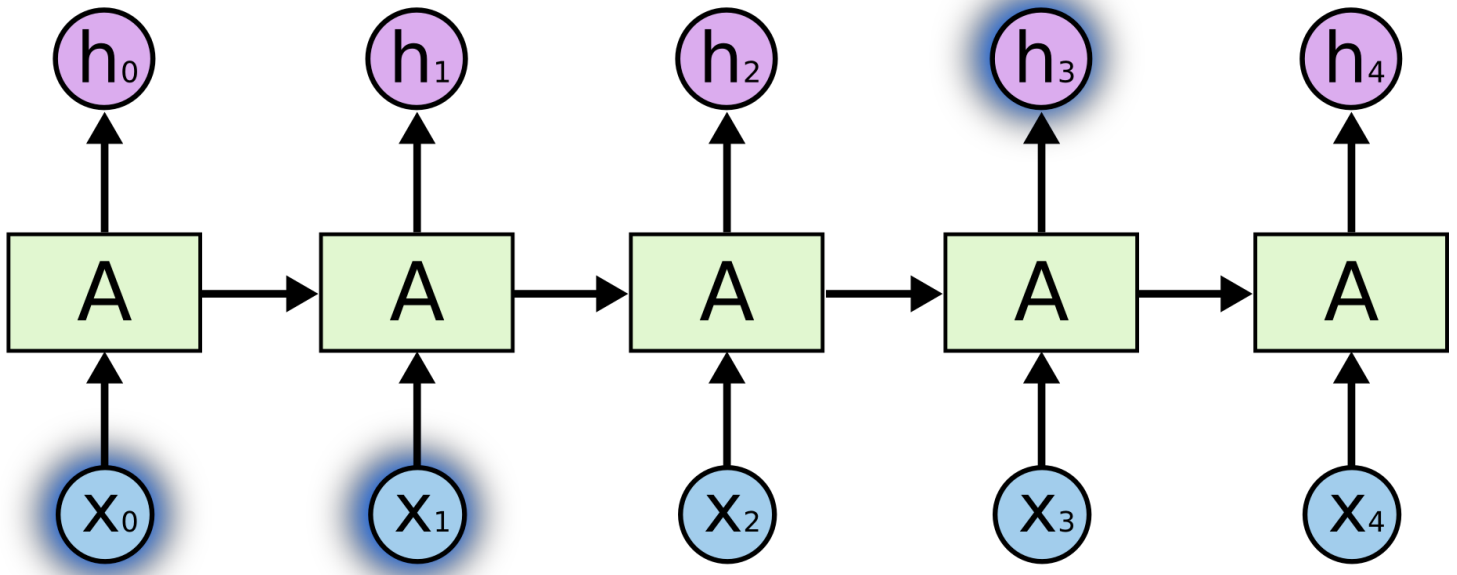
- Training Accuracy: 96.3%
- Validation Accuracy: 85.4%
- Test Accuracy: 84.7%

Not really better than using only the final state (whose accuracy is 84.4%).

Shortcomings of Simple RNN

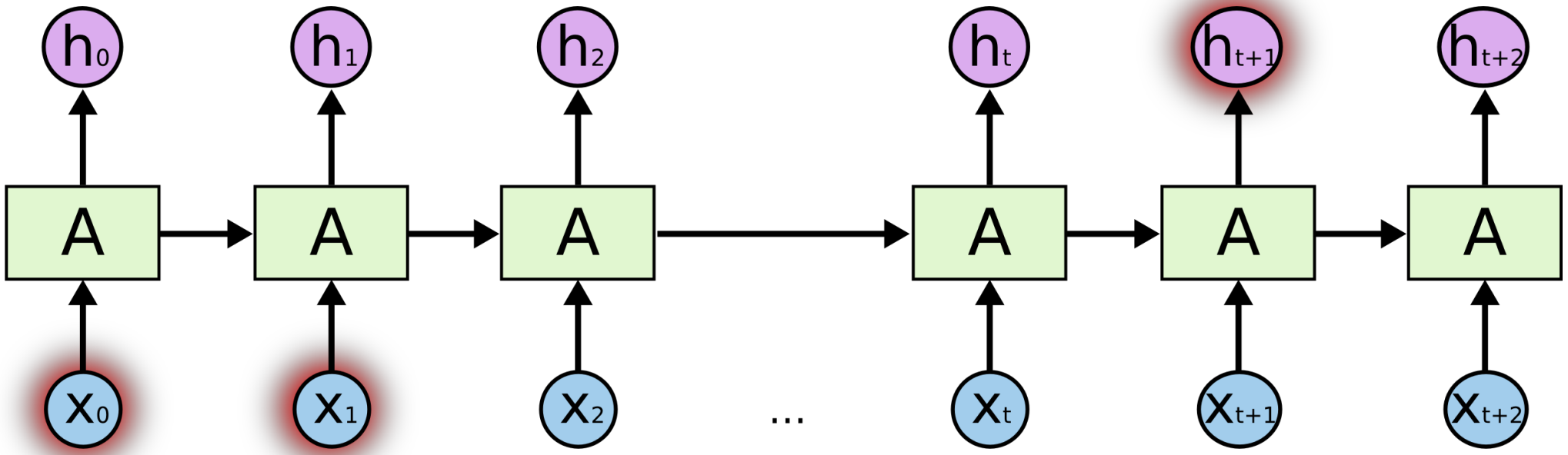
RNN is good at short-term dependence.

Predicted next words: sky



Input text: clouds are in the

RNN is bad at long-term dependence.

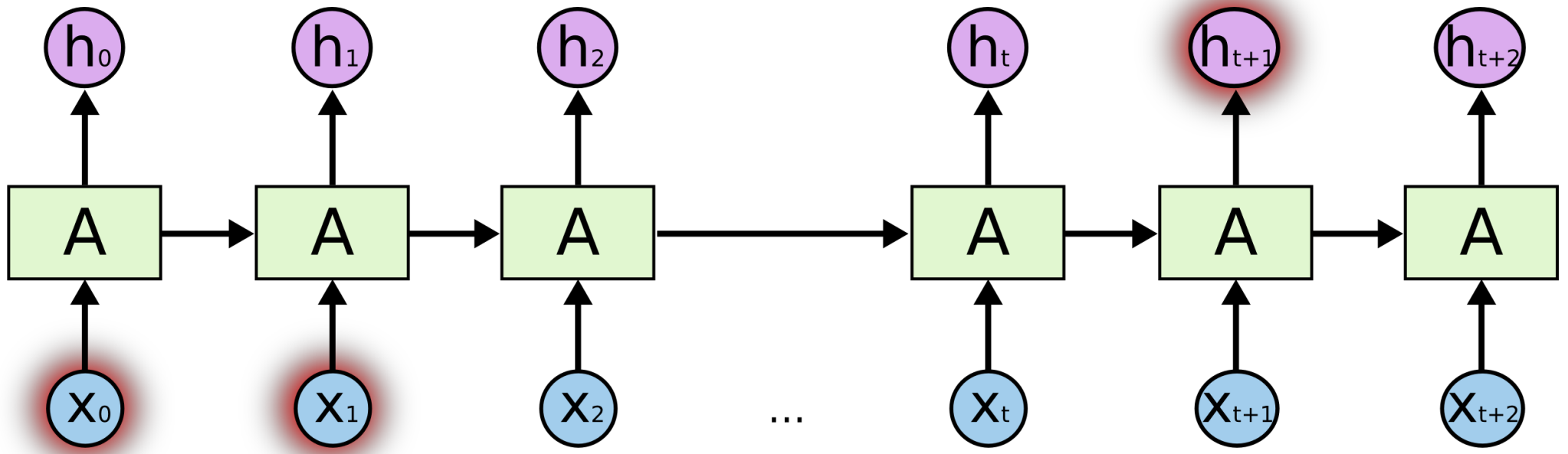


Vanishing gradient: $\frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{x}_1}$ is near zero.

RNN is bad at long-term dependence.

Predicted next words:

France



Input text:

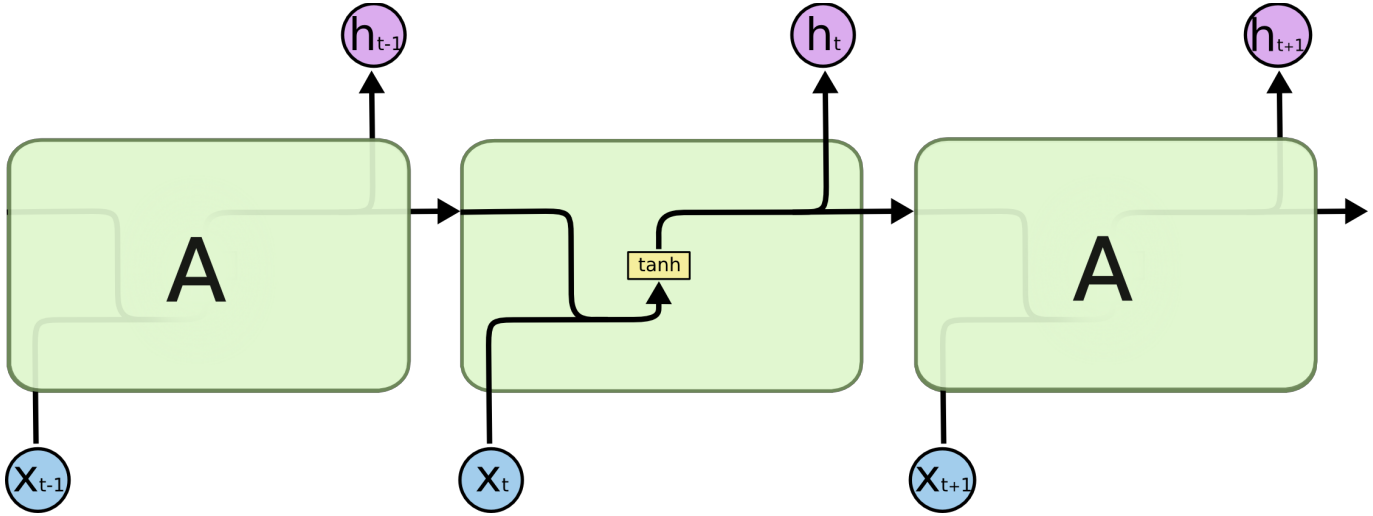
in | France | speak | fluent

Long Short Term Memory (LSTM)

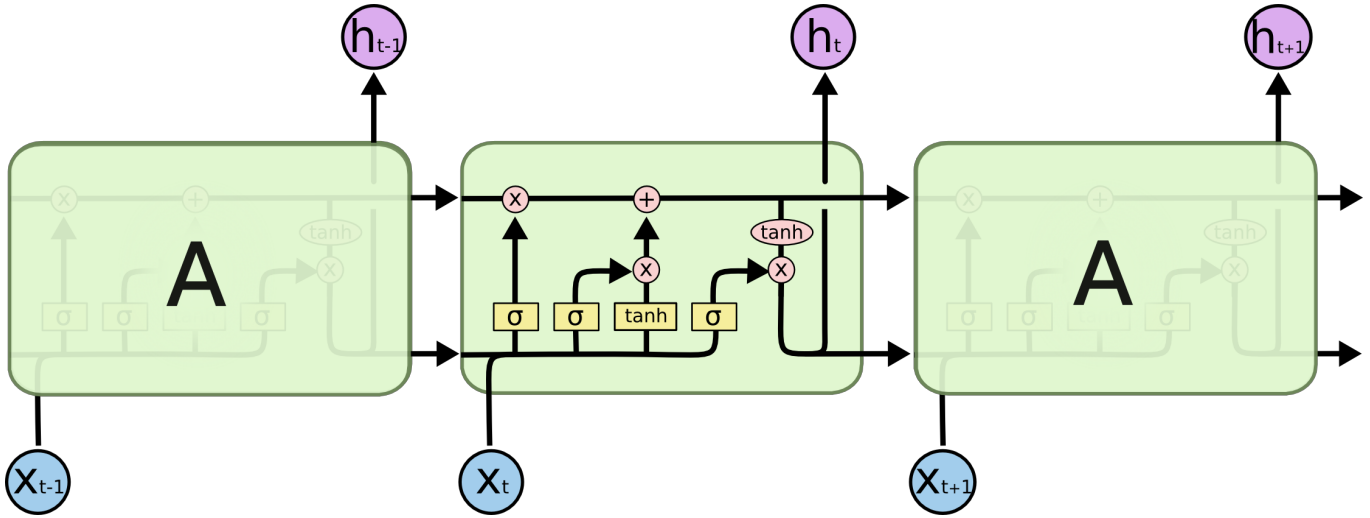
Reference

1. Hochreiter and Schmidhuber. Long short-term memory. *Neural computation,* 1997.

LSTM Networks



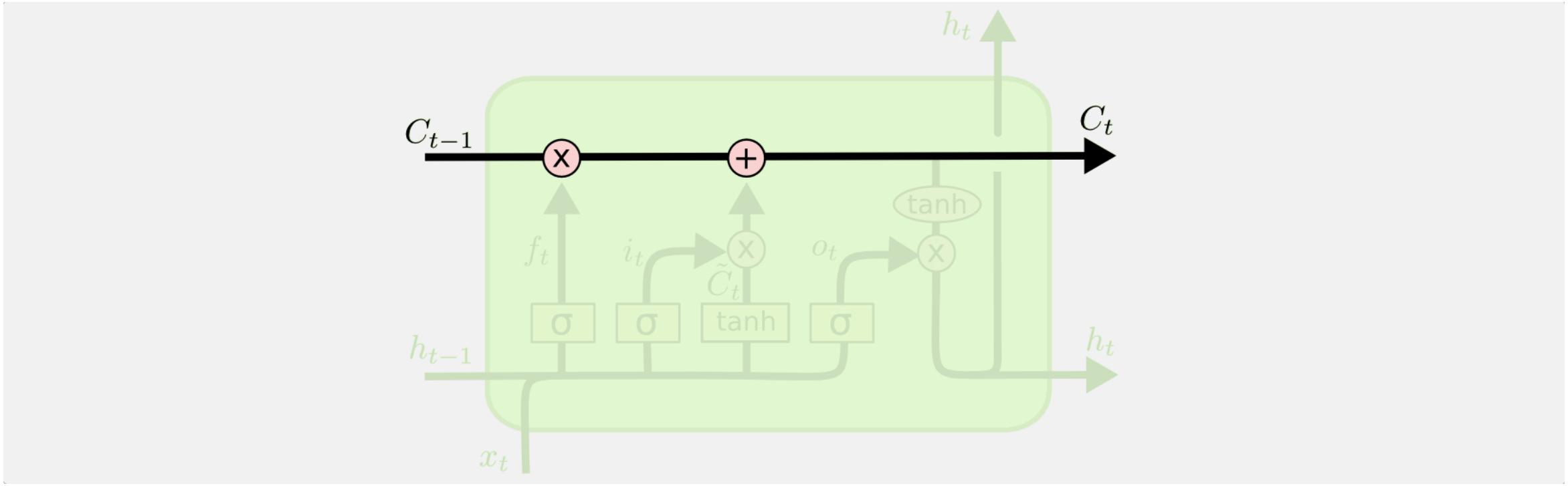
Simple RNN



LSTM

LSTM: Conveyor Belt

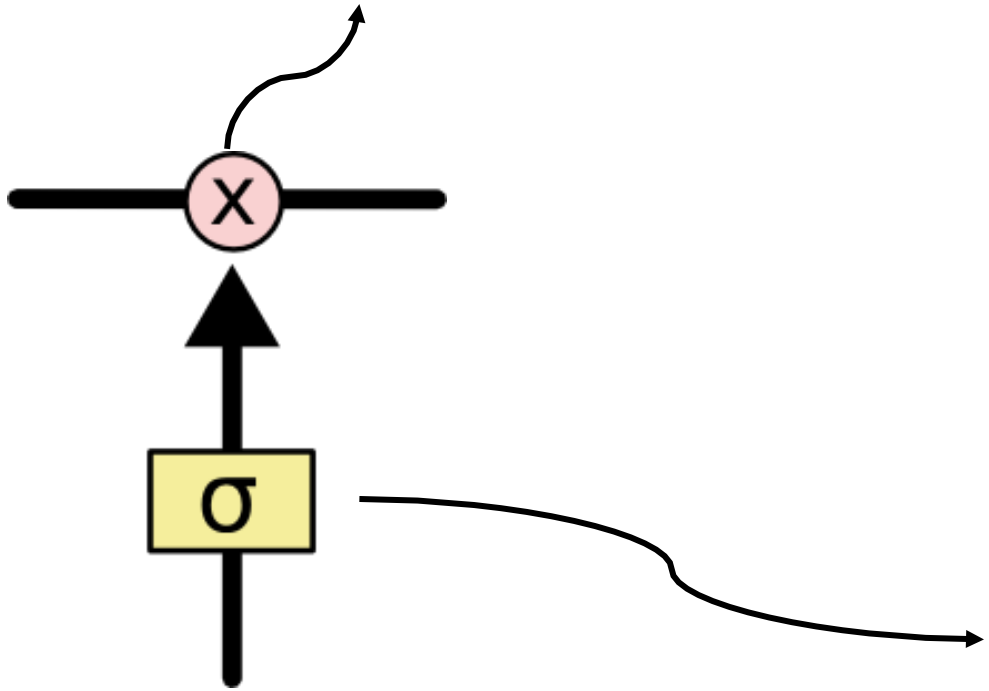
- Conveyor belt: the past information directly flows to the future.

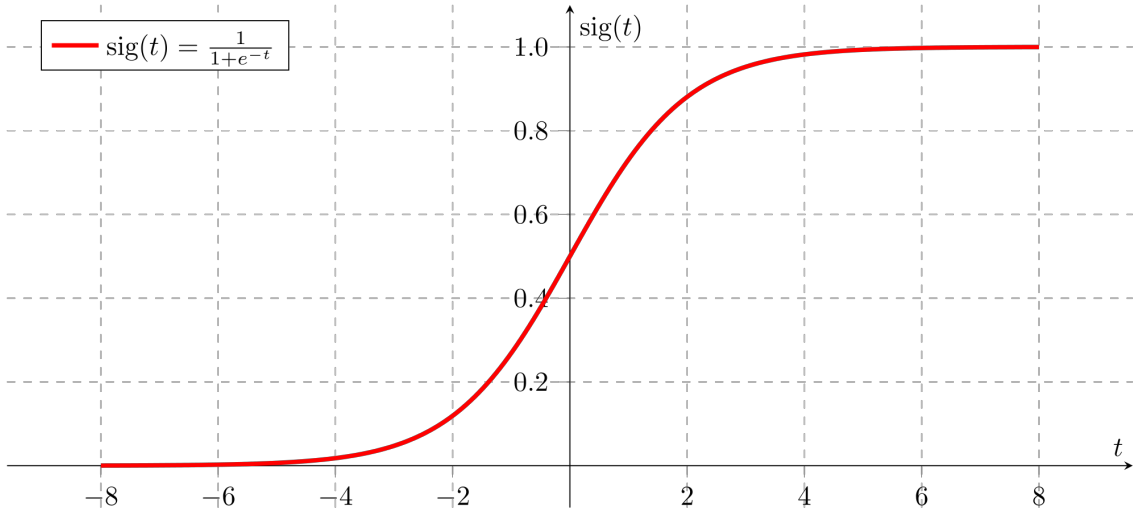


LSTM: Forget Gate

- Forget gate (f_t): a collection of logistic regressions.
  - A value of zero means "let nothing through".
  - A value of one means "let everything through!"

Elementwise multiplication of 2 vectors.

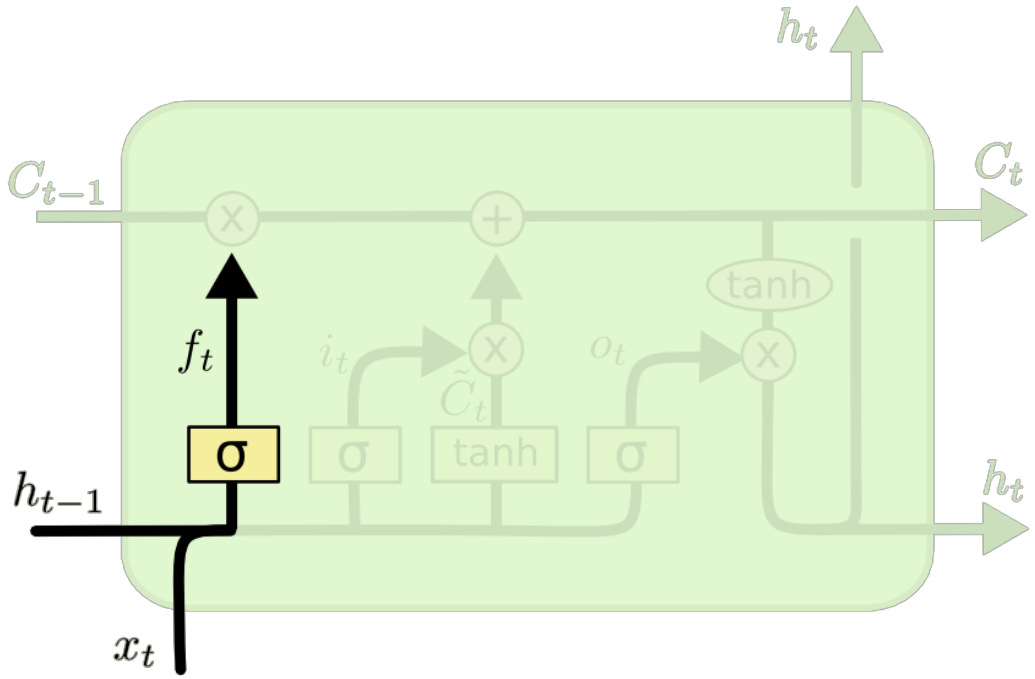




Sigmoid function: between 0 and 1.

LSTM: Forget Gate

- Forget gate (f_t): a collection of logistic regressions.
  - A value of zero means "let nothing through".
  - A value of one means "let everything through!"



parameters

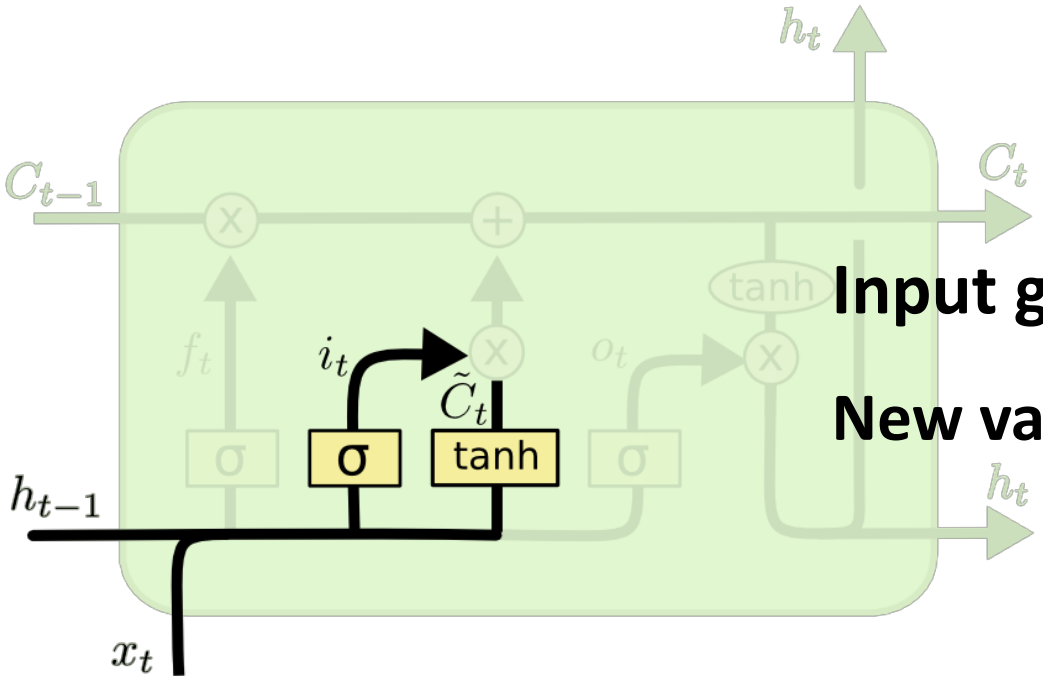
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Its shape is the same to C_t

$\boldsymbol{h}, \mathbf{x}, \mathbf{f}$ are vectors

LSTM: Input Gate

- Input gate (i_t): decides which values of the conveyor belt we'll update.



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

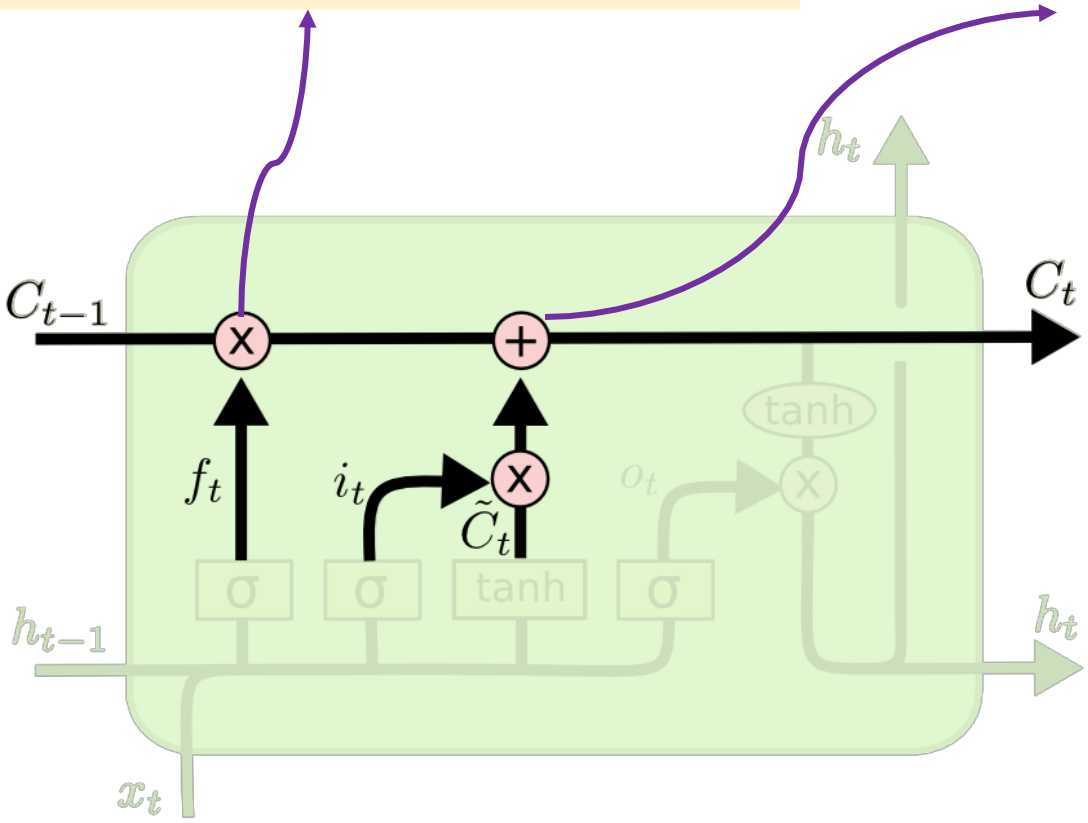
New value: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$

their dimensions are the same to C_t

LSTM: Update the Conveyor Belt

optionally let information through

add new information

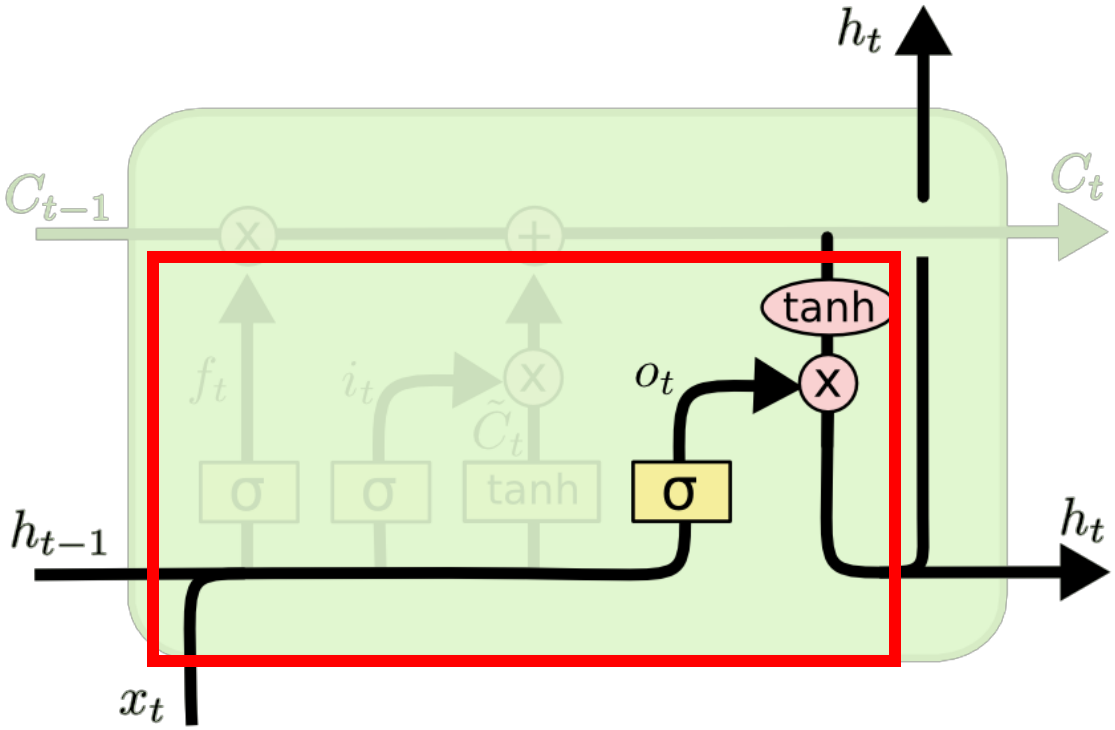


$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$$

elementwise multiplication of vectors

LSTM: Output Gate

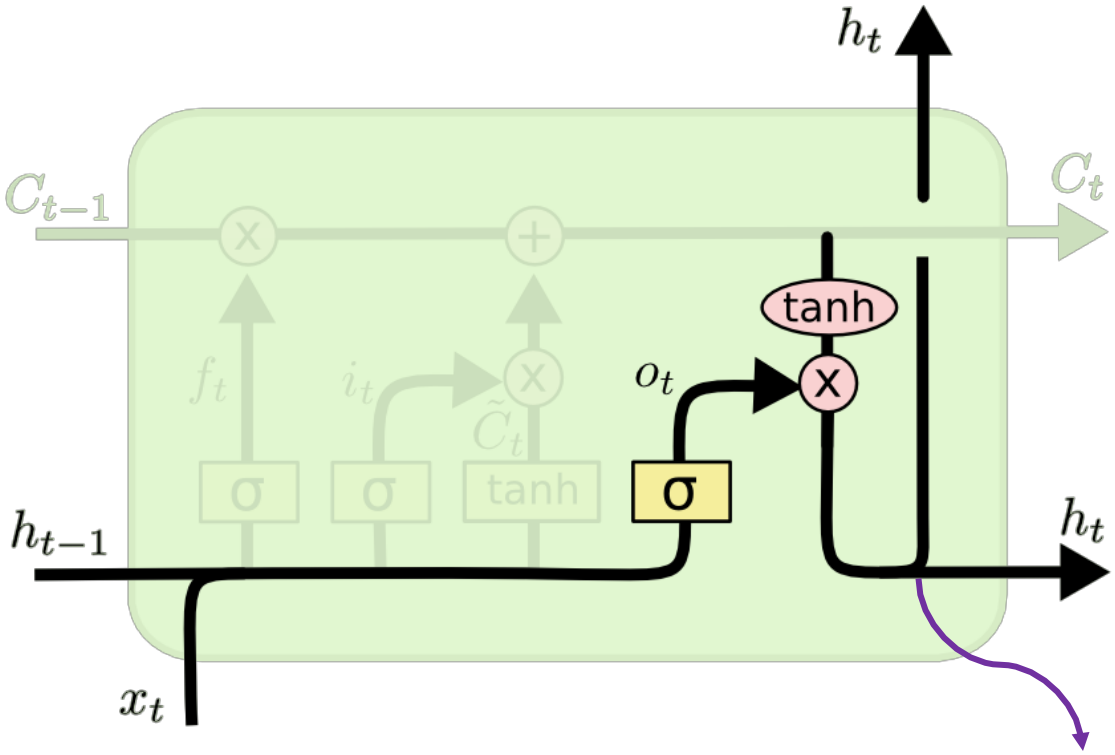
- Output gate (o_t): decide what flows from the conveyor belt C_{t-1} to the hidden state h_t.



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

LSTM: Output Gate

- Output gate (o_t): decide what flows from the conveyor belt C_{t-1} to the hidden state h_t.

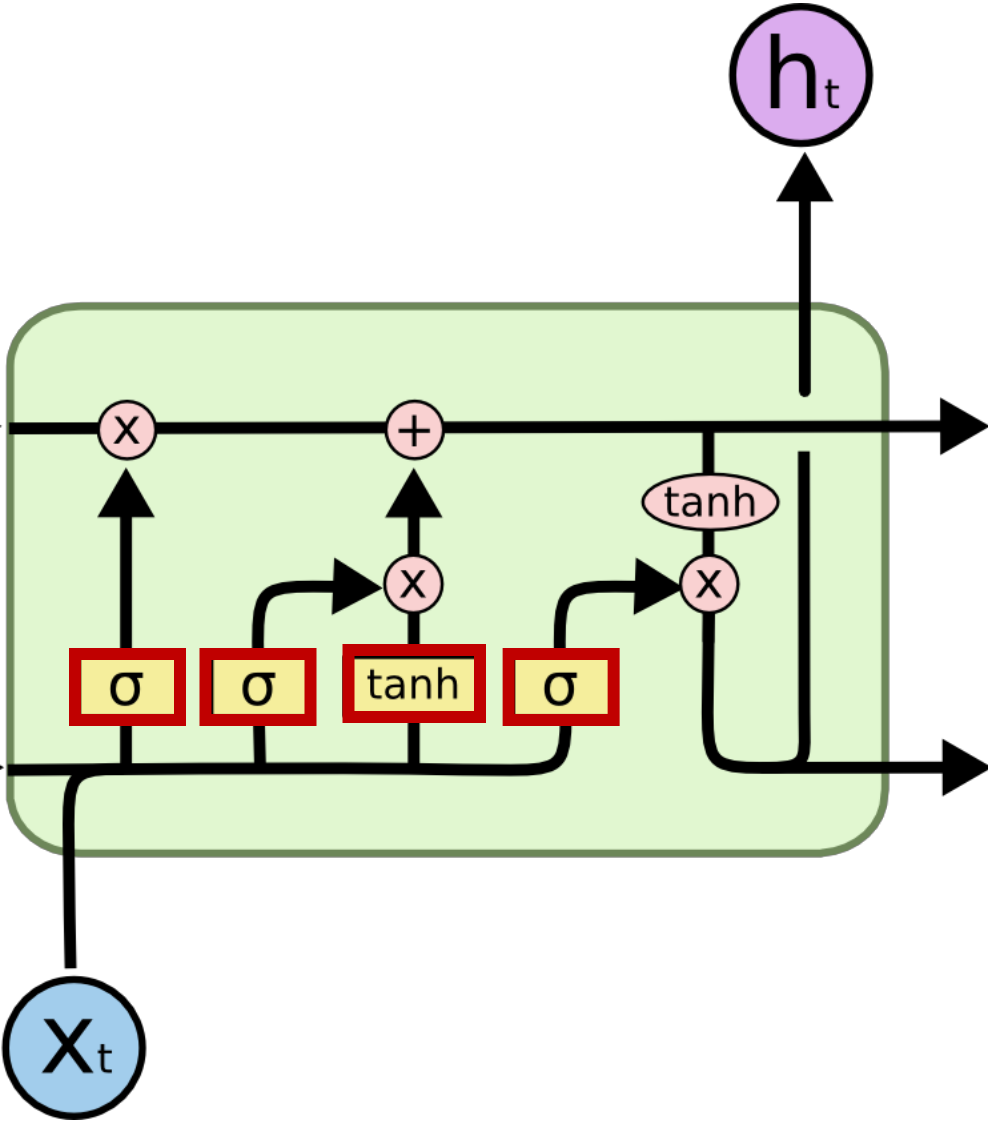


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

Two copies of h_t

LSTM: Number of Parameters



#parameters: 4 times as many as SimpleRNN

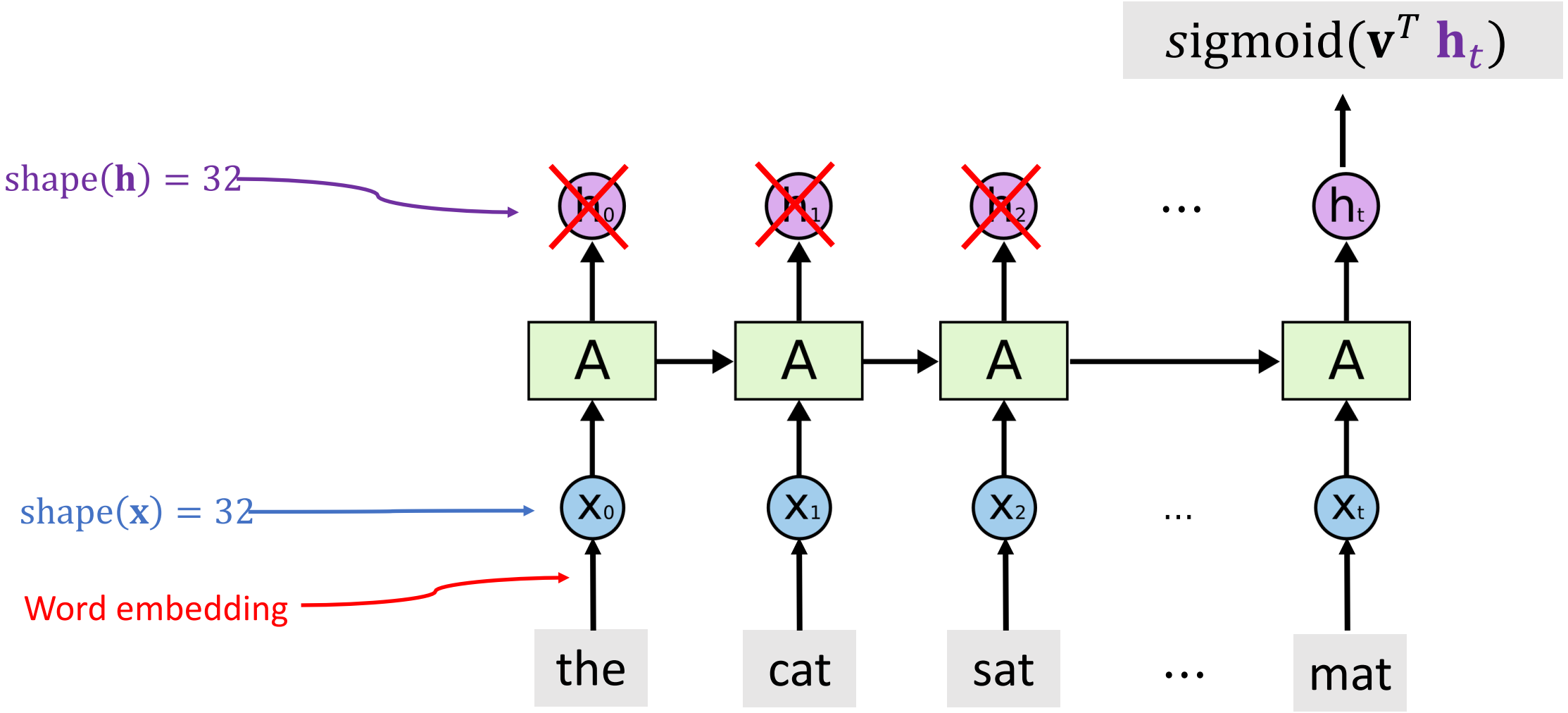
- 4 parameter matrices, each of which has
  - `#rows: shape(h)`
  - `#cols: shape(h)+shape(x)`

- #parameter (do not count intercept):

`4 × shape(h) × [shape(h)+shape(x)]`

LSTM Using Keras

LSTM for IMDB Review



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

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 for IMDB Review

- Training Accuracy: 91.8%
- Validation Accuracy: 88.7%
- Test Accuracy: 88.6%

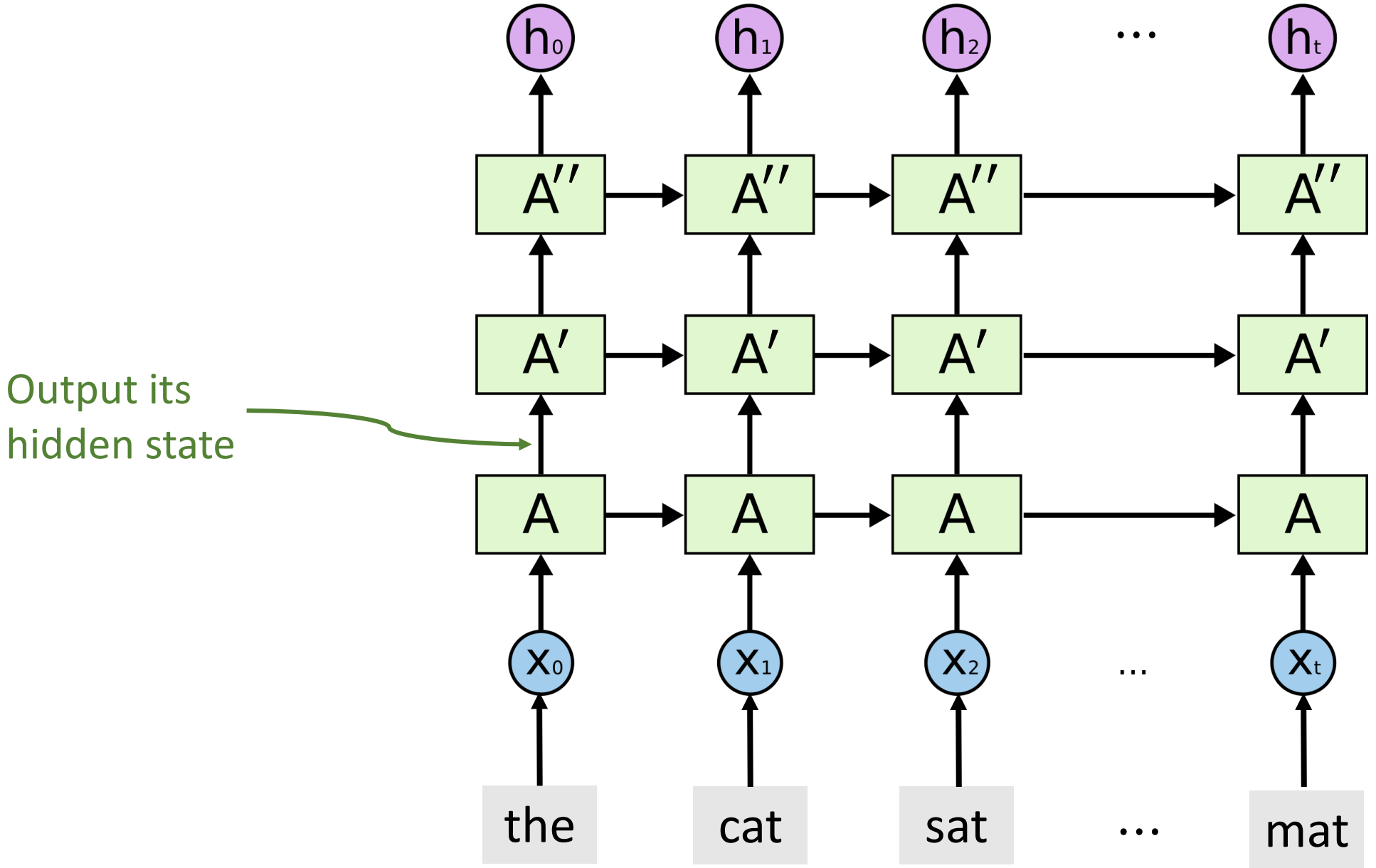
Substantial improvement over SimpleRNN (whose accuracy is 84%).

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()
```

Stacked RNN

Stacked RNN (LSTM)



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'))
```

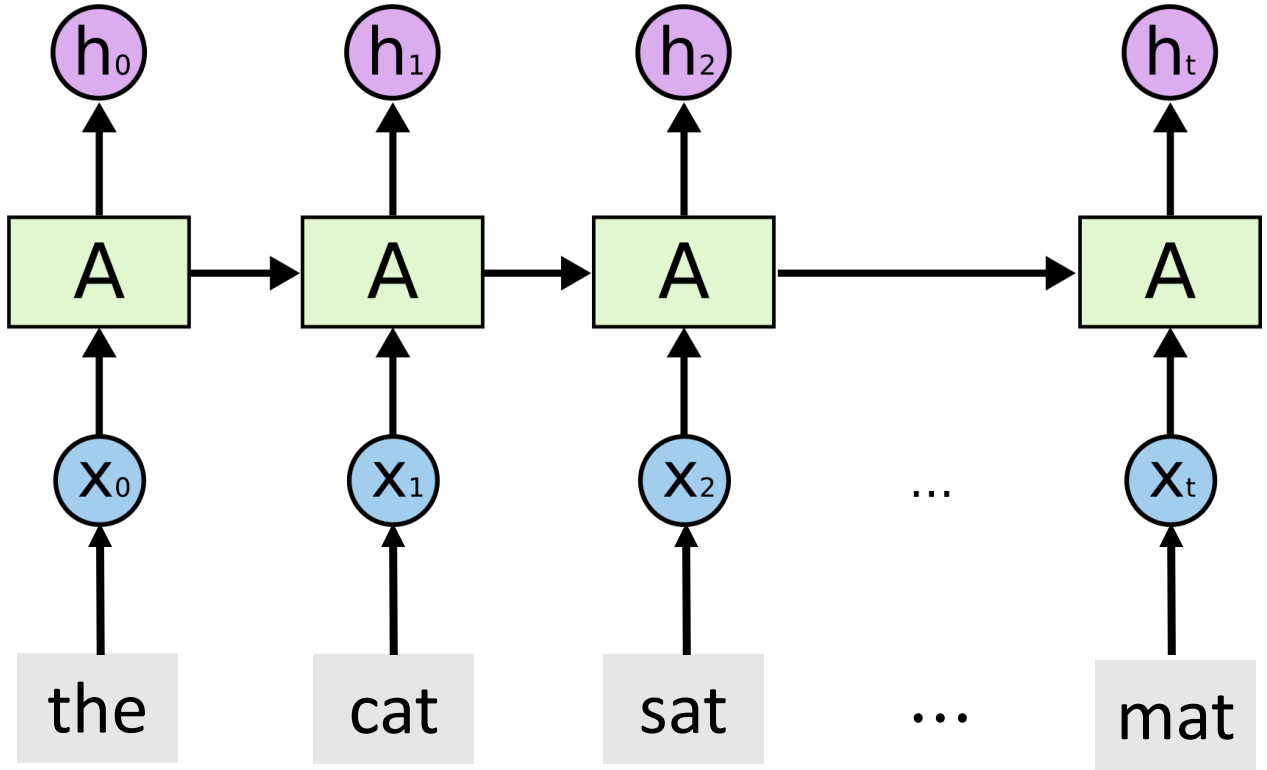
Stacked LSTM

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 32)	320000
lstm_1 (LSTM)	(None, 500, 32)	8320
lstm_2 (LSTM)	(None, 500, 32)	8320
lstm_3 (LSTM)	(None, 32)	8320
dense_1 (Dense)	(None, 1)	33

Total params: 344,993
Trainable params: 344,993
Non-trainable params: 0

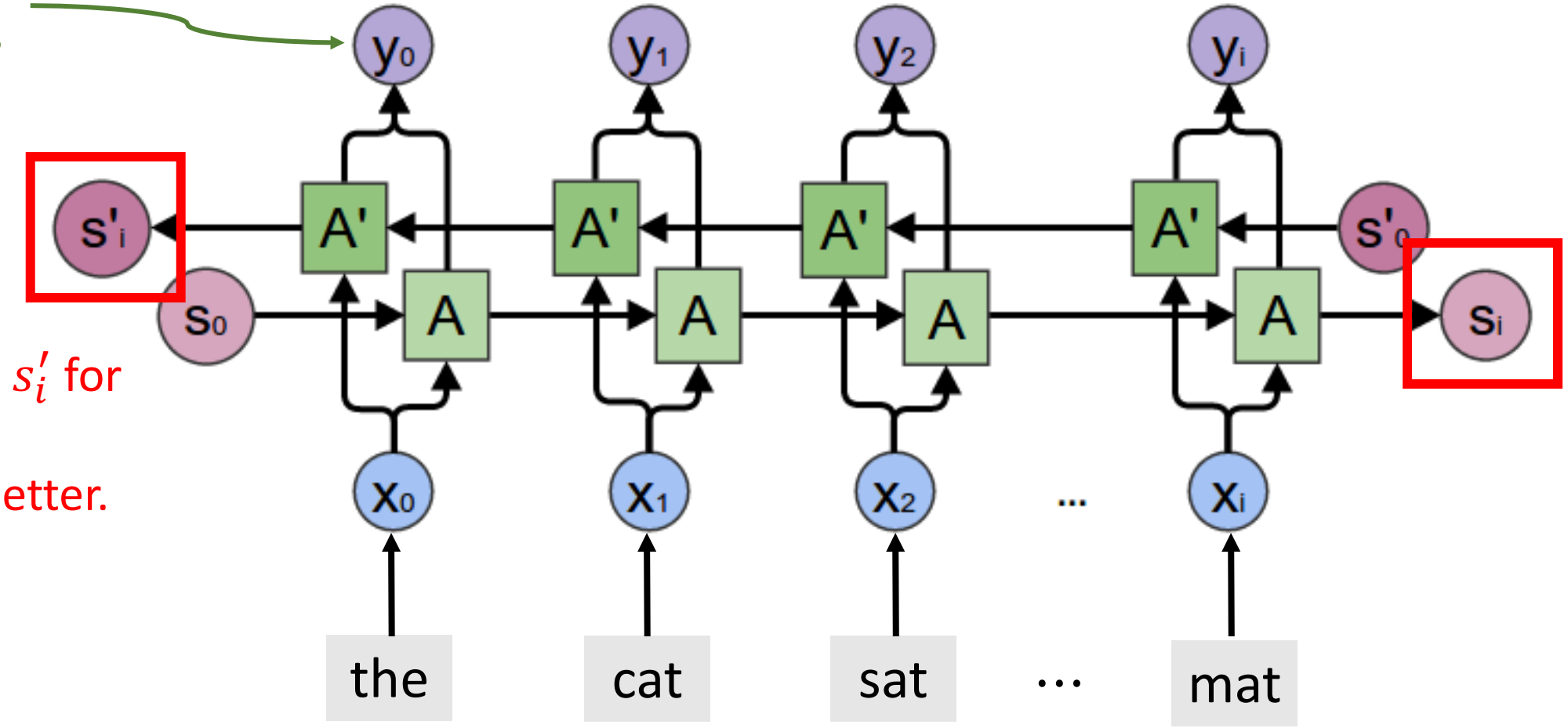
Bidirectional RNN

RNN (LSTM) in One Direction



Bidirectional RNN (LSTM)

Stack of the 2 hidden states



Using both s_i and s_i' for classification, the accuracy will be better.

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

Pretrain

Why and How Pretraining?

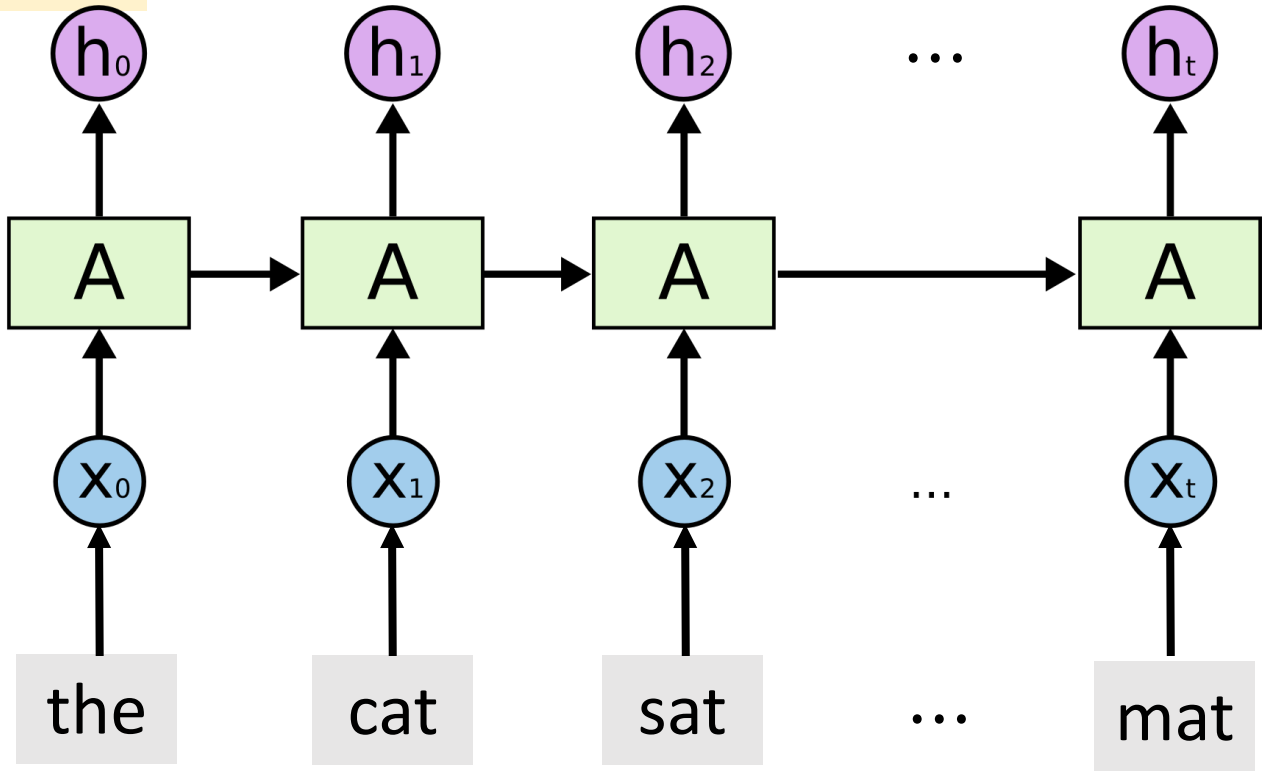
Observation: The embedding layer contributes most of the parameters!

```
_________________________________________________________________
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
```

Trick: Pretrain the Embedding Layer

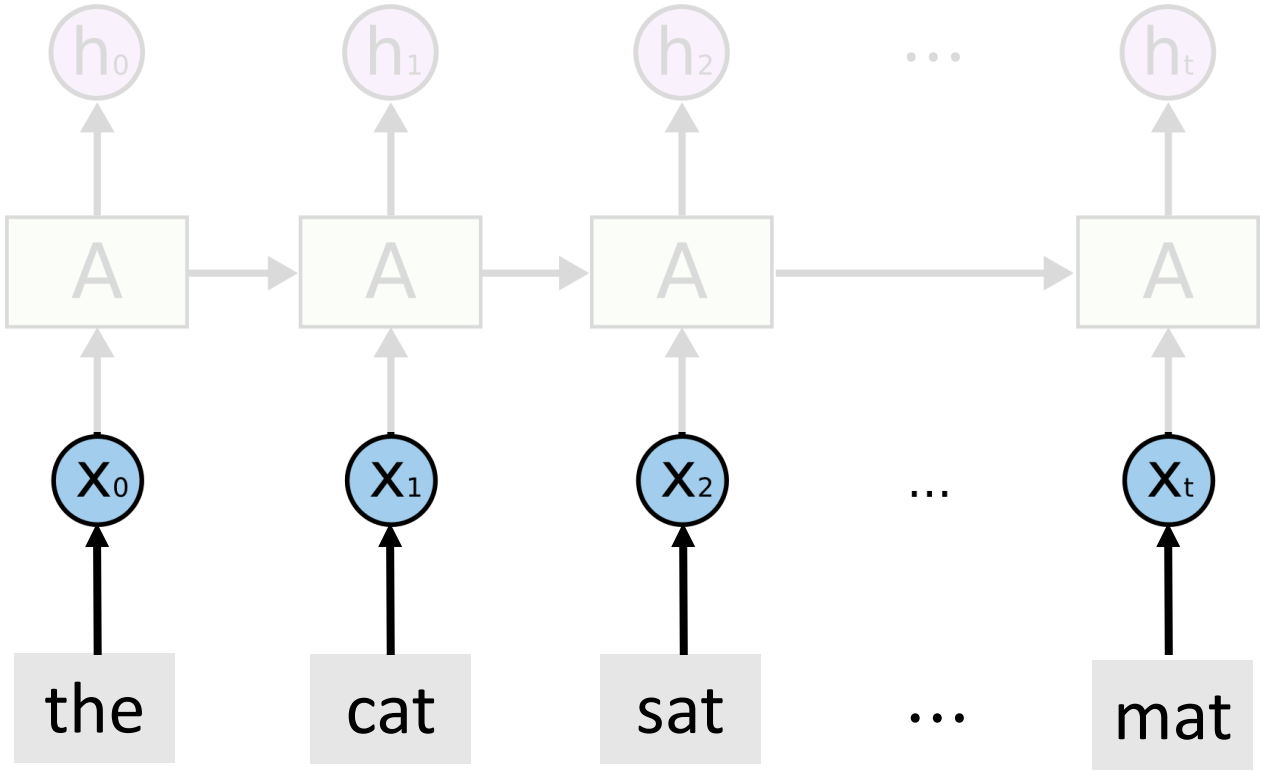
Step 1: Train a model on large dataset.

- Perhaps different problem.
- Perhaps different model.



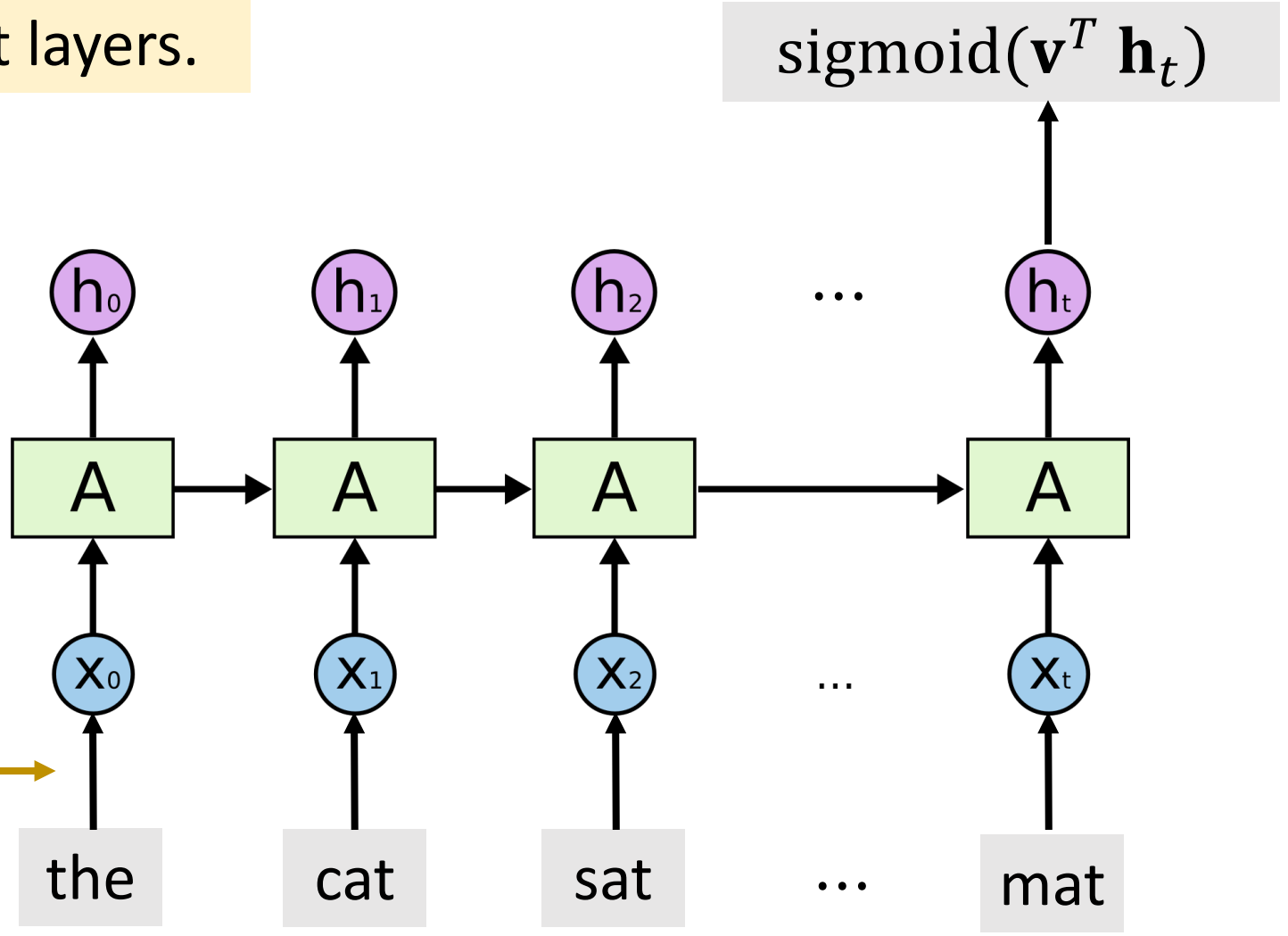
Trick: Pretrain the Embedding Layer

Step 2: Keep only the embedding layer.



Trick: Pretrain the Embedding Layer

Step 3: Train new LSTM and output layers.

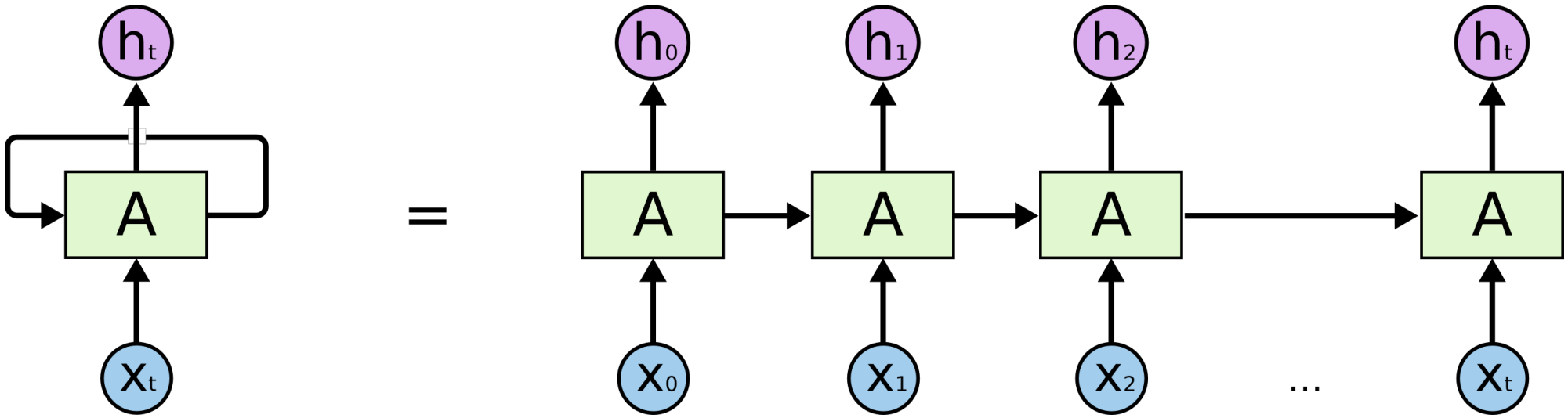


Set the embedding layer non-trainable.

Summary

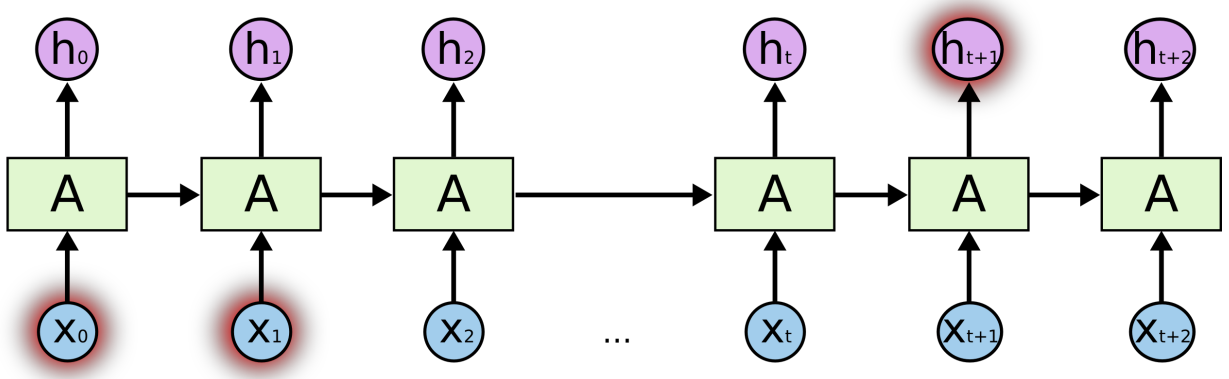
Summary

- RNN for text, audio, and time series data.
- Hidden state $\mathbf{h}_t$ aggregates information in the inputs $\mathbf{x}_0, \cdots, \mathbf{x}_t$.



Summary

- RNN for text, audio, and time series data.
- Hidden state $\mathbf{h}_t$ aggregates information in the inputs $\mathbf{x}_0, \cdots, \mathbf{x}_t$.
- All the RNN suffers from vanishing gradient problem.
  - It forgets what it has seen early on.
  - If t is large, $\mathbf{h}_t$ is almost irrelevant to $\mathbf{x}_0$.



Summary

- RNN for text, audio, and time series data.
- Hidden state $\mathbf{h}_t$ aggregates information in the inputs $\mathbf{x}_0, \cdots, \mathbf{x}_t$.
- All the RNN suffers from vanishing gradient problem.
  - It forgets what it has seen early on.
  - If t is large, $\mathbf{h}_t$ is almost irrelevant to $\mathbf{x}_0$.
- LSTM uses a "conveyor belt" to alleviate the vanishing gradient problem.
  - LSTM is better than SimpleRNN.
  - However, LSTM forgets what it has seen hundreds of steps ago.

Number of Parameters

- SimpleRNN has a parameter matrix (and perhaps an intercept vector).
- Shape of the parameter matrix is

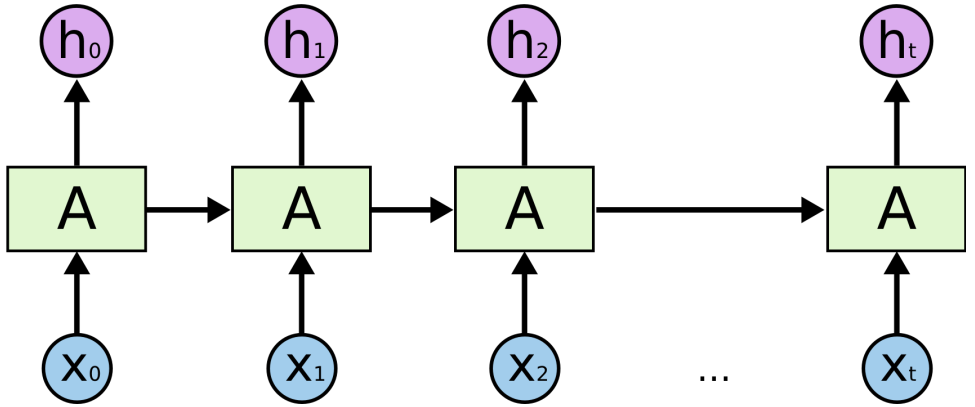
$$\texttt{shape(h)} \times \texttt{[shape(h)+shape(x)]}.$$

Number of Parameters

- SimpleRNN has a parameter matrix (and perhaps an intercept vector).
- Shape of the parameter matrix is

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

- Only one such parameter matrix, no matter how long the sequence is.
  - All the modules share the parameter matrix.



Number of Parameters

- SimpleRNN has a parameter matrix (and perhaps an intercept vector).
- Shape of the parameter matrix is

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

- Only one such parameter matrix, no matter how long the sequence is.
  - All the modules share the parameter matrix.

- LSTM has 4 such parameter matrices.
  - Thus 4 times as many parameters as SimpleRNN.

Rules of Thumb

- Always use LSTM instead of SimpleRNN.
- Always use LSTM dropout to alleviate overfitting.
- Use Bi-LSTM whenever possible.
- Stacked LSTM instead of a single LSTM layer (if the sample size is big).
- Pretrain the embedding layer (if the sample size is small).