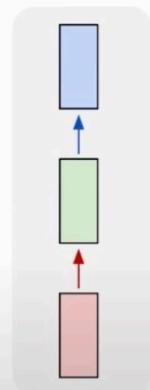
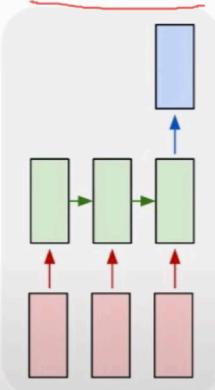


How to model sequential data?

one to one



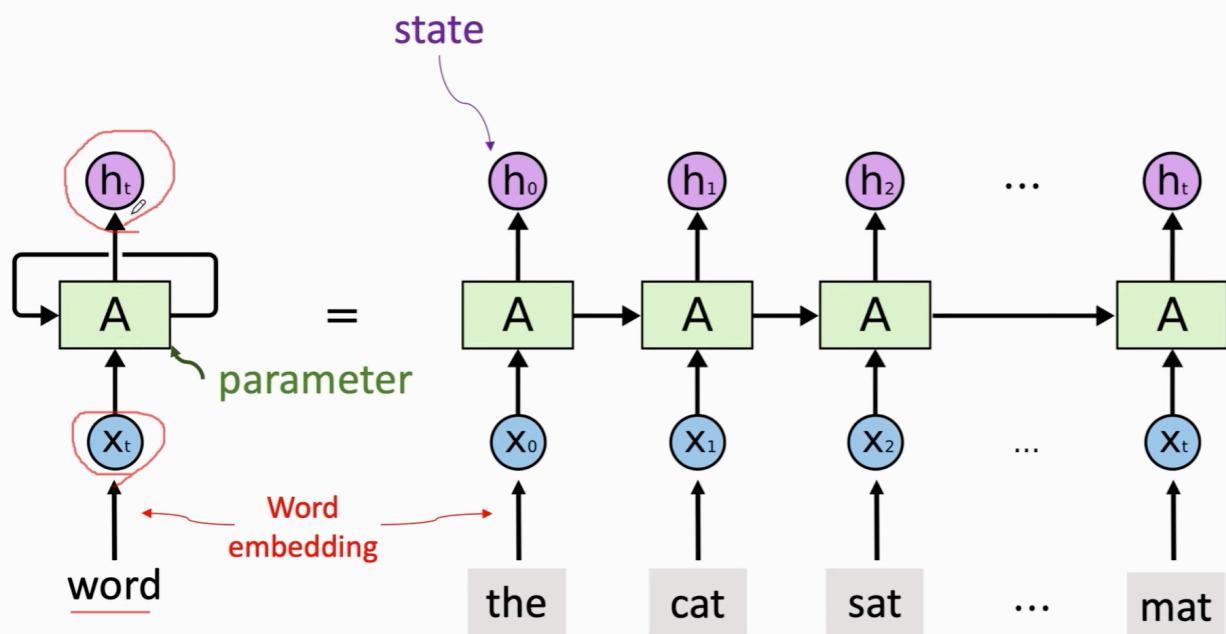
many to one



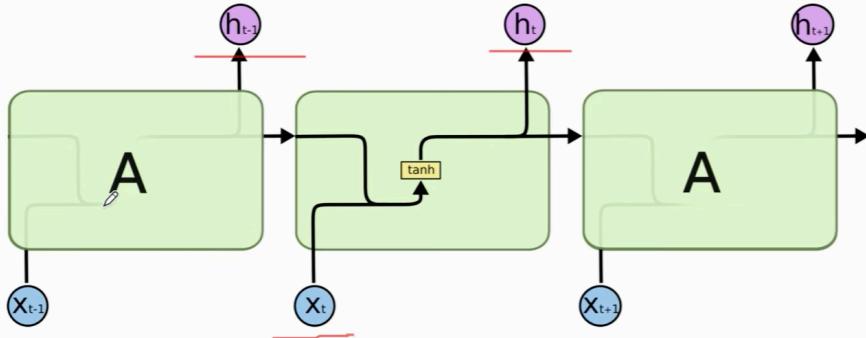
- Limitations of FC Nets and ConvNets:

- Process a paragraph as a whole.
- Fixed-size input (e.g., image).
- Fixed-size output (e.g., predicted probabilities).
- RNNs are better ways to model the sequential data (e.g., text, speech, and time series).

Recurrent Neural Networks (RNNs)



Simple RNN



$$h_t = \tanh \left[A \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \right]$$

hyperbolic tangent function

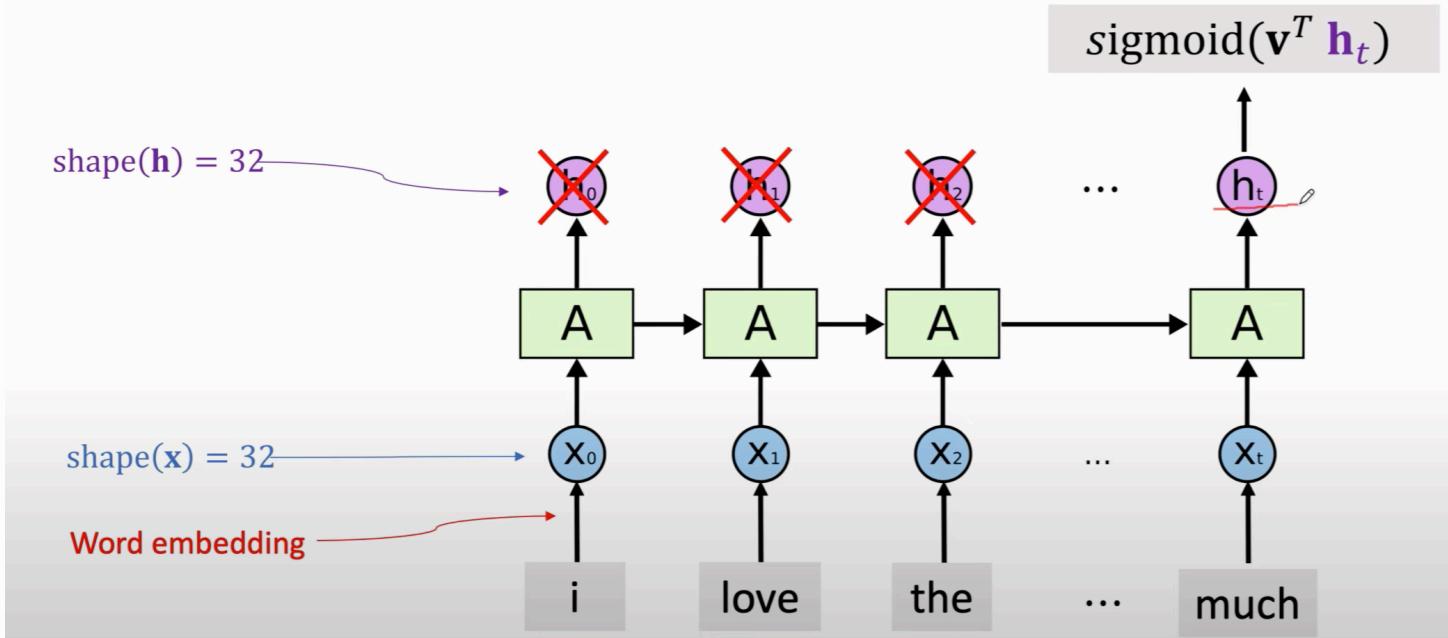
Simple RNN

Trainable parameters: matrix **A**

- #rows of **A**: shape (**h**)
- #cols of **A**: shape (**h**) + shape (**x**)
- Total #parameter: $\frac{\text{shape}(\mathbf{h})}{\cancel{\text{shape}(\mathbf{h})}} \times [\text{shape}(\mathbf{h}) + \text{shape}(\mathbf{x})]$

$$h_t = \tanh \left[A \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \right]$$

Simple RNN for IMDB Review



Simple RNN for IMDB Review

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

vocabulary = 10000 #unique words in the dictionary
embedding_dim = 32 shape(x) = 32
word_num = 500 sequence length
state_dim = 32 shape(h) = 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()
```

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	#parameters in RNN:	
Non-trainable params: 0	$2080 = 32 \times (32 + 32) + 32$	

shape(h) = 32
 shape(x)

Simple RNN for IMDB Review

```
from keras import optimizers  
  
epochs = 3  → Early stopping alleviates overfitting  
  
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))
```

```
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
```

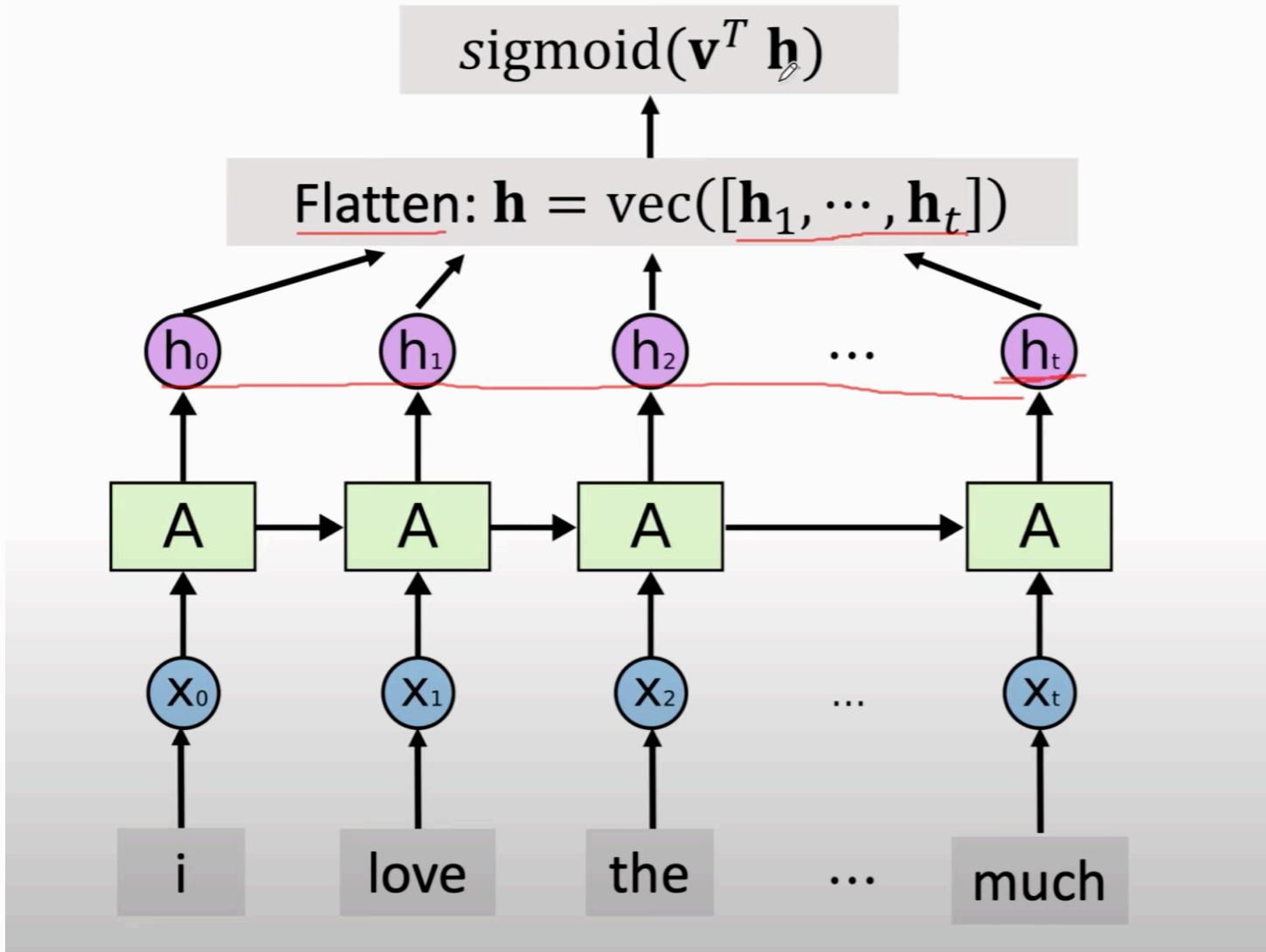
Simple RNN for IMDB Review

```
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



Simple RNN for IMDB Review

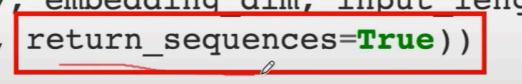
```
from keras.models import Sequential
from keras.layers import SimpleRNN, Embedding, Dense

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'))

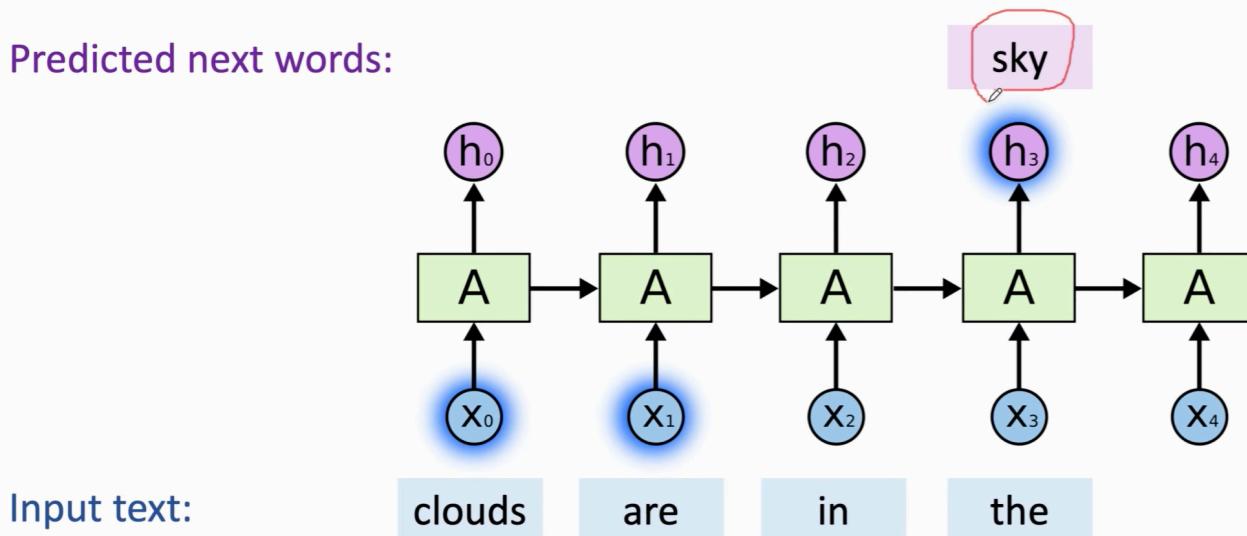
model.summary()
```

Return all the states h_1, \dots, h_t .



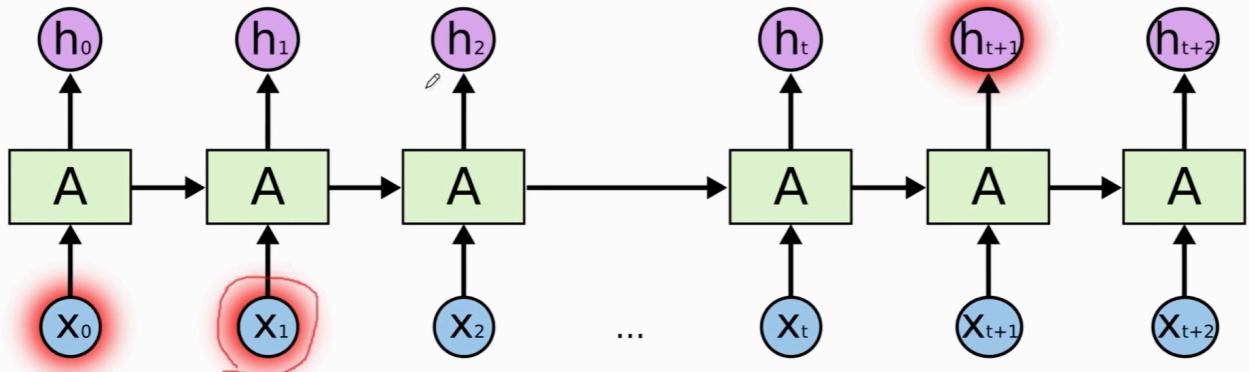
SimpleRNN is good at short-term dependence.

Predicted next words:



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

SimpleRNN is bad at long-term dependence.

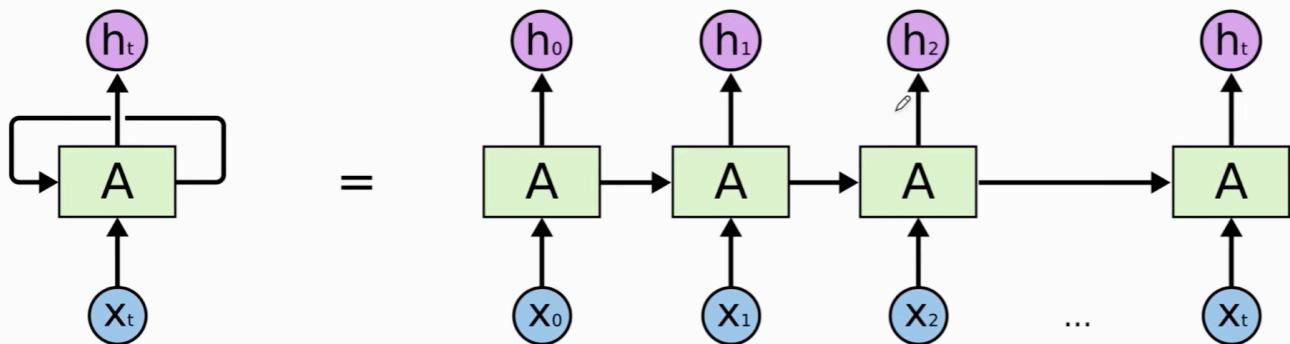


h_{100} is almost irrelevant to x_1 : $\frac{\partial h_{100}}{\partial x_1}$ is near zero.

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

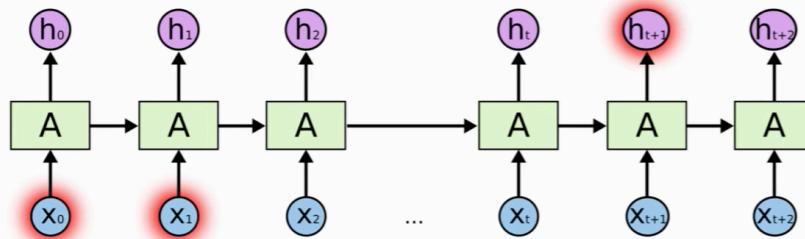
Summary

- RNN for text, speech, and time series data.
- Hidden state h_t aggregates information in the inputs x_0, \dots, x_t .



Summary

- RNN for text, speech, and time series data.
- Hidden state \mathbf{h}_t aggregates information in the inputs $\mathbf{x}_0, \dots, \mathbf{x}_t$.
- RNNs can forget early inputs.
 - It **forgets** what it has seen early on.
 - If t is large, \mathbf{h}_t is almost irrelevant to \mathbf{x}_0 .



Number of Parameters

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

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