RNN & LSTM

Dealing with text data:

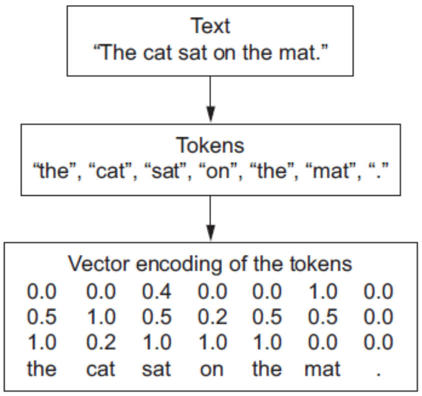
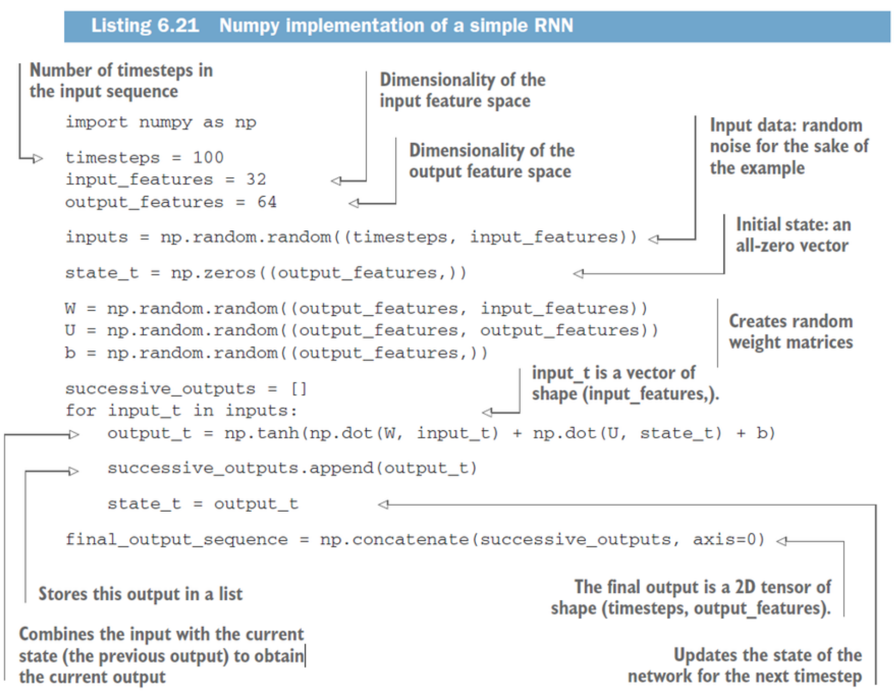


Figure 6.1 From text to tokens to vectors

from token to vector:

1. one hot (too sparse)
2. Ebedding(word index → embedding layer → coresponding word vector):
   a. embeding with your data for your task
   b. emdeding with pretrained data(works for small dataset)

Understanding of Recurrent neural networks



Listing 6.21 Numpy implementation of a simple RNN

```
import numpy as np

timesteps = 100
input_features = 32
output_features = 64

inputs = np.random.random((timesteps, input_features))

state_t = np.zeros((output_features,))

W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))

successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)

    successive_outputs.append(output_t)

    state_t = output_t

final_output_sequence = np.concatenate(successive_outputs, axis=0)
```



```
Setting: input : (100,32) output(100,64)

for each time t:
  output = F(input_t, state_t)
  state_t = output_t   #update the state with cur output
  result_lis.append(output)  # store the output

```

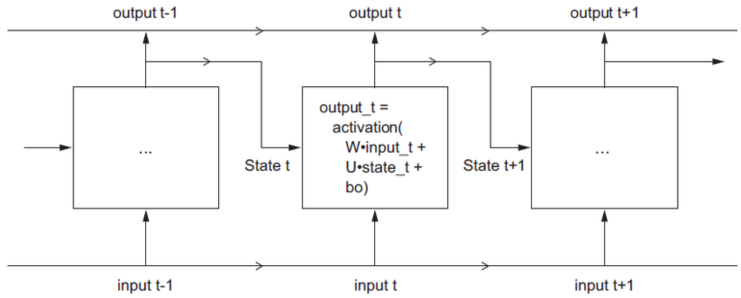


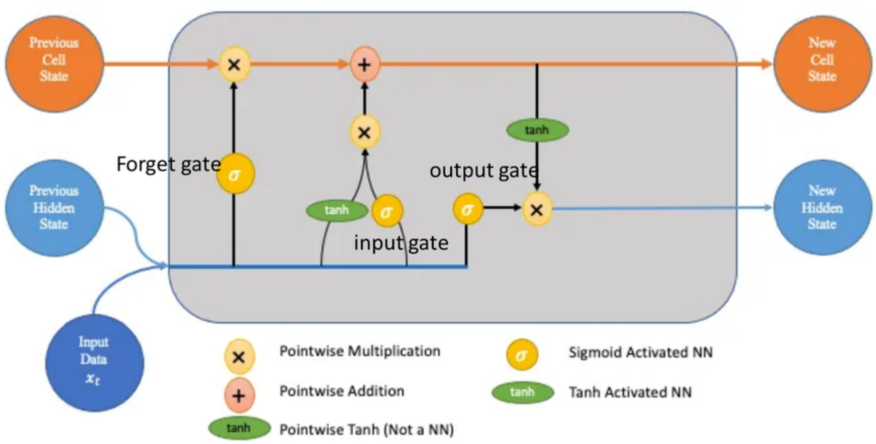
Figure 6.10 A simple RNN, unrolled over time

The key point here is that each cur output take the cur state as input, where store the previous output

For this reason, you don't need this full sequence of outputs, just take the last output (result = result_lis[-1])

Problem: gradient vanishing. long sequence stored in the state goes through forward propagation and results in deep function series. When do differentiate in backpropagation with chain rule, the grandient will be super small.

LSTM



LSTM Diagram

How does LSTM prevent gradient vanishing?

- The trick is gate/filter. All three sigmoid functions are serve as filter, because they convert value to [0,1] as [iirelavent, relavant]. Follow with point multiplcation as application of the filter to different states/data with different purposes
- Those filters simlipify the deep function series by filtering out some irrelvant data

How do different filters work?

- input into all three filteres are the **same(the weight are diff): input data X-t and hidden staste_t**, meaning that filters created in this way generally work for the whole current network (not sure why)
- Three filters work at different part of the network:
  - **Forget gate**: apply to previous cell states to forget irrelevant information in the carry dataflow
  - **Input gate**: apply to input data (map to [-1,1] by tech) to filter out irrelevant input

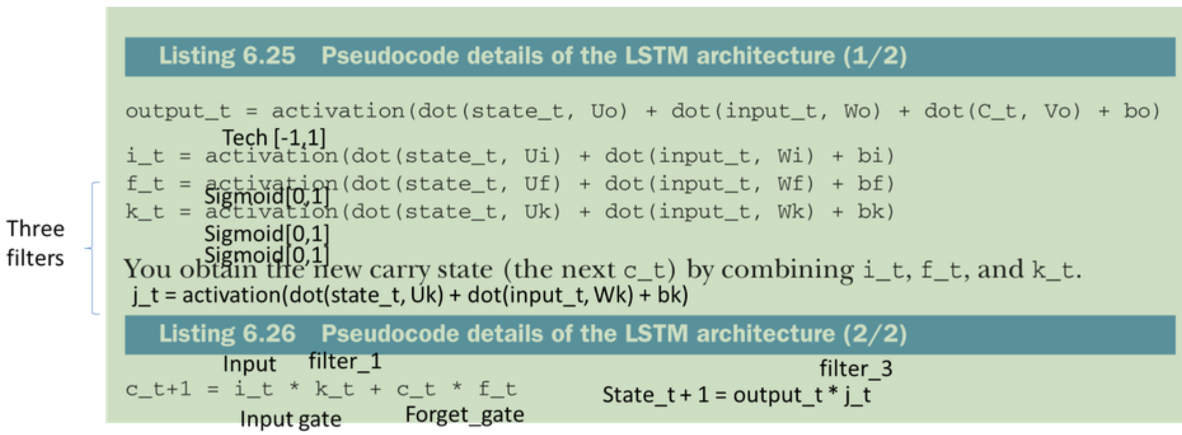
- **output gate**: apply to output data(sum [forget_gate(filtered cell state) + input_gate(filtered input data)] and map to [-1,1] by tech) to filter out irrelevant output

How to update different states?

- cell states are updated by the sum [forget_gate(filtered cell state) + input_gate(filtered input data)]
- hidden States are updated by output gate (filtered updated cell state with tech)

Here note cell states only carry filtered input&pre_cell_state from sigmoid and didn't map to [-1,1]

Pseudocode code of LTSM



Listing 6.25 Pseudocode details of the LSTM architecture (1/2)

```
output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(C_t, Vo) + bo)

i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

You obtain the new carry state (the next c_t) by combining i_t, f_t, and k_t.

Listing 6.26 Pseudocode details of the LSTM architecture (2/2)

```
c_t+1 = i_t * k_t + c_t * f_t
```