DEEP LEARNING

BY M.J.PASSLAR

COMPUTERONIC – TEHRAN – IRAN

2022

CHAPTER 10 : RNN LAYERS

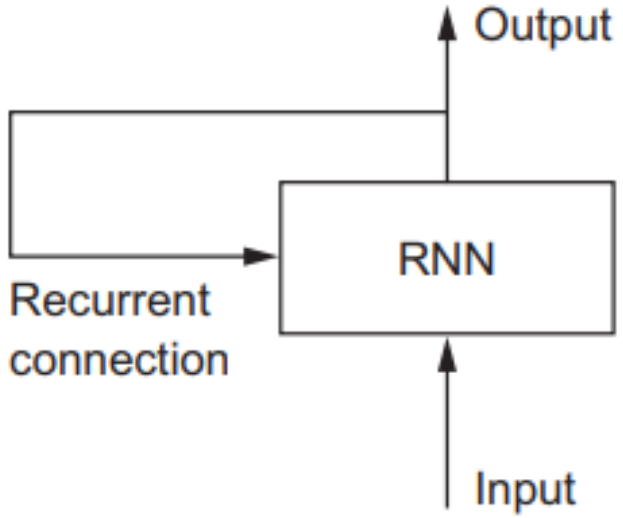
Up to now, there is no memory ...

Feedforward N

- Feed all data set at a time
- There is no memory
- Large tensors

Recurrent N

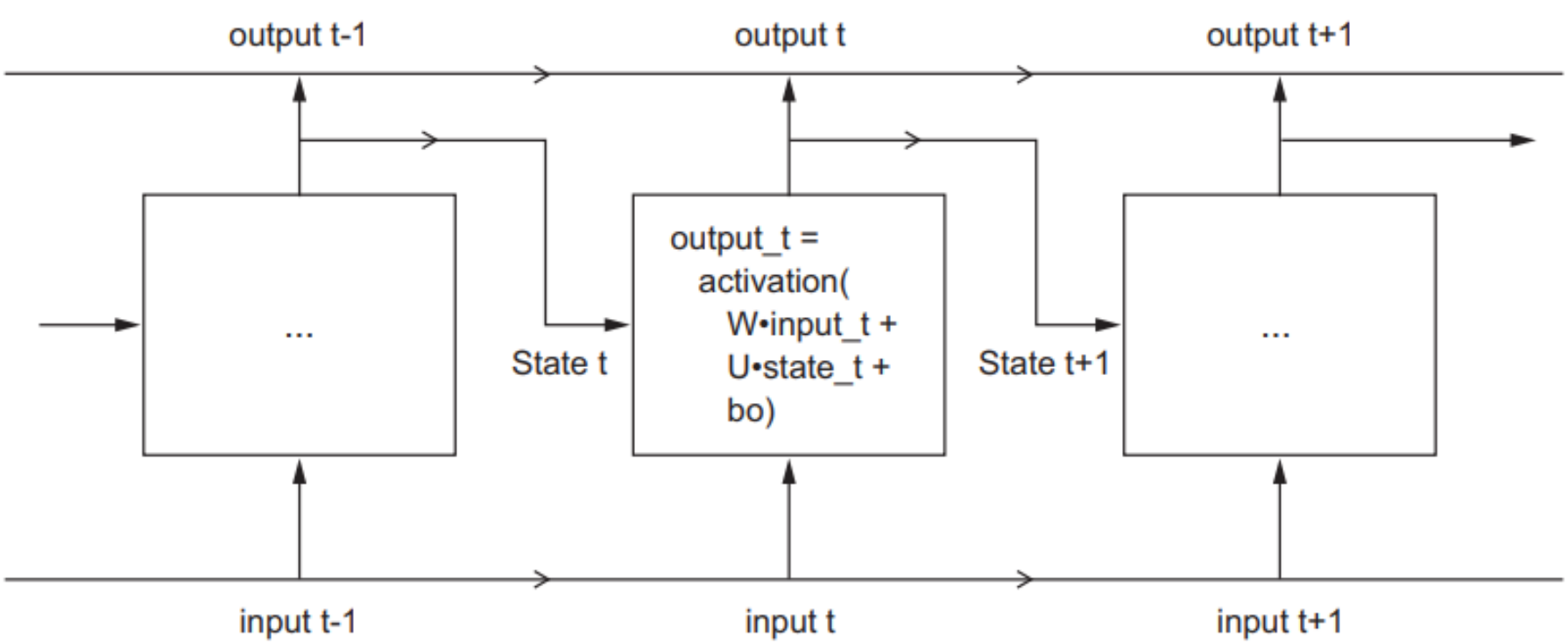
- Feed one data at a time
- there is memory in the network
- Multiple small tensors



RNN layer with numpy

```
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)
```

How it's working?



IMDB with new approach – preparing dataset

```
from keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_features = 10000
maxlen = 500
batch_size = 32
print('Loading data...')
(input_train, y_train), (input_test, y_test) = imdb.load_data(
num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = pad_sequences(input_train, maxlen=maxlen)
input_test = pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

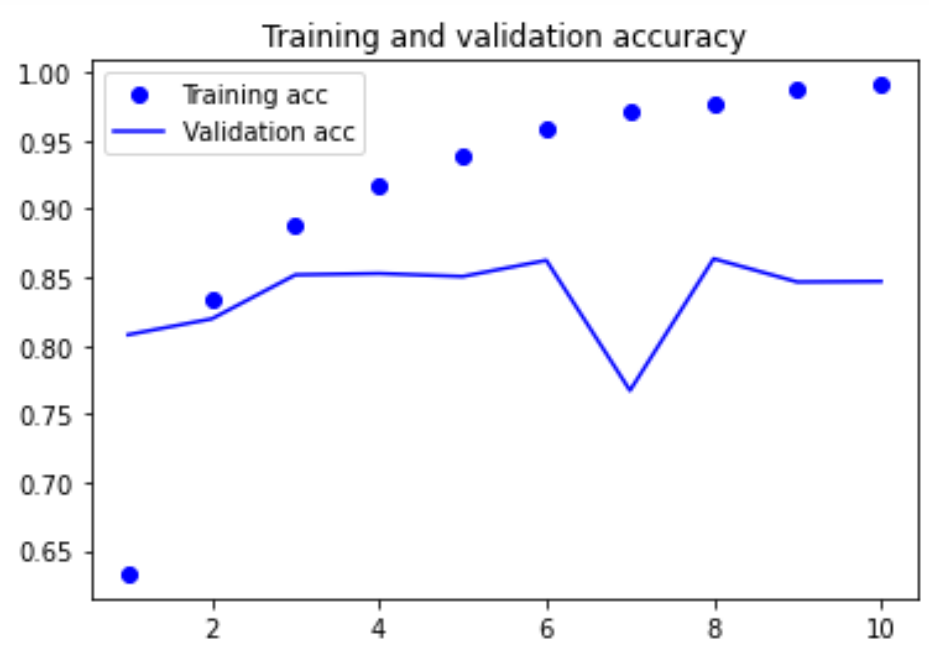
Training the model with RNN and Embedded layer

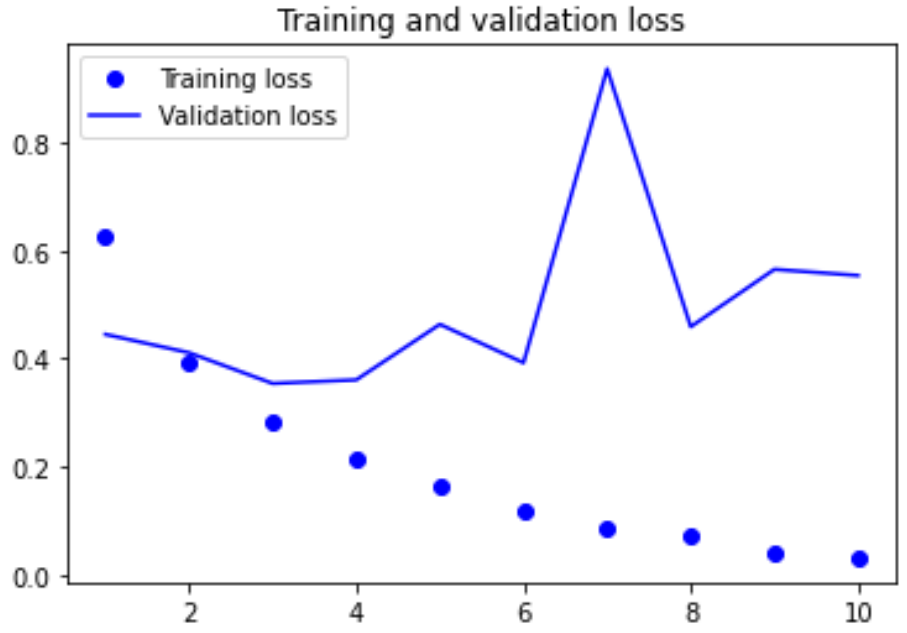
```
from keras.layers import Dense
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

Plotting the result

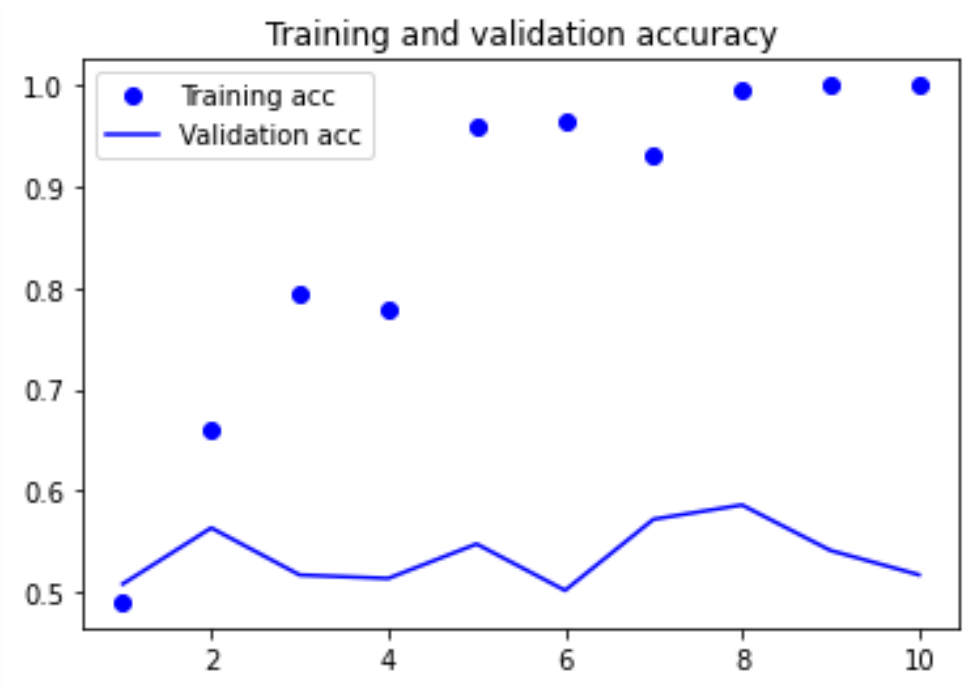
```
import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Loss and validation plot

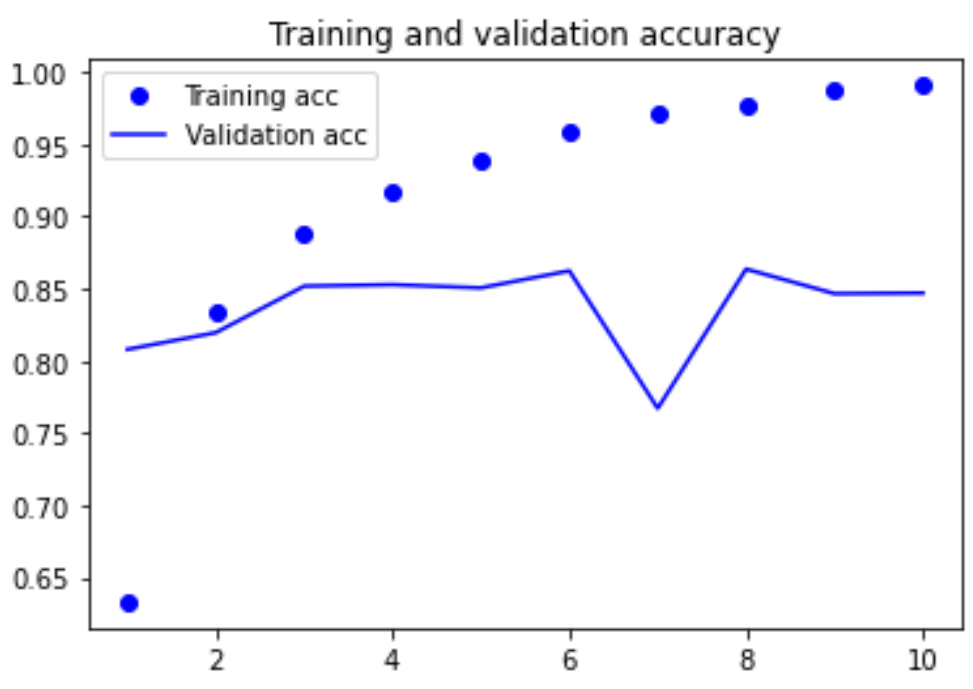




FFN vs RNN

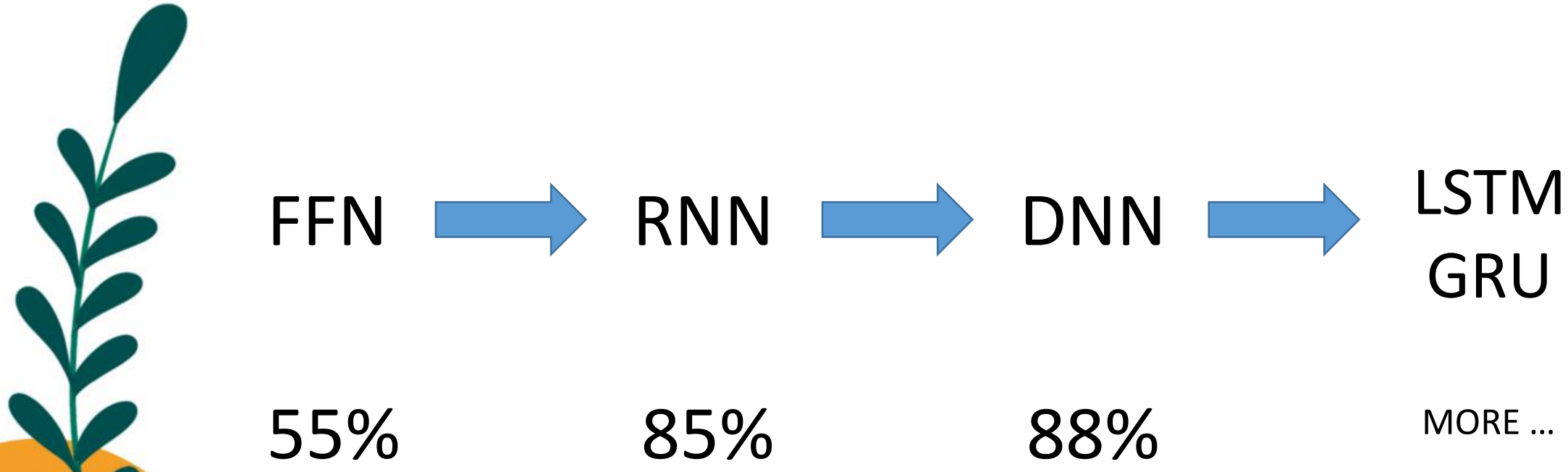


FFN
50%

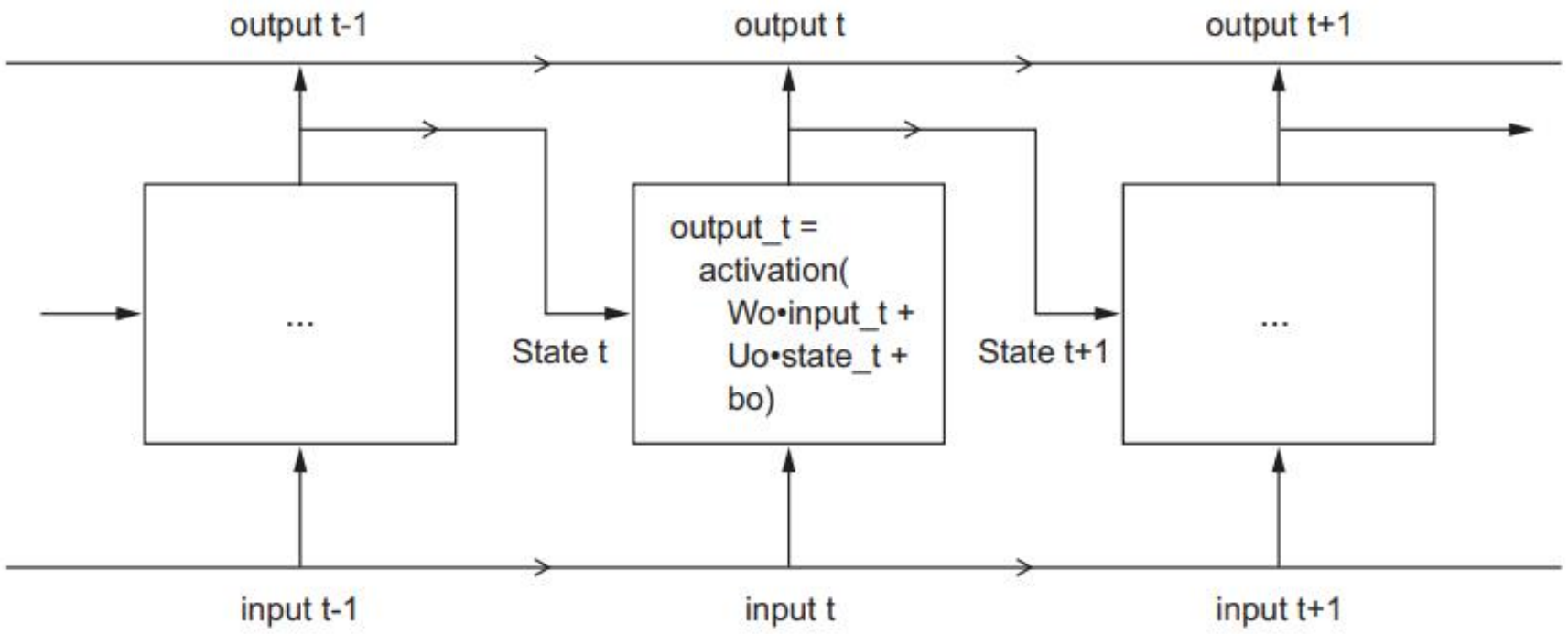


RNN
85%

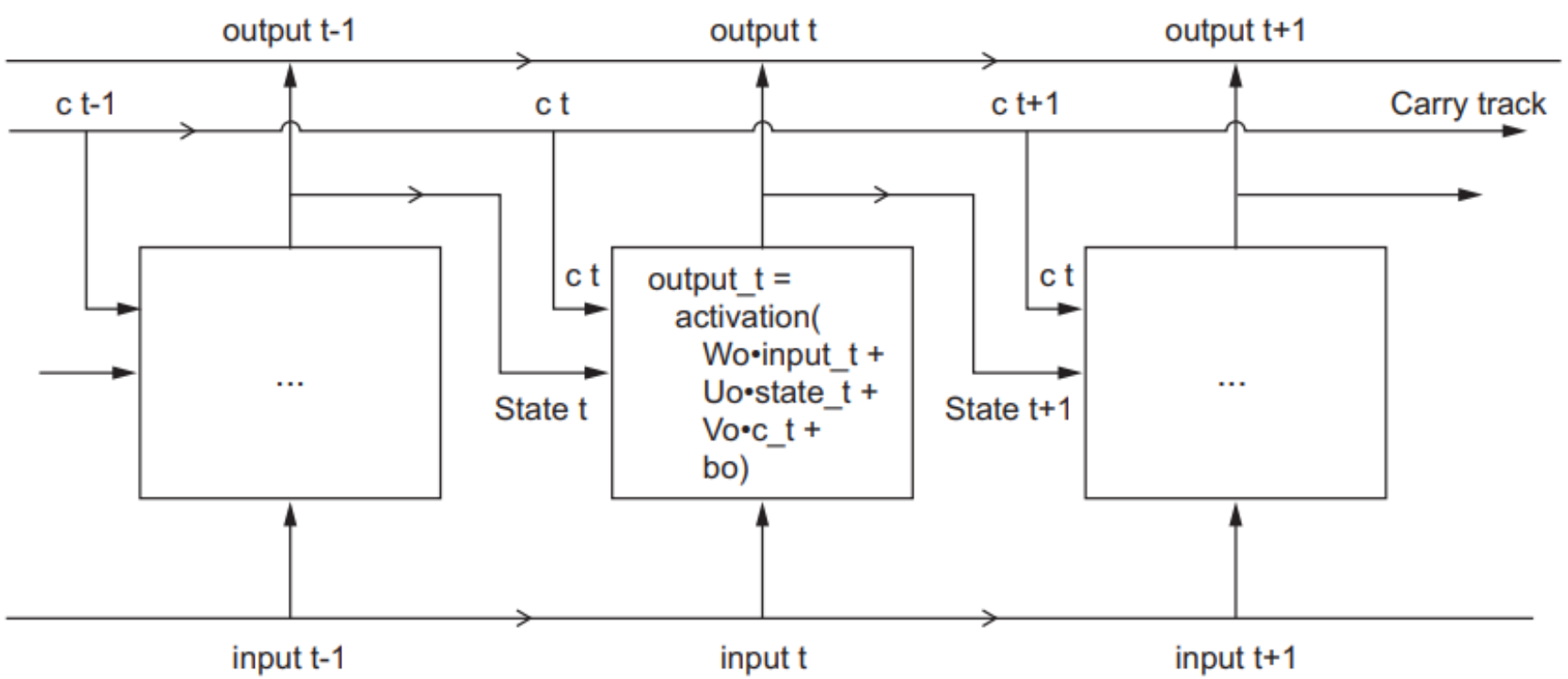
Not enough yet ...



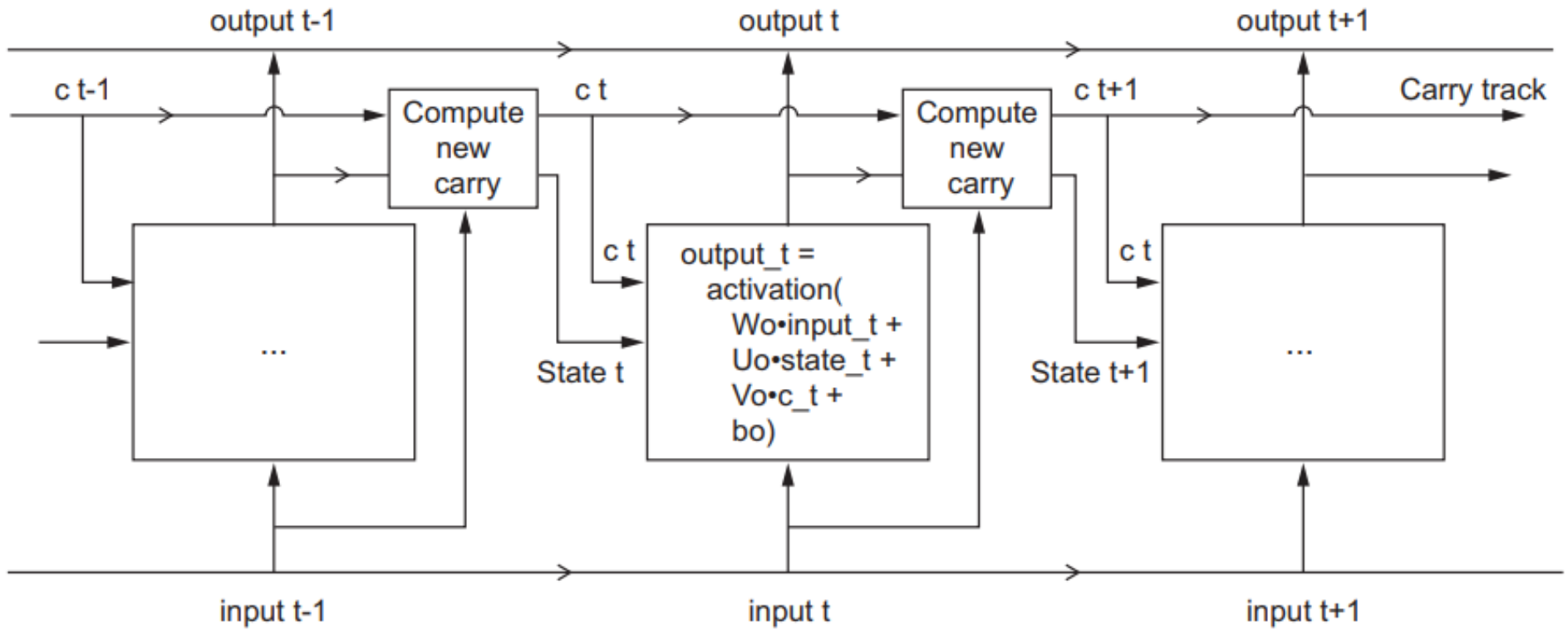
What's LSTM ?


output t-1
output t
output t+1
...
State t
output_t =
activation(
Wo•input_t +
Uo•state_t +
bo)
State t+1
...
input t-1
input t
input t+1

Carry track


output t-1
output t
output t+1
c t-1
c t
c t+1
Carry track
c t
output_t =
activation(
Wo•input_t +
Uo•state_t +
Vo•c_t +
bo)
c t
...
State t
State t+1
...
input t-1
input t
input t+1

Compute new Carry



So what ????

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

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)
```

The best result ...