Language and Other Temporal things Recurrent Nets

Language Modeling

Predict the next word. We'll start with random "weights" for the embeddings and other parameters and run SGD. How do we set up a training set?

Thou shalt not make a machine in the likeness of a human mind

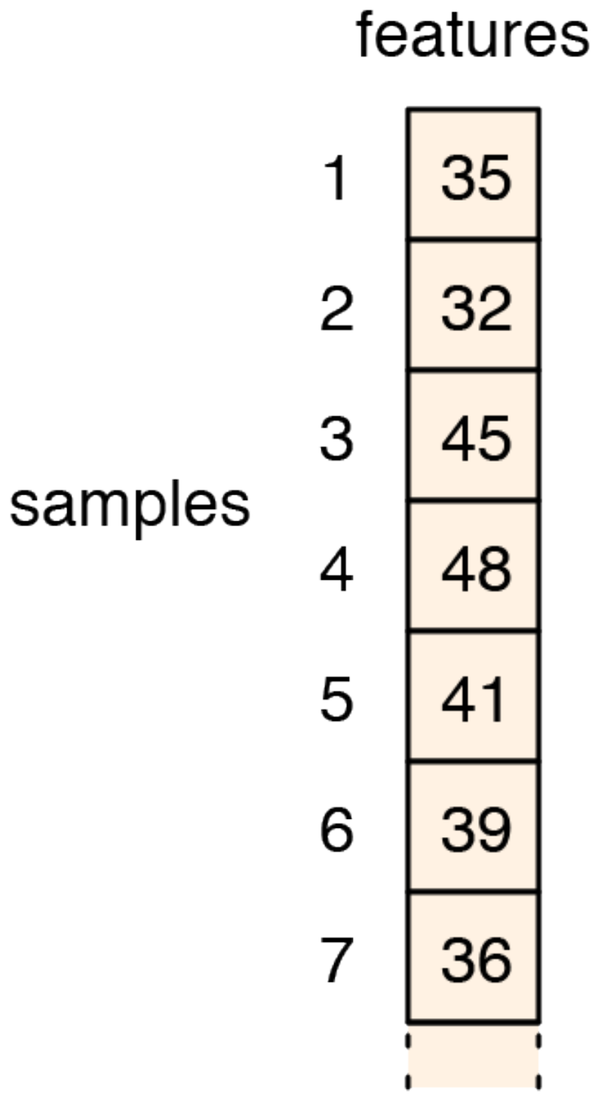
Sliding window across running text

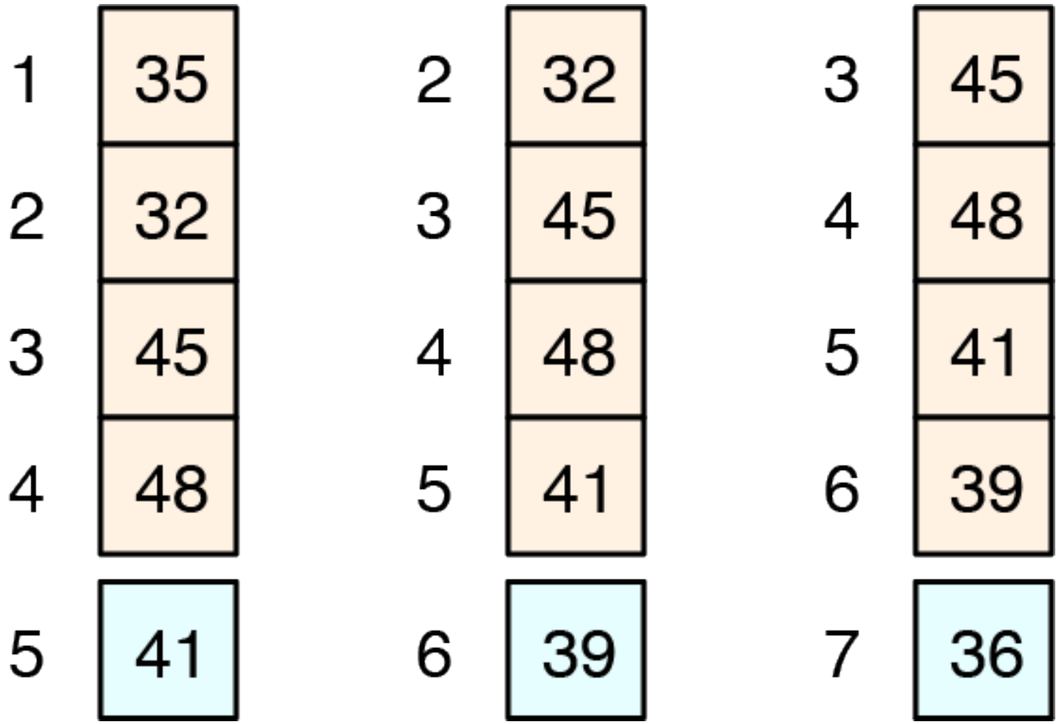
thou	shalt	not	make	a	machine	in	the	…
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	
thou	shalt	not	make	a	machine	in	the	

Dataset

input 1	input 2	output
thou	shalt	not
shalt	not	make
not	make	a
make	a	machine
a	machine	in

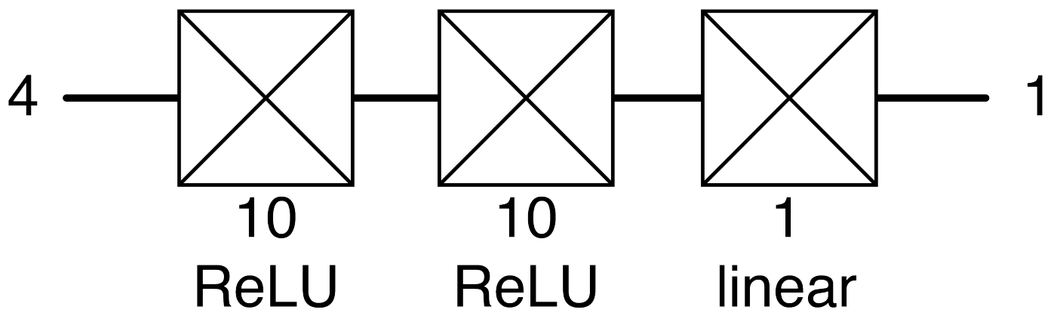
Dealing with Sequences



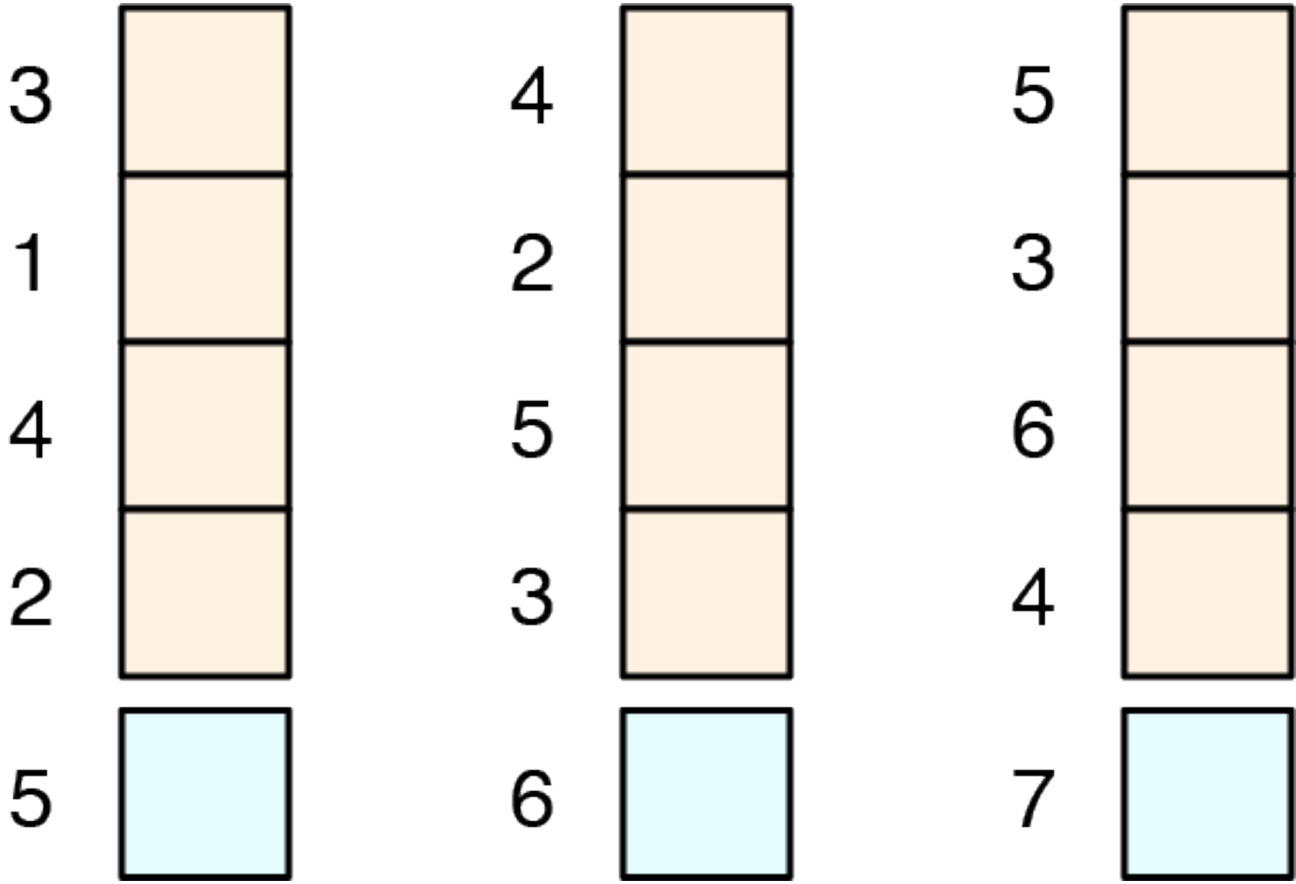


This is a windowed dataset, with a window of size 4 and overlapping windows

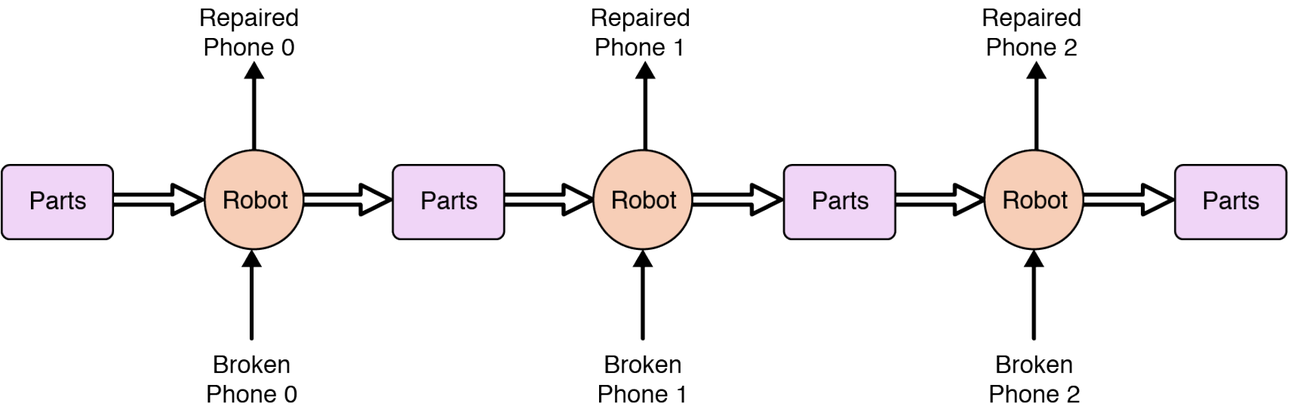
Fitting with a MLP

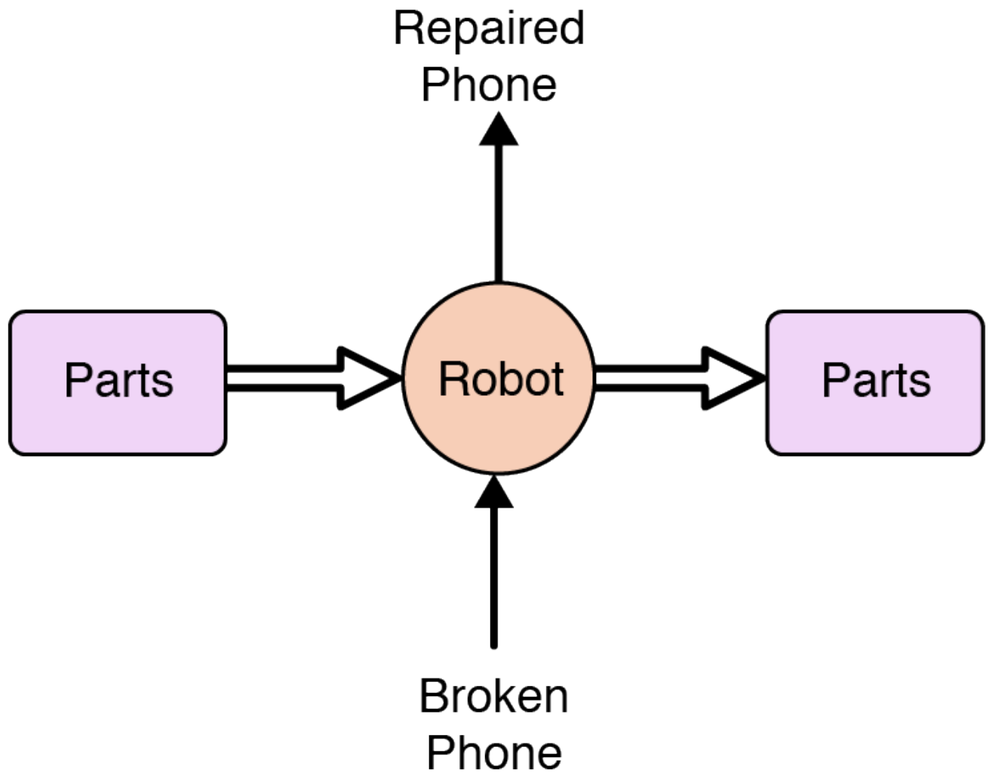


But now, order does not matter! So the model on the right gives equivalent results.



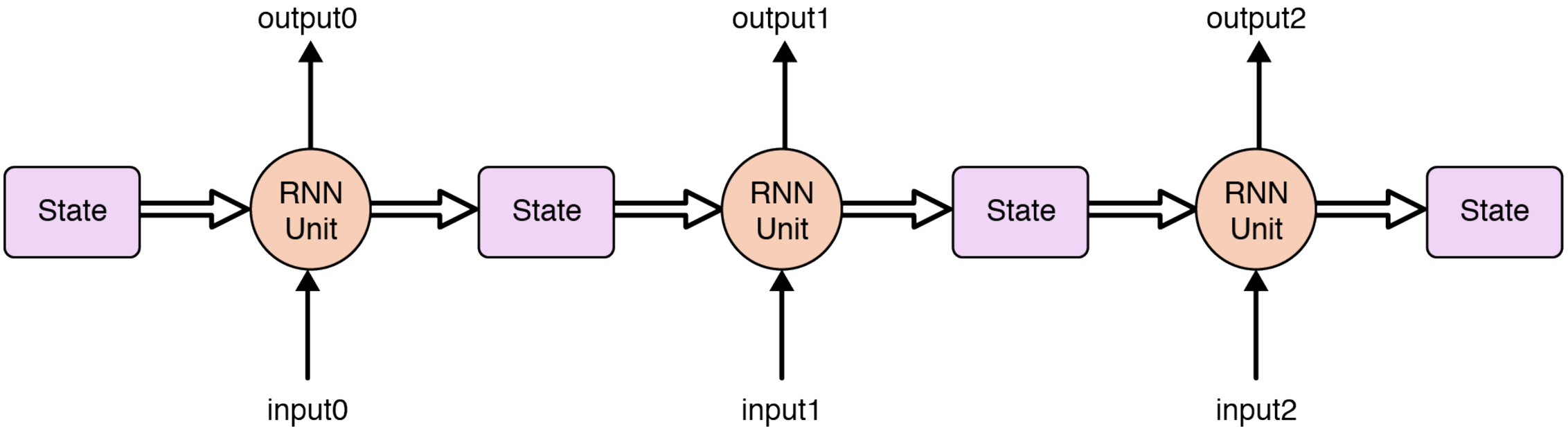
The idea of state





Consider a robot that repairs phones using parts from a part box. As time progresses, the state of the parts box changes. This is the idea behind the simple RNN.

Updating state

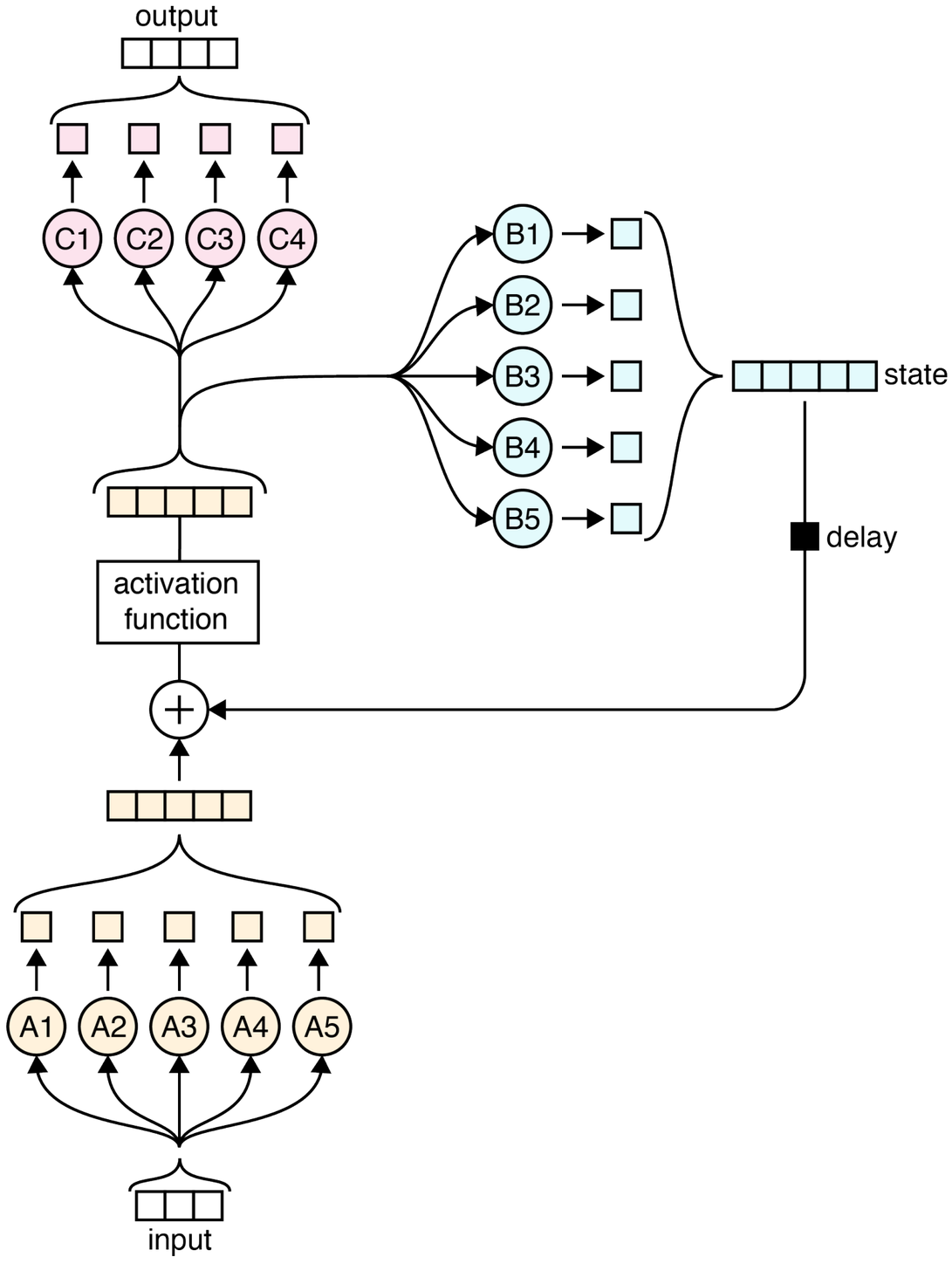


Structure of a SimpleRNN

Input size: 3, State size: 5, Output size: 4

"*An input of 3 values is processed simultaneously by five neurons lettered A to create a list of 5 values. This is added, element by element, to the state. This result then goes through five neurons lettered B to create a new state, which then goes into the delay step. The result of the addition also goes into the 4 neurons lettered C to produce an output.*"

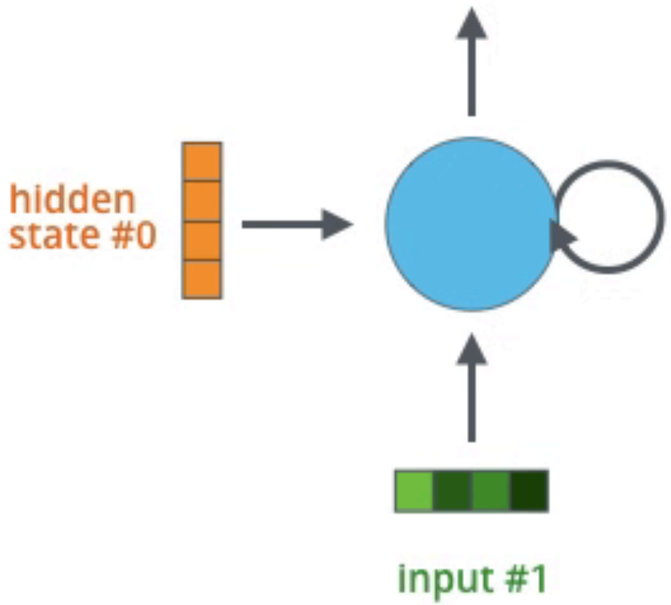
(Glassner, Andrew. Deep Learning, Vol. 2: From Basics to Practice)



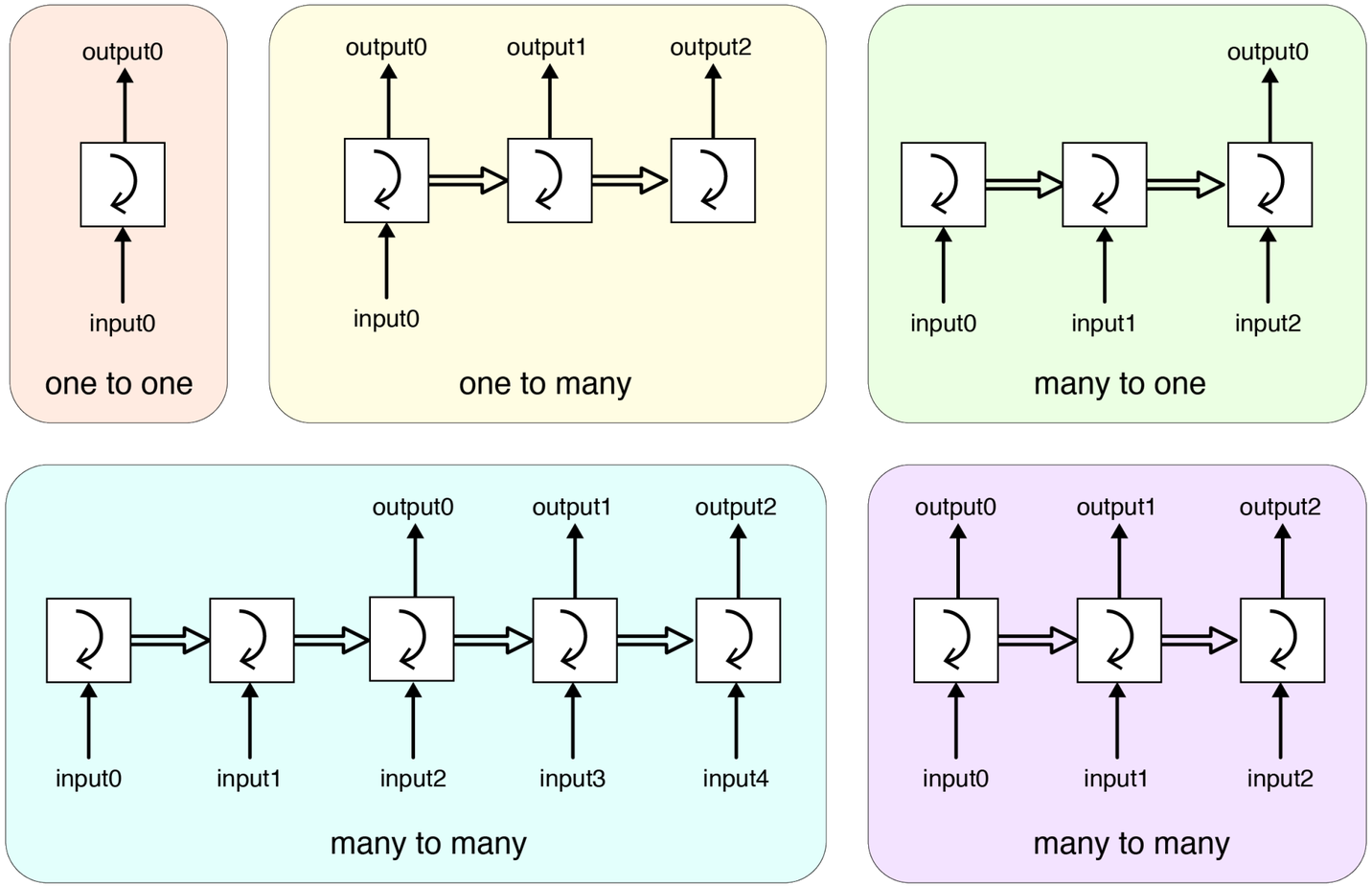
Recurrent Neural Network

Time step #1:

An RNN takes two input vectors:



Outputs from RNNs



- **one to many** structure takes in a single piece of data and produces a sequence.
- **many to one** structure reads in a sequence and gives us back a single value. e.g. sentiment analysis
- **many to many** structures are in some ways the most interesting. Delays are useful in translation. For example, English: "T*he black dog jumped over the cat*" to Italian "*Il cane nero salt*ò *sopra il gatto*"
  Other examples include video description and movement classification.

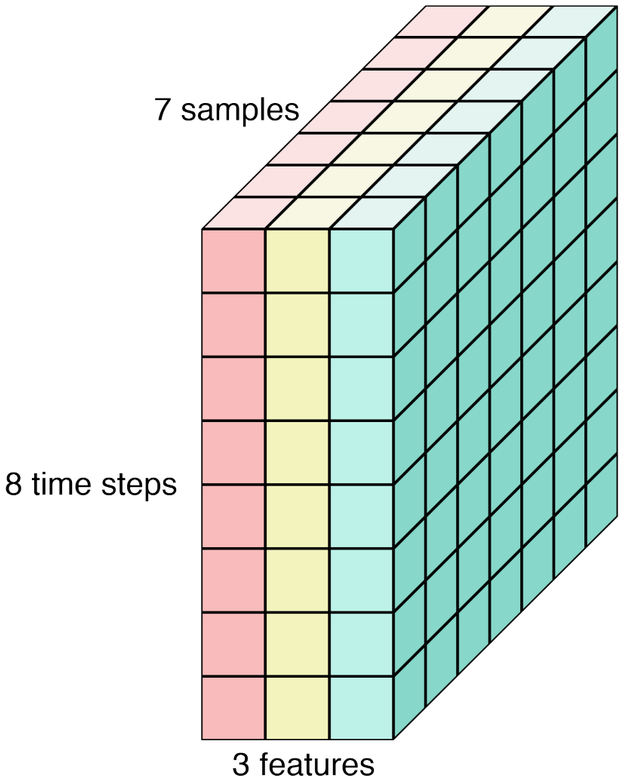
Keras example

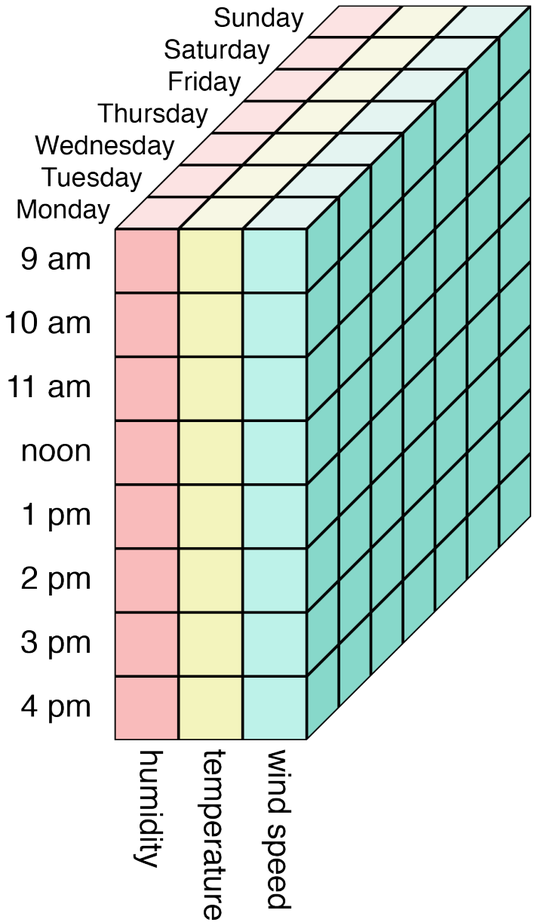
```
max_features = 10000
maxlen = 500
batch_size = 32

(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
```

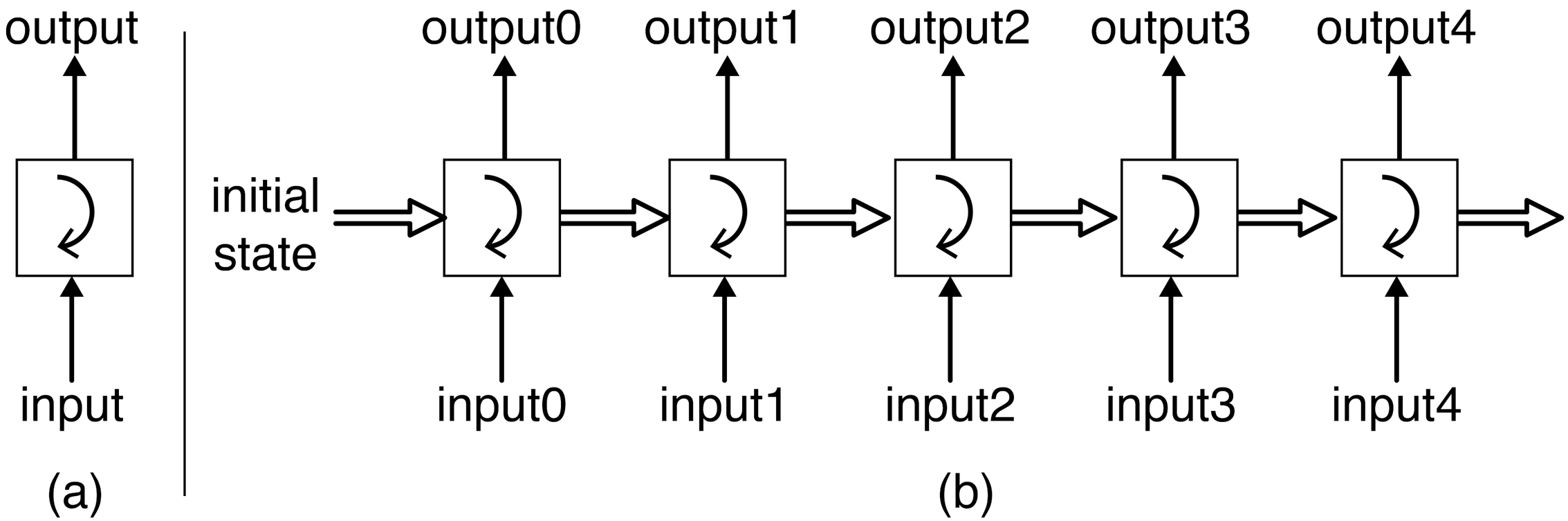
How is data fed to Keras?





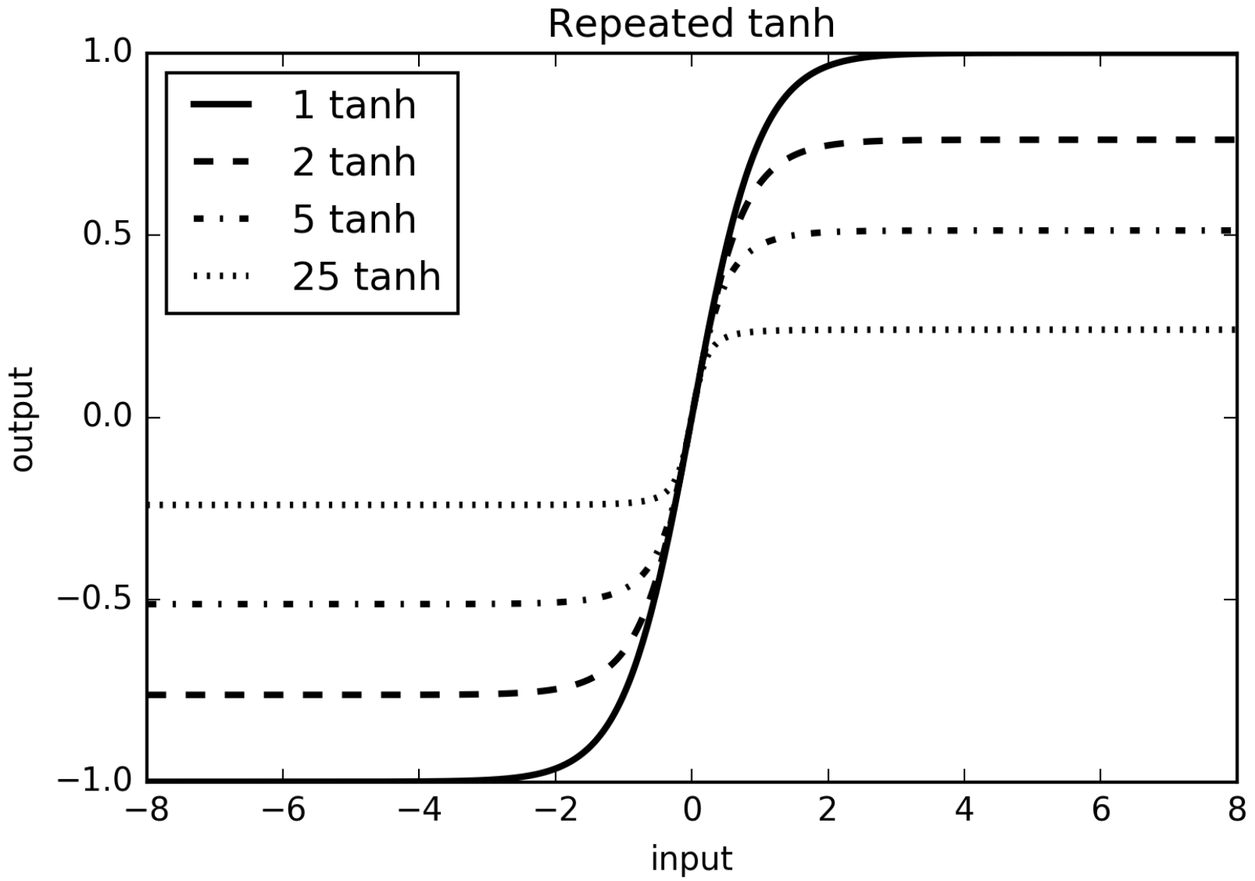
Training a RNN

Backprop in a RNN is called Back Propagation Through Time (BPTT). First unroll the network and backprop like a regular MLP

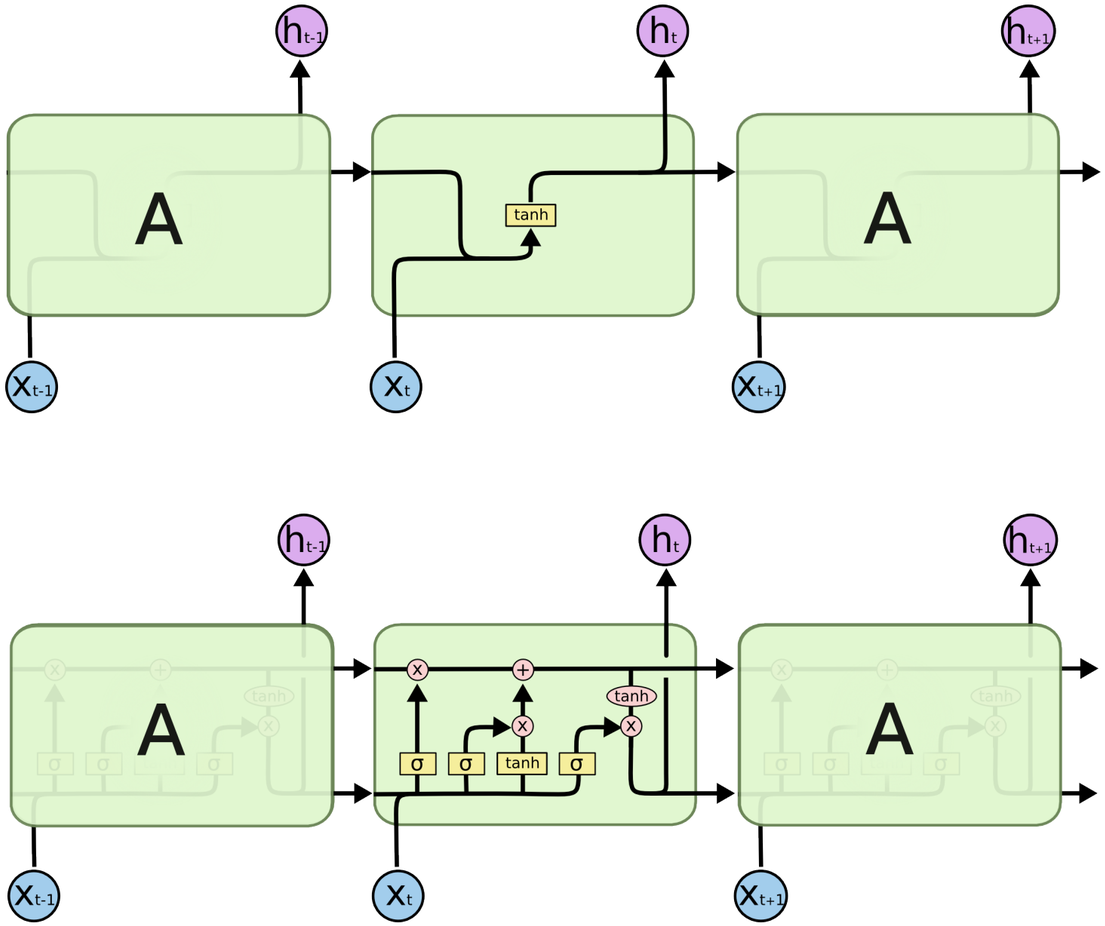


Problems

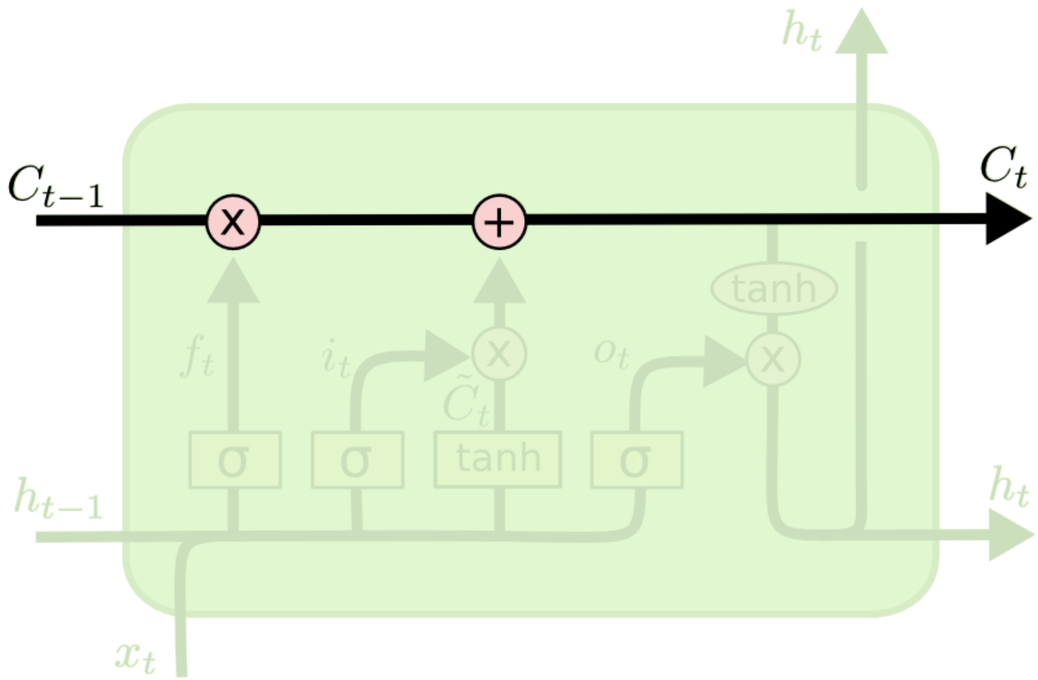
- repeated application of non-linearities lead to 0 gradients and no learning.
- The dynamic range of the network is also reduced.
- This problem happens in deep CNNs as well, and has many solutions
- The finiteness of the RNN memory means that not enough state may be propagating through. This is called that **Long Term Dependency Problem**.



From Simple RNNs to LSTMs

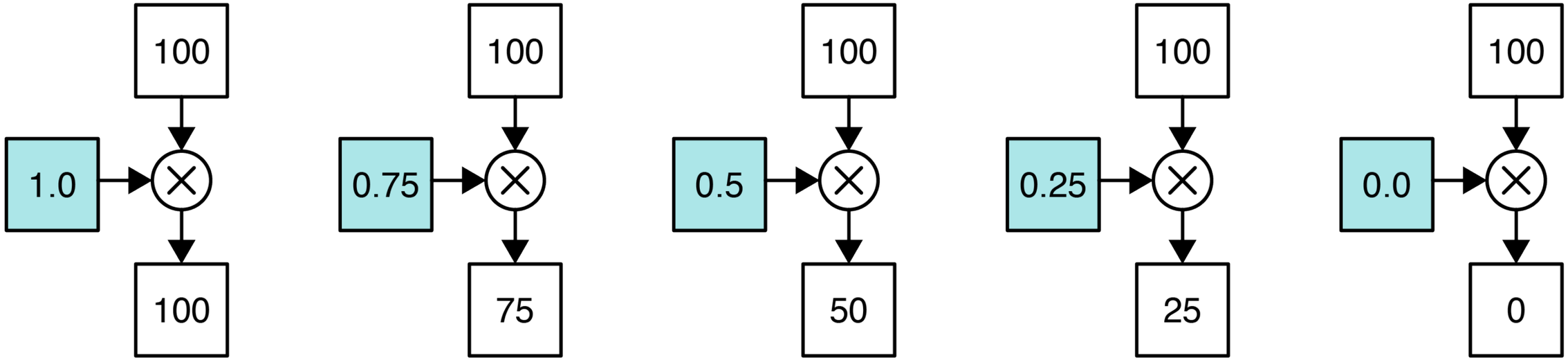


The main idea behind LSTM



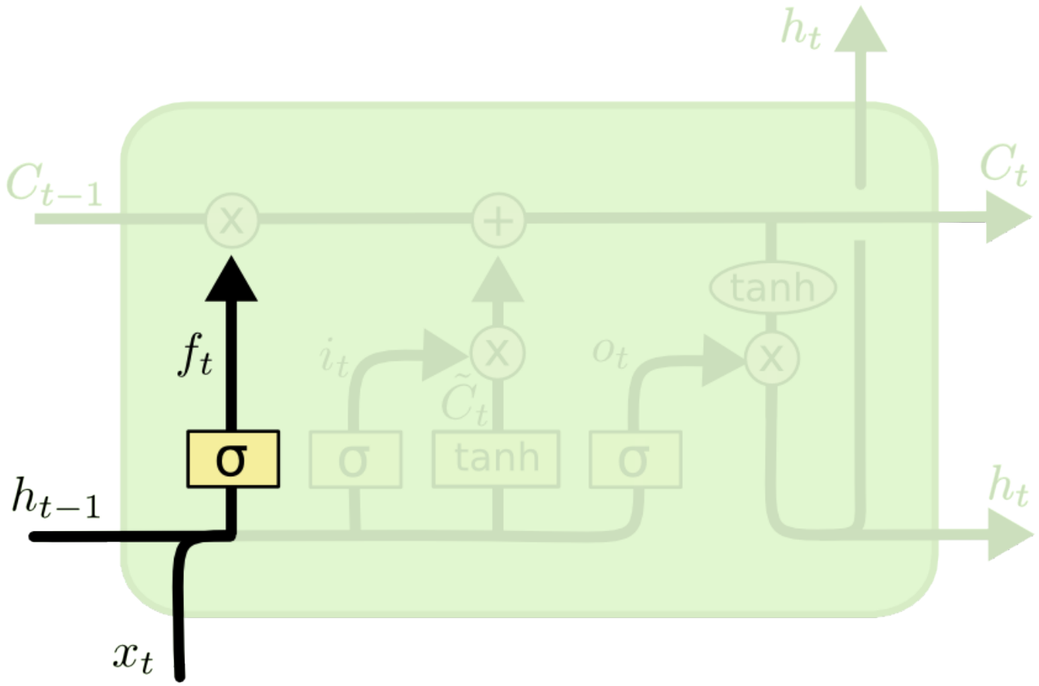
- takes short term memory and makes it longer
- The key is the cell state, the horizontal line running through the top of the diagram.
- memory runs straight down the entire chain
- a key innovation is the gate, which only allows partial information through

Gates



Values calculated through a regression and a sigmoid regulate "How much of a number" comes through..

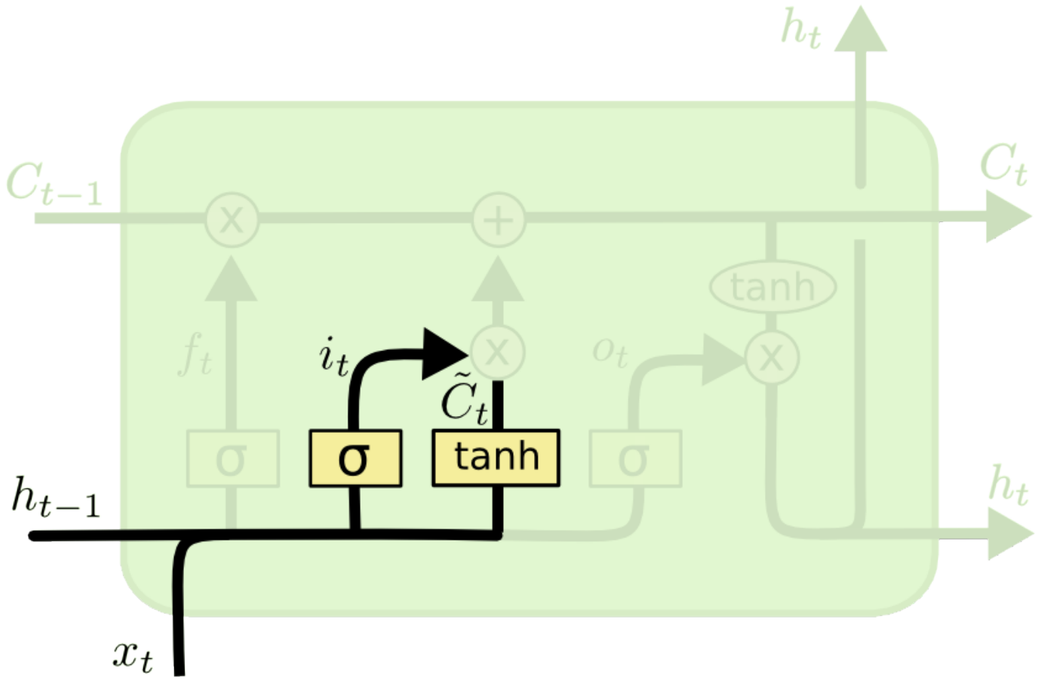
The forget gate



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

Remembering.

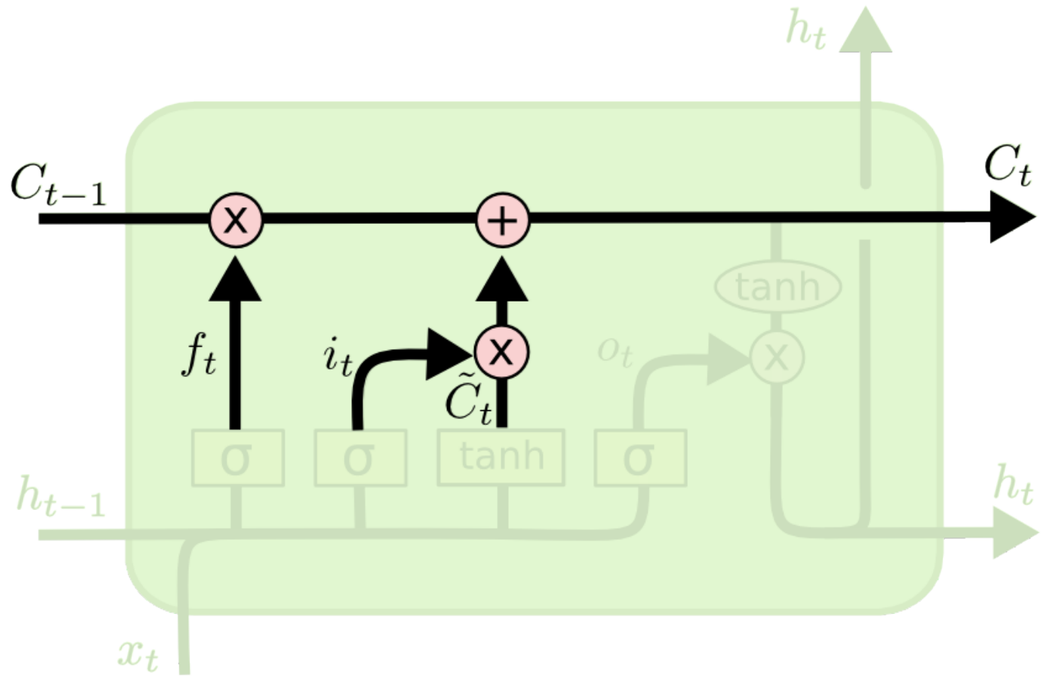
- a sigmoid layer called the “input gate layer” i_t that decides which values we’ll update.
- a tanh layer creates a vector of new candidate values, $\tilde{C}_t$ that could be added to the state.



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

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

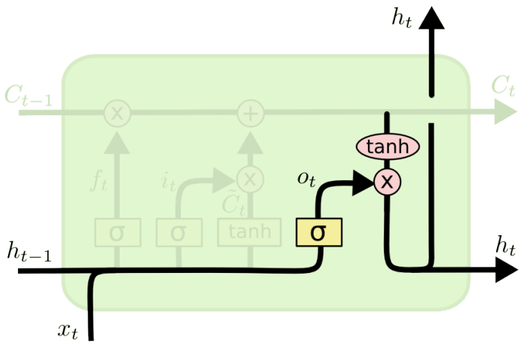
How is the state updated?



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

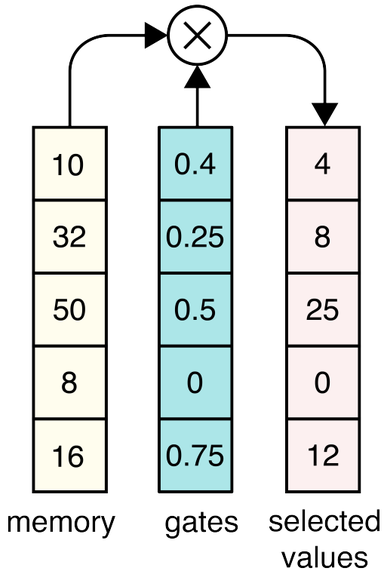
Multiply old state C_{t-1} by f_t to forget things, then add that to $\tilde{C}_t$, the new candidate value, multiplied by i_t

The output or selected values

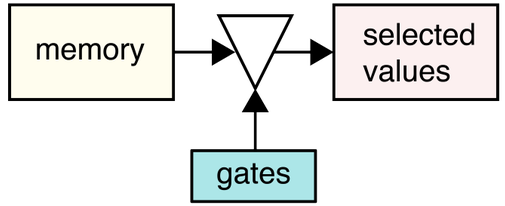


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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



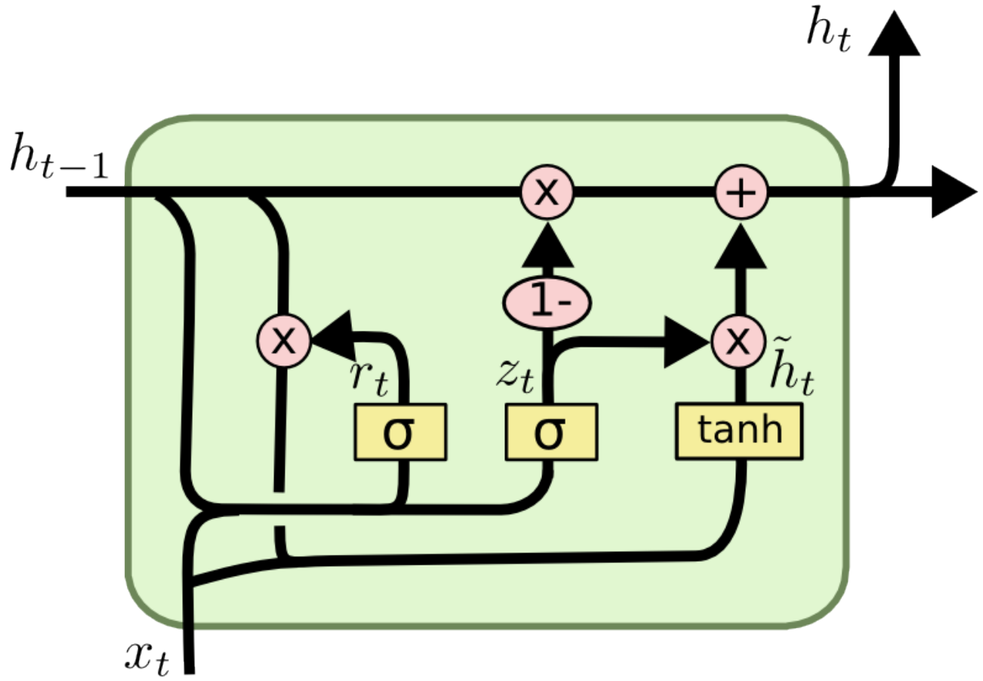
(a)



(b)

Output is a filtered version of the cell state. First, sigmoid to decide what parts of the cell state to output. Then multiply by tanh(cell_state).

GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Combines forget and input gates into single “update gate.” Merges the cell state and hidden state. Faster. Try Both.

Keras Example

```
max_features = 10000
maxlen = 500
batch_size = 32

(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM/GRU/CuDNNLSTM/CuDNNGRU(32))
model.add(Dense(1, activation='sigmoid'))
```