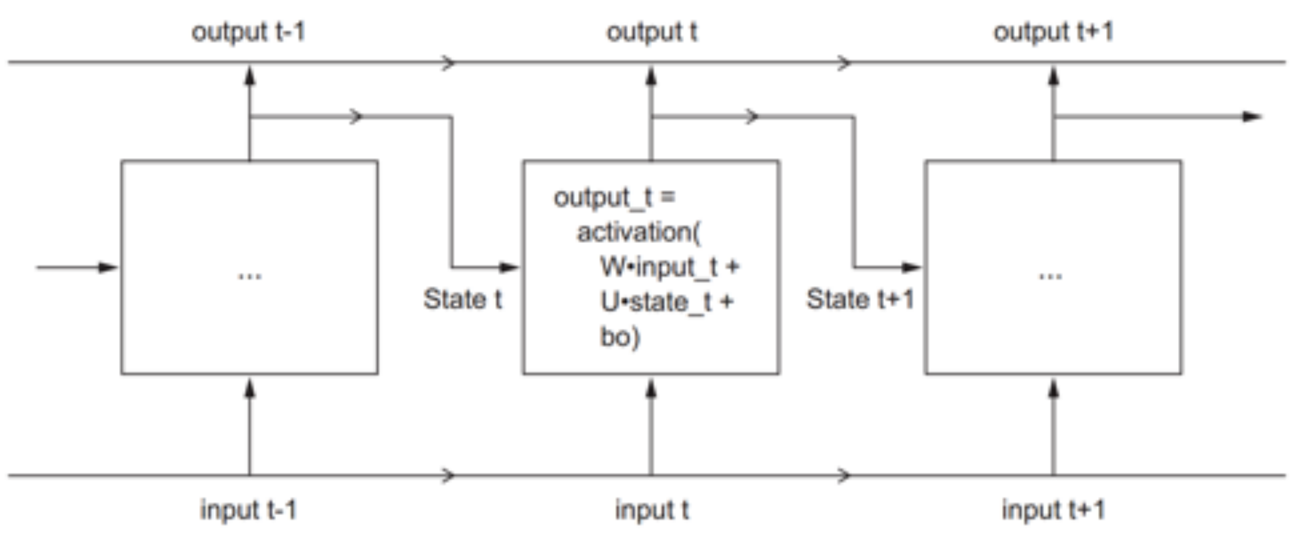
STAT430: Machine Learning for Financial Data

LarryHua.com/teaching
Spring 2019

Recurrent neural networks

Simple Recurrent neural networks (RNN)

- RNN processes sequences by iterating through the sequence elements and maintaining a state containing information it has seen
- recurrent networks vs feed-forward networks
- each output at t contains information from time step 1 to t, and is referred to as `hidden state`
- Simple RNN is not useful for handling long sequences




```
state_t <- 0
for (input_t in input_sequence) {
output_t <- activation(dot(W, input_t) + dot(U, state_t) + b)
state_t <- output_t
}
```

Simple RNN

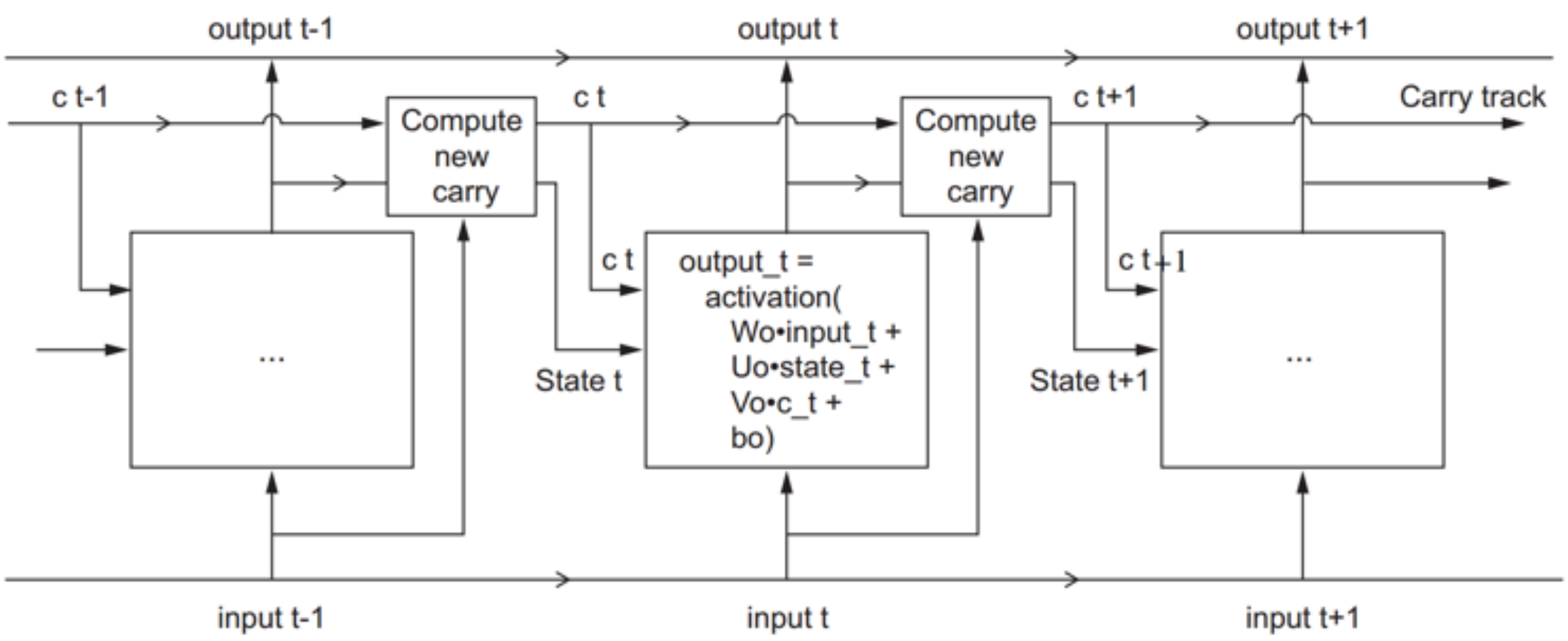
- `layer_simple_rnn()`: fully-connected RNN where the output is to be fed back as part of input
  - input: 3D tensor of the shape (batch_size, timesteps, input_features)
    - `input_shape`: excluding batch axis, required when using this layer as the first layer
  - output: two options controlled by `return_sequences = T/F`
    - T: 3D tensor of the shape (batch_size, timesteps, output_features)
    - F: 2D tensor of the shape (batch_size, output_features)
    - parameter `units` specifies the dimensionality of the output
      - eg, `layer_simple_rnn(unit = 16)`
- with several recurrent layers one after the other, the intermediate layers should return full sequences
- Try R

Vanishing gradient problem

- for feed-forward networks with many layers or simple RNN for long sequences, gradients become vanishingly small preventing the weight from updating
  - using backpropagation, gradients for the weights of earlier layers involves multiplication of many small gradients, thus become too small
- solutions: allows past information to be re-injected at a later time, thus fighting the vanishing gradient problem and accounting for long term dependence
  - Long short-term memory (LSTM)
  - Gated recurrent unit (GRU)

RNN with Long Short Term Memory (LSTM)

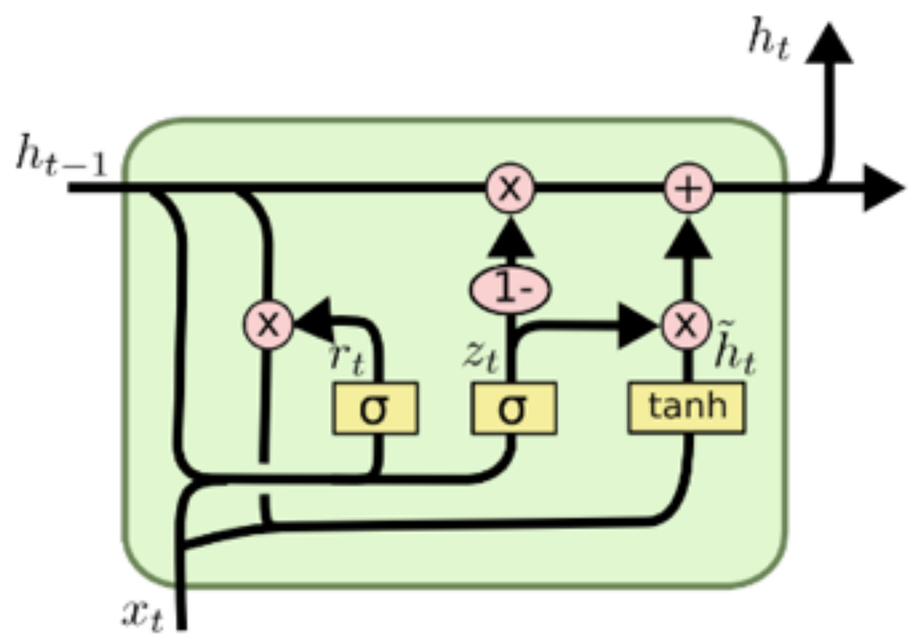
- use an additional data flow (C_t: carry / cell state) to carry information across time steps
- **`sigmoid()`** (σ) **with range (0,1) plays the role of gates**
- **`tanh()`** **with range (-1, 1) plays the role of generating new output**



- [Understanding LSTM Networks]()

RNN with Gated Recurrent Unit (GRU)

- a variant of LSTM that has less parameters and more computational efficiency
- the forget and input gates from LSTM unit are merged, and the carry data flow and hidden state are merged
- z_t: update gate vector
- r_t: reset gate vector



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

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

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

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

LSTM / GRU in Keras

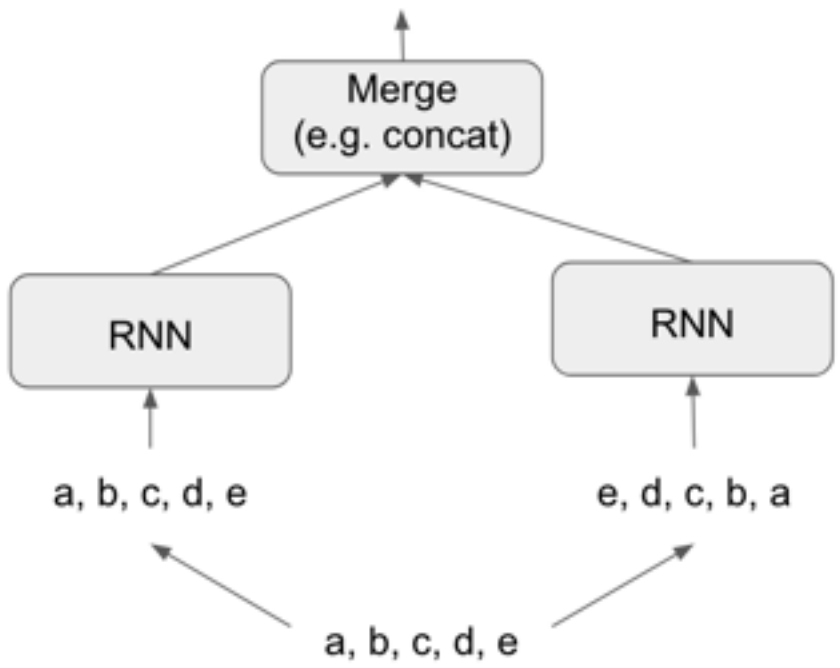
- `layer_lstm()`
  - parameter `units` specifies the dimensionality of the output
    - eg, `layer_lstm(units = 16)`
- `layer_cudnn_lstm()`
  - **much faster** LSTM with Nvidia GPU cuDNN + Tensorflow backend
- for GRU: `layer_gru()` and `layer_cudnn_gru()`
- Try R

Advanced use of RNN

- Recurrent dropout
  - specific, built-in way to use dropout to fight overfitting in recurrent layers
- Stacking recurrent layers
  - to increase the representational power of the network (at the cost of higher computational loads)
- Try R

Advanced use of RNN

- Bidirectional recurrent layers
  - which presents the same information to a recurrent network in different ways, increasing accuracy and mitigating forgetting issues
  - `bidirectional()`: takes a recurrent layer instance as an argument, and creates a second, separate instance of this recurrent layer
    - one for chronological order and the other for reversed order
  - a bidirectional layer has twice more parameters than a chronological LSTM



Going even further

- adjust the number of units in each recurrent layer in the stacked setup. The current choices are largely arbitrary and thus probably suboptimal
- adjust the learning rate used by the `RMSprop` optimizer
- try using `layer_lstm()` instead of `layer_gru()`
- try using a bigger densely connected regressor on top of the recurrent layers: that is, a bigger dense layer or even a stack of dense layers
- don't forget to eventually run the best-performing models on test set
- deep learning is more an art than a science, and one has to evaluate different strategies empirically

Wrapping up

- establish common-sense baselines for your metric of choice
- try simple models before expensive ones, to justify the additional expense
- for data where temporal ordering matters, recurrent networks are a great fit and easily outperform models that first flatten the temporal data
- for recurrent networks, use a time-constant dropout mask and recurrent dropout mask
- stacked RNNs provide more representational power than a single RNN layer, but much more expensive and thus not always worth it
- bidirectional RNNs are useful on NLP, but aren’t strong performers on sequence data where the recent past is much more informative than the beginning of the sequence
- [Back to Course Scheduler]