Lecture Notes for

Machine Learning in Python

Professor Eric Larson

Lecture: RNN Demo

Lecture Agenda

- Logistics
  - RNNs due **During Finals Time**
- Recurrent Networks
  - *Overview*
  - *Problem Types*
  - *Embeddings*
  - *Types of RNNs*
  - **Demo A**
  - **CNNs and RNNs**
  - **Demo B**
  - **Ethics Case Study**
  - **Course Retrospective**

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn

Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,

Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn

Detailed mathematics for simple optimization

intuition for advanced optimization

Neural Networks and Back Prop.

Numpy

Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

Recurrent Networks

Keras, Tensorflow

Intuition, Detailed implement.

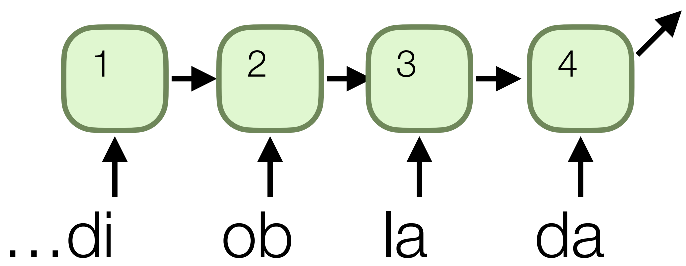
Ethics in Language Models

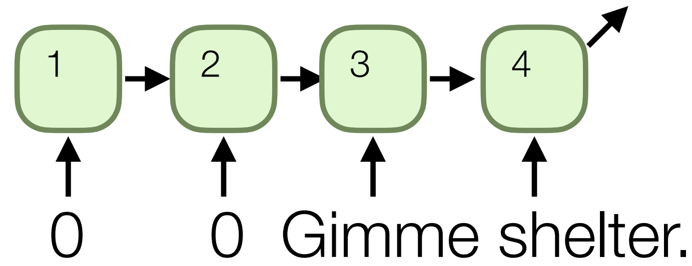
ConceptNet

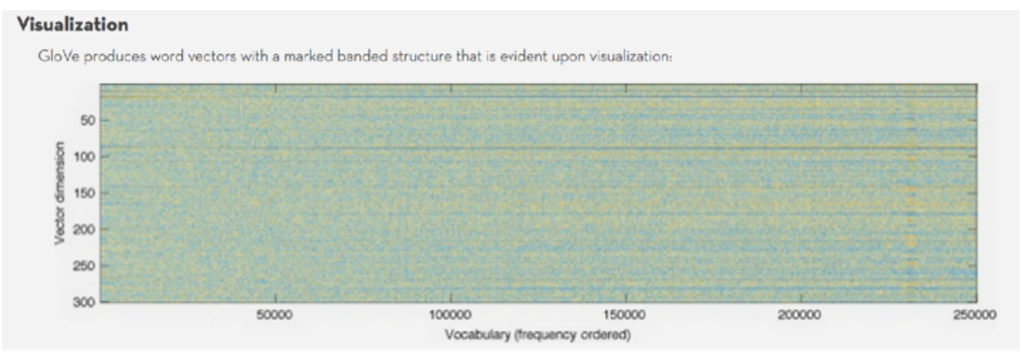
Case studies

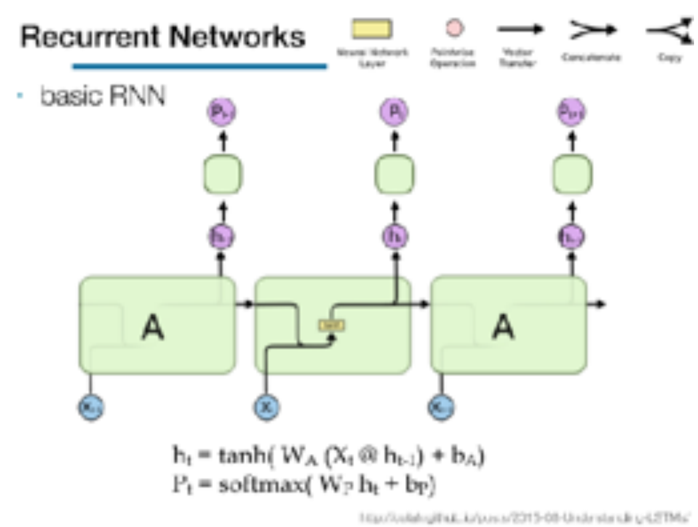
Last Time

- padding/clipping



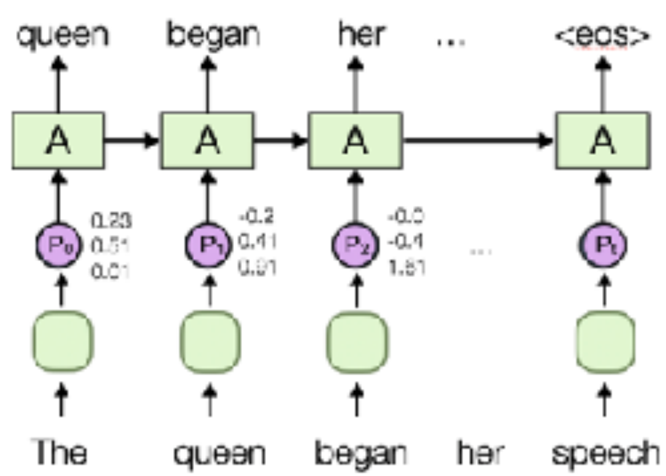






Self Test

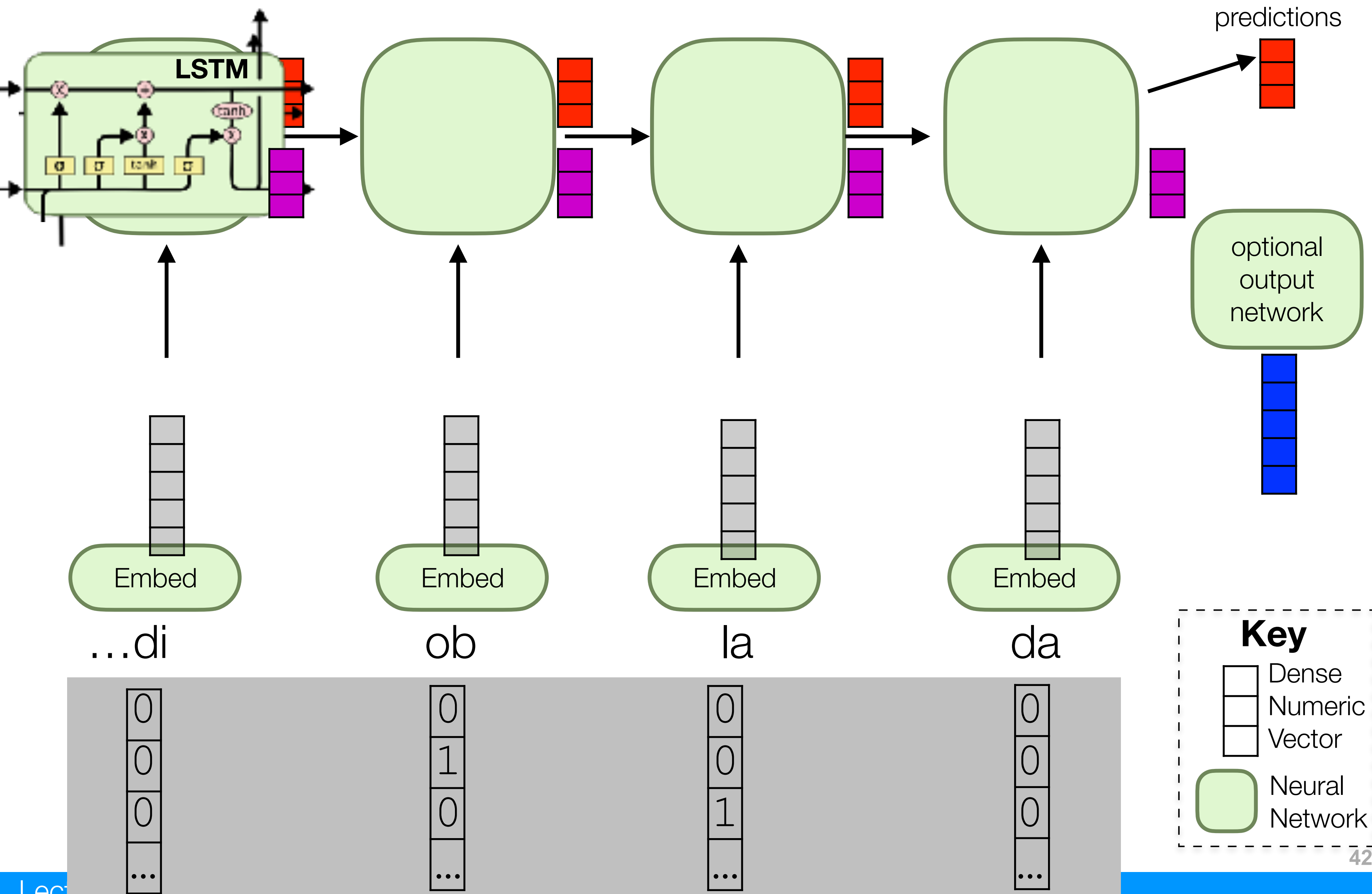
- T/F: In Recurrent Neural Networks that are “rolled out”, each RNN cell can be run in parallel.
  - A. **True**, state vectors can be added later
  - B. **True**, but parallelization must use forward backward (like Viterbi)
  - C. **False**, state vectors must be found sequentially
  - D. **False**, input changes due to sequential nature of X_t



Commonly Used RNN Nodes

I like this version better.

General recurrent flow (many to one)



Recurrent Networks: GRUs

- gated recurrent units

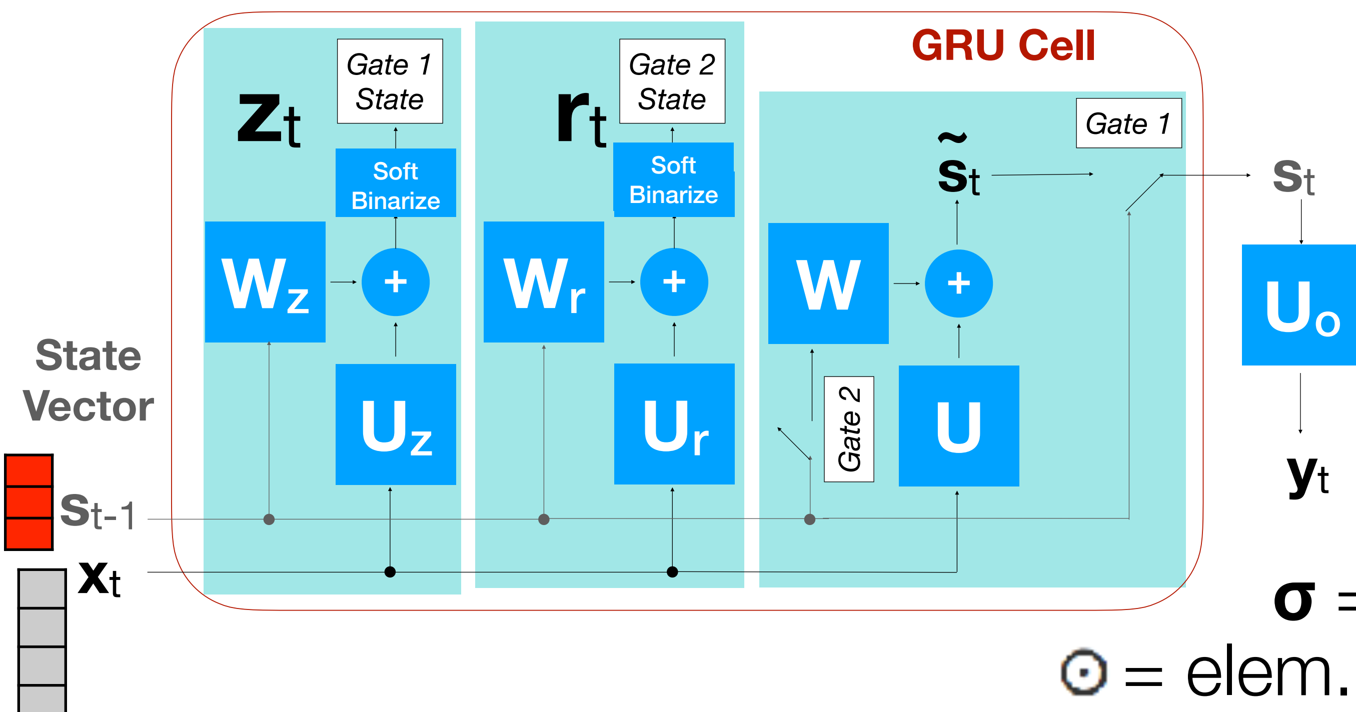
Selectivity controls, gates (**0 or 1**)

$$r_t = \sigma(W_r \underbrace{\boxed{s_{t-1}}}_{\text{past state}} + U_r \underbrace{\boxed{x_t}}_{\text{current input}} + b_r)$$

$$z_t = \sigma(W_z \boxed{s_{t-1}} + U_z \boxed{x_t} + b_z)$$

$$\tilde{s}_t = \phi(\overbrace{\boxed{W(r_t \odot s_{t-1})}}^{\text{selectively remember}} + \overbrace{\boxed{Ux_t}}^{\text{with influence}} + b)$$

$$s_t = \underbrace{\boxed{z_t \odot s_{t-1}}}_{\text{remember only past}} + \underbrace{\boxed{(1 - z_t) \odot \tilde{s}_t}}_{\text{OR remember with input}}$$



σ = sigmoid

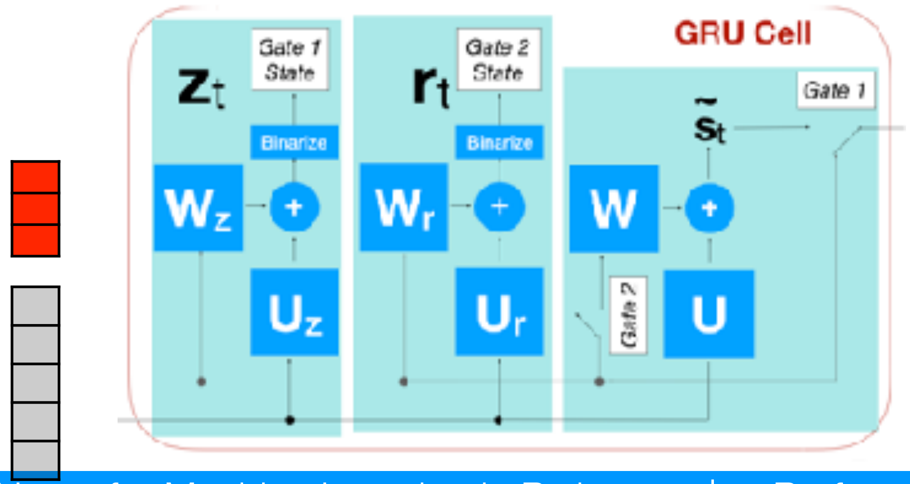
⊙ = elem. multiplication

Self Test

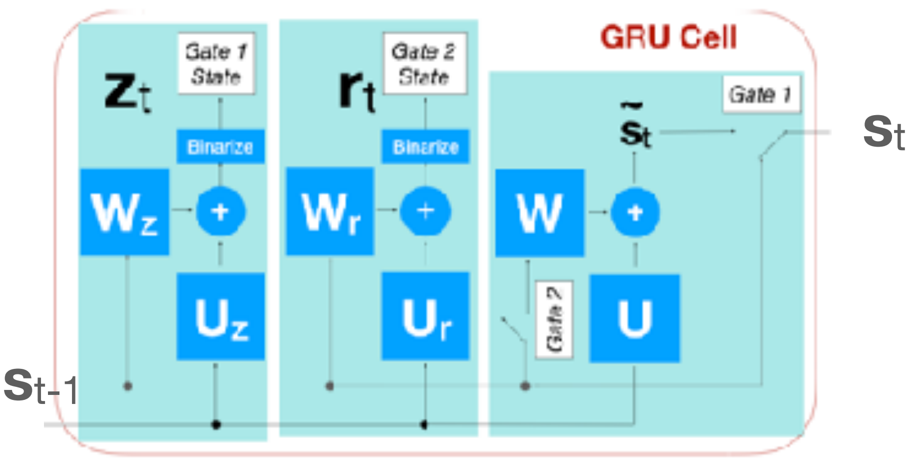
- What element of the GRU helps with vanishing and exploding gradients?
- A. derivative of σ
- B. no activation function
- C. derivative of ϕ
- D. ϕ

$$r_t = \sigma(W_r s_{t-1} + U_r x_t + b_r)$$
$$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$$

$$\tilde{s}_t = \phi(W(r_t \odot s_{t-1}) + U x_t + b)$$
$$s_t = z_t \odot s_{t-1} + (1 - z_t) \odot \tilde{s}_t$$



Derivative of GRU



$$r_t = \sigma(W_r s_{t-1} + U_r x_t + b_r)$$
$$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$$

$$\tilde{s}_t = \phi(W(r_t \odot s_{t-1}) + U x_t + b)$$
$$s_t = \boxed{z_t \odot s_{t-1}} + \boxed{(1 - z_t) \odot \tilde{s}_t}$$

Product Rule

Product Rule

To back propagate, we need sensitivity of state vector, w.r.t previous state

$$\partial s_t \,/\, \partial s_{t-1} = (\partial z_t \text{ x } s_{t-1}) + (\partial s_{t-1} \text{ x } z_t) + \partial \tilde{s}_t - (\partial z_t \text{ x } \tilde{s}_t) - (\partial \tilde{s}_t \text{ x } z_t)$$

likely vanish

could vanish, depending on ϕ

likely vanish

could vanish, depending on ϕ

hard to vanish unless $z_t = 0$

Recurrent Networks: Gen 1 LSTM

- LSTM prototype

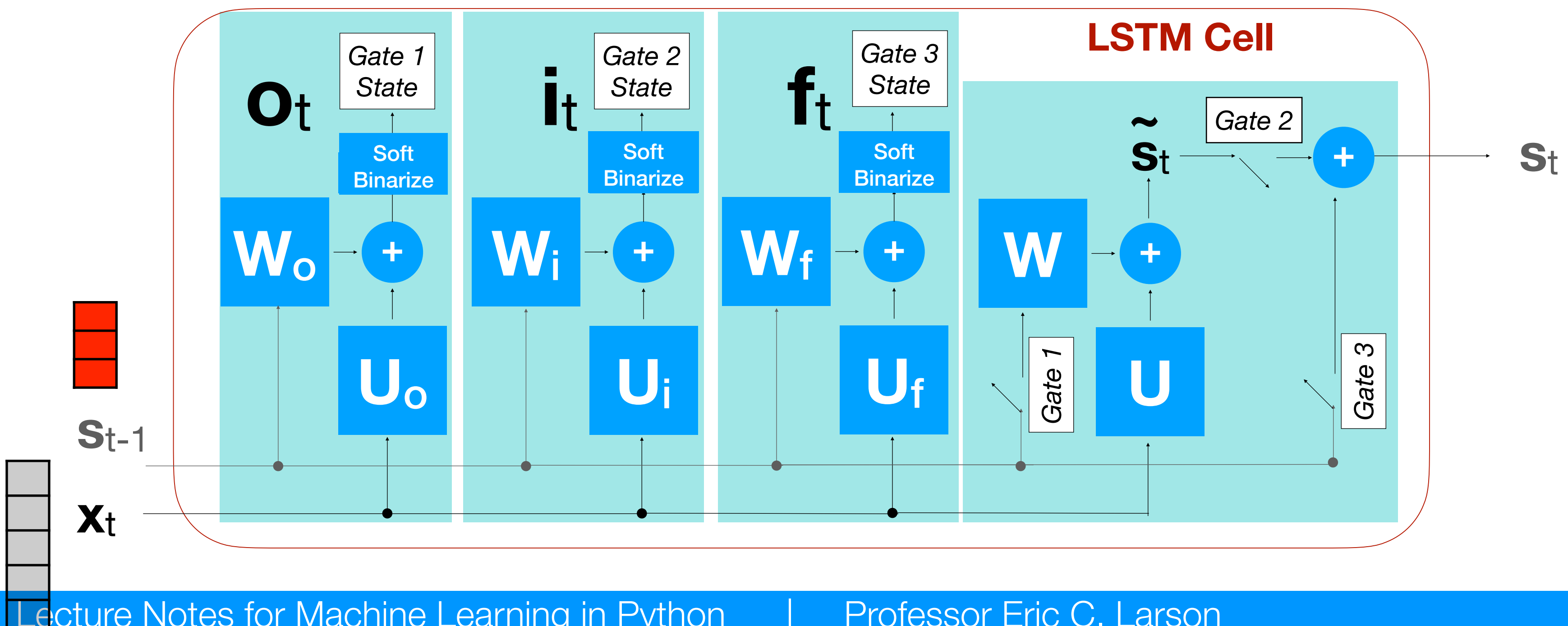
Selectivity controls (**gates, 0 or 1**)

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$
$$f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$$

selectively remember past — with influence

$$\tilde{s}_t = \phi(W\boxed{(o_t \odot s_{t-1})} + \boxed{Ux_t + b})$$
$$s_t = \boxed{f_t \odot s_{t-1}} + \boxed{i_t \odot \tilde{s}_t}$$

selectively remember past — with past weighted influence



Recurrent Networks: Gen 2 LSTM

- LSTM in TensorFlow

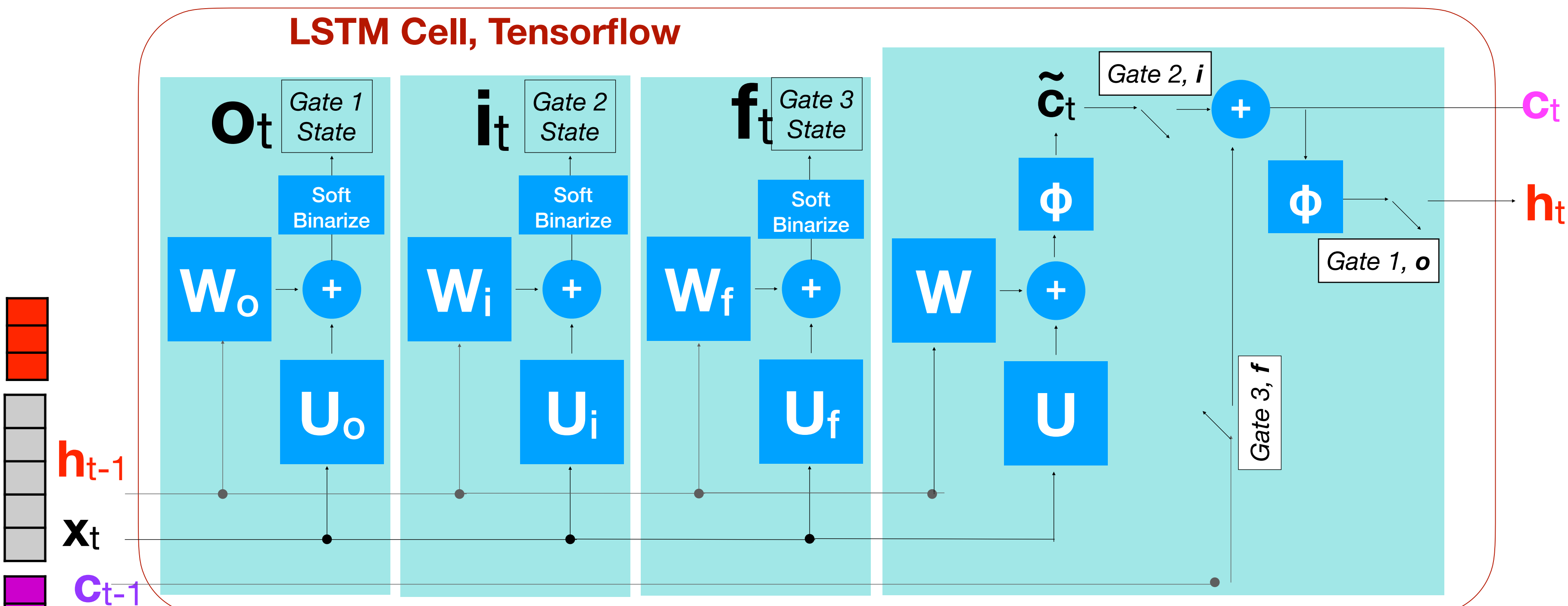
Selectivity controls (**gates, 0 or 1**)

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

explicit remembering state

$$\tilde{c}_t = \phi(W h_{t-1} + U x_t + b)$$
$$c_t = \underbrace{\boxed{f_t \odot c_{t-1}}}_{\text{remember previous state}} + \underbrace{\boxed{i_t \odot \tilde{c}_t}}_{\text{update with output, } h_t}$$
$$h_t = \boxed{o_t \odot \phi(c_t)}$$

get next ht for selecting gates



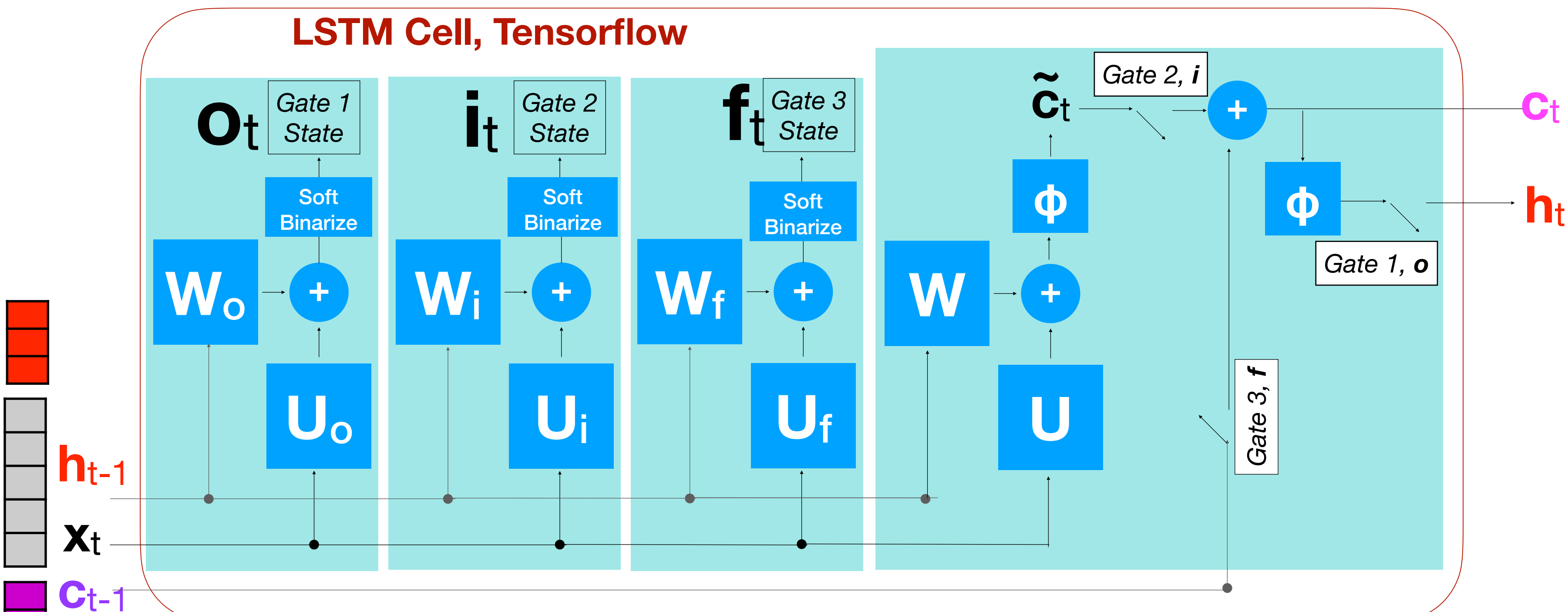
LSTM Dropout

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$
$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$
$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

Recurrent Dropout

Input Dropout

The days of training **without** using **dropout** are **over**.



What to choose?

- There is no hard and fast rule
  - try both
  - basic LSTM has had great success
  - GRU also sometimes is easier to train
  - you will see many variations
    - peephole LSTM
    - hierarchical LSTM
    - and many more…

Final Demo - Part A

Many to one:
- Simple RNNs
- GRUs
- LSTMs

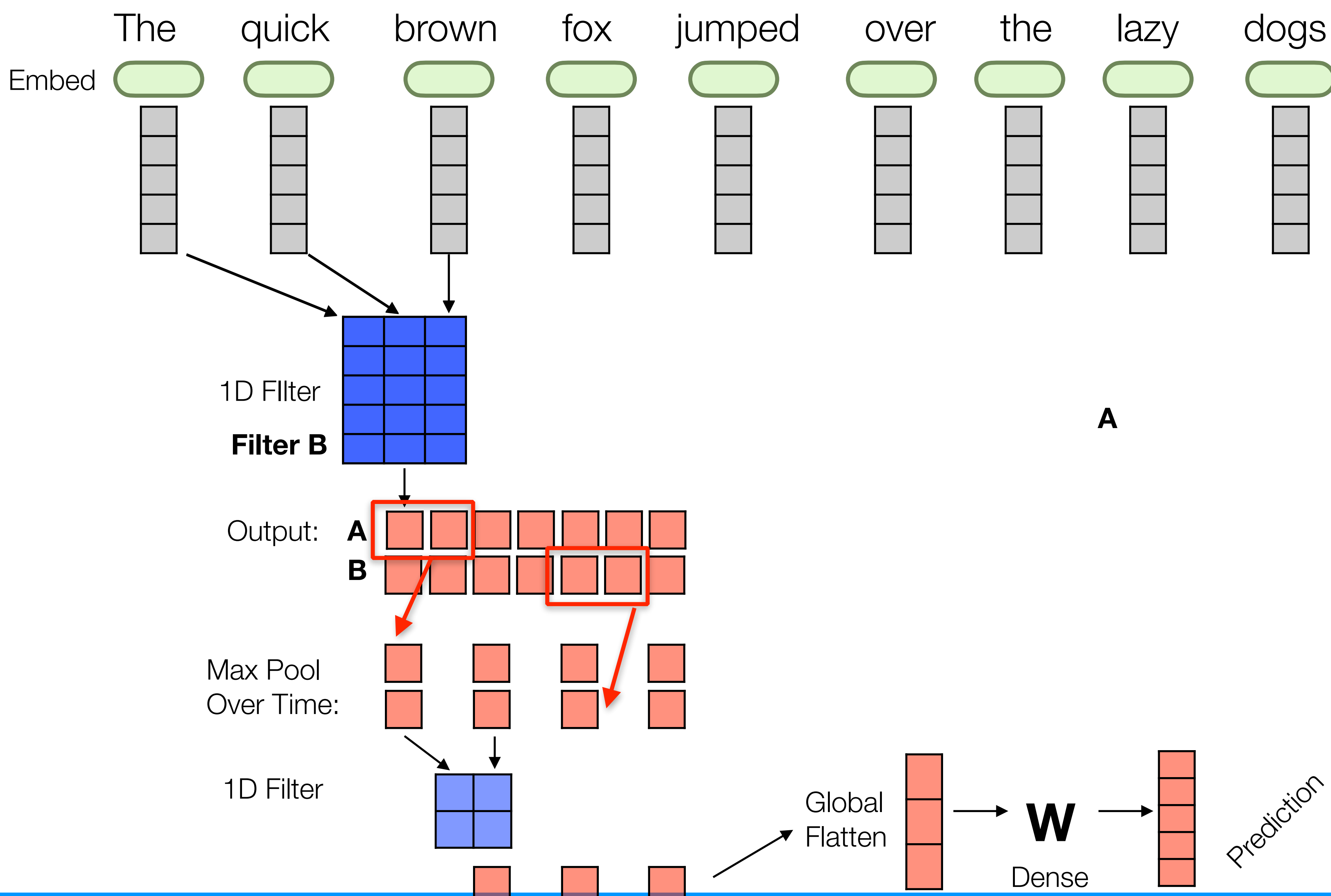
More examples:
https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/skflow
http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html
http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Seq2Seq:
https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/examples/skflow/neural_translation_word.py

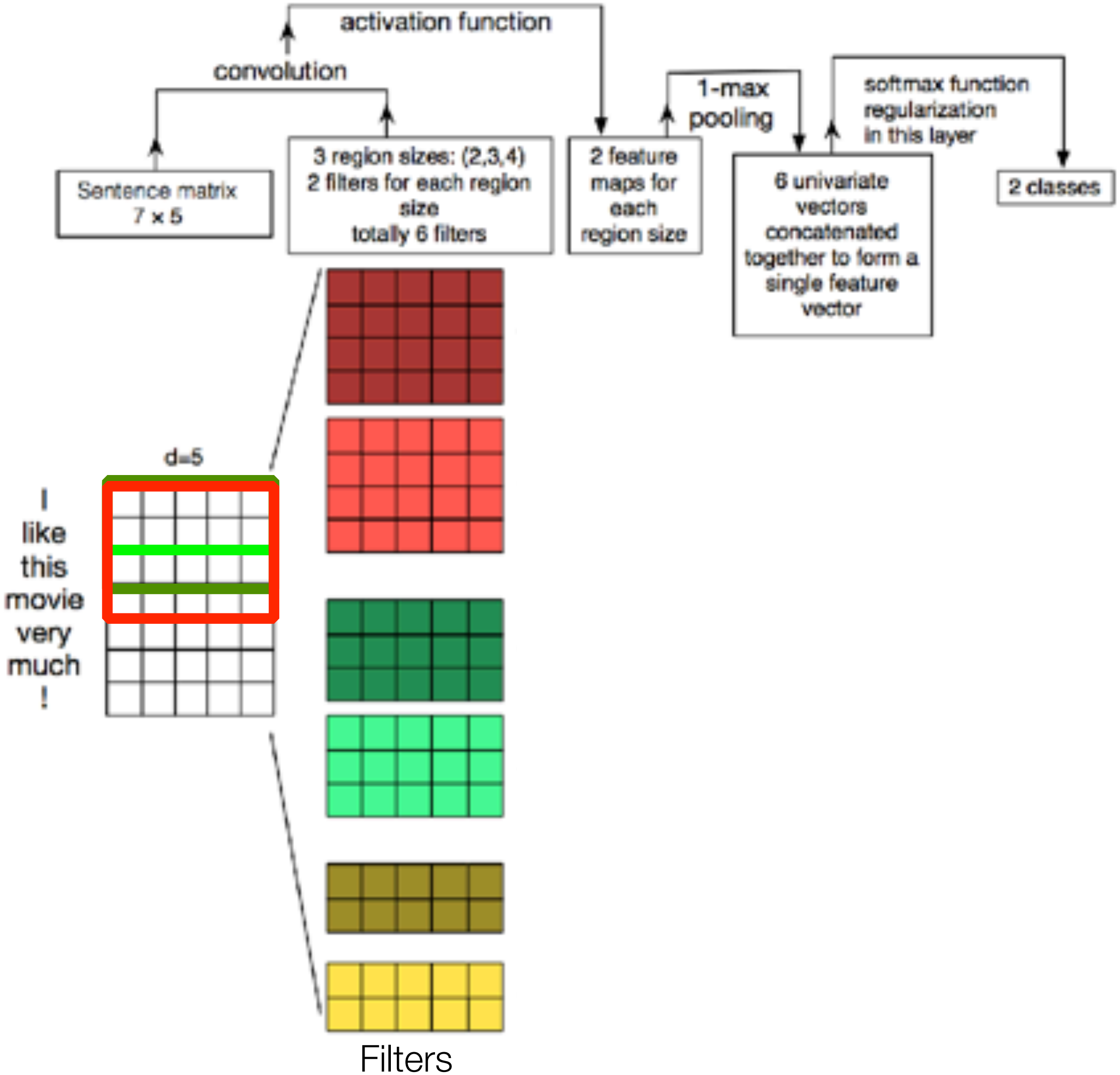
CNNs for Sequences

CNNs and RNNs

is an RNN similar to a CNN?



CNNs with Multiple Region Sizes



http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Demo - Part B

Back to the CNN

More examples:

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

Seq2Seq:

https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/examples/skflow/neural_translation_word.py