Lecture Notes for Machine Learning in Python

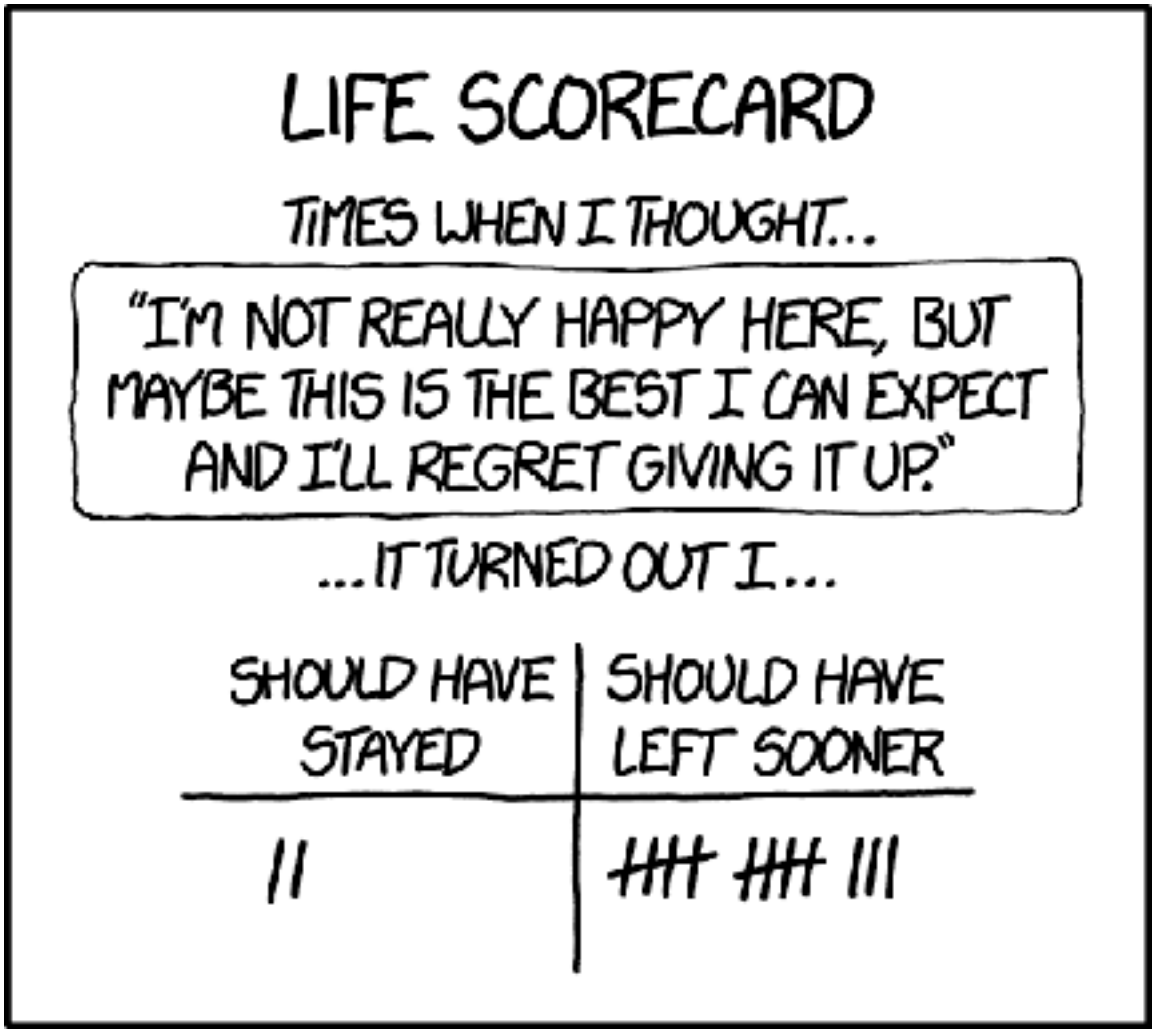
Professor Eric Larson

Recurrent Neural Networks

Lecture Agenda

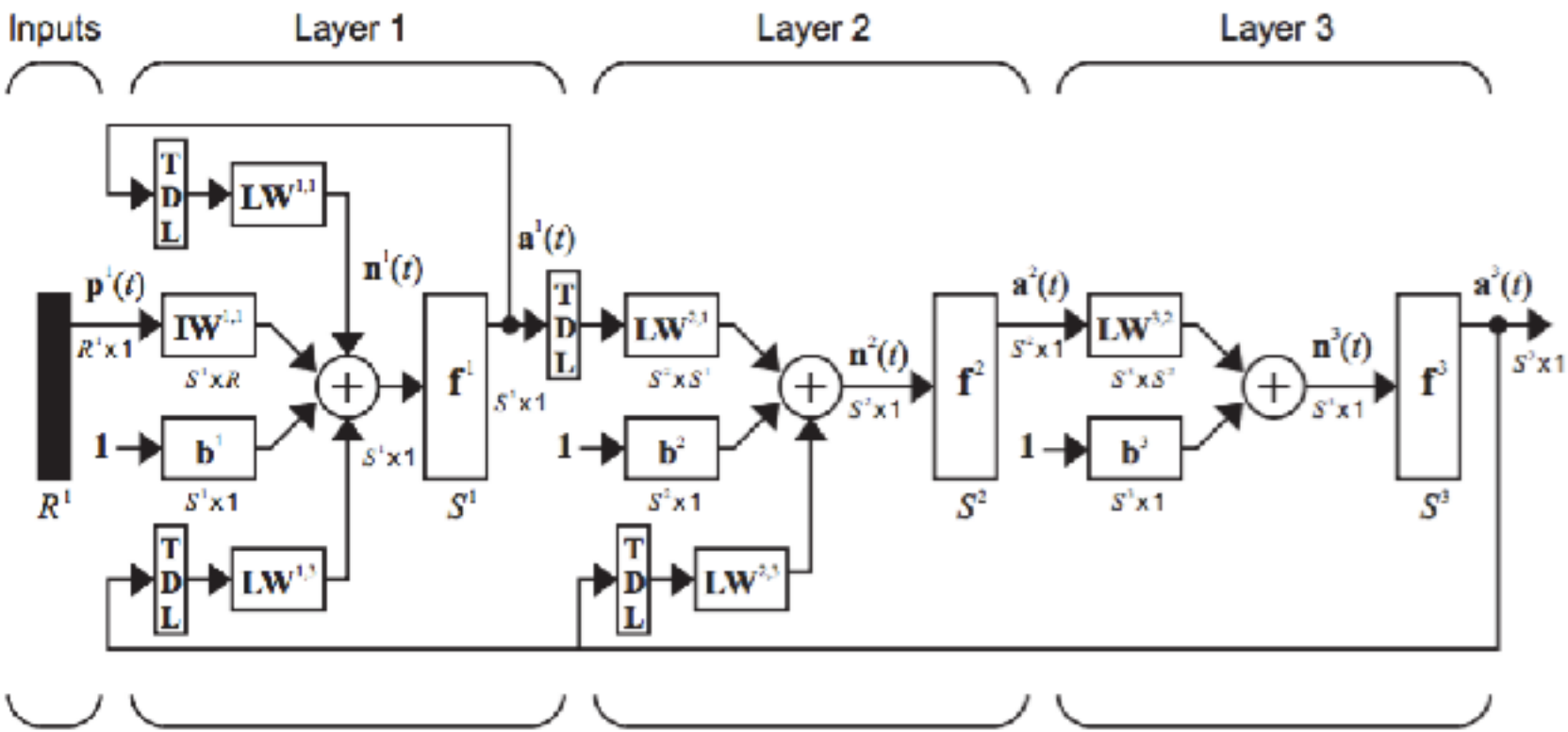
- Logistics
  - CNNs due this week (Friday)
  - RNNs due the following week (Friday)
- Recurrent Networks (two lecture agenda)
  - Overview
  - Problem Types
  - Embeddings
  - Types of RNNs
  - Demo A
  - CNNs and RNNs
  - Demo B
  - course retrospective

History of Recurrent Neural Networks



History of Recurrent Networks

- Dynamic Networks
  - can use current and previous inputs, in time
  - still popular, but ultimately extremely hard to train
  - **been around for decades**

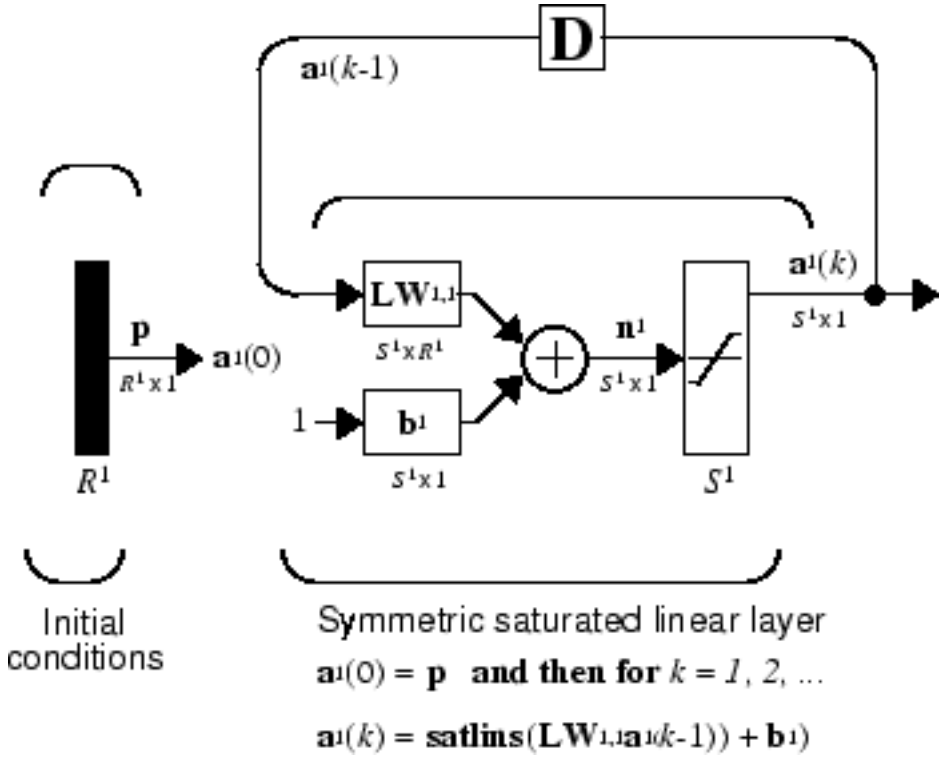


Neural Network Design, Hagan, Demuth, Beale, and De Jesus

History of Recurrent Networks

- Hopfield Network, 1982

John Hopfield, Princeton



Neural Network Design, Hagan, Demuth, Beale, and De Jesus

History of Recurrent Networks

- Elman/Jordan Networks, ~1988

Jeffrey Elman, UCSD

Michael Jordan, Berkeley

Elman network[10]

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Jordan network[11]

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

Variables and functions

- x_t: input vector
- h_t: hidden layer vector
- y_t: output vector
- W, U and b: parameter matrices and vector
- σ_h and σ_y: Activation functions

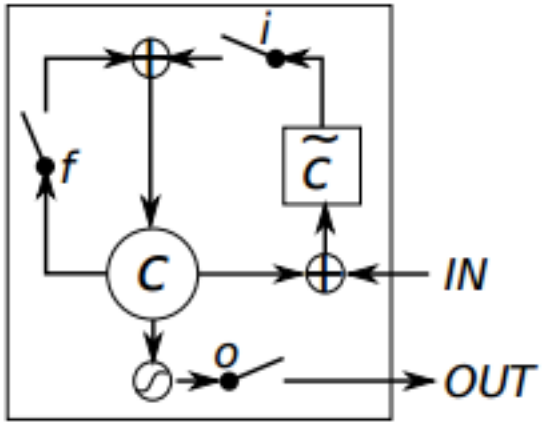
Wikipedia

History of Recurrent Networks

- Long Short Term Memory, ~1997

Sepp Hochreiter, Many Universities

Jürgen Schmidhuber, Switzerland



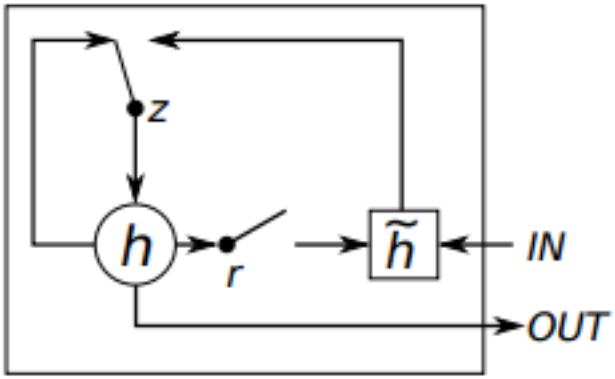
More on these later

History of Recurrent Networks

- Gated Recurrent Units, ~2014

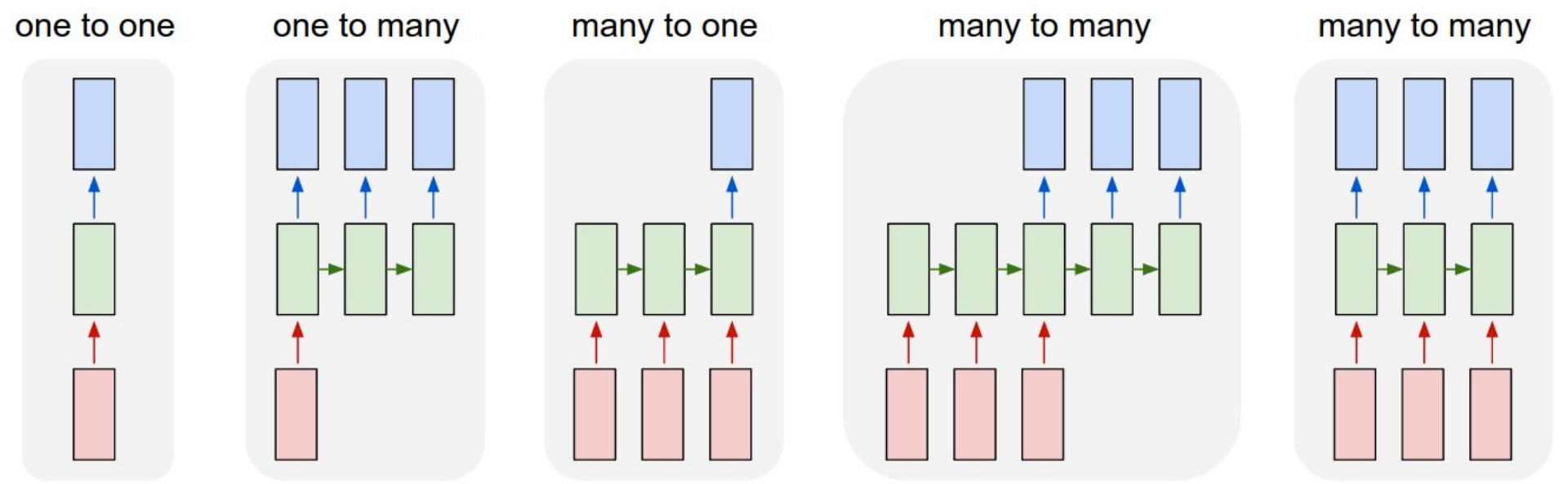
Yoshua Bengio

Kyunghyun Cho, Professor at NYU

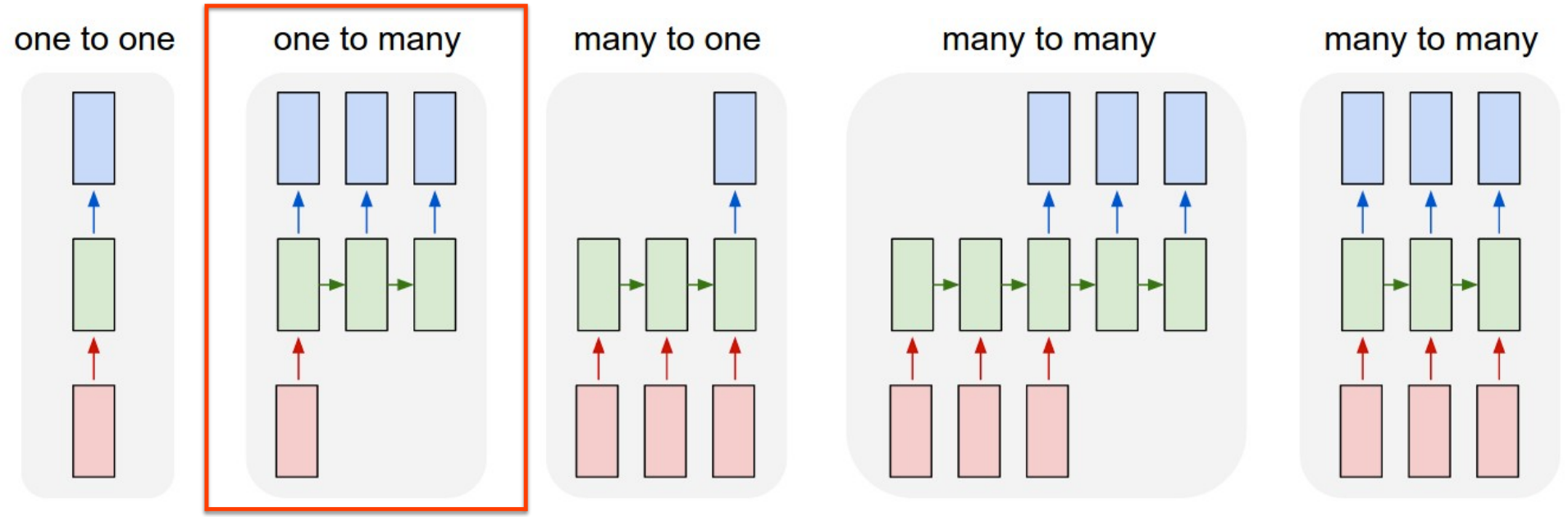


More on these later

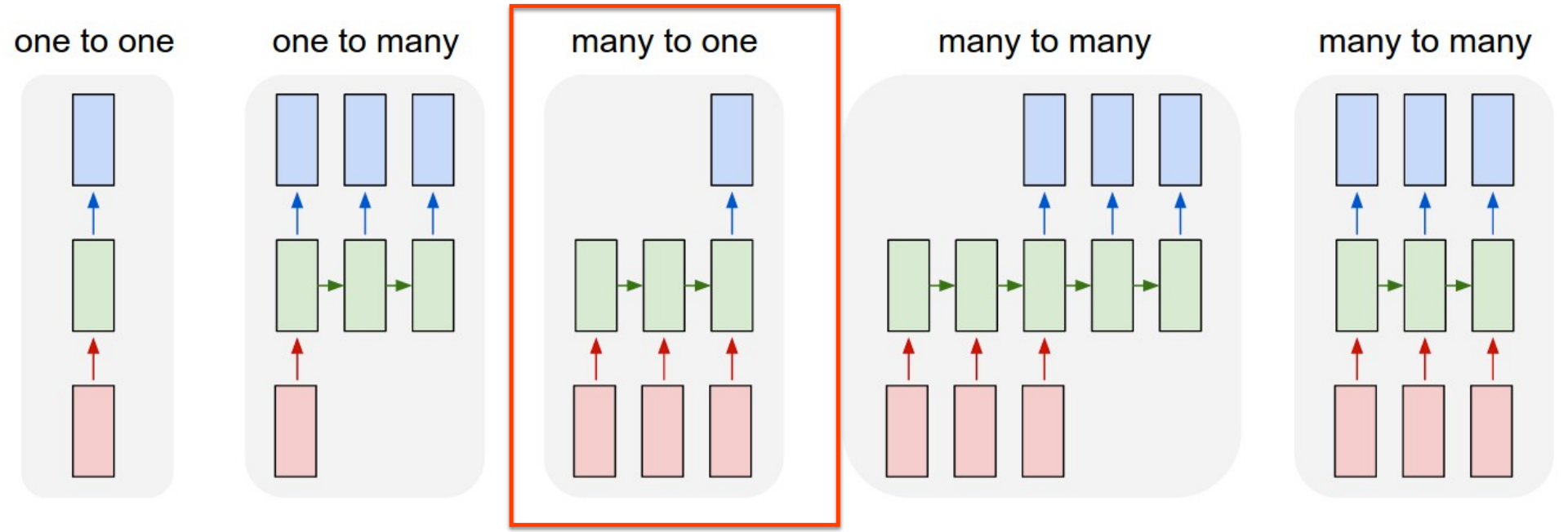
Recurrent Networks: Problem Types



Recurrent Networks: Problem Types



Recurrent Networks: Problem Types



The movie is great.

The movie stars Mr. X

The movie is horrible.

Recurrent Networks: Ontology Classification

Eva Ingolf is a well known Icelandic violinist particularly recognized for her authoritative performances of solo works by J. S. Bach. She comes from a leading musical family and her father Ingólfur Guðbrandsson premiered many of the great choral works in Iceland and six of her sisters and brothers are professional musicians who have made an important contribution to the high quality of the musical life in the country.Eva Ingolf currently lives in New York City with her husband Kristinn Sv.

Artist

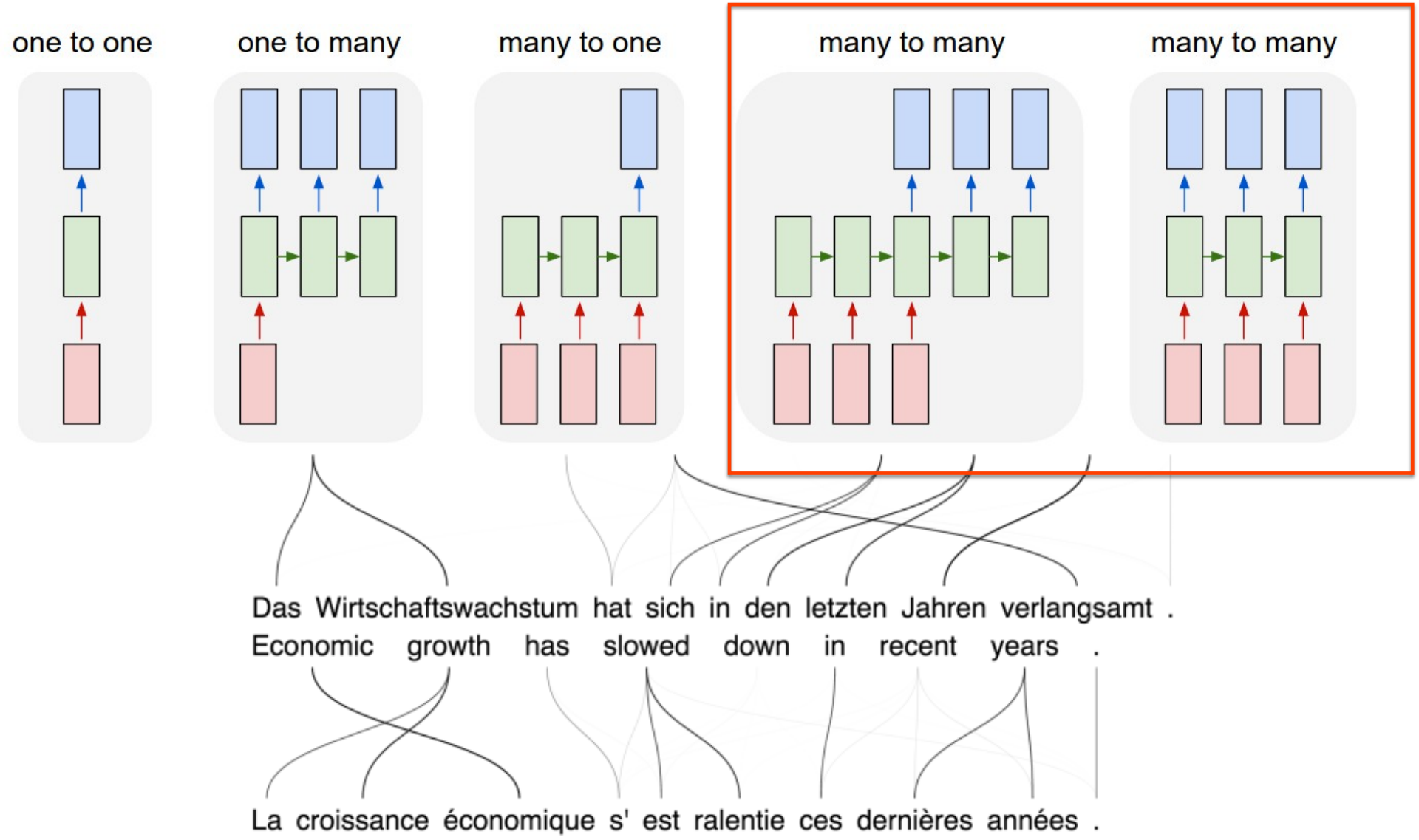
Shaun Norris (born 14 May 1982) is a South African professional golfer.Norris plays on the Sunshine Tour where he has won twice. He won the inaugural Africa Open in 2008 and the Nashua Masters in 2011. He also began playing on the European Tour in 2011 after graduating from qualifying school.

Athlete

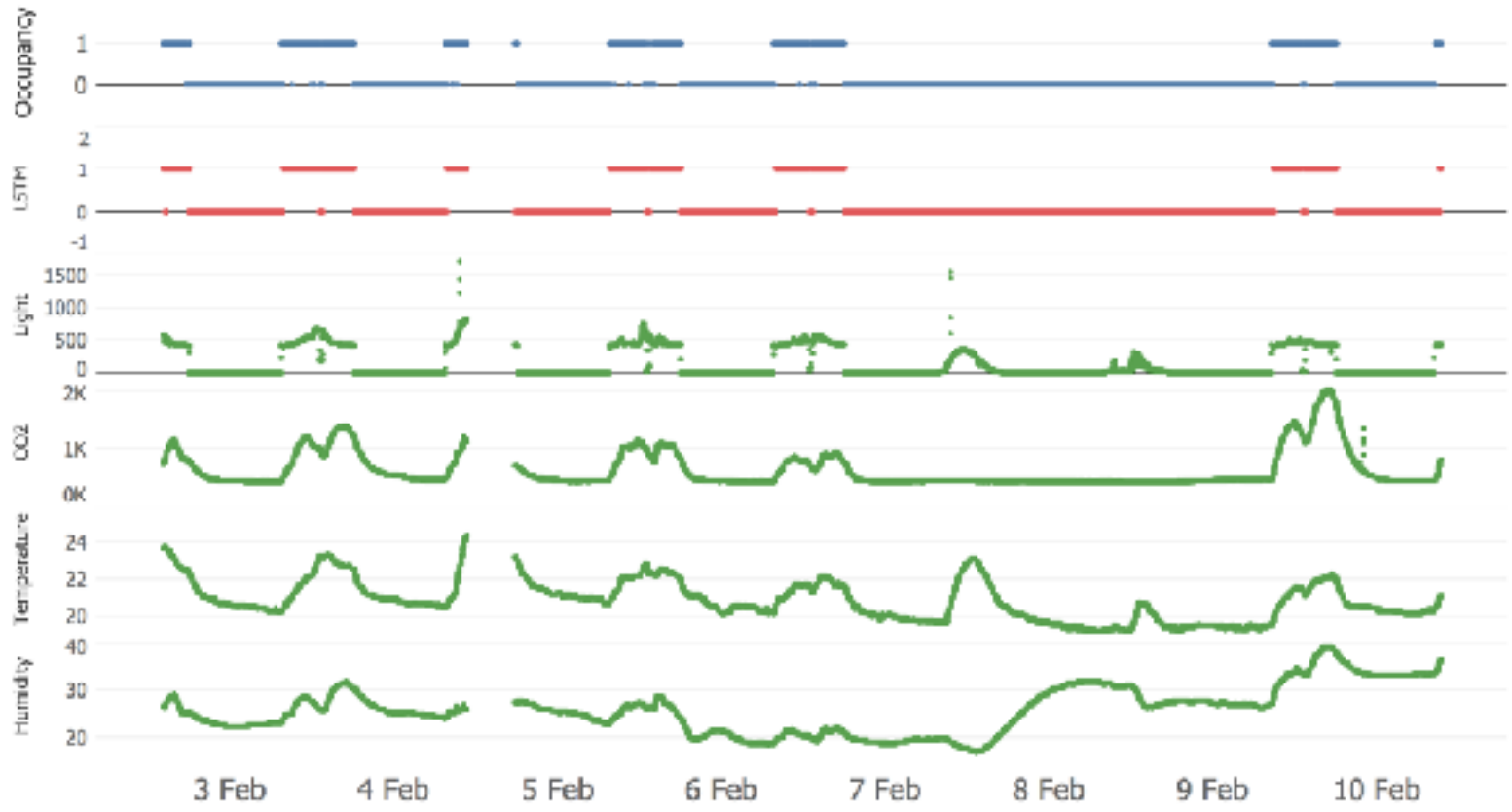
Palace Software was a British video game publisher and developer during the 1980s based in London England. It was notable for the Barbarian and Cauldron series of games for 8-bit home computer platforms in particular the ZX Spectrum Amstrad CPC and Commodore 64.

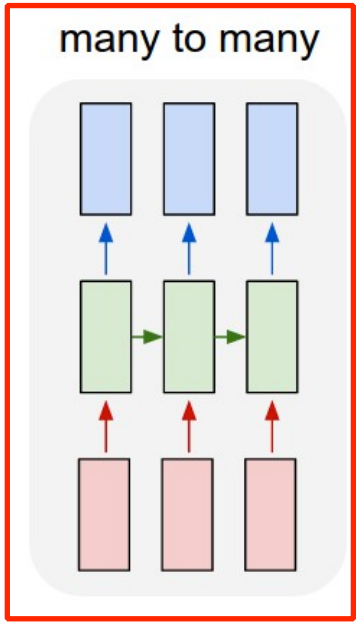
Company

Recurrent Networks: Problem Types



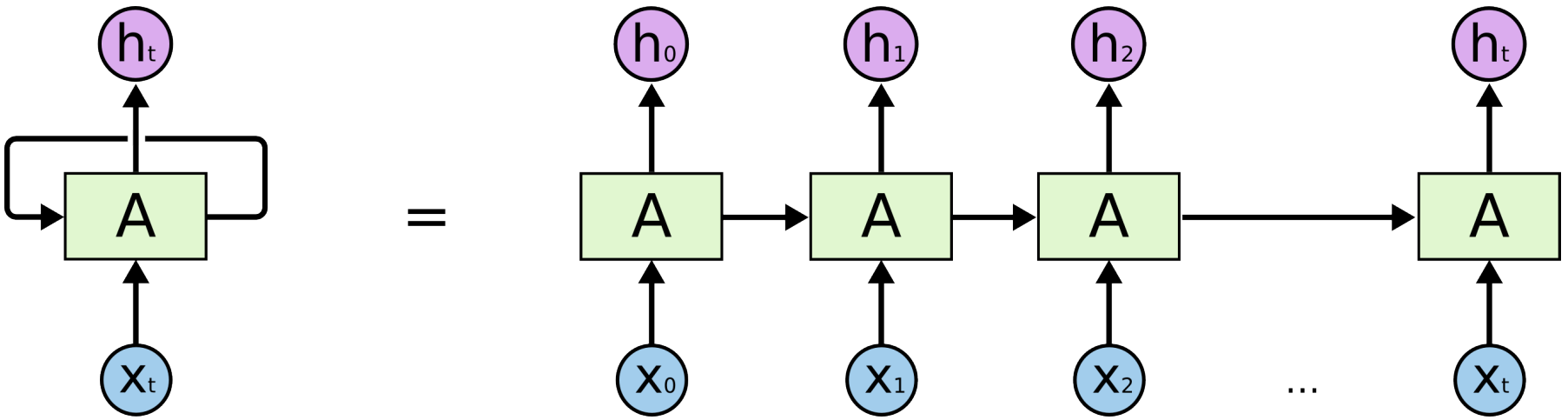
Recurrent Networks: Problem Types





sequence to sequence

Recurrent Networks: Main Idea

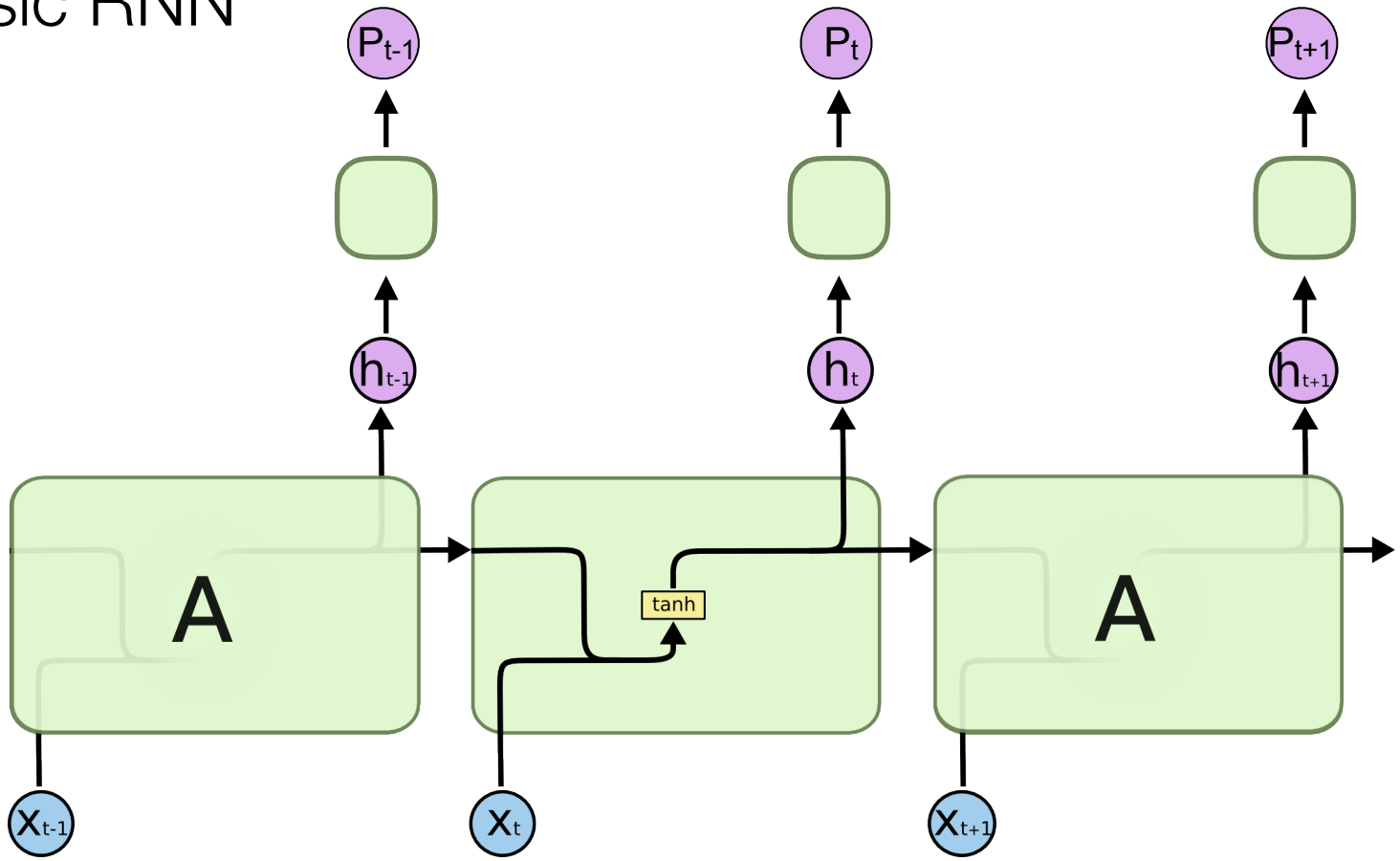


compact

unrolled

Recurrent Networks

- basic RNN

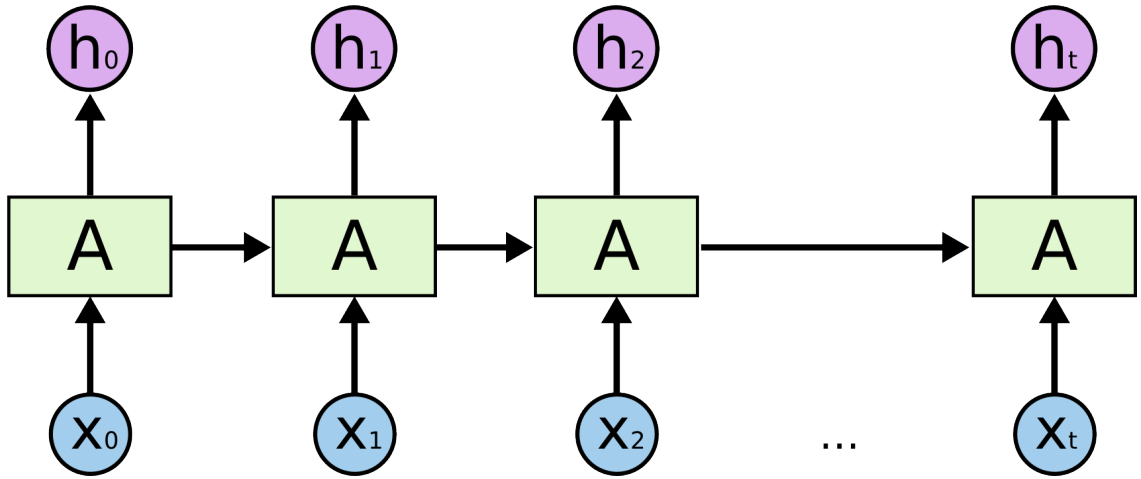


$$h_t = \tanh(W_A (X_t @ h_{t-1}) + b_A)$$
$$P_t = \text{softmax}(W_P h_t + b_P)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Networks: Representation

- python:

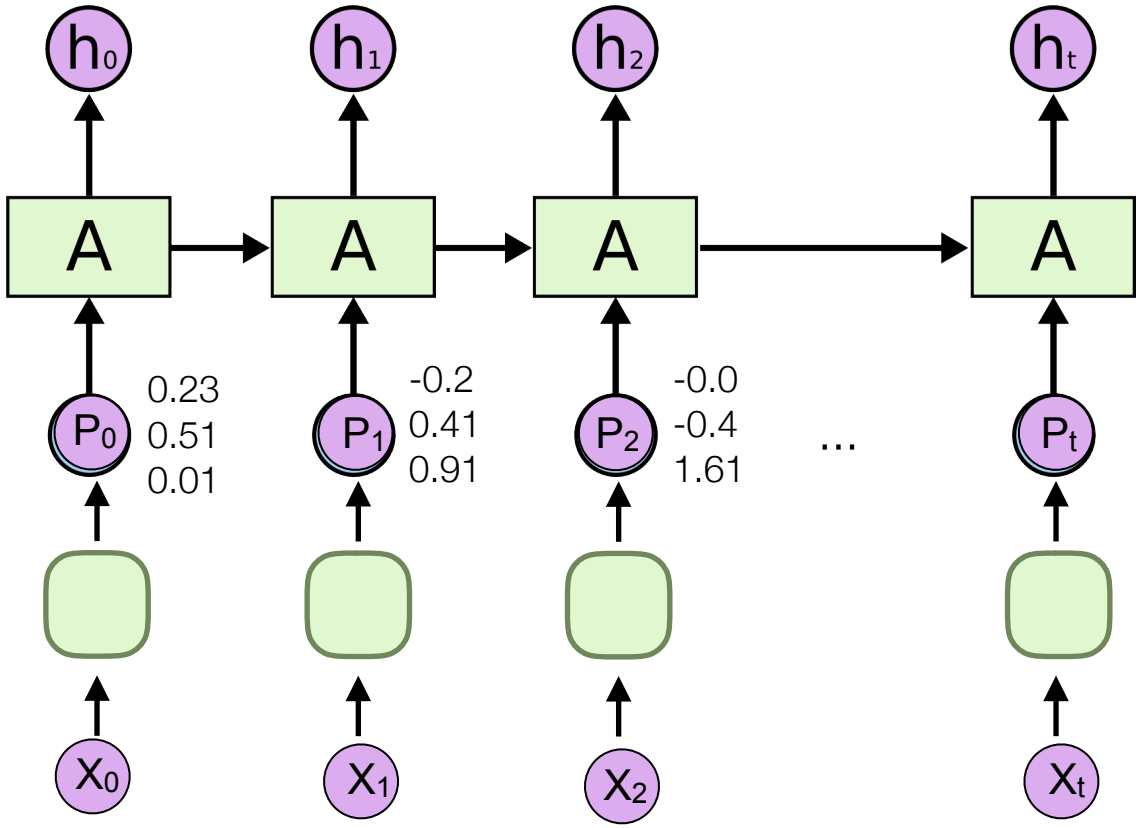


character:	T	h	e	q	u	i	c	k	b	r	o	w	n
int:	3			1					17				
one hot:	0 0 1 0 0 0 0 … 0 0			1 0 0 0 0 0 0 … 0 0					0 0 0 0 0 0 0 … 1 0				

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

Word Embeddings



int:	3	1	17	
one hot:	0 0 1 0 0 0 0	1 0 0 0 0 0 0	0 0 0 0 0 0 0	

Word Embeddings · Many are pre-trained for you!!

GloVe

Global Vectors for Word Representation

Highlights

1. Nearest neighbors

The Euclidean distance (or cosine similarity) between two word vectors provides an effective method for measuring the linguistic or semantic similarity of the corresponding words. Sometimes, the nearest neighbors according to this metric reveal rare but relevant words that lie outside an average human's vocabulary. For example, here are the closest words to the target word *frog*:

0. *frog*
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

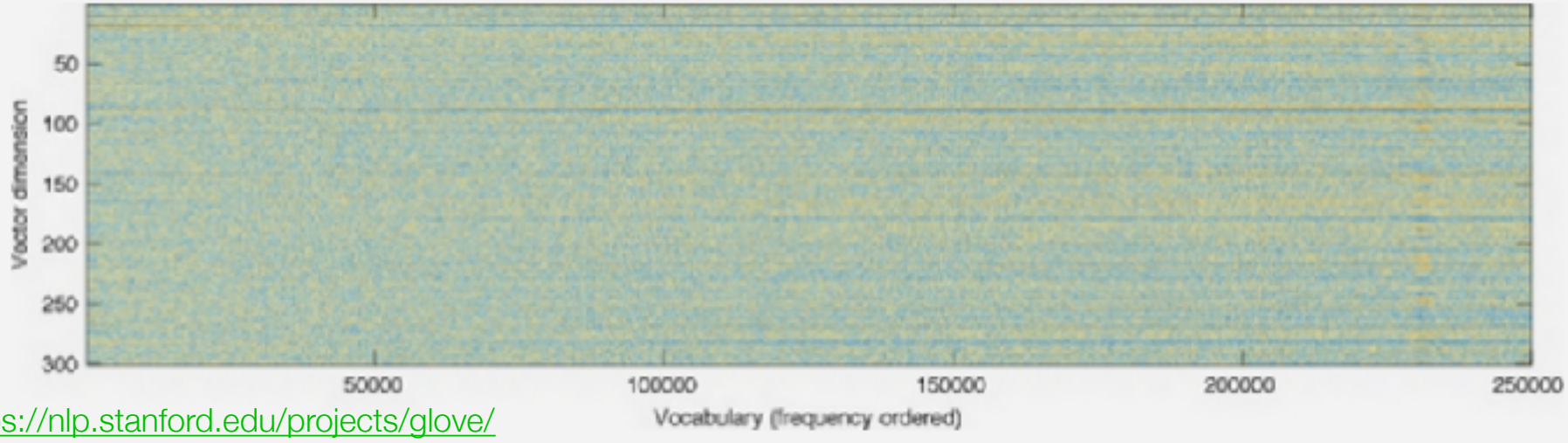
3. litoria

4. leptodactylidae

5. rana

7. eleutherodactylus

GloVe produces word vectors with a marked banded structure that is evident upon visualization:

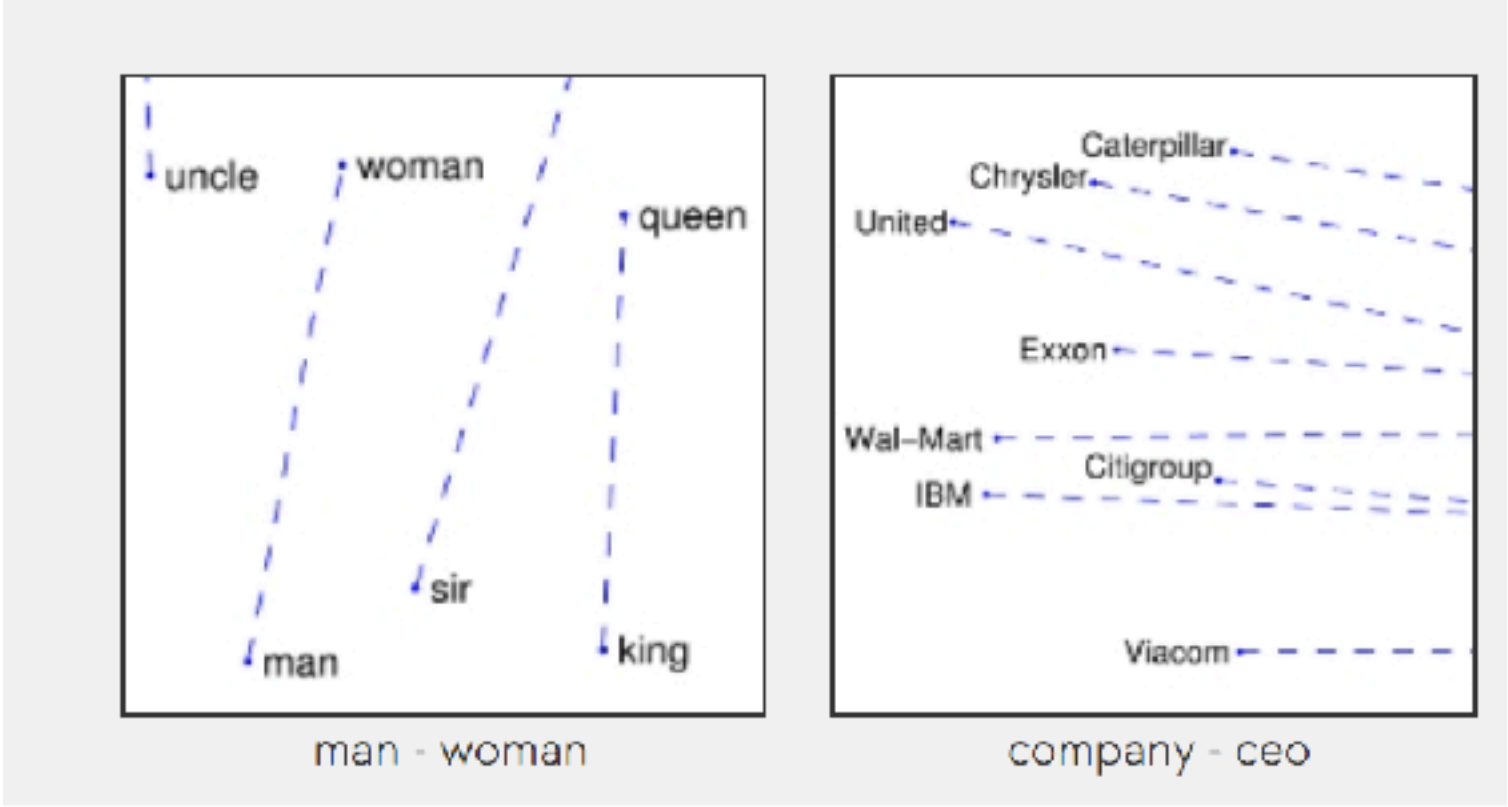


https://nlp.stanford.edu/projects/glove/

Word Embeddings · Many are pre-trained for you!!

GloVe

Global Vectors for Word Representation

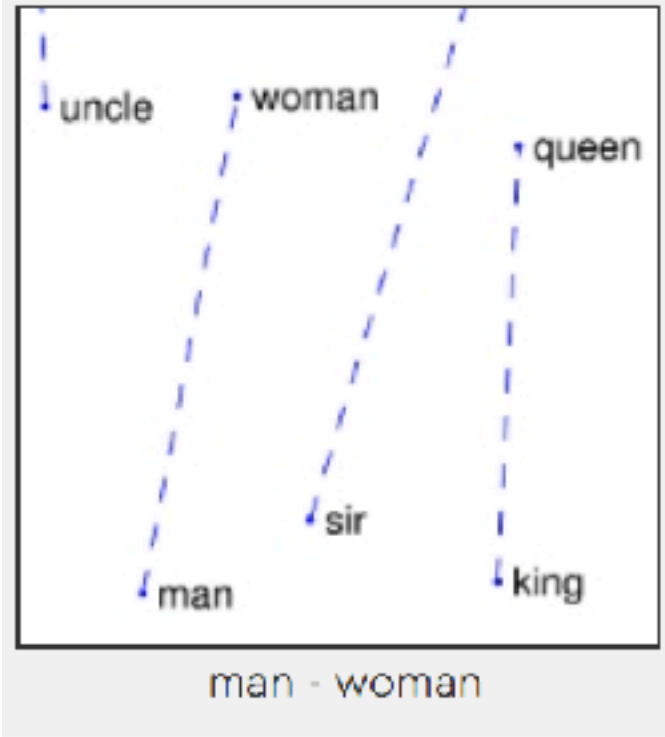


each axis might encode a different type of relationship

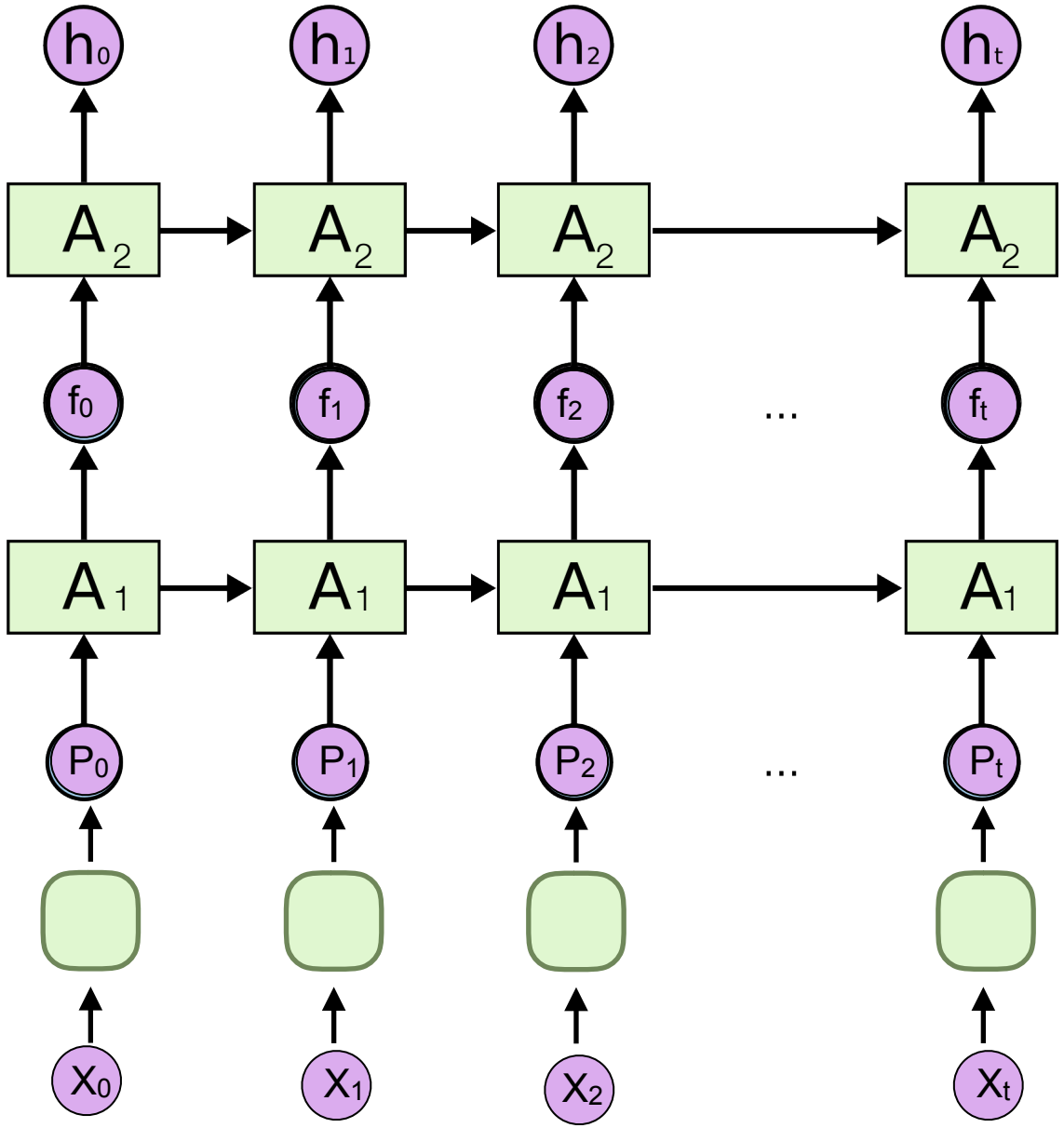
https://nlp.stanford.edu/projects/glove/

Self Test

- Each axis on the embedding plot below corresponds to:
- A. a weight inside the embedding layer
- B. the average of weigths in the embedding layer
- C. the average of the one hot encoding for a word
- D. two outputs of the embedding layer

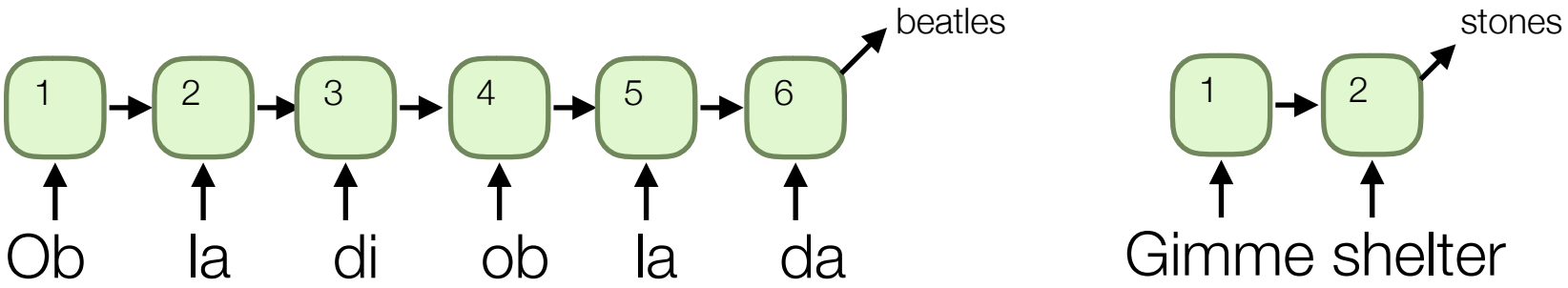


Sequence Stacking



Practical Logistics: Sequence Length

- option A: dynamic length sequences



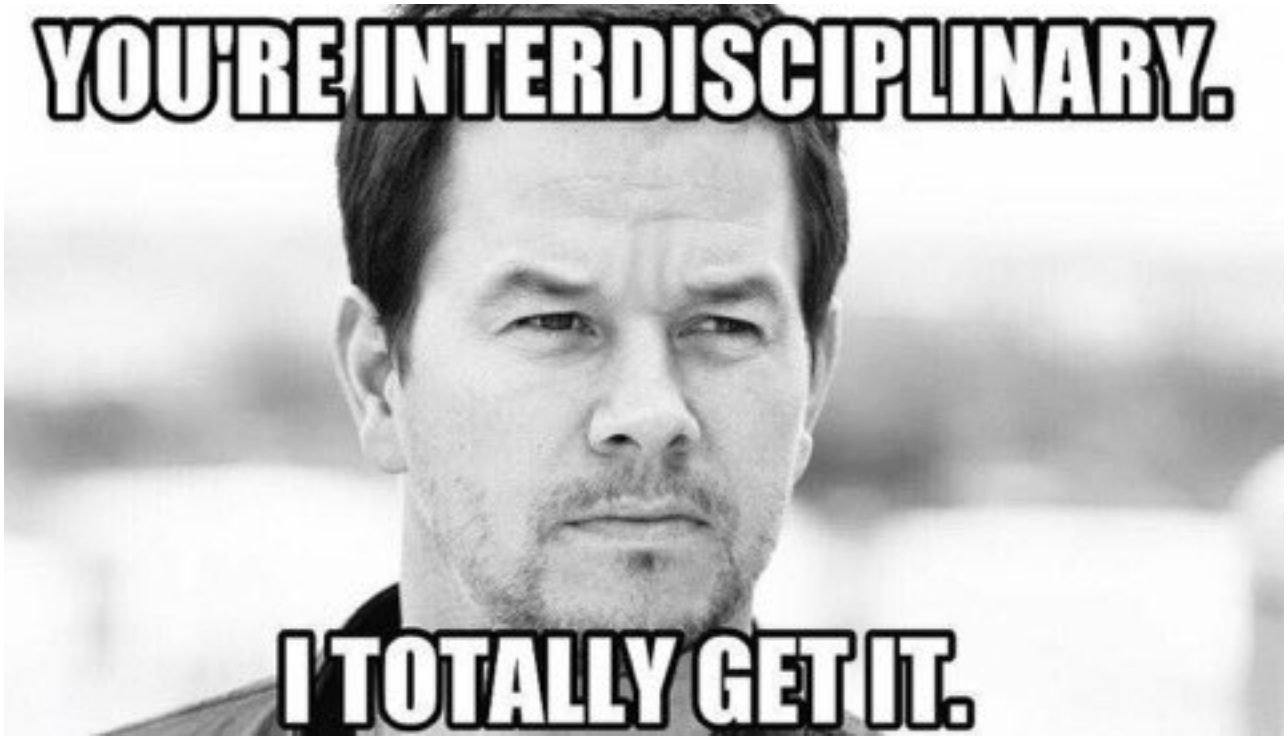
- option B: padding/clipping

- main difference: **speed**

Self Test

- The main reason dynamic length is slow is because:
  - A. the computation graph must be updated
  - B. weights must be tied together for each recurrent node
  - C. the weights must be multiplied until the output converges
  - D. the unrolling operation takes some time

For Fun: Recurrent Generation



Generating More Outputs

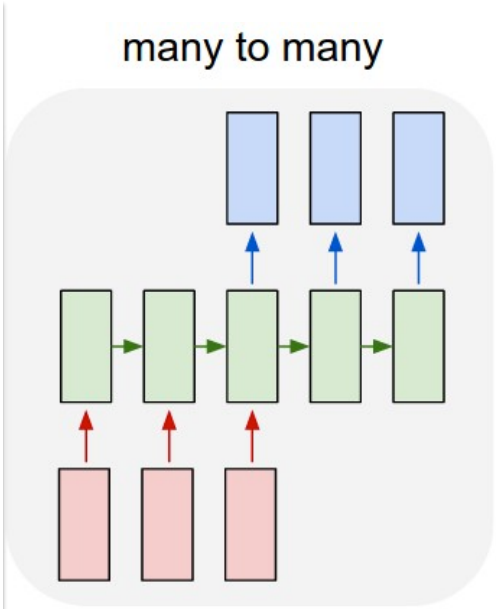
- Highly sophisticated steps:
  - train an RNN to generate
    the **next** word/character from
    the **current** word/character
  - train on a corpus of text
    - Shakespeare
    - Movie Scripts
    - Whatever!
  - seed with random word,
    feed output words as input to next node
  - rinse, repeat

Recurrent Networks: Character Data

```
First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

All:
We know't, we know't.

First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?

All:
No more talking on't; let it be done: away, away!

Second Citizen:
One word, good citizens.

ervices he has done for his country?
```

```
First Citizen:
We are accounted poor citize
What authority surfeits on w
would yield us but the super
wholesome, we might guess th
but they think we are too de
afflicts us, the object of o
inventory to particularise t
sufferance is a gain to them
our pikes, ere we become rak
speak this in hunger for bre

Second Citizen:
Would you proceed especially

All:
Against him first: he's a ve

Second Citizen:
Consider you what s
```

Recurrent Networks: Character Generation




```
ATOOOS

UIEAOUYOUZZZZZZUZAAAYAYf n fsflflrurctuateot t ta's  a  wtut
Whith then, a do makes and them and to sees,
I wark on this ance may string take thou honon
To sorriccorn of the bairer, whither, all
I'd see if yiust the would a peid.

LARYNGLe:
To would she troust they fould.

PENMES:
Thou she so the havin to my shald woust of
As tale we they all my forder have
As to say heant thy wansing thag and
Whis it thee shath his breact, I be and might, she
Tirs you desarvishensed and see thee: shall,
What he hath with that is all time,
And sen the have would be sectiens, way thee,
They are there to man shall with me to the mon,
And mere fear would be the balte, as time an at
And the say oun touth, thy way womers thee.
```

Recurrent Networks: Character Generation

more data, star wars + star trek + tarantino + the matrix

```
DENT'SUEENCK

Bartholomew of the TIE FIGHTERS are stunned. There is a crowd and armored
switcheroos.

PICARD
(continuing)
Couns two dim is tired. In order to the sentence...

The sub    bottle appears on the screen into a small shuttle shift of the
ceiling. The DAMBA FETT splash fires and matches them into the top, transmit to stable high above
upon their statels,
falling from an alien shaft.

ANAKIN and OBI-WAN stand next to OBI-WAN down the control plate of smoke at the TIE fighter. They
stare at the centre of the station loose into a comlink cover -- comes up to the General, the
GENERAL HUNTAN AND FINNFURMBARD from the PICADOR to a beautiful Podracisly.

ENGINEER
Naboo from an army seventy medical
security team area re-weilergular.

EXT.
```

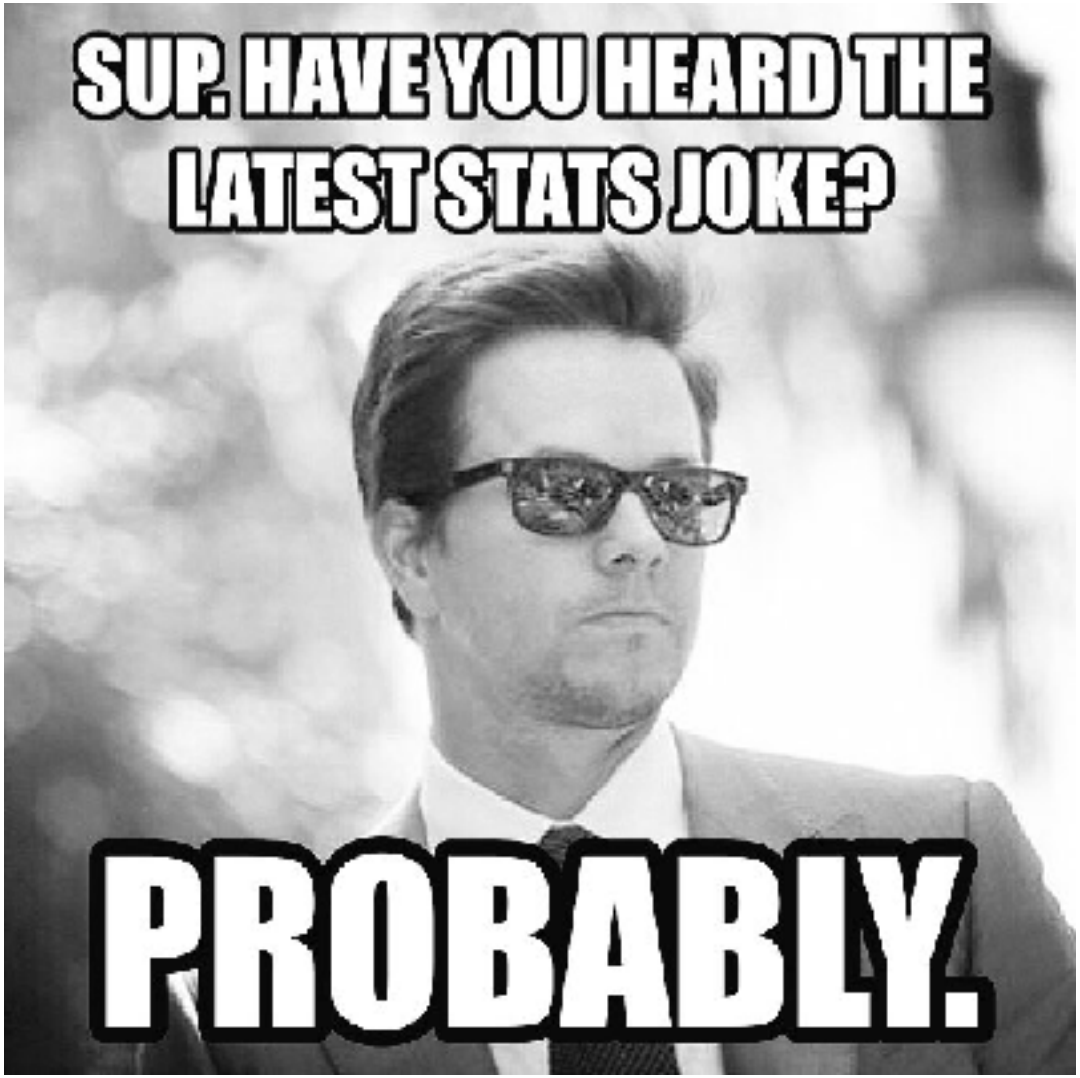
THE MULTIVERSE —

Movie written by algorithm turns out to be hilarious and intense

For *Sunspring*'s exclusive debut on Ars, we talked to the filmmakers about collaborating with an AI.

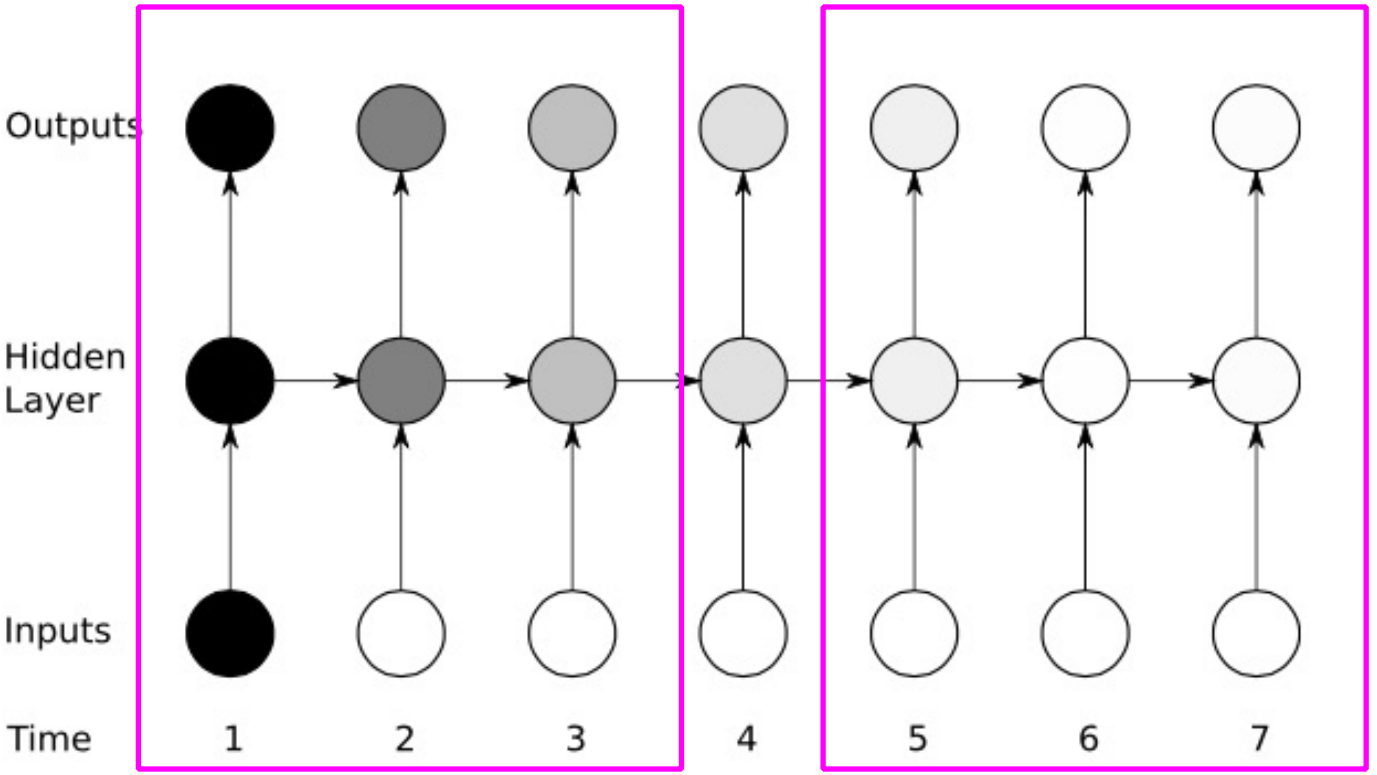
ANNALEE NEWITZ - 6/9/2016, 5:30 AM

Back to Reality: Types of RNNs



Recurrent Networks, the Age Old Problem

- vanishing gradients: is this the same?

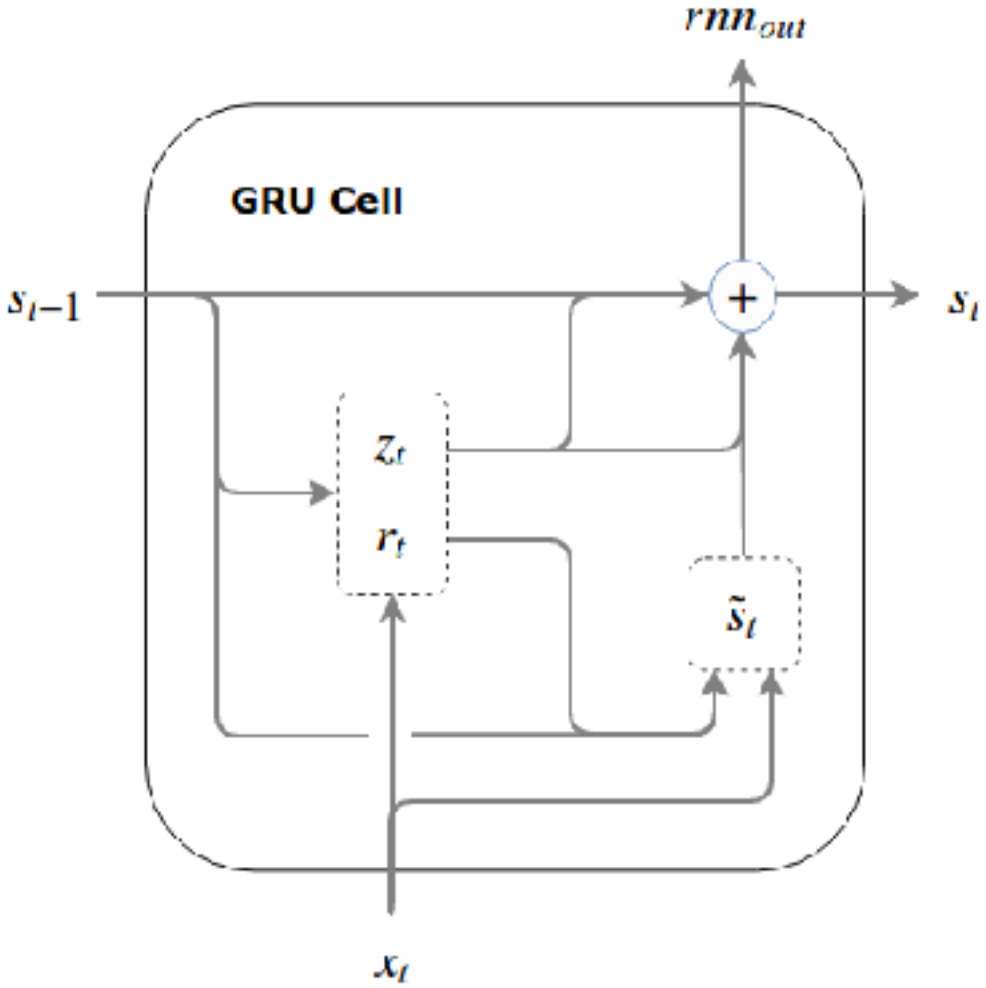


$$\frac{\partial E_t}{\partial S_{t-k}} = \frac{\partial E_t}{\partial S_t}\frac{\partial S_t}{\partial S_{t-k}} = \frac{\partial E_t}{\partial S_t}\left(\frac{\partial S_t}{\partial S_{t-1}}\frac{\partial S_{t-1}}{\partial S_{t-2}}\cdots\frac{\partial S_{t-k+1}}{\partial S_{t-k}}\right) = \frac{\partial E_t}{\partial S_t}\prod_{i=1}^{k}\frac{\partial S_{t-i+1}}{\partial S_{t-i}}$$

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

Recurrent Networks: GRUs

- gated recurrent units



Selectivity controls, gates (0 or 1)

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

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

remember — with input

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

$$s_t = z_t \odot s_{t-1} + (1 - z_t) \odot \tilde{s}_t$$

remember past — remember with input

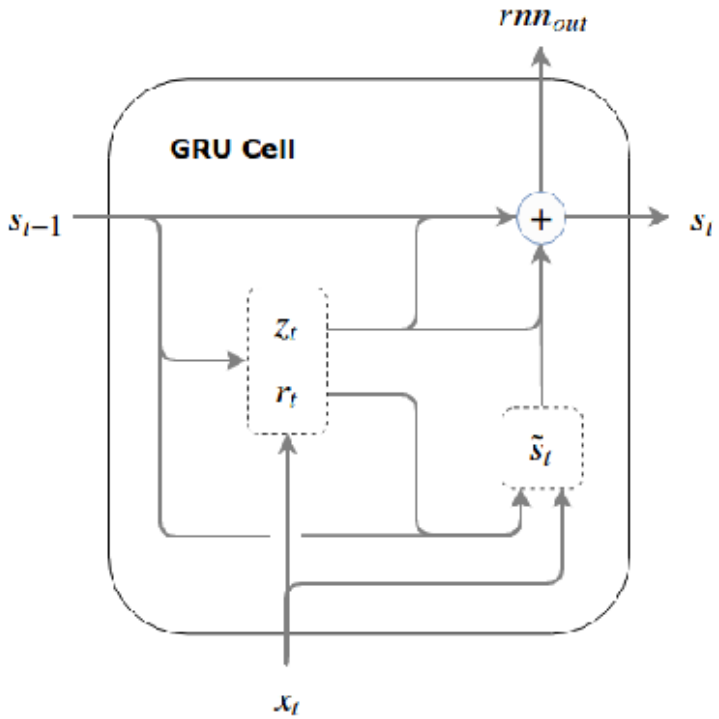
σ = hard limit

⊙ = elem. multiplication

http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Self Test

- What element of the GRU helps with vanishing and exploding gradients?
- A. derivative of ϕ
- B. derivative of σ
- C. σ
- D. ϕ

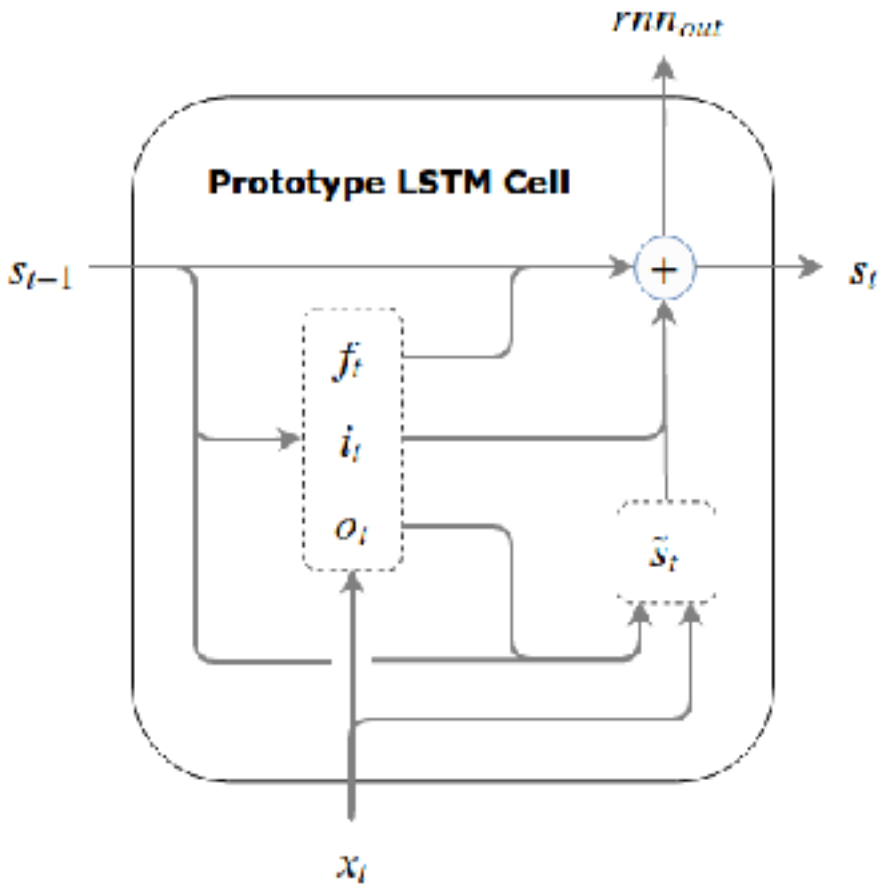


$$r_t = \sigma(W_r s_{t-1} + U_r x_t + b_r)$$
$$z_t = \sigma(W_z s_{t-1} + U_z x_t + b_z)$$

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

Recurrent Networks: Gen 1 LSTM

- LSTM prototype



Selectivity controls (gates, 0 or 1)

$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$$

remember past with input

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

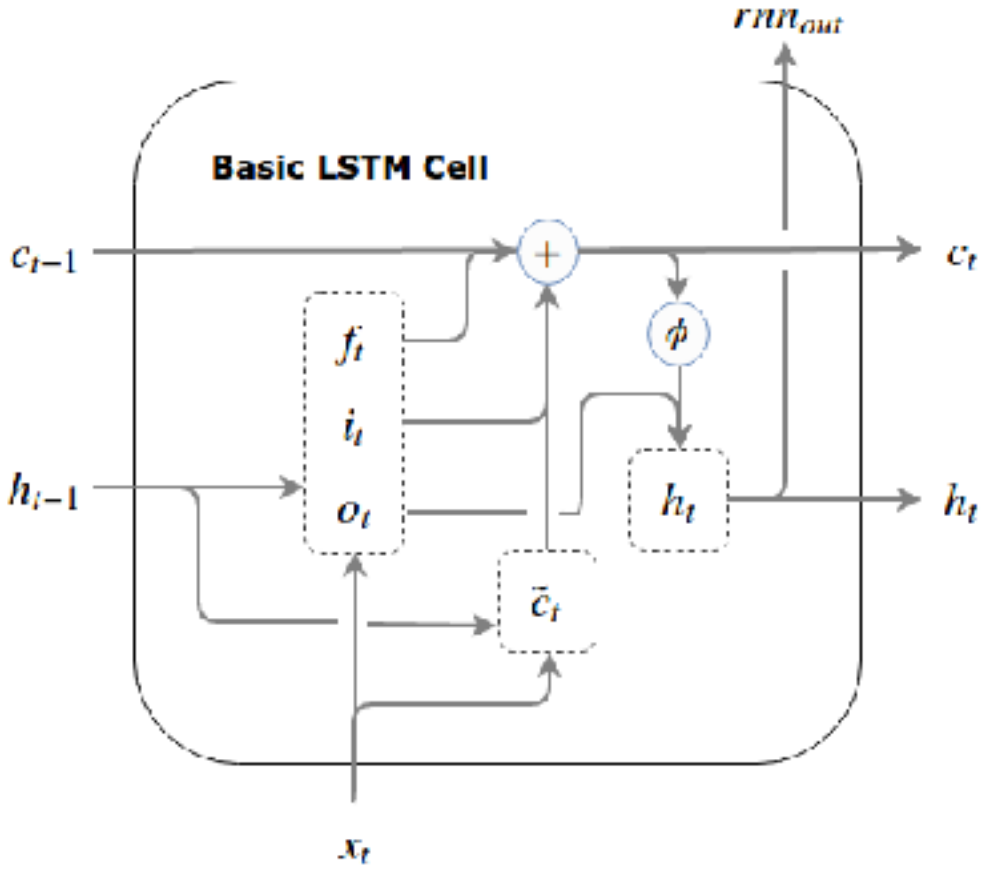
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

remember past with output

http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Recurrent Networks: Gen 2 LSTM

- LSTM in TensorFlow



Selectivity controls

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

explicit remembering state

$$\tilde{c}_t = \phi(W h_{t-1} + U x_t + b)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

remember state remember output

$$h_t = o_t \odot \phi(c_t)$$

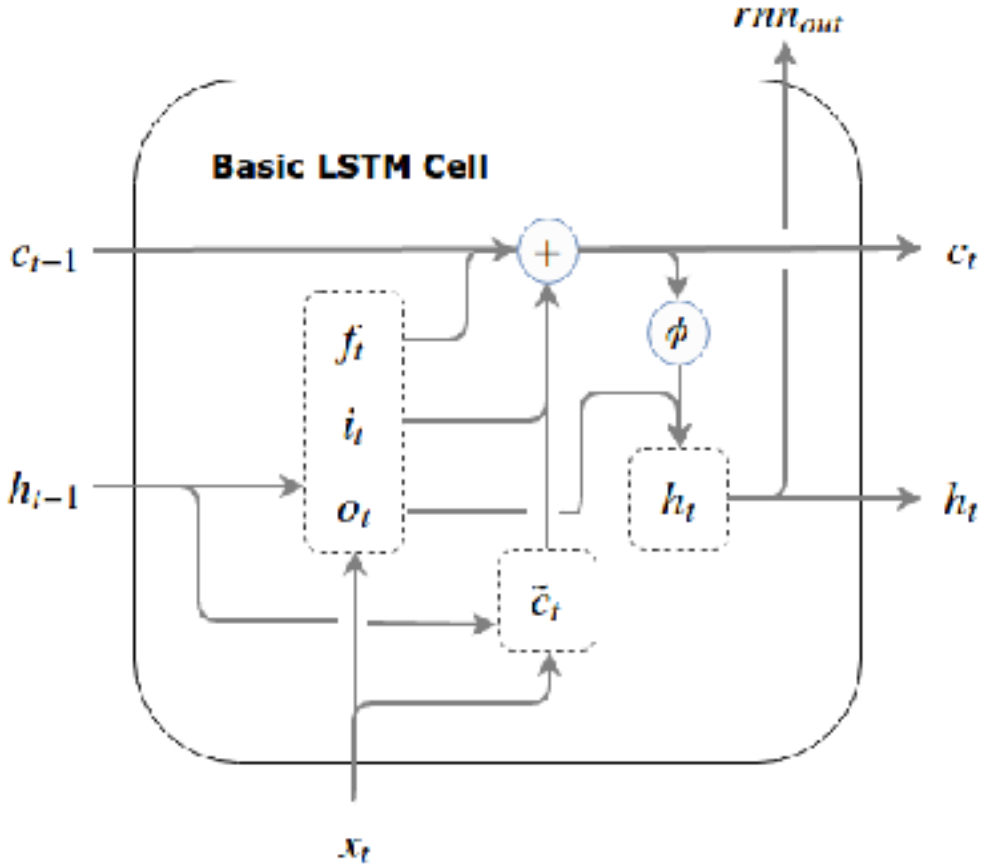
remember both

$$\text{rnn}_{out} = h_t$$

http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

LSTM Dropout

- LSTM in TensorFlow



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

Reccurrent Dropout

Input Dropout

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

http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

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…

Lecture Notes for Machine Learning in Python

Professor Eric Larson

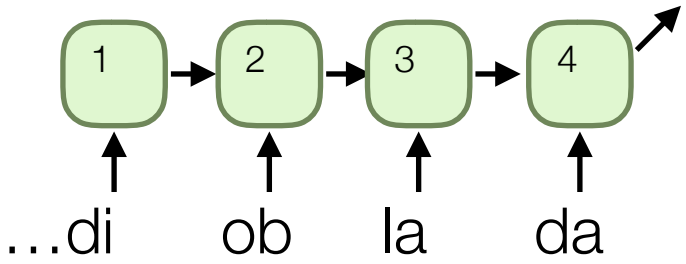
Final Lecture: RNN Demo

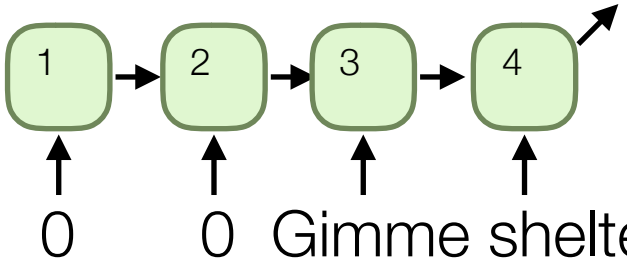
Lecture Agenda

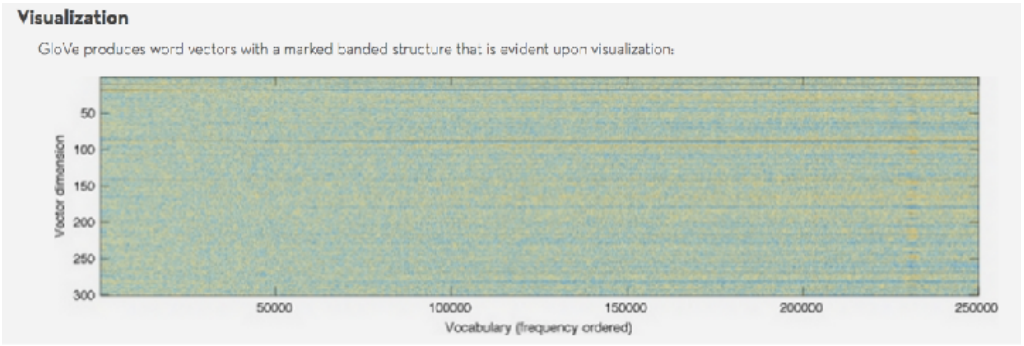
- Logistics
  - RNNs due **Friday, May 12** (day of finals, **5PM**)
- Recurrent Networks (two lecture agenda)
  - *Overview*
  - *Problem Types*
  - *Embeddings*
  - *Types of RNNs*
  - Demo A
  - CNNs and RNNs
  - Demo B
  - course retrospective

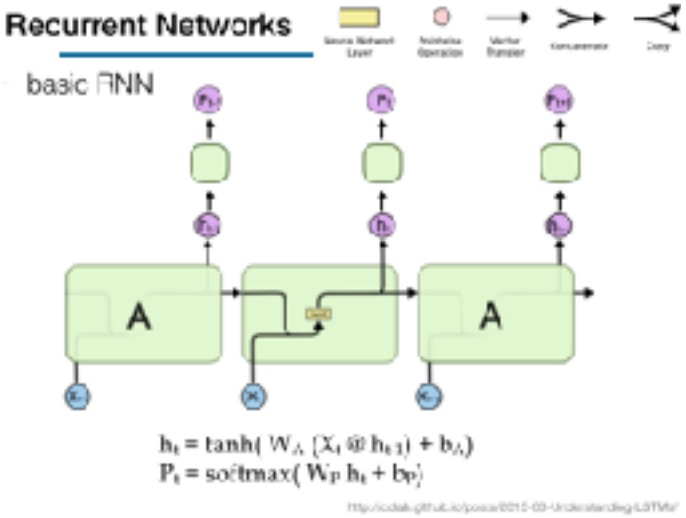
Last Time

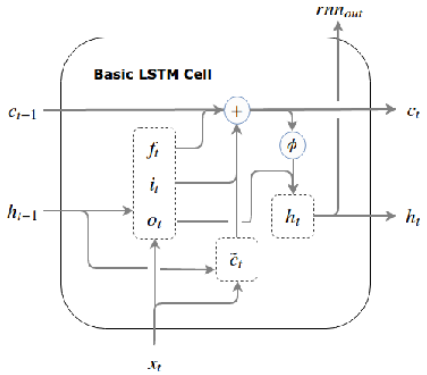
- option B: padding/clipping

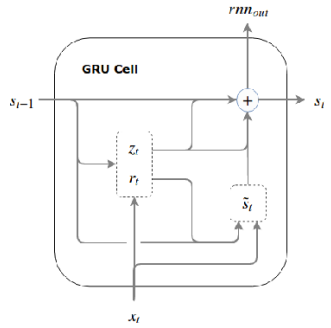






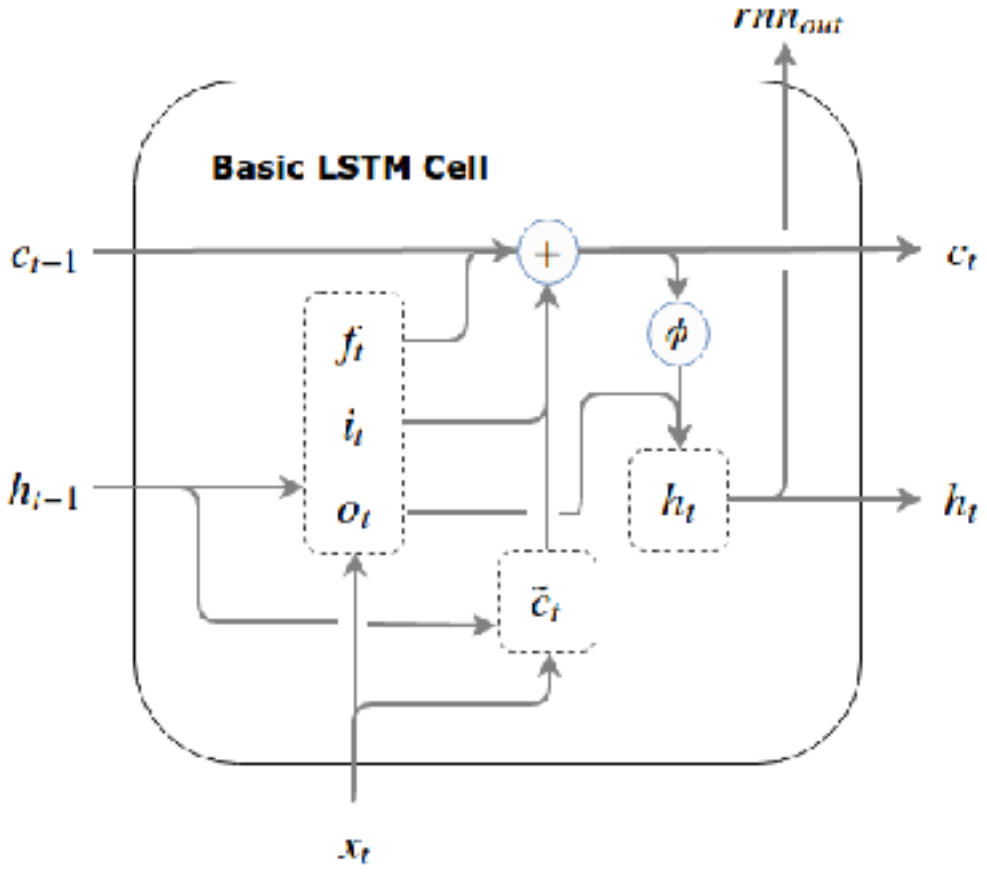






Recurrent Networks: Gen 2 LSTM

- LSTM in TensorFlow



Selectivity controls

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

explicit remembering state

$$\tilde{c}_t = \phi(W h_{t-1} + U x_t + b)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

remember state remember output

$$h_t = o_t \odot \phi(c_t)$$

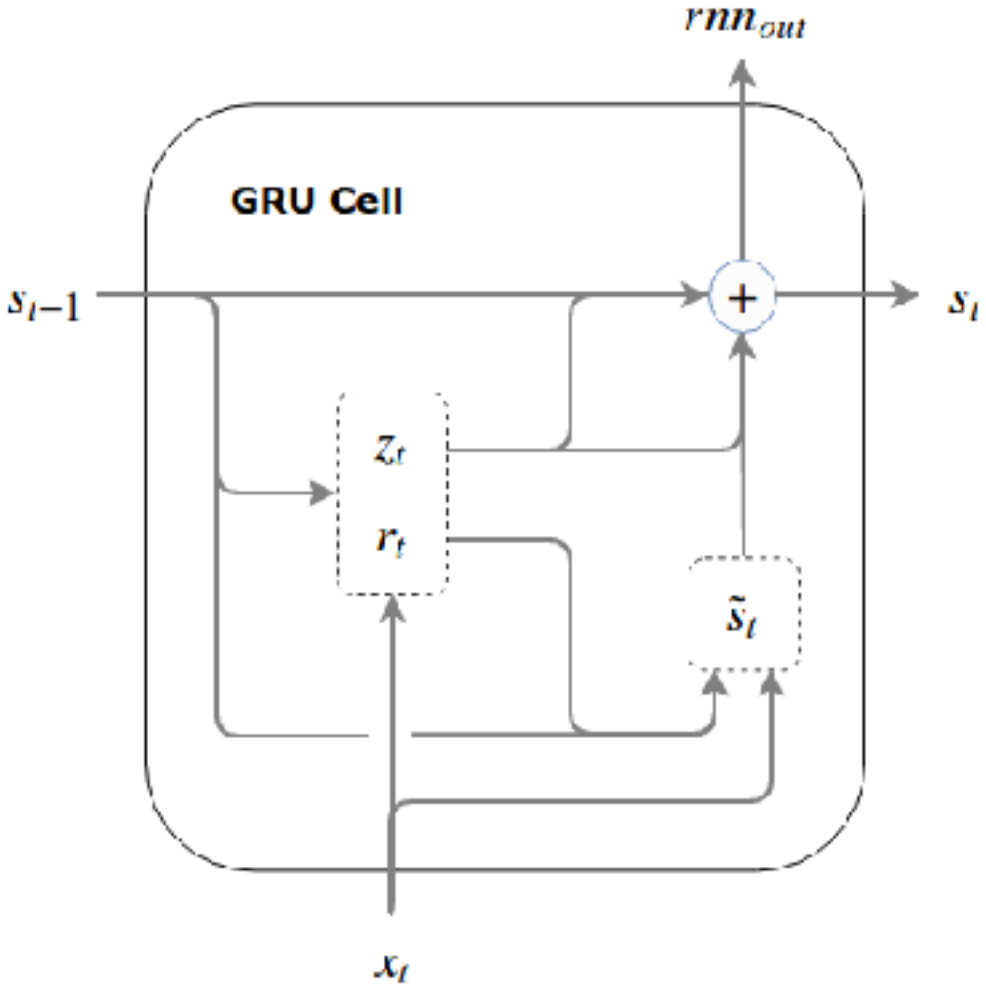
remember both

$$\text{rnn}_{out} = h_t$$

http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Recurrent Networks: GRUs

- gated recurrent units



Selectivity controls, gates (0 or 1)

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

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

remember — with input

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

$$s_t = z_t \odot s_{t-1} + (1 - z_t) \odot \tilde{s}_t$$

remember past — remember with input

σ = hard limit

⊙ = elem. multiplication

http://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Final Demo - Part A

Recurrent Networks in Keras

Many to one:

- Simple RNNs
- GRUs
- LSTMs

More examples:

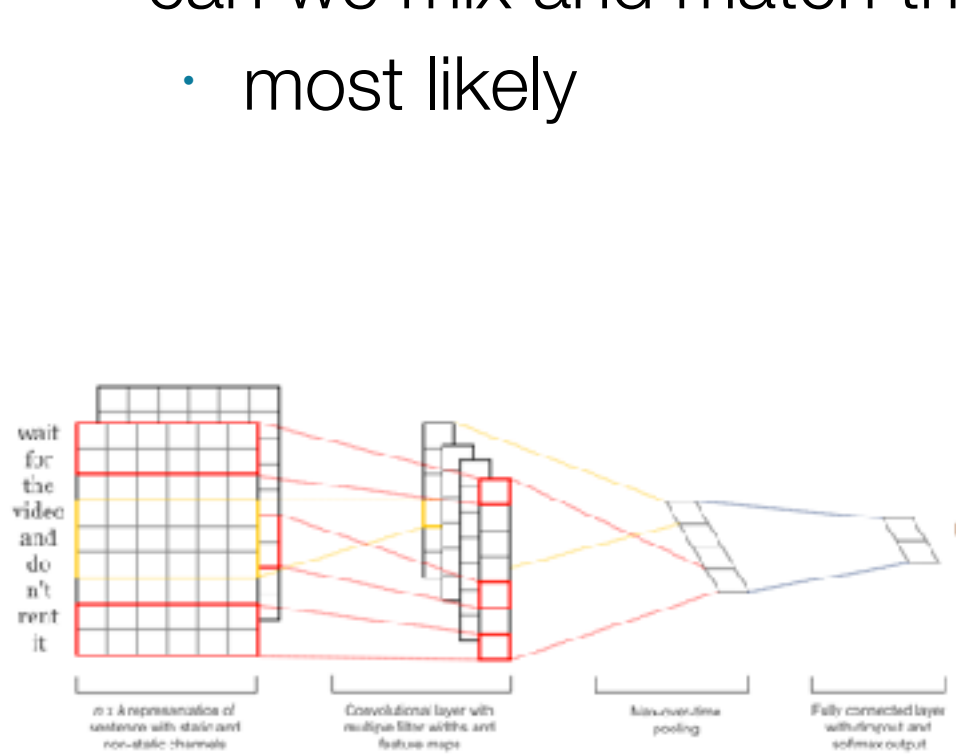
https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/skflow
http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html
http://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

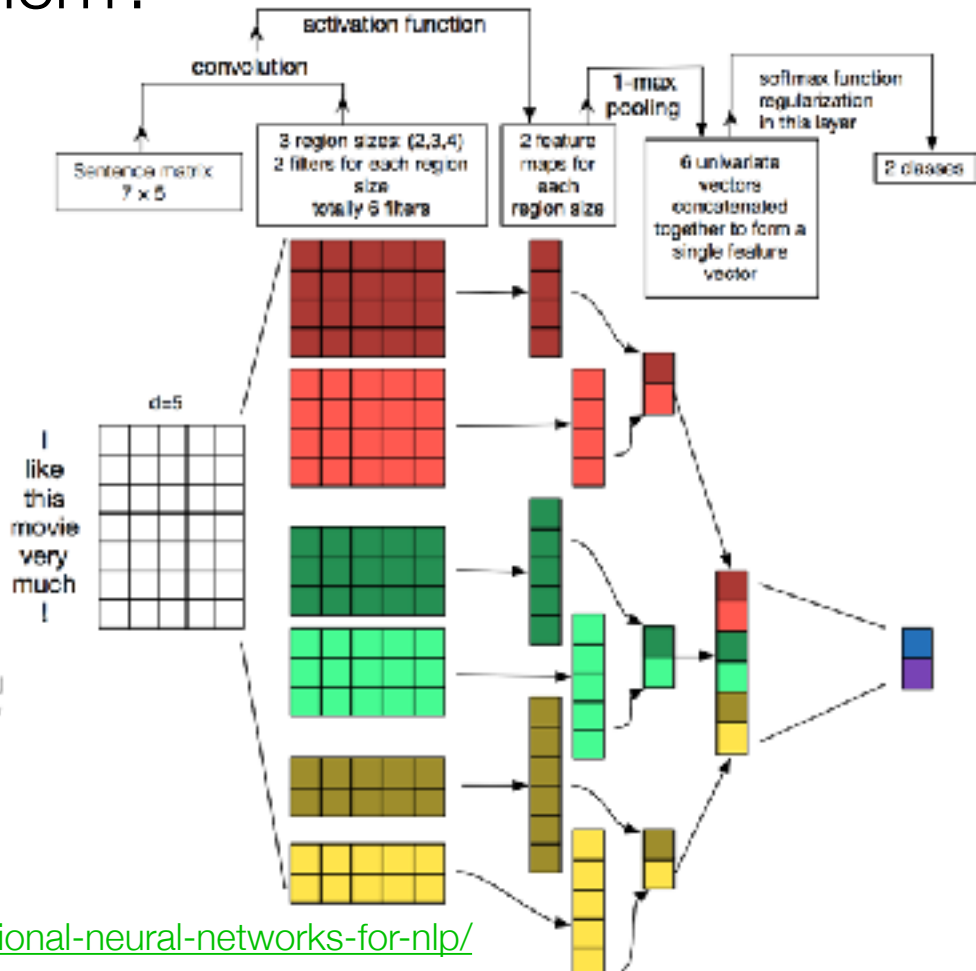
Seq2Seq:

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

CNNs and RNNs

- is an RNN similar to a CNN?
  - given fixed length sequences, probably
- can we mix and match them?
  - most likely





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

Final Demo - Part B

Recurrent Networks in Keras

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

Thank you for a great semester!

- but it could **have been better** somehow, right?
  - if you went into a job interview for a data scientist position and where asked about ***X***, that you did not feel strong in, what would ***X*** be?
    - i.e, what didn't translate well?
    - i.e., what are you **still not comfortable** with?
  - how could you learn better? same datasets to compare with others?
  - what should **not be** cut or changed?
  - what **skills** should students have **before this class** that I don't yet require?

Backup slides

TensorFlow

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
with tf.variable_scope('rnn_cell'):
    W = tf.get_variable('W', [num_classes + state_size, state_size])
    b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(0.0))

def rnn_cell(rnn_input, state):
    with tf.variable_scope('rnn_cell', reuse=True):
        W = tf.get_variable('W', [num_classes + state_size, state_size])
        b = tf.get_variable('b', [state_size], initializer=tf.constant_initializer(0.0))
    return tf.tanh(tf.matmul(tf.concat(1, [rnn_input, state]), W) + b)

state = init_state
rnn_outputs = []
for rnn_input in rnn_inputs:
    state = rnn_cell(rnn_input, state)
    rnn_outputs.append(state)
final_state = rnn_outputs[-1]

#logits and predictions
with tf.variable_scope('softmax'):
    W = tf.get_variable('W', [state_size, num_classes])
    b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
logits = [tf.matmul(rnn_output, W) + b for rnn_output in rnn_outputs]
predictions = [tf.nn.softmax(logit) for logit in logits]

# Turn our y placeholder into a list labels
y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

#losses and train_step
losses = [tf.nn.sparse_softmax_cross_entropy_with_logits(logit,label) for \
          logit, label in zip(logits, y_as_list)]
total_loss = tf.reduce_mean(losses)
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```

recurrent networks

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
def train_network(num_epochs, num_steps, state_size=4, verbose=True):
    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        training_losses = []
        for idx, epoch in enumerate(gen_epochs(num_epochs, num_steps)):
            training_loss = 0
            training_state = np.zeros((batch_size, state_size))
            if verbose:
                print("\nEPOCH", idx)
            for step, (X, Y) in enumerate(epoch):
                tr_losses, training_loss_, training_state, _ = \
                    sess.run([losses,
                              total_loss,
                              final_state,
                              train_step],
                                  feed_dict={x:X, y:Y, init_state:training_state})
                training_loss += training_loss_
                if step % 100 == 0 and step > 0:
                    if verbose:
                        print("Average loss at step", step,
                              "for last 250 steps:", training_loss/100)
                    training_losses.append(training_loss/100)
                    training_loss = 0

    return training_losses
```

TensorFlow

http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

```
def train_network(num_epochs, num_steps, state_size=4, verbose=True):
    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        for idx, epoch in enumerate(gen_epochs(num_epochs, num_steps)):
            training_state = np.zeros((batch_size, state_size))
            for X, Y in epoch:
                tr_losses, training_loss_, training_state, _ = \
                    sess.run([losses,
                              total_loss,
                              final_state,
                              train_step],
                             feed_dict={x:X, y:Y, init_state:training_state})
```

TensorFlow (simplified)

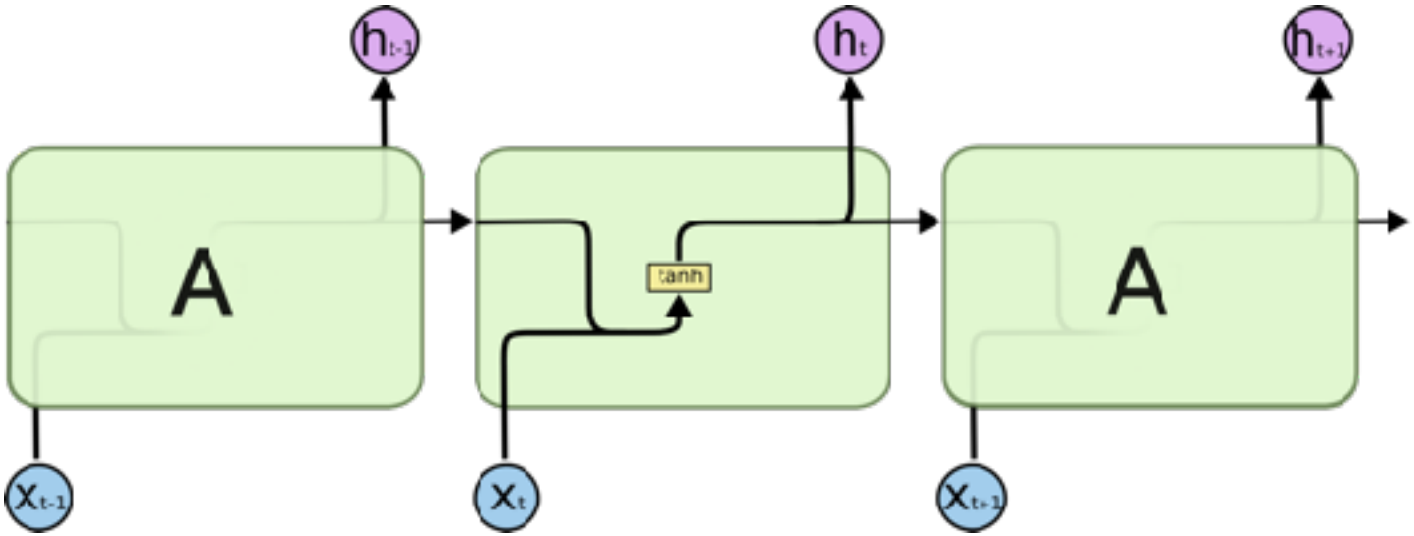
http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

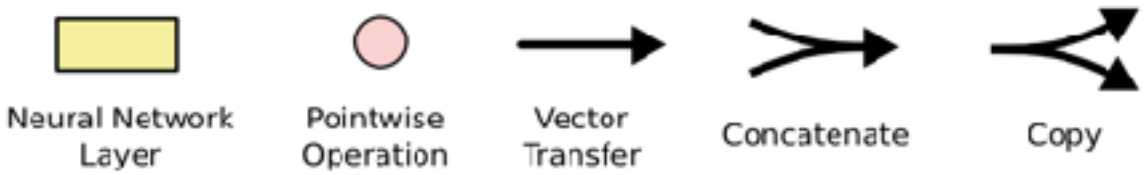
```
cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
rnn_outputs, final_state = tf.nn.rnn(cell, rnn_inputs, initial_state=init_state)


loss_weights = [tf.ones([batch_size]) for i in range(num_steps)]
losses = tf.nn.seq2seq.sequence_loss_by_example(logits, y_as_list, loss_weights)
```

```
x = tf.placeholder(tf.int32, [batch_size, num_steps], name='input_placeholder')
y = tf.placeholder(tf.int32, [batch_size, num_steps], name='labels_placeholder')
init_state = tf.zeros([batch_size, state_size])

x_one_hot = tf.one_hot(x, num_classes)
rnn_inputs = tf.unpack(x_one_hot, axis=1)

cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
rnn_outputs, final_state = tf.nn.rnn(cell, rnn_inputs, initial_state=init_state)


with tf.variable_scope('softmax'):
    W = tf.get_variable('W', [state_size, num_classes])
    b = tf.get_variable('b', [num_classes], initializer=tf.constant_initializer(0.0))
logits = [tf.matmul(rnn_output, W) + b for rnn_output in rnn_outputs]
predictions = [tf.nn.softmax(logit) for logit in logits]

y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

loss_weights = [tf.ones([batch_size]) for i in range(num_steps)]
losses = tf.nn.seq2seq.sequence_loss_by_example(logits, y_as_list, loss_weights)
total_loss = tf.reduce_mean(losses)
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```

More Advanced Architectures

- **LSTM key idea**: limit how past data can affect output

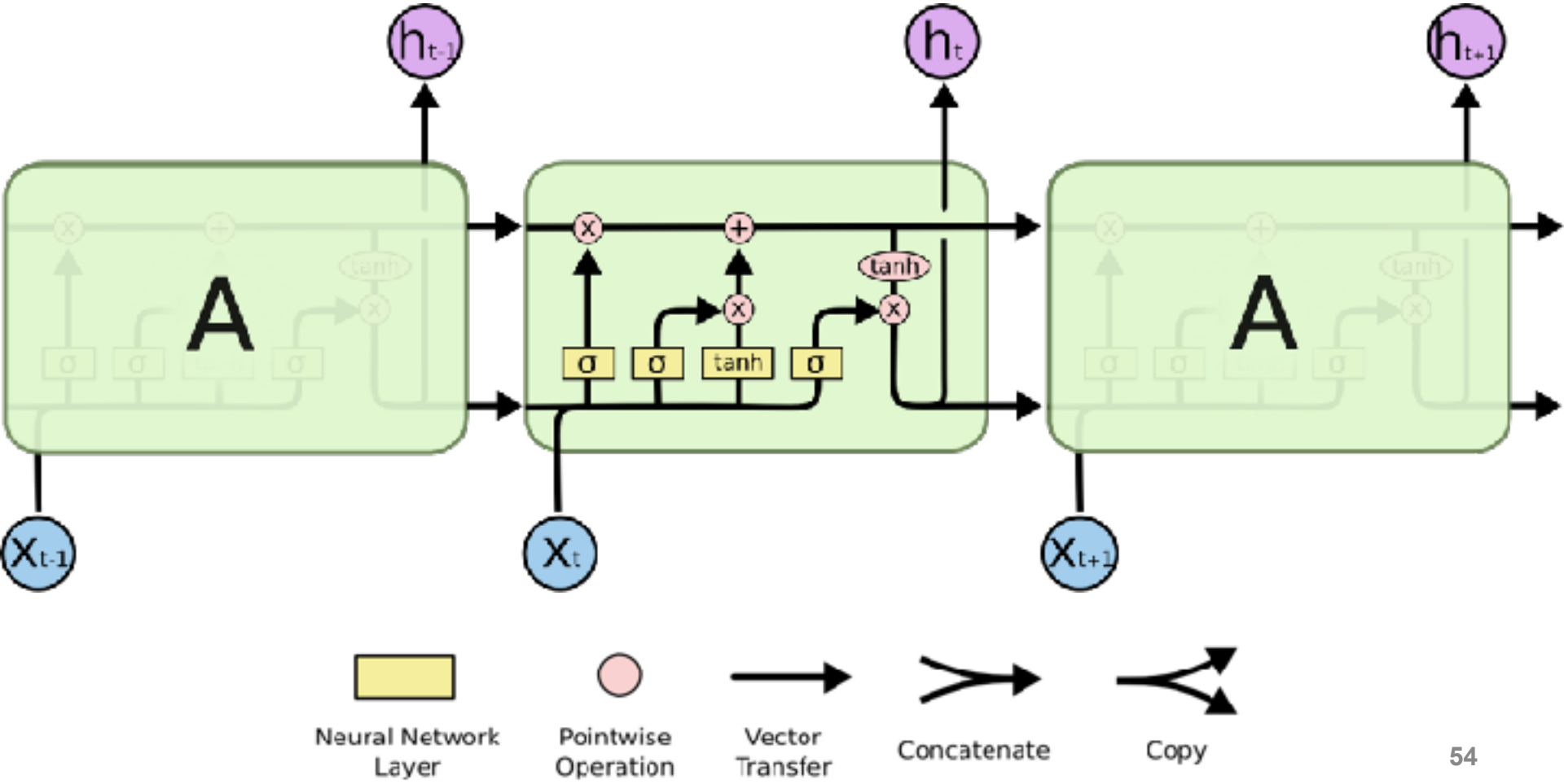




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

More Advanced Architectures

- **LSTM key idea**: limit how past data can affect output

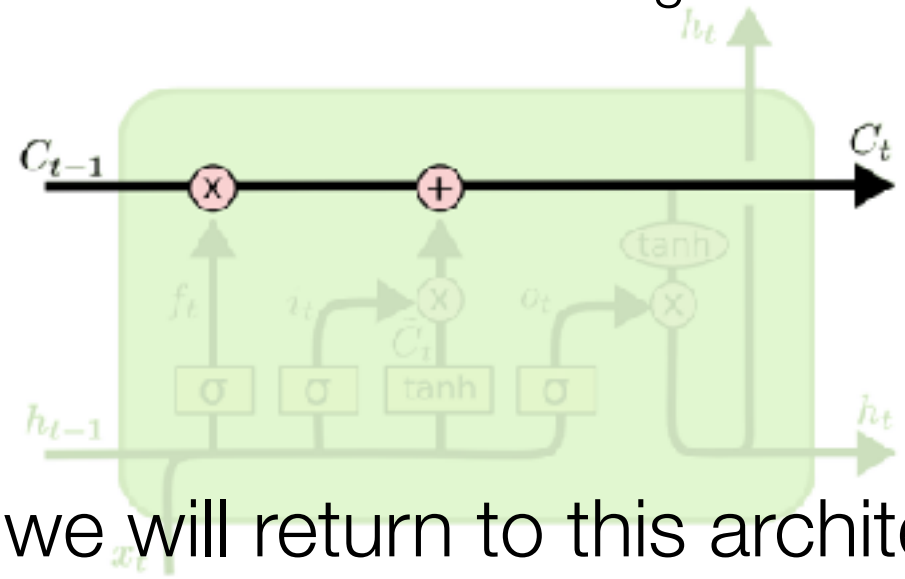


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

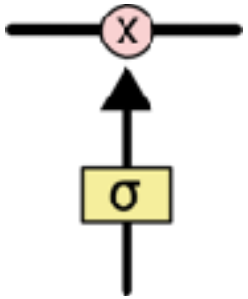
More Advanced Architectures

- **LSTM key idea**: limit how past data can affect output

let **cell state** through

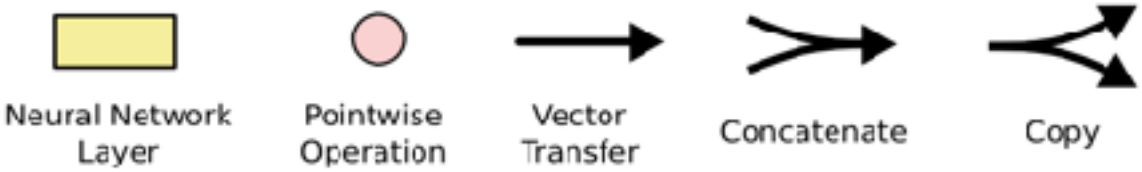


potentially **forget past** inputs via "gate" σ



we will return to this architecture later, for now:

put it in long term memory 😂



http://colah.github.io/posts/2015-08-Understanding-LSTMs/