DEPARTAMENTO DE SEÑALES, SISTEMAS Y RADIOCOMUNICACIONES

Deep Learning Seminar
Day-3

Master of Science in Signal Theory and Communications
TRACK: Signal Processing and Machine Learning for Big Data

Departamento de Señales, Sistemas y Radiocomunicaciones
E.T.S. Ingenieros de Telecomunicación
Universidad Politécnica de Madrid

What's new with Recurrent Nets?

- They allow us to operate over sequences of vectors
- Applications areas are broad:
  - Signals: video, speech, sensors,....
  - Time series: financial series, log messages,..
  - Sequence of characters: Natural Language Processing,...
- RNN can use sequences in the input, the output, or both.

Tenemos entradas salidas y estados

Recurrent Neural Networks (RNN) concepts

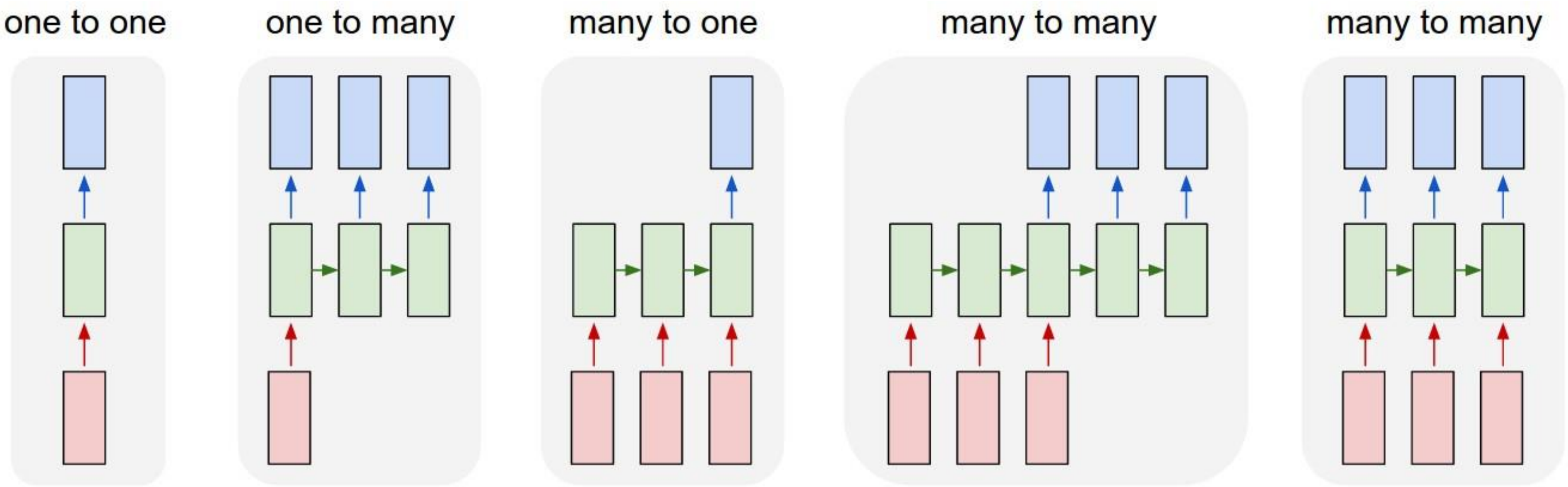


Image description using text

Emotion Recognition

Language Model

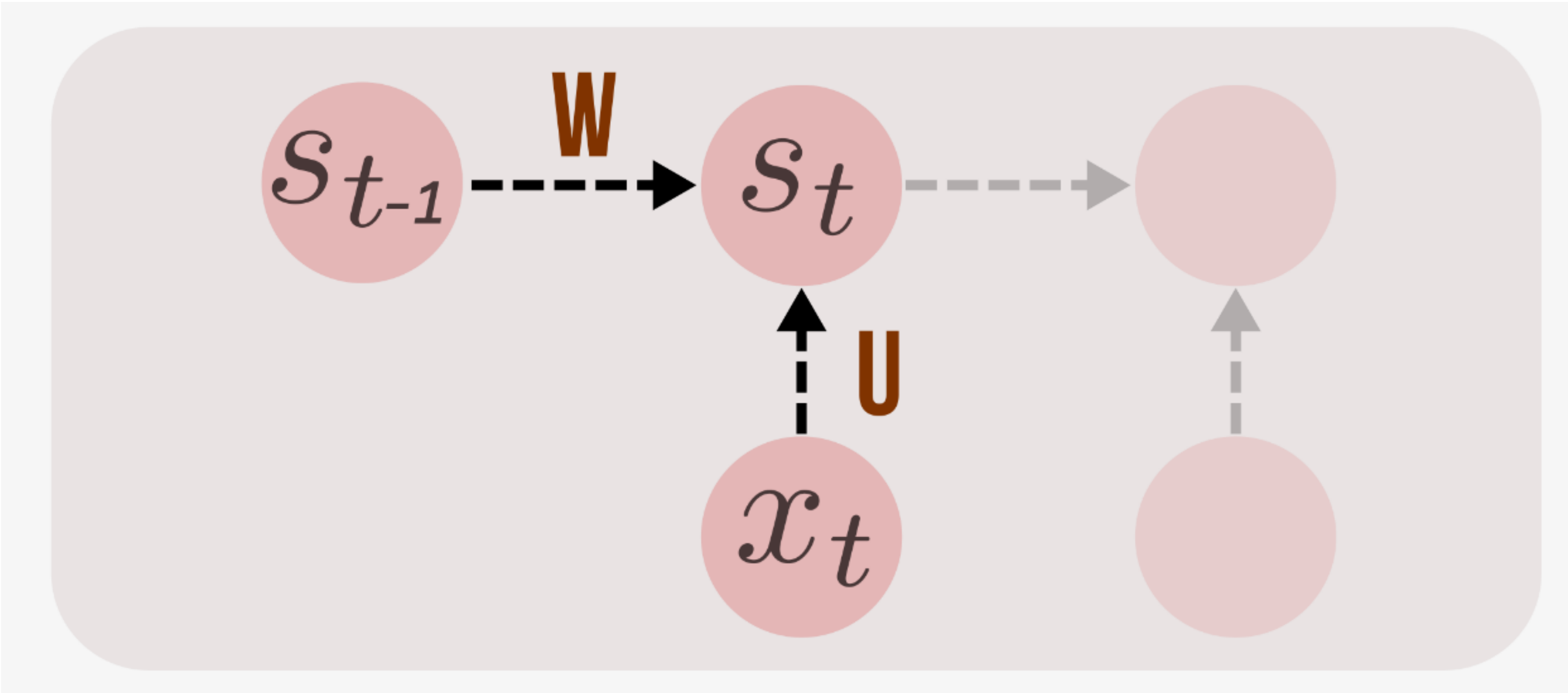
Machine Translation

Video frame labelling

Phoneme Recognition

http://suriyadeepan.github.io/2017-02-13-unfolding-rnn-2/

- RNNs combine the **input vector** with their **state vector** with a fixed (but learned) function to **produce a new state vector**.



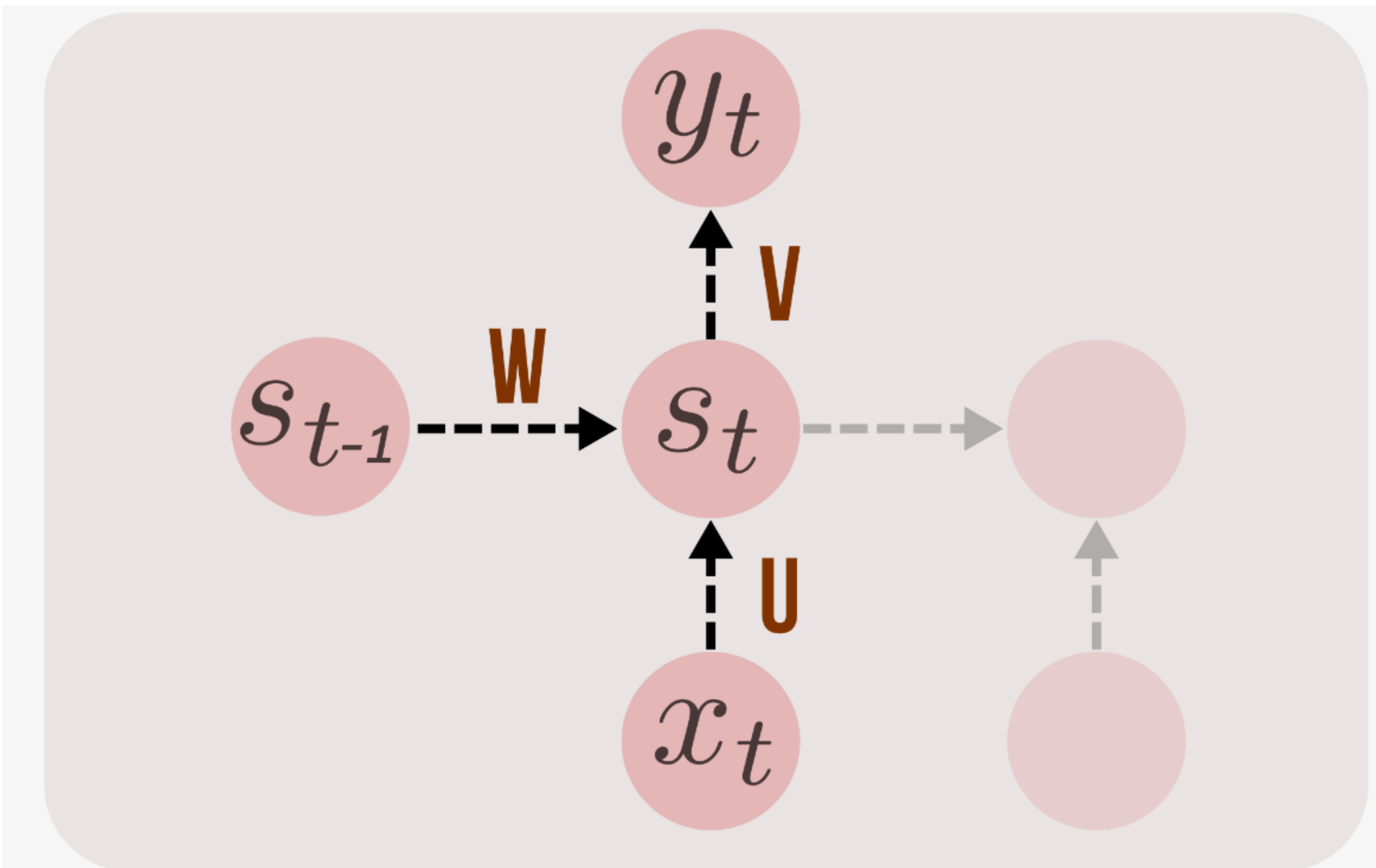
Moving from one state to a new one using conections (W) and activation functions

$$s_t = tanh(Ux_t + Ws_{t-1}),$$

OUTPUT y_t

logits = tf.matmul(state, V) + b

predictions = tf.nn.softmax(logits)



RNN essentially describe programs:

inputs + some internal variables.

- In fact, it is known that [RNNs are Turing-Complete] in the sense that they can to simulate arbitrary programs (with proper weights).

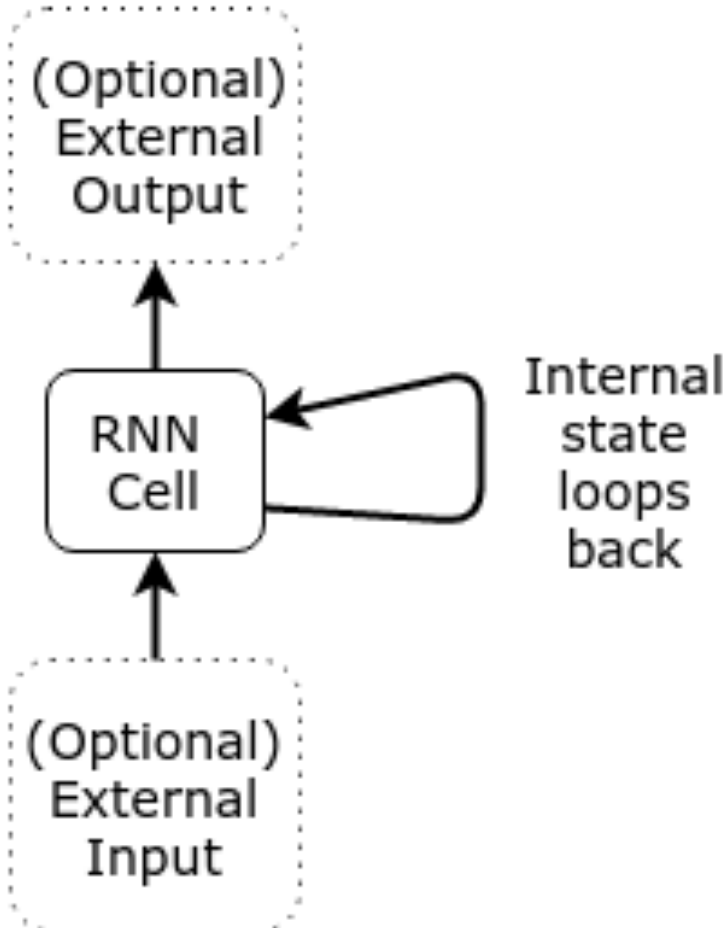
...path to **AI**??

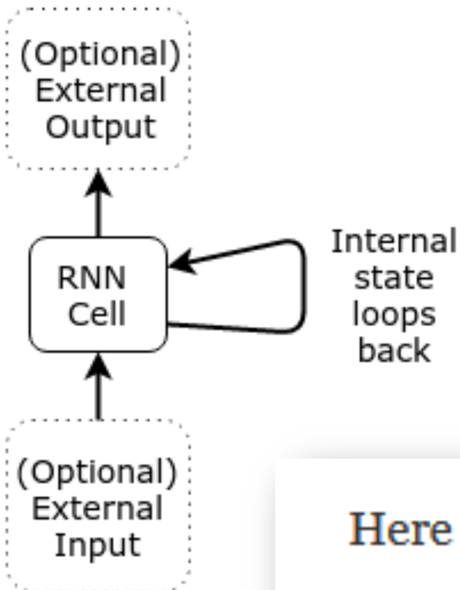
.... But " forget I said anything."

Andrej Karpathy
http://karpathy.github.io/2015/05/21/rnn-effectiveness

Written Memories: Understanding, Deriving and Extending the LSTM

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



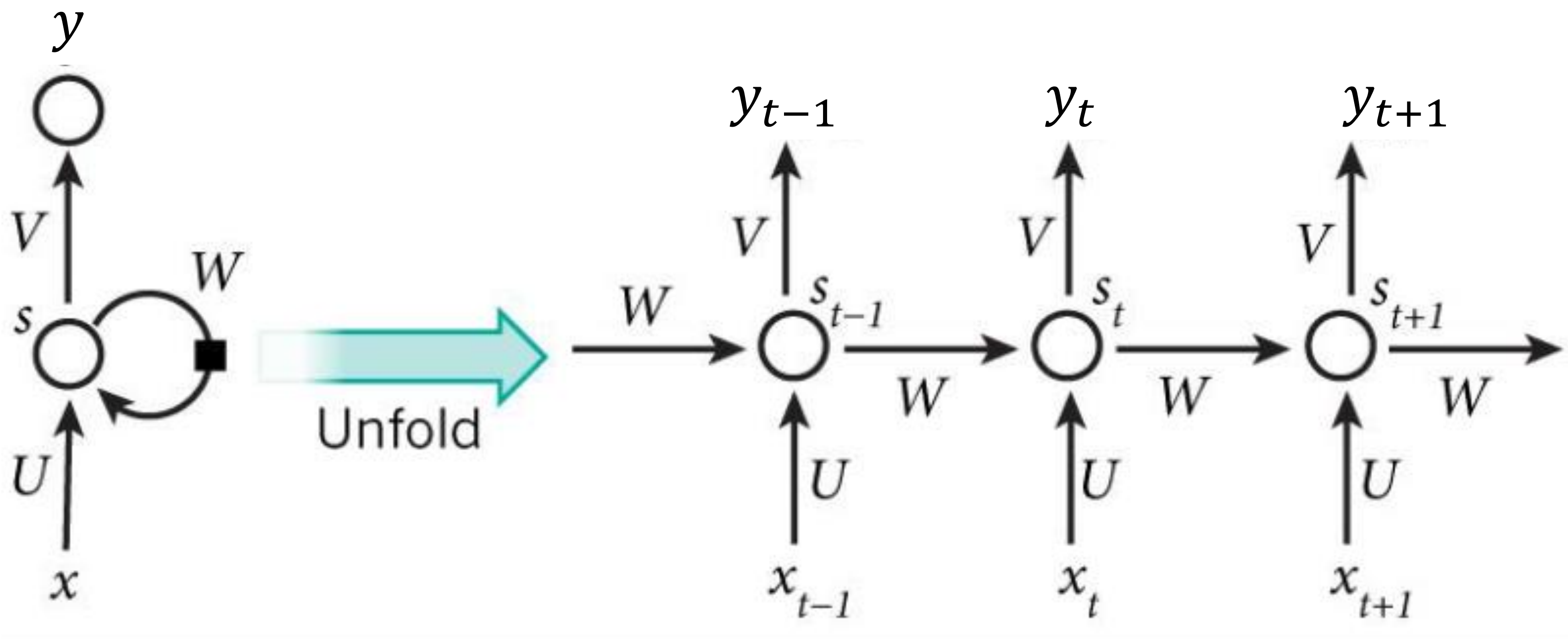


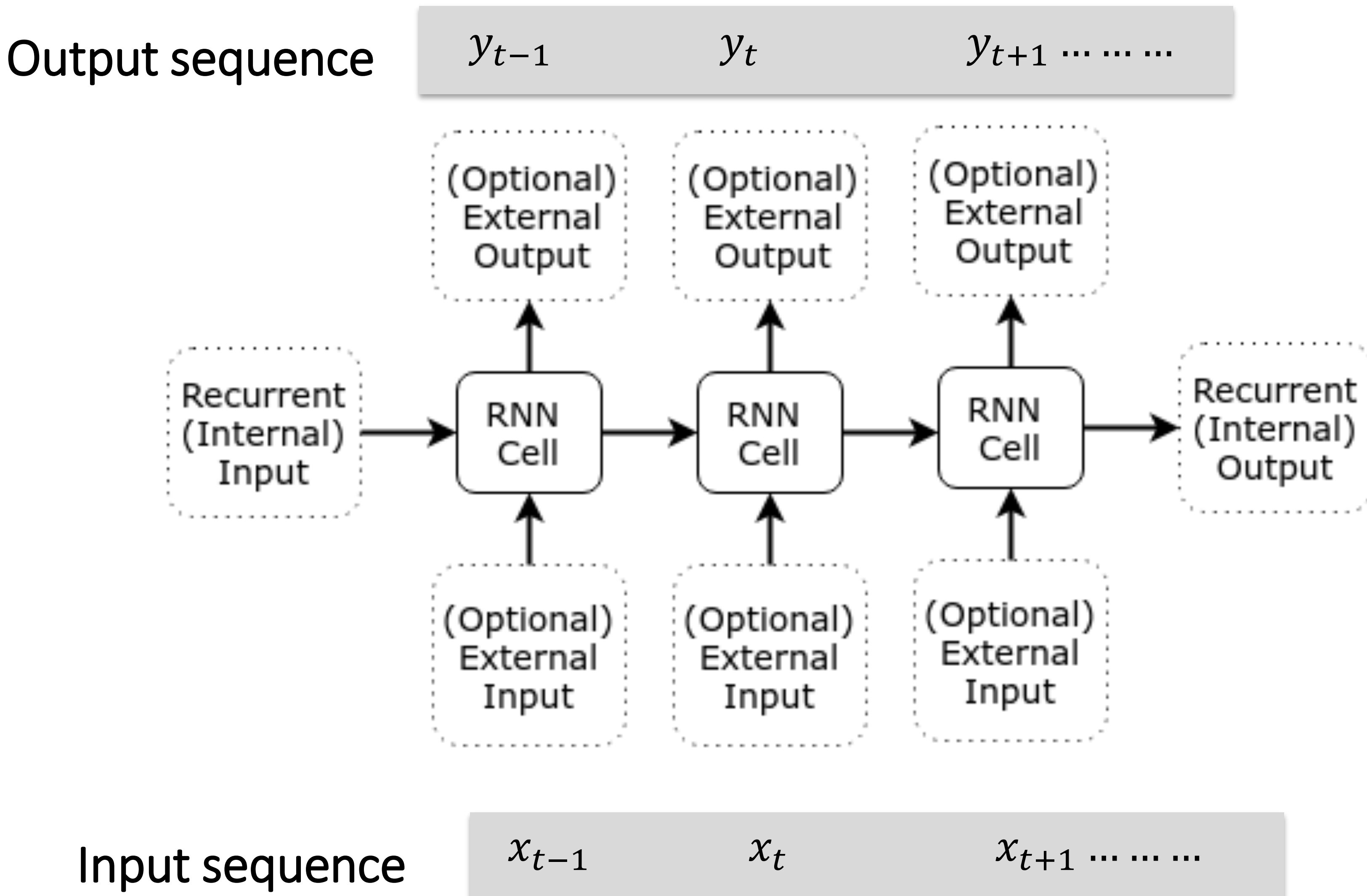
Here is the algebraic description of the RNN cell:

$$\begin{pmatrix} s_t \\ o_t \end{pmatrix} = f \begin{pmatrix} s_{t-1} \\ x_t \end{pmatrix}$$

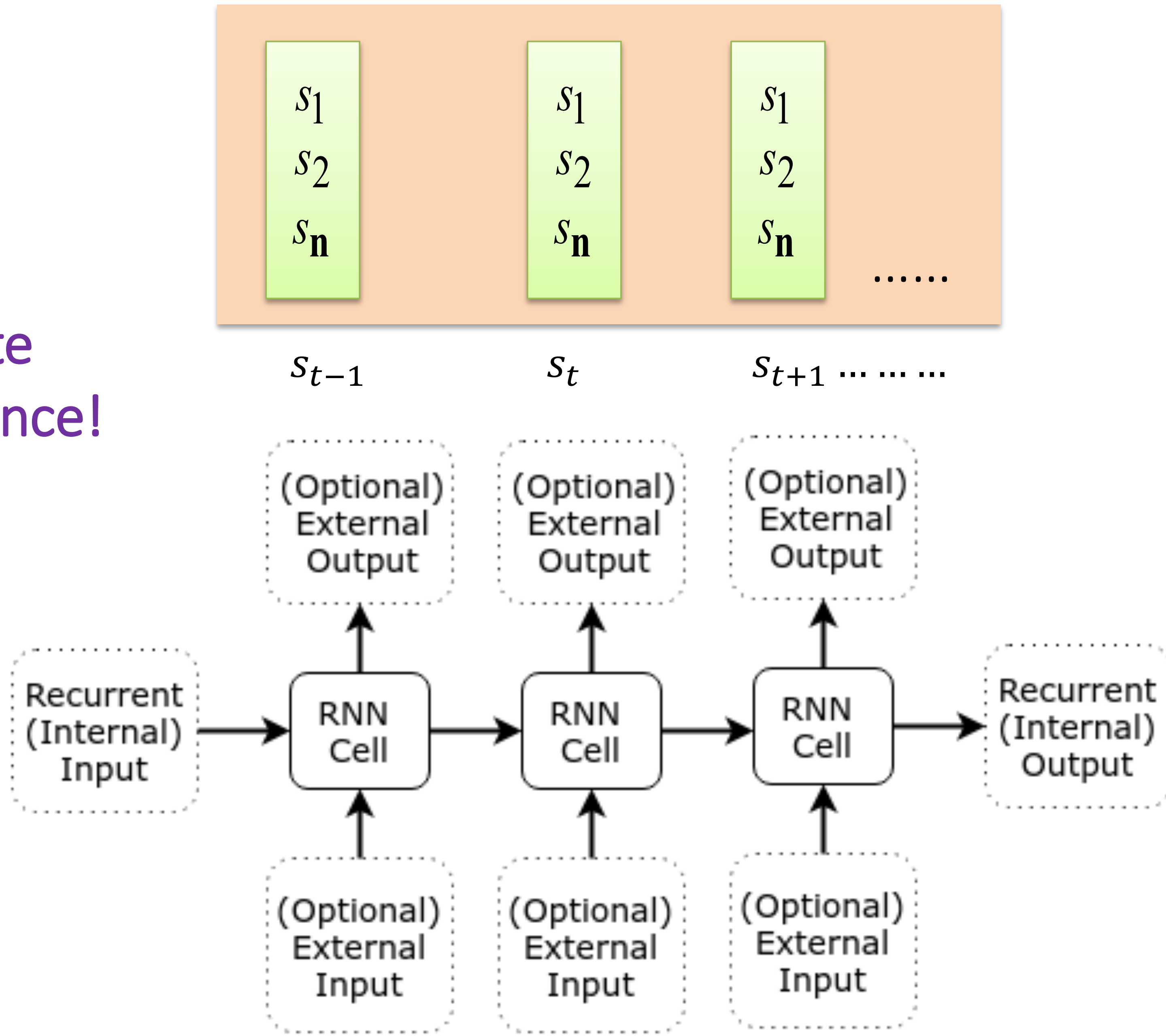
where:

- s_t and s_{t-1} are our current and prior states,
- o_t is our (possibly empty) current output,
- x_t is our (possibly empty) current input, and
- f is our recurrent function.

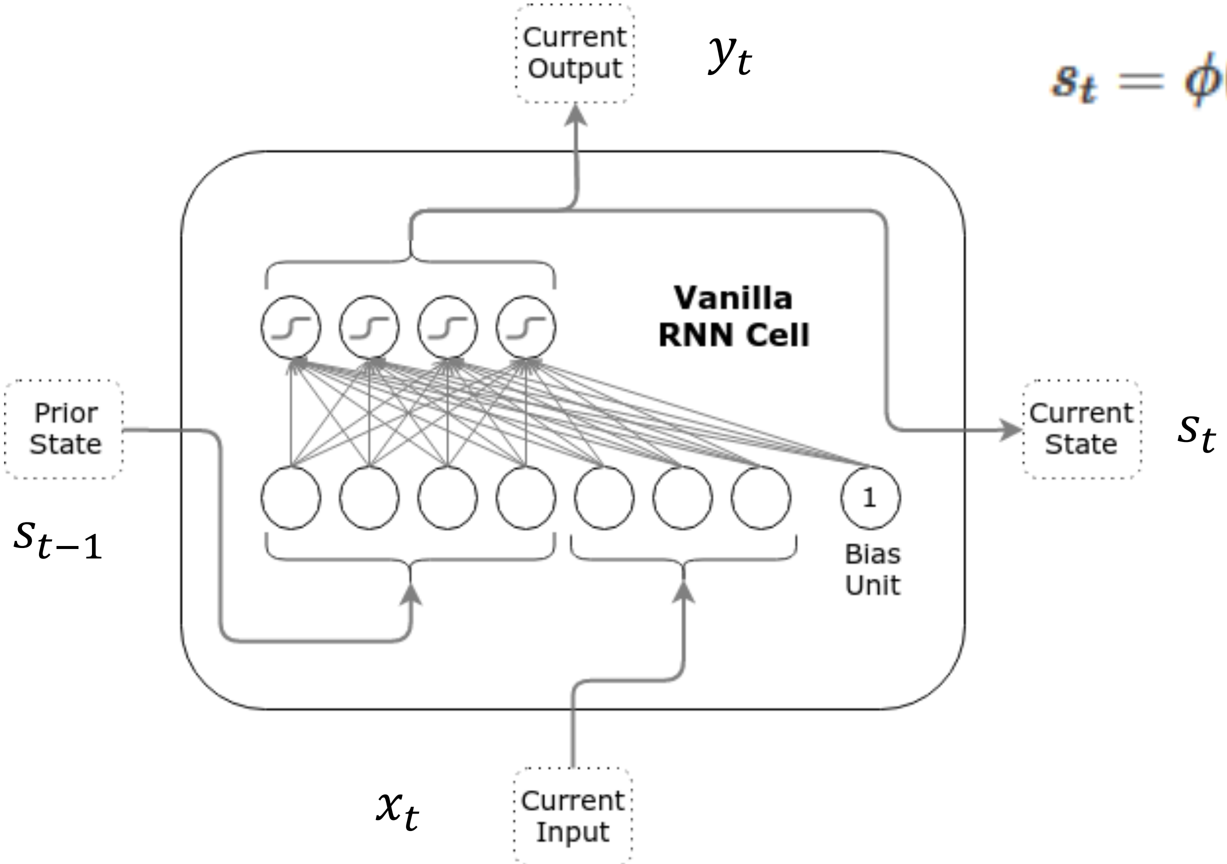

y
V
W
s
U
x
Unfold
y_{t-1}
y_t
y_{t+1}
V
V
V
W
s_{t-1}
s_t
s_{t+1}
W
W
W
U
U
U
x_{t-1}
x_t
x_{t+1}


Output sequence
y_{t-1} y_t y_{t+1} … … …
(Optional) External Output
(Optional) External Output
(Optional) External Output
Recurrent (Internal) Input
RNN Cell
RNN Cell
RNN Cell
Recurrent (Internal) Output
(Optional) External Input
(Optional) External Input
(Optional) External Input
Input sequence
x_{t-1} x_t x_{t+1} … … …

State sequence!



$$s_t = \phi(Ws_{t-1} + Ux_t + b)$$

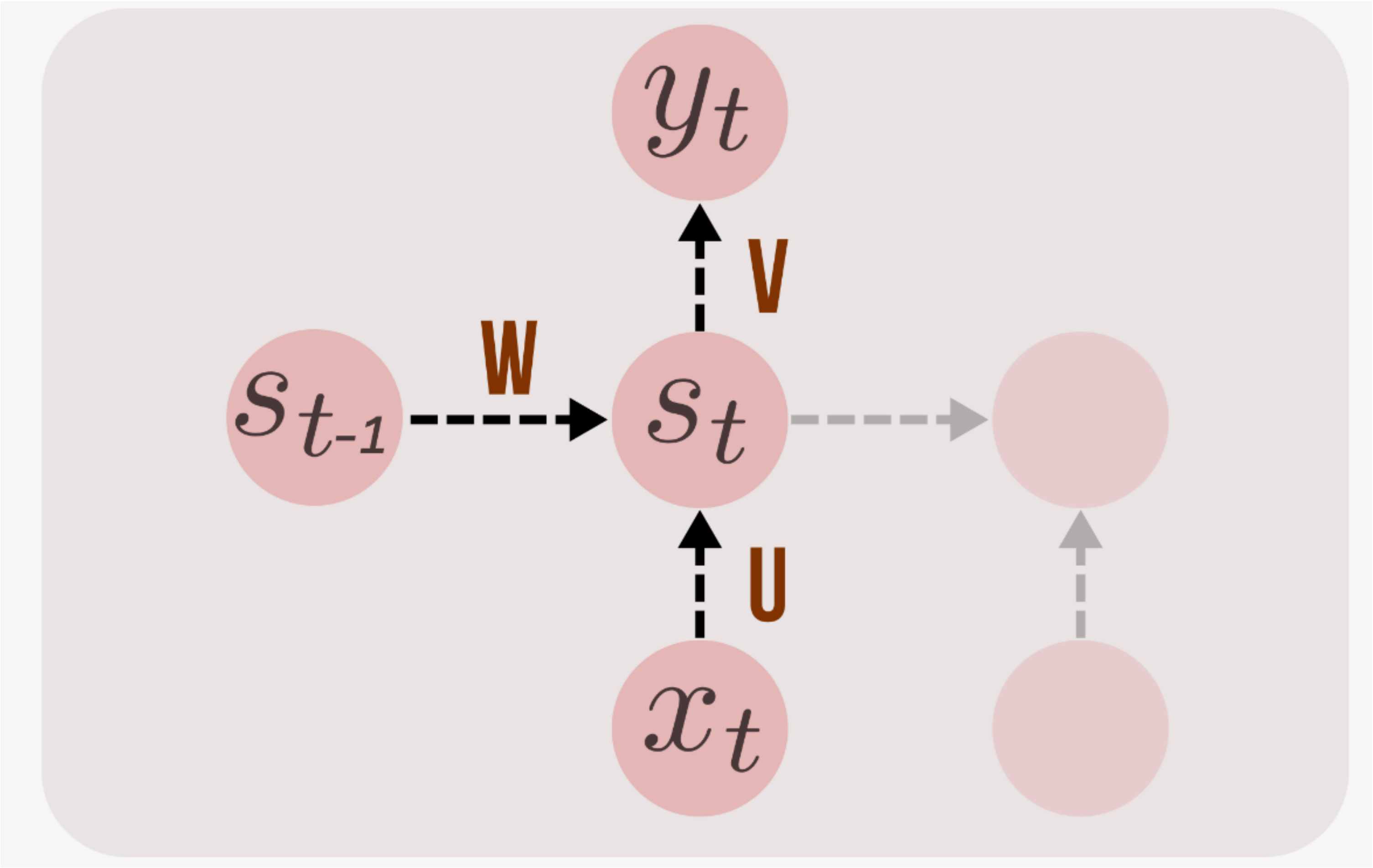


where:

- ϕ is the activation function (e.g., sigmoid, tanh, ReLU),
- $s_t \in \mathbb{R}^n$ is the current state (and current output),
- $s_{t-1} \in \mathbb{R}^n$ is the prior state,
- $x_t \in \mathbb{R}^m$ is the current input,
- $W \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^n$ are the weights and biases, and
- n and m are the state and input sizes.

TRAINING RNN: *W, U, V ?*

- **BPTT:** *Backpropagation Through Time*
- Truncated



Backpropagation Through Time (BPTT)

- Because the parameters are shared by all time steps in the network

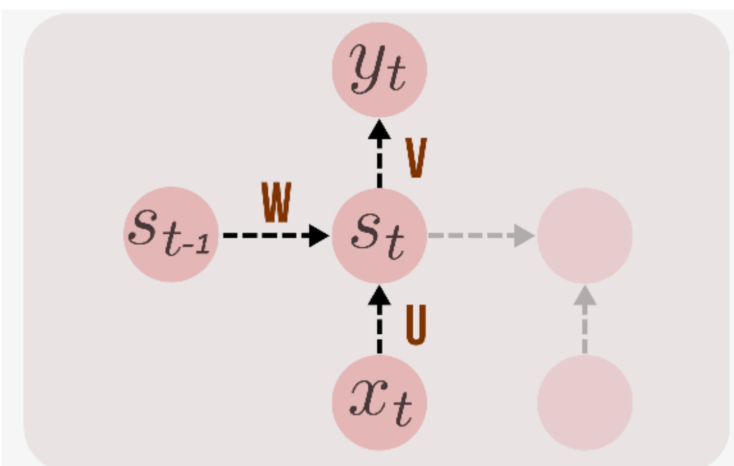
- The gradient at each output depends not only on the calculations of the current time step, but also the previous time steps.

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/

Backpropagation Through Time (BPTT)

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$

Cross entropy loss: $E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) \quad = -\sum_t y_t \log \hat{y}_t$

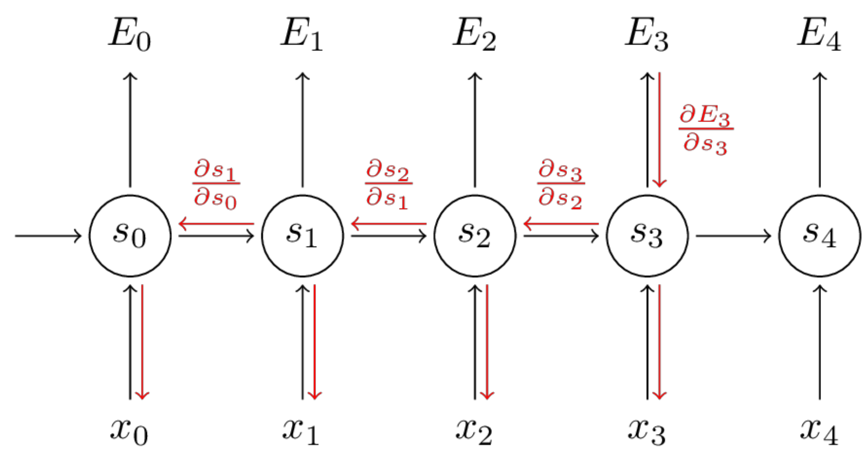


- The output value does depend on the state of the hidden layer,
- which depends on all previous states of the hidden layer
- and thus, all previous inputs

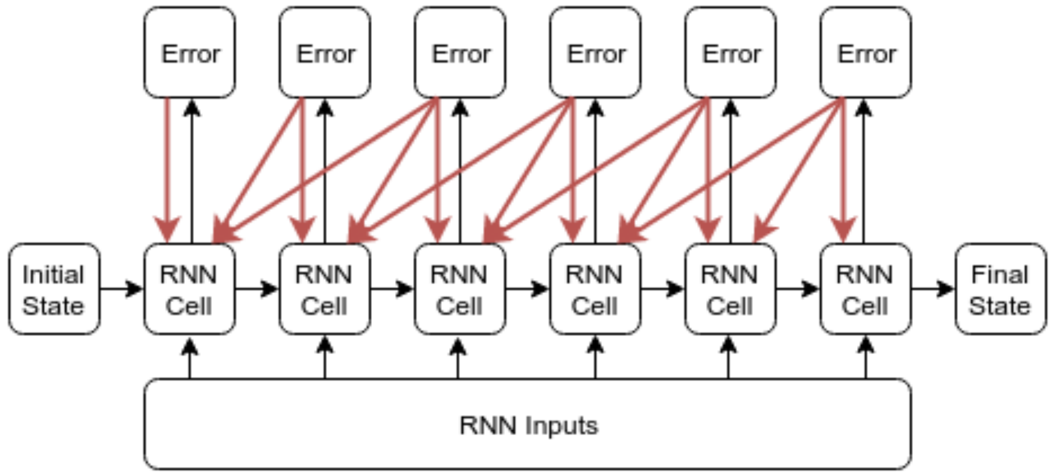
Backpropagation Through Time (BPTT)

- But the recurrent net can be seen as a (very deep) feedforward net with shared weights
  - The forward pass builds up a stack of the activities of all the units at each time step.
  - The backward pass peels activities off the stack to compute the error derivatives at each time step.
  - After the backward pass we add together the derivatives at all the different times for each weight

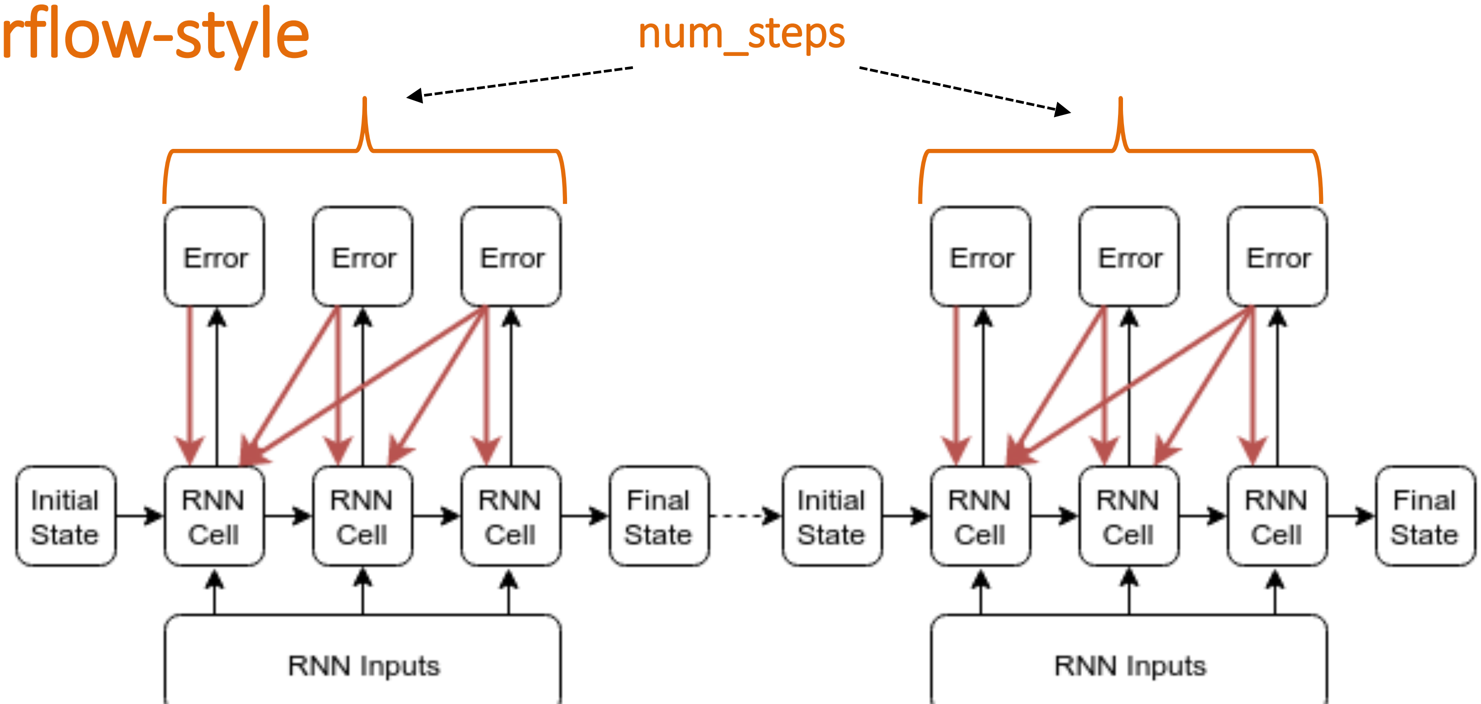
$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial y_y} \frac{\partial y_t}{\partial s_t} \boxed{\frac{\partial s_t}{\partial s_k}} \frac{\partial s_k}{\partial W}$$



Styles of Truncated Backpropagation



Tensorflow-style



Vanishing gradient problem

In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.

Led to the development of **LSTM**s and **GRU**s, two of the currently most popular and powerful models

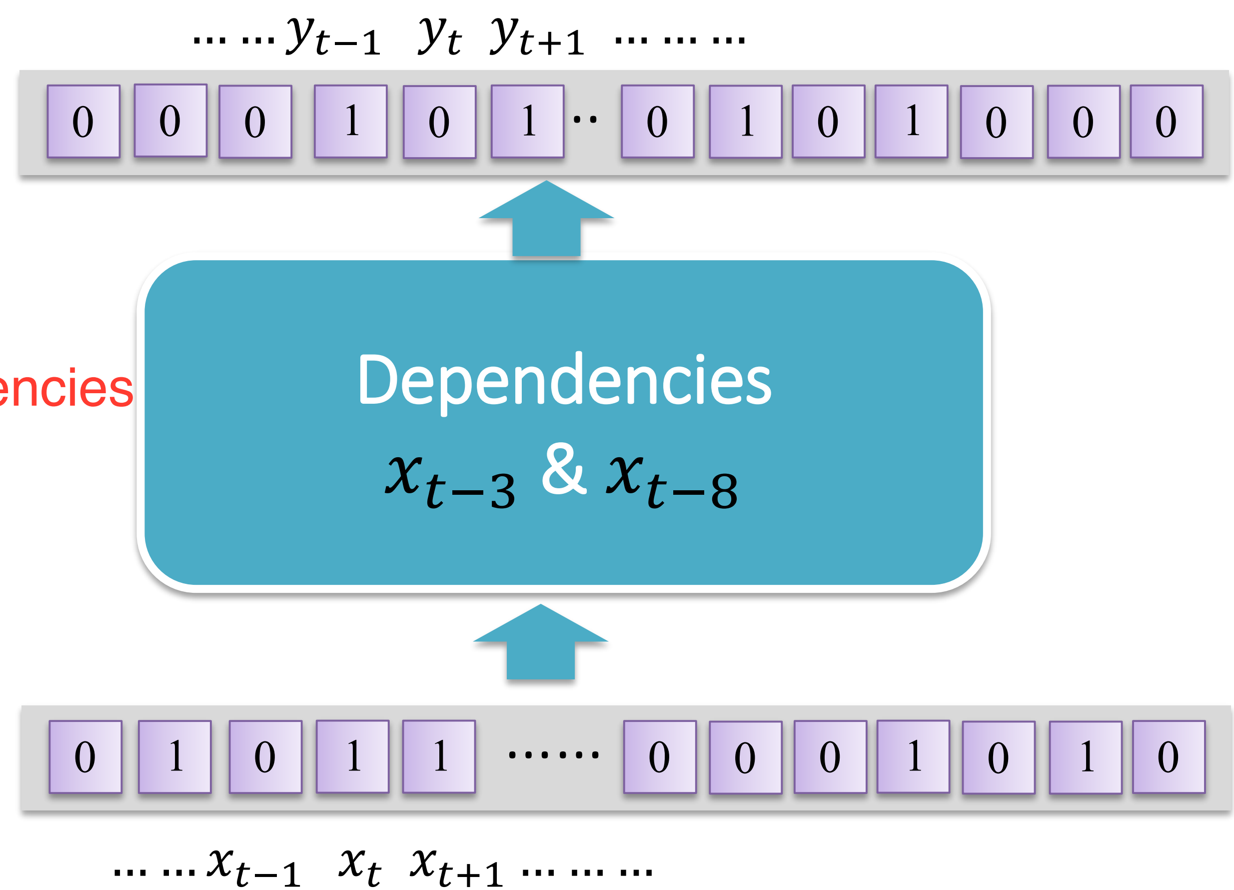
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/

Simple Example: RNN for a Binary Sequence

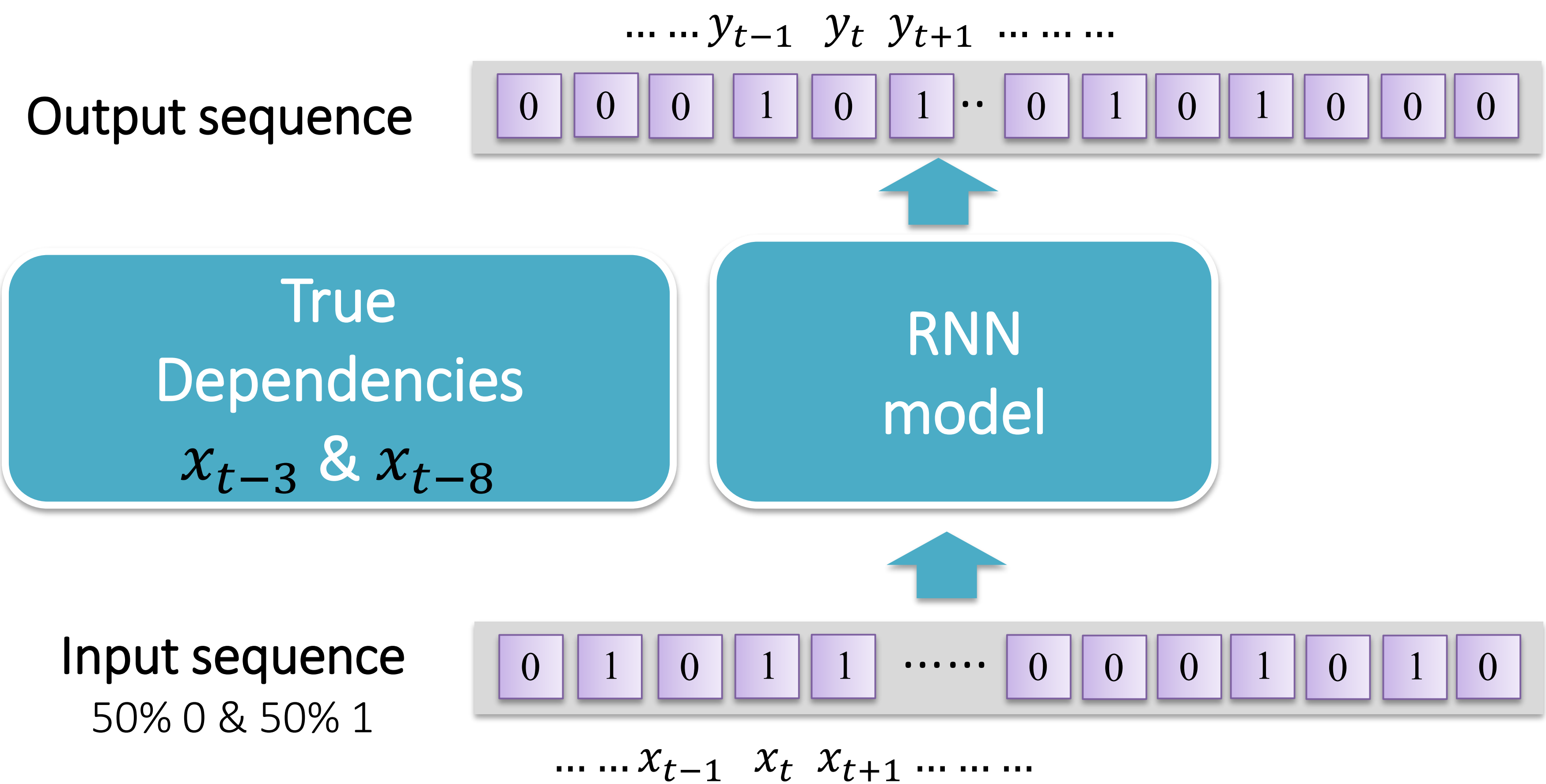
http://r2rt.com/recurrent-neural-networks-in-tensorflow-i.html

Output sequence

Including time dependencies



Input sequence
50% 0 & 50% 1



Expected cross entropy loss if the model:

- learns neither dependency: 0.661563238158
- learns first dependency: 0.519166699707
- learns both dependencies: 0.454454367449

We will start with a BasicRNNCell

MSTC_RNN_1.ipynb

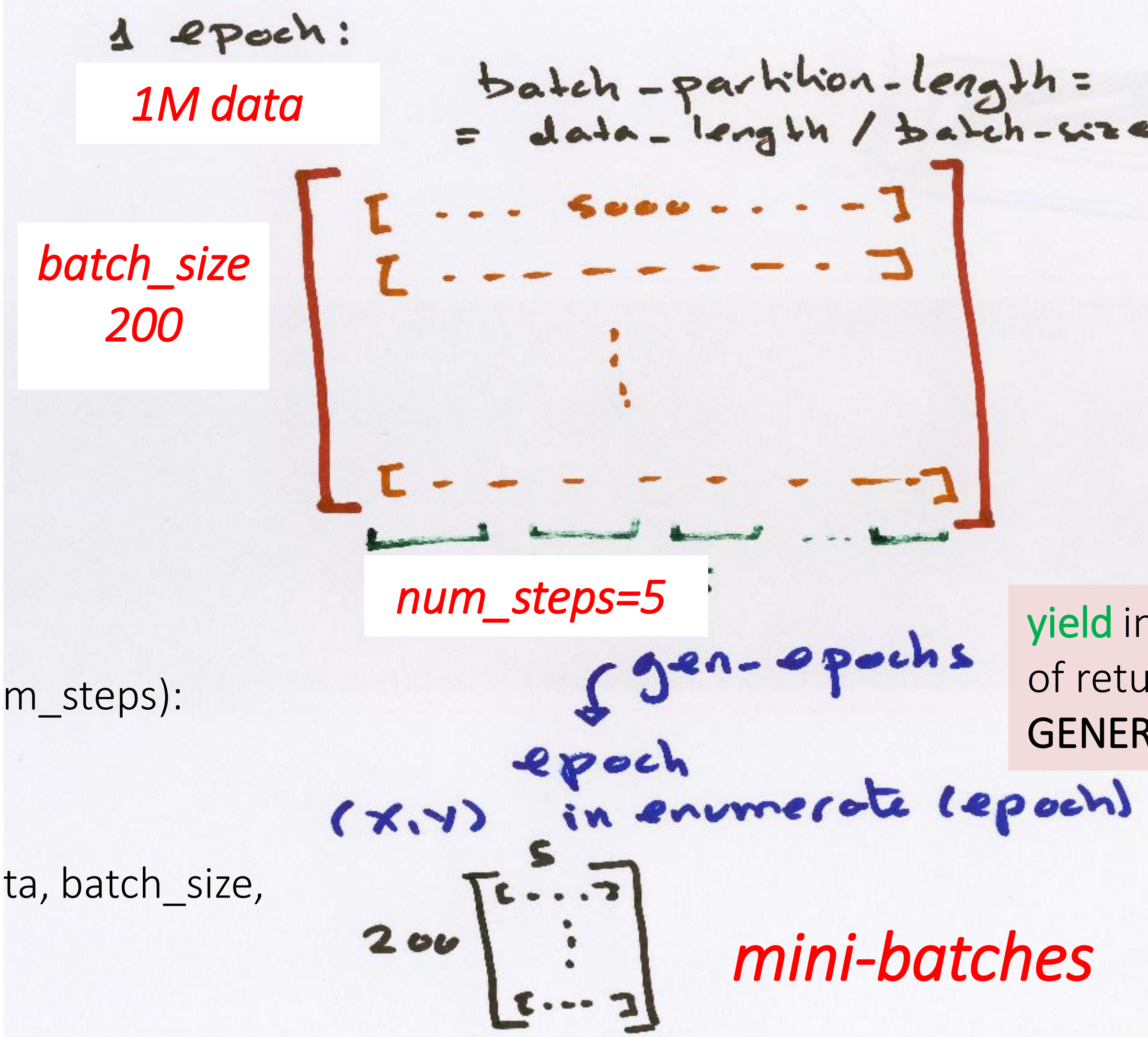
First: dealing with data

- Generate our binary sequences
- Prepare for feeding data into the graph: *from raw data to batches*

Prepare for feeding data into the graph: *from raw data to batches*

- Remember: Neural networks are trained by approximating the gradient of loss function using only a small subset of the data, also known as a mini-batch.

Prepare for feeding data into the graph: *from raw data to batches*

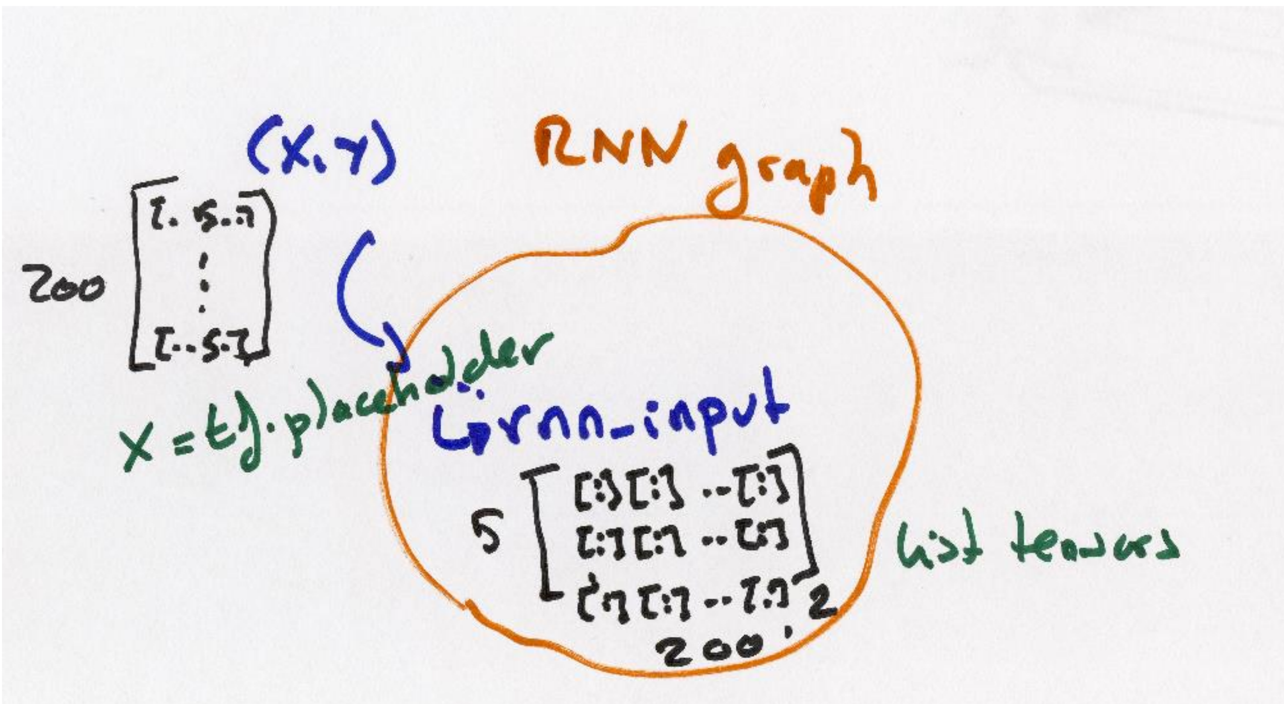


yield instead of return
GENERATOR

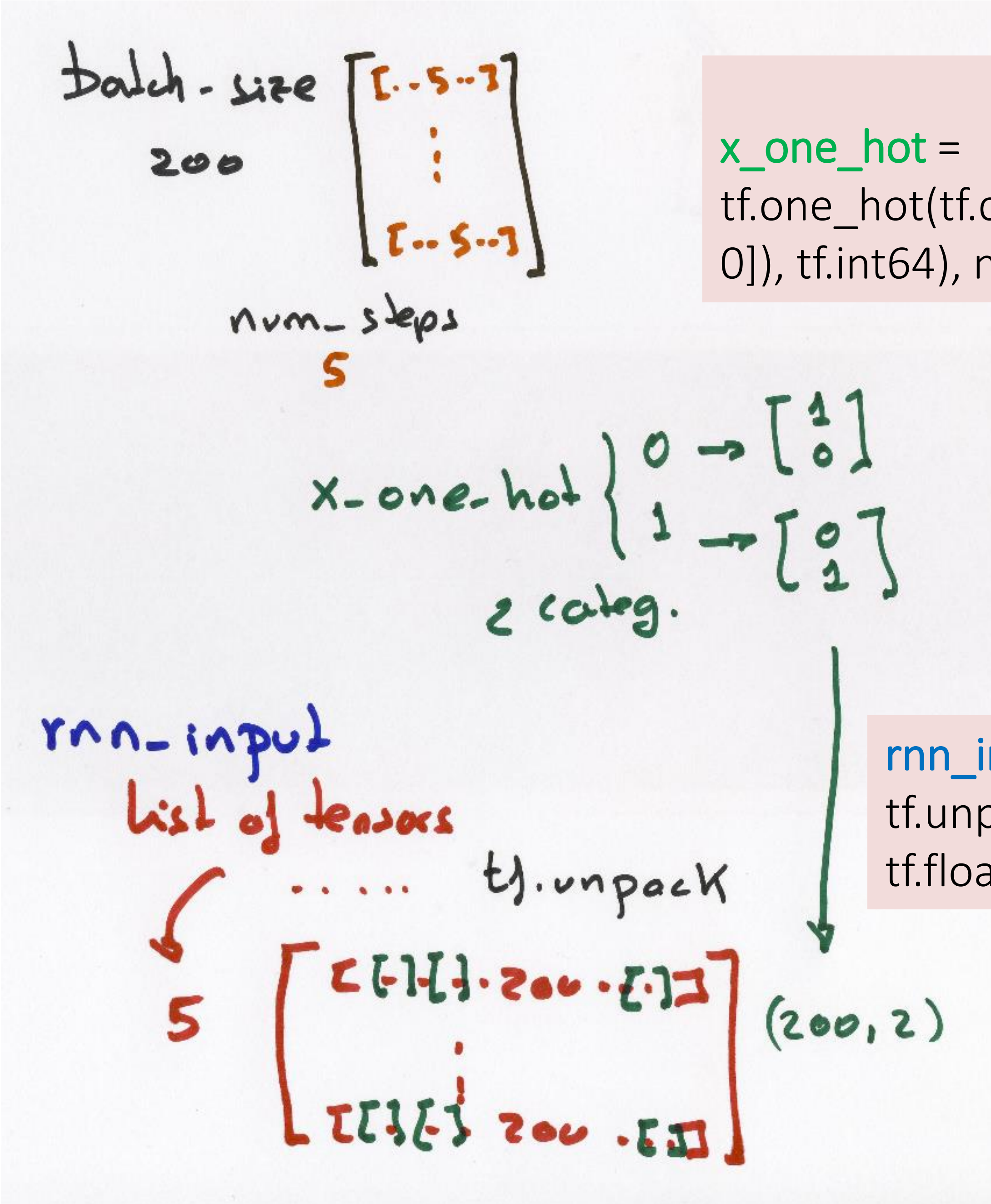
def gen_epochs(n, num_steps):

def gen_batch(raw_data, batch_size, num_steps):

```
def train_network(num_epochs, num_steps, state_size=4, verbose=True):
          .....
          for idx, epoch in enumerate(gen_epochs(num_epochs, num_steps)):
                    for step, (X, Y) in enumerate(epoch):
                              sess.run([losses_last,total_loss,final_state,train_step],
                                        feed_dict={x:X, y:Y, init_state:training_state})
```



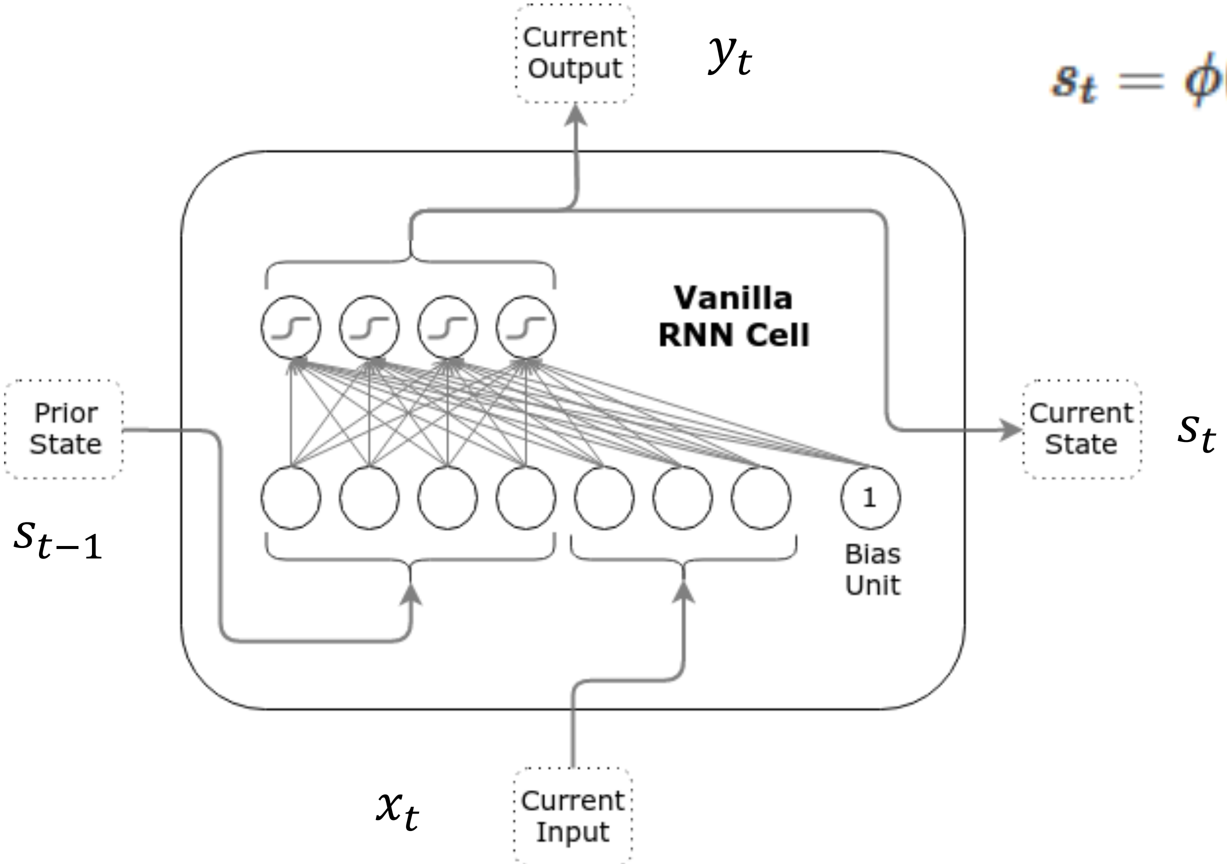

```
x = tf.placeholder(tf.int32, [batch_size, num_steps],
name='input_placeholder')
```




```
x_one_hot = tf.one_hot(tf.cast(tf.transpose(x, perm=[1, 0]), tf.int64), num_classes, 1,0)
```

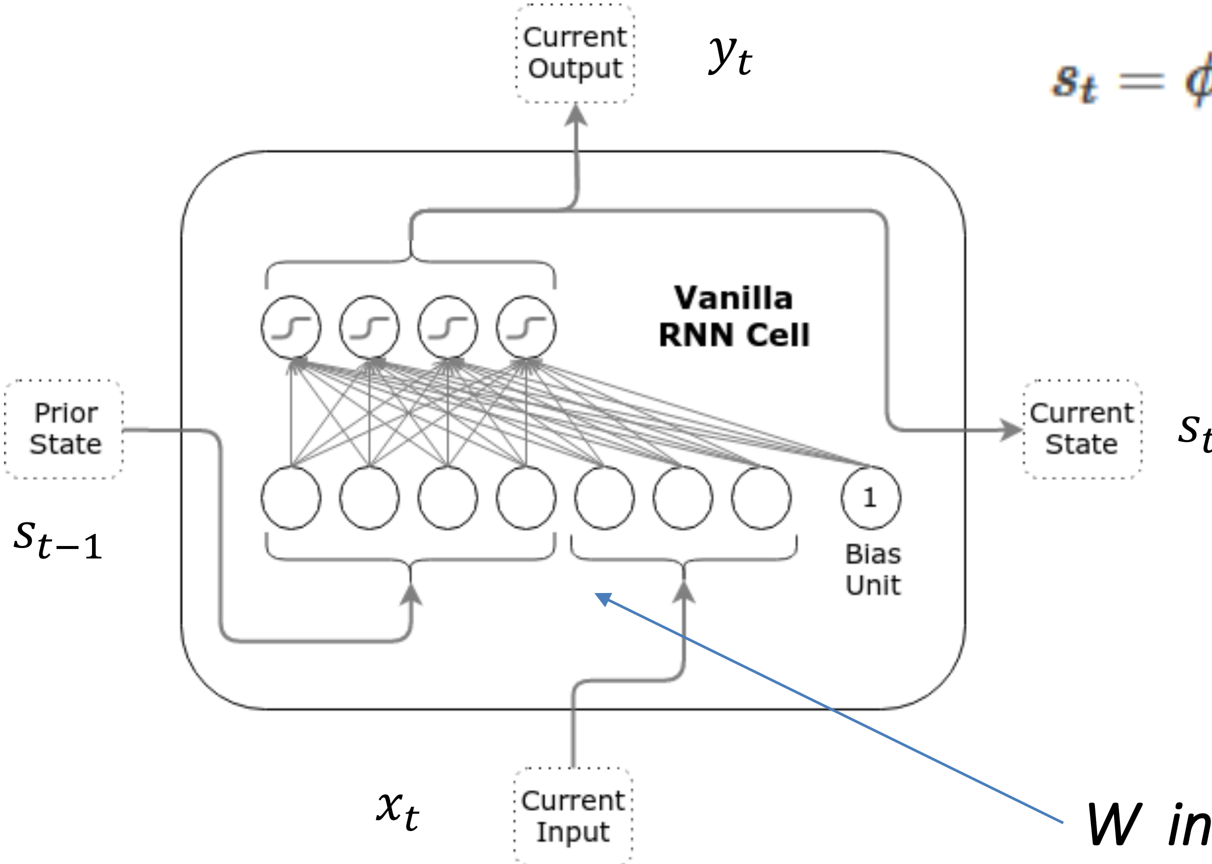
```
rnn_inputs = tf.unpack(tf.cast(x_one_hot, tf.float32))
```

$$s_t = \phi(W s_{t-1} + U x_t + b)$$



where:

- ϕ is the activation function (e.g., sigmoid, tanh, ReLU),
- $s_t \in \mathbb{R}^n$ is the current state (and current output),
- $s_{t-1} \in \mathbb{R}^n$ is the prior state,
- $x_t \in \mathbb{R}^m$ is the current input,
- $W \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^n$ are the weights and biases, and
- n and m are the state and input sizes.



$$s_t = \phi(W s_{t-1} + U x_t + b)$$

W includes W and U

```
Definition of rnn_cell
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)
```

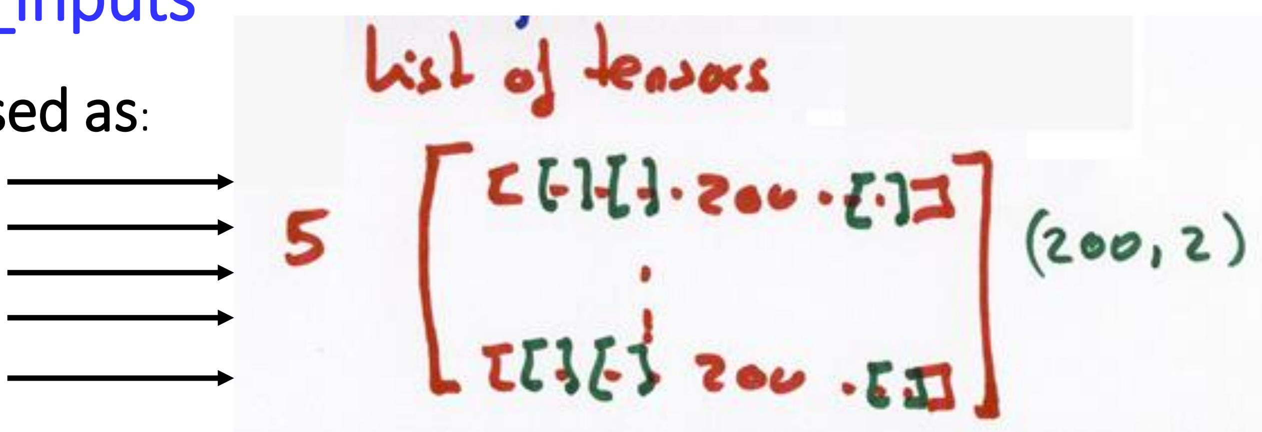
Adding rnn_cells to graph

```
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]
```

rnn_inputs

Processed as:

rnn_input




```
return tf.tanh(tf.matmul(tf.concat(1, [rnn_input, state]), W) + b)
```

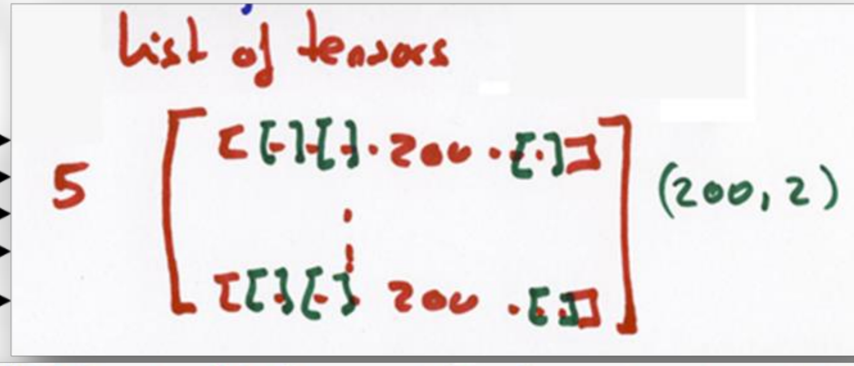
Styles of Truncated Backpropagation

state=
init_state

rnn_inputs

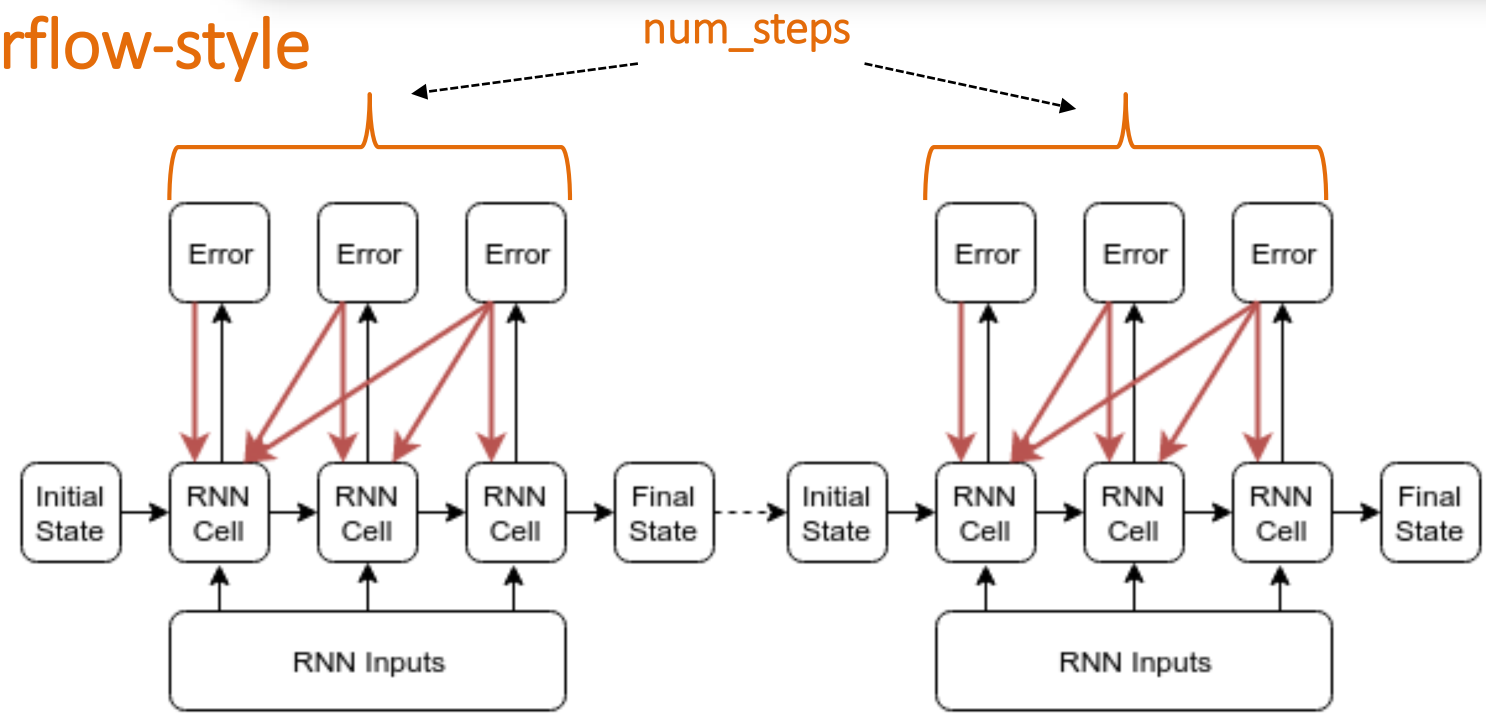
Processed as:

rnn_input

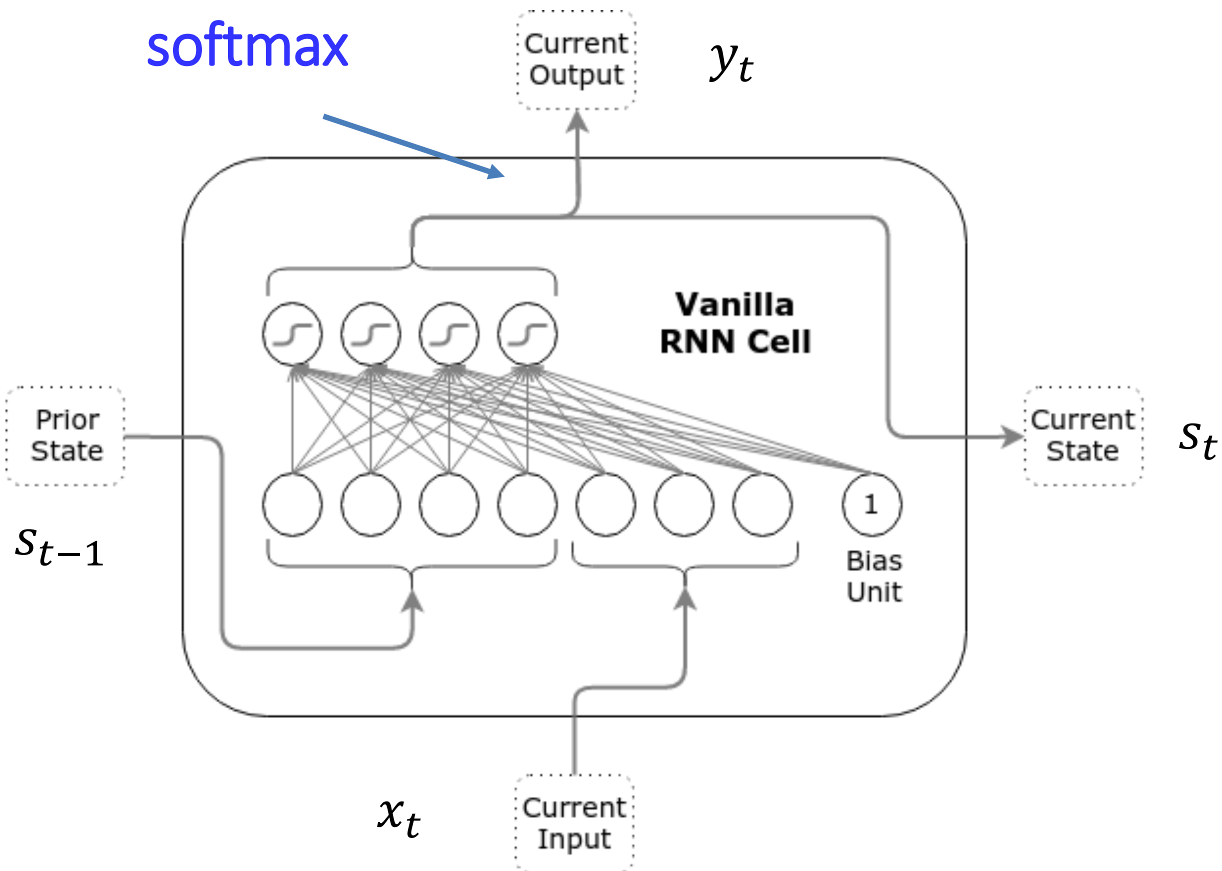



```
return tf.tanh(tf.matmul(tf.concat(1, [rnn_input, state]), W) + b)
```

Tensorflow-style



```
#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]
```



- **Loss function and training step**

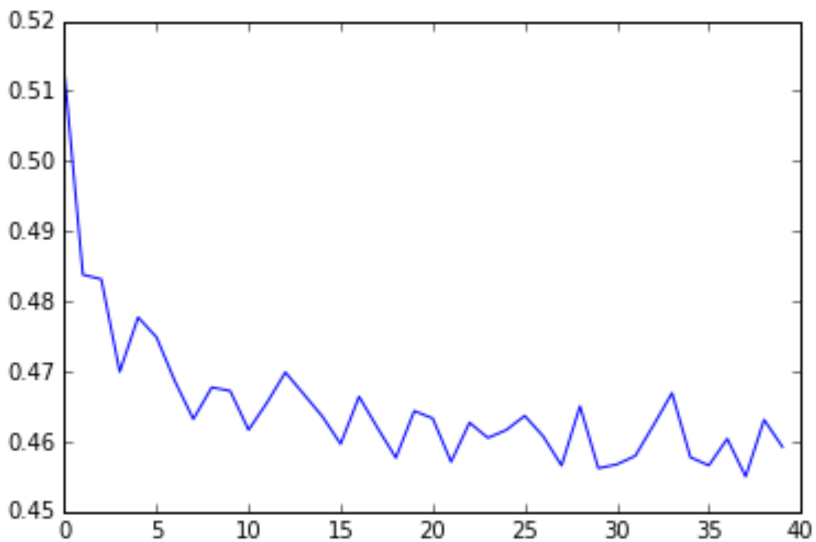
```
# Turn our y placeholder into a list labels
y_as_list = [tf.squeeze(i, squeeze_dims=[1]) for i in tf.split(1, num_steps, y)]

#losses and train_step
losses = [tf.nn.sparse_softmax_cross_entropy_with_logits(logit,label) for \
          logit, label in zip(logits, y_as_list)]
losses_last=losses[-1]
total_loss = tf.reduce_mean(losses[-1])
train_step = tf.train.AdagradOptimizer(learning_rate).minimize(total_loss)
```

Understand your results

- **Plotting training losses**

```
plt.plot(training_losses)
```

```
[<matplotlib.lines.Line2D at 0x7f30b0df9790>]
```



- **Set GLOBAL Configuration Variables**

```
# Global config variables
num_epochs=10
num_steps = 10 # number of truncated backprop steps
batch_size = 200
num_classes = 2
state_size = 16
learning_rate = 0.1
```

Now:

- Translating our model to Tensorflow
- Using a dynamic RNN

MSTC_RNN_2_dynamic.ipynb

Tensorflow RNN static vs. dynamic

- Tensorflow contains two different functions for RNNs: tf.nn.rnn and tf.nn.dynamic_rnn.

- Internally, **tf.nn.rnn** creates an **unrolled static-graph for a fixed RNN length**.
  - First, graph creation is slow.
  - Second, you're unable to pass in longer sequences than you've originally specified.

- **tf.nn.dynamic_rnn** solves this. It uses a tf.While loop to dynamically construct the graph when it is executed.
  - Graph creation is faster;
  - and **you can feed batches of variable size**.

What about performance? dynamic is faster?

In short, just use tf.nn.dynamic_rnn. There is no benefit to tf.nn.rnn and I wouldn't be surprised if it was deprecated in the future.

http://www.wildml.com/2016/08/rnns-in-tensorflow-a-practical-guide-and-undocumented-features/

Tensorflow RNN static vs. dynamic

rnn_inputs are different

- rnn_inputs to **tf.nn.rnn** is a list of tensors

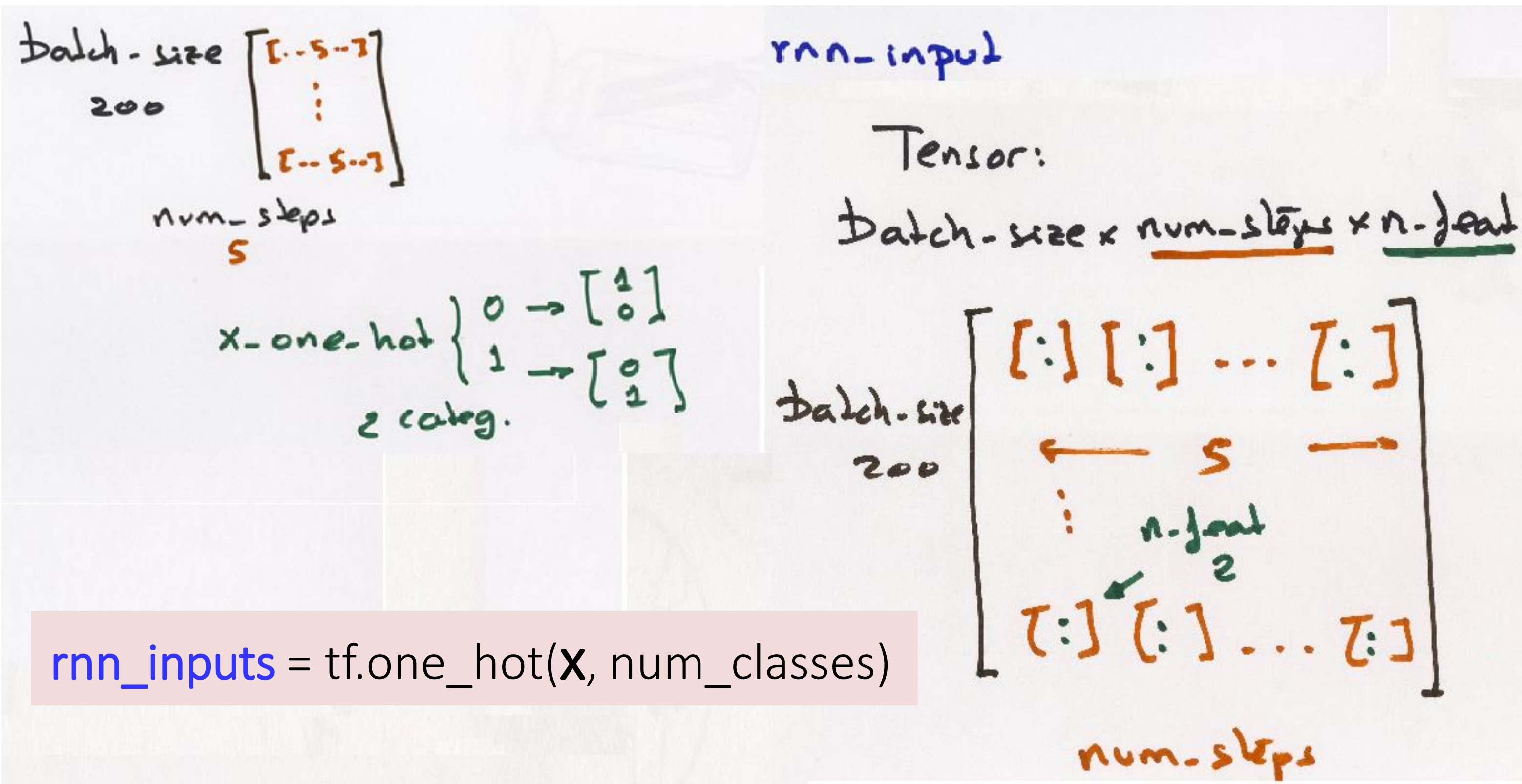
- rnn_inputs **to tf.nn.dynamic_rnn** is:

  A Tensor with dimension
  [batch_size, n_step, n_input]

- rnn_inputs **to tf.nn.dynamic_rnn** is:

A Tensor with dimension
[batch_size, n_step, n_input]

x



rnn_inputs = tf.one_hot(**X**, num_classes)

Translating our model to a BasicRNNCel in Tensorflow's API is easy:

- We simply replace two sections by two lines!!!
- We use: *tf.nn.dynamic_rnn*

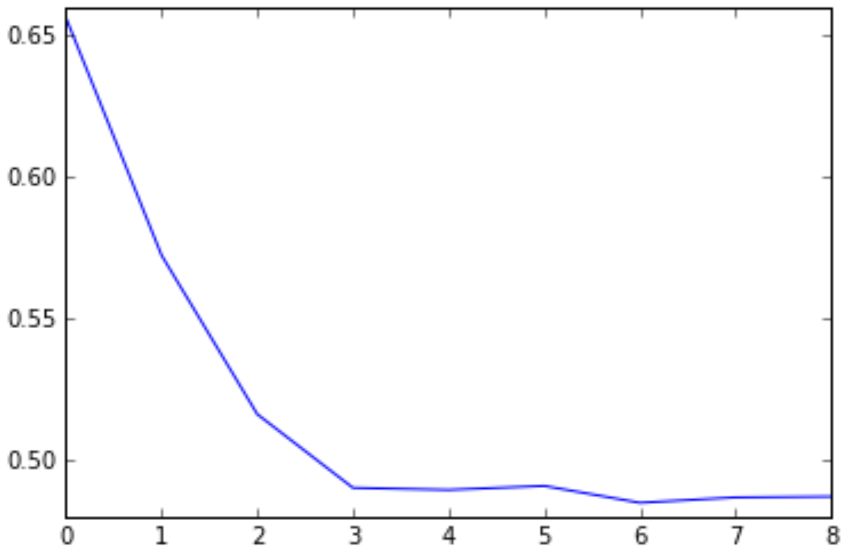
```
"""
Definition of rnn_cell in TENSORFLOW's API
"""


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

Understand your results

- **Plotting training losses**

```
plt.plot(training_losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa6b940ef90>]
```



- **Set GLOBAL Configuration Variables**

```
# Global config variables
num_epochs=1
num_steps = 5 # number of truncated backprop steps
batch_size = 200
num_classes = 2
state_size = 8
learning_rate = 0.1
```

Finally:

- We will use LSTM and GRU

- For a Simple NLP Task:
  - character-level language model to generate character sequences

MSTC_RNN_3.ipynb

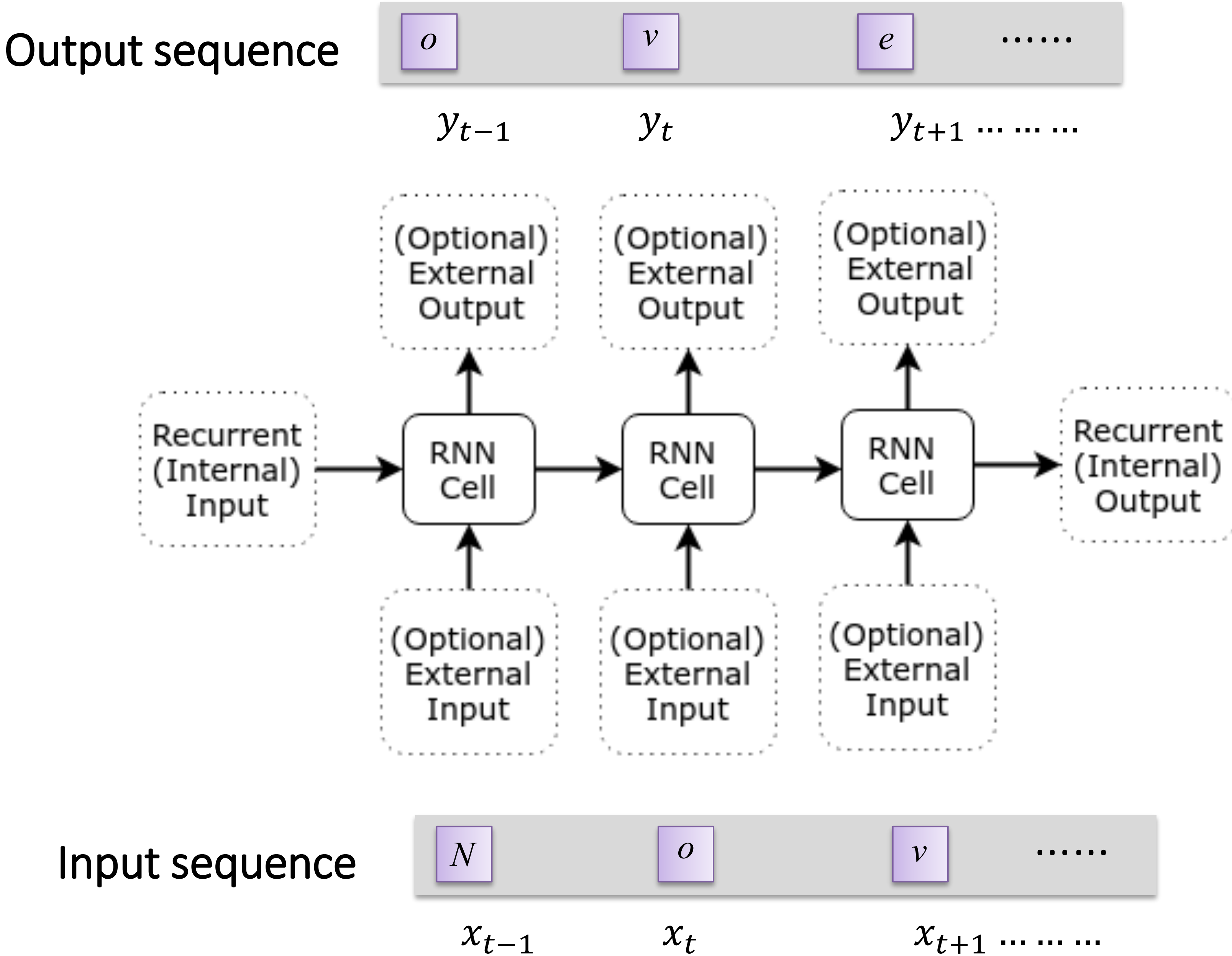
Simplified from:
http://r2rt.com/recurrent-neural-networks-in-tensorflow-ii.html

The NLP Task: language modeling to capture the statistical relationship among words **/ chars**

The objective of the network is to observe the input *x*, one character at a time and produce the output sentence *y*, one character at a time

X (string)	**Y (string)**
November 2016If you'	ovember 2016If you'r
re a California vote	e a California voter
r, there is an impor	, there is an import
tant proposition on	ant proposition on y
your ballot this yea	our ballot this year

http://suriyadeepan.github.io/2017-02-13-unfolding-rnn-2/


Output sequence
o
v
e
……
y_{t-1}
y_t
y_{t+1} … … …
(Optional) External Output
(Optional) External Output
(Optional) External Output
Recurrent (Internal) Input
RNN Cell
RNN Cell
RNN Cell
Recurrent (Internal) Output
(Optional) External Input
(Optional) External Input
(Optional) External Input
Input sequence
N
o
v
……
x_{t-1}
x_t
x_{t+1} … … …

For this NLP Task:

- We read a txt file: *tinyshakespeare.txt , cervantes.txt*

- Read all characters = **vocab** and generating a mapping from chars to numbers **vocab_to_idx** and viceversa **idx_to_vocab**

```
: print vocab

set(['\n', '!', ' ', '$', "'", '&', '-', ',', '.', '3', ';', ':', '?', 'A', 'C', 'B', 'E',
'Q', 'P', 'S', 'R', 'U', 'T', 'W', 'V', 'Y', 'X', 'Z', 'a', 'c', 'b', 'e', 'd', 'g', 'f',
'r', 'u', 't', 'w', 'v', 'y', 'x', 'z'])

: vocab_size

: 65
```

```
raw_data[0:10]
```

'First Citi'

vocab_to_idx

```
data[0:10]
```

[19, 46, 57, 56, 59, 2, 14, 46, 59, 46]

idx_to_vocab

```
recover_data = [idx_to_vocab[c] for c in data]
```

```
recover_data[0:10]
```

['F', 'i', 'r', 's', 't', ' ', 'C', 'i', 't', 'i']

Epochs and Batches are generated in a similar way as in MSTC_RNN_1:

- PTB_iterator from *Penn Tree Bank (PTB)* dataset is used

Example for: num_steps=200 batch_size=32

X (string)	Y (string)
November 2016If you'	ovember 2016If you'r
re a California vote	e a California voter
r, there is an impor	, there is an import
tant proposition on	ant proposition on y
your ballot this yea	our ballot this year

```
X information....
(32, 200)
<type 'numpy.ndarray'>
[[19 46 57 ..., 62 52 58]
 [45 52 57 ..., 50 39 62]
 [ 2 14 52 ..., 19 46 57]
 ...,
 [52 58  2 ..., 58 56 55]
 [ 2 47 52 ..., 45  2 60]
 [ 2 45 52 ..., 52 56 42]]
```

```
Y information....
(32, 200)
<type 'numpy.ndarray'>
[[46 57 56 ..., 52 58  2]
 [52 57  2 ..., 39 62  2]
 [14 52 57 ..., 46 57 56]
 ...,
 [58  2 60 ..., 56 55 46]
 [47 52 60 ...,  2 60 52]
 [45 52 52 ..., 56 42  2]]
```

Embedings

```
-
X information....
(32, 200)
<type 'numpy.ndarray'>
[[19 46 57 ..., 62 52 58]
 [45 52 57 ..., 50 39 62]
 [ 2 14 52 ..., 19 46 57]
 ...,
 [52 58  2 ..., 58 56 55]
 [ 2 47 52 ..., 45  2 60]
 [ 2 45 52 ..., 52 56 42]]
```

- Indices by themselves, carry no semantic meaning
- This is where embedding comes in; more commonly known as *word vector* or *word embedding*.

http://sebastianruder.com/word-embeddings-1/

Embedings

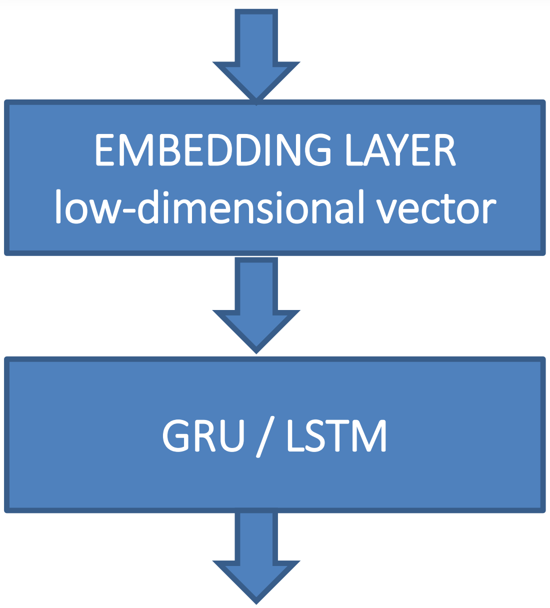
```
-
X information....
(32, 200)
<type 'numpy.ndarray'>
[[19 46 57 ..., 62 52 58]
 [45 52 57 ..., 50 39 62]
 [ 2 14 52 ..., 19 46 57]
 ...,
 [52 58  2 ..., 58 56 55]
 [ 2 47 52 ..., 45  2 60]
 [ 2 45 52 ..., 52 56 42]]
```

In this case, we will **map the characters to low dimensional vectors of size state_size.**

low-dimensional vector
(state_size = n_inputs)

http://sebastianruder.com/word-embeddings-1/

```
X information....
(32, 200)
<type 'numpy.ndarray'>
[[19 46 57 ..., 62 52 58]
 [45 52 57 ..., 50 39 62]
 [ 2 14 52 ..., 19 46 57]
 ...,
 [52 58  2 ..., 58 56 55]
 [ 2 47 52 ..., 45  2 60]
 [ 2 45 52 ..., 52 56 42]]
```



http://stackoverflow.com/questions/40184537/what-does-embedding-do-in-tensorflow

In this example embedings are done by:

- creating an embedding matrix of shape
  [vocab_size=num_classes, state_size]

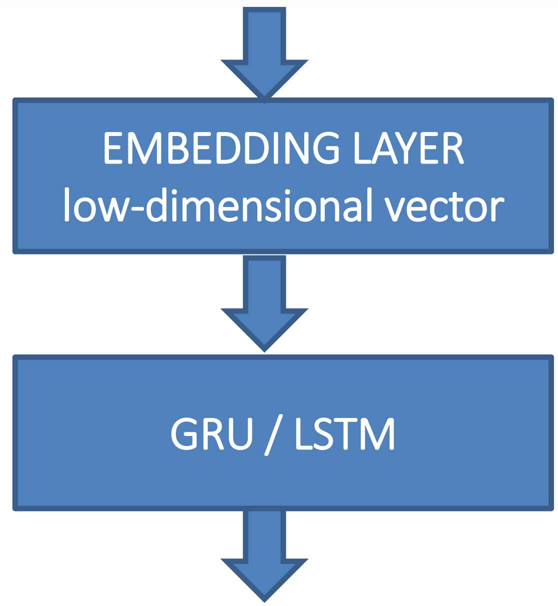
- and selecting a row of the matrix by index of character

```
embeddings = tf.get_variable('embedding_matrix',
[num_classes, state_size])

# Note that our inputs are no longer a list, but a tensor of dims
batch_size x num_steps x state_size  we will use DYNAMIC
RNN
rnn_inputs = tf.nn.embedding_lookup(embeddings, x)
```

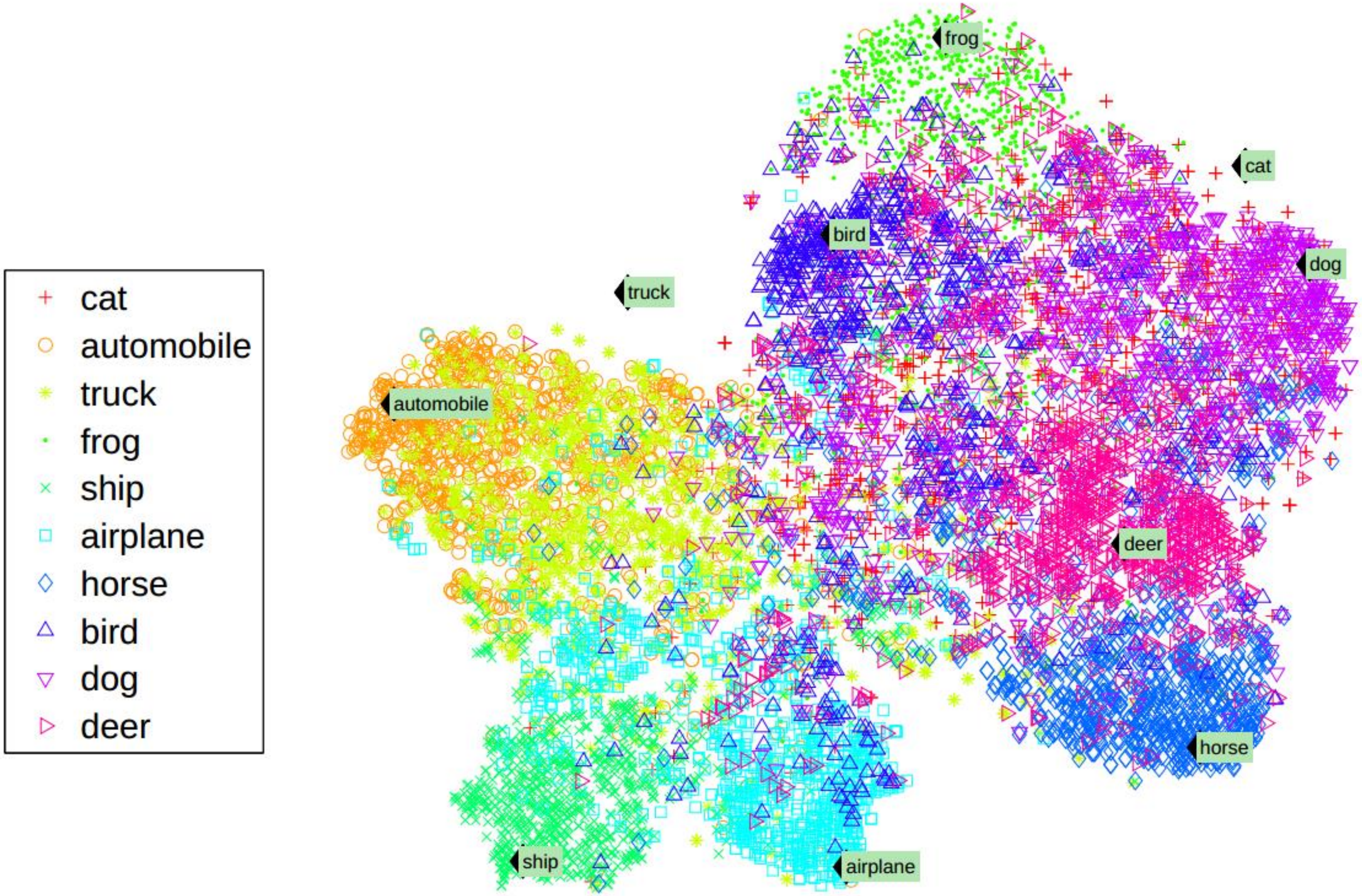
http://sebastianruder.com/word-embeddings-1/

- The embedding_matrix is a “layer” so Tensorflow does backprop of the error INTO this lookup table,
- and hopefully what starts off as a randomly initialized dictionary will gradually become “significant”

```
X information....
(32, 200)
<type 'numpy.ndarray'>
[[19 46 57 ..., 62 52 58]
 [45 52 57 ..., 50 39 62]
 [ 2 14 52 ..., 19 46 57]
 ...,
 [52 58  2 ..., 58 56 55]
 [ 2 47 52 ..., 45  2 60]
 [ 2 45 52 ..., 52 56 42]]
```



http://stackoverflow.com/questions/40184537/what-does-embedding-do-in-tensorflow



http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/

See also:

Word2vec https://www.tensorflow.org/tutorials/word2vec

Vector space models (VSMs) represent (embed) words in a continuous vector space where **semantically similar words** are mapped to nearby points ('are embedded nearby each other')

Now we can test using LSTM:

```
#LSTM
cell = tf.nn.rnn_cell.LSTMCell(state_size,
state_is_tuple=True)
cell = tf.nn.rnn_cell.MultiRNNCell([cell] * num_layers,
state_is_tuple=True)
```

or GRU

```
# GRU
cell = tf.nn.rnn_cell.GRUCell(state_size)
```

Now we can test using LSTM or GRU

\+ much more to consider…

#Dropout

Regularization

Batch normalization

Adding noise…

Models LSTM or GRU can be saved:

```
saver = tf.train.Saver()
saver.save(sess, 'RNN_GRU_model_shakespeare')
```

Models LSTM or GRU can be used to generate TEXT

- NOTE that for this we need to rebuild the graph so as to accept a single character at a time

A se dije a su la caral mar entus de la mucho al disto escundas el maloras de sus entre de su como lo haballe que de sin cuanto en llaba en el muestra me camiminor las, coralas des como ello, que estroras de esprento, y asigo, podros compastra de aquellao es a todo de lo que a sercido al vene de la contuento entaron esa camo, separ en esta estusando a lo hallar de su amo, y con la alvando.

Y le caballero, que lleguado al por el mano, a su por mas quiero a toda el pocina dejor la porte en serventar a la cual de algún estar en alguna dejuna me de ella embro que se le cabrese las ma

Basic concepts: LSTM & GRU

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

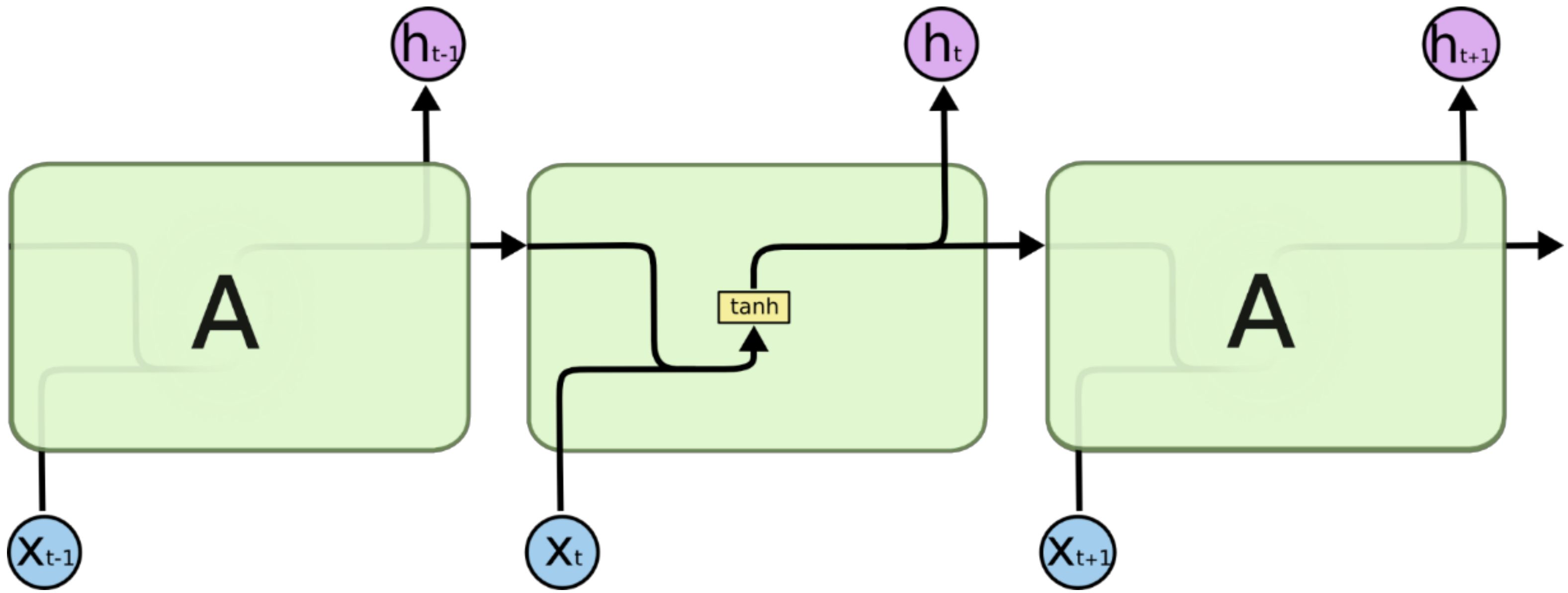
- **Vanishing gradients**: a problem?

it may be that for some tasks we want gradients to vanish completely, and for others, it may be that we want them to grow

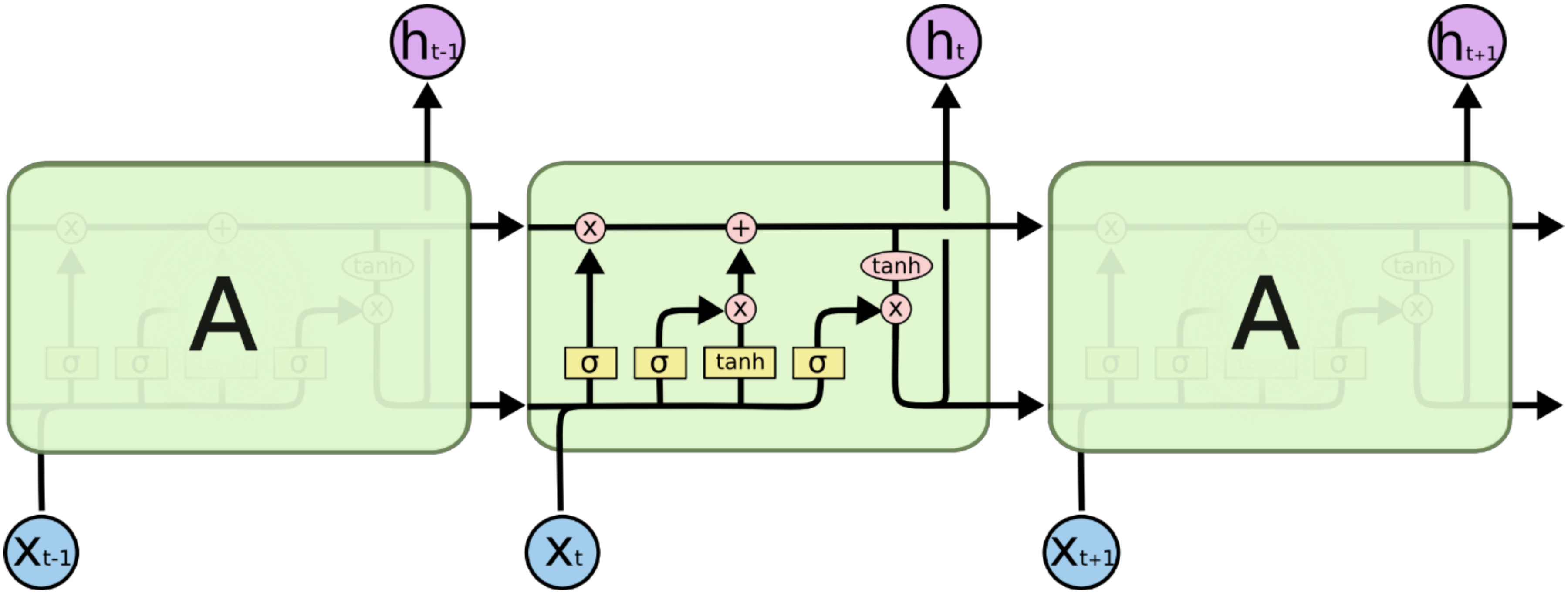
- Are RNNs capable of handling "**long-term dependencies**?"

Long Short Term Memory networks – LSTMs
Hocreiter and Schmidhuber (1997)

$NOTATION: states = h_t$

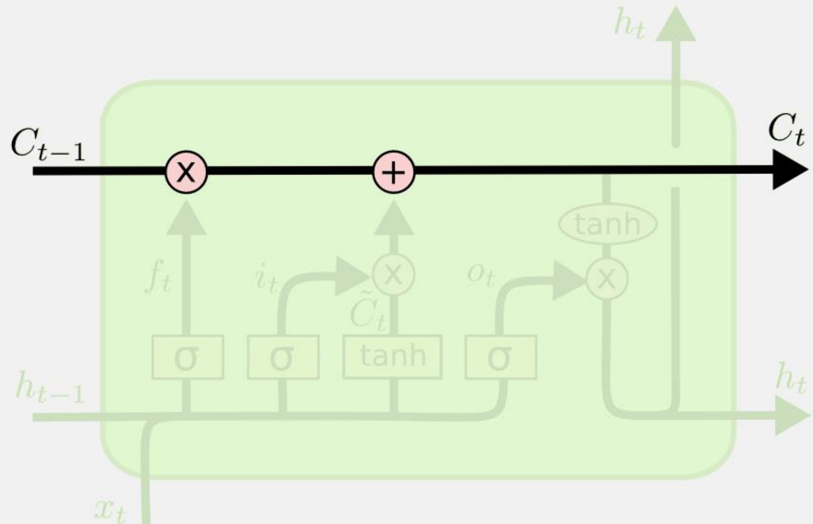


The repeating module in a standard RNN contains a single layer



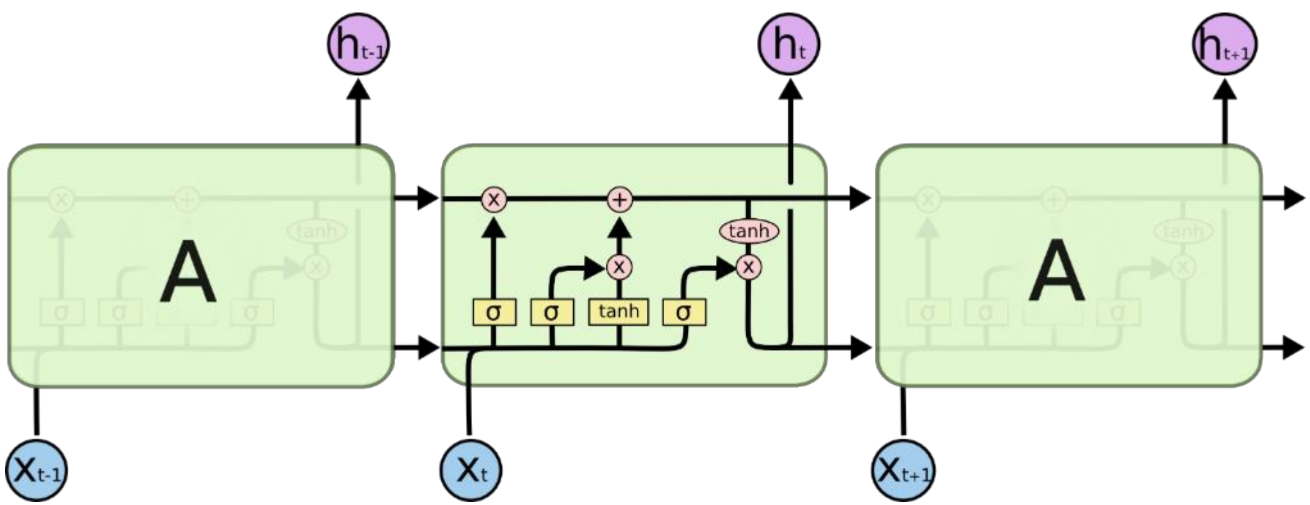
The repeating module in LSTM contains four interacting layers

Core Idea Behind LSTM



CELL STATE $= \boldsymbol{c_t}$

$states = h_t$



Tensorflow NOTE:

cell = tf.nn.rnn_cell.LSTMCell(state_size, state_is_tuple=True)

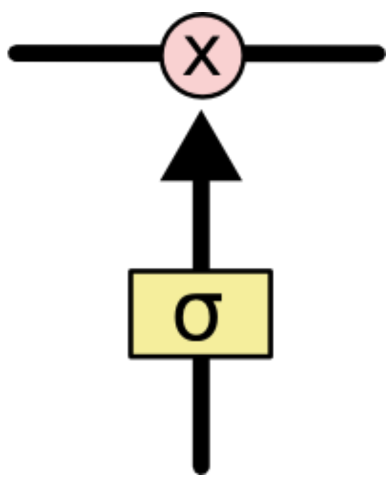
- “Hidden State" h_t and "Cell State" ,C_t

Core Idea Behind LSTM

- The LSTM does have the ability to **remove or add information to the cell state,** carefully regulated by structures called *gates*.

- **Gates** are a way to optionally let information through.

- They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state

Basic concepts: LSTM & GRU

Hocreiter and Schmidhuber (1997)

- The fundamental principle of LSTMs: to ensure the integrity of our messages in the real world, we **write them down**

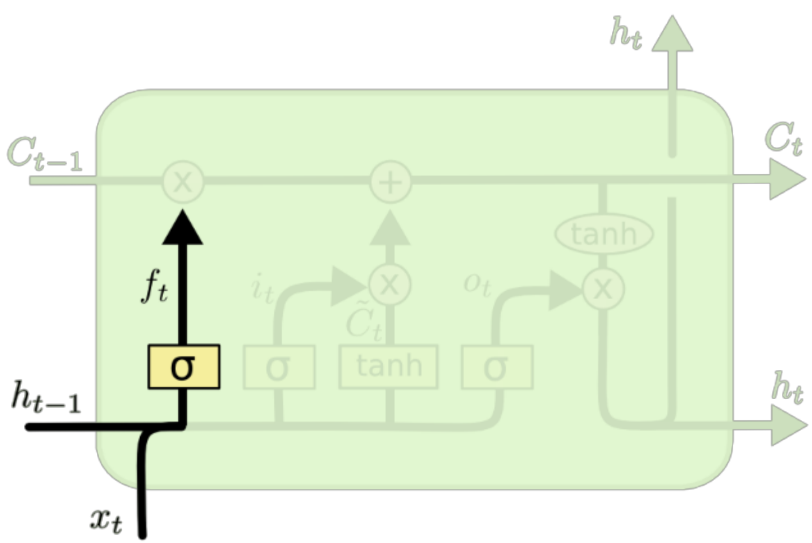
- The fundamental challenge of LSTMs and **keeping our state under control** is to be selective in three things:
  1. what we write (write selectivity),
  2. what we read (because we need to read something to know what to write) (read selectivity)
  3. and what we forget (because obsolete information is a distraction and should be forgotten) (forget selectivity)

Gates as a mechanism for selectivity

Step-by-Step LSTM Walk Through

Forget gate: The first step in our LSTM is to decide what information we're going to throw away from the cell state.

When we "see" something new relevant we decide forget….

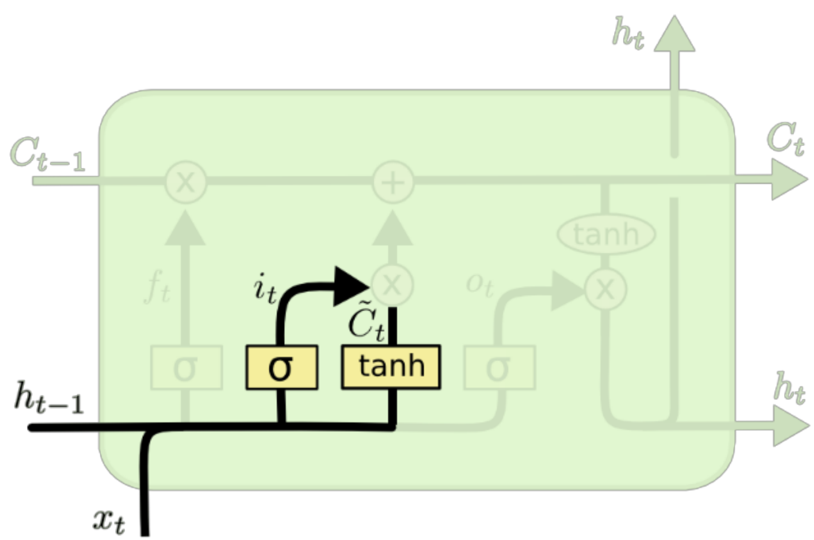


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Step-by-Step LSTM Walk Through

The next step is to decide what new information we're going to store in the cell state. This has two parts.

- First, a sigmoid layer called the "**input gate layer**" decides which values we'll update.
- Next, a **tanh layer** creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.



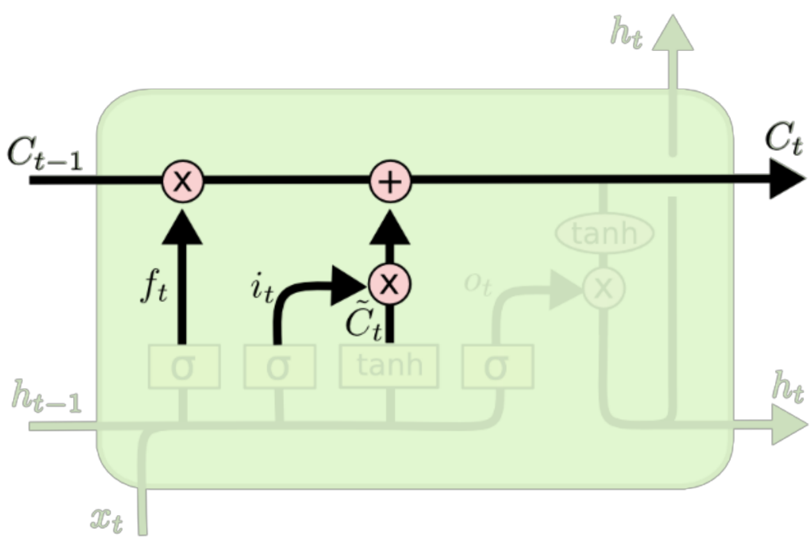
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Step-by-Step LSTM Walk Through

It's now time to **update the old cell state, C_{t-1}, into the new cell state C_t**

- The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t(forget gate) then we add $i_t * \tilde{C}_t$

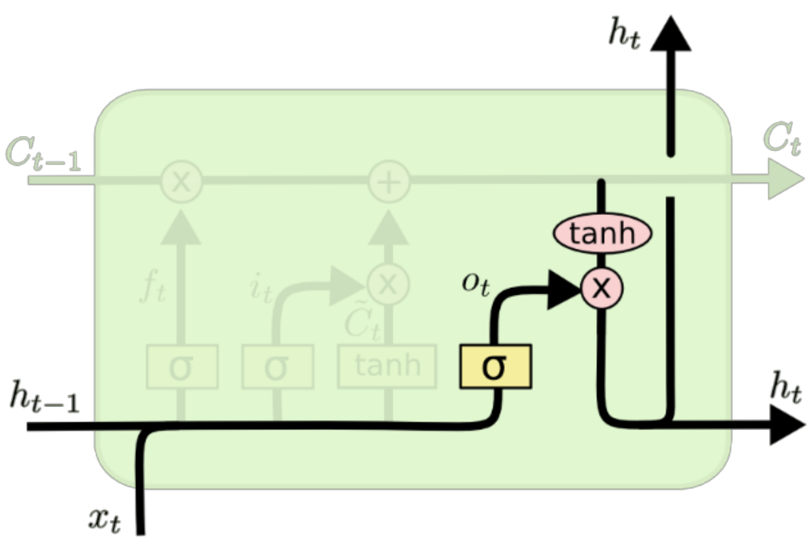


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step-by-Step LSTM Walk Through

Finally, we need to decide what we're going to output h_t. This output will be based on our cell state, but will be a filtered version.

- First, we run a sigmoid layer which decides what parts of the cell state we're going to output.
- Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



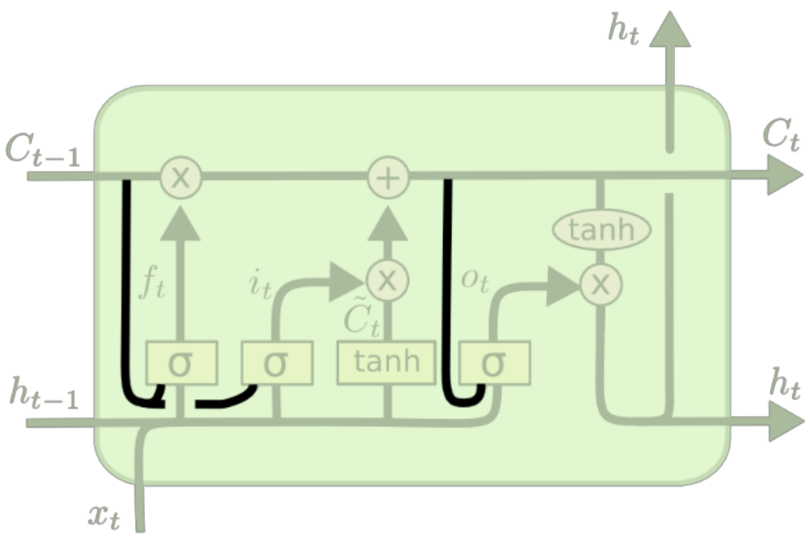
$$o_t = \sigma\left(W_o \, [h_{t-1}, x_t] \, + \, b_o\right)$$

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

Variants on Long Short Term Memory

One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding "peephole connections."

- This means that we let the gate layers look at the cell state.



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

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

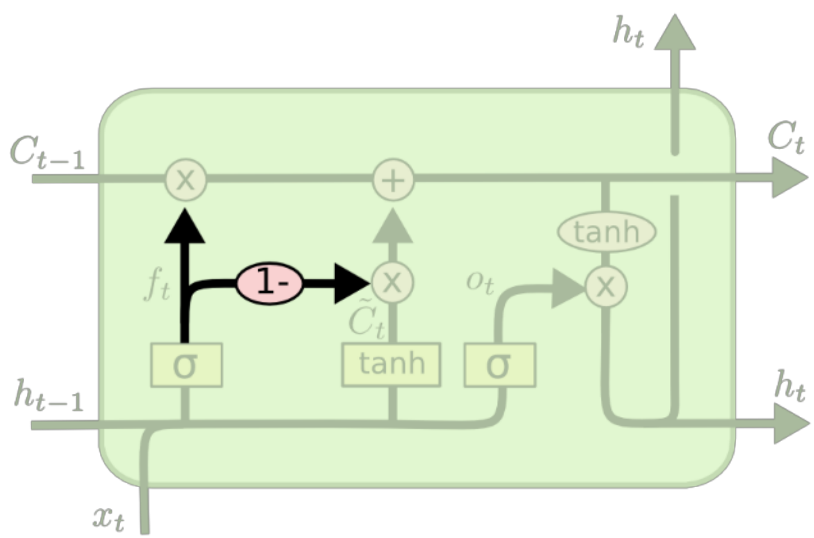
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

The above diagram adds peepholes to all the gates, but many papers will give some peepholes and not others.

Variants on Long Short Term Memory

Another variation is to use coupled forget and input gates.

- We only input new values to the state when we forget something older



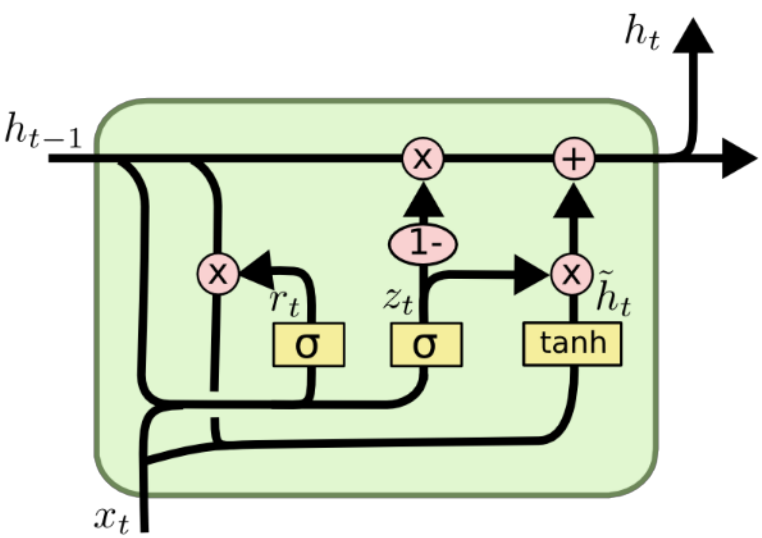
$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Variants on Long Short Term Memory

Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014).

- It combines the forget and input gates into a single “update gate.”
- It also merges the cell state and hidden state, and makes some other changes.

The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.



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

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

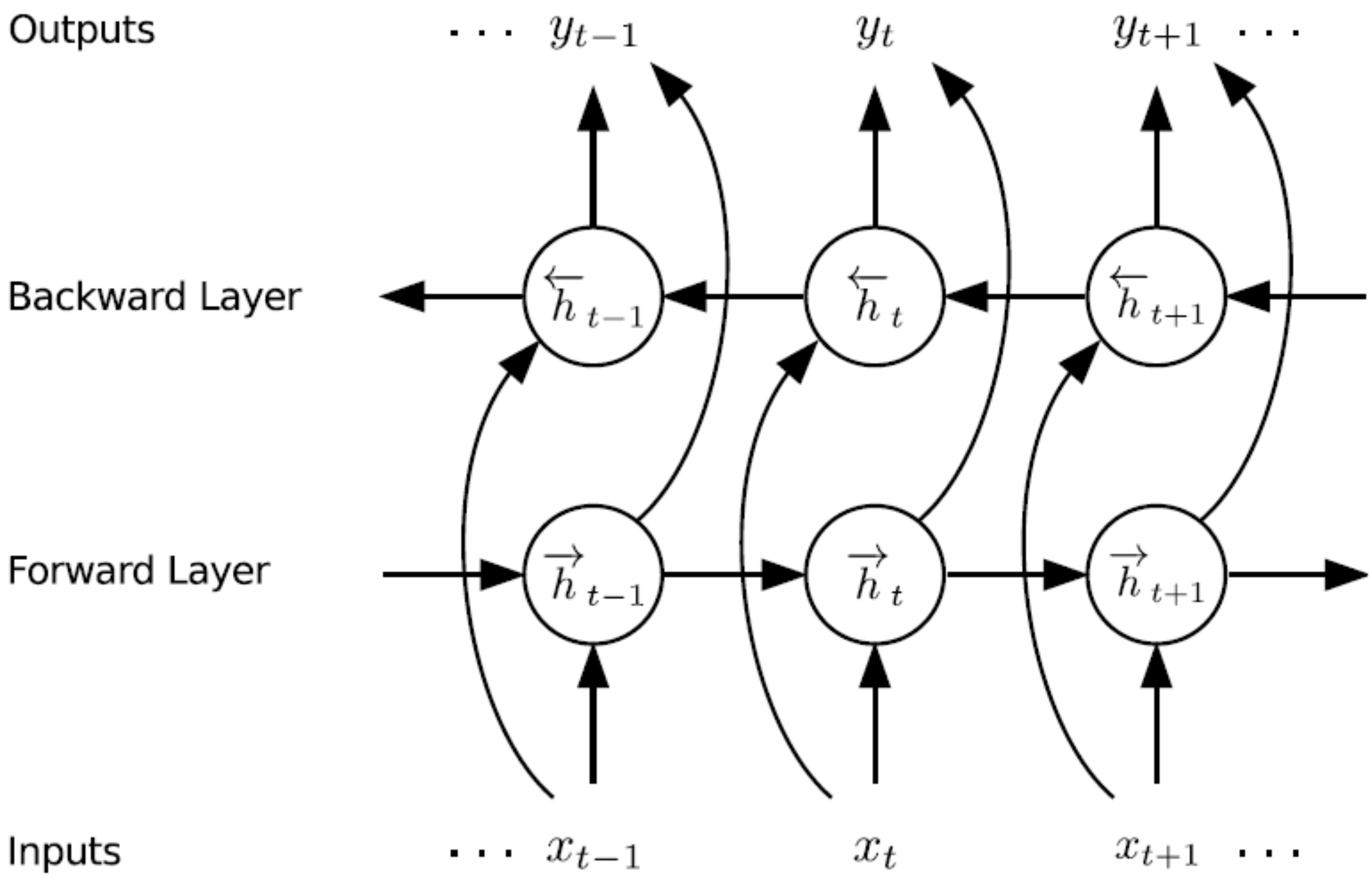
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Variants on Long Short Term Memory

These are only a few of the most notable LSTM variants.

See for example: **BLSTM** Bi-directional LSTM

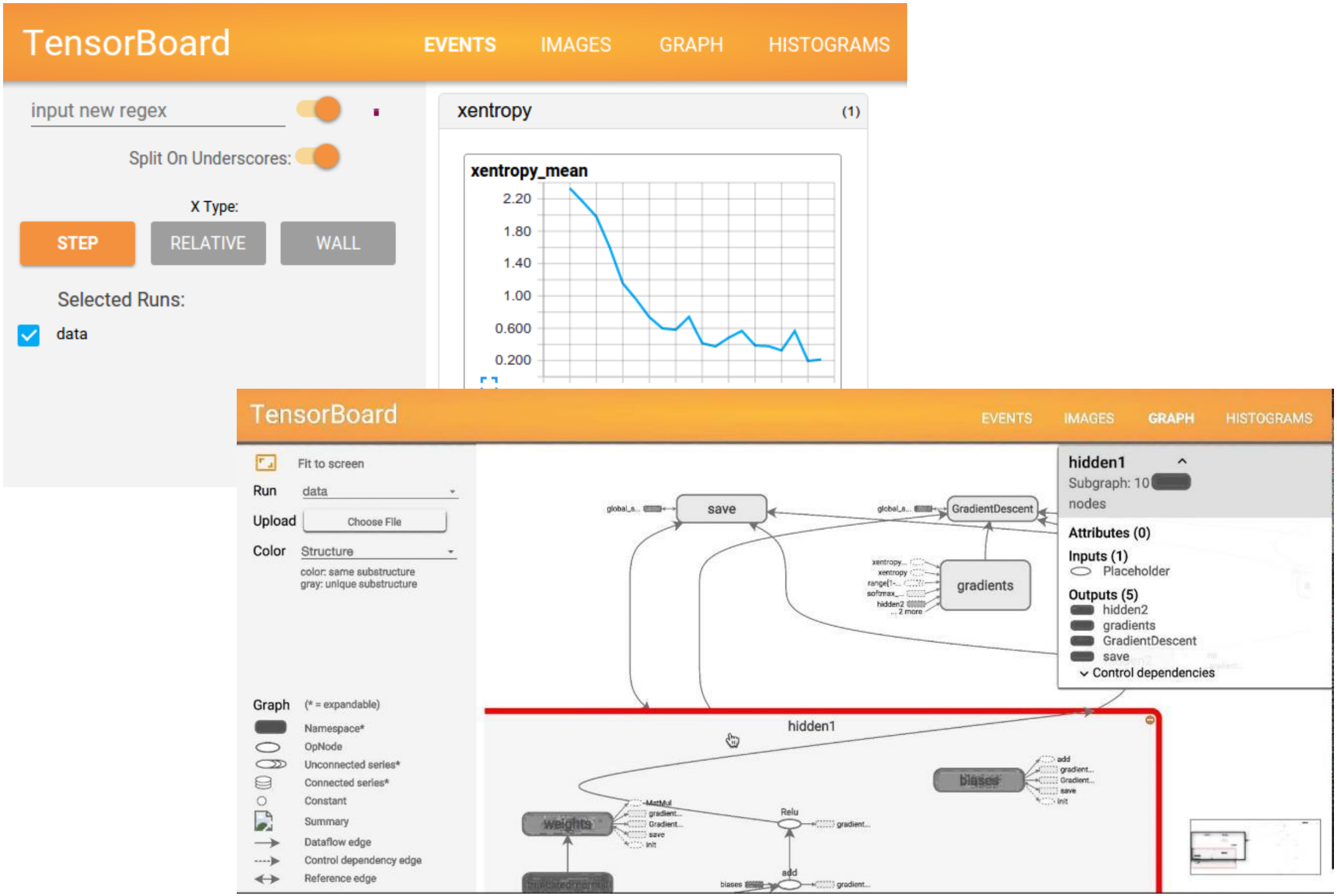


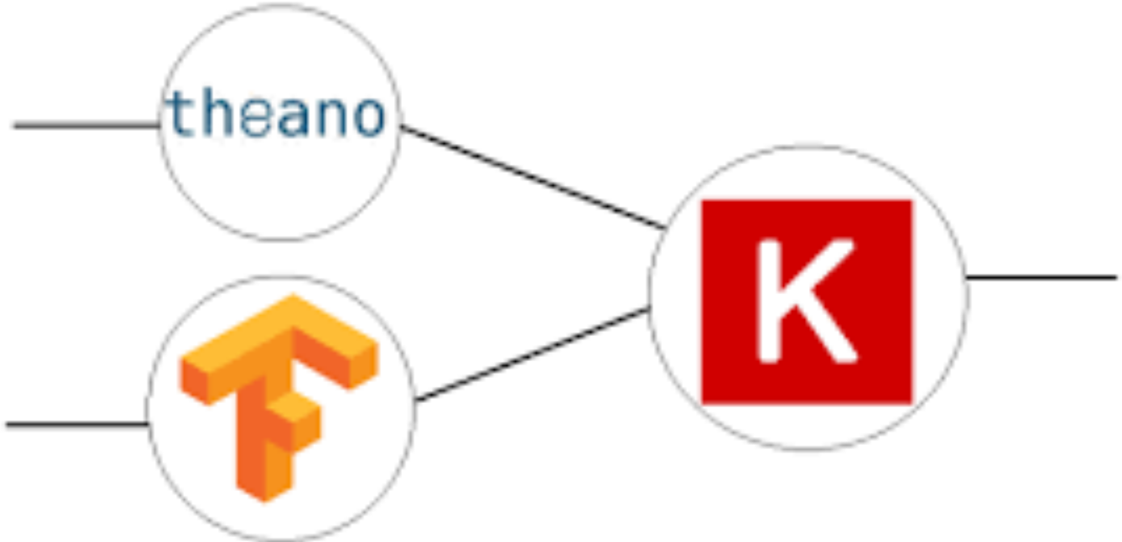
Variants on Long Short Term Memory

Which of these variants is best? Do the differences matter?

- Greff, et al. (2015) do a nice comparison of popular variants, finding that they're all about the same.
- Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

TensorBoard: Visualizing Learning





https://keras.io/

Scikit Flow

http://terrytangyuan.github.io/2016/03/14/scikit-flow-intro/

References (I)

https://theneuralperspective.com/2016/10/04/05-recurrent-neural-networks-rnn-part-1-basic-rnn-char-rnn/

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

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/

https://github.com/suriyadeepan/rnn-from-scratch

http://stats.stackexchange.com/questions/241985/understanding-lstm-units-vs-cells

http://monik.in/a-noobs-guide-to-implementing-rnn-lstm-using-tensorflow/

https://medium.com/@erikhallstrm/hello-world-rnn-83cd7105b767

http://suriyadeepan.github.io/2017-01-07-unfolding-rnn/

http://archive.eetindia.co.in/www.eetindia.co.in/VIDEO_DETAILS_700001601.HTM

References (I)

http://r2rt.com/styles-of-truncated-backpropagation.html

http://akkikiki.github.io/assets/LSTM+and+GRU.html

http://campuspress.yale.edu/yw355/deep_learning/

http://datascience.stackexchange.com/questions/12964/what-is-the-meaning-of-the-number-of-units-in-the-lstm-cell

http://stackoverflow.com/questions/37901047/what-is-num-units-in-tensorflow-basiclstmcell