DEPARTAMENTO DE SEÑALES, SISTEMAS Y RADIOCOMUNICACIONES

Acoustic Signal Processing

Máster Universitario en Ingeniería de Telecomunicación

Prof. Eduardo López Gonzalo
eduardo.lopez@upm.es
Prof. Luis A. Hernández Gómez
luisalfoso.hernandez@upm.es

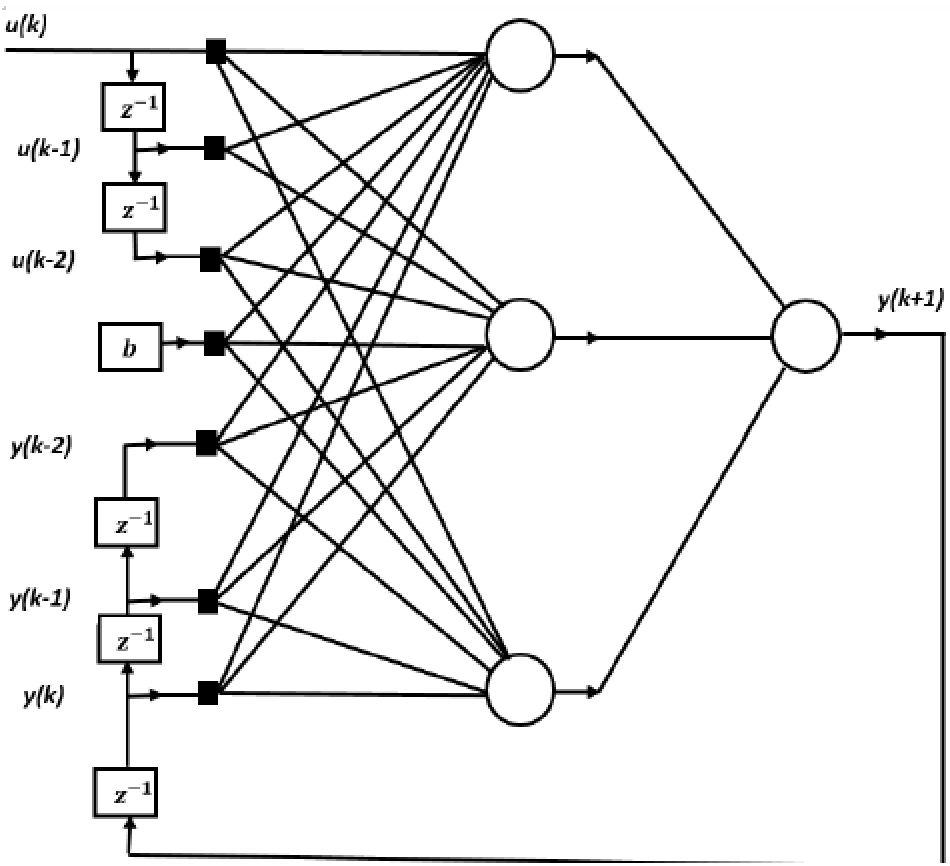
Departamento de Señales, Sistemas y Radiocomunicaciones
E.T.S. Ingenieros de Telecomunicación
Universidad Politécnica de Madrid

Modelling sequences

- Applications areas are broad:
  - **Signals**: video, speech, sensors,....
  - **Time series**: financial series, log messages,..
  - **Sequence of characters**: Natural Language Processing,...
  - **Sequences of observations**, actions, rewards: **Reinforcement Learning** (robots, bots, autonomous vehicles, dialog systems..)

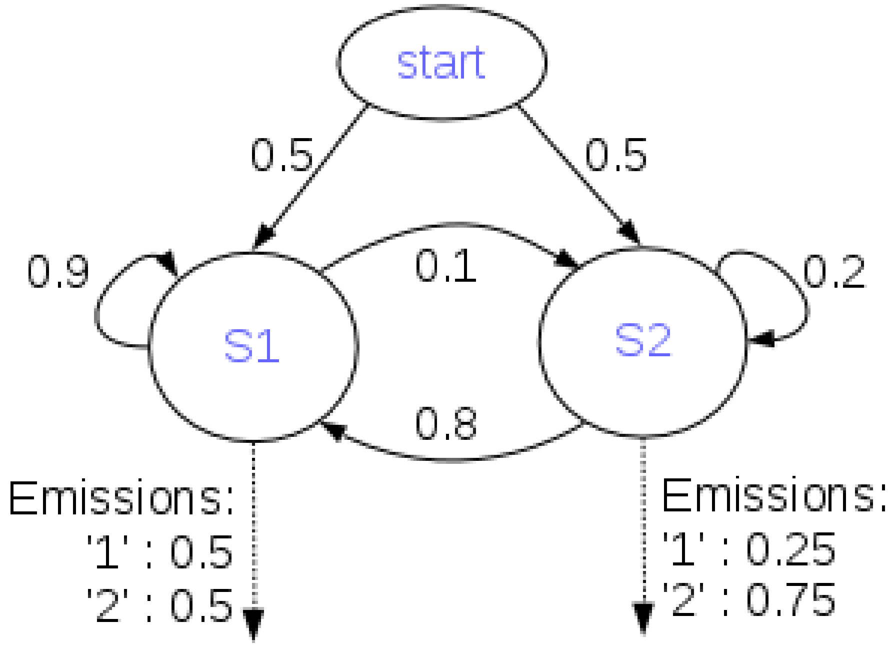
Memoryless models for sequences

- **NARX** model: **N**onlinear **A**uto **R**egressive Model with e**x**ternal inputs.
- Another name: **TDNN** (time-delay neural network)



Dynamic Generative Models:
as for example Hidden Markov Models

- Hidden Markov Models have a discrete one-of-N hidden state.
- Transitions between states are stochastic and controlled by a transition matrix.
- The outputs produced by a state are stochastic.



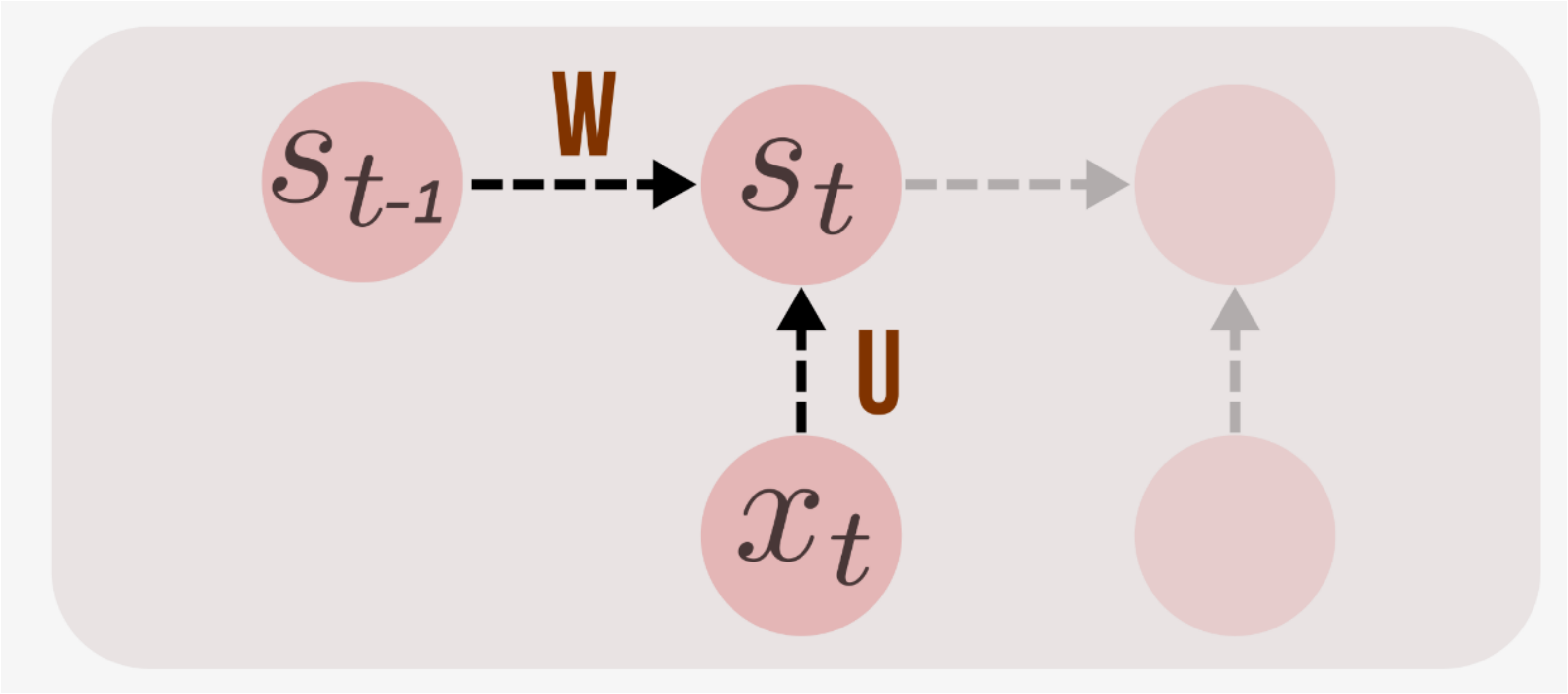
A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data.
  - At each time step it must select one of its hidden states.
  - So with N hidden states it can only remember log(N) bits about what it generated so far.

From: Geoffrey Hinton
CSC2535 2013: Advanced Machine Learning

Recurrent Neural Networks (RNN)

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



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

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

Recurrent Neural Networks (RNN)

- RNNs are very powerful, because they combine two properties:
  – Distributed hidden state that allows them to store a lot of information about the past efficiently.
  – Non-linear dynamics that allows them to update their hidden state in complicated ways.

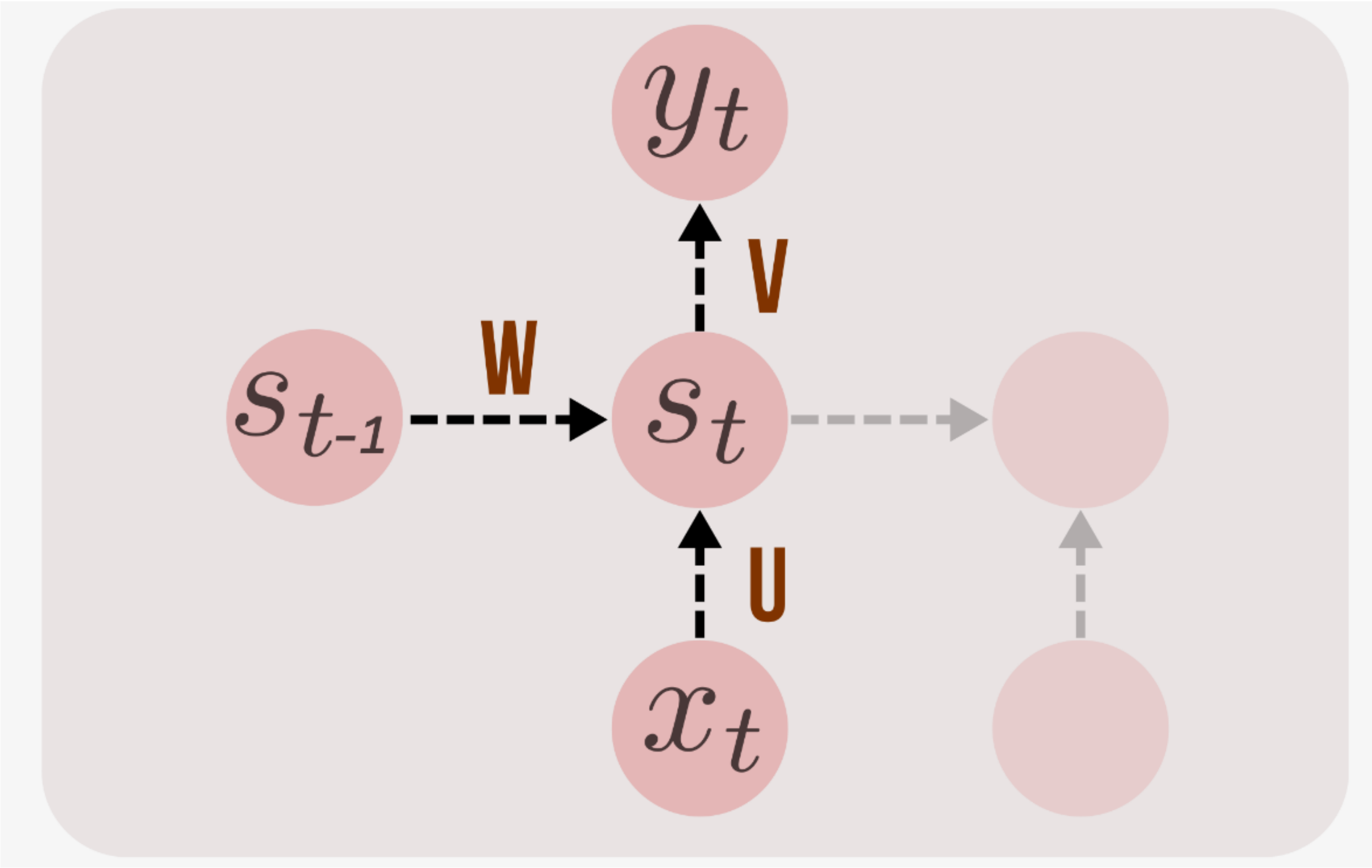
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.

From: Geoffrey Hinton
CSC2535 2013: Advanced Machine Learning

OUTPUT y_t

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

predictions = tf.nn.softmax(logits)



RNN essentially describe programs:

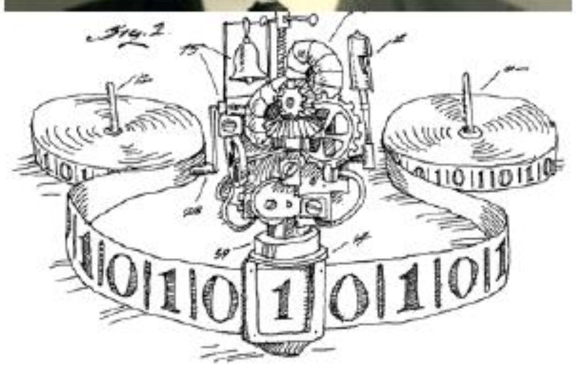
inputs + some internal variables.

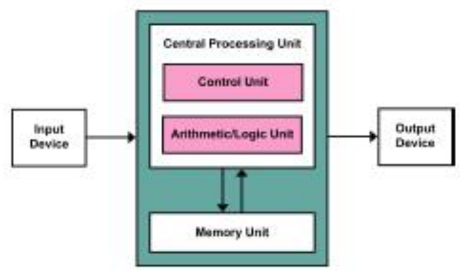
In fact, it is known that [RNNs are Turing-Complete] in the sense that they can to simulate arbitrary programs (with proper weights).

Neural Turing Machines

Can neural nets learn programs?

Alex Graves
Greg Wayne
Ivo Danihelka





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

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

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

Recurrent Neural Networks Applications

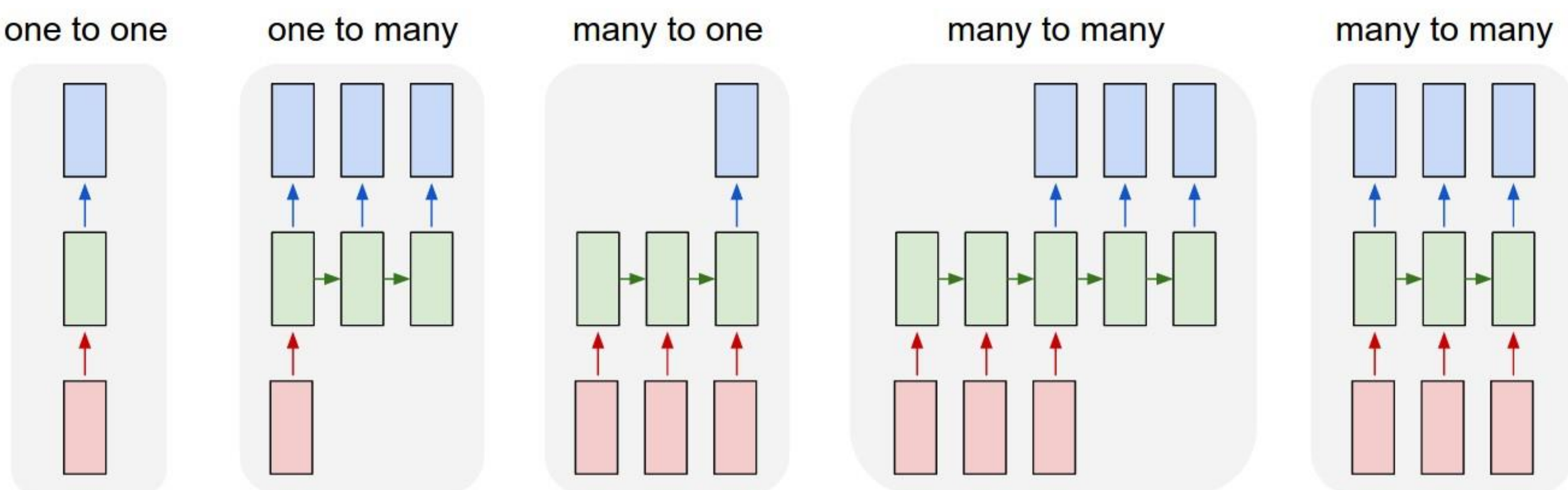


Image description using text

Emotion Recognition

Language Model

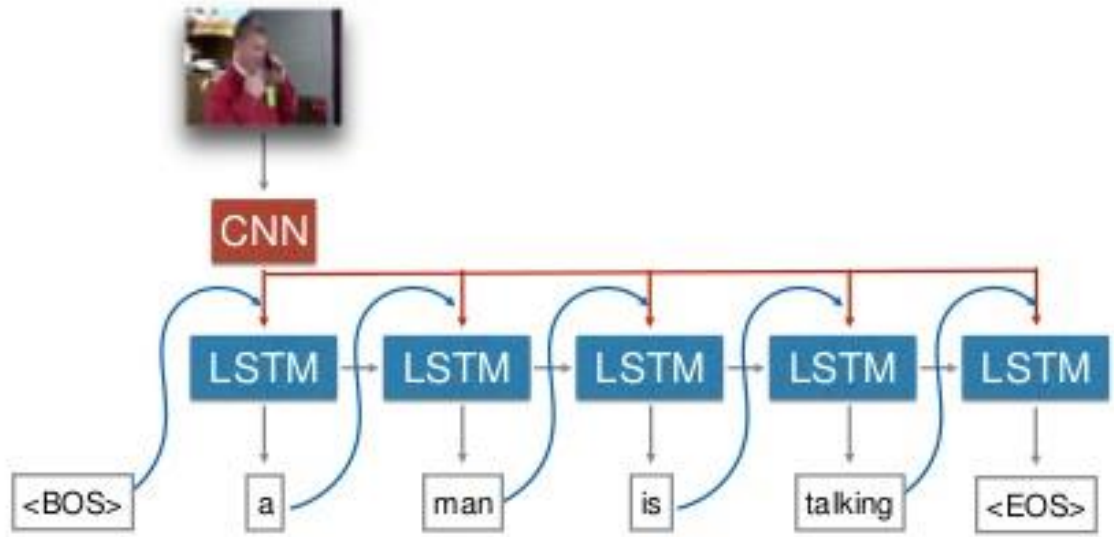
Machine Translation

Video frame labelling

Phoneme Recognition

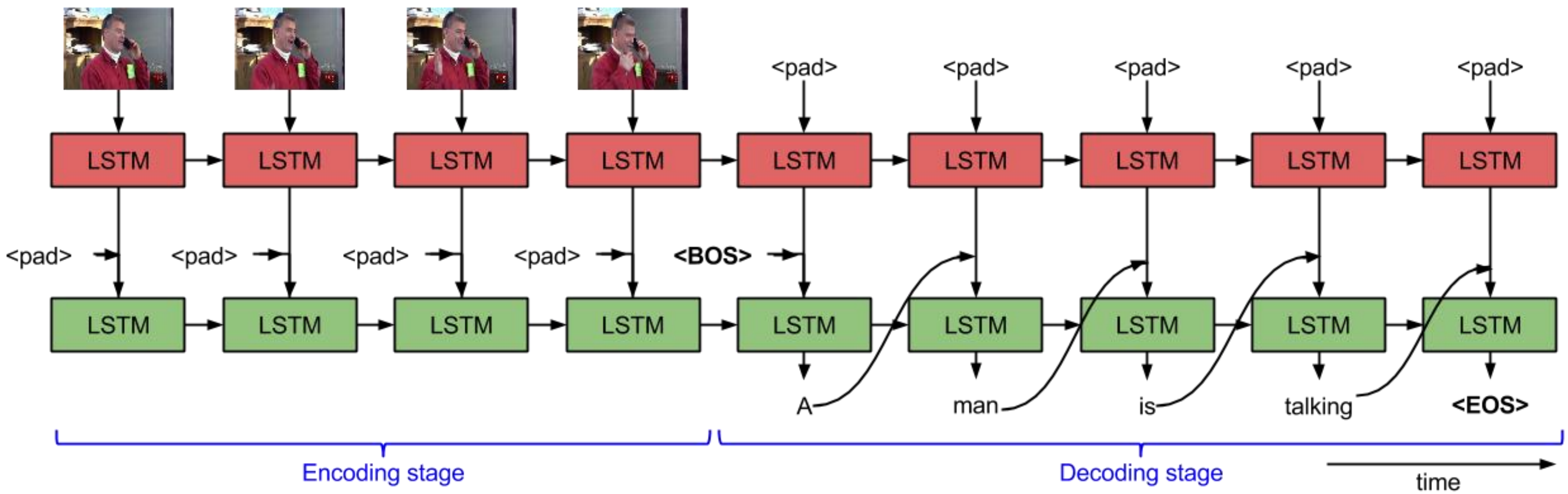
Image Description

sequential
output



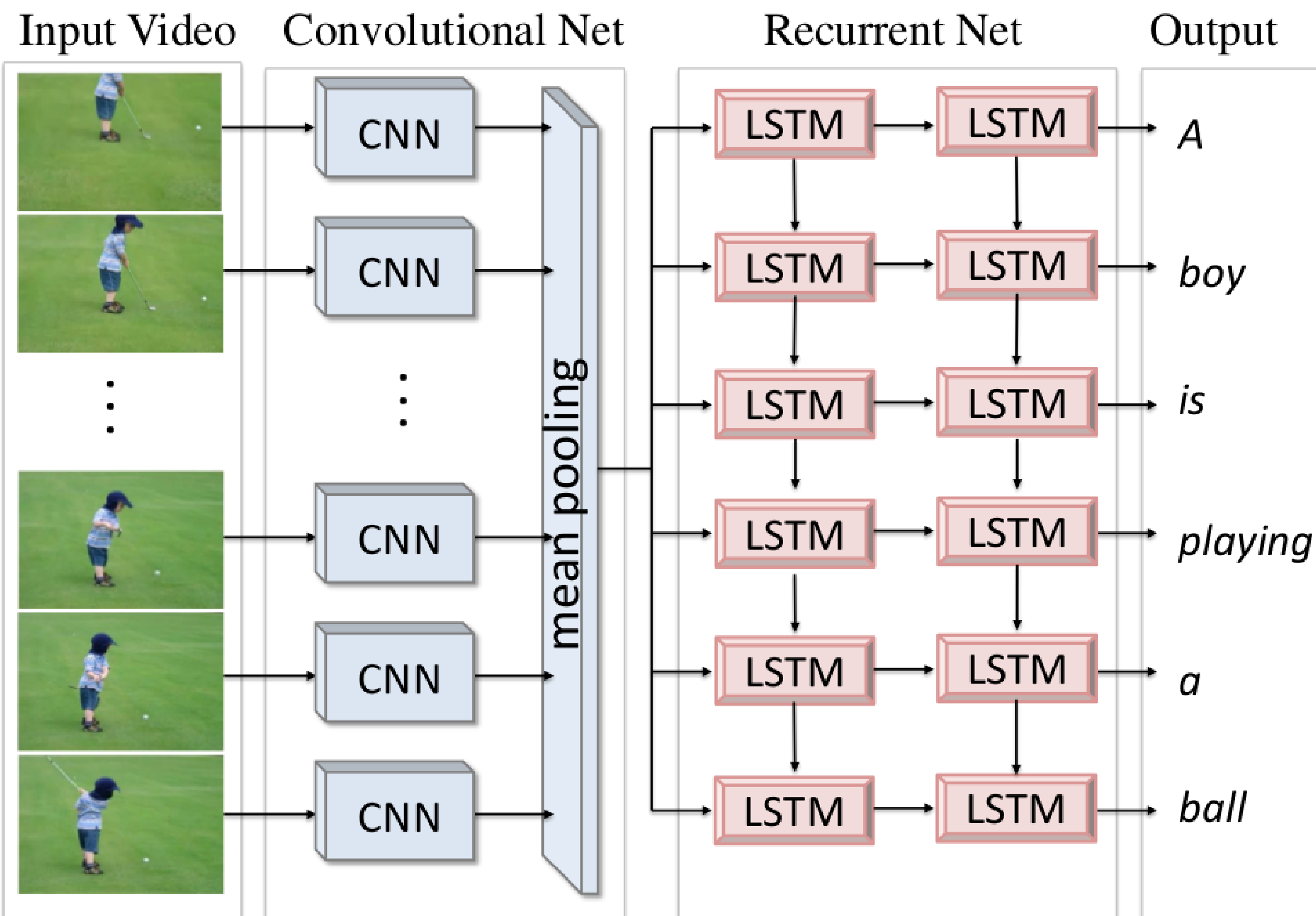
- Jeff Donahue, CVPR Caffe Tutorial, June 6, 2015

Sequence to Sequence - Video to Text
Subhashini Venugopalan, et al.



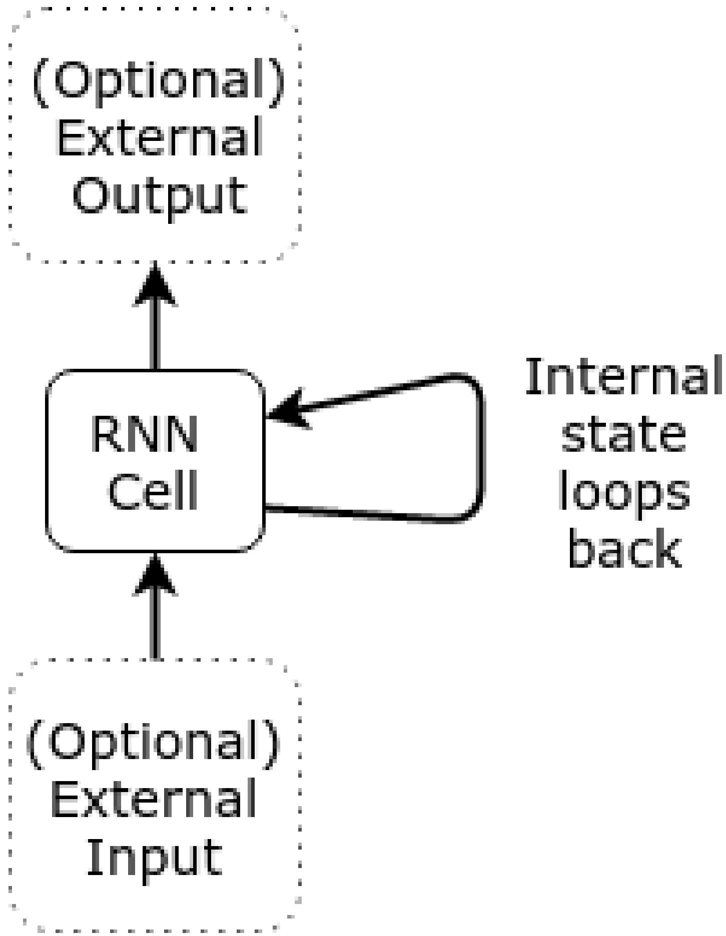
Translating Videos to Natural Language Using Deep Recurrent Neural Networks

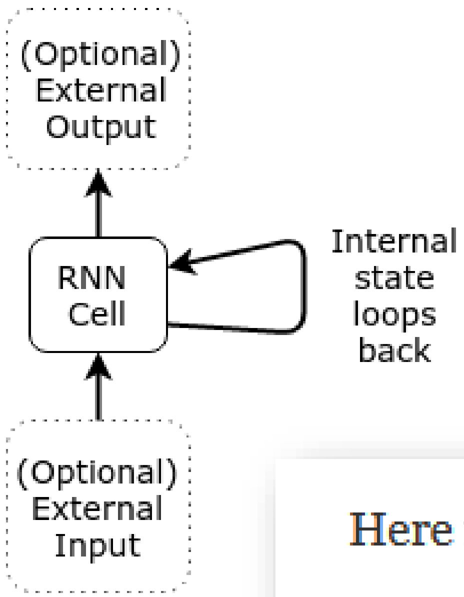
Subhashini Venugopalan, et al.



Written Memories: Understanding, Deriving and Extending the LSTM

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



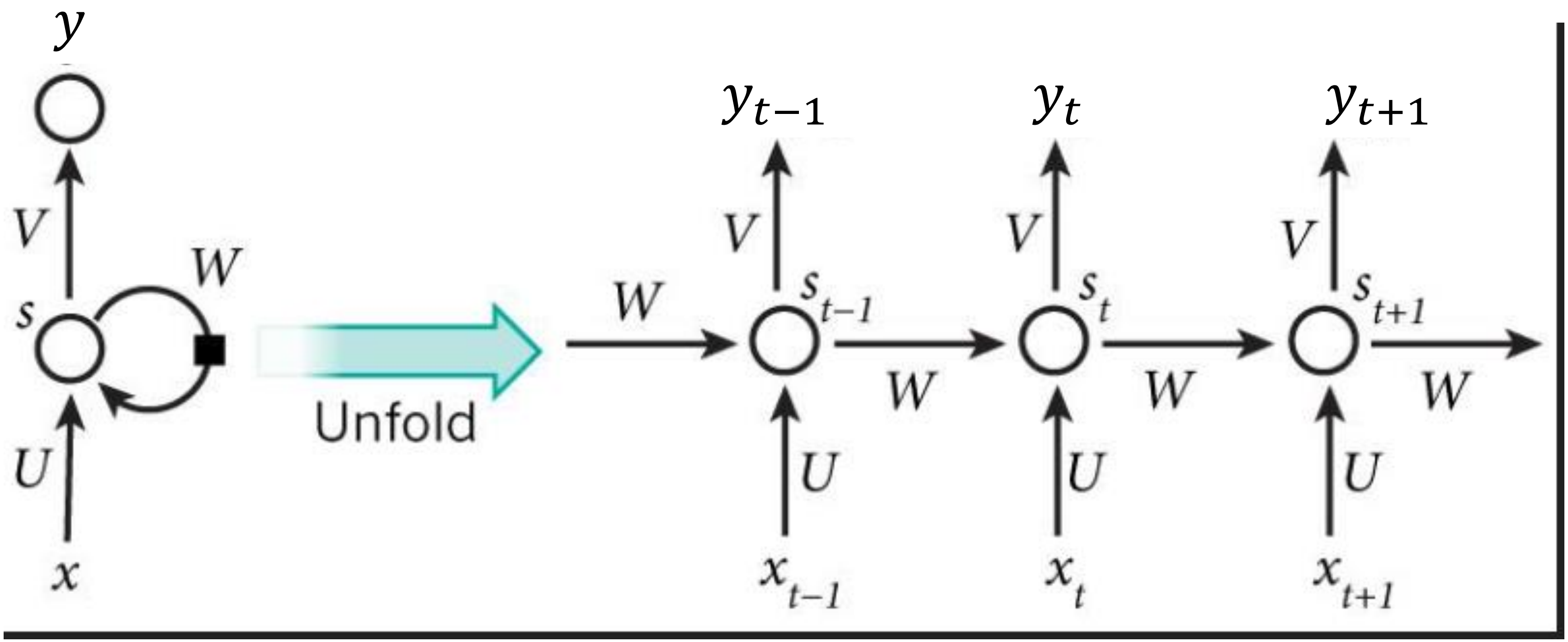


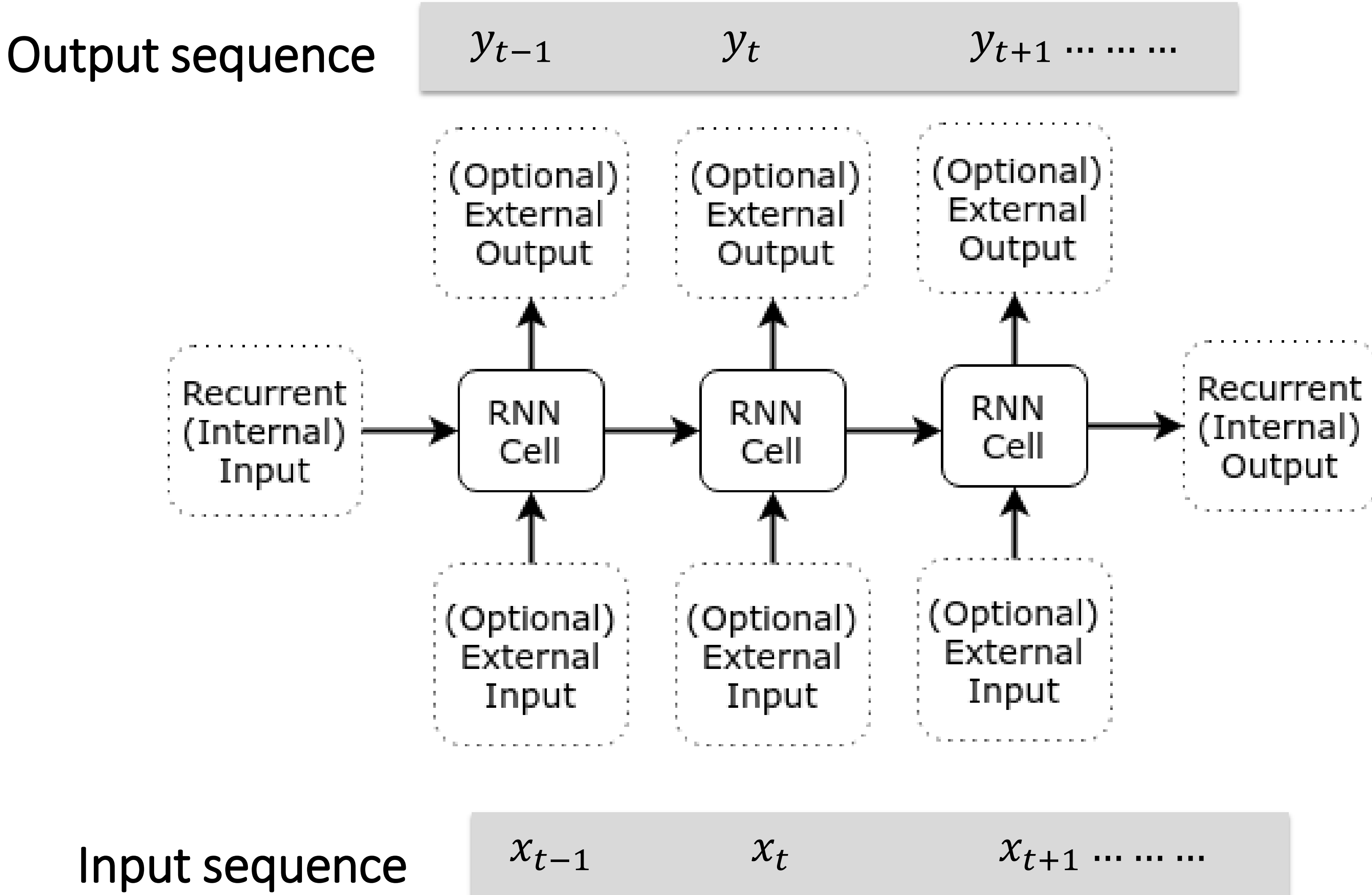
Here is the algebraic description of the RNN cell:

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

where:

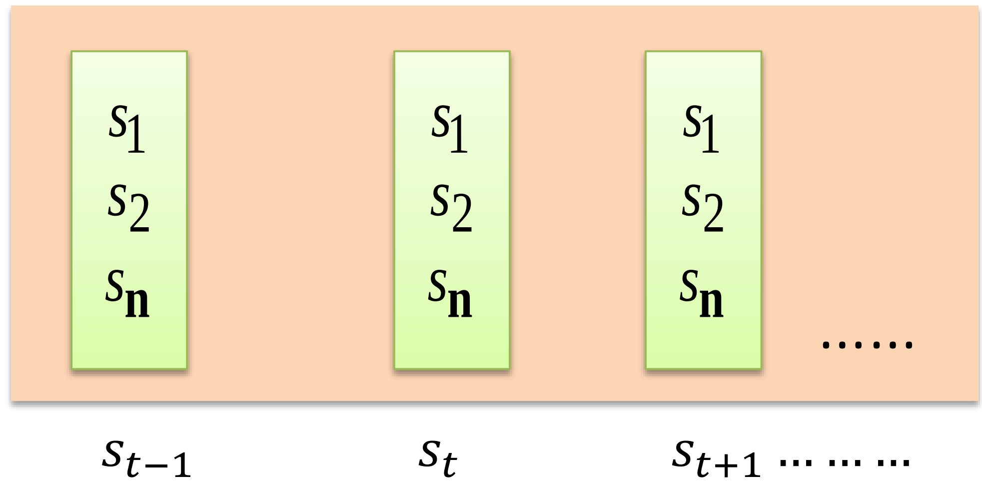
- s_t and s_{t-1} are our current and prior states,
- o_t is our (possibly empty) current output,
- x_t is our (possibly empty) current input, and
- f is our recurrent function.

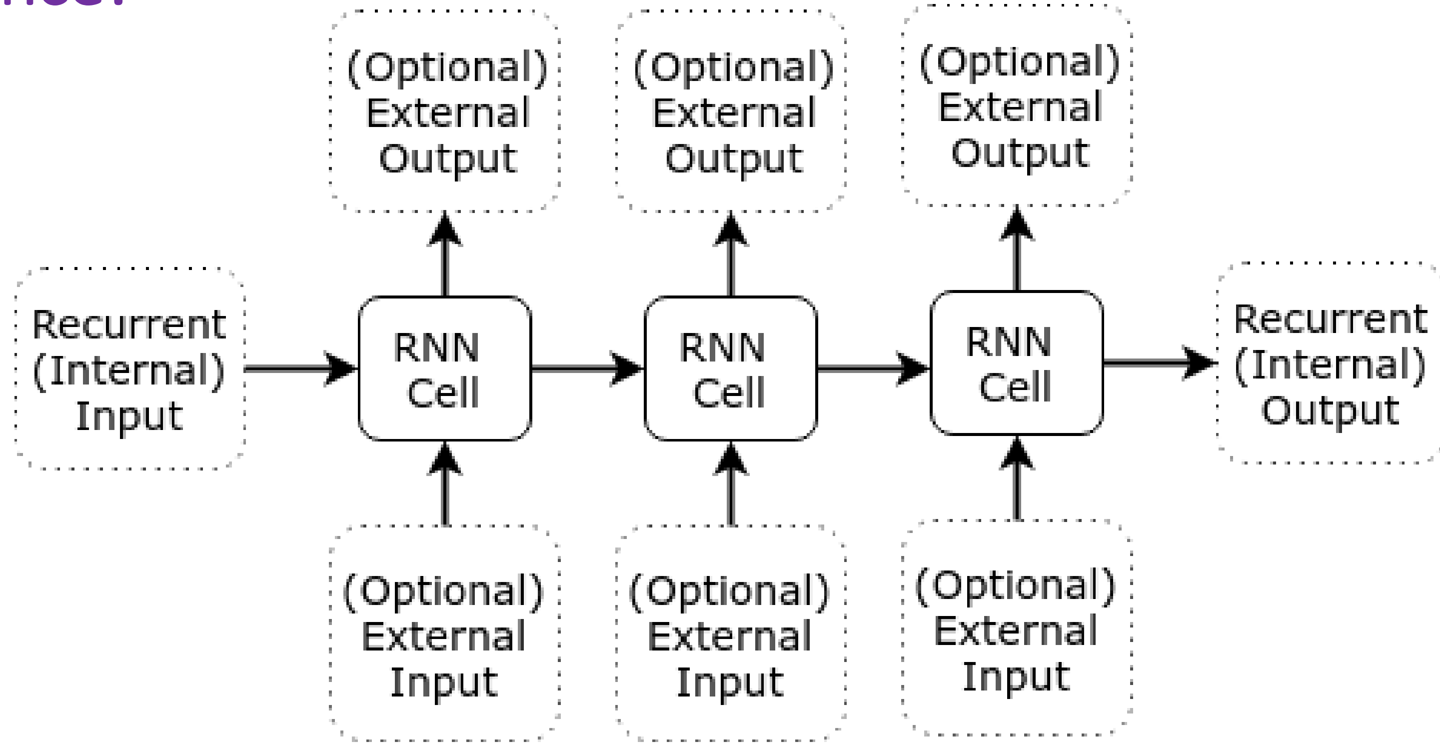

y
V
W
s
U
x
Unfold
W
y_{t-1}
V
s_{t-1}
W
U
x_{t-1}
y_t
V
s_t
W
U
x_t
y_{t+1}
V
s_{t+1}
W
U
x_{t+1}



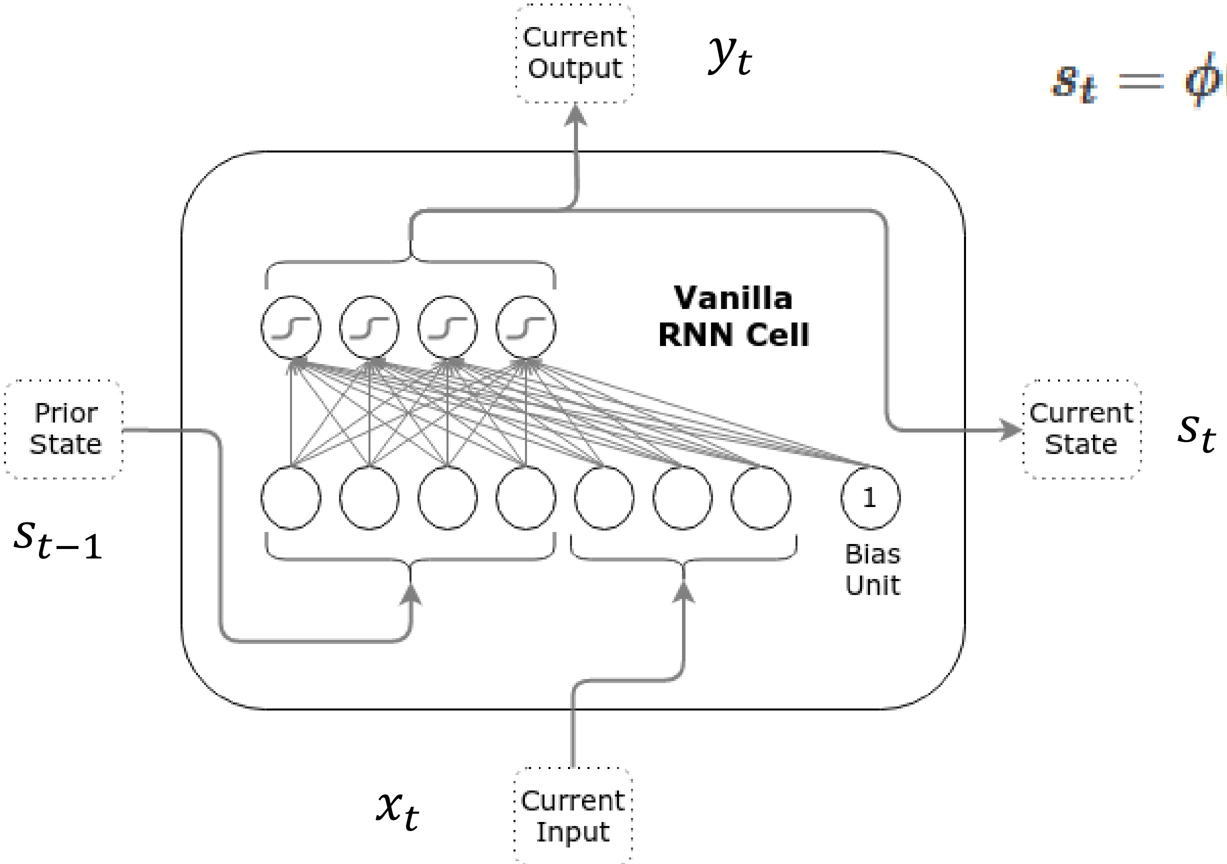
Output sequence
y_{t-1} y_t y_{t+1} … … …
(Optional) External Output
(Optional) External Output
(Optional) External Output
Recurrent (Internal) Input
RNN Cell
RNN Cell
RNN Cell
Recurrent (Internal) Output
(Optional) External Input
(Optional) External Input
(Optional) External Input
Input sequence
x_{t-1}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

State sequence!





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

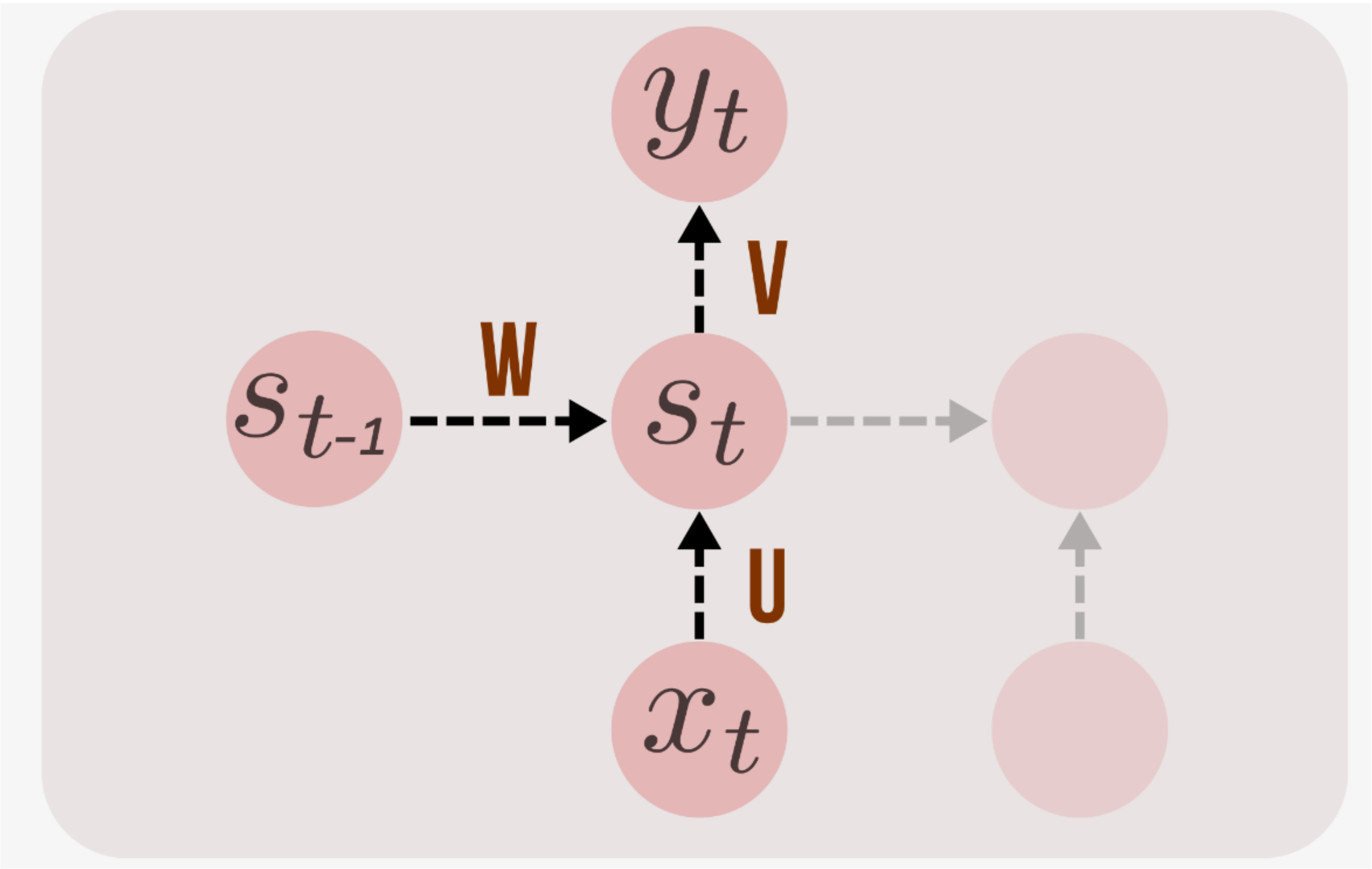


where:

- 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.

TRAINING RNN: W, U, V ?

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



Backpropagation Through Time (BPTT)

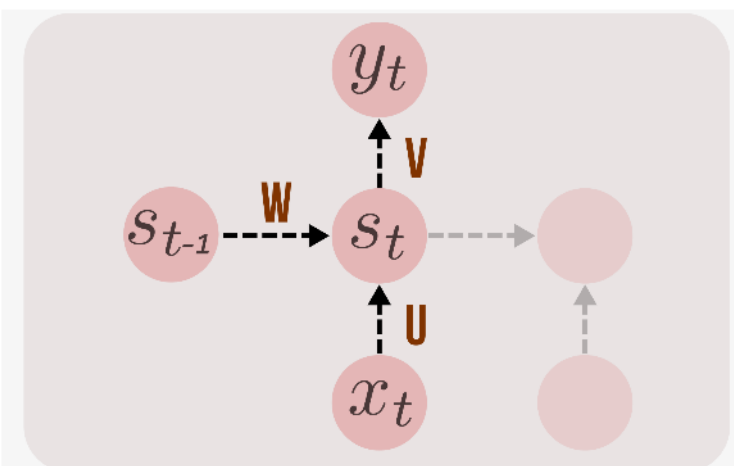
- Because the parameters are shared by all time steps in the network
- The gradient at each output depends not only on the calculations of the current time step, but also the previous time steps.

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

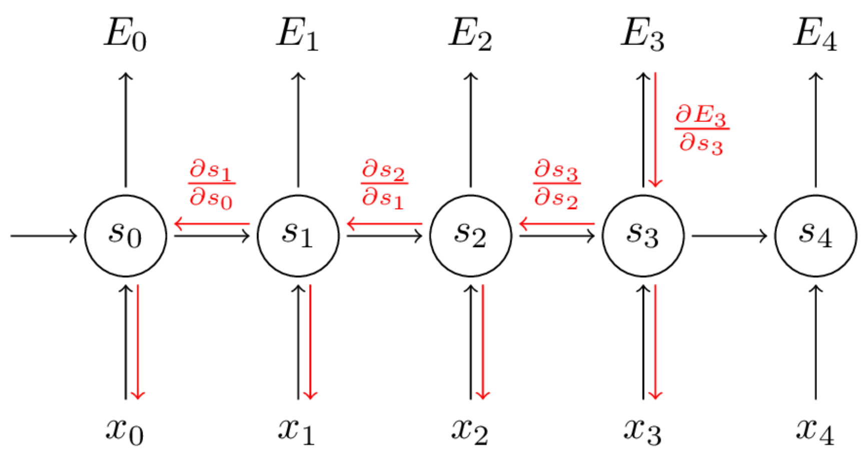


- The output value does depend on the state of the hidden layer,
- which depends on all previous states of the hidden layer
- and thus, all previous inputs

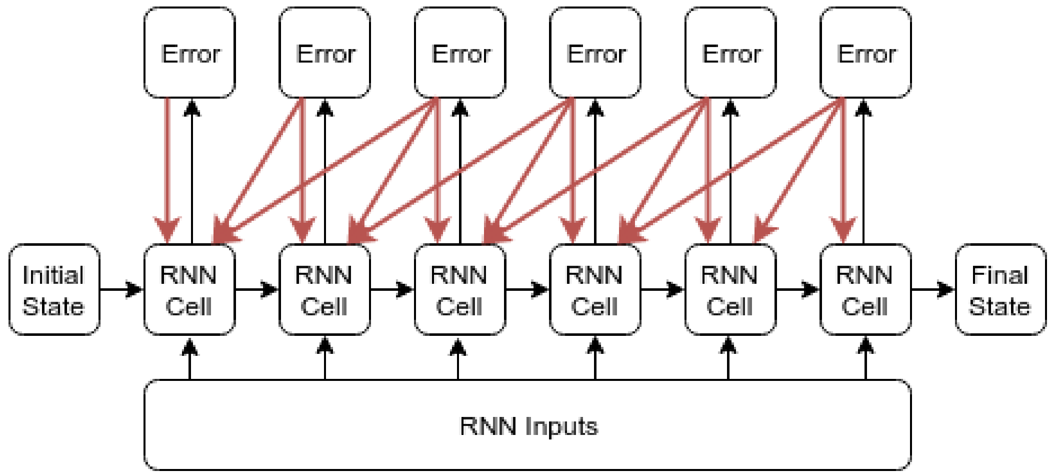
Backpropagation Through Time (BPTT)

- 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

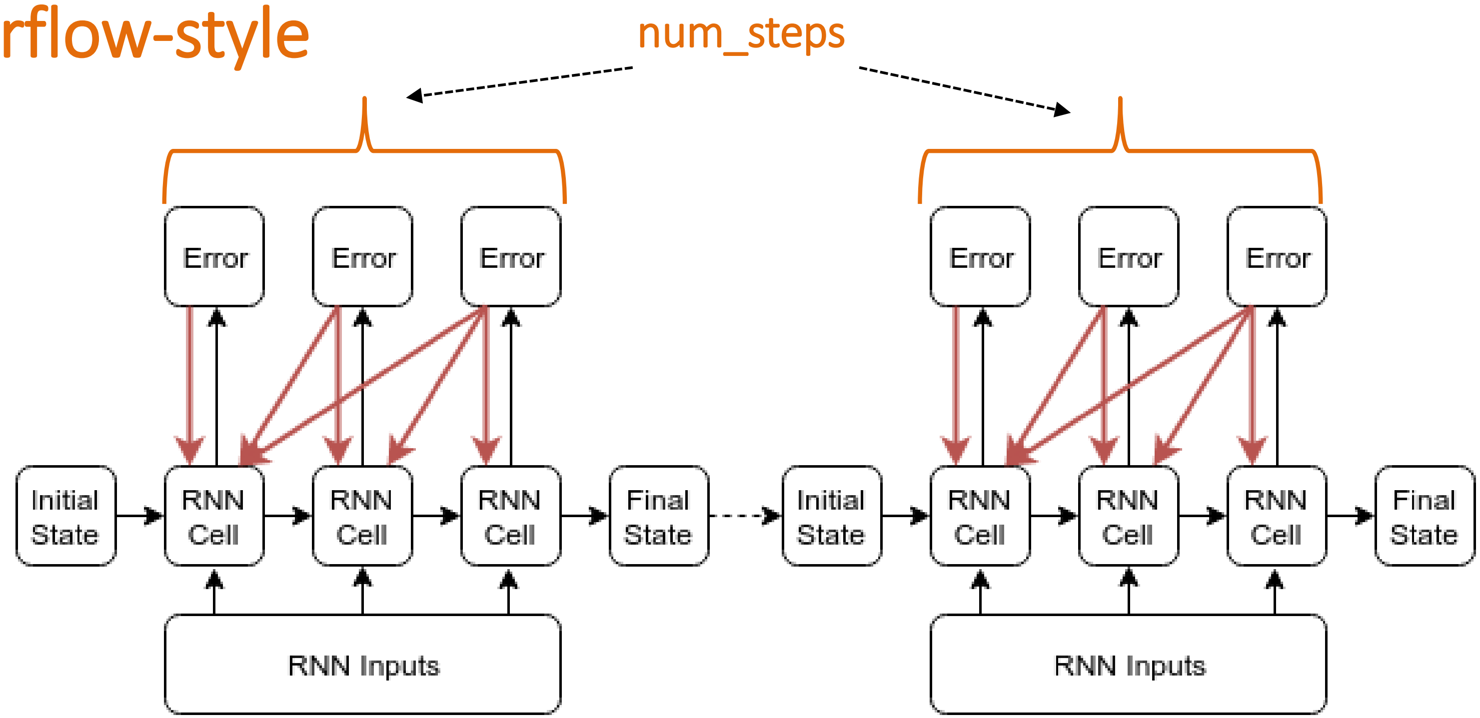
$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial y_y} \frac{\partial y_t}{\partial s_t} \boxed{\frac{\partial s_t}{\partial s_k}} \frac{\partial s_k}{\partial W}$$



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/

Let’s practices for **series prediction**

in TensorFlow with a BasicRNNCell

https://github.com/MUIT-TSA/DeepLearning_TF_Keras

TSA_RNN_1_Series_2018.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**.

Tensorflow RNN static vs. dynamic

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]

See Neural networks and deep learning by Aurélien Géro

https://www.safaribooksonline.com/library/view/neural-networks-and/9781492037354/ch04.html

In particular Fig 9 to understand reshape

- You can try with financial data (hard) and synthetic signals (easy)
- As well as with LSTM and GRU

Now let's try a simple NLP task

Predicting Text Characters

https://github.com/MasterMSTC/DeepLearning_TF_Keras

TSA_Keras_RNN_2_Text_2018.ipynb

Text prediction & generation using RNN

Keras

Text Prediction/Generation with Keras using LSTM: *Long Short Term Memory* networks

In this example we will work with the book: Alice's Adventures in Wonderland by Lewis Carroll.

We are going to learn the dependencies between characters and the conditional probabilities of characters in sequences so that we can in turn generate wholly new and original sequences of characters.

Adapted from:

Text Generation With LSTM Recurrent Neural Networks in Python with Keras

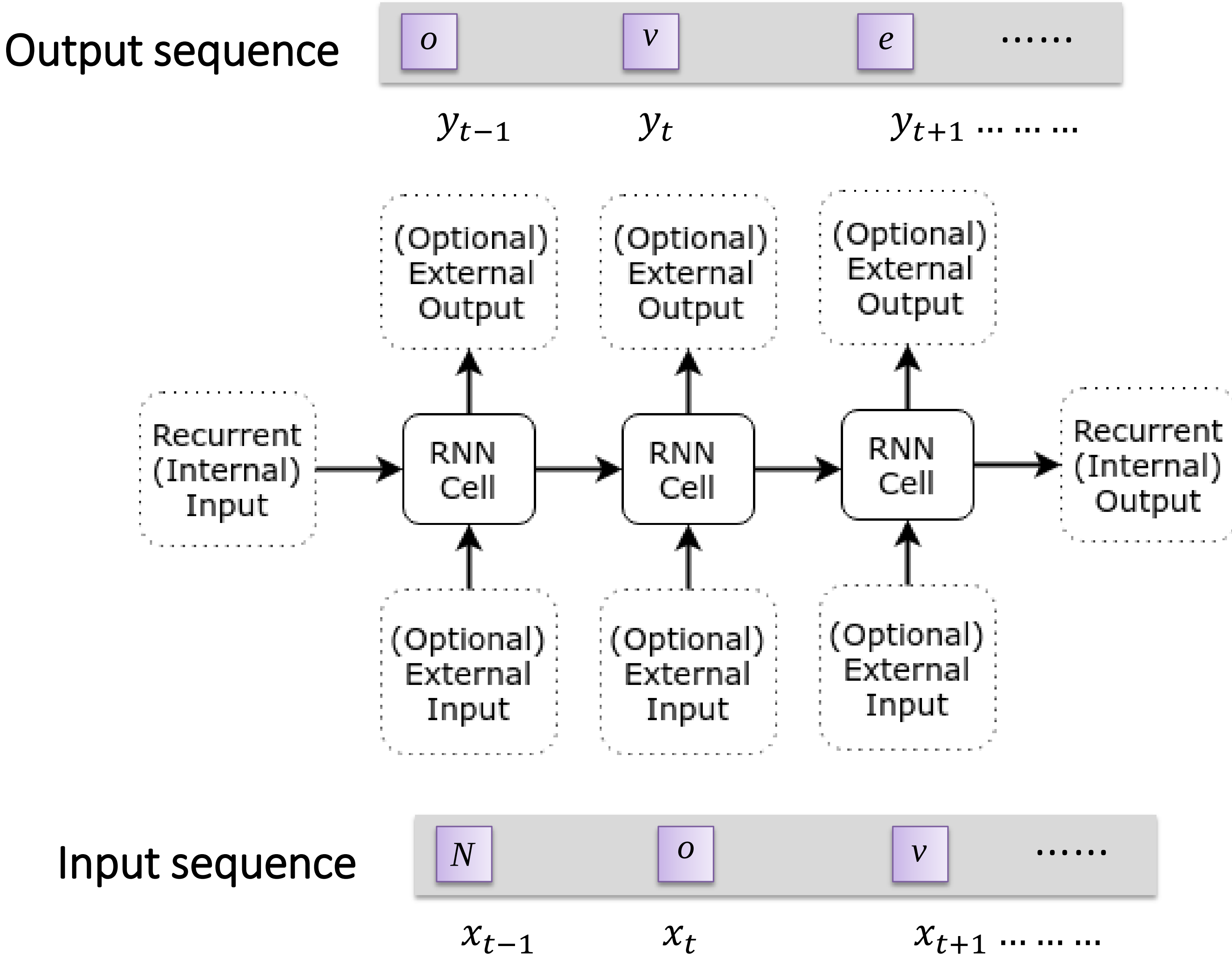
By Jason Brownlee

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} ... ...

Now we can test using LSTM or GRU

+ much more to consider…

#Dropout

Regularization

Batch normalization

Adding noise…

RNN are Generative Models

Vanilla (Elman), LSTM or GRU can be used to generate TEXT

100 th iteration

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

↓ train more

300 th iteration

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

↓ train more

700 th iteration

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ train more

2000 th iteration

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”

Basic concepts: LSTM & GRU

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

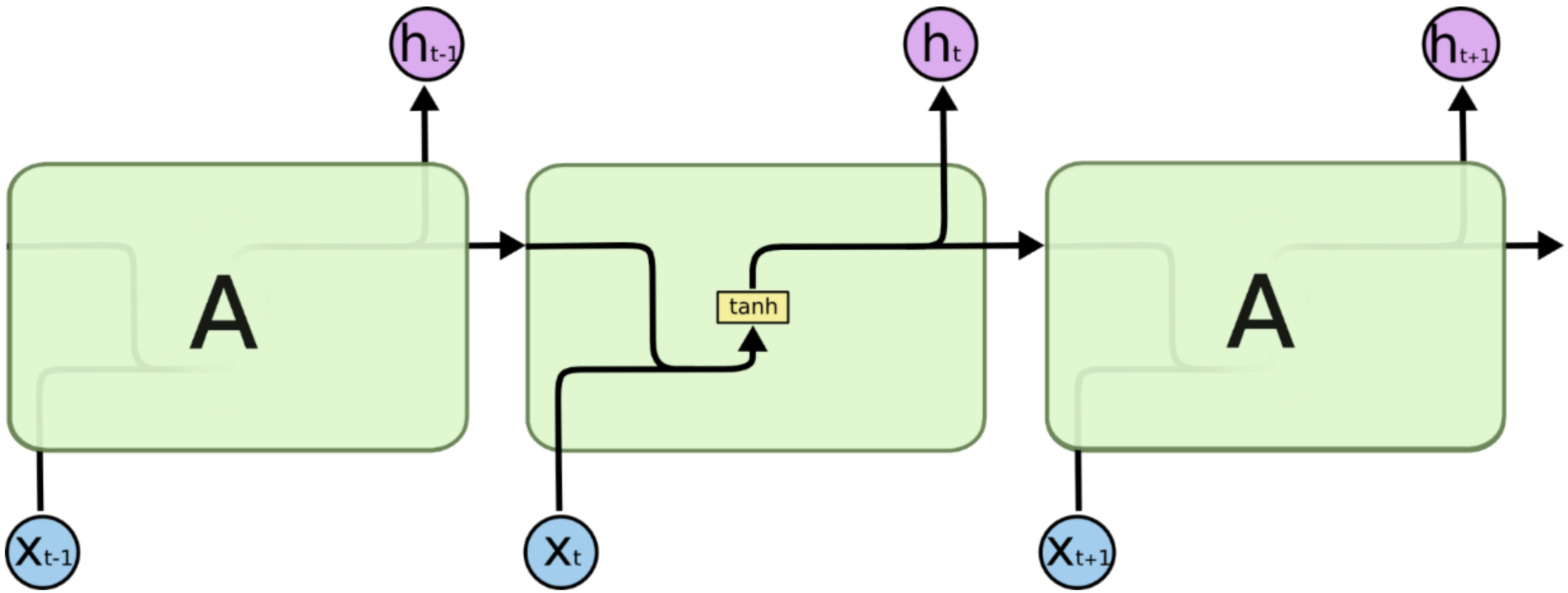
- **Vanishing gradients**: a problem?

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

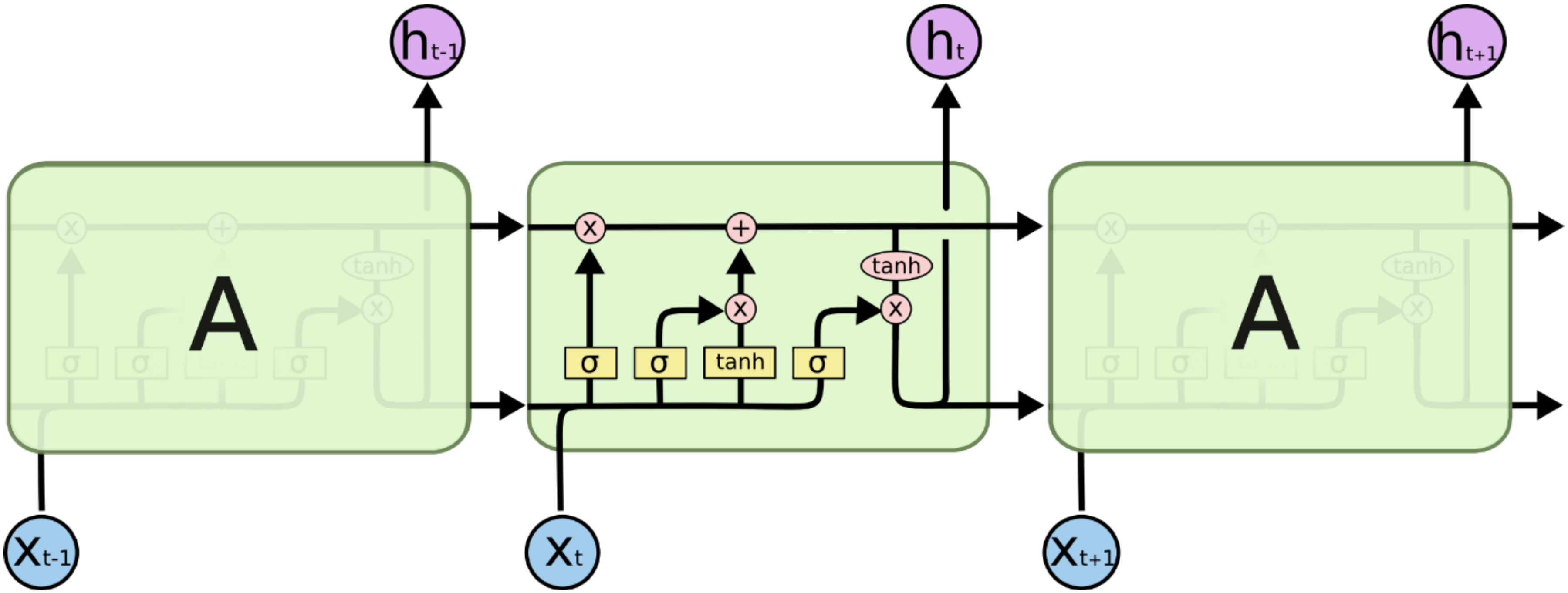
- Are RNNs capable of handling "**long-term dependencies**?"

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

$NOTATION: states = h_t$

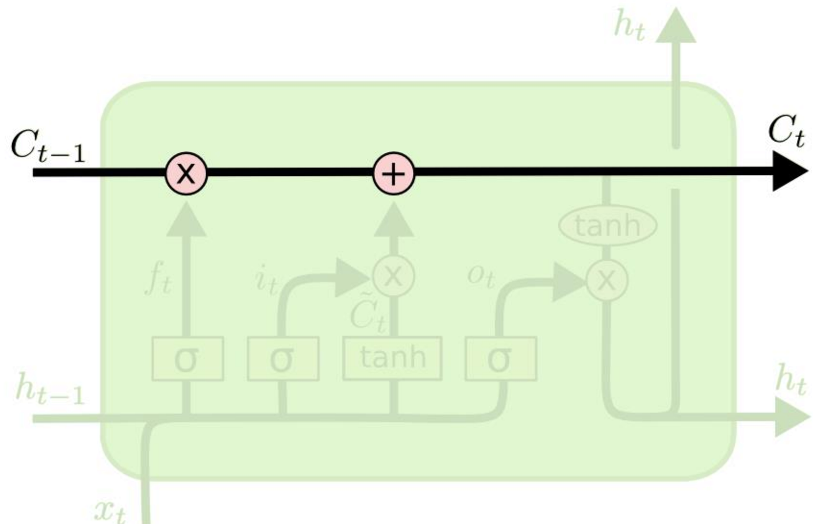


The repeating module in a standard RNN contains a single layer



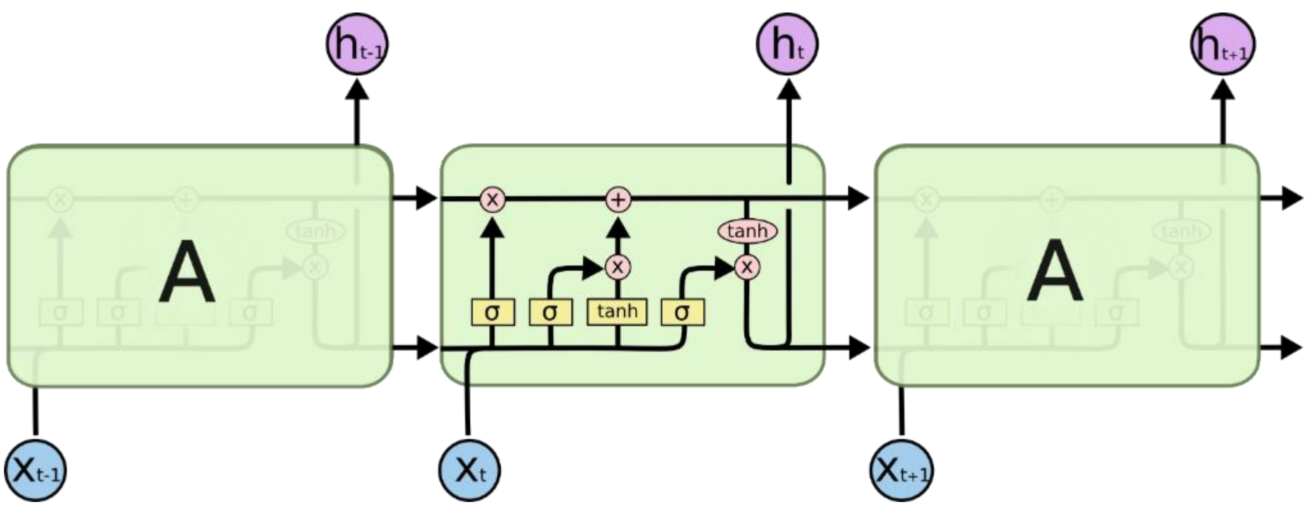
The repeating module in LSTM contains four interacting layers

Core Idea Behind LSTM



CELL STATE $= \boldsymbol{c_t}$

$states = h_t$



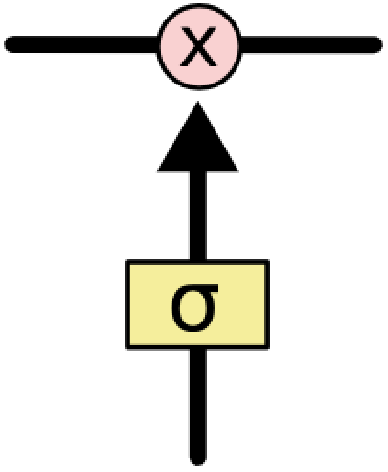
Tensorflow NOTE:

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**

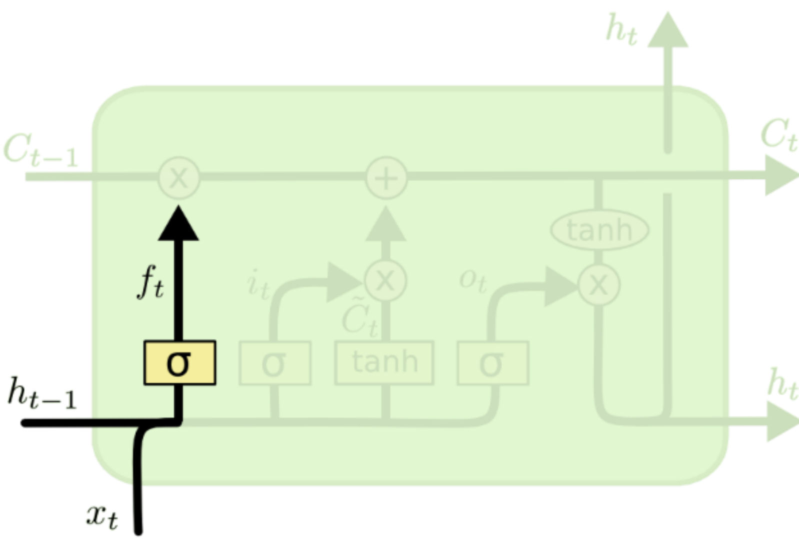
- The fundamental challenge of LSTMs and **keeping our state under control** is to be selective in three things:
  1. what we write (write selectivity),
  2. what we read (because we need to read something to know what to write) (read selectivity)
  3. and what we forget (because obsolete information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

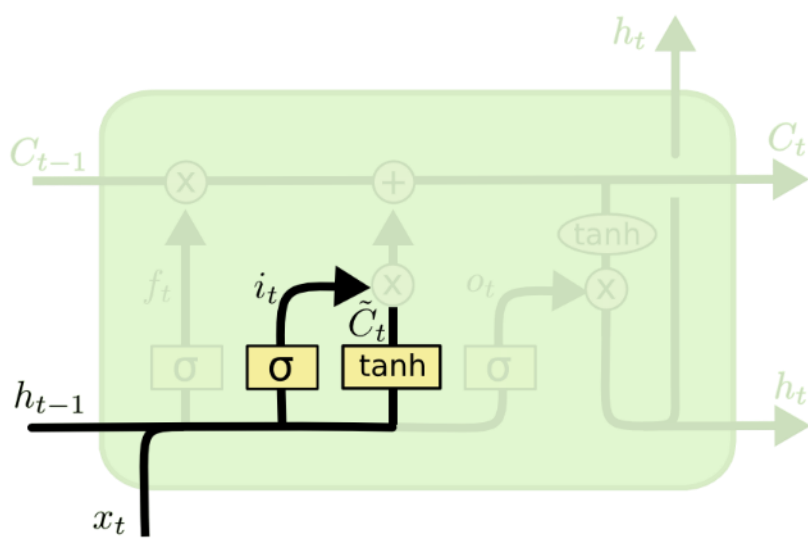


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

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.

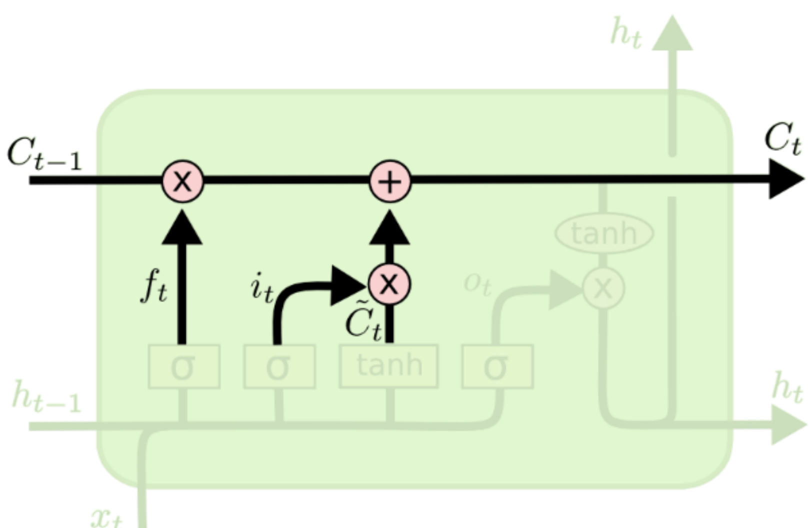


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Step-by-Step LSTM Walk Through

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

- 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 $i_t * \tilde{C}_t$



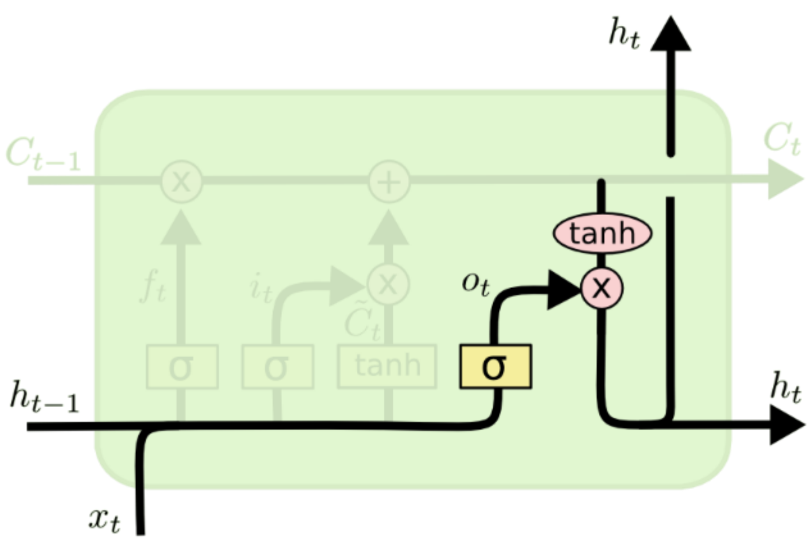
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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.



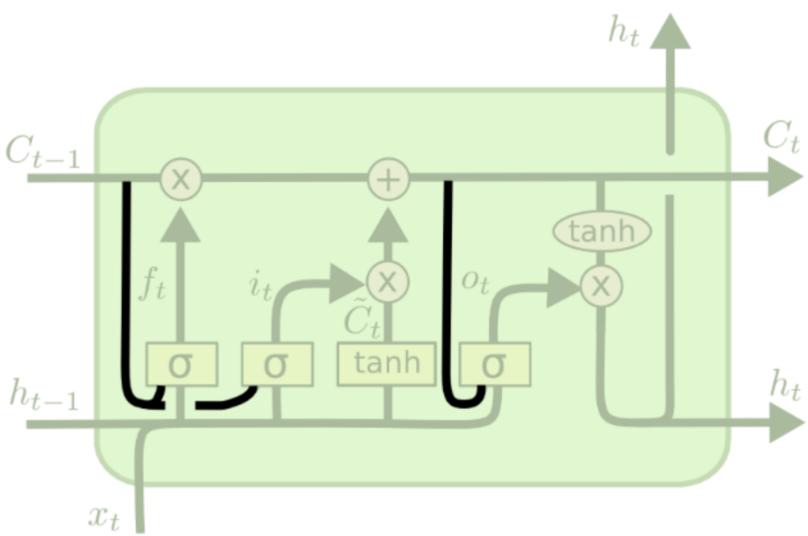
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

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.



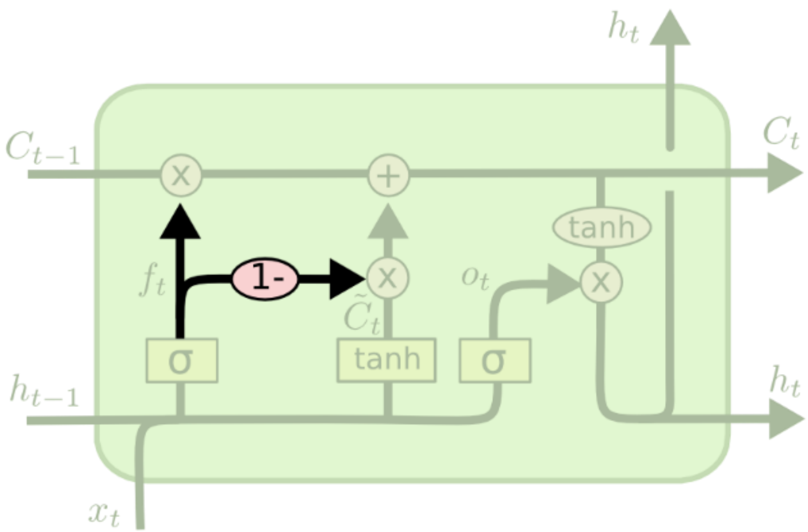
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

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



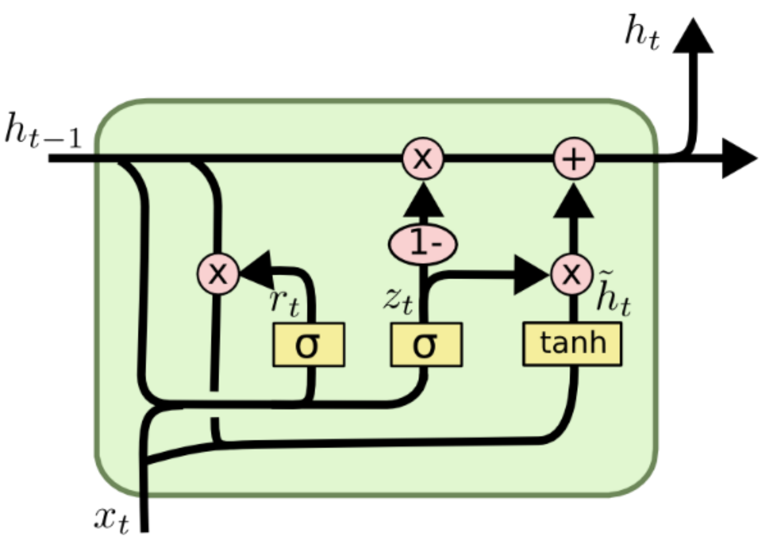
$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

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.



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

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

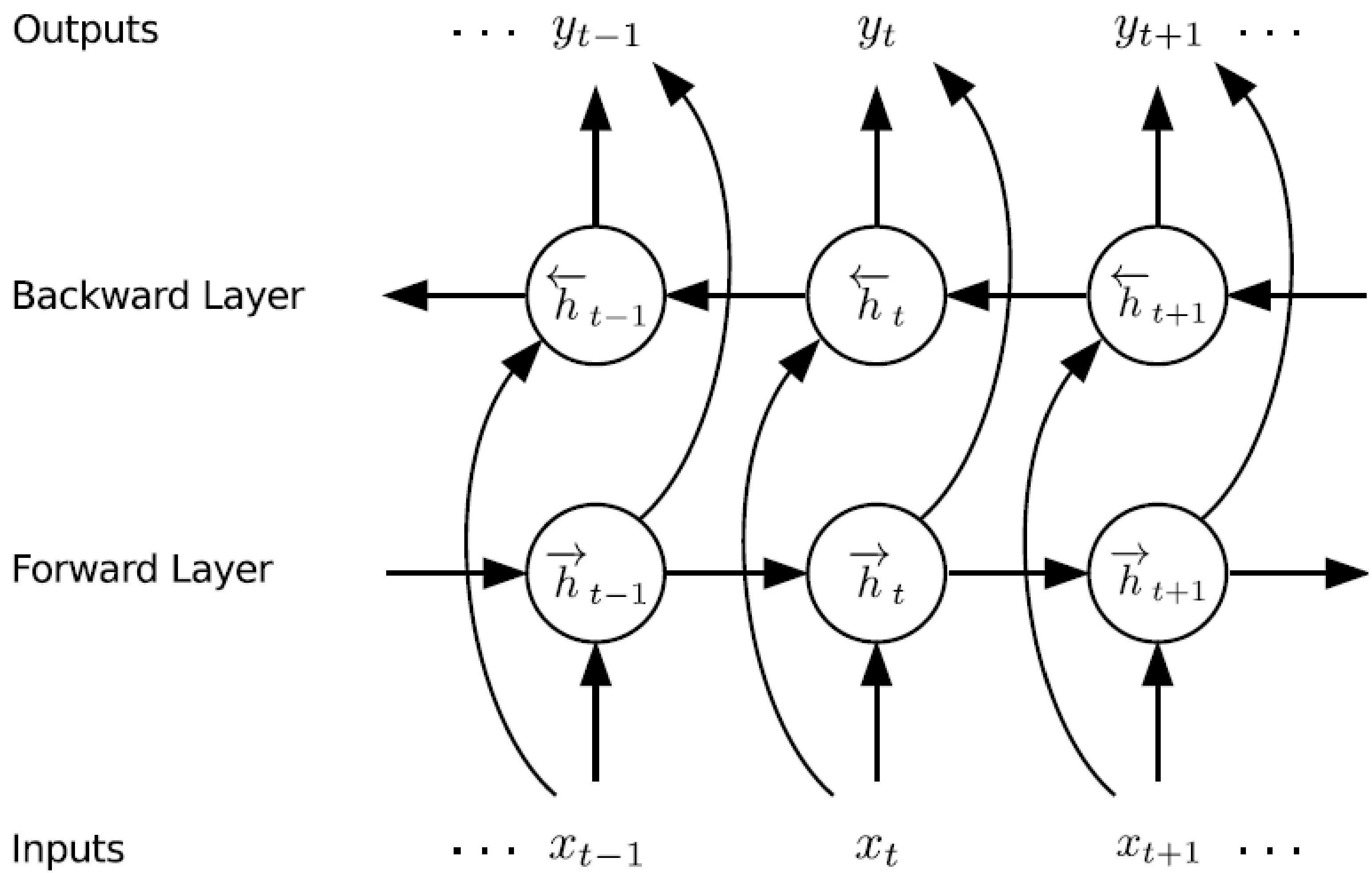
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Variants on Long Short Term Memory

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

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

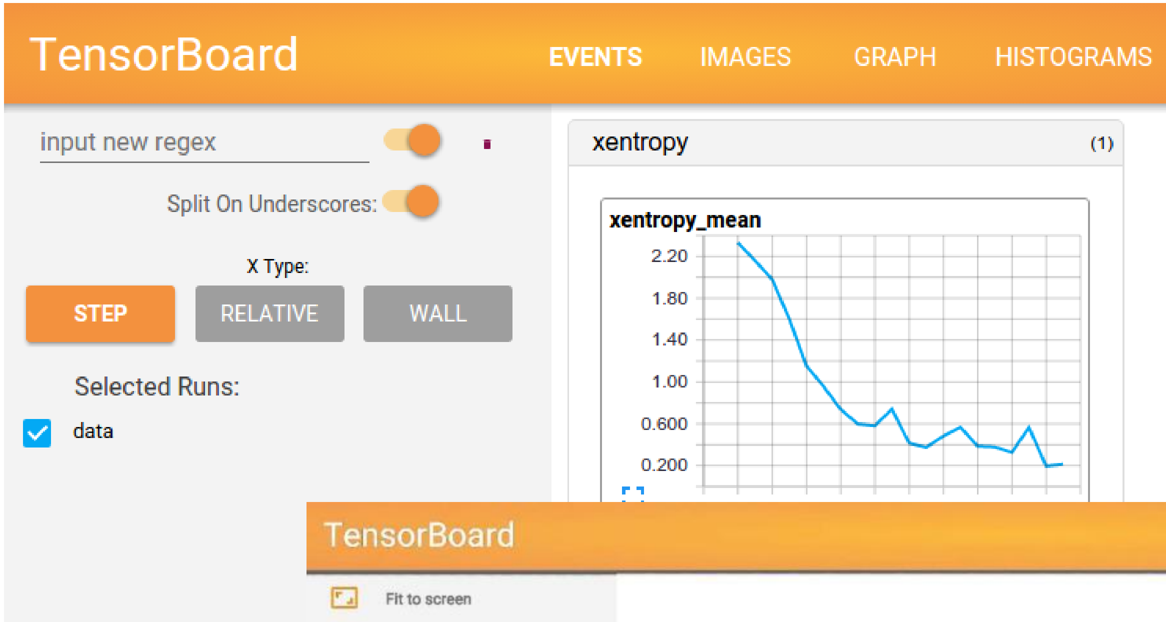


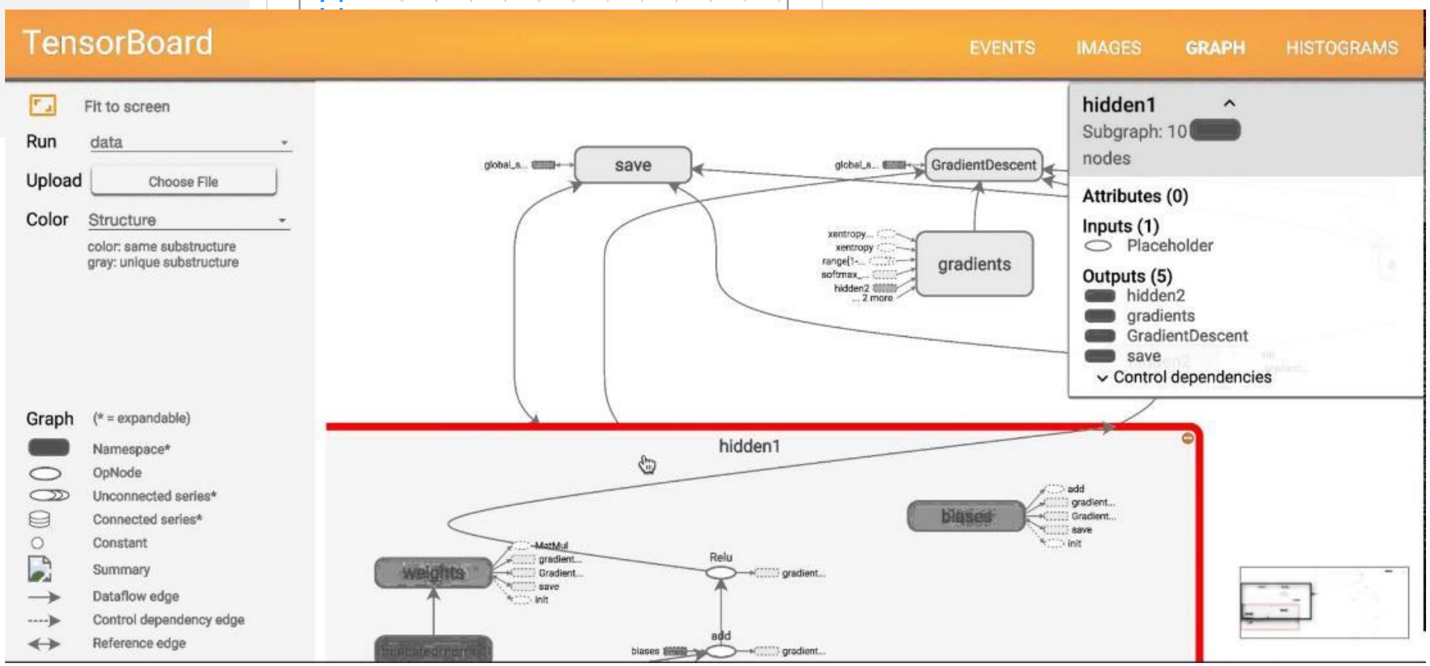
Variants on Long Short Term Memory

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.

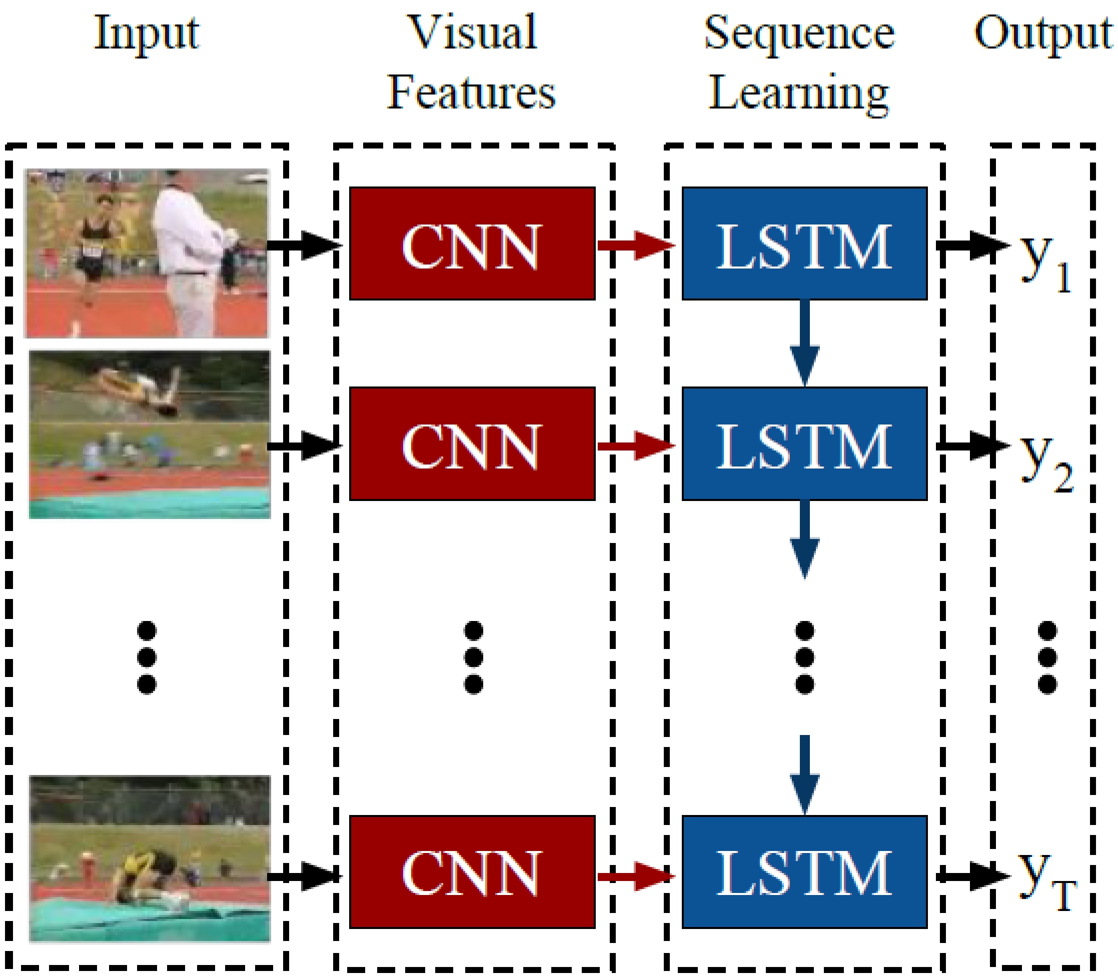
TensorBoard: Visualizing Learning





Remember you can combine different DL architectures

…sometimes CNNs are considered as feature extraction after RNN…



..for Eric Martín...

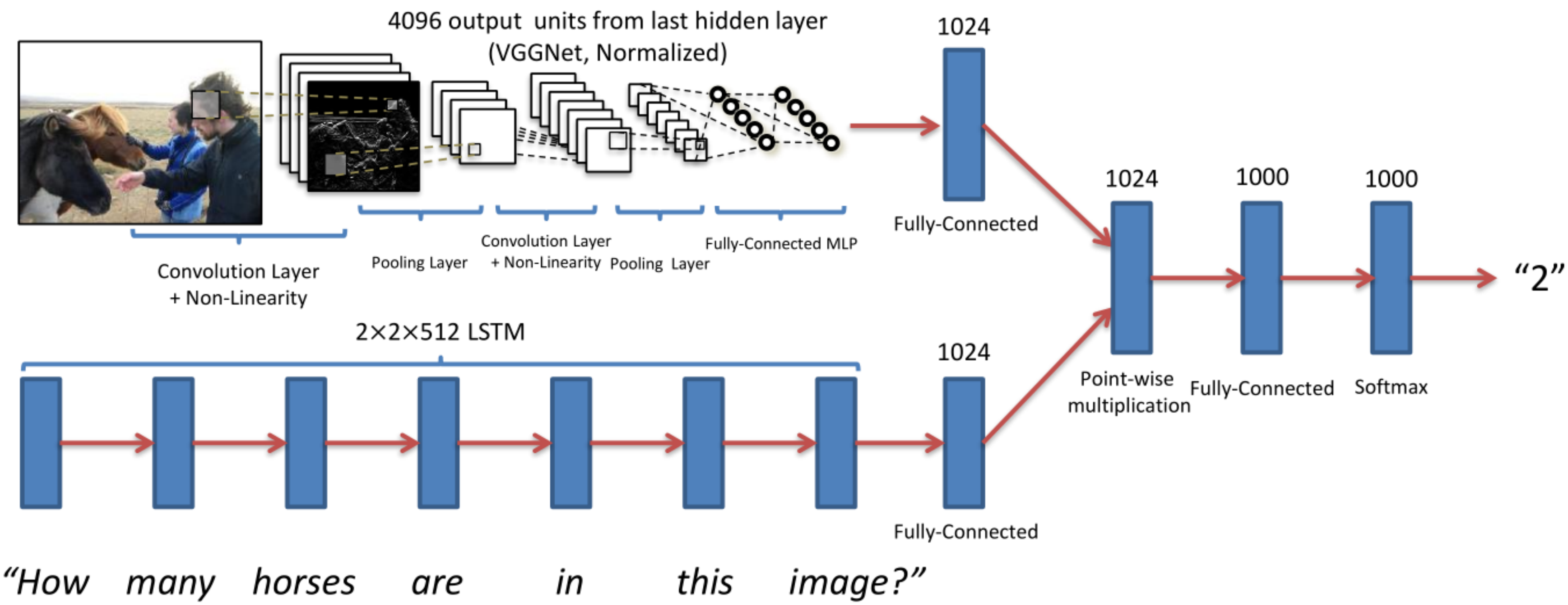
https://www.dukascopy.com/fxcomm/fx-article-contest/?Automated-High-Frequency-Trading-With=&action=read&id=1835

With the LSTM **we don't need to use any indicator because it's going to build its own indicators that are totally hidden from us**.

We only need to feed it the past tick to tick movements, one at a time, over the period we've chosen and it will predict the following movement.

Remember you can combine different DL architectures

…sometimes CNNs and RNNs are combined for modelling different multimodal sources…



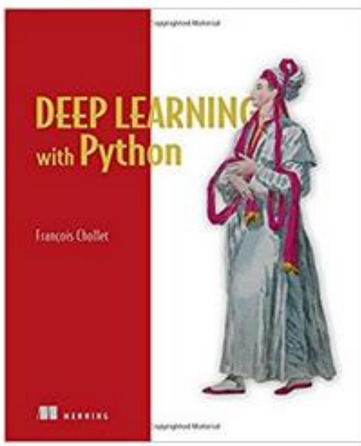
Embedding:

Indices by themselves, carry no **semantic** meaning

https://github.com/MasterMSTC/DeepLearning_TF_Keras

MSTC_Keras_RNN_3_Word_Embeddings_2018.ipynb

Word Embeddings with Keras



Adapted from:

6.1-using-word-embeddings

By François Chollet

Keras

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*.

Embedings

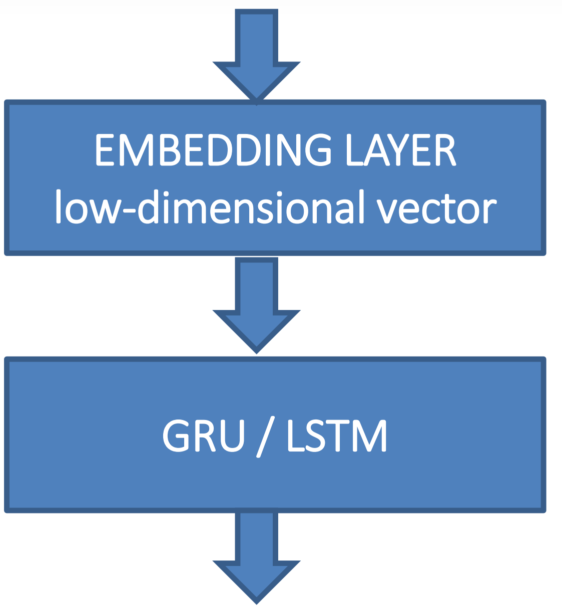
```
-
X information....
(32, 200)
<type 'numpy.ndarray'>
[[19 46 57 ..., 62 52 58]
 [45 52 57 ..., 50 39 62]
 [ 2 14 52 ..., 19 46 57]
 ...,
 [52 58  2 ..., 58 56 55]
 [ 2 47 52 ..., 45  2 60]
 [ 2 45 52 ..., 52 56 42]]
```

low-dimensional vector
(state_size = n_inputs)

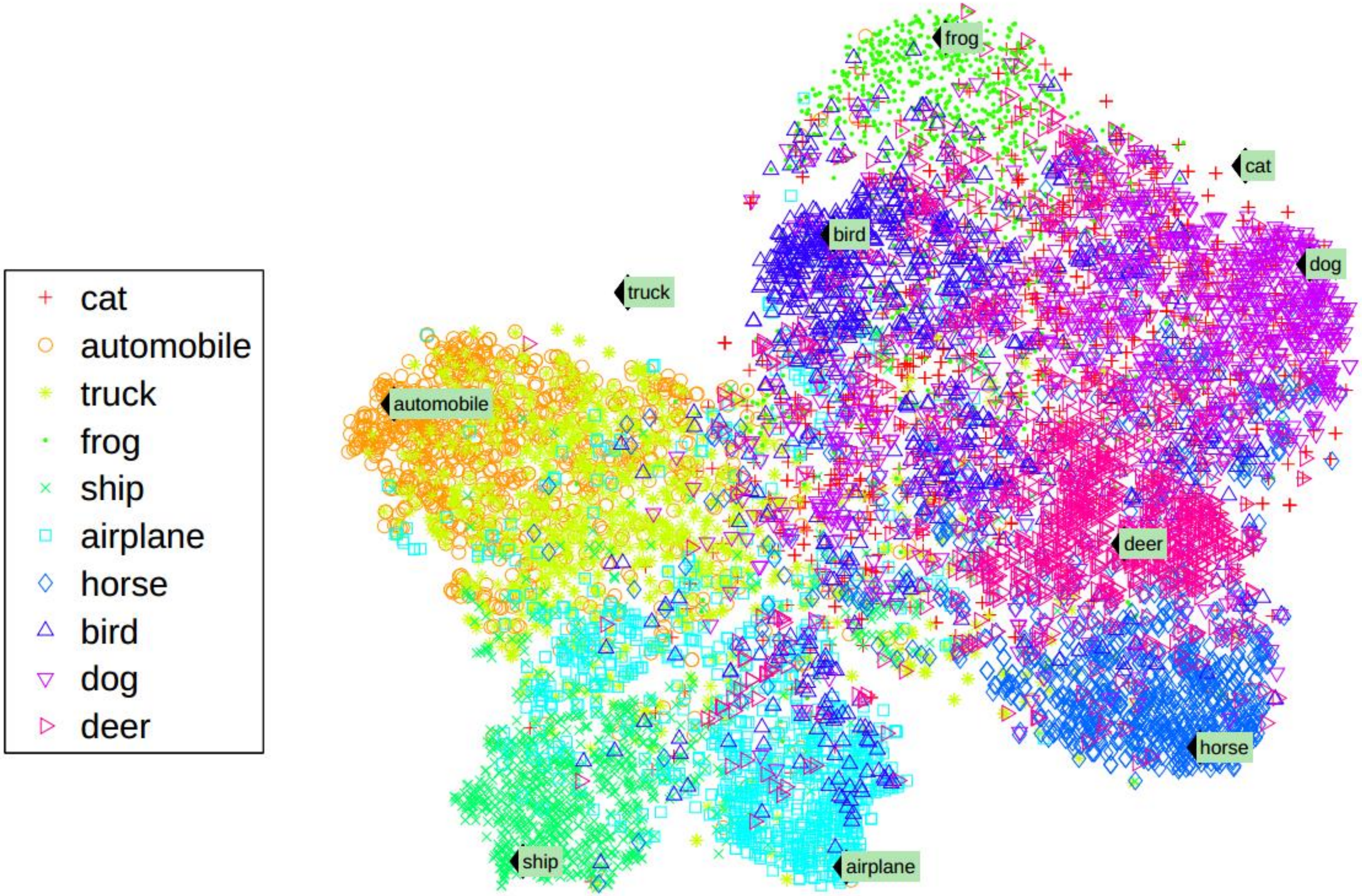
In this case, we will **map the characters to low dimensional vectors of size state_size.**

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



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

See also:

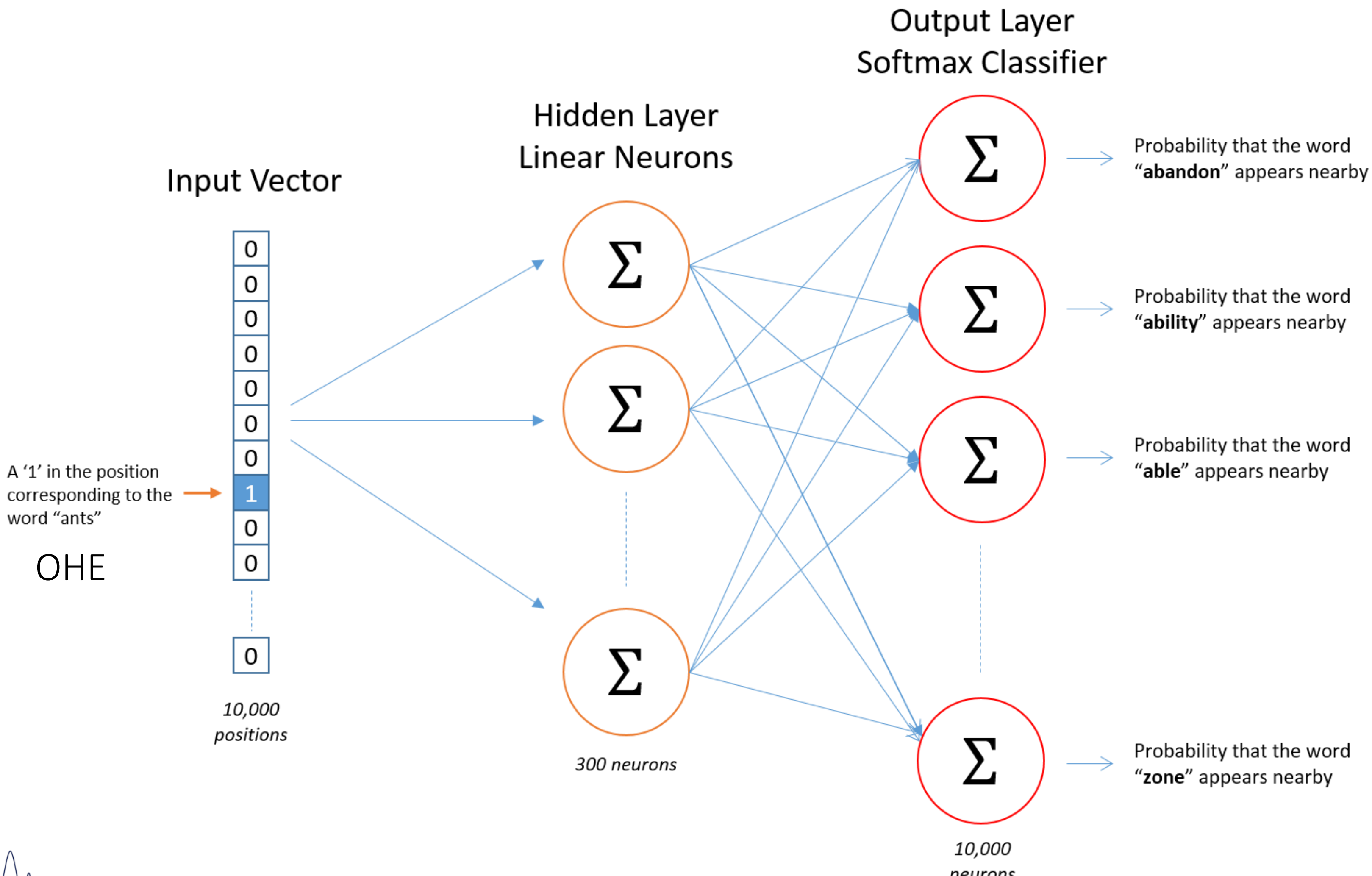
Word2vec https://www.tensorflow.org/tutorials/word2vec

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')

Word2Vec

https://www.youtube.com/watch?v=BD8wPsr_DAI

Sampled Softmax



Word2Vec

word2vec

(WATER - WET) + FIRE = FLAMES

(PARIS - FRANCE) + ITALY = ROME

(WINTER - COLD) + SUMMER = WARM

(MINOTAUR - MAZE) + DRAGON = SIMCITY

https://ronxin.github.io/wevi/

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