Recurrent Neural Networks (Ch. 15-16)

Wojtek Kowalczyk

wojtek@liacs.nl

Resources (old!)

Recurrent Neural Networks Tutorial (Denny Britz):

http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

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

http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

LSTM and GRU networks(Chris Olah):

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

General Intro (Andrej Karpathy):

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

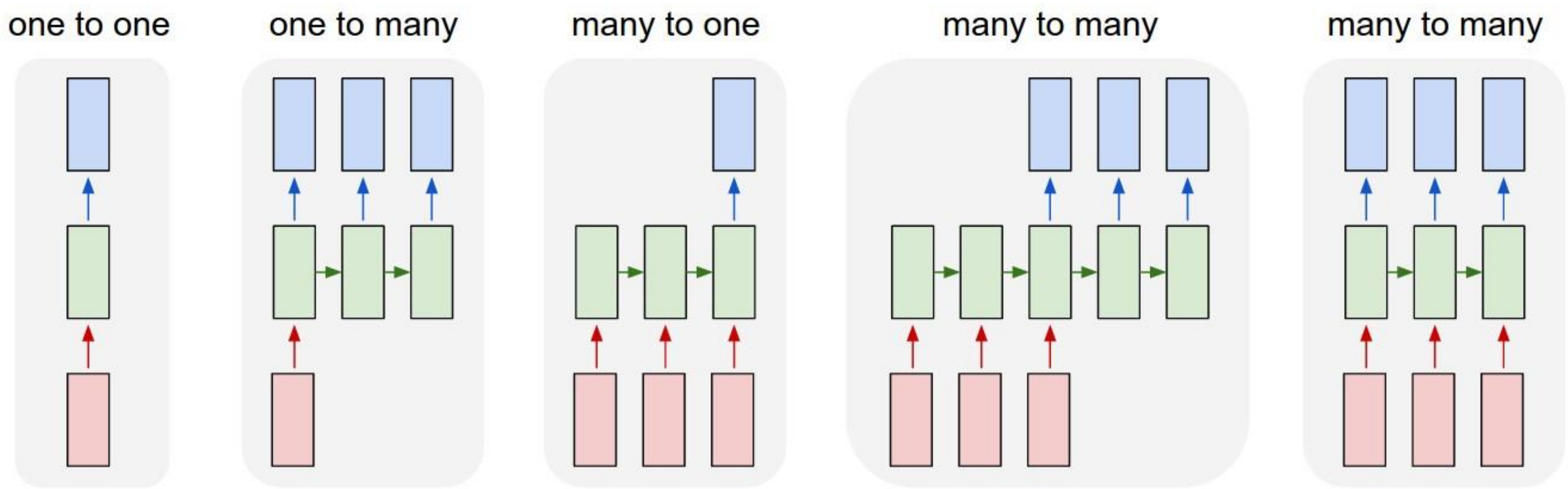
Agenda

- Motivation
- “Vanilla” Recurrent Networks
- Backpropagation Through Time
- Example: Language Modeling
- LSTM and GRU networks
- Word2Vec
- A **‘123+34’ -> ‘157’** example (adding two 3-digit integers)

Motivation: Sequential Data

- Forecasting weather conditions, traffic jams, exchange rates, ...
- Natural Language Processing (NLP):
  e. g., document classification, sentiment analysis, language translation, speech2text
- Automatic music composition
- Image/Video captioning
- Control (controlling robot arms/legs, steering a car, production processes ...)
- ...

Various Scenario’s



(1) Single Input -> Single Output (no recurrence)
(2) Single Input -> Sequence Output (e.g. image captioning)
(3) Sequence Input ->Single Output (e.g. sentiment analysis)
(4) Sequence Input -> Sequence Output (e.g. Machine Translation)
(5) Synced sequence input and output (e.g. classification of frames in a video)

(Karpathy, 2015)

Example: Language Modeling

Given a collection of correct sentences in English (sequences of words) – a training set - build a model can automatically complete unfinished sentences (or generates correct sentences from scratch).

V: a set of possible words (vocabulary)

For an unfinished sentence - a sequence of words (w1, w2, .., wk) we want to know, for every word v from V, the probability that v follows wk: P(w|w1, ..., wk).

E.g.:

P(cat |this, is, a)=0.03,

P(dog |this, is, a)=0.05,

P(this |this, is, a)=0.00,

...

Traditional Approach:

N-grams

- Task: compute probability of a sentence *W*

$$P(W) = \prod_i P(w_i | w_1 \ldots w_{i-1})$$

- Often simplified to trigrams:

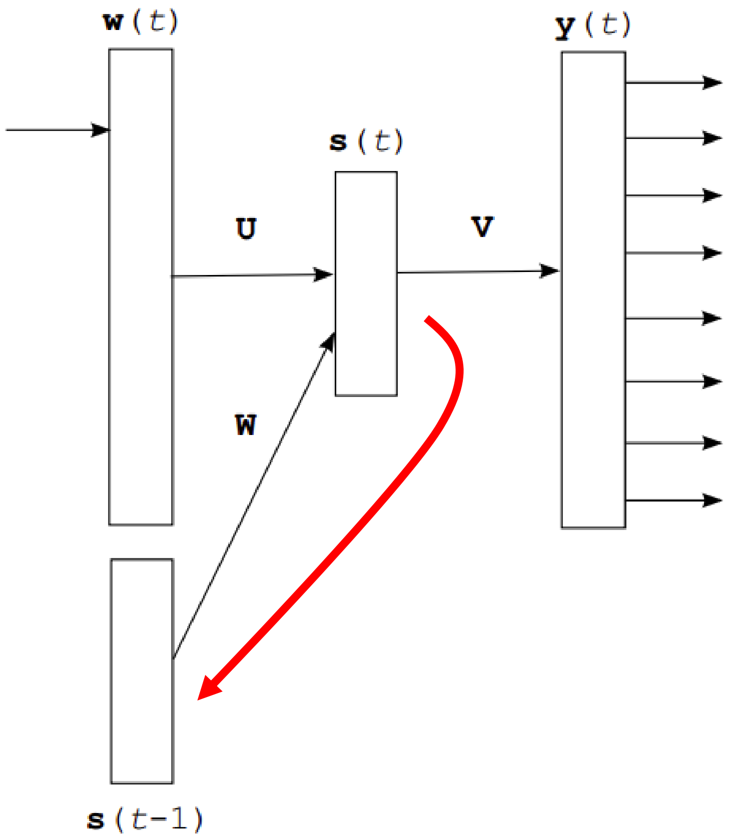
$$P(W) = \prod_i P(w_i | w_{i-2}, w_{i-1})$$

- The probabilities are computed using counting on training data

Huge number of parameters to estimate => doesn't work!

“Vanilla” Recurrent Neural Network

- |V|=n (Vocabulary: n words)
- words represented with “1 of n” coding: vectors of length n; all zero’s except of 1 that corresponds to the word
- input and output layers have n nodes
- output layer activated by soft-max function (can represent probability distribution over words)



“Vanilla” Recurrent Neural Network

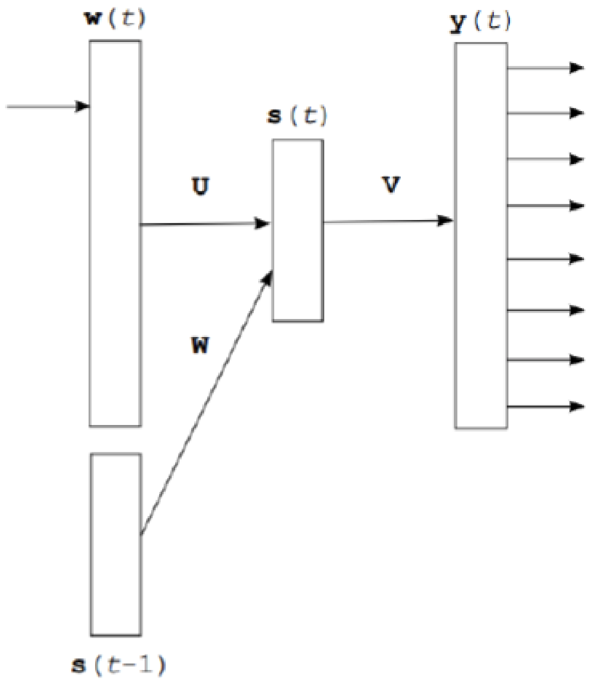
$$\mathbf{s}(t) = f(\mathbf{U}\mathbf{w}(t) + \mathbf{W}\mathbf{s}(t-1))$$
$$\mathbf{y}(t) = g(\mathbf{V}\mathbf{s}(t))$$

$f()$ is often sigmoid activation function:

$$f(z) = \frac{1}{1+e^{-z}}$$

$g()$ is often the softmax function:

$$g(z_m) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$



Backpropagation Through Time (BPTT)

- How to train the recurrent nets?
- 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)
- Recurrent net can be seen as a (very deep) feedforward net with shared weights

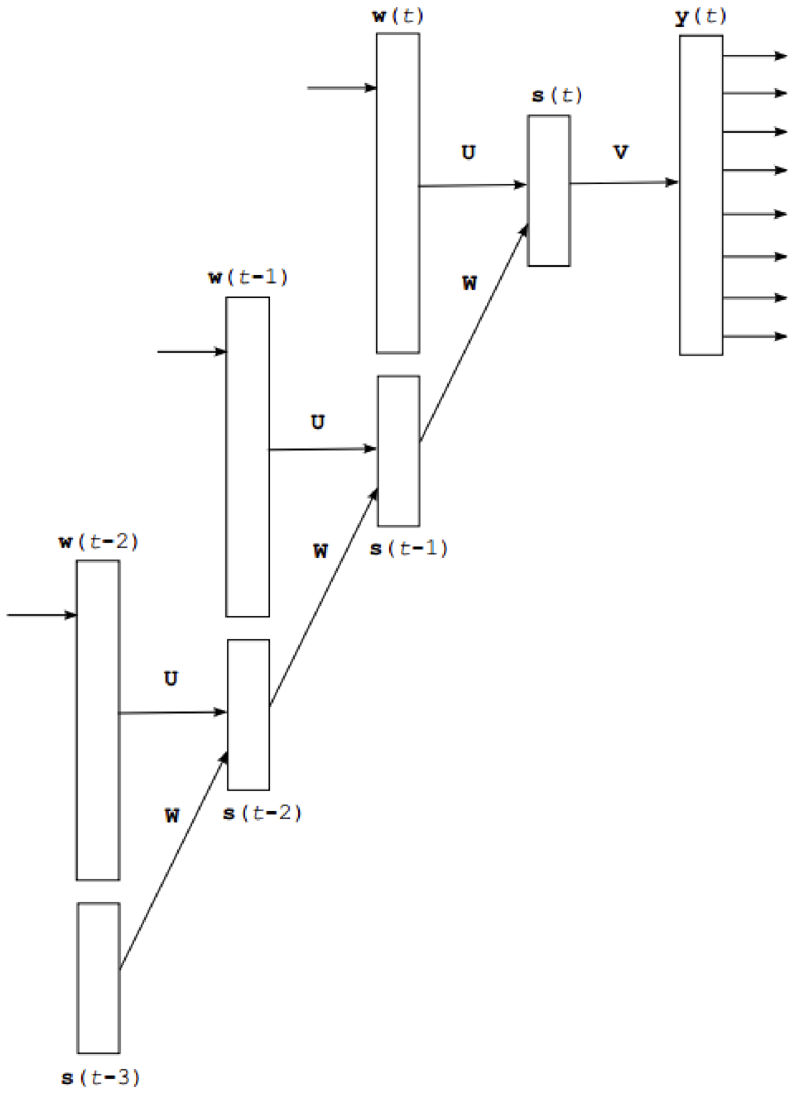
Backpropagation Through Time (BPTT)

“unfold the network over time”

- show the first word, compute the hidden state and error
- show the second word, compute the hidden state and the error
- ...

error(w_1, ..., w_k)=f(U,W, V, w_1., ..., w_k)

- use SGD to find the minimum !



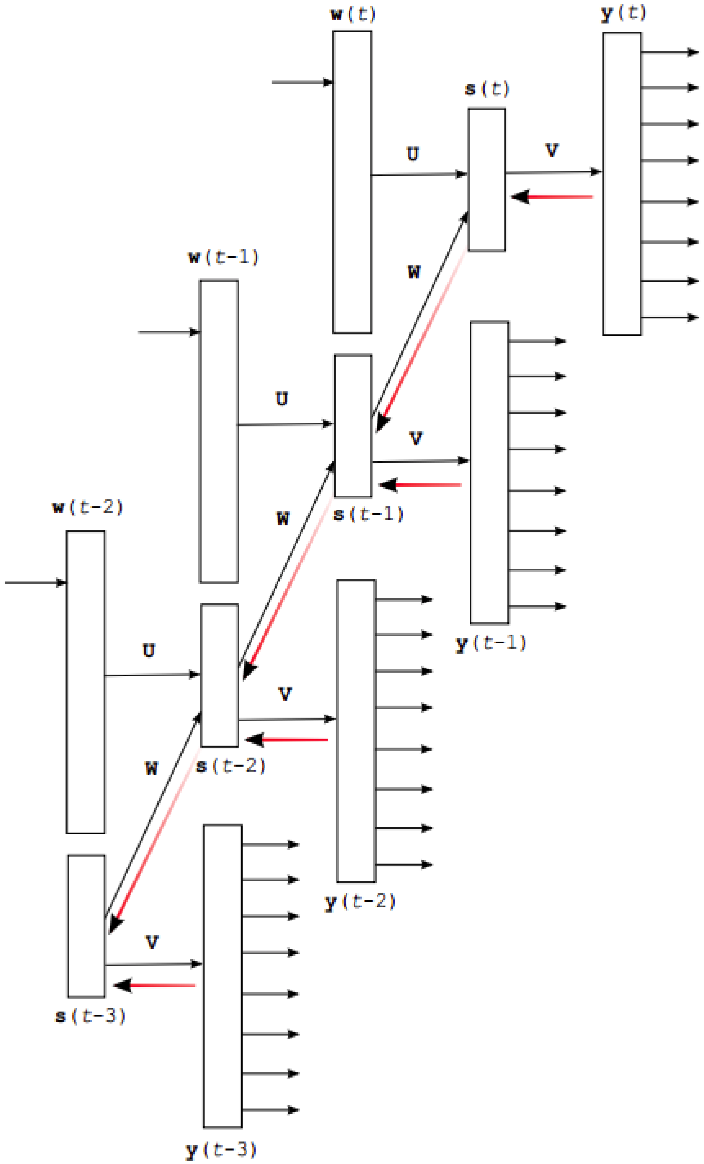
Backpropagation Through Time (BPTT)

“unfold the network over time”

error(w_1, ..., w_k)=f(U,W, V, w_1., ..., w_k)

SGD -> calculating gradients

- the longer the sequence the deeper error propagation!
- the problem of vanishing/exploding gradients!
- **Only short (5-10) sequences can be modeled this way!**

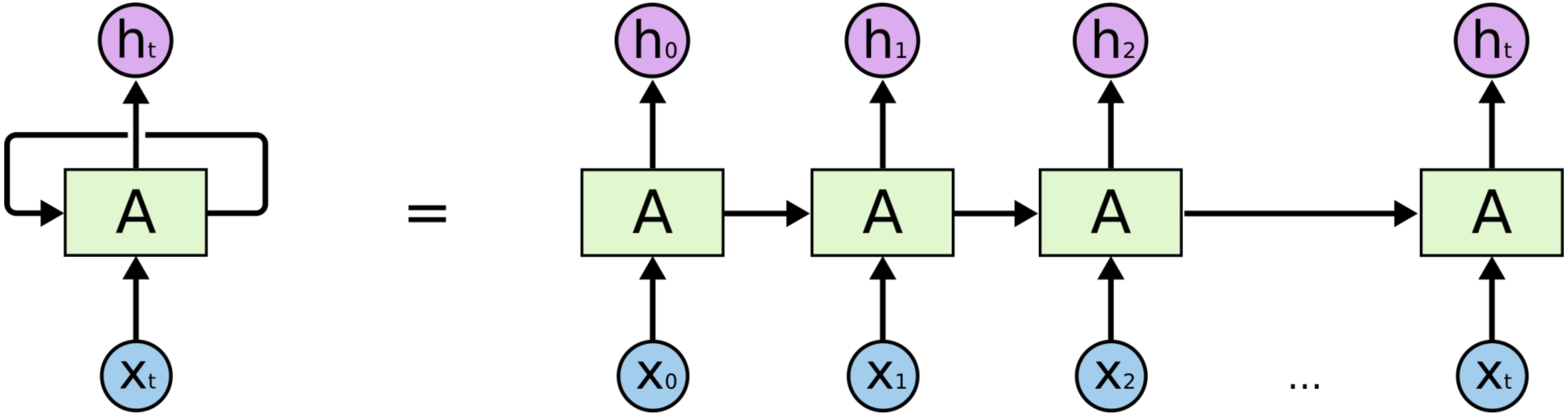


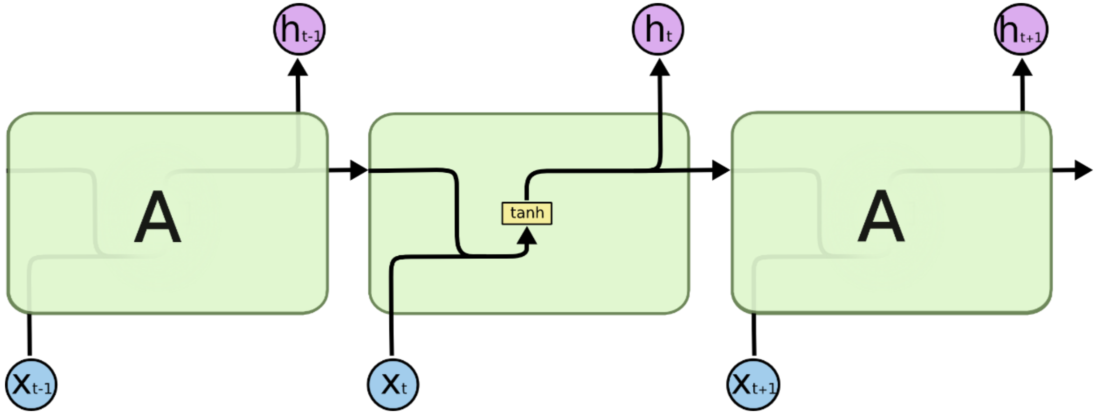
Long-Short Memory Networks

Key Ideas:

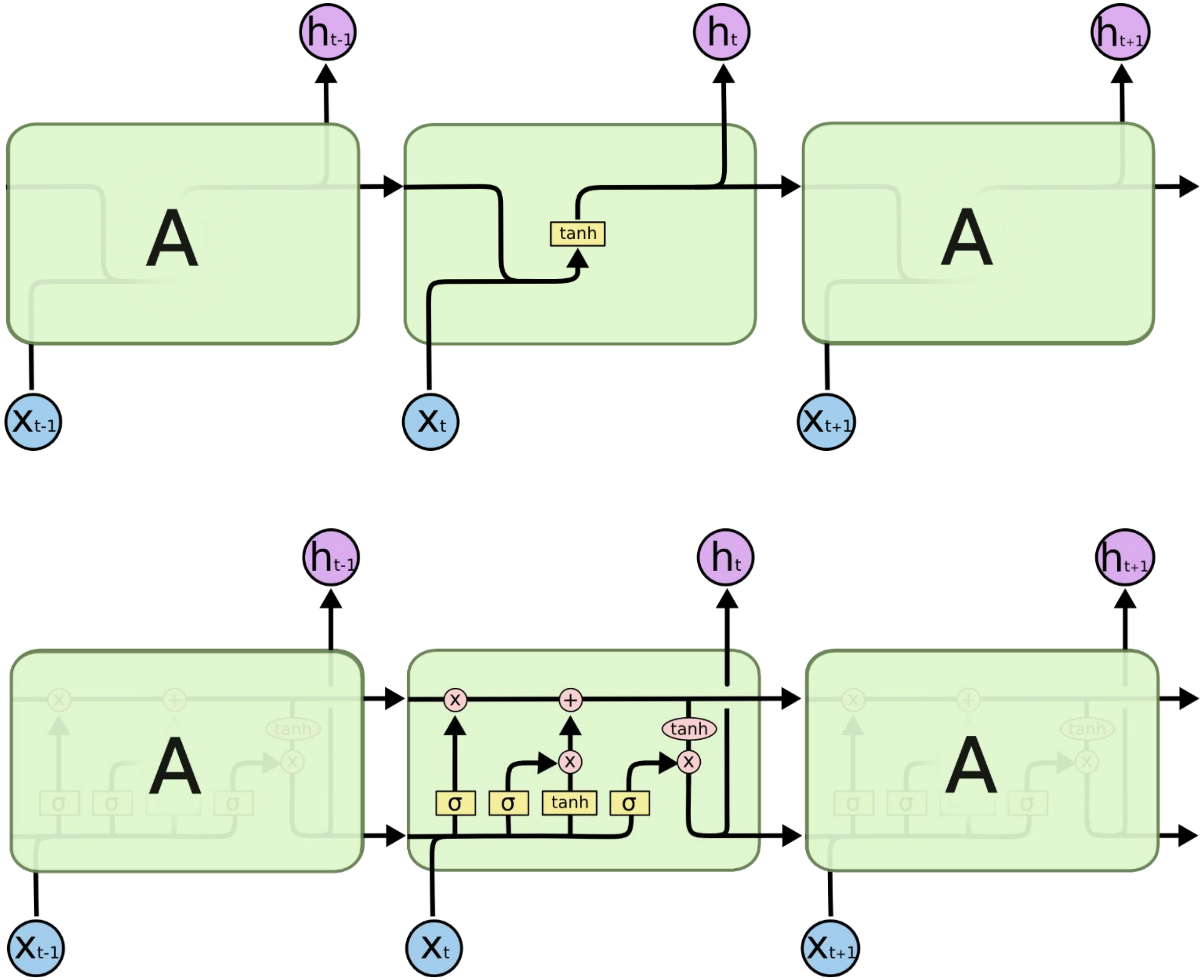
- extend the “short memory” hidden layer of a RNN by a mechanism that allows to “learn” which information should be preserved for a longer period and how should it be combined with the current data
- this “meta-information” should be kept in a “cell state” vector
- the hidden layer is replaced by a specially designed network

“Vanilla” Recurrent Network

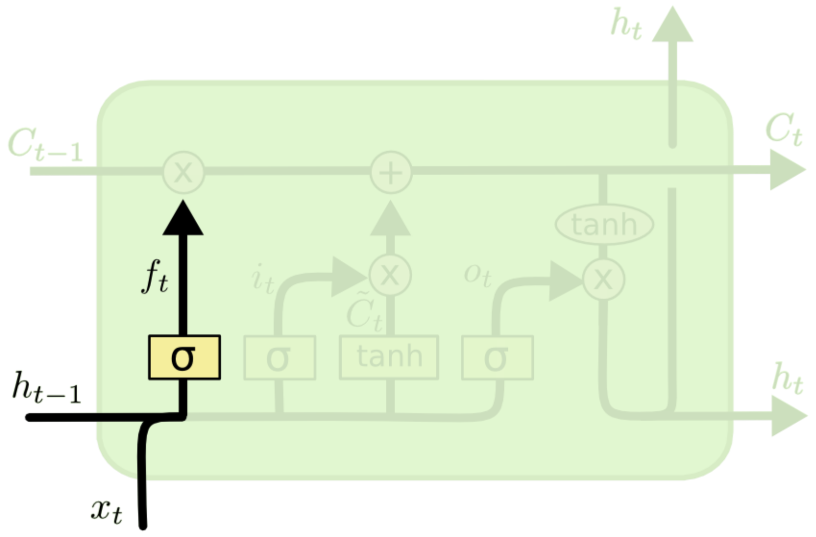




RNN versus LSTM Building Blocks



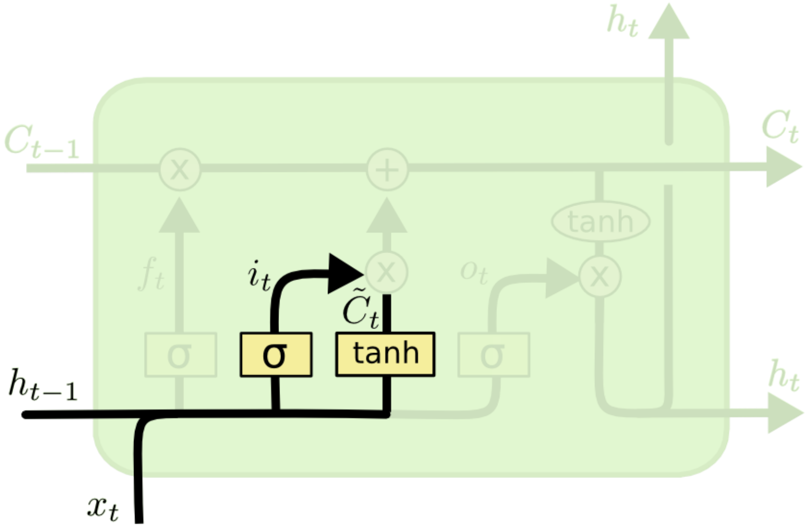
LSTM: forget gate



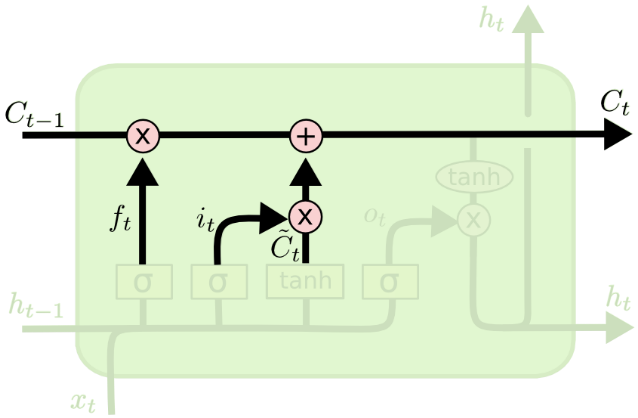
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

decides which information from the internal state should be *neglected* (sigmoid takes values in (0,1)

LSTM: input gate



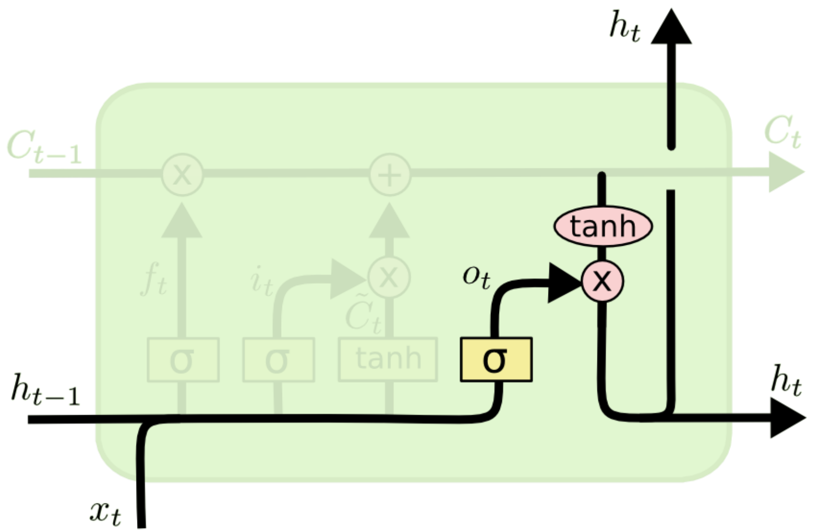
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



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

decides which information should be *added* to the internal state (*tanh* generates new information; *sigma* filters it)

LSTM: output gate

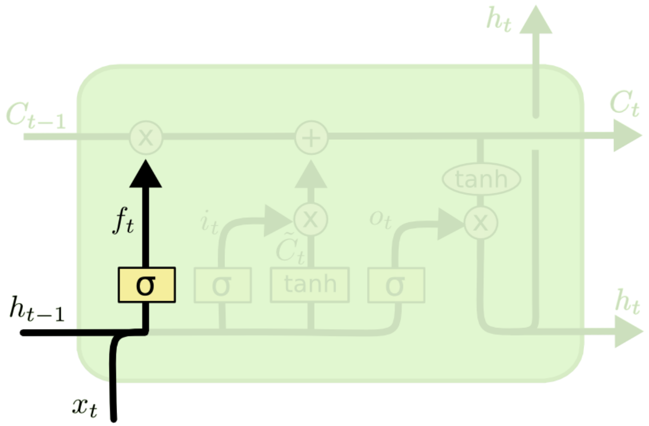


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

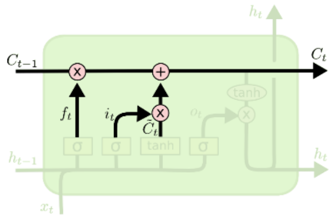
$$h_t = o_t * \tanh\left(C_t\right)$$

computes the output of the LSTM cell by combining
3 pieces of information: *new input*, *previous output*, *cell state*

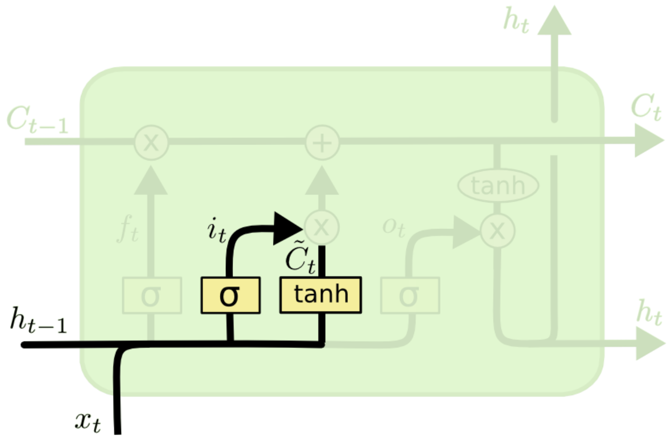
LSTM: Summary of parameters



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

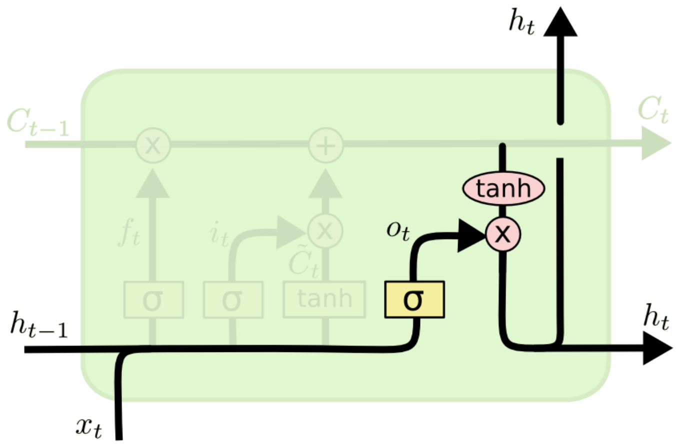


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



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

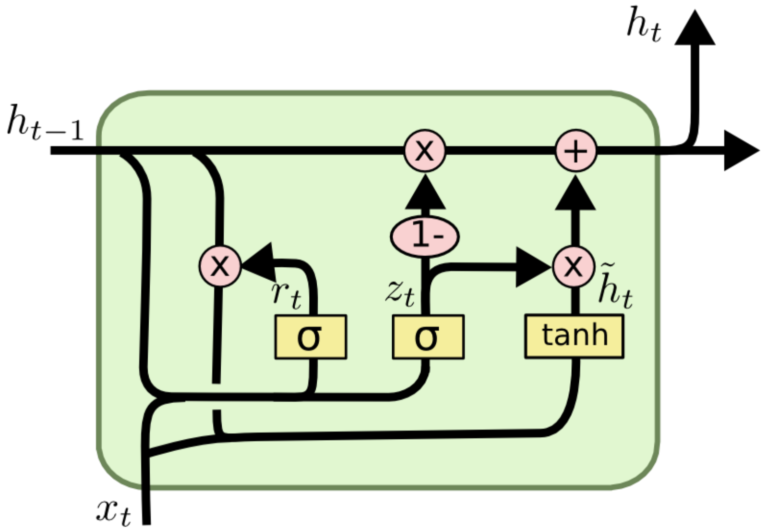
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



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

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

GRU: Gated Recurrent Unit



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

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

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

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

A simplified version of LSTM: only 2 gates and 3 matrices; no “cell state” (implicit in the output)

Remarks

- We discussed only “single recurrent layer” recurrent networks
- It is possible (and common) to stack several recurrent layers – the output of the first layer serves as an input to the recurrent layer, etc.
- It is possible to combine convolutional or dense layers with recurrent layers
- There are several variants of LSTM/GRU layers – it is not clear which one is best
- Additional links can be found at: https://github.com/kjw0612/awesome-rnn
- **Study Geron’s Chapters 15 and 16 for the recent state of the art!**

RNNs in Keras

1. https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html
2. https://github.com/keras-team/keras/blob/master/examples/addition_rnn.py
3. https://github.com/keras-team/keras/blob/master/examples/lstm_seq2seq.py
4. https://github.com/keras-team/keras/blob/master/examples/lstm_stateful.py

LSTM in more detail

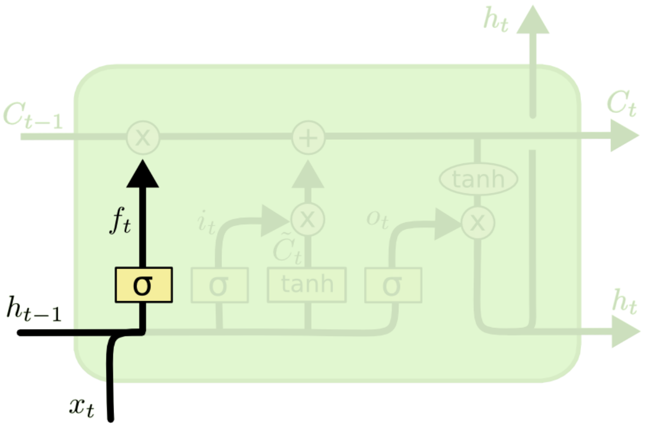
- h_t = “hidden state” = output of LSTM cell = “short term memory”
- C_t = “cell state” = “long term memory”
- “number of nodes of the LSTM layer” = $|h_t| = |C_t|$
- How many parameters has an LSTM with n inputs and m outputs?

- Two types of training: “stateless” and “statefull”
  - stateless: after processing a batch, network hidden state (h_t and C_t) is RESET
  - statefull: after processing a batch, network hidden state (h_t and C_t) is NOT RESET
- DEMO:

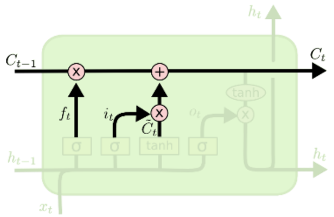
  https://keras.io/examples/nlp/addition_rnn/

- count trainable parameters
- change statefull to stateless and compare results

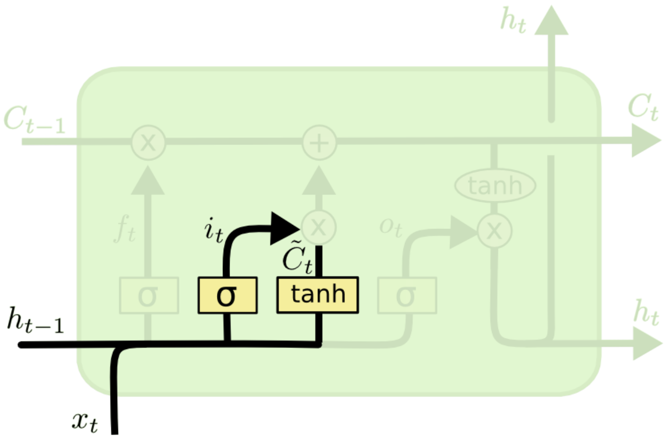
LSTM: Summary of parameters



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

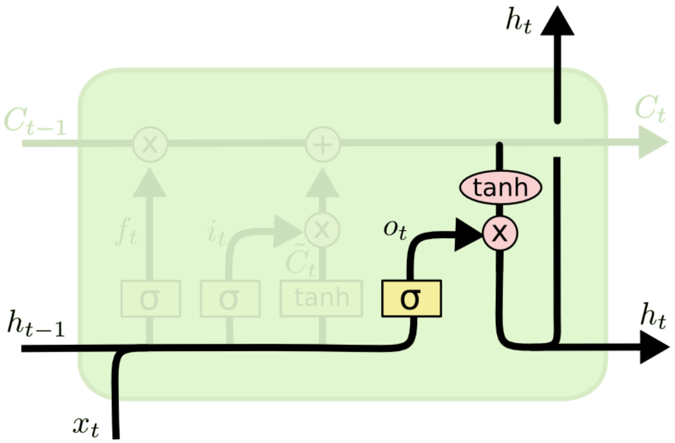


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



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

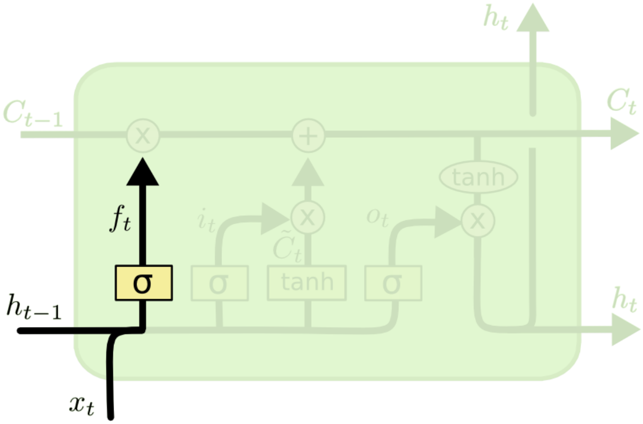
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



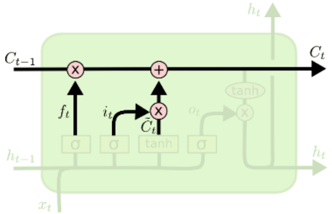
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

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

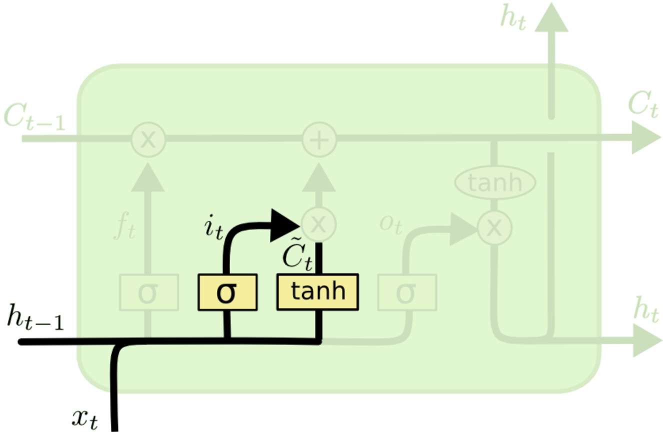
LSTM: Summary of parameters



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

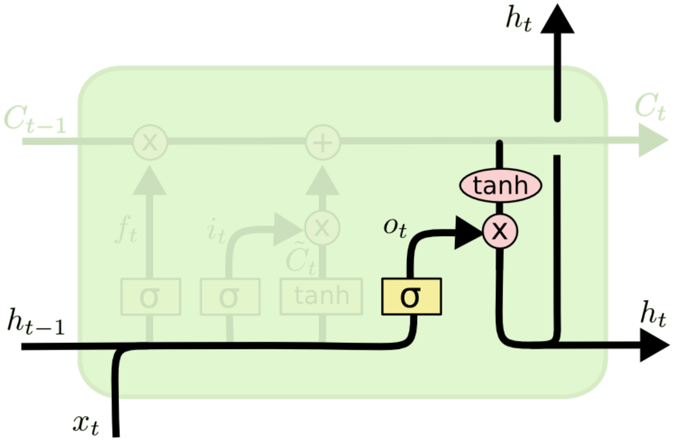


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



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

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



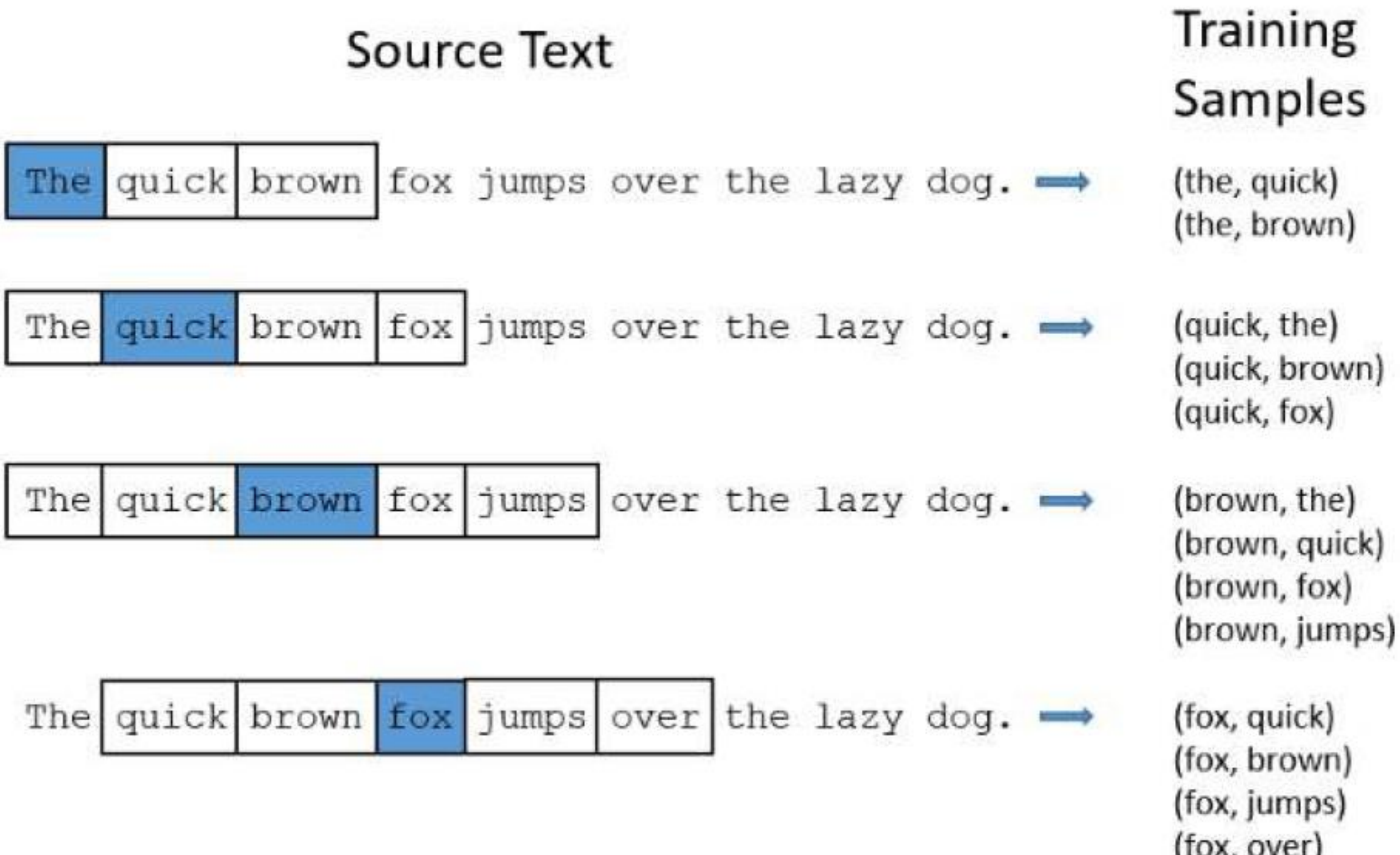
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

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

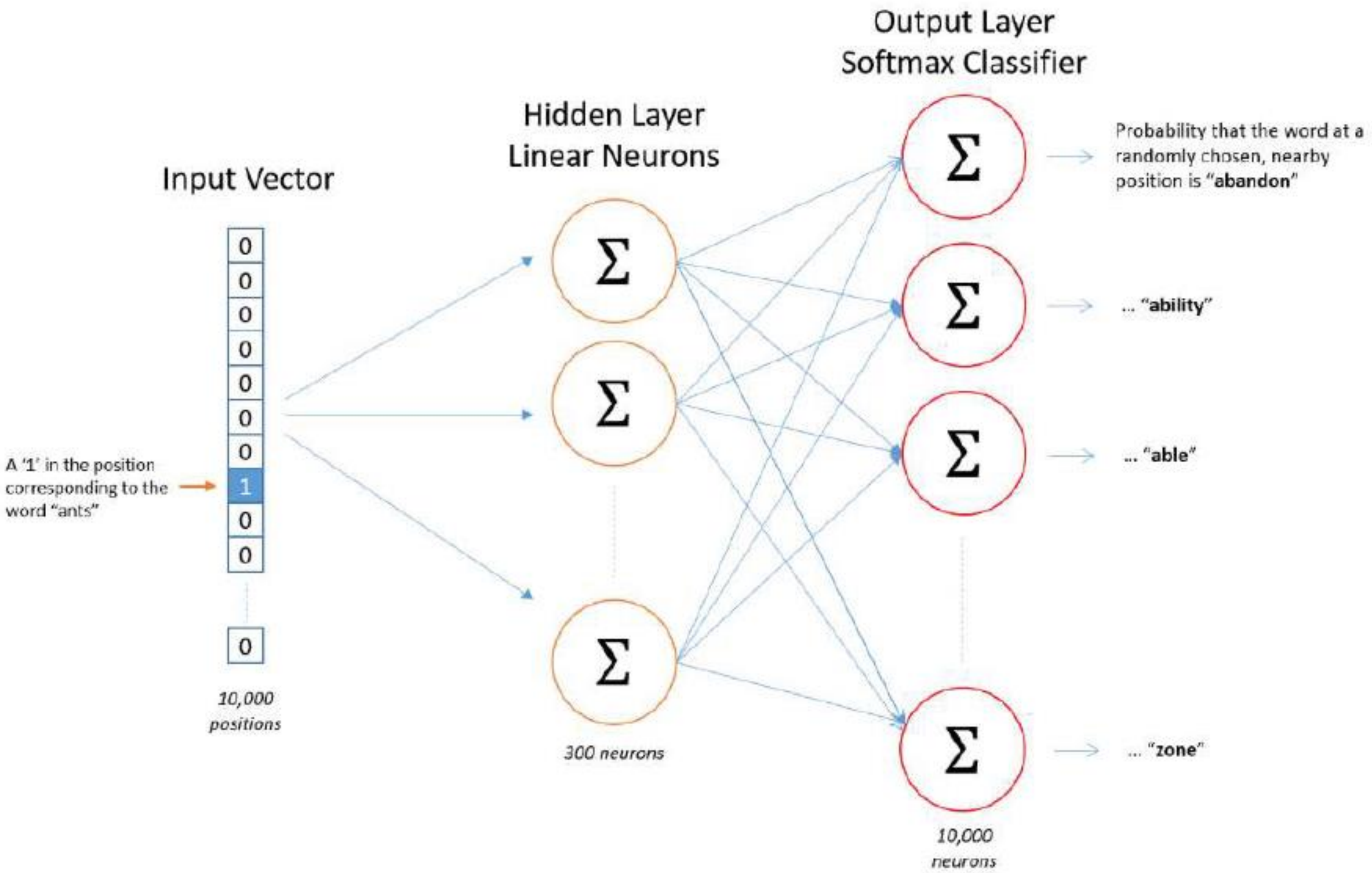
Word2Vec

- Key idea: instead of long “one-hot” vector representation of words we represent words with much shorter “vectors of floats”.
- Approach: take a large collection of documents, formulate a **simple task** (eg., predicting neighboring words in a sentence, next word, middle word, ...
- Train a network on your task, using one-hot word representation and treat the output of the first hidden layer (e.g., 100 nodes) as a vector that represents input words.

Word2Vec: the training set



Word -> 300 numbers



Resources

http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

https://code.google.com/archive/p/word2vec/

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

http://mccormickml.com/2016/04/27/word2vec-resources/

Mikolov's papers:
http://arxiv.org/pdf/1301.3781.pdf
http://arxiv.org/pdf/1310.4546.pdf