MSE | MASTER OF SCIENCE IN ENGINEERING

Recurrent Neural Networks

RNNs part 2

Long-term dependencies

LSTM - GRU

Word Embedding

Jean Hennebert
Martin Melchior

Overview

Noe Lutz - Engineering Lead in Google AI Brain Applied team in Zürich

Recaps RNN1

Long term dependencies

LSTM - GRU

Word embeddings - Word2Vec

Practical work - continuation

RNN1 recaps

RECAP - Sequence Models

Input sequence x with length T_x

$$x = (x_1, x_2, \ldots, x_{T_x})$$

→

Output sequence y with length T_y

$$y = (y_1, y_2, \ldots, y_{T_y})$$

Typically, the elements of x and y are ***multi-dimensional vectors*** and are ***parametrised*** by an ***index*** *or* ***'time'*** to describe the position in the sequence.

	image tagging, music generation	**sentiment analysis**	**named entity recognition**	**language translation, speech recognition, chatbots**
	one-to-many	**many-to-one**	**many-to-many**	**many-to-many**
out				
in				
	$T_x = 1, T_y \geq 1$	$T_x \geq 1, T_y = 1$	$T_x \geq 1, T_y \geq 1$ $T_x = T_y$	$T_x \geq 1, T_y \geq 1$ $T_x \neq T_y$

Example Today, the weather above the fog line is beautiful.

x_1 x_2 x_3 $\cdots$ x_9

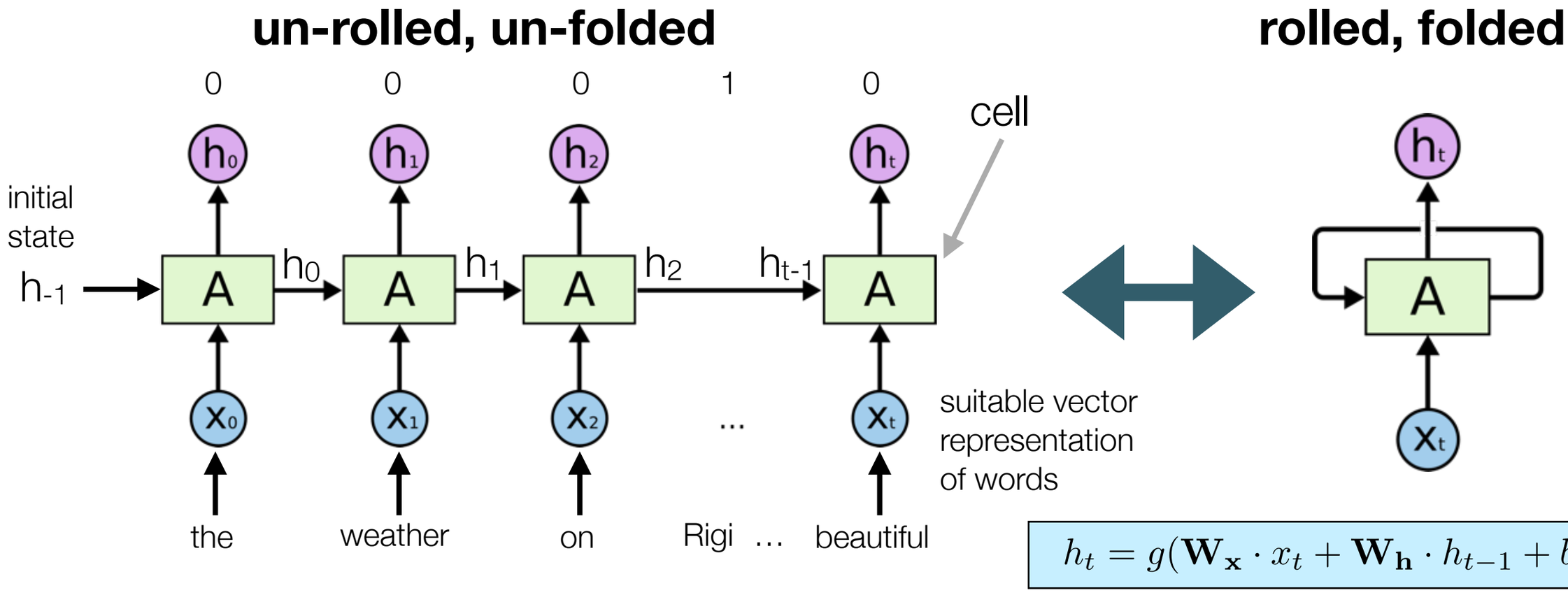
$T_x = 9$, word representation vectors

Heute ist das Wetter oberhalb der Nebelgrenze schön.

y_1 y_2 y_3 $\cdots$ y_8

$T_y = 8$, word representation vectors

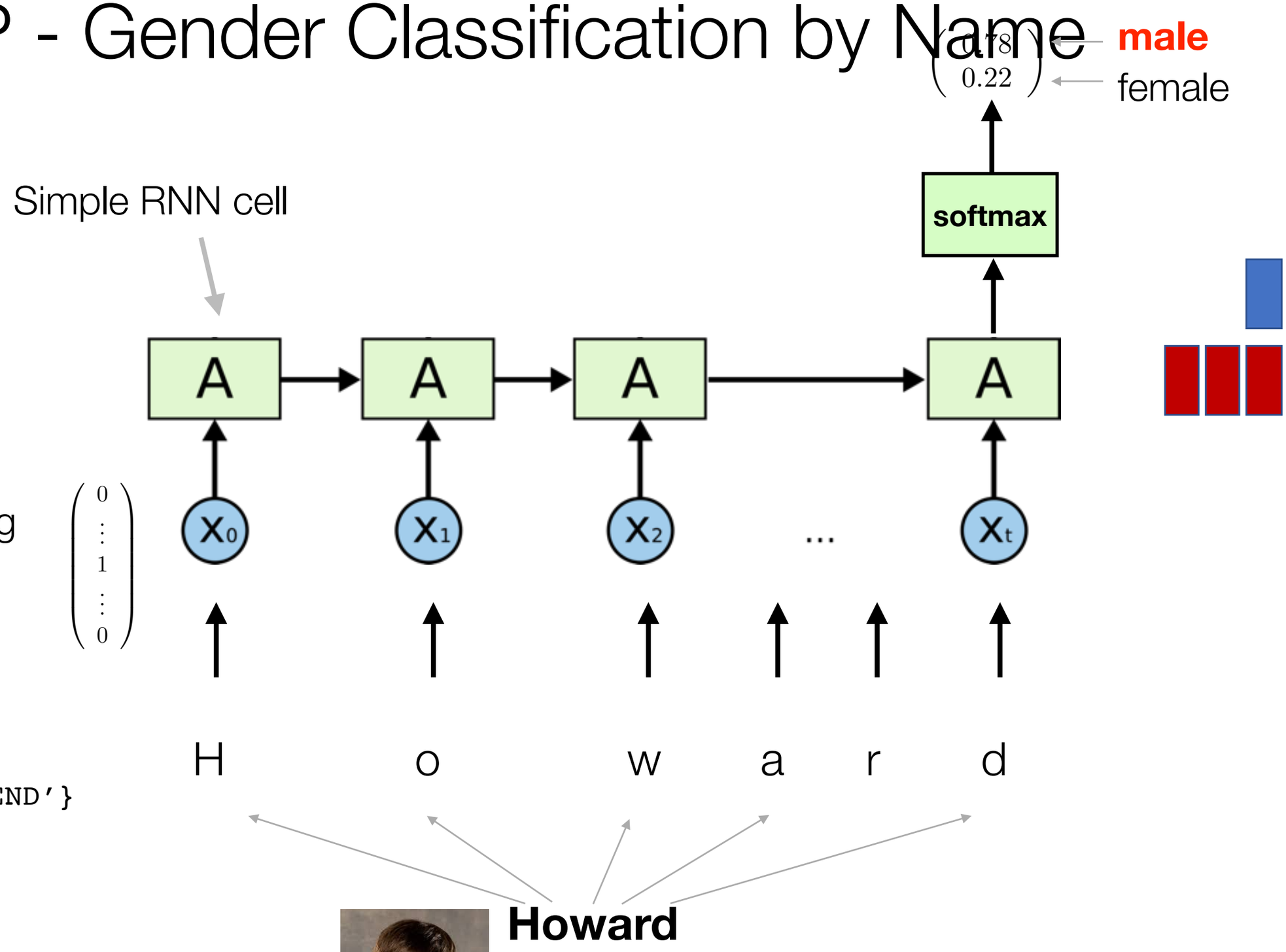
RECAP - Recurrent Cells



- **State** h is updated through a succession of steps. The state **memorises** (part of) the “history”.
- At update *t*, it consumes the next element of the input sequence (if available) and the previous state.
- State is passed on to
  - the next cell associated with step t+1 or
  - possibly to another layer (e.g. output layer)
- Before the first step, the state is suitably initialised: typically with zero values.
- All the cells of a layer share the parameters.

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

RECAP - Gender Classification by Name



one-hot vector with a ‘1’ at the position according to the position of the letter in the alphabet

$$\begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

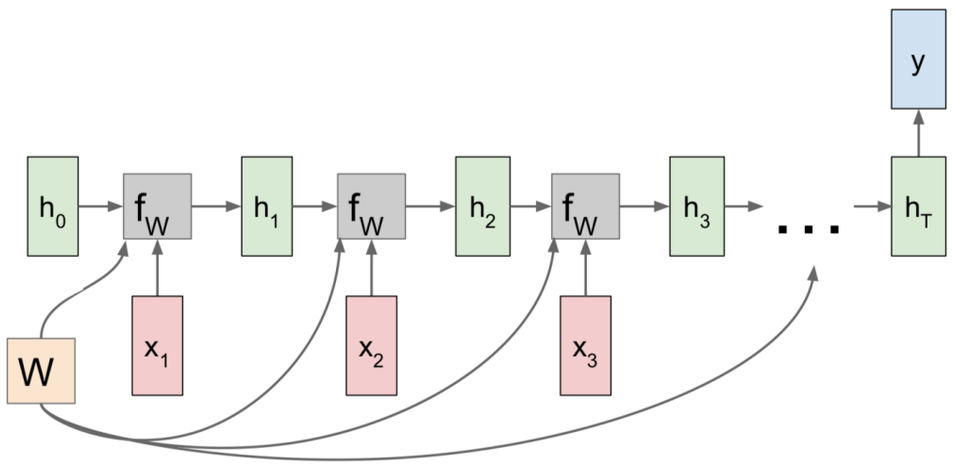
```
{'A', 'B', ‘C’,…,
 'a', 'b', ‘c’,…,
 ‘-‘, “‘“, ‘ ‘,’END’}
```

56 dimensions

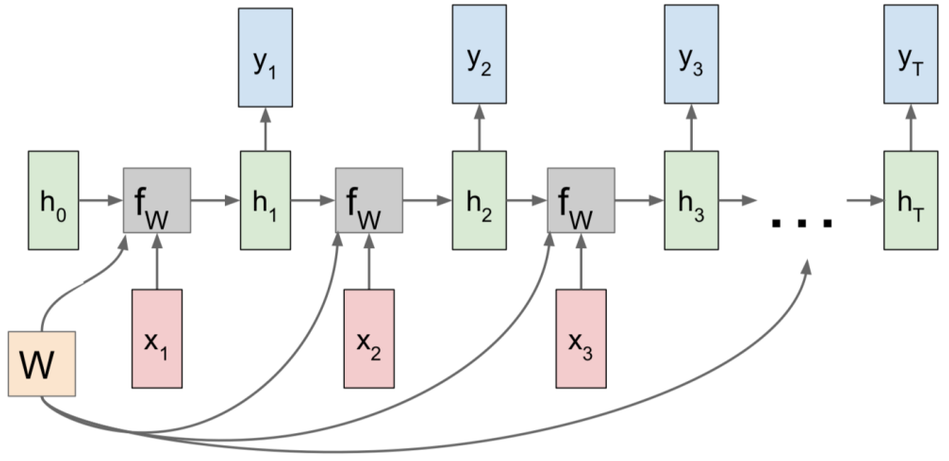
RECAP - Generative RNNs

A **generative system** is a system able to generate new consistent data from a **seed**. By consistent, we mean respecting temporal or spatial “structures” that have been learned from the input space.

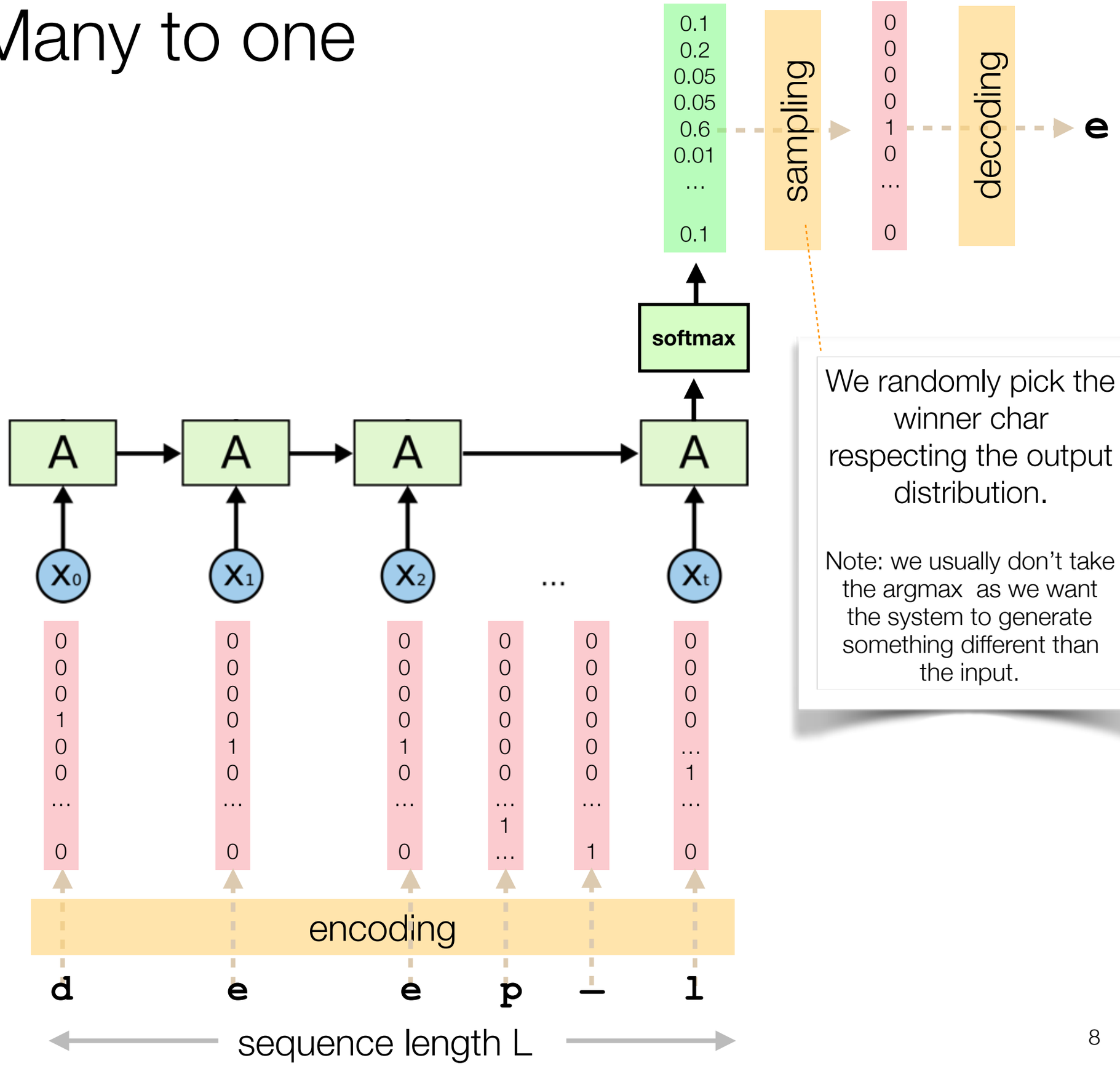
Many to one



Many to many



RECAP - Many to one approach



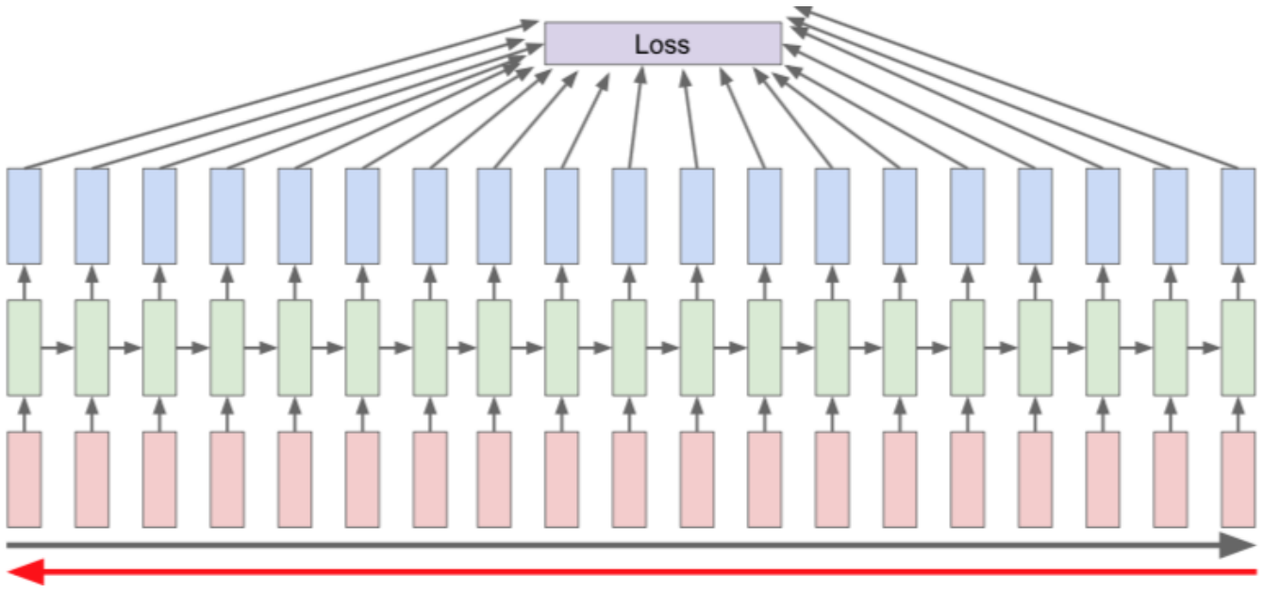
We randomly pick the winner char respecting the output distribution.

Note: we usually don't take the argmax as we want the system to generate something different than the input.

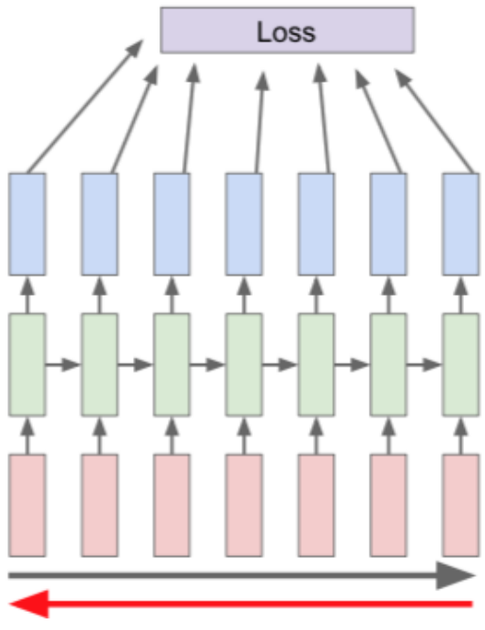
One-hot vector with a '1' at the position according to the position of the letter in the alphabet size D

```
{'a', 'b', 'c',…,
 'A', 'B', 'C',…,
 '-', "'",
'_','END'}
```

RECAP - Many to many approach: Truncated back propagation through time



In theory: forward through the entire sequence to compute loss, then backward through the entire sequence to compute gradient. But not really feasible if the sequence is long.



In practice: run forward and backward through chunks of the sequence instead of whole sequence

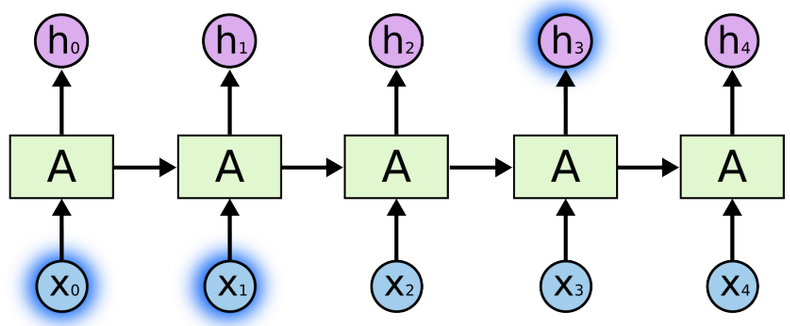
Long-Term Dependencies

Problem of Long-Term Dependencies

RNNs help wherever we need context from the previous input. They connect information from the past to perform a current task.

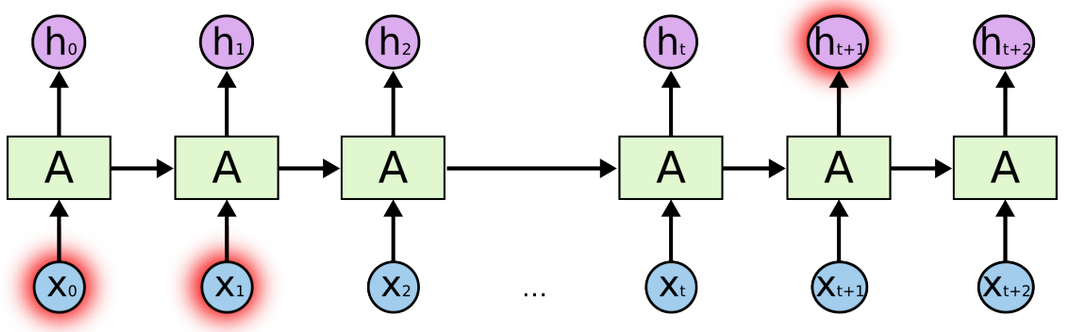
- For simple RNN's this works well if only the recent past or the close environment is needed.

  Example: "the clouds are in the ____" [sky]

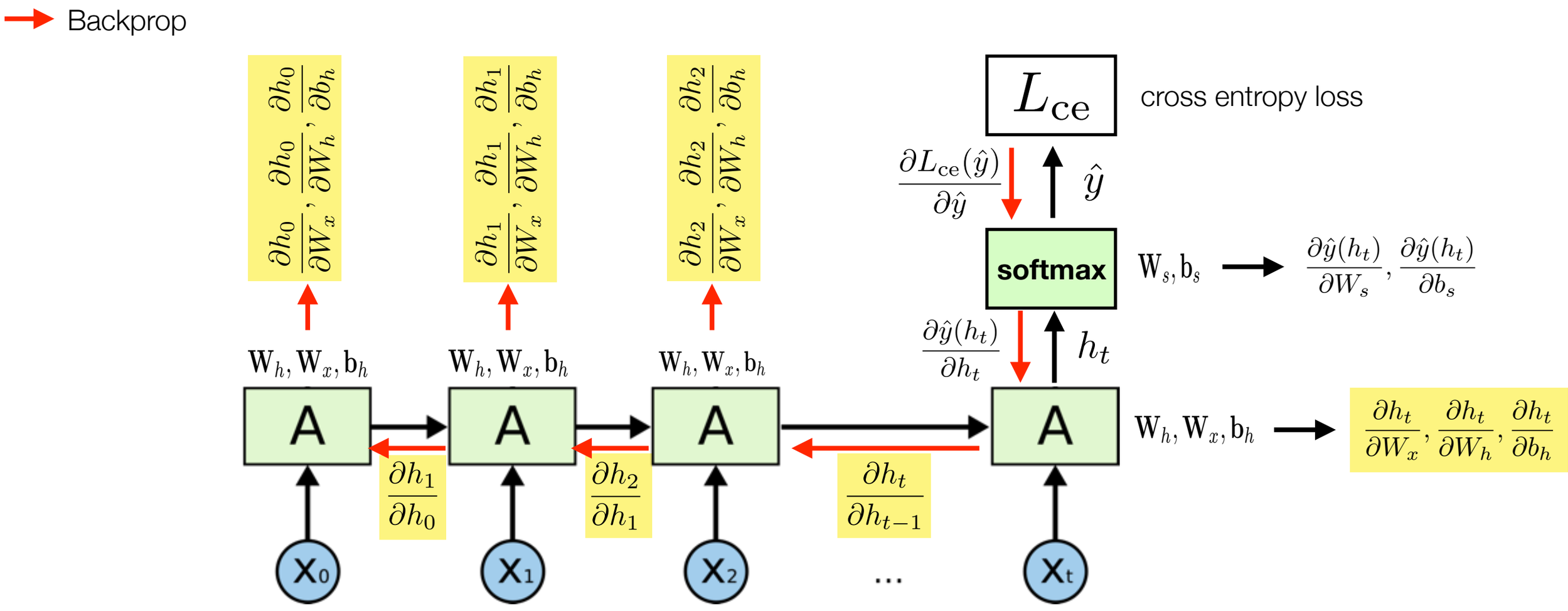


- It fails if a wider context is needed and the gap between the words grows too large.

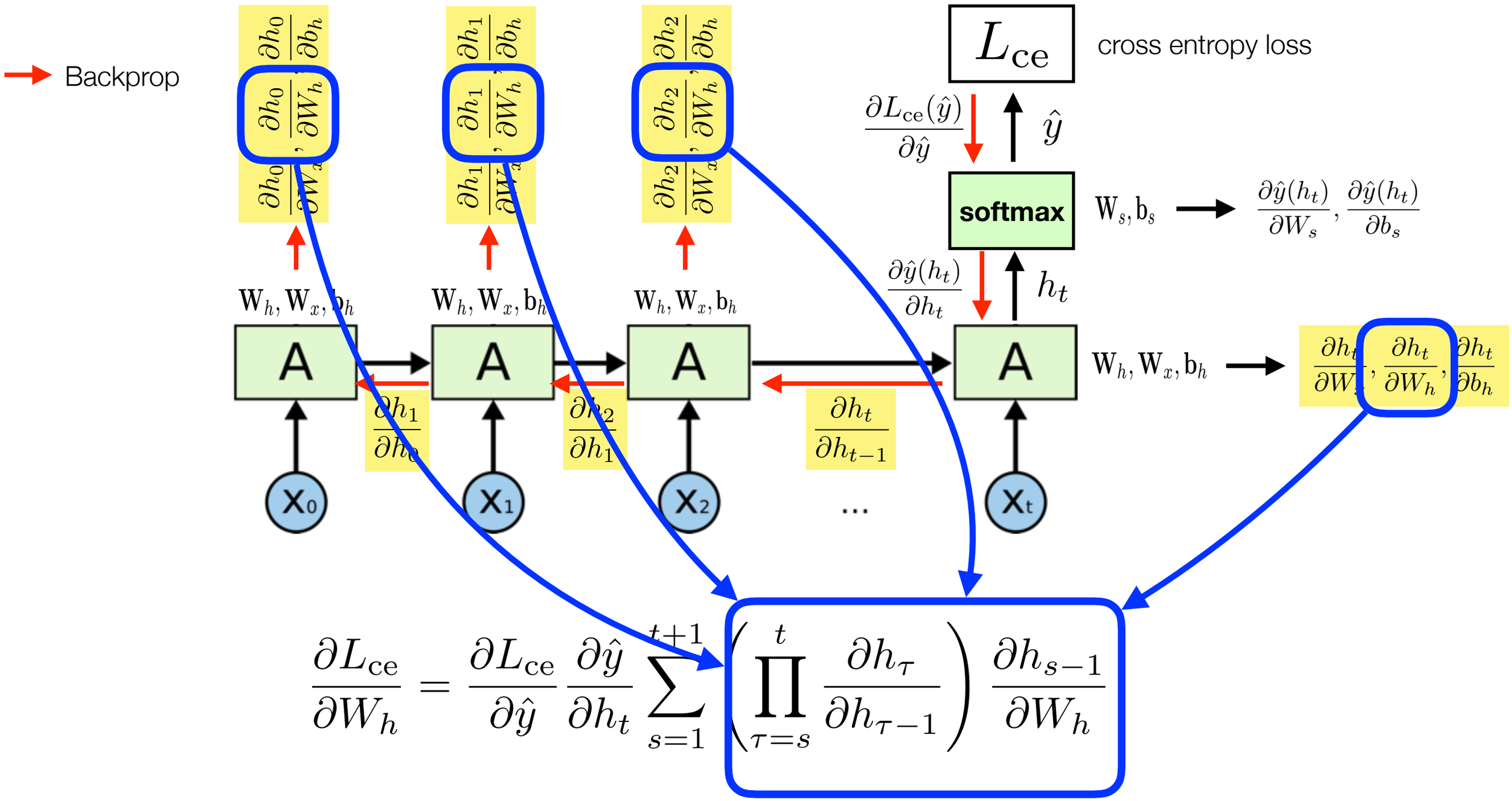
  Example: "Yesterday, the two top candidates for for federal council - a woman from the Canton Valais and a woman from Canton St. Gallen - were both elected in the first round".



Backprop with a Single-Layer Simple RNN



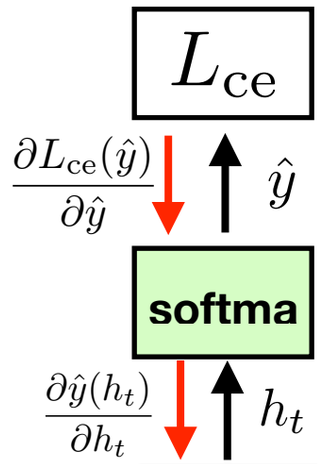
Gradient for Weights Matrix of a Layer

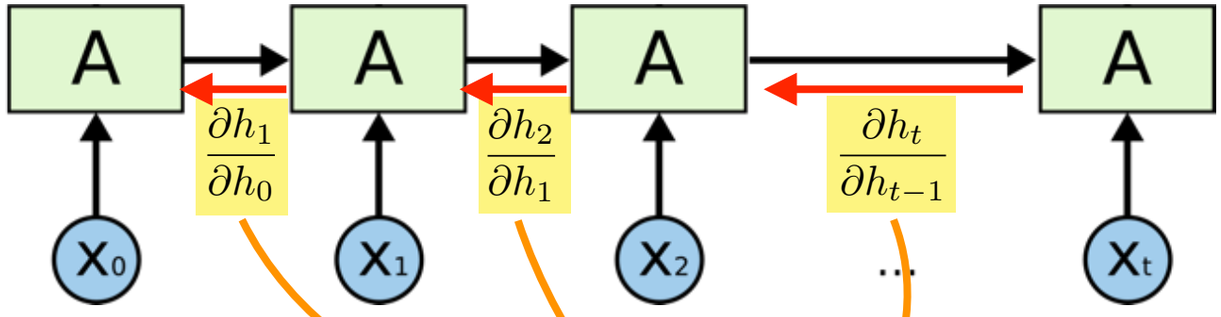


$$\frac{\partial L_{\text{ce}}}{\partial W_h} = \frac{\partial L_{\text{ce}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_t} \sum_{s=1}^{t+1} \left(\prod_{\tau=s}^{t} \frac{\partial h_\tau}{\partial h_{\tau-1}} \right) \frac{\partial h_{s-1}}{\partial W_h}$$

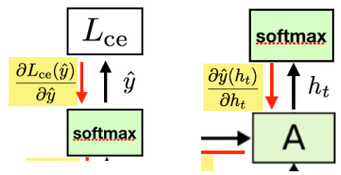
Backpropagation through Time

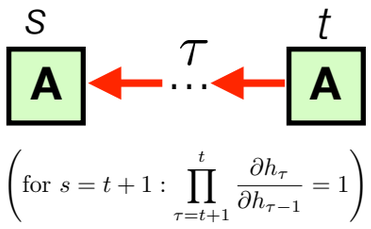
Backprop





$$\frac{\partial L_{\text{ce}}}{\partial W_h} = \frac{\partial L_{\text{ce}}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_t} \sum_{s=1}^{t+1} \left(\prod_{\tau=s}^{t} \frac{\partial h_\tau}{\partial h_{\tau-1}} \right) \frac{\partial h_{s-1}}{\partial W_h}$$





Product with many factors, each of the form

$$\mathbf{W}_h^T \cdot g'(\mathbf{z}_\tau)$$

can become arbitrarily small or large.

Backprop Through Time (BPTT)

- Product terms that handles backprop through time.

$$\prod_{\tau=s}^{t} \frac{\partial h_\tau}{\partial h_{\tau-1}}, \quad \frac{\partial h_\tau}{\partial h_{\tau-1}} = W_h^T \cdot g'(z_\tau)$$

bounded and small e.g. for sigmoid *g*

- If *t*-*s* is large (i.e. for long sequences) the product term can become
  - very small: gradient contributions from “far away” steps become zero, and the state at those steps doesn’t contribute to what you are learning —> vanishing gradient
  - very large: training does not converge —> exploding gradients.
- Long-range dependencies are hard to learn.

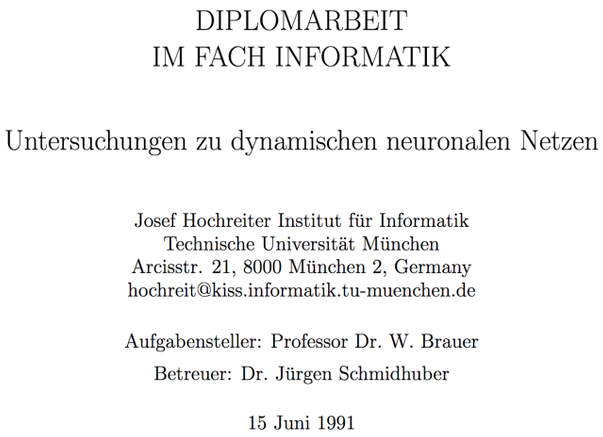
scalar case (for illustration)

$$\left.\begin{array}{l} 0 < w < 1 \\ t - s \gg 1 \end{array}\right\} \Rightarrow w^{t-s} \ll 1$$

$$\left.\begin{array}{l} w > 1 \\ t - s \gg 1 \end{array}\right\} \Rightarrow w^{t-s} \gg 1$$

Understanding the Long-Term Dependencies Problem

- In theory, RNNs are capable of keeping a memory incl. long-term dependencies.
  See e.g. https://stats.stackexchange.com/questions/220907/meaning-and-proof-of-rnn-can-approximate-any-algorithm/221142
- However, in practice, RNNs have difficulties in learning them due to the vanishing and exploding gradients problem — investigated by
  - Hochreiter, Schmidhuber (1991)
  - Bengio, Simard, Frasconi (1994)



DIPLOMARBEIT
IM FACH INFORMATIK

Untersuchungen zu dynamischen neuronalen Netzen

Josef Hochreiter Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
hochreit@kiss.informatik.tu-muenchen.de

Aufgabensteller: Professor Dr. W. Brauer
Betreuer: Dr. Jürgen Schmidhuber

15 Juni 1991

- Different attempts to solve these issues:

  (1) **Long-Short-Term Memory** (LSTM) **Cells** (Hochreiter, Schmidhuber, 1997), other **Gated Recurrent Units** (e.g. Cho et al., 2014)

  (2) **Suitable Initialisation of Weights**

Weights Initialisation Strategies

- **MLPs / CNNs** (as learned in week 6):
  - Properly initialise weights (Xavier/He) by using random numbers (uniform or normal) with mean=0 and suitably scaled stdev.
  - Use non-saturating activation functions (ReLU or Leaky ReLU) to alleviate the vanishing gradients problem
  - Batch Normalisation
  - Clipping gradients

- **RNNs** with ‘vanilla’ recurrent units:
  - Properly initialise weights (see next slide)
  - Use non-saturating activation functions (ReLU or Leaky ReLU) to alleviate the vanishing gradients problem
  - Clipping gradients (see last weeks example on activity recognition)

Weights Initialisation: Two Approaches

- **IRNN**:
  - Proposed by Le, Jaitly, Hinton in 2015 https://arxiv.org/pdf/1504.00941.pdf
  - Use ReLU to alleviate the vanishing gradient problem
  - Identity matrix for the recurrent weights so that the state variable remains unchanged in case there are no inputs (x).
  - Random parameters used as inputs weights (applied to x).
  - Support in Keras

$$W_h = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

```
SimpleRNN(hidden_units,
          kernel_initializer=initializers.RandomNormal(stddev=…),
          recurrent_initializer=initializers.Identity(gain=1.0),
          activation='relu',
          input_shape=…)
```

W_x

W_h

- **np-RNN**:
  - Proposed by Talathi, Vartak in 2016 https://arxiv.org/pdf/1511.03771v3.pdf
  - Use ReLU to alleviate the vanishing gradient problem
  - Recurrent weight matrix initialised to a normalized-positive definite matrix with the highest eigenvalue = 1 and all the others <1.
  - Random weights applied to the inputs.
  - No direct support in Keras (?), use custom initialiser

$$\begin{array}{rcl} W_h & = & \frac{1}{e}(A+I) \\ A & = & \frac{1}{N} R^T \cdot R \\ e & = & \max(\lambda(A+I)) \\ R & : & \text{Standard Gaussian} \\ \lambda(x) & : & \text{Set of eigenvalues of X} \end{array}$$

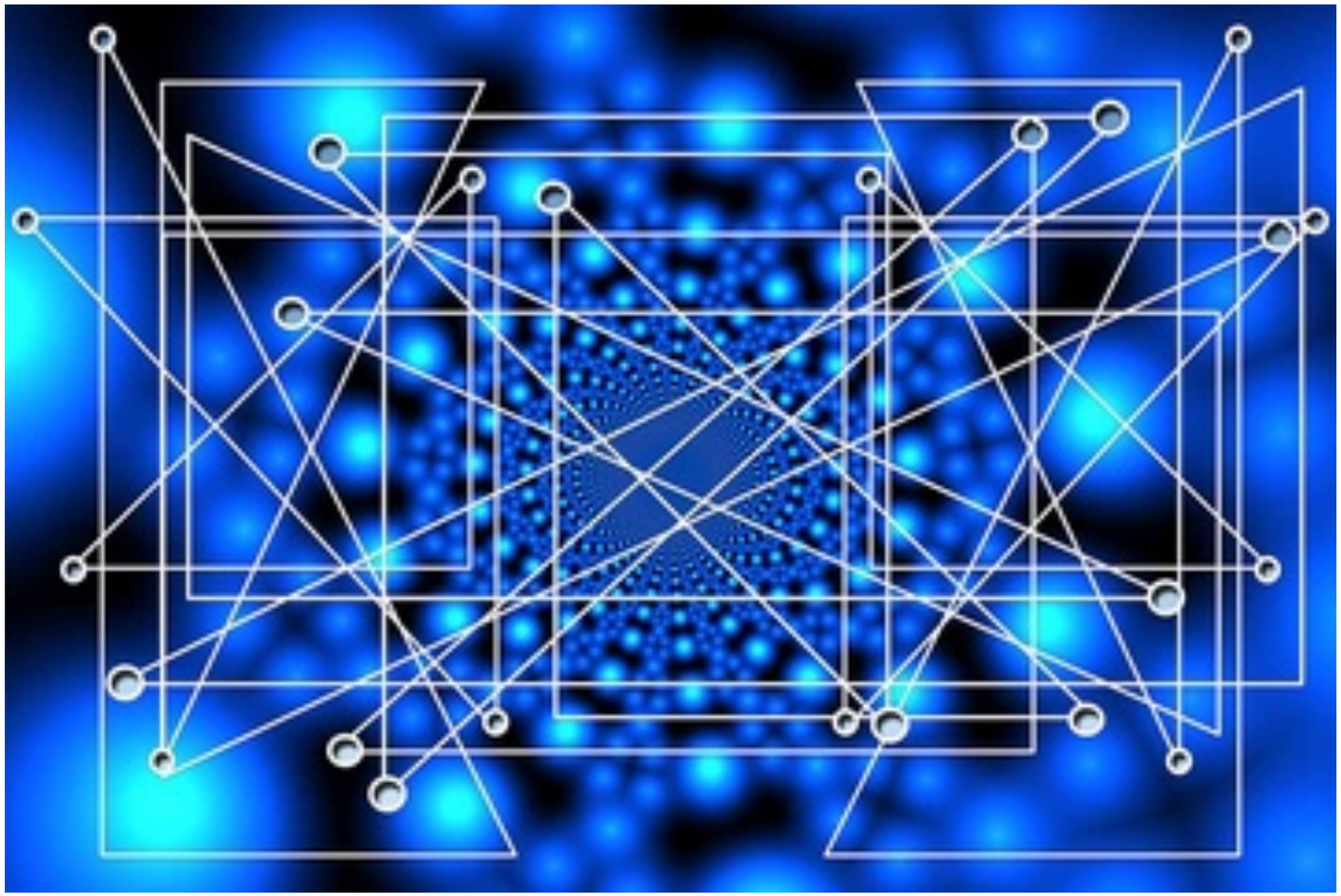
Comparison

RNN Type	Test Accuracy	Number of Parameters	Sensitivity to Parameters
IR-NN	67.0%	1 x	high
np-RNN	75.2%	1 x	low
LSTM	78.5%	4 x	low

See Talathi, Vartak ICLR 16, https://arxiv.org/pdf/1511.03771.pdf

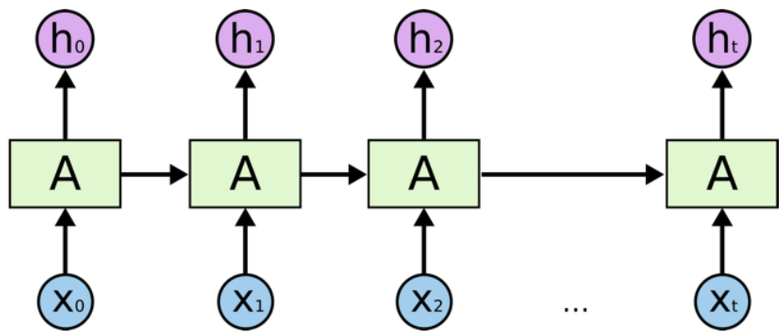
Tested on the UCF101 benchmark dataset — activity classification from video clips data.

LSTM
GRU



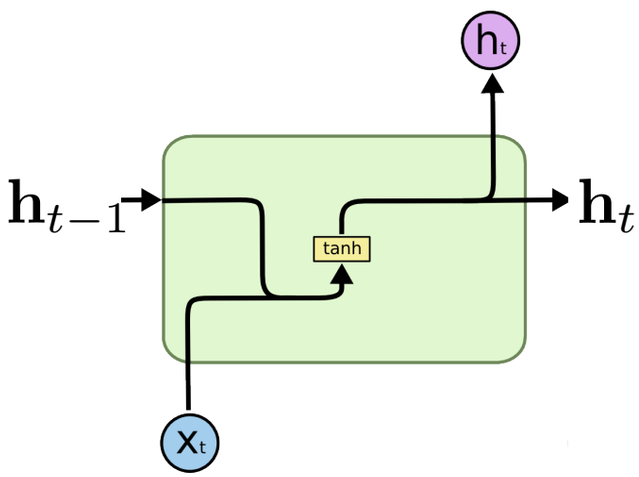
Long Short Term Memory Cells (LSTM)

- Introduced by Hochreiter und Schmidhuber (1997)
- LSTM networks have the same general structure of cyclicly updated cells.

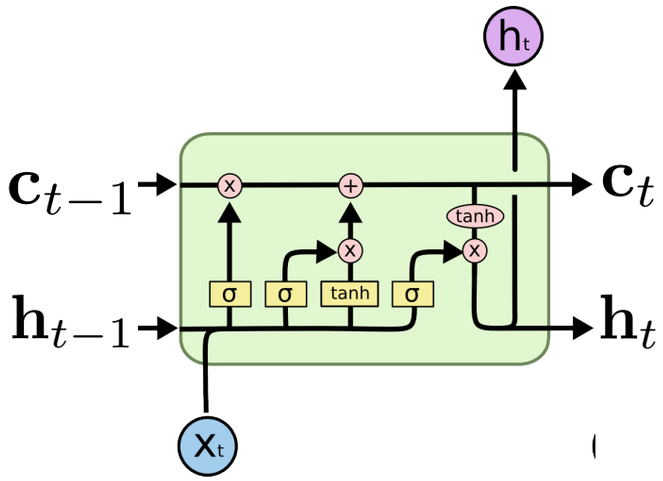


- The cell has a much richer structure: It is designed to keep a ***long-term memory*** that is kept as additional state variable $\mathbf{c}_t$ to be updated.

Simple RNN Unit



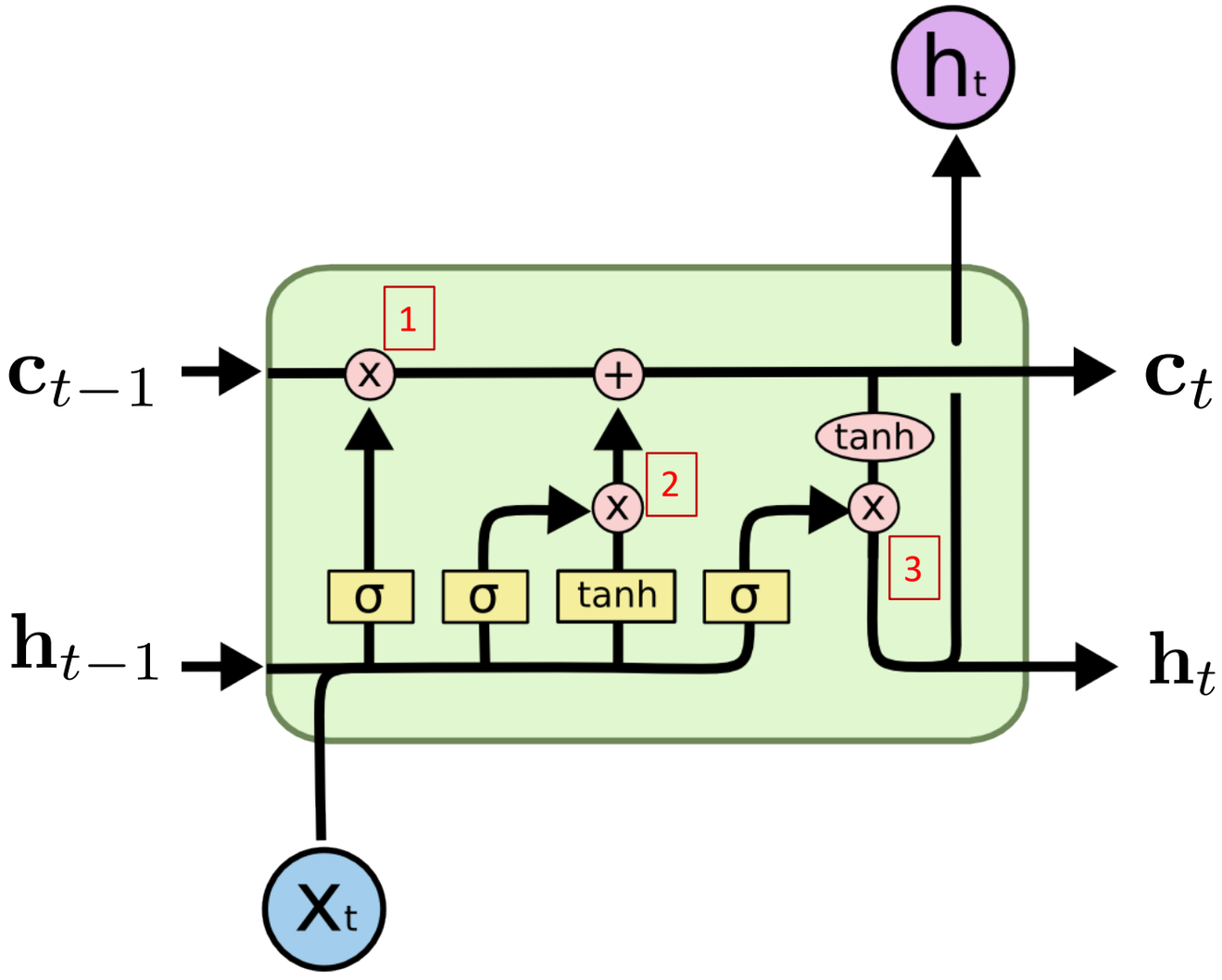
LSTM Unit



long-term memory

short-term memory
working memory

Internals of a LSTM Cell



Long-term memory is updated through **Gates** (marked with (x)): Gates control how the information flows between short-term state, input and long-term state

- Forget Gate 1
- Input Gate 2
- Output Gate 3

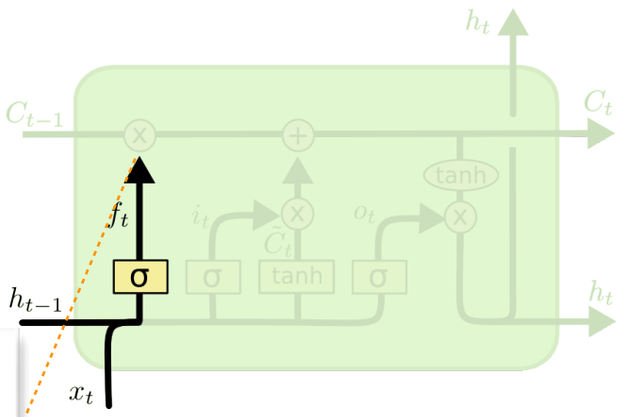
The gates are implemented by

- affine transformation of inputs
- activation function
- element-wise multiplication.

Internals of a LSTM Cell

Forget Mechanism

(—> long-term memory)



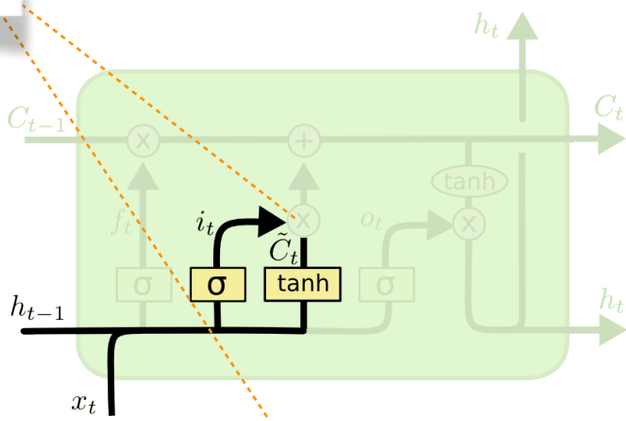
$$f_t = \sigma\left(\mathbf{W}_{f,x} \cdot x_t + \mathbf{W}_{f,h} \cdot h_{t-1} + \mathbf{b}_f\right)$$

If a scene (e.g. in a movie) ends, the model should forget some scene-specific information (location, the time of day) — but should remember other information (e.g. if a character dies).

Element-wise Multiplication

Save Mechanism

(—> long-term memory)



$$i_t = \sigma\left(\mathbf{W}_{i,x} \cdot x_t + \mathbf{W}_{i,h} \cdot h_{t-1} + \mathbf{b}_i\right)$$

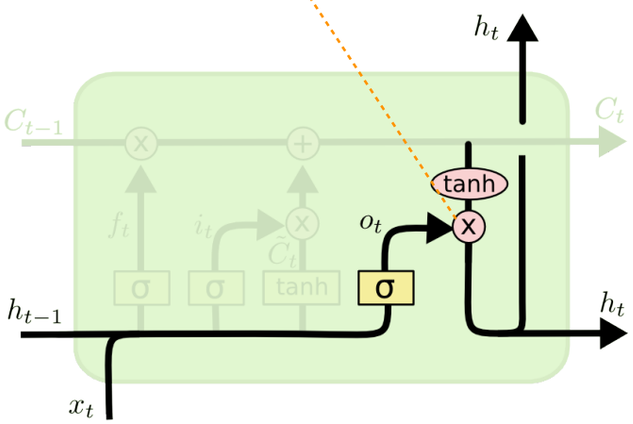
$$\tilde{c}_t = \tanh\left(\mathbf{W}_{c,x} \cdot x_t + \mathbf{W}_{c,h} \cdot h_{t-1} + \mathbf{b}_c\right)$$

When the model sees a new image, it needs to learn whether any information about the image is worth using and saving.

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

Output Mechanism

(—> short-term memory)

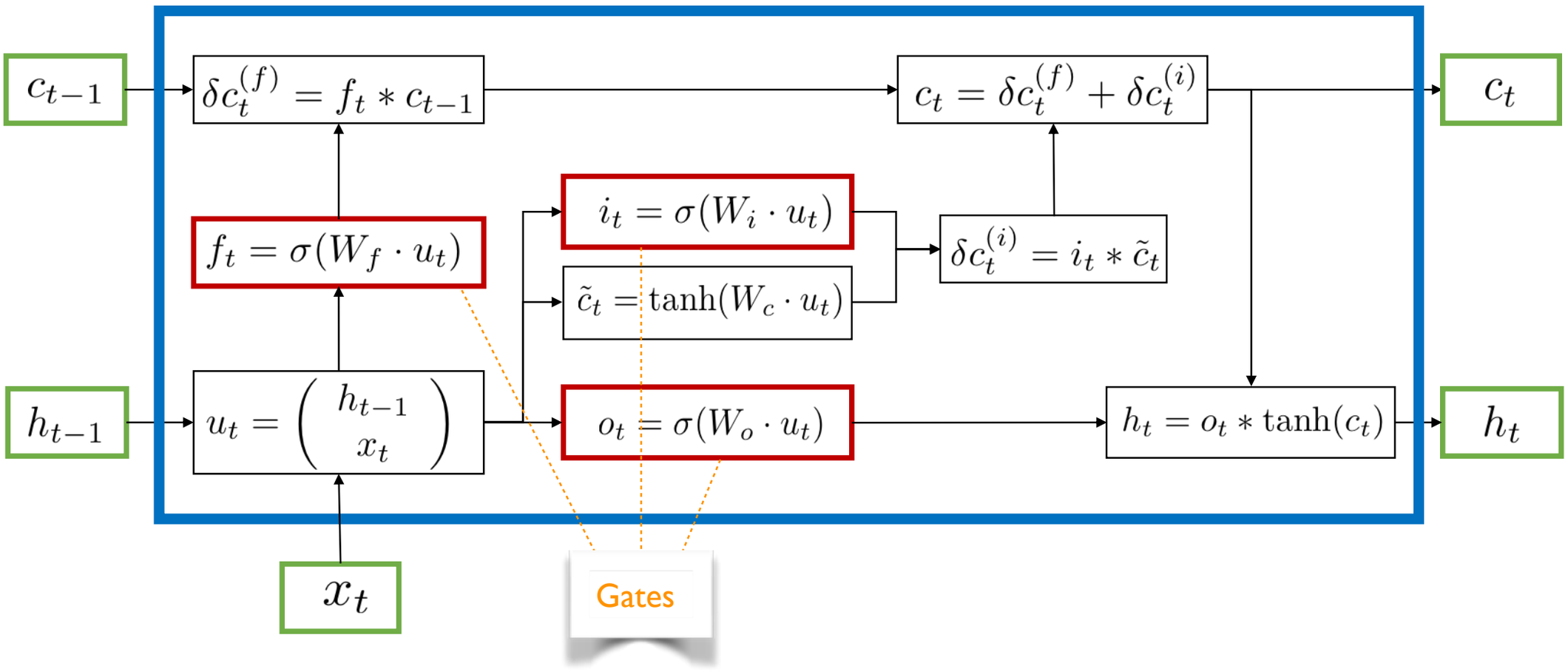


$$o_t = \sigma\left(\mathbf{W}_{o,x} \cdot x_t + \mathbf{W}_{o,h} \cdot h_{t-1} + \mathbf{b}_o\right)$$

The model needs to learn which part of the long-term memory is useful in the given situation.

$$h_t = o_t * \tanh(c_t)$$

Computational Graph for Single LSTM Step



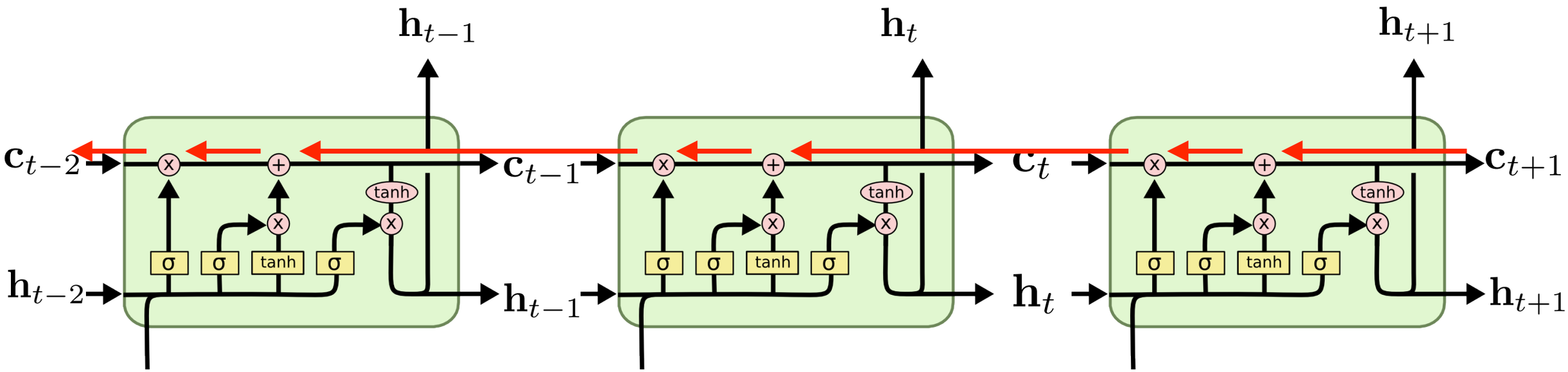
The matrices

$$W_f = (W_{f,h}\, W_{f,i})\,, W_i = (W_{i,h}\, W_{i,i})\,, W_c = (W_{c,h}\, W_{c,i})\,, W_o = (W_{o,h}\, W_{o,i})$$

have shape $n_h \times (n_h + n_x)$ with n_h the dimension of the hidden state variables h_t, c_t and n_x the dimension of the input x_t.

Backprop with LSTMs

"Super-Highway for Backprop"



$$\mathbf{c}_t = f_t * \mathbf{c}_{t-1} + i_t * \tilde{\mathbf{c}}_t \quad \Rightarrow \quad \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = f_t$$

time-dependent, in]0,1[

$$\Rightarrow \quad \frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_s} = \prod_{\tau=s+1}^{t} \frac{\partial \mathbf{c}_\tau}{\partial \mathbf{c}_{\tau-1}} = \prod_{\tau=s+1}^{t} f_\tau$$

A similar principle for stabilising backprop also in:

- Highway Networks (Sivastava, ICML 2015)
- ResNet

Implementation in Keras

Human Activity Recognition Example

Model with a single LSTM layer. Since we classify on the output of the last cell, we again use **`return_sequences=True`**

```
n_steps = len(X_train[0])  # 128 timesteps per series
n_input = len(X_train[0][0])  # 9 input parameters per timestep

n_hidden = 32 # Hidden layer num of features
n_classes = 6 # Total classes (should go up, or should go down)
reg_param = 0.005

model = Sequential()
model.add(LSTM(units=n_hidden, return_sequences=False, \
                    input_shape=(n_steps,n_input),\
                    kernel_regularizer=l2(reg_param)))
model.add(Dense(n_classes, kernel_regularizer=l2(reg_param), \
                activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',\
              metrics=['accuracy'])
model.summary()
```

n_hidden n_input

W_k	$[32 \times (32+9)]$	:	1312
b_k	$[32 \times 1]$	:	32
W_{fc}	$[6 \times 32]$	:	192
b_{fc}	$[6 \times 1]$	:	6

4x (k=f,i,c,o)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 32)                5376
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

Results including Gradient Clipping

Human Activity Recognition Example

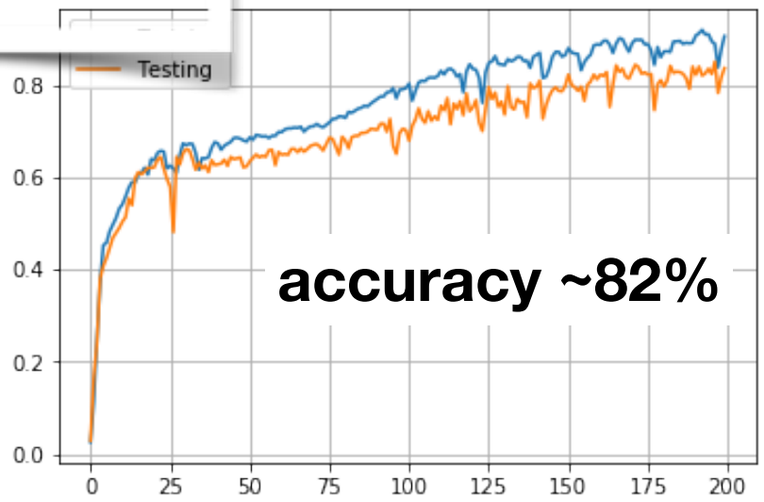
Simple RNN

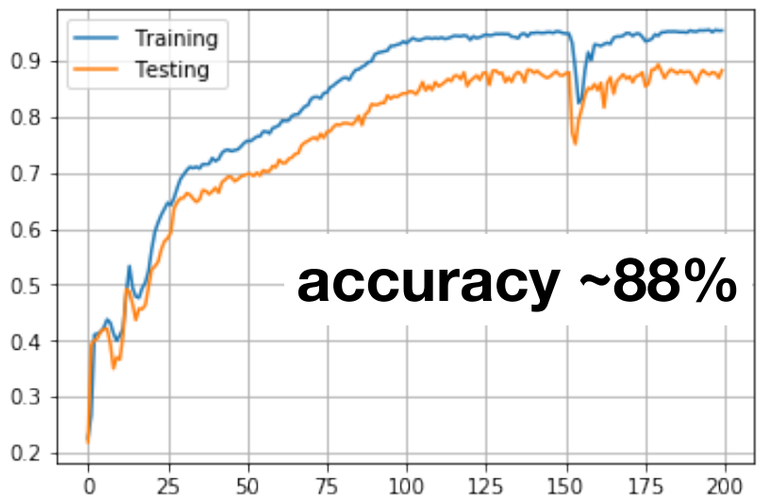
(clipping gradient by value)

LSTM

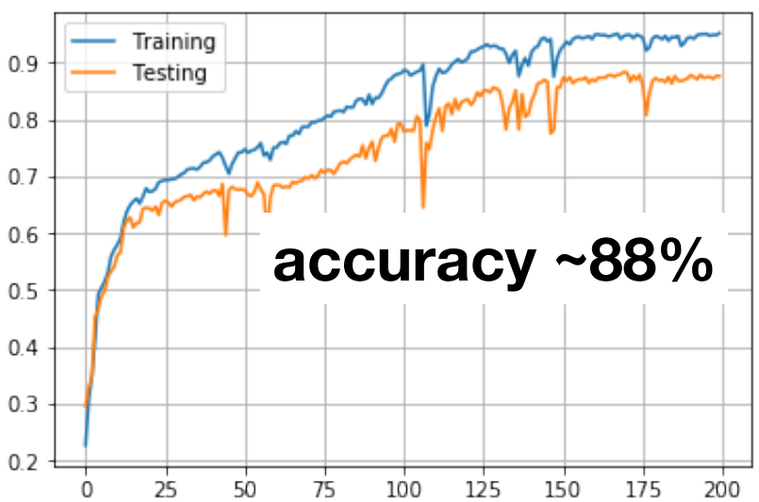
(w/o gradient clipping)

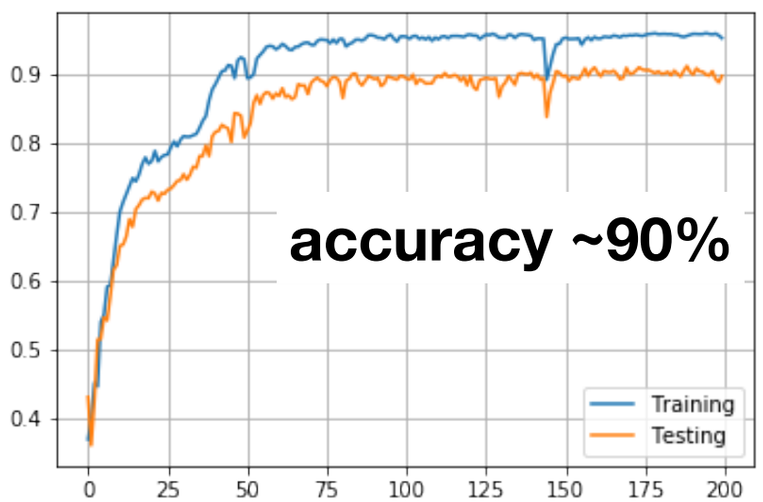
One Layer





Two Layers





- 32 hidden units in each layer, moderate L2-Regularisation
- Training with batch size 1500 over 200 epochs

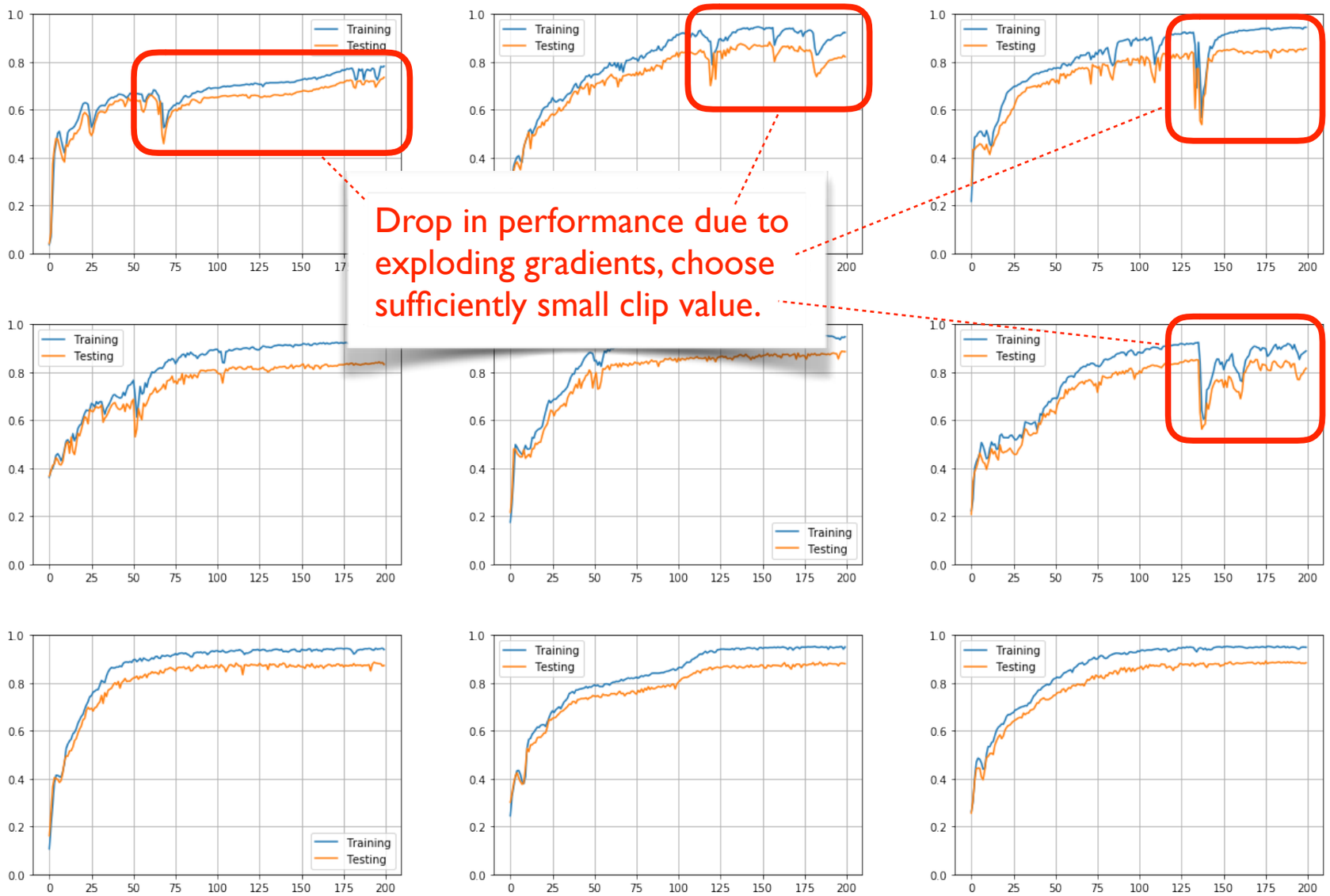
Comparing Different Runs of Same Model

Single Layer LSTM, 32 units

w/o gradient clipping

with gradient clipping clipvalue=0.5

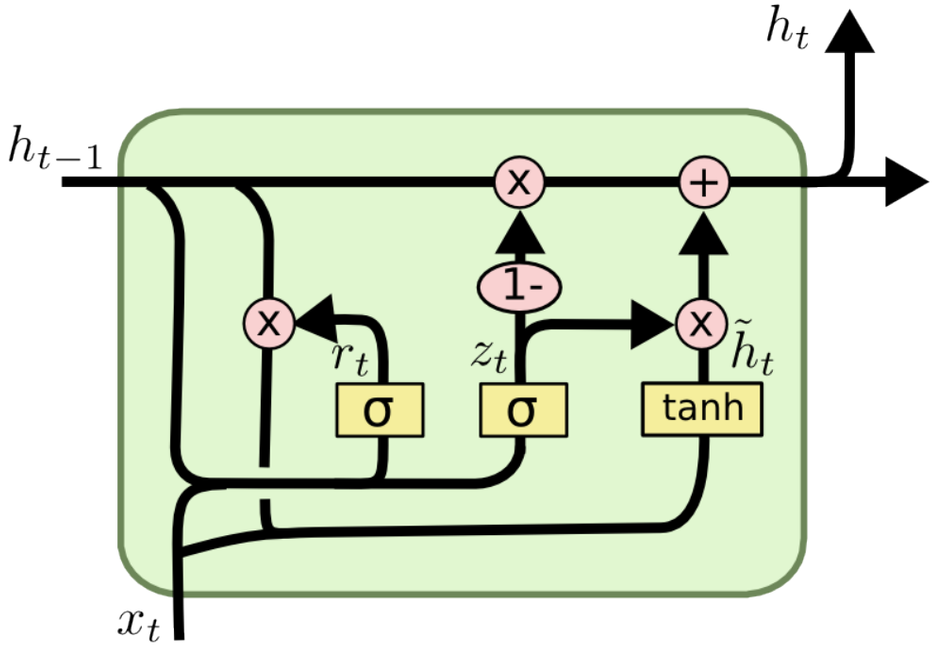
with gradient clipping clipvalue=0.1



Gated Recurrent Unit (GRU)

Several variations of LSTM. The most popular probably **GRU**:

- No separate long-term memory.
- Forget gate and input gate merged.
- Relevance gate (or reset gate)



Less parameters, since only **3** weight matrices of dimension $n_h \times (n_h + n_x)$: W_r, W_c, W_u

Relevance Gate

$$r_t = \sigma(W_{rh} \cdot h_{t-1} + W_{rx} \cdot x_t + b_r)$$

Candidate State

$$\tilde{h}_t = \tanh(W_{ch} \cdot (r_t * h_{t-1}) + W_{cx} \cdot x_t + b_c)$$

Update Gate

$$u_t = \sigma(W_{uh} \cdot h_{t-1} + W_{ux} \cdot x_t + b_u)$$

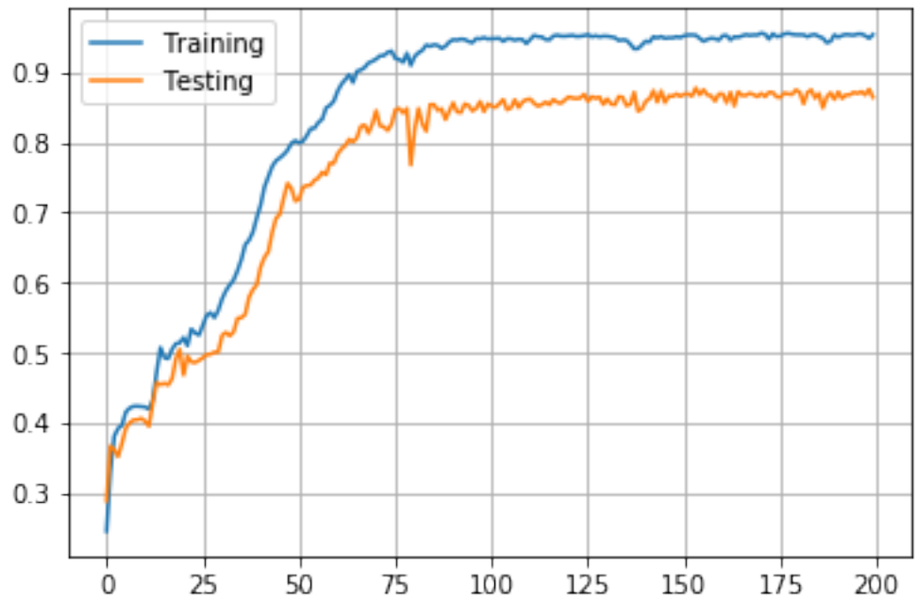
Update

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

Similar Performance with GRU as with LSTM

Human Activity Recognition Example

Single layer GRU with 32 units — otherwise same settings as above



Evaluated systematically by different authors on suitable example/benchmark tasks.

Evaluation on music and speech modeling tasks

(Chung 2014, https://arxiv.org/pdf/1412.3555.pdf)

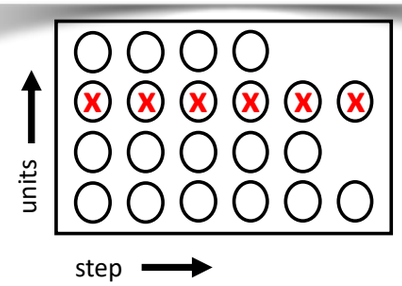
- Both, GRU and LSTM perform better than Simple RNN with tanh
- GRU performs comparably to LSTM
- No clear consensus between GRU and LSTM

A Note on Regularisation Schemes

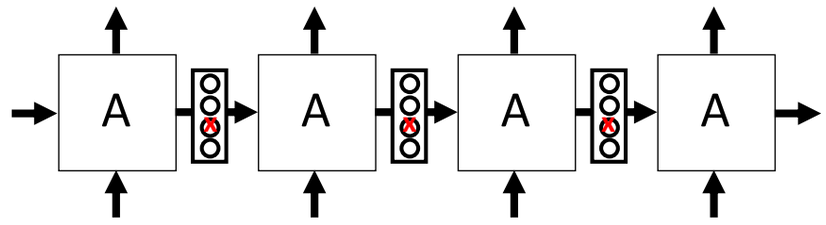
- Batch Normalisation for LSTMs
  - RNNs deepest along temporal dimension
  - Repeated scaling could cause exploding gradients
  - Batch-normalise hidden state variable *h* and input *x* independently (at each time step); use separate but fixed scaling parameters
  - No separate batch-normalisation of the long-term memory state *c*.
  - See Cooijmans, et al. (2016) https://arxiv.org/pdf/1603.09025.pdf

- Dropout in LSTMs
  - See Gal, Ghahramani (2016) https://arxiv.org/pdf/1512.05287.pdf
  - *Variational dropout* (as implemented in `keras`)
    - Same mask applied in all time steps.
    - Mask applied to **input/output units** (keyword `dropout`, or `Dropout` layer)
    - Mask applied to **hidden state units** (keyword `recurrent_dropout`)

Masking **input units** (uniformly for all steps) - in the example, the 3rd unit is masked (as indicated with **x**)



Masking **hidden state units** (uniformly for all steps) - in the example, the 2nd unit is masked (as indicated with **x**)



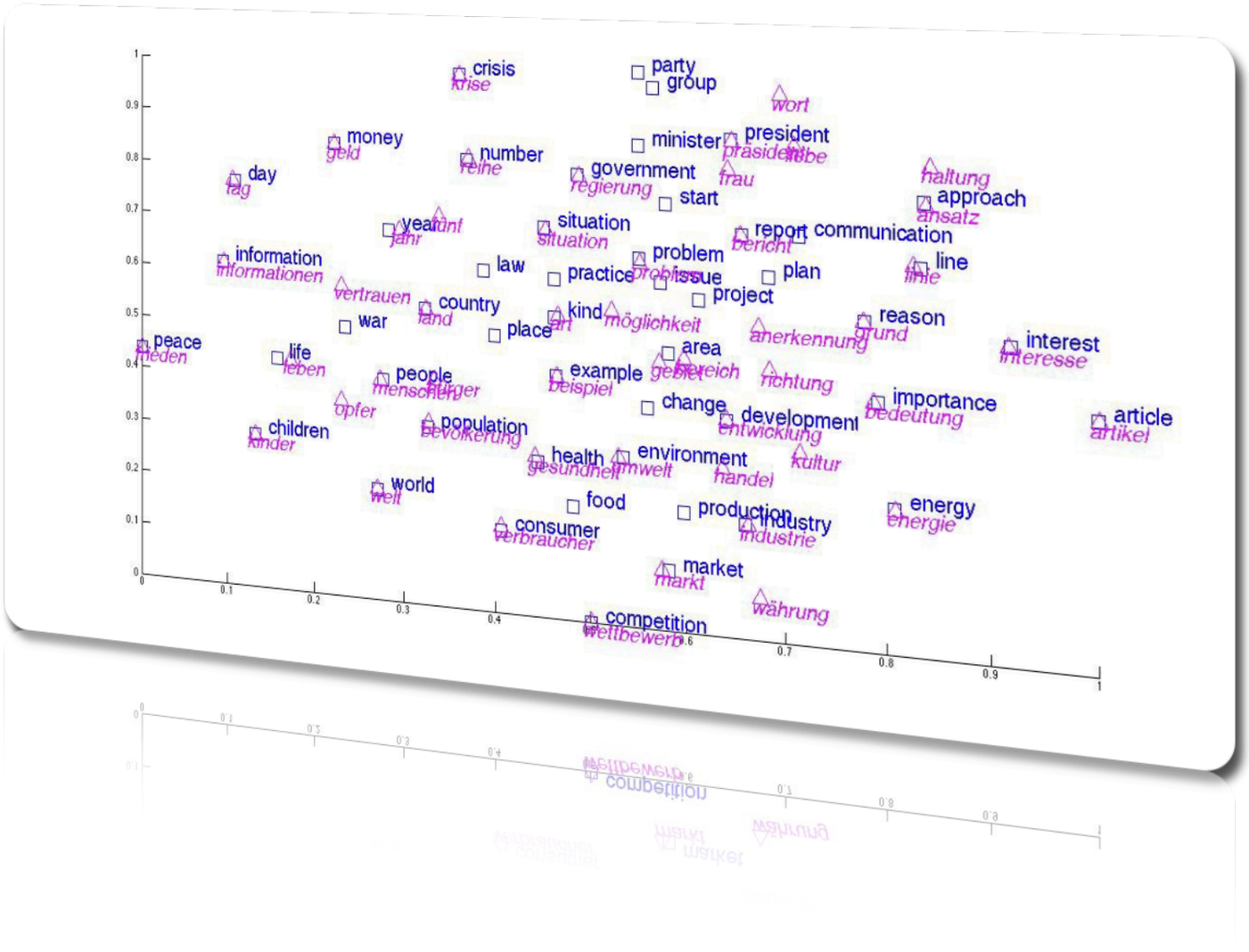
Word Embedding

Why embedding

Principles

Training strategies

Use of word embedding



Why word embedding?

- Inputs of networks need to be numerical continuous values
- 1-hot “atomic” representation of words
  - Too sparse! Let’s imagine a 50’000 word vocabulary application with 1-hot representation.

apple	[0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0]
orange	[0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 ... 0 0 0 0 0 0]
car	[0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0]

  - Does not carry information.

Figures taken from Xin Ring - School of Information - University of Michigan

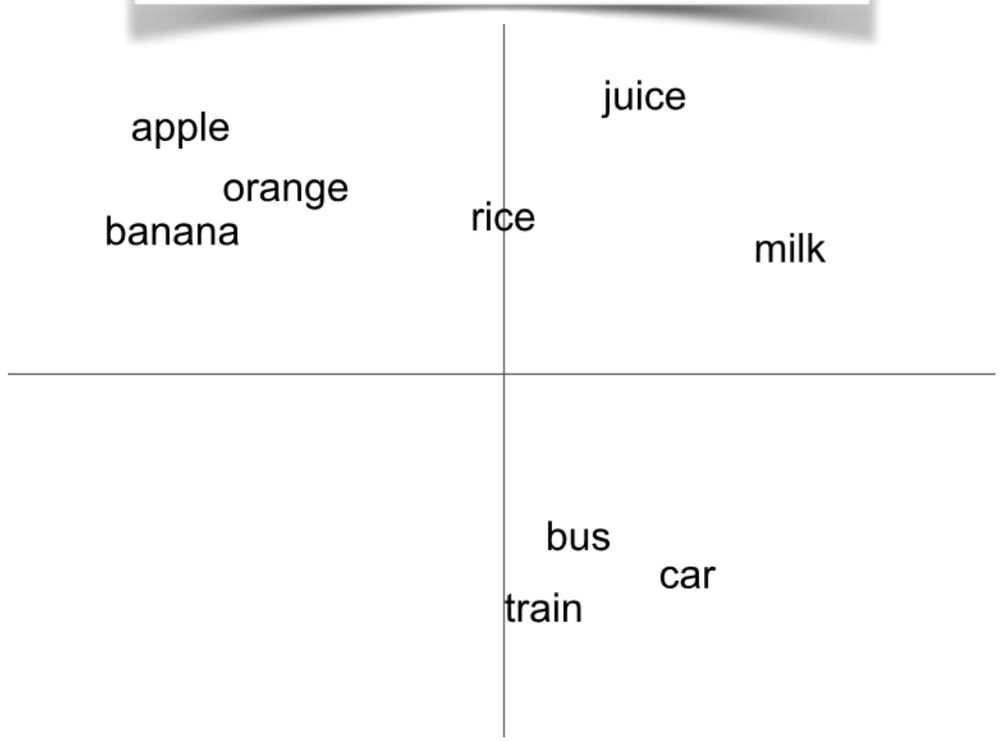
Word embedding

Word embedding is a projection of a word into vectors of real numbers, i.e. a mapping of a space where each word has its own dimension to a space that is of lower dimensionality.

Example: projection into a 3 dimensional space

anarchism	0.5	0.1	−0.1
originated	−0.5	0.3	0.9
as	0.3	−0.5	−0.3
a	0.7	0.2	−0.3
term	0.8	0.1	−0.1
of	0.4	−0.6	−0.1
abuse	0.7	0.1	−0.4

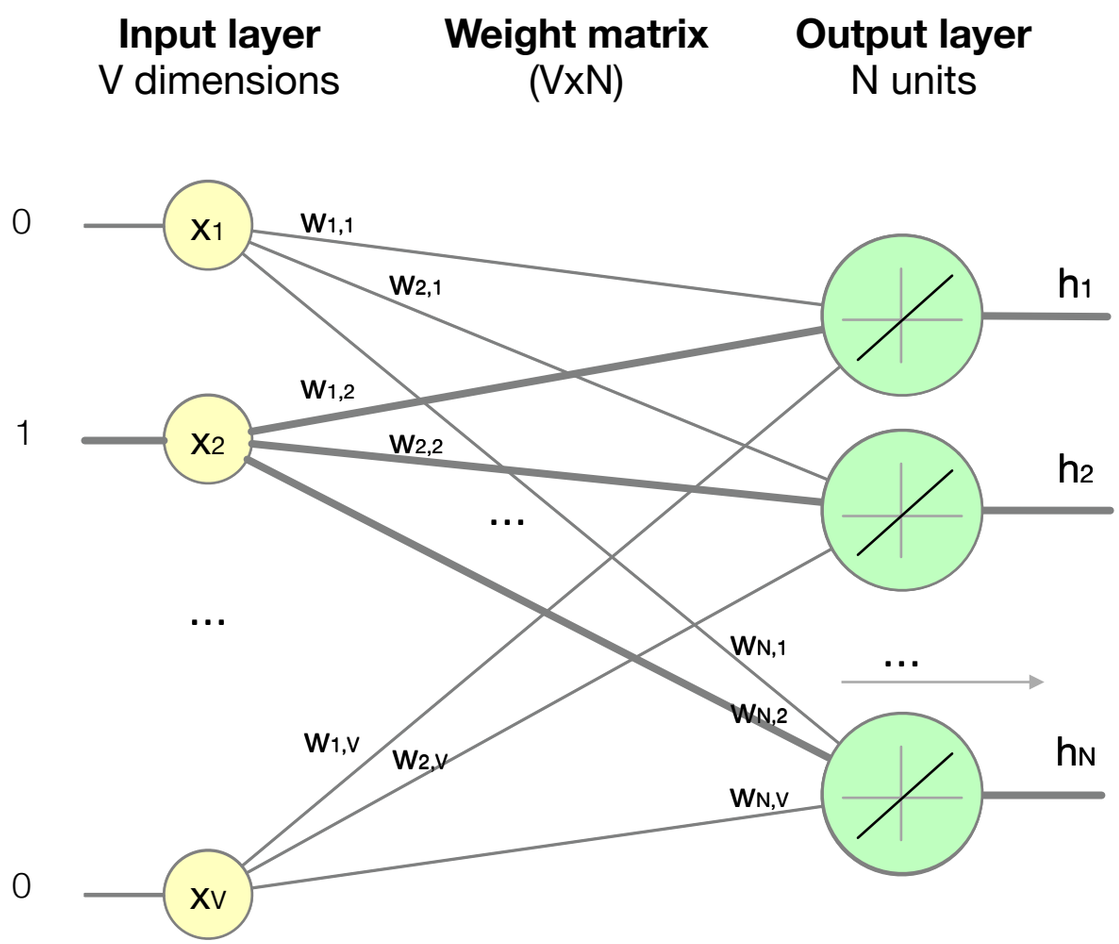
Example: projection into a 2 dimensional space



Embedding layer

- Input: 1-hot encoding of the word in a vocabulary of V dimension
  - Or just the index of the input that is hot
- Output: 1 vector of N dimension

Example: 'orange' is the 2nd word in the vocabulary



Output for 'orange' is a vector such as [0.3, 0.1, -0.2, …]

$$\mathbf{h} = \mathbf{x}^T\mathbf{W} := \mathbf{v}_{w_I}$$

As 'orange' is 1-hot, say in position 2 of vector x, h is then a simple lookup of the values in line 2 of the input weight matrix

Use of embedding layers

- Analogy with CNN for image recognition
  - The embedding layer is the first layer of a NLP network
  - It performs a kind of feature extraction for words
- 3 strategies:
  - Train the embedding layer on a given data set and for a given task
    - The embedding is going to be specific to the dataset and the task
  - Use a pre-trained “generic” embedding such as word2vec and freeze the layer - similar to *transfer learning* concept
  - Do both: init the embedding layer with pre-trained weights and train it

Embedding layer in Keras

- Read the API of the embedding layer in Keras
  - https://keras.io/layers/embeddings/
- Read the `IMDb_Sentiment.ipynb` notebook
  - IMDb dataset: 25'000 movie reviews from IMDb web site, labelled by sentiment (positive/negative)
    - https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification
    - `<START> this film was just brilliant casting location scenery story direction`

This is an example where we **train** the embedding layer to perform a task of sentiment analysis.

word2vec - generic representations

word2vec relates to models producing word embedding into space with **contextual similarity** or words, i.e. words that share common context are located in close proximity.

- Two original papers from Google - Mikolov et al. (2013)

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

https://arxiv.org/pdf/1301.3781.pdf

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

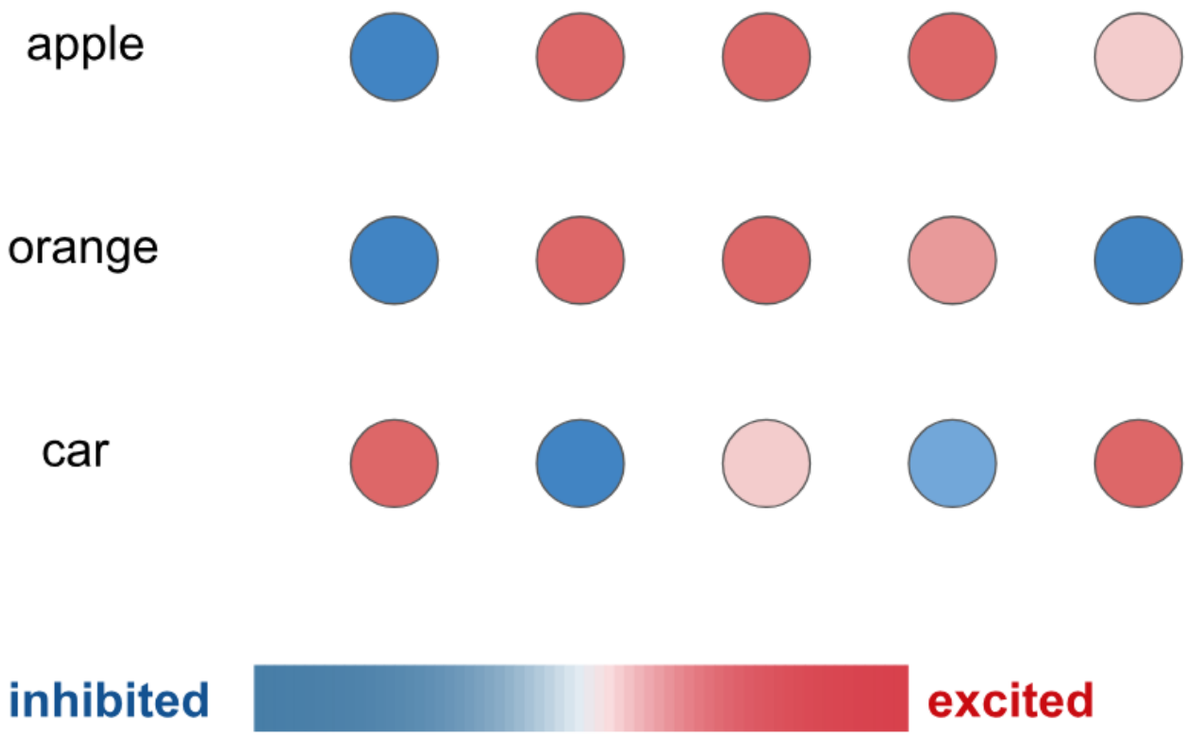
Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

https://arxiv.org/abs/1310.4546

word2vec - distributed representation

- We would like to represent words with “continuous” levels of activations where words in similar context share similar activations.



- How to create such representations?

Figures taken from Xin Ring - School of Information - University of Michigan

word2vec - contextual representation

- Hypothesis of work: a word is represented by its "context"

Another hypothesis: the order of the context words is disregarded. This is the "**bag-of-word**" assumption.

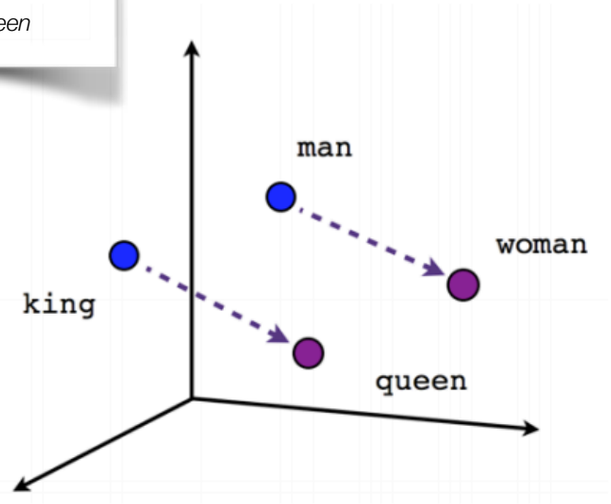
A window around the target word is created and words falling in the window are added to the "bag", disregarding the order.

Figures taken from Xin Ring - School of Information - University of Michigan

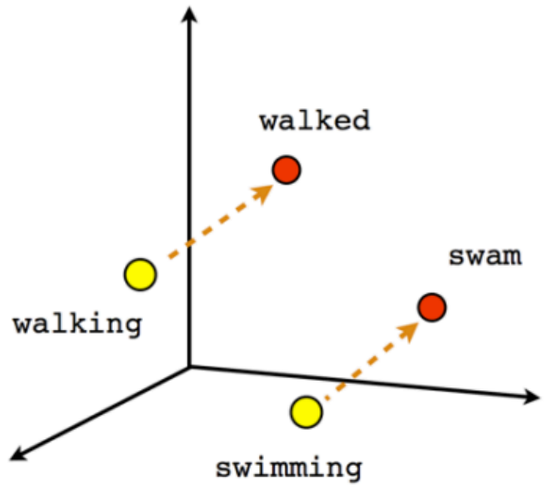
word2vec - geometric operations

- The word2vec mapping is not only about clustering related words, it is also able to preserve some **“path meaning”**

In vector terms:
$v_{woman} + (v_{king} - v_{man})$
is close to v_{queen}

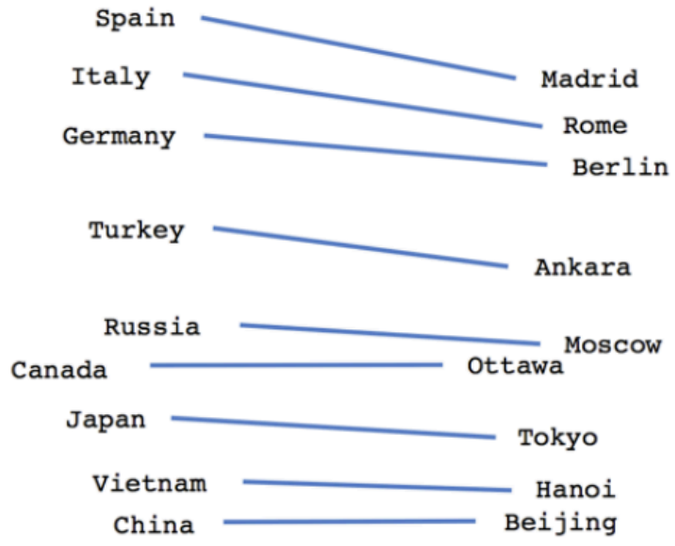


Male-Female



Verb tense

The path from a point to the other captures a semantic relationship: “capital of”

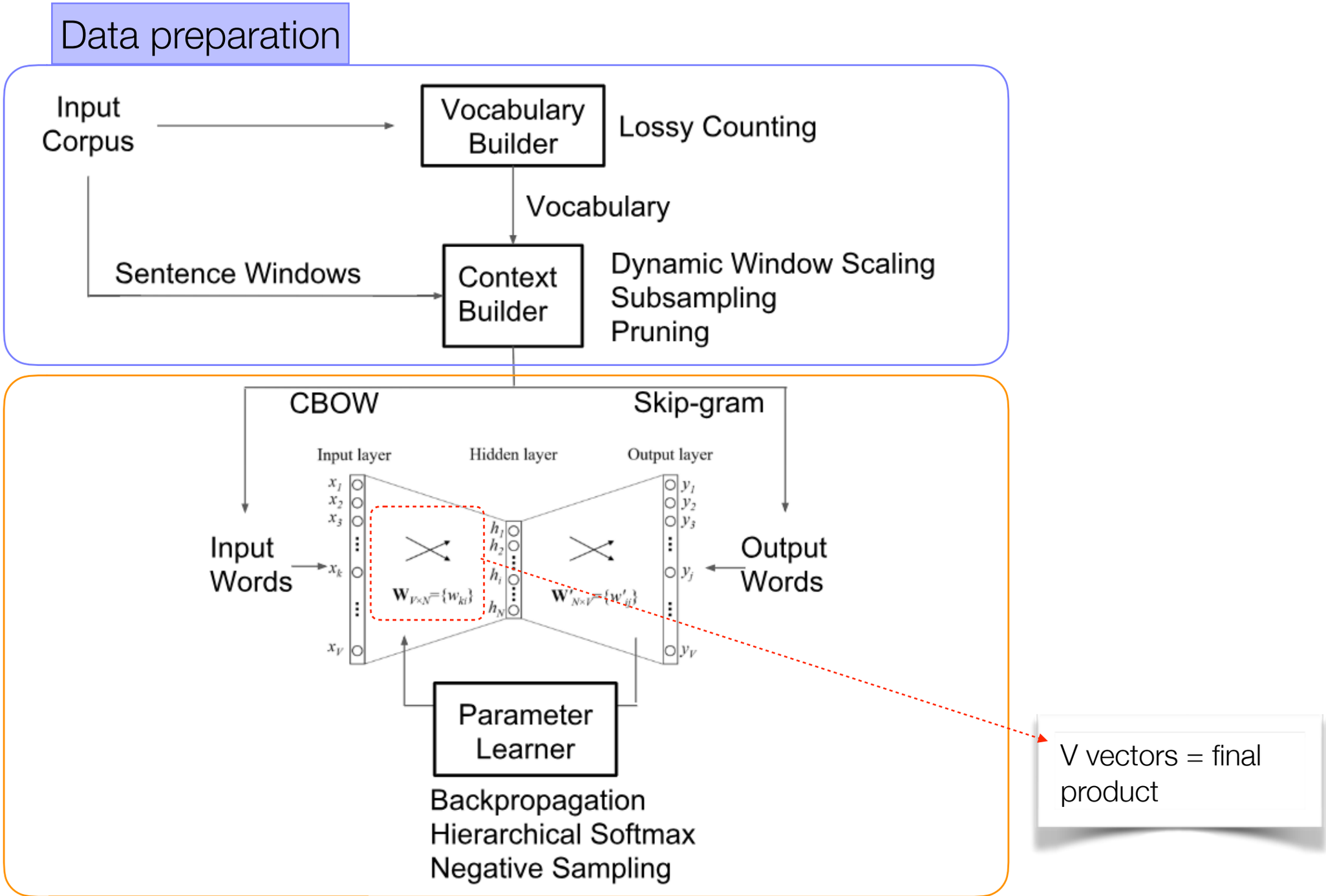


Country-Capital

word2vec - applications

- Many machine learning and deep learning tasks use *word2vec* as a preliminary transformation of the word, i.e. as a feature extraction
- Once in the *vec* space, we can apply more efficiently classification, prediction, clustering, etc.
  - Named entity recognition
  - Document classification
  - Sentiment analysis
  - Paraphrase detection
  - Word clustering
  - Machine translation
  - …

word2vec - training overview



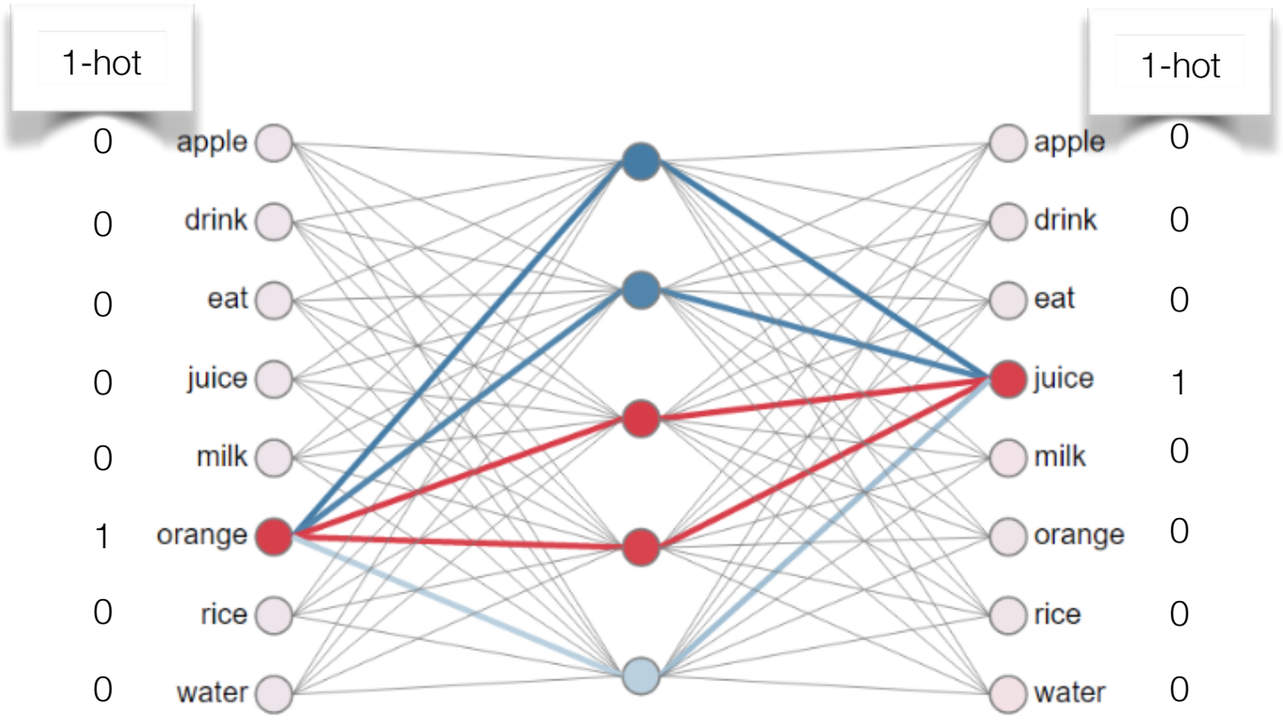
Figures taken from Xin Ring - School of Information - University of Michigan

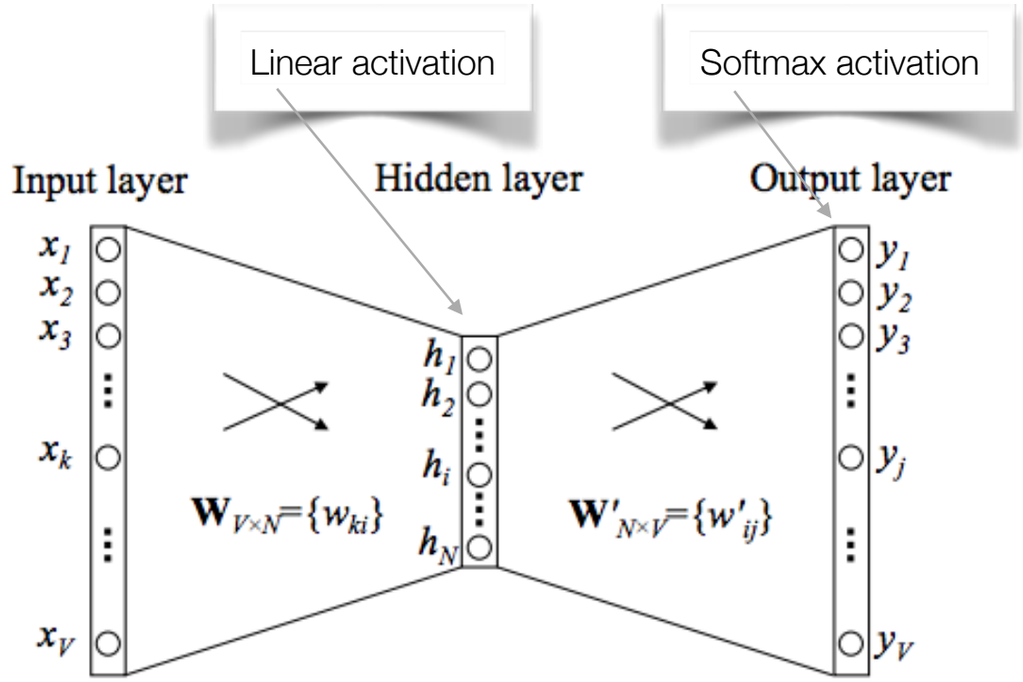
word2vec - training strategies

- One word context
  - Not really used, typically for educational purposes
- Continuous bag-of-word model - CBOW
  - Can be seen as extension of one word context to multi word context
- Skip-gram model
- Strategies are summarised in the next slides but if you want to dig further:
  - https://arxiv.org/pdf/1411.2738.pdf
  - https://www.youtube.com/watch?v=D-ekE-Wlcds

One word context

- Neat online demo: https://ronxin.github.io/wevi/





As ‘orange’ is 1-hot, say in position 6 of vector x, h is then a simple lookup of the values in line 6 of the input weight matrix

$$\mathbf{h} = \mathbf{x}^T\mathbf{W} := \mathbf{v}_{w_I}$$

Embedding of word ‘orange’

Before the softmax, the output logit of, say ‘juice’, is the dot product of h by the 4th column vector of the output weight matrix

$$u_j = \mathbf{v}'_{w_j}{}^T \cdot \mathbf{h}$$

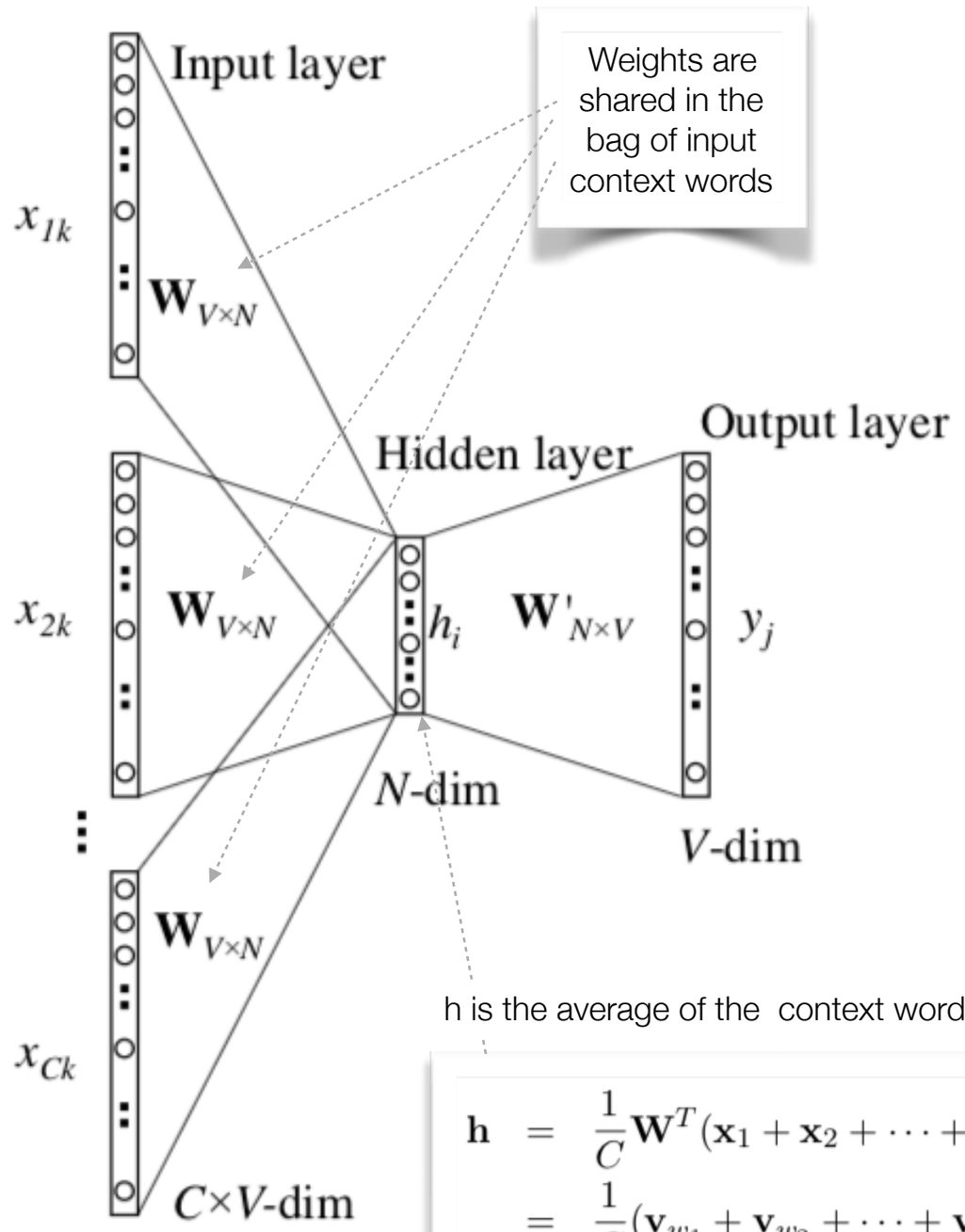
The softmax function can become problematic if we have large vocabulary. The denominator term can be cpu intensive.

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V}\exp(u_{j'})}$$

Figures taken from Xin Ring - School of Information - University of Michigan

CBOW - Continuous Bag of Word

- Can be seen as an extension of one word context to multi-word context at the input layer
- From a bag of context words, predict a word



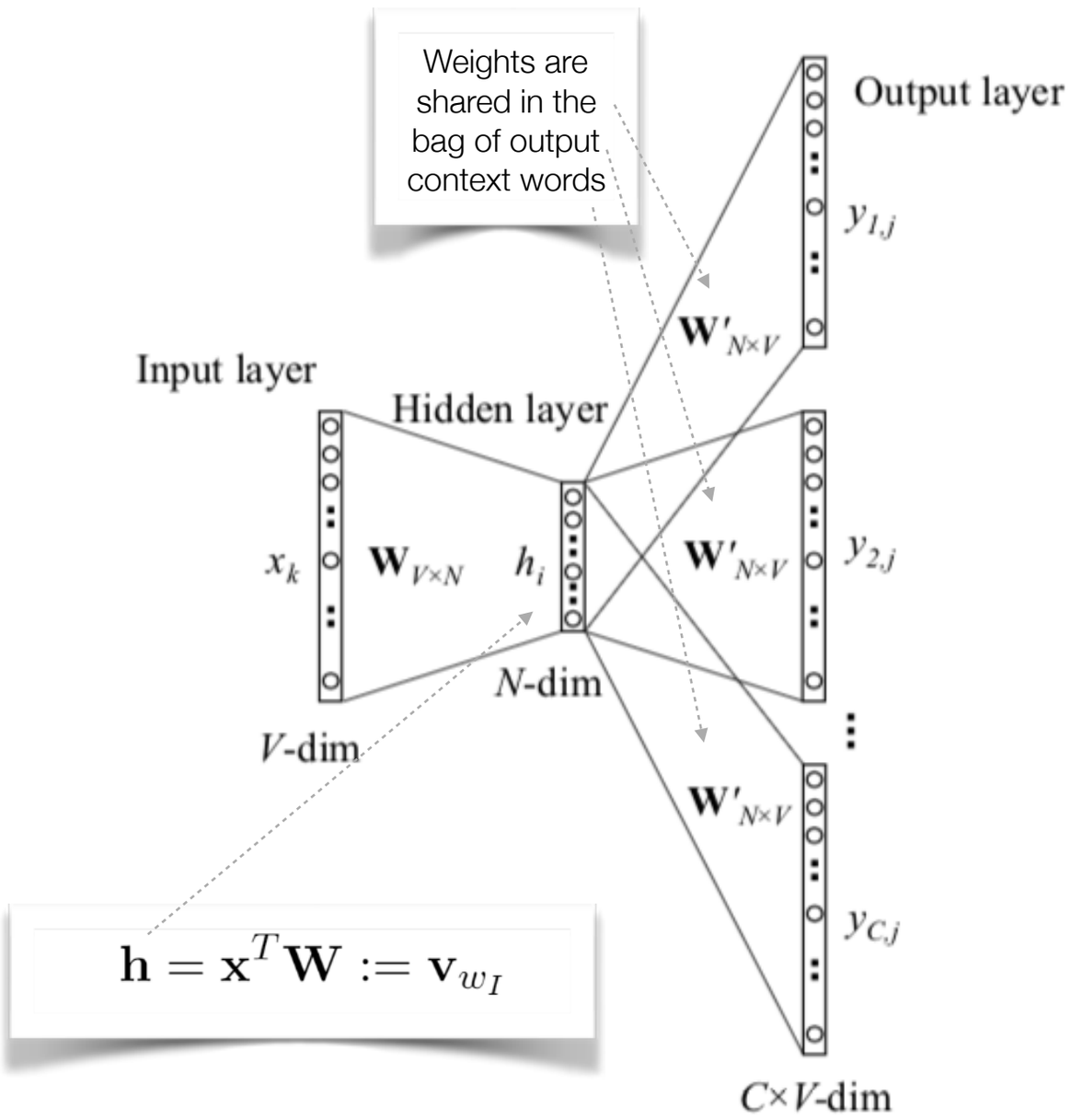
h is the average of the context word vectors

$$\begin{aligned}\mathbf{h} &= \frac{1}{C}\mathbf{W}^T(\mathbf{x}_1+\mathbf{x}_2+\cdots+\mathbf{x}_C)\\ &= \frac{1}{C}(\mathbf{v}_{w_1}+\mathbf{v}_{w_2}+\cdots+\mathbf{v}_{w_C})^T\end{aligned}$$

Figures taken from Xin Ring - School of Information - University of Michigan

Skip Gram model

- Can be seen as an extension of one word context to multi word context at the output layer
- From a bag of context words, predict a word

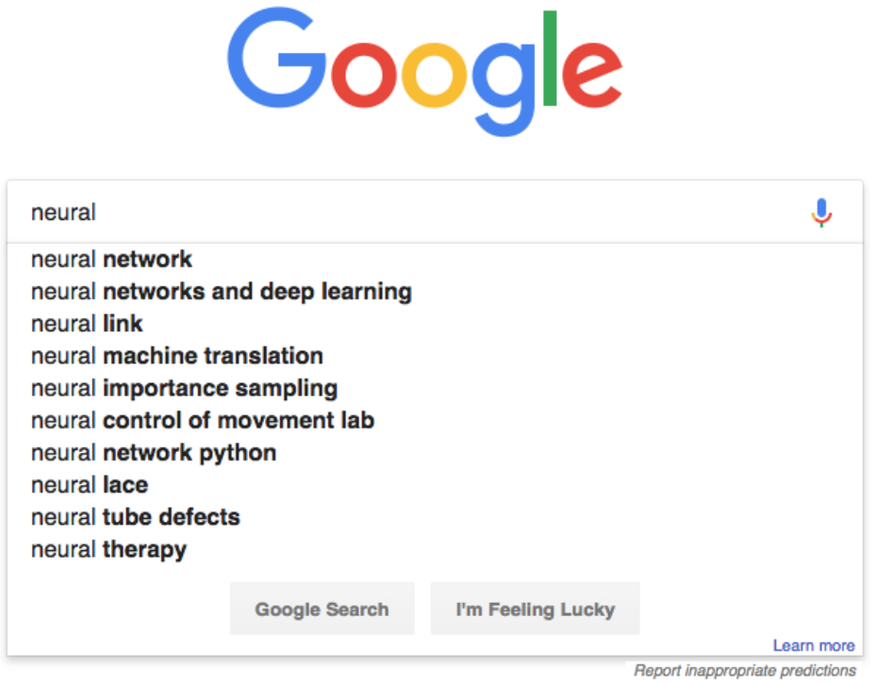


Figures taken from Xin Ring - School of Information - University of Michigan

word2vec - CBOW for word prediction

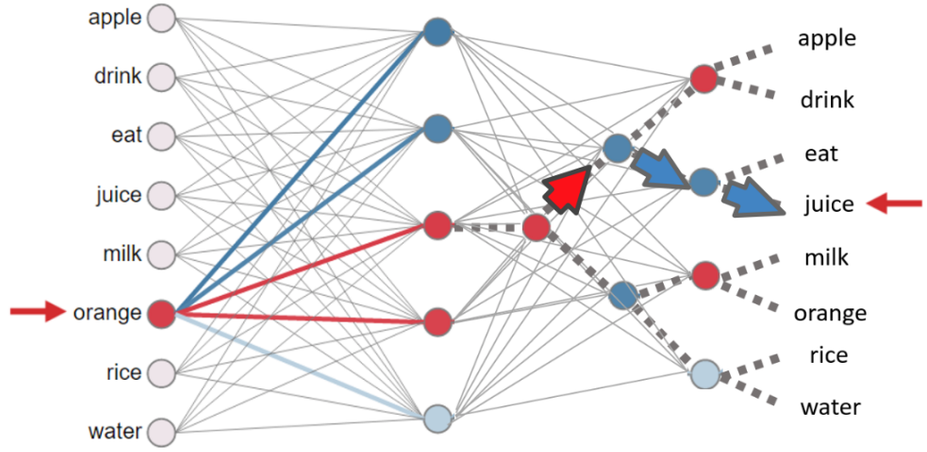
Activity

- Could we use CBOW for word prediction in the context of a search engine keyword completion?



word2vec - practical considerations on softmax

- The softmax and the derived update rules need to iterate through all the vocabulary
  - Impractical if V is large and if training corpora are large, which is the case most of the time
- 2 solutions
  - **Hierarchical Softmax** - Morin and Bengio, 2005
    - Build a binary tree on top of the output layer
    - https://www.youtube.com/watch?v=B95LTf2rVWM
  - **Negative sampling** - Mikolov et al, 2013
    - Update only a sample of the outputs, instead of all of them



word2vec - implementations

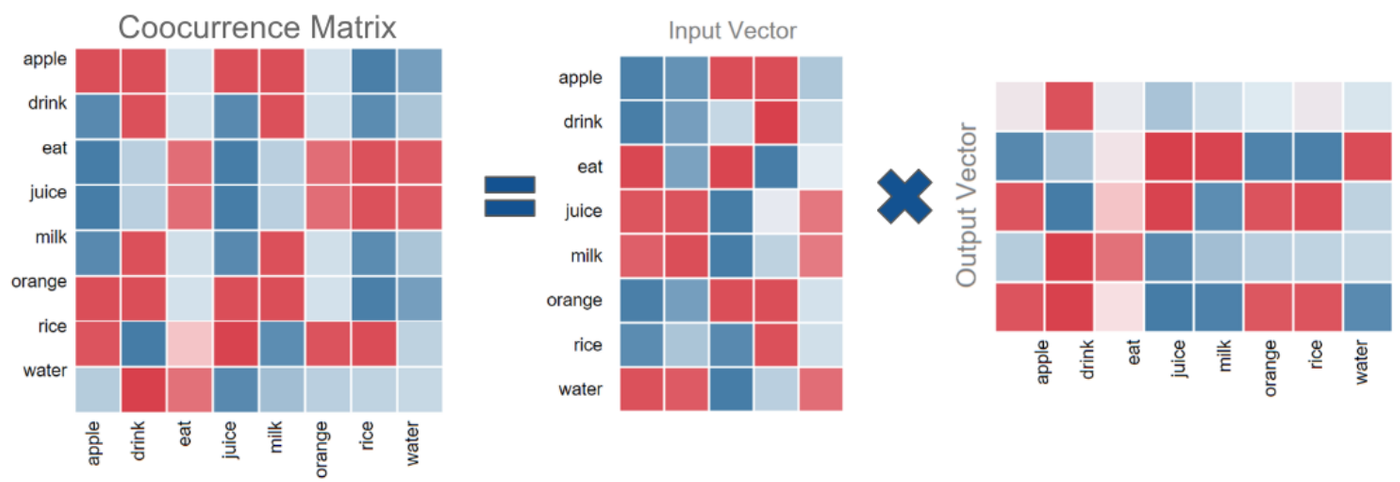
- A popular implementation is gensim
  - https://radimrehurek.com/gensim/models/word2vec.html
  - Pre-trained models and ready-to-download datasets: https://github.com/RaRe-Technologies/gensim-data

```
from gensim.models.word2vec import Word2Vec
import gensim.downloader as api

corpus = api.load('text8')  # download the corpus and return it opened as an iterable
model = Word2Vec(corpus)  # train a model from the corpus
model.most_similar("car")

"""
output:

[(u'driver', 0.8273754119873047),
 (u'motorcycle', 0.769528865814209),
 (u'cars', 0.7356342077255249),
 (u'truck', 0.7331641912460327),
 (u'taxi', 0.718338131904602),
 (u'vehicle', 0.7177008390426636),
 (u'racing', 0.6697118878364563),
 (u'automobile', 0.6657308340072632),
 (u'passenger', 0.6377975344657898),
 (u'glider', 0.6374964714050293)]

"""
```

Other popular embeddings

- GloVe - Global Vectors for Word Representation
  - https://nlp.stanford.edu/projects/glove/
  - Not based on ANN approaches but on statistical models
    - Dimensionality reduction of co-occurence word X context
- *word2vec* and approaches based on co-occurence matrices are actually related

O. Levy and Y. Goldberg, “Neural Word Embedding as Implicit Matrix Factorization”, NIPS 2014



Figures taken from Xin Ring - School of Information - University of Michigan

Wrap-up

- **Long term dependencies**
  - If “too long term”, we could face problems of vanishing / exploding gradients
  - Gradient clipping should be activated
  - Good weight initialisation strategies could help
- **LSTM - GRU**
  - Designed to capture long-term dependencies and to let the gradient flow more easily through time
- **Word Embedding**
  - Projection of a word into vectors of real numbers
  - word2vec allows to preserve contextual information in the embedding space