COMP SCI 7327 Concepts in Artificial Intelligence & Machine Learning
-Text Classification

By Dr Wei Zhang

Text Classification

- Text Categorization
- Sentiment Analysis
- Spam Detection
- Grammar Error Detection
- Malware Detection
- Relation Extraction
- ...

Text Classification

- Text Representation
  - Sentence Representation
  - Word Representation
  - Character Representation
- Pre-processing
- Simple DNN Examples

Text Representation

- Sentence Representation
  - N-gram
  - Word-level
  - Character-level

Text Representation (Cont.)

Example: *Discussing things you care about can be difficult*

- N-gram
  - 2-gram (bigram): Discussing things, things you, you care, care about, about can, …
  - 3-gram: Discussing things you, things you care,…
- Word level: Discussing, things, you, care, about, can, be, difficult
- Character-level: D, i, s …

Sentence Representation

- Bag-of-Words Model
  - TF/IDF

Sentence Representation

- Bag-of-Words Model

D1 - "I am feeling very happy today"
D2 - "I am not well today"
D3 - "I wish I could go to play"

Unique list of words:
I am feeling very happy today not well wish could go to play

	I	am	feeling	very	happy	today	not	well	wish	could	go	to	play
D1	1	1	1	1	1	1	0	0	0	0	0	0	0
D2	1	1	0	0	0	1	1	1	0	0	0	0	0
D3	2	0	0	0	0	0	0	0	1	1	1	1	1

Source: Anirban Majumder, 2017

Sentence Representation

- TF/IDF (term frequency-inverse document frequency)

Term Frequency (TF): is a scoring of the frequency of the word in the current document.

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

Inverse Document Frequency (IDF): is a scoring of how rare the word is across documents.

$$IDF(t) = log_e(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

TF/IDF score: TF*IDF

Sentence Representation

- TF/IDF Scikit-Learn code example

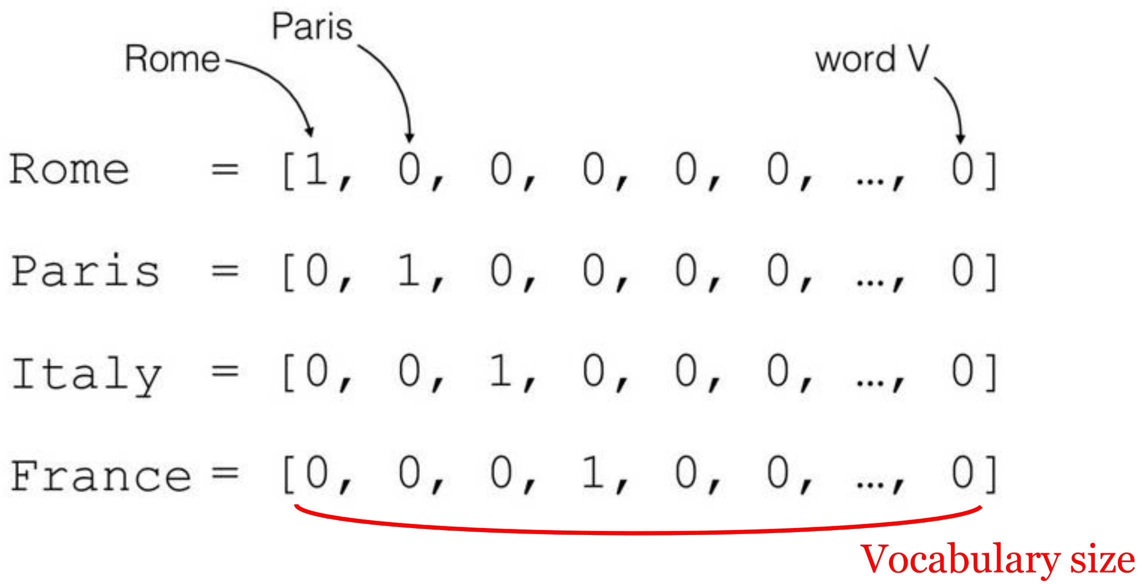
```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer()
# use TreeankWordTokenizer
 from nltk.tokenize import TreebankWordTokenizer
 tokenizer = TreebankWordTokenizer()
vect.set_params(tokenizer=tokenizer.tokenize)
# remove English stop words
vect.set_params(stop_words='english’)
# include 1-grams and 2-grams
 vect.set_params(ngram_range=(1, 2))
 # ignore terms that appear in more than 50% of the documents
vect.set_params(max_df=0.5)
# only keep terms that appear in at least 2 documents
vect.set_params(min_df=2)
```

Word Representation

- One-hot Encoding
- Word Embedding
  - Distributed representation: word2vec, Glove etc.
- SubWord
  - WordPiece
  - BPE

Word Representation (Cont.)

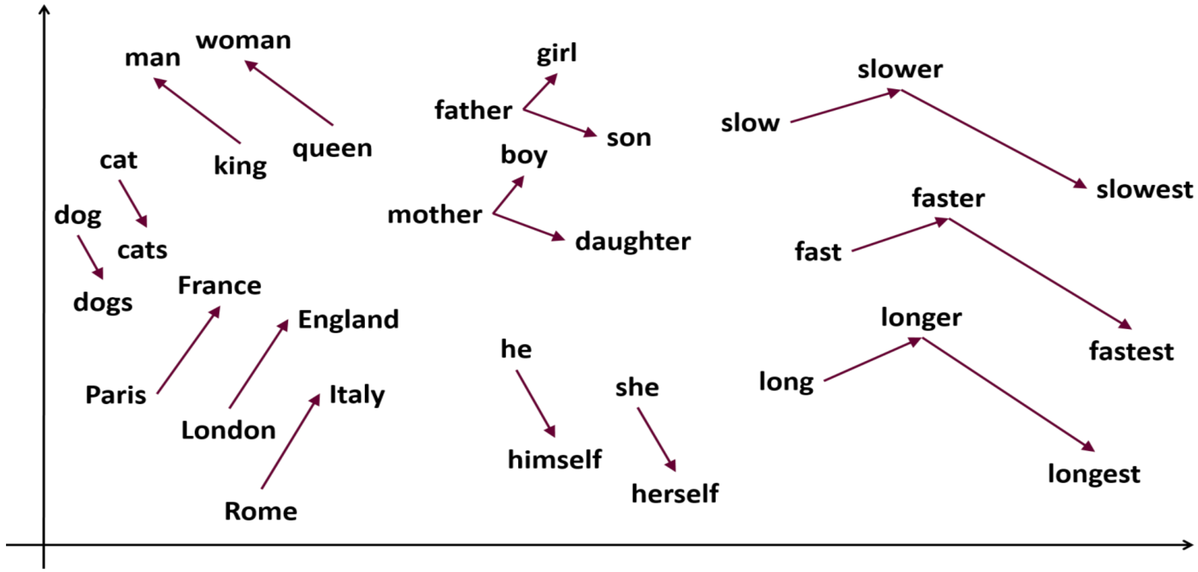
- One-hot Encoding



Source :(Marco Bonzanini, 2017)

Word Representation (Cont.)

- Word Embedding
  - Distributed representation



Word Embedding

- Word2Vec

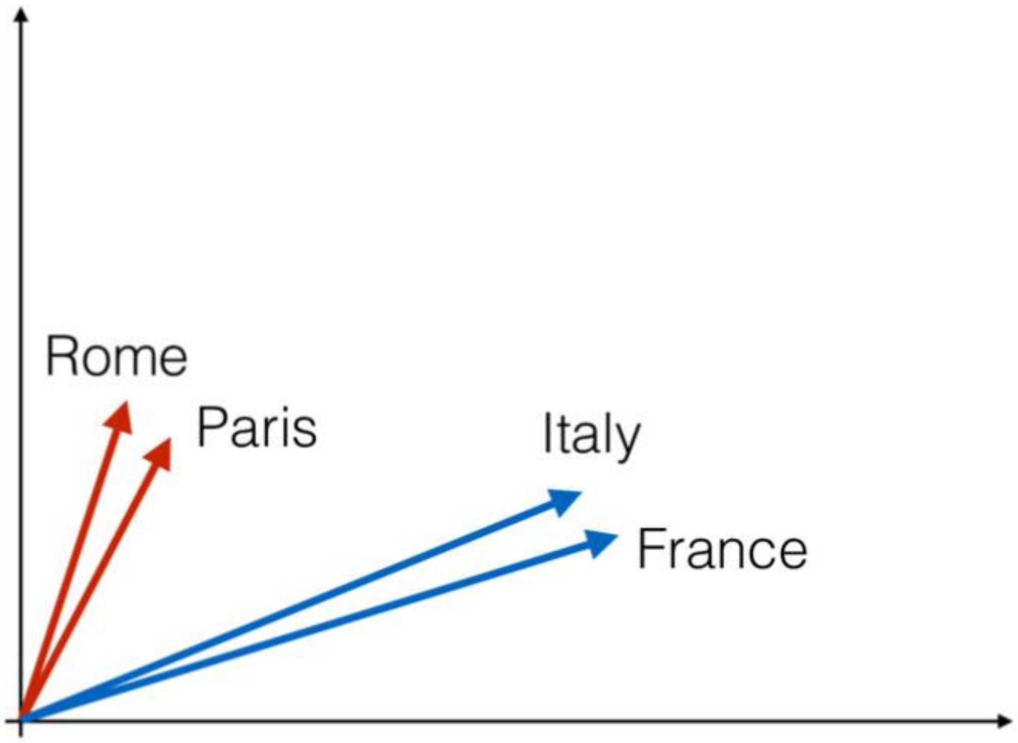
Rome = [0.91, 0.83, 0.17, …, 0.41]

Paris = [0.92, 0.82, 0.17, …, 0.98]

Italy = [0.32, 0.77, 0.67, …, 0.42]

France = [0.33, 0.78, 0.66, …, 0.97]

Word2Vec



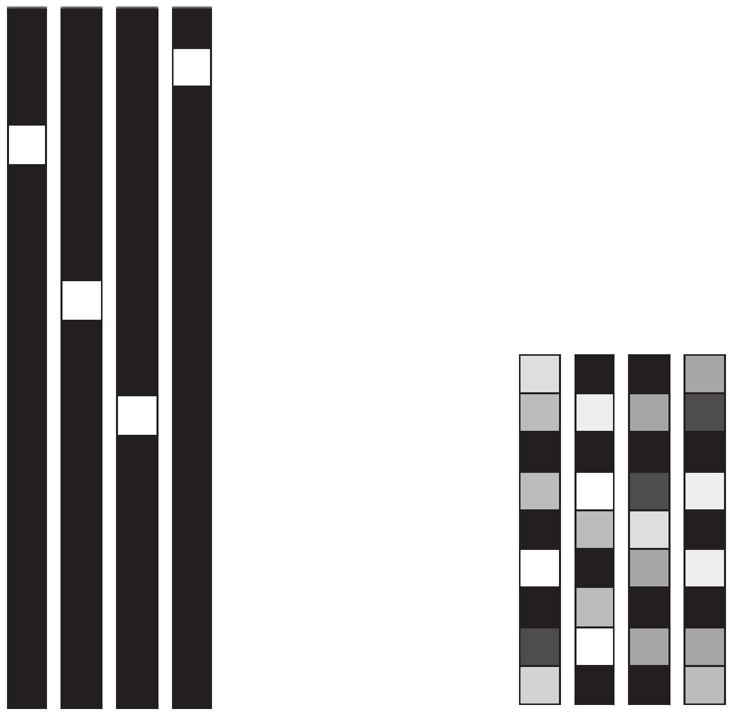
Word2Vec

- Learnt by Neural Networks

- Two models: Skip-Gram (SG) and Continuous Bag-of-Words (CBOW)
  - SG: predicting the context given a word
  - CBOW: predicting the word given its context

- In the process of predicting the target word, Word2Vec learns the vector representation of the target word.

One-hot vs Word Embedding



One-hot word vectors:
- Sparse
- High-dimensional
- Hardcoded

Word embeddings:
- Dense
- Lower-dimensional
- Learned from data

Source: *Deep Learning with Python* by Francois Chollet

Word2Vec in Gensim

- **from gensim.models import KeyedVectors**
- *# Load vectors directly from the file*
- model = KeyedVectors.load_word2vec_format('data/GoogleGoogleNews-vectors-negative300.bin', binary=True)
- *# Access vectors for specific words with a keyed lookup:*
- vector = model[rome']
- *# see the shape of the vector (300,)*
- vector.shape

Word2Vec in Gensim: Train your own

```
from gensim.models import Word2Vec
# define training data
sentences = [['this', 'is', 'the', 'first', 'sentence', 'for', 'word2vec'],
['this', 'is', 'the', 'second', 'sentence'],
['the', ‘final', 'sentence']]

# train model
model = Word2Vec(sentences, min_count=1)

# summarize vocabulary
words = list(model.wv.vocab)

# access vector for one word
print(model['sentence'])

# save model
model.save('model.bin')

# load model
new_model = Word2Vec.load('model.bin')
```

Word2Vec in Spacy

- **import spacy**
- *# Load the spacy model that you have installed*
- nlp = spacy.load('en_core_web_md')
- *# process a sentence using the model*
- doc = nlp("This is some text that I am processing with Spacy")
- *# It's that simple - all of the vectors and words are assigned after this point*
- *# Get the vector for 'text':*
- doc[3].vector
- *# Get the mean vector for the entire sentence (useful for sentence classification etc.)*
- doc.vector

Sub-Word

- Word representation cannot handle unseen word or rare word well. Character-level representation is one of the solution to overcome out-of-vocabulary (OOV). However, it may too fine-grained any missing some important information.
- Subword is in between word and character. It is not too fine-grained while able to handle unseen word and rare word.
  - Byte Pair Encoding (BPE)
  - WordPiece

Sub-Word: BPE

- Algorithm
  1. Prepare a large enough training data (i.e. corpus)
  2. Define a desired subword vocabulary size
  3. Split word to sequence of characters and appending suffix "</w>" to end of word with word frequency. So the basic unit is <u>character in this stage</u>. For example, the frequency of "low" is 5, then we rephrase it to "l o w </w>": 5
  4. Generating a new subword according to the high frequency occurrence.
  5. Repeating step 4 until reaching subword vocabulary size which is defined in step 2 or the next highest frequency pair is 1.

e.g., "low: 5", "lower: 2", "new**es**t: 6" and "wid**es**t: 3"

es is the highest frequency subword

Sub-Word: WordPiece

- Algorithm
    1. Prepare a large enough training data (i.e. corpus)
    2. Define a desired subword vocabulary size
    3. Split word to sequence of characters
    4. <u>Build a languages model </u>based on step 3 data
    5. Choose the new word unit out of all the possible ones that increases the likelihood on the training data the most when added to the model.
    6. Repeating step 5until reaching subword vocabulary size which is defined in step 2 or the likelihood increase falls below a certain threshold.

Character Representation

- One-hot Encoding
  - 26/52-dimensional vector to represent a character
  - Easy to form vector for unseen words
  - For misspelling words
  - Handles infrequent words better than Word2Vec

- A good paper to read:

  Zhang et al. *Character-level Convolutional Networks for Text Classification*. NIPS 2015.

Text Classification

- Text Representation
  - Sentence Representation
  - Word Representation
  - Character Representation
- Pre-processing
- Simple DNN Examples

Pre-processing

- Tokenization
- Remove stop words
- Lowercase/uppercase
- (Check misspelling)
- **Lemmatization or Stemming**
- …

Lemmatization & Stemming

- The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

Connect
Connections------> Connect
Connected-----> Connect
Connecting-----> Connect
Connection-----> Connect

Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
- *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*
- Lemmatization: have to find correct dictionary of headword form

Stemming

- Morphemes:
  - The small meaningful units that make up words
  - Stems: The core meaning-bearing units
  - Affixes: Bits and pieces that adhere to stems
- Stemming:
  - Reduce terms to their stems
  - *Stemming* is crude chopping of affixes
    - language dependent
    - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

Porter's algorithm
The most common English stemmer

Step 1a

```
sses → ss    caresses → caress
ies  → i     ponies   → poni
ss   → ss    caress   → caress
s    → ø     cats     → cat
```

Step 1b

```
(*v*)ing → ø  walking   → walk
(*v*)ed  → ø  plastered → plaster
…
```

Step 2 (for long stems)

```
ational→ ate  relational→ relate
izer→ ize     digitizer → digitize
ator→ ate     operator  → operate
…
```

Step 3 (for longer stems)

```
al   → ø  revival    → reviv
able → ø  adjustable → adjust
ate  → ø  activate   → activ
…
```

Lemmatization in NLTK

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# You will have to download the set of stop words the first time
nltk.download('stopwords’)
 # Load stop words
stop_words=stopwords.words(‘english’)
example_sent = "This is a sample sentence”
word_tokens = word_tokenize(example_sent)

wnl = WordNetLemmatizer()
wnl.lemmatize(“sample")
```

'sample'

Stemming in NLTK

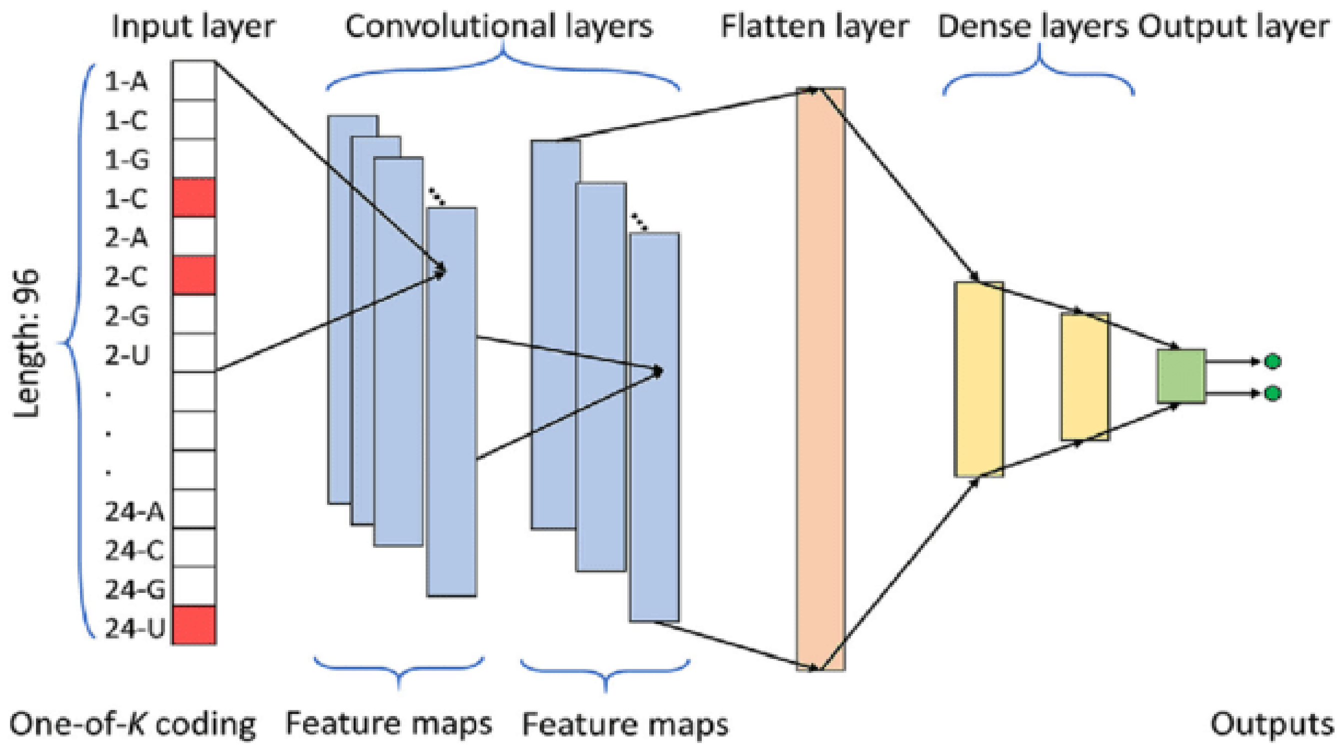
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import *

# You will have to download the set of stop words the first time
nltk.download('stopwords’)
 # Load stop words
stop_words=stopwords.words(‘english’)
example_sent = "This is a sample sentence”
word_tokens = word_tokenize(example_sent)

porter = PorterStemmer()
porter.stem(“sample")
```

'sampl'

Text Classification

- Text Representation
  - Sentence Representation
  - Word Representation
  - Character Representation
- Pre-processing
- Simple DNN Examples

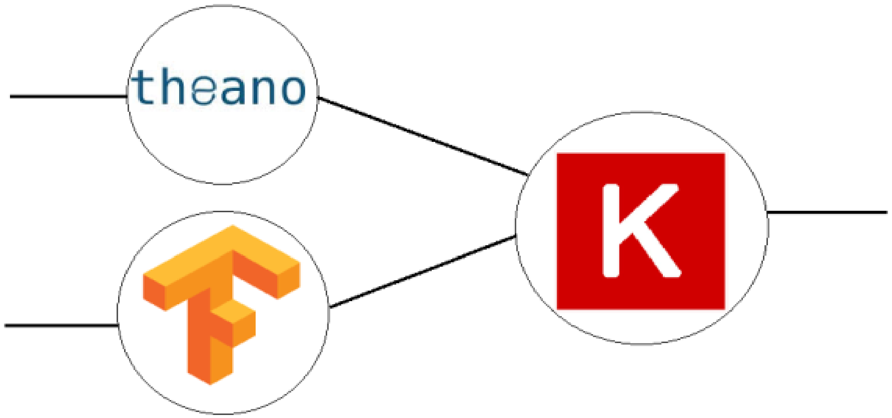
Text Classification Model: 1D CNN



Source: Qi Zhao et al. 2018

Keras

- High-level neural networks API, written in Python.
- On top of TensorFlow, CNTK, Theano
- For fast experimentation
- Support both CPU and GPU

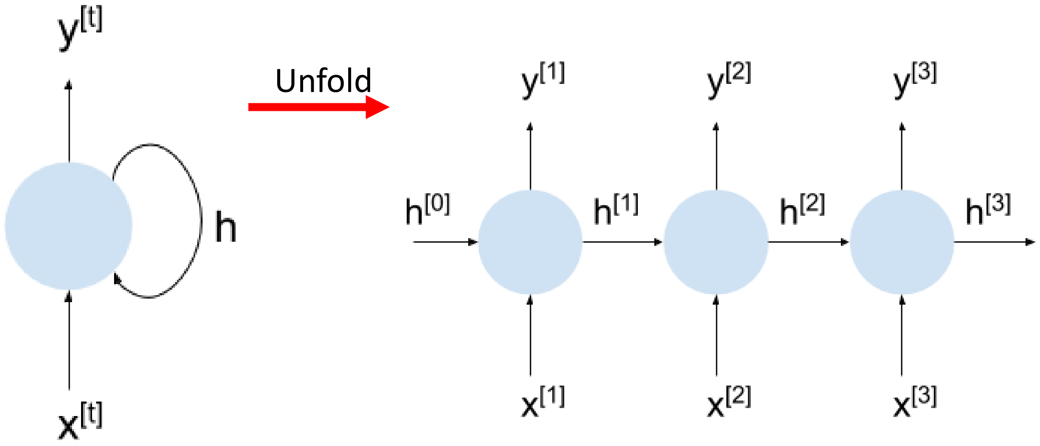


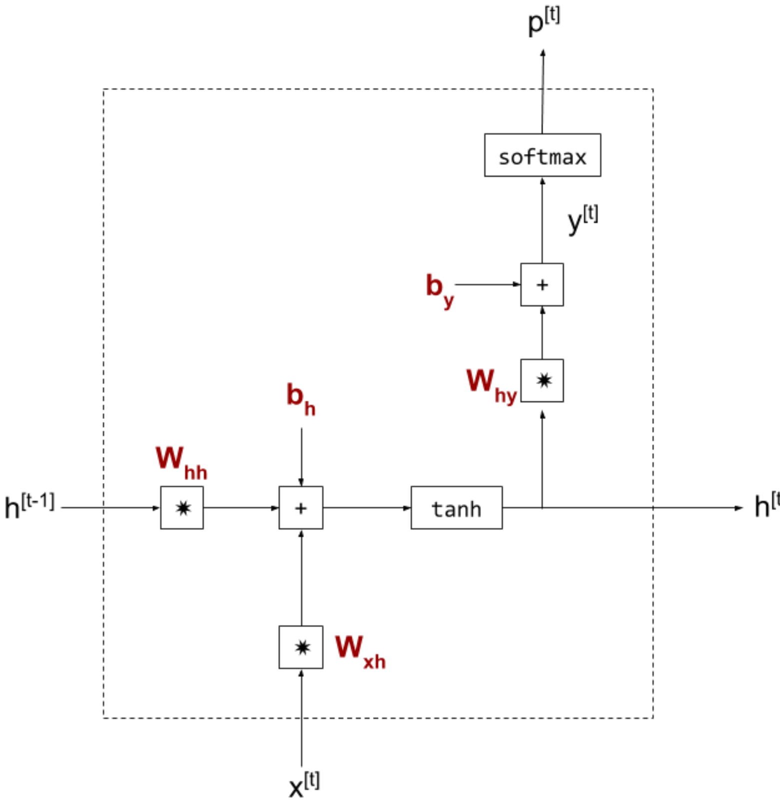
Keras - CNN

This is not complete code!!

```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, GlobalMaxPooling
1D, Dense, Dropout, Flatten, MaxPooling1D
..
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_len,weigh
ts=[embedding_matrix],trainable=False))
model.add(Conv1D(512, 3, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])


model.fit(X_train, y_train,batch_size = 50, epochs =10 , verbose = 1,
         validation_split=0.1)

 loss, accuracy = model.evaluate(X_test, y_test, verbose=True)
 print("Testing Accuracy:  {:.4f}".format(accuracy))
```

Text Classification Model: RNN



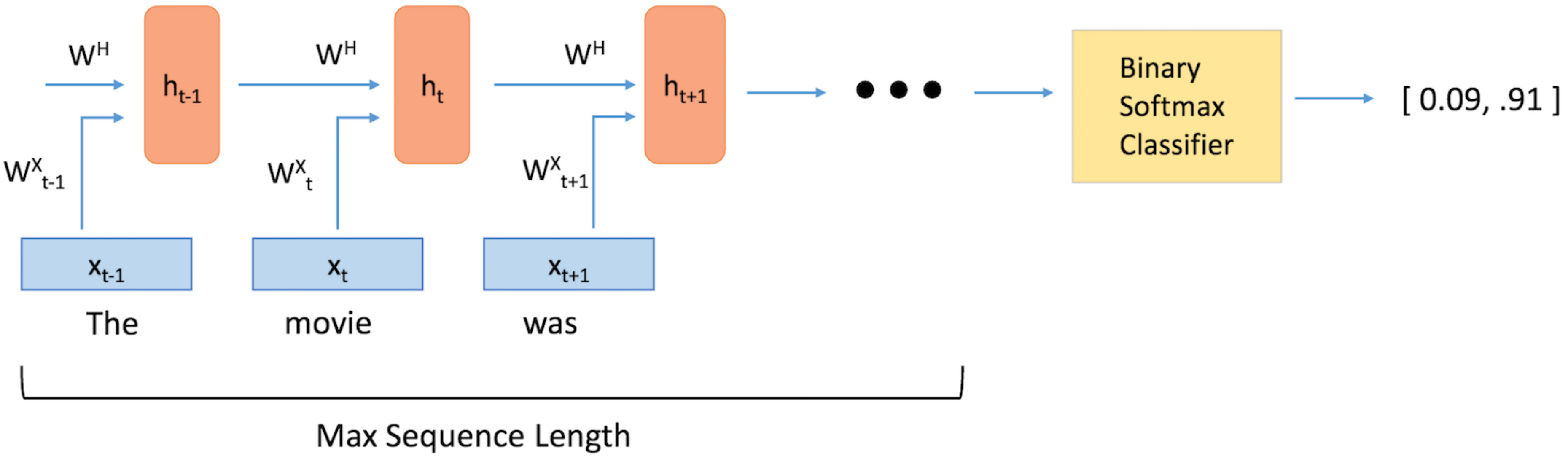


x is the input vector (at time step *t*), *y* is the output vector and *h* is the *state vector* kept inside the model.

Internal-structure of the RNN cell

Source: Eli Bendersky 2018

Text Classification Model: RNN



Keras - RNN

```
from keras.layers import Embedding, SimpleRNN
…
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_le
ngth=max_len,weights=[embedding_matrix],trainable=False
))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossen
tropy', metrics=['acc'])

model.fit(X_train, y_train,
                    epochs=2,
                    batch_size=128,
                    validation_split=0.2)


loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
```

Google Codelab for Deep Learning

- A free cloud service
- Can save notebooks to Google Drive
- Jupyter Notebooks: Tensorflow, Keras, PyTorch
- Free GPU
  - Nvidia T4: 16GB of GPU memory
  - “The best available hardware is prioritized for users who use Colaboratory interactively rather than for long-running computations.”
- More details here: https://colab.research.google.com/notebooks/welcome.ipynb