Deep Learning & NLP

Francesco Pugliese, PhD

Italian National Institute of Statistics, Division "Information and Application Architecture", Directorate for methodology and statistical design

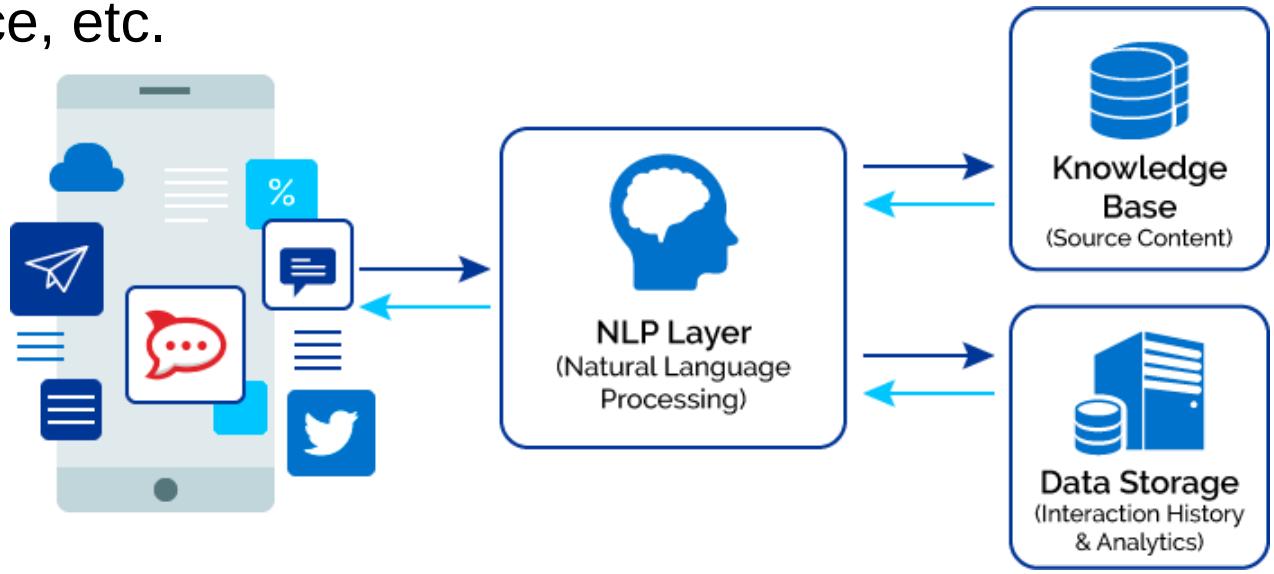
Email **Francesco Pugliese** :

f.pugliese@deeplearningitalia.com

francesco.pugliese@istat.it

Textual Big Data alias The problem of the Natural Languale Processing - NLP

- Understanding **complex language utterances is one of the hardest challenge** for Artificial Intelligence (AI) and Machine Learning (ML).
- **NLP** is everywhere because people communicate most everything: web search, advertisement, emails, customer service, etc.



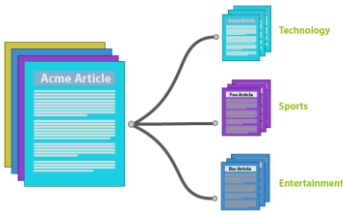
Deep Learning and NLP

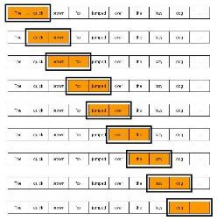
- "Deep Learning" approaches have obtained very high performance across many different **NLP** tasks. These models can often be trained with a **single end-to-end model** and do not require traditional, task-specific feature engineering.
  (*Stanford University School Of Engineering – CS224D*)

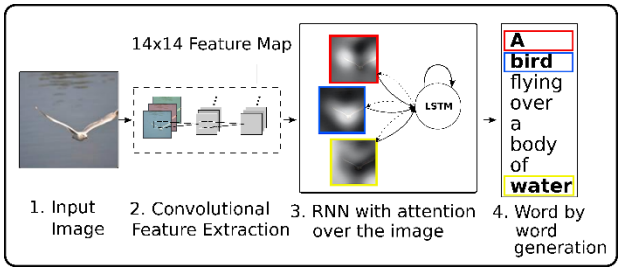
- **Natural language processing** is shifting from statistical methods to **Neural Networks**.

7 NLP applications where Deep Learning achieved «state-of-art» performance

- **1 <u>Text Classification</u>**: Classifying the topic or theme of a document (i.e. **Sentiment <u>Analysis</u>**).
- **2 <u>Language Modeling</u>**: Predict the **next word given the previous words**. It is fundamental for other tasks.
- **3 <u>Speech Recognition</u>**: Mapping an **acoustic signal** containing a spoken natural language utterance into the corresponding sequence of words intended by the speaker.
- **4 <u>Caption Generation</u>**: Given a **digital image**, such as a photo, generate a **textual description** of the contents of the image.







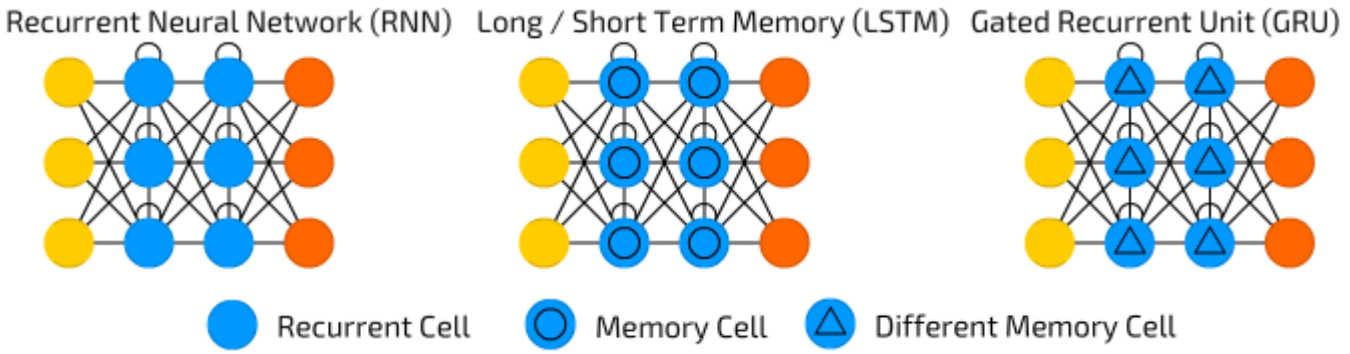
7 NLP applications where Deep Learning achieved «state-of-art» performance

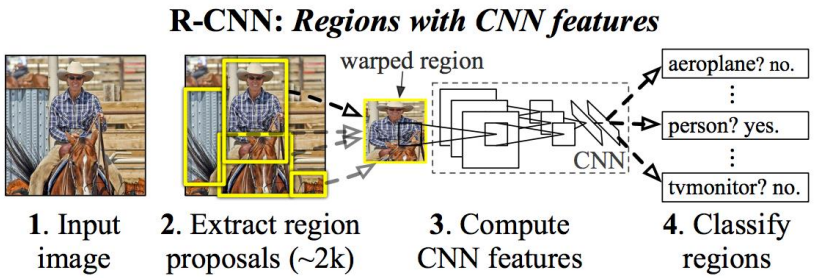
- **5 Machine Translation**: Automatic translation of text or speech from one language to another, is one [of] the most important applications of NLP.
- **6 Document Summarization**: It is the task where a short description of a text document is created.
- **7 Question Answering**: It is the task where the system tries to answer a user query that is formulated in the form of a question by returning the appropriate noun phrase such as a location, a person, or a date. (i.e. Who killed President Kennedy? Oswald)



Text Classification Models

- **RNN, LSTM, GRU, ConvLstm, RecursiveNN, RNTN, RCNN**
- The modus operandi for text classification involves the use of a pre-trained **word embedding** for **representing words** and a **deep neural networks** for **learning how to discriminate documents** on classification problems.



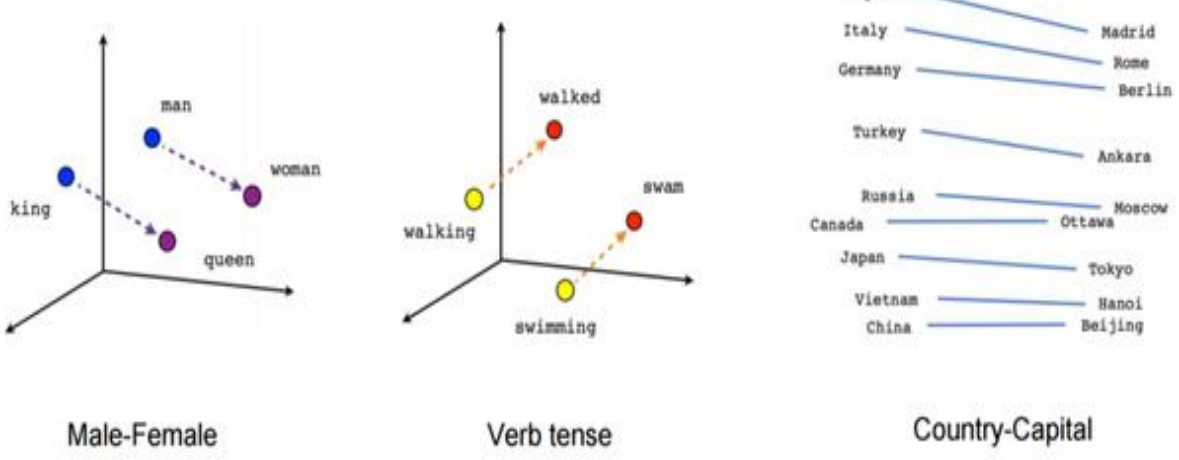


- The **non-linearity of the NN** leads to superior classification accuracy.

Word Embedding & Language Modeling

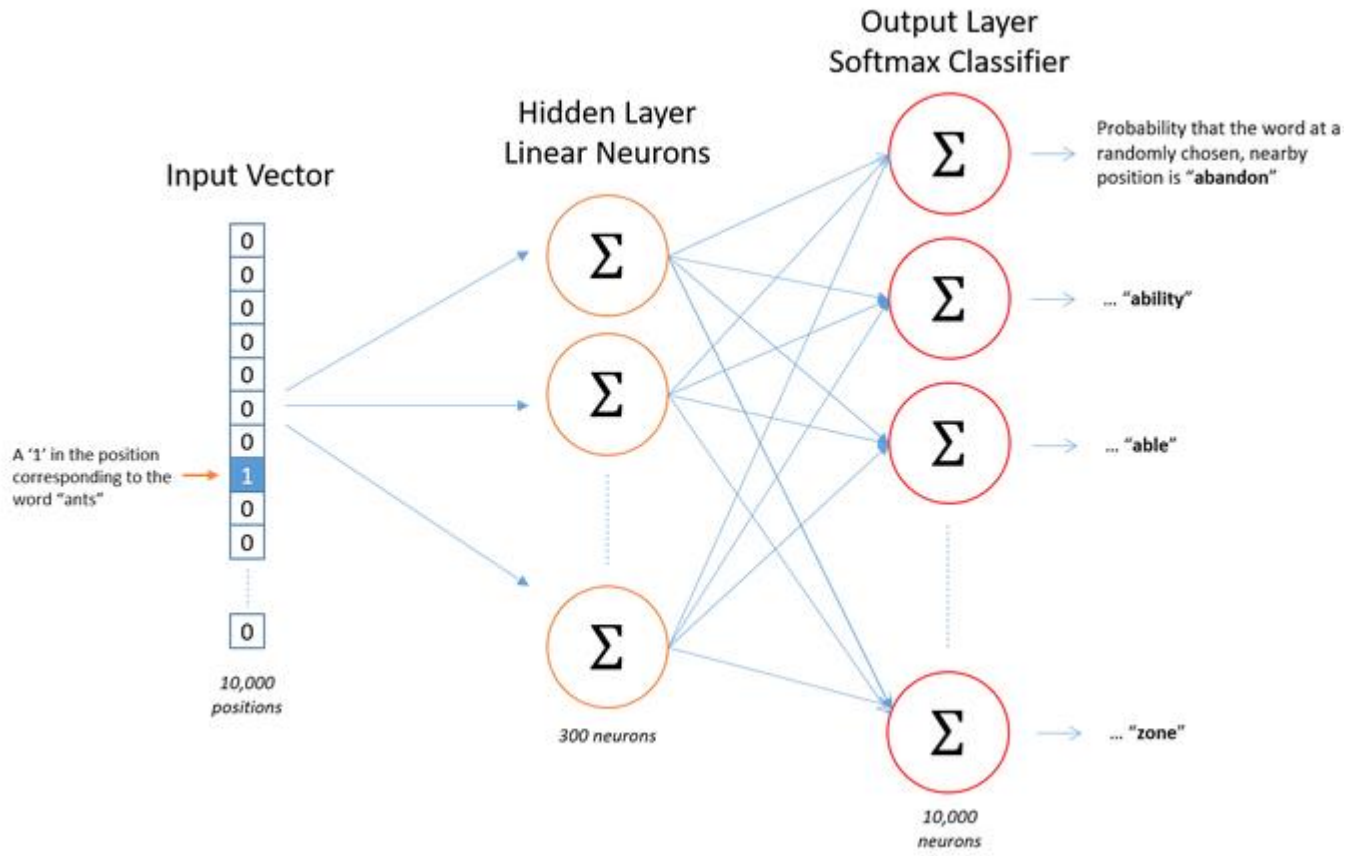
- Word embedding is the collective name for a set of language modeling and feature learning techniques for natural language processing (NLP) where words or sentences from the vocabulary are mapped to vectors of real numbers.

- These vectors are semantically correlated by metrics like cosine distance

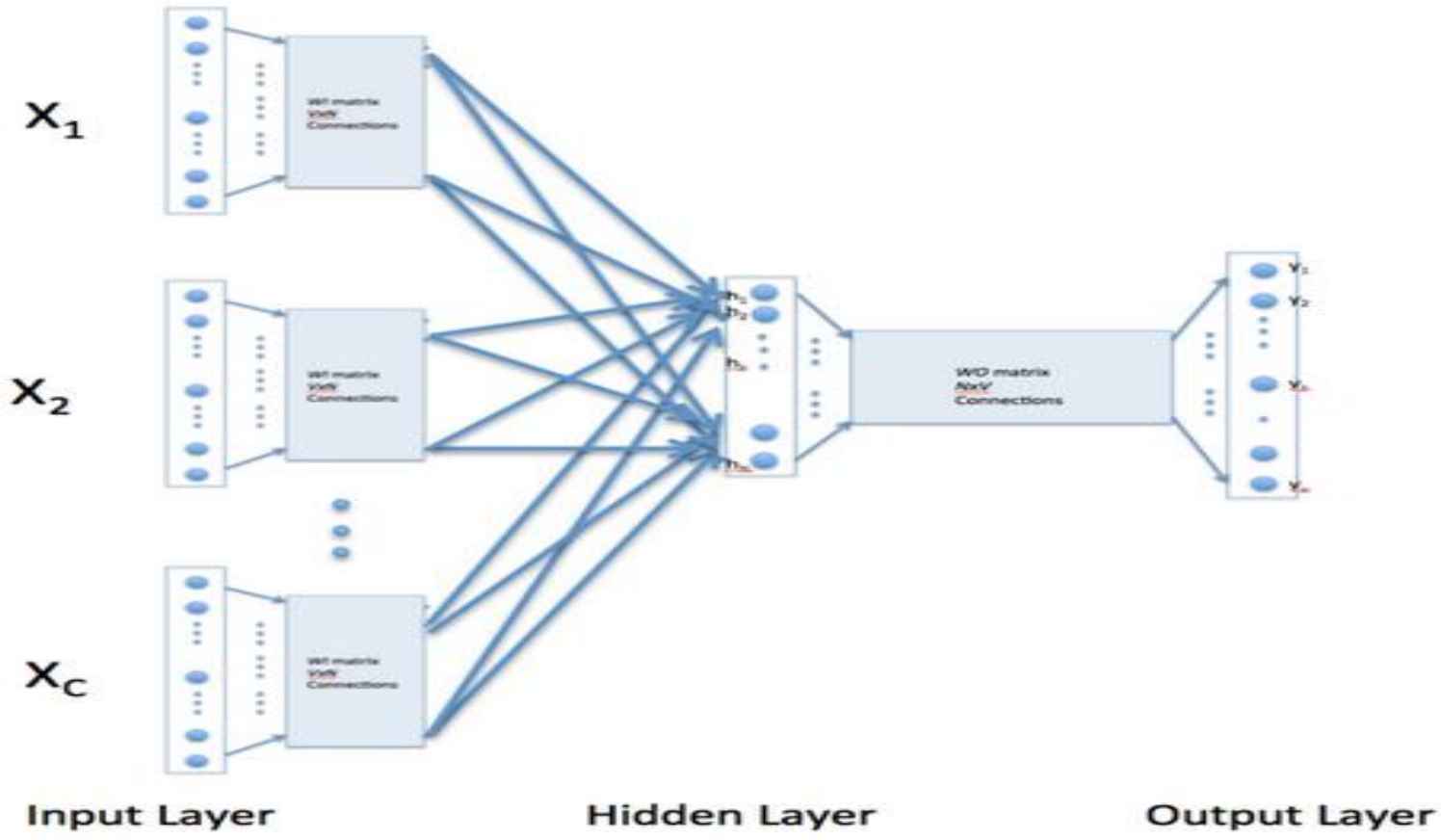


Male-Female | Verb tense | Country-Capital

Skip-Gram Model (Mikolov, et. al., 2013)



C-BOW Model (Bow, et al., 2003).

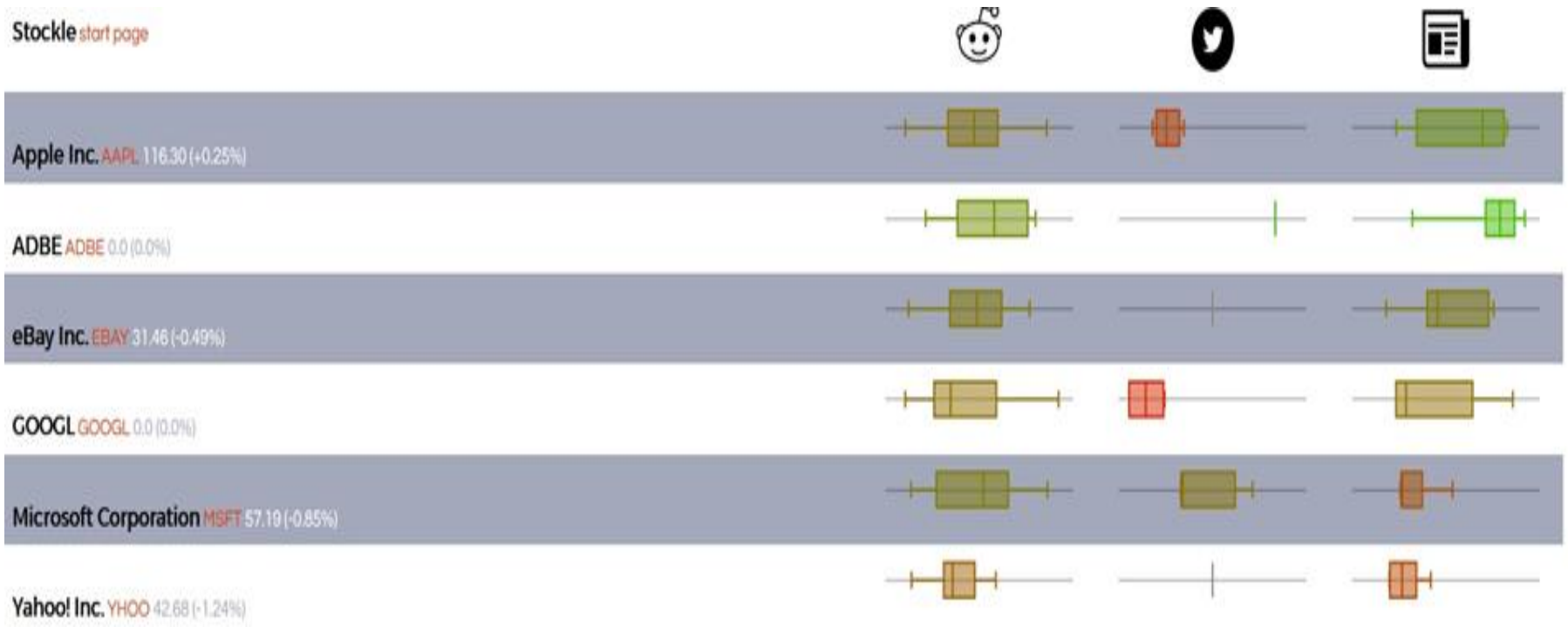


Sentiment Analysis (*Ain, et al. 2017*)

- **Sentiments** of users that are expressed on the web has great influence on the readers, product vendors and politicians.
- **Sentiment Analysis** refers to text organization for the classification of mind-set or feelings in different manners such as negative, positive, favorable, unfavorable, thumbs up, thumbs down, etc. Thanks to DL, the SA can be visual as well.

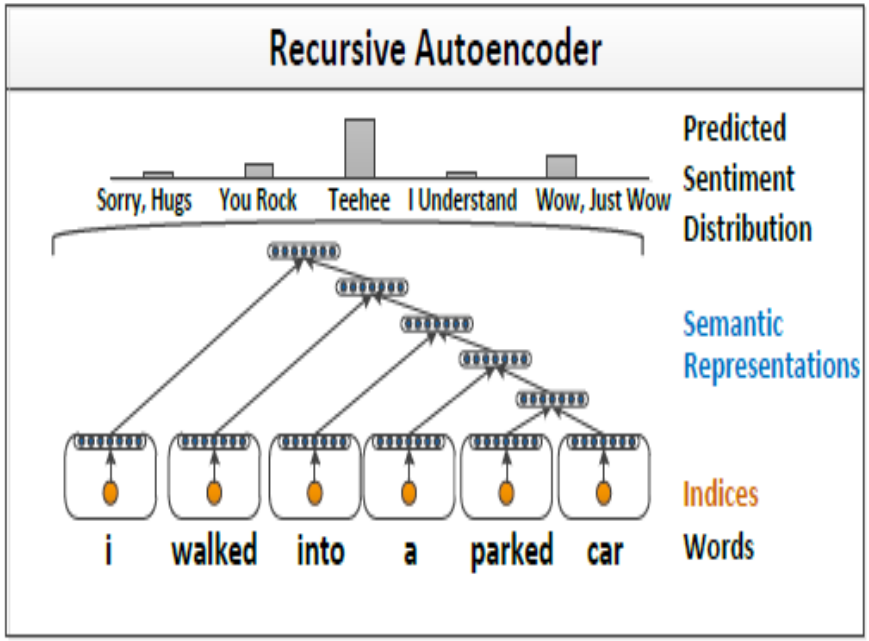
Discovering people opinions, emotions and feelings about a product or service

Sentiment Analysis with Feedback



Recursive Neural Tensor Networks (RecursiveNN) (Socher, R., et al., 2011b)

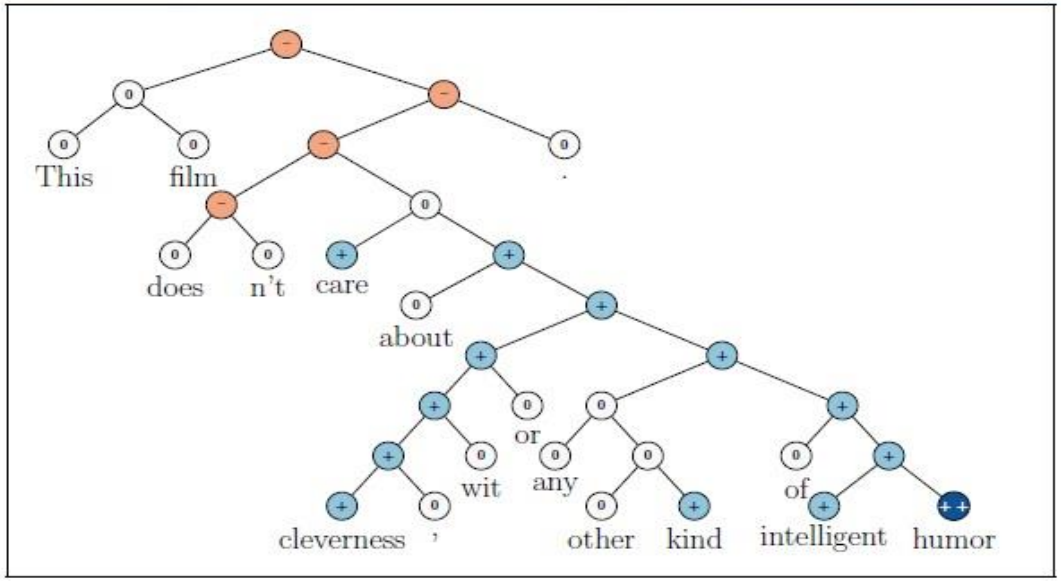
- This models are recursive auto-encoders which learn semantic vector representations of phrases. Word indices (orange) are first mapped into a semantic vector space (blue).
- Then they are recursively merged by the same auto-encoder network into a fixed length sentence representation. The vectors at each node are used as features to predict a distribution over text labels.



Recursive Neural Tensor Networks (RNTN) (Socher, R., et al. 2013)

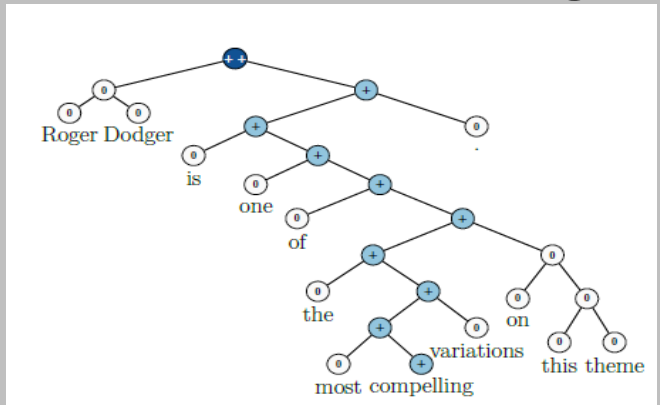
- **The Stanford Sentiment Treebank is the first corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language.**

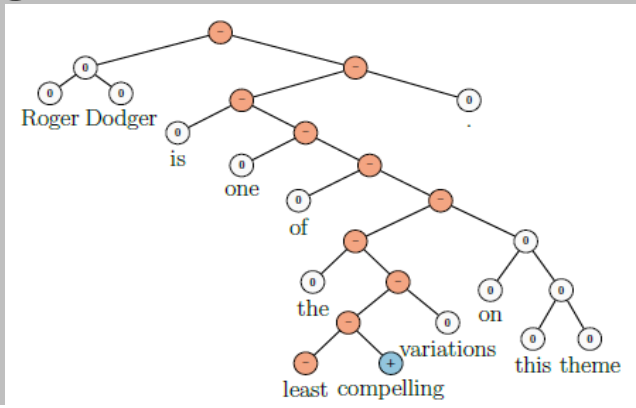
- **RNTNs compute parent vectors in a bottom up fashion using a compositionality function and use node vectors as features for a classifier at that node.**



RNTN – Upside and Downside

- RNTNS are very efficient in terms of constructing sentence representations.
- RNTNs capture the semantics of a sentence via a tree structure. Its performance heavily depends on the performance of the textual tree construction.
- Constructing such a textual tree exhibits a time complexity of at least O(n2), where n is the length of the text.
- RNTNs are unsuitable for modeling long sentences or documents.



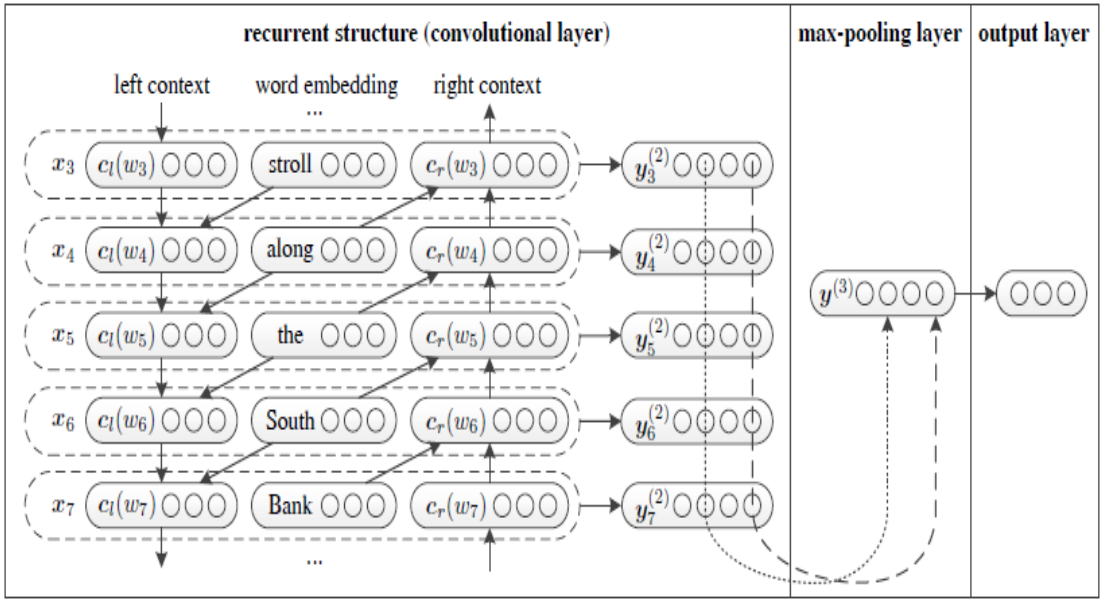


Recurrent Convolutional Neural Networks (RCNN) (*Lai, S., et al. 2015*)

- They adopt a a recurrent structure to **capture contextual information** as far as possible when learning word representations, which may introduce considerably **less noise compared** to traditional window-based neural networks.

- The **bi-directional recurrent structure** of RCNNs.

- **RCNNs** exhibit a time complexity of O(n)



RCNN Equations

- RCNNs exhibit a **time complexity of O(n)** , which is linearly correlated with the length of the text length.

- **7 equations** defining all the Neural Network topology

- **Input length** can be variable

$$c_l(w_i) = f(W^{(l)} c_l(w_{i-1}) + W^{(sl)} e(w_{i-1})) \quad (1)$$

$$c_r(w_i) = f(W^{(r)} c_r(w_{i+1}) + W^{(sr)} e(w_{i+1})) \quad (2)$$

$$x_i = [c_l(w_i); e(w_i); c_r(w_i)] \quad (3)$$

$$y_i^{(2)} = \tanh\left(W^{(2)} x_i + b^{(2)}\right) \quad (4)$$

$$y^{(3)} = \max_{i=1}^{n} y_i^{(2)} \quad (5)$$

$$y^{(4)} = W^{(4)} y^{(3)} + b^{(4)} \quad (6)$$

$$p_i = \frac{\exp\left(y_i^{(4)}\right)}{\sum_{k=1}^{n} \exp\left(y_k^{(4)}\right)} \quad (7)$$

RCNN in Keras

```
class SentimentModelRecConvNet:
    @staticmethod

    def build(input_length, vector_dim):
        hidden_dim_RNN = 200
        hidden_dim_Dense = 100

        embedding = Input(shape=(input_length, vector_dim))

        left_context = LSTM(hidden_dim_RNN, return_sequences = True)(embedding)                                  # Equation 1
        # left_contex: batch_size x tweet_length x hidden_state_dim
        right_context = LSTM(hidden_dim_RNN, return_sequences = True, go_backwards = True)(embedding)            # Equation 2
        # right_cntext: come left_contex
        together = concatenate([left_context, embedding, right_context], axis = 2)                               # Equation 3
        semantic = TimeDistributed(Dense(hidden_dim_Dense, activation = "tanh"))(together)                       # Equation 4
        pool_rnn = Lambda(lambda x: backend.max(x, axis = 1), output_shape = (hidden_dim_Dense, ))(semantic)      # Equation 5
        pool_rnn_args = Lambda(lambda x: backend.argmax(x, axis=1), output_shape = (hidden_dim_Dense, ))(semantic)

        output = Dense(1, input_dim = hidden_dim_Dense, activation = "sigmoid")(pool_rnn)                        # Equations 6, 7

        deepnetwork = Model(inputs=embedding, outputs=output)
        deepnetwork_keywords = Model(inputs=embedding, outputs=pool_rnn_args)

        return [deepnetwork, deepnetwork_keywords]
```

RCNN: Feature Extraction

- RCNNs employ a max-pooling layer that automatically judges which words play key roles in text classification to capture the key components in texts.

- The most important words are the information most frequently selected in the max-pooling layer.

- Contrary to the most positive and most negative phrases in RNTN, RCNN does not rely on a syntactic parser, therefore, the presented n-grams are not typically “phrases”.

	RCNN
P	well *worth* the; a *wonderful* movie; even *stinging* at; and *invigorating* film; and *ingenious* entertainment; and *enjoy* .; ’s *sweetest* movie
N	A *dreadful* live-action; Extremely *boring* .; is *n’t* a; ’s *painful* .; Extremely *dumb* .; an *awfully* derivative; ’s *weaker* than; incredibly *dull* .; very *bad* sign;
	RNTN
P	an amazing performance; most visually stunning; wonderful all-ages triumph; a wonderful movie
N	for worst movie; A lousy movie; a complete failure; most painfully marginal; very bad sign

RCNN applied to Extractive Text Summarization

- **Best keywords** lead to **best contextes** ---> **Summarization**

```
Tweet 29: "GiÃ  avete letto 136 pagine del piano scuola? #Fenomeni #labuonascuola"

Sentiment: -0.95 - -1
Keywords: pagine, avete, fenomeni, piano

Tweet 30: "\"Per l'#aternanza #scuola #lavoro bisogna passare da 11a 100milioni di euro\" #labuonascuola http://t.co/zGAzkni8rv"

Sentiment: -0.81 - -1
Keywords: euro, t, scuola, lavoro



Most significant keywords driving the sentiment decision:

Eccolo

Siamo

Scuola

Giuste

Escluso

Most significant sentences driving the sentiment decision:

...cambierÃ  solo se noi metteremo al centro...

...solo se noi metteremo al centro la...

...piÃ¹ grande spettacolo mai visto passodopopasso scuola...

...mai visto passodopopasso scuola labuonascuola...

...nessuno si senta escluso la buona scuola...
```

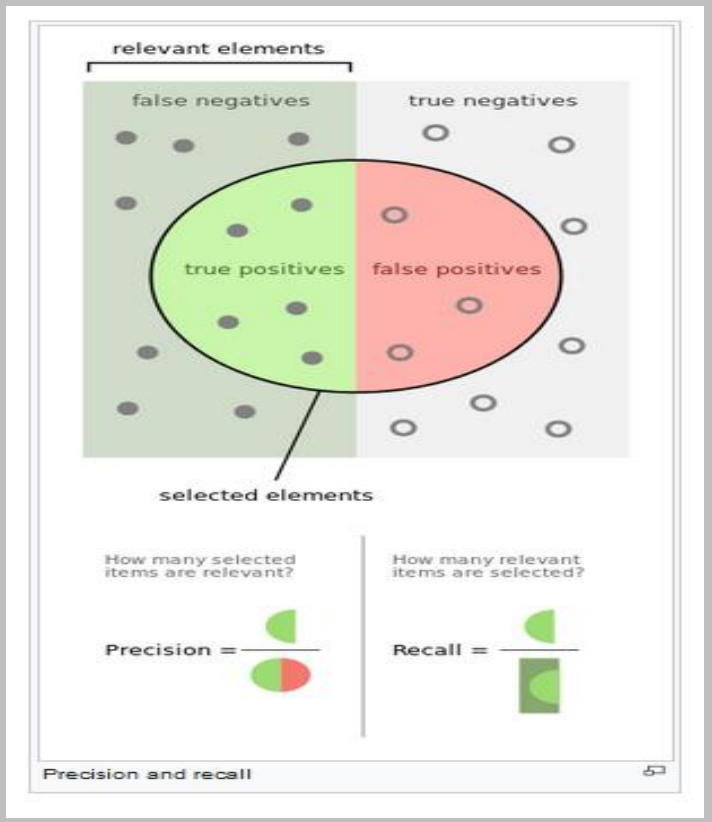
Recurrent Neural Networks are able to understand negations and other things

- Thanks to **word embeddings** semantics **RNNs** can recognize **nagations**, and complex **forms** of **language utterances**.

```
Tweet: This is a bad thing
- Sentiment: -0.72 - -1

Keywords: bad, thing, a, is

Tweet: This is not a bad thing
- Sentiment: 0.46 - +1

Keywords: not, thing, bad, a

Tweet: This is a positive thing
- Sentiment: 0.94 - +1

Keywords: positive, thing, a, is

Tweet: This is a very positive thing
- Sentiment: 0.91 - +1

Keywords: positive, very, thing, a

Tweet: I like Renzi politics
- Sentiment: 0.70 - +1

Keywords: like, renzi, politics, i

Tweet: I don't agree with Renzi Politics
- Sentiment: 0.16 - 0

Keywords: don't, agree, politics, renzi

Tweet: Renzi did a wrong international Politics
- Sentiment: -0.34 - -1

Keywords: wrong, did, renzi, international

Tweet: Renzi did a very good international Politics
- Sentiment: 0.74 - +1

Keywords: did, renzi, good, very

Tweet: Istat is a very good Institute of research
- Sentiment: 0.84 - +1

Keywords: good, very, research, istat

Tweet: Istat is not a good Institute of research - Sentiment: -0.78 - -1

Keywords: not, research, istat, institute
```

Classification Metrics

F-score



sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

specificity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

negative predictive value (NPV)

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}}$$

miss rate or false negative rate (FNR)

$$\mathrm{FNR} = \frac{\mathrm{FN}}{P} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{N} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

false discovery rate (FDR)

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$

false omission rate (FOR)

$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$$

accuracy (ACC)

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{P + N} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

F1 score

is the harmonic mean of precision and sensitivity

$$F_1 = 2 \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{\mathrm{PPV} + \mathrm{TPR}} = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}$$

Use Case 2: Classification of Cifar 10 with CNN in Keras

Metrics

		True condition			
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{\text{TPR}}{\text{FPR}}$	Diagnostic odds ratio (DOR) = $\frac{\text{LR+}}{\text{LR−}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{\text{FNR}}{\text{TNR}}$	F_1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$

NLP with Word2Vec: Gensim library

class `gensim.models.word2vec.Word2Vec`(*sentences=None, corpus_file=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75, cbow_mean=1, hashfxn=<built-in function hash>, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False, callbacks=(), max_final_vocab=None*)

Bases: `gensim.models.base_any2vec.BaseWordEmbeddingsModel`

Train, use and evaluate neural networks described in **https://code.google.com/p/word2vec/**.

Once you're finished training a model (=no more updates, only querying) store and use only the `KeyedVectors` instance in *self.wv* to reduce memory.

The model can be stored/loaded via its `save()` and `load()` methods.

The trained word vectors can also be stored/loaded from a format compatible with the original word2vec implementation via *self.wv.save_word2vec_format* and `gensim.models.keyedvectors.KeyedVectors.load_word2vec_format()`.

Some important attributes are the following:

`wv`

`Word2VecKeyedVectors` – This object essentially contains the mapping between words and embeddings. After training, it can be used directly to query those embeddings in various ways. See the module level docstring for examples.

NLP with Word2Vec: KeyedVectors

class `gensim.models.keyedvectors.Word2VecKeyedVectors`(*vector_size*)

Bases: `gensim.models.keyedvectors.WordEmbeddingsKeyedVectors`

Mapping between words and vectors for the `Word2Vec` model. Used to perform operations on the vectors such as vector lookup, distance, similarity etc.

`accuracy`(***kwargs*)

Compute accuracy of the model.

The accuracy is reported (=printed to log and returned as a list) for each section separately, plus there's one aggregate summary at the end.

Parameters:

- **questions** (*str*) – Path to file, where lines are 4-tuples of words, split into sections by ": SECTION NAME" lines. See *gensim/test/test_data/questions-words.txt* as example.
- **restrict_vocab** (*int, optional*) – Ignore all 4-tuples containing a word not in the first *restrict_vocab* words. This may be meaningful if you've sorted the model vocabulary by descending frequency (which is standard in modern word embedding models).
- **most_similar** (*function, optional*) – Function used for similarity calculation.
- **case_insensitive** (*bool, optional*) – If True - convert all words to their uppercase form before evaluating the performance. Useful to handle case-mismatch between training tokens and words in the test set. In case of multiple case variants of a single word, the vector for the first occurrence (also the most frequent if vocabulary is sorted) is taken.

Returns: Full lists of correct and incorrect predictions divided by sections.

Return type: list of dict of (str, (str, str, str)

NLP with Keras: Embedding Layer

Embedding

[source]

```
keras.layers.Embedding(input_dim, output_dim, embeddings_initializer='uniform', embeddings_regularizer=None, activ
```

Turns positive integers (indexes) into dense vectors of fixed size. eg. [[4], [20]] -> [[0.25, 0.1], [0.6, -0.2]]

This layer can only be used as the first layer in a model.

Example

```
model = Sequential()
model.add(Embedding(1000, 64, input_length=10))
# the model will take as input an integer matrix of size (batch, input_length).
# the largest integer (i.e. word index) in the input should be
# no larger than 999 (vocabulary size).
# now model.output_shape == (None, 10, 64), where None is the batch dimension.

input_array = np.random.randint(1000, size=(32, 10))

model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
assert output_array.shape == (32, 10, 64)
```

NLP with Keras: Embedding Layer

Arguments

- **input_dim:** int > 0. Size of the vocabulary, i.e. maximum integer index + 1.
- **output_dim:** int >= 0. Dimension of the dense embedding.
- **embeddings_initializer:** Initializer for the `embeddings` matrix (see initializers).
- **embeddings_regularizer:** Regularizer function applied to the `embeddings` matrix (see regularizer).
- **embeddings_constraint:** Constraint function applied to the `embeddings` matrix (see constraints).
- **mask_zero:** Whether or not the input value 0 is a special "padding" value that should be masked out. This is useful when using recurrent layers which may take variable length input. If this is `True` then all subsequent layers in the model need to support masking or an exception will be raised. If mask_zero is set to True, as a consequence, index 0 cannot be used in the vocabulary (input_dim should equal size of vocabulary + 1).
- **input_length:** Length of input sequences, when it is constant. This argument is required if you are going to connect `Flatten` then `Dense` layers upstream (without it, the shape of the dense outputs cannot be computed).

Input shape

2D tensor with shape: `(batch_size, sequence_length)`.

Output shape

3D tensor with shape: `(batch_size, sequence_length, output_dim)`.

NLP with Keras: Conv1D Layer

Conv1D

[source]

```
keras.layers.Conv1D(filters, kernel_size, strides=1, padding='valid', data_format='channels_last', dilation_rate=1
```

1D convolution layer (e.g. temporal convolution).

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, does not include the batch axis), e.g. `input_shape=(10, 128)` for time series sequences of 10 time steps with 128 features per step in `data_format="channels_last"`, or `(None, 128)` for variable-length sequences with 128 features per step.

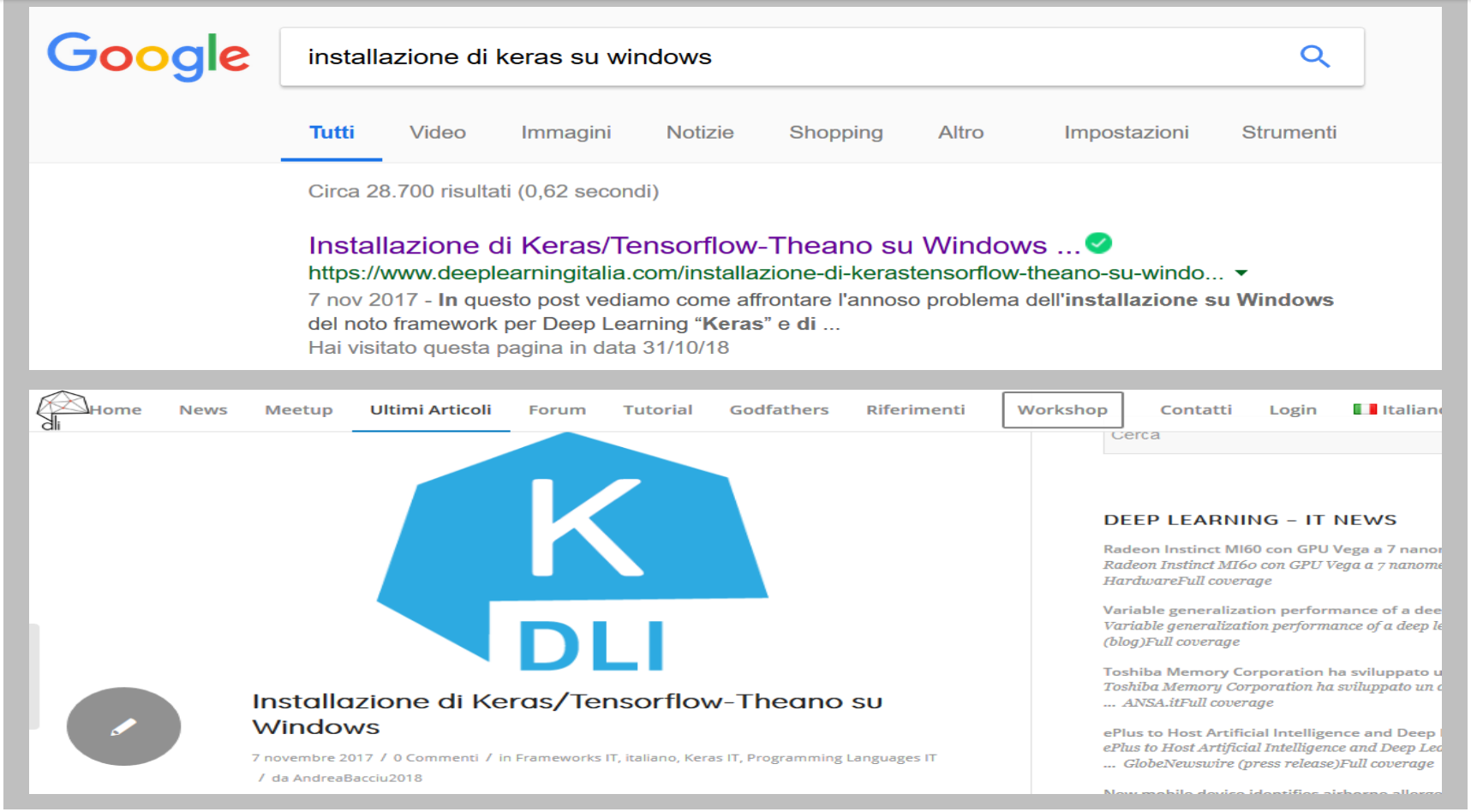
Arguments

- **filters:** Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size:** An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- **strides:** An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.

NLP with Keras: Conv1D Layer

- **padding:** One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"valid"` means "no padding". `"same"` results in padding the input such that the output has the same length as the original input. `"causal"` results in causal (dilated) convolutions, e.g. `output[t]` does not depend on `input[t + 1:]`. A zero padding is used such that the output has the same length as the original input. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.
- **data_format:** A string, one of `"channels_last"` (default) or `"channels_first"`. The ordering of the dimensions in the inputs. `"channels_last"` corresponds to inputs with shape `(batch, steps, channels)` (default format for temporal data in Keras) while `"channels_first"` corresponds to inputs with shape `(batch, channels, steps)`.
- **dilation_rate:** an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- **activation:** Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **use_bias:** Boolean, whether the layer uses a bias vector.
- **kernel_initializer:** Initializer for the `kernel` weights matrix (see initializers).
- **bias_initializer:** Initializer for the bias vector (see initializers).
- **kernel_regularizer:** Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **bias_regularizer:** Regularizer function applied to the bias vector (see regularizer).
- **activity_regularizer:** Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
- **kernel_constraint:** Constraint function applied to the kernel matrix (see constraints).
- **bias_constraint:** Constraint function applied to the bias vector (see constraints).

Exercise 3: Text Classifier in Python/Keras



Exercise 3: Text Classifier in Python/Keras

NLP: A Textual Classifier with Keras – 20NewsGroup Dataset

GloVe word embeddings

We will be using GloVe embeddings, which you can read about here. GloVe stands for "Global Vectors for Word Representation". It's a somewhat popular embedding technique based on factorizing a matrix of word co-occurence statistics.

Specifically, we will use the 100-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia. You can download them here (warning: following this link will start a 822MB download).

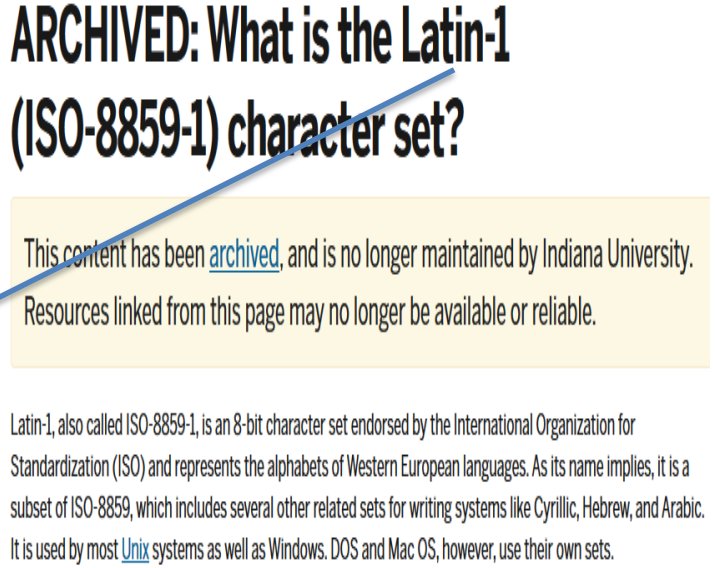
20 Newsgroup dataset

The task we will try to solve will be to classify posts coming from 20 different newsgroup, into their original 20 categories --the infamous "20 Newsgroup dataset". You can read about the dataset and download the raw text data here.

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

NLP Preprocessing: Dataset Loading

```
texts = []  # list of text samples
labels_index = {}  # dictionary mapping label name to numeric id
labels = []  # list of label ids
for name in sorted(os.listdir(TEXT_DATA_DIR)):
    path = os.path.join(TEXT_DATA_DIR, name)
    if os.path.isdir(path):
        label_id = len(labels_index)
        labels_index[name] = label_id
        for fname in sorted(os.listdir(path)):
            if fname.isdigit():
                fpath = os.path.join(path, fname)
                if sys.version_info < (3,):
                    f = open(fpath)
                else:
                    f = open(fpath, encoding='latin-1')
                t = f.read()
                i = t.find('\n\n')  # skip header
                if 0 < i:
                    t = t[i:]
                texts.append(t)
                f.close()
                labels.append(label_id)

print('Found %s texts.' % len(texts))
```



ARCHIVED: What is the Latin-1 (ISO-8859-1) character set?

This content has been archived, and is no longer maintained by Indiana University. Resources linked from this page may no longer be available or reliable.

Latin-1, also called ISO-8859-1, is an 8-bit character set endorsed by the International Organization for Standardization (ISO) and represents the alphabets of Western European languages. As its name implies, it is a subset of ISO-8859, which includes several other related sets for writing systems like Cyrillic, Hebrew, and Arabic. It is used by most Unix systems as well as Windows. DOS and Mac OS, however, use their own sets.

Building Texts List and Labels List

NLP Preprocessing: Dataset Loading

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(nb_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
```

Stop-words Cleaning and tokenization

```
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)
```

Building of the Word Index

Sequence 1D 0-Padding

```
labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

# split the data into a training set and a validation set
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

To categorical variables conversion

Shuffle of Data and Labels

Data and Labels Splitting

NLP Preprocessing: Embedding Layer Setting-Up

```
embeddings_index = {}
f = open(os.path.join(GLOVE_DIR, 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Found %s word vectors.' % len(embeddings_index))
```

Embedding vectors loading and check

```
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

Embedding matrix arrangement

```
from keras.layers import Embedding

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False)
```

Keras Embedding Layer primitives

NLP: Cov1D Model

Training a 1D convnet

Finally we can then build a small 1D convnet to solve our classification problem:

```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
x = Conv1D(128, 5, activation='relu')(embedded_sequences)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(5)(x)
x = Conv1D(128, 5, activation='relu')(x)
x = MaxPooling1D(35)(x)  # global max pooling
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
preds = Dense(len(labels_index), activation='softmax')(x)

model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

# happy learning!
model.fit(x_train, y_train, validation_data=(x_val, y_val),
          epochs=2, batch_size=128)
```

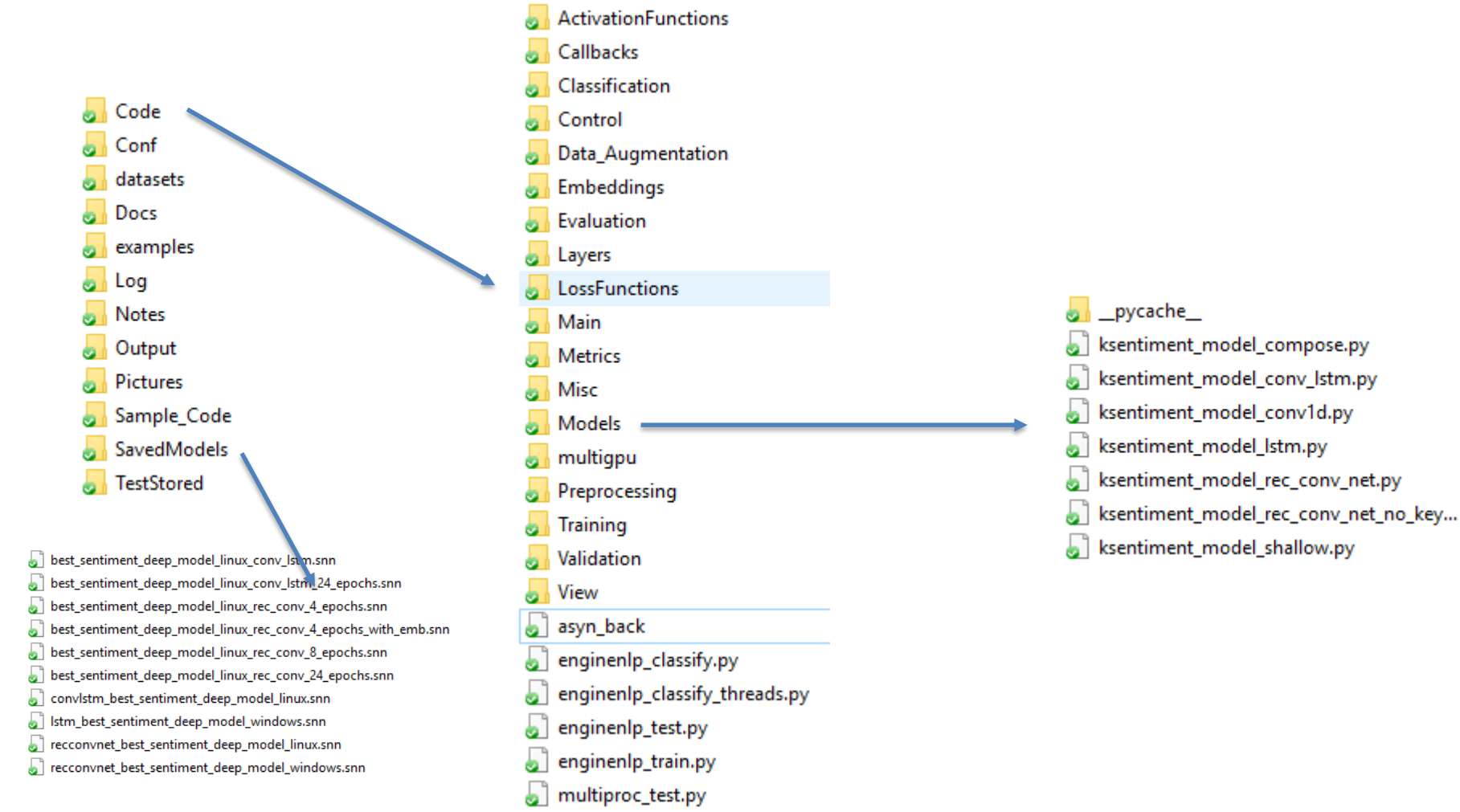
This model reaches **95% classification accuracy** on the validation set after only 2 epochs. You could probably get to an even higher accuracy by training longer with some regularization mechanism (such as dropout) or by fine-tuning the `Embedding` layer.

We can also test how well we would have performed by not using pre-trained word embeddings, but instead initializing our `Embedding` layer from scratch and learning its weights during training. We just need to replace our `Embedding` layer with the following:

```
embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            input_length=MAX_SEQUENCE_LENGTH)
```

After 2 epochs, this approach only gets us to **90% validation accuracy**, less than what the previous model could reach in just one epoch. Our pre-trained embeddings were definitely buying us something. In general, using pre-trained embeddings is relevant for natural processing tasks were little training data is available (functionally the embeddings act as an injection of outside information which might prove useful for your model).

Anatomy of an advanced Text Classification Application by Deep Learning in Keras



REFERENCES

Vinyals, O., & Le, Q. (2015). A neural conversational model. *arXiv preprint arXiv:1506.05869*.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Hochreiter S, Schmidhuber J. Long Short-Term Memory (1997). Neural Computation. 1997;9(8):1735–80. pmid:9377276

Bliemel F. Theil's (1973) Forecast Accuracy Coefficient: A Clarification. Journal of Marketing Research. 10(4):444.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., ... & Socher, R. (2016, June). Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning* (pp. 1378-1387).

Bow, C., Hughes, B., & Bird, S. (2003, July). Towards a general model of interlinear text. In *Proceedings of EMELD workshop* (pp. 11-13).

REFERENCES

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training gans. In *Advances in Neural Information Processing Systems* (pp. 2234-2242).

AKNOWLEDGEMENTS

THANK YOU

FOR YOUR ATTENTION

Francesco Pugliese