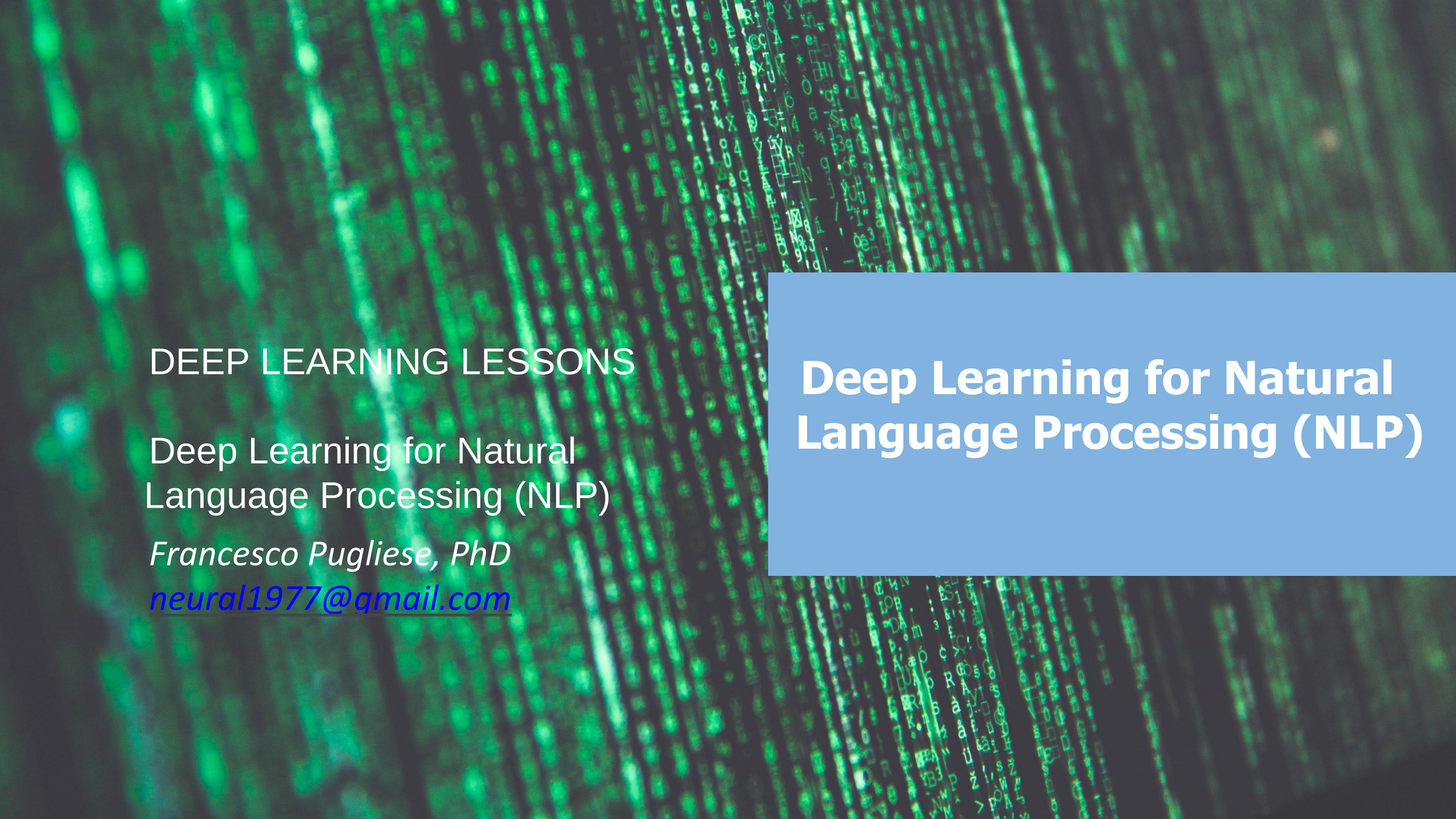


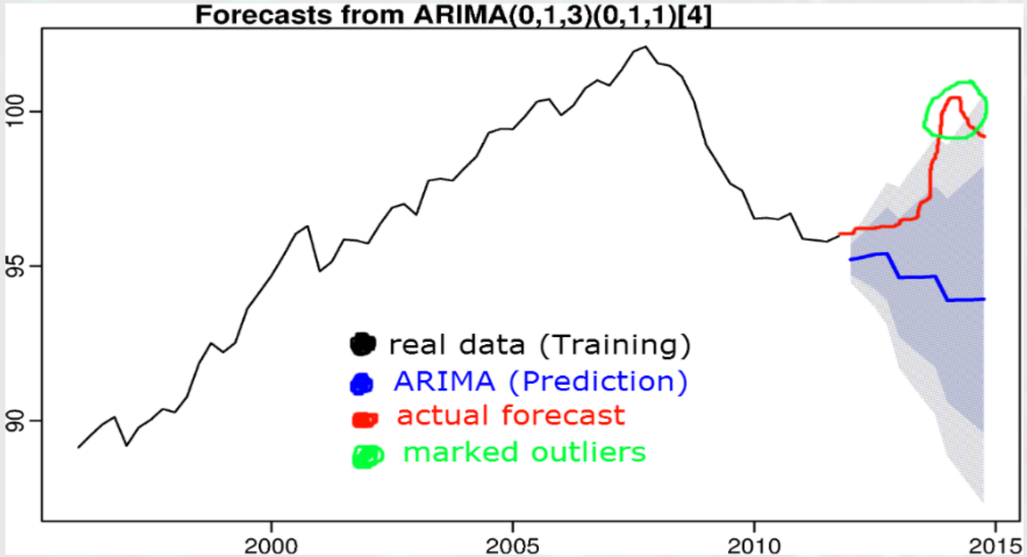
DEEP LEARNING LESSONS

Deep Learning for Natural Language Processing (NLP)

Francesco Pugliese, PhD
neural1977@gmail.com

Deep Learning for Natural Language Processing (NLP)

Deep Learning for Time-Series Prediction



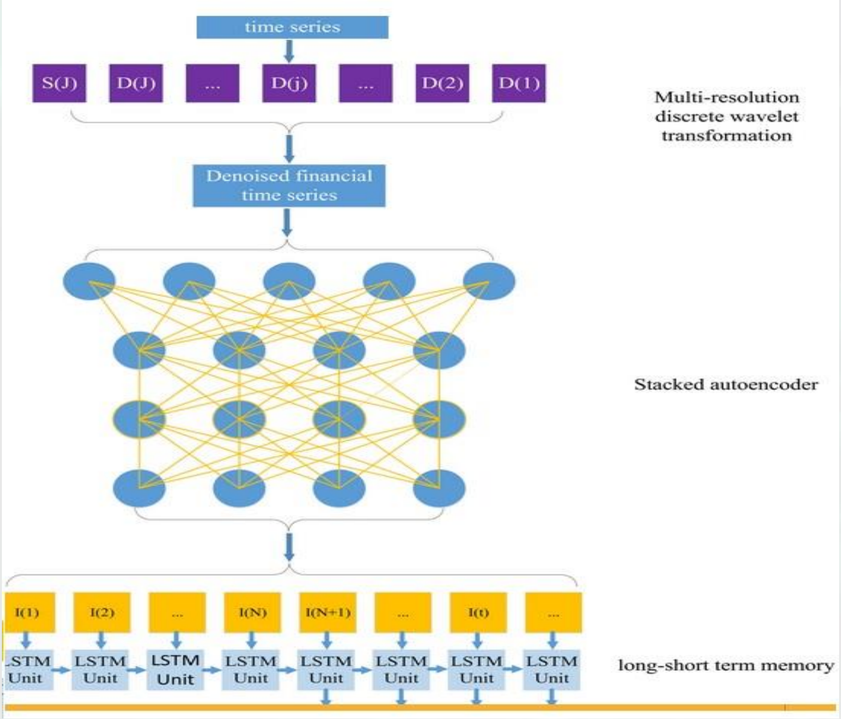
- The application of **Deep Learning** approaches to time-series prediction has received a great deal of attention from both entrepreneurs and researchers. Results show that deep learning models outperform other statistical models in predictive accuracy **(Bao, et al., 2017)**.

$$X_t - \alpha_1 X_{t-1} - \cdots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q},$$

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t$$

- The application of classic time series models, such as **Auto Regressive Integrated Moving Average (ARIMA),** usually requires strict assumptions regarding the distributions and stationarity of time series. For complex, non-stationary and noisy time-series it is necessary for one to know the properties of the time series before the application of classic time series models **(Bodyanskiy and Popov, 2006).** Otherwise, the forecasting effort would be ineffective.

Advantages of Artificial Neural Networks (ANNs) in Time-Series Prediction

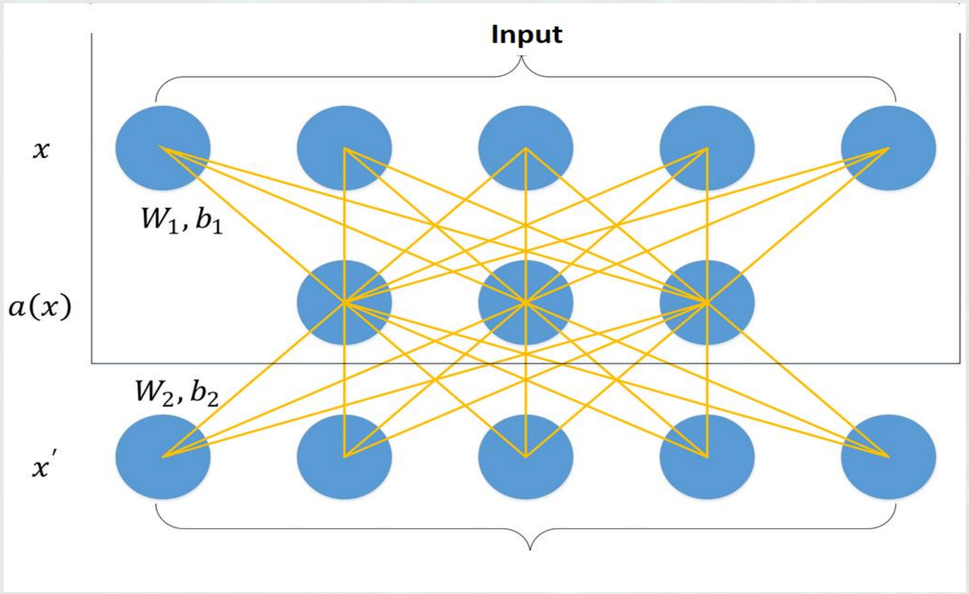


- However, by using ANNs, a priori analysis as ANNs do not require prior knowledge of the time series structure because of their black-box properties **(Nourani, et al., 2009).**

- Also, the impact of the stationarity of time series on the prediction power of ANNs is quite small. It is feasible to relax the stationarity condition to non-stationary time series when applying ANNs to predictions **(Kim, et al., 2004).**

- ANNs allow **multivariate time-series forecasting** whereas classical linear methods can be difficult to adapt to multivariate or multiple input forecasting problems.

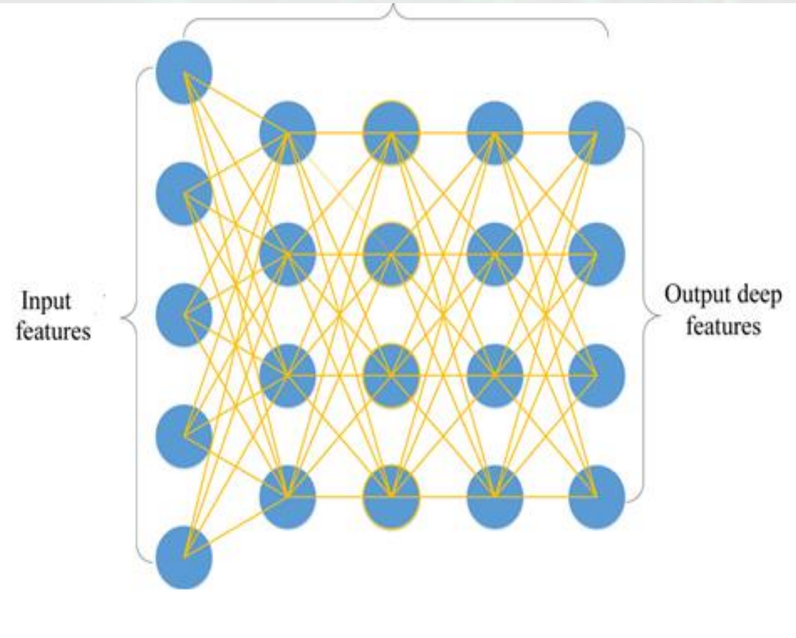
Stacked Auto-encoders (SAEs)



- According to recent studies, better approximation to nonlinear functions can be generated by stacked deep learning models than those models with a more shallow structure.
- A Single layer **Auto-Encoder (AE)** is a three-layer neural network. The first layer and the third layer are the input layer and the reconstruction layer with k units, respectively. The second layer is the hidden layer with n units, which is designed to generate the deep feature for this single layer AE.
- The aim of training the single layer AE is to minimize the error between thee input vector and the reconstruction vector by **gradient descent.**

$$a(x) = f(\mathbf{W}_1 x + b_1)$$
$$x' = f(\mathbf{W}_2 a(x) + b_2)$$

Stacked Auto-encoders (SAEs): 4 Auto-encoders



- Stacked auto-encoders (SAEs) are constructed by stacking a sequence of single-layer AEs layer by layer **(Bengio Y, et. Al. 2007).**
- After training the first single-layer auto-encoder, the reconstruction layer of the first single layer auto-encoder is removed (included weights and biases), and the **hidden layer** is reserved as the input layer of the second single-layer auto-encoder.
- **Depth** plays an important role in **SAE** because it determines qualities like invariance and abstraction of the extracted feature.
- **Wavelet Transform (WT)** can be applied as input to SAEs to handle data particulary non-stationary **(Ramsey, (1999) .**

Metrics

- There exist several indicators to measure the predictive accuracy of each model **(Hsieh, et. al., 2011; Theil, 1973)**
- **RMSE (Root Mean Square Error):** Represents the sample standard deviation of the differences between predicted values and observed values.

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{n}}$$

- **MAPE (Mean Absolute Percentage Error):** Measures the size of the error in percentage terms. Most people are comfortable thinking in percentage terms, making the MAPE easy to interpret.

$$MAPE = \frac{\sum_{t=1}^{N}\left|\frac{y_t - y_t^*}{y_t}\right|}{N}$$

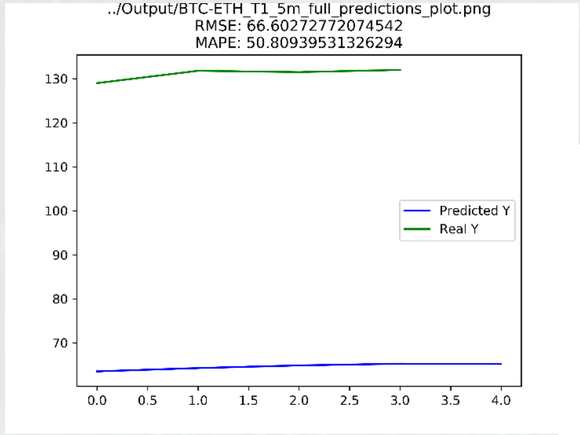
- **Theil U:** Theil U is a relative measure of the difference between two variables. It squares the deviations to give more weight to large errors and to exaggerate errors.

$$Theil\ U = \frac{\sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - y_t^*)^2}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t)^2} + \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t^*)^2}}$$

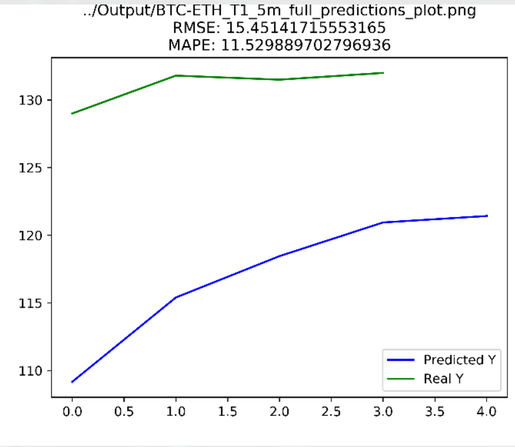
- In these equations, yt is the actual value and is the predicted value.

Italian GDP (Gross Domestic Production) Time Series Prediction – Yearly - 1980 – 2016 (Istat)

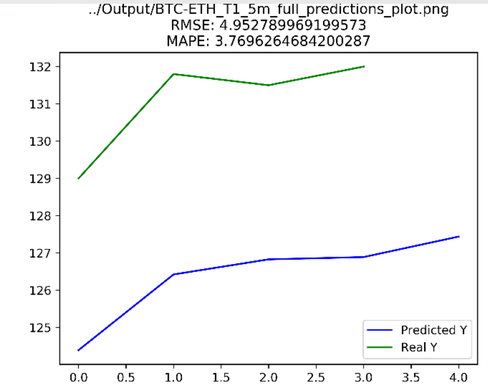
Test Set : 10%



1 Epoch



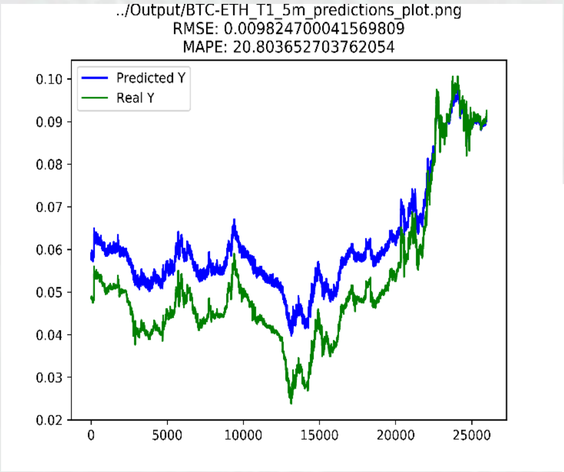
100 Epochs



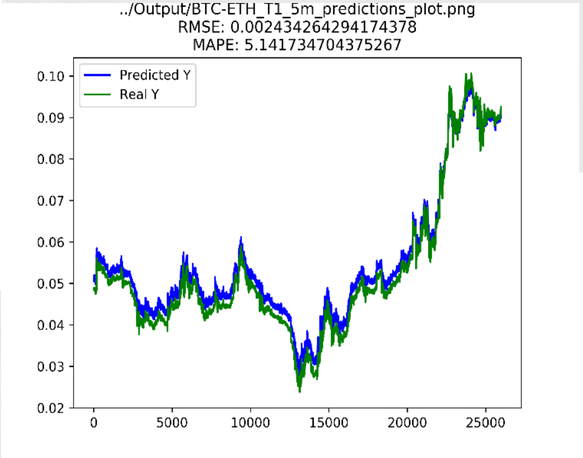
1000 Epochs

Bitcoin BTC-ETH exchange Time Series Prediction – 5 mins (Poloniex)

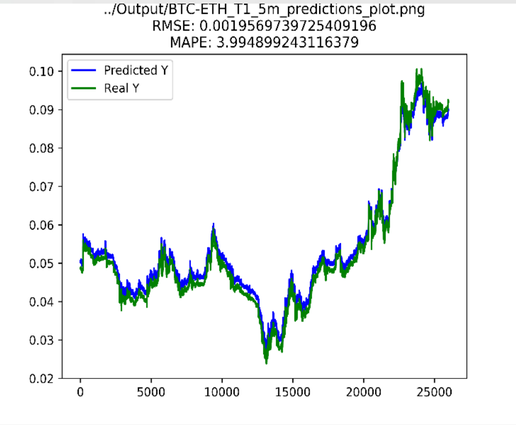
Test Set : 10%



1 Epoch



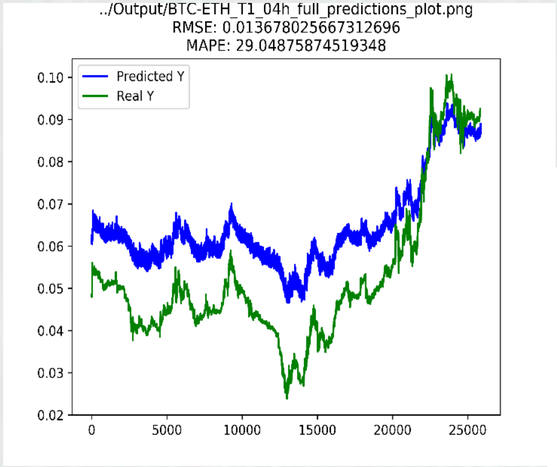
100 Epochs



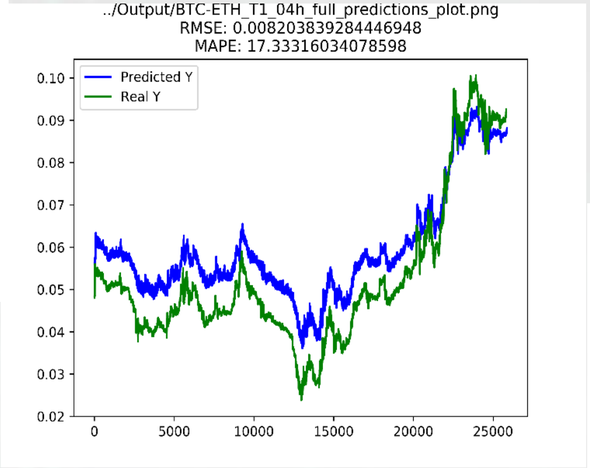
1000 Epochs

Bitcoin BTC-ETH exchange Time Series Prediction – 4 hours (Poloniex)

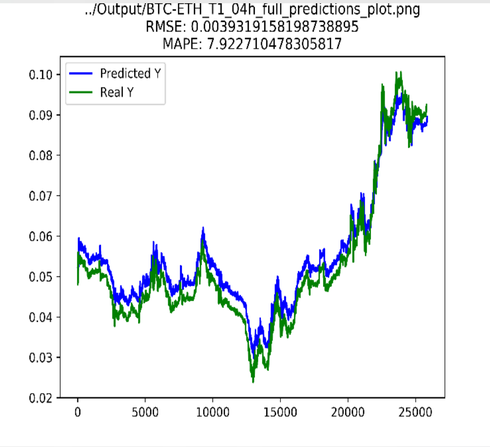
Test Set : 10%



1 Epoch

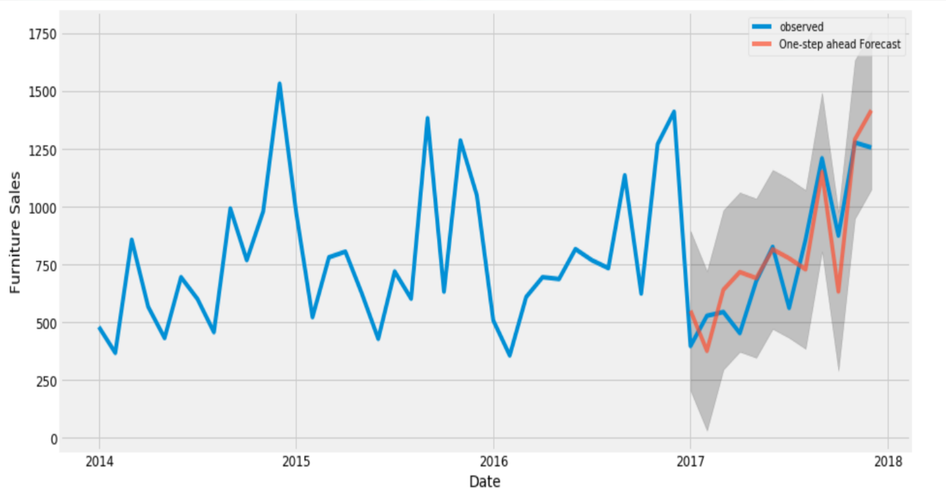


100 Epochs



1000 Epochs

Time Series Forecasting Exercise : Multivariate Time-Series Forecasting in Keras



1. Air Pollution Forecasting

In this tutorial, we are going to use the Air Quality dataset.

This is a dataset that reports on the weather and the level of pollution each hour for five years at the US embassy in Beijing, China.

The data includes the date-time, the pollution called PM2.5 concentration, and the weather information including dew point, temperature, pressure, wind direction, wind speed and the cumulative number of hours of snow and rain. The complete feature list in the raw data is as follows:

1. **No**: row number
2. **year**: year of data in this row
3. **month**: month of data in this row
4. **day**: day of data in this row
5. **hour**: hour of data in this row
6. **pm2.5**: PM2.5 concentration
7. **DEWP**: Dew Point
8. **TEMP**: Temperature
9. **PRES**: Pressure
10. **cbwd**: Combined wind direction
11. **Iws**: Cumulated wind speed
12. **Is**: Cumulated hours of snow
13. **Ir**: Cumulated hours of rain

https://raw.githubusercontent.com/jbrownlee/Datasets/master/pollution.csv

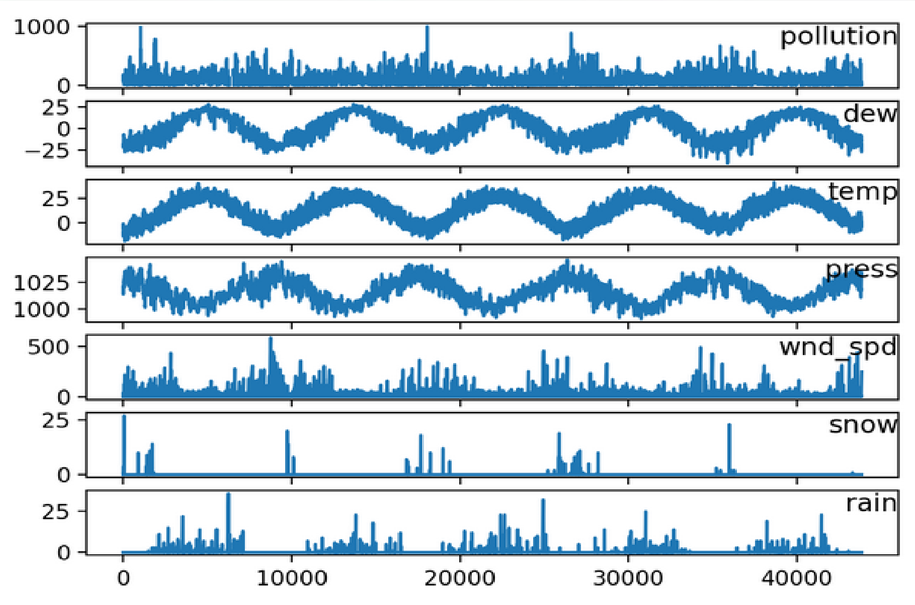
```
No,year,month,day,hour,pm2.5,DEWP,TEMP,PRES,cbwd,Iws,Is,Ir
1,2010,1,1,0,NA,-21,-11,1021,NW,1.79,0,0
2,2010,1,1,1,NA,-21,-12,1020,NW,4.92,0,0
3,2010,1,1,2,NA,-21,-11,1019,NW,6.71,0,0
4,2010,1,1,3,NA,-21,-14,1019,NW,9.84,0,0
5,2010,1,1,4,NA,-20,-12,1018,NW,12.97,0,0
6,2010,1,1,5,NA,-19,-10,1017,NW,16.1,0,0
7,2010,1,1,6,NA,-19,-9,1017,NW,19.23,0,0
8,2010,1,1,7,NA,-19,-9,1017,NW,21.02,0,0
9,2010,1,1,8,NA,-19,-9,1017,NW,24.15,0,0
10,2010,1,1,9,NA,-20,-8,1017,NW,27.28,0,0
11,2010,1,1,10,NA,-19,-7,1017,NW,31.3,0,0
```

Time-Series: A pollution predictor with Keras – PM2.5 concentration

```python
def pollution_prepare_data(pred_dataset_path='', pred_dataset_file_in='', pred_dataset_file_out=''):
    # load data
    def parse(x):
        return datetime.strptime(x, '%Y %m %d %H')
    dataset = read_csv(pred_dataset_path+'/'+pred_dataset_file_in,  parse_dates = [['year', 'month', 'day', 'hour']], index_col=0, date_parser=parse)
    dataset.drop('No', axis=1, inplace=True)
    # manually specify column names
    dataset.columns = ['pollution', 'dew', 'temp', 'press', 'wnd_dir', 'wnd_spd', 'snow', 'rain']
    dataset.index.name = 'date'
    # mark all NA values with 0
    dataset['pollution'].fillna(0, inplace=True)
    # drop the first 24 hours
    dataset = dataset[24:]
    # summarize first 5 rows
    print(dataset.head(5))
    # save to file
    dataset.to_csv(pred_dataset_path+'/'+pred_dataset_file_out)
```

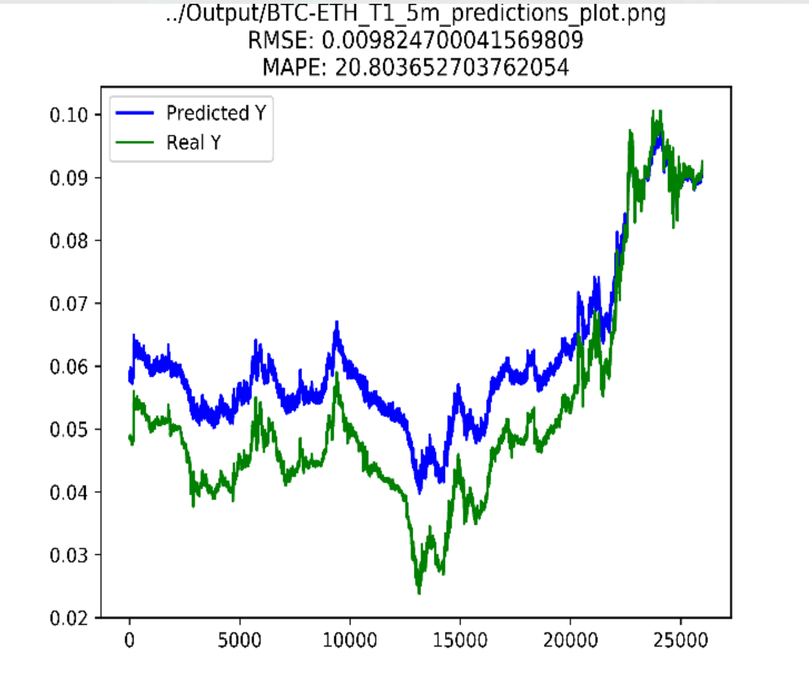
	pollution	dew	temp	press	wnd_dir	wnd_spd	snow	rain
date								
2010-01-02 00:00:00	129.0	-16	-4.0	1020.0	SE	1.79	0	0
2010-01-02 01:00:00	148.0	-15	-4.0	1020.0	SE	2.68	0	0
2010-01-02 02:00:00	159.0	-11	-5.0	1021.0	SE	3.57	0	0
2010-01-02 03:00:00	181.0	-7	-5.0	1022.0	SE	5.36	1	0
2010-01-02 04:00:00	138.0	-7	-5.0	1022.0	SE	6.25	2	0

Time-Series: View Model




```python
# Plot inputs
def plot_dataset(pred_dataset_path='', pred_dataset_file=''):
    # load dataset
    dataset = read_csv(pred_dataset_path+'/'+pred_dataset_file, header=0, index_col=0)
    values = dataset.values
    # specify columns to plot
    groups = [0, 1, 2, 3, 5, 6, 7]
    i = 1
    # plot each column
    pyplot.figure()
    for group in groups:
        pyplot.subplot(len(groups), 1, i)
        pyplot.plot(values[:, group])
        pyplot.title(dataset.columns[group], y=0.5, loc='right')
        i += 1
    pyplot.show(block=False)
    pyplot.pause(1)
```

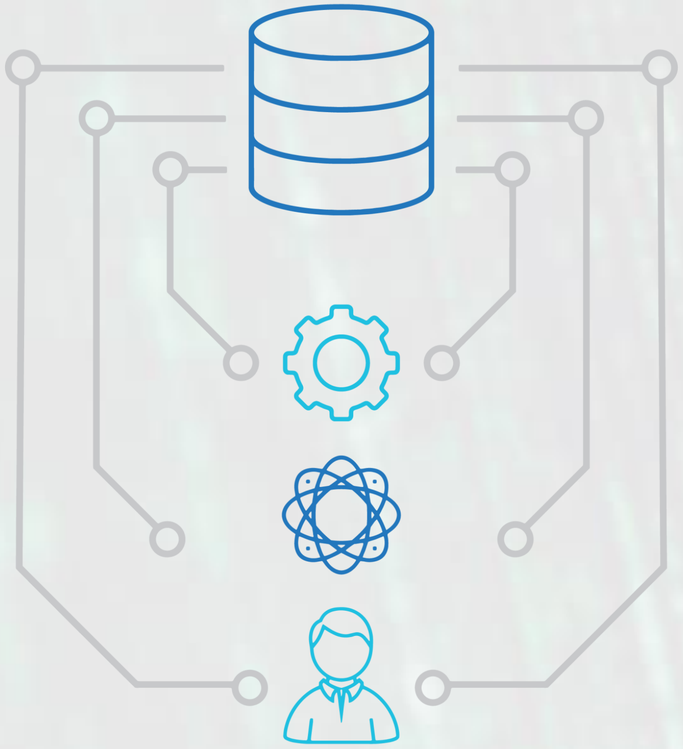
Time-Series: View Model




```python
# Plot the loss function chart
def plot_loss(history = None):
    pyplot.figure()                                          # generate a new window
    pyplot.plot(history['loss'], label='train')
    pyplot.plot(history['val_loss'], label='test')
    pyplot.legend()
    pyplot.show(block=False)
    pyplot.pause(1)


# Plot predictions and real y chart
def plot_predictions(predictions_view_path_file, rmse, mape, inv_yhat = None, inv_y = None, save = False, pause_time = 1):
    pyplot.figure()                                          # generate a new window
    pyplot.plot(inv_yhat, label='Predicted Y', color = 'blue')
    pyplot.plot(inv_y, label='Real Y', color = 'green')
    pyplot.legend()
    pyplot.title(predictions_view_path_file+"\nRMSE: "+str(rmse)+'\nMAPE: '+str(mape))
    pyplot.savefig(predictions_view_path_file, dpi = 600)
    pyplot.show(block=False)
    pyplot.pause(pause_time)
```

Data Preparation : From Time-Series a Supervised Problem



```python
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

Data Preparation : From Time-Series a Supervised Problem

```
0
1
2
3
4
5
6
7
8
9
```

features | label

example	i	$x^{(1)}$	$x^{(2)}$	$x^{(3)}$	...	$x^{(j)}$	y
	1	75	1.77	25	...	...	yes
	2	68	1.65	32	...	...	X
	3	57	1.60	27	...	...	no
	4	84	1.90	41	...	...	X
	...	...	...	...	...	...	...
	N	...	...	...	...	...	...

OKPEDIA

```
X, y
1  2
2, 3
3, 4
4, 5
5, 6
6, 7
7, 8
8, 9
```

Data Preparation : Pandas shift function


```
1 from pandas import DataFrame
2 df = DataFrame()
3 df['t'] = [x for x in range(10)]
4 df['t-1'] = df['t'].shift(1)
5 print(df)
```

```
1     t  t-1
2 0   0  NaN
3 1   1  0.0
4 2   2  1.0
5 3   3  2.0
6 4   4  3.0
7 5   5  4.0
8 6   6  5.0
9 7   7  6.0
10 8  8  7.0
11 9  9  8.0
```

Data Preparation : Multi-Step Univariate Analysis

```
1 data = series_to_supervised(values, 3)
```

	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
3	0.0	1.0	2.0	3
4	1.0	2.0	3.0	4
5	2.0	3.0	4.0	5
6	3.0	4.0	5.0	6
7	4.0	5.0	6.0	7
8	5.0	6.0	7.0	8
9	6.0	7.0	8.0	9

Data Preparation : Multi-Step Univariate Analysis

```
data = series_to_supervised(values, 3)
```

	var1(t-3)	var1(t-2)	var1(t-1)	var1(t)
3	0.0	1.0	2.0	3
4	1.0	2.0	3.0	4
5	2.0	3.0	4.0	5
6	3.0	4.0	5.0	6
7	4.0	5.0	6.0	7
8	5.0	6.0	7.0	8
9	6.0	7.0	8.0	9

Data Preparation : Multi-Step or Sequence Forecasting

```
data = series_to_supervised(values, 2, 2)
```

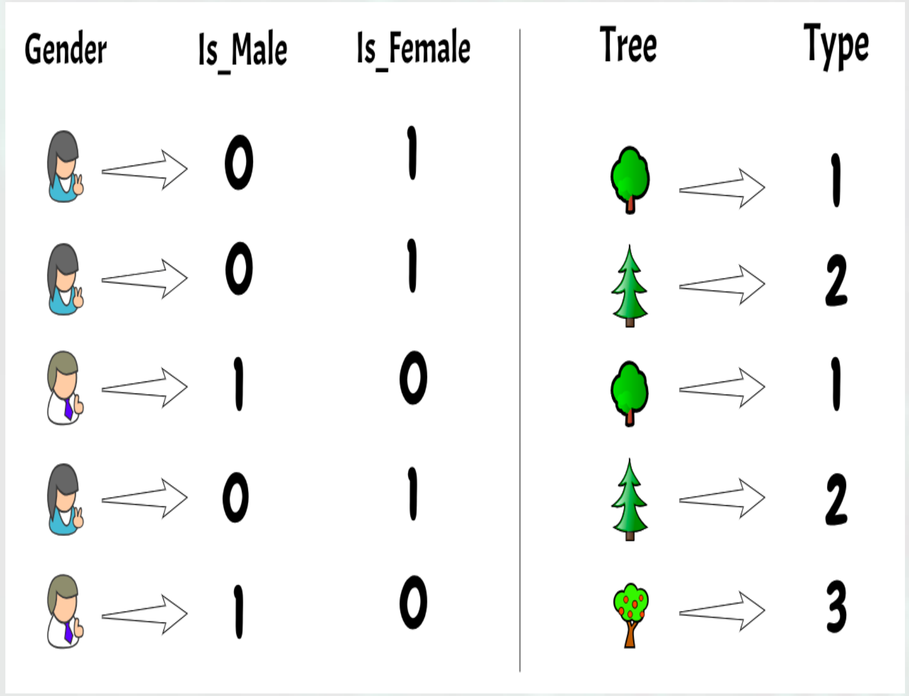
	var1(t-2)	var1(t-1)	var1(t)	var1(t+1)
2	0.0	1.0	2	3.0
3	1.0	2.0	3	4.0
4	2.0	3.0	4	5.0
5	3.0	4.0	5	6.0
6	4.0	5.0	6	7.0
7	5.0	6.0	7	8.0
8	6.0	7.0	8	9.0

Data Preparation : Multivariate Forecasting

		var1(t-1)	var2(t-1)	var1(t)	var2(t)
1	1	0.0	50.0	1	51
2	2	1.0	51.0	2	52
3	3	2.0	52.0	3	53
4	4	3.0	53.0	4	54
5	5	4.0	54.0	5	55
6	6	5.0	55.0	6	56
7	7	6.0	56.0	7	57
8	8	7.0	57.0	8	58
9	9	8.0	58.0	9	59

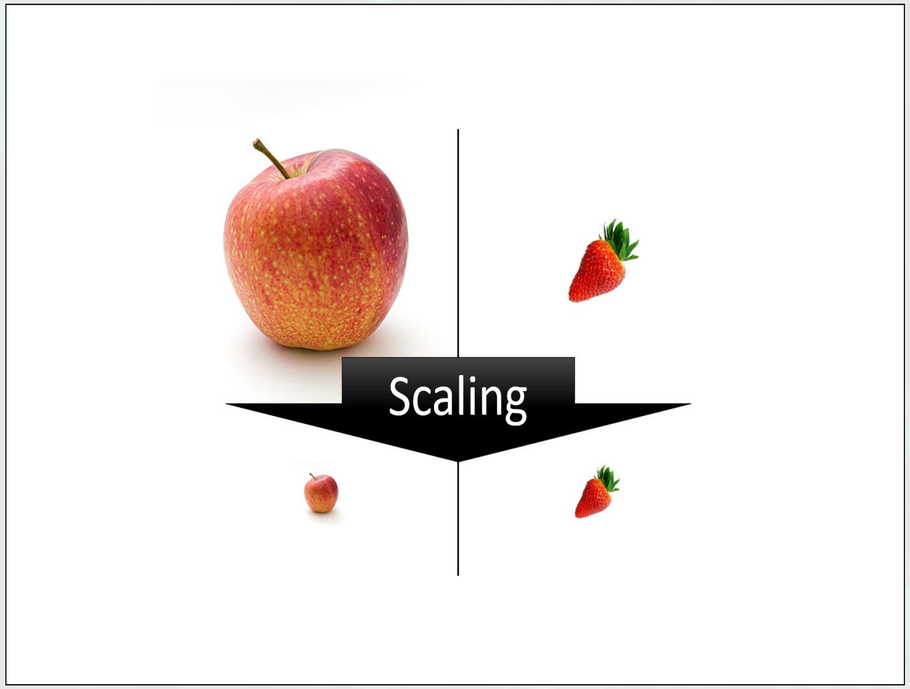
		var1(t-1)	var2(t-1)	var1(t)	var2(t)	var1(t+1)	var2(t+1)
1	1	0.0	50.0	1	51	2.0	52.0
2	2	1.0	51.0	2	52	3.0	53.0
3	3	2.0	52.0	3	53	4.0	54.0
4	4	3.0	53.0	4	54	5.0	55.0
5	5	4.0	54.0	5	55	6.0	56.0
6	6	5.0	55.0	6	56	7.0	57.0
7	7	6.0	56.0	7	57	8.0	58.0
8	8	7.0	57.0	8	58	9.0	59.0

Data Preparation : Data Encoding and Normalization




```
# load dataset
dataset = read_csv('pollution.csv', header=0, index_col=0)
values = dataset.values
# integer encode direction
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
print(reframed.head())
```

Data Preparation : Data splitting and scaling

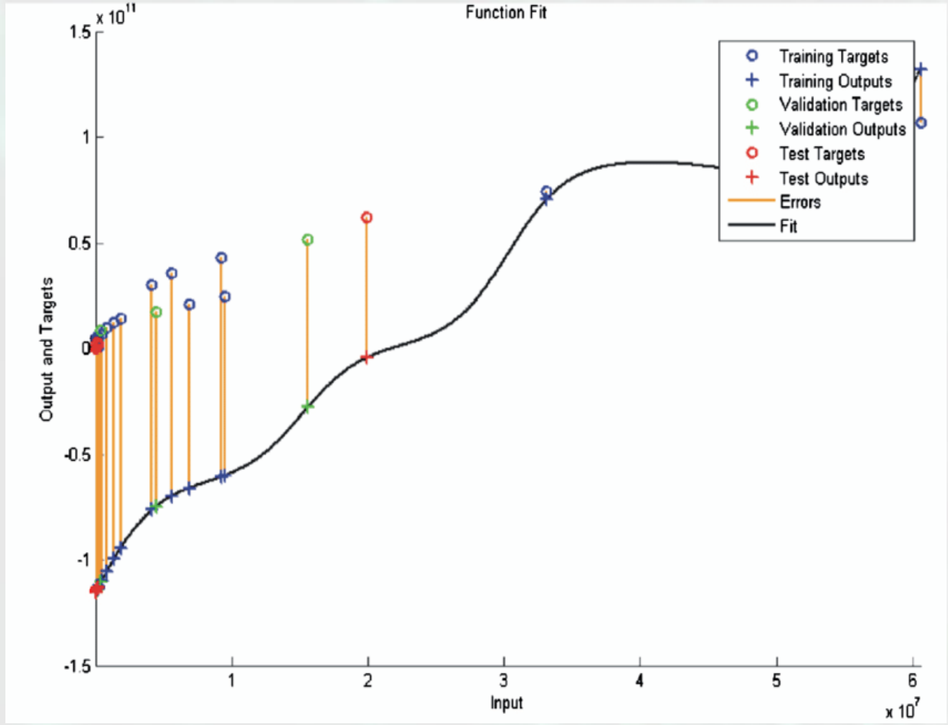



```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)  var6(t-1)  \
1   0.129779   0.352941   0.245902   0.527273   0.666667   0.002290
2   0.148893   0.367647   0.245902   0.527273   0.666667   0.003811
3   0.159960   0.426471   0.229508   0.545454   0.666667   0.005332
4   0.182093   0.485294   0.229508   0.563637   0.666667   0.008391
5   0.138833   0.485294   0.229508   0.563637   0.666667   0.009912

   var7(t-1)  var8(t-1)   var1(t)
1   0.000000        0.0  0.148893
2   0.000000        0.0  0.159960
3   0.000000        0.0  0.182093
4   0.037037        0.0  0.138833
5   0.074074        0.0  0.109658
```

```python
# split into train and test sets
values = reframed.values
n_train_hours = 365 * 24
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

Model Fitting and Prediction




```
# fit network
history = model.fit(train_X, train_y, epochs=50, batch_size=72, validation_data=(t
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()

# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, 1:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, 1:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

Natural Language Toolkit

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

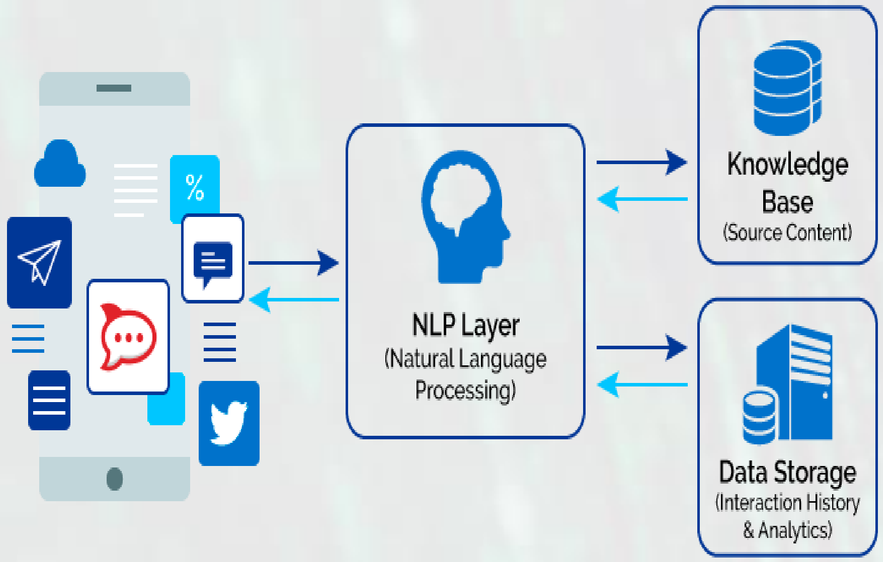
Natural Language Toolkit

Some simple things you can do with NLTK

Tokenize and tag some text:

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',
'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),
('Thursday', 'NNP'), ('morning', 'NN')]
```

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).

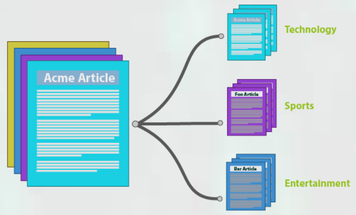
- **NLP** is everywhere because people communicate most everything: web search, advertisement, emails, customer service, etc.

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)

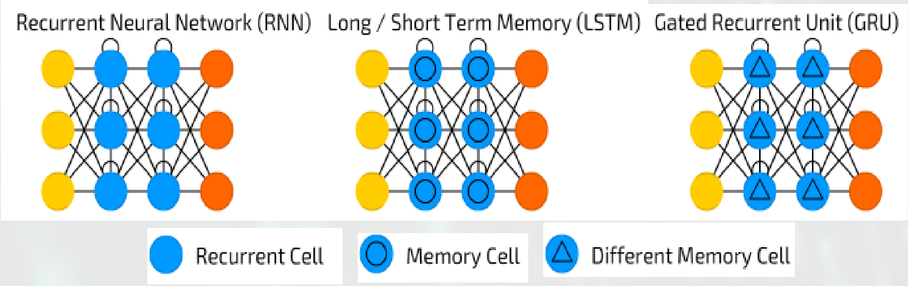
- **Natural language Processing** is shifting from statistical methods to **Neural Networks**.

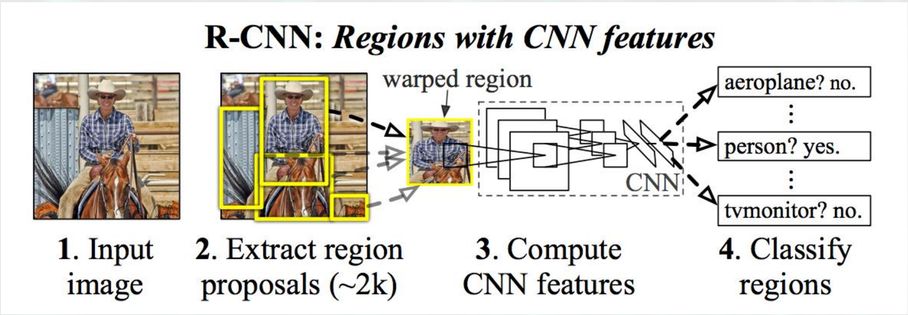
7 NLP applications where Deep Learning achieved «state-of-art» performance



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

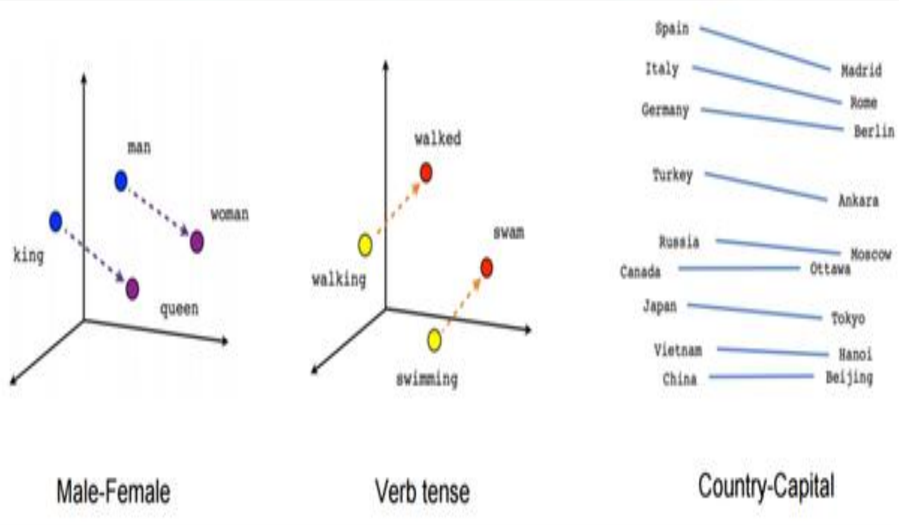
Text Classification Models





- **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.

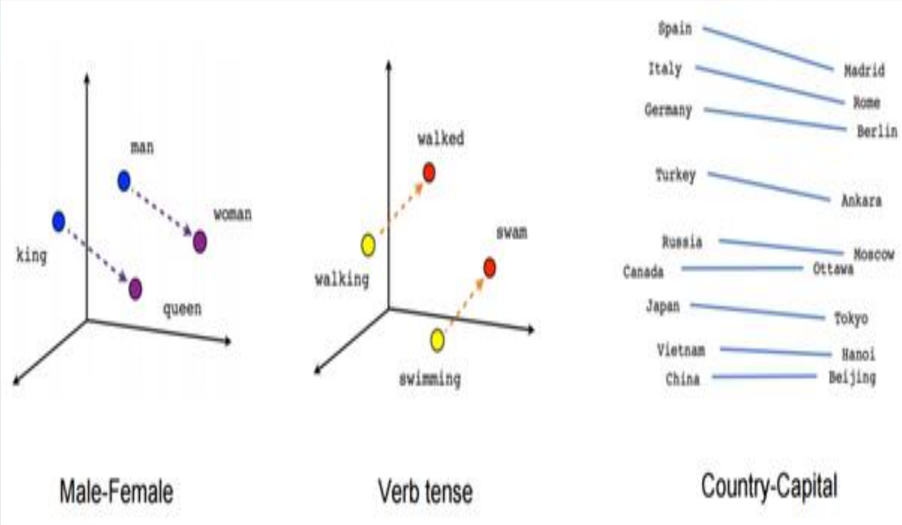
Word Embedding & Language Modeling

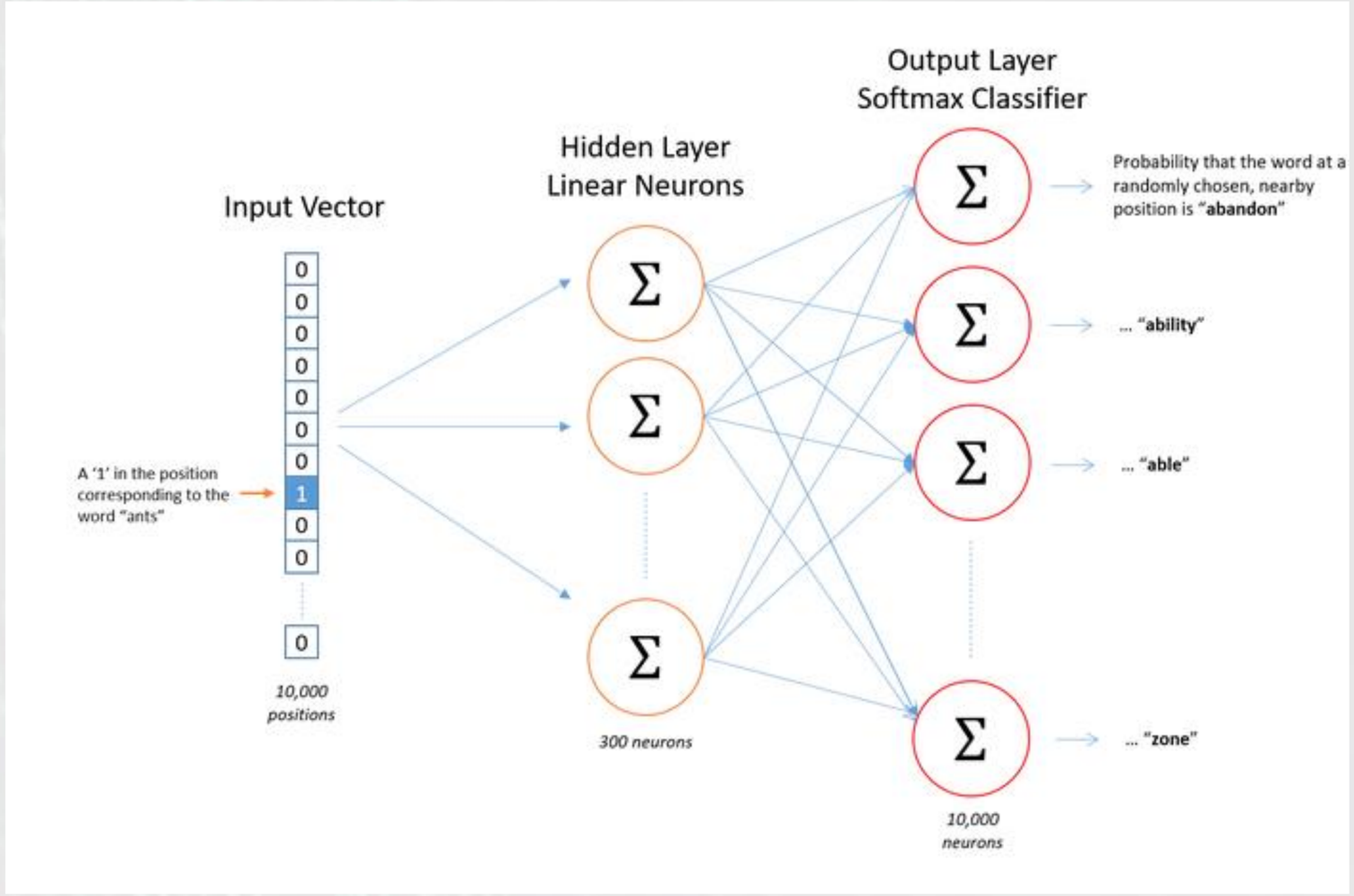


- **Word embedding** is the collective name for a set of language modeling and feature learning techniques for natural language processing (**NLP**) where words or sentences from the vocabulary are mapped to vectors of real numbers.

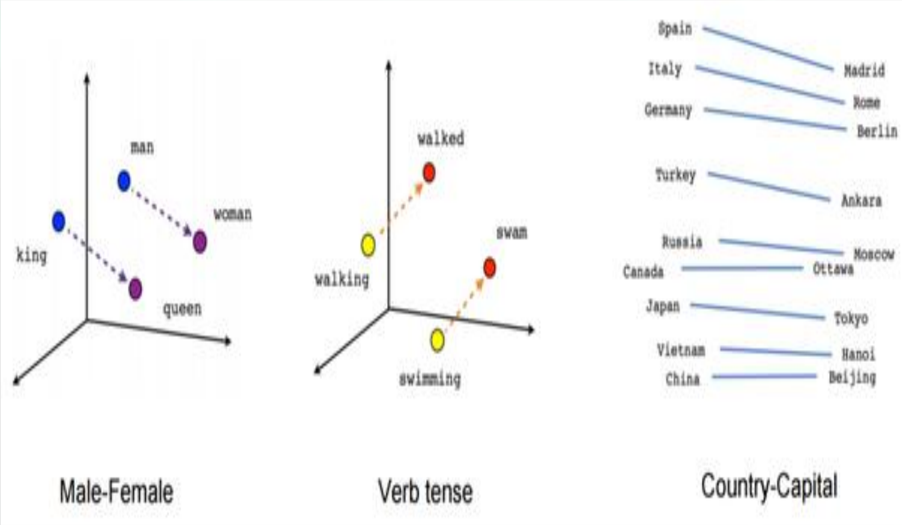
- These vectors are semantically correlated by metrics like **cosine distance.**

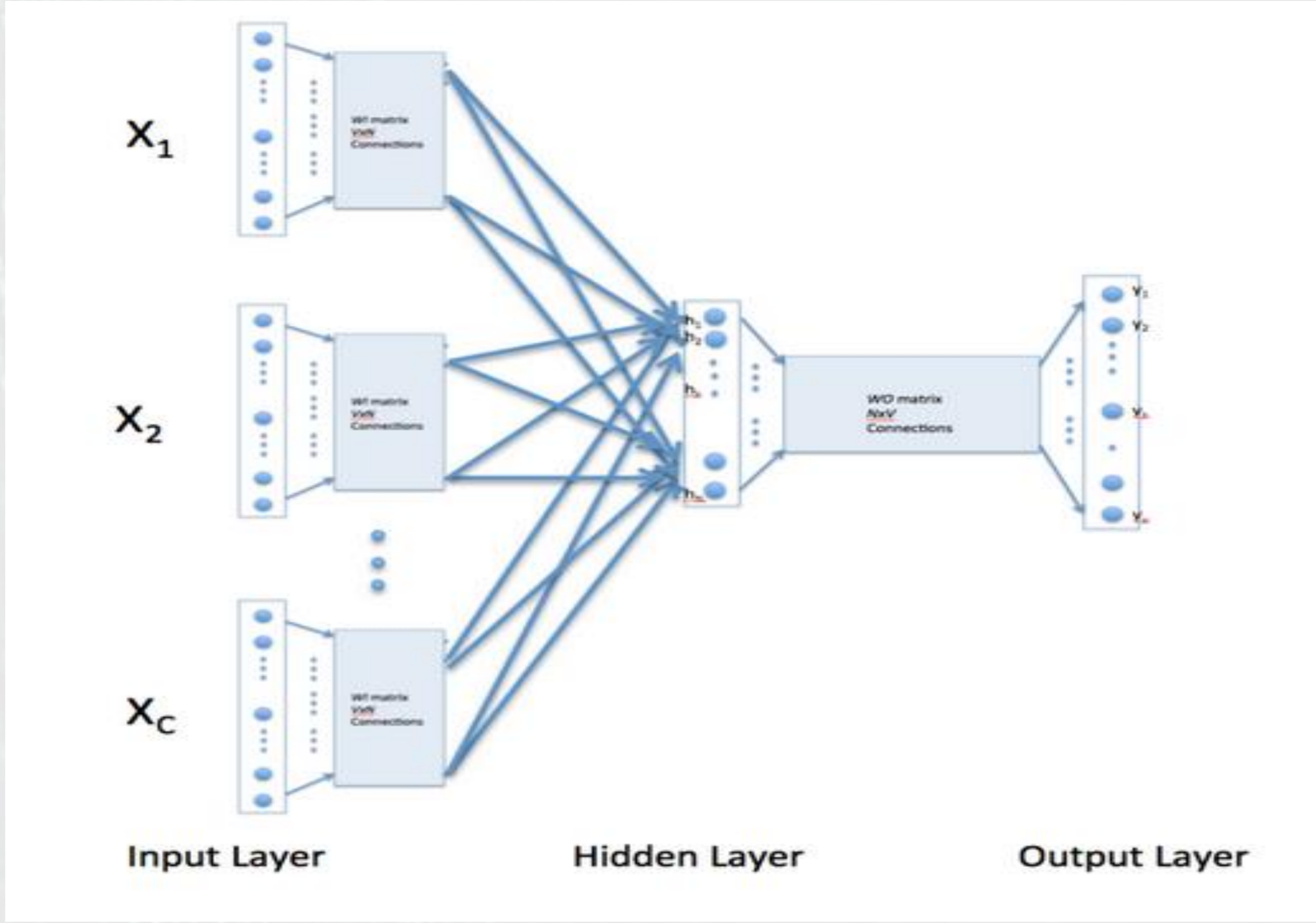
Skip-Gram Model (Mikolov, et. al., 2013)





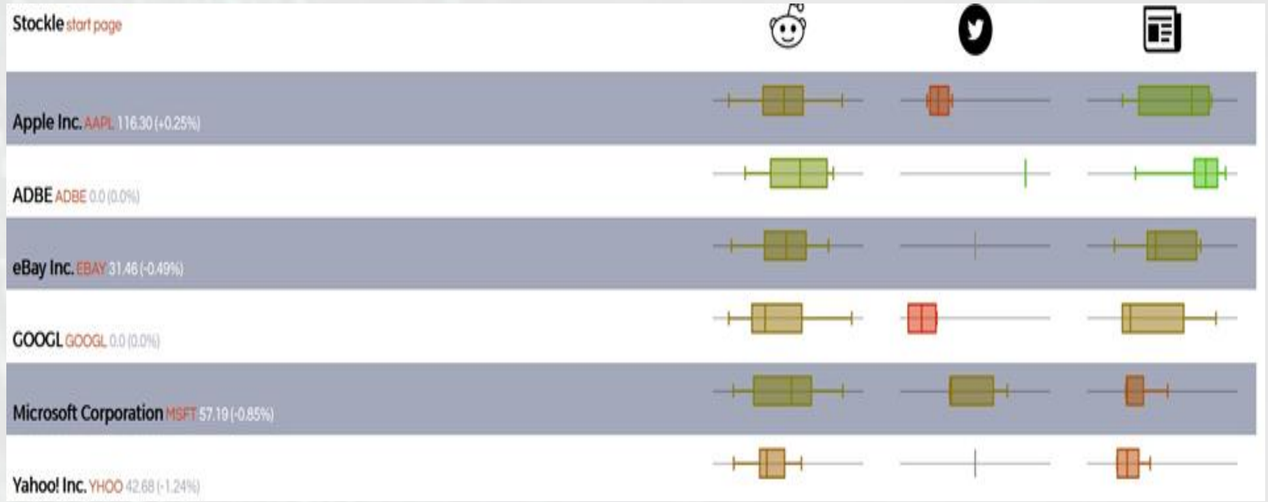
C-BOW Model (Bow, et al., 2003).



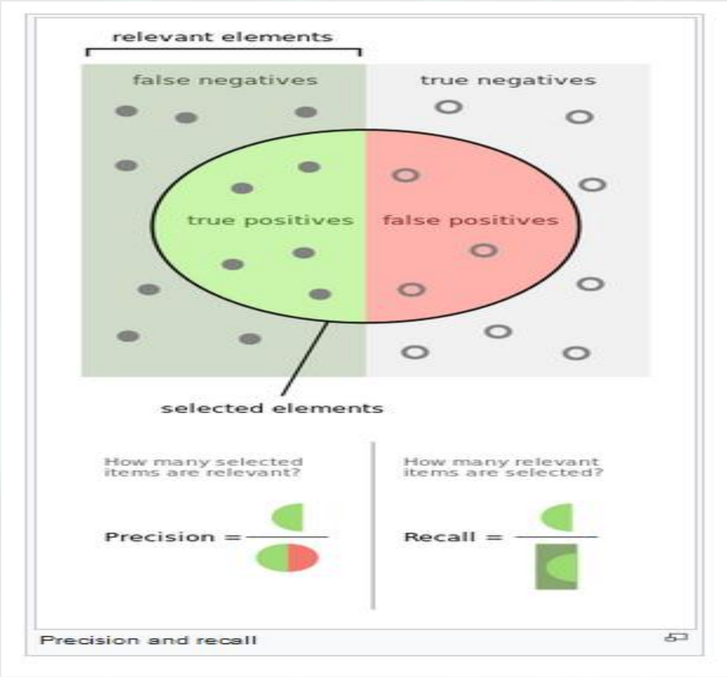


Sentiment Analysis (Ain, et al. 2017)

- **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.



Classification Metrics



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

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

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:

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.

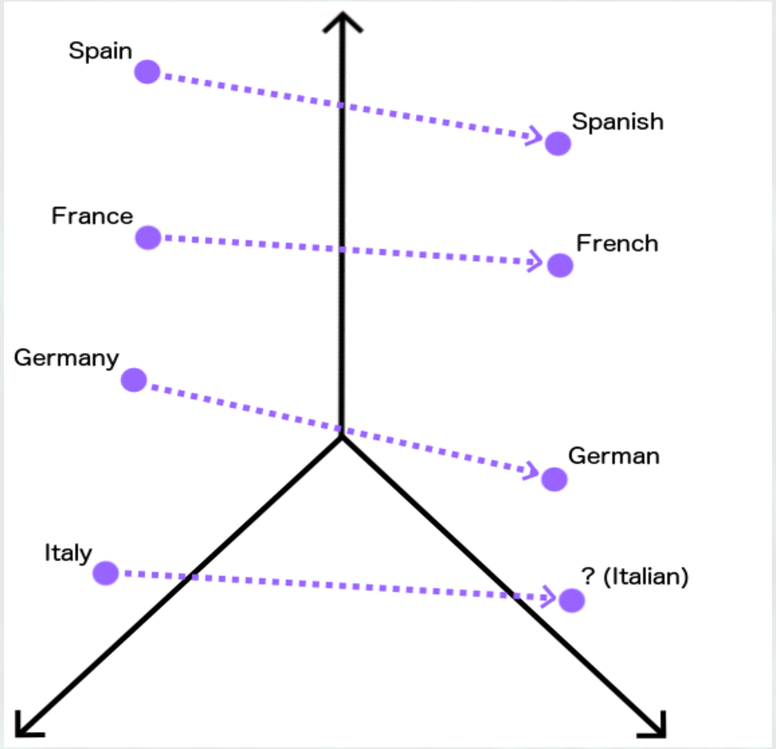
Parameters:

- **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.

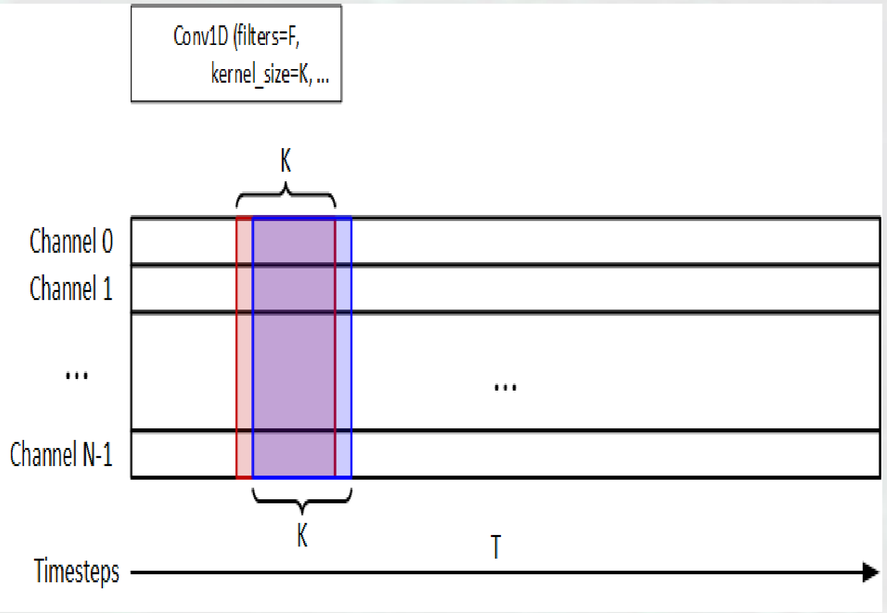
Example

```
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: 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).

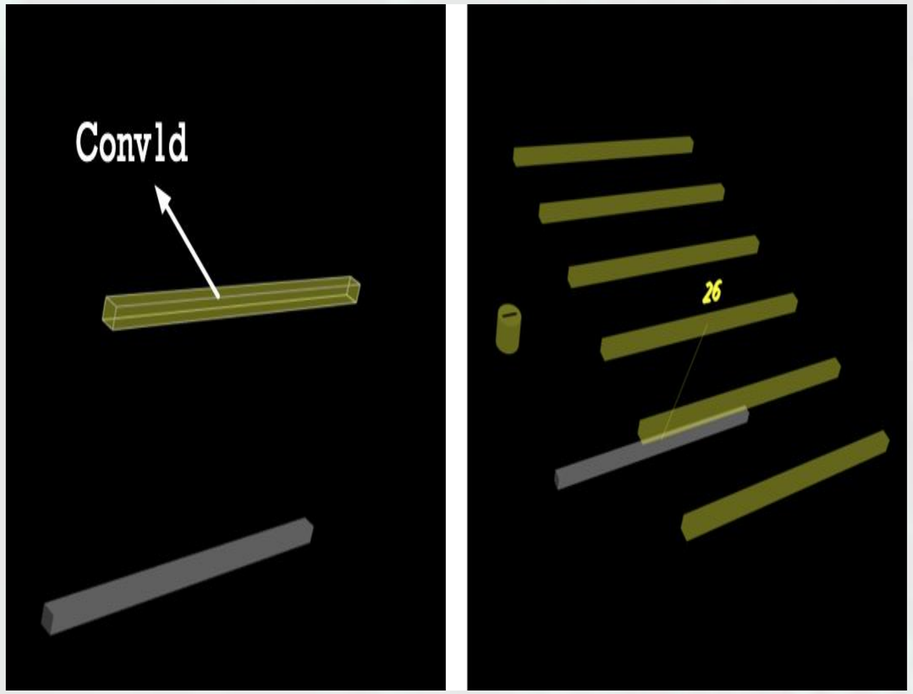
This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as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

Deep LSTM/GRU Architectures

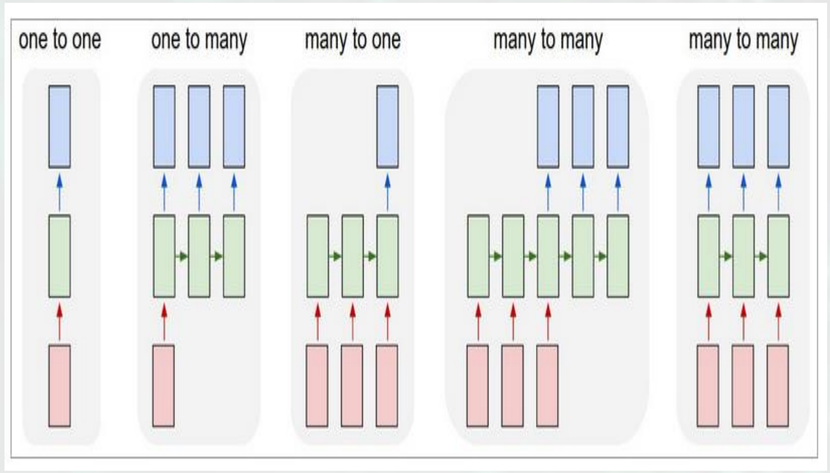
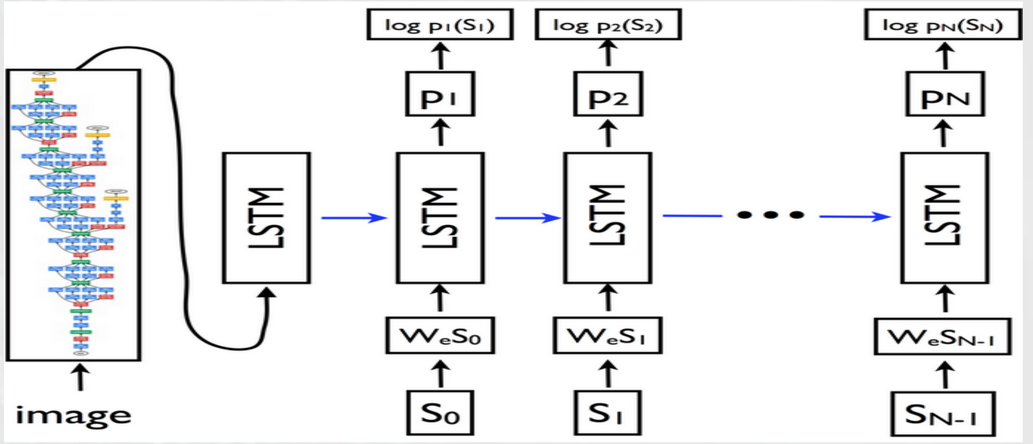
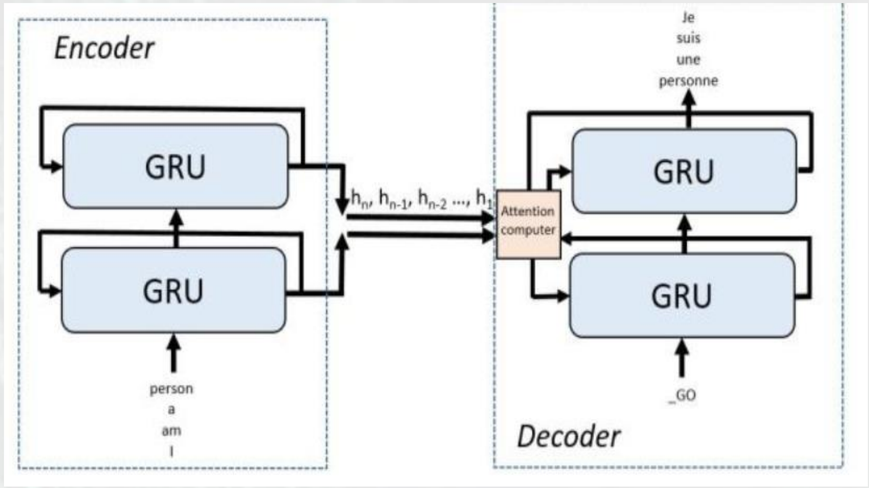


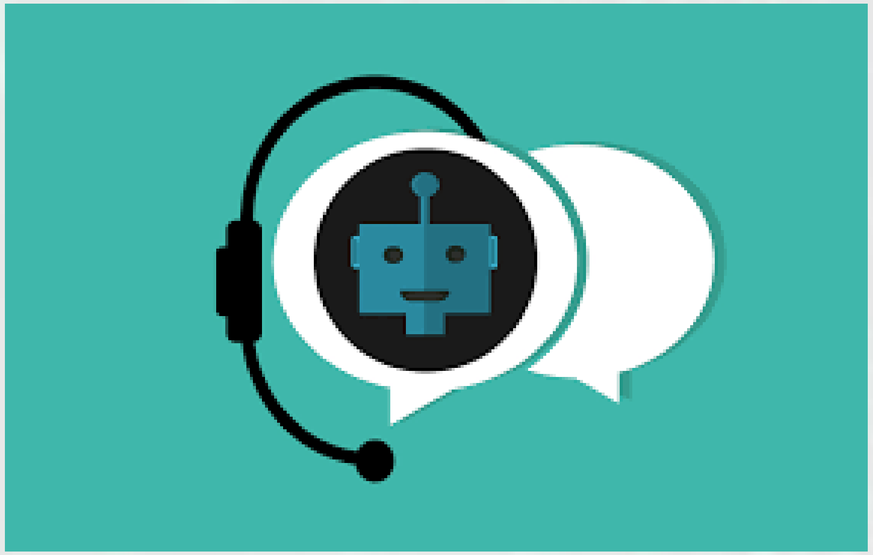
Image Caption Generator

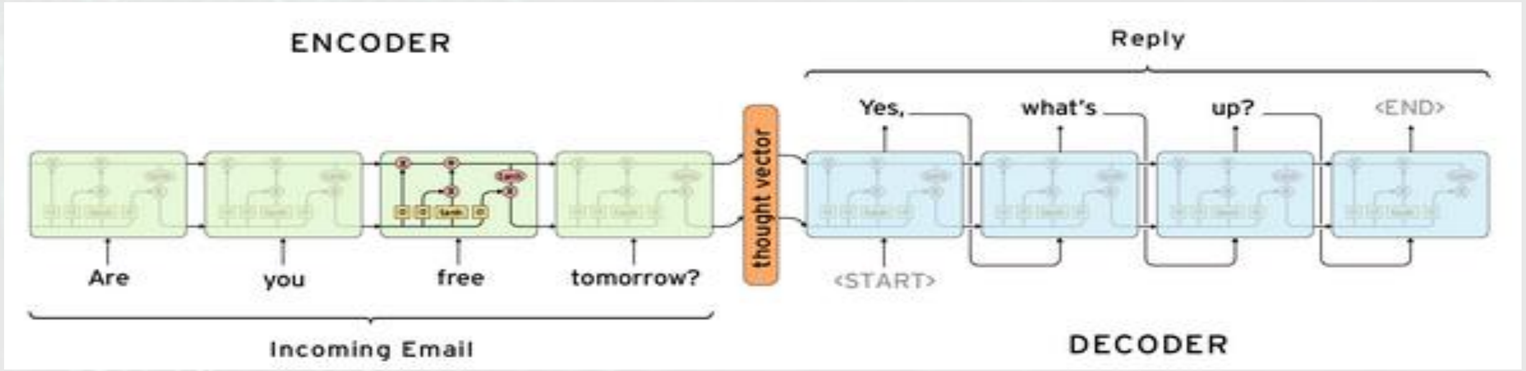


Seq2seq model



Neural Conversational Models (Vinyals, & Le., 2015)





Conversation model – chatbot?

- Training on a set of conversations. The input sequence can be the concatenation of what has been conversed so far (the context), and the output sequence is the reply.

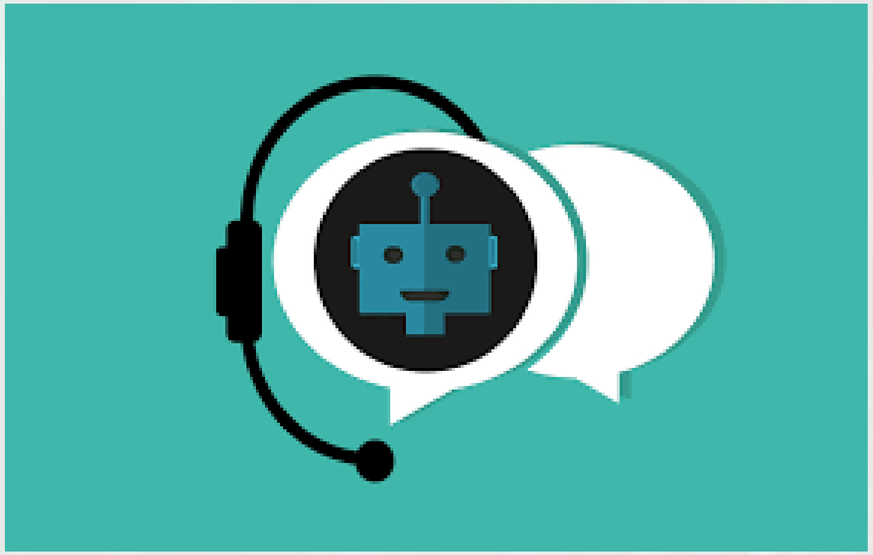
Conversation 4: Philosophical Q&A

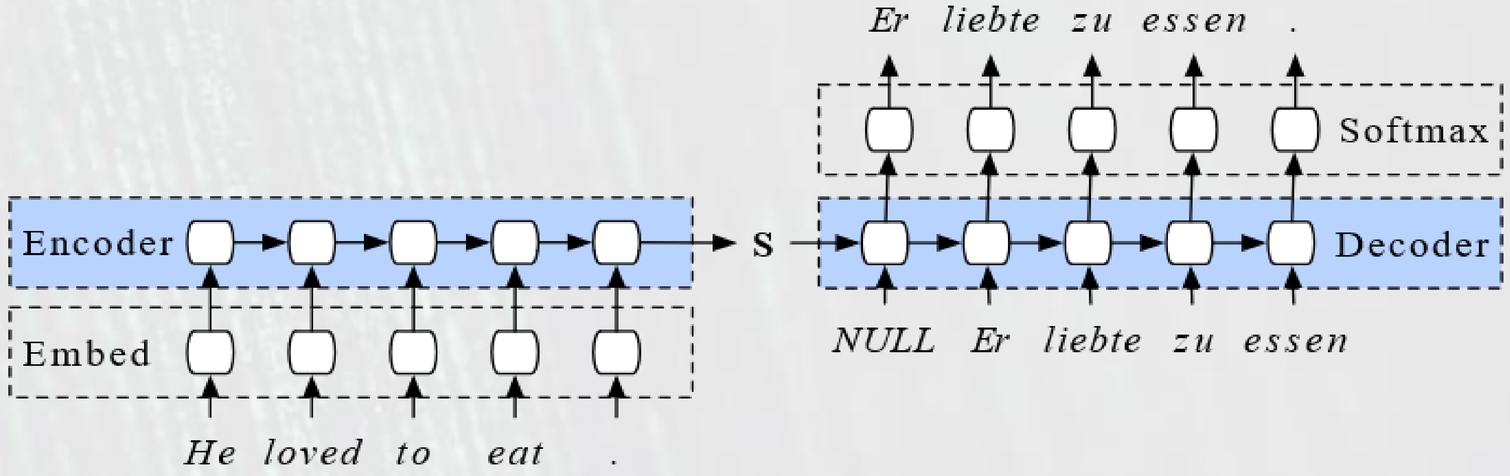
Machine: *is it a windows machine ?*
Machine: *we have linux on your machine .*
Machine: *is it windows xp or windows 7 ?*
Human: *i am using linux*
Machine: *let me take a remote of your machine and check if issue has been pushed properly .*
Human: *ok*
Machine: *please open terminal and run the below commands in one computer :*
Machine: *sudo apt - get update*
Machine: *sudo apt - get upgrade*
Machine: *sudo <COMMAND> - updater*
Human: *done*
Machine: *let me try connecting now*
Human: *ok*

Human: *what is the purpose of life ?*
Machine: *to serve the greater good .*
Human: *what is the purpose of living ?*
Machine: *to live forever .*
Human: *what is the purpose of existence ?*
Machine: *to find out what happens when we get to the planet earth .*
Human: *where are you now ?*
Machine: *i 'm in the middle of nowhere .*
Human: *what is the purpose of dying ?*
Machine: *to have a life .*
Human: *what is the purpose of being intelligent ?*
Machine: *to find out what it is .*
Human: *what is the purpose of emotions ?*
Machine: *i don 't know .*

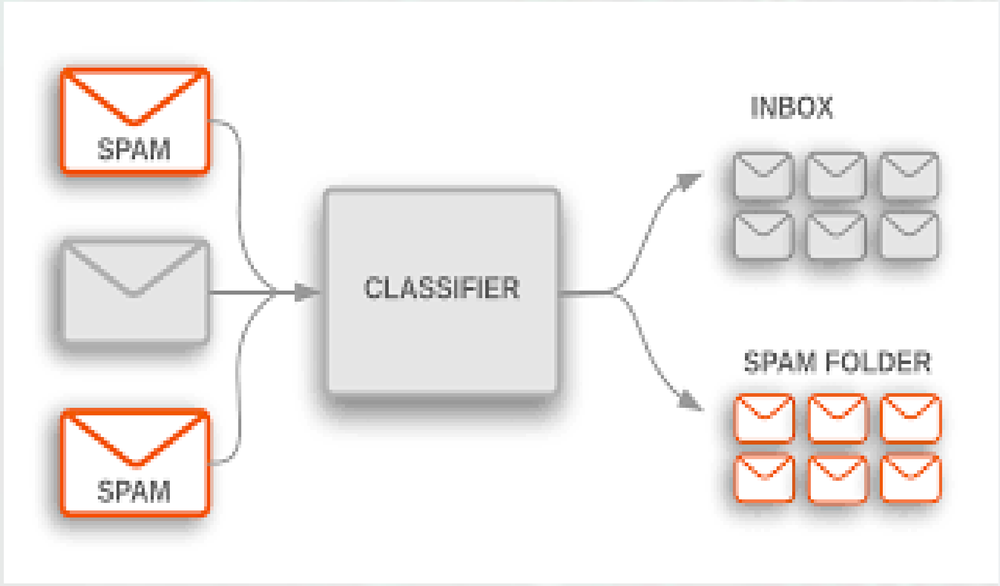
Neural Machine Translation

Input sentence:	*Translation (PBMT):*	*Translation (GNMT):*	*Translation (human):*
李克強此行將啟動中加總理年度對話機制，與加拿大總理杜魯多舉行兩國總理首次年度對話。	Li Keqiang premier added this line to start the annual dialogue mechanism with the Canadian Prime Minister Trudeau two prime ministers held its first annual session.	Li Keqiang will start the annual dialogue mechanism with Prime Minister Trudeau of Canada and hold the first annual dialogue between the two premiers.	Li Keqiang will initiate the annual dialogue mechanism between premiers of China and Canada during this visit, and hold the first annual dialogue with Premier Trudeau of Canada.

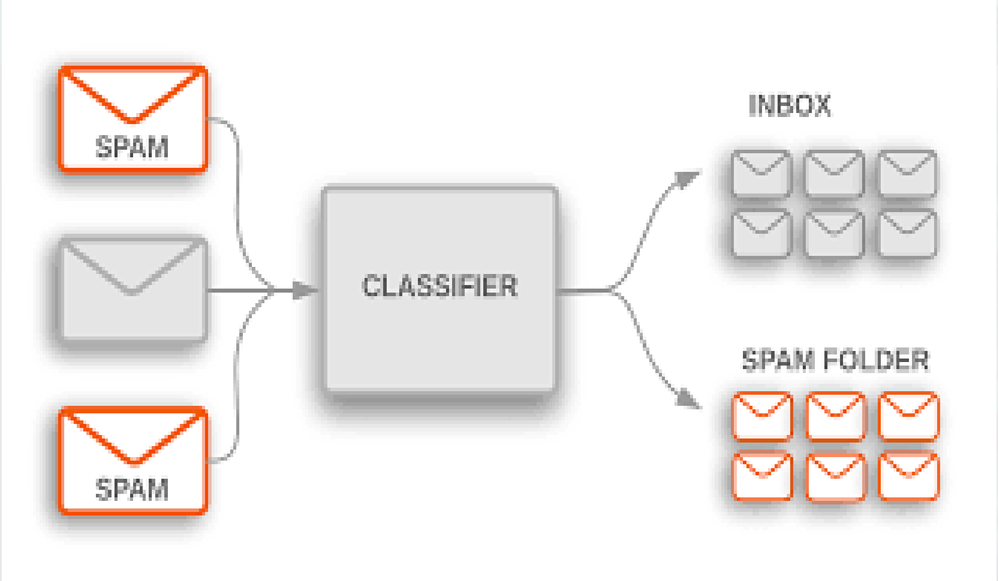




Exercise: Text Classifier in Python/Keras



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).

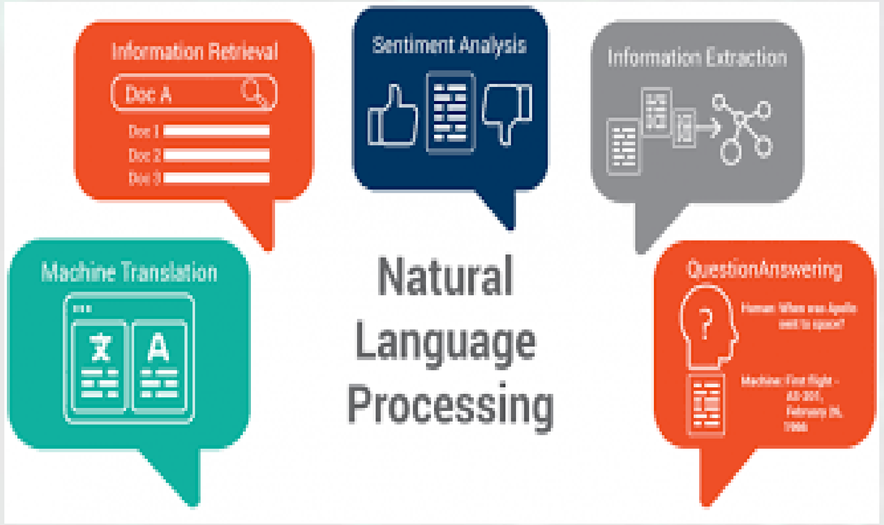
20 Newsgroup dataset

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.

44

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

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))
```

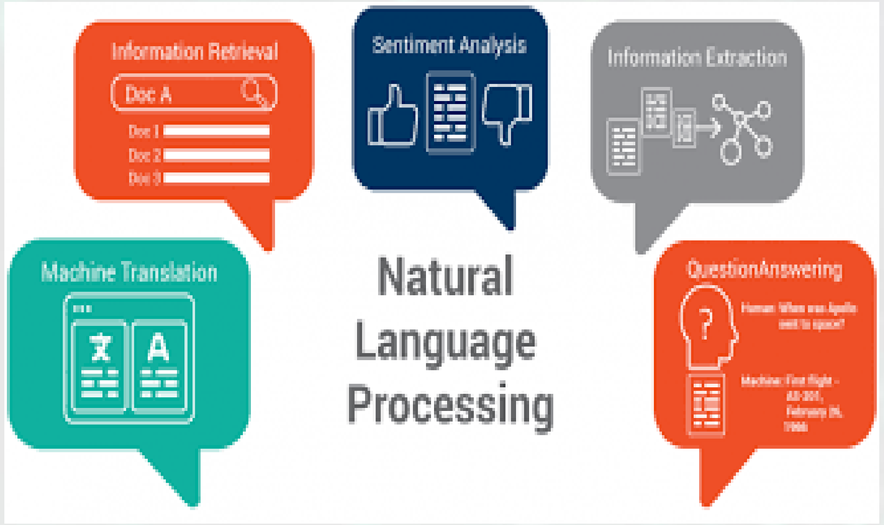
ARCHIVED: What is the Latin-1 (ISO-8859-1) character set?

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

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))
```

Building of the Word Index

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

Sequence 1D 0-Padding

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

To categorical vari ables conversion

```
# 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:]
```

Shuffle of Data an d Labels

Data and Labels S plitting

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).

DEEP LEARNING LESSONS

Deep Learning for Natural Language Processing (NLP)

Francesco Pugliese, PhD
neural1977@gmail.com

Thank You