Deep Learning

for Natural Language Processing(NLP)

Francesco Pugliese

DIRM-DCME-MEC
frpuglie@istat.it

Istat

Programma del Corso del 4 Dicembre 2019

4/12 – Deep Learning per Natural Language Processing (NLP):
Mattina - Teoria:

- Reti Neurali Ricorrenti (RNN): Elman RNN, Rated Recurrent Unit (RRU), Gated Recurrent Unit (GRU), Long Short Term Memories (LSTM)
- Predizione di Serie Storiche, Back Proagation Trough Time (BPTT)
- Modelli ricorrenti Seq2seq per chatbots e neural machine translation

Pomeriggio – Seminario & Pratica:

- Diego Zartetto: Seminario su Estrazione di Statistiche da Mappe Satellitari
- Esercizi di implementazione in Keras di Classificatori di Testo
- Use Case 2: Deep Learning per NLP

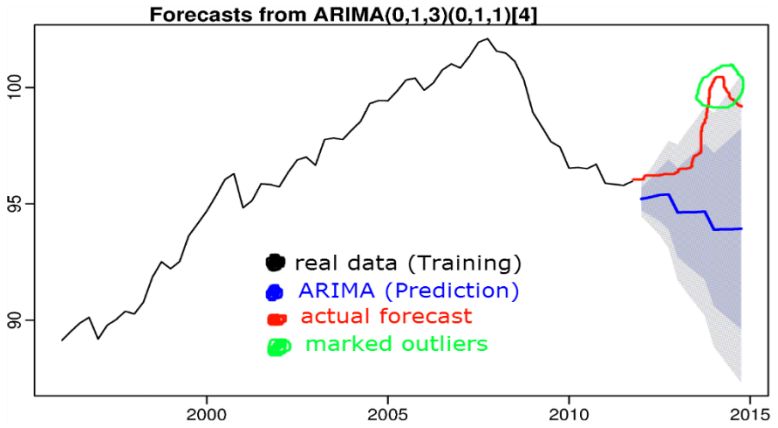
Deep Learning for Time-Series Prediction

- 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).**

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.

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

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



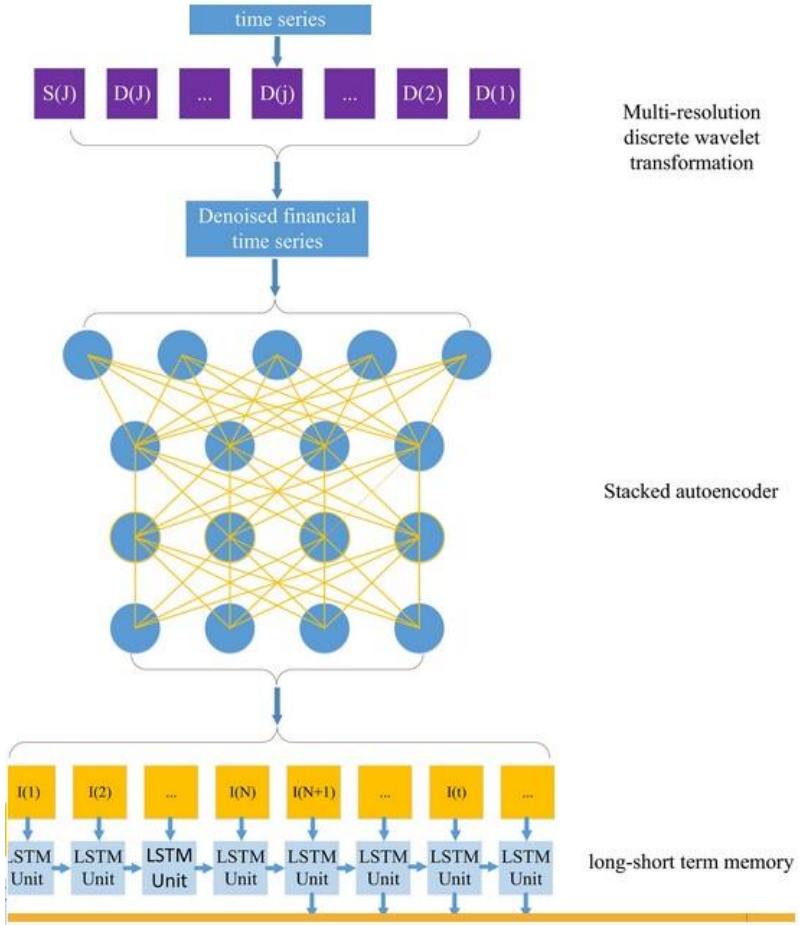
Istat

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.



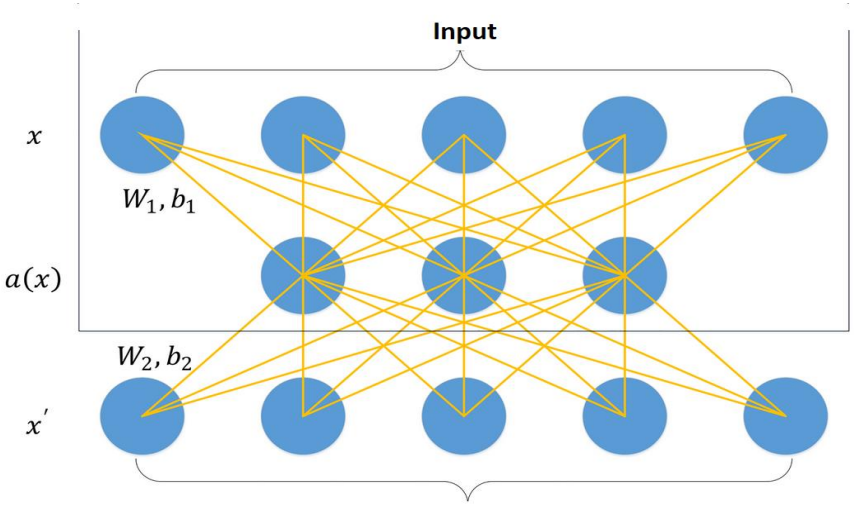
Istat

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 AEE is to minimize the error between thee input vector and the reconstruction vector by **gradient descent**.



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

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

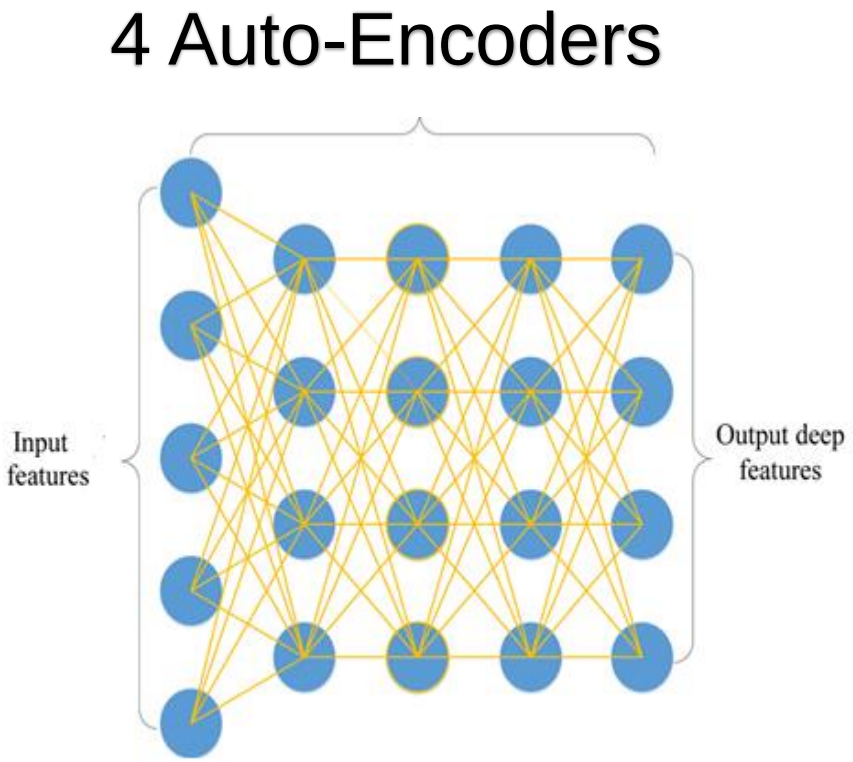
Stacked Auto-encoders (SAEs)

- 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)** .



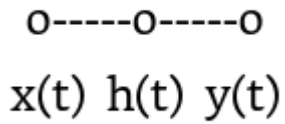
Recurrent Neural Networks (RNNs) : Elman's Architecture

- There exist seveal indicators to measure the predictive accuracy of each model (**Hsieh, et. al.,** 2011; **Theil**, 1973)

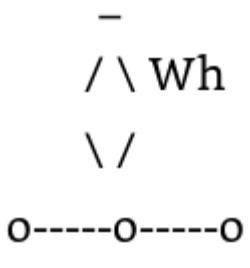
- **RMSE (Root Mean Square Error): R**epresents the sample standard deviation of the differences between predicted values and observed values.
- **MAPE (Mean Absolute Percentage Error):** Measures the size of the error in percentage terms. Most people are comfortable thinking in percentage terms, making the MAPE easy to interpret.
- Thanks to its recursive formulation, RNNs are not limited by the **Markov assumption** for sequence modeling:

$$p\{ x(t) \mid x(t-1), \ldots, x(1) \} = p\{ x(t) \mid x(t-1) \}$$

Simple Feed Forward Artificial Neural Network (



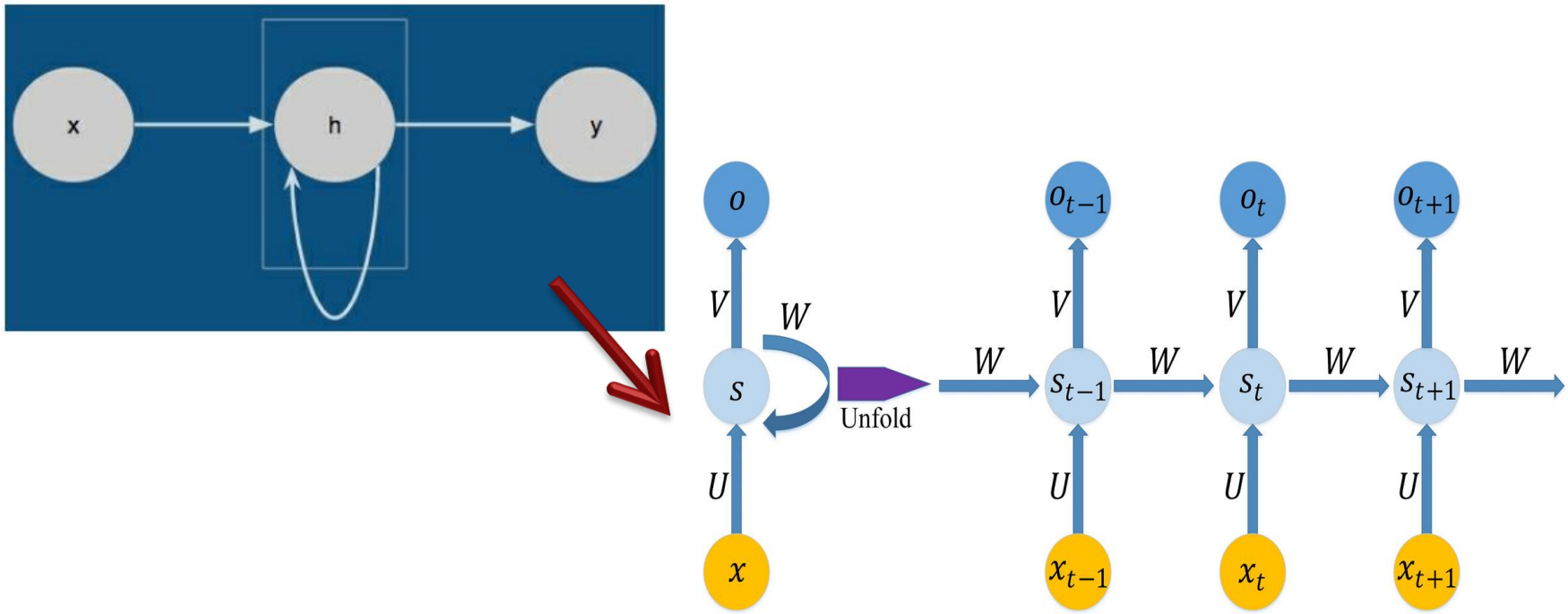
Recurrent Neural Network (Elman's Architectu



$$h(t) = f(x(t)W_i + h(t-1)W_h + b_h)$$

Unfolding RNNs

- Although RNN models the time series well, it is hard to learn long-term dependencies because of the vanishing gradient problem in Back-Propagation Through Time (BPTT) **(Palangi H, et al., 2016)**



Back Propagation Through Time (BPTT)

- In BPTT updating weights is going to look exactly the same.
- **We can prove that the derivative of the loss function "Cross-Entropy" passes through the derivative of and t** $p_j = \frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}}$
- **Things are going to be multiplied together over and over again, due to the chain rule of calculus:**

$$d[W_h{}^T h(t-1)] / dW_h$$

- **The result is that gradients go down through the time (vanishing gradient problem) or they get very large very quickly (exploding gradient problem)**
- **RRNNs, GRUs, LSTMs solve the gradient problems with BPTT**

$$y(t)=\text{softmax}(W_o{}^T h(t))$$

$$y(t)=\text{softmax}(W_o{}^T f(W_h{}^T h(t-1)+W_x{}^T x(t)))$$

$$y(t)=\text{softmax}(W_o{}^T f(W_h{}^T f(W_h{}^T h(t-2)+W_x{}^T x(t-1))+W_x{}^T x(t)))$$

$$y(t)=\text{softmax}(W_o{}^T f(W_h{}^T f(W_h{}^T f(W_h{}^T h(t-3)+W_x{}^T x(t-2))+W_x{}^T x(t-1))+W_x{}^T x(t)))$$

- We drop the bias in ordcer to display things simply

Istat

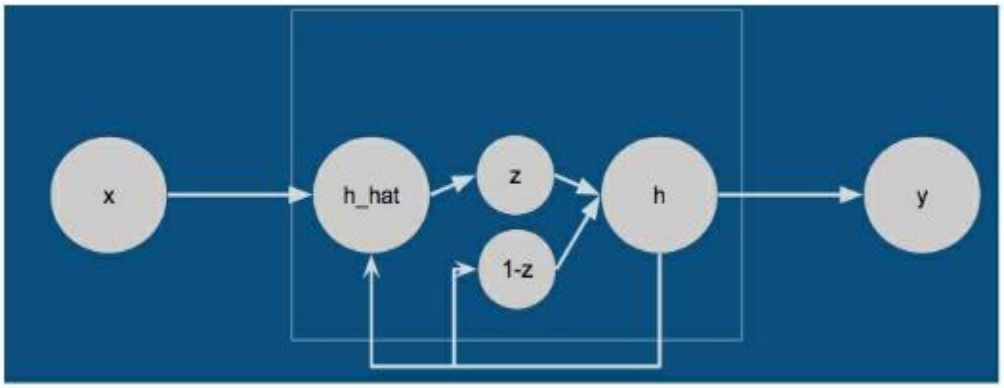
Rated Recurrent Neural Networks (RRNNs)

- The idea is to weight *f(x, h(t-1)),* which is the output of a simple RNN and *h(t-1)* which is the previous state **(Amari, et al., 1995)**.

- **We add a rating operation between what would have been the output of a simple RNN and the previous output value.**

- **This new operation can be seen as a gate since it takes a value between 0 an 1, and the other gate has to take 1 minus that value**

- **This is a gate that is choosing between 2 things: a) taking on the old value or taking the new value. As result we get a mixture of both.**



$$h\_hat(t) = f(x(t)W_x + h(t-1)W_h + b_h)$$

$$z(t) = sigmoid(x(t)W_{xz} + h(t-1)W_{hz} + b_z)$$

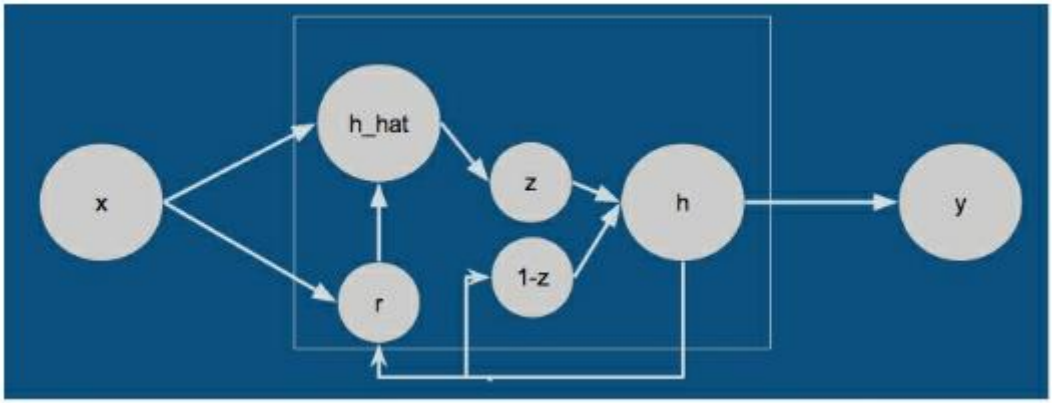
$$h(t) = (1 - z(t)) * h(t-1) + z(t) * h\_hat(t)$$

- **Z(t) is called the "rate"**

Gated Recurrent Neural Networks (GRUs)

- Gated Reccurrent Units were introduced in 2014 and are a simpler versione of LSTM. They have less parameters but same concepts **(Chung, et al., 2014).**

- **Recent research has also show that the accuracy between LSTM and GRU is comparable and even better with the GRUs in some cases.**
- **In GRUs we add one more gate with regard to RRNNs: the "reset gate r(t)" controlling how much of the previous hidden we will consider when we create a new candidate hidden value. In other words, it can "reset" the hidden value.**
- **The old gate of RRNNs is now called "update gate z(t)" balancing previous hidden values and new candidate hidden value for the new hidden value.**



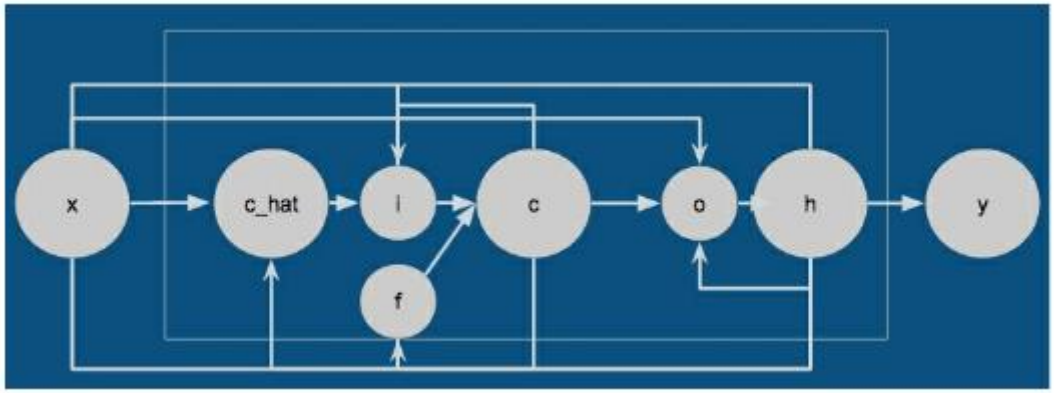
$$r_t = \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r)$$
$$z_t = \sigma(x_t W_{xz} + h_{t-1} W_{hz} + b_z)$$
$$\hat{h}_t = g(x_t W_{xh} + (r_t \odot h_{t-1}) W_{hh} + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t.$$

Istat

Long-Short Term Memories (LSTMs)

- LSTM is an effective solution for combating vanishing gradients by using memory cells **(Hochreiter, et al., 1997).**

- **A memory cell is composed of four units: an input gate, an output gate, a forget gate and a self-recurrent neuron**
- **The gates control the interactions between neighboring memory cells and the memory cell itself. Whether the input signal can alter the state of the memory cell is controlled by the input gate. On the other hand, the output gate can control the state of the memory cell on whether it can alter the state of other memory cell. In addition, the forget gate can choose to remember or forget its previous state.**



$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi} + c_{t-1} W_{ci} + b_i)$$
$$f_t = \sigma(x_t W_{xf} + h_{t-1} W_{hf} + c_{t-1} W_{cf} + b_f)$$
$$c_t = f_t c_{t-1} + i_t \tanh(x_t W_{xc} + h_{t-1} W_{hc} + b_c)$$
$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho} + c_t W_{co} + b_o)$$
$$h_t = o_t \tanh(c_t)$$

Metrics

- There exist several indicators to measure the predictive accuracy of each model (**Hsieh, et. al.,** 2011; **Theil**, 1973)

- **RMSE (Root Mean Square Error): R**epresents the sample standard deviation of the differences between predicted values and observed values.
- **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.
- **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.

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

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

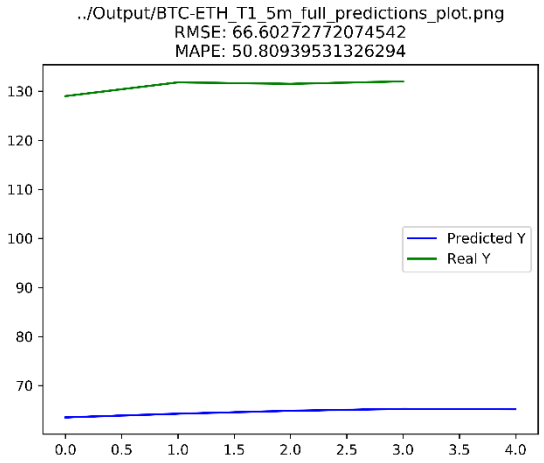
$$Theil\ U = \frac{\sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t - y_t^*)^2}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t)^2} + \sqrt{\frac{1}{N}\sum_{t=1}^{N}(y_t^*)^2}}$$

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

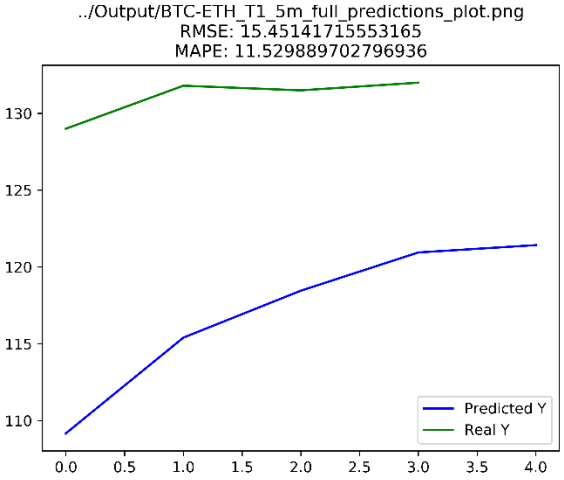
Istat

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

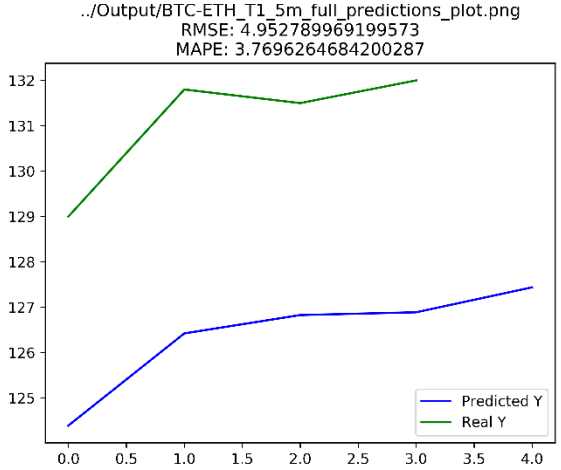
Test Set : 10%



1 Epoch



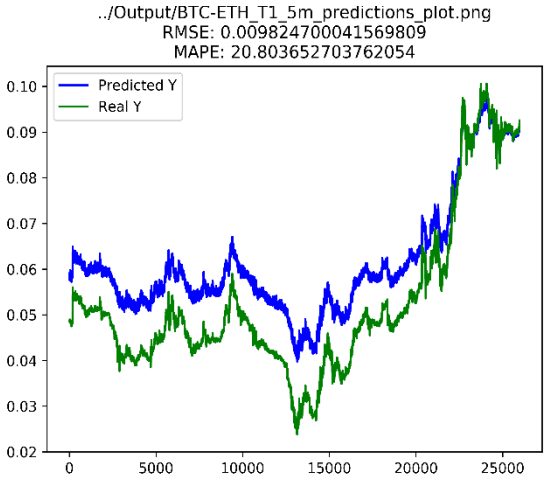
100 Epochs



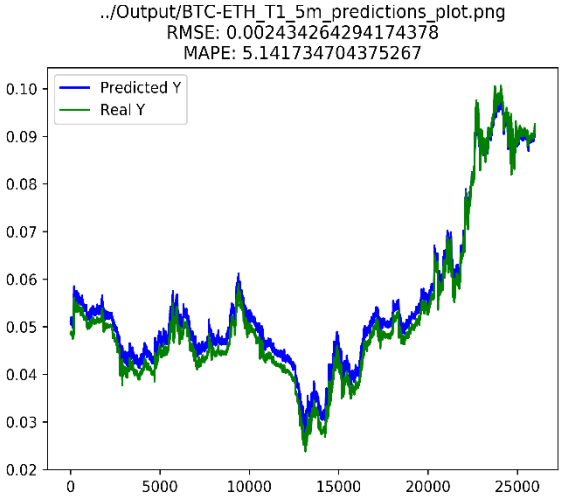
1000 Epochs

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

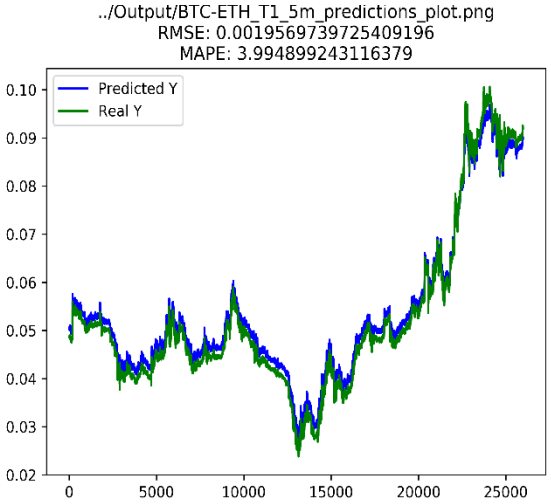
Test Set : 10%



1 Epoch



10 Epochs

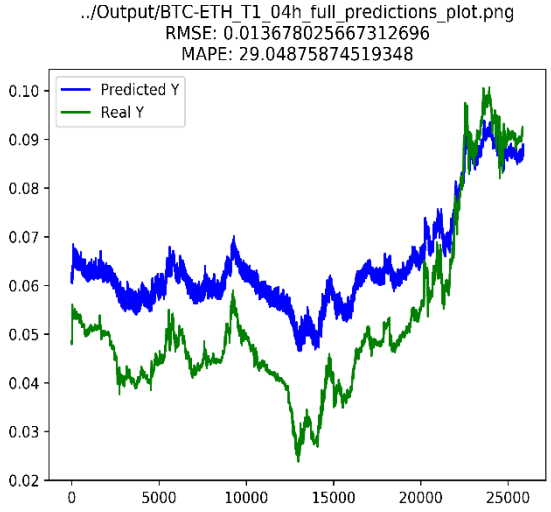


100 Epochs

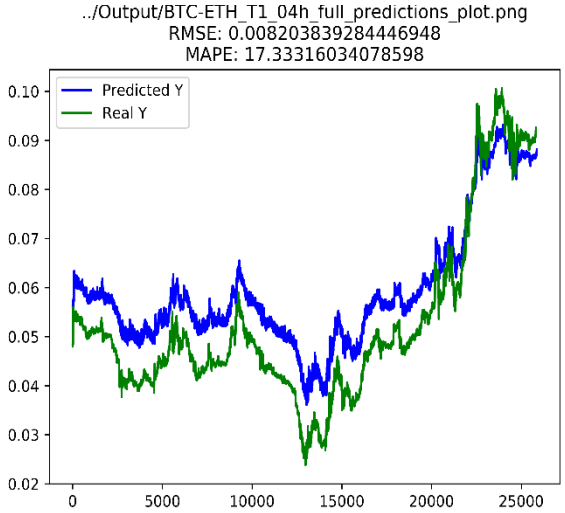
Istat

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

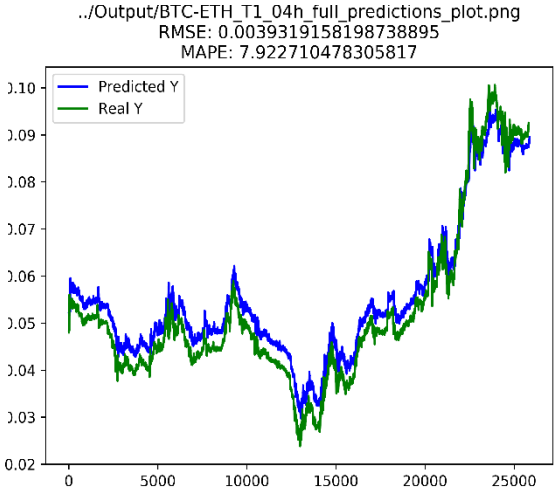
Test Set : 10%



1 Epoch



10 Epochs



100 Epochs

Istat

Time Series Forecasting with Keras: Layers.Recurrent.RNN

RNN [source]

```
keras.layers.RNN(cell, return_sequences=False, return_state=False, go_backwards=False, stateful=False, unroll=Fals
```

Base class for recurrent layers.

Arguments

- **cell**: A RNN cell instance. A RNN cell is a class that has:
  - a `call(input_at_t, states_at_t)` method, returning `(output_at_t, states_at_t_plus_1)`. The call method of the cell can also take the optional argument `constants`, see section "Note on passing external constants" below.
  - a `state_size` attribute. This can be a single integer (single state) in which case it is the size of the recurrent state (which should be the same as the size of the cell output). This can also be a list/tuple of integers (one size per state).
  - a `output_size` attribute. This can be a single integer or a TensorShape, which represent the shape of the output. For backward compatible reason, if this attribute is not available for the cell, the value will be inferred by the first element of the `state_size`.

  It is also possible for `cell` to be a list of RNN cell instances, in which cases the cells get stacked on after the other in the RNN, implementing an efficient stacked RNN.
- **return_sequences**: Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **return_state**: Boolean. Whether to return the last state in addition to the output.
- **go_backwards**: Boolean (default False). If True, process the input sequence backwards and return the reversed sequence.
- **stateful**: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch

Istat

Time Series Forecasting with Keras: Layers.Recurrent.RNN

- **unroll:** Boolean (default False). If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.
- **input_dim:** dimensionality of the input (integer). This argument (or alternatively, the keyword argument `input_shape`) is required when using this layer as the first layer in a model.
- **input_length:** Length of input sequences, to be specified 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). Note that if the recurrent layer is not the first layer in your model, you would need to specify the input length at the level of the first layer (e.g. via the `input_shape` argument)

Input shape

3D tensor with shape `(batch_size, timesteps, input_dim)`.

Output shape

- if `return_state`: a list of tensors. The first tensor is the output. The remaining tensors are the last states, each with shape `(batch_size, units)`.
- if `return_sequences`: 3D tensor with shape `(batch_size, timesteps, units)`.
- else, 2D tensor with shape `(batch_size, units)`.

Time Series Forecasting with Keras: Layers.Recurrent.SimpleRNN

SimpleRNN [source]

```
keras.layers.SimpleRNN(units, activation='tanh', use_bias=True, kernel_initializer='glorot_uniform', recurrent_ini
```

Fully-connected RNN where the output is to be fed back to input.

Arguments

units: Positive integer, dimensionality of the output space. activation: Activation function to use (see activations). Default: hyperbolic tangent (`tanh`). If you pass `None` , 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, used for the linear transformation of the inputs (see initializers). recurrent_initializer: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state (see initializers). bias_initializer: Initializer for the bias vector (see initializers). kernel_regularizer: Regularizer function applied to the `kernel` weights matrix (see regularizer). recurrent_regularizer: Regularizer function applied to the `recurrent_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` weights matrix (see constraints). recurrent_constraint: Constraint function applied to the `recurrent_kernel` weights matrix (see constraints). bias_constraint: Constraint function applied to the bias vector (see constraints). dropout: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs. recurrent_dropout: Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state. return_sequences: Boolean. Whether to return the last output in the output sequence, or the full sequence. return_state: Boolean. Whether to return the last state in addition to the output. go_backwards: Boolean (default False). If True, process the input sequence backwards and return the reversed sequence. stateful: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. unroll:

Time Series Forecasting with Keras: Layers.Recurrent.GRU

GRU [source]

```
keras.layers.GRU(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, kernel_initializer=
```

Gated Recurrent Unit - Cho et al. 2014.

There are two variants. The default one is based on 1406.1078v3 and has reset gate applied to hidden state before matrix multiplication. The other one is based on original 1406.1078v1 and has the order reversed.

The second variant is compatible with CuDNNGRU (GPU-only) and allows inference on CPU. Thus it has separate biases for `kernel` and `recurrent_kernel`. Use `'reset_after'=True` and `recurrent_activation='sigmoid'`.

Arguments

- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use (see activations). Default: hyperbolic tangent (`tanh`). If you pass `None`, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **recurrent_activation:** Activation function to use for the recurrent step (see activations). Default: hard sigmoid (`hard_sigmoid`). If you pass `None`, 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, used for the linear transformation of the inputs (see initializers).
- **recurrent_initializer:** Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state (see initializers).
- **bias_initializer:** Initializer for the bias vector (see initializers).
- **kernel_regularizer:** Regularizer function applied to the `kernel` weights matrix (see regularizer).

Time Series Forecasting with Keras: Layers.Recurrent.GRU

- **recurrent_regularizer:** Regularizer function applied to the `recurrent_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` weights matrix (see constraints).
- **recurrent_constraint:** Constraint function applied to the `recurrent_kernel` weights matrix (see constraints).
- **bias_constraint:** Constraint function applied to the bias vector (see constraints).
- **dropout:** Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- **recurrent_dropout:** Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.
- **implementation:** Implementation mode, either 1 or 2. Mode 1 will structure its operations as a larger number of smaller dot products and additions, whereas mode 2 will batch them into fewer, larger operations. These modes will have different performance profiles on different hardware and for different applications.
- **return_sequences:** Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **return_state:** Boolean. Whether to return the last state in addition to the output.
- **go_backwards:** Boolean (default False). If True, process the input sequence backwards and return the reversed sequence.
- **stateful:** Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- **unroll:** Boolean (default False). If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.
- **reset_after:** GRU convention (whether to apply reset gate after or before matrix multiplication). False = "before" (default), True = "after" (CuDNN compatible).

Time Series Forecasting with Keras: Layers.Recurrent.LSTM

LSTM

[source]

```
keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, kernel_initializer
```

Long Short-Term Memory layer - Hochreiter 1997.

Arguments

- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use (see activations). Default: hyperbolic tangent (`tanh`). If you pass `None` , no activation is applied (ie. "linear" activation: `a(x) = x`).
- **recurrent_activation:** Activation function to use for the recurrent step (see activations). Default: hard sigmoid (`hard_sigmoid`). If you pass `None` , 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, used for the linear transformation of the inputs. (see initializers).
- **recurrent_initializer:** Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see initializers).
- **bias_initializer:** Initializer for the bias vector (see initializers).
- **unit_forget_bias:** Boolean. If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"` . This is recommended in Jozefowicz et al. (2015).
- **kernel_regularizer:** Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **recurrent_regularizer:** Regularizer function applied to the `recurrent_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` weights matrix (see constraints).

Time Series Forecasting with Keras: Layers.Recurrent.LSTM

- **bias_constraint:** Constraint function applied to the bias vector (see constraints).
- **dropout:** Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- **recurrent_dropout:** Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.
- **implementation:** Implementation mode, either 1 or 2. Mode 1 will structure its operations as a larger number of smaller dot products and additions, whereas mode 2 will batch them into fewer, larger operations. These modes will have different performance profiles on different hardware and for different applications.
- **return_sequences:** Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **return_state:** Boolean. Whether to return the last state in addition to the output.
- **go_backwards:** Boolean (default False). If True, process the input sequence backwards and return the reversed sequence.
- **stateful:** Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- **unroll:** Boolean (default False). If True, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.

Time Series Forecasting with Keras: Layers.Recurrent.ConvLSTM2D

ConvLSTM2D [source]

```
keras.layers.ConvLSTM2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1,
```

Convolutional LSTM.

It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional.

Arguments

- **filters:** Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size:** An integer or tuple/list of n integers, specifying the dimensions of the convolution window.
- **strides:** An integer or tuple/list of n integers, specifying the strides of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding:** One of `"valid"` or `"same"` (case-insensitive).
- **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, time, ..., channels)` while `"channels_first"` corresponds to inputs with shape `(batch, time, channels, ...)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be `"channels_last"`.
- **dilation_rate:** An integer or tuple/list of n integers, 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`).
- **recurrent_activation:** Activation function to use for the recurrent step (see activations).
- **use_bias:** Boolean, whether the layer uses a bias vector.
- **kernel_initializer:** Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see initializers).

Time Series Forecasting with Keras: Layers.Recurrent.ConvLSTM2D

- **bias_initializer:** Initializer for the bias vector (see initializers).
- **unit_forget_bias:** Boolean. If True, add 1 to the bias of the forget gate at initialization. Use in combination with `bias_initializer="zeros"`. This is recommended in Jozefowicz et al. (2015).
- **kernel_regularizer:** Regularizer function applied to the `kernel` weights matrix (see regularizer).
- **recurrent_regularizer:** Regularizer function applied to the `recurrent_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` weights matrix (see constraints).
- **recurrent_constraint:** Constraint function applied to the `recurrent_kernel` weights matrix (see constraints).
- **bias_constraint:** Constraint function applied to the bias vector (see constraints).
- **return_sequences:** Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **go_backwards:** Boolean (default False). If True, process the input sequence backwards.
- **stateful:** Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- **dropout:** Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.
- **recurrent_dropout:** Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

Time Series Forecasting with Keras: Layers.Recurrent.CuDNNGRU

CuDNNGRU [source]

```
keras.layers.CuDNNGRU(units, kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initial
```

Fast GRU implementation backed by CuDNN.

Can only be run on GPU, with the TensorFlow backend.

Arguments

units: Positive integer, dimensionality of the output space. kernel_initializer: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see initializers). recurrent_initializer: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see initializers). bias_initializer: Initializer for the bias vector (see initializers). kernel_regularizer: Regularizer function applied to the `kernel` weights matrix (see regularizer). recurrent_regularizer: Regularizer function applied to the `recurrent_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` weights matrix (see constraints). recurrent_constraint: Constraint function applied to the `recurrent_kernel` weights matrix (see constraints). bias_constraint: Constraint function applied to the bias vector (see constraints). return_sequences: Boolean. Whether to return the last output. in the output sequence, or the full sequence. return_state: Boolean. Whether to return the last state in addition to the output. stateful: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

Time Series Forecasting with Keras: Layers.Recurrent.CuDNNLSTM

CuDNNLSTM [source]

```
keras.layers.CuDNNLSTM(units, kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initia
```

Fast LSTM implementation with CuDNN.

Can only be run on GPU, with the TensorFlow backend.

Arguments

units: Positive integer, dimensionality of the output space. kernel_initializer: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see initializers). unit_forget_bias: Boolean. If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"`. This is recommended in Jozefowicz et al. (2015). recurrent_initializer: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see initializers). bias_initializer: Initializer for the bias vector (see initializers). kernel_regularizer: Regularizer function applied to the `kernel` weights matrix (see regularizer). recurrent_regularizer: Regularizer function applied to the `recurrent_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` weights matrix (see constraints). recurrent_constraint: Constraint function applied to the `recurrent_kernel` weights matrix (see constraints). bias_constraint: Constraint function applied to the bias vector (see constraints). return_sequences: Boolean. Whether to return the last output. in the output sequence, or the full sequence. return_state: Boolean. Whether to return the last state in addition to the output. stateful: Boolean (default False). If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

Exercise : Multivariate Time-Series Forecasting in Keras

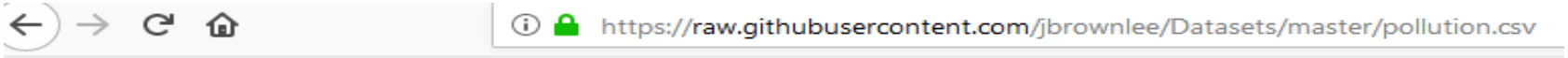
1. Air Pollution Forecasting

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




```
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
```

Istat

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

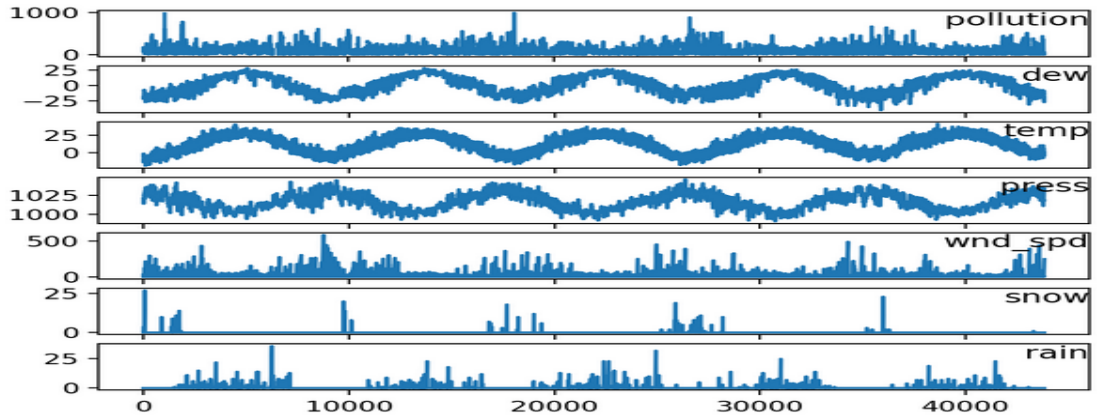
```
def pollution_prepare_data(pred_dataset_path='', pred_dataset_file_in='', pred_dataset_file_out=''):
    # load data
    def parse(x):
        return datetime.strptime(x, '%Y %m %d %H')
    dataset = read_csv(pred_dataset_path+'/'+pred_dataset_file_in,  parse_dates = [['year', 'month', 'day', 'hour']], 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

Istat

Time-Series: View Model

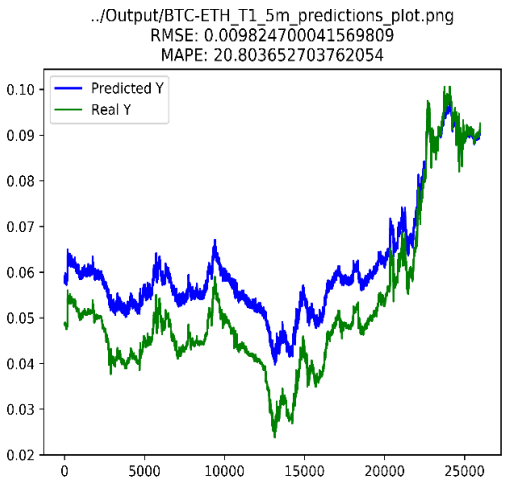
```
# Plot inputs
def plot_dataset(pred_dataset_path='', pred_dataset_file=''):
    # load dataset
    dataset = read_csv(pred_dataset_path+'/'+pred_dataset_file, header=0, index_col=0)
    values = dataset.values
    # specify columns to plot
    groups = [0, 1, 2, 3, 5, 6, 7]
    i = 1
    # plot each 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)
```



Istat

Time-Series: View Model

```
# 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)
```



Istat

Multivariate LSTM Forecast Model

- Neural networks like Long Short-Term Memory (LSTM) recurrent neural networks are able to almost seamlessly model problems with multiple input variables.
- This is a great benefit in time series forecasting, where classical linear methods can be difficult to adapt to multivariate or multiple input forecasting problems.

```
class PredModelLSTM:
    @staticmethod
    def build(input_length, vector_dim, output_size, lstm_n_hiddens, summary):

        # initialize the model
        deepnetwork = Sequential()
        deepnetwork.add(LSTM(lstm_n_hiddens, input_shape = (input_length, vector_dim)))
        deepnetwork.add(Dense(output_size))

        if summary==True:
            deepnetwork.summary()

        return deepnetwork
```

Data Preparation : From Time-Series a Supervised Problem

```
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
```

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

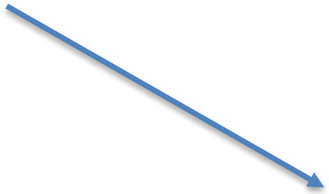
Data Preparation : Pandas shift function

```
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)
```

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

Data Preparation : Multi-Step Univariate Analysis

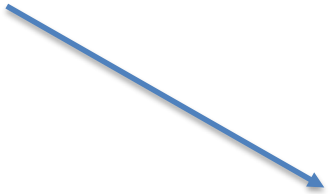
```
1 data = series_to_supervised(values, 3)
```



	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 or Sequence Forecasting

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



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

Data Preparation : Multivariate Forecasting

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

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

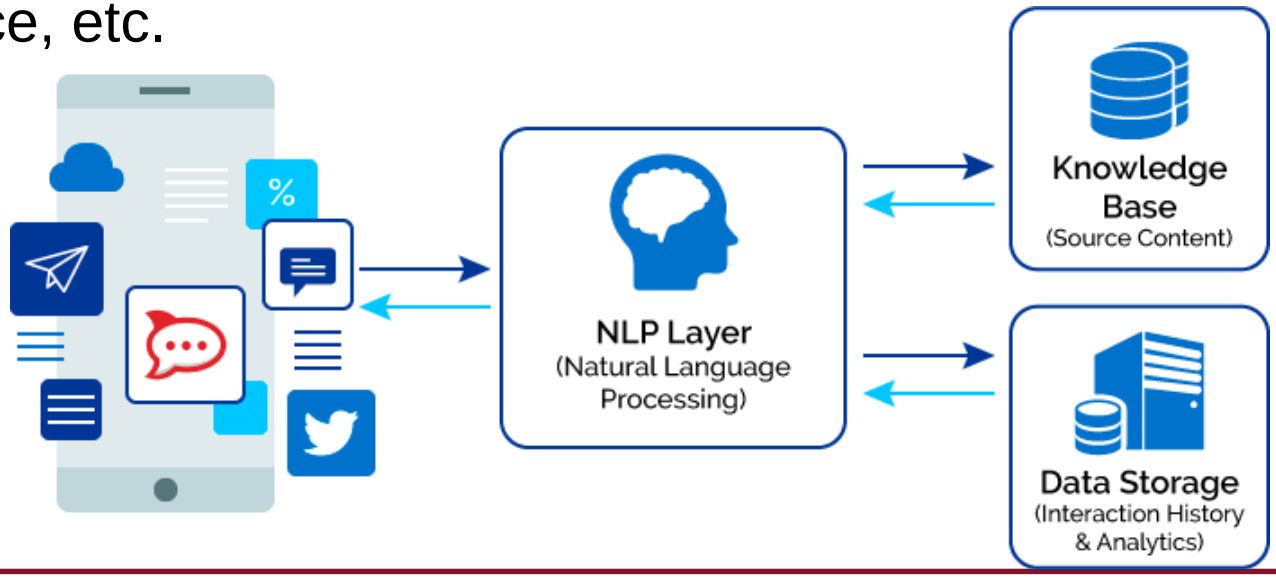
Istat

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)
```

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.



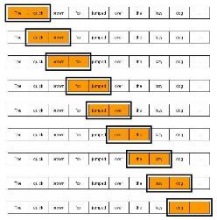
Deep Learning and NLP

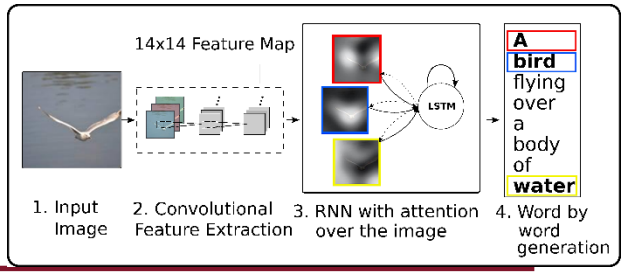
- “Deep Learning” approaches have obtained very high performance across many different **NLP** tasks. These models can often be trained with a **single end-to-end model** and do not require traditional, task-specific feature engineering.
  (*Stanford University School Of Engineering – CS224D*)

- **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.



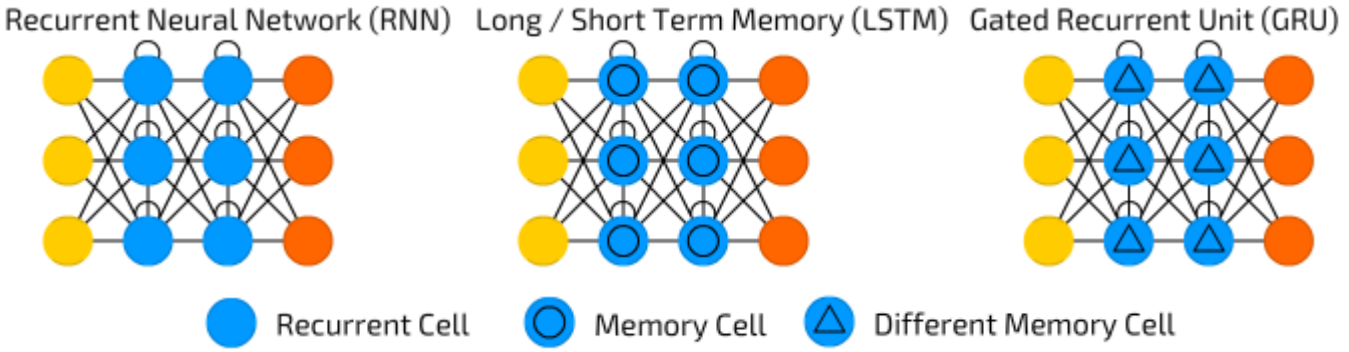


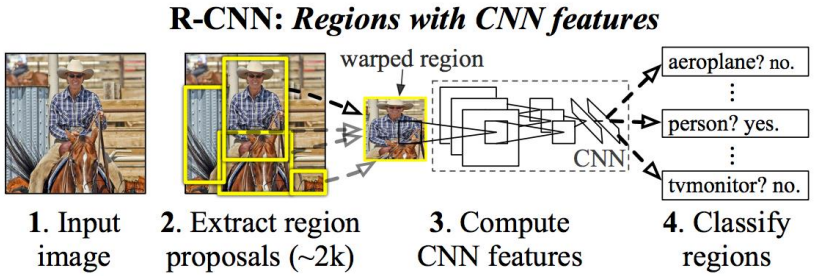
7 NLP applications where Deep Learning achieved «state-of-art» performance

- **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.





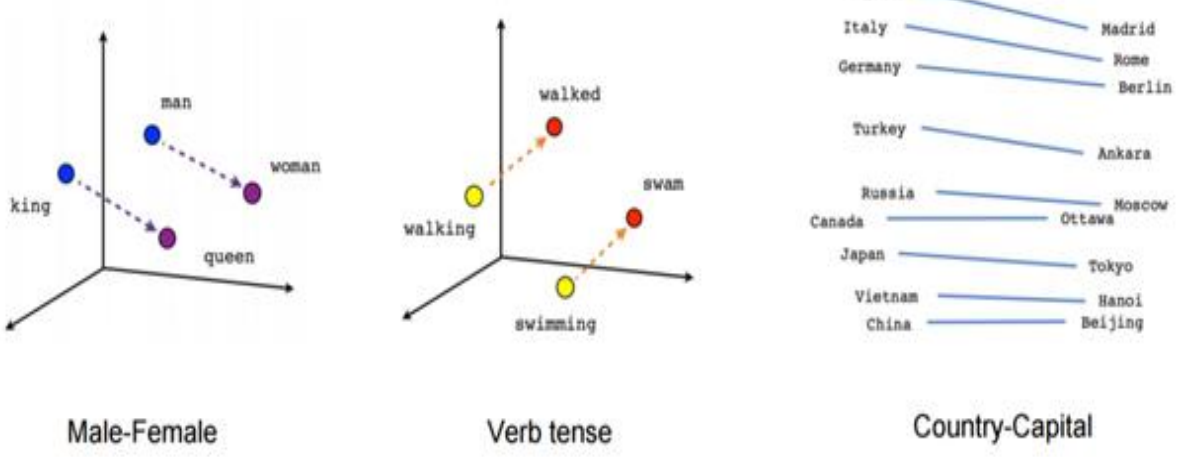
- The **non-linearity of the NN** leads to superior classification accuracy.

Istat

Word Embedding & Language Modeling

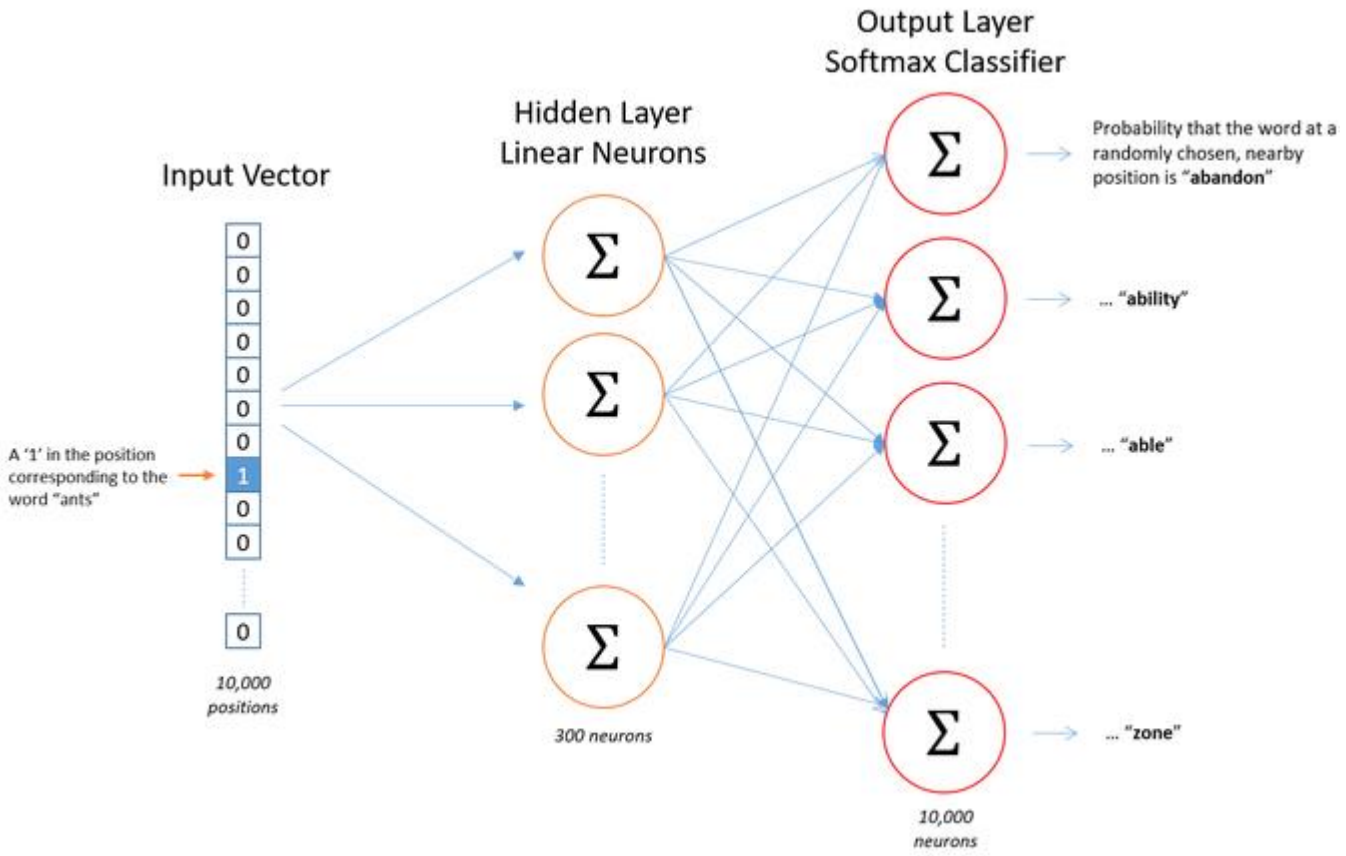
- Word embedding is the collective name for a set of language modeling and feature learning techniques for natural language processing (NLP) where words or sentences from the vocabulary are mapped to vectors of real numbers.

- These vectors are semantically correlated by metrics like cosine distance

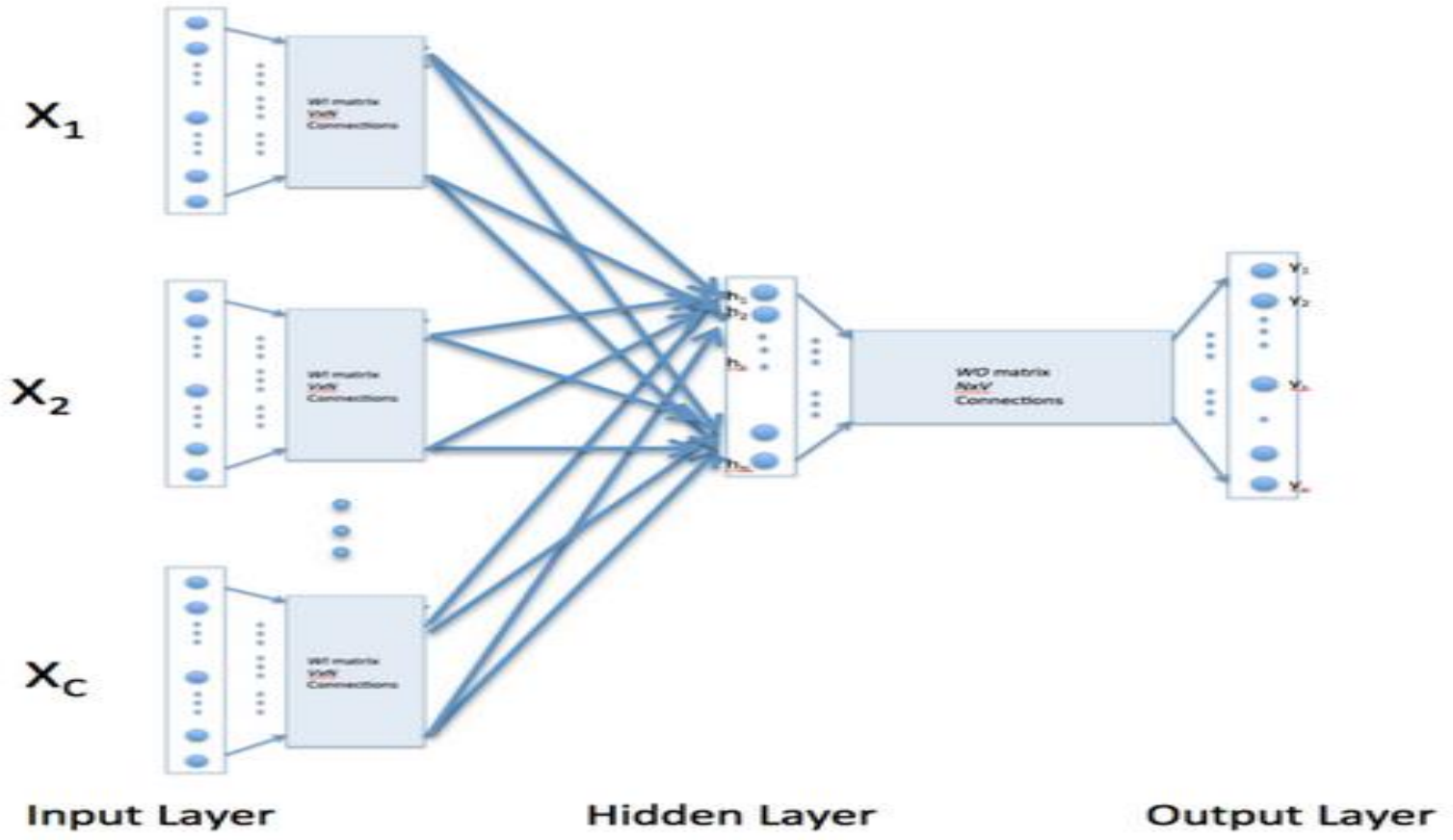


Male-Female | Verb tense | Country-Capital

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



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

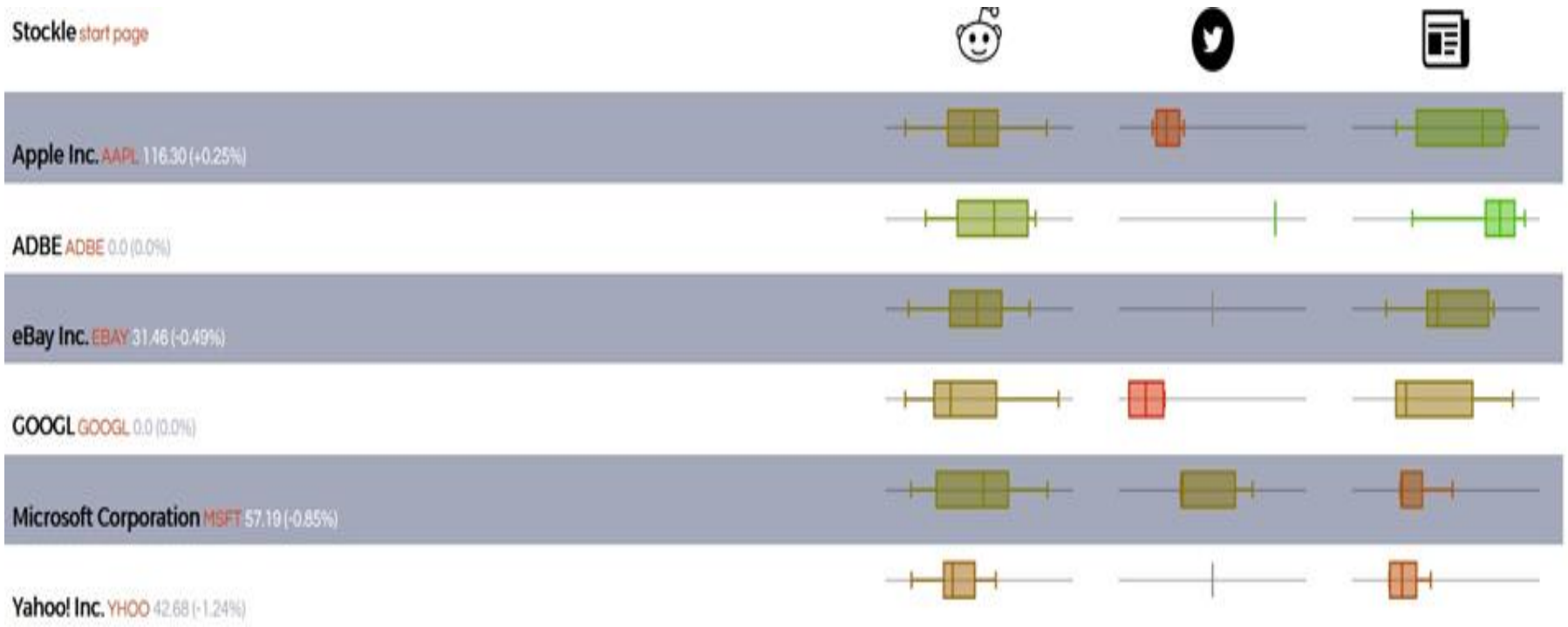


Sentiment Analysis (Ain, et al. 2017)

- **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

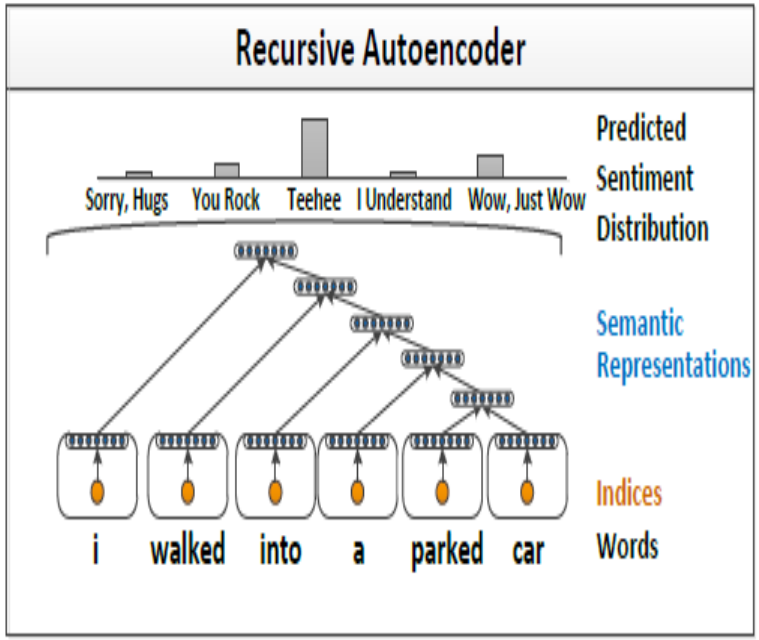
Sentiment Analysis with Feedback



Istat

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

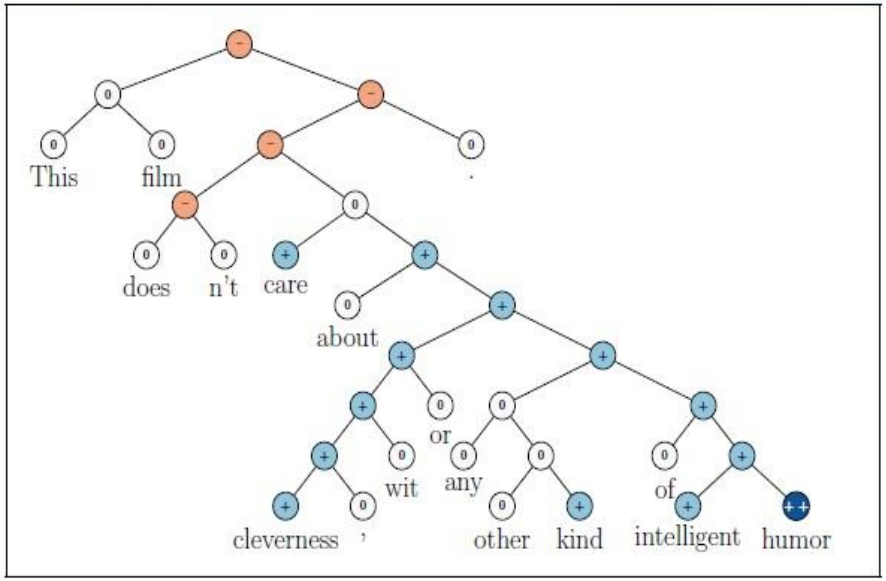
- **This models are recursive auto-encoders which learn semantic vector representations of phrases. Word indices (orange) are first mapped into a semantic vector space (blue).**
- **Then they are recursively merged by the same auto-encoder network into a fixed length sentence representation. The vectors 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.**

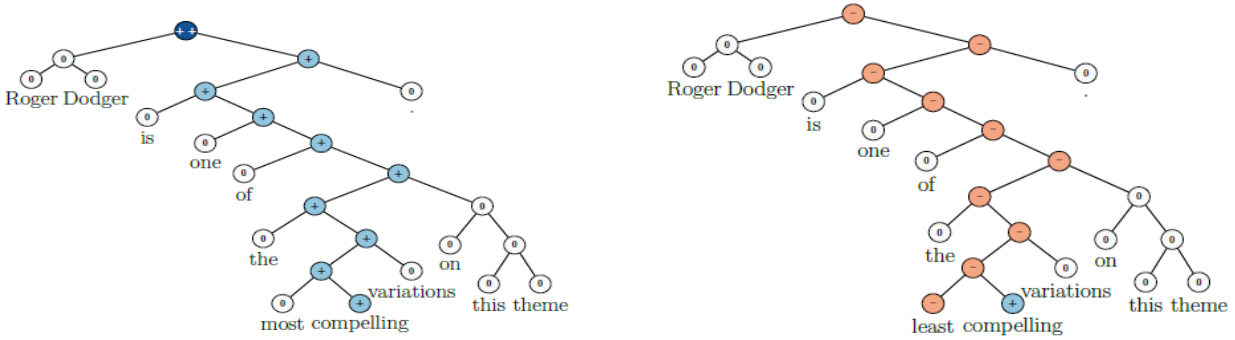
- **<u>RNTNs</u> compute parent vectors in a bottom up fashion using a compo-sitionality function and use node vectors as features for a classifier at that node.**



Istat

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 sentencesor documents.**

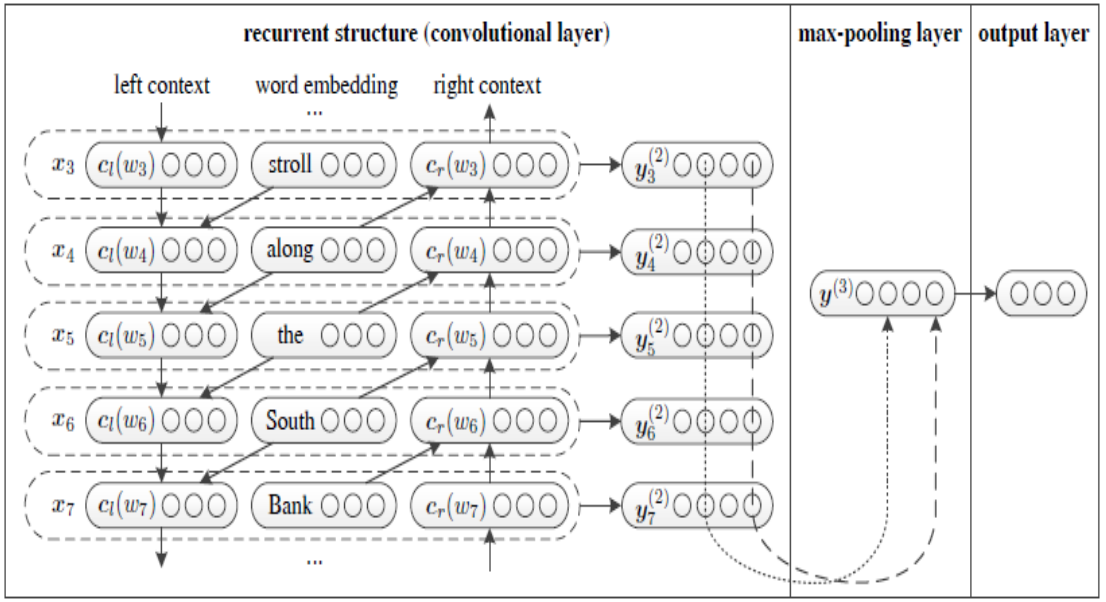


Recurrent Convolutional Neural Networks (RCNN) (Lai, S., et al. 2015)

- 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]
```

Istat

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...
```

Istat

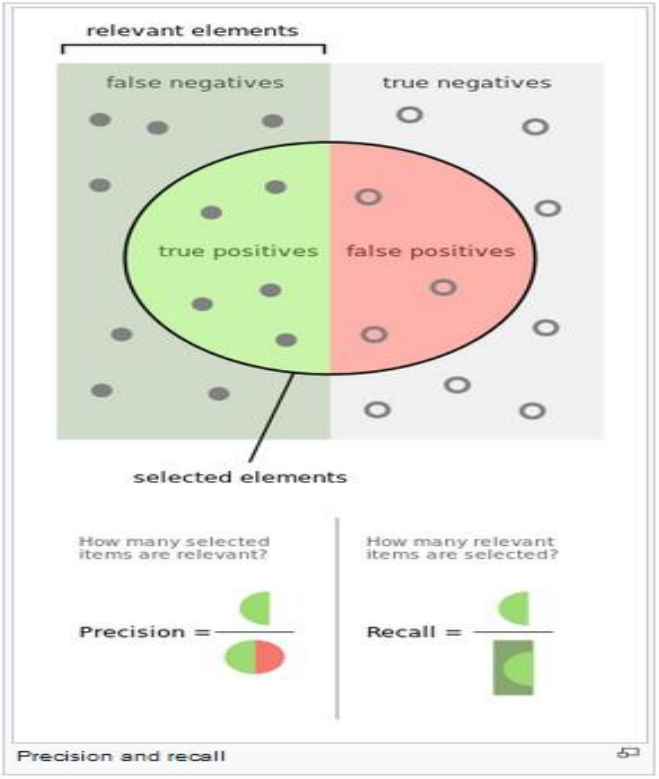
Recurrent Neural Networks are able to understand negations and other things

- 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



Precision and recall

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

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

Istat

Metrics

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 = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$

Istat

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)

Istat

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).

Deep LSTM/GRU Architectures

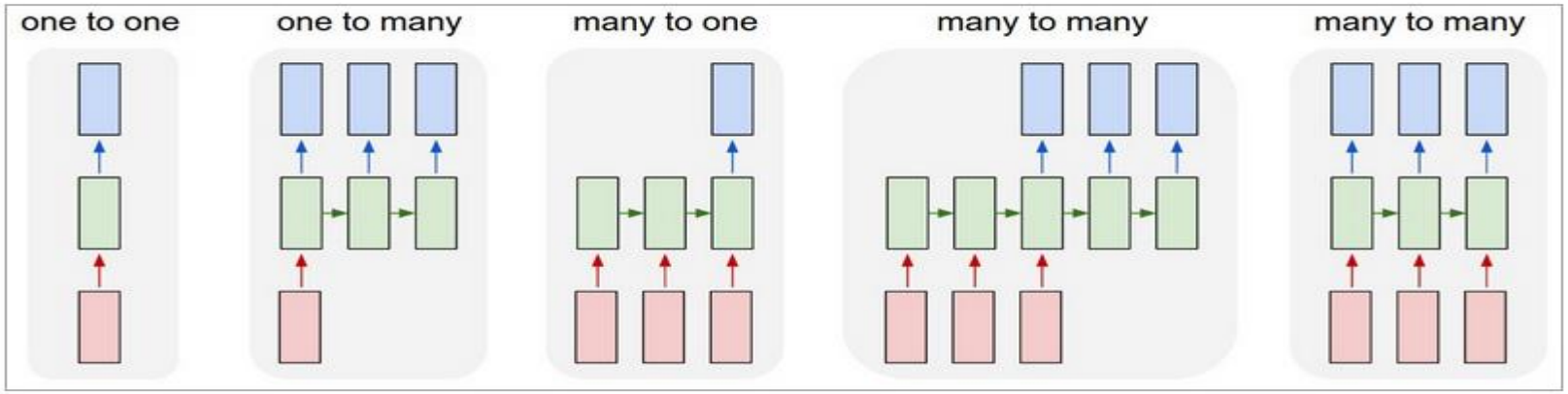
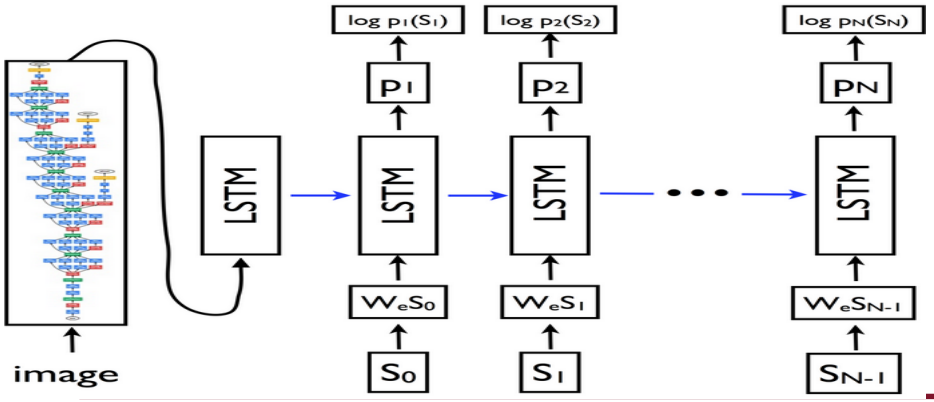
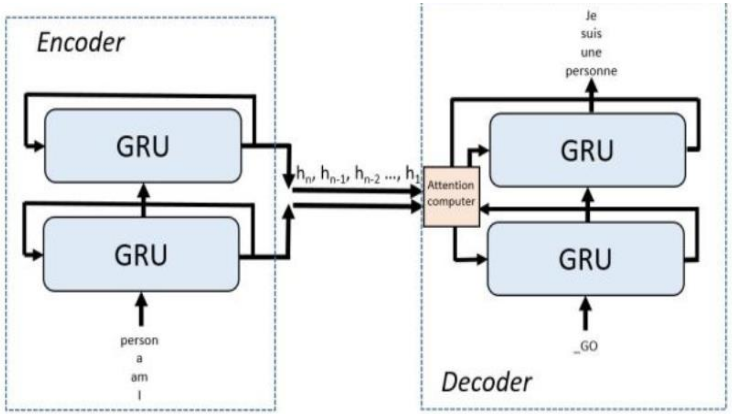


Image Caption Generator

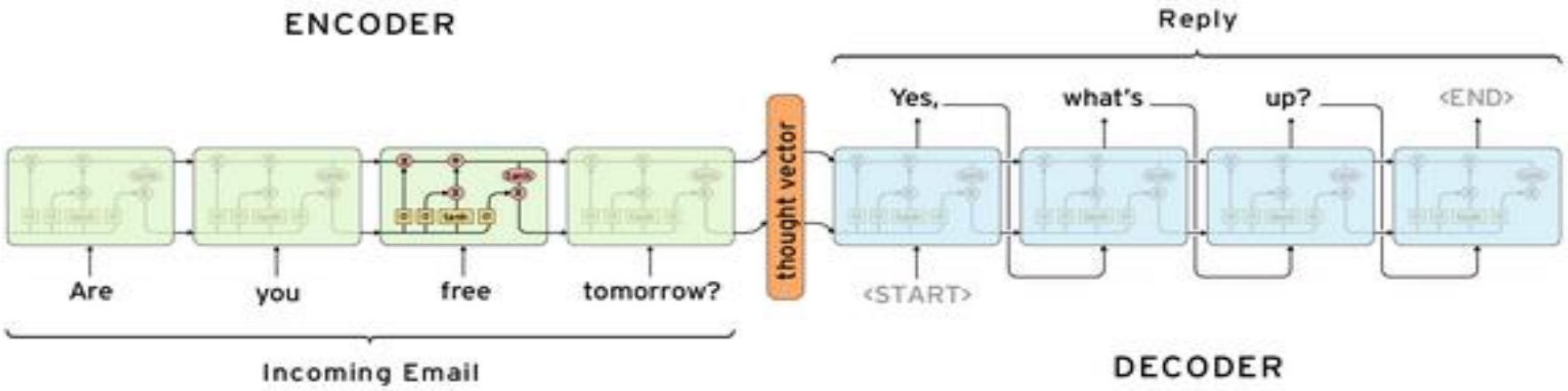


Seq2seq model



Istat

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.

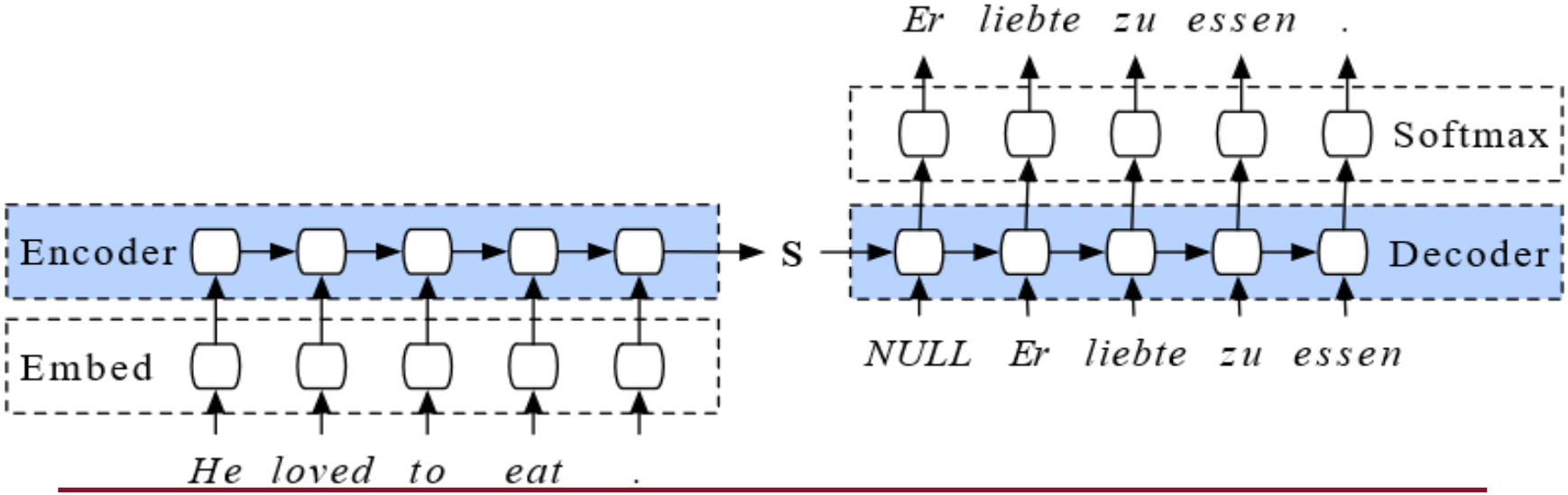
Conversation 4: Philosophical Q&A

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*

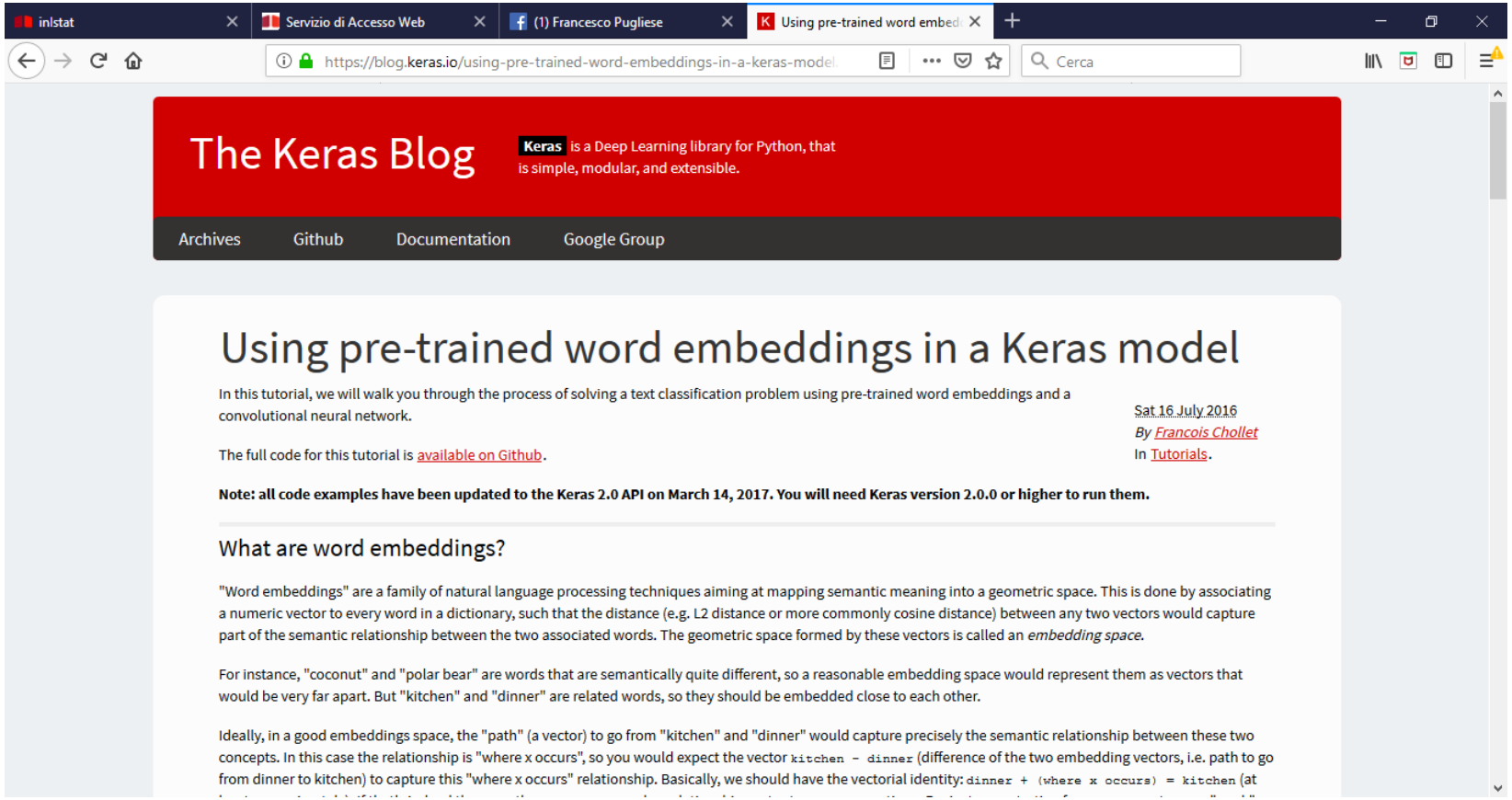
Human: *what is the purpose of life ?*
Machine: *to serve the greater good .*
Human: *what is the purpose of living ?*
Machine: *to live forever .*
Human: *what is the purpose of existence ?*
Machine: *to find out what happens when we get to the planet earth .*
Human: *where are you now ?*
Machine: *i '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).

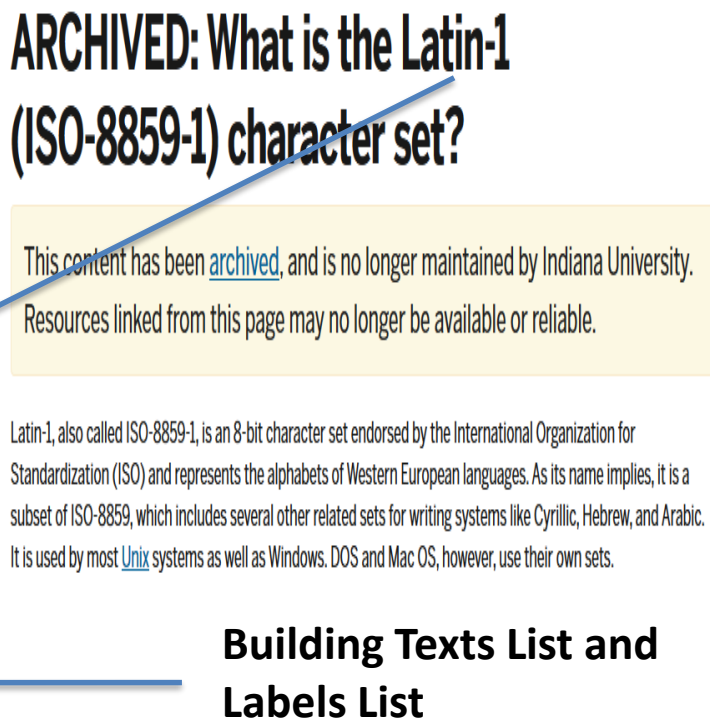
20 Newsgroup dataset

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.

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))
```



Building Texts List and Labels List

Istat

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 conversi on

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 arrange- ment

```
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)
```

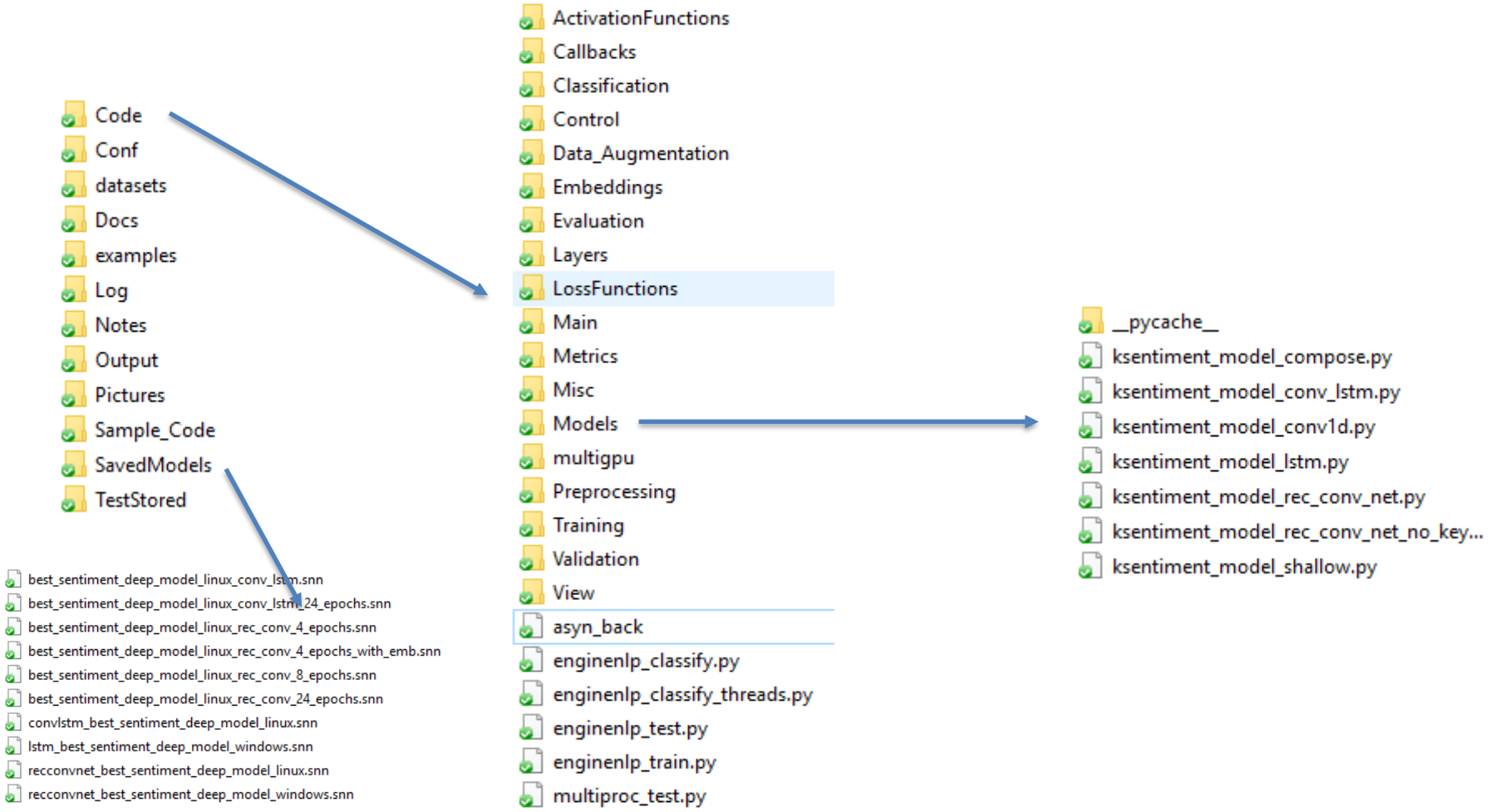
This model reaches **95% classification accuracy** on the validation set after only 2 epochs. You could probably get to an even higher accuracy by training longer with some regularization mechanism (such as dropout) or by fine-tuning the `Embedding` layer.

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



Istat

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).

REFERENCES

Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PloS one, 12(7), e0180944.

Bodyanskiy Y, Popov S. (2006) Neural network approach to forecasting of quasiperiodic financial time series. Eur J Oper Res. 175(3):1357–66.

Nourani V, Komasi M, Mano A. (2009) A multivariate ANN-wavelet approach for rainfall-runoff modeling. Water Resources Management. 2009;23(14):2877.

Kim TY, Oh KJ, Kim C, Do JD. (2004) Artificial neural networks for non-stationary time series. Neurocomputing.

Hsieh TJ, Hsiao HF, Yeh WC. (2011) Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. Applied Soft Computing. 2011;11(2):2510–25. 61(C):439–47.

Kim TY, Oh KJ, Kim C, Do JD. Artificial neural networks for non-stationary time series. Neurocomputing. 2004;61(C):439–47.

REFERENCES

Sutskever I, Hinton GE (2008). Deep, narrow sigmoid belief networks are universal approximators. Neural Computation. 20(11):2629–36. pmid:18533819

Roux NL, Bengio Y. (2010) Deep Belief Networks Are Compact Universal Approximators. Neural Computation.22(8):2192–207.

Bengio Y, Lamblin P, Popovici D, Larochelle H. (2007) Greedy layer-wise training of deep networks. Advances in neural information processing systems. 19:153.

Ramsey JB. (1999) The contribution of wavelets to the analysis of economic and financial data. Philosophical Transactions of the Royal Society B Biological Sciences. 357(357):2593–606.

Palangi H, Ward R, Deng L. (2016) Distributed Compressive Sensing: A Deep Learning Approach. IEEE Transactions on Signal Processing. 64(17):4504–18.

Amari, S. I., Cichocki, A., & Yang, H. H. (1995, December). Recurrent neural networks for blind separation of sources. In *Proc. Int. Symp. NOLTA* (pp. 37-42).

REFERENCES

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.

Acknowledgements

THANK YOU FOR YOUR ATTENTION

FRANCESCO PUGLIESE