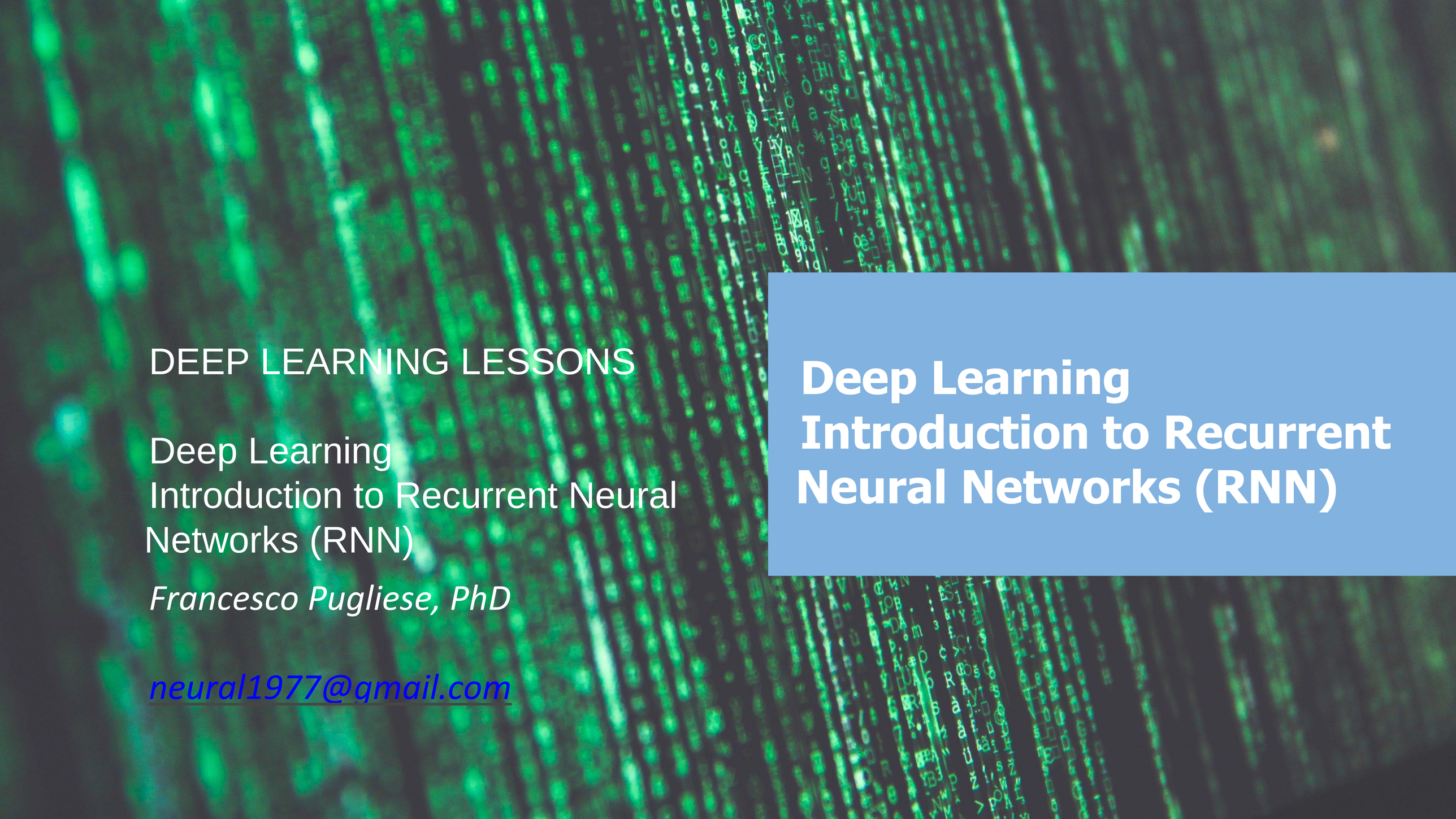


DEEP LEARNING LESSONS

Deep Learning
Introduction to Recurrent Neural Networks (RNN)

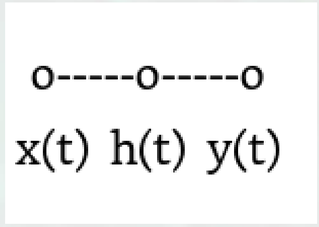
Francesco Pugliese, PhD

neural1977@gmail.com

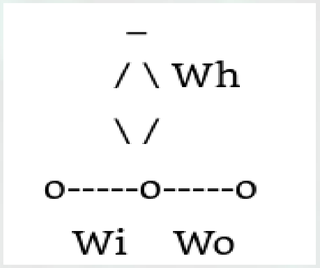
Deep Learning Introduction to Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) : Elman's Architecture

Simple Feed Forward Artificial Neural Network (MLP)



Recurrent Neural Network (Elman's Architecture)



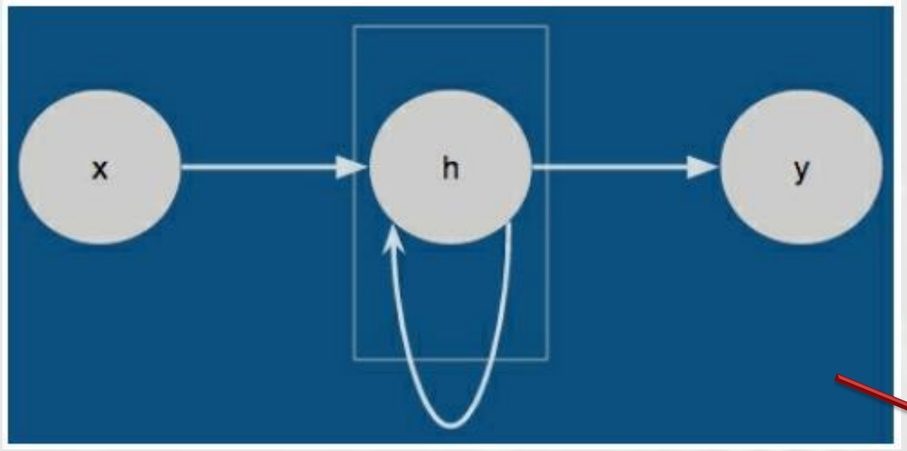
$$h(t) = f(x(t)W_i + h(t-1)W_h + b_h)$$

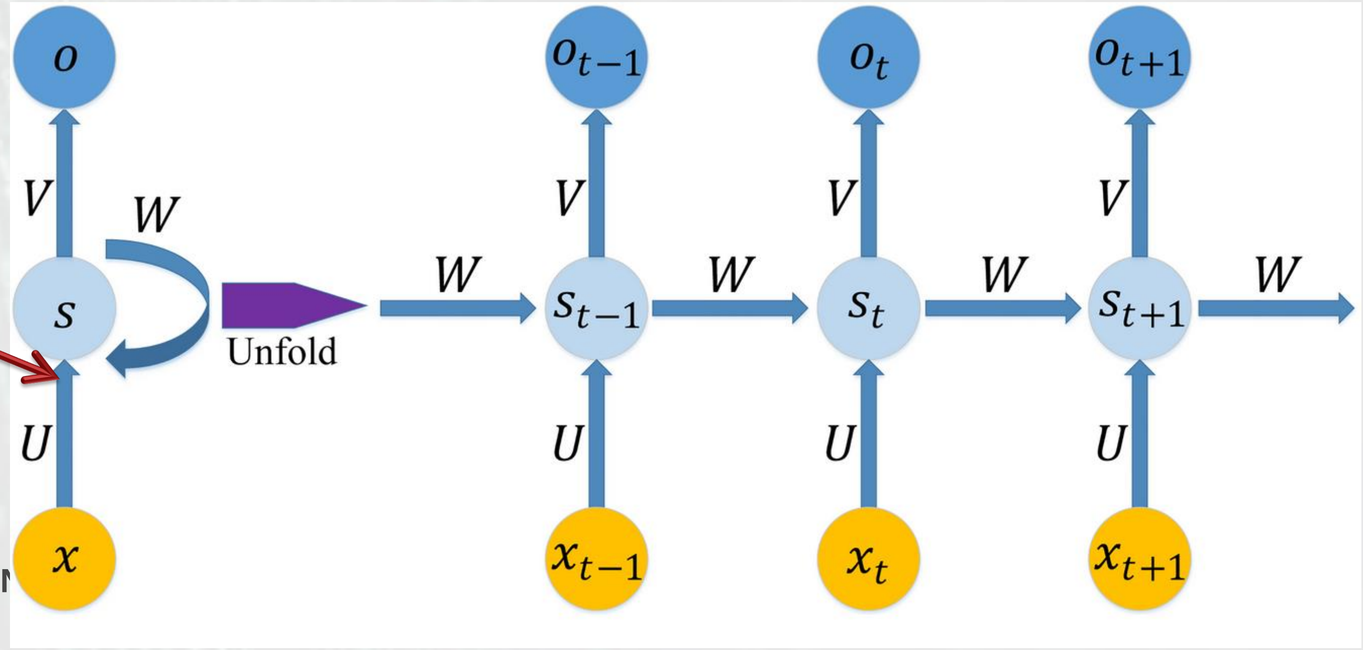
- There exist seveal indicators to measure the predictive accuracy of each model **(Hsieh, et. al., 2011; Theil, 1973)**
- **RMSE (Root Mean Square Error):** Represents the sample standard deviation of the differences between predicted values and observed values.
- **MAPE (Mean Absolute Percentage Error):**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

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

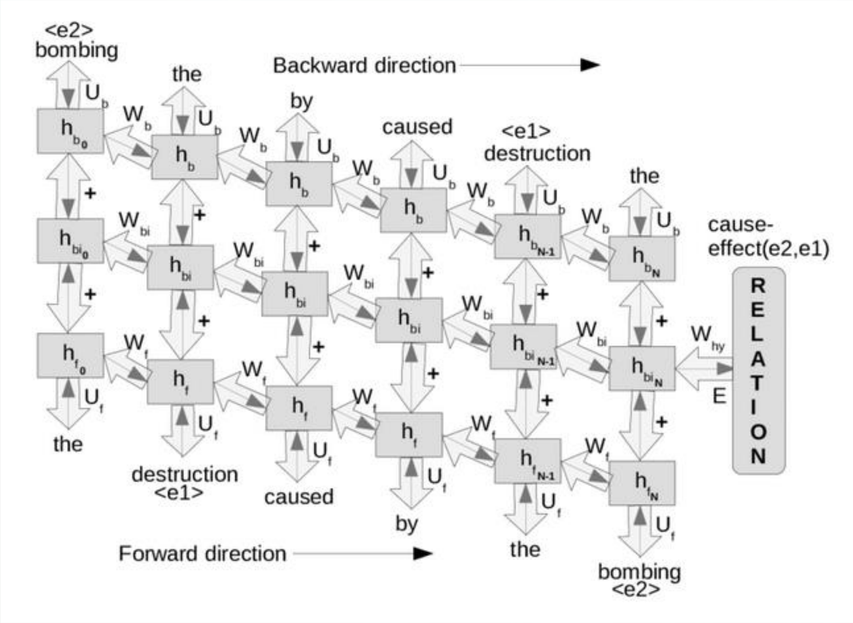
Unfolding RNNs

- 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)

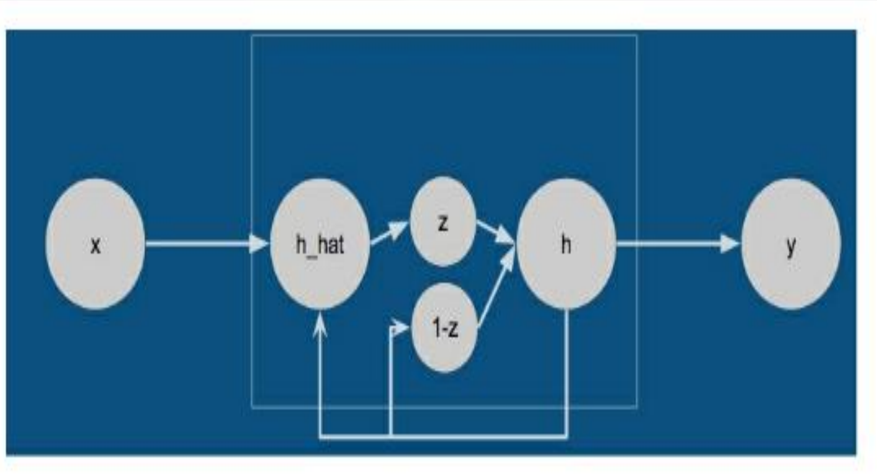


- In BPTT updating weights is going to look exactly the same.
- We can prove that the derivative of the loss function “Cross-Entropy” passes through the derivative of and the Softmax.
- Things are going to be multiplied together over and over again, due to the chain rule of calculus:

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

- 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

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.

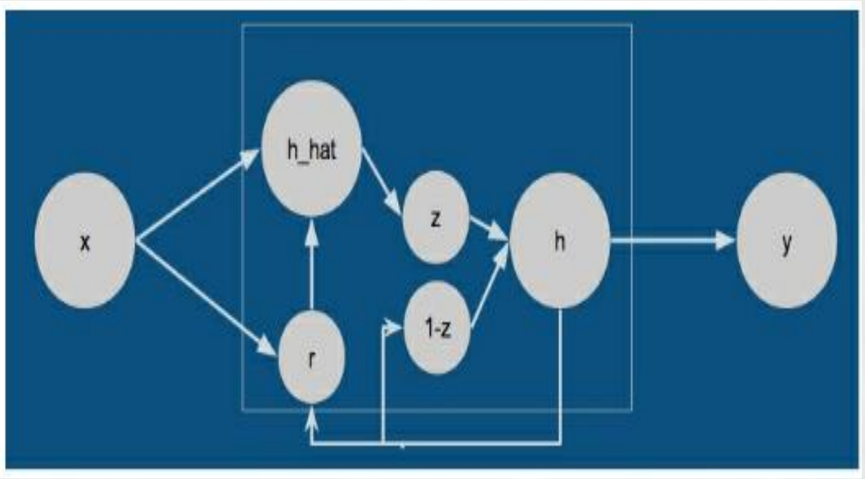
$$h\_hat(t) = f(x(t)W_x + h(t-1)W_h + b_h)$$

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

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

- **Z(t) is called the “rate”**

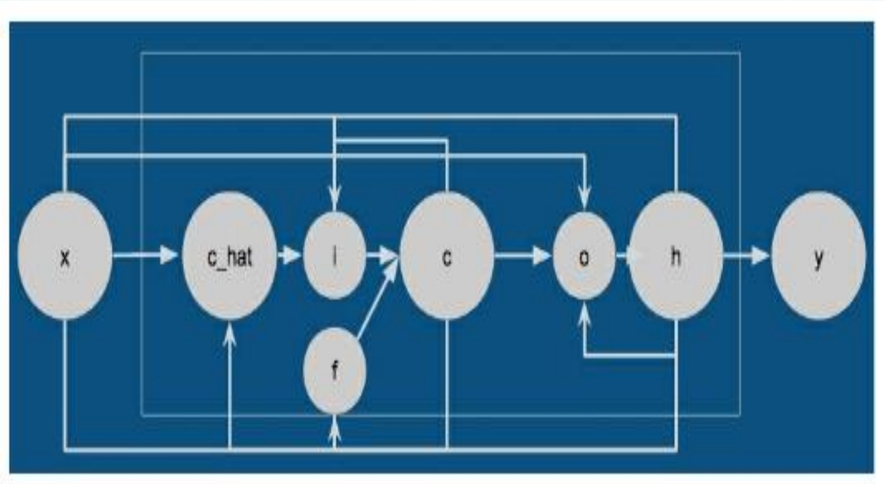
Gated Recurrent Neural Networks (GRUs)



- 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.

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

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.

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

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.

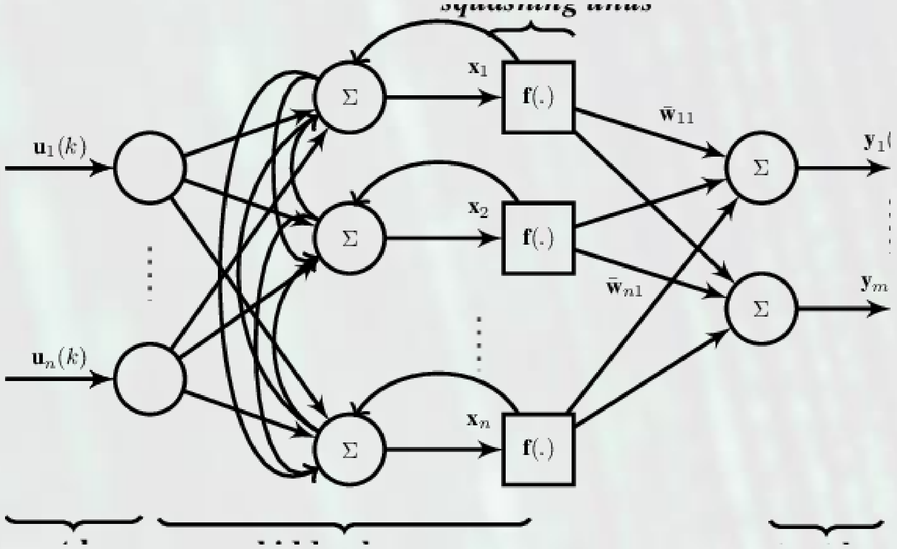
$$MAPE = \frac{\sum_{t=1}^{N}\left|\frac{y_t - y_t^*}{y_t}\right|}{N}$$

- **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.

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
```

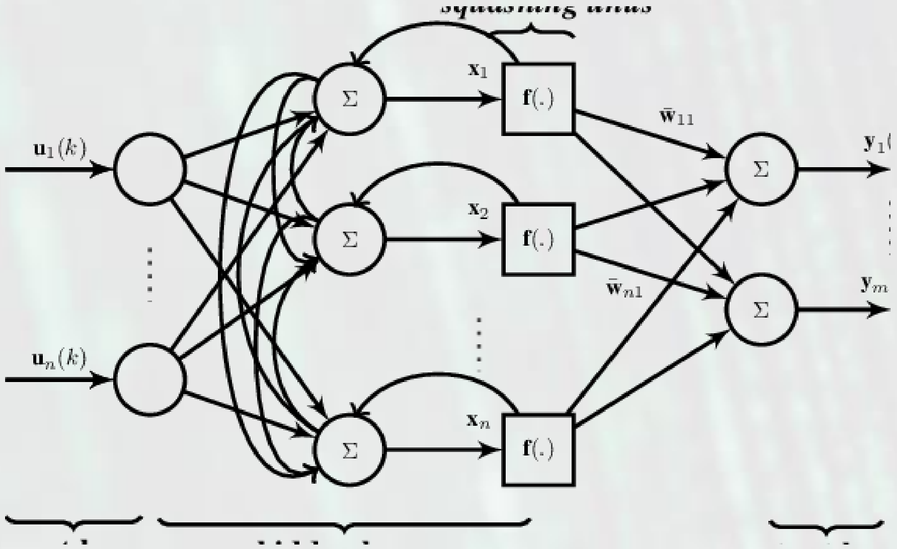
Base class for recurrent layers.

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

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)

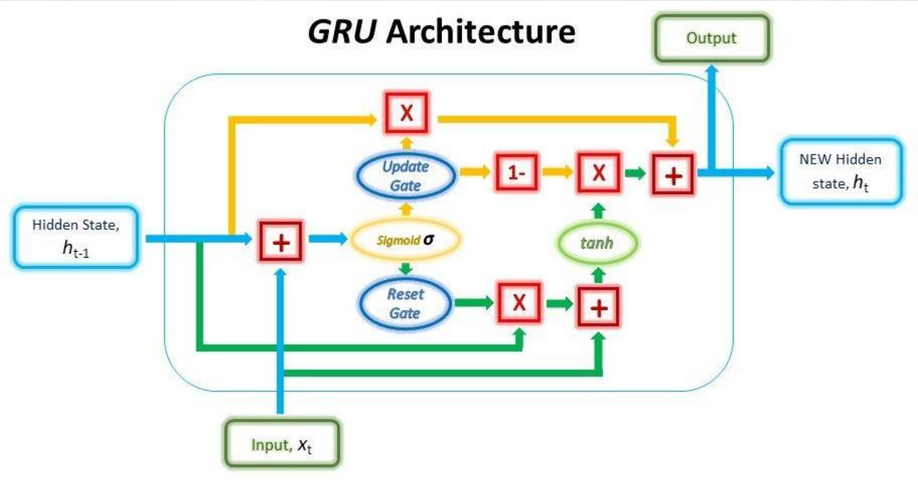
Input shape

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.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.

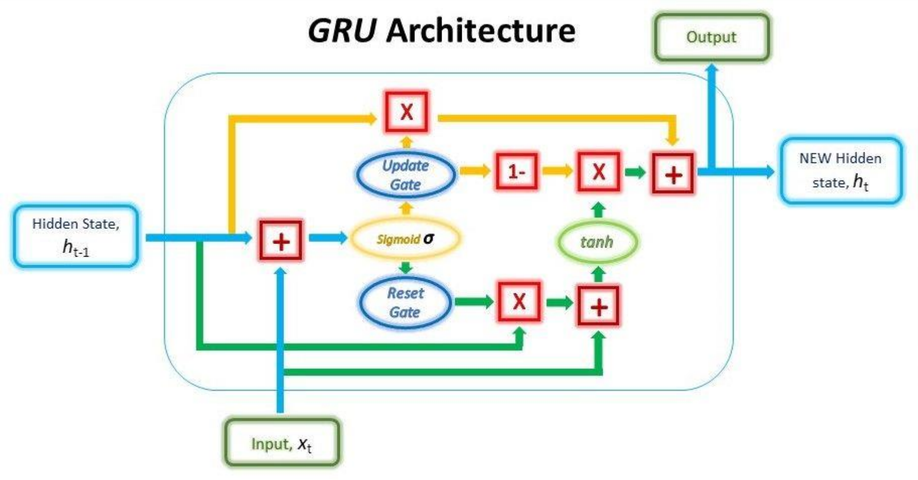
There are two variants. The default one is based on 1406.1078v3 and has reset gate applied to hidden state before matrix multiplication. The other one is based on original 1406.1078v1 and has the order reversed.

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

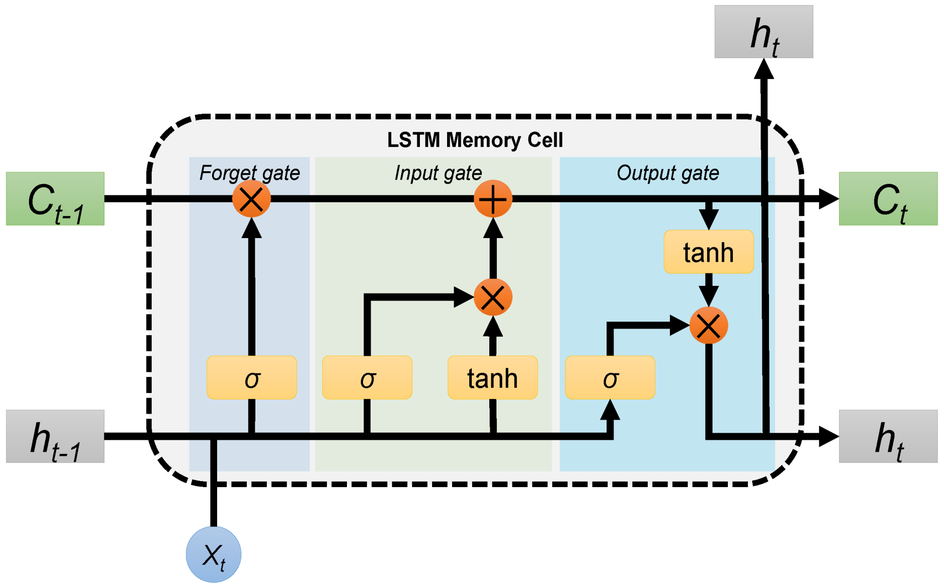
- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use (see activations). Default: hyperbolic tangent (`tanh`). If you pass `None`, no activation is applied (ie. "linear" activation: `a(x) = x`).
- **recurrent_activation:** Activation function 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]

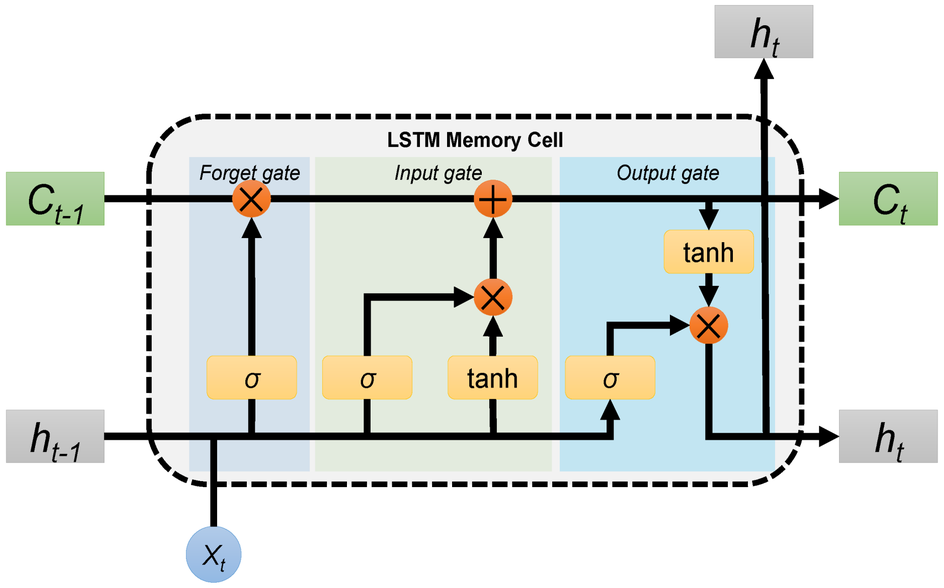
```
keras.layers.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, kernel_initializer
```

Long Short-Term Memory layer - Hochreiter 1997.

Arguments

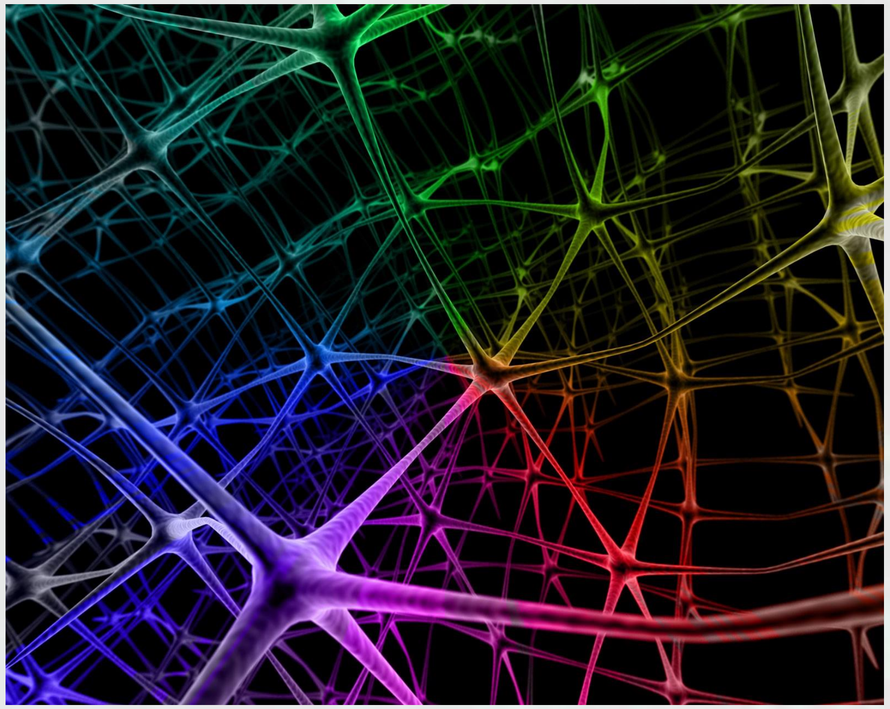
- **units:** Positive integer, dimensionality of the output space.
- **activation:** Activation function to use (see activations). Default: hyperbolic tangent (`tanh`). If you pass `None` , no activation is applied (ie. "linear" activation: `a(x) = x`).
- **recurrent_activation:** Activation function 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.

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
```

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

DEEP LEARNING LESSONS

Deep Learning
Introduction to Recurrent Neural Networks (RNN)

Francesco Pugliese, PhD

neural1977@gmail.com

Thank You