Table of Contents

- 1 Deep learning in Time Series Analysis
  - 1.1 General about Deep Learning in TSA
    - 1.1.1 Fully-Connected network
    - 1.1.2 Convolution neural networks
    - 1.1.3 Recurrent neural networks
    - 1.1.4 Attention based network
      - 1.1.4.1 General about the attention mechanism
      - 1.1.4.2 Self-Attention
      - 1.1.4.3 Query-key-value Attention and Multi-head Attention
      - 1.1.4.4 Transformer block and Transform Architecture

1 Deep learning in Time Series Analysis

1.1 General about Deep Learning in TSA

Disadvantaged of Traditional Machine Learning

- Outliers and Missing values can dramatically affect the performance of the models.
- Classical ML are not able to recognize complex patterns in the data.
- Classical ML usually work well only in few-steps forecasts, not in long term forecast.
- Base-line ensemble of methods have comparable to neural-network time of learning.

In opposite to classical ml approaches

Advantages of the neural network:

- ability to approximate arbitrary nonlinear functions;
- ability to handle noise (robustness to noise);
- robustness to outliers and missed values;
- ability to work with irregular and misaligned time series (with irregular and non-uniform time steps);
- good performance on the multivariate inputs with any number of features ;
- ability to perform long-multi-step forecasts.

.

In the complex case the time series classification can be performed using deep learning neural networks.

The main approaches here are:

Fully-Connected network (FCN)

- Multilayer perceptron (MLP, FCNN);
- Force-back FCN;
- Non-linear autoregressive network (nar, narx, narimax);
- Some transformers without attention.

Convolution neural network (CNN)

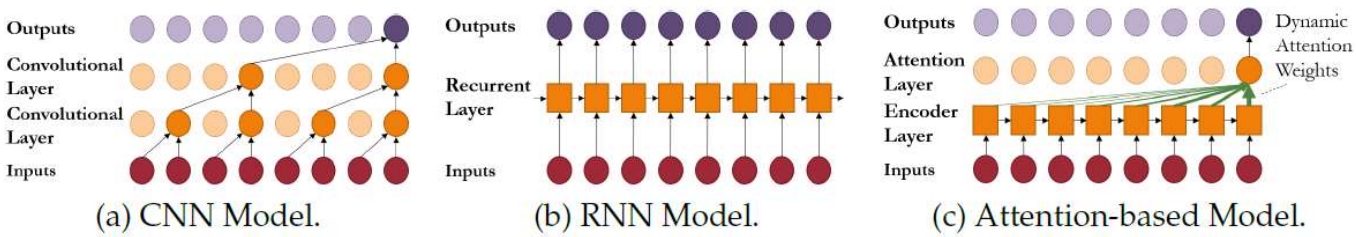
- 2d-conv for multivariate data;
- 2d-conv for data transformed to 2-dimensional (segmentation, sliding, other);
- 1d-conv for sliding windowed dataset;
- 1d-wave-conv (Dilated convolution, casual convolution, Temporal Convolutional Network (TCN)) for sliding windowed dataset;

Recurrent neural network (RNN)

- Simple approach (RNN,LSTM, GRU);
- bidirectional approach (RNN,LSTM, GRU);
- stacked or deep RNN approach (RNN,LSTM, GRU);
- hybrid (Recurrent neural network + CNN) approach;

Attention-based models

- encoder attention (RNN,LSTM, GRU);
- Self-attention (LSTM, GLU, CNN);
- Transformer-based (CNN, FCN);
- hybrid approach (CNN-attention, e.t.c.).

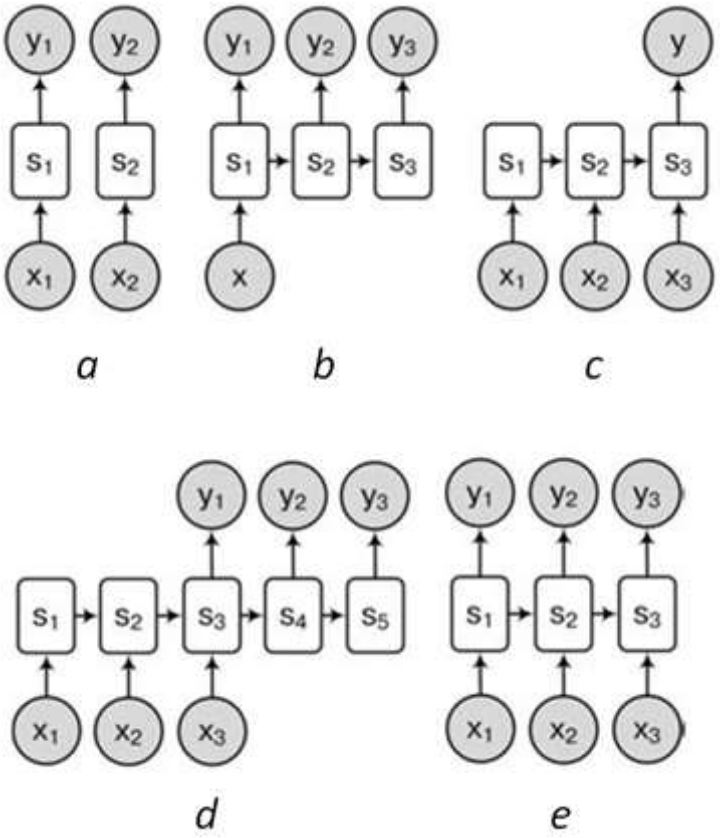


(a) CNN Model. (b) RNN Model. (c) Attention-based Model.

DL time series approaches are

- **One-to-One (a)**: one input and one output. Typical example is the case where you have an full segment (or full time series) as input and you want to predict a single label for the segment.
- **One-to-Many (b)**: a full segment (or full time series) as input and a sequence of outputs. A typical example is an segment on input and its corresponding metrics, parameters or its decomposition.
- **Many-to-One (c)**: sequence of data as input and we have to predict a single output. A typical example where we have an input set of segments (or obtained by sliding window) and we want to predict a single output tag (forecast or label).
- **Sequential Many-to-Many (d)**: a sequence input and a sequence output. For instance, stock prices of 7 days as input and stock prices of next 7 days as outputs.
- **Synchronous Many-to-Many (e)**: a sequence input and a sequence output. For instance, video-processing as time series problem or any on-line problem.

Examples of the approaches for recurrent neural networks.



The most universal approach of Deep learning in time series analysis is Many-to-Many (or **Sequence2Sequence, Seq2Seq**).

Seq2Seq approach can be interpreted as encoder-decoder model.

In this case an encoder is using to summarise past information (i.e. targets, observed inputs and a priori known inputs), and a decoder to make new predictions, classification, estimation and other.

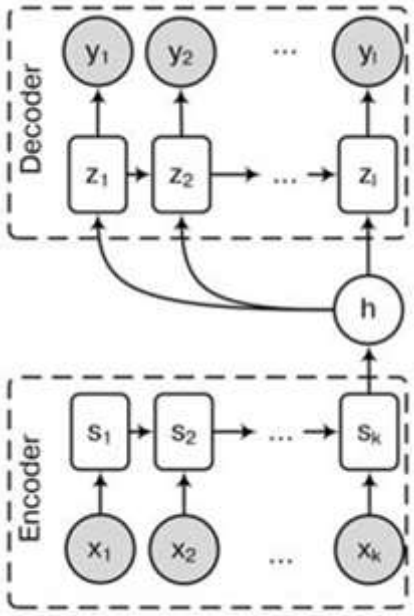
Note

The other popular approach is many-to-one but it could be considered as specific case of Many-to-Many.

Example of encoder-decoder rnn network.

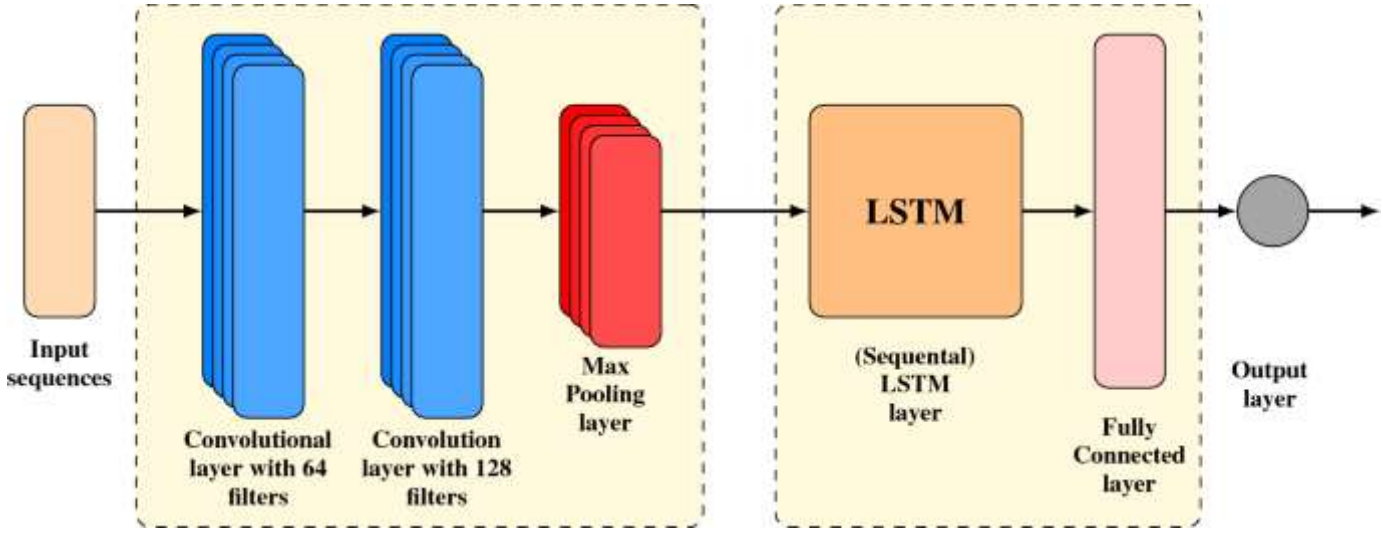
Here *h* denote the hidden (or latency, embedding) space of any fixed dimension.

The meaning of this space is the feature extraction (i.e. summarization of some meaningful information of the whole source sequence).



In general the type of encoder and decoder can be different.

For instance, The CNN encoder and RNN decoder can be joined.



1.1.1 Fully-Connected network

Simple Fully-Connected network approach is not popular in the time series analysis due to the several **drawbacks**:

- Lack of account for the connectivity between segments (obtained by sliding window) - MLP networks does not use similarity and sequential behavior of sequential input data (e.g. in RNN hidden state implement "memory" of sequential segments features).
- Huge amount of parameters (due to the lack of weights re-using like in cnn or rnn).

Note

The drawback are overcome with transform architecture (attention-based architecture based on the full-connected networks).

Force-back FCN approach suppose to the following and previous values to be predicted.

For instance, backward prediction can be used to error calculation and error reduction.

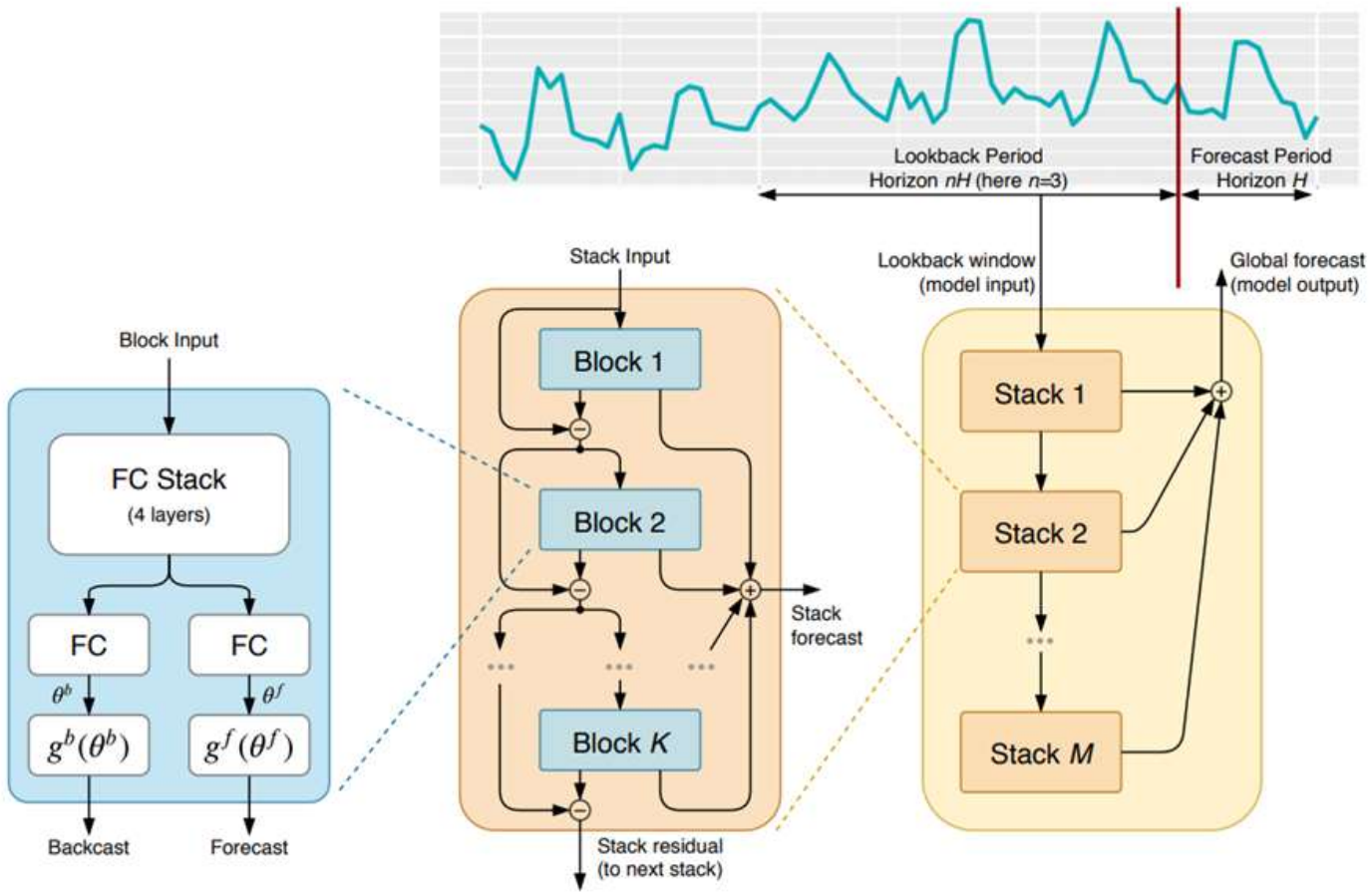
The N-BEATS network propose to use stack of FCN-Force back network each of them predict result

based on the error of the previous one.

Example of the popular **force-back prediction Full-connected network N-BEATS**

The idea under **N-BEATS** is to:

- 1 block - make force and back forecast (in-sample in backward direction and out-of-sample in forward direction).
- calculate error backward prediction.
- 2 block - make prediction of error in force and back direction
- correct forward forecast (made from block 1).
- calculate error of error prediction based on the known value of in-sample forecast.
- repeat stage 2 and clarify out-of-sample forecast.

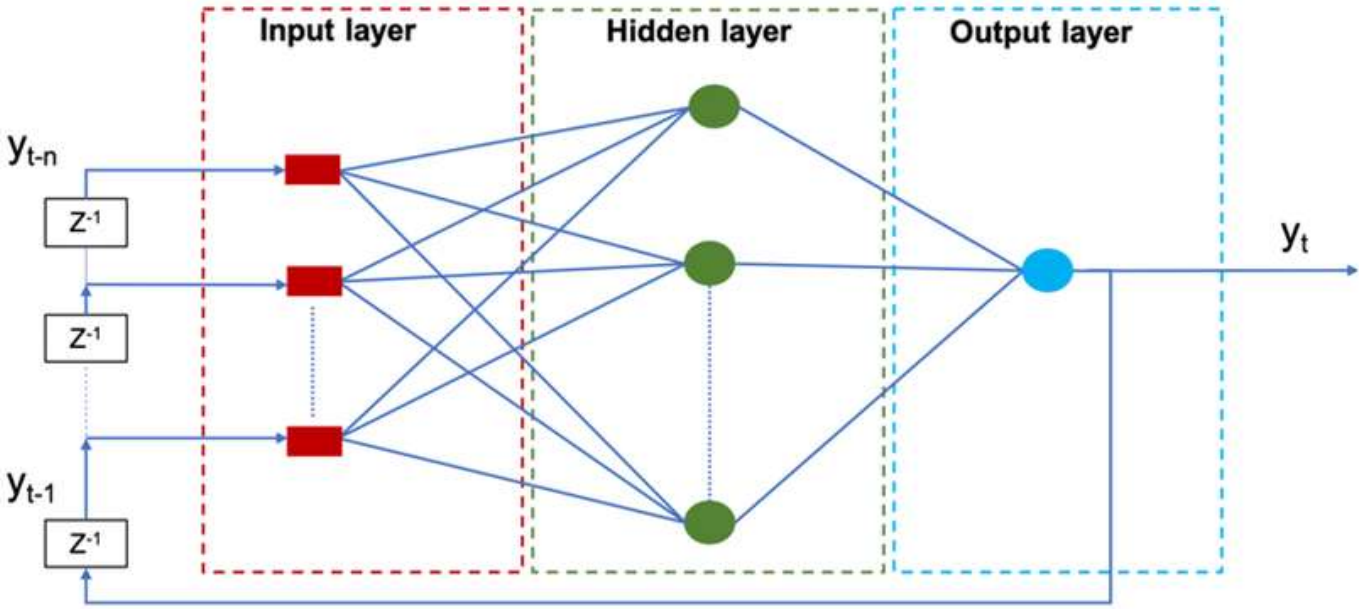


The other FCN approach is using each predicted output (or set of outputs) as part of the input data.

In this case "pseudo-recurrent-connection" can be implemented by feed-back connection.

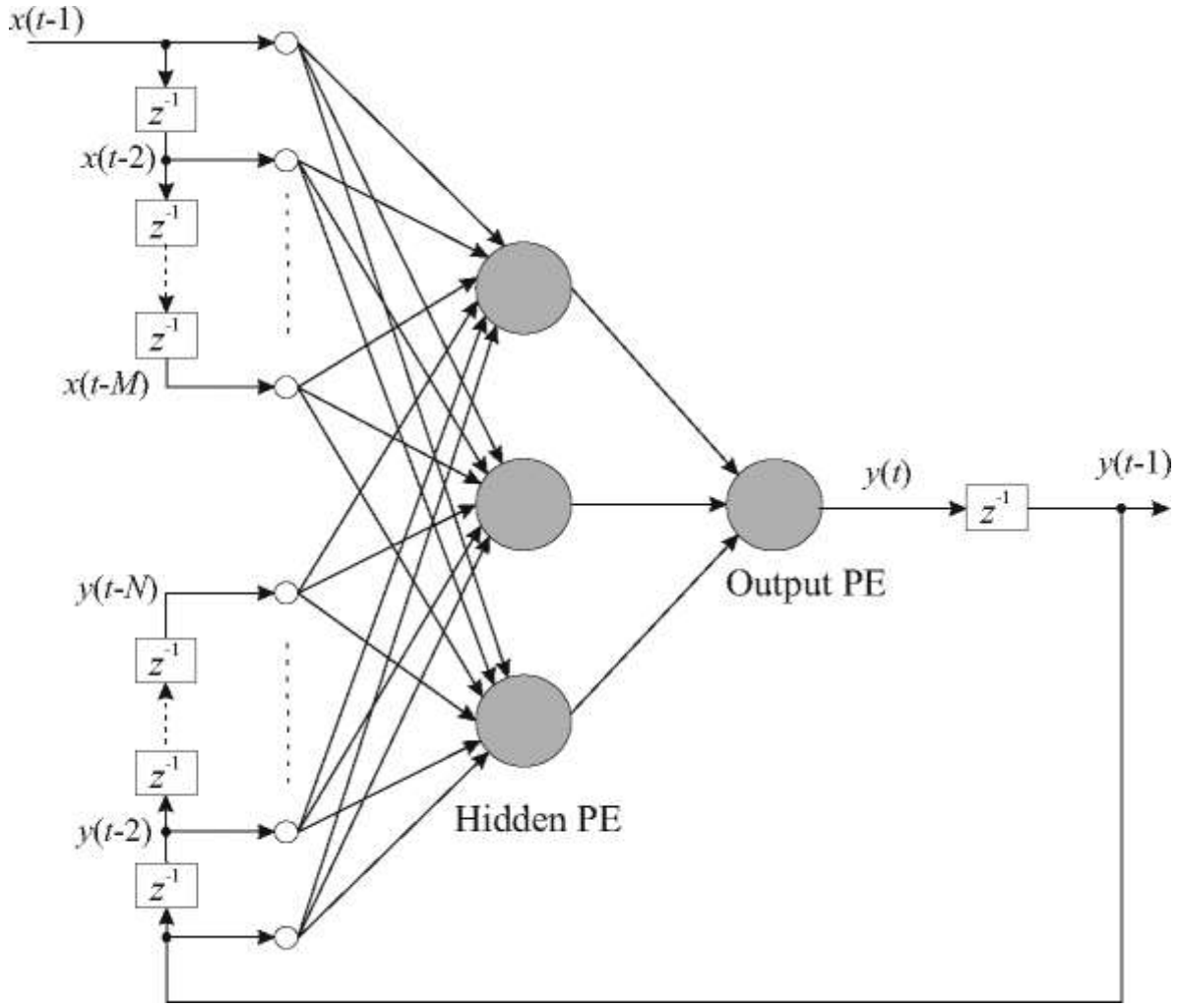
These networks embody the idea of nonlinear autoregressive model (or **non-linear autoregressive network, NAR**), where weights are the autoregressive parameters.

Example of non-linear autoregressive network.



If in non-linear autoregression beside of re-using predictions as inputs the other data are using the approach can be called **non-linear autoregressive with exogenous factors (NARX) network**.

Example of non-linear autoregressive with exogenous factors (NARX) network.



NAR and NARX approaches are not popular up-today , due to low robustness to output problems (outliers, gradient explosion and e.t.c.).

1.1.2 Convolution neural networks

The most popular approach in neural network application in time series is convolution networks.

In many cases of the only **1-dimensional convolution** is necessary (due to 1-dimensional representation of time series).

The 1d convolution can be described as

$$f(x) = W \cdot X_{lag},$$

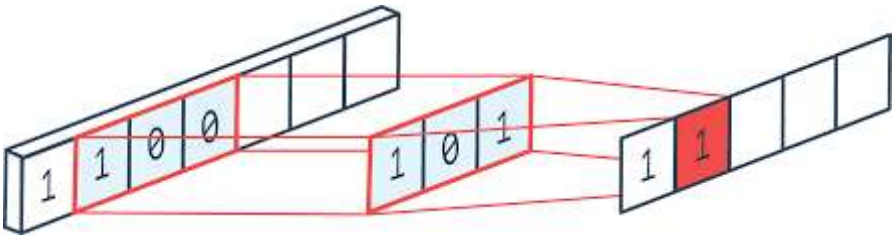
where:

- W is the weights matrix, $W = (W_0, W_1, \ldots W_{K-1})$, K is the kernel size.
- X_{lag} is the so-called convolution or lag matrix of input x = (x_0, x_1, ...x_{N-1}),

  N is the input size,

$$X_{lag} = \begin{bmatrix} x_0 & x_1 & \ldots & x_{N-2} & x_{N-1} \\ x_1 & x_2 & \ldots & x_{N-1} & 0 \\ \ldots & & & & \\ x_{K-1} & x_K & \ldots & 0 & 0 \end{bmatrix}$$

  Example of 1-d convolution



Note

- The conv-1d operation corresponds to the to finite impulse response (FIR) filters in digital signal processing.
- The filter kernel size K corresponds to the receptive field size in classical CNN approach.
- In line with the spatial invariance assumptions for standard CNNs 2d CNN for image), temporal CNNs (1d CNN for time series) assume that relationships are time-invariant – thus it is possible to using the same set of filter weights at each time step and across all time.
- The filter kernel size needs to be tuned carefully to ensure that the model can make use of all relevant historical information.
- It is worth noting that a single causal CNN layer is equivalent to an auto-regressive (AR) model.

The 1d convolution can be described as

$$r = k * x = k \cdot \vec{x},$$

where:

- k is the weights vector $k = (k_0, k_1, \ldots, k_{K-1})$, K is the kernel size.,
- x is the input vector, $x = (x_0, x_1, \ldots, x_{N-1})$, N is the input size, and $\vec{x} = x^T$,
- k is the convolution matrix.

$$k = \begin{bmatrix} k_0 & k_1 & \ldots & k_{K-1} & 0 & \ldots & 0 & \ldots & 0 \\ 0 & k_0 & \ldots & k_{K-2} & k_{K-1} & \ldots & 0 & \ldots & 0 \\ \ldots & & & & & & & & \\ 0 & 0 & \ldots & 0 & 0 & \ldots & k_0 & \ldots & k_{K-1} \end{bmatrix}$$

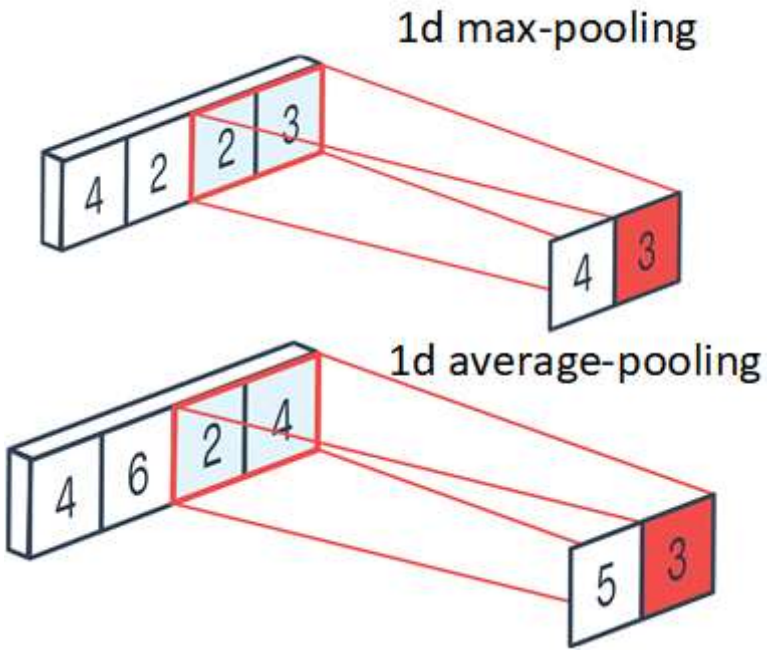
$$r = k * x = k \cdot \vec{x} = \begin{bmatrix} k_0 & k_1 & \ldots & k_{K-1} & 0 & \ldots & 0 & \ldots & 0 \\ 0 & k_0 & \ldots & k_{K-2} & k_{K-1} & \ldots & 0 & \ldots & 0 \\ \ldots & & & & & & & & \\ 0 & 0 & \ldots & 0 & 0 & \ldots & k_0 & \ldots & k_{K-1} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

$$= \begin{bmatrix} k_0 \cdot x_0 + k_1 \cdot x_1 + \cdots + k_{K-1} \cdot x_{K-1} \\ k_0 \cdot x_1 + k_1 \cdot x_2 + \cdots + k_{K-1} \cdot x_K \\ k_0 \cdot x_2 + k_1 \cdot x_3 + \cdots + k_{K-1} \cdot x_{K+1} \\ \vdots \\ k_0 \cdot x_{n-1-K} + k_1 \cdot x_{n-K} + \cdots + k_{K-1} \cdot x_{n-1} \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ \vdots \\ r_{n-1} \end{bmatrix}$$

Example of 1-d convolution

> *Note*:
>
> The filter kernel size K corresponds to the receptive field size in classical CNN approach.

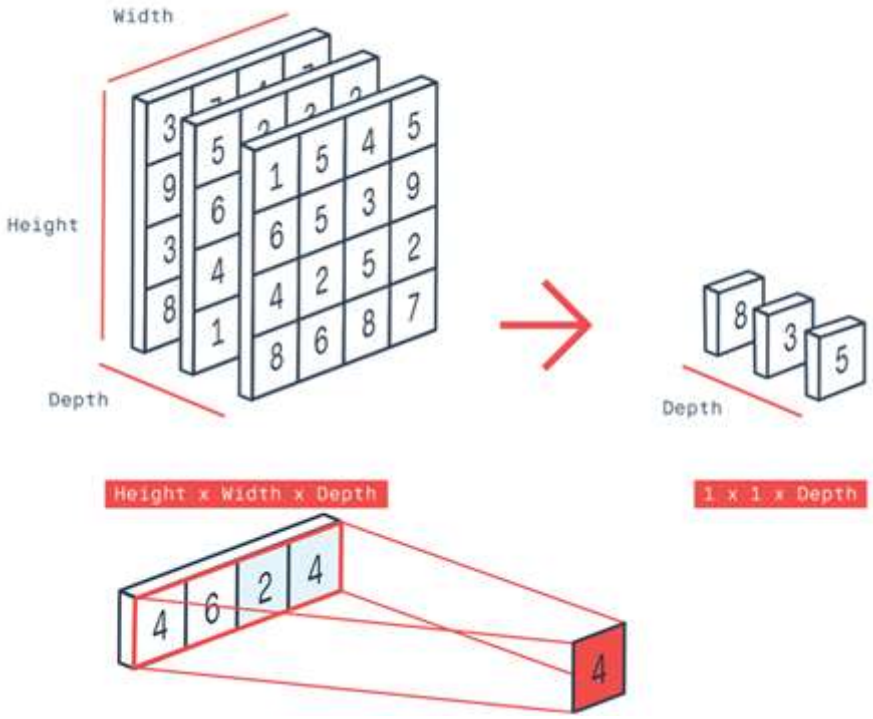
Beside the 1d convolution it also could be **1-d pooling**.



Beside the 1d convolution it also could be **global pooling**.

The global average pooling is used everywhere in 1d-cnn, however you can try to use as global max pooling as flatten and other layers instead of them.

Example of global-average pooling (is the alternative to flatten layer).



Example of 1-d convolution network with input drop-out

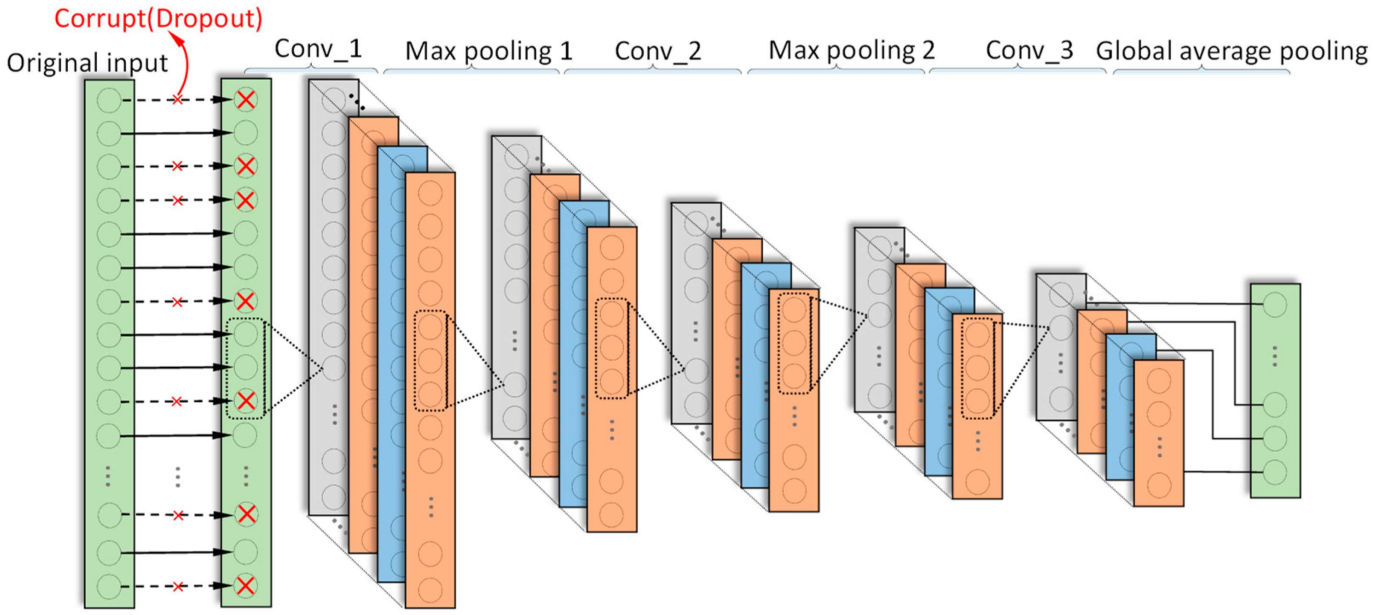
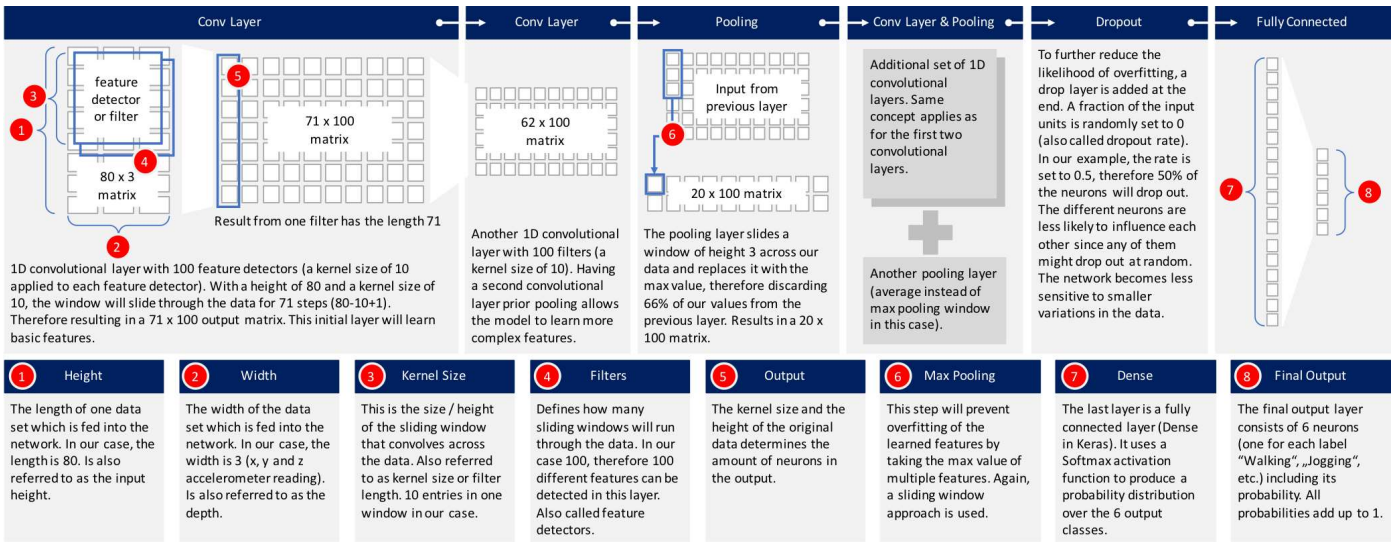


Illustration of the of 1-d convolution network for multi-dimension data with explanation.

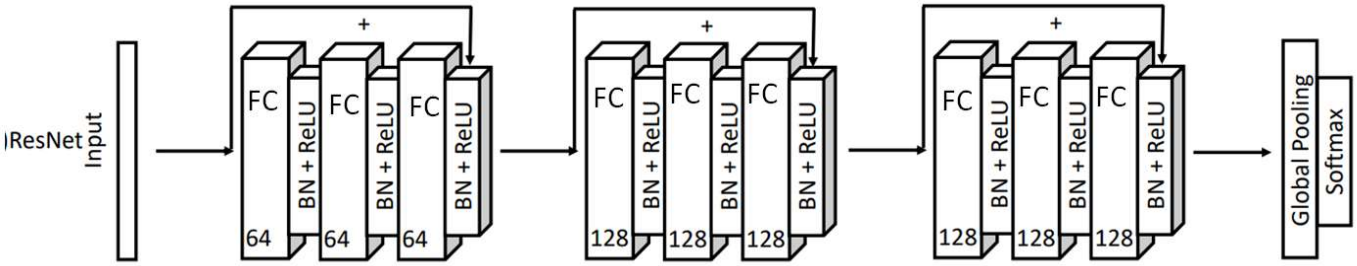


Note

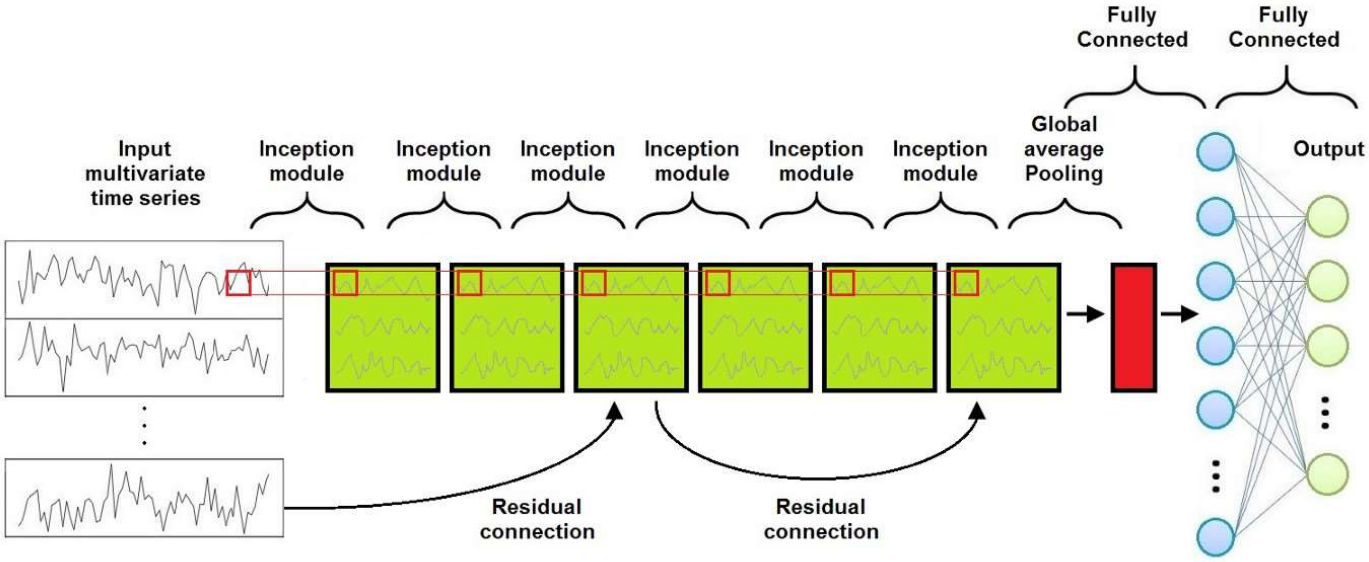
The most of the **regularization approaches** from for 2d-convolutions, are applicable for 1-d time series networks, such as:

- dropput,
- skipped-connections,
- batch-normalization,
- 1x1-convolutions and other .

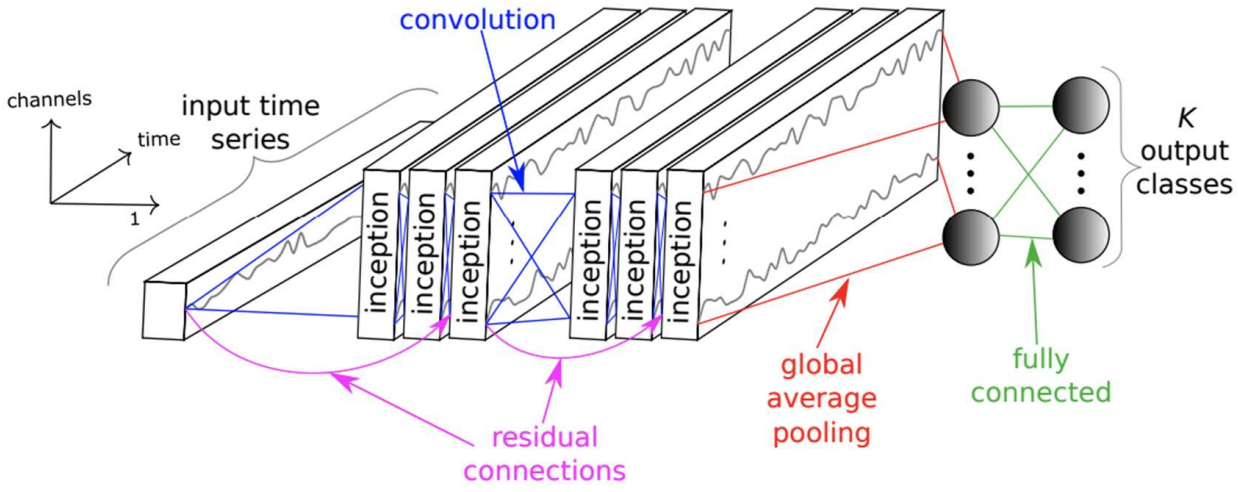
Example of **1d ResNet** fo time series classification.

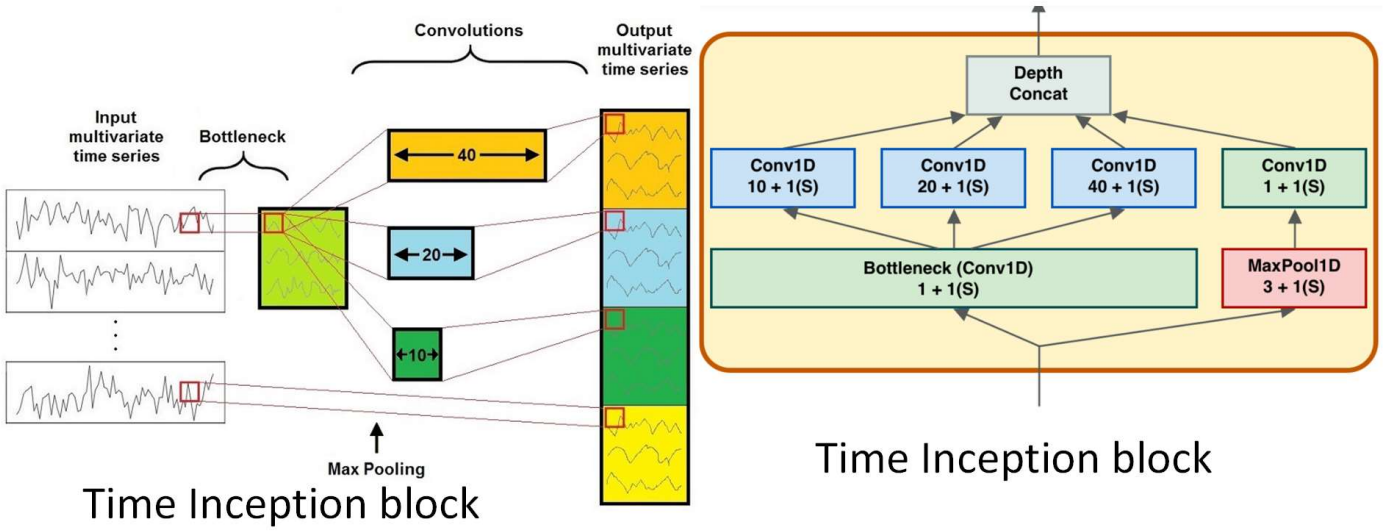


Example of **1d multi-variate ResNet** for time series classification.



Example of **1d InceptionTime network** for time series classification.



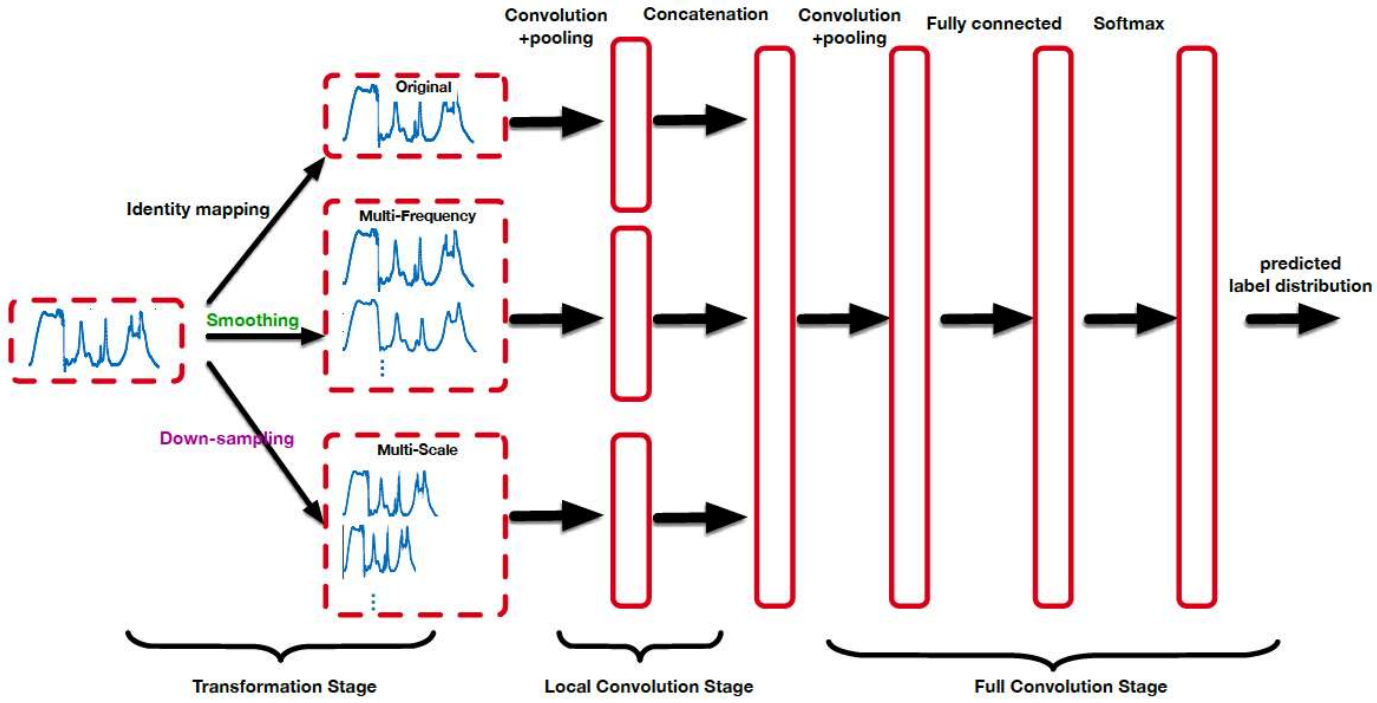


In some cases The network input can be formed by using some feature extraction from the time seres segment.

For instant, the data itself can be combined with its spectrum, decomposition and other transformations.

In this case so-called local convolutions can be applied as the first layer.

Example of **feature-selection for local-convolution layer.**



Dilated Convolutions

Standard 1d-convolutional layers can be computational challenging where long-term dependencies are significant.

Due to the number of parameters scales directly with the size of the receptive field.

To overcame the problem of receptive field size, and handle long-term dependencies frequently the dilated convolution layers is applied.

The dilated convolution allow to increase the receptive filed without breaking the sequence of the time series but by increasing the depth of network. In the simple causal convolution on the plot below, you can see that only the 5 most

recent timesteps can influence the highlighted output.

The dilated convolutions, which allow the receptive field to increase exponentially as a function of the convolution layer depth.

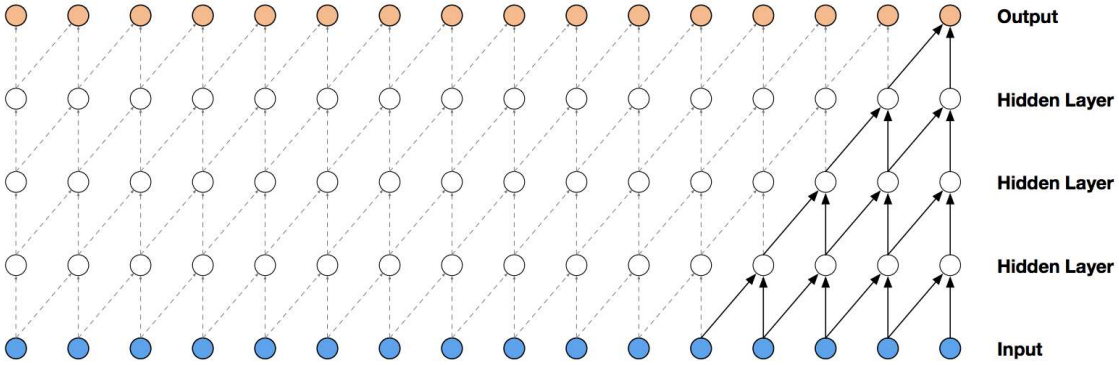


Figure 2: Visualization of a stack of causal convolutional layers.

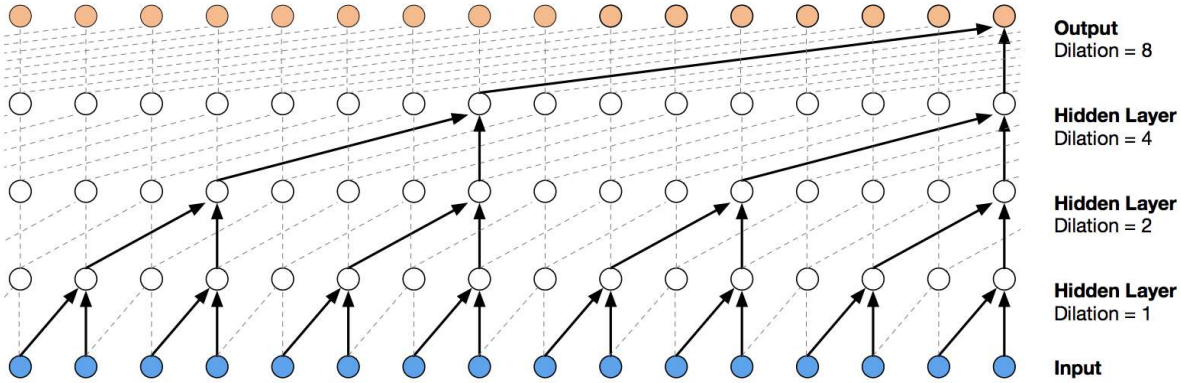
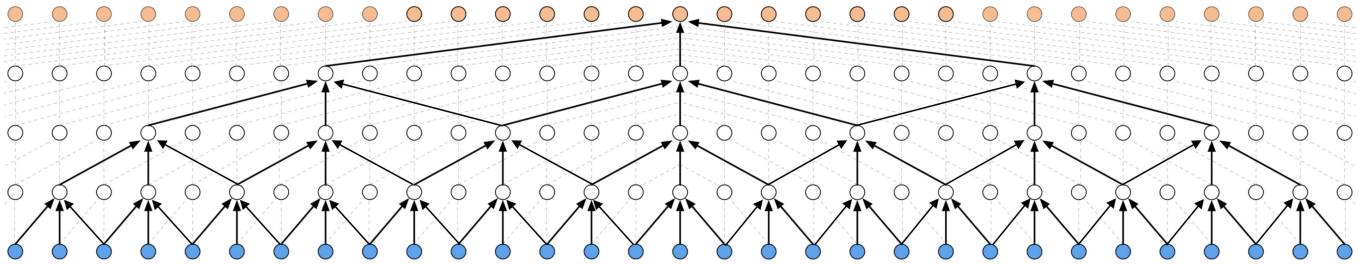


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Beside the the dilated casual convolution it can be **non-casual dilated convolution,** obtained by bi-directional sliding of convolution kernels.

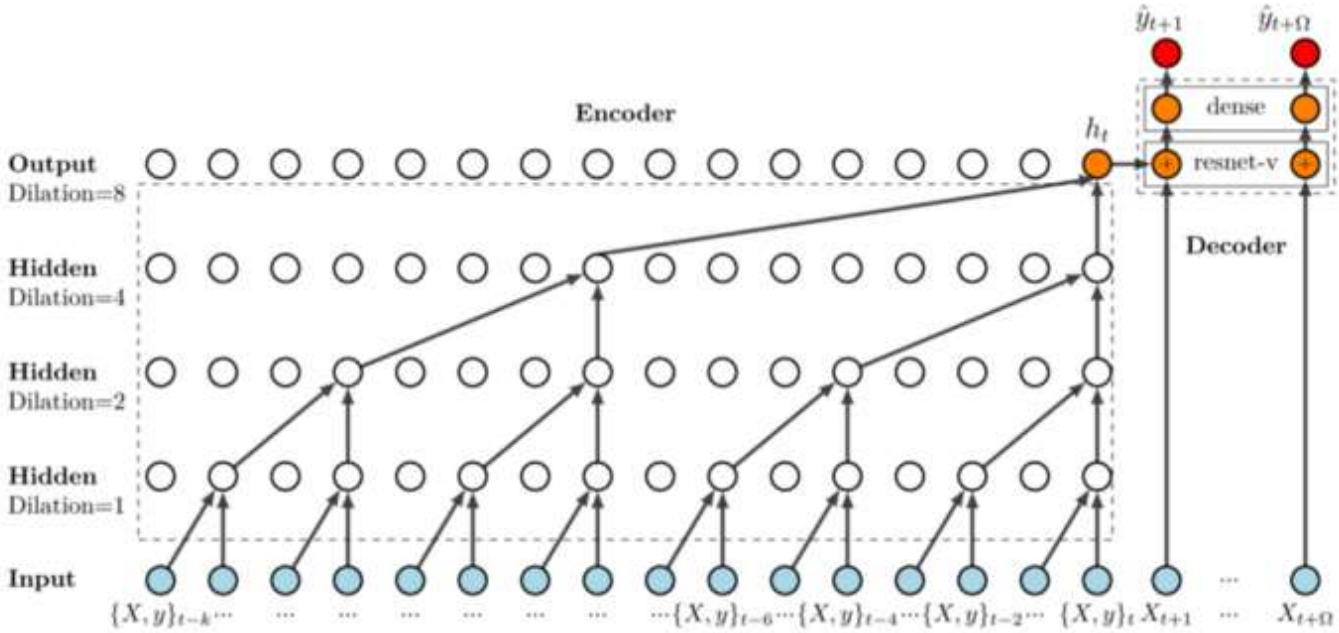


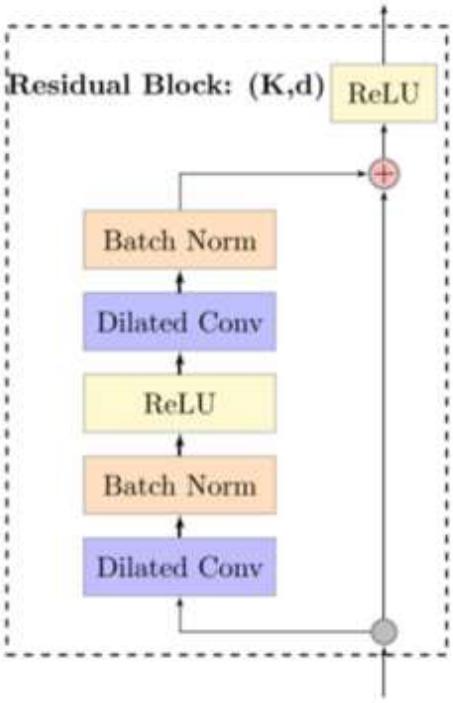
The example of modern implementation of the dilated convolution is the

Deep-temporal convolution network (deepTCN).

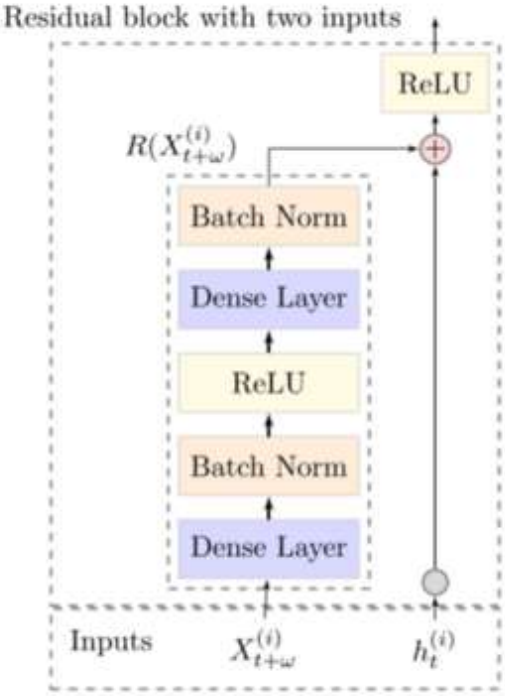
- **Encoder part:** stacked dilated causal convolutions are constructed to capture the long-term temporal dependencies;
- **Decoder part:** a variant of residual block is designed to cooperate both historical covariates and future covariates.

In the picture below X are the exogenous factors, y are the inputs; h is the hidden state and k is the full receptive field.



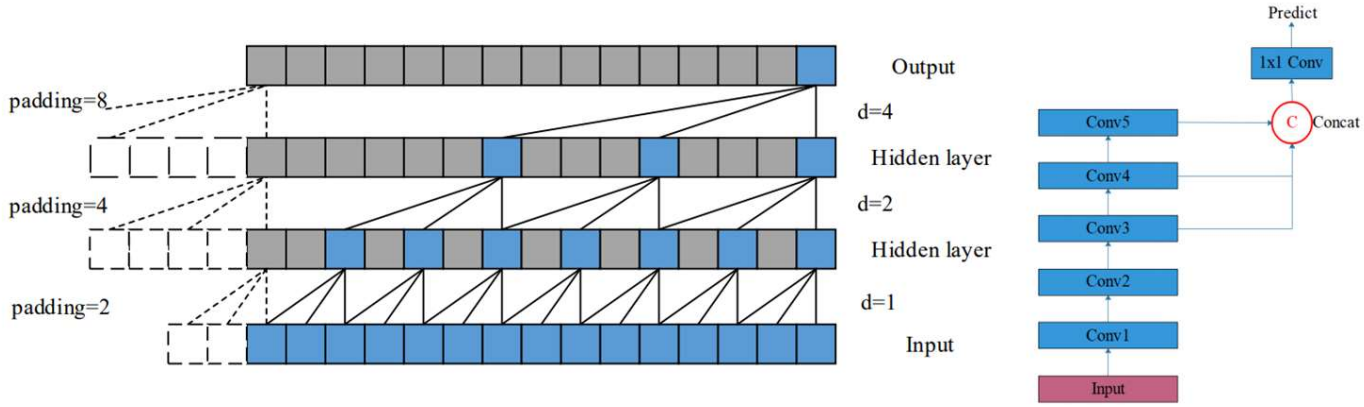


(b) Encoder module



(c) Decoder module

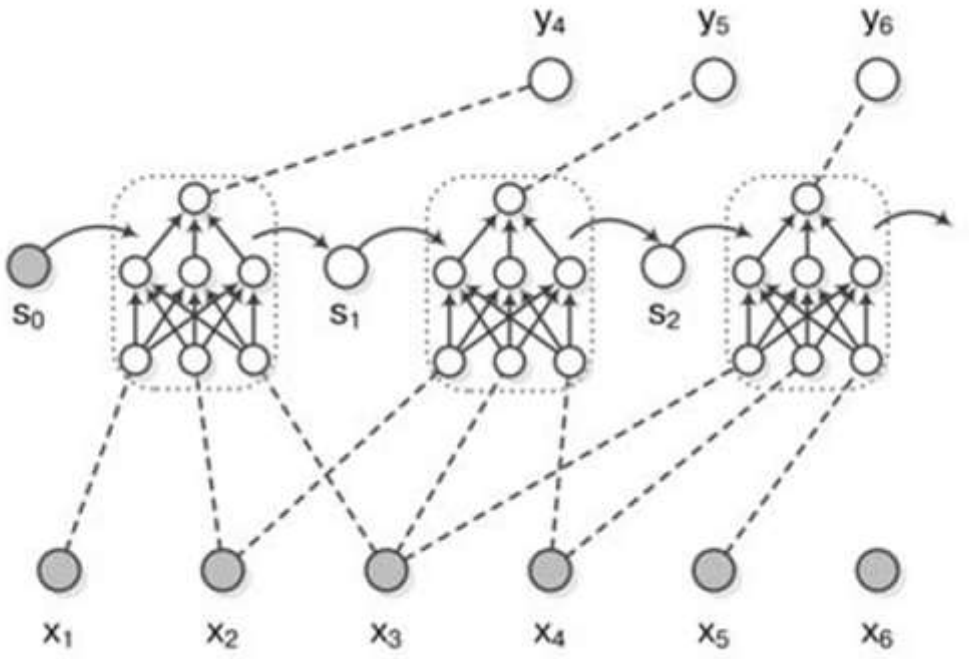
Example of multi-scaled dialed convolution prediction



1.1.3 Recurrent neural networks

Recurrent neural networks (RNNs) have had a historically been used in sequence modeling, with strong results on a variety of tasks. The traditional application of RNN is natural language processing. However in recent years many RNN-based architectures have been developed for temporal forecasting applications. The mentioned above popularity in sequence processing is due to is RNN cells contain an internal memory state which acts as a compact summary of past information.

Example of RNN work for time series many-to-one problem (three inputs and 1 out-put) with some hidden state s



The one of the **main problem of simple RNN (vanilla RNN)** is the **weights (or gradient) explosion and vanishing**,

when long sequence is under processing.

Thus only some part of the series can be processed as input of RNN.

The solutions of this simple RNN problem are:

- truncation of the input size (truncated back propagation).
- bi-directional learning (if it possible).
- dilated temporal learning.
- using of advanced recurrent networks (LSTM, GRU and other).
- regularization of learning (dropout, bath-norm, L1,L2 reg, gradient (or it norm) restrictions, self-normalization activation-functions and e.t.c.).

Note

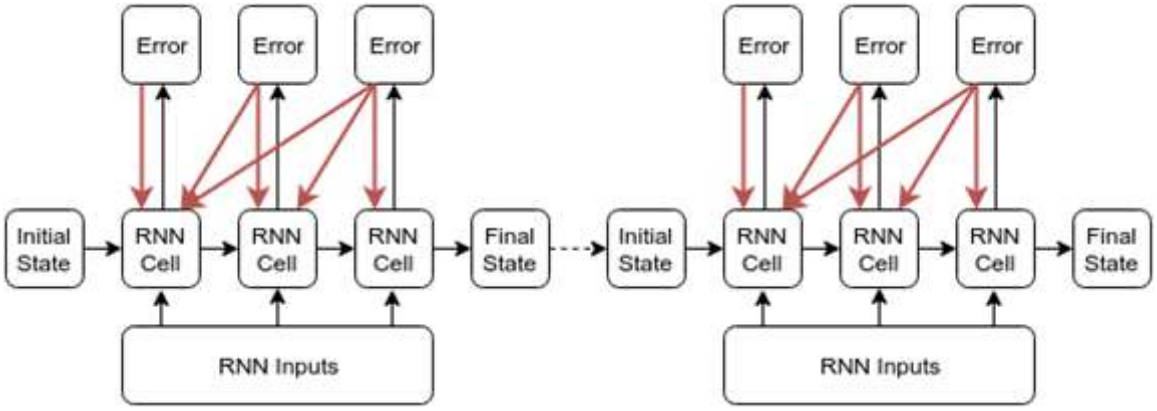
- An error gradient by the definition is the direction and magnitude calculated during the training of a neural network that is used to update the network weights in the right direction and by the right amount.
- In deep recurrent neural networks, error gradients can accumulate during an update and result in very large gradients.

  Large gradients in turn result in large updates of the network weights, and in turn, an unstable network.

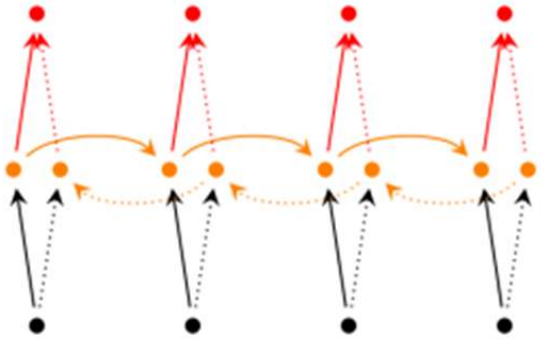
  At an extreme, the values of weights can become so large as to overflow and result in NaN values.
- **The following indicators show the gradient explosion**
  - The model is unable to get traction on your training data (e.g. poor loss).
  - The model is unstable, resulting in large changes in loss from update to update.
  - The model loss goes to NaN during training.
  - The model weights quickly become very large during training.

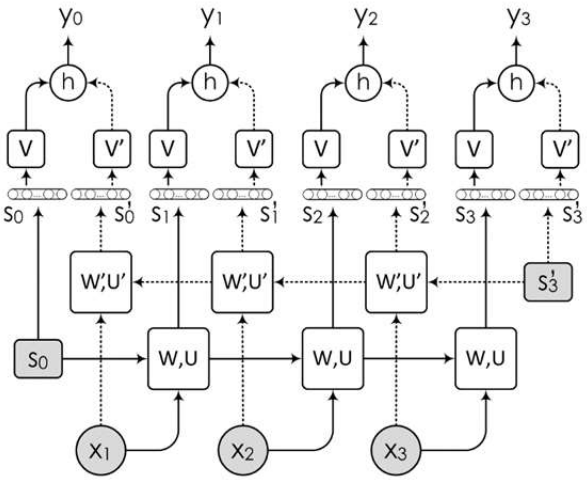
- The model weights go to NaN values during training.
- The error gradient values are consistently above 1.0 for each node and layer during training.

Example of truncated back-propagation

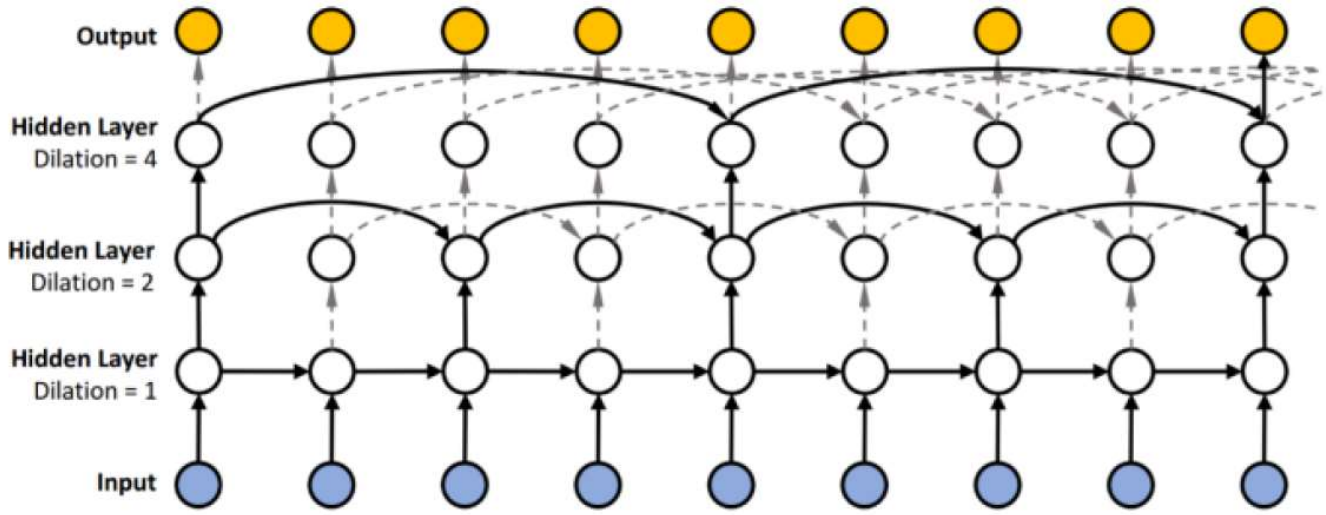


Example of bi-direction learning

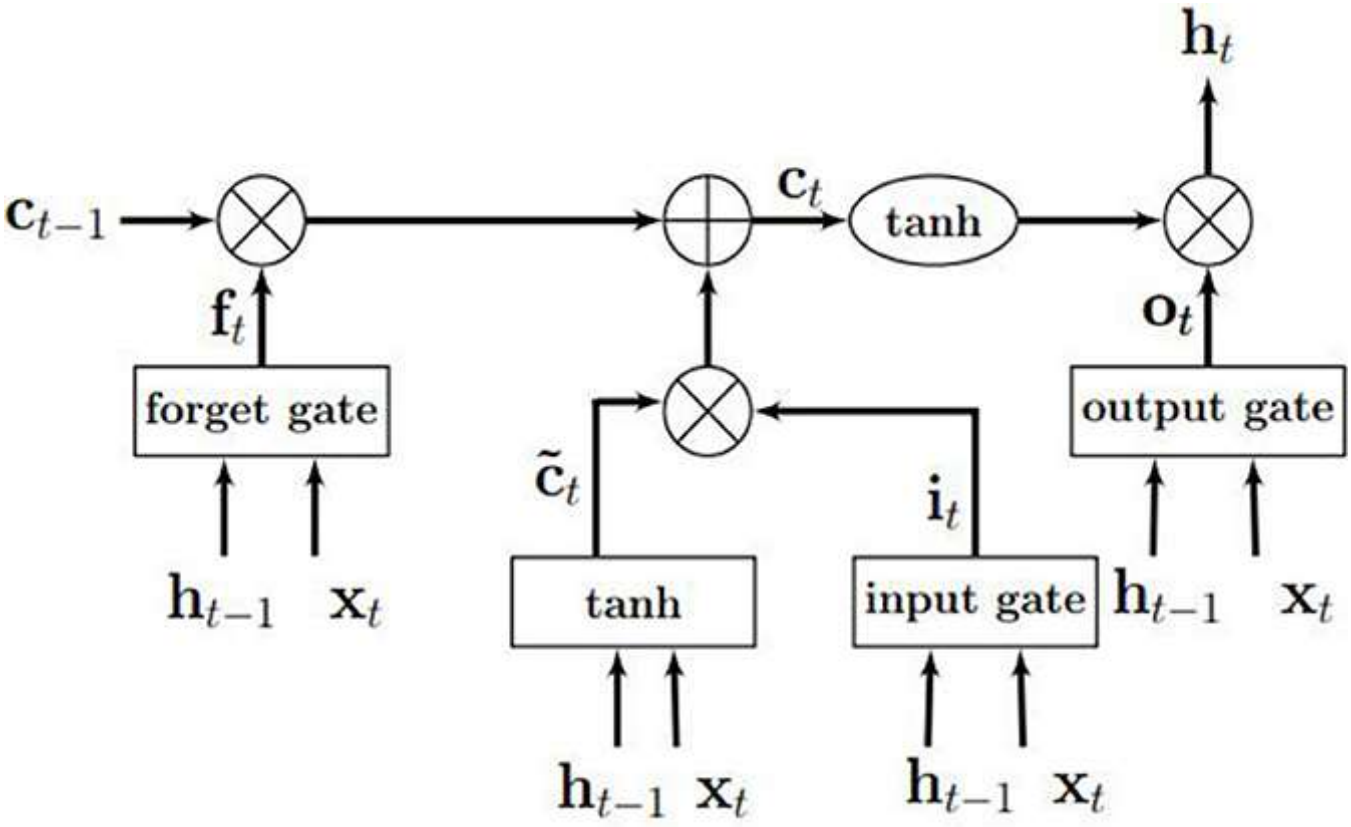


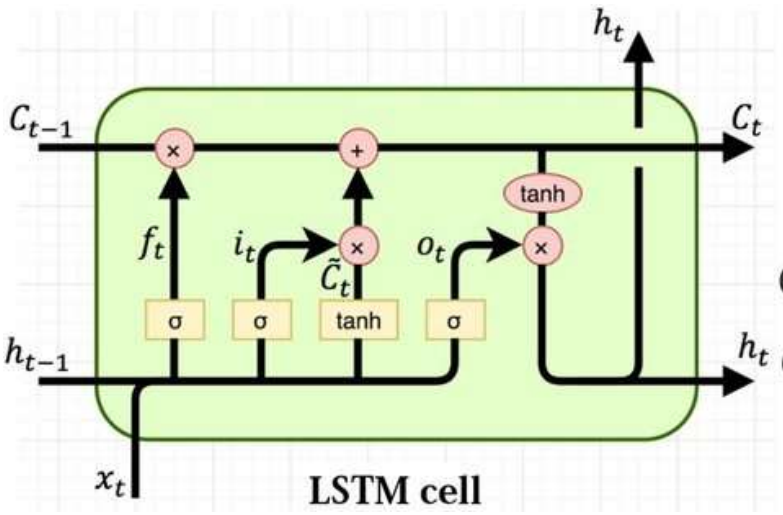


Example of dilated temporal RNN learning



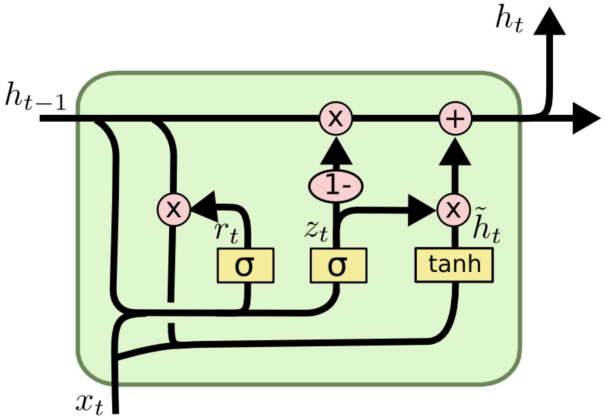
Long Short-Term Memory networks (LSTMs) were developed to address the RNN long-memory limitations. This is achieved through the use of a cell state which stores separate long and short term information, modulated through a series of gates.





$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right)$$
$$\tilde{C}_t = \tanh\left(x_t U^g + h_{t-1} W^g\right)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

The **Gated Recurrent Unit (GRU)** Cell can be considered as alternative for LSTM one



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Autoregression recurrent learning

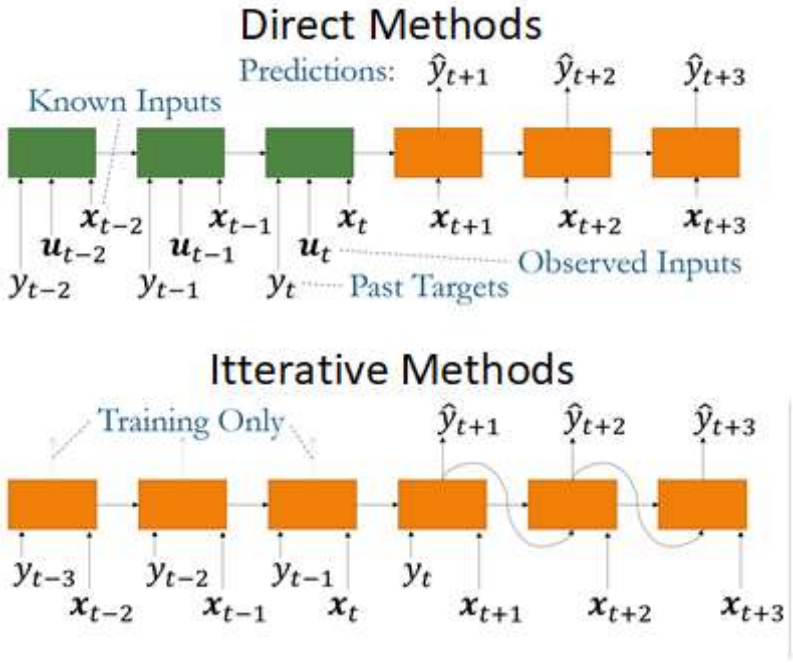
- **Iterative learning** is the development of the naive recurrent learning idea designed for time series application.
- **The main idea in the iterative learning** is to use auto-regression principle in the recurrent network framework.

- In differ with tradition (direct) method in Iterative learning the series is learning to predict on only one step forward on training stage (note that hidden state is also keep passed).
- On the validate stage network use its previous predictions as addition inputs.
- Multi-horizon forecasts producing by recursively feeding samples of the target (one-step forecast) into future time steps (by repeating the inference procedure).

Note

The direct approach (traditional) is typically sequence-to-sequence architectures using an encoder to summarise past information (i.e. targets, observed inputs and a priori known inputs), and a decoder to combine them with known future inputs.

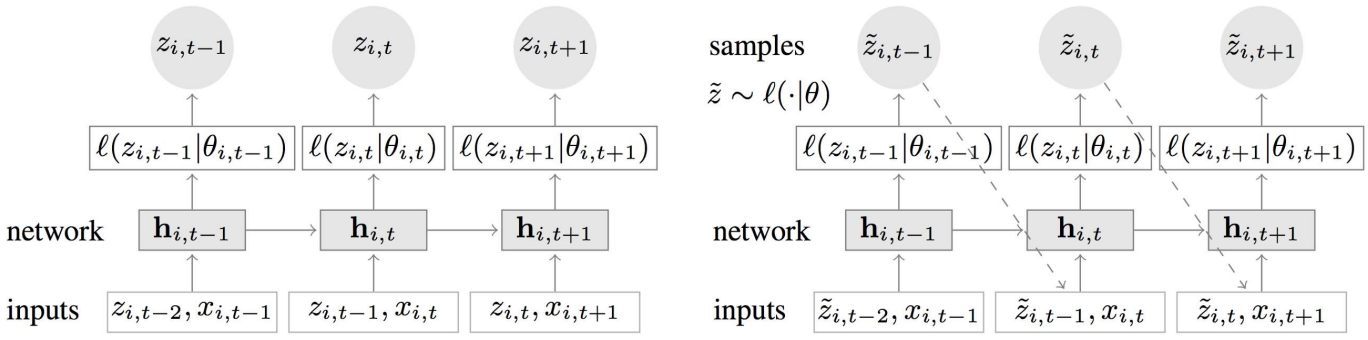
The drawback of direct approach is while avoiding the need for recursion, direct methods require the specification of a maximum forecast horizon, with predictions made only at predefined discrete intervals.



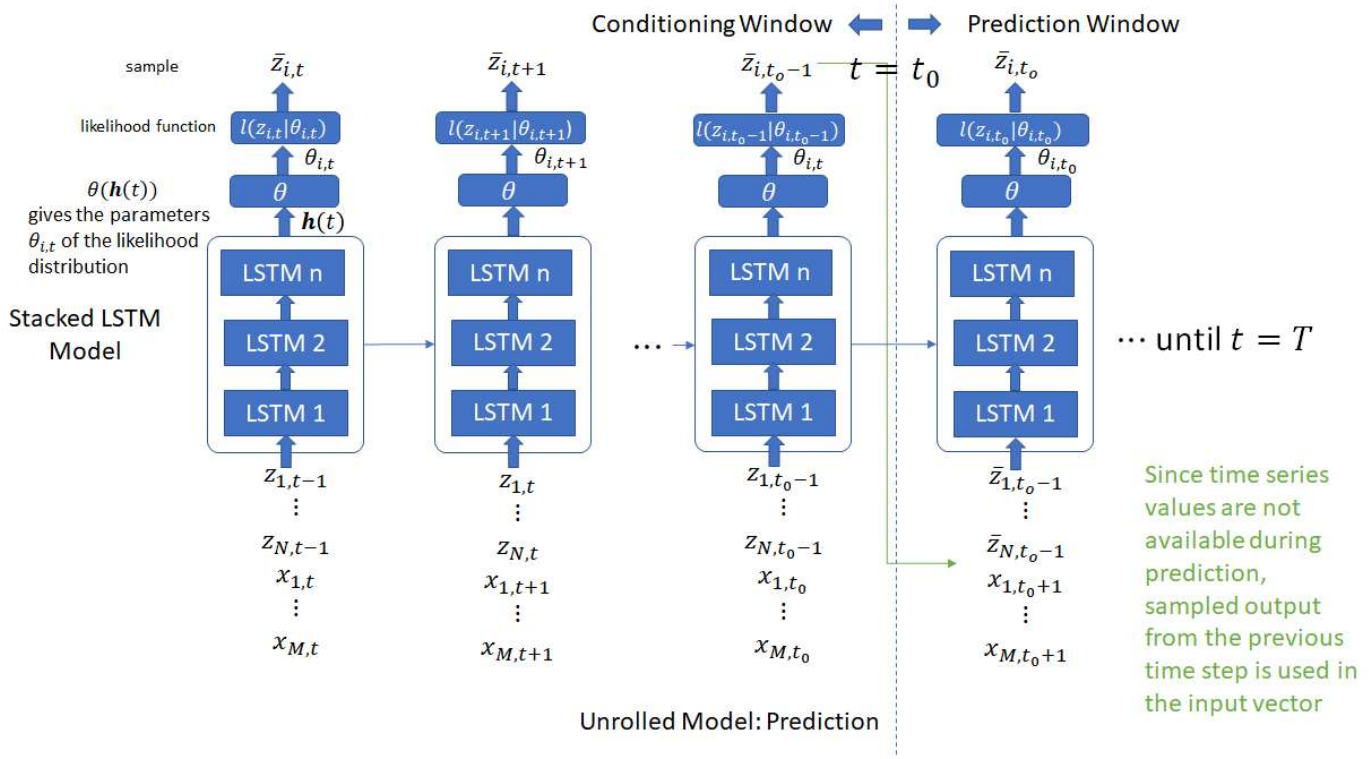
Deep AR

The Deep AR network is the most famous implementation of the iterative learning in combination with lstm cells and so-called **probabilistic supervised learning.**

Official Deep AR architecture representation on training stage(left) and test stages (right) .



Unofficial Deep AR architecture representation.



In the **Probabilistic learning** we are not only try to forecast the one prediction value, but also its variance.

In other words we try to build some distribution and minimize its likelihood (sum of its values).

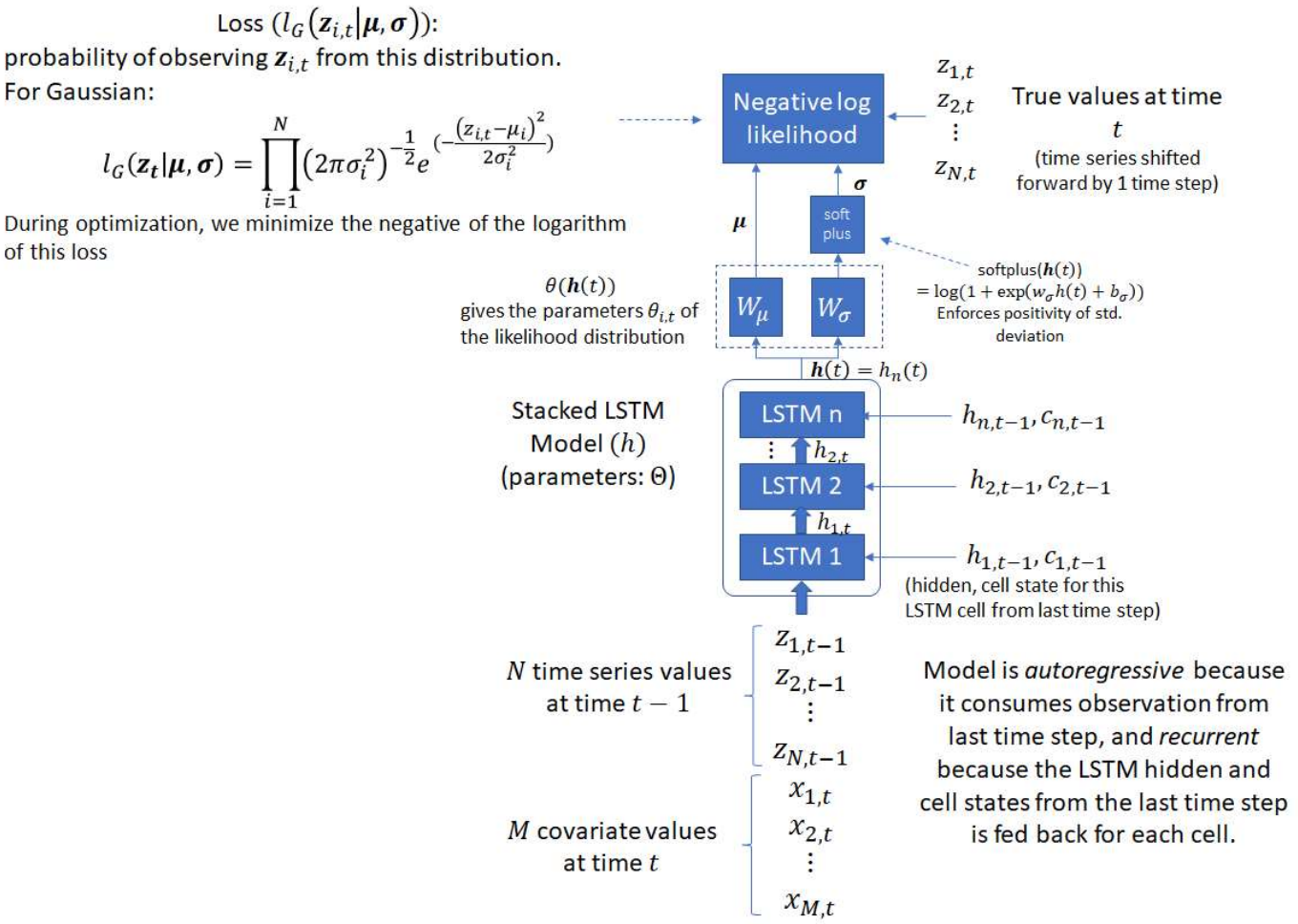
In the **Probabilistic iterative learning**

- On the **training stage** the goal is to predict at each time step the following one value and its distribution parameters (μ and α-variance for Gaussian distribution, horizon=1).

  The information is propagated to the hidden layer (h) and up to the loss function (likelihood function, which is a score function).

  The likelihood function can be Gaussian or Negative Binomial.

- On the **validate stage** forward propagation is carried out using input z_{i-1} (along with [optional] covariates or one-hot-encoded categorical features) and obtained on training stage parameters μ and α.



Gaussian likelihood loss function

$$\ell_{\mathrm{G}}(z|\mu,\sigma) = (2\pi\sigma^2)^{-\frac{1}{2}}\exp(-(z-\mu)^2/(2\sigma^2))$$

$$\mu(\mathbf{h}_{i,t}) = \mathbf{w}_\mu^T\mathbf{h}_{i,t} + b_\mu \quad \text{and} \quad \sigma(\mathbf{h}_{i,t}) = \log(1+\exp(\mathbf{w}_\sigma^T\mathbf{h}_{i,t} + b_\sigma))\ .$$

Negative Binomial Likelihood loss function

$$\ell_{\mathrm{NB}}(z|\mu,\alpha) = \frac{\Gamma(z+\frac{1}{\alpha})}{\Gamma(z+1)\Gamma(\frac{1}{\alpha})}\left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}}\left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{z}$$

$$\mu(\mathbf{h}_{i,t}) = \log(1+\exp(\mathbf{w}_{\mu}^{T}\mathbf{h}_{i,t}+b_{\mu})) \quad \text{and} \quad \alpha(\mathbf{h}_{i,t}) = \log(1+\exp(\mathbf{w}_{\alpha}^{T}\mathbf{h}_{i,t}+b_{\alpha}))\ ,$$

Advantages of DeepAR

- ability to training several hundred or thousands of time-series simultaneously, significant model scalability.
- Minimal Feature Engineering.
- The model learns seasonal behavior on given covariates across time series.
- predict on items with little history.
- Variety of likelihood functions: DeepAR does not assume Gaussian noise for data flexibility.

Drawbacks of DeepAR

- Requires the full set of data when making new predictions.
- No model tuning for individual time series.
- Data must be formatted in a particular way recommended by the developer.
- Allow to obtain comparably high performance only for huge joined data set. Thea accuracy for specific data series (or few amount of series) lower then for ARIMA

and ETS.

1.1.4 Attention based network

1.1.4.1 General about the attention mechanism

The attention is the mechanism to improvements in long-term dependency learning in neural-network.

The attention mechanism provides several key benefits.

- allow directly **attend to any significant rare events that occur**.

  In retail forecasting applications, for example, this includes holiday or promotional periods which can have a positive effect on sales.

- **learn regime-specific temporal dynamics** – by using distinct attention weight patterns for each regime.
- **learn controlled-length sequences (quite long sequences)**
- **work as regularization to avoid of gradient vanishing and explosion**
- **allow to exclude recurrent networks (in transformers) by replacing it on positions encoding**

The first attempt to use attention was **Attention for Seq2Seq RNN Model**(encoder-decoder rnn model)

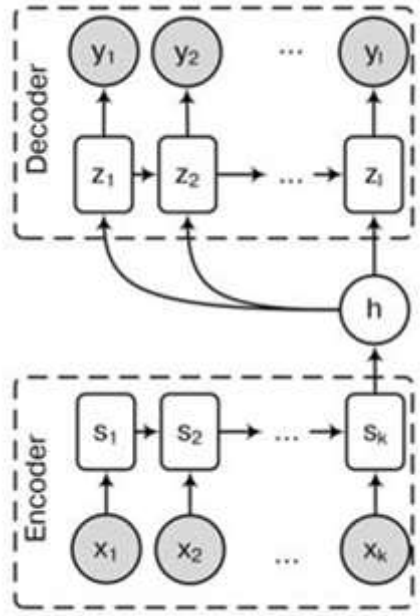
In the simple (or naive) seq2seq implementation the model consists of two part:

- encoder (feature extraction)
- and decoder, which is initialized with the context vector to emit the transformed output.

In the simplest case the last states of the encoder network is used as the decoder additional information state.

however, often it has forgotten the earliest part once it completes processing the whole input.

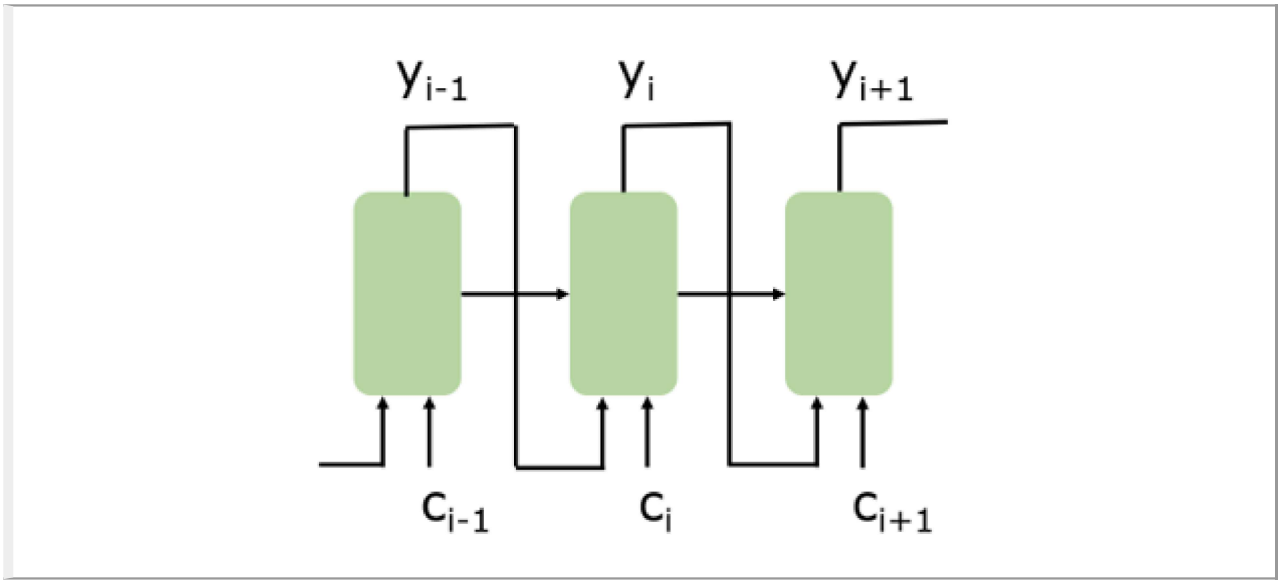
> *Note*
>
> *A potential problem with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector.*



The attention mechanism was designed to help memorize long source sentences in neural machine translation.

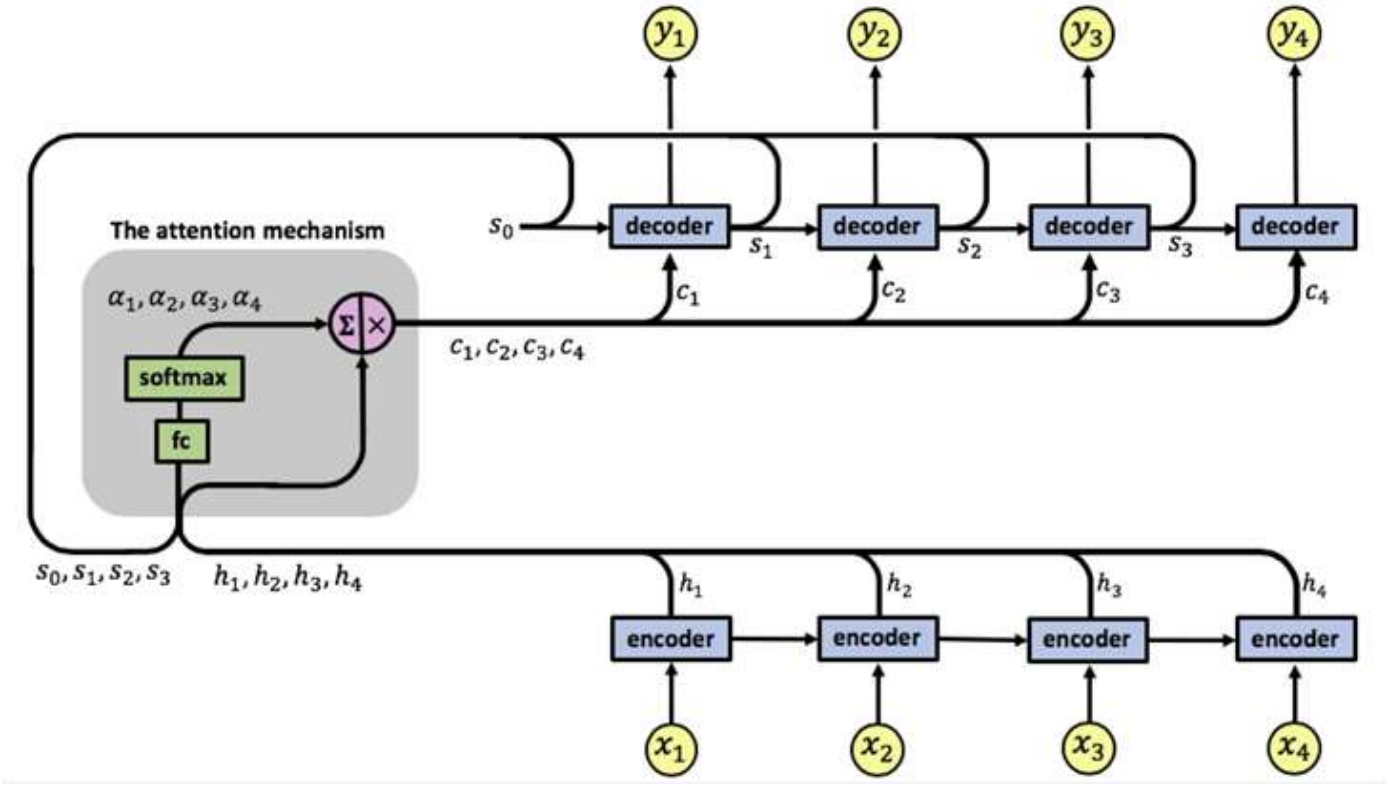
For understanding the attention assume you have:

- a sequential model for predict outputs y_i,
- a sequential model input h_i;
- the h_i can be as y_{i-1} as its combination with other data (in the picture below it is y_{i-1});
- decoder keep its hidden state s_{i-1} and pass it to the each next step for prediction y_i
- in addition decoder has as input context vector c_i that point on the impact factor of the $i-th$ time step.



Let's now considering

Attention Encoder-Decoder (seq2seq) model.



In the original idea it was proposed to using perceptron with soft-max output (shortcut connections c_i) above the hidden space vector of seq2seq model (last encoder layer).

The weights of these shortcut connections a_{ij} are customizable for each output element.

In other words

The attention network (or layer) represents a relation between the hidden vector and the entire input sequence.

In accordance to the original idea the hidden state vector c_t fro the each of the output step t can be calculated as $$ \begin{aligned} \mathbf{c}_t &= \sum_{i=1}^n \alpha_{t,i} \boldsymbol{h}_i & \small{\text{; Context vector for output }y_t} \\\\ \alpha_{t,i} &= \text{align}(y_t, x_i) & \small{\text{; How well two words }y_t\text{ and }x_i\text{ are aligned.}} \\\\ &= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^n \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))} & \small{\text{; Softmax of some predefined alignment score.}} \\\\ \rm and & \\\\ &\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_{a1}\boldsymbol{s}_t+\mathbf{W}_{a2}\boldsymbol{h}_i) \end{aligned} $$

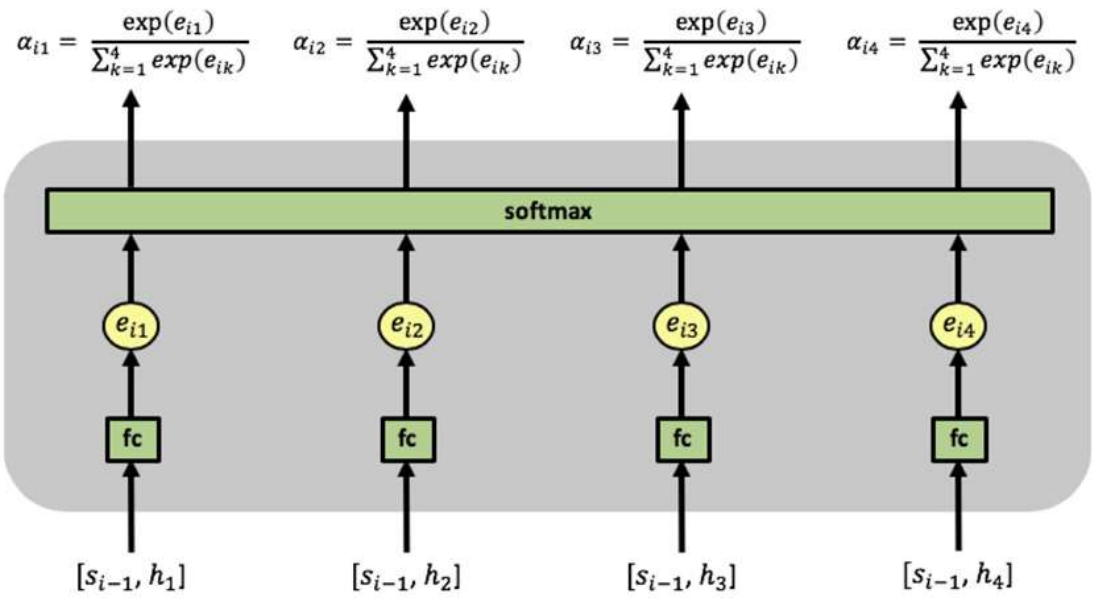
where

- c_t is the context vector - the sum of hidden states h_i of the input sequence, weighted by alignment scores $a_{t,i}$.
- h_i is the hidden state of the encoder on the i-th time step.
- s_t is the hidden state of the decoder on the t-th time step.
- $a_{t,i}$ is the alignment scores assigns as a score is the learned degree in which each output y_t corresponds to the each input x_i.
- $score$ is the score function, which can be different, but here is originally proposed one is shown (so-called additive attention or Bahdanau attention). The decoder uses these attention scores to decide on how much attention to pay to input at every decoding time step. In some sources multiplicative attention is proposed to be more effective ($\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = h_i^\top \mathbf{W}_a s_j$ - Luong attention).
- v_a and W_a are weight matrices to be learned in the alignment model.
- n is the dimension of the source hidden state.

Note

- Here $score$ is the alignment model which scores how well the inputs around position j and the output at position i match.
- The alignment model can be approximated by a small neural network, and the whole model can then be optimised using any gradient optimisation method such as gradient descent.

In such manner, for instance, for one output y_i the attention weights $\alpha_{ij},j=1,2,3,4$ will be generated as it is shown below



After $\alpha_i = \{\alpha_{i1},\alpha_{i2},\alpha_{i3},\alpha_{i4}\} $ calculation the context vector c_i will be calculated and taken into account for prediction of

y_i and for forming hidden state s_i.

This operation will be repeated for all 4 y.

Here $c_{i} = \sum_j \alpha_{ij} h_{j}$ is weighting of decoder hiden states h_{j} for each output y_i

Notes

- The core idea of attention is to focus on the most relevant parts of the input sequence for each output.

  In other words we learn addition network to predict how previous hidden state (s_{i-1}) and current and previous inputs ($h_{i-n}:h_{i-1}; \ h_{i}$) influence on the current output y_i.
- By providing a direct path to the inputs, attention also helps to alleviate the vanishing gradient problem.
- Many articles are proposed to use full-connected network in the attention (instead of using just one perceptron).

Notes 2

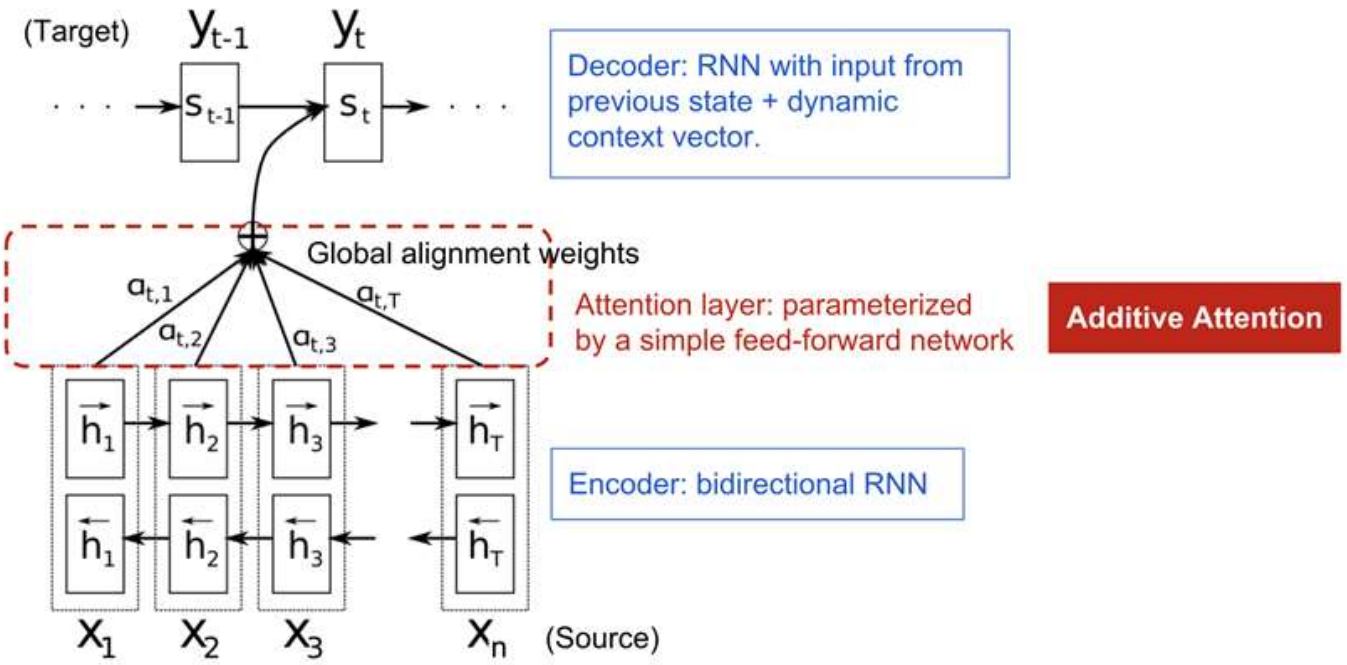
- In the original idea the bi-directional LSTM was used for encoder.
- In the originally proposed architecture instead of encoding the input sequence into a single fixed context vector, the attention model develops a context vector that is filtered specifically for each output time step.
- The original idea was to solve both the aligment and translation problems in natural language processing.

    Alignment is the problem in machine translation that identifies which parts of the input sequence are relevant to each word in the output.

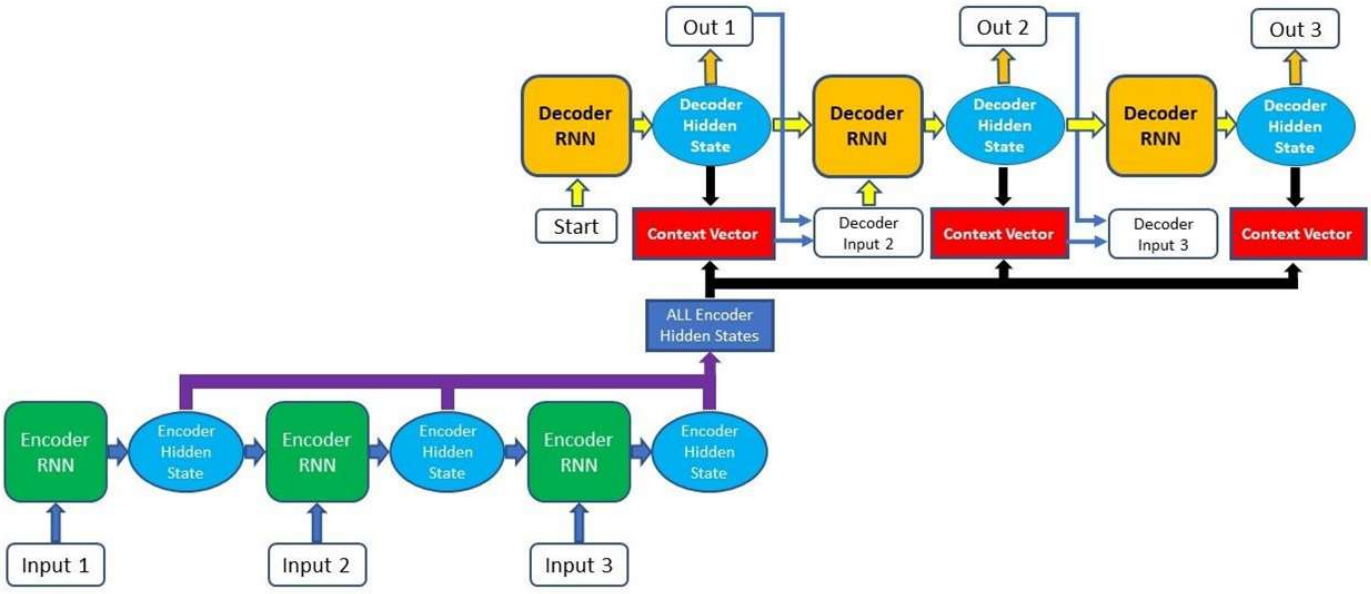
    The translation is the process of using the relevant information to select the appropriate output.

    In this idea attention aim is to (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated.

original idea:



scheme of original attention work:



In the above we calculate attention (score) as $\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_{a1}\boldsymbol{s}_t+\mathbf{W}_{a2}\boldsymbol{h}_i)$,

however it could be other variants of score functions

A several popular score function:

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

Here are a summary of broader categories of attention mechanisms:

1.1.4.2 Self-Attention

The modification of the simple attention approach is the **self-attention layer**.

The main idea of Self_attention is to learn form the only inputs.

The method is also called **intra-attention**.

Method of Self-Attention has been used successfully in a variety of tasks.

And made the revolution in some areas of DL, such as NLP.

The main advantage here is that you can pretrain encoder unsupervised on the any data to predict it farther values (like temporal autoencoder)

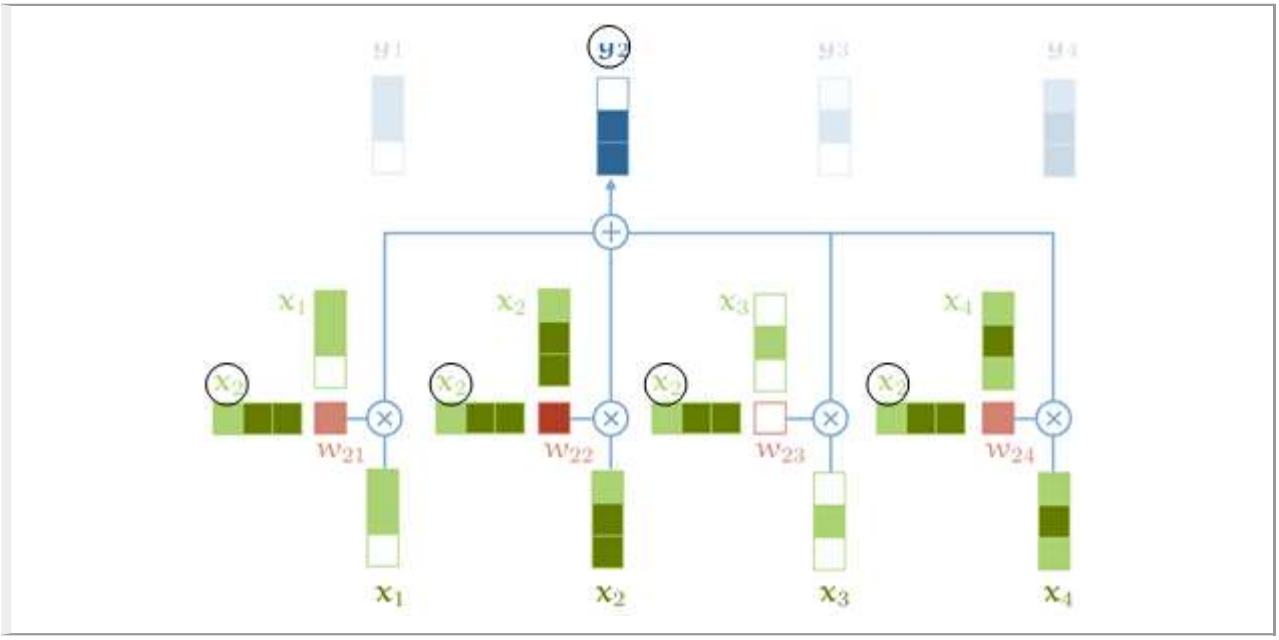
In the case of self-attention we can simplify the additive attention to the follows: $$ score(\mathbf{h}_i) = \mathbf{v}_a{}^\top \text{tanh}(\mathbf{W}_a \mathbf{h}_i) $$ Thus we can calculate the representation $ c $ in the matrix form as follows: $$ \begin{align} \begin{split} \mathbf{a} &= \text{softmax}(\mathbf{v}_a \text{tanh}(\mathbf{W}_a \mathbf{H}^\top)) \\ \mathbf{c} & = \mathbf{H} \mathbf{a}^\top \end{split} \end{align}, $$ where:

- $H={h_1,...,h_n}$ is the hidden state matrix;
- a is the attention vector;
- c is the sequence representation (context vector).

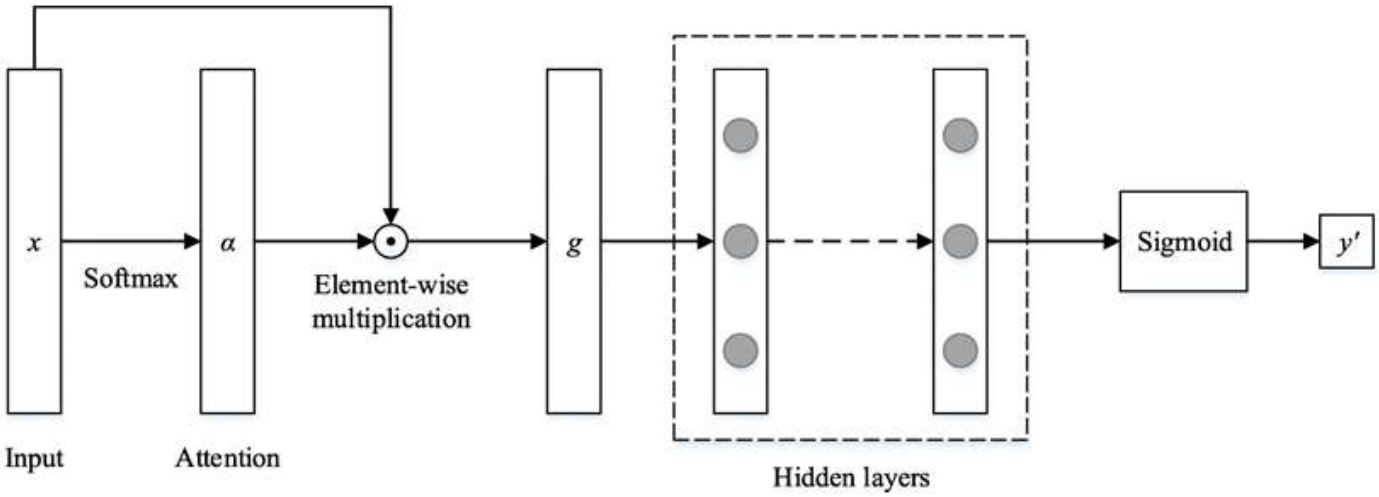
Note

In the other way of explanation of self-attention for each context vector y_i: $$y_i = \sum_{{j}} w_{{i}{j}} x_{j} $$ where j indexes over the whole input sequence (here x).

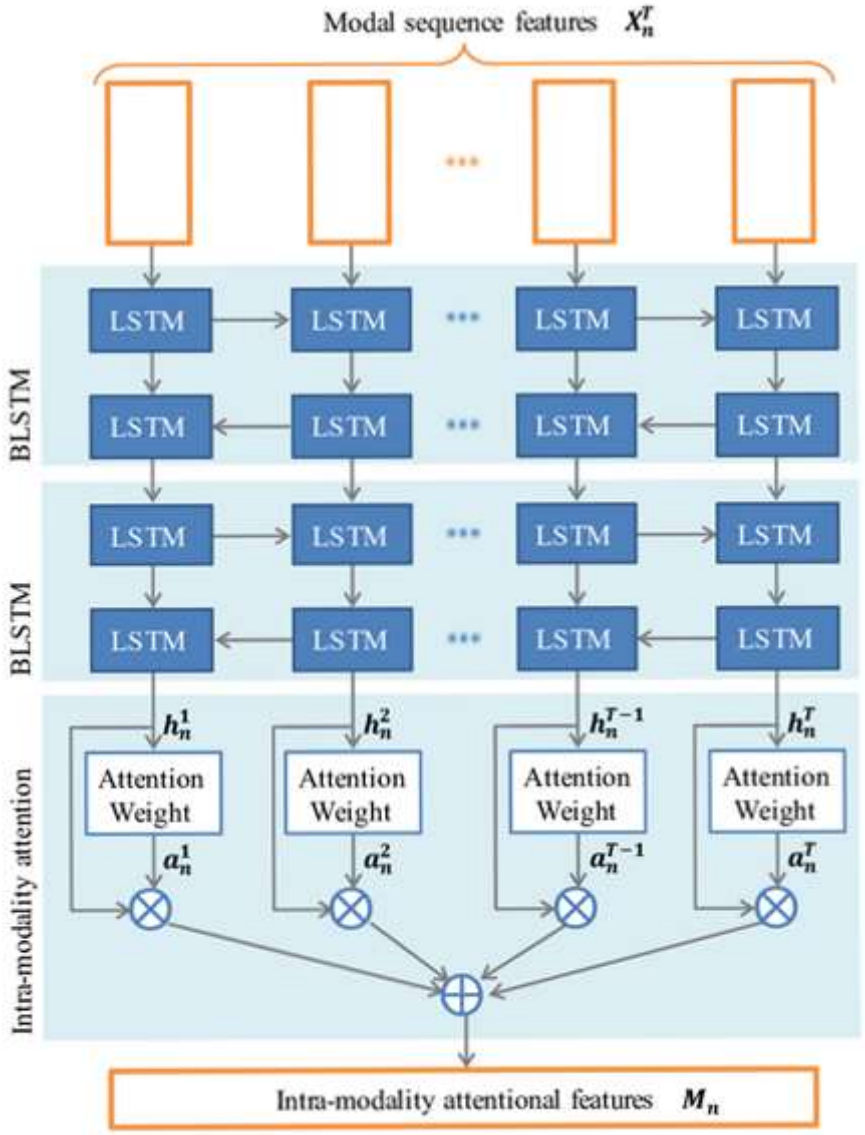
The weight w_{ij} is not a parameter, as in a normal neural net, but it is derived from a function over x_i and x_j. For instance, $w_{{i}{j}} = softmax(x_j\cdot x_i)$



Example of self-attention network for clinical diagnose prediction based on the patient metrics.



Self-attention based encoder scheme of the context vector values producing for multivariate time series.



Key-value self-attention

Key-value attention is the self-attention modification that propose to splits each hidden vector

h_i into a key k_i and a value v_i such that $[k_i;v_i]=h_i$.

The keys part are used for calculating the attention distribution a_i using additive attention: $$ \mathbf{a}_i = \text{softmax}(\mathbf{v}_a{}^\top \text{tanh}(\mathbf{W}_1 [\mathbf{k}_{i-L}; \ldots; \mathbf{k}_{i-1}] + (\mathbf{W}_2 \mathbf{k}_i)\mathbf{\hat1}^\top)) $$ where L is the length of the attention window and $\mathbf{\hat1}$ is a vector of ones.

The values part are then used to obtain the context representation c_i: $$ \mathbf{c}_i = [\mathbf{v}_{i-L}; \ldots; \mathbf{v}_{i-1}] \mathbf{a}^\top $$ Thus we introduce separate weights for current input and previous inputs.

Note

You may choose as key ans value based on your task.

for instance, key could be all expect the last values and value -last value of sequence.

1.1.4.3 Query-key-value Attention and Multi-head Attention

Query-key-value self-attention

Actually, it was proposed that every input vector x_i can be used in three different ways in the attention operation:

- **Query**: x_i is compared to $x_{i-1}:x_{i-n}$ (every other vector) to calculate the weights for its own output y_i.
- **key**: x_i is compared to $x_{i-1}:x_{i-n}$ to calculate the weights for the output of the j-th vector y_j; $j\ne i$.
- **value**: x_i is used as part of the weighted sum to compute each output vector once the weights have been calculate.

In correspondence with previously saied we can introduced three vectors (obtained by three weights matrix): $$ \begin{align*} q_{i} &= W_q x_{i} & k_{i} &= W_k x_{i} & v_{i} &= W_v x_{i} \end{align*} $$ All three weights matrix should have dimension $n\times n$, where n is the window (or input) dimension.

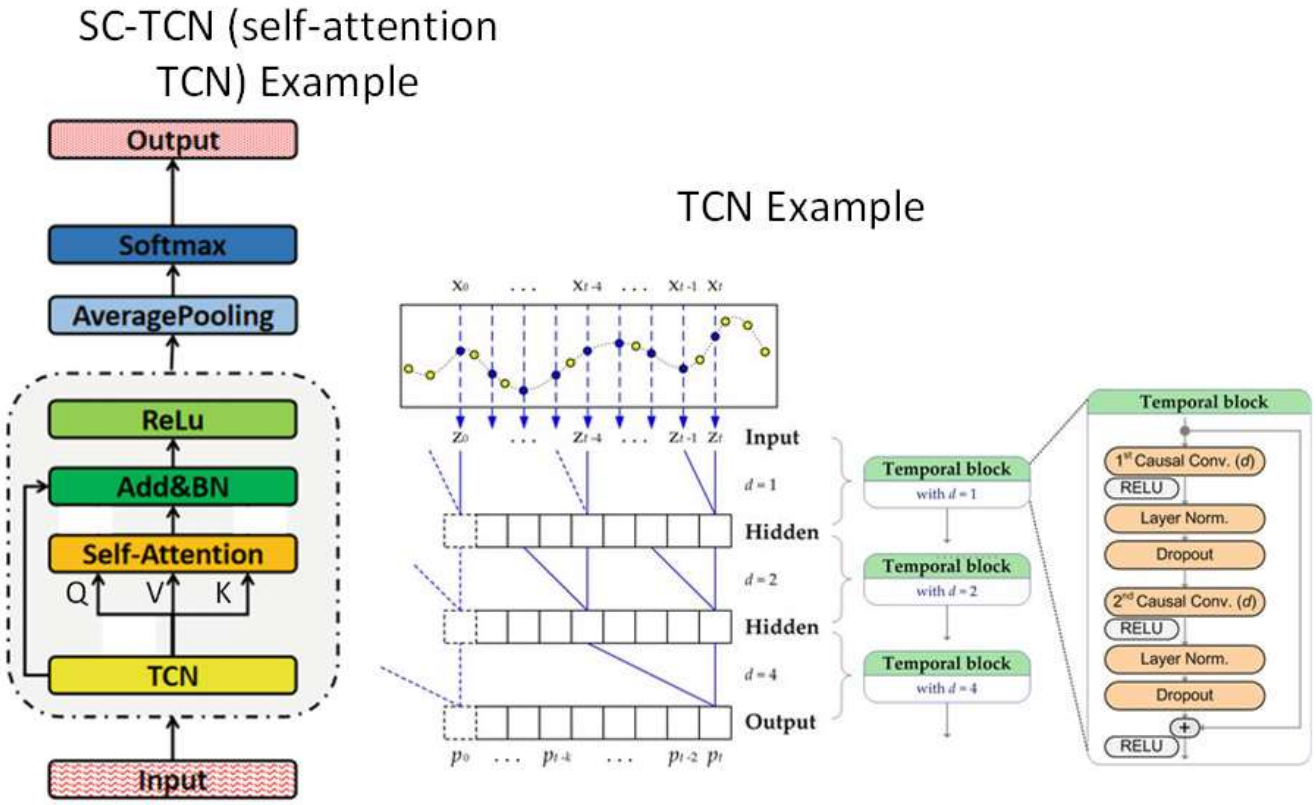
In this case output vector can be calculated as **"scaled-dot-product"** attention: $$\begin{aligned} w'_{ij} &= \frac{{q_{i}}^T k_{j}}{\sqrt n} \\\\ y_{i} &= \sum_{j} w_{ij} v_{j} \\\\ \rm where:&\\\\ w_{ij} &= softmax(w'_{ij}) = \frac{exp(w'_{ij})}{\sum_{j} exp(w'_{ij})} \\\\\text{and } n& \text{ is the dot-product matrix dimension (for matrix $n\times n$) } \end{aligned}$$

Note **The scale $\sqrt n$ in w' equation** need to the follows.

The softmax function can be sensitive to very large input values. This can kill the gradient, and slow-down learning, or cause it to stop altogether.

Since the average value of the dot product grows with the embedding dimension k, it helps to scale the dot product back a little to stop the inputs to the softmax function from growing too large.

Example of Temporal-convolution - self attention (query-key-value) network for multivariate time series classification.



Multi-head attention

The discriminative power of self-attention can be increased by using its ensembling.

The Ensembling can be implemented combining several self attention layers in parallel (which we'll index with r), each with different weights matrix W^r_q, W^r_k,W^r_v.

These are called **attention heads**.

For each input x_i each attention head produces a different output vector y^r_i. We concatenate these, and pass them through a linear transformation to reduce the dimension back to n. *Note*

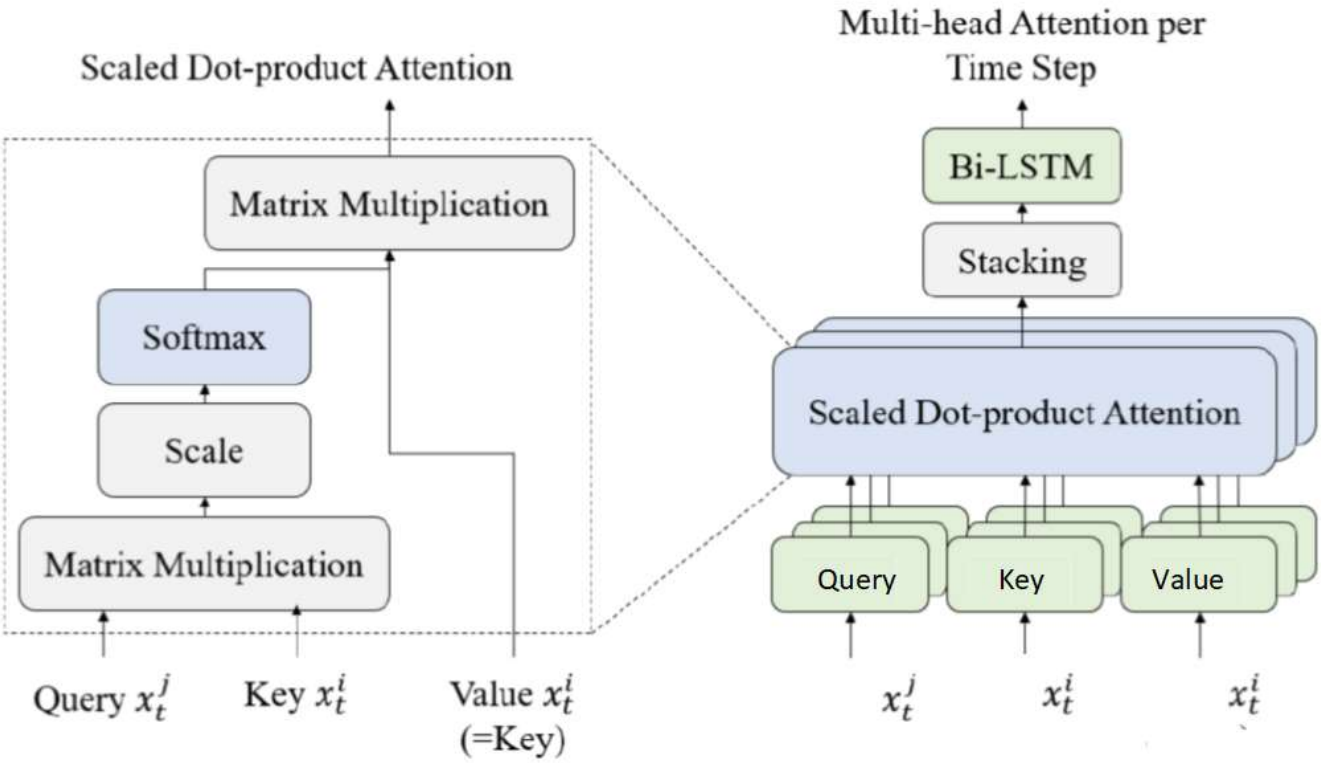
Actually it can be distinguished:

- **Narrow multi-head self-attention**: cut input vector into r parts (if the vector has 256 dimensions, and we have 8 attention heads, we cut it into 8 chunks of 32 dimensions).
- **wide multi-head self-attention**: produce r matrices with whole size (for 256 dim produce 8 attention heads, with 256×256 dim. -This way produce a better result at the cost of more memory and time.

In the Multi-head self-attention context vector c can be given as: $$ c = \rm MultiHead(Q, K, V) = W^O \cdot \rm concat[head_1; \dots; head_h], $$ where:

- $ \rm head_i = Attention(Q_i, K_i, V_i)$
- $$ \rm Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{n}})V $$
- W^O is the weights matrix of each head impact.

Example of multi-head attention simple network



Masked Attention

The drawback of the simple self-attention is that it is attended by influence of forward and previous inputs on each output.

That lead to violations of causality requirements.

For avoiding this the weights matrix have to be transformed to casual-form (i.e. triangle matrix).

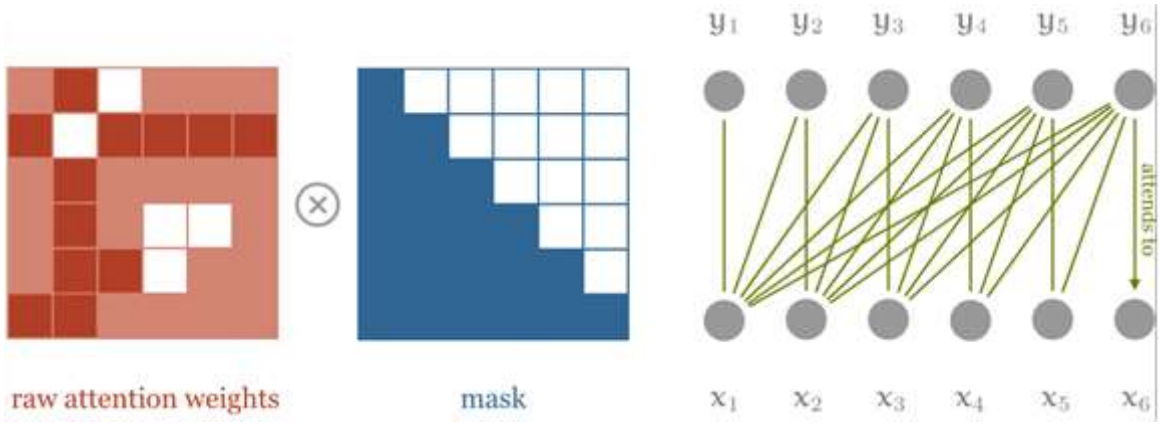
Such operation can be performed by its multiplication with corresponding matrix mask.

Masking the self-attention allow to use it as an autoregressive model analogue.

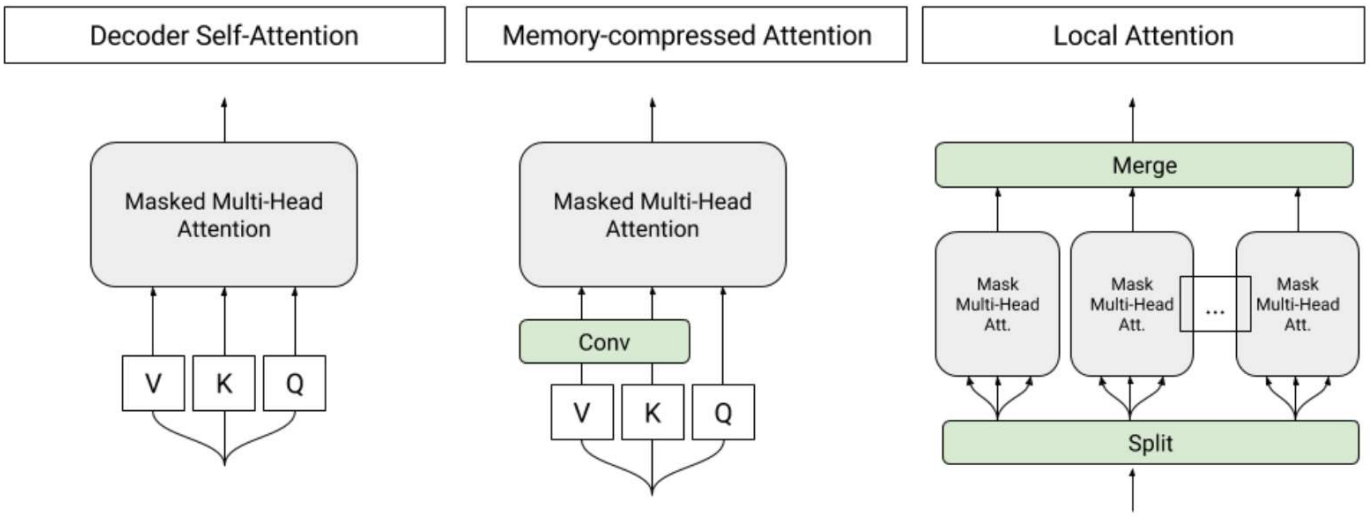
In the case of masking the attention can be given as $$ \rm Attention(Q, K, V) = softmax(M\frac{QK^\top}{\sqrt{n}})V $$ where M is masking matrix - upper triangle, in the following form $$ \begin{bmatrix} 0 & -\infty & -\infty & \dots & -\infty \\ 0 & 0 & -\infty & \dots & -\infty \\ 0 & 0 &0 & \dots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} $$ Where we use $-\infty$ since we want these elements to be zero after the softmax.

Illustration of masked attention

Note Masking the self attention, to ensure that elements can only attend to input elements that precede them in the sequence. Note that the multiplication symbol is slightly misleading: we actually set the masked out elements (the white squares) to $-\infty$.



Example of a several approach for multi-head masked self-attention.



In memory compressed attention case $\rm conv$ - mean $1x1 \rm conv$ for reduce the number of convolution in self attention. It is like the denoising - you drop empty of data features.

In the decoder self-attention you processing all the data together. In local attention here you process data from each sliding window position separately and the merge results.

1.1.4.4 Transformer block and Transform Architecture

The transformer block

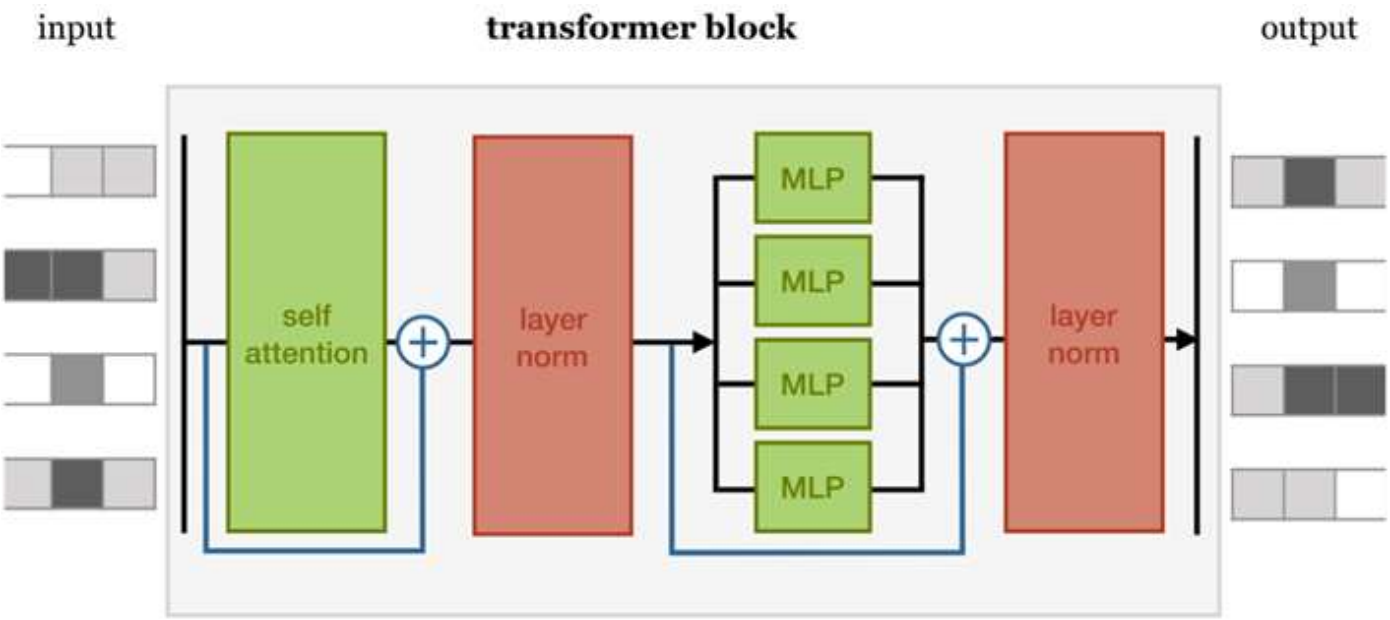
As usual the multi-head attention is used in the content of the so-called **Transformer block**.

Transformer block consists of a:

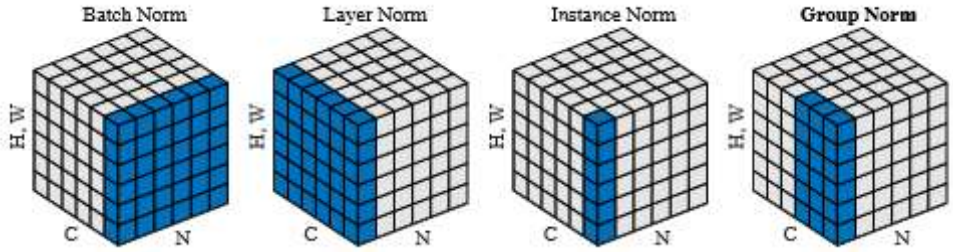
- a self attention layer,
- layer normalization,
- a feed forward layer (a single MLP applied independently to each vector),
- and another layer normalization.
- Residual connections are added around both, before the normalization.

Note

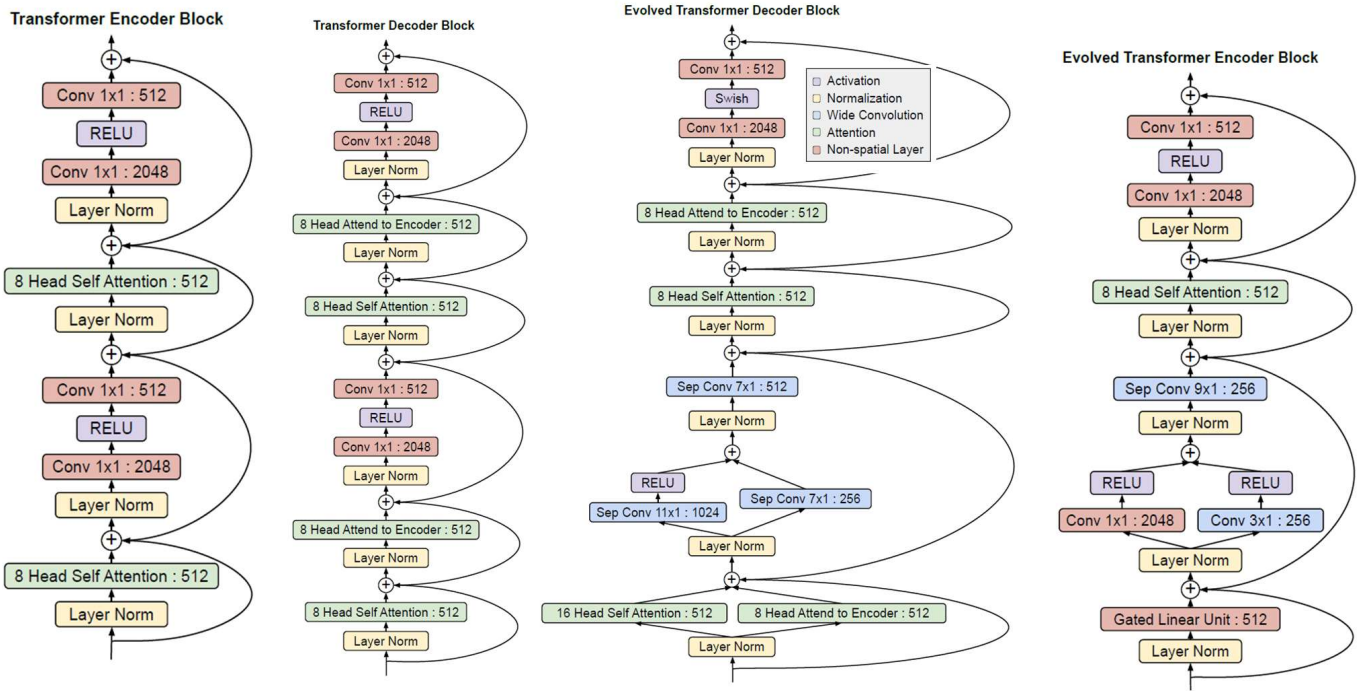
- The order of the various components is not set in stone; the important thing is to combine self-attention with a local feedforward, and to add normalization and residual connections.
- in transformer you may have multihead self attention and each MLP will work with different head.
- The architecture of transform block contains no recurrence - it is the Full-connected architecture in the original idea



Please note that classical implementation of the transformer use layer normalization instead of batch normalization



The architecture of transform blocks in stack.



Transformer with Positional Encoding

The main drawback of simple transformer (FC) implementation that it can-not distiguish an order of the sequence.

> One possible solution is to give the model some sense of order - add a piece of information about its position to each sample in the series. We call this "piece of information", **the positional encoding.**

To incorporate information about the order of the sequence the position encoding is included in the transformer architecture.

> The encoding is performed by mapping time steps $t:t+\tau$ values to the lookup table which is allowing to obtain d-dimensional code for each time step.

The positional encoding have to be carried out during both training and evaluation.

The positional encoding can be done as:

- randomized encoding (fixed as hyper parameter).
- learned encoding (learned as hyper parameter).
- fixed function encoding (in the original proposition).

Example of the most simple fixed function encoding as binary representation of dimension 4. $$ \begin{align} \text{position } 0: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{0}} & & \text{position } 8: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{0}} \\ \text{position } 1: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{1}} & & \text{position } 9: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{1}} \\ \text{position } 2: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{0}} & & \text{position } 10: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{0}} \\ \text{position } 3: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{1}} & &

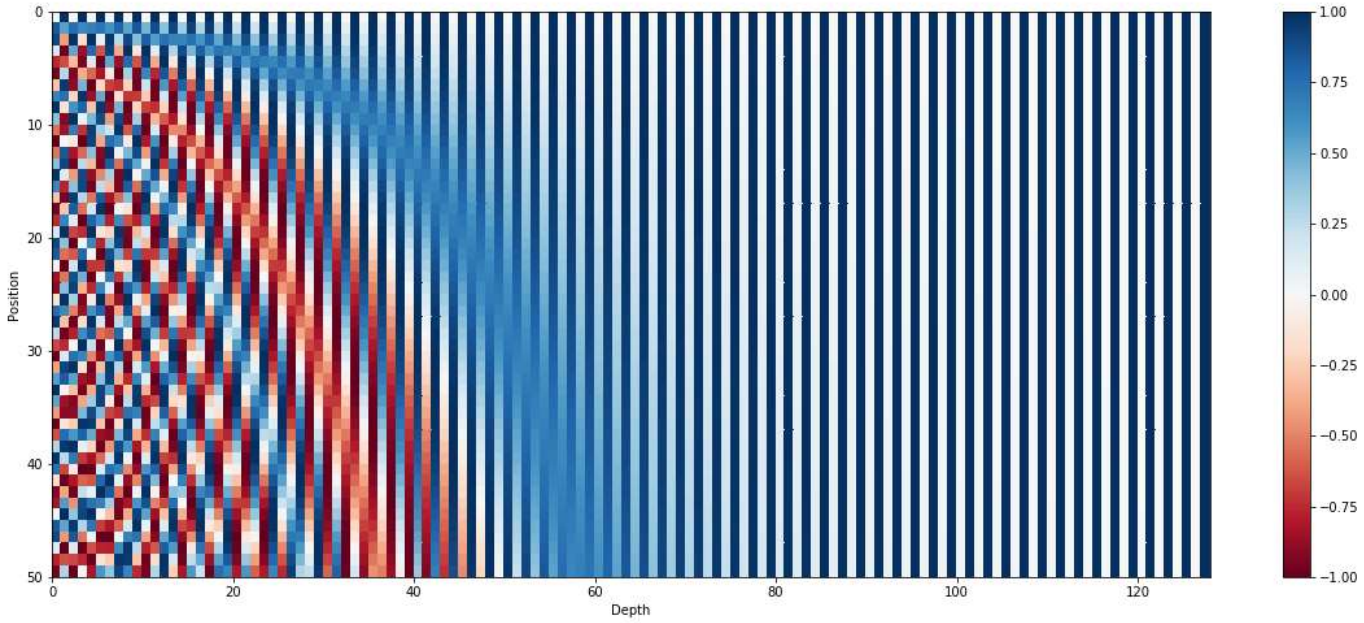
\text{position } 11: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{0}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{1}} \\ \text{position } 4: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{0}} & & \text{position } 12: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{0}} \\ \text{position } 5: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{1}} & & \text{position } 13: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{0}} \ \ \color{red}{\texttt{1}} \\ \text{position } 6: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{0}} & & \text{position } 14: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{0}} \\ \text{position } 7: \ \ \ \ \color{orange}{\texttt{0}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{1}} & & \text{position } 15: \ \ \ \ \color{orange}{\texttt{1}} \ \ \color{green}{\texttt{1}} \ \ \color{blue}{\texttt{1}} \ \ \color{red}{\texttt{1}} \\ \end{align} $$

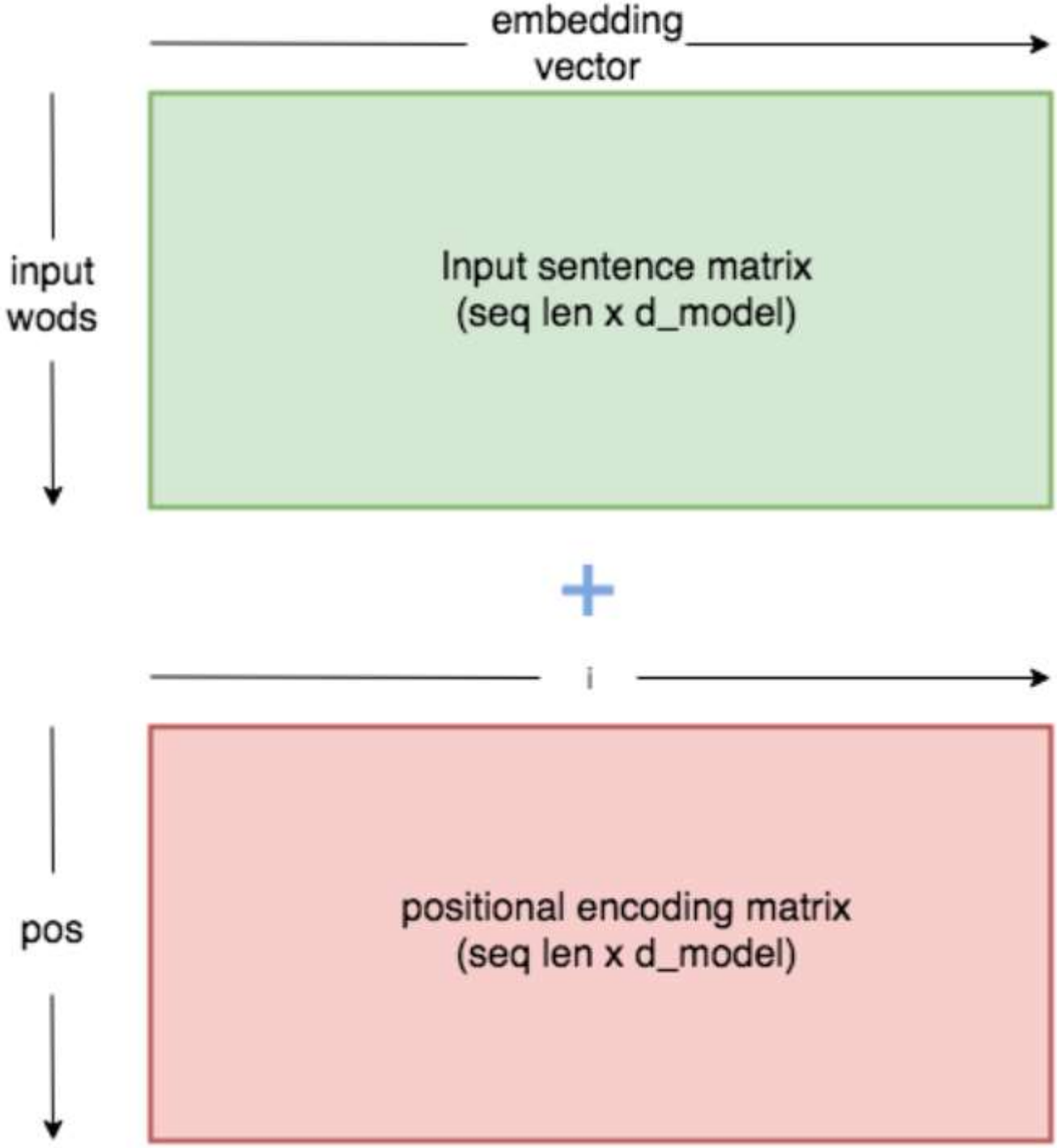
The originally proposed representation of the position encoding $$ \vec{p_t} = \begin{bmatrix} \sin({\omega_1}.t)\\ \cos({\omega_1}.t)\\ \\ \sin({\omega_2}.t)\\

\cos({\omega_2}.t)\\ \\ \vdots\\ \\ \sin({\omega_{d/2}}.t)\\ \cos({\omega_{d/2}}.t) \end{bmatrix}_{d \times 1} $$

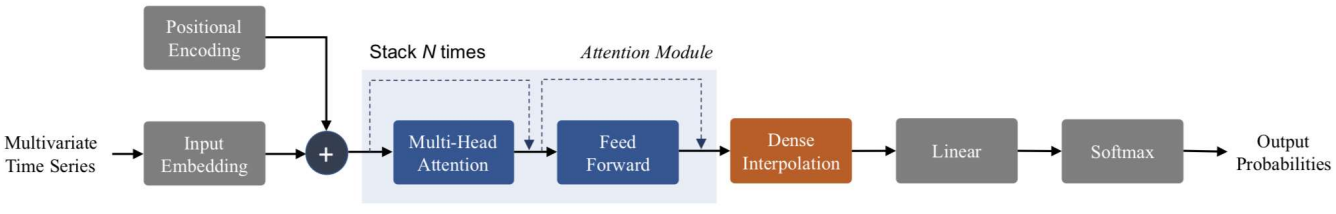
Example of the 128-dimensional positional encoding for a sequence with the maximum length of 50.

Each row represents the embedding vector $\vec{p_t}$





Example of multi-head attention network for the time-series classification



As it is shown above The time series transformer based feature extraction consists of the Input embedding performed of full-1-d conv network and position encoder.

The reason why position encoding is add (instead of concatination)

is due to the high dimensionality of the input (N feature maps - feature embeddings).

By add position encoding matrix you get features approximately orthogonality.

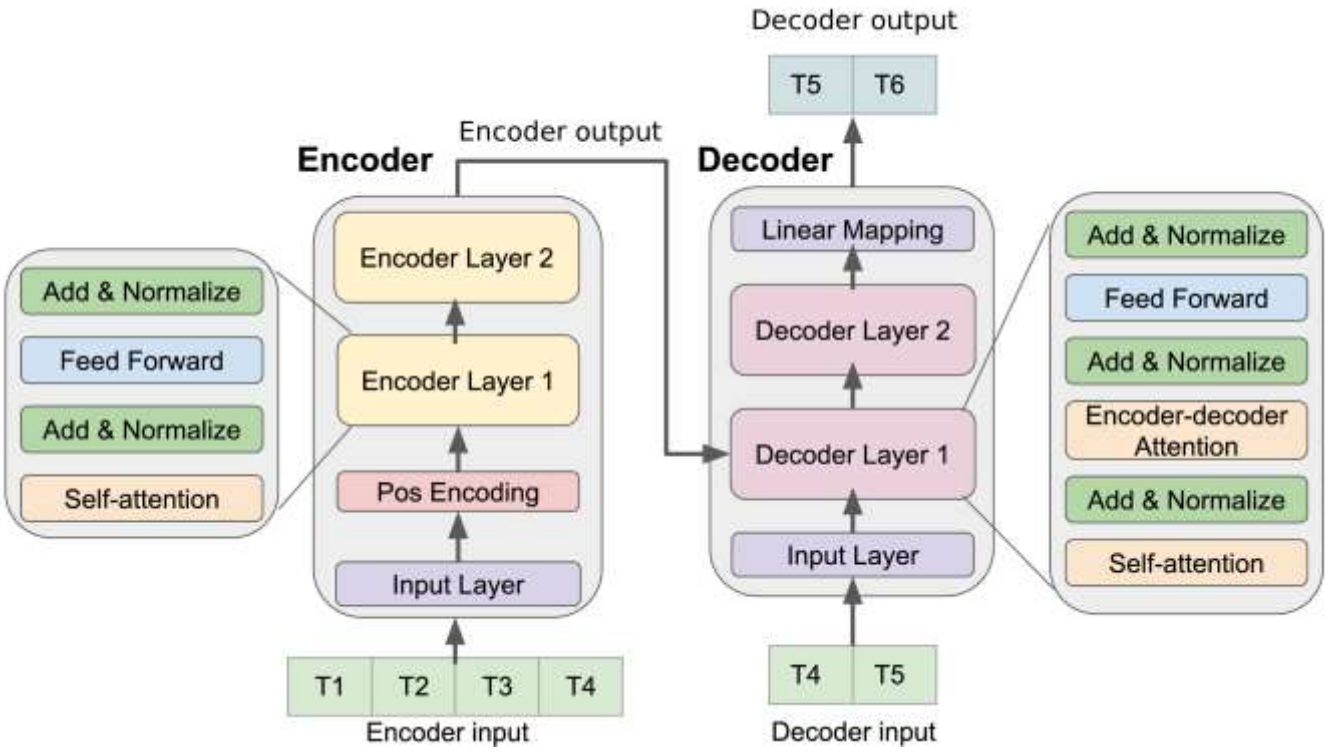
This is meaning full in multihead attention

Position Encoding add just small pies of inferences to the data (like noises) - it is expect to have no effect on the resulted accuracy

Note

The transformer block have to be destinguished from the **transformer encoder-decoder model**

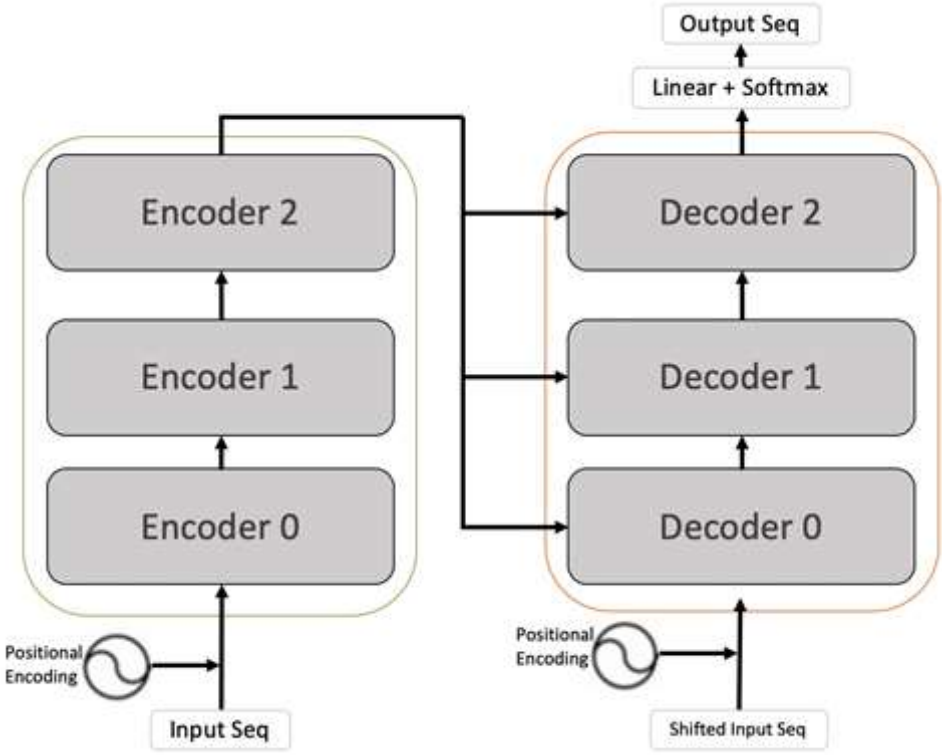
Example of the time series prediction transformer encoder-decoder model



Note

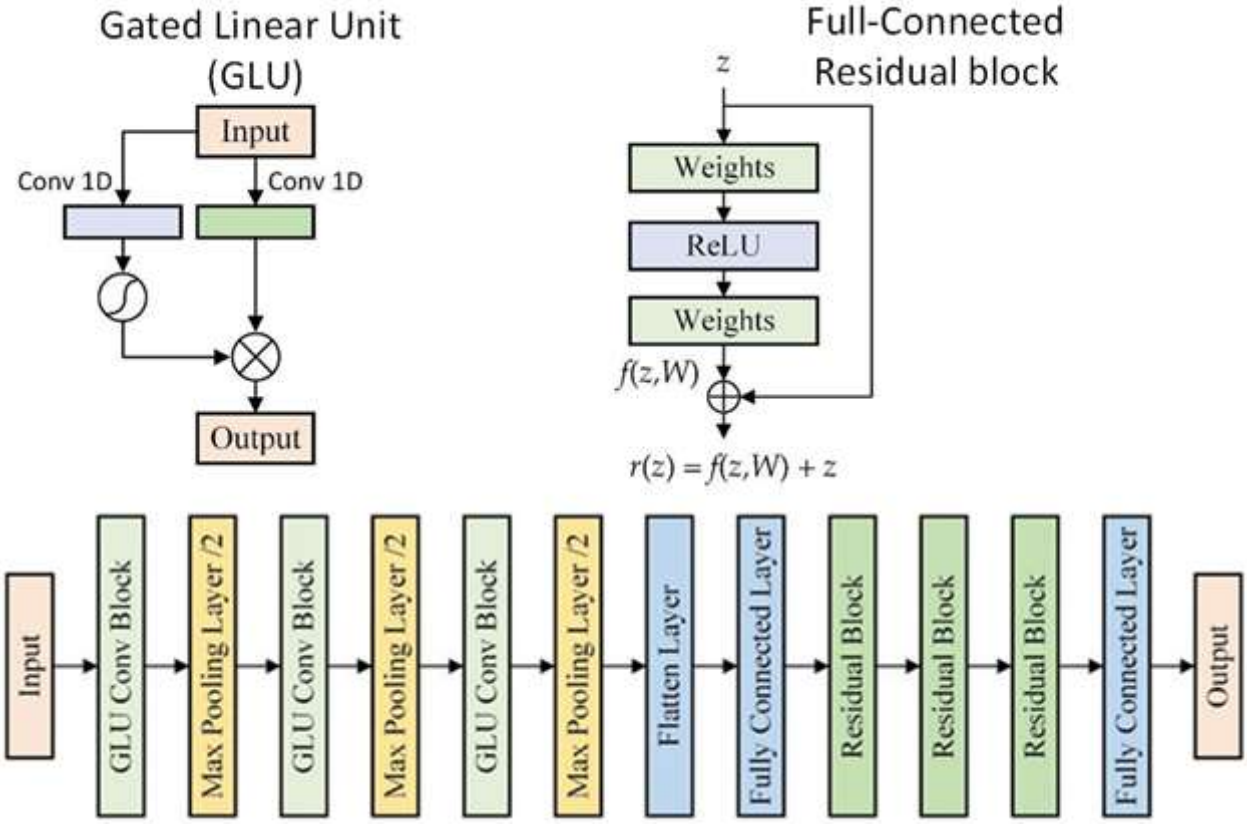
In the decoder-encoder attention query and key are gotten from encoder and value from decoder input.

Example of the time series prediction stacked transformer encoder-decoder model

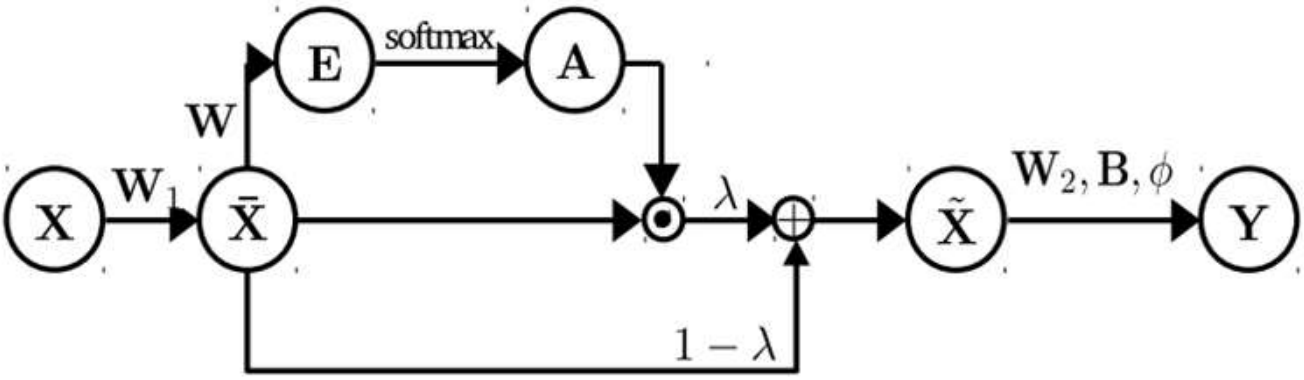


A several papers proposed to a number of attention modification for time series

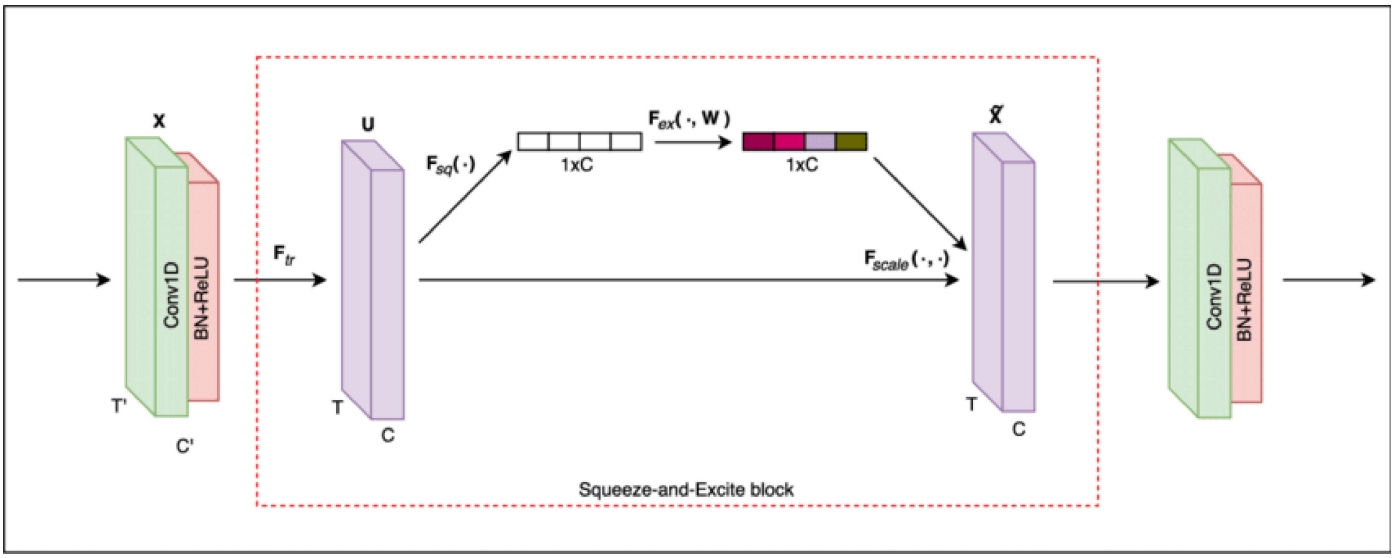
The one of the most popular is the **Gated linear Unit**



The other one is the temporal-attention block



Squeeze-And-Excitation block as alternative attention



In [ ]: