Sequence Modeling

S.Calderara, R. Di Carlo

UNIMORE

December 3, 2020

Agenda

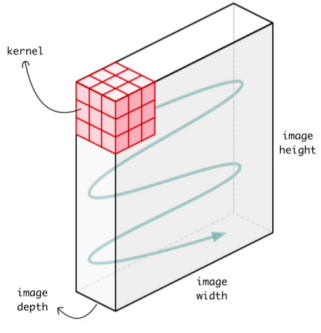
A sequence model is a model that takes in input a sequence of items (words, letters, time series, audio signals, etc) and outputs one item or another sequence of items.

$< x_1 >$	$< x_2 >$	$< x_3 >$		$< y_1 >$	$< y_2 >$	$< y_3 >$	$< y_4 >$
Je	suis	étudiant	$\longrightarrow$	I	am	a	student

Some of the tasks using seq-to-seq models are: Machine translation, Time-series forecasting, Video Captioning, Audio generation, Text Summarization and more.

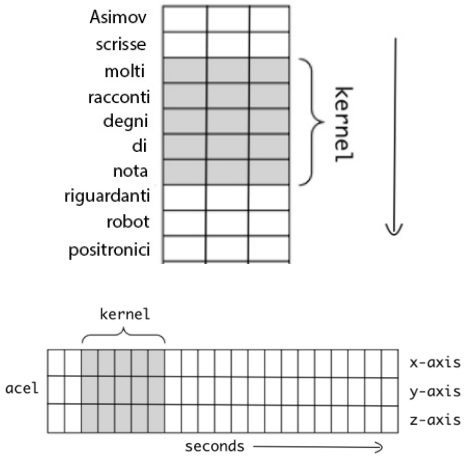
Even if sequence modeling is very often associated with recurrent neural networks, yet recent results indicate that other neural network architectures can outperform RNN on many tasks.

The standard Convolution Neural Network was first introduced in Lenet-5 architecture. Conv2D is generally used on Image data. It is called 2 dimensional CNN because the kernel slides along 2 dimensions on the data.



In **Conv1D** the kernel slide only in one dimension. Which kind of data requires kernel sliding in only one dimension and have spatial properties?

1D Convolution



We can apply it to time series, as well as to text, since we can represent each word as a vector of fixed length.

The discrete 1D convolution is defined as:

$$(\boldsymbol{x} * \omega)(n) = \sum_{i=0}^{k-1} \boldsymbol{x}(n-i)\omega(i)$$

Where $\boldsymbol{x}$ is our input vector of length $\boldsymbol{n}$, and $\boldsymbol{\omega}$ our kernel of length $\boldsymbol{k}$.

1D Convolution - example

Asimov	0.2	0.1	-0.3	0.4
scrisse	0.5	0.2	-0.3	-0.1
molti	-0.1	-0.3	-0.2	0.4
racconti	0.3	-0.3	0.1	0.1
sui	0.2	-0.3	0.4	0.2
robot	0.1	0.2	-0.1	-0.1
positronici	-0.4	-0.4	0.2	0.3

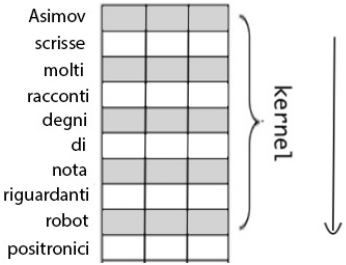
Apply a **filter**(or **kernel**) of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

Asimov scrisse molti	-1
scrisse molti racconti	-0.5
molti racconti sui	-3.6
racconti sui robot	-0.2
sui robot positronici	0.3

Dilated convolution

A major disadvantage of using CNN for sequences is that in order to achieve a long effective history size, we need an extremely deep network or very large filter.



To mitigate this problem we can increase the receptive field using dilated convolution.

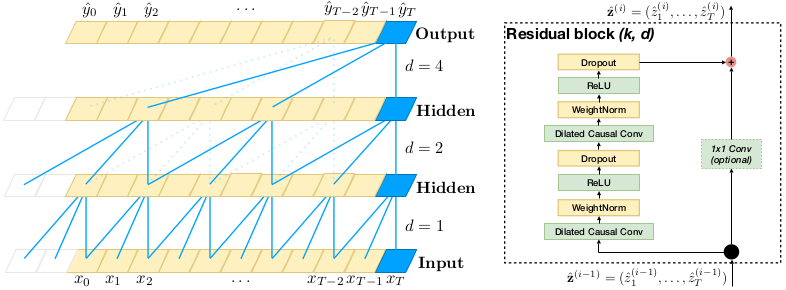
$$(\boldsymbol{x} * \omega)(n) = \sum_{i=0}^{k-1} \boldsymbol{x}(n - d \cdot i)\omega(i)$$

Where $\boldsymbol{d}$ is the dilation factor

Temporal Convolutional Network

1D-Conv layers are used in many models as a valid alternative to RNN in different tasks.

(*b*) The **TCN** [1] is a model composed of residual blocks of dilated causal convolutions
(*a*) Each layer use a different dilation factor **d**= 1,2,4 and filter size **k**= 3



(a) (b)

Transformer

Sequence-to-sequence modelling using RNNs? No thanks, **attention** is all I need. [5]

Challenges with RNNs

- Long range dependencies
- Gradient vanishing and explosion
- Large # of training steps
- Recurrence prevents parallel computation

Transformer networks

- Facilitate long range dependencies
- No gradient vanishing and explosion
- Fewer training steps
- No recurrence, facilitate parallel computation

Transformer architecture

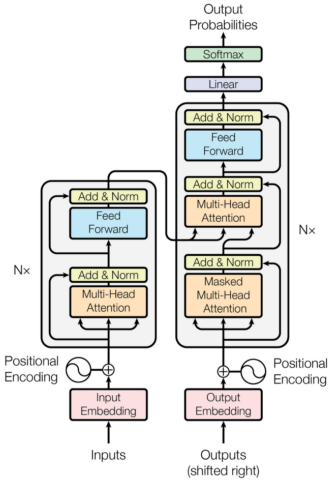


Figure 1: The Transformer - model architecture.

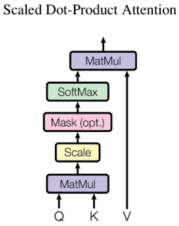
Transformer networks are composed by two main stacks:

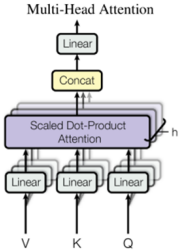
- Encoder
- Decoder

Both are composed of a sub-stacks of ***N*** modules consisting mainly of **Multi-Head Attention** and **Feed Forward** layers.

Attention

Let's start with the description of the attention-mechanism. It's can be described by the following equation:



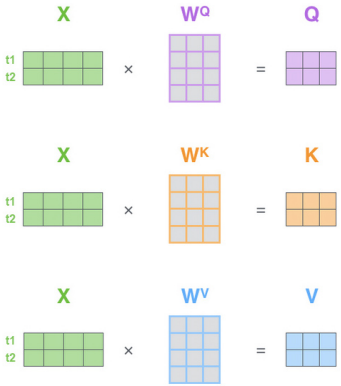


$$Attention(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{\boldsymbol{d}_k}}\right)\boldsymbol{V}$$

Where **Q, K, V** are vectors representing query, keys and values. $\boldsymbol{d}_k$ is just a scaling factor to prevent dot product growing large in magnitude, pushing the softmax into regions where it has small gradients.

Query, keys and values

Query, keys and values are generated using self-attention from each of the encoder's input vectors.



$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^{Q}$$

$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^{K}$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^{V}$$

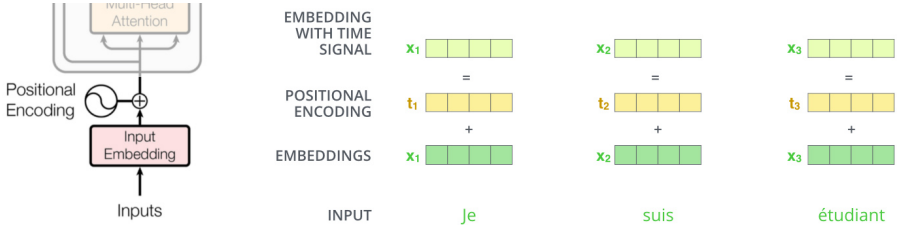
Query, keys, and values have dimension $\boldsymbol{d_k}$.
All sub-layers in the model, as well as the embedding layers, produce outputs of dimension $\boldsymbol{d_{model}} = 512$.

$$d_k = d_{model}/N_{heads}$$

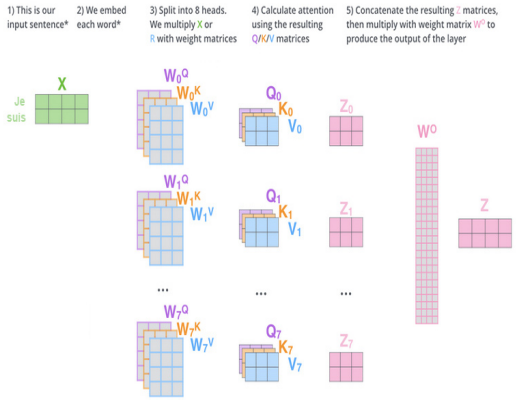
Positional encoding

Since the Transformer contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens.

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k+1 \end{cases} \quad \textit{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$



Multi-head attention



A **multi-head attention** layer is composed of many attention layers, each of which has its own set of queries, keys and values obtained by randomly initializing the weights matrices.

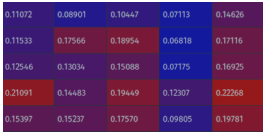
Then, after training, each set is used to project the input embeddings into a different representation subspace.

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = Concat(head_1, ..., head_h)\boldsymbol{W}^O$$

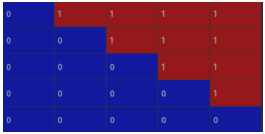
where:

$$head_i = \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right)$$

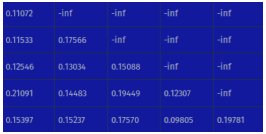
Decoding



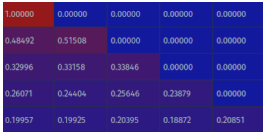
Attention matrix



Mask



Attention masked



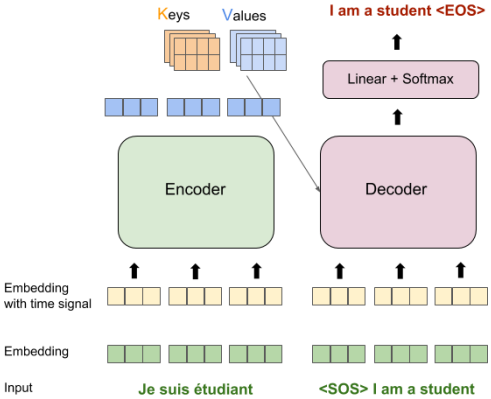
Attention Softmax

During the training phase, the input of the decoder stack is the GT sequence shifted right to add the start token.

$< y_1 >$	$< y_2 >$	$< y_3 >$	$< y_4 >$	$< y_5 >$
SOS	I	am	a	student

Also we need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

Decoding



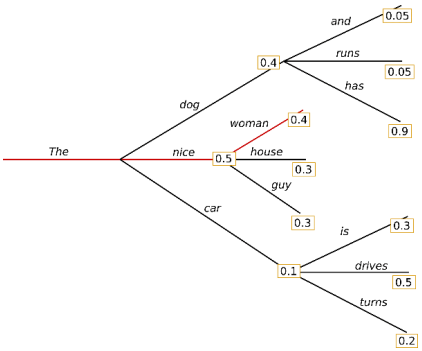
In "encoder-decoder attention" layers, the queries $\boldsymbol{Q}$ come from the previous decoder layer,and the memory keys $\boldsymbol{K}$ and values $\boldsymbol{V}$ come from the encoder output.

Inference

At inference time we do not have the GT sequence to provide as input to the decoder. So we use the transformer in auto-regressive mode.

Encoder input (x)	Decoder input ($\hat{y}_{t-1}$)	Decoder Output ($\hat{y}_t$)
Je suis étudiant	<SOS>	I
Je suis étudiant	<SOS>I	I am
Je suis étudiant	<SOS>I am	I am a
Je suis étudiant	<SOS>I am a	I am a student
Je suis étudiant	<SOS>I am a student	I am a student <EOS>

Greedy Search



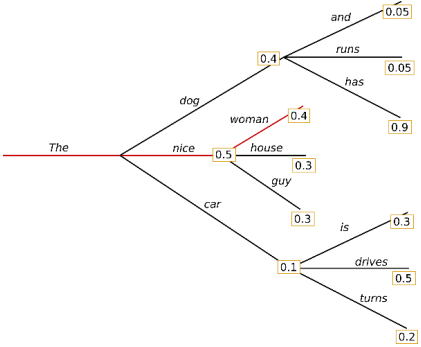
When we predict a sentence, the last Softmax layer gives us a probability distribution on the words in our dictionary. If we always choose the one with the highest probability, we are applying a greedy search.

A greedy search is not always the best choice, in fact if we assume the probability distribution of a word sequence as:

$$P\left(w_{1:T} \mid W_0\right) = \prod_{t=1}^{T} P\left(w_t \mid w_{1:t-1}, W_0\right)$$

$\boldsymbol{W_0}$ being the initial context word sequence

Greedy Search

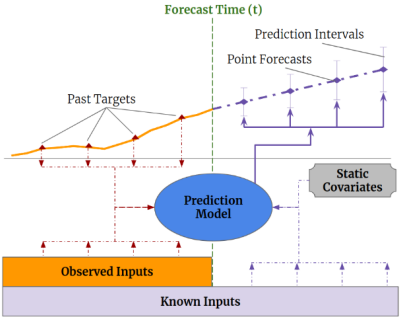


We obtain:

$P(\text{"The","nice","woman"}) = 0.2$
$P(\text{"The", "dog", "has"}) = 0.36$

Beam search is one of the techniques to mitigate this problem, reducing the risk of missing hidden high probability word sequences by keeping the most likely n_{beams} of hypotheses at each time step.

Temporal Fusion Transformer

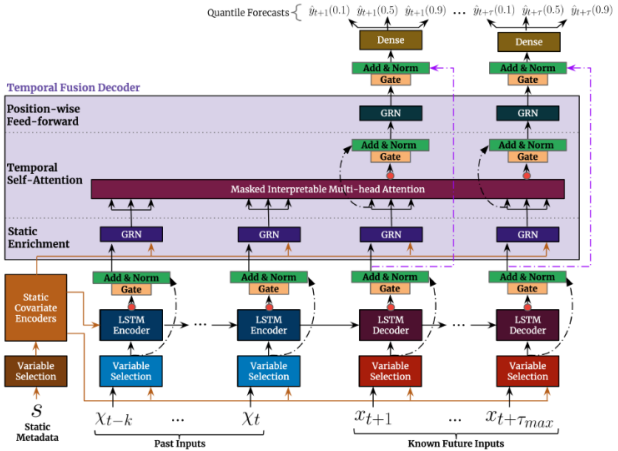


Real world time series datasets consist of many type of input features:

- Past observed inputs
- Apriory known future inputs
- Static features

TFT (Temporal fusion transformer) [3] is a novel interpetrable forecasting deep neural network model from Google AI.

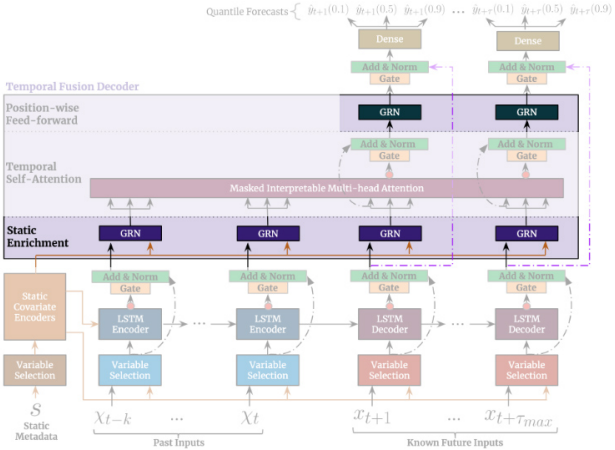
Architecture

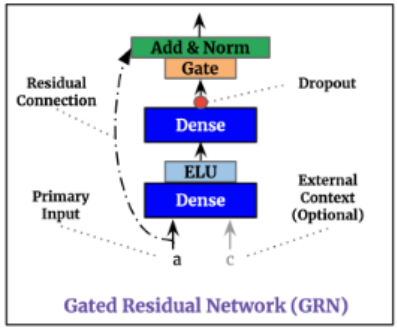


TFT is based on sequential processing using both:

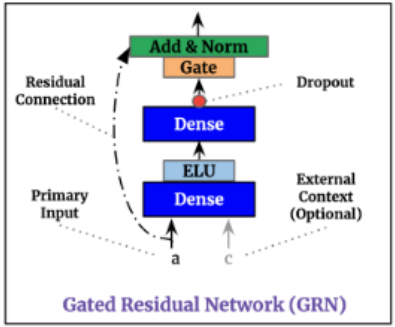
- Recurrent layers, to capture local dynamics
- Attention layers, to capture long-term context

GRN layer


Quantile Forecasts
Dense
Add & Norm
Gate
Temporal Fusion Decoder
Position-wise
Feed-forward
GRN
Temporal
Self-Attention
Masked Interpretable Multi-head Attention
Static
Enrichment
Static
Covariate
Encoders
LSTM
Encoder
LSTM
Decoder
Variable
Selection
Static
Metadata
Past Inputs
Known Future Inputs


Add & Norm
Gate
Residual
Connection
Dropout
Dense
ELU
Dense
Primary
Input
External
Context
(Optional)
a
c
Gated Residual Network (GRN)

GRN layer



Gated Residual Network (**GRN**) is a building block of TFT. The GRN takes in a primary input $\boldsymbol{a}$ and an optional context vector $\boldsymbol{c}$

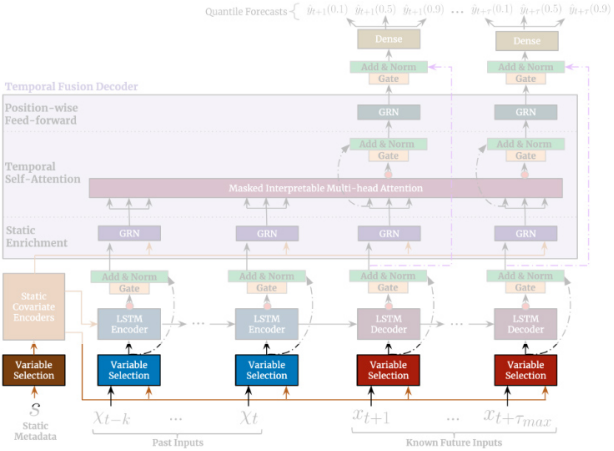
$$\begin{aligned}\mathrm{GRN}_{\omega}(\boldsymbol{a}, \boldsymbol{c}) = \ & \text{LayerNorm } (\boldsymbol{a} + \mathrm{GLU}_{\omega}(\boldsymbol{\eta}_1)) \\ \boldsymbol{\eta}_1 = \ & \boldsymbol{W}_{1,\omega}\boldsymbol{\eta}_2 + \boldsymbol{b}_{1,\omega} \\ \boldsymbol{\eta}_2 = \ & \mathrm{ELU}\left(\boldsymbol{W}_{2,\omega}\boldsymbol{a} + \boldsymbol{W}_{3,\omega}\boldsymbol{c} + \boldsymbol{b}_{2,\omega}\right)\end{aligned}$$

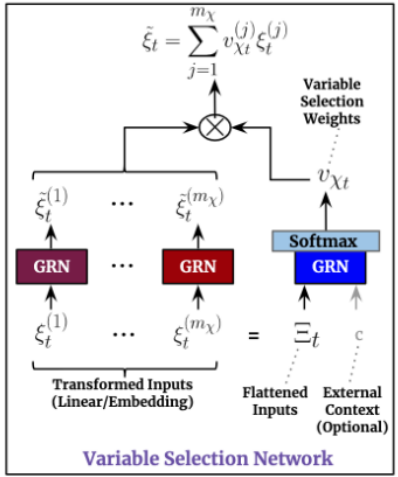
where:

$$\mathrm{GLU}_{\omega}(\gamma) = \sigma\left(\boldsymbol{W}_{4,\omega}\gamma + \boldsymbol{b}_{4,\omega}\right) \odot \left(\boldsymbol{W}_{5,\omega}\gamma + \boldsymbol{b}_{5,\omega}\right)$$

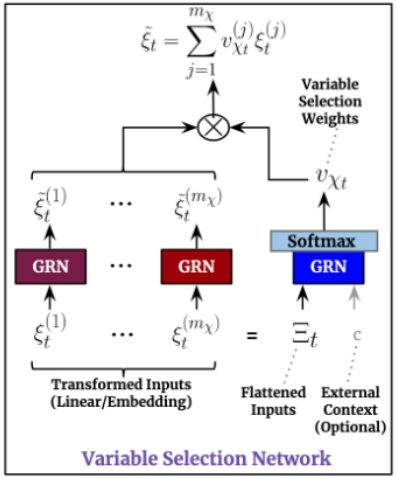
ELU is the Exponential Linear Unit activation function. $\sigma(.)$ is the Sigmoid activation function.

Variable Selection Network





Variable Selection Network



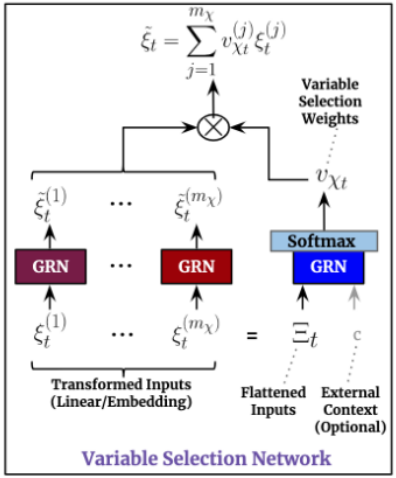
Variable selection also allows TFT to remove any unnecessary noisy inputs which could negatively impact performance.

Being $\xi_t^{(j)}$ the embedding of the $\boldsymbol{j}$-th variable at time $\boldsymbol{t}$ and $\Xi_t = \left[\xi_t^{(1)^T}, \ldots, \xi_t^{(m_\chi)^T}\right]^T$ a vector of past inputs at time $\boldsymbol{t}$:

$$\boldsymbol{v}_{\chi_t} = \text{Softmax}\left(\text{GRN}_{v_\chi}\left(\Xi_t, \boldsymbol{c}_s\right)\right)$$

c_s is a context vector and v_{χ_t} is a vector of variable selection weights.

Variable Selection Network



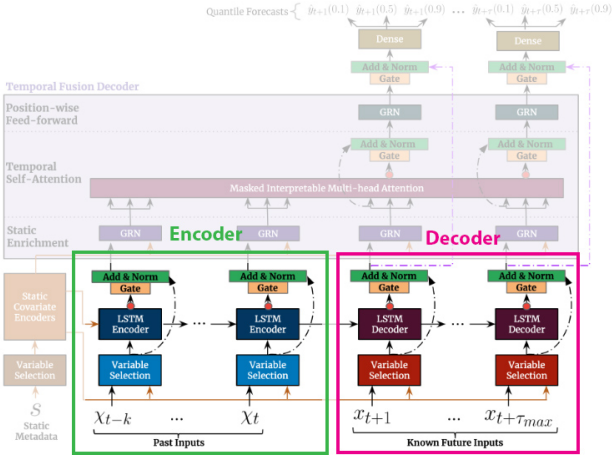
At each time step, an additional layer of non-linear processing is employed by feeding each $\xi_t^{(j)}$ through its own GRN:

$$\tilde{\xi}_t^{(j)} = \text{GRN}_{\tilde{\xi}^{(j)}}\left(\xi_t^{(j)}\right)$$

Where each GRN share weights across all time steps $\boldsymbol{t}$. Finally input variables are weighted and combined:

$$\tilde{\xi}_t = \sum_{j=1}^{m_\chi} v_{\chi_t}^{(j)} \tilde{\xi}_t^{(j)}$$

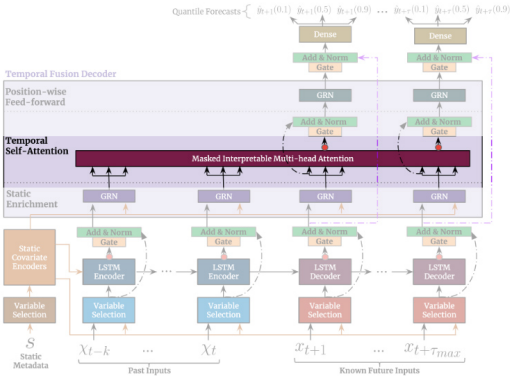
Sequence to sequence layer



The encoding and decoding part is done using LSTM layers, feeding $\xi_{t-k:t}$ into the encoder and $\xi_{t+1:t+\tau_{max}}$ into the decoder.

At inference time a vector of zeros is the input of the LSTM decoder.

Interpretable multi-head attention



This layer is the same multi-head attention proposed in Transformer architecture.

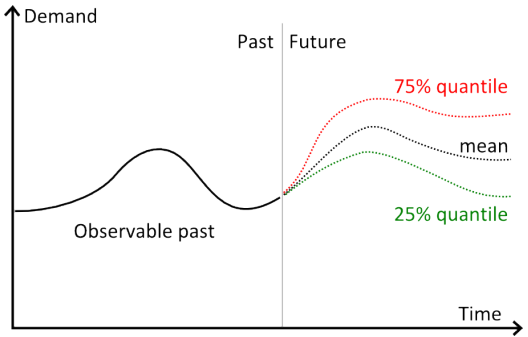
$$\text{InterpretableMultiHead}\left(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\right) = \tilde{\boldsymbol{H}}\boldsymbol{W}_H$$

But each head shares the same value weights and the heads are aggregated with the average instead of concatenation

$$\tilde{H} = 1/H \sum_{h=1}^{m_H} \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_Q^{(h)}, \boldsymbol{K}\boldsymbol{W}_K^{(h)}, \boldsymbol{V}\boldsymbol{W}_V\right)$$

$\boldsymbol{W}_V$ are value weights shared across all heads.

Quantile Regression



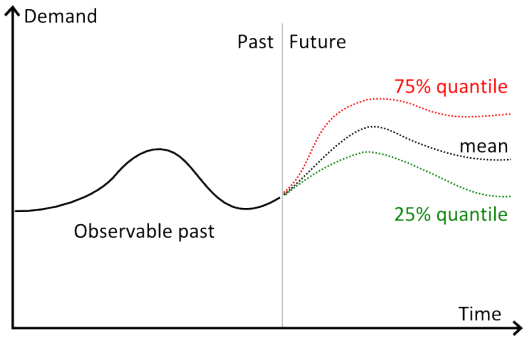
TFT generates prediction intervals on top of point forecasts. This is achieved by the simultaneous prediction of various percentiles (e.g. 10th, 50th and 90th) at each time step.

The prediction intervals are learned by jointly minimizing the quantile loss, summed across all quantile outputs:

$$QL(\boldsymbol{y}, \hat{\boldsymbol{y}}, \boldsymbol{q}) = \boldsymbol{q}(\boldsymbol{y} - \hat{y})_+ + (1 - \boldsymbol{q})(\hat{\boldsymbol{y}} - \boldsymbol{y})_+$$

Where $(.)_+ = max(0, .)$

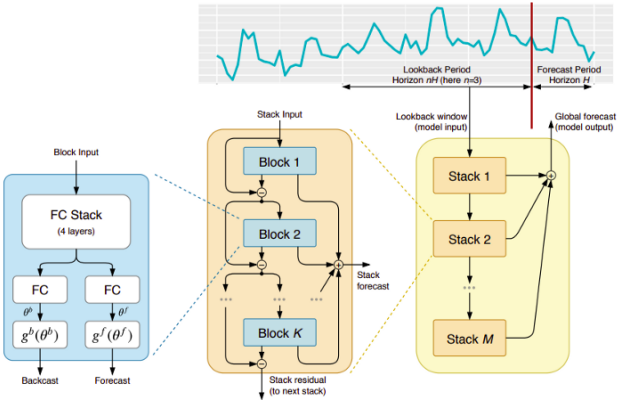
Quantile Regression



$$\mathcal{L}(\Omega, \boldsymbol{W}) =$$
$$= \sum_{y_t \in \Omega} \sum_{q \in \mathcal{Q}} \sum_{\tau=1}^{\tau_{\max}} \frac{QL\left(y_t, \hat{y}(q, t-\tau, \tau), q\right)}{M \tau_{\max}}$$

Where $\boldsymbol{\Omega}$ is the domain of training data containing $\boldsymbol{M}$ samples, $\boldsymbol{W}$ represents the weights of TFT and $\boldsymbol{Q} = \{0.1, 0.5, 0.9\}$ is the set of output quantiles.

N-Beats



N-Beats [4] is a block-based deep neural network model for univariate time series forecasting.

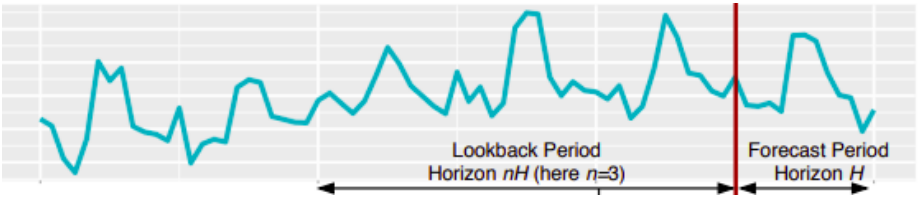
The main key principles of this architecture are:

- Being simple and generic
- Not rely on time-series-specific feature engineering or input scaling

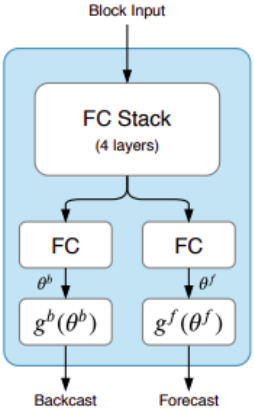
Input

The model takes a time serie $(x_0, x_1, ...x_t)$ as input, called **Lookback Period** and predict $(x_{t+1}, ..., x_{t+h})$ called **Forecast Period**.

h is the forecast window length.



Basic block

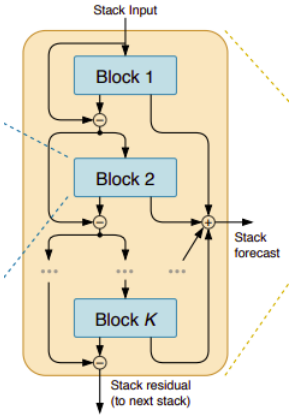


The basic building block consists of a 4-layers FC (fully connected) network with RELU nonlinearities.

It is then divided into two parts:

- 2-layers FC to predict the **lookback period**, so basically it learns to reconstruct the input.
- 2-layers FC to predict the **forecast period**.

Stack basic blocks



The basic blocks are organized into stacks using doubly residual stacking principle. The **lookback period output** is used to compute the residual and becomes the input of the next basic block.

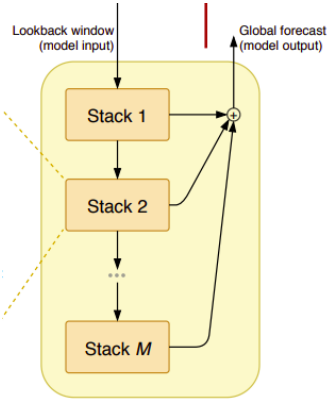
$$x_\ell = x_{\ell-1} - \hat{x}_{\ell-1}$$

Where ℓ indicate the ℓ-th block

The **forecast period outputs** from each block are simply aggregated:

$$\hat{y} = \sum_\ell \hat{y}_\ell$$

Combined stacks



Stacks are combined in a higher-level structure. Each stack ouputs a **residual vector** (this vector represents learnings not learnt by the stack) and a **forecast vector**.

The residual vector of each stack becomes the input of the next one and the forecast vectors are aggregated in the same way seen before.

The paper also propose a different configuration of the architecture, augmented with certain inductive biases, to be interpretable.

- **Trend model** A typical characteristic of trend is that most of the time it is a monotonic function, or at least a slowly varying function. In order to mimic this behaviour we constrain the last layer of the basic block:

$$\hat{y}_{s,\ell}^{tr} = \boldsymbol{T}\theta_{s,\ell}^{f}$$

where $\theta_{s,\ell}^{f}$ are polynomial coefficients predicted by a FC network of layer ℓ of stack s and $T = [1, t, ..., t^{p}]$ is the matrix of powers of t.

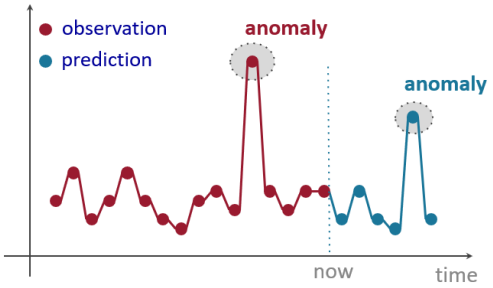
The paper also propose a different configuration of the architecture, augmented with certain inductive biases, to be interpretable.

- **Seasonality model** Typical characteristic of seasonality is that it is a regular, cyclical, recurring fluctuation. In order to mimic this behaviour we constrain the last layer of the basic block:

$$\hat{y}_{s,\ell}^{\text{seas}} = \boldsymbol{S}\theta_{s,\ell}^{f}$$

where $\theta_{s,\ell}^{f}$ are polynomial coefficients predicted by a FC network of layer ℓ of stack s and $S = [1, \cos(2\pi t), \ldots \cos(2\pi \lfloor H/2 - 1 \rfloor t)), \sin(2\pi t), \ldots, \sin(2\pi \lfloor H/2 - 1 \rfloor t))]$ is the matrix of sinusoidal waveforms.

Unsupervised Anomaly Detection on Time Series



An **anomaly**, or an **outlier**, is a data point which is significantly different from the rest of the data. [2]

Generally, the data in most applications is created by one or more generating processes that reflect the functionality of a system. When the underlying generating process behaves in an unusual way, it creates outliers.

- Classical approach:

  Detecting an anomaly based on **PCA** has been widely used in practice because the only information needed is a dataset describing the normal process operation. The most common PCA-based monitoring statistic is **Hotelling's T^2**:

$$T^2 = \sum_{i=1}^{n.comp} \frac{t_i^2}{s_i^2}$$

  Where s_i is the i-th standard deviation of the t_i score. For testing new data, when the value of **T** exceeds a threshold value then is a fault.

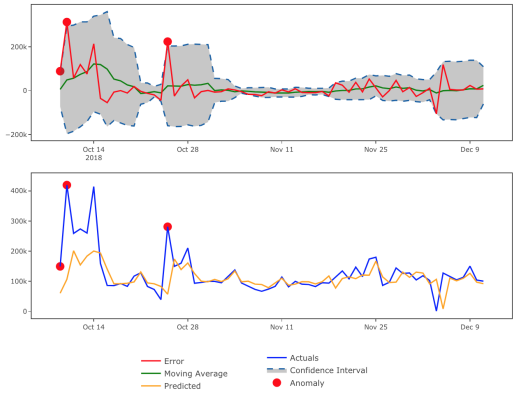
- Classical approach:

Q-statistic denotes the change of the events that are not explained by the model of principal components. It is a measure of the difference, or residual between a sample and its projection into the model.

$$Q_i = \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^{\mathrm{T}} = \boldsymbol{x}_i \left(\mathrm{I} - \mathrm{PP}^{\mathrm{T}} \right) x_i^{\mathrm{T}}$$

Where **P** is the matrix of principal components obtained using **PCA**. When a vector of new data is available, the Q-statistic is calculated and compared with a threshold value, if the confidence limit is violated then a fault is declared.

Unsupervised Anomaly Detection on Time Series



- Forecasting Based Approaches:

  In this methodology, a prediction is performed with a forecasting model (such as a deep learning model) for the next time period.

  When a new forecasted value is out of confidence interval, or exceeds a certain threshold, the sample is flagged as anomaly.

Credits I

These slides heavily borrow from a number of awesome sources. I'm really grateful to all the people who take the time to share their knowledge on this subject with others.

In particular:

- Stanford CS224n: Natural Language Processing with Deep Learning
  http://web.stanford.edu/class/cs224n/
- Understanding 1D and 3D Convolution Neural Network
  https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610
- The illustrated Transformer
  http://jalammar.github.io/illustrated-transformer/

Credits II

- The Positional Encoding
  https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- Hugging Face
  https://huggingface.co/
- N-BEATS https://kshavg.medium.com/
  n-beats-neural-basis-expansion-analysis-for-interpretable-time-series-for

References I

[1] S. Bai, J. Z. Kolter, and V. Koltun.
An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.
arXiv:1803.01271, 2018.

[2] A. Bl'azquez-Garc'ia, A. Conde, U. Mori, and J. Lozano.
A review on outlier/anomaly detection in time series data.
ArXiv, abs/2002.04236, 2020.

[3] B. Lim, N. Loeff, S. Arik, and T. Pfister.
Temporal fusion transformers for interpretable multi-horizon time series forecasting.
2020.

References II

[4] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio.
N-beats: Neural basis expansion analysis for interpretable time series forecasting.
In *International Conference on Learning Representations*, 2020.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin.
Attention is all you need.
In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.