Transformers applied to numerical simulator modelling

Max Cohen

March 09 2020

Outline

1. Neural networks
2. Deep learning for Time Series
3. Transformer
4. Application on Oze data

Definition

Consider a problem where we want to predict Y given an input X. A neural network is a function F_θ with parameters θ called "weights" that computes:

$$F_\theta(X) = \mathbb{P}_\theta(Y|X)$$

in a classification setting (Y lies in a discrete set) or

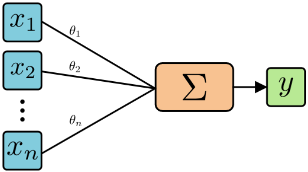
$$F_\theta(X) = \widehat{Y}$$

in a regression setting (Y lies in a subset of $\mathbb{R}^d$).

Weights are estimated using a dataset $(X^{(i)}, Y^{(i)})_{1 \leq i \leq m}$, by minimizing a loss function $L : (x, y) \mapsto \mathbb{R}^+$:

$$\theta \mapsto \frac{1}{n} \sum_{i=1}^{n} L(F_\theta(X^{(i)}), Y^{(i)}) .$$

Fully connected neuron



A linear transform followed by a nonlinear activation:

$$F_\theta(x) = \sigma(\theta \cdot x + b), \quad \theta \in \mathbb{R}^n, b \in \mathbb{R}\,,$$

where σ is a non-linearity, for instance a sigmoid function $\sigma : x \mapsto \frac{1}{1+e^{-x}}$ or a ReLU $\sigma : x \mapsto max(0, x)$.

Bigger architectures

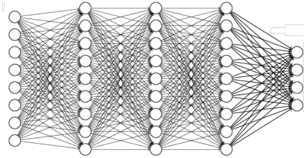


Figure: Fully connected network

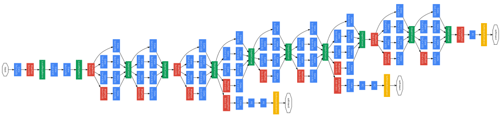
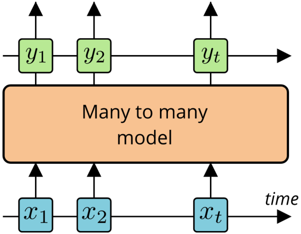


Figure: Inception network

Problem definition



We consider a **many to many regression problem**, for instance predicting the indoor temperature given weather data, or solving a Natural Language Processing (NLP) task.

$$X : (T, n)$$
$$Y : (T, d)$$

Example: consumptions data

→ **Predict consumptions and temperatures in a building**

- **Input** $X = (X_1, \dots, X_T)$:

  - Time series such as humidity, outside temperatures.
  - Received hourly.

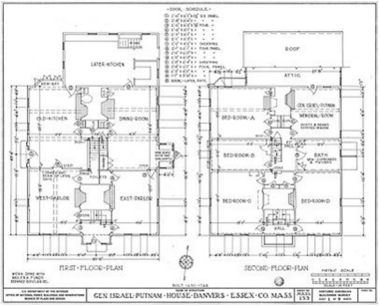
- **Output** $Y = (Y_1, \dots, Y_T)$:

  - Inside temperatures in several spots.
  - Consumptions (cooling, heating, ventilation).

- **Controlled system** :

  - The time series also depend on a building management system (not mentionned today).

Example: consumptions data



Model based on a **schematic view of the building** (geometry, etc.) and a **simulator for transient systems**.

Example: consumptions data

Physics based approaches to propose simulators are widespread in the most complex situations.

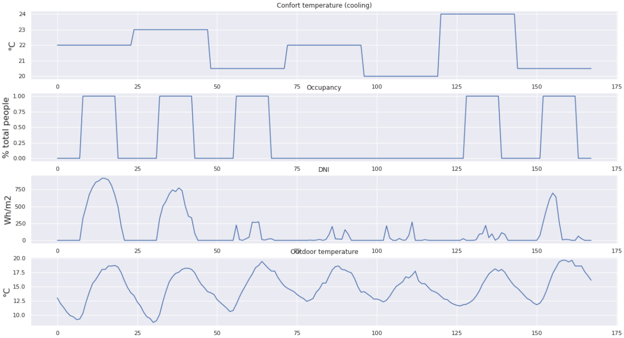
The main drawbacks of highly sophisticated softwares to simulate the behavior of transient systems: significant computational time required to train such models and integration of many noisy sources.

1. Millions of data obtained from a a physics based approach / software.
2. **(STEP 1)** Design a new flexible model (deep learning for time series) and train model parameters during an initial phase with these measurements.

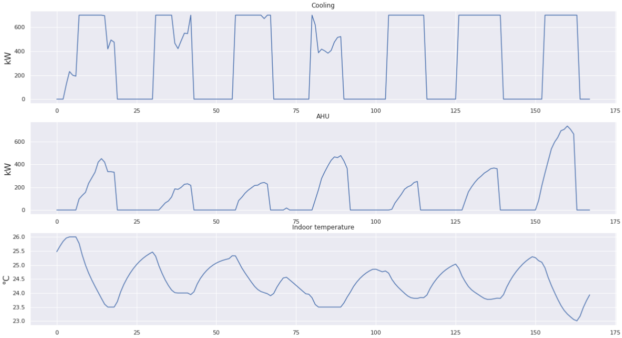
Example: consumptions data

1. Millions of data obtained from a a physics based approach / software.
2. **(STEP 1)** Design a new flexible model (deep learning for time series) and train model parameters during an initial phase with these measurements.
3. **(STEP 2a)** Analysis of the statistical predictive efficiency of the model. Uncertainty quantification in the prediction using sensor data.
4. **(STEP 2b)** Tuning of commands to ensure energy savings with no renovation works.

Example: input data



Example: output data



Recurrent Neural Networks

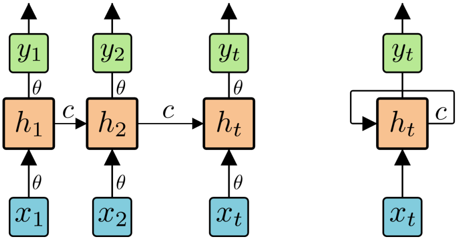
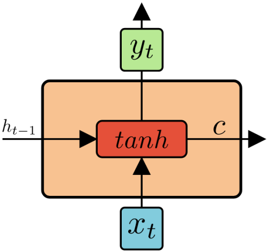


Figure: RNN architecture

Until recently, **recurrent neural networks (RNN)** had been established as the go-to architecture for sequence modelling. They easily scale to long sequences, with a complexity in $\mathcal{O}(T \cdot n^2)$.

Traditional architecture

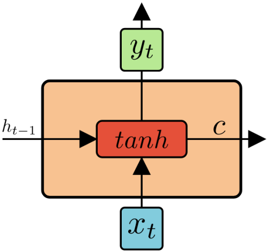


At time t, a **hidden state of the network** is computed as follows:

$$h_t = \sigma_h(W_x\, x_t + W_h\, h_{t-1} + b_h)\,,$$

where σ_h is a nonlinear activation function.
W_x, b_h and W_h are the unknown parameters of the state update.

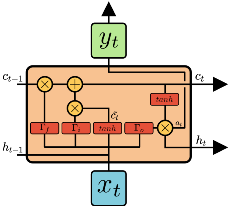
Traditional architecture



At time t, the **hidden state of the network** is used to predic Y_t as follows:

$$\widehat{y}_t = F_\theta(x_t, h_{t-1}) = \sigma_y(W_y \cdot h_t + b_y)\,.$$

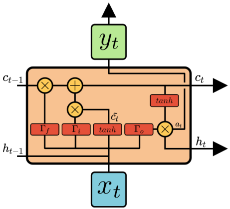
Long Short Term Memory (LSTM)



The LSTM cell is a more complex recurrent neural network. It contains three gates, input, forget, output gates and a memory cell. The **gates at time t** are:

$$\Gamma_i = \sigma(W_i[h_{t-1}, x_t] + b_i)$$
$$\Gamma_f = \sigma(W_f[h_{t-1}, x_t] + b_f)$$
$$\Gamma_o = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

Long Short Term Memory (LSTM)



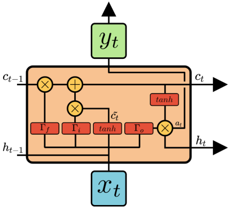
The previous hidden state h_{t-1} and the current input x_t are used to compute a candidate $\tilde{c}_t$:

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)$$

The **cell memory** c_t, is updated as:

$$c_t = \Gamma_i * \tilde{c}_t + \Gamma_f * c_{t-1}$$

Long Short Term Memory (LSTM)



The hidden state h_t, is computed as

$$h_t = \Gamma_o * \tanh(c_t)$$

The **predicted output** is:

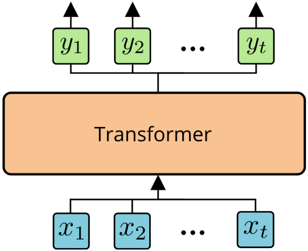
$$\widehat{y}_t = \sigma_y(W_y\, h_t + b_y)\,.$$

Introduction for NLP

Transformer were introduced for NLP as an alternative to sequential models. They rely on **attention mechanisms** to address the lack of long term memory of LSTM.

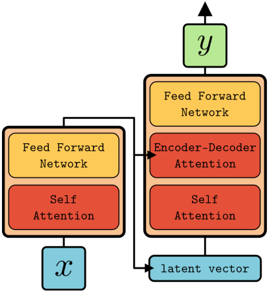
- Sequences are computed in parallel.
- Complexity is in $\mathcal{O}(T^2 \cdot n)$.
- Memory is replaced by attention.

Computing in parallel



$$F_\theta(x) = P(y_1 \cdots y_T | x_1 \cdots x_T)$$

Encoder - Decoder structure



The encoder computes a latent vector from the input data, which is fed to the decoder in order to predict the outputs. These sub-networks are trained jointly and are supposed to encourage the Transformer to learn a **meaningful representation** of the data.

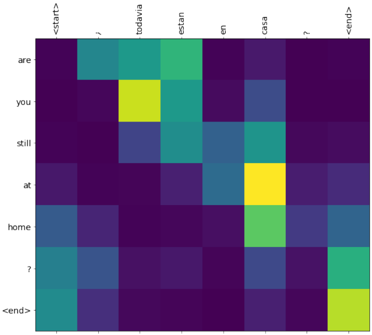
Feed Forward Network

The Feed Forward Network is a combination of linear transformations, with a ReLU activation function:

$$\texttt{FFN}(x) = max(0, x \cdot W_1 + b_1) \cdot W_2 + b_2$$

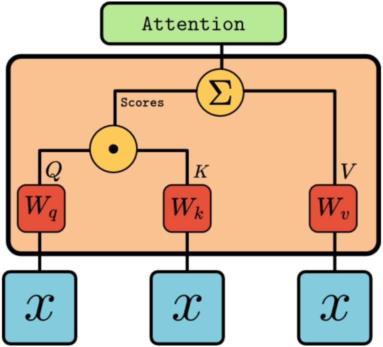
It act as the main **non linearity** of the Transformer.

Self attention mechanism - intuition



At each time step, what are the part of the entire sequence should we focus on to get the best interpretation ?

Self attention mechanism - illustration



Self attention mechanism

$$
\begin{aligned}
Q &= W_q x & (T, q) \quad & \texttt{Queries} \\
K &= W_k x & (T, k) \quad & \texttt{Keys} \\
V &= W_v x & (T, v) \quad & \texttt{Values} \\
\texttt{Scores} &= \frac{QK^T}{\sqrt{q}} & (T, T) \quad &
\end{aligned}
$$

With the softmax function $\texttt{softmax}(z) \mapsto \frac{e^{z_i}}{\sum_j e^{z_j}}$:

Attention

$$
Attention = \texttt{softmax}(\frac{Q \cdot K}{\sqrt{d_k}})V
$$

Decoder mechanism - single layer example

Transformers-based approaches propose a **model for the conditional distribution of a data Y_t given past observations**.

Data representations: for all $1 \leqslant s \leqslant t$ and all $1 \leqslant h \leqslant n_h$, define,

$$q^h(s) = W^{h,q} X_s\,, \quad \kappa^h(s) = W^{h,\kappa} X_s\,, \quad v^h(s) = W^{h,v} X_s\,,$$

where $W^{h,q}$, $W^{h,\kappa}$ and $W^{h,v}$ are unknown matrices.
Scores:

$$\begin{aligned} \mathrm{score}^h(t) &= (q^h(t))^T K^h\,, \\ \pi^h(t) &= \mathrm{softmax}\left(\mathrm{score}^h(t)/\sqrt{r}\right), \end{aligned}$$

where the columns of K^h are the $\kappa^h(t-s)$, $1 \leqslant s \leqslant t$.

Decoder mechanism - single layer example

Self attention: for all $1 \leqslant s \leqslant t$, as,

$$z^h(t) = \sum_{s=1}^{t} \pi_s^h(t) v^h(t-s) \,,$$

The output is then computed with a Layer normalization step:

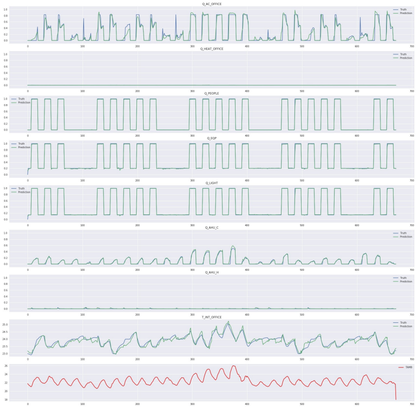
$$u(t) = \mathcal{N}(z(t) + X_t) \,,$$
$$\hat{Y}_t = \mathcal{N}(\texttt{FFN}(u(t)) + u(t)) \,,$$

where $\mathcal{N}$ is a normalizing function.

Implementation

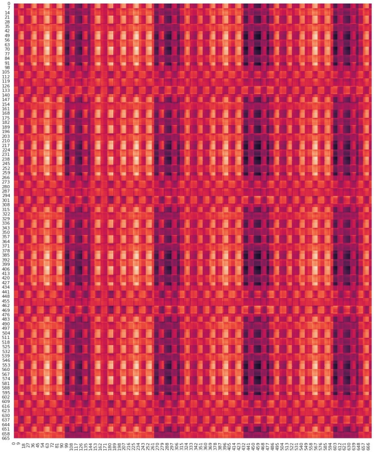
Our transformer model is implemented using **PyTorch**, the code is available at github.com/maxjcohen/transformer.
Because we do not yet have access to powerful **computing units**, some parameters have been lowered (latent dimension, query/key/value dimension).

Results: visual interpretation



- Visually, the predictions match the dataset;
- Interior temperature seems much harder to predict;
- Lack of local attention is visible (during the night for example).

Results: attention maps



We can identify day/night cycles, as well as week/weekend.