TAYSIR

User Manual

21 Mar 2023

Table of Content

About the competition

This competition is an on-line challenge on **extracting simpler models from already trained neural networks**. These neural nets are trained on sequential categorial (=symbolic) data. Some of these data are artificial, some come from real world problems (NLP, Bioinformatics, Software Engineering, etc.).

The competition offers 2 tracks:

Track 1: Binary classification. The neural nets were trained on data belonging to 2 classes. This corresponds to a language in the formal language theory sense.

Track 2: Language modelling. The neural nets were trained to provide a distribution over the potential next symbols given the beginning of a sequence (a prefix). They then are seen as models assigning a probability to any sequence (density estimator):

$$p(\rtimes a_1a_2\ldots a_n\ltimes) = p(a_1|\rtimes)p(a_2|\rtimes a_1) \ldots p(\ltimes|\rtimes a_1a_2\ldots a_n)$$

where a_i are the integers forming the sequence and $\rtimes$ and $\ltimes$ are the start and final symbols, respectively.

The website presenting the competition can be found at https://remieyraud.github.io/TAYSIR/

The platform hosting the competition, where you can access the data and submit your simple models is codalab: https://codalab.lisn.upsaclay.fr/ You will need to create an account there to be able to participate. This platform has its drawbacks - like the need for manual acceptance to be able to participate, or a useless timeline right in the middle of the page - but it is helping us a lot.

There are **two competitions on that platform**: one for the first Track, one for the second. We had a third one to beta test both our framework and yours.
Each competition is divided into **several phases: each phase corresponds to one Neural Net** from which to extract a simpler model. Phases might not all start at the same time, but they will all end on April 30th.
Beta link: https://codalab.lisn.upsaclay.fr/competitions/11055 (finished on March 2nd)
Track 1 link: https://codalab.lisn.upsaclay.fr/competitions/11249
Track 2 link: https://codalab.lisn.upsaclay.fr/competitions/11619

We provide a **starter kit with notebooks** to show you how to play with our models and how to generate the archive that will be needed to submit your surrogate models.

We created a **discord server** dedicated to the competition: https://discord.gg/ZubJfV2gKd

The Datasets

All the data used for training the neural nets are **sequential and symbolic**. Practically, they are sequences of integers.
We do not provide the training datasets since this is not a learning competition. However, we provide the datasets that were used for **validation** (around 10% of all the data).
The provided files are in the following format :

```
[Number of sequences] [Alphabet size]
[Length of sequence] [List of symbols]
[Length of sequence] [List of symbols]
[Length of sequence] [List of symbols]
…
[Length of sequence] [List of symbols]
```

For example the following dataset:

```
5 10
6 8 6 5 1 6 7 4 9
12 8 6 9 4 6 8 2 1 0 6 5 9
7 8 9 4 3 0 4 9
4 8 0 4 9
8 8 1 5 2 6 0 5 3 9
```

is composed of 5 sequences and has an alphabet size of 10 (so symbols are integers between 0 and 9) and the first sequence is composed of 6 symbols: [8, 6, 5, 1, 6, 7, 4, 9]. Notice that here 8 is the start symbol and 9 is the end symbol. All sequences start and end with them. For instance, in this example, the empty sequence is coded [8, 9].

The Neural Nets

The complete code used for the models can be found in the folder containing each model.

RNNs

We provide the RNNs as MLFlow pytorch models. This allows the models to be used cross-platform with easy access to their inner processes.
The architectures cover a large spectrum of hyperparameters:

- Type of recurrent layers: SRN, GRU, LSTM[1]
- Number of neurons per recurrent layers
- Number of recurrent and dense layer(s)
- Batch size
- Patience

None of these architectures contains an embedding layer as we want to focus on the recurrent part.

The implementation of the RNN is given in the file *tnetwork.py* that you can find inside all MLFlow models.
Here are the main variable of the class:

- *n_layers*: number of recurrent layers
- *neurons_per_layer*: number of cells per recurrent layer
- *dropout*: whether dropout is used
- *cell_type*: the type of recurrent cell
- *final_type*: the type of feedforward used for the final dense layer(s)
- *bidirectional*: whether the network is bidirectional
- *split_dense*: whether there is one or two dense final layer(s)
- *task*: "binary" or "lm" depending on the competition track
- *hides_pairs*: boolean, True if the cell type is LSTM

The module comes with a handful of functions that we hope will be useful:

one_hot_encode(self, word): This function takes a sequence as a list of integers and returns a tensor containing the sequence one-hot encoded (without padding)

predict(self, x): This function takes a one-hot encoded sequence (or several) and returns the prediction of the model. This prediction is a float (or a tensor of floats) that contains:

- 0 or 1 for Track 1 (Binary Classification).
- The probability assigned to the sequence for Track 2 (Language Modelling). This probability is defined as $p(\rtimes a_1a_2\ldots a_n\ltimes) = p(a_1|\rtimes)p(a_2|\rtimes a_1) \ldots p(\ltimes|\rtimes a_1a_2\ldots a_n)$ where a_i are the integers forming the sequence and $\rtimes$ and $\ltimes$ are the start and final symbols, respectively.

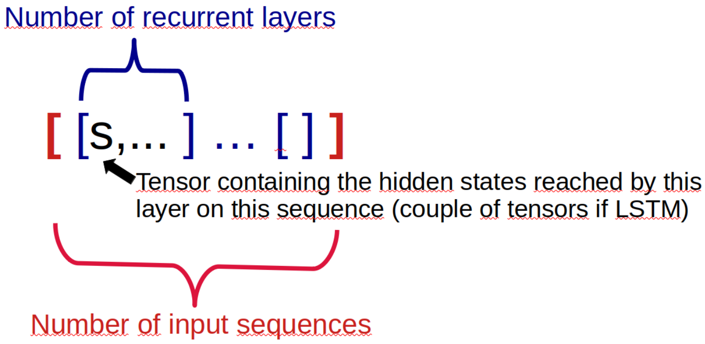
forward(self, x, hidden=None, full_ret=False): usual forward. *hidden=None* corresponds to the initialisation of the hidden state of the model. You do not need to know what *full_ret* is doing ;-)

forward_bin(self,x, hidden=None, full_ret=False): This function is the usual forward for binary classification. However, it can be used on a RNN trained for language modelling: in this case it provides a tuple whose first vector is containing the probability of each symbol to be the next one once *x* is fully parsed.

[1] The pytorch LSTM does not give access to the values of the carry during the parsing, so we use our own designed module *lstmx.py*, based on torch.nn.LSTMCell

forward_lm(self, x, hidden=None, full_ret=False): This function provides a tensor containing vectors of next symbols probability after each symbol in *x*.

reached_hidden(self, x, hidden=None): This function parses a 1-hot encoded sequence (or a list of sequences) from the hidden state passed in argument (uses initial state if None, otherwise requires a list of tensors, one per recurrent layer) and returns a list of lists of hidden states encountered. The number of lists is the number of words. Each list is made of a list of tensors whose number corresponds to the number of hidden recurrent layers. Each tensor contains the hidden states encountered at this layer while parsing the word (if LSTM, these elements are tuples of two tensors, one for the hidden state and one for the carry). The dimension of one tensor is (1, sequence length, size of the layer). An example of its use is given in the Track 1 notebook for baseline.


Number of recurrent layers
[[S,...] ... [ ]]
Tensor containing the hidden states reached by this layer on this sequence (couple of tensors if LSTM)
Number of input sequences

feed_dense(self, h): This function takes the output of the last recurrent layer and computes the output of the dense layer(s)

Transformers

The models that are going to be offered are DistilBERT trained on the hidden datasets from scratch. The reason why DistilBERT is used is that it has a simple and standard Transformer-type structure. The models were adjusted to have smaller hyperparameters than the original version as far as their test accuracies did not largely drop, so that participants easily dealt with them with lower computational cost.

Submission

We provide a **tool** that allows you to save any python function in the MLFlow needed format. You then will need to wait for us to run your model on the (not known to participants) test set (our server works in a FIFO mode and assigns a **max of 5 minutes to each submission**). When this computation is done, you will see the feedback on the website (its scores if everything goes well, an error log otherwise).

For each Track, the maximal **number of submissions per day is fixed** (20) and will not vary during the competition. There is also **a limit on the total amount of submissions per Track (1000)**. This aims at having a smoothly working server. But then you have to use them with sparsity!

Your submission will appear in the leaderboard **only if its global score** (see next Section) is **better than the previous best one** you had on that particular phase (e.g., 1.1)

Evaluation

The submissions will be evaluated on 3 criterias:

- **Score:** This is the error rate for the binary classification and the mean square error (time 10^6 for magnitude and readability) for language modelling. We compare the output of your surrogate model with the one of the trained Neural Net.
- **Memory usage:** The memory footprint of your model during a prediction, in mebibytes.
- **CPU time:** The CPU time spent by your model during a prediction, in milliseconds.

Your goal is to reach the smallest score possible on all these criterias.

Based on these 3 scores, we compute a **global score** this way:

$$\frac{1}{2}\boldsymbol{Score} \ + \frac{1}{4}\boldsymbol{memory\_usage\_ratio} + \frac{1}{4}\boldsymbol{CPU\_time\_ratio}$$

Where *Score* is either the error rate (Track1) or MSE*10^6 (Track 2), *memory_usage_ratio* [*resp. CPU_time_ratio*] is the division of the memory usage [*resp.* CPU time] of your model by the one of the original Neural Net on the same data set.

Advice about Memory Usage. There exists a "fixed offset" which is due to the overhead of using MLflow to run your models: in our local test it is around 120mb (with python 3.8 and MLFlow 1.25.1). It is possible that another combination will obtain a smaller offset: we did not grid search these parameters (because that is not really what the competition is about).
A detail that can help you is that MLFlow is using cloudpickle under the hood. And when executed in a notebook, cloudpickle is serialising every variables that have been declared since the notebook is run, this is how MLFlow can magically run any model but the downside to that is that if you declare a variable that is not used by your model then it will be loaded in memory for nothing. So in order to reduce your memory footprint, we advise to **run ONLY the cells that are needed to execute your model (or run save_function in a dedicated python file).**

Advice about CPU time. The first time you are submitting something to a Track, a virtual environment is created for you. This means that the very first submission can take more time than supposed, because the CPU time is accounting for the installation of the eventual packages needed. The same environment will then be used for all your submissions (in a given Track) so this could happen only once.

Video

A video presenting the competition can be found here:
https://www.youtube.com/watch?v=LY36ek4EcwA
Warning: this talk was given months before the competition, some elements are not relevant anymore - starting with the name of the competition that changed! Another difference is that

we are **only providing MLFlow models** (but they are able to do way more than what we thought at the time of this talk). **Track 2 is only about language modelling neural nets** (and not general regression as stated in the talk). The timeline and the URL also changes, but if you are reading this, you already know that ;-)

Timeline

February 2023:	Beginning of the competition, starting with Track Beta, then Track 1, and finally Track 2
April 30th 2023:	End of the competition
May 15th 2023:	Submission deadline for the extended abstracts presenting your work
July 10-13th 2023:	ICGI 2023 in Morocco, half a day dedicated to the competition