ExplaiNN: interpretable and transparent neural networks for genomics

A tutorial, by **Oriol Fornes**

Research Associate & Deputy Group Leader @ Wasserman Lab
University of British Columbia

December, 2022

oriol@cmmt.ubc.ca
@OFornes

What is ExplaiNN?

- ExplaiNN (Explainable Neural Networks) is a glass-box deep learning model for genomic tasks trained on one-hot encoded sequences
- ExplaiNN performs well compared to more complex models and provides similar interpretation to more complex and time-consuming approaches
- Preprint: https://doi.org/10.1101/2022.05.20.492818
- GitHub: https://github.com/wassermanlab/ExplaiNN

Contents of the GitHub repository

- **conda** – contains a bash script to create a conda environment for the tutorial
- **data** – contains different datasets (e.g. for this tutorial)
- **explainn** – i.e., the explain library
- **notebooks** – contains different Jupyter notebooks (e.g. paper figures)
- **scripts** – contains different Python scripts (e.g. to train models)

The ExplaiNN library

- explainn/
  - interpretation/
    - **interpretation.py** – contains code to interpret ExplaiNN models
  - models/
    - **networks.py** – contains different models (e.g. ExplaiNN, DanQ, …)
  - train/
    - **train.py** – contains code to train ExplaiNN models
    - **test.py** – contains code to test ExplaiNN models
  - utils/
    - **tools.py** – contains general functions (e.g. one-hot encoder)

Conda prerequisites

- Make sure that `conda` is available

```
conda init bash
source ~/.bashrc
```

- Add `libmamba` solver and custom packages directory

```
conda config --set solver libmamba
conda config --add pkgs_dirs ${HOME}/.conda/pkgs
```

Installing ExplaiNN

- Clone ExplaiNN from GitHub and switch to the beta branch

```
git clone https://github.com/wassermanlab/ExplaiNN.git
cd ExplaiNN
```

- Install all requirements via conda

```
./conda/create-conda-env.sh
conda activate explainn
```

- Verify that the requirements were installed correctly

`>>> import torch` ← *from the Python console*

`>>> torch.cuda.is_available()` ← *should return "True"*

`tomtom` ← *from the Terminal; should return the help message*

Installing ExplaiNN

- Install the ExplaiNN library (developer mode)

```
python setup.py install
```

- Verify that the ExplaiNN library was installed correctly
  - On GPURTX / GPURTX-2:

```
jupyter notebook --notebook-dir=./notebooks --no-browser --port=XXXX
```

  - On your local machine:

```
ssh -N -f -L localhost:YYYY:localhost:XXXX user@gpurtx.cmmt.ubc.ca
open http://localhost:YYYY/notebooks/test.ipynb
```

Where XXXX and YYYY can be any two numbers

Python scripts (e.g. ExplaiNN library wrappers)

- scripts/
  - **train.py** – trains ExplaiNN models
  - **interpret.py** – interprets ExplaiNN models
  - **test.py** – provides performance of ExplaiNN models on the test set
  - **predict.py** – computes predictions using ExplaiNN models for a set of sequences in a FASTA file

TSV

The ExplaiNN TSV file format

- Files can be gzipped
- 1st column – sequence identifier
- 2nd column – the sequence
- 3rd to nth columns – i.e., the values for the different classes

ERR1002407.188226	AATTAGCTCACATTCCCTGCGTGGCGCAACATACGCTGCG	0.0	0.0	1.0	0.0
ERR1002409.315766	CATTCACCGCGAGCTACCTAGCGCAGTGCATTGCACAACG	0.0	0.0	0.0	1.0
ERR1002405.154226	TTCTAGCGTCTCACCAGCATGGCGCAACTTCAGGATTGCG	0.0	1.0	0.0	0.0
ERR1002403.79059	GCATGGAACATATAACGTAGAAAAACCCACCGACACAGGG	1.0	0.0	0.0	0.0
ERR1002405.194629	GCATTGCTTGCATCATAGATACACGGCGCATCCGCCACTA	0.0	1.0	0.0	0.0
ERR1002407.57779	CGCCTACAAAGGAGCACGCACTTACGCAATGAGCCGTCCA	0.0	0.0	1.0	0.0
ERR1002407.418287	ACCTTGGCCCTGTCGATGGTCCACACGAGAGTATTGCGAA	0.0	0.0	1.0	0.0
ERR1002405.78846	GAACAATTACCTCTCCTAGCGGGTAAAGACCCGTTGCAAA	0.0	1.0	0.0	0.0
ERR1002405.187854	GCAAGATCTGCAGTGTTGCCCCGCCGTCACACCACGCCAT	0.0	1.0	0.0	0.0
ERR1002407.111275	GAGTGATGCAAGTCCCAGAGTGTACTGTACACTAGTTTAT	0.0	0.0	1.0	0.0
…					

Parsers from different formats to the ExplaiNN format

- scripts/parsers/
  - **bed2explainn.py** – from BED to ExplaiNN train/validation/test files
  - **fasta2explainn.py** – from FASTA to ExplaiNN train/validation/test files
  - **fastq2explainn.py** – from FASTQ to ExplaiNN train/validation/test files
  - **pbm2explainn.py** – from PBM to ExplaiNN train/validation/test files
  - **json2explainn.py** – from JSON to ExplaiNN train/validation/test files

Utilities

- scripts/utils/
  - **match-seqs-by-gc.py** – given two or more FASTA files, subsample the same number of sequences from each file while accounting for the %GC

JSON

  - **subsample-seqs-by-gc.py** – subsample *n* sequences from FASTA file while accounting for the original %GC distribution
  - **resize.py** – resizes intervals in a BED file to desired size
  - **jaspar2logo.py** / **meme2logo.py** – convert PWMs in JASPAR / MEME formats to motif logos
  - **meme2clusters.py** / **tomtom.py** – clusters PWMs in MEME format based on Tomtom similarities

Example 1: training a ExplaiNN model that predicts the binding of CTCF to the human genome

- Download non-redundant human CTCF ChIP-seq peaks from ReMap

```
cd ./data/tutorial
URL=https://remap.univ-amu.fr/storage/remap2022/hg38/MACS2/TF/CTCF
wget -P CTCF ${URL}/remap2022_CTCF_nr_macs2_hg38_v1_0.bed.gz
```

output: `./CTCF/remap2022_CTCF_nr_macs2_hg38_v1_0.bed.gz`

- Download the human genome assembly hg38 using genomepy

```
genomepy install -p UCSC -g ./genomes -t 8 -f hg38
```

output: (folder "`./genomes/hg38/`") `hg38.fa`, `hg38.fa.sizes`, etc.

- Download the JASPAR CORE collection of vertebrates profiles

```
URL=https://jaspar.genereg.net/download/data/2022/CORE
wget -P JASPAR ${URL}/JASPAR2022_CORE_vertebrates_non-redundant_pfms_meme.txt
```

output: `./JASPAR/JASPAR2022_CORE_vertebrates_non-redundant_pfms_meme.txt`

Example 1: training a ExplaiNN model that predicts the binding of CTCF to the human genome

- Extract the ChIP-seq peak summits

```
zless ./CTCF/remap2022_CTCF_nr_macs2_hg38_v1_0.bed.gz | \
    awk '{print $1"\t"$7"\t"$8}' > ./CTCF/CTCF_summits.bed
```

output: `./CTCF/CTCF_summits.bed`

- Extend the summits 100 bp in each direction for a total size of 201 bp

```
PY_SCRIPT=../../scripts/utils/resize.py
${PY_SCRIPT} -o ./CTCF/CTCF_201bp.bed \
    ./CTCF/CTCF_summits.bed ./genomes/hg38/hg38.fa.sizes 201
```

output: `./CTCF/CTCF_201bp.bed`

- Get the FASTA sequences of the resized peaks

```
bedtools getfasta -fi ./genomes/hg38/hg38.fa -fo ./CTCF/CTCF_201bp.fa \
    -bed ./CTCF/CTCF_201bp.bed
```

output: `./CTCF/CTCF_201bp.fa`

Example 1: training a ExplaiNN model that predicts the binding of CTCF to the human genome

- Subsample 10K sequences for training (i.e., for training faster)

```
PY_SCRIPT=../../scripts/utils/subsample-seqs-by-gc.py
${PY_SCRIPT} -o ./CTCF/CTCF_201bp_10K.fa --subsample 10000 ./CTCF/CTCF_201bp.fa
```

output: `./CTCF/CTCF_201bp_10K.fa`

- Create training/validation/test splits (% = 80/10/10); negative sequences are obtained by dinucleotide shuffling CTCF peak sequences using BiasAway

```
PY_SCRIPT=../../scripts/parsers/fasta2explainn.py
${PY_SCRIPT} -o ./CTCF/ -p CTCF ./CTCF/CTCF_201bp_10K.fa
```

output: (folder "`./CTCF/`") `CTCF.train.tsv.gz`, `CTCF.validation.tsv.gz`, `CTCF.test.tsv.gz`

Example 1: training a ExplaiNN model that predicts the binding of CTCF to the human genome

- Train an ExplaiNN model that predicts the binding of CTCF

```
PY_SCRIPT=../../scripts/train.py
OUT_DIR=./ExplaiNN/CTCF
${PY_SCRIPT} -o ${OUT_DIR} --input-length 201 --criterion BCEWithLogits \
    --rev-complement ./CTCF/CTCF.train.tsv.gz ./CTCF/CTCF.validation.tsv.gz
```

output: (folder "`./ExplaiNN/CTCF/`") `losses.tsv`, `model_epoch_best_3.pth` (name can differ), `parameters-train.py.json`

- Measure the model's performance on the test set

```
PY_SCRIPT=../../scripts/test.py
${PY_SCRIPT} -o ${OUT_DIR} ${OUT_DIR}/model_epoch_best_3.pth \
    ${OUT_DIR}/parameters-train.py.json ./CTCF/CTCF.test.tsv.gz
```

output: `./ExplaiNN/CTCF/performance-metrics.tsv`

- The performance of the model on the test set is **~0.85**

Example 1: training a ExplaiNN model that predicts the binding of CTCF to the human genome

- Interpret the model

```
PY_SCRIPT=../../scripts/interpret.py
${PY_SCRIPT} -o ${OUT_DIR} --exact-match \
    ${OUT_DIR}/model_epoch_best_3.pth ${OUT_DIR}/parameters-train.py.json \
    ./CTCF/CTCF.train.tsv.gz
```

output: (folder "`./ExplaiNN/CTCF/`") `filters.meme`, `importances.tsv`, `importances.tsv.gz`, `weights.tsv`

- Obtain a logo for each filter in PNG format (option "-f")

```
PY_SCRIPT=../../scripts/utils/meme2logo.py
${PY_SCRIPT} -c 8 -f png -o ${OUT_DIR}/logos ${OUT_DIR}/filters.meme
```

output: (folder "`./ExplaiNN/CTCF/logos/`") `filter0.fwd.png`, `filter0.rev.png`, etc.

Example 1: training a ExplaiNN model that predicts the binding of CTCF to the human genome

- Visualize the logos of CTCF-like filters

```
PY_SCRIPT=../../scripts/utils/tomtom.py
${PY_SCRIPT} -c 8 -o ${OUT_DIR}/tomtom ${OUT_DIR}/filters.meme \
    ./JASPAR/JASPAR2022_CORE_vertebrates_non-redundant_pfms_meme.txt
```

output: `./ExplaiNN/CTCF/tomtom/tomtom.tsv.gz`

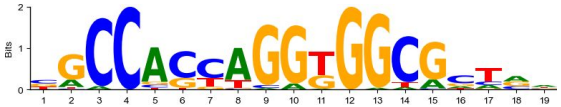
```
zgrep MA0139.1 ${OUT_DIR}/tomtom/tomtom.tsv.gz
```

filter31	MA0139.1	-2	4.27616e-09	3.59625e-06	3.57994e-06	...
filter21	MA0139.1	-1	1.33438e-10	1.12221e-07	2.199e-07	...
filter3	MA0139.1	-1	6.02406e-09	5.06624e-06	9.93355e-06	...

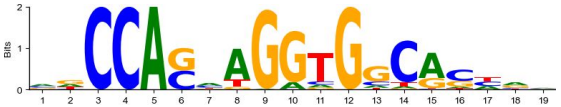
(results can differ)

filter31

filter21



filter3



Example 2: training a more realistic ExplaiNN model that predicts the binding of CTCF to the human genome

- To train the previous ExplaiNN model, we used a negative set of CTCF sequences (i.e., not bound) obtained by dinucleotide shuffling
- For this example, we will use a more realistic negative set obtained from subsampling accessible regions across 733 human biosamples from ENCODE that do not overlap any CTCF ChIP-seq peak

```
URL=https://www.encodeproject.org/files/ENCFF503GCK/@@download
FILE=ENCFF503GCK.tsv
wget -P ENCODE ${URL}/${FILE}
```

output: `./ENCODE/ENCFF503GCK.tsv`

- Remove accessible regions that overlap with CTCF peaks

```
tail -n +2 ENCODE/${FILE} | bedtools subtract -A -a - \
    -b ./CTCF/remap2022_CTCF_nr_macs2_hg38_v1_0.bed.gz > ENCODE/no_CTCF.bed
```

output: `./ENCODE/no_CTCF.bed`

Example 2: training a more realistic ExplaiNN model that predicts the binding of CTCF to the human genome

- Extract the summits and extend them 100 bp in each direction (size = 201 bp)

```
less ENCODE/no_CTCF.bed | awk '{print $1"\t"$7-1"\t"$7}' \
    > ./ENCODE/no_CTCF_summits.bed
PY_SCRIPT=../../scripts/utils/resize.py
${PY_SCRIPT} -o ./ENCODE/no_CTCF_summits_201bp.bed \
    ./ENCODE/no_CTCF_summits.bed ./genomes/hg38/hg38.fa.sizes 201
```

output: (folder "./ENCODE/") no_CTCF_summits.bed, no_CTCF_summits_201bp.bed

- Get the FASTA sequences of the resized regions

```
bedtools getfasta -fi ./genomes/hg38/hg38.fa -bed \
    ./ENCODE/no_CTCF_summits_201bp.bed -fo ./ENCODE/no_CTCF_summits_201bp.fa
```

output: ./ENCODE/no_CTCF_summits_201bp.fa

Example 2: training a more realistic ExplaiNN model that predicts the binding of CTCF to the human genome

- Subsample 100K sequences for training by matching the %GC content between CTCF peaks (i.e., positives) and ENCODE regions (i.e., negatives)

```
PY_SCRIPT=../../scripts/utils/match-seqs-by-gc.py
${PY_SCRIPT} -o ./CTCF/CTCF+ENCODE_201bp_100K.json -s 100000 \
    ./CTCF/CTCF_201bp.fa ./ENCODE/no_CTCF_summits_201bp.fa
```

output: `./CTCF/CTCF+ENCODE_201bp_100K.json`

- Create training/validation/test splits (% = 80/10/10)

```
PY_SCRIPT=../../scripts/parsers/json2explainn.py
${PY_SCRIPT} -o ./CTCF/ -p CTCF+ENCODE ./CTCF/CTCF+ENCODE_201bp_100K.json
```

output: (folder "`./CTCF/`") `CTCF+ENCODE.train.tsv.gz`, `CTCF+ENCODE.validation.tsv.gz`, `CTCF+ENCODE.test.tsv.gz`

Example 2: training a more realistic ExplaiNN model that predicts the binding of CTCF to the human genome

- Train, test and interpret a second ExplaiNN model that predicts CTCF binding

```
PY_SCRIPT=../../scripts/train.py
OUT_DIR=./ExplaiNN/CTCF+ENCODE
${PY_SCRIPT} -o ${OUT_DIR} --input-length 201 --criterion BCEWithLogits \
    --rev-complement ./CTCF/CTCF+ENCODE.train.tsv.gz \
    ./CTCF/CTCF+ENCODE.validation.tsv.gz
PY_SCRIPT=../../scripts/test.py
${PY_SCRIPT} -o ${OUT_DIR} ${OUT_DIR}/model_epoch_best_5.pth \
    ${OUT_DIR}/parameters-train.py.json ./CTCF/CTCF+ENCODE.test.tsv.gz
PY_SCRIPT=../../scripts/interpret.py
${PY_SCRIPT} -o ${OUT_DIR} --exact-match \
    ${OUT_DIR}/model_epoch_best_5.pth ${OUT_DIR}/parameters-train.py.json \
    ./CTCF/CTCF+ENCODE.train.tsv.gz
```

output: (folder "`./ExplaiNN/CTCF+ENCODE/`") `losses.tsv`, `model_epoch_best_5.pth` (name can differ), `parameters-train.py.json`, `filters.meme`, `importances.tsv`, etc.

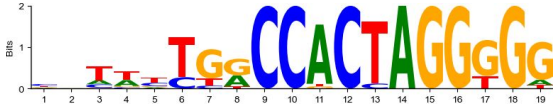
Example 2: training a more realistic ExplaiNN model that predicts the binding of CTCF to the human genome

- This time, the performance of the model on the test set is **~0.7** (it is harder to make predictions on genomic sequences than on shuffled sequences)
- Again, the model learned multiple instances of the CTCF motif

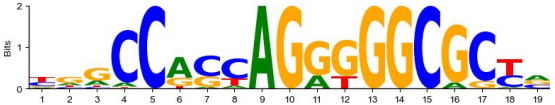
```
PY_SCRIPT=../../scripts/utils/meme2logo.py
${PY_SCRIPT} -c 8 -f png -o ${OUT_DIR}/logos ${OUT_DIR}/filters.meme
PY_SCRIPT=../../scripts/utils/tomtom.py
${PY_SCRIPT} -c 8 -o ${OUT_DIR}/tomtom ${OUT_DIR}/filters.meme \
      ./JASPAR/JASPAR2022_CORE_vertebrates_non-redundant_pfms_meme.txt
zgrep MA0139.1 ${OUT_DIR}/tomtom/tomtom.tsv.gz
```

filter81	MA0139.1	-5	2.25891e-06	0.00189975	0.00189975	...
filter50	MA0139.1	0	1.05544e-16	8.87626e-14	1.75871e-13	...
filter56	MA0139.1	0	6.75372e-17	5.67988e-14	1.1315e-13	...

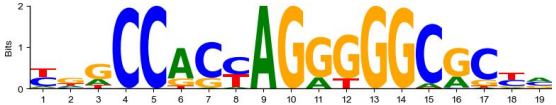
filter81



filter50



filter56



Example 2: training a more realistic ExplaiNN model that predicts the binding of CTCF to the human genome

- Now, let's use the model to make predictions of CTCF binding
- Download CTCF and accessible regions in the mouse genome

```
URL=https://downloads.wenglab.org/cCREs
FILE=mm10-CTCF.bed
wget -P ENCODE ${URL}/${FILE}
URL=https://www.encodeproject.org/files/ENCFF910SRW/@@download
FILE=ENCFF910SRW.tsv
wget -P ENCODE ${URL}/${FILE}
```

output: (folder "`./ENCODE/`") `mm10-CTCF.bed`, `ENCFF910SRW.tsv`

- Download the mouse genome assembly mm10 using genomepy

```
genomepy install -p UCSC -g ./genomes -t 8 -f mm10
```

output: (folder "`./genomes/mm10/`") `mm10.fa`, `mm10.fa.sizes`, etc.

Example 3: training ExplaiNN on the AI-TAC dataset

- Download the AI-TAC dataset

```
URL=https://www.dropbox.com/s
wget -P AI-TAC ${URL}/r8drj2wxc07bt4j/ImmGenATAC1219.peak_matched.txt
wget -P AI-TAC ${URL}/7mmd4v760eux755/mouse_peak_heights.csv
```

output: (folder "./AI-TAC/") ImmGenATAC1219.peak_matched.txt, mouse_peak_heights.csv

- Extract the ATAC-seq peak centers

```
cut -f 2-4 AI-TAC/ImmGenATAC1219.peak_matched.txt | \
    awk '{C=$2+int(($3-$2)/2);printf $1"\t%.0f\t%.0f\n",C-1,C;}' \
    > ./AI-TAC/AI-TAC_centers.bed
```

output: ./AI-TAC/AI-TAC_centers.bed

Example 3: training ExplaiNN on the AI-TAC dataset

- Extend the centers 125 bp in each direction (size = 251 bp, as in AI-TAC)

```
PY_SCRIPT=../../scripts/utils/resize.py
${PY_SCRIPT} -o ./AI-TAC/AI-TAC_251bp.bed \
    ./AI-TAC/AI-TAC_centers.bed ./genomes/mm10/mm10.fa.sizes 251
```

output: `./AI-TAC/AI-TAC_251bp.bed`

- Get the FASTA sequences of the resized peaks

```
bedtools getfasta -fi ./genomes/mm10/mm10.fa -fo ./AI-TAC/AI-TAC_251bp.fa \
    -bed ./AI-TAC/AI-TAC_251bp.bed
```

output: `./AI-TAC/AI-TAC_251bp.fa`

Example 3: training ExplaiNN on the AI-TAC dataset

- Create training/validation splits (% = 90/10)

```
grep -v "^>" ./AI-TAC/AI-TAC_251bp.fa | cut -c 2- > ./AI-TAC/AI-TAC_sequences.txt
tail -n +2 AI-TAC/mouse_peak_heights.csv | cut -d "," -f 1 \
    > ./AI-TAC/AI-TAC_ids.txt
tail -n +2 AI-TAC/mouse_peak_heights.csv | cut -d "," -f 2- | \
    tr "," "\t" > ./AI-TAC/AI-TAC_heights.txt
paste -d "\t" ./AI-TAC/AI-TAC_ids.txt ./AI-TAC/AI-TAC_sequences.txt \
    ./AI-TAC/AI-TAC_heights.txt > AI-TAC/AI-TAC_251bp.tsv
awk 'BEGIN {srand()} {f = FILENAME (rand() <= 0.1 ? ".validation" : ".train");
print > f}' ./AI-TAC/AI-TAC_251bp.tsv
```

output: (folder "./AI-TAC/") `AI-TAC_251bp.tsv.train`, `AI-TAC_251bp.tsv.validation`

Example 3: training ExplaiNN on the AI-TAC dataset

- Train (same parameters as in the preprint; it can take a few hours) and test

```
PY_SCRIPT=../../scripts/train.py
OUT_DIR=./ExplaiNN/AI-TAC
${PY_SCRIPT} -o ${OUT_DIR} --input-length 251 --criterion Pearson \
    --num-units 300 ./AI-TAC/AI-TAC_251bp.tsv.train \
    ./AI-TAC/AI-TAC_251bp.tsv.validation
PY_SCRIPT=../../scripts/test.py
${PY_SCRIPT} -o ${OUT_DIR} ${OUT_DIR}/model_epoch_best_8.pth \
    ${OUT_DIR}/parameters-train.py.json ./AI-TAC/AI-TAC_251bp.tsv.validation
```

output: (folder "`./ExplaiNN/AI-TAC/`") `losses.tsv`, `model_epoch_best_8.pth` (name can differ), `parameters-train.py.json`, `performance-metrics.tsv`

- The performance (i.e., PCC) of ExplaiNN on the AI-TAC dataset was **~0.35**, which is very similar to the performance reported in the preprint (see Fig. 4A)

Example 3: training ExplaiNN on the AI-TAC dataset

- Interpret the model

```
PY_SCRIPT=../../scripts/interpret.py
${PY_SCRIPT} -t -o ${OUT_DIR} --correlation 0.75 --num-well-pred-seqs 1000 \
    ${OUT_DIR}/model_epoch_best_8.pth ${OUT_DIR}/parameters-train.py.json \
    ./AI-TAC/AI-TAC_251bp.tsv.train
```

output: (folder "./ExplaiNN/AI-TAC/") `filters.meme`, `importances.tsv`, etc.

- Cluster the filters (i.e., remove redundancy)

```
PY_SCRIPT=../../scripts/utils/meme2clusters.py
$PY_SCRIPT -c 8 -o ${OUT_DIR}/clusters ${OUT_DIR}/filters.meme
```

output: ./ExplaiNN/AI-TAC/clusters/") `clusters.meme`, `clusters.tsv.gz`, etc.

- Obtain a logo for each cluster in PNG format (option "-f")

```
PY_SCRIPT=../../scripts/utils/meme2logo.py
${PY_SCRIPT} -c 8 -f png -o ${OUT_DIR}/logos ${OUT_DIR}/clusters/clusters.meme
```

output: (folder "./ExplaiNN/CTCF/logos/") `filter0.fwd.png`, `filter0.rev.png`, etc.

Example 3: training ExplaiNN on the AI-TAC dataset

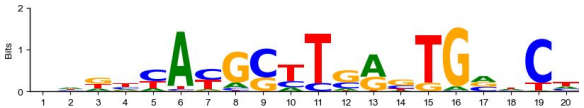
- Visualize the logos of CEBP and PAX clusters (i.e., highlighted in the preprint)

```
PY_SCRIPT=../../scripts/utils/tomtom.py
${PY_SCRIPT} -c 8 -o ${OUT_DIR}/tomtom ${OUT_DIR}/clusters/clusters.meme \
    ./JASPAR/JASPAR2022_CORE_vertebrates_non-redundant_pfms_meme.txt
```

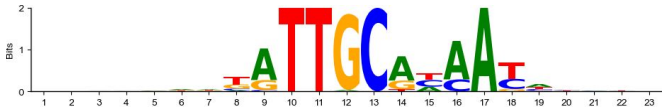
output: `./ExplaiNN/AI-TAC/tomtom/tomtom.tsv.gz`

```
zgrep -e MA0069.1 -e MA0102.4 ${OUT_DIR}/tomtom/tomtom.tsv.gz
cluster85       MA0069.1        -2      2.16186e-07     0.000181813     0.000181813     ...
cluster42       MA0102.4        -6      6.09912e-07     0.000512936     0.00028401      ...
```

cluster85 (i.e., PAX)



cluster42 (i.e., CEBP)



- More complex visualization can be achieved by using Jupyter notebooks (or similar)