RelativePerformer

Relative Encoding of `Performer` architecture. Parts of this code are based on the implementation available here. Files which were taken from this repository retain the authors original copyright notice.

Structure of the project, results and trained models

Most of the code regarding the Performer model implementations can be found in the subfolder `relative_performer`. Jupyter notebooks with experiments and exploratory analysis can be found in the folder `notebooks`. The pipeline for training the CNNs can be found in the folder `CNN`. The raw results used for in our work can be found in the folder `results`. Pretrained models can be downloaded here.

Models

The code implements the following models:

- `Performer`: Performer model with absolute positional encodings and without any further modifications
- `NoPosPerformer`: Performer model without any positional encodings. - `RelativePerformer`: Adapted Performer model using constrained weight matrices for projection of positional embeddings (corresponding to *strategy 1 of the paper*)
- `ClippedRelativePerformer`: Adapted Performer model which directly learns relative positional embeddings (corresponding to *strategy 2 of the paper*)

Installation

Install poetry in a python environment with python version larger than 3.7. Alternatively use pyenv to install an appropriate python version. Pyenv and poetry are compatible, for further details check out the section on pyenv in the poetry documentation.

Then install the package in a new virtual environment using

```
poetry install --dev
```

You can then enter the virtual environment using `poetry shell` or run commands inside the virtual environment using `poetry run <command>`.

Training

The main script for training models is `relative_performer/train.py`, it allows to define training and model parameters and the dataset that should be used to train the model.

```
$ poetry run relative_performer/train.py --help

usage: train.py [-h] [--log_path LOG_PATH] [--exp_name EXP_NAME]
                [--version VERSION] [--batch_size BATCH_SIZE]
                [--embedding_type {linear,MLP,lookup}]
                {Performer,RelativePerformer,NoposPerformer,ClippedRelativePerfor
                {FashionMNIST,MNIST,CIFAR10}

positional arguments:
  {Performer,RelativePerformer,NoposPerformer,ClippedRelativePerformer}
                        The model to train
  {FashionMNIST,MNIST,CIFAR10}
                        The dataset to train on

optional arguments:
  -h, --help            show this help message and exit
  --log_path LOG_PATH   Logging path
  --exp_name EXP_NAME   Experiment name
  --version VERSION     Version of experiment
  --batch_size BATCH_SIZE
                        Batch size used for training.
  --embedding_type {linear,MLP,lookup}
                        Embedding type used to embed pixel values (default:
                        linear)
```

The logs of successful runs will be stored in the path `<log_path>/<exp_name>/<version>`. If `--version` is not explicitly defined it will automatically be set to the pattern `version_X` where `X` is automatically incremented for each run. By default (if no command line parameters are provided) the path is `lighting_logs/default/version_X`.

Missing datasets will automatically be downloaded to the path `<project_root>/data`.

Model specific arguments

Each model can additionally have specific arguments associated which may also be defined via the command line

```
$ poetry run relative_performer/train.py Performer MNIST --help

usage: train.py [-h] [--log_path LOG_PATH] [--exp_name EXP_NAME]
```

```
usage: train.py [-h] [--log_path LOG_PATH] [--exp_name EXP_NAME]
                [--version VERSION] [--batch_size BATCH_SIZE]
                [--embedding_type {linear,MLP,lookup}]

                [--learning_rate LEARNING_RATE] [--warmup WARMUP]
                [--schedule {constant,noam}] [--dim DIM] [--depth DEPTH]
                [--heads HEADS] [--attn_dropout ATTN_DROPOUT]
                [--ff_dropout FF_DROPOUT]
                [--feature_redraw_interval FEATURE_REDRAW_INTERVAL]
                [--no_projection]
                {Performer,RelativePerformer,NoposPerformer,ClippedRelativePerfor
                {FashionMNIST,MNIST,CIFAR10}

positional arguments:
  {Performer,RelativePerformer,NoposPerformer,ClippedRelativePerformer}
                        The model to train
  {FashionMNIST,MNIST,CIFAR10}
                        The dataset to train on

optional arguments:
  -h, --help            show this help message and exit
  --log_path LOG_PATH   Logging path
  --exp_name EXP_NAME   Experiment name
  --version VERSION     Version of experiment
  --batch_size BATCH_SIZE
                        Batch size used for training.
  --embedding_type {linear,MLP,lookup}
                        Embedding type used to embed pixel values (default:
                        linear)
  --learning_rate LEARNING_RATE
  --warmup WARMUP
  --schedule {constant,noam}
  --dim DIM
  --depth DEPTH
  --heads HEADS
  --attn_dropout ATTN_DROPOUT
  --ff_dropout FF_DROPOUT
  --feature_redraw_interval FEATURE_REDRAW_INTERVAL
  --no_projection
```

Which shows that the `Performer` model additionally supports the arguments `--learning_rate`, `--dim`, `--depth` and `--heads` etc.

Example - Training on MNIST

Training the performer model on MNIST with default parameters can be achieved using the command below:

```
$ poetry run relative_performer/train.py MNIST

No correct seed found, seed set to 2111136583
GPU available: False, used: False
TPU available: None, using: 0 TPU cores

  | Name                  | Type                        | Params
-------------------------------------------------------------------
0 | content_embedding     | Linear                      | 256
1 | output_layer          | Linear                      | 1.3 K
2 | loss                  | CrossEntropyLoss            | 0
3 | train_acc             | Accuracy                    | 0
4 | val_acc               | Accuracy                    | 0
5 | test_acc              | Accuracy                    | 0
6 | positional_embedding  | LearnableSinusoidEncoding   | 32
7 | performer             | Performer                   | 793 K
-------------------------------------------------------------------
794 K     Trainable params
0         Non-trainable params
794 K     Total params
Running with random seed: 2111136583
Called log hparams
Validation sanity check: 100%|██████████| 2/2 [00:03<00:00,  1.55s/it]
Training: Epoch 0:   0%|          | 2/3749 [00:10<5:14:18,  5.03s/it, loss=3.22, 
```

Testing

After a model is run its performance can be evaluated on the test split of the dataset. This is implemented in the files `relative_performer/test.py` and `relative_performer/test_recursive.py`, where the former tests a single model and the latter traverses a directory testing all trained models it finds in the hierarchy. In both cases the `--output` argument expects a path to a csv file where the results should be stored.

```
$ poetry run relative_performer/test.py

usage: test.py [-h] --output OUTPUT run_dir
```

```
positional arguments:
  run_dir

optional arguments:
  -h, --help       show this help message and exit
  --output OUTPUT

$ poetry run relative_performer/test_recursive.py

usage: test_recursive.py [-h] [--output OUTPUT] run_dirs

positional arguments:
  run_dirs

optional arguments:
  -h, --help       show this help message and exit
  --output OUTPUT
```

Testing shifted images

As an additional means of evaluating the models the script `relative_performer/test_shifted.py` tests the model on a subset of shifted versions of the datasets. This script currently only supports being applied to runs from either MNIST or FashionMNIST. The option `--min_shift` determines the shifting range in pixels. The performance is only evaluated on digits that support the full span of shifts of min_shift pixels to the left and to the right. `--labels` allows to filter the classes which should be shifted. In our experiments this was always set to 1 as it allowed the maximal amount of shift for both MNIST and FashionMNIST. Finally, `--output` expects a path to a csv file for storing the results and `run_dirs` expects a list of multiple folders containing runs on which the object shift experiment should be run.

```
$ poetry run relative_performer/test_shifted.py --help

usage: test_shifted.py [-h] [--labels LABELS [LABELS ...]]
                       [--min_shift MIN_SHIFT] --output OUTPUT
                       run_dirs [run_dirs ...]

positional arguments:
  run_dirs

optional arguments:
  -h, --help            show this help message and exit
  --labels LABELS [LABELS ...]
  --min_shift MIN_SHIFT
  --output OUTPUT
```