Reproduce results

Paul Lodder

January 28, 2021

Introduction

This file serves the purpose of documenting our code and providing the commands that produce the results we report in our paper.

Getting started

First of all, make sure to have followed the steps documented in README.org. Make sure that you have activated the virtualenvironment that you initialized to run any of our scripts.

General overview

To train and evaluate a model for any given dataset, we run

```
python scripts/train.py --dataset <dataset name>
```

where `<dataset name>` is one of `adult`, `german`, `yaleb`, `cifar10`, `cifar100`. The correct model for the given dataset is loaded by taking the (hyper)parameters specified in src/defaults.py. E.g. if we are working with `adult` data, the following variable is used to determine all parameters:

```
DEFAULTS_ADULT = {
    "z_dim": 2,
    "max_epochs": 2,
    "batch_size": 64,
    "lambda_od": 0.037,
    "gamma_od": 0.8,
    "encoder_hidden_dims": [64],
    "lambda_entropy": 0.55,
    "gamma_entropy": 1.66,
    "input_dim": 108,
    "target_output_dim": 1,
    "sens_output_dim": 1,
    "target_disc_hidden_dims": [64, 64],
    "sens_disc_hidden_dims": [64, 64],
    "target_disc_batch_norm": False,
    "predictor_epochs": 10,
    "encoder_lr": 1e-3,
    "encoder_weight_decay": 5e-4,
    "discs_lr": 1e-3,
    "discs_weight_decay": 5e-4,
```

```
    "loss_components": "entropy,kl,orth",
    "step_size": 30,
}
```

Most of these parameters can also be changed by explicitly providing a value as a command line argument:

```
python scripts/train.py --dataset <dataset name> --step_size 100
```

The loaded configuration is passed to functions defined in src/iniitializers.py, like `get_fodvae(config)` which returns the correct model by looking at the given config and associated hyperparameters.

Reproducing our results

Table 1 summarizes the commands that produce the results presented in the corresponding listing our paper. For more information

Table 1: The commands that reproduce the results presented in the corresponding listing from our paper.

Listing in our paper	**Command to reproduce**
Figure 1	`bash scripts/train_and_plot_target_and_sens_accs.sh`
Figure 2	`bash scripts/all_ablative.sh`
Figure 3	`bash scripts/produce_sensitivity_plot.sh`
Figure 4 (Appendix)	`python scripts/show_yaleb_positions.py`
Table 3	`python scripts/run_cifar.py`
Table 4 (Appendix)	`python scripts/show_dataset_details.py`

For more specific information about each script and the handling of results, see the sections below.
Now that you a main idea of our code, let us talk you through reproducing the specific results that we present in our paper.

Figure 1 - Comparing performance with other methods

Figure 1 shows multiple bar plots that compare Sarhan et al's (2020) sensitive and target accuracies achieved on the raw data, embeddings from a regular VAE, and the target representations produced by the proposed model. They performed this experiment only on the Adult, German, and YaleB dataset. To produce all plots for this figure, we ran:

```
bash scripts/train_and_plot_target_and_sens_accs.sh
```

This script runs the following script for each of the three datasets:

```
python scripts/make_fig2.py --dataset <dataset-name>
```

It saves two figures per dataset in `FIGURES_DIR` (as specified in your `.env`): `<dataset-name>_sens.png` and `<dataset-name>_target.png` which show the sensitive and target accuracies, respectively.

Figure 2 - Ablative study

Figure 2 shows multiple bar plots that compare the target and sensitive accuracies when turning on and off particular components of the loss term. To support this experiment, we added an additional command line argument `--loss_components` which must take one of the following values:

`none`	Nor entropy loss nor OD loss is used
`entropy`	Entropy loss is used but OD loss is not
`kl,orth`	OD loss is used and the prior means are orthogonal, entropy loss is not used
`entropy,kl`	Entropy loss and OD loss are used, but the prior means are not orthogonal
`entropy,kl,orth`	Both entropy loss and OD loss are used, the prior means are orthogonal

The results are averaged over 5 random seeds. To produce the results for any of our datasets, we run:

```
bash ablative.sh <dataset name>
```

This will run our model on the given dataset for 5 different random seeds for each of the possible `--loss_components` values. The results for stored in `RESULTS_DIR` in the following format: `ablative.<dataset_name>.<loss-components>.<seed>.json` and each file contains the sensitive and target performance as follows:

```
{
  "target": <output of classification_report output_dict>,
  "sensitive": <output of classification_report output_dict>
}
```

To visualize the results of the ablative study, we run:

```
python scripts/visualize_ablative.py --dataset <dataset-name>
```

This script will look at all the corresponding files in the `RESULTS_DIR`, and produce the corresponding bar plot for the dataset, averaging over the random seeds and including the std as an error bar. The resulting figure will be saved to `FIGURES_DIR/ablative.<dataset-name>.png`
To produce the results and plots for all of the datasets, we run:

```
bash scripts/all_ablative.sh
```

Which just runs the `ablative.sh` to produce the results, and subsequently the `visualize_ablative.py` for each dataset.

Figure 3 - Sensitivity analysis

Figure 3 shows a sensitivy analysis done on the Adult dataset by varying λ_{OD}-λ_E and γ_{OD}-γ_E. To produce the results for this plot, we run:

```
python scripts/sensitivity_analysis.py --dataset adult
```

It trains and evaluates our model on the Adult dataset for 64 different combinations of λ_{OD}-λ_E, where λ_{OD} varies from 0.01 to 0.09, and λ_E varies from 0.1 to 1 and for 64 different combinations of γ_{OD}-γ_E where γ_{OD} varies from 1 and 2 and γ_E varies from 0.8 to 1.7. The results were produced for 5 different random seeds, and were saved in files formatted as:
`sensitivity&<dataset-name>&lambda_entropy=<value0>_lambda_od=<value1>&<seed>.json` and `sensitivity&<dataset-name>&gamma_entropy=<value0>_gamma_od=<value1>&<seed>.json`. To visualize the results, we run

```
python scripts/visualize_sensitivity.py --dataset adult
```

This stores the produced plot in `FIGURES_DIR/sensitivity.<dataset-name>.png`

Figure 4 Appendix - YaleB lightning positions

Figure 4 in our appendix visualizes the clustering of YaleB lightning positions that constitute our sensitive classes. To produce the plot, we run:

```
python scripts/show_yaleb_positions.py
```

which saves `yaleb_lighting_positions.png` to your `FIGURES_DIR`.

Table 3

This table presents the target and sensitive accuracy achieved on the Cifar-10 and Cifar-100 dataset. We produced these results by running:

```
python scripts/run_cifar.py
```

This script trains the correct models for Cifar-10 and Cifar-100 for 3 different random seeds, and pickles the results stored in a pickled `pd.DataFrame` in the `RESULTS_DIR` defined in your `.env`. The results presented in our table are the average of the results across the random seeds.

Table 4 Appendix - Dataset details

To produce the overview of the dataset details, we run

```
python scripts/show_dataset_details.py
```

This prints some of the details presented in the table to stdout.