Fashion Encoder Training - User Documentation

This is a user documentation for a package designated for training and evaluating the Fashion Encoder model.

Table of Contents

Environment Setup

We tested this package using conda environment manager, so we recommend using it. However, you should be able to install the dependencies manually.

Hardware requirements:
To run the experiments, we recommend using GPU with CUDA support as the model contains a convolutional neural network. We tested the experiments on NVIDIA Tesla V100 16/32GB.

Software requirements:
When using conda, you don't need to install any aditional software such as CUDA SDK (conda takes care of this).

> For more information about using with conda GPU, see this guide

Conda installation:

1. Install conda using this installation guide
2. Run `conda env create --file environment.yml` in the project root to create our environment
3. When using this project, activate the environment using `conda activate outfit-recommendation`. Then you can run the commands that are described below

Requirements:

In case, you can't use conda, you will need to install these dependencies:

- Python 3.7
- Tensorflow >= 2.1 (preferably the GPU version)
- pillow
- scipy
- jupyter
- keras-tuner

Prepare Datasets

Before running the experiment you will need to download and build the datasets.

1. Download the Datasets

Download Maryland Polyvore

1. Download the dataset files for the Maryland Polyvore Dataset from this link https://www.kaggle.com/dnepozitek/polyvore-dataset
2. Move the folder `maryland` into `data/raw`
3. Download the images from Maryland Polyvore dataset from this link https://www.kaggle.com/dnepozitek/maryland-polyvore-images
4. Move the folder `images` into `data/raw/maryland`

Download Polyvore Outfits

1. Download Polyvore Outfits dataset from this link https://www.kaggle.com/dnepozitek/polyvore-outfits
2. Move the whole folder `polyvore_outfits` into `data/raw/`

2. Build the TFRecord Datasets

We have prepared scripts to build the datasets in the `bin` folder. You can execute the following commands to build the corresponding datasets:

- `bin/build_mp.sh`
- `bin/build_mp_images.sh`
- `bin/build_po.sh`

- `bin/build_po_images.sh`
- `bin/build_pod.sh`
- `bin/build_pod_images.sh`

Each script builds a training dataset, a validation FITB task and a test FITB task. The names have the following meaning: `mp` stands for Maryland Polyvore, `po` is Polyvore Outfits and `pod` is Polyvore Outfits Disjoint. The scripts with its names ending with `_images.sh` build the datasets with raw images, the other scripts extracts the visual features from the images using InceptionV3.

> Note that the building the dataset may take a few hours

Running Experiments

> Make sure you have installed all the neccesary dependencies and that you have prepared the datasets before trying to run the experiments

Training

In order to train and evaluate the model, you can use the `src.models.encoder.encoder_main` Python module, that is run `python -m "src.models.encoder.encoder_main"` with appropriate parameters.

We have prepared sets of parameters for the basic tasks in `src/models/encoder/params.py`. The set can be selected using the `--param-set`. Some of the parameters can be overriden with a corresponding CLI parameter (the full list of CLI parameters is bellow).

Examples

1. **Train the best models**
   To run the best model on Maryland Polyvore, you can use the set `MP_BEST` as follows:

   ```
   python -m "src.models.encoder.encoder_main" --param-set "MP_BEST"
   ```

   We also include `PO_BEST` and `POD_BEST` for training the best models of Polyvore Outfits
2. **Override some hyperparameters**
   Imagine that you want to change some hyperparameters of a model with multiplication category embedding on Polyvore Outfits. You can use the `PO_MUL` parameter set as a base and override only the particular parameters:

```
python -m "src.models.encoder.encoder_main" \
    --param-set "PO_MUL" \
    --num-heads 4 \
    --category-dim 32 \
    --hidden-size 32 \
    --filter-size 64 \
    --batch-size 16
```

> Make sure that you change the `category_dim` to match the `hidden_size` when using multiplication or addition for category embedding

3. **Debug the model**

   To show a trace of a model, you can use an arbitrary set of parameters and set the `mode` to debug:

```
python -m "src.models.encoder.encoder_main" \
    --param-set "POD_BEST" \
    --mode "debug"
```

4. **Train the model on images**

   To train the model together with the CNN, you have to use the `with-cnn` switch and change the filenames(according to the building scripts). Also you will probably need to reduce the batch size:

```
python -m "src.models.encoder.encoder_main" \
    --param-set "POD_BEST" \
    --train-files "data/processed/tfrecords/pod-images-train-000-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-001-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-002-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-003-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-004-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-005-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-006-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-007-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-008-9.tfrecord" \
        "data/processed/tfrecords/pod-images-train-009-9.tfrecord" \
    --valid-files "data/processed/tfrecords/pod-fitb-images-valid.tfrecord" \
    --test-files "data/processed/tfrecords/pod-fitb-images-test.tfrecord" \
    --with-cnn \
    --batch-size 6
```

> Be aware that training with the CNN requires a lot of memory (both of graphics and standard). Also, the framework was optimised for use with extracted features, so training with CNN is rather experimental.

Parameters of `src.models.encoder.encoder_main`

`--mode {train,debug}`
Either "train" or "debug",

`--param-set PARAM_SET`
Name of the hyperparameter set to use as base

`--train-files TRAIN_FILES [TRAIN_FILES ...]`
Paths to train dataset files

`--valid-files VALID_FILES`
Paths to validation dataset files

`--test-files TEST_FILES`
Paths to test dataset files

`--batch-size BATCH_SIZE`
Batch size

`--filter-size FILTER_SIZE`
Encoder filter size

`--epoch-count EPOCH_COUNT`
Number of epochs

`--hidden-size HIDDEN_SIZE`
Hidden size (dimension of the preprocessor's output)

`--num-heads NUM_HEADS`
Number of self-attention heads

`--num-hidden-layers NUM_HIDDEN_LAYERS`
Number of hidden layers (encoder blocks)

`--checkpoint-dir CHECKPOINT_DIR`
Path to a directory with checkpoints (resumes the training)

`--with-weights WITH_WEIGHTS`
Path to the directory with saved weights. The directory

`--masking-mode {single-token,category-masking}`
Mode of item masking.

`--valid-mode {fitb,masking}`
Validation mode. "fitb" by default, but the training task can be used for validation, as well. Note that you

have to generate the validation dataset of a training-type, if you want to use `masking` validation.

`--learning-rate LEARNING_RATE`
Optimizer's learning rate

`--valid-batch-size VALID_BATCH_SIZE`
Batch size of a validation dataset (only used when `valid-mode` set to masking)

`--with-cnn {True,False}`
Train the model with the CNN. Make sure that you have changed the dataset files accordingly.

`--category-embedding {True,False}`
Apply learned category embedding to image feature vectors

`--categories-count CATEGORIES_COUNT`
Number of categories

`--with-mask-category-embedding {True,False}`
Apply category embedding to the mask token

`--category-attention {True,False}`
Compute keys and queries from categories

`--target-gradient-from TARGET_GRADIENT_FROM`
Value of valid accuracy, when gradient is let through target tensors, -1 for stopped targets gradient

`--info INFO`
Arbitrary additional information about the configuration that is logged

`--with-category-grouping {True,False}`
Categories are mapped into high-level groups

`--category-dim CATEGORY_DIM`
Dimension of category embedding (must match the `hidden_size` when using multiplication or addition for embedding)

`--category-merge {add,multiply,concat}`
Mode of category embedding merge. Applies only when `--category-embedding` is set to `True`

`--category-file CATEGORY_FILE`
Path to Polyvore Outfits categories file

`--categorywise-train {True,False}`
Compute loss function only between items from the same category

`--early-stop-patience EARLY_STOP_PATIENCE`
Number of epochs to wait for improvement

`--early-stop-delta EARLY_STOP_DELTA`
Minimum change to qualify as improvement

`--early-stop {True,False}`
Enable early stopping

`--loss {cross,distance}`
Loss function

`--margin MARGIN`
Margin of the distance loss function

Hyperparameter Tuning

The hyperparameter tuning functionality is implemented in a module `src.models.encoder.param_tuning`. You can edit the `build` method to restrict the tuning to only some parameters or to modify the search space. As the file uses Keras Tuner in a straightforward way, we refer you to the official Keras Tuner documentation.

To execute the hyperparameter tuning, run `bin/hypertuning.sh`.

> We decided not to implement a CLI for the hyperparameter tuning because the Keras Tuner library provides a convenient way of setting up the tuning programmatically.