hexia

Release 0.0.1

Jun 30, 2019

Tutorials

This is **Hexia**. A **PyTorch** based framework for building visual question answering models. Hexia provides a mid-level API for seamless integration of your VQA models with pre-defined data, image preprocessing and natural language proprocessing pipelines.

CHAPTER 1

Quickstart

Author: Ali Gholami

In this section, we'll walk through general examples that help you get hexia up and running with your own vqa models.

1.1 Installation

1. Clone the repository and enter it:

```
git clone https://github.com/aligholami/hexia && cd hexia
```

1. Run the **setup.py** to install dependencies:

```
python3 setup.py install --user
```

1.2 Examples

1.2.1 Preparing Images

Image preprocessing with a custom CNN architecture:

```
import torch.nn as nn
from hexia.backend.cnn.resnet import resnet
from hexia.preprocessing.vision import Vision
from hexia.tests import config

# define your own image feature extractor which inherits pytorch nn module
class myCNN(nn.Module):

    def __init__(self):
```

(continues on next page)

(continued from previous page)

```
        super(myCNN, self).__init__()
        self.model = resnet.resnet101(pretrained=True)

        def save_output(module, input, output):
            self.buffer = output

        self.model.layer4.register_forward_hook(save_output)

    def forward(self, x):
        self.model(x)
        return self.buffer

# perform image preprocessing with your custom CNN
my_cnn = myCNN().cuda()
visual_preprocessor = Vision(
    transforms_to_apply=['none'],
    cnn_to_use=myCNN,
    path_to_save=config.preprocessed_path,
    path_to_train_images=config.train_path,
    path_to_val_images=config.val_path,
    batch_size=config.preprocess_batch_size,
    image_size=config.image_size,
    keep_central_fraction=config.central_fraction,
    num_threads_to_use=8
)
visual_preprocessor.initiate_visual_preprocessing()
```

The above code will load your defined model (either pre-trained or from scratch) and saves the extracted features to **path_to_save**.

1.2.2 Preparing Questions and Answers

Text preprocessing with one of the available models (GloVe or Word2Vec):

```
from hexia.preprocessing.language import Language
from hexia.tests import config

language_preprocessor = Language(
    max_answers=config.max_answers,
    save_vocab_to=config.vocabulary_path
)
language_preprocessor.initiate_vocab_extraction()

# Remove this line if you want to use Word2Vec embeddings
language_preprocessor.extract_glove_embeddings(
    dims=50,
    path_to_pretrained_embeddings=config.glove_embeddings,
    save_vectors_to=config.glove_processed_vectors,
    save_words_to=config.glove_words,
    save_ids_to=config.glove_ids
)
```

The above code will tokenize the VQA dataset and saves extracted GloVe vectors to **save_vectors_to**. If you want to use random Word2Vec embeddings, simply remove the call to **extract_glove_embeddings**; the framework will automatically switch to randomly initialized Word2Vec embeddings.

1.2.3 Training, Validation and Checkpointing

Model training, validation and checkpointing:

```
import torch
import torch.nn as nn
import torch.optim as optim
from tensorboardX import SummaryWriter
import os
from hexia.backend.utilities import utils
from hexia.tests import config
from hexia.backend.monitoring.tracker import Tracker
from hexia.runtime.train_val import TrainValidation
from hexia.vqa.models.joint import M_Resnet18_randw2v_NoAtt_Concat as model

# Prepare dataset
train_loader, val_loader = utils.prepare_data_loaders(path_to_feature_maps=config.
↪preprocessed_path,
                                                      batch_size=config.batch_size,␣
↪num_workers=config.data_workers)

# Build the model
net = nn.DataParallel(model.Net(train_loader.dataset.num_tokens)).cuda()
optimizer = optim.Adam([p for p in net.parameters() if p.requires_grad])
tracker = Tracker()
train_writer = SummaryWriter(config.visualization_dir + 'train')
val_writer = SummaryWriter(config.visualization_dir + 'val')

# Separate objects for train and validation (enables auto-resume on valid path␣
↪settings)
vqa_trainer = TrainValidation(net, train_loader, optimizer, tracker, train_writer,␣
↪train=True, prefix='train',
                              latest_vqa_results_path=config.latest_vqa_results_path)
vqa_validator = TrainValidation(net, val_loader, optimizer, tracker, val_writer,␣
↪train=False, prefix='val', latest_vqa_results_path=None)

best_loss = 10.0
best_accuracy = 0.1
for epoch in range(config.num_epochs):
    _ = vqa_trainer.run_single_epoch()
    r = vqa_validator.run_single_epoch()

train_writer.close()
val_writer.close()
```

The above code will create two separate data loaders for train and validation (with their respective image paths). Then you may use **Tracker** and **TensorboardX.SummaryWriter** classes to track and visualize the training and validation process. Note that it is important to use the parameter **latest_vqa_results_path** only once among all objects. To enable **auto_resume** functionality while re-running the code, you can add the following to your higher-level training and validation loop:

```
if r['epoch_accuracy'] > best_accuracy and r['epoch_loss'] < best_loss:

    # Update best accuracy and loss
    best_accuracy = r['epoch_accuracy']
    best_loss = r['epoch_loss']
```

(continues on next page)

(continued from previous page)

```
    # Clear path from previus saved models and pre-evaluation files
    try:
        os.remove(config.best_vqa_weights_path)
        os.remove(config.best_vqa_answers_to_eval)
    except FileNotFoundError as fe:
        pass

    # Save the new model weights
    weights = net.state_dict()
    torch.save(weights, config.best_vqa_weights_path)

    # Save answ, idxs and qids for later evaluation
    utils.save_for_vqa_evaluation(r['answ'], r['ids'], r['qids'])

checkpoint = {
    'epoch': r['epoch'],
    'model_state_dict': net.state_dict(),
    'optimizer_state_dict': optimizer.state_dict(),
    'tracker': tracker.to_dict(),
    'vocab': train_loader.dataset.vocab,
    'train_iters': r['train_iters'],
    'val_iters': r['val_iters'],
    'prefix': r['prefix'],
    'train': r['train'],
    'loader': r['loader']
}

torch.save(checkpoint, config.latest_vqa_results_path)
```

In this code, variable **r** contains crucial information about the last training and validation step, writer, tracker and optimizable model parameters status.

Note: If you would like to use your custom VQA model, you may import it as a replacement of this line:

```
from hexia.vqa.models.joint import M_Resnet18_randw2v_NoAtt_Concat as model
```

your model should inherit the nn.Module and its overrided forward function should look like this:

```
def forward(self, v, q, q_len):
    # your vqa logic to compute an answer
    return answer
```

1.2.4 Model Evaluation

Evaluting the trained model (based on official evaluation class):

```
# Import the evaluation module
from hexia.vqa.evaluation.evaluator import VQAEvaluator

v_evaluator = VQAEvaluator(
    data_directory=config.data_directory,
    best_model_results_directory=config.best_vqa_answers_to_eval)
```

(continues on next page)

(continued from previous page)

```
v_evaluator.evaluate_best_vqa_model()
```

Parameter **data_directory** specifies the directory of data (make sure its according to the official format), and **best_model_results_directory** is the path to the generated JSON file as the answers to the best model.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search