SimREC

SimREC Authors

Sep 03, 2022

CONTENTS

CHAPTER

ONE

THE BASICS

1.1 Installation

Here we provide the basic installation guide for SimREC, including setups of the environments, datasets and the pretrained weights.

1.1.1 Environment Setup

- Clone this repo

```
git clone https://github.com/luogen1996/SimREC.git
cd SimREC
```

- Create a conda virtual environment and activate it

```
conda create -n simrec python=3.7 -y
conda activate simrec
```

- Install Pytorch following the official installation instructions
- Install mmcv following the installation guide
- Install Spacy and initialize the GloVe and install other requirements as follows:

```
pip install -r requirements.txt
wget https://github.com/explosion/spacy-models/releases/download/en_vectors_web_lg-2.
↪1.0/en_vectors_web_lg-2.1.0.tar.gz -O en_vectors_web_lg-2.1.0.tar.gz
pip install en_vectors_web_lg-2.1.0.tar.gz
```

1.1.2 Dataset Setup

Prepare the datasets before running experiments.

The project directory is `$ROOT`and current directory is located at `$ROOT/data` to generate annotations.

1. Download the cleaned referring expressions datasets and extract them into `$ROOT/data` folder:
1. Prepare mscoco train2014 images, original Flickr30K images, ReferItGame images and Visual Genome images, and unzip the annotations. Then the file structure should look like:

```
$ROOT/data
|-- refcoco
    |-- instances.json
    |-- refs(google).p
    |-- refs(unc).p
|-- refcoco+
    |-- instances.json
    |-- refs(unc).p
|-- refcocog
    |-- instances.json
    |-- refs(google).p
    |-- refs(umd).p
|-- refclef
    |-- instances.json
    |-- refs(berkeley).p
    |-- refs(unc).p
|-- images
    |-- train2014
    |-- refclef
    |-- flickr
    |-- VG
```

1. Run data_process.py to generate the annotations. For example, running the following code to generate the annotations for **RefCOCO**:

```
cd $ROOT/data
python data_process.py --data_root $ROOT/data --output_dir $ROOT/data --dataset␣
↪refcoco --split unc --generate_mask
```

- `--dataset={'refcoco', 'refcoco+', 'refcocog', 'refclef'}` to set the dataset to be processd.

For **Flickr** and **merged pre-training data**, we provide the pre-processed json files: flickr.json, merge.json.

Note: The merged pre-training data contains the training data from RefCOCO *train*, RefCOCO+ *train*, RefCOCOg *train*, Referit *train*, Flickr *train* and VG. We also remove the images appearing the validation and testing set of RefCOCO, RefCOCO+ and RefCOCOg.

1. At this point the directory `$ROOT/data` should look like:

```
$ROOT/data
|-- refcoco
    |-- instances.json
    |-- refs(google).p
    |-- refs(unc).p
|-- refcoco+
    |-- instances.json
    |-- refs(unc).p
|-- refcocog
    |-- instances.json
    |-- refs(google).p
    |-- refs(umd).p
|-- anns
    |-- refcoco
        |-- refcoco.json
    |-- refcoco+
        |-- refcoco+.json
    |-- refcocog
```

(continues on next page)

(continued from previous page)

```
        |-- refcocog.json
    |-- refclef
        |-- refclef.json
    |-- flickr
        |-- flickr.json
    |-- merge
        |-- merge.json
|-- masks
    |-- refcoco
    |-- refcoco+
    |-- refcocog
    |-- refclef
|-- images
    |-- train2014
    |-- refclef
    |-- flickr
    |-- VG
|-- weights
    |-- pretrained_weights
```

1.1.3 Pretrained Weight Setup

We provide the pretrained weights of visual backbones on MS-COCO. We remove all images appearing in the *val+test* splits of RefCOCO, RefCOCO+ and RefCOCOg. Please download the following weights into `$ROOT/data/weights`.

We also provide the weights of **SimREC** that are pretrained on 0.2M images.

1.2 SimREC Config

We adopted the `Lazy Config System` design from detectron2, You can refer to d2 tutorials for more details aboud the syntax and usage of lazy config. In this section, we provide the basic examples of the lazy config usage in SimREC.

1.2.1 Configs in SimREC

We simply summarized the config namespaces into `model`, `dataset`, `optim`, `train` as follows:

model

This is the configuration for the model definition. You can refer to `configs/common/models` for more examples:

Here is the example of loading `SimREC` model config file:

```python
# configs/common/models/simrec.py
import torch.nn as nn

from simrec.config import LazyCall
from simrec.models.simrec import SimREC
from simrec.models.backbones import CspDarkNet
from simrec.models.heads import REChead
```

(continues on next page)

(continued from previous page)

```
from simrec.models.language_encoders import LSTM_SA
from simrec.layers.fusion_layer import SimpleFusion, MultiScaleFusion, GaranAttention

model = LazyCall(SimREC)(
    visual_backbone=LazyCall(CspDarkNet)(
        pretrained=False,
        pretrained_weight_path="./data/weights/cspdarknet_coco.pth",
        freeze_backbone=True,
        multi_scale_outputs=True,
    ),
    language_encoder=LazyCall(LSTM_SA)(
        depth=1,
        hidden_size=512,
        num_heads=8,
        ffn_size=2048,
        flat_glimpses=1,
        word_embed_size=300,
        dropout_rate=0.1,
        # language_encoder.pretrained_emb and language.token_size is meant to be set
        # before instantiating
        freeze_embedding=True,
        use_glove=True,
    ),
    multi_scale_manner=LazyCall(MultiScaleFusion)(
        v_planes=(512, 512, 512),
        scaled=True
    ),
    fusion_manner=LazyCall(nn.ModuleList)(
        modules = [
            LazyCall(SimpleFusion)(v_planes=256, out_planes=512, q_planes=512),
            LazyCall(SimpleFusion)(v_planes=512, out_planes=512, q_planes=512),
            LazyCall(SimpleFusion)(v_planes=1024, out_planes=512, q_planes=512),
        ]
    ),
    attention_manner=LazyCall(GaranAttention)(
        d_q=512,
        d_v=512
    ),
    head=LazyCall(REChead)(
        label_smooth=0.,
        num_classes=0,
        width=1.0,
        strides=[32,],
        in_channels=[512,],
        act="silu",
        depthwise=False
    )
)
```

```
# my_own_config_file.py
from common.models.simrec import model

model.visual_backbone.pretrained = True # modify config according to your own needs
```

Note: due to the lazy instantiate benefits of LazyCall, some param like `model.language_encoder.pretrained_emb` will be set in train loop before instantiating the model.

train

This is the configuration for training. The default dataset config can be found in `configs/common/train.py`. The details of training config are as follows:

```
# Basic training-related configs.

train = dict(

    # Directory where to save the output files.
    output_dir = "./test",

    # Warmup epochs and total epochs for training.
    warmup_epochs=3,
    epochs = 25,

    # Learning rate settings for lr-scheduler.
    base_lr=1e-4,
    warmup_lr=1e-7,
    min_lr=1e-6,

    # Total batch size, if you run SimREC on 4 GPUs,
    # each gpu will handle only (batch / 4) samples.
    batch_size=8,

    # Evaluation configuration, if set sequential=True, which will
    # use SequentialSampler during validating.
    evaluation=dict(
        eval_batch_size=8,
        sequential=False
    ),

    # Log the training infos every log_period times of iterations.
    log_period=1,

    # Save the checkpoints every save_period times of iterations.
    save_period=1,

    # Basic data config.
    data=dict(
        pin_memory=True,
        num_workers=8,
        mean=[0., 0., 0.],
        std=[1., 1., 1.],
    ),

    # Scheduler config.
    scheduler=dict(
        name="cosine",
        decay_epochs=[30, 35, 37],
        lr_decay_rate=0.2,
    ),

    # Enable automatic mixed precision for training which does not
    # change model's inference behavior.
    amp=dict(enabled=False),

    # Distributed training settings.
```

(continues on next page)

(continued from previous page)

```
    ddp=dict(
        backend="nccl",
        init_method="env://",
    ),

    # Enable model ema during training or not.
    ema=dict(enabled=True, alpha=0.9997, buffer_ema=True),

    # Automatically convert batchnorm to sync batchnorm layers.
    sync_bn=dict(enabled=False),

    # Clip gradient.
    clip_grad_norm=0.15,

    # Resume training.
    auto_resume=dict(enabled=True),
    resume_path="",
    vl_pretrain_weight="",

    # Multi-scale training.
    multi_scale_training=dict(
        enabled=True,
        img_scales=[[224,224],[256,256],[288,288],[320,320],[352,352],
                    [384,384],[416,416],[448,448],[480,480],[512,512],
                    [544,544],[576,576],[608,608]]
    ),

    # Log image during training.
    log_image=False,

    # Training seed.
    seed = 123456,
)
```

optim

This is the configuration for the optim definition. Please refer to `configs/common/optim.py` for the default optim config:

```
from torch.optim import Adam

from simrec.config import LazyCall

optim = LazyCall(Adam)(
    # optim.params is meant to be set before instantiating
    lr=0.0001,
    betas=(0.9, 0.98),
    eps=1e-9
)
```

dataset

This is the configuration for dataset. The default dataset config can be found in `configs/common/dataset.py`. The details of dataset config are as follows:

```
import albumentations as A
from torchvision.transforms import transforms

from simrec.config import LazyCall
from simrec.datasets.dataset import RefCOCODataSet
from simrec.datasets.transforms.randaug import RandAugment

from .train import train

dataset = LazyCall(RefCOCODataSet)(
    # the dataset to be created
    # choose from ["refcoco", "refcoco+", "refcocog", "referit", "vg", "merge"]
    dataset = "refcoco",

    # path to the files
    ann_path = {
                'refcoco':'./data/anns/refcoco.json',
                'refcoco+': './data/anns/refcoco+.json',
                'refcocog': './data/anns/refcocog.json',
                'referit': './data/anns/refclef.json',
                'flickr': './data/anns/flickr.json',
                'vg': './data/anns/vg.json',
                'merge':'./data/anns/merge.json'
            },
    image_path = {
                'refcoco': './data/images/coco',
                'refcoco+': './data/images/coco',
                'refcocog': './data/images/coco',
                'referit': './data/images/refclef',
                'flickr': './data/images/flickr',
                'vg':'./data/images/VG',
                'merge':'./data/images/'
            },
    mask_path = {
                'refcoco': './data/masks/refcoco',
                'refcoco+': './data/masks/refcoco+',
                'refcocog': './data/masks/refcocog',
                'referit': './data/masks/refclef'
            },

    # original input image shape
    input_shape = [416, 416],
    flip_lr = False,

    # basic transforms
    transforms=LazyCall(transforms.Compose)(
        transforms = [
            LazyCall(transforms.ToTensor)(),
            LazyCall(transforms.Normalize)(
                mean=train.data.mean,
                std=train.data.std,
            )
        ]
```

(continues on next page)

(continued from previous page)

```
    ),

    # candidate transforms
    candidate_transforms = {
        # "RandAugment": RandAugment(2, 9),
        # "ElasticTransform": A.ElasticTransform(p=0.5),
        # "GridDistortion": A.GridDistortion(p=0.5),
        # "RandomErasing": transforms.RandomErasing(
        #     p = 0.3,
        #     scale = (0.02, 0.2),
        #     ratio=(0.05, 8),
        #     value="random",
        # )
    },

    # the max truncked length for language tokens
    max_token_length = 15,

    # use glove pretrained embeddings or not
    use_glove = True,

    # datasets splits
    split = "train",
)
```

Use the default config in your own config file

The users do not have to rewrite all the config every time. You can use the default config file provided in SimREC by importing them as the python file. For example:

```
# import the default config
from simrec.config import LazyCall
from .common.dataset import dataset
from .common.train import train
from .common.optim import optim
from .common.models.simrec import model

# modify them according to your own needs
# refine the cfg
train.output_dir = "./your/own/path"
train.batch_size = 32
train.save_period = 1
train.log_period = 10
train.evaluation.eval_batch_size = 32
train.sync_bn.enabled = False
```

1.3 Getting Started

Here we provide the basic tutorials about the usage SimREC. Make sure you've already install the environments for SimREC, please refer to Installation.

1.3.1 Training

In SimREC, we provide `tools/train_engine.py` and `tools/eval_engine.py` for training and evaluation.

The following script will start training `simrec` model on `refcoco` dataset on a single GPU:

```
$ bash tools/train.sh configs/simrec_refcoco_scratch.py 1
```

All of the `checkpoints`, `logs` and `tensorboard` logs will be saved to `cfg.train.output_dir`, you can modify them in the config file, we highly recommend the users to put their own config file under `/configs` to easily reuse the default config files:

```
# config.py
from .common.train import train

train.output_dir = "/your/own/path"
```

For **distributed data parallel training**, you can modify the training script as follows:

```
$ bash tools/train.sh configs/simrec_refcoco_scratch.py 4
```

To run ddp training mode by simply modifing the last number of the training scripts to `4`.

1.3.2 Override the config in command line

You can override the config file in command line. For example, you can enable the `SyncBatchNorm` in scripts for ddp training like:

```
$ bash tools/train.sh configs/simrec_refcoco_scratch.py 4 train.sync_bn.enabled=True
```

which may give you a better result.

1.3.3 Resume training

In SimREC, we support two resume training ways adopted from Swin-Transformer:

- Automatically resume training:

SimREC automatically saves `last_checkpoint.pth` during training time to `cfg.train.output_dir`, if set `cfg.train.auto_resume.enabled=True`, before training, SimREC will find if there is `last_checkpoint.pth` in `cfg.train.output_dir` and automatically resume from it.

- Resume training from specific checkpoint:

Firstly, you should disable `auto-resume` function which will override the `cfg.train.resume_path` by setting `cfg.train.auto_resume.enabled=False`, and you should update `cfg.train.resume_path` to the specific checkpoint you want to resume from as follows:

```
# config.py
from .common.train import train

train.auto_resume.enabled=False
train.resume_path = "path/to/specific/checkpoint.pth"
```

1.3.4 Evaluation

Run `bash tools/eval.sh` under to evaluate the saved checkpoint.

```
bash tools/eval.sh config/simrec_refcoco_scratch.py 4 /path/to/checkpoint
```