UBAI PRACTICE - Captioning

hs.hwang

2024-11-04

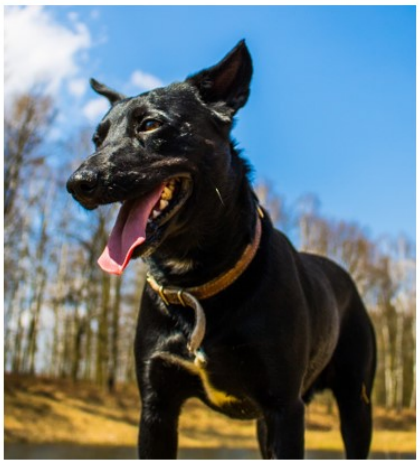
Table of contents

1 Image Captioning

Python , (Image Captioning) .

, .

, , “A black dog sitting among leaves in a forest, surrounded by trees.(.)” .



.

1.

, .

. captioning . Terminal , cd (captioning) .

`pwd` .

`cd` .

.

(1) enroot

, enroot .

OS enroot .

.

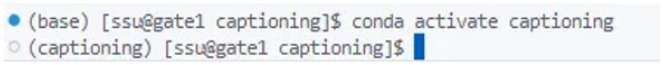
```
# gpu5
srun --pty -p gpu5 -c 2 /bin/bash

# dockerhub
enroot import docker://eclipse/ubuntu_python

#       ubuntu.sqsh
enroot create -n mycontainer eclipse_ubuntu_python.sqsh

#
enroot start --root --rw --mount .:/mnt ubuntu-test /bin/bash
```

(2) conda

conda .

`conda create -n captioning python=3.8` python 3.8 captioning .

`conda activate captioning` .



`pip install -r requirements.txt` .

OS ! !

```
# requirements.txt

torch==1.13.1+cu118
torchvision==0.14.1+cu118
transformers==4.24.0
matplotlib==3.5.3
jupyter==1.0.0
```

2.

. **Microsoft COCO (MS COCO)** .

MS COCO Object detection( ), Segmentation( ), Captioning ,
.

MS COCO shell .

```
#!/bin/bash

# COCO dataset directory
mkdir -p /data/coco

# Download COCO Train2014 images and captions
cd /data/coco
wget http://images.cocodataset.org/zips/train2014.zip
wget http://images.cocodataset.org/zips/val2014.zip
wget http://images.cocodataset.org/annotations/annotations_trainval2014.zip
```

```
# Unzip the dataset
unzip train2014.zip
unzip val2014.zip
unzip annotations_trainval2014.zip
```

`mkdir` data . mkdir .

`cd` .

`wget` MS COCO dataset , .

`unzip` dataset zip , .

, shell .

`datset_download.sh` shell , "permission denied ( )" . ,
`chmod` .

`chmod` , .

```
chmod [references][operator][modes] file1 ...
```

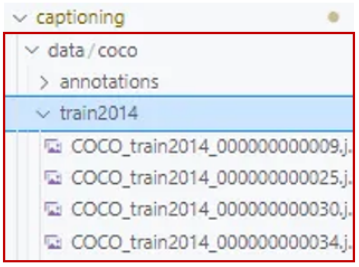
operator	role
r	(read)
w	(write)
x	(execute)

`chmod +x [file_name.sh]` +x [file_name.sh] .

```
(base) [ssu@gate1 captioning]$ ./dataset_download.sh
bash: ./dataset_download.sh: Permission denied
(base) [ssu@gate1 captioning]$ chmod +x dataset_download.sh
```

.

, .



3.

, .

Transformer . Transformer 2017 Google , .

transformer.py , transformer.py .

slurm . , HPC slurm `transformer.sh` .

```
#!/bin/bash
#SBATCH --job-name=captioning
#SBATCH --output=./output/training_captioning_%n_%j.out
#SBATCH --error=./output/training_captioning_%n_%j.err
#SBATCH --nodes=2
#SBATCH --partition=gpu3
#SBATCH --gres=gpu:4
#SBATCH --cpus-per-task=16
#SBATCH --mem=128G
```

```
#SBATCH --time=24:00:00

echo "start at:" `date` #
echo "node: $HOSTNAME" #
echo "jobid: $SLURM_JOB_ID" # jobid

# Load modules
module load cuda/11.8

# Train the transformer-based image captioning model
python transformer.py
```

#SBATCH --job-name=captioning job-name captioning .

output error output training_captioning .

#SBATCH --nodes=2 , node 2 . , node 2 .

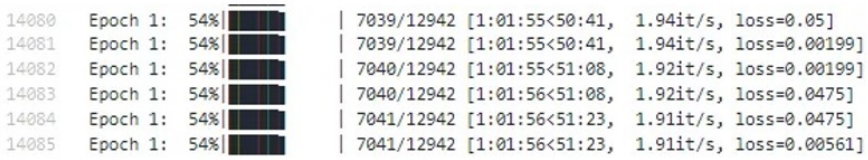
#SBATCH --gres=gpu:4 gpu 4 .

module load cuda/11.8 module cuda 11.8 version .

python transformer.py transformer.py .

sbatch transformer.sh (job) .

tqdm , error .



out log , .

“a shoe rack with some shoes and a dog sleeping on them” caption .

Captioning .

2 transformer.py

```
########################################
#### Chapter01. Environment Setting #####
########################################

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import math
from tqdm.notebook import trange, tqdm
import random

import torch
import torch.nn as nn
from torch import optim
from torch.utils.data import DataLoader
import torch.nn.functional as F
from torch.distributions import Categorical

import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torchvision

from transformers import AutoTokenizer
os.environ["TOKENIZERS_PARALLELISM"] = "false"

torch.backends.cuda.matmul.allow_tf32 = True


########################################
######## Chapter02. Model Define ########
########################################

# Define the root directory of the dataset
```

```
data_set_root='./data/coco'
train_set ='train2014'
validation_set ='val2014'

train_image_path = os.path.join(data_set_root, train_set)
train_ann_file = '{}/annotations/captions_{}.json'.format(data_set_root, train_set)

val_image_path = os.path.join(data_set_root, validation_set)
val_ann_file = '{}/annotations/captions_{}.json'.format(data_set_root, validation_set)

class SampleCaption(nn.Module):
    def __call__(self, sample):
        rand_index = random.randint(0, len(sample) - 1)
        return sample[rand_index]

tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")

class TokenDrop(nn.Module):

    def __init__(self, prob=0.1, blank_token=1, eos_token=102):
        self.prob = prob
        self.eos_token = eos_token
        self.blank_token = blank_token

    def __call__(self, sample):
        mask = torch.bernoulli(self.prob * torch.ones_like(sample)).long()
        can_drop = (~(sample == self.eos_token)).long()
        mask = mask * can_drop
        mask[:, 0] = torch.zeros_like(mask[:, 0]).long()
        replace_with = (self.blank_token * torch.ones_like(sample)).long()
        sample_out = (1 - mask) * sample + mask * replace_with
        return sample_out

def extract_patches(image_tensor, patch_size=16):
    bs, c, h, w = image_tensor.size()
    unfold = torch.nn.Unfold(kernel_size=patch_size, stride=patch_size)
    unfolded = unfold(image_tensor)
    unfolded = unfolded.transpose(1, 2).reshape(bs, -1, c * patch_size * patch_size)
    return unfolded

class SinusoidalPosEmb(nn.Module):
    def __init__(self, dim):
```

```
        super().__init__()
        self.dim = dim

    def forward(self, x):
        device = x.device
        half_dim = self.dim // 2
        emb = math.log(10000) / (half_dim - 1)
        emb = torch.exp(torch.arange(half_dim, device=device) * -emb)
        emb = x[:, None] * emb[None, :]
        emb = torch.cat((emb.sin(), emb.cos()), dim=-1)
        return emb

class AttentionBlock(nn.Module):
    def __init__(self, hidden_size=128, num_heads=4, masking=True):
        super(AttentionBlock, self).__init__()
        self.masking = masking
        self.multihead_attn = nn.MultiheadAttention(hidden_size, num_heads=num_heads, batch_

    def forward(self, x_in, kv_in, key_mask=None):
        if self.masking:
            bs, l, h = x_in.shape
            mask = torch.triu(torch.ones(l, l, device=x_in.device), 1).bool()
        else:
            mask = None
        return self.multihead_attn(x_in, kv_in, kv_in, attn_mask=mask, key_padding_mask=key_m

class TransformerBlock(nn.Module):
    def __init__(self, hidden_size=128, num_heads=4, decoder=False, masking=True):
        super(TransformerBlock, self).__init__()
        self.decoder = decoder
        self.norm1 = nn.LayerNorm(hidden_size)
        self.attn1 = AttentionBlock(hidden_size=hidden_size, num_heads=num_heads, masking=mas
        if self.decoder:
            self.norm2 = nn.LayerNorm(hidden_size)
            self.attn2 = AttentionBlock(hidden_size=hidden_size,
                                        num_heads=num_heads, masking=False)
        self.norm_mlp = nn.LayerNorm(hidden_size)
        self.mlp = nn.Sequential(nn.Linear(hidden_size, hidden_size * 4), nn.ELU(), nn.Linear

    def forward(self, x, input_key_mask=None, cross_key_mask=None, kv_cross=None):
        x = self.attn1(x, x, key_mask=input_key_mask) + x
        x = self.norm1(x)
```

```
        if self.decoder:
            x = self.attn2(x, kv_cross, key_mask=cross_key_mask) + x
            x = self.norm2(x)
        x = self.mlp(x) + x
        return self.norm_mlp(x)

class Decoder(nn.Module):
    def __init__(self, num_emb, hidden_size=128, num_layers=3, num_heads=4):
        super(Decoder, self).__init__()
        self.embedding = nn.Embedding(num_emb, hidden_size)
        self.embedding.weight.data = 0.001 * self.embedding.weight.data
        self.pos_emb = SinusoidalPosEmb(hidden_size)
        self.blocks = nn.ModuleList([TransformerBlock(hidden_size, num_heads, decoder=True) i
        self.fc_out = nn.Linear(hidden_size, num_emb)

    def forward(self, input_seq, encoder_output, input_padding_mask=None,
                encoder_padding_mask=None):
        input_embs = self.embedding(input_seq)
        bs, l, h = input_embs.shape
        seq_indx = torch.arange(l, device=input_seq.device)
        pos_emb = self.pos_emb(seq_indx).reshape(1, l, h).expand(bs, l, h)
        embs = input_embs + pos_emb
        for block in self.blocks:
            embs = block(embs, input_key_mask=input_padding_mask, cross_key_mask=encoder_pado
        return self.fc_out(embs)

class VisionEncoder(nn.Module):
    def __init__(self, image_size, channels_in, patch_size=16, hidden_size=128,
                 num_layers=3, num_heads=4):
        super(VisionEncoder, self).__init__()

        self.patch_size = patch_size
        self.fc_in = nn.Linear(channels_in * patch_size * patch_size, hidden_size)

        seq_length = (image_size // patch_size) ** 2
        self.pos_embedding = nn.Parameter(torch.empty(1, seq_length, hidden_size).normal_(sto
        self.blocks = nn.ModuleList([TransformerBlock(hidden_size, num_heads, decoder=False,
    def forward(self, image):
        bs = image.shape[0]
        patch_seq = extract_patches(image, patch_size=self.patch_size)
        patch_emb = self.fc_in(patch_seq)
        embs = patch_emb + self.pos_embedding
```

```
        for block in self.blocks:
            embs = block(embs)
        return embs

class VisionEncoderDecoder(nn.Module):
    def __init__(self, image_size, channels_in, num_emb, patch_size=16,
                 hidden_size=128, num_layers=(3, 3), num_heads=4):
        super(VisionEncoderDecoder, self).__init__()

        self.encoder = VisionEncoder(image_size=image_size, channels_in=channels_in, patch_si

        self.decoder = Decoder(num_emb=num_emb, hidden_size=hidden_size, num_layers=num_layer

    def forward(self, input_image, target_seq, padding_mask):
        bool_padding_mask = padding_mask == 0
        encoded_seq = self.encoder(image=input_image)
        decoded_seq = self.decoder(input_seq=target_seq,
                                   encoder_output=encoded_seq,
                                   input_padding_mask=bool_padding_mask)
        return decoded_seq


########################################
####### Chapter03. Model Training #######
########################################

# Define the learning rate for the optimizer
learning_rate = 1e-4

# Image size
image_size = 128

# Define the number of epochs for training
nepochs = 3

# Define the batch size for mini-batch gradient descent
batch_size = 128

# GPU
device = torch.device(1 if torch.cuda.is_available() else 'cpu')

# Embedding Size
```

```
hidden_size = 192

# Number of Transformer blocks for the (Encoder, Decoder)
num_layers = (6, 6)

# MultiheadAttention Heads
num_heads = 8

# Size of the patches
patch_size = 8

# Create model
caption_model = VisionEncoderDecoder(image_size=image_size, channels_in=test_images.shape[1]

# Initialize the optimizer with above parameters
optimizer = optim.Adam(caption_model.parameters(), lr=learning_rate)
scaler = torch.cuda.amp.GradScaler()

# Define the loss function
loss_fn = nn.CrossEntropyLoss(reduction="none")

td = TokenDrop(0.5)

# Initialize the training loss logger
training_loss_logger = []

# Transforms
train_transform = transforms.Compose([transforms.Resize(image_size),
                                      transforms.RandomCrop(image_size),
                                      transforms.ToTensor(),
                                      transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0
                                      transforms.RandomErasing(p=0.5)])

transform = transforms.Compose([transforms.Resize(image_size),
                                transforms.CenterCrop(image_size),
                                transforms.ToTensor(),
                                transforms.Normalize(mean=[0.485, 0.456, 0.406],std=[0.229, 

train_dataset = datasets.CocoCaptions(root=train_image_path,
                                      annFile=train_ann_file,
                                      transform=train_transform,
                                      target_transform=SampleCaption())
```

```
val_dataset = datasets.CocoCaptions(root=val_image_path,
                                    annFile=val_ann_file,
                                    transform=transform,
                                    target_transform=SampleCaption())

# Data Load
data_loader_train = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_worke
data_loader_val = DataLoader(val_dataset, batch_size=batch_size, shuffle=True, num_workers=8

dataiter = next(iter(data_loader_val))
test_images, test_captions = dataiter

# Iterate over epochs
for epoch in trange(0, nepochs, leave=False, desc="Epoch"):
    # Set the model in training mode
    caption_model.train()
    steps = 0
    # Iterate over the training data loader
    for images, captions in tqdm(data_loader_train, desc="Training", leave=False):
        images = images.to(device)

        # Tokenize and pre-process the captions
        tokens = tokenizer(captions, padding=True, truncation=True, return_tensors="pt")
        token_ids = tokens['input_ids'].to(device)
        padding_mask = tokens['attention_mask'].to(device)
        bs = token_ids.shape[0]

        # Shift the input sequence to create the target sequence
        target_ids = torch.cat((token_ids[:, 1:],
                                torch.zeros(bs, 1, device=device).long()), 1)
        tokens_in = td(token_ids)
        with torch.cuda.amp.autocast():
            # Forward pass
            pred = caption_model(images, tokens_in, padding_mask=padding_mask)

        # Compute the loss
        loss = (loss_fn(pred.transpose(1, 2), target_ids) * padding_mask).mean()

        # Backpropagation
        optimizer.zero_grad()
        scaler.scale(loss).backward()
        scaler.step(optimizer)
```

```
        scaler.update()

        # Log the training loss
        training_loss_logger.append(loss.item())


########################################
###### Chapter04. Model Inference #######
########################################

# Create a dataloader itterable object
dataiter = next(iter(data_loader_val))
# Sample from the itterable object
test_images, test_captions = dataiter

# Choose an index within the batch
index = 0
test_image = test_images[index].unsqueeze(0)

# Lets visualise an entire batch of images!
plt.figure(figsize = (3,3))
out = torchvision.utils.make_grid(test_image, 1, normalize=True)
_ = plt.imshow(out.numpy().transpose((1, 2, 0)))
print(test_captions[index])
```