exam 개인학습

Lec 3.4

AI summary Try on this page

Type something...

01) Self-Supervised Learning for Image Classification

01_Data Augmentation

02_Contrastive Loss

2.1 Sim

2.2 simclr_loss_naive

2.3 sim_positive_pairs

2.4 compute_sim_matrix

2.5 simclr_loss_vectorized

03_Train

01) Self-Supervised Learning for Image Classification

Pretrained Weights

For your convenience, we have given you pretrained weights (trained for ~18 hours on CIFAR-10) for the SimCLR model. Run the following cell to download pretrained model weights to be used later. (This will take ~1 minute)

```
%%bash
DIR=pretrained_model/
if [ ! -d "$DIR" ]; then
    mkdir "$DIR"
fi

URL=http://downloads.cs.stanford.edu/downloads/cs231n/pretrained_simclr_model.pth
FILE=pretrained_model/pretrained_simclr_model.pth
if [ ! -f "$FILE" ]; then
    echo "Downloading weights..."
    wget "$URL" -O "$FILE"
fi
```

- 미리 weight들을 pretrained하였다. 약 18시간정도 걸린다고 한다.
- ResNet50 + MLP들을 훈련시켰다.

01_Data Augmentation

```
train_transform = transforms.Compose([
    ###############################################################################
    # TODO: Start of your code.                                                   #
    #                                                                             #
    # Hint: Check out transformation functions defined in torchvision.transforms #
    # The first operation is filled out for you as an example.                    #
    ###############################################################################
    # Step 1: Randomly resize and crop to 32x32.
    transforms.RandomResizedCrop(32),
    # Step 2: Horizontally flip the image with probability 0.5
    transforms.RandomHorizontalFlip(p = 0.5),
    # Step 3: With a probability of 0.8, apply color jitter (you can use "color_jitter" defined above.
    transforms.RandomApply([color_jitter], p =0.8),
    # Step 4: With a probability of 0.2, convert the image to grayscale
    transforms.RandomGrayScale(p = 0.2),
    ###############################################################################
    #                                END OF YOUR CODE                             #
    ###############################################################################
    transforms.ToTensor(),
    transforms.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010])])
return train_transform
```

- `RandomResizedCrop` , `RandomHorizontalFlip` , `RandomApply` , `RandomGrayScale` 로 Data Augmentation을 구성한다.
- 확률로 다른 Data Augmentation을 만든다.

```
if self.transform is not None:
    ###############################################################################
    # TODO: Start of your code.                                                   #
    #                                                                             #
    # Apply self.transform to the image to produce x_i and x_j in the paper #
    ###############################################################################
    x_i = self.transform(img)
    x_j = self.transform(img)
```

- image를 transform에 넣어서 x_i와 x_j를 각각만들어준다.

```
class Model(nn.Module):
    def __init__(self, feature_dim=128):
        super(Model, self).__init__()

        self.f = []
        for name, module in resnet50().named_children():
            if name == 'conv1':
                module = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
            if not isinstance(module, nn.Linear) and not isinstance(module, nn.MaxPool2d):
                self.f.append(module)
        # encoder
        self.f = nn.Sequential(*self.f)
        # projection head
        self.g = nn.Sequential(nn.Linear(2048, 512, bias=False), nn.BatchNorm1d(512),
                               nn.ReLU(inplace=True), nn.Linear(512, feature_dim, bias=True))

    def forward(self, x):
        x = self.f(x)
        feature = torch.flatten(x, start_dim=1)
        out = self.g(feature)
        return F.normalize(feature, dim=-1), F.normalize(out, dim=-1)
```

- f랑 g는 모두 구현이 되어있다.

02_Contrastive Loss

$$l\left(i,j\right)=-\log\frac{\exp\left(\,\mathrm{sim}(z_i,z_j)\,/\,\tau\right)}{\sum_{k=1}^{2N}1_{k\neq i}\exp\left(\,\mathrm{sim}(z_i,z_k)\,/\,\tau\right)}$$

- 각각의 loss들을 더해 평균을 낼 것이다.
- sim함수와 loss함수를 구현해보자.

The total loss is computed across all positive pairs (i, j) in the batch. Let $z = [z_1, z_2, \ldots, z_{2N}]$ include all the augmented examples in the batch, where $z_1 \ldots z_N$ are outputs of the left branch, and $z_{N+1} \ldots z_{2N}$ are outputs of the right branch. Thus, the positive pairs are (z_k, z_{k+N}) for $\forall k \in [1, N]$.

Then, the total loss L is:

$$L = \frac{1}{2N} \sum_{k=1}^{N} [\, l(k,\ k+N) + l(k+N,\ k)\,]$$

- 논문에서는 positive pair의 위치를 (k, k+1)로 두었지만 과제에서는 편의상 (k, k + N) pair로 두었다.
- 양쪽 pair 각각의 loss를 더한 후 평균을 내준다.

2.1 Sim

```
##############################################################################
# TODO: Start of your code.                                                  #
#                                                                            #
# HINT: torch.linalg.norm might be helpful.                                  #
##############################################################################

norm_dot_product = torch.dot(z_i.T, z_j) / (torch.linalg.norm(z_i) * torch.linalg.norm(z_j))
```

- 먼저 sim함수를 구현해주자.

2.2 simclr_loss_naive

```
num1 = torch.exp(sim(z_k, z_k_N)/tau)
deno1 = 0
for i in range(2*N):
  if i == k:
    continue
  temp1 = torch.exp(sim(z_k, out[i])/tau)
  deno1 += temp1
loss1 = -torch.log(num1 / deno1)

num2 = torch.exp(sim(z_k_N, z_k)/tau)
deno2 = 0
for i in range(2*N):
  if i == k + N:
    continue
  temp2 = torch.exp(sim(z_k_N, out[i])/tau)
  deno2 += temp2
loss2 = -torch.log(num2 / deno2)

total_loss += loss1 + loss2
```

- navie한 방법으로 반복문을 돌아서 각각 sim을 구해주고 나눠주는 방식으로 구현하였다.
- 2번째 인덱스가 k + N인 것을 조심해서 작성하자.

2.3 sim_positive_pairs

- 행렬 사이즈가 같을 때 곱셈을 해주면 element-wise product로 똑같은 위치에 있는 성분끼리 곱셈을 진행한다.
- 원소는 행(row)기준이므로 dim = 1을 이용해서 합을 계산해주었다.

```
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
pos_pairs = torch.sum(out_left * out_right, dim = 1, keepdims = True)
pos_pairs /= torch.linalg.norm(out_left, dim = 1, keepdims = True)
pos_pairs /= torch.linalg.norm(out_right, dim = 1, keepdims = True)
```

- 차원을 유지해줘야하기 때문에 keepdims = True로 해주었다.

2.4 compute_sim_matrix

```
norm_out = out / torch.linalg.norm(out, dim = 1, keepdim = True)
sim_matrix = torch.matmul(norm_out, norm_out.T)
```

- i,j에 해당하는 값이 내적값이므로 원래 행렬과 전치 행렬의 행렬곱으로 표현할 수 있다.
- 곱하기 전에 norm으로 나눠줘서 미리 scale를 조정한다.

2.5 simclr_loss_vectorized

```
# Step 1: Use sim_matrix to compute the denominator value for all augmented samples.
# Hint: Compute e^{sim / tau} and store into exponential, which should have shape 2N x 2N.
exponential = torch.exp((sim_matrix / tau))

# This binary mask zeros out terms where k=i.
mask = (torch.ones_like(exponential, device=device) - torch.eye(2 * N, device=device)).to(device).bool()

# We apply the binary mask.
exponential = exponential.masked_select(mask).view(2 * N, -1)  # [2*N, 2*N-1]

# Hint: Compute the denominator values for all augmented samples. This should be a 2N x 1 vector.
denom = torch.sum(exponential, dim = 1, keepdim = True) # 2N x 1

# Step 2: Compute similarity between positive pairs.
# You can do this in two ways:
# Option 1: Extract the corresponding indices from sim_matrix.
# Option 2: Use sim_positive_pairs().
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

sim_pairs = torch.cat([sim_positive_pairs(out_left, out_right), sim_positive_pairs(out_left, out_right)],
dim = 0) # 2N x 1로 이어붙이기
```

- mask를 통해서 i = j인 값들을 0으로 만들어준다.
- sim_positive_pairs를 이용해서 값을 구해준다음 `cat dim = 0` 를 이용해서 위아래로 붙인다.

```
# Step 3: Compute the numerator value for all augmented samples.
numerator = None
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

numerator = torch.exp(sim_pairs / tau) # 2N x 1

# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# Step 4: Now that you have the numerator and denominator for all augmented samples, compute the total loss.
loss = None
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

loss = -torch.mean(torch.log(numerator/denom))
```

- 아까와 동일한 방식으로 loss를 계산한다.

03_Train

```
_, out_left = model.forward(x_i)
_, out_right = model.forward(x_j)
loss = simclr_loss_vectorized(out_left, out_right, temperature, device)
```

- forward를 통해서 out_left, out_right를 구해준 후 `simclr_loss_vectorized` 를 통해서 loss를 구해준다.
- 여기서 tau = temperature이다.

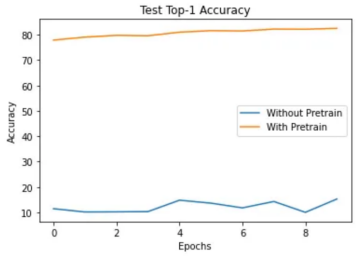
```python
class Classifier(nn.Module):
    def __init__(self, num_class):
        super(Classifier, self).__init__()

        # Encoder.
        self.f = Model().f

        # Classifier.
        self.fc = nn.Linear(2048, num_class, bias=True)

    def forward(self, x):
        x = self.f(x)
        feature = torch.flatten(x, start_dim=1)
        out = self.fc(feature)
        return out
```

- 간단한 Linear Classifier를 달아서 정확도를 확인해보자.



- pretrain된 모델의 정확도가 82%로 꽤 높은 것을 확인할 수 있다.