IIPL 인턴 2주차 보고

정윤희

INDEX

과제 수행 사항

수행사항 **기본적인 PyTorch 구현 및 cs231n 내용 숙지**

- 지난 주차 보충

Q. FC layer의 결과 점수화 과정?

- Output = input x weight + bias, 입력 데이터의 선형 결합 계산하여 점수 생성.

Q. ResNet이 vanishing gradient를 해결하기 위해 나온 모델인가?

- Degradation problem을 해결하고자 나온 모델인데, vanishing gradient도 완화된 것.

Q. Validation data 쓰는 이유?

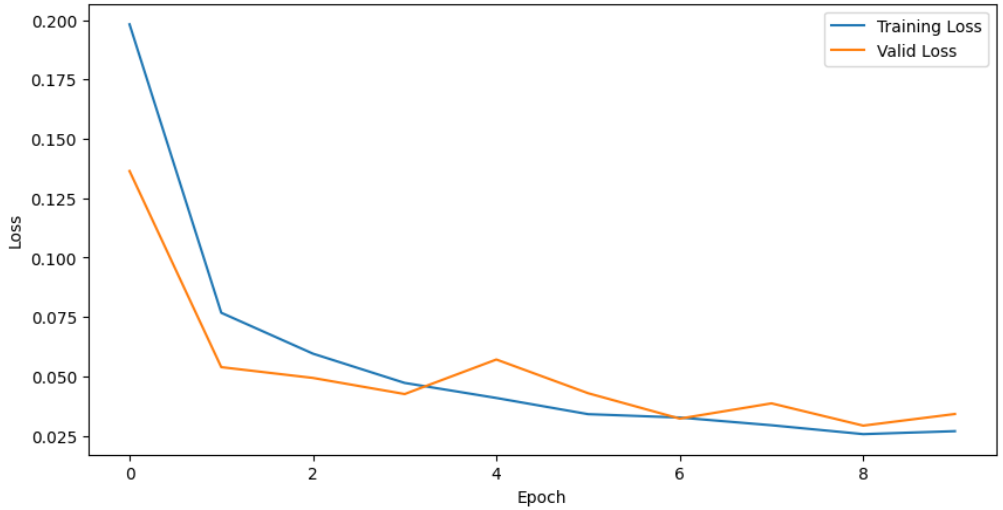
- 하이퍼파라미터 정하기 위함. (val loss가 가장 작을 때까지만 학습 -> 총 epoch 수)
- Train 데이터만으로 학습하기에는 몇 epoch에 끝내야 할지 모름. -> val data 보고 선정.

과제 수행 사항

- MNIST Data validation

  - train : valid : test = 50000 : 10000 : 10000 split

```
model.eval()
valid_loss = 0.0
with torch.no_grad():
    for data, target in valid_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        loss = criterion(output, target)

        valid_loss += loss.item() * data.size(0)

valid_loss = valid_loss / len(valid_loader.dataset)
valid_losses.append(valid_loss)

print(f'Epoch: {epoch + 1}, Training Loss: {train_loss:.4f}, Validation Loss: {valid_loss:.4f}')
```



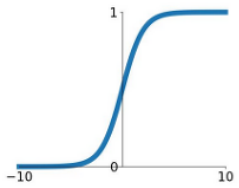
과제 수행 사항

- cs231n 6강

  - Training Neural Network
  - 활성화함수, 데이터 전처리, 가중치 초기화, BN 등

Sigmoid

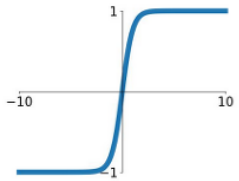
$\sigma(x) = \frac{1}{1+e^{-x}}$



- [0,1], 역전파시에 0 곱해지면서 vanishing gradient 발생.
- 출력이 zero-centered하지 않아 gradient 업데이트 비효율적

tanh

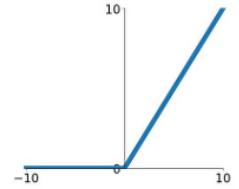
$\tanh(x)$



- [-1,1], zero-centered 문제는 해결되나, saturation으로 인해 gradient 소실.

ReLU

$\max(0, x)$

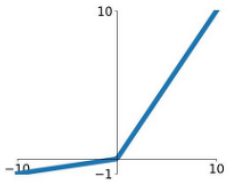


- Non zero-centered
- 양의 수에서는 saturation 되지 않으나, 음에서는 됨. (dead ReLU)

과제 수행 사항

Leaky ReLU

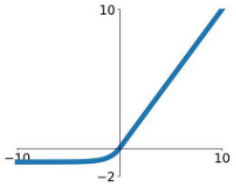
$$\max(0.1x, x)$$



- f(x) = max(0.01x, x)
- PReLU -> f(x) = max(αx, x) 기울기를 backprop으로 학습시키는 파라미터를 α로 결정, 좀 더 유연할 것.

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- ReLU와 Leaky ReLU 사이에 해당
- 0에서의 미분 불가점을 미분 가능하게 smooth 시킴.
- 음의 영역에서 기울기가 소실되고, exp 연산도 다시 생겼으므로 효율은 떨어짐.

- Data Preprocessing
  - zero-centered -> 모든 특징이 동일한 범위에 있도록 해서 결과에 동등하게 기여하도록 함.
  - Normalize -> 이미지는 이미 각 차원 안에서 스케일이 맞춰져 있으므로 할 필요 x

과제 수행 사항

- zero-centered -> 오직 첫 layer에서만 zero-centering 되기 때문에 첫 layer 정도에만 의미 있음. -> 입력값이 전부 양수이면 w가 전부 음수나 양수가 되어 업데이트가 일방향으로만 일어나는 문제 해결하지는 x.

- Weight Initialization

  - Xavier initialization: 랜덤의 가우시안 분포 값에서 np.sqrt()으로 나누어서 스케일링을 해 주는 방식. (입력값의 개수에 따라 상대적으로 값을 조절해 initialization 함.)

  - ReLU에서 뉴런의 절반이 죽는다는 근거로 np.sqrt(fan_in/2)를 해 줌 -> He initialization

- Batch Normalization (입력 스케일만 살짝 조정)

  - 현재 batch에서 평균과 분산을 이용해 normalize 수행하여 정규 분포를 따르게 함.

  - tanh로 가는 입력값들을 linear한 부분(기울기가 존재하는 범위)으로 범위를 강제함.

과제 수행 사항

- cs231n 7강

  - Training Neural Network
  - Optimization, Regularization 등

* BN: activation이 zero-mean과 unit variance가 될 수 있도록 레이어 하나 추가하는 방법

  왜? norm 전에는 손실함수가 아주 약간의 가중치 변화에도 예민했으나, norm 후 덜 민감해짐.

- SGD의 문제점
  - Local minima: 극대값들 사이의 극소값에 안착하는 경우.
  - Saddle point: 바라보는 방향에 따라 다른 값을 가지는 것. (극대, 극소)
  - Stochastic problem: 미니배치마다 loss를 계산해서 전진하는데 비효율적.

과제 수행 사항

SGD + Momentum(운동량)

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99

- Local minima에 도달해도 velocity 가지므로 gradient=0일 때도 움직일 수 있음.
- loss에 민감한 수직 방향 변동 줄이고, 수평 방향 움직임 가속화.

Adagrad

- Velocity term 대신 grad squared term 이용.
  - Grad가 작아지기 때문에 stepsize가 오랜 시간 지속되는 상황에서 느려짐.

과제 수행 사항

- RMSProp

  - if) gradient가 y축으로 가파르다 -> 불균형적으로 update할 것.

  - 경사를 보고 너무 가파르면 신중하게 이동 -> lr을 각 파라미터별로 다르게 준 셈.

- Adam

  - 초기 step이 엄청 커져 잘못될 수 있기에 bias correction term 추가.

overshoot하긴 하나, momentum만큼은 x	+	각 차원의 상황을 따로 고려해 step 이동(RMSProp)

- Regularization

  - 모델이 training data에 fit하는 것(Overfitting)을 막고, 한 번도 보지 못한 데이터(test data)에서의 성능 향상 목적. (ex. Dropout: 순전파에서 임의로 일부 뉴런 0으로 세팅)

과제 수행 사항

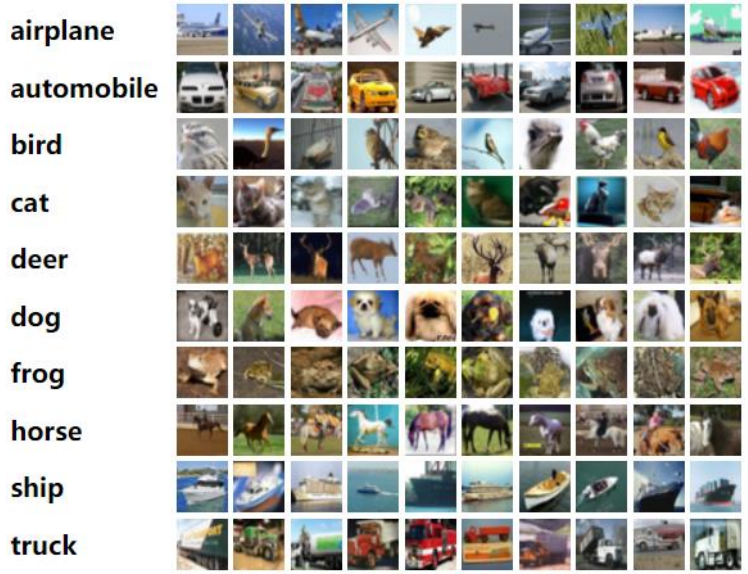
- CIFAR-10 -> 32*32 크기의 RGB 이미지가 50,000개의 train, 10,000개의 test로 구분된 set로, 비행기, 자동차, 새, 고양이, 사슴 등 10개의 정답 class로 구성.

CLASS torchvision.datasets.CIFAR10(*root: Union[str, Path], train: bool = True, transform: Optional[Callable] = None, target_transform: Optional[Callable] = None, download: bool = False*) [SOURCE]

CIFAR10 Dataset.

Parameters:

- **root** (str or `pathlib.Path`) – Root directory of dataset where directory `cifar-10-batches-py` exists or will be saved to if download is set to True.
- **train** (*bool, optional*) – If True, creates dataset from training set, otherwise creates from test set.
- **transform** (*callable, optional*) – A function/transform that takes in a PIL image and returns a transformed version. E.g, `transforms.RandomCrop`
- **target_transform** (*callable, optional*) – A function/transform that takes in the target and transforms it.
- **download** (*bool, optional*) – If true, downloads the dataset from the internet and puts it in root directory. If dataset is already downloaded, it is not downloaded again.



과제 수행 사항

- Model(VGG)

  Layer의 개수에 따라 VGG-16, VGG-19로 부름.

```
class pretrained_vgg(nn.Module):
  def __init__(self, model):
    super(pretrained_vgg, self).__init__()
    self.features = nn.Sequential(
        *list(model.features.children())
    )

    self.linear = nn.Sequential(
        nn.Linear(512, 512),
        nn.ReLU(),
        nn.Linear(512, 128),
        nn.ReLU(),
        nn.Linear(128, 10)
    )

  def forward(self, x):
    out = self.features(x).squeeze()
    out = self.linear(out)
    return out
```

- VGG 모델의 특징 추출 레이어 가져오고,
순차적 모듈로 묶어서 할당.

- self.features 통과해 특징 추출
  -> .squeeze() 통해 불필요한 차원 제거

과제 수행 사항

- Import parser

```
parser = argparse.ArgumentParser()

parser.add_argument('--epochs', type=int, default=10)
parser.add_argument('--batch_size', type=int, default=64)
parser.add_argument('--lr', type=float, default=0.001)

args = parser.parse_args()
```

```
usage: colab_kernel_launcher.py [-h] [--epochs EPOCHS] [--batch_size BATCH_SIZE] [--lr LR]
colab_kernel_launcher.py: error: unrecognized arguments: -f /root/.local/share/jupyter/runtime/kernel-36d1df04-8789-47fc-b9cf-caad6ad2e992.json
An exception has occurred, use %tb to see the full traceback.

SystemExit: 2

/usr/local/lib/python3.10/dist-packages/IPython/core/interactiveshell.py:3561: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

- argparse 라이브러리 사용시 jupyter나 colab 외에 터미널, 다른 프레임워크에서 실행해야 하는 것으로 보임.

- easydict 라이브러리 사용시 args 변수에 같은 값 저장 가능.

```
args = easydict.EasyDict({
    "batch_size":64,
    "epochs":10,
    "lr":0.001
})
```

과제 수행 사항

- 모듈화

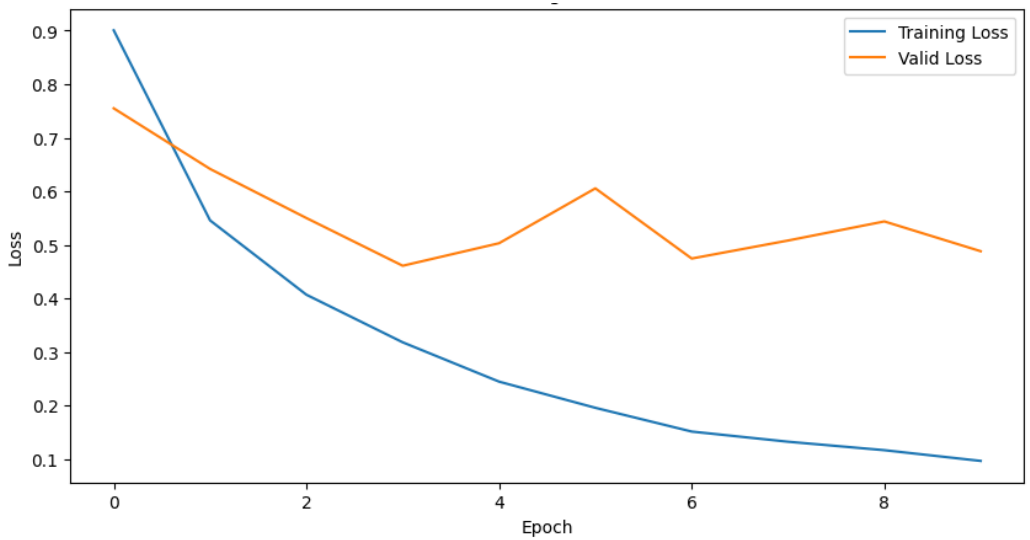
```python
def CIFAR10_DATA(batch_size=args.batch_size):
    cifar_train = datasets.CIFAR10(root='./', download = True, train=True, transform=transforms.T
    cifar_test = datasets.CIFAR10(root='./', download = True, train=False, transform=transforms.T

    train_data, valid_data = random_split(cifar_train, [40000, 10000])

    #train_dataset = CustomCi(cifar_train)
    train_dataset = CustomCi(train_data)
    valid_dataset = CustomCi(valid_data)
    test_dataset = CustomCi(cifar_test)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    valid_loader = DataLoader(valid_dataset, batch_size=batch_size, shuffle=False)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    return train_loader, valid_loader, test_loader
```

```python
def train(vgg16, train_loader, criterion, optimizer, device):
    vgg16.train()
    train_loss = 0.0
    for data, target in train_loader:
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = vgg16(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()

        train_loss += loss.item() * data.size(0)
        train_loss = train_loss / len(train_loader.dataset)
    return train_loss
```

과제 수행 사항

- Training

Epoch 10




```
Epoch: 1, Training Loss: 0.9006, Validation Loss: 0.7550
Epoch: 2, Training Loss: 0.5457, Validation Loss: 0.6418
Epoch: 3, Training Loss: 0.4069, Validation Loss: 0.5501
Epoch: 4, Training Loss: 0.3181, Validation Loss: 0.4610
Epoch: 5, Training Loss: 0.2447, Validation Loss: 0.5031
Epoch: 6, Training Loss: 0.1958, Validation Loss: 0.6055
Epoch: 7, Training Loss: 0.1514, Validation Loss: 0.4745
Epoch: 8, Training Loss: 0.1325, Validation Loss: 0.5081
Epoch: 9, Training Loss: 0.1166, Validation Loss: 0.5438
Epoch: 10, Training Loss: 0.0966, Validation Loss: 0.4882
```

- 시도할 수 있는 방안 제시

  - 데이터에 비해 모델이 너무 복잡할 수 있음 -> 정규화 통한 복잡도 조절 또는 다른 모델 구조 시도

  - Data augmentation -> 훈련 데이터 늘려서 모델의 일반화 능력 향상

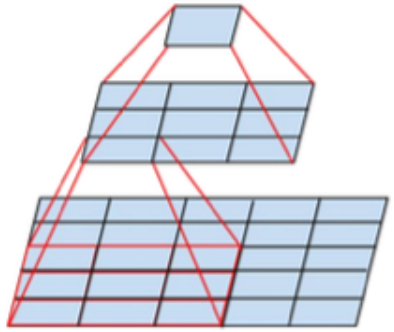
개인 수행 사항

- VGG Net(Very Deep Convolutional Networks for Large-scale Image Recognition) 논문 리뷰

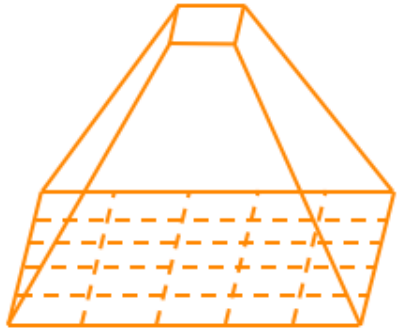
https://arxiv.org/pdf/1409.1556.pdf

Main idea

- 3 x 3 Conv, stride 1, pad 1과 2 x 2 MAX POOL stride 2만 사용.
- 더 깊게(Deeper) layer를 쌓음.



two successive
3x3 convolutions



5x5 convolution

CNN에서 Filter의 Receptive field의 효과 살린 것.

Filter가 한 번 훑을 때 보는 영역, 영역이 클수록 input image의 많은 특징을 잡을 수 있음.

-> Receptive field는 넓게 가져가면서 FC layer와 같아지지 않게 하기 위한 방법 제안.

개인 수행 사항

- VGG Net의 특징

  - 학습이 필요한 parameter를 줄임.

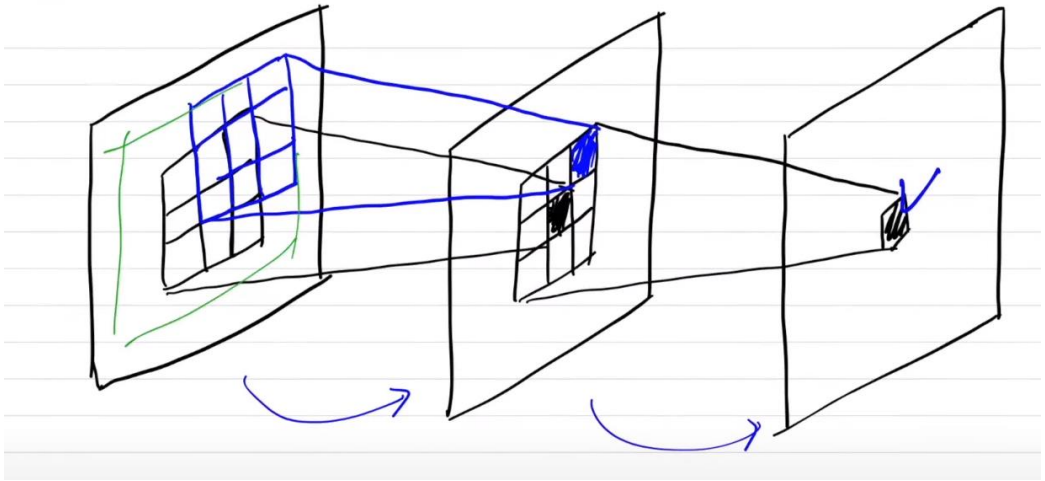
    3 x 3 filter 2 layer -> 2 x (3 x 3) = 18

    5 x 5 filter 1 layer -> 1 x (5 x 5) = 25

  - 동시에 layer가 많아짐에 따라 비선형성 더 반영할 수 있음.

    - 비선형성 커질수록 표현력 높아짐. -> 복잡한 관계를 더 잘 표현할 수 있다.

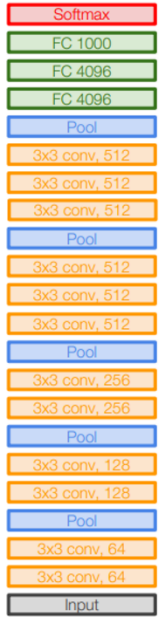
  - 3 x 3 conv layer를 깊게 쌓아 7 x 7 과 동일한 receptive field 효과를 냄.



개인 수행 사항

Layer	Output Size	Memory	Number of Parameters
INPUT	[224x224x3]	224*224*3=150K	0
CONV1-1	[224x224x64]	224*224*64=**3.2M**	(3*3*3)*64 = 1,728
CONV1-2	[224x224x64]	224*224*64=**3.2M**	(3*3*64)*64 = 36,864
POOL1	[112x112x64]	112*112*64=800K	0
CONV2-1	[112x112x128]	112*112*128=1.6M	(3*3*64)*128 = 73,728
CONV2-2	[112x112x128]	112*112*128=1.6M	(3*3*128)*128 = 147,456
POOL2	[56x56x128]	56*56*128=400K	0
CONV3-1	[56x56x256]	56*56*256=800K	(3*3*128)*256 = 294,912
CONV3-2	[56x56x256]	56*56*256=800K	(3*3*256)*256 = 589,824
CONV3-3	[56x56x256]	56*56*256=800K	(3*3*256)*256 = 589,824
POOL3	[28x28x256]	28*28*256=200K	0
CONV4-1	[28x28x512]	28*28*512=400K	(3*3*256)*512 = 1,179,648
CONV4-2	[28x28x512]	28*28*512=400K	(3*3*512)*512 = 2,359,296
CONV4-3	[28x28x512]	28*28*512=400K	(3*3*512)*512 = 2,359,296
POOL4	[14x14x512]	14*14*512=100K	0
CONV5-1	[14x14x512]	14*14*512=100K	(3*3*512)*512 = 2,359,296
CONV5-2	[14x14x512]	14*14*512=100K	(3*3*512)*512 = 2,359,296
CONV5-3	[14x14x512]	14*14*512=100K	(3*3*512)*512 = 2,359,296
POOL5	[7x7x512]	7*7*512=25K	0
FC6	[1x1x4096]	4096	7*7*512*4096 = **102,760,448**
FC7	[1x1x4096]	4096	4096*4096 = 16,777,216
FC8	[1x1x1000]	1000	4096*1000 = 4,096,000

***Total Memory: 24M * 4 bytes ~= 96MB / image (only forward! x2 for backward)**
***Total Parameters: 138M Parameters**



VGG16

Details

- Dropout 0.5 for FC6, FC7
- Batch size 256
- SGD + Momentum 0.9
- Initial lr 0.01, reduced to 1/10 twice
- L2 weight decay 5e-4

향후 활동

향후활동

- cs231n 및 cs224n 공부
- data augmentation 및 state_dict() 활용한 가중치 값 저장 및 로드 진행
- Gradient clipping, schedular 등 통한 성능 향상 (목표 정확도 85%)
- 최근 기술 동향 또는 이슈 논문 1~2편 리뷰

정윤희

감사합니다