Parrot Deep Learning

Session 04.
Batch, Epoch, DL flow

박준서(p_jun)

- 좋은 모델이란 뭘까?
- 확률과 가능도
- 어떤 파라미터의 값을 '얼마나' 변경시켜야 할까?

Batch Size & Epoch

- 수 GB ~ 수백 GB의 데이터를 학습에 이용한다
- 따라서 모든 데이터를 한 번에 올리기엔 램과 메모리가 모자라다
- 데이터를 나누어 계산을 동시에 처리하자 (GPU)

- e.g. batch size = 128 -> 128개의 데이터를 한번에 학습

Batch Size & Epoch

- 만약 4096장의 이미지, batch가 128이라면?
- 32개의 batch를 이용하여 학습하는 것

- 이 32을 iteration 또는 step이라 부른다.
- 데이터 수 / batch size = iteration

Batch Size & Epoch

- 만약 4096장의 이미지, batch가 128이라면?
- 32개의 batch를 이용하여 학습하는 것

- 이 32을 iteration 또는 step이라 부른다.
- 데이터 수 / batch size = iteration

Batch Size & Epoch

- Batch size 선택 가이드
- 보편적으로 사용하는 배치 정규화로 인해 배치 크기에 따라 학습이 영향을 받습니다

- Batch size가 크다면?
- Training set 분포를 더 근사하여 추정하므로, noise가 줄고 잘 수렴한다
- 그렇기에 overfitting이 일어날 수도 있다

- Batch size가 작다면?
- 적은 값을 샘플링하므로 noise가 커지고, 모델이 불안정해 진다
- 따라서 overfitting을 방지하여 모델 성능을 키울 수도 있다

Batch Size & Epoch

- Batch size 선택 가이드
- 보편적으로 사용하는 배치 정규화로 인해 배치 크기에 따라 학습이 영향을 받습니다

- Batch size가 크다면?
- Training set 분포를 더 근사하여 추정하므로, noise가 줄고 잘 수렴한다
- 그렇기에 overfitting이 일어날 수도 있다

- Batch size가 작다면?
- 적은 값을 샘플링하므로 noise가 커지고, 모델이 불안정해 진다 (정규화가 강해진다)
- 따라서 overfitting을 방지하여 모델 성능을 키울 수도 있다

Batch Size & Epoch

- 실험적으로 진행해보자!

- 32 ~ 128 정도의 batch size를 선택하는게 좋은 것 같다..
- Rethinking 'Batch' in BatchNorm(Facebook) - 2021

Batch Size & Epoch

- Epoch는 전체 데이터를 통과한 횟수이다
- 학습을 반복할 횟수라 생각하다

- e.g. epoch = 3, 전체 데이터를 3번 학습함

- 에포크가 적으면 충분히 학습이 되지 않고, 많으면 과적합이 일어난다
- -> 적당히 시각화를 해보며 확인하자

Softmax

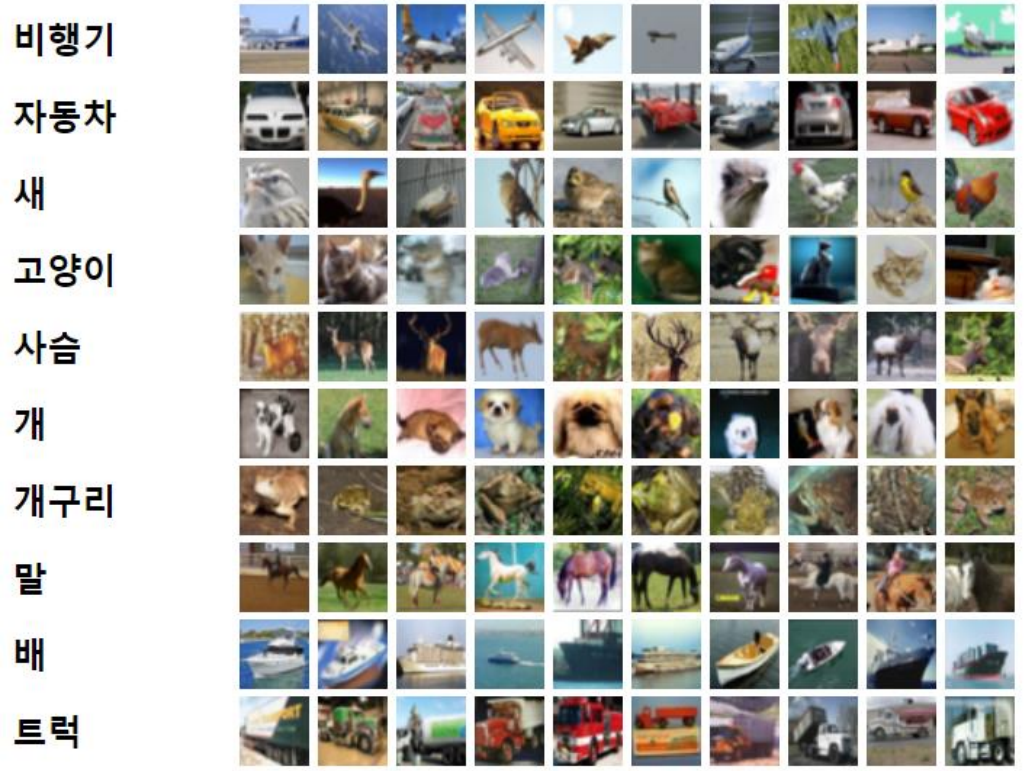
- Softmax는 활성화 함수의 일종으로 입력 값을 0~1로 정규화 하여 출력해 준다
- 총합이 1이 되도록 만들어 준다 (확률)

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- Class 분류 문제의 마지막 layer의 활성화 함수로 사용해 주자

DL flow

- 10가지 클래스가 있는 CIFAR10 데이터 셋을 통해 딥러닝 코드를 알아봅시다



Transforms

- 데이터 선처리 함수입니다
- 크기 조절, agumentation, 정규화 등등 다양한 기능이 있습니다
- Pandas에서 df을 다룬 것 처럼 pytorch에선 tensor를 다룹니다

- (x - 0.5)/0.5 분포로 정규화 해 주는 코드입니다

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
```

Datasets, Dataloder

- Datasets에는 다양한 데이터가 있습니다
- 그 중 CIFAR10을 가져와 데이터 로더를 만들어줍니다

```
[ ]  trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                             download=True, transform=transform)
     train_loader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                               shuffle=True, num_workers=2)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:05<00:00, 29324628.46it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
```

```
[ ]  valset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                           download=True, transform=transform)
     val_loader = torch.utils.data.DataLoader(valset, batch_size=4,
                                             shuffle=False, num_workers=2)
```

```
Files already downloaded and verified
```

- 데이터 로더는 배치 크기만큼 데이터를 뽑아 분배해주는 파이프라인입니다

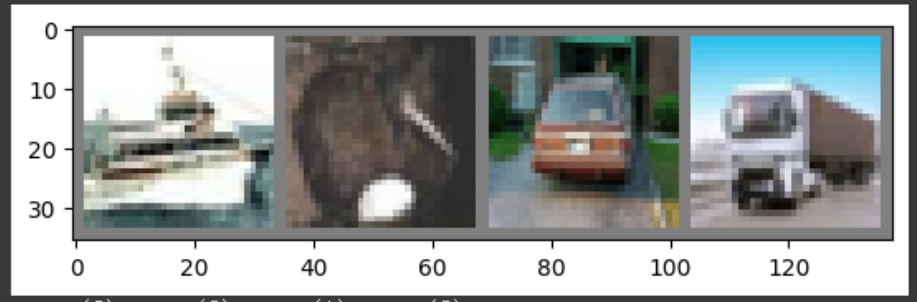
Datasets, Dataloder

- **무작위로 이미지가 잘 있는지 확인 해 봅시다**

```
[ ]  def imshow(img):
         img = img / 2 + 0.5     # unnormalize
         npimg = img.numpy()
         plt.imshow(np.transpose(npimg, (1, 2, 0)))
         plt.show()
```

```
[ ]  dataiter = iter(train_loader)
     images, labels = next(dataiter)

     imshow(torchvision.utils.make_grid(images))
     print(' '.join('%5s' % labels[j] for j in range(4)))
     #classes = 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'
```




```
tensor(8) tensor(2) tensor(1) tensor(9)
```

Model

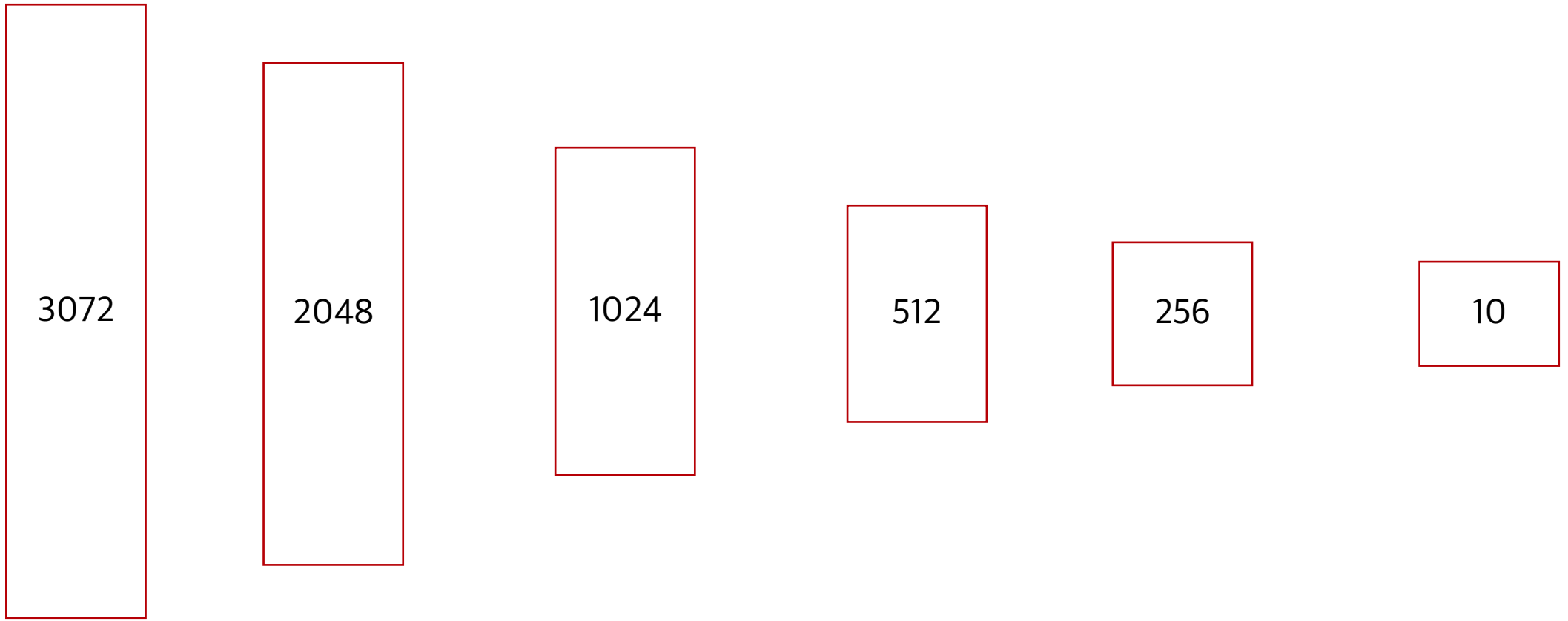
- nn.Module을 상속받아 모델을 만들 수 있습니다
- nn.Linear은 선형 레이어를 추가해 줍니다

- forward에서 모델의 각 레이러를 통과 시켜줍니다
- 각 레이어의 끝에 relu 활성화 함수를 적용시킵니다
- relu가 아닌 sigmoid, tanh등등 선택 가능합니다

```
class Mymodel(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(3072, 2048)
        self.fc2 = nn.Linear(2048, 1024)
        self.fc3 = nn.Linear(1024, 512)
        self.fc4 = nn.Linear(512, 256)
        self.fc5 = nn.Linear(256, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = self.fc5(x)
        return x
```

Model



- 모델의 구조는 위와 같습니다
- 첫 레이어는 입력, 마지막 레이어는 클래스의 수로 부터 왔습니다

Model

- Nn의 Sequential을 사용하면 다음과 같은 방법으로 동일한 모델 정의가 가능합니다

```
class SequentiaMymodel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear_model = nn.Sequential(
            nn.Linear(3072, 2048),
            nn.ReLU(),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        x = self.linear_model(x)
        return x
```

GPU

- Cuda를 GPU라고 생각하면 됩니다
- GPU가 없다면 cpu를 할당

```
model = Mymodel()

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)
```

- to(device)는 해당 데이터를 device로 처리하겠다는 코드

- 이 코드를 잘 적어줘야 GPU 연산이 가능합니다

training

```
for epoch in range(EPOCH):
    train_loss = 0
    val_loss = 0
    total = 0
    correct = 0
    for i, data in enumerate(train_loader):
        x, y = data
        x = x.reshape(-1, 32*32*3) # flatten
        x = x.to(device)
        y = y.to(device)

        optimizer.zero_grad() #gradient 초기화

        y_pred = model(x)

        loss = criterion(y_pred, y)

        loss.backward() #backpropagation
        optimizer.step() #update

        train_loss += loss.item()
    train_loss/=len(train_loader)
    train_loss_history.append(train_loss)
```

- Epoch 만큼 전체 데이터를 반복합니다
- Batch 크기 만큼 한번에 가져와 연산을 해줍니다
- Backpropagation은 함수가 잘 해줍니다
- Batch_size로 나눠준 이유는 평균 계산 위함

validation

```
with torch.no_grad(): # val set 학습을 꺼둠
    for i, data in enumerate(val_loader):
        x, y = data
        x = x.reshape(-1, 32*32*3) # flatten
        x = x.to(device)
        y = y.to(device)

        output = model(x)
        loss = criterion(output, y)

        val_loss += loss.item()


        _, pred = torch.max(output.data, 1)
        total += y.size(0)
        correct += (pred == y).sum().item()
val_loss /= len(val_loader)
val_loss_history.append(val_loss)
val_acc = 100 * correct / total
```

- 학습과 거의 동일한 코드입니다
- 다만 기울기 계산으로 학습을 하지 않습니다

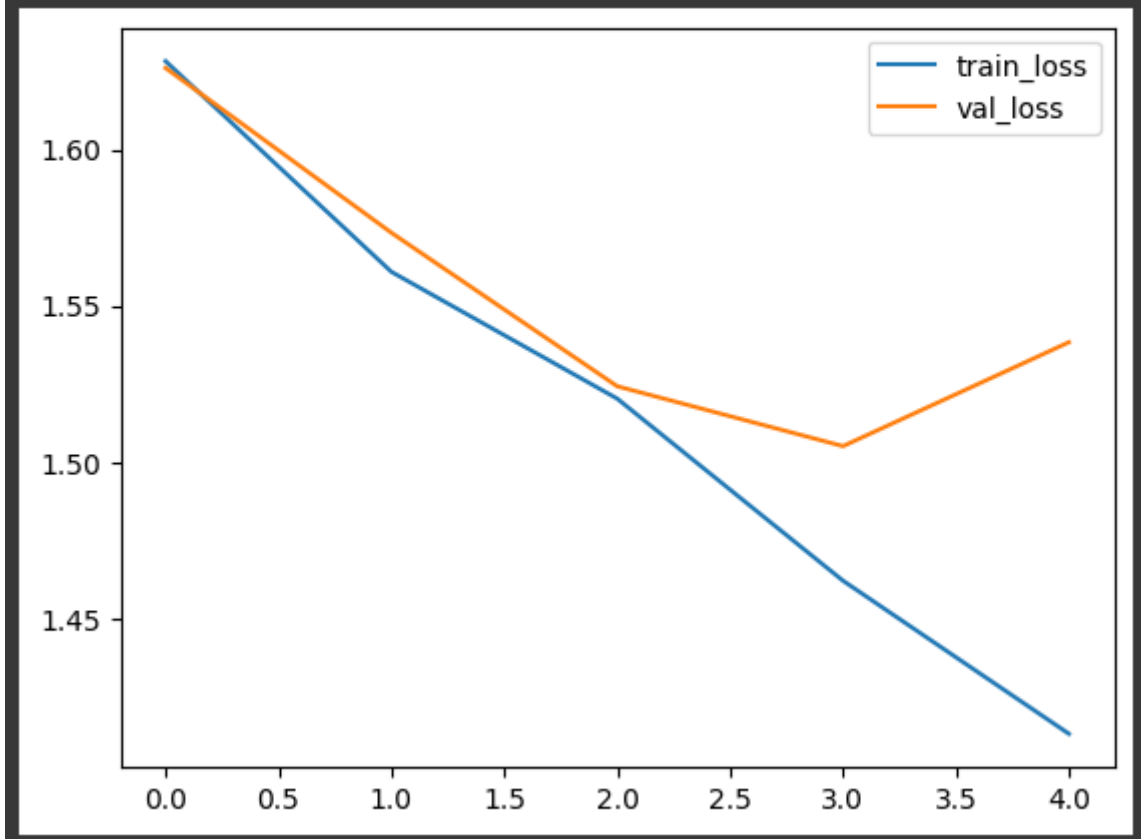
- 이 코드는 accuracy 계산을 위한 코드입니다

test

- Test set 코드는 생략하였습니다.. 만
- Valid 처럼 model에 x를 넣어주고, max인 라벨을 찾아주면 됩니다

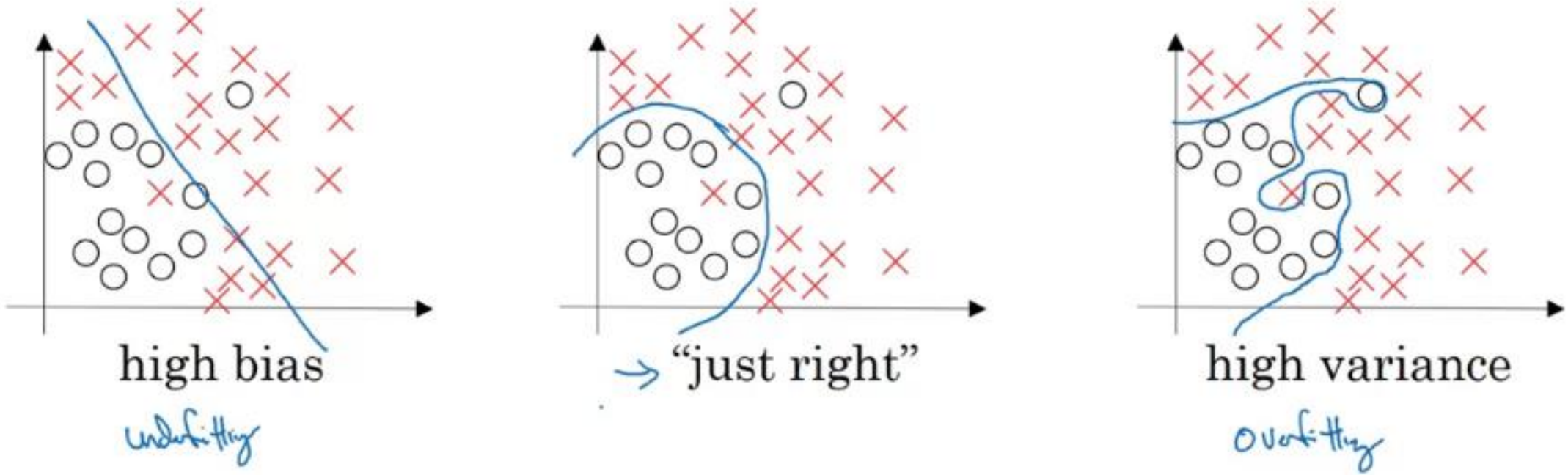
visualization

```
plt.plot(np.arange(epoch+1), train_loss_history, label='train_loss')
plt.plot(np.arange(epoch+1), val_loss_history, label='val_loss')
plt.legend()
plt.show()
```



- 계산된 loss의 그래프입니다
- 4번 에포크 이후로 val이 증가하였습니다
- Overffiting?

Bias / Variance



- Train err는 낮지만, valid err가 높으면? Overfitting
- Train err가 높고 valid err도 높으면? Undefitting

Bias / Variance

- Bias와 variance 모두 낮아야 학습이 잘 진행되었다고 할 수 있다
- Train 예측 잘하고, valid도 예측 잘함 (물론 test에 대해 낮으면 둘 모두에 variance 높음)

- 과거에는 bias / variance trade off로 둘 중 하나를 포기해야만 했지만
- 최근에는 데이터를 많~~~이 구하거나 네트워크 변경으로 둘 모두 좋은 성능을 얻을 수 있다
- Ex) transformer

Bias / Variance

- Bias가 높다면?
- 모델의 크기를 키운다 (깊이를 깊게 하고, 유닛을 늘리고, 다른 알고리즘 사용)
- 학습을 더 오래 시킨다 (Epoch 증가)
- 다른 형태의 네트워크를 사용한다 (직선 말고 곡선, CNN 등)

- Variance가 높다면?
- 데이터 수를 늘린다
- 정규화를 적용시킨다
- 다른 네트워크 아키텍터를 적용시킨다

QnA

- 질문 있으신가요?