겨울학기 연구참여 Final Project Report

20190857 신중혁

1. 개요

PyTorch 는 다양한 pretrained model 을 비롯해 여러 종류의 loss function, optimizer 등을 제공하여 Transfer Learning 을 간편하게 실험해볼 수 있게 해준다. 본 과제에서는 다양한 세팅을 통해 Image Net 에서 pretrained 된 CNN 모델들의 성능을 분석해볼 것이다. 발표 및 Lecture 에서 다룬 model 들인 AlexNet, VGGNet (vgg16), 그리고 ResNet(resnet18)의 Image Net Pretrained Model 들을 가지고 CIFAR-10에 맞게 fine-tuning 실험을 진행하였고, Initial Learning Rate, Batch Size, Optimizer 등에 따른 실행 시간과 수렴 속도 그리고 정확도를 측정하였다. 보다 자세한 세팅은 학습 방법 및 환경에서 이야기 하겠다. 모든 경우에 대해 테스트를 할 수 있다면 좋겠지만, 사용할 수 있는 리소스와 시간의 제약으로 대표성을 띌 수 있는 총 14개의 세팅에서 실험을 진행하였다.

2. 코드 요약

총 14개의 세팅에서 실험을 진행하는 데 사용된 14개의 notebook 을 첨부하였다. 처음에는 하나의 notebook 에서 cell 들을 나누어 한 번에 실행시키고자 하였지만, 실험할 세팅이 14개인데다 구글 colab 및 로컬 데스크톱에서 나누어 실행시키기 위해 단순히 14개를 만들었다. 이렇게 하면 이전 cell 에서 실행시킨 뒤 남아있는 데이터의 영향을 받아 새로운 학습에 지장이 생기는 문제 역시 간단히 피해갈 수 있다.
코드 자체는 PyTorch 공식 웹페이지의 tutorial 을 많이 참고하였다. 다만 텐서보드를 통해 실험 과정을 분석하기 위해 약간의 수정을 거쳤고, 모델은 다음과 같이 구성하였다.
모델은 PyTorch Docs 에 나와있는 방식으로 직관적으로 구현하였다.

```
'''
#AlexNet
alexnet = models.alexnet(pretrained=True)
num_ftrs = alexnet.classifier[6].in_features
alexnet.classifier[6]=nn.Linear(num_ftrs, len(classes))
alexnet = alexnet.to(device)

#VGG16
vgg16 = models.vgg16(pretrained=True)
num_ftrs = vgg16.classifier[6].in_features
vgg16.classifier[6]=nn.Linear(num_ftrs, len(classes))
vgg16 = vgg16.to(device)

'''

#Resnet18
resnet18 = models.resnet18(pretrained=True)
num_ftrs = resnet18.fc.in_features #number of output nodes
resnet18.fc = nn.Linear(num_ftrs, len(classes))
resnet18 = resnet18.to(device)
```

train_model 함수를 비롯한 상당 부분이 tutorial 의 코드와 유사하므로 불필요한 설명은 생략하도록 하겠다.

3. 학습 방법 및 환경

사실 이번 과제를 진행하면서 가장 많이 신경을 썼던 부분이다. 실험해보고자 하는 세팅에 비해 구글 colab 의 속도나 세션 제한이 비교적 타이트하여, 가지고 있는 데스크톱으로 실험을 일부 진행하였다. 로컬 데스크톱의 경우 RTX 3070 그래픽 카드를 사용하였는데, 아직 RTX 30시리즈들을 많은 딥러닝 라이브러리들이 정식 지원하지 않아 드라이버 관련 많은 문제를 겪었다. 또한 비교적 최신 gpu 이다 보니 colab 의 gpu 보다 실행 시간이 거의 2배 가까이 빨라 실행 시간을 비교하고자 했던 부분들은 모두 colab 이나 local 중 하나로 통일하여 진행하였다. 아래는 실험한 세팅들이다. C 는 구글 Colab 을 사용하여 진행한 경우, L 은 Local 데스크톱을 이용하여 진행한 실험이다.

Batch Size	Model	Loss Function	Optimizer - Initial Learning Rate - Momentum (For LR decay, exponential lr scheduler was used for all case)					
			SGD-0.1–0.9	SGD-0.001-0.9	SGD-0.0001-0.9	SGD-0.001-0.45	SGD-0.001- 0.0	Adam-0.001
16	AlexNet	CrossEntropy		C				C
	VGG16			C				C
	Resnet18		L	L, C	L	L	L	C
8				L				
32				L				
64				L				

Model 로는 위에서 언급했듯, AlexNet, VGG16, Resnet18을 사용하였고, Optimizer 로는 SGD 와 Adam, Initial Learning rate 는 0.1~0.00001, 그리고 momentum 은 0.9, 0.45, 0.0을 기준으로 실험을 진행하였다. 원래는 Loss Function 의 종류, LR scheduler 의 유형과 step size 도 변수로 두고 싶었으나, 시간이 너무 오래 걸려 Loss Function 은 CrossEntropy 로 통일했고, LR decay 는 exponential scheduler 에 5 번째 epoch 마다 LR 을 0.1배 하는 것으로 통일하였다. Epoch 의 경우, colab 으로 진행한 실험은 30, 그리고 로컬에서 진행한 실험은 20번 진행하였다.

colab 으로는 서로 다른 모델 간 SGD, Adam 을 이용한 성능을 분석했고, Local 데스크톱으로는 Optimizer SGD 의 Learning Rate 과 Momentum, 그리고 batch size 에 따른 성능을 분석하였다. Batch Size, SGD 의 Learning Rate 와 Momentum 을 비교하는 데에는 resnet18을 표준으로 이용하였다.

같은 row, column 의 세팅들끼리 비교하면 다른 조건이 대부분 동일한 상황이므로 합리적인 비교라고 할 수 있다.

4. 학습 결과 및 분석

참고 : trained weight 을 저장하는 것을 깜빡하고, 대부분의 세팅에서 torch.save 를 진행하지 못한 채로 노트북 / colab 을 종료하여 trained weight 이 없습니다. 다만 SGD-

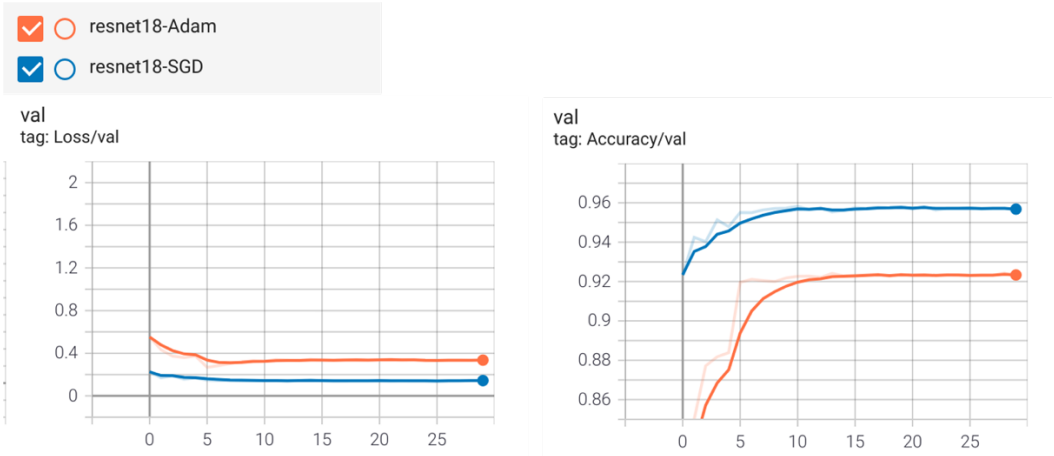
0.1– 0.9, SGD-0.001– 0.9, SGD-0.00001– 0.9, 그리고 SGD-0.001-0.45의 4 경우는 다시 학습시켜, trained weight 을 저장하였습니다. (일부는 텐서보드 결과로 확인해주시면 감사하겠습니다.) colab 에서 실행한 **4.1**과 같은 일부 세팅에서 colab 이 세션 활성화 시간 제한으로 학습 종료 후 바로 끊겨버려, 텐서보드 저장 값도 함께 없어졌습니다. 이들의 경우 남아 있는 것들이 output 밖에 없어 output 을 기반으로 보고서를 적었습니다.
주말에 학습 시킨 후 화요일에 이 부분을 알게 되었는데, colab 에서 gpu 사용량 제한이 걸려서 다시 돌리기는 어려웠던 점 양해 부탁드립니다.

4.1 AlexNet vs VGG16 vs Resnet18 (with SGD, Adam)

Epoch 당 평균 실행 시간 (sec) (총 30 epoch 실행)	AlexNet	VGG16	Resnet18
Adam Optimizer	147	946	125
SGD Optimizer	166	909	122

Best Accuracy	AlexNet	VGG16	Resnet18
Adam Optimizer	0.6171	0.1201	0.9233
SGD Optimizer	0.9466	0.9455	0.9568

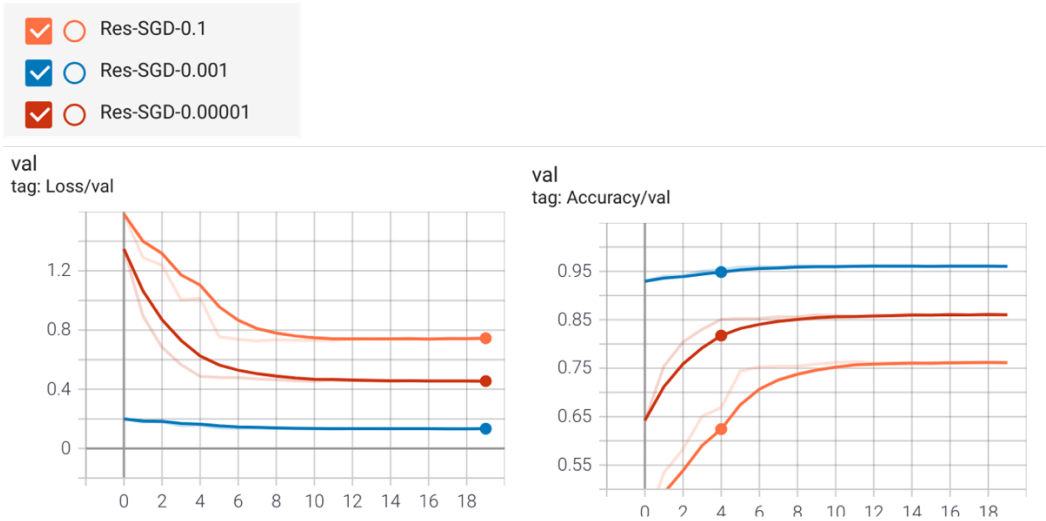
아래는 Resnet18의 epoch 에 따른 그래프이다.



위에서 언급한 대로 모두 initial learning rate 0.001로 시작했고, batch size 와 loss function 은 16과 cross entropy loss 로 고정했다.
가장 빠른 실행 시간과 높은 정확도를 보여준 것 모두 Resnet18이다. 눈에 띄는 점이 있다면 VGG16의 경우에는 실행하는데 매우 긴 시간이 걸렸다는 점이고, Adam Optimizer 의 경우 AlexNet 과 VGG16에서 충분히 좋지 못한 값으로 수렴하지 못하고 있는 점을 확인할 수 있다. AlexNet 의 경우 learning rate 을 조금 수정하거나 hyper parameter 를 바꾸면 보다 나은 결과를 얻을 수 있을 것으로 보이나, VGG16의 경우에는 accuracy 가 0.12라 사실상 학습을 제대로 하지 못한 것으로 보인다. CS231n 앞 쪽에서 잠깐 언급한 것처럼 Initialization 이 아예 잘못된 것으로 보인다.
일반적으로 Adam 은 SGD Momentum, RMSProp 등의 장점을 합쳐 만든 만큼, 더

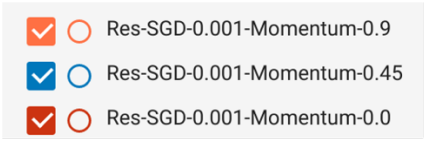
좋은 성능을 낸다고 하는데, 어디에서 문제가 생겼는지는 아마 더 실험을 해봐야 알 수 있을 것 같다.
전반적으로 image net 에서 이미 충분히 학습된 pretrained model 을 이용해서인지, epoch 1부터 매우 높은 accuracy 를 보여줌을 확인할 수 있고, 이로 인해 모델들 간의 오차율 차이는 매우 미미한 수준임을 확인할 수 있다. pretrained model 을 사용하지 않고 base 부터 학습했다면 epoch 30 수준에서 유의미한 차이를 보였을 것이다.

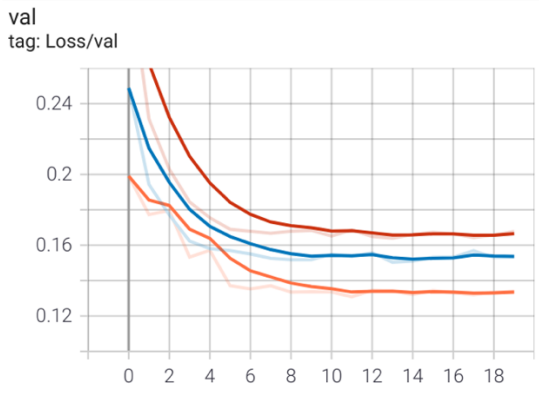
4.2 Effect of initial Learning Rate in SGD (0.1 vs 0.001 vs 0.00001)

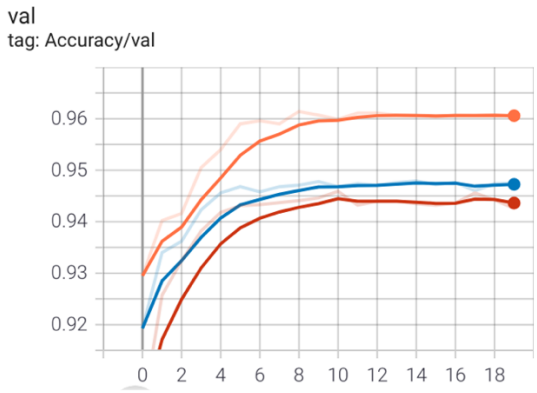


Resnet18 – Batch size 16 – Cross Entropy – SGD Momentum 0.9 – Exponential LR decay 를 고정한 채로 SGD 의 initial learning rate 에 따른 수렴 속도를 살펴보자.
실행 시간에 있어서는 모두 1초 미만의 평균 차이를 보여 오차범위 내 모두 동일한 시간이 걸렸다고 볼 수 있다.
그래프를 통해 Initial Learning rate 가 0.001 – 0.00001 – 0.1 순으로 높은 정확도를 가지고 있음을 확인할 수 있다. 이론대로라면 Learning rate 가 너무 큰 경우, optimum 값에 도달하지 못할 가능성이 있고, 너무 작을 경우 수렴하는 속도가 너무 느릴 수 있다. 사실 학습 속도가 가장 느린 경우는 LR 이 0.001일 때였는데, 아무래도 첫 epoch 부터 Loss 가 매우 적은, 사실상 optimum 상태로 시작하여 천천히 학습된 것으로 보인다. initial learning rate 가 0.1로 큰 경우에는 epoch 20 때 좋은 optimum 에 도달하지 못한 것을 확인할 수 있다. (물론 LR decay 가 있기 때문에 반복한다면 조금은 더 나아질 여지가 있긴 하다)
전반적으로 LR 이 0.001일 때 가장 좋은 결과를 보였기 때문에 아래 실험들에서는 모두 LR 을 0.001로 고정하였다.

4.3 Effect of Momentum in SGD (0.9 vs 0.45 vs 0.0)





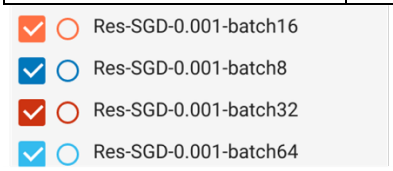


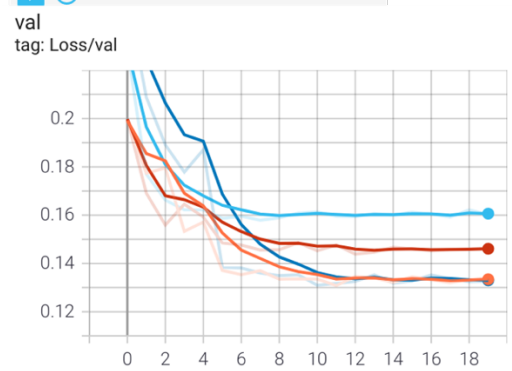
Resnet18 – Batch size 16 – Cross Entropy – SGD LR 0.001 – Exponential LR decay 로 고정한 채로 실험을 진행하였다. 모멘텀은 Gradient Descent 에 관성과 같은 효과를 더해주어 local minimum 에 빠져도 쉽게 빠져나와 global minimum 에 빠르게 수렴할 수 있도록 해준다.

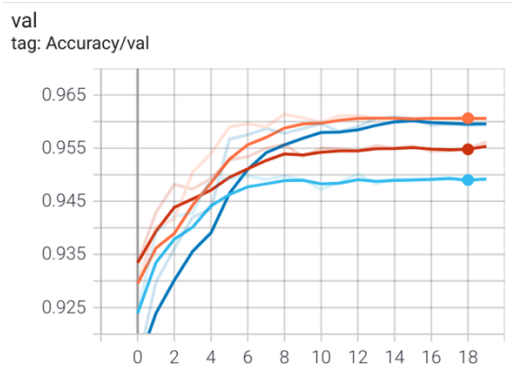
다행히도 이론과 일치하는 결과를 얻을 수 있었다. Momentum 이 0.9, 즉 가장 컸을 때 가장 Global Minimum 에 가까운 weight 으로 수렴한 것으로 보인다. 0.45나 0 이었을 때에는 Local Minimum 에 빠진 것으로 해석할 수 있다.

4.4 Effect of Batch Size

Epoch 당 평균 실행시간 (total 20 epoch)	Batch Size 8	Batch Size 16	Batch Size 32	Batch Size 64
	115.7	96.25	84.2	82.65

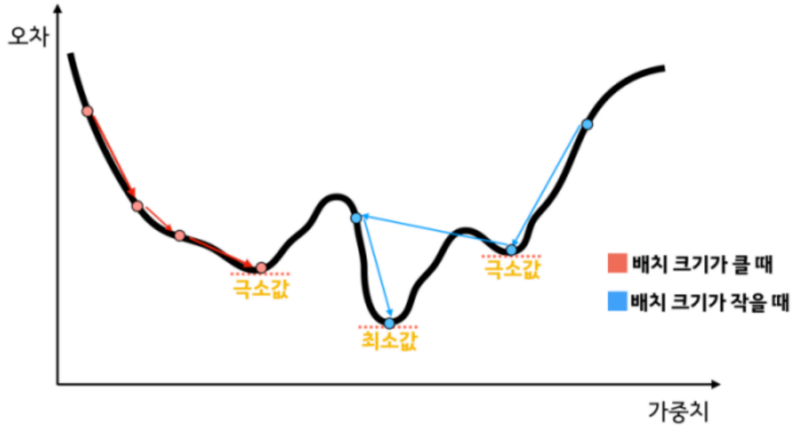






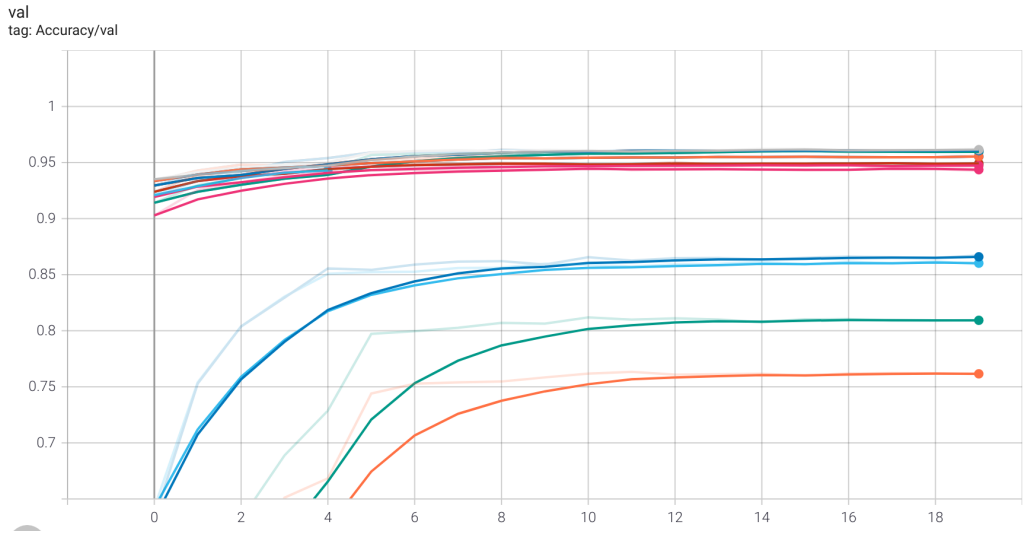
적절한 Batch Size 를 정하는 것은 일반적으로 매우 중요하다. Batch Size 가 크면 위 실험 결과에서도 볼 수 있듯 총 훈련 시간은 줄어든다. Epoch 당 가중치를 업데이트 해야하는 횟수가 적기 때문이다. 하지만 이 경우 모델의 최적화와 일반화가

어려워진다. 또한 Batch Size 가 큰 경우, 오차가 local minimum 에 빠지기 쉬운데, 그 이유는 이미 오차를 최소화하는 방향이 정해졌을 가능성이 크기 때문이다.



이에 기반해 실험 결과를 살펴보면, batch size 16 – 8- 32- 64 순으로 Accuracy 가 나왔으며, Batch Size 8과 16은 거의 오차범위내 유사함을 확인할 수 있다. 즉, Batch Size 가 큰 경우 global minimum 보다는 local minimum 쪽에 빠졌다고 해석할 수 있겠다.

5. 결론



Pretrained Model 들을 사용하여 평균적으로 매우 높은 Accuracy 를 얻어냈지만, 실제로 이렇게까지 Train 되려면 많은 시간이 걸린다. Base 부터 시작할 때 논문에 따르면 epoch 100에 accuracy 80% 정도를 Resnet50이 달성했다고 한다. (더 얕은 Resnet18은 더 낮을 것이다.) 대부분의 세팅에서 모델이 수렴한 95~96% 정도면 일반적인 사람의 Accuracy 와 비슷하거나 능가하는 수준이다.

본 과제에서는 사실 모델 그 자체보다는 Hyper Parameter 들이 모델 성능에 미치는 영향과 그 배경이 되는 개념들을 실제로 실험해보았다. Pretrained 된 모델을 사용했지만, 코드 몇 줄로 이 정도의 Neural Network 를 구성할 수 있는 점이 인상 깊었고, 처음으로 PyTorch 코드를 제대로 공부해볼 수 있는 기회여서 좋았던 것 같다. 사용 가능한 리소스의 제한으로 몇몇 Trained Weights 들이 날라간 점은 아쉽지만, 이번에

연구참여에서 공부한 내용들 및 실제 논문을 구현해볼 수 있게 되어 미래 많은 도움이 될 것 같다.