기계학습 Assignment #3

2016-15788
정원석

[1] Better-Net Architecture

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 1. GoogLeNet incarnation of the Inception Architecture
("Going Deeper with Convolutions" 논문 발췌)

Inception-module을 사용한 Convolution Neural Network의 시초를 제시한 것이 바로 Going Deeper with Convolution 논문([1])의 GoogLeNet이다. GoogLeNet을 out-perform 하는 ResNet의 등장으로 인해 간단한 아이디어로 deep network를 구축하는 것이 핵심이라는 것을 최초로 제시한 의미 있는 논문이다. 이후에도 Google에서 inception module을 발전시킨 Inception v2, v3, v4 등의 모델을 제시하였으며 현재 image-classification 분야에서는 human-error보다 작은 error를 기록하는 등 놀라운 성과를 보여주고 있다.

그 중, 우리 과제에서 사용가능한 모델은 GoogLeNet으로 한정된다. Inception v2, v3, v4에서 사용한 inception block은 과제에서 제시된 inception block과 다른 형태를 띄고 있기 때문이다. 따라서 GoogLeNet을 구현하기로 결정했으며, Figure 1의 parameter dimension을 참조하여 네트워크를 구축하였다. 단, 원래의 auxilary classifier을 붙여서 학습을 진행하려면 def train(.) 코드를 수정해야 하는 소요가 생겨서, 제한 사항으로 인해 auxilary classifier를 네트워크에 stack하진 않았다. 하지만, 마찬가지로 GoogLeNet 논문을 참조하여 auxilary block class를 구현해보았으며, 그 구조는 Table 1에서 확인할 수 있다.

```
class AuxilaryBlock(nn.Module):
```

```
    def __init__(self, in_planes, num_classes):
        super(AuxilaryBlock, self).__init__()
        self.conv = nn.Conv2d(in_planes, 128, kernel_size=1)
        self.fc1 = nn.Linear(2048, 1024)
        self.fc2 = nn.Linear(1024, num_classes)
        self.dropout = nn.Dropout(p = 0.7)

    def forward(self, x):
        # N x 512 x 14 x 14
        # N x 528 x 14 x 14
        x = nn.AdaptiveAvgPool2d(x, (4, 4))
        x = self.conv(x)
        x = torch.flatten(x, 1)
        x = F.relu(x)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        # N x 10

        return x
```

Table 1. Auxilary block 구조

기본적인 network의 스켈레톤은 GoogLeNet을 모방하여 구현하였으며, 추가로 학습을 진행하는 데 있어 다음의 hyper-parameter tuning을 통해 BetterNet의 성능을 조절하였다. 조절 가능한 hyper-parameter의 종류는 다음과 같다.

Index	Hyper-parameter	
1	Batch-size	4, 8, 16, 32, 64, 128, 256
2	Filter-size, number of filters	GoogLeNet 구조를 차용
3	Network Architecture	GoogLeNet 구조를 차용
4	Pooling vs Strided Convolution	GoogLeNet 구조를 차용
5	Optimizers	SGD(0.9), Adam, RMSprop
6	Activation functions	GoogLeNet 구조를 차용
7	Regularizations	Not necessary
8	Model Ensembles	

Table 2. Hyper-parameter in our interest

본 과제에서 주어진 조절가능한 Hyper-parameter의 종류는 Table 1과 같다. 그 중, 2~4의 구현은 앞에서도 언급하였듯이 GoogLeNet을 구현한 논문인 “Going Deeper with Convolutions”을 참조하여 작성하였다. 다음과 같은 이유로 구현을 생략한 구조도 존재한다.

1) Local-Resp-Norm: 'ResNet' 논문에서 큰 효과가 없는 block임이 밝혀졌고, 그 이후에도 잘 쓰이지 않으므로 구현을 생략하였다.
2) Auxilary-structure: auxilary 구조를 구현하려고 하면, loss를 back-prop할 때 조금

특별한 treat이 필요하다. 하지만 본문의 training 코드를 건드리지 못하므로, auxilary 구조의 구현을 포기하였다. 다음은 Auxilary-block의 구현이다.
3) Regularization의 경우, epoch이 2밖에 되지 않으므로, overfit보다는 정확도인 underfit을 방지하는 데 초점을 맞춰야 한다. 따라서, regularization penalty-term은 따로 두지 않았다.

따라서, 우리가 조절하면서 실험할 hyper-parameter의 후보군은 1,5,8번 hyper-parameter로 좁혀진다. 따라서, 다음과 같은 hyper-parameter 후보군을 두고 grid-search를 진행하였다. Table 3은 grid-search를 통해서 얻은 test-set accuracy이다. optimizer index는 Table 2에서 위에서 아래로 차례대로 1부터 9이다.

Index	Hyper-parameter	Grids	
1	Batch-size	4, 8, 16, 32, 64, 128, 256	
5	Optimizers	Adam	lr=0.01, betas=(0.9, 0.999), eps=1e-08 lr=0.001, betas=(0.9, 0.999), eps=1e-08 lr=0.0001, betas=(0.9, 0.999), eps=1e-08
		SGD	lr=0.01, momentum=0.9 lr=0.001, momentum=0.9 lr=0.0001, momentum=0.9 lr=0.001, momentum=0.8 lr=0.001, momentum=0.7
		RMS-prop	lr=0.01, alpha=0.99, eps=1e-08
8	Model Ensembles	Dropout으로 비슷한 효과	

Table 3. Hyper-parameter in our interest

	1	2	3	4	5	6	7	8	9
4	62.21	66.42	72.94	66.47	70.14	58.6	69.26	68.28	53.33
8	65.21	66.3	76.86	70.09	74.65	58.68	72.69	69.94	60.31
16	x	68.51	78.36	74.57	74.73	55.3	71.06	67.61	67.25
32	x	71.4	78.51	76.39	72.55	47.76	66.36	63.55	66.88
64	x	73.16	75.32	78.14	68.28	39.78	61.8	57.43	58.85
128	x	73.33	71.26	76.96	62.24	31.22	54.31	46.69	56.92
256	x	73.68	66.54	74.38	52.05	25.98	44.02	37.65	46.94

Table 4. Accuracy Matrix(drop_rate = 0.4)

X를 test-set-accuracy를 가지는 random-variable이라고 생각하면 그 분포는 optimizer와 batch_size에 의해 정해진다고 생각할 수 있다. Table 4에서도 확인할 수 있듯이, 우리는 63개의 grid 중에 78% 정도의 가지는 3개의 후보군을 발견하였다. Optimizer의 종류마다 batch_size 증가에 따른 정확도의 경향성이 천차만별이었다. 학습속도는 batch_size가 커질수록 빨라지는 것을 확인하였다. Figure 4는 hyperparameter를 tuning하지 않은 원래 baseline GoogLeNet과 grid-search를 통해서 찾은 최적의 BetterNet의 loss 그래프를 보여주고 있다. 더 좋은 성능을 보여주는 것을 확인할 수 있다.
추가적으로 위의 hyper-parameter를 고정시켜가며 drop_ratio를 0부터 1까지 0.1 단위로 변화시켜가며 test-set accuracy를 측정해 본 결과 drop_ratio=0.2일 때가 가장 높게 측정

되었으며, 큰 효과는 보지 못하였다. Drop_out 기법이 ensemble의 효과를 가져다 줄 것이라 예상했지만 큰 효과를 주지 못하였다. 하지만, 더 많은 epoch에서 훈련하게 될 경우 regularization 등에서 효과를 드러낼 것이라고 생각한다. 하지만, 해당 보고서에서 언급하였듯이, 해당 network의 설계는 overfit보다는 underfit에 신경을 썼기 때문에, optimizer의 weight-decay 항등을 건드리지 않았다. 하지만, 더 많은 epoch에서 학습을 진행할 경우, overfit을 신경써야 할 것이다. 요약하자면, 현재의 설정으로, 78%의 정확도를 달성하였다.

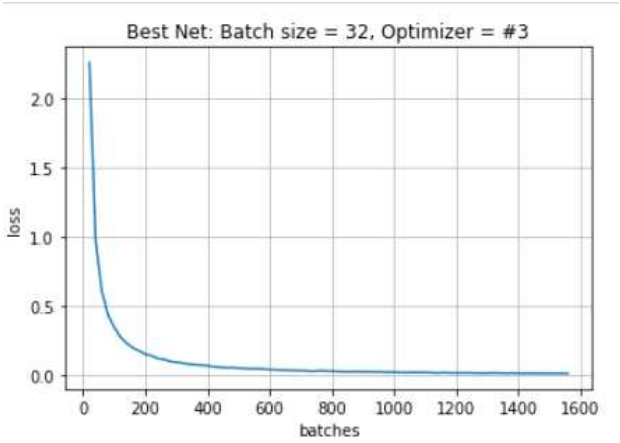

Figure 2. Best-Net Loss-curve

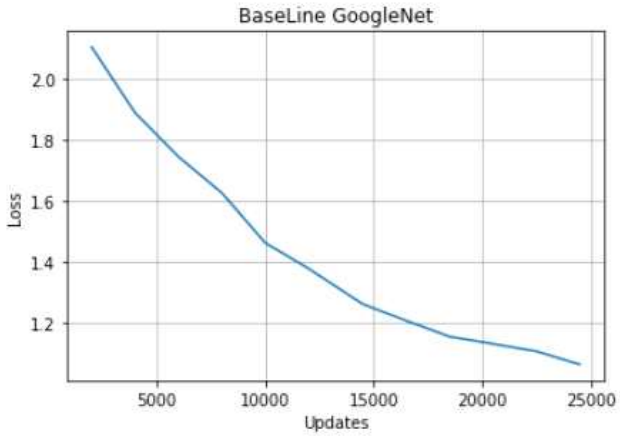

Figure 3. Baseline Loss-curve