



논문 스터디 3주차

이선민, 김영호

Densely connected convolutional networks

ADAM : A method for stochastic optimization

Index

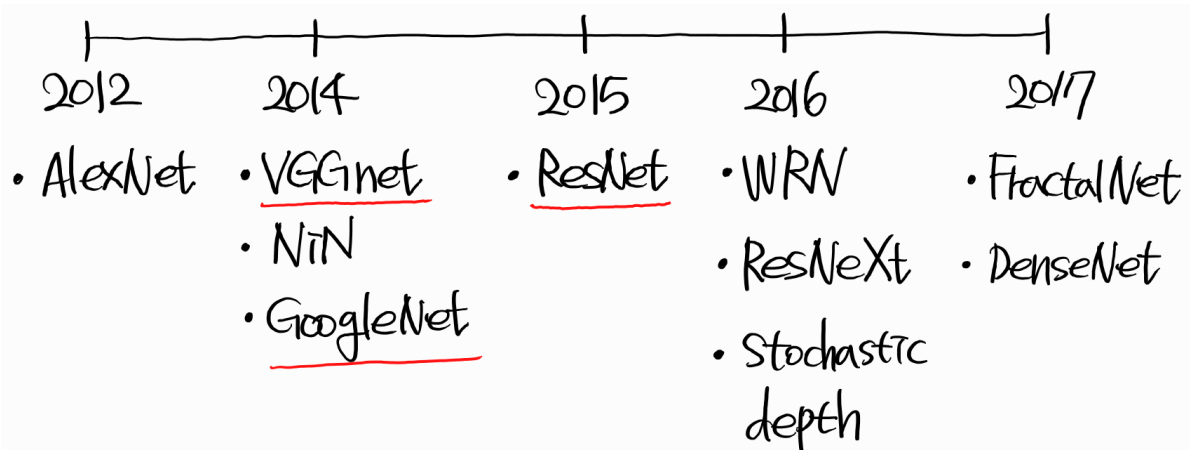
1. Densely connected convolutional networks
 - 1.1. ResNet, DenseNet, and FractalNet
 - 1.2. DenseNet 논문 정리
2. ADAM : A method for stochastic optimization
 - 2.1. Gradient Descent with Momentum
 - 2.2. 지수 가중 평균
 - 2.3. ADAM 논문 정리

1. Densely connected convolutional networks

얼렁뚱땅

1.1. ResNet, DenseNet, and FractalNet

본 논문의 초반부에는 다양한 CNN Architecture들이 등장한다. 최초의 CNN모델이라 할 수 있는 Lenet5에서 시작하여 inception module과 auxiliary classifier를 활용한 GoogLeNet, 기본적인 CNN 구조를 활용하여 layer를 깊이 쌓은 VGG, residual connection을 활용한 ResNet, 기존의 ResNet에서 layer에 dropout을 적용한 Stochastic Depth, , shallow pathway와 deep pathway를 동시에 활용하고 dropout을 적용 시켜 일부 경로만 활용하여 학습을 진행하는 FractalNet 등이 등장한다. 그러나 본문에서는 **ResNet과 DenseNet과의 비교**를 중점적으로 다루고 있기 때문에 이 두 architecture에 대해 집중적으로 다룰 것이며, 추가적으로 **FractalNet**에 대해 간단하게 살펴볼 것이다. CNN 기본 구조와 관련된 내용을 여러분들이 어느 정도 알고 있다고 가정하고, architecture의 구조적인 관점 위주로 전개해보겠다.



- ResNet의 기본 구조

1. 하나의 residual block은 두 개의 3*3 conv layer로 이루어져 있다.

3. 마지막 Convolution 이후 pooling 시에는 global average pooling을 사용한다. **global average pooling**은 하나의 activation map 전체를 average pooling하는 것을 의미한다. 학자들은 global average pooling을 활용한 이유로 residual block을 통해 효과적으로 feature-vector를 뽑아냈기 때문에 average pooling만 적용해도 충분하다고 주장한다. average pooling으로도 분류가 가능하고 overfitting 문제를 회피할 수 있으며 연산량이 대폭 줄어든다는 이점도 얻을 수 있다.

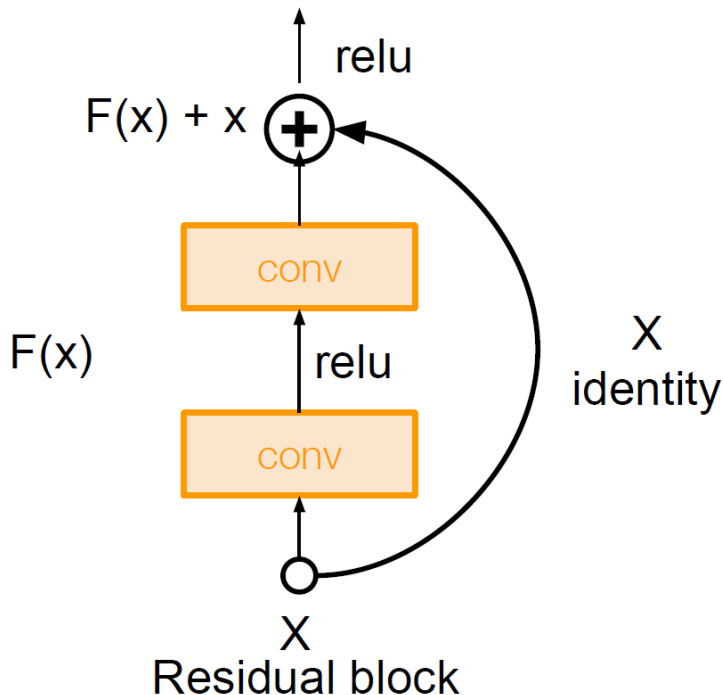
3. 맨 끝 부분에는 최종적으로 1000개의 class를 분류하는 FC layer만 넣어준다.



- residual connection

먼저 ResNet의 핵심인 residual connection이 등장한 배경부터 살펴보겠다. 학자들이 처음 호기심을 가졌던 부분은 CNN을 더 깊게 쌓게 되면 어떤 일이 발생하는가에 관한 것이었다. 20개의 layer를 쌓은 경우와 56개의 layer를 쌓은 결과를 비교해보았는데, 놀랍게도 56개를 쌓은 경우의 train error와 test error가 20개를 쌓은 경우보다 더 높은 것으로 나타나 **층을 깊게 쌓는다고 해서 높은 성능을 보이지 않는다는 점을 발견**하였다. 여기서 그들은 더 깊어질수록 optimizaition이 어렵다는 가설을 생각해냈고, 층을 많이 쌓은 모델(deep model)이 적어도 층을 적게 쌓은 모델(shallow model) 만큼의 성능이 나와야 한다고 추론했다. 그들은 먼저 **shallow model의 가중치 W 를 deep model의 일부 층에 복사한 후, 나머지 층은 identity mapping을 진행하면 되겠다는 아이디어**를 떠올려낸다. 이렇게 구성하면 shallow model의 성능 만큼은 보장될 것이기 때문이다. 이러한 점들을 종합하여 그들은 residual mapping이 가능하도록 모델을 디자인하게 되고, 이로부터 등장한 것이 **residual connection**이다.

residual connection을 그림으로 나타내면 아래와 같다. 입력 x 는 이전 layer에서 흘러 들어온 입력이다. 그리고 layer가 직접 $H(x)$ 를 학습하기보다 $H(x)-x$ 를 학습할 수 있도록 만들어준다. 먼저 x 를 가중치 W 를 고려하지 않고 identity mapping으로 그대로 출력쪽으로 전달해주는데, 이를 **skip connection 또는 shortcut**이라 한다. 그러면 실제 layer는 x 에 대한 residual에 해당하는 변화량 $F(x)$ 만 학습하면 되고, 최종 출력값은 $x + \text{residual}$ 이 되는데, 이 값은 x 에 가까운 값이다. ResNet을 제안한 학자들은 이 방법을 사용하면 학습이 더 쉬워지게 된다고 주장하는데, $F(x)$ 가 0일수록 좋은 것이므로 모든 W 를 0으로 만들어주면 그만이기 때문이다. Full mapping을 이용하여 학습하는 것보다 **입력값과 얼마나 차이가 나는가를 학습하는 것이 더 쉬울 것**이라고 생각한 것이다. 추가적으로 입력에 residual을 더하는 방식으로 표현되기 때문에 feature를 **summation**한다고 표현한다. 본 논문에서 summed된다는 표현이 등장하기 때문에 짚고 넘어간다.

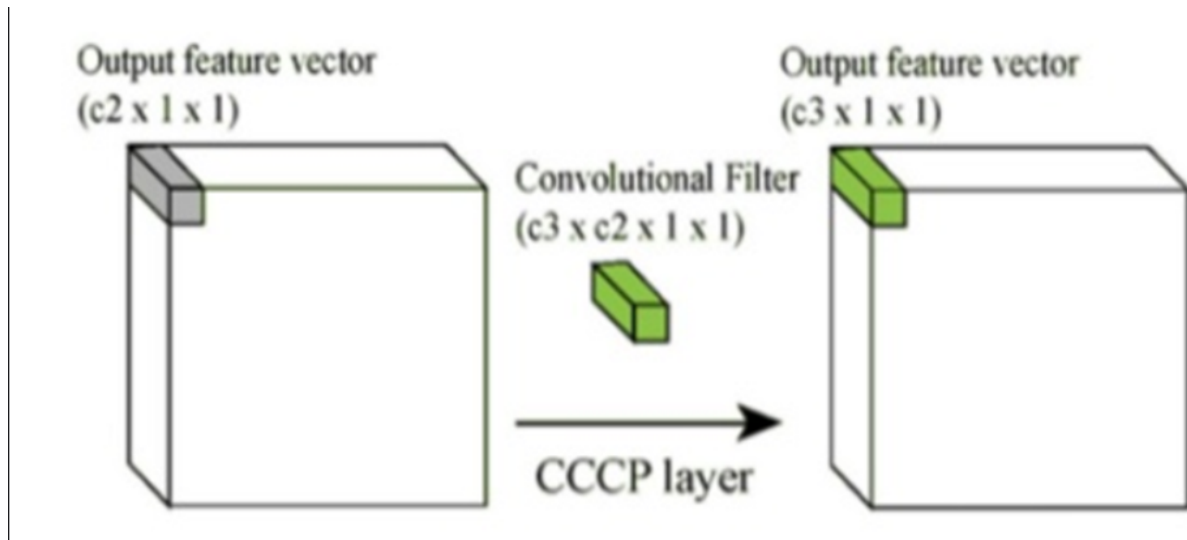


Use layers to fit residual
 $F(x) = H(x) - x$
 instead of
 $H(x)$ directly

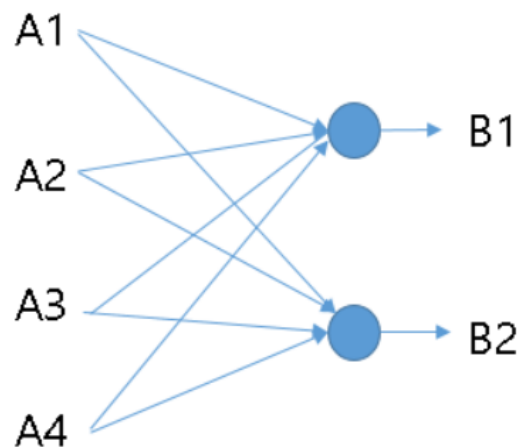
- Bottleneck layer (왜 1*1 convolution을 사용하는가?)

50개 이상의 layer 쌓아 깊게 학습을 진행하는 경우 ResNet에서는 efficiency를 높이기 위해 bottleneck layer를 추가한다. bottleneck layer는 쉽게 말해 1*1 convloution을 사용해 차원을 축소시키기 위함이다. 1*1 convloution을 하게 되면 여러 개의 feature map으로부터 비슷한 성질을 갖는 것들을 묶어낼 수 있고, feature map의 숫자를 줄일 수 있으며, 결과적으로 연산량을 줄일 수 있게 되기 때문에 유용하게 사용된다. ResNet 이전에 등장한 GoogLeNet에서도 이를 이용하고 있으니 추가적으로 공부하고 싶은 사람이 있다면 참고하자.

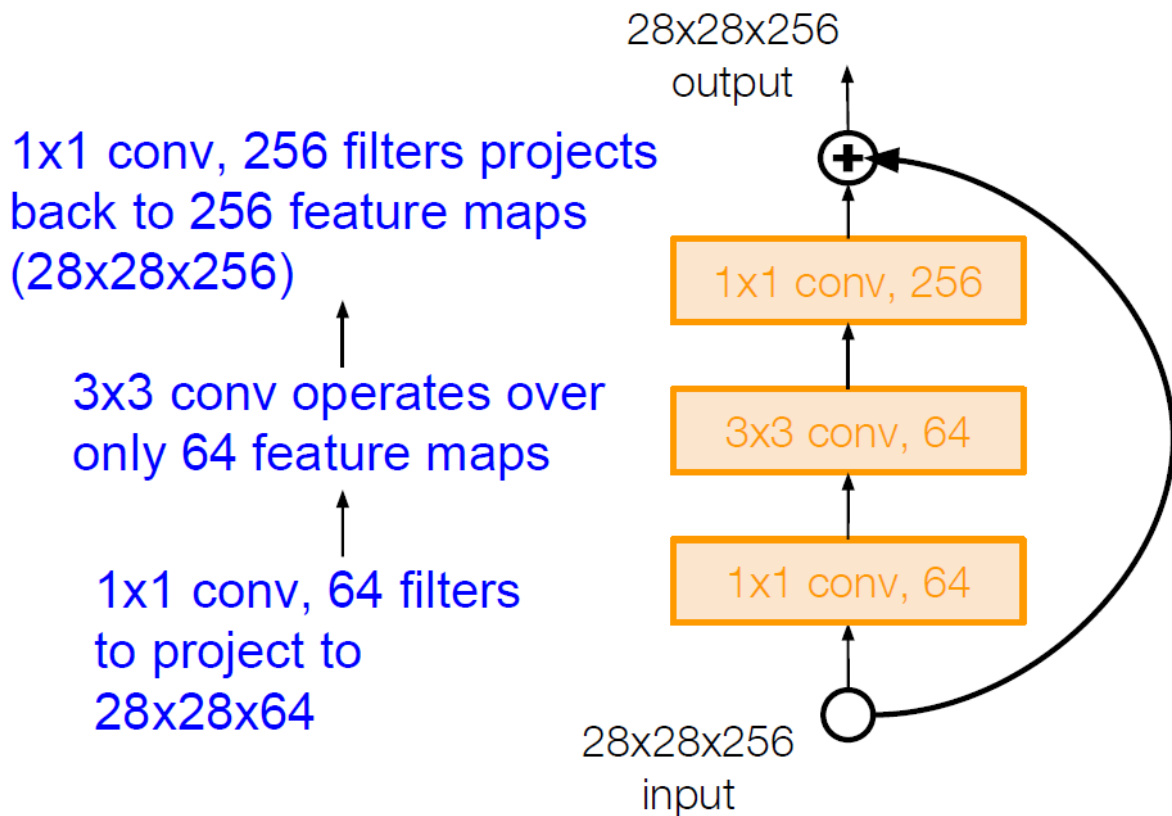
이제 1*1 convolution의 연산이 어떠한 방식으로 이루어져 차원 축소의 효과를 가져오는지 살펴보자. **1*1 convolution은 쉽게 말해 현재 feature map의 개수에 더 적은 feature map의 개수만큼 fully connected 시켜 feature map의 특정 위치에 정보를 담아내는 것이다.** 이렇게 글로 표현하면 무슨 말인지 쉽게 와 닿지 않기 때문에 아래의 그림을 통해 설명하기로 한다.



위에서 설명한 것을 단순화하여 만약 입력 feature map의 개수 $c2$ 가 4이고, 출력 feature map의 개수 $c3$ 가 2라면, 1×1 convolution은 아래와 같이 fully-connected 형태로 표현 가능하며, 결과적으로 4개의 feature map이 2개의 feature map으로 결정되어 차원이 축소되고 연산량이 줄어들게 된다.

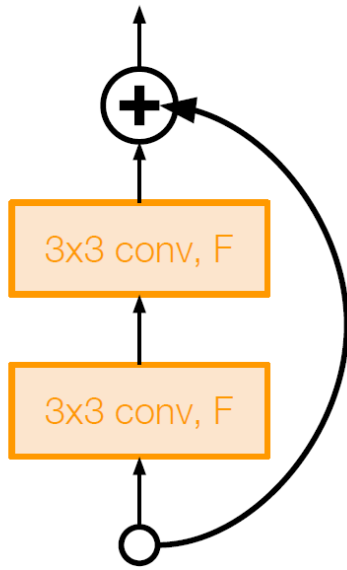


ResNet에서는 다음과 같은 형태로 Bottleneck layer를 추가하고 있다.

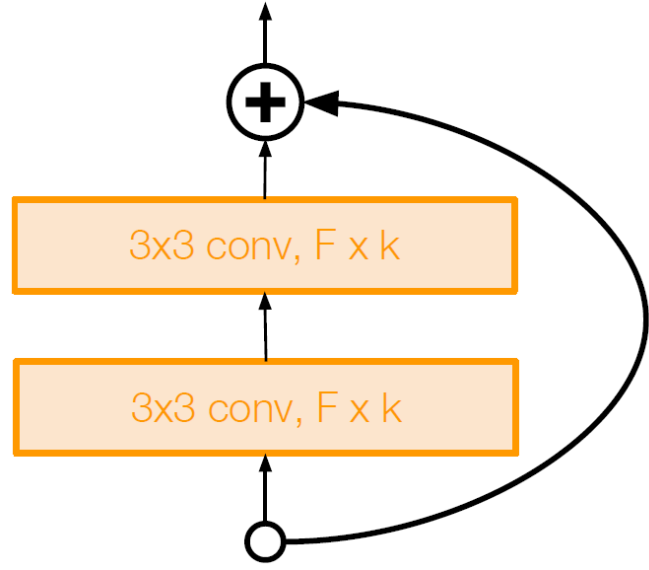


- Original ResNet의 변주 등장!(WRN; Wide ResNet)

기존의 ResNet은 layer를 깊게 쌓으면서 feature를 재사용함으로써 학습이 느려진다는 문제점을 갖고 있다고 보고 중요한 것은 depth가 아닌 residual 그 자체 주장하면서 등장했다. residual connection이 있으면 네트워크가 더 깊어질 필요는 없다고 주장했다. 그래서 **residual block의 width를 늘렸다**. 이는 **residual block 내에서 더 깊은 학습이 이루어지도록 했다고** 이해하면 된다. 기존에는 F 개의 필터를 사용하였지만, WRN에서는 k 배 만큼을 더 곱한 $F \times k$ 개의 필터를 사용한다. 학자들은 적절한 k 값을 찾아 $F \times k$ 개의 필터를 사용한 결과, 50개 층만 있어도 152개의 층을 사용한 기존의 ResNet보다 성능이 더 좋음을 입증했다. 이는 **residual block 자체가 중요하며, depth의 효과는 부수적인 것임을** 의미한다. 기존의 ResNet과 WRN의 residual block을 비교하여 나타낸 그림은 아래와 같다.

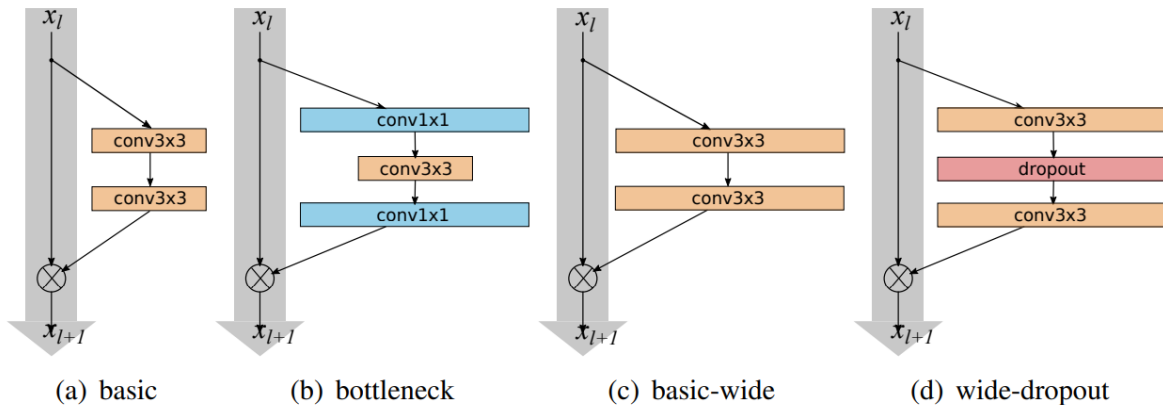


Basic residual block



Wide residual block

추가적으로 WRN에서 활용한 다양한 유형의 residual block은 다음과 같다.



(2) DenseNet

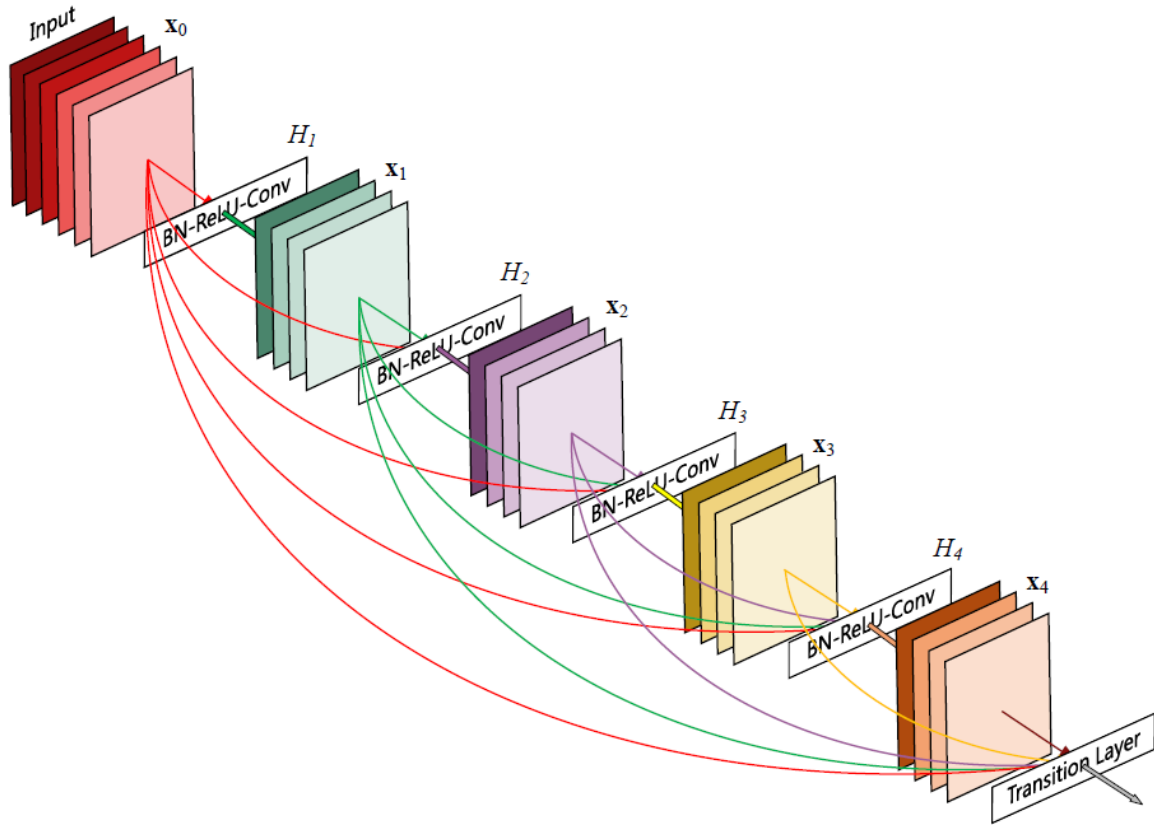
• Original DenseNet

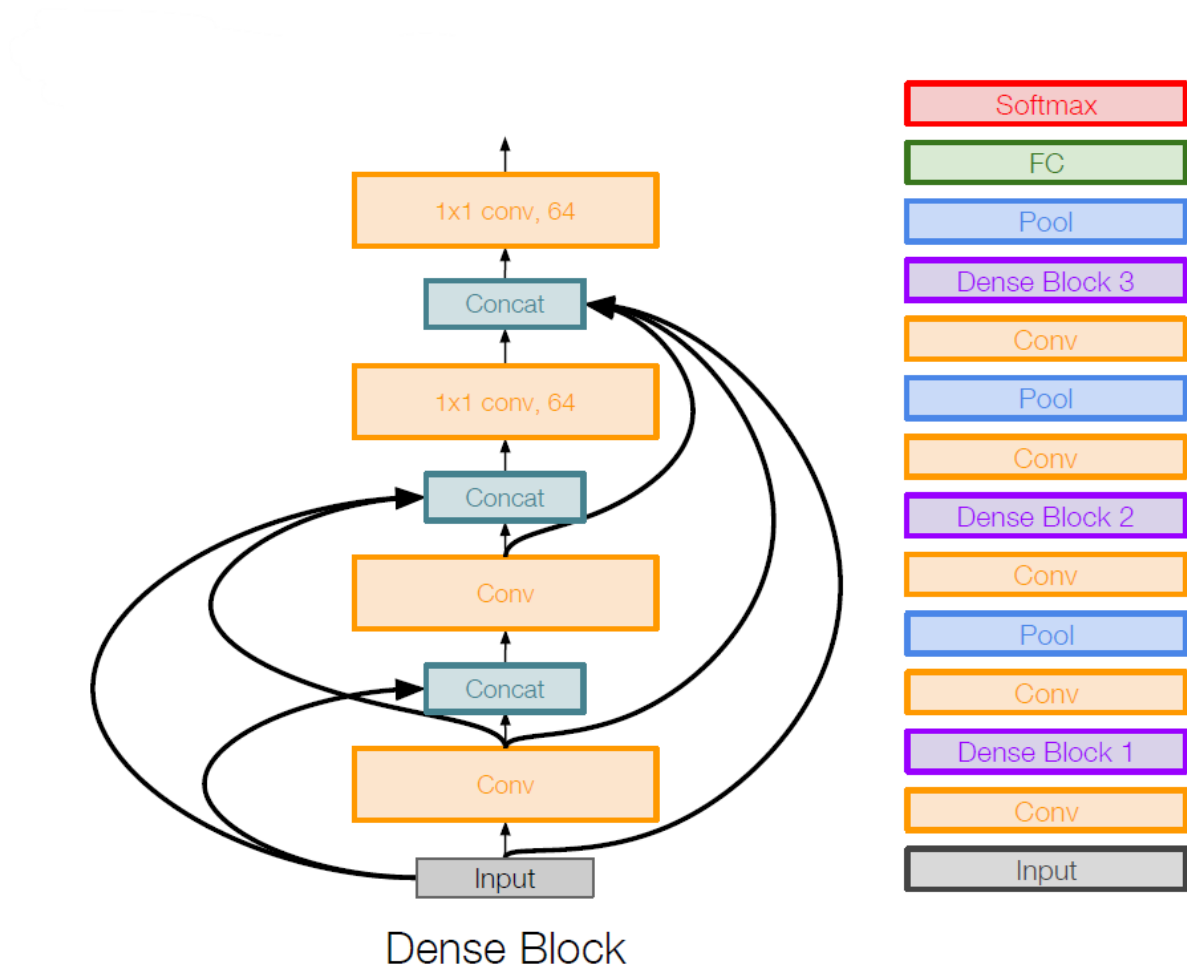
2017년 발표된 DenseNet은 이전 layer의 feature map을 그 이후의 모든 layer의 feature map에 연결한다. ResNet에서는 입력에 residual이 더해지는 형태였기 때문에 summation한다고 표현했지만, DenseNet에서는 feature map을 계속해서 연결해 나간다는 의미에서 concatenation한다고 표현한다. 이 또한 논문에서 concatenated 된다고 표현하기 때문에 짚고 넘어간다. dense connection은 각 layer의 출력을 다른 layer에도 여러 번 사용하기 때문에 feature를 더 잘 전달할 수 있도록 하는 역할을 한다. 이를 수식으로 나타내면 다음과 같다.

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$$

기존의 CNN 모델은 처음 layer의 feature map이 다음 layer의 입력값으로 전달되어 많은 layer를 통과하게 되면 처음 layer의 feature map에 대한 정보는 사라질 수 있다. 학자들은 DenseNet이 처음 layer의 feature map을 마지막 layer의 feature map까지 연결하기 때문에 gradient가 소실되는 문제를 완화시킬 수 있다고 주장한다. 또한 각 layer의 feature map을 연결하여 다음

layer로 전달하기 위해 적은 채널 수의 feature map을 생성하고, 이는 이전 layer의 feature map과 결합하여 다음 layer로 전달되어야 하므로 **파라미터 수가 적다**. 아래의 두 그림은 DenseNet의 구조를 도식화한 것이다.





추가적으로 본 논문에 등장하는 정의와 공식을 정리해보면 다음과 같다.

$$\begin{aligned}
 \# \text{ total layers} &: L \\
 \# \text{ total direct connections} &: \frac{L(L+1)}{2} \\
 \# \text{ channels in input layer} &: k_0 \\
 \# \text{ channels of dense block} &: k \\
 \# \text{ feature maps of } l^{\text{th}} \text{ layer} &: k_0 + k \times (l-1)
 \end{aligned}$$

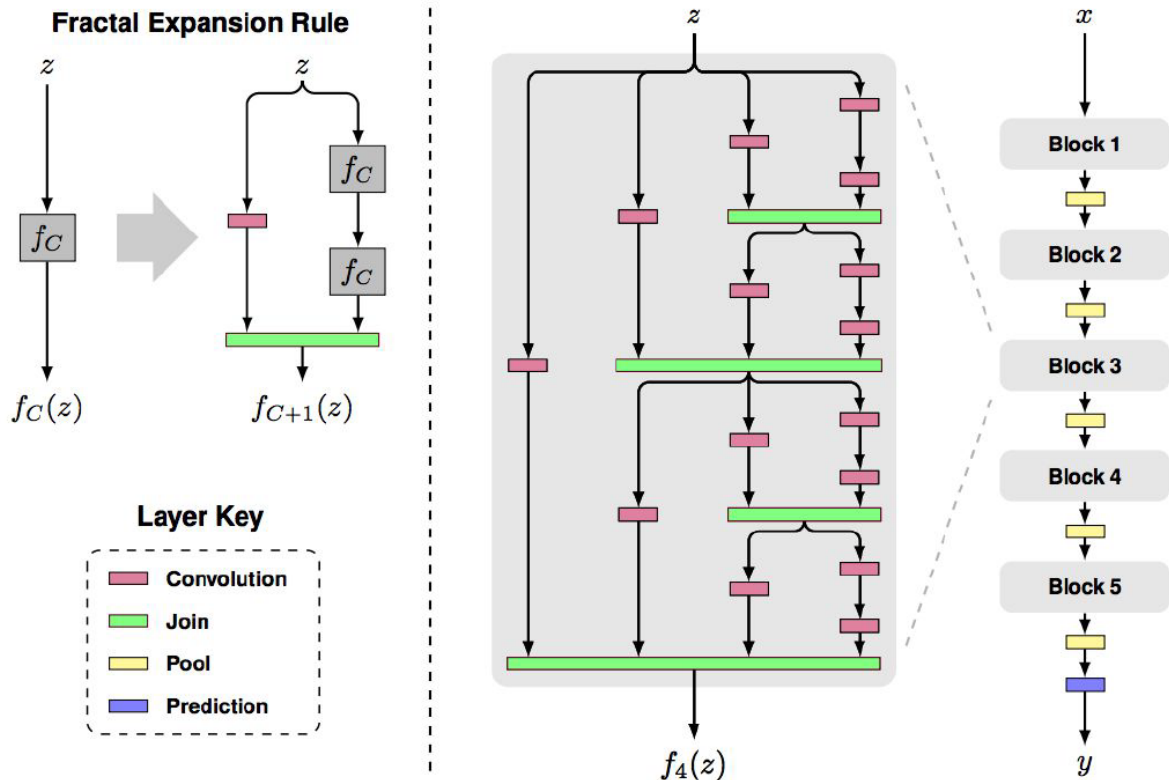
여기서 두 가지만 더 짚고 넘어가자면 논문에서는 dense block 내의 채널의 수 k 를 **growth rate**로 정의하고 있는데, 이는 각 layer가 전체에 어느 정도 기여할지 결정한다.

• Original DenseNet의 변주 등장! (DenseNet-B, DenseNet-C, DenseNet-BC)

Original DenseNet model에 bottleneck layer를 추가하는 경우 **DenseNet-B**로 정의한다. 여기서 논문에서는 bottleneck layer의 채널 수로 $4k$ 개를 사용했다고 한다. 또한 Compression과 model compactness에 대해 설명하면서 dense block내의 채널 수를 조절하는 파라미터 θ 를 **compression factor**로 명명한다. θ 는 0과 1 사이의 값을 갖는다. θ 를 가지고 기존 dense block 보다 더 적은 채널 수를 사용하기로 한 것이다. 그리고 bottleneck layer를 사용하지 않고 $\theta = 0.5$ 를 주는 DenseNet model을 **DenseNet-C**, bottleneck layer를 사용하고 $\theta = 0.5$ 를 주는 DenseNet model을 **DenseNet-BC**를 정의한다.

(3) FractalNet

FractalNet은 residual을 이용한 학습은 불필요하다고 주장하며 ResNet의 반기를 들고 등장하였다. 대신 shallow/deep network의 정보를 모두 잘 전달하는 것이 중요하다고 생각하여 **shallow/deep path를 출력에 모두 연결하는 방식**으로 디자인하였다. train 시에는 drop path를 이용하여 일부 path만으로 학습을 진행하며, test 시에는 모든 path를 사용한다. 본 논문에서 FractalNet이 나오기 때문에 이런 모델이구나 정도만 이해하고 넘어가자. FractalNet의 구조를 그림으로 나타내보면 다음과 같다.



1.2. DenseNet 논문 정리

• Data description

Dataset으로는 CIFAR, SVHN, Imagenet을 활용하였다. 먼저 **CIFAR**의 경우 32×32 컬러 이미지를 10개의 class로 분류하는 CIFAR-10(C10)과 100개의 class로 분류하는 CIFAR-100(C100)을 활용하였다(검색해보면 알겠지만 C10의 경우 airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck 10개의 범주로 분류!). 여기서 한 가지 notation을 사용하는데, **data augmentation이 진행된 경우와 구분하기 위해 끝에 +기호를 표시하여 각각 C10+, C100+로 표기한다**. 추후 result 분석 시 사용될 notation이므로 숙지하기 바란다. **SVHN(The Street View House Numbers)**의 경우 32×32 컬러 숫자 이미지 dataset으로 data augmentation은 적용하지 않는다. 1000개의 class로 분류하는 **Imagenet** 역시 data augmentation을 적용하며, crop을 통해 이미지의 사이즈를 224×224 로 조정하였다. 또한 CIFAR과 SVHN의 경우 pixel값 범위 조정을 위해 normalize를 해주거나 255로 나눠주었다. 아래의 블로그를 참조하면 CIFAR, SVHN, Imagenet을 비롯한 다양한 image dataset과 관련된 설명을 확인할 수 있다.

Image Dataset(1)

Computer Vision에서 주로 다루게 되는 문제들은 크게 Classification/Semantic Segmentation/Object Detection/Instance Segmentation과 같이 4가지로 분류할 수 있다. 이 중, 이번 포스트에서는 Classification(Image Recognition) 문제에서 주로 사용되는 데이터셋들을 위주로 살펴보도록 하겠다.
https://kjhov195.github.io/2020-02-09-image_dataset_1/



• Training

기본적으로 mini-batch 단위로 gradient를 구하고 가중치 W 를 update시키는 **SGD(Stochastic gradient descent ; 확률적 경사하강법)**를 활용하여 학습을 진행한다. **CIFAR와 SVHN**의 경우 batch size는 64, epoch 수는 각각 300과 40으로 하였고, 초기 learning rate로는 0.1을 사용하였는데, epoch 수의 50%와 75% 구간에서 절반으로 나눠 이를 조정하였다. **ImageNet**의 경우 batch size는 256, epoch 수는 90으로 하였고, 앞선 경우와 마찬가지로 초기 learning rate로는 0.1을 사용하였으며, epoch수가 30일 때와 60일 때 절반으로 나눠 조정해주었다. 또한 가중치 W 를 update 시키기 위해 weight decay 값으로는 10^{-4} 를 사용하였으며, **Nesterov momentum**을 이용하였다. momentum의 decay rate 값으로는 0.9를 사용하였다. 일반적으로 0.9 혹은 0.99를 사용한다고 알려져 있다. 추가적으로 **data augmentation을 진행하지 않은 C10, C100, SVHN**의 경우 dropout rate를 0.2로 주어 layer dropout을 진행한다. layer dropout 개념은 dropout의 개념을 layer에 적용했다고 생각하면 된다. 즉, 학습 시 일부 node들을 random하게 건너뛰고 학습을 진행하여 overfitting이 발생하지 않도록 하는 것과 마찬가지로 이를 layer에 적용한 것이다.

이제 이를 바탕으로 모델을 적합 시킨 결과를 표와 그래프로 설명된 자료 위주로 살펴보기로 한다. 본 논문에서는 앞서 살펴본 모델인 ResNet, DenseNet, DenseNet-B, DenseNet-C, DenseNet-BC를 모델링하였다. 특히 앞서 살펴본 **layer의 depth를 나타내는 L 과 growth rate에 해당하는 K** 에 따라 어떻게 달라지는지 주목할 필요가 있다는 점을 기억하자.

• Results

(1) Classification Results of CIFAR and SVHN

아래의 table은 CIFAR과 SVHN dataset에 따른 test error rate(%)를 나타낸 것이다. 논문에 나와있는 Table2에서 필요한 부분만을 가져왔다.

CIFAR dataset에 data augmentation을 해준 **C10+와 C100+**에 DenseNet-BC ($L = 190, k = 40$)를 적용할 경우 Wide ResNet보다 좋은 성능을 보였다. CIFAR dataset에 data augmentation을 해주지 않은 **C10과 C100**에서는 DenseNet ($k = 12$)부터 DenseNet-BC ($k = 24$)에 이르기까지 적용한 모든 경우에서 FractalNet에 Dropout/Drop-path를 적용했을 때보다 좋은 성능을 나타냈다. SVHN dataset에서는 DenseNet ($L = 100, k = 24$)를 적용할 경우 dropout을 적용한 Wide ResNet보다도 좋은 성능을 보이며 가장 좋은 성능을 보여주었다.

original DenseNet을 적용한 경우 L 과 k 가 증가함에 따라 **general trend**를 보이는 것으로 나타났다. C10+와 C100+에서 가장 잘 보여주고 있는데, 이들에 DenseNet을 적용한 경우 L 과 k 를 증가시키에 따라 파라미터 수가 증가하고 error rate가 줄어드는 경향을 보이고 있다. 이는 DenseNet이 크고 깊어질수록 이미지의 feature를 학습하는 능력이 향상되고 있음을 보여준다.

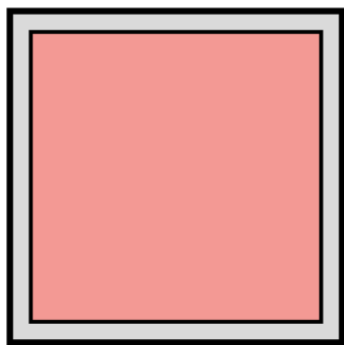
DenseNet은 FractalNet과 Wide ResNet에 비해 **parameter efficiency**가 우수한 것으로 나타났다. FractalNet의 경우 21개의 층을 쌓았음에도 불구하고 38.6M개의 파라미터가 필요하고, Wide ResNet의 경우 28개의 층을 쌓았음에도 불구하고 36.5M개의 파라미터가 필요하다. 이에 반해, 250개의 층을 쌓은 DenseNet-BC의 경우 15.3M개의 파라미터만 필요한 것을 확인할 수 있다. 파라미터가 많을수록 많은 연산을 필요로 하기 때문에 feature를 추출해냄에 있어 계산하는 시간이 짧을수록 효율적이라고 볼 수 있다.

다만, C10에 original DenseNet을 적용했을 때 error rate를 살펴보면 $k = 24$ 일 때 소폭 증가함을 확인할 수 있는데, 이 부분에서 potential overfitting이 일어나고 있다. 그러나 이는 DenseNet-BC에서 Bottle neck과 compression layer를 활용함으로써 해결됨을 확인할 수 있다.

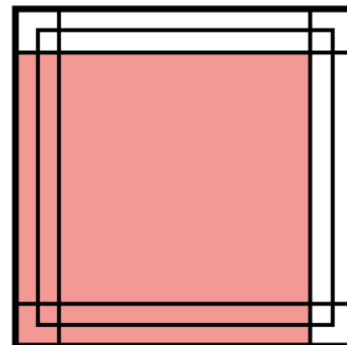
Method	Depth	Params	C10	C10+	C100	C100+	SVHN
FractalNet [17] with Dropout/Drop-path	21	38.6M	10.18	5.22	35.34	23.30	2.01
	21	38.6M	7.33	4.60	28.20	23.73	1.87
Wide ResNet [42] with Dropout	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
	16	2.7M	-	-	-	-	1.64
DenseNet ($k = 12$)	40	1.0M	7.00	5.24	27.55	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	5.77	4.10	23.79	20.20	1.67
DenseNet ($k = 24$)	100	27.2M	5.83	3.74	23.42	19.25	1.59
DenseNet-BC ($k = 12$)	100	0.8M	5.92	4.51	24.15	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	5.19	3.62	19.64	17.60	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	3.46	-	17.18	-

(2) Classification Results of Imagenet

CIFAR과 SVHN dataset과는 달리 Imagenet의 경우 validation set의 error를 이용한다. 이 때 error를 validation set에 augmentation을 적용할 때 single-crop을 적용한 경우와 10-crop을 적용한 경우로 나누고 있다. 여기서 single-crop과 10-crop이 무엇인지 잠깐 설명하자면, 먼저 single-crop은 이미지의 중앙을 중심으로 크기에 맞게 잘라 이미지를 보게 된다. 10-crop은 image를 총 10번 잘라서 보는 것인데, 큰 모서리 4번, 작은 모서리 4번, 작은 중앙 1번, 큰 중앙 1번 잘라서 이미지를 보게 된다. 그림으로 나타내면 다음과 같다.



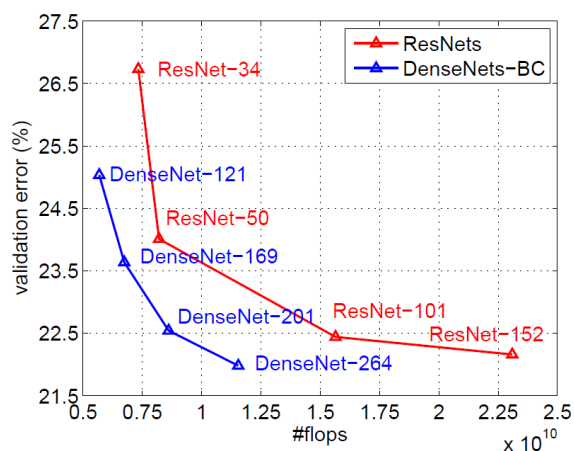
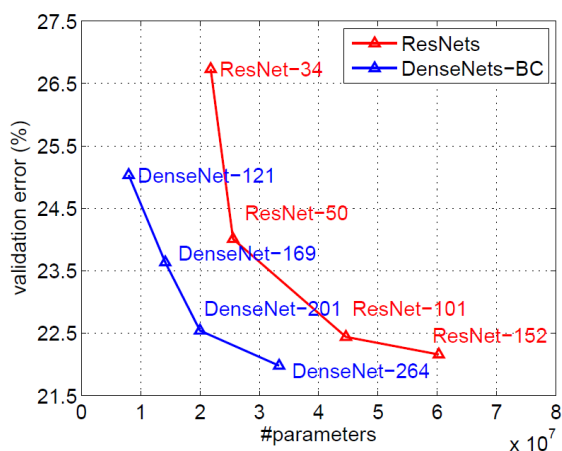
(a) Single Center Crop



(b) Part of a 10-Crop

이러한 개념을 바탕으로 본 논문의 Figure3에서는 validation set에 single-crop 수행 시 DenseNet-BC와 ResNet을 적용했을 때 파라미터 수와 1초 동안 몇 번의 연산을 수행하는지 나타내는 FLOPS(FLoating point Operations Per Second)에 따른 top-1 validation error의 변화를 그래프로 보여주고 있다. 여기서 **top1-validation error**란 가장 높은 확률로 정답의 class를 맞췄을 때 나타나는 error를 의미한다. 추가적으로 논문에 해당 내용이 등장하는 부분을 보면 state-of-art ResNet이라는 용어와 함께 사용하고 있는데, **state-of-art**란 가장 뛰어난 성능을 보이는 정도의 의미로 생각하면 될 것 같다.

이제 본격적으로 Figure3에 나타난 결과를 살펴보기로 한다. 왼쪽의 parameter에 따른 validation error를 살펴보면 동일한 validation error를 보이는 상황에서 DenseNet-BC가 ResNet보다 훨씬 더 적은 파라미터를 사용하고 있음을 알 수 있다. 파라미터 수가 적기 때문에 연산을 효율적으로 수행한다는 점에서 오른쪽의 FLOPs에 따른 validation error를 살펴보더라도 비슷한 상황을 보이고 있음을 확인할 수 있다. 따라서 앞서 살펴본 Table2와 마찬가지로 **DenseNet은 parameter efficiency 측면에서 상당히 우수한 면을 보인다.**



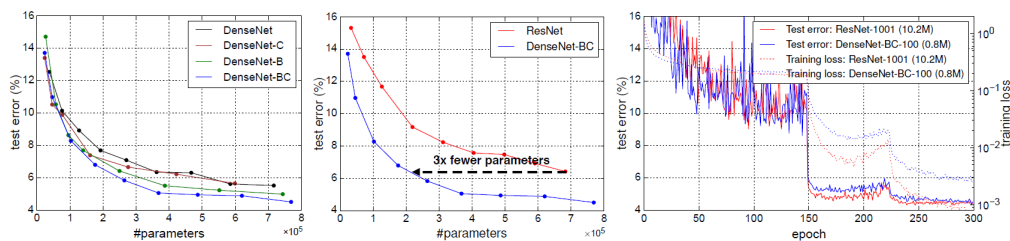
• Discussion

저자는 앞서 3개의 dataset의 classification 결과에서 살펴본 것처럼 DenseNet이 여러모로 우수한 성능을 보이고 있다는 점을 다시 한 번 강조하고 있다. 중요하다고 생각되는 몇몇 포인트들만 살펴보기로 한다.

먼저 **model compactness**에 대해 이야기를 하고 있는데, 이는 앞서 설명했던 것처럼 ResNet의 경우 이전 결과와 현재 결과를 summation으로 이어주는 구조였다면, DenseNet의 경우에는 모든 결과를 concatenation으로 싸그리 concatenation으로 연결시키는 구조이기 때문이다. 이렇게 함으로써 feature reuse가 가능하고 결과적으로 이러한 점들 때문에 compact한 model이라고 할 수 있는 것이다. 맥락 상으로 concatenation, feature reuse, compact 모두 비슷한 느낌이라고 가져가면 될 것 같다.

본 논문의 **Figure4**에는 아래의 3개 그래프가 등장한다. **왼쪽 그래프**는 original DenseNet과 DenseNet에 변수를 준 모델들의 parameter efficiency를 나타내고 있고, **중간 그래프**는 DenseNet-BC와 ResNet의 parameter efficiency를 나타내고 있다.

parameter efficiency는 파라미터 수에 따라 test error가 어느 정도 되는지를 나타내는 지표로, 동일한 파라미터를 사용함에도 불구하고 test error가 낮다면 parameter efficiency가 높다고 볼 수 있다. 또한 여기서 저자는 왜 ResNet 모델과 비교하였는지 잠시 언급하고 있는데, ResNet이 이에 앞서 등장했던 모델인 AlexNet과 VGG-net에 비해 더 적은 파라미터를 이용하여 좋은 성능을 보였기 때문이라고 설명한다. 이는 실제로 AlexNet, VGG-net, ResNet의 구조를 살펴보고 파라미터를 계산해보면 알 수 있는데, 이 부분은 더 깊게 공부해보고 싶은 사람이 있다면 추가적으로 학습해보기 바란다. **오른쪽 그래프**는 ResNet이 100개의 층을 쌓은 DenseNet-BC만큼의 성능을 보이기 위해서는 1001개의 층이 필요하고, 파라미터 수를 살펴보면 DenseNet에 비해 ResNet이 엄청난 양의 연산을 수행해야 하기 때문에 비효율적임을 보여준다.

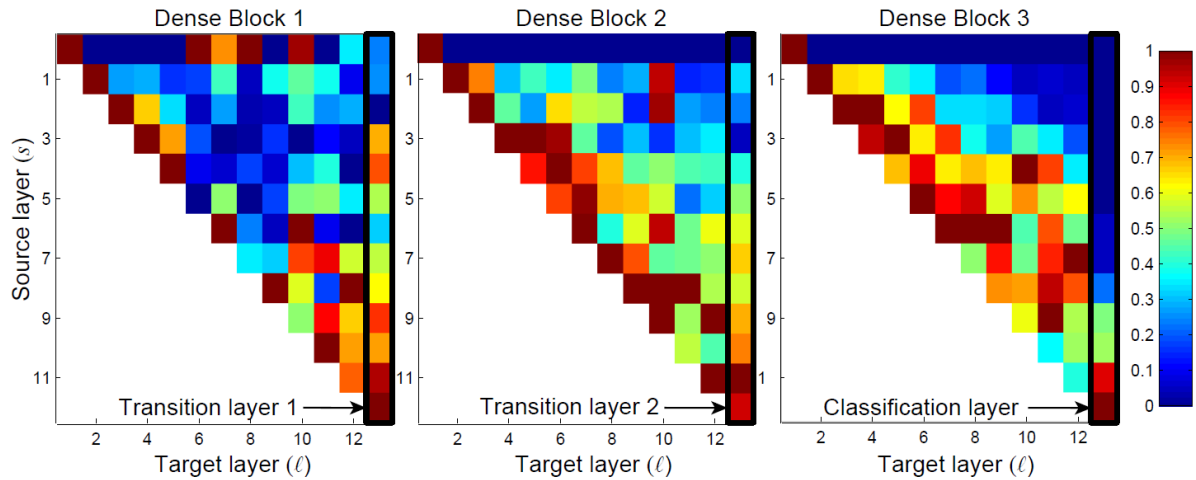


다음으로 저자는 DenseNet이 **deep supervision**을 수행한다고 주장한다. layer가 점점 쌓이면서 shorter connection을 통한 loss fuction으로부터 additional supervision을 학습하기 때문에 성능이 향상되었다고 보고 있다. 이는 DenseNet이 concatenated되는 구조이기 때문에 나타나는 것과 관련 있는 것이라고 생각할 수 있는데, 모든 layer가 loss function과 직접 맞닿아 있어 loss function과 gradient가 덜 복잡한 구조를 갖고 있기 때문이다.

마지막으로 **feature reuse**의 측면을 제시하고 있다. 이를 알아보기 위해 본 논문은 Figure5에서 heatmap 시각화를 보여주고 있다. C10+에 DenseNet ($L = 40, k = 12$)를 적용하여 train했을 때 현재 layer l 에 대해 그 이전까지에 해당하는 각각의 layer s 에 할당된 average absolute weight(weighted의 average L1 norm)를 계산한 후 각 block의 heatmap을 시각화한 것이다. 즉, **선행 layer들과 현재 layer와의 dependency의 정도를 color를 다르게 줌으로써 시각화한 것으로 이해하면 된다. color가 빨간색에 가까울수록 현재 layer l 이 선행 layer 중 하나인 s 의 feature를 적극적으로 활용하고 있는 것이고, 파란색에 가까울 경우에는 그 반대이다.** 이 때 시각화할 block의 개수는 3개로 한정하였으며, 이를 통해 DenseNet의 feature reuse 측면을 살펴볼 수 있다.

heatmap을 통해 알 수 있는 특징은 다음의 3가지로 정리해볼 수 있다.

1. 모든 layer는 동일한 block 내에 많은 input의 weight를 분산시킨다.
2. transition layer가 중복된 feature의 경우에는 평균 weight가 낮은 feature를 주로 출력하며, 이는 출력이 compression되는 DenseNet-BC의 결과와 정확히 일치함을 알 수 있다. 여기서 **transition layer**란 dense block 사이에서 convolution과 pooling을 수행하는 layer를 뜻한다.
3. 가장 오른쪽에 표시된 classification layer도 전체 dense block의 weight를 사용하기는 하지만, 최종 feature map에 집중함을 확인할 수 있다.



• Conclusion

결과적으로 파라미터의 수를 증가 시킴에 따라 DenseNet이 degradation이나 overfitting 문제 없이 꾸준히 성능을 끌어올렸고, 적은 수의 파라미터 및 연산량으로도 다른 모델들보다 좋은 성능을 보였기 때문에 DenseNet이 좋은 모델임을 강조한다. 다만, detail한 조건을 주는 부분에 있어 자세한 설명이 빈약하다는 생각이 든다. 가령, 왜 bottleneck layer의 개수를 $4k$ 로 했는지, momentum으로 비주류인 Nesterov momentum을 사용했는지가 그러하다.

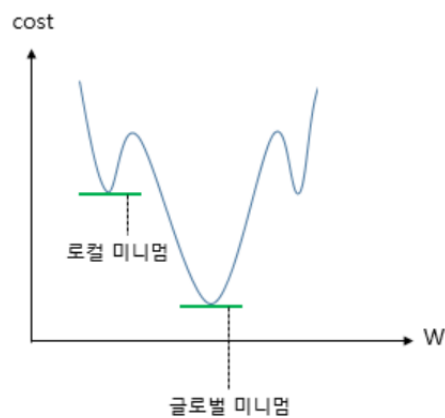
2. ADAM: A method for stochastic optimization

2.1. Gradient Descent with Momentum

우리는 모델을 훈련시키는 방법으로서 Gradient Descent가 예측값과 정답값 간의 차이인 손실함수의 크기를 최소화시키는 파라미터를 찾는 방법이라고 다룬 바 있다.

간단히 말해, 모멘텀은 알고리즘에 운동량을 추가하는 방법이다. 즉, 하강을 통해 최적의 파라미터를 찾는 경사하강법과 달리 ‘얼마나 빨리’ 하강 하는지를 고려해주는 것이다. 그렇다면 단순 경사와 더불어 속도까지 함께 고려해야 하는 이유는 무엇일까?

첫 번째 이유는 SGD의 local minima 문제와 관련이 있다. SGD는 local minimum에 도달했을 때 기울기가 0이므로 이를 global minimum으로 잘못 인식하는 위험을 가지고, 심지어는 최적의 값이라고 판단되는 값이 local minimum인지 global minimum인지 구분할 수 없는 가능성까지 존재한다.



두 번째 이유는 안장점과 관련있다. 미분이 0일 경우 업데이트를 중단하여 안장점을 벗어나지 못하는 SGD의 문제를 Momentum을 통해 해결할 수 있는 것.

Momentum의 사전적 의미는 외부에서 힘을 받지 않는 한 정지해 있거나 운동 상태를 지속하려는 성질을 뜻한다. 이러한 관점에서 고안된 Momentum은 기존의 SGD로 parameter를 이동시킬 때 운동량, 즉 관성을 부여하는 최적화 기법이다.

Momentum은 SGD에서 계산된 점선의 기울기, **한 시점 이전**의 점선의 기울기값을 일정한 비율만큼 반영한다. 파라미터가 얼마나 빨리 하강하는지 알아내는 방법은, 해당 파라미터가 이전에 얼마나 빨리 하강했는지를 확인하는 것이기 때문이다. 이를 통해 Local minimum에 도달할 경우 이전 기울기의 크기를 고려해 추가적으로 이동함으로써 이를 빠져나갈 수 있는 것이고, global minimum에 도달할 경우 추가적인 관성을 받아도 더 올라갈 수 없기 때문에 해당 지점을 global minimum으로 판단해줄 수 있다.

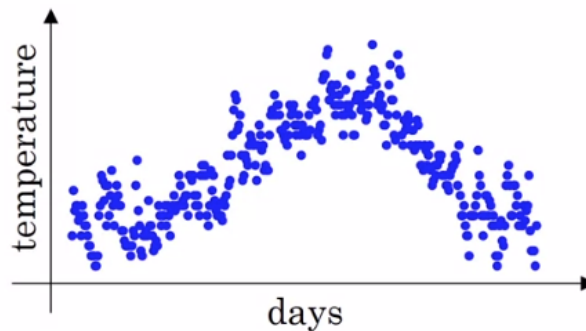
SGD와 비교해봤을 때, 중구난방으로 최적점을 찾아가는 SGD와 달리 그 진동을 줄이고, 최적점에 연관된 방향으로 더 가속을 해 준다는 장점을 지닌다.

2.2. 지수 가중 평균

• Momentum

Momentum을 좀 더 자세히 이해해보자. Momentum 알고리즘을 이해하기 위해서는 지수 가중 평균(지수 이동 평균)을 이해해야 한다.

지수 가중 평균이란 데이터의 이동 평균을 구할 때, 오래된 데이터의 영향을 지수적으로 감소시키는 방법이다.



예를 들어, 위 그림에서 날짜에 따른 기온의 트렌드를 산출하기 위해서 단순 평균을 내는 방법은 매우 위험하다. 단순 평균은 현재를 기점으로, 아주 오래된 데이터의 영향과 가장 가까운 데이터의 영향을 동일하게 만들어 버리므로 시간에 따라 변화하는 추세를 적절하게 나타낼 수 없기 때문이다. 이를 보완하기 위해 각 데이터의 시점에 따라 weight를 다르게 부여하는 방식이 지수 가중 평균인 것.

이러한 지수 가중 평균 식은 다음과 같다. (자세한 유도 과정은 reference 참고)

$$v_t = \beta v_{t-1} + (1 - \beta)\theta_t \quad (1)$$

$$0 \leq \beta \leq 1, \theta : \text{새로운 데이터}, v : \text{현재 경향}$$

이때 v_{t-1} 는 다시

$$v_{t-1} = \beta v_{t-2} + (1 - \beta)\theta_{t-1} \quad (2)$$

로 나타낼 수 있으므로, 식 (1)을 다시 나타내면

$$v_t = \beta\{\beta v_{t-2} + (1 - \beta)\theta_{t-1}\} + (1 - \beta)\theta_t \quad (3)$$

이고, v_{t-2} 앞에 β^2 가 곱해지는 것으로 보아, 오래된 데이터일수록 현재 경향에 미치는 영향이 지수적으로 줄어드는 경향성을 확인할 수 있다.

이러한 알고리즘을 가지고 가중치를 업데이트하는 예시를 살펴해보도록 하겠습니다.

$$v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W} \quad (4)$$

$$W \leftarrow W + v \quad (5)$$

- L : loss function value
- W : weights
- η : learning rate
- α : 가속도를 의미하는 하이퍼파라미터

(4)번 수식은 v 의 영향으로 인해 가중치가 감소하던 혹은 증가하던 방향으로 더 많이 변화하는 것을 의미한다. 최초의 v 값은 당연히 0이다.

간단한 예를 들어보자.

첫 번째 스텝의 기울기를 5, 두 번째 스텝의 기울기를 3, 학습률을 0.1이라고 가정한 경우이다.

$$\frac{\partial L}{\partial W_1} = 5, \frac{\partial L}{\partial W_2} = 3 \quad (6)$$

$$W \leftarrow W - \eta \frac{\partial L}{\partial W_1} = W - 0.5 \quad (7)$$

$$W \leftarrow W - \eta \frac{\partial L}{\partial W_2} = W - 0.3 \quad (8)$$

SGD의 경우 두 번째 스텝에서 가중치는 -0.3만큼 변화한다.

Momentum을 적용한 경우를 살펴보자. α 의 값은 0.9이다.

$$v_1 \leftarrow -\eta \frac{\partial L}{\partial W_1} = -0.5 \quad (9)$$

$$W \leftarrow W + v_1 = W - 0.5 \quad (10)$$

$$v_2 \leftarrow \alpha v_1 - \eta \frac{\partial L}{\partial W_2} = -0.45 - 0.3 = -0.75 \quad (11)$$

$$W \leftarrow W + v_2 = W - 0.75 \quad (12)$$

SGD보다 두 번째 스텝에서 -0.45만큼 가중치가 더 변하는 것을 확인할 수 있다. 즉 Momentum을 사용했을 때 가속도가 고려되어 SGD보다 더 빠르게 하강하는 것.

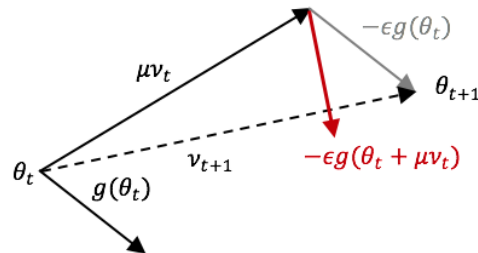
• Nesterov Momentum

또 다른 momentum 알고리즘의 종류로, DenseNet 논문에서 다룬 기법이기에 간략하게 짚고 넘어가보자. 일반 momentum 알고리즘의 공식과 비교하여 살펴보겠다.

$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t \quad (13)$$

$$v_t = \beta v_{t-1} + (1 - \beta)(\theta_t + \beta v_t) \quad (14)$$

Nesterov momentum 은 일반 momentum 의 공식과 거의 같다. 다만 현재 위치의 gradient θ_t 를 이용하는 것이 아니고 현재 위치에서 속도 βv_t 만큼 전진한 후의 gradient $(1 - \beta)(\theta_t + \beta v_t)$ 을 이용한다. 이는 선형적으로 혹은 모험적으로 먼저 진행한 후 에러를 교정한다라고 표현하는데, 현재 파라미터 위치에서 앞쪽의 gradient 를 구한다는 것이 특징이다. 아래 그림은 일반 momentum 의 이동 결과와 nesterov momentum 의 결과를 비교한 것.



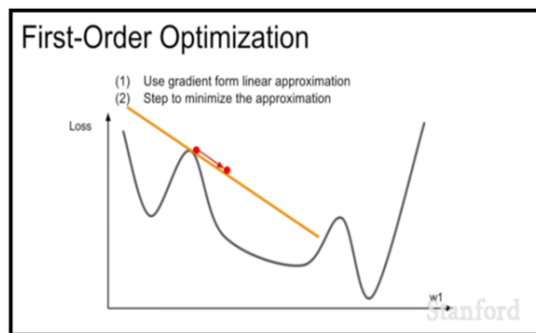
2.3. ADAM 논문 정리

ADAM은 Adaptive Moment Estimation 은 딥러닝 최적화 기법 중 하나로서 Momentum과 RMSProp 의 장점을 결합한 알고리즘이다.

(1) 용어 정리

First - Order Gradient

1차 함수 기울기를 뜻한다. First - order optimization은 한 번 미분한 weight만 optimize에 반영한다는 의미이다. 그림을 보면, first - order optimization은 1차 함수 방향, 즉 직선 방향으로만 최적화가 진행되는 것을 볼 수 있고, 이러한 이유로 gradient의 업데이트가 제한적이다.



이를 보완하기 위해 고차 함수 optimization이 등장하였으나, 이를 이용할 경우 시간 복잡도가 엄청나게 증가하여 아직까지는 first - order optimization이 널리 사용되고 있다. SGD, AdaGrad, RMSProp, Adam 은 모두 이에 해당된다.

Stochastic Object Function

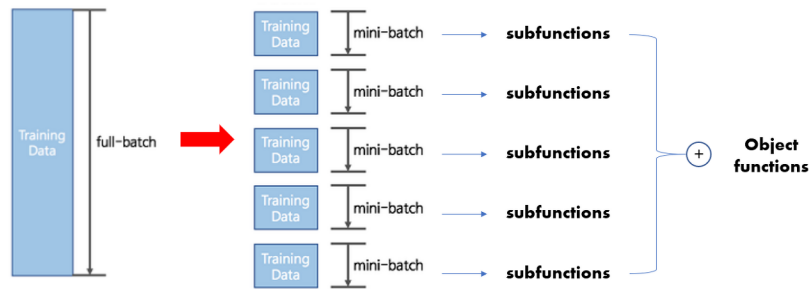
mini-batch 방식과 같이 랜덤하게 데이터 샘플을 선택함으로써 매번 loss function 값이 달라지는 함수를 의미한다. 쉽게 말하면 loss function 이라고 이해하면 되고, 그 종류에는 MSE와 Cross Entropy 등이 있다.

(2) Introduction

Object Function 의 특징

- Objective Function 은 stochastic

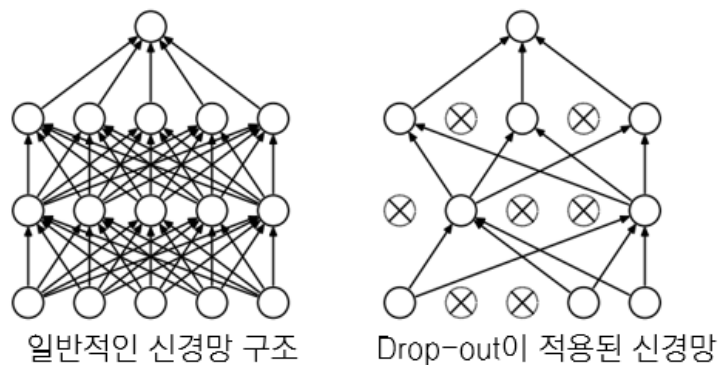
- Objective Function 은 다른 subsample 데이터에서 평가된 subfunctions의 합으로 구성



- 따라서 각 mini-batch마다 학습의 방향과 크기, 즉 gradient steps 를 다르게 줄 수 있기 때문에 학습에 효율적

SGD 알고리즘의 단점

기본적으로, objective function 에 noise가 발생하면 optimization 의 성능이 크게 저하된다.



Objective function 에 노이즈가 발생한다면 적절한 가중치를 부여해주기 위해 더 효율적인 stochastic optimization 이 요구된다.

본문에서는 이러한 stochastic optimization, 그리고 high dimensional 문제에서는 higher - order optimization 보다는 first - order method가 더 적합하다고 판단하여 ADAM 을 제시한다.

(3) Algorithm

ADAM은 첫 번째 moment의 추정치와 두 번째 moment의 추정치로부터 각 파라미터에 대한 개별적인 학습률을 계산한다.

- 첫 번째 moment 의 추정치 : momentum optimizer
- 두 번째 moment 의 추정치 : AdaGrad / RMSProp
- 따라서, ADAM 은 Momentum 과 RMSProp 의 장점을 결합한 최적화 기법

ADAM 의 두 번째 momentum 을 구성하는 두 알고리즘에 대해 짚고 넘어가보자.

AdaGrad

AdaGrad 는 가중치의 업데이트 횟수에 따라 learning rate를 조절하는 옵션이 추가된 최적화 기법이다. 이러한 원리 덕분에 sparse 한 gradient 최적화에도 적합하다고 알려져 있다. 즉, 이전의 gradient 변화량을 참고하는 것(이러한 점에서 Momentum 의 원리가 반영된 것을 확인할 수 있다.)

$$\theta_t = \theta_{t-1} - \alpha \frac{g_t}{\sqrt{\sum_{i=1}^t g_i^2}} \quad (13)$$

위 수식을 해석해보자면, 이미 충분히 업데이트 된 변수들은 optimum 에 거의 도달했다고 보고 stepsize, 즉 learning rate (α 값) 를 작게하고 싶은 것이며, 반대로 업데이트가 많이 되지 않는 weight 들은 learning rate 를 크게 해주는 것이다. 따라서 G_t 라는 새로운 변수를 도입하여 여태까지의 gradient 의 L2 norm (즉 여태까지의 gradient 의 유클리디안 노름) 을 반영하는 것을 확인할 수 있다.

그러나, AdaGrad 는 iteration 이 계속될수록 G_t 가 증가해져서 stepsize 가 너무 작아져 업데이트 강도가 약해진다는 문제점이 있다. 이러한 문제를 보완하기 위해 RMSProp 을 도입한다.

RMSProp

RMSProp 은 AdaGrad 와 같이 과거의 모든 기울기를 균일하게 더하지 않고(L2 norm), 먼 과거의 기울기를 작게, 새로운 기울기 정보를 크게 반영하는 방법이다. 이를 통해 optimum 을 찾기 전에 learning rate 가 0에 수렴할 수 있는 AdaGrad 의 문제를 해결해준다.

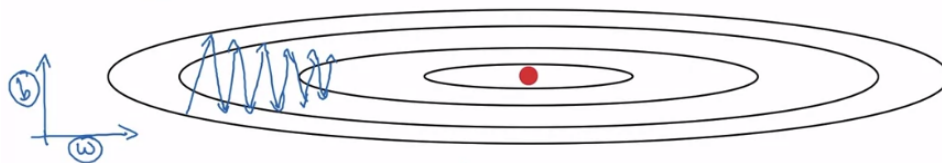
$$G_t = \gamma G_{t-1} + (1 - \gamma) g_t^2 \quad (14)$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t \quad (15)$$

(ϵ : G_t 가 매우 작아졌을 때 0으로 나뉘지는 것을 방지하기 위한 값으로, 1보다 매우 작은 양수)

Momentum 알고리즘과 마찬가지로 지수 이동 평균이 적용된 것을 확인할 수 있다.

예를 들어 RMSProp 을 이해해보자면, iteration 이 진행됨에 따라 개별 parameter 의 크기를 조절할 수 있다는 의미를 가진다.



그림과 같은 상황을 생각해보자. 효율적으로 optimum 으로 도달하기 위해서는 파라미터가 가로축으로 빠르게, 세로축으로는 천천히 진행되어야 한다. 이러한 경우 가로축에 해당하는 파라미터의 g_t 값, 즉 t 시점의 gradient 를 크게하여 $\sqrt{G_t}$ 를 크게할 수 있다. 이는 해당 파라미터 방향으로 학습을 빠르게 진행되도록 조정하는 과정인 것이다.

정리하자면, RMSProp 이 가지는 의미는 각 parameter 별로 적절히 learning rate 의 크기를 조절해줄 수 있다는 데 있으며 전체가 아닌 최근 반복에서 비롯된 gradient 만 누적하여 고려한다는 점에서 AdaGrad 와의 차별점을 두고 있는 것이다.

ADAM

ADAM 은 momentum 이 적용된 first momentum 과 AdaGrad, RMSProp 이 적용된 second momentum 으로 구성되어 있다.

파라미터

- step size α
- decay rates β_1, β_2
- stochastic objective function $f(\theta)$
- parameters(weights) $\theta \Rightarrow$ initial parameter vector θ_0

알고리즘

Require: α : Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0 : Initial parameter vector
 $m_0 \leftarrow 0$ (Initialize 1st moment vector)
 $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 $t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
 $t \leftarrow t + 1$
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
 $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
 $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
 $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
 $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

- i. first moment m_0 , second moment v_0 , time step t 초기화
- ii. parameter θ_t 가 더 이상 수렴하지 않을 때까지 iteration 반복
 - i. time step t 를 증가하여 업데이트
 - ii. stochastic objective function 으로 이전 time step 의 gradient 계산

$$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$$

- iii. biased first & second moment 값 계산

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

- iv. 초기의 momentum 값이 0으로 초기화 되는 경우를 방지하기 위해, bias-correction 을 적용

$$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$$

$$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$$

- v. 최종 parameter 업데이트

$$\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$$

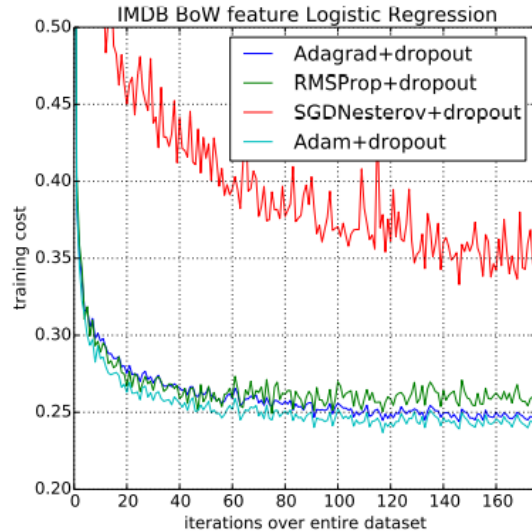
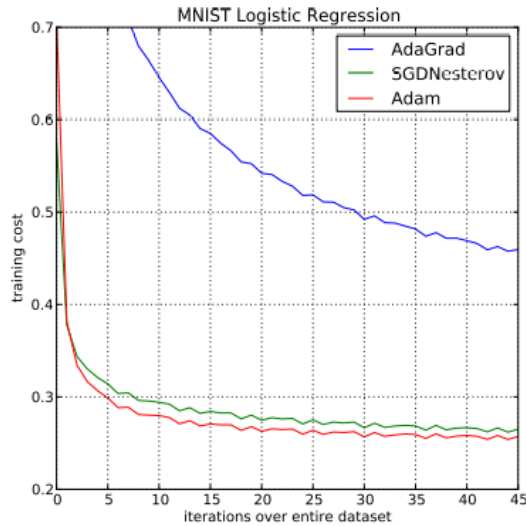
(ϵ : G_t 가 매우 작아졌을 때 0으로 나뉘지는 것을 방지하기 위한 값으로, 1보다 매우 작은 양수)



second moment v_t 가 AdaGrad, RMSProp 역할을 하여 parameter 의 업데이트 횟수에 따라 학습률을 달리 한다. 따라서 업데이트를 많이 한 parameter 일수록 v_t 값이 커진다.

Experiment

본 논문에서는 MNIST 데이터셋과 IMDB 영화 리뷰 데이터 셋을 이용하여 multi-class logistic regression 을 통해 ADAM 알고리즘 성능을 평가하고자 하였다.



왼쪽 그래프는 MNIST 데이터로 최적화를 시행했을 때의 결과인데, ADAM 의 training cost 가 가장 적은 것을 알 수 있다. 오른쪽 IMDB 데이터 셋을 활용했을 때 역시 ADAM 의 경우가 가장 작은 것으로 보이는데, 종합적으로 ADAM 은 SGD 와 유사하게 수렴하고 AdaGrad 보다 빠르게 수렴하는 것을 확인할 수 있었다.

classification 외에 multi-layer neural network 과 CNN 에서의 optimization 을 진행하였는데, dropout 을 적용한 optimizer 와 하지 않은 optimizer 모두 ADAM 의 수렴 속도가 가장 빠른 것을 알 수 있었다.

Conclustion

Stochastic objective function의 최적화를 위해 간단하고 효율적인 최적화 알고리즘인 ADAM을 살펴보았다. 대량의 데이터 셋과 고차원 parameter 공간에 대한 문제 해결에 집중할 수 있고, AdaGrad와 RMSProp을 조합하여 개별 parameter에 대한 기울기 조정을 통해 효율적인 optimization을 진행할 수 있었다. Logistic regression, multi-layer neural network, CNN 등에서 최적화 성능이 다른 알고리즘보다 압도적으로 높은 것으로 평가되었다. 물론 현재 AdaMax, NAdam 등 bias correction 과정을 생략할 수 있거나 Nesterov gradient와 ADAM을 섞은 기법 등 ADAM을 보완할 수 있는 여러 기법들이 존재하고 있기도 하다.

Reference

[Paper Review] Introduction to Image Classification Basic Networks : AlexNet, ZFNet, VGGNet, ResNet

[Paper Review] Introduction to Image Classification Basic Networks : AlexNet, ZFNet, VGGNet, ResNet

1) 발표자: DSBA 연구실 석사과정 정익석[2] 발표 논문: 본 발표는 Image Classification Task의 기반이 되는 논문 4개를 소개합니다.- AlexNet : ImageNet Classification with Deep Convolutional Neural...

▶ https://www.youtube.com/watch?v=vRtM4K8e_Q4

Papers You Must Read

Deep Learning Image Classification Overview

AlexNet, ZFNet, VGGNet, ResNet

고려대학교 산업경영공학과 정익석

2020.11

[Paper Review] ResNet

ResNet 논문 리뷰

논문 링크 : <https://arxiv.org/pdf/1512.03385.pdf> 딥러닝에서 neural networks가 깊어질수록 성능은 더 좋지만 train이 어렵다는 것은 알려진 사실이다. 그래서 이 논문에서는 residual를 이용한 잔차학습(residual learning framework)를 이용해서 깊은 신경망에서도 training이 쉽게 이뤄질 수 있다는 것을 보이고 방법론을

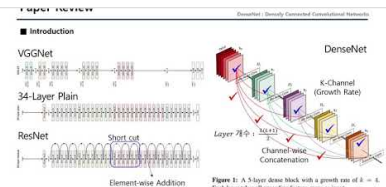
<https://codebaragi23.github.io/machine%20learning/3.-ResNet-paper-review/>

[Paper Review] DenseNet

[Paper Review] DenseNet

1. 발표자 : DSBA 연구실 석사과정 김호영 2. 발표 논문 : DenseNet (<https://arxiv.org/pdf/1608.06993.pdf>)
3. 개요 : - 본 리뷰에서는 Image Classification Task의 Model 중 하나인 DenseNet을 다룹니다....

 <https://www.youtube.com/watch?v=GKo6Mcozl0o&t=1s>



DenseNet Tutorial [1] Paper Review & Implementation details

안녕하세요, 오늘은 오랜만에 Image Classification 분야의 논문을 리뷰하고, 코드로 구현하는 과정을 설명드릴 예정입니다. 오늘 리뷰할 논문은 DenseNet으로 잘 알려져 있는 CNN architecture를 다룬 "Densely Connected Convolutional Networks" 이라는 논문입니다. 2017년 CVPR Best Paper Award를 받았으며

 <https://hoya012.github.io/blog/DenseNet-Tutorial-1/>

hoya012
Deep Learning
Blog!

DenseNet - Organize everything I know documentation

ResNet은 이전 결과와 현재 결과를 Summation으로 이어주는 구조였다면, DenseNet은 모든 결과를 Concatenation으로 다 연결시키는 구조라고 볼 수 있다. 그래서 ResNet은 결국 최종적으로 마지막 Layer의 결과로 Classification을 하는데, DenseNet은 전체 Layer 결과를 모두 반영하여 Classification을 할 수 있다. 그렇기 때문에 저자들은 DenseNet이 더 좋은 성능을 낼 수 있다고 주장한다.

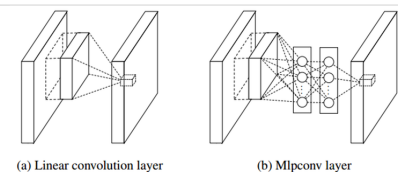
 https://oi.readthedocs.io/en/latest/computer_vision/cnn/densenet.html

1*1 convolution

[Part V . Best CNN Architecture] 5. GoogLeNet [2] - 라온�플 머신러닝 아카데미 -

Part I. Machine Learning Part V. Best CNN Architecture Part VII. Semantic Segmentat...

 <https://m.blog.naver.com/PostView.naver?blogId=laonple&logNo=220692793375&proxyReferer=>



Lecture 5 | Convolutional Neural Networks

Lecture 5 | Convolutional Neural Networks

In Lecture 5 we move from fully-connected neural networks to convolutional neural networks. We discuss some of the key historical milestones in the developme...


 <https://www.youtube.com/watch?v=bNb2fEVKeEo>

Lecture 5:
Convolutional Neural Networks

Lecture 7 | Training Neural Networks II

Lecture 7 | Training Neural Networks II

Lecture 7 continues our discussion of practical issues for training neural networks. We discuss different update rules commonly used to optimize neural netwo...

 https://www.youtube.com/watch?v=_JB0AO7QxSA&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRf3EO8sYv&index=7

Lecture 7:
Training Neural Networks,
Part 2

Lecture 9 | CNN Architectures

Lecture 9 | CNN Architectures

In Lecture 9 we discuss some common architectures for convolutional neural networks. We discuss architectures which performed well in the ImageNet challenges...

 <https://www.youtube.com/watch?v=DAOcjFr1Y&t=14s>

Lecture 9:
CNN Architectures

Google Colaboratory

https://colab.research.google.com/drive/1AUo9vrbEGCznzpvj89ZztuZO0bZn_CPx?usp=sharing

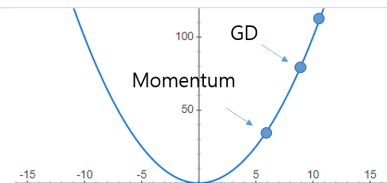


Momentum

딥러닝 용어정리, Momentum, AdaGrad 설명

제가 공부한 내용을 정리하는 글입니다. 제가 나중에 다시 보기 위해 작성하는 것이므로 본문은 편의상 반말로 작성했습니다. 잘못된 내용이 있다면 지적 부탁드립니다. 감사합니다. Momentum 은 Gradient descent(이하 GD) 기반의 optimization algorithm 이다. 수식으로 표현하면 아래와 같다. L : loss function value W :

📄 <https://light-tree.tistory.com/140>



Momentum을 이용한 최적화기법 - ADAM

※ 본 포스팅은 Andrew Ng 교수님의 강의를 정리한 것입니다. Python 라이브러리를 이용한 딥러닝 학습 알고리즘에 관련된 tutorial들에서 거의 대부분 optimization을 수행할 때 Gradient Descent 대신에 ADAM Optimizer를 이용해 O...

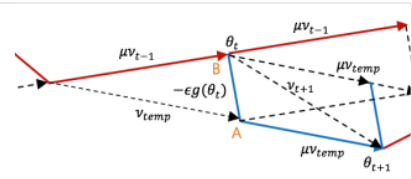
📄 https://angeloyeo.github.io/2020/09/26/gradient_descent_with_momentum.html

Nesterov Momentum

Momentum & Nesterov momentum

경사하강법 gradient descent 최적화 알고리즘의 한 종류인 모멘텀 momentum과 네스테로프 모멘텀 nesterov momentum 방식은 여러 신경망 모델에서 널리 사용되고 있습니다. 비교적 이 두가지 알고리즘은 직관적이고 쉽게 이해할 수 있습니다. 이 글에서는 두 알고리즘이 실제 구현에서는 어떻게 적용되는지 살펴 보겠습니다.

📄 <https://tensorflow.blog/2017/03/22/momentum-nesterov-momentum/>



지수 가중 평균

[딥러닝] Optimization Algorithm (최적화 알고리즘)

이번 포스팅은 Neural Network를 빠르게 훈련시키는 최적화 알고리즘에 관한 내용입니다. 딥러닝은 크기가 큰 데이터의 경우 잘 작동하는데, 데이터의 크기가 클수록 훈련 속도는 느려집니다. 이런 경우 효율성을 높이기 위한 최적화 알고리즘을 잘 선택해야 합니다. 여러가지 최적화 알고리즘의 개념에 대해 알아보고

📄 <https://velog.io/@minjung-s/Optimization-Algorithm>

[딥러닝]

RMSProp

[Deep Learning] 최적화(Optimizer): (3) RMSProp

AI & 빅데이터/머신러닝-딥러닝 Tony Park 2022. 5. 21. 02:16 RMSProp는 딥러닝 최적화 기법 중 하나로써 Root Mean Square Propagation의 약자로, 알엠에스프롭(R.M.S.Prop)이라고 읽습니다. 최적화 기법 중 하나인 AdaGrad는 학습이 진행될 때 학습률(Learning rate)이 꾸준히 감소하다 나중에는 0으로 수렴하여 학습

📄 <https://heytech.tistory.com/384>

최적화 기법
RMSProp

ADAM

[딥러닝] 딥러닝 최적화 알고리즘 알고 쓰자. 딥러닝 옵티마이저(optimizer) 총정리

이번 포스트에서는 딥러닝에 사용되는 최적화 알고리즘을 정리해보려고 한다. 지금까지 어떤 근거도 없이 Adam을 써왔는데, 최근에 잘 해결되지 않던 문제에 SGD를 써보니 성능이 훨씬 향상된 경험에 있다.. 이에 "최적화 알고리즘을 알고 쓰자!"라는 마음이 생겼고, 그래서 이 포스트를 작성하게 되었다. 이 포스트는 최적화

📄 https://hiddenbeginner.github.io/deeplearning/2019/09/22/optimization_algorithms_in_deep_learning.html



제가 공부한 내용을 정리하는 글입니다. 제가 보려고 쓰는 글이기 때문에 아래는 작성의 편의상 반말로 작성했습니다. 이전 AdaGrad 설명에서 언급했듯이 AdaGrad 는 최소값에 도달하기 전에 학습률을 0에 수렴하게 만들 수도 있다. AdaGrad 가 간단한 convex function 에선 잘 동작하지만, 복잡한 다차원 곡면 함수를 (상대적

🌳 <https://light-tree.tistory.com/141>

