컴퓨터비전특론 6th week summary

[Lec07: Training Neural Networks Part 1] 이어서

**Batch Normalization**

Layer의 각 차원이 unit gaussian 분포를 따르도록 정규화하는 과정.

예를 들어 총 K개의 차원이 있는 k-dimensional input 가 주어졌을 때, 에 대해 각 차원의 데이터마다 다음과 같은 식으로 정규화를 진행한다.

주로 Fully connected 혹은 Convolutional layer 이후와 tanh 혹은 ReLU 등의 비선형 함수 이전 단계에서 진행한다.

normalized된 input을 sigmoid함수에 입력할 경우, input의 범위가 0에서 1사이이므로 sigmoid function에 가운데 부분으로만 맵핑되어 linear function을 통과시킨 것과 같아진다. 따라서 normalized된 데이터가 일 때, nonlinear 형태로 변환시키고자 다음과 같은 affine transformation을 한번 더 거친다(scale and shift).

Batch normalization을 사용할 경우 gradient flow가 개선되며, learning rate를 크게 사용할 수 있게 한다. 또한 initialization에 대한 의존성을 줄일 수 있으며, overfitting을 방지하는 regularization의 역할을 한다.

Batch normalization은 그룹으로 묶어진 데이터에 대해 평균과 분산을 계산해야 하는데, test data의 경우 데이터가 하나씩 들어오므로 평균과 분산을 구하는 의미가 없다. 따라서 이를 해결하기 위해 train data를 적합하는 과정에서 평균과 분산에 대한 moving average를 계산하고 이를 test에서 사용한다.

Other normalization methods

- Layer norm: 배치 내에 속하는 각 image별로 독립적으로 normalize

- Instance norm: 배치 내에 속하는 각 image 내에서도 channel별로 normalize

- Group norm: 배치 내에 속하는 각 image 내에서 채널을 몇 개의 group으로 나눠 group별로 normalize

**Babysitting the learning process**

1. Preprocess the training data

: normalization 등

2. Choose the architecture

: learning rate의 parameter 설정 등에 따라 모델의 성능이 많이 달라지므로 적절한 parameter를 찾는 과정은 중요하다.

3. Adjust training parameters

**Hyperparameter optimization**

Hyperparameters

Network architecture

Learning rate와 이에 대한 decay schedule(decay constant), update type(SGD, Momentum, AdaGrad, RMSProp…)

Regularization strength (L2 penalty, dropout strength)

Cross-validation strategy

- First stage: 몇 개의 epoch만을 사용하여 대략적인 parameter의 range를 찾는 과정

- second stage부터는 finer search(더 긴 running time)

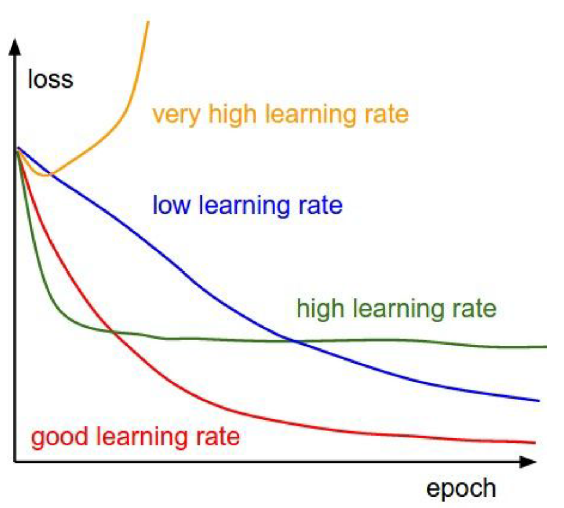
parameter search

그냥 찾는 것보다 log scale을 사용하는 것이 더 빠르고 정확하게 찾을 수 있다.

Grid search vs random search

Grid search: 전체 분포를 grid로 나누고 정해진 몇 개의 구간에 대해서만 탐색

Random search: 전체 분포의 여러 지점을 random하게 고려함 -> grid search보다 더 나은 결과

Learning rate에 따른 loss의 변화

- low learning rate의 경우 수렴 속도가 느린 대신 epoch가 많아질수록 high learning rate의 경우보다 더 낮은 loss를 얻는다.

- high learning rate의 경우 수렴이 빨리 되어서 초반에 loss가 급격히 줄어든다. 그러나 초반 이후에는 loss가 크게 감소하지 않는다.

- learning rate에 따라 모델의 성능이 달라지므로 수렴 속도와 loss의 lower bound 모두 적절한 learning rate를 찾는 것이 중요하다. 또는 초반에는 크게, 나중에는 작게 설정하는 learning rate decay를 사용할 수도 있다.

Gap between training and validation accuracy

Training과 validation의 accuracy가 크게 차이가 난다는 것은 해당 모델이 overfitting 되었음을 뜻한다. 이러한 경우 규제항 의 를 더 크게 하거나 dropout을 사용하는 등의 regularization strength를 높이는 방법을 사용하여 문제를 해결해볼 수 있다.

반대로 두 accuracy 사이의 gap이 아예 없다는 것은 model capacity가 너무 낮음을 뜻하므로, capacity를 증가시킬 필요가 있다.

[Lec08: Training Neural Networks Part 2]

**Optimization**

SGD(Stochastic gradient descent)

Loss function을 weight에 대해 미분한 값인 gradient에 learning rate를 곱한 값을 계산하고, 이 값을 weight에서 빼는 순서로 weight를 업데이트하는 방식.

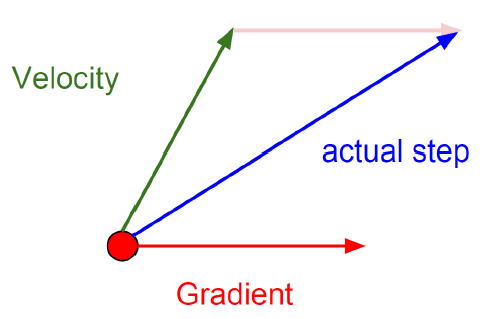
- SGD의 문제점

* Loss function의 high condition number(조건수): Hessian matrix의 가장 큰 eigen value와 가장 작은 value의 차이가 큰 경우. Contidtion number가 높을 경우 gradient의 이동 과정에 noise가 심하여 지그재그로 이동하게 된다.
* Local minima 또는 saddle point(미분값이 0인 point)에 갇힐 가능성이 있다.
* minibatch로부터 계산되는 gradient들이 noisy할 수 있다.

SGD + Momentum

속도 (velocity)의 개념을 도입: gradient가 기존에 어떤 속도로 움직이고 있었는지를 고려한다.

이 때 friction 는 주로 0.9 또는 0.99를 사용한다. 속도인 는 gradients들의 running mean이라고 할 수 있다.



기존의 SGD가 빨간 화살표 방향으로 움직였다면, momentum을 더한 SGD는 현재 속도인 velocity를 반영하여 파란색 화살표 방향으로 움직인다

이러한 방법은 현재 까지의 velocity를 반영해서 움직이게 하므로 local minima나 saddle point로부터 빠져나올 수 있게 하며, 기존의 SGD보다 smooth하게 optimal value를 찾아 나간다.

Nesterov momentum

현 지점이 아닌 방향을 따라 언정도 움직인 지점에서의 gradient를 계산한다. SGD에 momentum을 더한 계산 방법과 성능의 차이가 크게 나지는 않는다.

AdaGrad(Adaptive gradient)

Gradient를 각 component마다 다르게 준다. 즉 learning rate를 다르게 설정하는 것이다.

위 식 중 의 업데이트 식에서 이 learning rate의 역할을 한다. 이 term에서 사용되는 는 이전 단계까지의 각 dimension의 gradient의 historical sum과 같다. 이러한 식을 통해 많은 움직임이 있었던 element는 learning rate를 낮게 만들어 더 세밀하게 optimal value를 찾게 하고, 움직임이 적었던 element는 learning rate를 크게 하여 더 빨리 optimal value로 수렴하게 만든다.

그러나 이 과정이 계속해서 반복되면 gradient가 계속해서 더해지므로 learning rate가 0으로 간다는 문제점이 있다.

RMSProp

학습이 많이 진행될수록 learning rate가 0에 가까워지는 AdaGrad의 한계점을 해결하기 위해, gradient의 제곱합 대신 gradient의 moving average를 사용하여 기존의 정보는 어느정도 유지하면서 새로운 정보로 업데이트하게 하였다. Moving average를 사용함으로써 가 계속해서 커지지 않고 적절한 값을 유지할 수 있다.

Adam

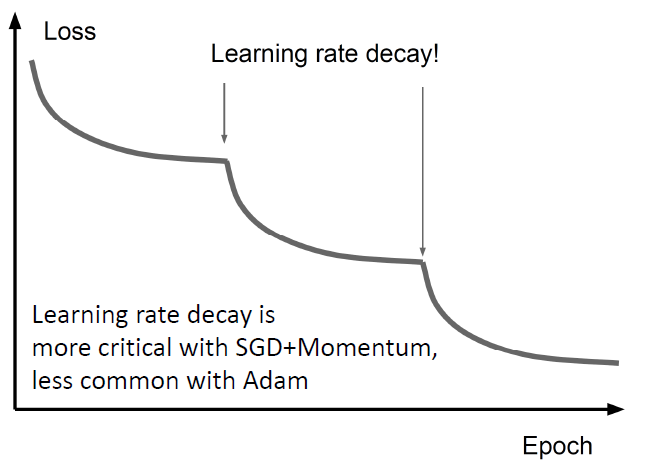
RMSProp with momentum과 같다. Learning rate 뿐만 아니라 gradient도 momentum update를 적용하였다.

위의 식에서 는 gradient momentum의 update, 는 learning rate momentum의 update 값이다.

이 때 gradient와 learning rate의 momentum은 0에서 시작하여 점점 업데이트를 거친다.

Learning rate as hyperparameter

가장 적합한 learning rate 찾기: step decay를 사용하자(몇번의 epoch이 지난 후 learning rate 줄이기)



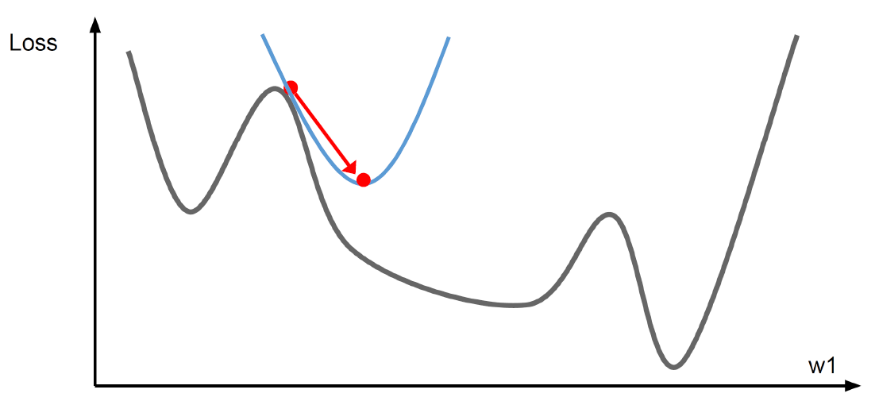
그러나 Adam의 경우 adaptive하게 learning rate가 설정되므로 learning rate decay가 큰 의미는 없음.

Second-order optimization

- First-order optimization의 경우, linear approximation으로부터 gradient를 사용

: 어떤 방향으로 갈지는 정해주지만 얼마나 가야 하는지는 지정해주지 않으므로 learning rate 설정이 필요하다.

- Second-order optimization

해당 위치에서 2차 함수를 fitting

: 2차 함수 내 최소값, 즉 미분값이 0이 되는 지점으로 업데이트

-> 이동해야 할 방향뿐만 아니라 어디로 이동해야 하는지도 계산되므로 learning rate가 필요하지 않다.

Second-order Taylor expansion

이 때 는 Jacobian matrix, 는 hessian matrix이다. 위의 식을 에 대해 미분하고 0이 되는 값을 찾기 위한 식을 세우면, update된 weight 를 다음과 같이 계산할 수 있다.

위에 식에는 별도의 hyperparameter가 없으며, hessian matrix가 learning rate의 역할을 한다.

그러나 hessian matrix의 역행렬을 구하는 시간복잡도는 으로 매우 크다. 이를 해결하기 위한 방법으로 BGFS와 L-BGFS가 있다.

BGFS(Broyden-Fletcher-Goldfarb-Shanno) 알고리즘은 Quasi-newton 방법을 사용하여 inverse hessian matrix를 시간복잡도가 가 되도록 approximate한다. 시간복잡도가 줄긴 했지만 여전히 많은 메모리를 차지하는 문제가 있다.

L-BFGS(Limited memory BFGS)는 block diagonal matrix의 형태를 사용하여, matrix 내 block에 대해서만 inverse를 계산한다. 그러나 이 방법은 full batch에 적합하며, mini-batch 방법을 사용하려면 모든 stochastic setting을 사용하지 않는 상태여야 한다.

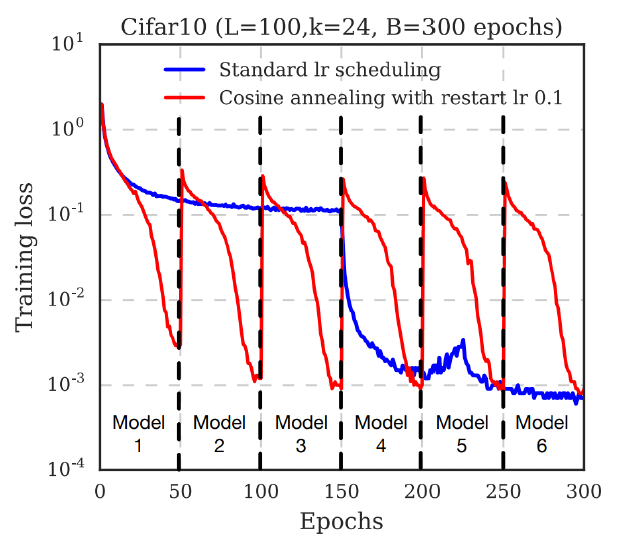
Early stopping

Train accuracy와 validation accuracy 간의 gap을 줄일 수 있는 방법으로, validation set의 accuracy가 특정 횟수 이상 감소하면 미리 설정한 training 횟수보다 빨리 적합을 멈추게 한다.

**Model Ensembles**

Independent한 다수의 모델을 학습하고, 모든 모델의 결과들을 종합하여 최종적인 예측값을 만드는 방법이다.

그러나 다수의 모델을 학습하는 것은 많은 시간이 소요된다. 이러한 문제를 막기 위해, 이 방법을 대신하여 하나의 모델을 한번 학습하는 중간에 여러 개의 weight parameter를 저장하는 multiple snapshots 방법을 사용할 수 있다.



옆의 그래프에서 빨간색으로 표시된 선은 cosine annealing을 사용하여 snapshot ensemble을 시도한 예시다. 총 6개의 다른 weight parameter가 얻어진 것을 알 수 있다. 매 시도마다 다른 initialization이 적용되기 때문에 이에 따라 weight도 매번 다른 값을 얻을 수 있다.

Polyak averaging

모델 fitting 과정에서 test를 위한 weight parameter를 moving average로 계산해둔다.