SGD: minibatch 내에서 data loss 를 계산하고, gradient 의 반대방향을 이용해 parameter vector 를 업데이트 하는 방식

단점

- 느림: 아래로 볼록한 모양의 loss function 이 있을 때, x 방향, y 방향의 합방향으로 gradient 가 업데이트 될 것. 이때, 수평방향으로는 기울기 변화가 작고, 수직방향으로는 기울기 변화가 큼. 따라서 한방향으로 내려가는 것이 아니라 지그재그 형태로 최저점에 도달하게 되어 오랜 시간이 걸림
- local minima, saddle point: local minima 의 경우, loss 함수에서 기울기가 0 인 부분에 도달하게 되면 더이상 update 를 진행하지 않아 global minima 를 찾기 전에 멈춰버릴 수 있음. saddle point 문제는 고차원 그래프에서 많이 발생하는데, 주변 기울기가 완만한 경우 wegit update 를 느리게 진행하고, 계속되는 경우, 머무르게 됨

SGD + momentum: local minima, saddle point 문제를 해결하기 위해 momentum 과 SGD 를 합친 방식

-> weight 를 update 할 때 velocity 의 개념을 추가한 것 (gradient vector 방향 + velocity vector)

-> rho: velocity 의 영향력

-> 속도 개념을 사용할 경우, gradient 가 0 이더라도 충분히 움직일 수 있음. 이는 local minima 와 saddle point 를 극복할 수 있게 함. update 가 잘 안되는 경우, 위에서 지그재그로 움직이는 경우에도 momentum 이 변동을 상쇄시켜 수평방향의 움직임을 좀 더 가속화할 것. 즉, 노이즈가 평균화됨

Nesterov Momentum: 현재지점에서 gradient 를 계산하고 velocity 를 추가해주는 기존 SGD momentum 과는 다르게 neterov 는 velocity 방향으로 먼저 움직인 후, 그 지점에서 gradient 를 계산하고, 둘을 합하는 것 (convex model 에서는 성능이 좋지만 nonconvex model 에서의 성능은 좋지 않음)

AdaGrad: 일종의 매개변수 맞춤형 update algorithm. velocity 대신에 grad squared 즉, 기울기값을 이용함

-> 학습이 계속 진행되면 학습 횟수가 늘어나고, update 속도가 느려짐.
-> Convex 할 경우, 유리하지만, nonconvex 의 경우, saddle point 에서 멈출 수 있어 NN 을 학습시킬 때 잘 사용하지 않음

RMSProp: AdaGrad 의 문제를 해결하기 위한 것. decay rate 을 이용해 step 의 속도를 조절할 수 있음

Adam: 가장 많이 사용하는 방법으로, first moment 와 second moment 를 이용해 이전의 정보를 유지시킴

- 일종의 RMSProp 과 momentum 을 합친 방식
- 초기 스텝에서는 second moment 를 0 으로 초기화하는데, 이 때 초기 step 이 엄청 커져서 잘못될 수 있음
- 이를 보정하는 항이 biased correction; first/second moments 를 update 하고, 현재 step 에 맞는 적절한 bias 를 부여해 값이 엄청 튀지 않도록 조절함
- 일반적으로 효과가 좋음

learning rate; step decay 라는 방식이 있음. learning rate decay 는 부차적인 하이퍼파라미터임. 학습 초기부터 고려하지 않기에 보통 학습 초기에는 learning rate decay 가 없다고 생각하고 learning rate 를 잘 선택하는 것이 중요함

지금까지는 1 차미분을 활용한 optimization 알고리즘을 살펴봤음. second-order approximation 의 정보를 추가적으로 활용하는 방법이 있을 수 있음. 이를 활용하면, minima 에 더 잘 근접할 수 있음.
ex) Hessian matrix 이용하기: learning rate 없이 minima 에 접근 가능 -> 기본적인 Newton's method 에서 learning rate 는 불필요함. 하지만, 실제로는 2 차 근사 또한 완벽하지 않으므로 learning rate 가 필요함. (minima 로 이동하는 것이 아니라 minima 의 방향으로 이동하기 때문)
-> Deep learning 에서는 사용할 수 없음
-> quasi-Newton methods 를 실제로 이용: Full Hessian 을 그대로 사용하기보다 근사시켜 Low-rank approximations 하는 방법임

L-BFGS: Hessian 을 근사시켜서 사용하는 방법. 사실상 DNN 에서는 잘 사용하지 않음

- L-BFGS 에서 2 차근사가 stochastic case 에서 잘 동작하지 않음
- non-convex problems 에 적합하지 않음

-> 결론적으로는, Adam 을 많이 씀. 하지만, full batch update 가 가능하고 stochasticity 가 적은 경우, L-BFGS 는 좋은 선택이 될 수 있음

Model ensemble: 한번도 보지 못한 데이터에서의 성능을 올리기 위해 사용하는 가장 빠르고 쉬운 방법. Machine learning 분야에서 사용

- 모델을 하나만 학습시키지 않고 여러개의 모델을 독립적으로 학습시키고 해당 모델 결과의 평균을 이용함
- model 의 수가 늘어날수록 overfitting 이 줄어들고, 성능은 향상됨
- 혹은 학습 도중에 중간 모델을 저장하고 앙상블로 사용할 수도 있음. 이 경우, test 시에 여러 snapshot 에서 나온 예측값을 평균 내서 사용함 -> 모델을 한번만 train 시켜도 좋은 성능을 얻을 수 있음

Regularization: 앙상블은 모델을 여러개 만들기 때문에 효율적이지 않음. 이때 사용하는 것이 regularization

- dropout: 일정 노드를 랜덤으로 out 시켜버리는 것. forward pass 과정에서 일부 뉴런을 0 으로 만듦. 이때, forward pass 할 때마다 0 이 되는 뉴런이 바뀜
    - 노드가 어떤 일부 feature 에만 의존하지 못하게 하여 overfitting 을 막아줌
    - 단일 모델로 ensamble 효과를 얻을 수 있음 -> dropout 은 서로 파라미터를 공유하는 서브네트워크 앙상블을 동시에 학습시키는 것이라고 생각할 수 있음
    - 이를 사용하면 NN 의 동작 자체가 변함

        - Network 에 z 라는 입력이 추가되는데, 이는 random dropout mask 임. 하지만, z 는 임의적이므로 test time 에는 적절하지 않음. 따라서 적분을 통해 그 임의성을 average out 시킴.
        - 여기서 문제는 이 적분이 어렵다는 것. 따라서 적분식을 근사화시켜서 이용함
        - dropout 을 사용하게 되면 네트워크 out 에 dropout probability 를 곱해줘야 함
- BN: train time 에서는 각 데이터에 대한 stochasticity 가 존재했지만, test time 에서는 minibatch 단위가 아닌 galobal 단위로 수행하여 stochasticity 를 평균화시켰음. -> dropout 과 유사한 regularization 효과
- data augmentation: 이미지를 조금씩 변형시키는 방식. 원본 이미지를 무작위로 변화시킨 이미지를 학습시킴
    - train time 에 input data 에 변환을 주게 되면 regularization 효과를 얻을 수 있음. <- train time 에는 stochasticity 가 추가되고 test time 에는 marginalize out 되기 때문
- 또다른 방식: DropConnect, fractional max pooling

Transfer Learning: CNN 을 학습할 때 많은 데이터가 필요하다는 편견을 깸

- 기존의 CNN layer 를 아주 큰 데이터셋으로 학습시킴
- 훈련된 model 을 작은 dataset 에 적용시킴
- 최종 FC layer 를 초기화시키고, 이전의 나머지 layer 의 weight 은 freeze 시킴
- 즉, 마지막 layer 만 가지고 데이터를 학습시키는 것
- data 가 좀 더 많다면, 좀 더 위의 layer 에서 fine tuning 함. 이때, 이미 model 이 잘 학습되어 있으므로 learning rate 는 작아도 됨