CS231n 7강 요약

NN을 훈련시키기 위해서 해야 하는 것들을 마저 이어 배운다.

지난 시간에는, 가중치 초기화, learning rate 설정, batch normalization 등에 대해 배웠고,

이번 시간에는 NN을 최적화시키기위한 gradient update 방법들, regularization 방법들을 배운다.

1. Gradient update 방법, Optimization 방법

1) SGD : Stochastic Gradient Descent 확률적 경사하강법

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

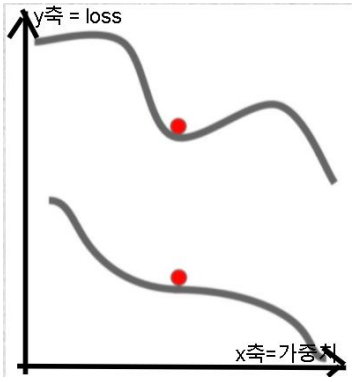
- αα = learning_rate

- dx = x 에서의 loss 에 대한 기울기

- 초기에 설정한 learning rate 을 이용하여 손실함수(loss)의 값이 최소가 되는 방향으로 가중치를 업데이트하게 된다

- 단점 1: 느리다.

- 단점 2: local minima (그림 위쪽) & saddle point (그림 아래쪽).



>> 기울기가 0 이 되어도 멈추지 않게 하는 힘이 필요한 것.

2) SGD + Momentum 방법

SGD 의 local minima 와 saddle 문제를 해결하기 위해 "Momentum"이란 방법을 결합하게 된다.

- 가중치 x 를 업데이트할때, (가)속도 vx 의 개념이 추가되었다.

- 모멘텀은 물리에서 운동량의 개념이다. 공이 언덕을 내려오다가 기울기가 0 인 지점을 만났더라도 이때까지 내려오던 속도에 의해 그 골짜기를 지나 계속 굴러갈 수 있을 것이다. 이렇게 이때까지 내려오던 속도의 개념을 추가했다.

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

모멘텀방식은 xt+1=xt−α(ρvt+∇f(xt))xt+1=xt−α(ρvt+∇f(xt))

SGD 방식은 xt+1=xt−α(∇f(xt))xt+1=xt−α(∇f(xt))

가중치업데이트에 직전스텝의 속도 vtvt 가 고려되었다는 점이 가장 큰 차이다.

이 vtvt 역시 이전의 모든스텝의 속도가 누적된 결과일 것이고, 왼쪽 그래프에서 보듯이, 이 속도항 때문에 SGD 에서 이동할 거리보다 모멘텀방식은 더 큰 스텝으로 뛰어가게 된다.

3) Nestero Momentum

모멘텀방식을 약간 더 업그레이드 한 방법이다.

기존의 모멘텀은 직전지점의 gradient 와 velocity 벡터의 합벡터로 다음 스텝의 지점을 구한다.

Nestero Momentum 방식은 직전지점의 velocity 벡터의 종착점을 예상해서, 그 지점에서의 gradient 벡터를 합하여 다음스텝의 지점을 구하게 된다.

3) AdaGrad update

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

+) 수직축과 수평축의 업데이트 속도를 적절히 맞춰준다.

-) cache 값이 계속 누적됨으로 인해 뉴럴네트워크의 학습이 진행되다보면 가중치가 0 이 되어 학습이 종료될 수 있다.

5) RMSProp update

AdaGrad 의 학습종료 문제를 개선한 알고리즘이다.

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

- 누적된 grad_squared 항에 **decay_rate** 를 곱하고, 현재의 dx 항에는 (1-decay_rate)를 곱해 grad_squared 가 누적되는 속도를 줄여준다.

- 보통 decay_rate 는 0.9 나 0.99 를 사용한다.

6) Adam

- 일종의 RMSProp + Momentum 을 합친 방식

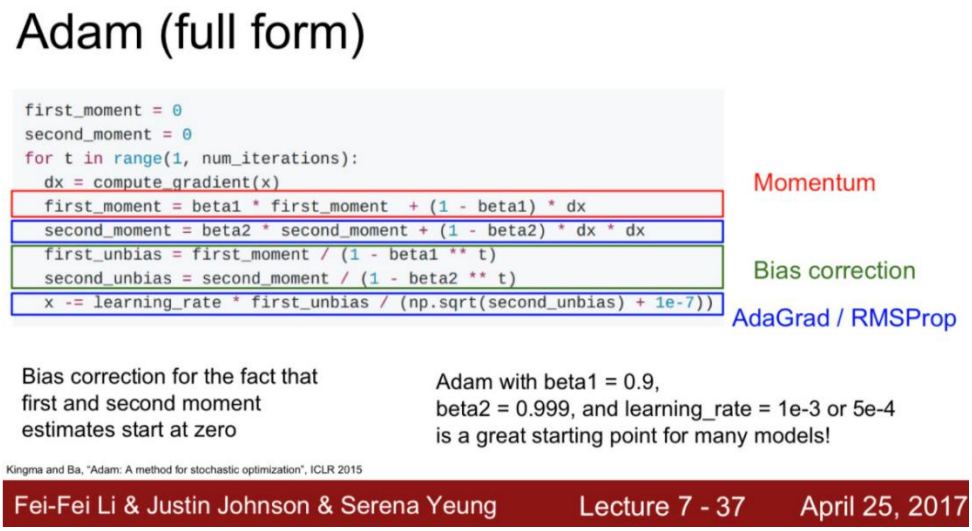
```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

momentum

RMSProp-like

Looks a bit like RMSProp with momentum

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



1차근사, 2차근사

Emsembles

- 이 때까지 training data 에 대하여 Loss 를 최소화시키는 알고리즘훈련방법에 대하여 배웠다.

- 그렇다면 처음보는 validation data 에도 이 알고리즘이 잘 작동하게 하려면 어떻게 할까?

이 때 앙상블(Ensembles)이라는 방법을 적용할 수 있다.

여러개의 모델을 각각 따로 학습시켜 그 평균값을 이용하는 것이다.

2. Regularization

1) Dropout

train data에 모델이 overfitting되는 것을 줄이고 처음 본 data에도 잘 작동하게 하기 위해서 일종의 regularization으로 dropout이라는 방법을 적용할 수 있다.

- 위와 같은 뉴럴네트워크에서 일부노드를 랜덤하게 0 으로 만들어 노드스위치를 꺼버린다.

- 중요한 것은 epoch 를 돌때마다 정방향전파과정에서 꺼지는 노드는 계속해서 랜덤하게 변한다는 것이다.

- 이 방법은 학습과정에서 **redundancy(중복성)을 줄인다**라는 의미가 있다.
- 일종의 ensemble 방식이다. train epoch 를 돌 때마다 랜덤으로 바뀌는 노드의 구성이 변하기 때문에 마치 여러모델의 값을 평균내는 ensemble 의 효과를 갖게 된다.

2) Data Augmentation

train data 에 변형을 줘볼 수도 있다.

1. 사진 좌우반전
2. 사진을 랜덤하게 자르기
3. 사진 크기 바꾸기

3) Drop Connect

- Dropout 은 노드의 activation 을 0 으로 만드는 것이었고,
- Dropconnect 는 노드의 weight matrix 를 0 행렬로 만드는 것이다.

4) Max pooling & Stochastic depth