-Lecture 6-

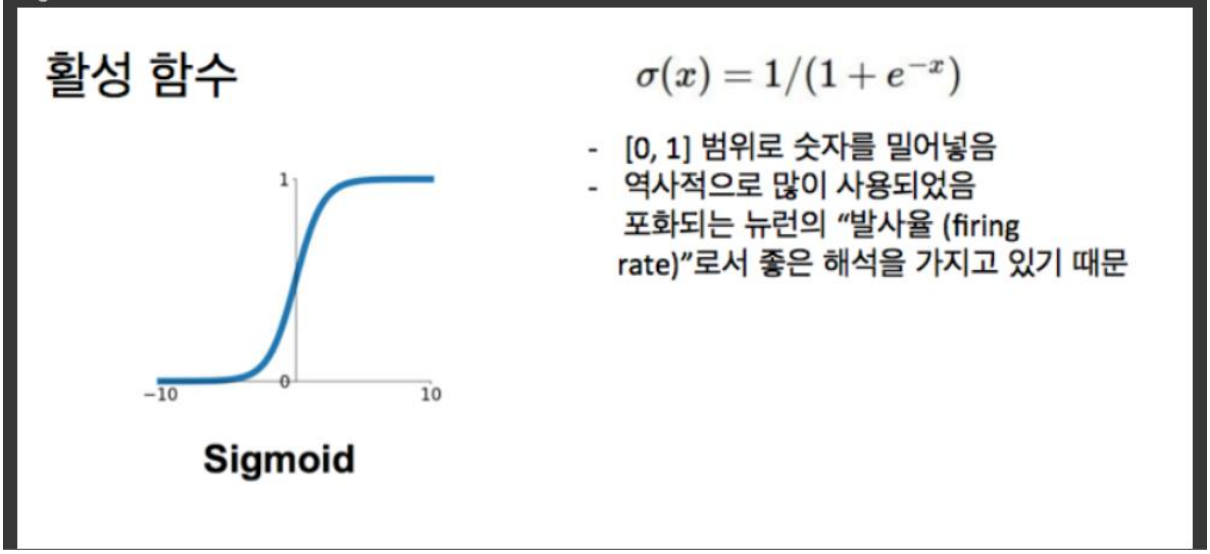
저번 시간 remind
-> 계산 그래프로 x,w값 계산.
CNN -> 합성곱 계층을 거쳐, 새로운 이미지를 얻는다.(이미지는 RGB형태, [32,32,3])

Training Neueral Networks

1. 활성화 함수
2. 데이터 전처리
3. 가중치(Weight Initialization)
4. Bacth Normalization
5. Babysitting the Learning Process
6. 하이퍼파라미터

-활성화 함수.

Sigmoid

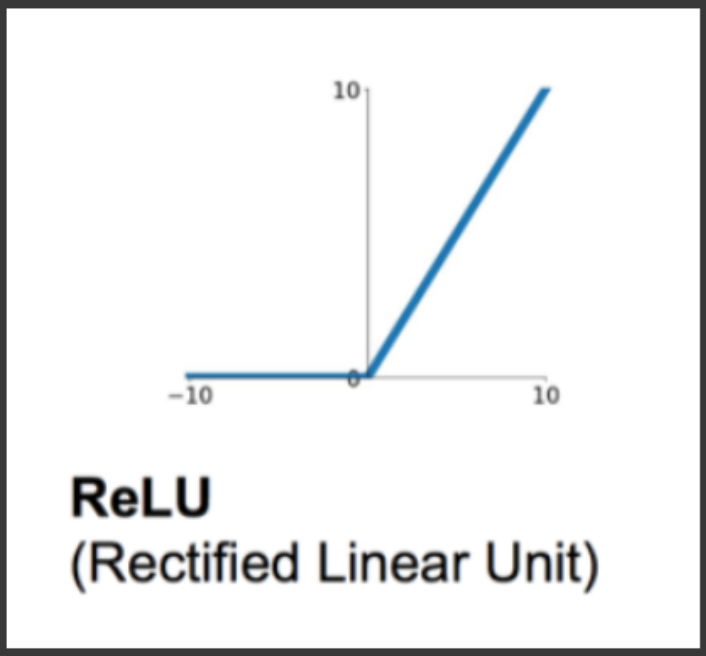


[0,1]사이의 값을 가짐.

-문제점

1. $f\left(\sum_i w_i x_i + b\right)$ 이런식으로 되는데, 계산 해주면, 항상 양수, 음수, 둘중에 하나여서, 기울기를 죽이는 결과.
2. 0 중심이 아니다.
3. e값 계산 때문에, 계산이 어렵다. (비용이 비싸다.)

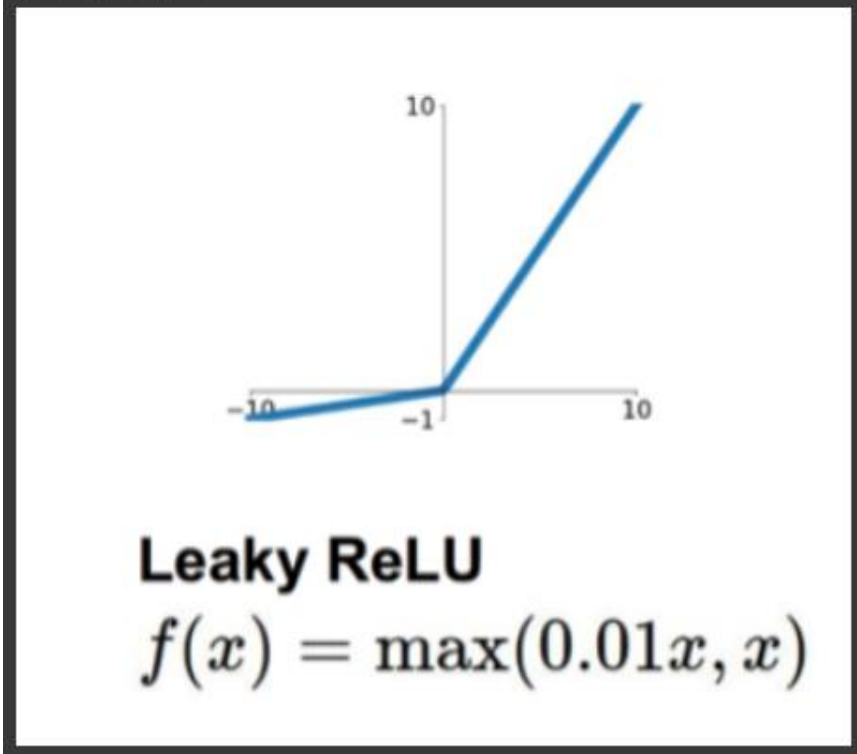
ReLU



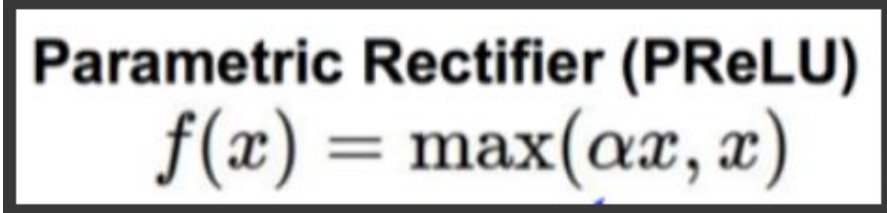
계산식은 f(x) = max(0,x)
양수 영역에서 x축에 평행하게 그려지지 않아(포화되지 않아) 계산할 때 효과적.
sigmoid, tanh보다 훨씬 빠르다.
현실과 더 잘 맞는다.

-문제점
0 중심 출력이 아님.
y축 음수 방향은 고려 못하기 때문에, 데이터를 전부 고려하지 못함.

Leaky ReLU



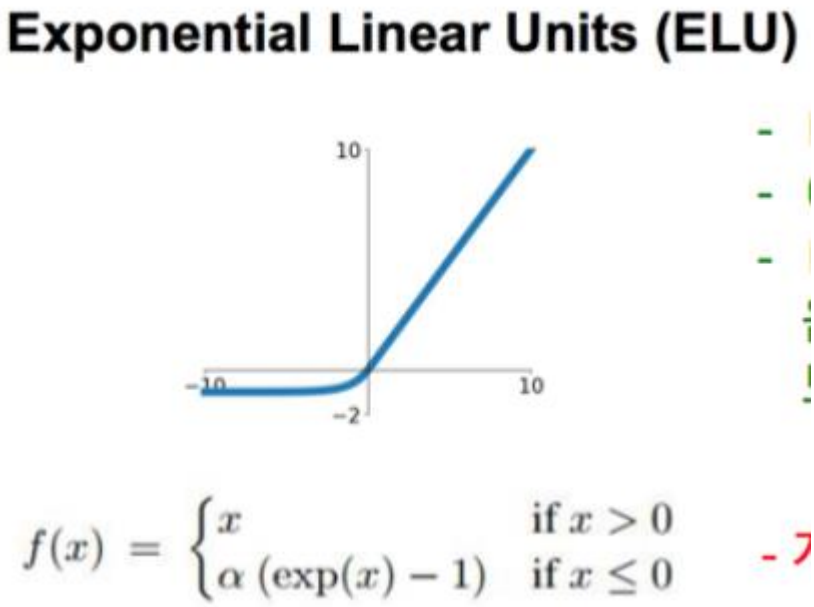
어디 축에 평행하지 않다.(포화되지 않음)
계산할때 효과적이다.
sigmoid, tanh보다 훨씬 빠르다.
죽지 않는다.



좀더 진보된 ReLU

0.01이 아닌,알파 값으로 둬서, 훨씬 더 유연.

ELU



ReLU의 장점만 가져옴

평균 출력 0에 가까움

Leaky ReLU보다 음의 방향쪽에 포화 영역이 있어 좀더 견고.

문제점

exp계산이 들어감.

맥스아웃(Maxout)뉴런

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

기존 함수보다 더 진보된 형태(ReLU, Leaky ReLU)를 섞음

포화되지 않음.

선형적

죽지 않는다.

현실적 조언:

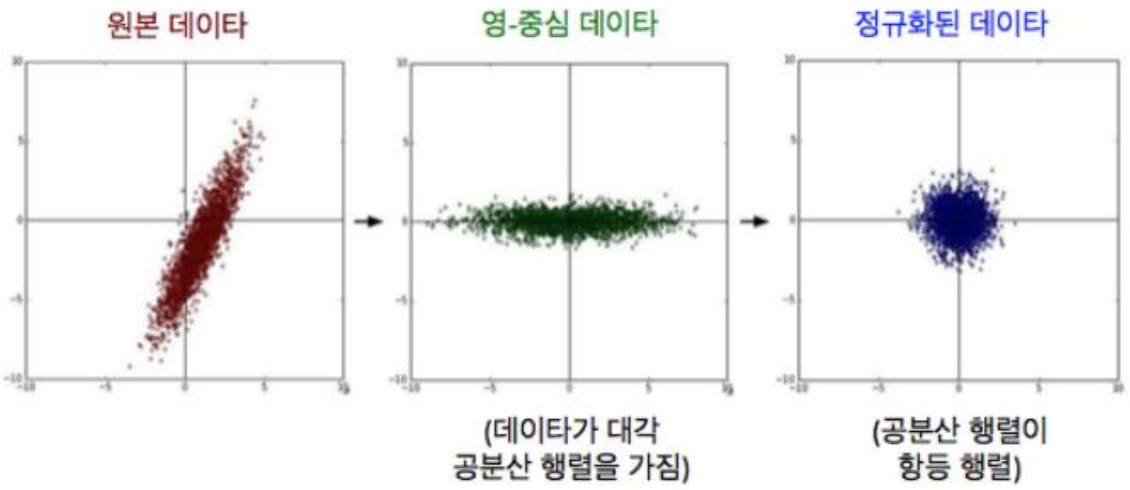
가장 좋은 건 ReLU(보통 많이 씀)

Leaky ReLU, Maxout, ELU, tanh 등 여러 개 실험해볼 것.

Sigmoid는 별로다.

-데이터 전처리.

머신러닝 에서는 주성분 분석(PCA), 화이트닝이 주로 쓰임.



이미지(영상)에서는 PCA, 화이트닝 안쓰임.

데이터 전처리(이미지):

1. 평균 이미지 빼기(AlexNet)

->평균 이미지[32,32,3] 배열

2. 채널당 평균 빼기(VGGNet)

-> 채널마다의 평균 (3개 숫자)

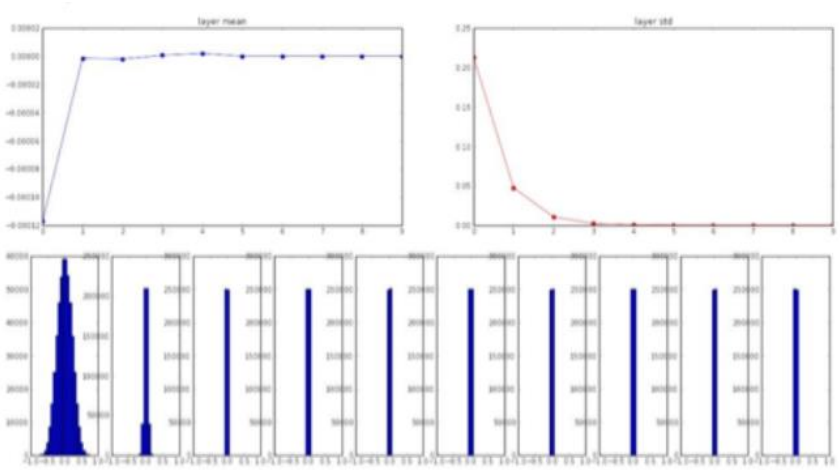
-가중치

가중치 값이 0 이라면, Wx + b를 생각했을 때, 모두 같은 뉴런들이 됨. (이러면 안된다)

```
W = 0.01* np.random.randn(D,H)
```

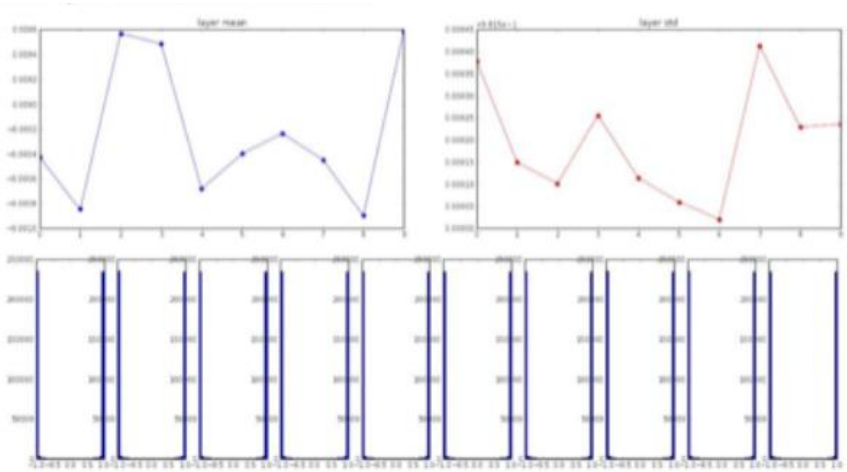
W값 설정. random하기 때문에, 딥러닝에서 문제발생.

난수 생성 0.01값:

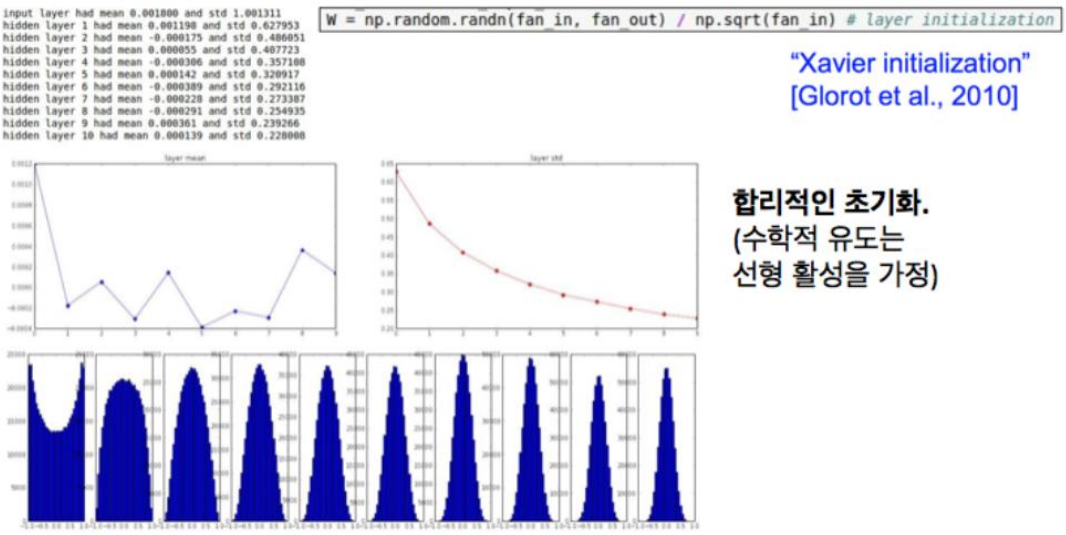


모든 값이 0으로 줄어드는 것 확인.

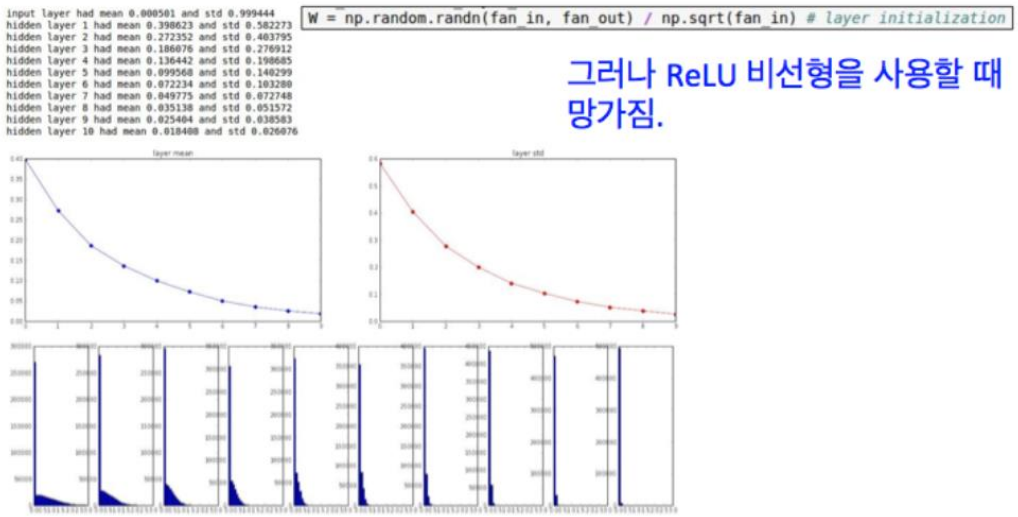
난수 생성 1.0값:



tanh함수를 통과시키면, 포화상태. 결국 모든 경사가 0이 됨.

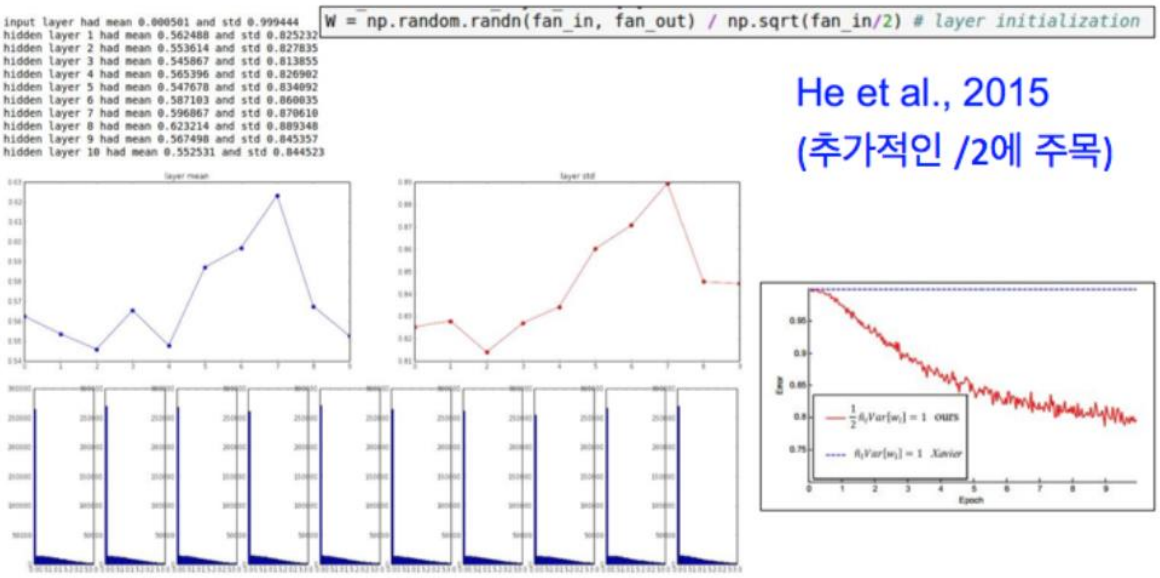


자비에 초기화 방식. 가장 이상적인 W값.



하지만 활성화 함수로 ReLU사용 시, 다 깨짐.

*ReLU함수 사용 시, 음의 값 고려 못하기 때문에.



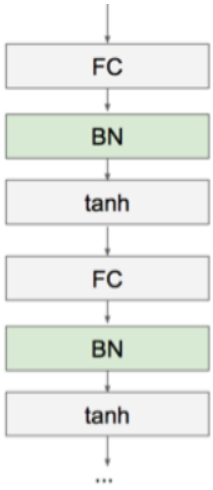
그래서 2로 나누어 보았는데, 상대적으로 잘 작동.

-Batch Normalization(배치 정규화).

계층적 정규분포(가우시안) 활성을 해야한다면,

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

위 바닐라 미분 함수를 적용.



위 함수를 적용하여, 이런식으로 계산.

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$

* 위의 식을 아래와 같이 간략히.

배치 정규화

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

주의: 테스트시에 배치정규화 계층은 다르게 동작함:

평균/표준편차가 배치에 기반해 계산되지 않고, 대신 훈련중의 하나의 고정된 경험적 활성 평균이 사용됨.

(예. 훈련시 이동평균 (running averages)으로 추정될 수 있음)

**

-BabySitting the Learning Process(학습과정 베이비시팅하기).

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0)
print loss
```

정규화 (regularization) 비활성화

```
2.30261216167
```

손실 ~2.3

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e3)
print loss
```

정규화 (regularization) 시작

```
3.06859716482
```

이 두개 값 바꿔가면서 확인.

학습률 1e-3값도 해보고, 1e-6값도 해보고, 손실이 서서히 내려가는 것을 확인해야 함.(하이퍼파라미터 영역)

손실이 내려가지 않음 -> 학습률이 낮음

손실이 급하게 올라감-> 학습률이 너무 높음.

-하이퍼파라미터.

1단계: 에포크 설정

2단계: 더 긴 실행시간, 세부적 탐색

학습률을 높이는 방법: 교차검증, 에포크 값 조절, 실행시간 늘리고, 세부적 샘플링.

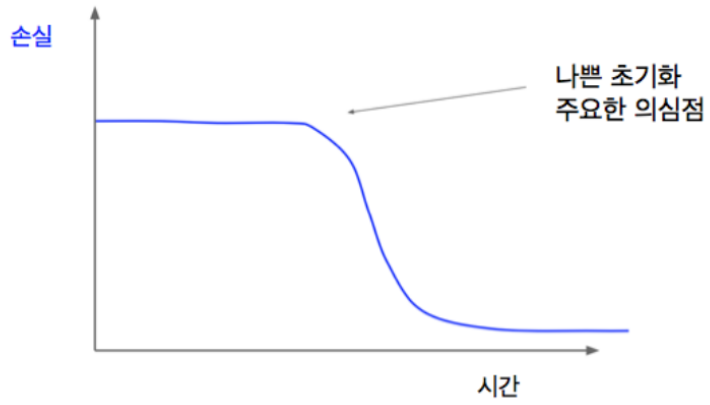
NaN값 감지 팁: 훈련 시작하고, 각 에포크 마다 확인.

만약 값이 이상하다면, 루프 빠져나오기.

랜덤 서치, 그리드 서치 사용.

랜덤 서치: 하이퍼파라미터 값 랜덤하게 넣고, 값 좋은 거 따와서 모델 생성(불필요한 탐색 횟수 감소).

그리드 서치: 순차적으로 입력 후, 가장 높은 성능 보이는 하이퍼파라미터 탐색(시간 오래 걸림).



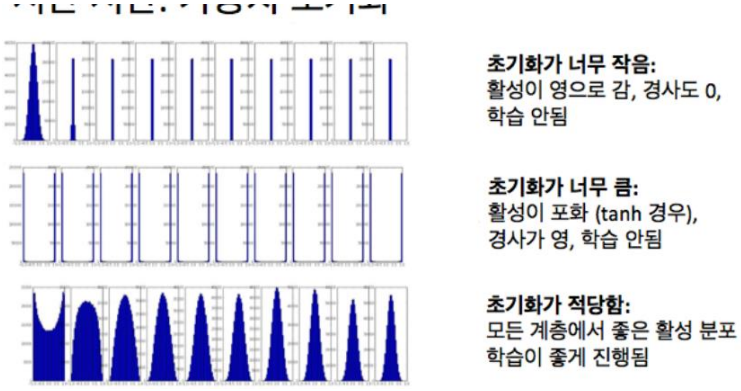
-> 초반에 아무것도 학습되지 않음. 그래서 잘못됐다.

Lecture 7

6강 remind

1. 보통 활성화 함수로 ReLU를 사용.(그냥 일반적으로)

2. W값 초기화(Initualization)



3. 데이터 전처리: 데이터 정규화(normalization)하는 것.

4. batch normalization.

5. babySitting Learning: 학습률에 따른 손실. 정확도에 대한 얘기.

6. 그리드 서치, 랜덤 서치 이야기

-하이퍼파라미터.

1. 최적화(fancier optimization)

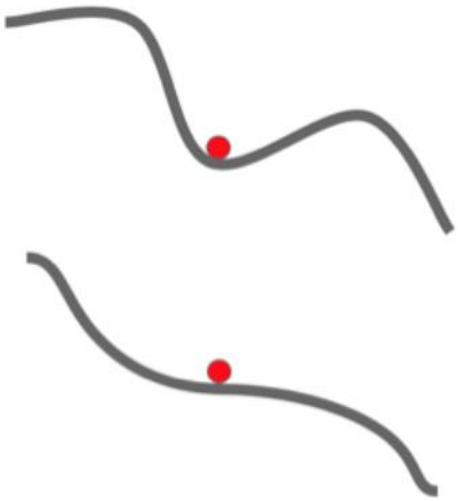
2. 정규화(Regularization)

3. 전이 학습(Transfer Learning)

-최적화(optimization)

SGD(확률적 경사 하강법)

목표 함수까지 가는데, 매우 지그재그 형태. (좋은 성능 안나옴)



문제점:

1.만약 손실함수가 위의 형태를 띄고 있다면(경사가 0인 형태), 멈추게 됨.(기울기가 0이어서)

2.시간도 오래 걸림

SGD + 모멘텀

SGD + 모멘텀 (momentum)

SGD

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

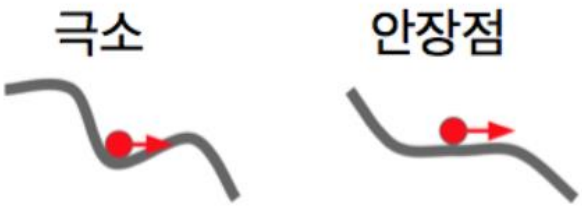
```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

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
```

경사의 방향이 아닌, 속도의 방향으로.



이런식으로, 속도의 방향으로 가기 때문에, 극소, 안장점을 만났을 때, 멈추지 않음.

Nesterov

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

짜증남, 우리는 보통 $x_t, \nabla f(x_t)$
에 대해 업데이트 하고자 함

$\tilde{x}_t = x_t + \rho v_t$ 변수의 변경과 재구성

$$v_{t+1} = \rho v_t - \alpha \nabla f(\tilde{x}_t)$$
$$\tilde{x}_{t+1} = \tilde{x}_t - \rho v_t + (1+\rho) v_{t+1}$$
$$= \tilde{x}_t + v_{t+1} + \rho(v_{t+1} - v_t)$$

```
dx = compute_gradient(x)
old_v = v
v = rho * v - learning_rate * dx
x += -rho * old_v + (1 + rho) * v
```

네스테로프는 현재 속도와 이전 속도 사이 오류 수정하는 항을 포함

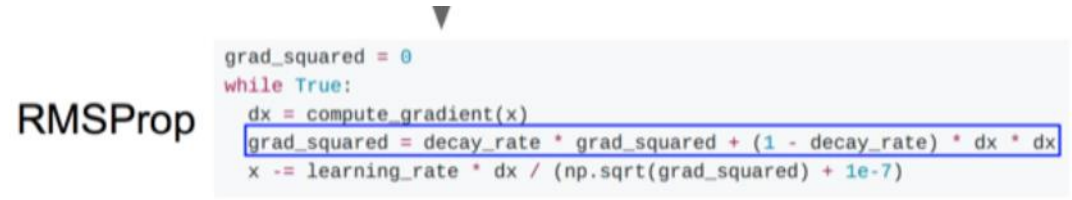
SGD, SGD Momentum보다 속도는 느리지만, 오버슈팅 동작이 최소화 됨.

AdaGrad

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

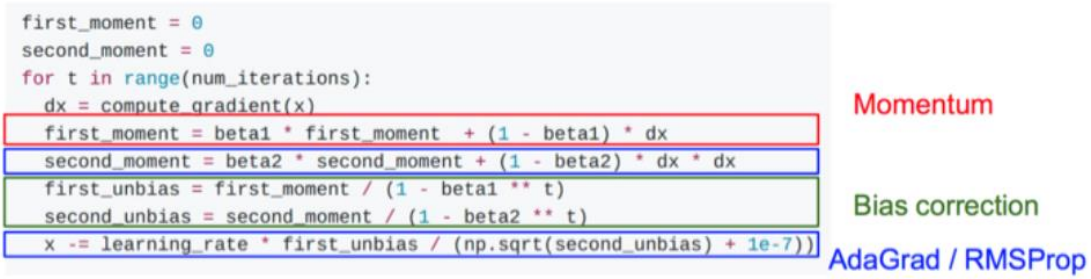
뒤에 나누기 때문에, 만약 grad_squared가 큰 값이라면, 전체 진행이 느려지게 됨.(업데이트 할수록 느려짐)

RMSProp



AdaGrad의 속도 느려지는 부분을 보완.

Adam

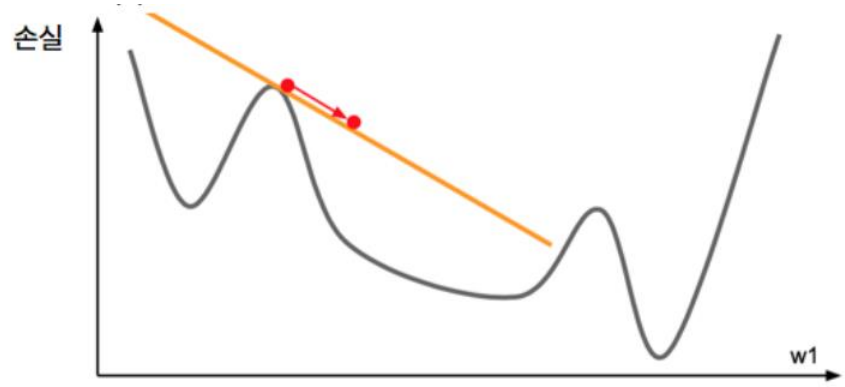


SGD Momentum, Adagrad, RMSProp 합친 형태.

변수 두개 0으로 고정하는 이유가 시작때부터 크게 움직이는 걸 막기 위함.

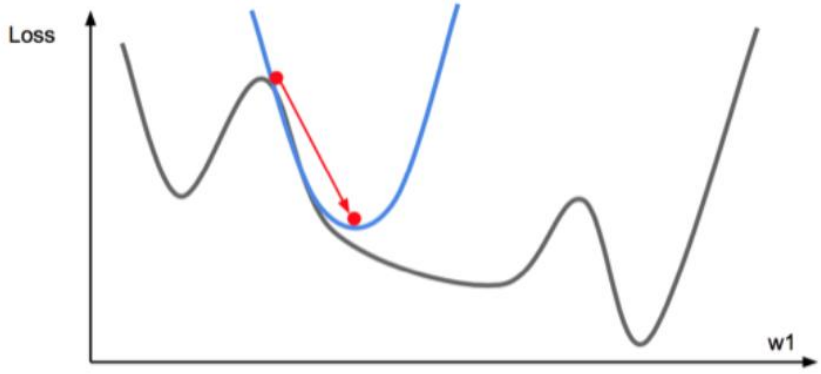
괜찮은 최적화 알고리즘.

1차 최적화



가장 기본적 경사. 미분값.

2차 최적화



$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

뉴턴 스텝(다차원 2차 최적화에 사용.)

학습률 존재 하지 않음.

매 걸음마다 최소의 걸음.

문제점:

O(N^2)원소 가짐. 역행렬에 O(N^3) 걸림. ->딥러닝에 안좋음.

BFGS(제한 조건 없는 함수에서 x를 최소화 시키는 것) : 뉴턴 방법 사용. 역행렬 만드는 대신, 1로 업데이트 해서.

L-BFGS(제한된 BFGS): 역행렬 전체 생성 혹은 저장 안함.

full batch에서는 적합. mini-batch에서는 안 좋음.

보통 Adam이 가장 일반적. full-batch 할 수 있음 L-BFGS쓸 것.

모델 앙상블(Model Ensembles):

하나의 모델을, 여러 개로 조화롭게 학습(여러 모델의 예측값의 평균).

훈련 과정에서 여러 snapshot 갖고 있다가 앙상블로 활용.

```
while True:
  data_batch = dataset.sample_data_batch()
  loss = network.forward(data_batch)
  dx = network.backward()
  x += - learning_rate * dx
  x_test = 0.995*x_test + 0.005*x # use for test set
```

훈련하는 동안, 계속 앙상블 갖는 것.

-정규화.

드롭아웃(Dropout):

forward pass 할 때, 임의의 뉴런들을 0으로 설정. (계층 몇 개 뺀다)

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

드롭아웃 사용 이유:

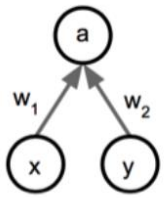
1.과적합 해결에 탁월.

2. 모델 내부에서 모델 앙상블을 하고 있음.

드랍아웃: 테스트시

적분을 근사하고 싶음 $$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$

단일 뉴런을 생각해 보기.



테스트시 갖고 있는 것: $E[a] = w_1 x + w_2 y$

훈련동안 갖고 있는 것:

$$\begin{aligned} E[a] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

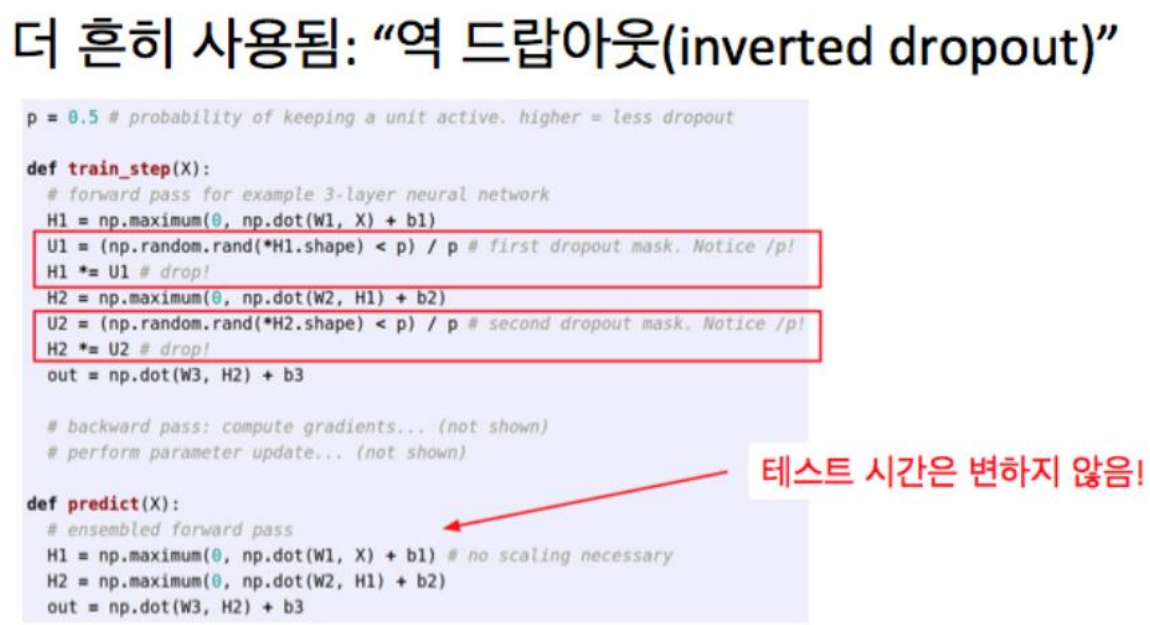
테스트 시 출력 = 훈련시 기대 출력. 이 두개가 같도록 해야 함.

드롭아웃:

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

테스트시 배율 조정 (scale)

역 드랍아웃



드롭아웃에서 predict할 때, '*p'가 되어있는 것을 확인 가능.

시간 절약을 위해, 역 드랍아웃 train때, '/ p' 해줌.

배치 정규화(Batch Normalization):

임의의 미니배치로 정규화

노이즈를 갖는다는 것만 보면 드롭아웃과 비슷.

파라미터값 제어는 불가능.

데이터 증강(data augmentation):

CNN에서 훈련하는 동안 이미지 변형.

ex) 1.고양이 사진 좌우 반전. 구역별 자르고 그걸 모아다가 평균.

2. 밝기 변경. 각 색들을 샘플링. 이걸 다 더하기. -> 좀 어려워서 잘 안 씀

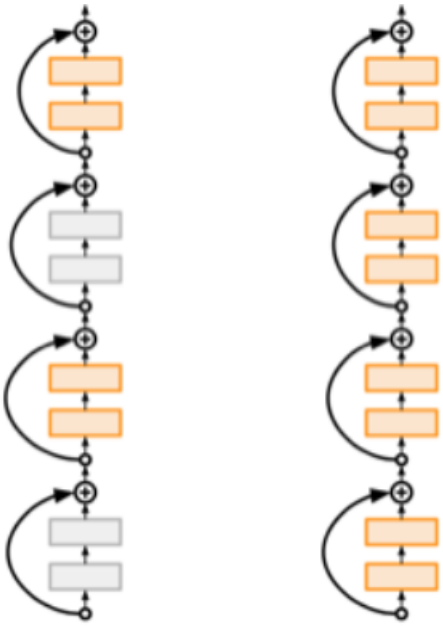
드랍커넥트(DropConnect):

드랍아웃처럼 활성을 0값 말고, 행렬 값 일부를 0으로.

Fractional Max Pooling(작은 맥스 풀링):

필터들을 더 작은 필터들로. (ex. 2x2 풀링을 임의로 1x1, 1x2, 2x1, 2x2로 설정해서 이걸 조합해 풀링)

확률적 깊이(stochastic depth):



훈련중에 계층을 임의로 드랍 하고, 테스트 할 때는 전체 계층 사용.

알려주는 사람이 극찬.

-전이학습(Transfor Learning).

학습된 모델을 다른 작업에 이용하는 것. CNN에 적용.

	매우 비슷한 데이타셋	매우 다른 데이타셋
매우 적은 데이타	제일 위 계층에서 선형분류기 사용	곤란한 상황에 처함... 여러 단계로부터 선형 분류기 시도 할 것
꽤 많은 데이타	몇개의 계층을 미세조정	더 많은 계층을 미세조정할 것

이미지 처리에 많이 사용.