Lecture5

neural network의 역사:

1957년 퍼셉트론(역전파 불가) ➔ 아다린,마다린(퍼셉트론 쌓은 것, 동일하게 역전파 불가)

➔ 루멀하트 논문(역전파, 문제가 쫌 많아서 암흑기) ➔ 2006년(역전파, 제대로 동작하는) ➔ 연구 활성화(딥러닝이라는 말이 나옴) ➔ 2010년대 초반, 딥러닝 폭발적 발전.

오늘날, CNN은 많은 곳에 사용된다. ex) 영상 분리, 자율주행(GPU에 의해 가능), 얼굴인식, 자세인식, 이미지 캡셔닝(이미지에 대한 문장 기술), 딥드림(요상한 그림)

그냥 대충 이런 흐름으로 발전해 왔다 정도로만.

Fully Connected Layer(완전 연결 계층):

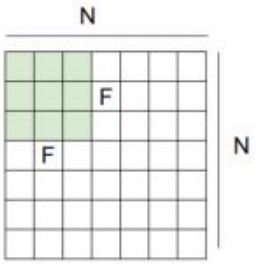
input -> W(가중치) -> activation

ex) 32 x 32 x 3의 이미지 -> 3027 x 1 배열.

Convolution Layer(합성곱 계층):

➔ 위와 다르게, 일렬로 늘리는 대신, 3차원 구조 유지(중간에 필터 껴넣는 형태).

CNN은 Convolution layer(합성곱 계층)의 결과, activation function(활성함수. ex ReLU)가 중간중간 껴있다.



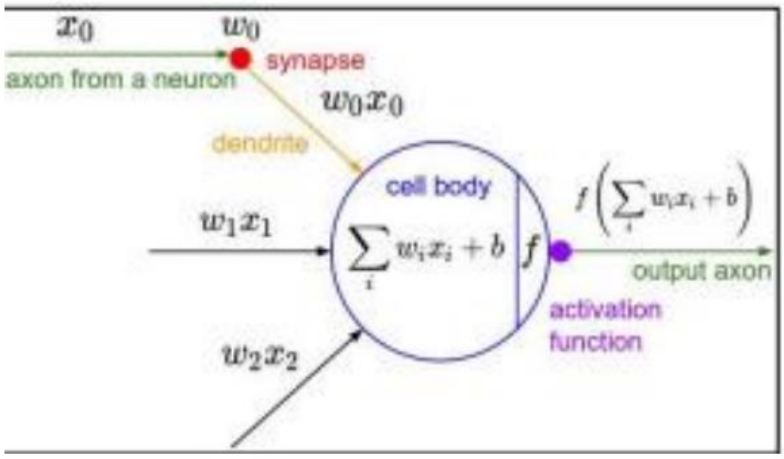
Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

➔ spartial dimension(공간 차원)에 맞춰서, stride 크기를 고려해야 한다.

실제 에서는 0으로 패딩(zero pad).

#그냥 Pytorch나 tensorflow로 CNN 계층 쌓을 때 그거 생각하면 될 듯.



여기서 파란 똥글뱅이는 local connectivity(지역 연결, 전체적인 연결 말고 단순 공간적 연결)

그리고 5x5 의 필터를 receptive field(수용장)라고도 부름.

Pooling layer(풀링 계층):

다운 샘플링을 위해서(더 작고, 관리하기 쉽게 하기 위함)

흔히 사용하는 Max Pooling.

➔ 풀링 계층에서는 다운샘플링을 위해서 0패딩을 사용하지 않는다.

Lecture 6

저번시간 remind

➔ 계산 그래프로 x,w 값 계산.

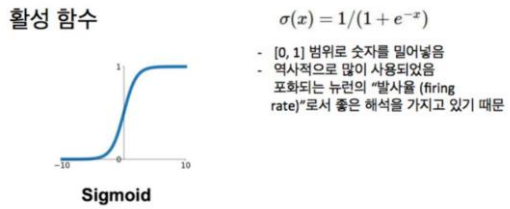
CNN → 합성곱 계층을 거쳐, 새로운 이미지를 얻는다.(이미지는 RGB 형태, [32,32,3])

Training Neural Networks

1. 활성화 함수
2. 데이터 전처리
3. 가중치(Weight initialization)
4. Batch Normalization
5. Babysitting the learning Process
6. Hyperparameters

-activation function-

Sigmoid



[0,1] 사이 값을 가진다는 특징.

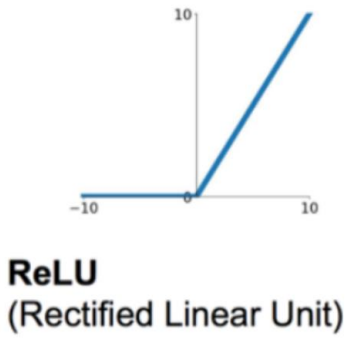
-문제점

1. $f\left(\sum_i w_i x_i + b\right)$ 이런식으로 되는데, 계산 해주면 항상 양수, 음수 둘중에 하나라는 특징. 그래서 기울기 죽이는 결과.

2. 0 중심이 아님.

3. e값 계산 때문에, 계산량 더러워짐.

ReLU



계산식은 f(x) = max(0, x)

양수 영역에서 x축에 평행하게 그려지지 않아(포화되지 않아) 계산할 때 효과적.

sigmoid, tanh보다 훨씬 빠름.

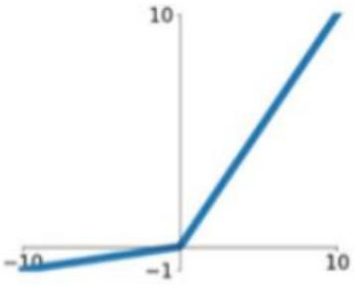
자연계 데이터 형태와 더 잘 들어맞음

-문제점

0 중심 출력 아님.

y 축 음수 방향은 고려 못하기 때문에, 데이터 전체를 고려하지 못함.

Leaky ReLU



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

어느 축에도 평행하지 않다는 특징(포화되지 않음)

계산할 때 효과적이다.

sigmoid, tanh보다 훨씬 빠르다.

죽지 않는다.

PReLU

Parametric Rectifier (PReLU)

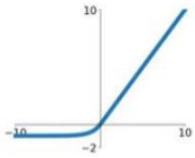
$$f(x) = \max(\alpha x, x)$$

좀더 진보된 ReLU

0.01이 아닌, 알파 값으로 둬서, 훨씬 더 유연.

ELU

Exponential Linear Units (ELU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

ReLU의 장점만 갖고왔다.

평균 출력 0에 가까움.

Leaky ReLU보다 음의 방향 쪽에 포화 영역이 있어 좀더 견고.

-문제점

e 계산이 들어감

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.

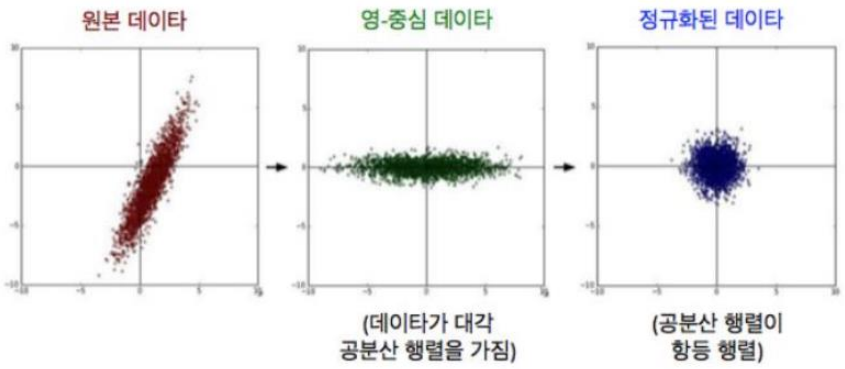
현실적 조언:

가장 좋은건 ReLU(그냥 많이 써서)

활성화 함수는 그냥 여러 개 실험해볼 것.(노가다 느낌으로)

-데이터 전처리

머신러닝에서는 주성분 분석(PCA), 화이트닝이 주로 쓰임



이미지에서는 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)

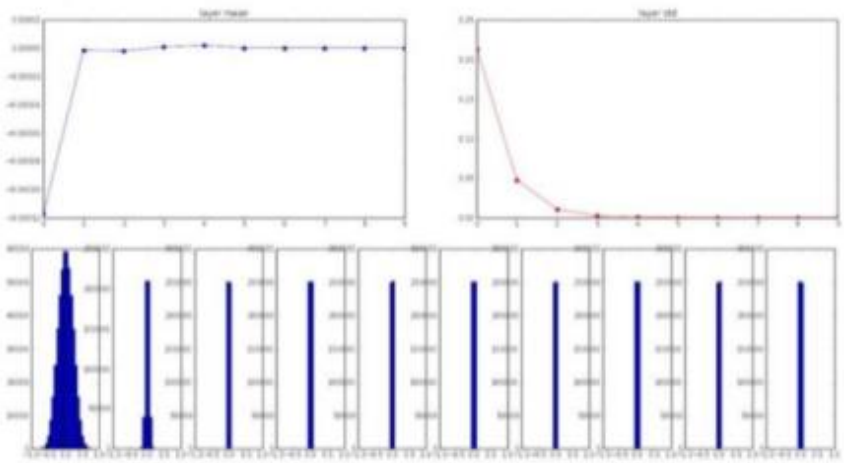
-가중치

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

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

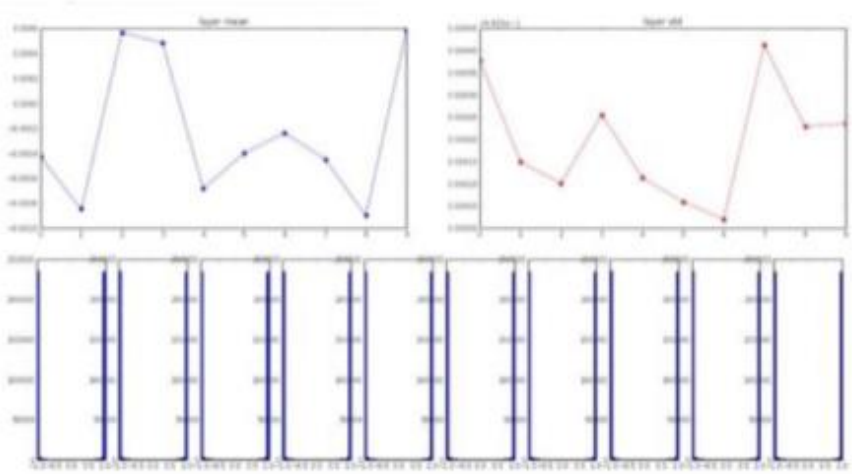
W값 설정. random하기 때문에, 딥러닝에서 문제발생.

난수 생성 0.01값:

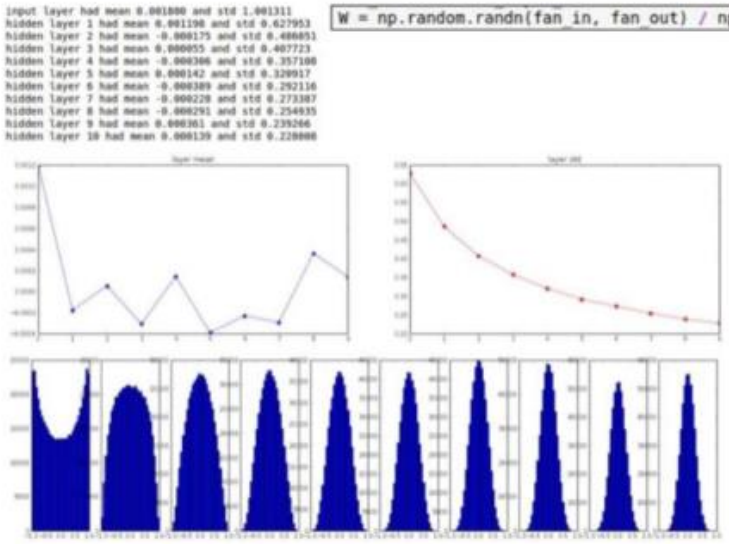


모든 값이 0으로 줄어드는 것 확인.

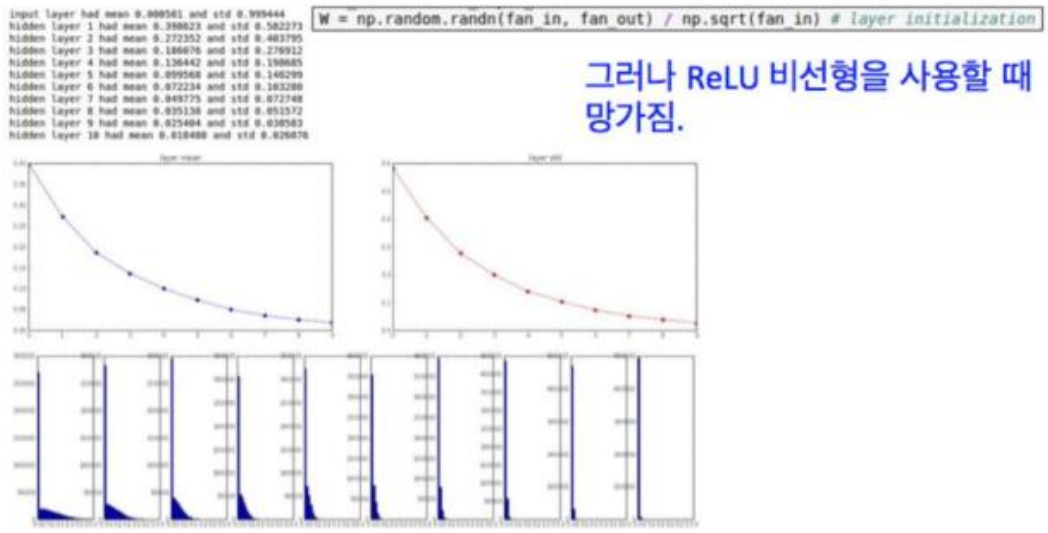
난수 생성 1.0값:



tanh함수를 통과시키면, 포화상태. 결국 모든 경사가 0이 됨.

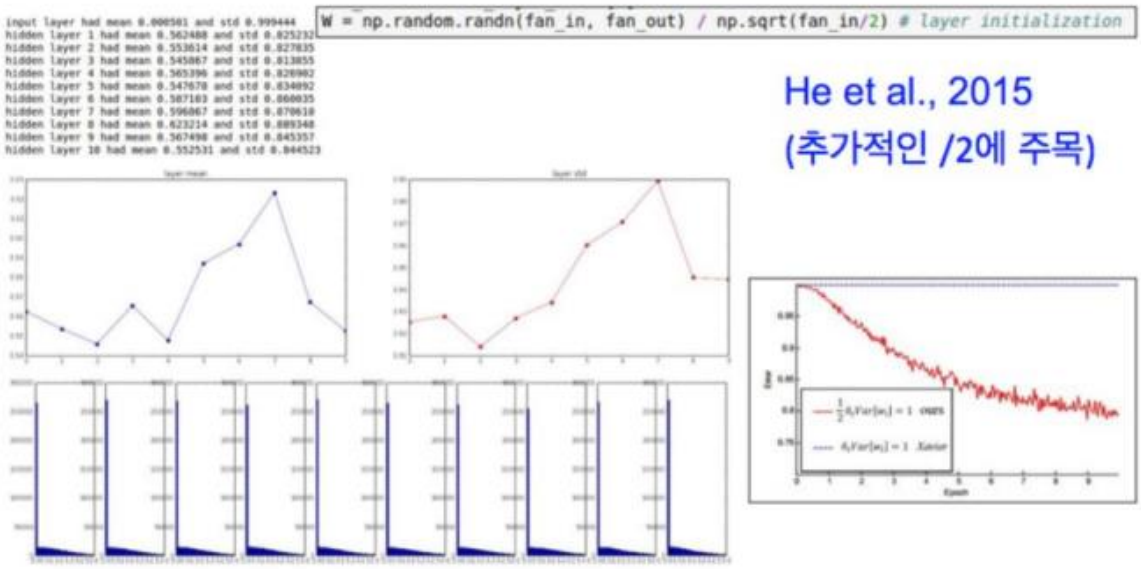


자비에 초기화 방식. 가장 이상적인 W값.



하지만 활성화 함수로 ReLU사용 시, 다 깨짐.

*ReLU함수 사용 시, 음의 값 고려 못하기 때문에.



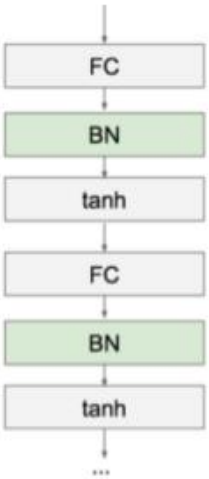
그래서 2로 나누어 보았는데, 상대적으로 잘 작동.

-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: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

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.

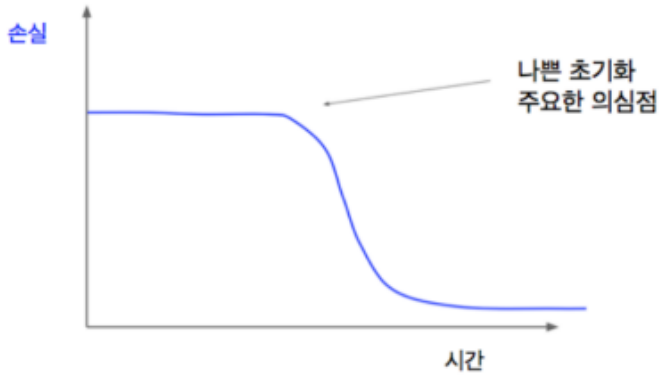
NaN값 감지 팁: 훈련 시작하고, 각 에포크 마다 확인.

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.