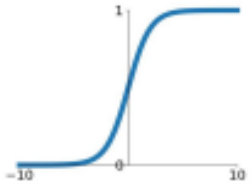
CS231N 6강
Training Neural Networks

Activation Functions(활성화 함수)

:딥러닝에서 비선형성을 가해주는 역할을 함.

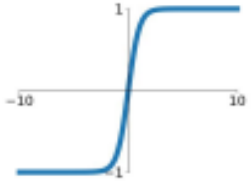
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



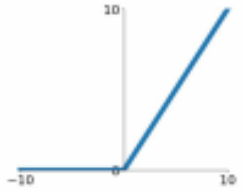
tanh

$$\tanh(x)$$



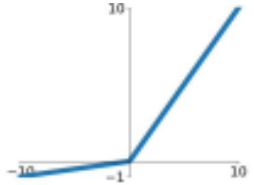
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

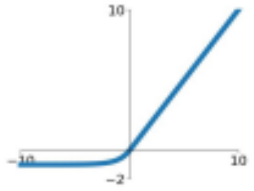


Maxout

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

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Use ReLU. Be careful with your learning rates
Try out Leaky ReLU / Maxout / ELU
Try out tanh but don't expect much
Don't use sigmoid

Sigmoid

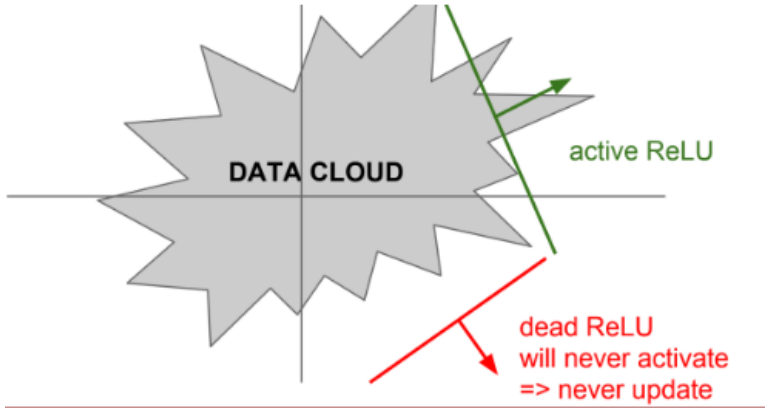
단점: 1)vanishing gradient 2)zero-center 3)exp의 연산 으로
activation 함수로 이용하지 않고 최종 output 낼때만 이용

ReLU

:기울기 소실 문제 해결, exp 연산 사라짐
But 입력값의 음의 부분은 전부 0으로 두어 vanishing gradient,zero-centered
문제가 여전히 있음

Dead ReLu
: reLu를 활성화 함수로 하는 모델에서
훈련 시 freeze 되는데 dead reLu에 빠진것

Leaky ReLu
: Dead ReLu 단점 보완(vanishing gradient 해결)



Maxout Neuron

: parameter을 더 두어서 출력 갑이 다른 2개의 함수 중에 max 값 선택하는 함수, 연산량 두배

Maxout "Neuron" [Goodfellow et al., 2013]

- Does not have the basic form of dot product -> nonlinearity
- Generalizes ReLU and Leaky ReLU
- Linear Regime! Does not saturate! Does not die!

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

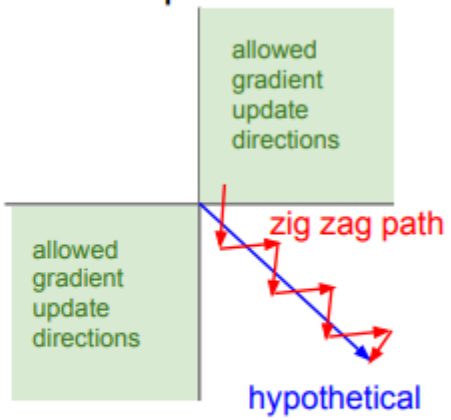
Problem: doubles the number of parameters/neuron :(

Data processing (데이터 전처리)

1)zero-centering
: 이미지 데이터는 픽셀 값이 같은 범위여서 zero-centering만 함, 입력값이 모두 양수이면 w가 전부 음수나 양수가 되어 업데이트가 일방향으로 일어남 -> zero-mean

Remember: Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?
Always all positive or all negative :(
(this is also why you want zero-mean data!)

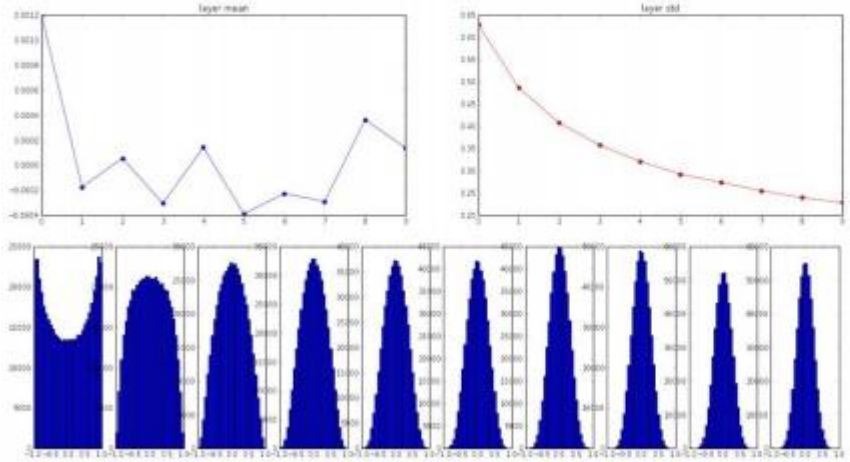
Weight Initialization

- Weight가 모두 0이면 입력값에서부터 모든 뉴런이 같은 값만을 출력함(layer 의미x)
- 맨 처음의 weight에 임의의 작은 값 부여하기: layer이 깊어질수록 0으로 수렴함 , update 잘 안됨
- Initialization이 w이 작으면 collapse 되고, 너무 크게 되면 saturate 됨
- **Xavier initialization**

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228088
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

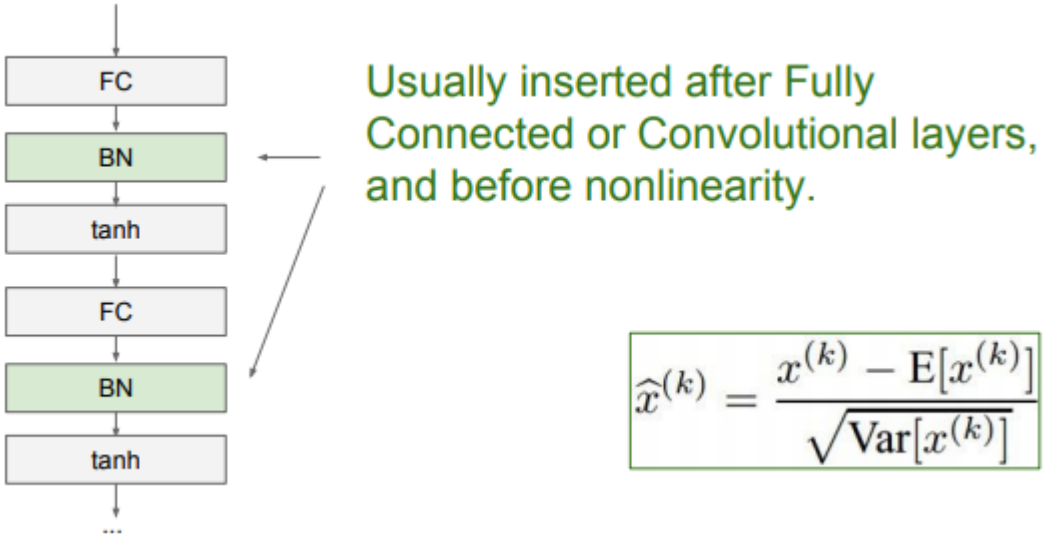
"Xavier initialization"
[Glorot et al., 2010]



Reasonable initialization.
(Mathematical derivation assumes linear activations)

Batch normalization

- Batch normalization을 하면 weight normalization 안해도 됨
- Training과정에서 gradient vanishing 문제를 일어나지 않게 함
- 각 층의 input distribution을 평균 0/표준편차 1로 만드는 것
- Activation layer 전에 잘 분포되도록 함



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

Learning process

Preprocess the data -> choose the architecture -> loss check -> train with few examples -> **hyperparameter select**

Hyperparameter Optimization

- 넓은 범위 -> 좁은 범위
- Grid Search<Random Search
- Network architecture,learning rate, regularization 등이 있음
- Validation의 acc 값과 training acc 값 차이 크면 과적합 일어나서 조심해야함