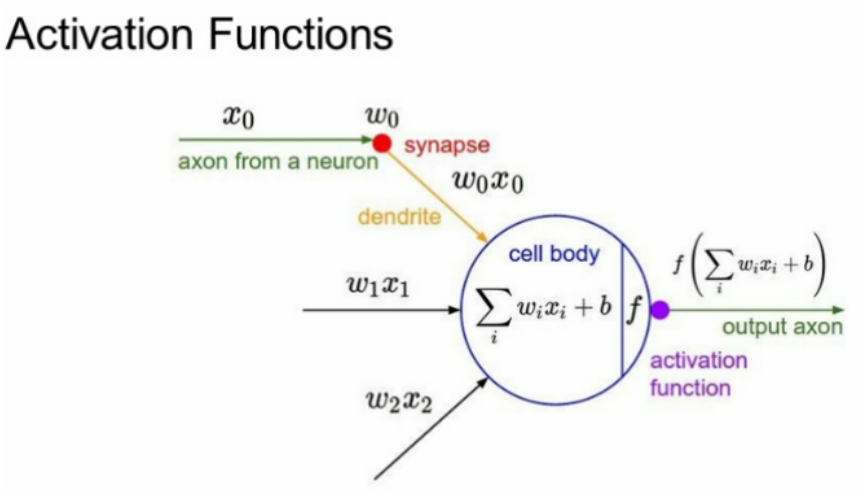
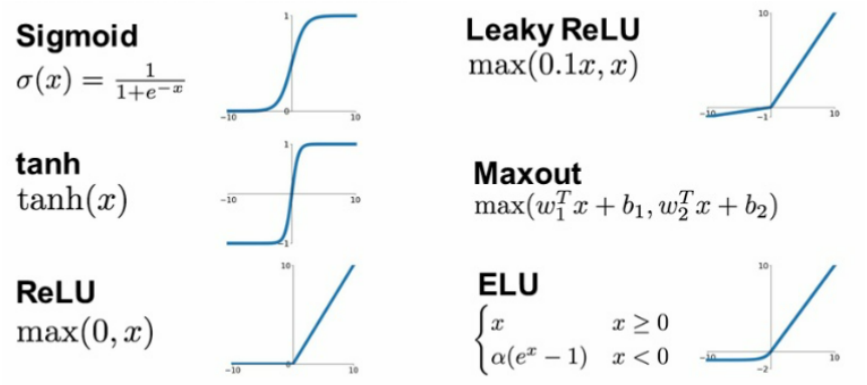
# Week6_Training Neural Networks 1

## Activation Functions




- Activation Function은 f = Wx 다음에 input값이 들어오면 이 값을 다음 노드로 보낼 때 어떻게 해서 보낼지를 결정한다.
- Non-linear function 사용⇒ 여러 layer를 1개의 layer처럼 취급할 수 있게됨
- 어떤 값을 활성화해서 다음 layer로 전달할지를 결정하는 것 ⇒ 활성화함수




### Activation Functions

1. Sigmoid Function
    a. 과거에 많이 사용
    b. 단일, 이진 분류에 많이 사용, 최종 출력에서만 사용
    c. hidden layer에는 사용하지 않음

**문제점**

1. gradient vanishing (Saturated neurons “kills” the gradients)
    a. 계속 backpropagation을 하게 되면 0과 가까운 값이 곱해져 0으로 기울기가 소실됨
2. Sigmoid outputs are not zero-centered
    a. 시그모이드 그래프는 0을 중심으로 되어 있지 않음 ⇒ 작은 변화에도 큰 변화 일어날 수 있음
3. exp() is a bit compute expensive
    a. exp 연산은 어려운 연산 ⇒ sigmoid는 잘 사용하지 않는다.

2. tanh() 하이퍼볼릭 탄젠트
    a. Zero centered

⇒ 여전히 gradient vanishing과 exp 연산 문제 존재

3. ReLU (Rectified Linear Unit)
    a. 가장 대중적으로 사용하는 activation function
    b. 계산 효율이 높음
    c. 수렴이 sigmoid 나 tanh 보다 빨리 일어남

**문제점**

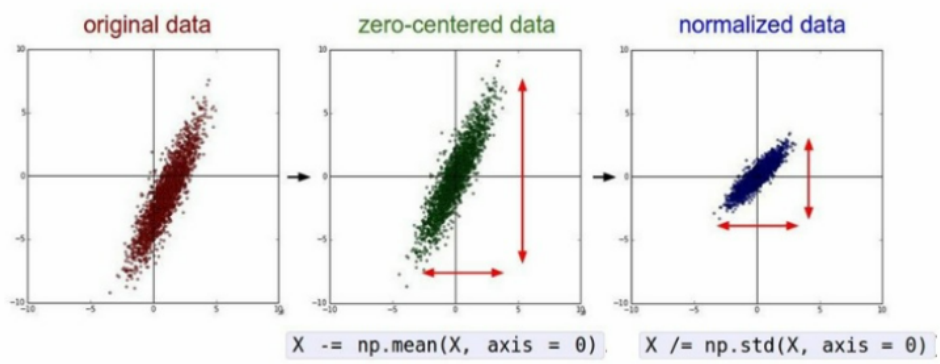
1. 0 이하 값을 아예 버림
2. zero centered 되어있지 않는다.

⇒ dead ReLU 문제

4. Leaky ReLU (ReLU의 보완)
    a. 0 이하의 값에 0.01x 값을 줘서 dead ReLU문제를 해결하려함
    b. PReLU : f(x) = max(ax , x)
5. ELU(Exponential Linear Units)
    a. ReLU의 모든 장점을 가지고 있음
    b. exp() 계산을 해야함
6. Maxout
    a. Max()값을 통해 2개의 파라미터를 줘서 더 좋은 파라미터를 선택하도록

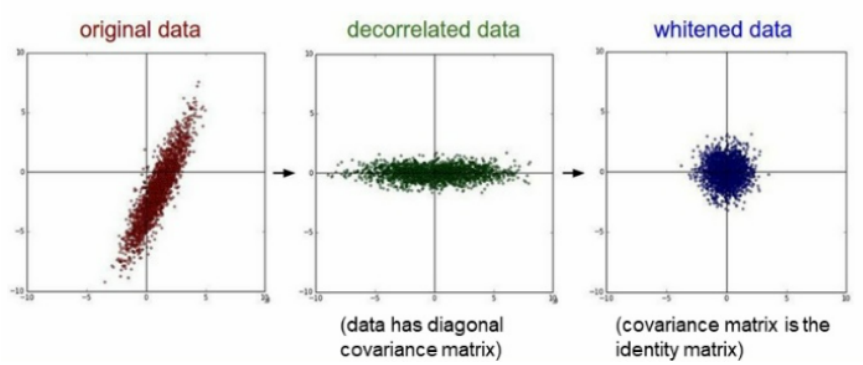
⇒ 연산량이 2개가 되는 문제가 존재

# Data Preprocessing




zero-centered, normalized를 주로 사용한다.

normalizaed의 경우 표준편차로 값을 나눠주어 특정 범위 안에 값들을 모아 놓은것 ⇒ 이미지는 0~255로 범위가 정해져 있기에 굳이 사용하진 않는다.




공분산에 따라 차원을 감소시켜주는 PCA와 whitened data는 이미지에 잘 사용하지 않는다.

⇒ 이미지에서는 zero-centered를 사용

# Weight Initialization

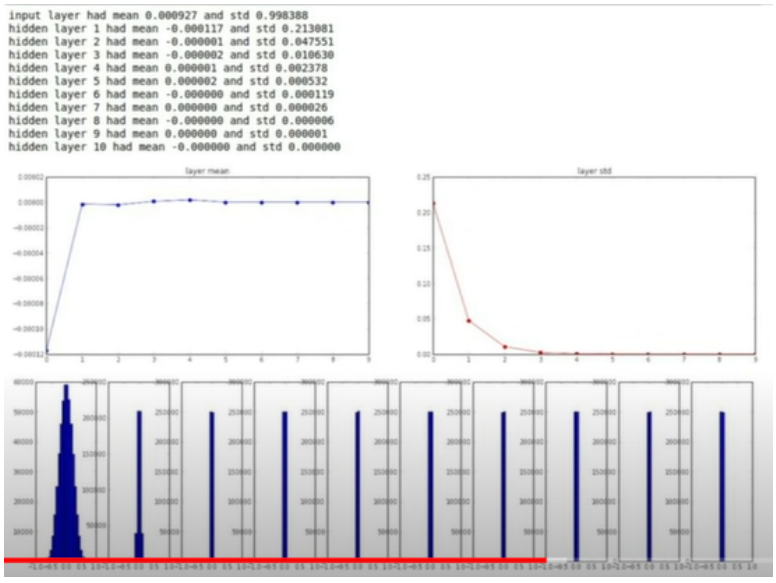
Q : 만약 W=0 일 경우 ⇒ Gradient Vanishing 발생

1. First idea : 작은 랜덤값을 대입

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

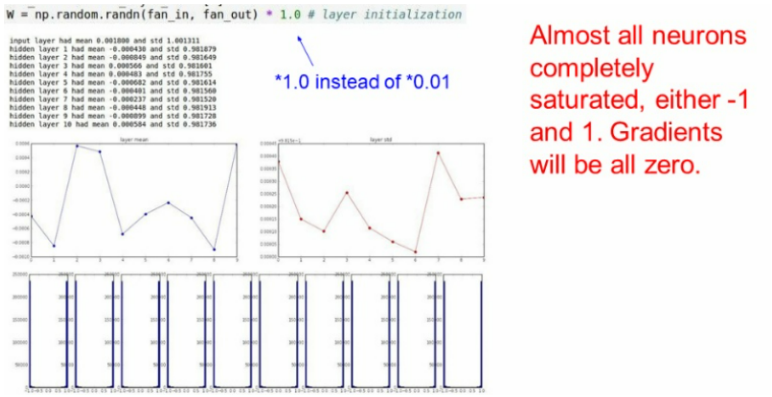
   작은 network에서는 괜찮지만, network가 깊어질수록 문제가 발생

10-layer, 500 neurons on each layer, tanh activation function 인 예시를 보자.

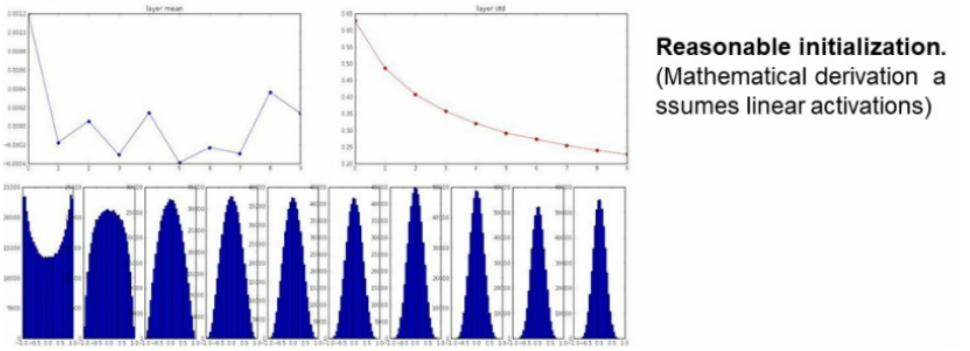


레이어가 깊어질수록 값들이 전부 날라간다.

tanh 그래프에서 기울기가 0인 부분은 vanish되고 0근처의 기울기아 0이 안되는 지점만 살아남는다.




W값을 크게 할 경우 -1과 1에 값이 포화된다.




⇒ Xavier initialization

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

fan_in(노드의 갯수) 를 normalization 한다. ⇒ input의 갯수에 따른 weight값이 정해져서 합리적으로 weight를 초기화한다.

⇒ tanh 에는 잘 적용되나 ReLU함수에는 잘 적용되지 않는다.

⇒ `W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2)`

/2를 추가시키니 잘 동작되었다.

# Batch Normalization

Gradient Vanishing 문제가 나오지 않도록 하는 아이디어.

## Batch Normalization

[Ioffe and Szegedy, 2015]

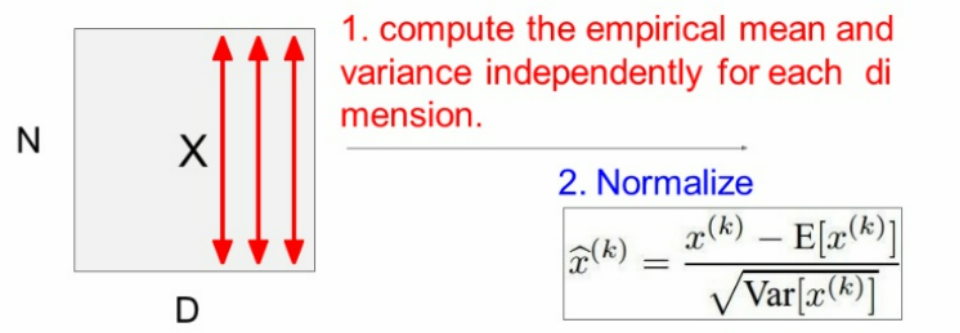
"you want unit gaussian activations? just make them so."

consider a batch of activations at some layer.
To make each dimension unit gaussian, apply:

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

this is a vanilla differentiable function...

internal covariance shift 를 방지한다. 각 층의 input distribution을 평균0, 표준편차 1 인 분포로 만든다.




보통 연산을 할 때 N * D 의 batch 로 나누어 처리하는데, 이때 batch가 들어오면 normalization을 한다.

보통 activation 이전에 BN 진행

FC → BN → activation 순으로 들어간다.

Q. BN의 필요성에 대한 판단은 어떻게 하는지?

⇒ 판단이 학습 가능하다.

Normalize:

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

And then allow the network to squash the range if it wants to:

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

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

위 식으로 BN을 do, undo 할지를 선택한다.

# Babysitting the Learning Process

1. Preprocess the data(보통 zero-centered data)
2. Choose the architecture
3. Double chek that the loss is reasonable + sanity check
4. Training ⇒ 작은 데이터셋을 우선 넣는다.
    a. train accuracy = 1 이 나오면 overfitting이 되어 나오고 모델이 잘 동작하고 있다는 의미
5. Regularization과 learning rate찾기
6. Hyperparameter Optimization
    a. coarse → fine 넓은 범위에서 좁은 범위로