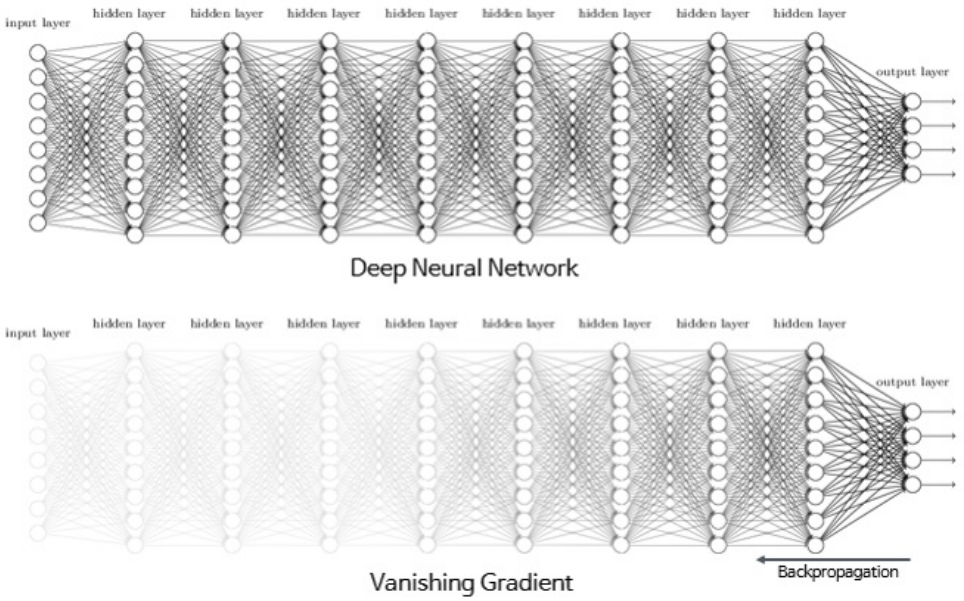
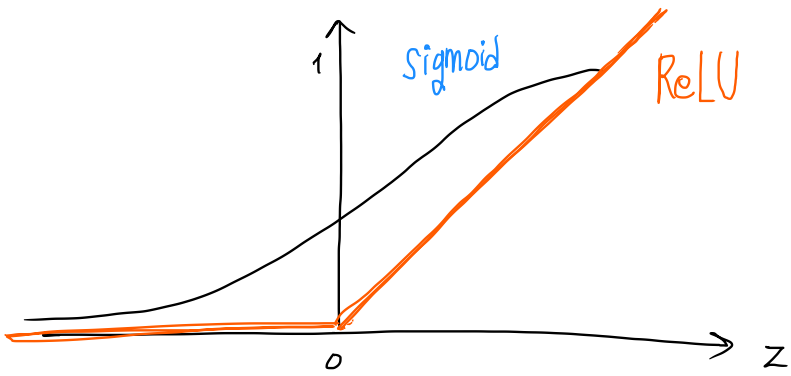
ReLU: Rectified Linear Unit

- back propagation 알고리즘은 2-3개의 레이어에서는 동작이 잘 되지만 더 deep 해지면 **결과가 잘 나오지 않는 문제가** 존재했다.

- **Vanishing gradient**: 미분값이 연속적으로 곱해지면서 값이 **0과** 가까워져버려 기울기를 구하지 못하는 현상

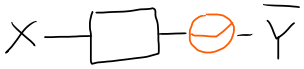


- 그래서 등장한 것이 **ReLU**! ReLU 는 값이 **0과 가까우면 0으로** 만들어 버리고 **0보다 크면 값을 키워준다**



- NN에서 Sigmoid 대신 relu 를 사용하면 된다.

L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

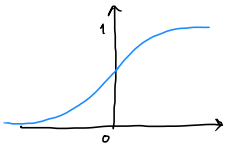


L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

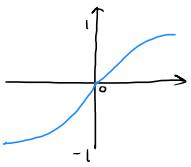
다른 알고리즘들

〈 Sigmoid 〉 → 〈tanh〉

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



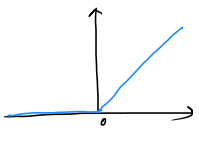
$\tanh(x)$



중심점을 0으로 옮김

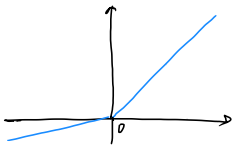
〈 ReLU 〉 → 〈 Leaky ReLU 〉

$\max(0, x)$



0보다 작으면 무조건 0

$\max(0.1x, x)$



0말고 0보다 더 작은 값주자

→ 〈ELU〉

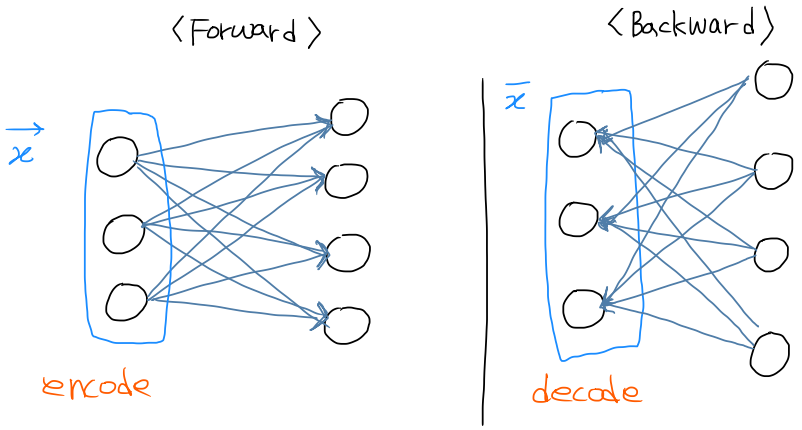
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

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

Weight 초기값을 잘 주는 방법

- 초기값을 0으로 주면? 미분할때 0이 되서 **값이 사라져버림**!! **절대 초기값을 0으로 주면 안됨!!**
- **RBM (Restricted Boltzmann Machine)** 방법을 사용하자

RBM



- encode 값과 decode 값을 비교해서 최대한 같은 값이 나오도록 weight 를 조정한다.
- 꼭 RBM 써야 하나? 그건 아님

fan_in / fan_out

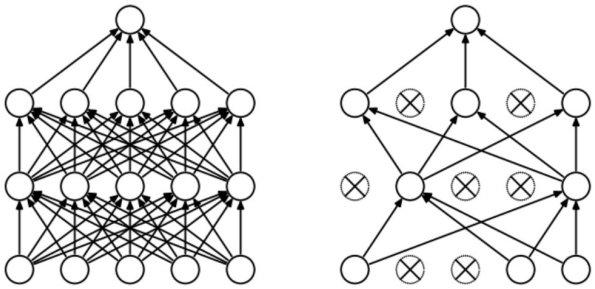
- RBM이 아닌 단순한 랜덤값을 넣는 방법

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

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

Dropout

- **over fitting 된 model을** 사용하면 학습데이터에는 잘 돌아가다가 처음 접하는 데이터는 잘 못돌리게 된다!
- 학습데이터의 양을 늘리거나 **Regularization (일반화)** 를 통해 해결할 수 있었다.
- 위의 두가지 방법과는 다르게 접근한 것이 **Dropout** 이다.



(a) Standard Neural Net (b) After applying dropout.

- 노드들의 복잡한 연결 중 랜덤하게 **일부를 끊고** (= 0으로 만든다) **일부만으로 학습시키는** 알고리즘!
- 보통 dropout rate 를 0.7 정도로 두고 사용한다 (노드를 70% 사용한다는 것)
- 학습시킬 때만 dropout rate 를 1이 아니게 주고 **Model을 사용할때는 반드시 1로** 줘야 한다!

model ensemble

- 학습시킬 수 있는 장비가 많을 때 사용하는 방법

- 여러 독립적인 Training Set을 만들고 합쳐서 결과를 내는 방법이다
- 적게는 2% 많게는 5%까지 성능이 향상된다.