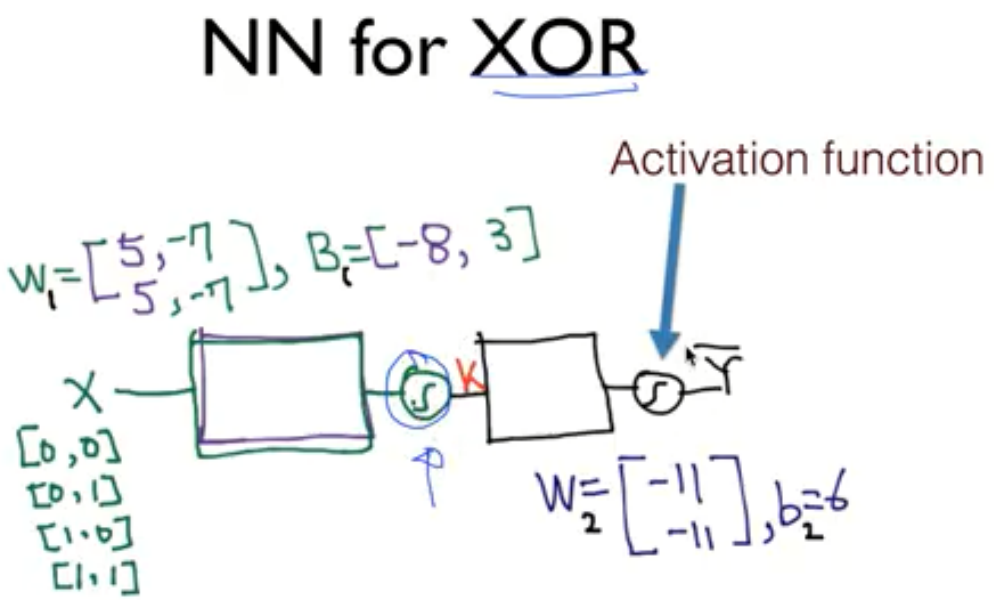
# 6주차 ML 스터디

발표자
박재형

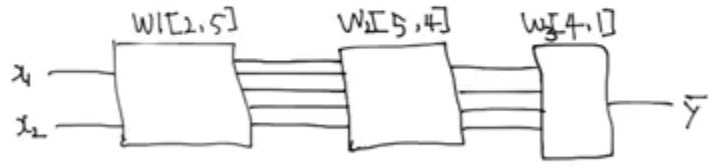
- NN에서 XOR을 연산하기 위해 deep learning이 효과적



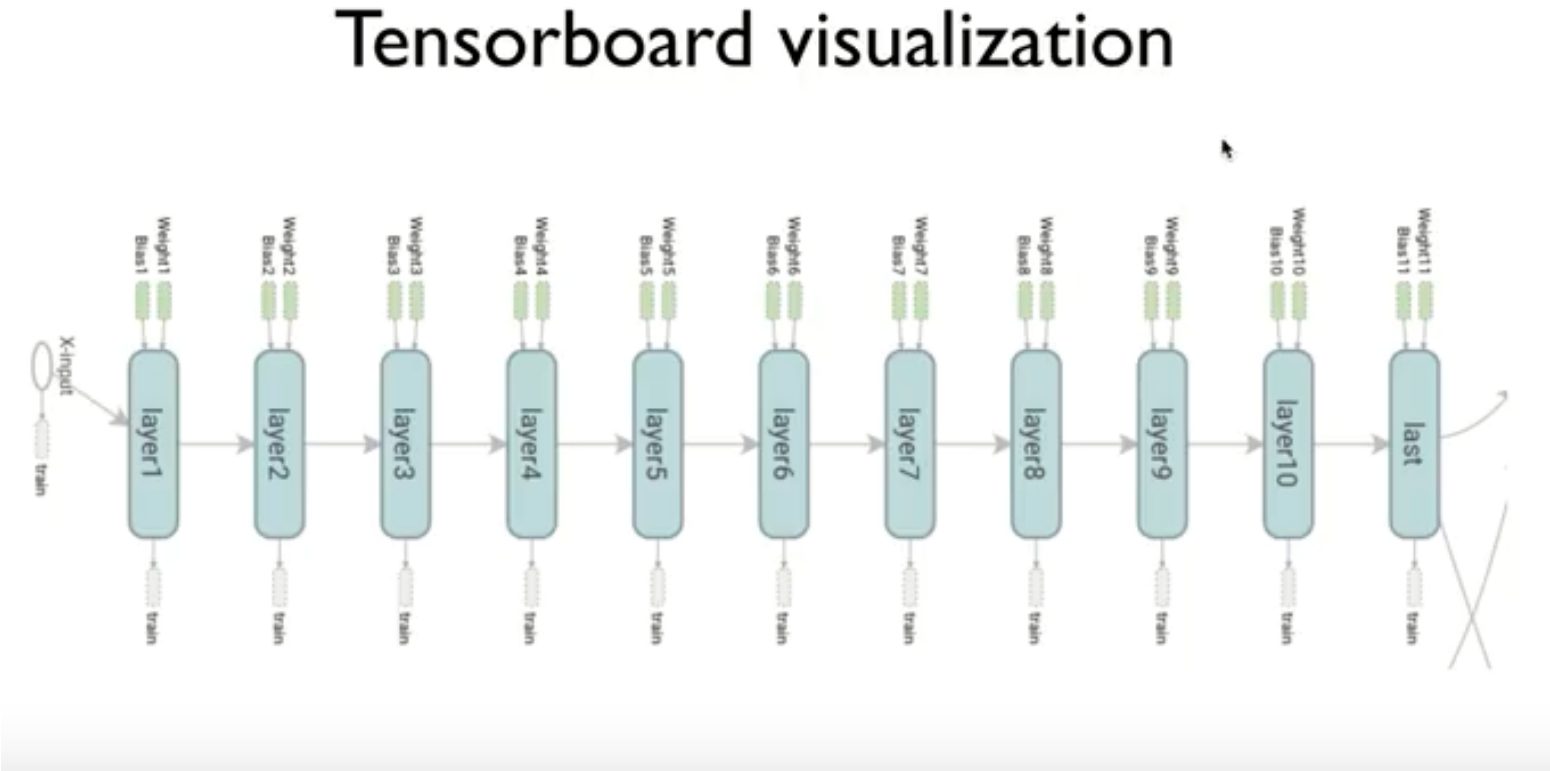
- Wide 보다는 deep 훨씬 효과적
- 왜? -> 해보니까 그렇더라

## Let's go deep & wide!

```tensorflow
W1 = tf.Variable(tf.random_uniform([2, 5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5, 4], -1.0, 1.0))
W3 = tf.Variable(tf.random_uniform([4, 1], -1.0, 1.0))

b1 = tf.Variable(tf.zeros([5]), name="Bias1")
b2 = tf.Variable(tf.zeros([4]), name="Bias2")
b3 = tf.Variable(tf.zeros([1]), name="Bias2")

# Our hypothesis
L2 = tf.sigmoid(tf.matmul(X, W1) + b1)
L3 = tf.sigmoid(tf.matmul(L2, W2) + b2)
hypothesis = tf.sigmoid(tf.matmul(L3, W3) + b3)
```

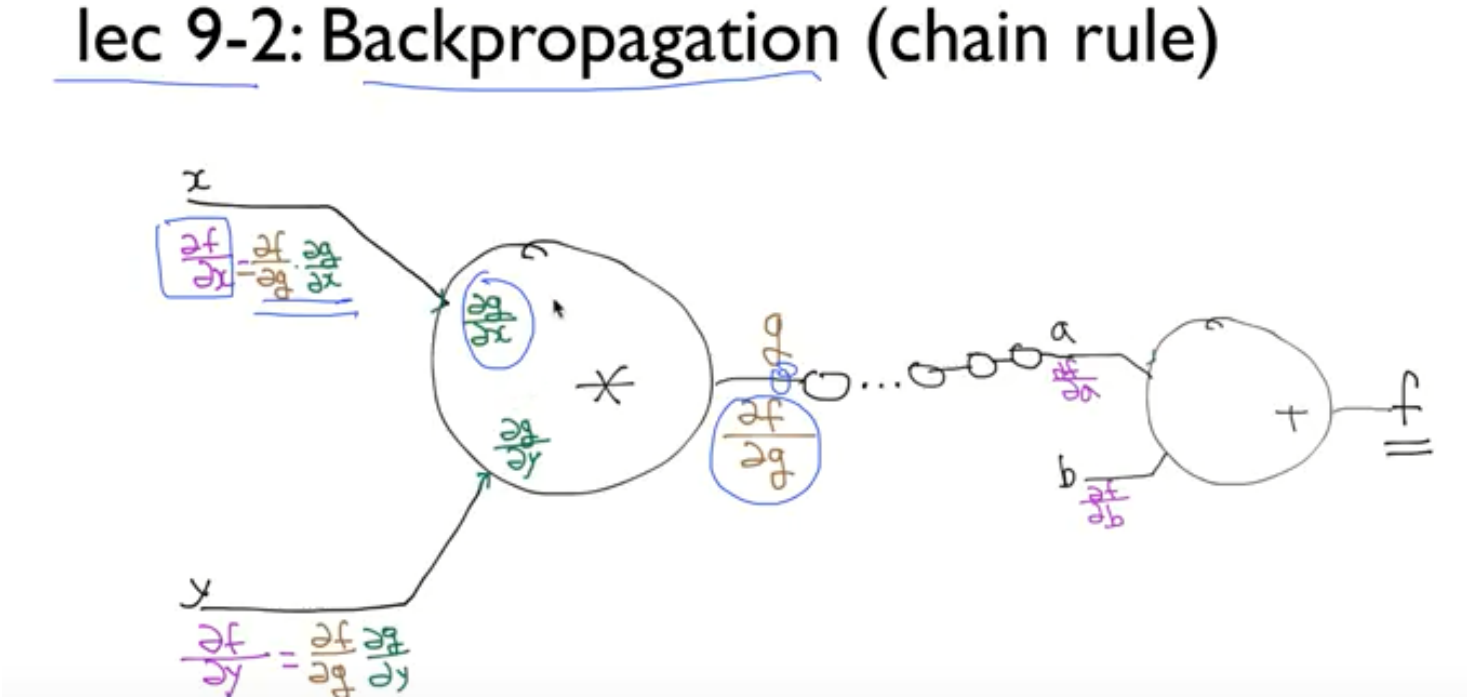


# Deep이 최고


Tensorboard visualization
X-input
train
Weight1
Bias1
layer1
Weight2
Bias2
layer2
Weight3
Bias3
layer3
Weight4
Bias4
layer4
Weight5
Bias5
layer5
Weight6
Bias6
layer6
Weight7
Bias7
layer7
Weight8
Bias8
layer8
Weight9
Bias9
layer9
Weight10
Bias10
layer10
Weight11
Bias11
last

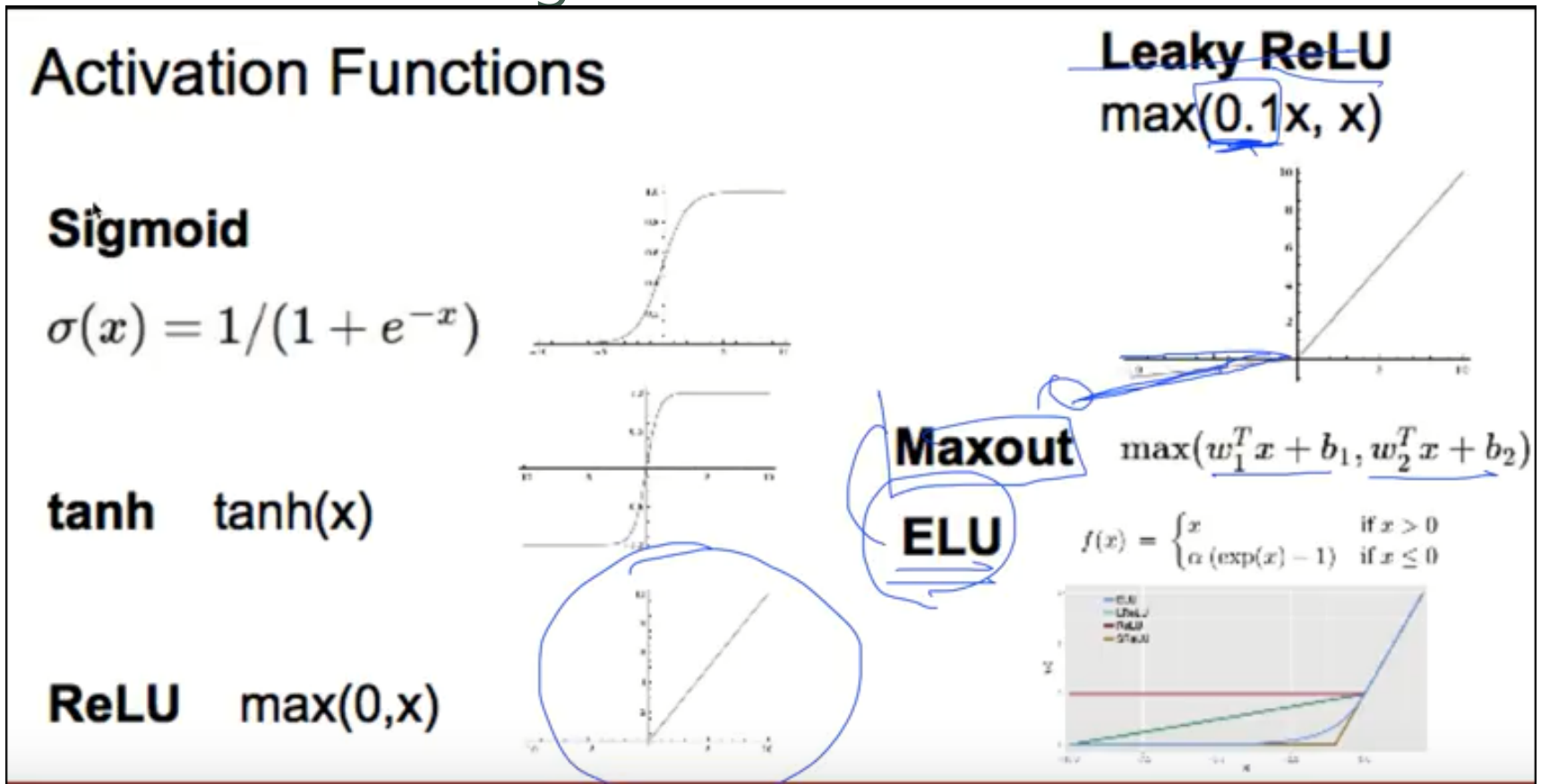
# Deep learning의 문제점

- 항상 두개 vanishing gradient and overfitting
- Backpropagation의 vanishing gradient 문제점



# Vanishing gradient 해결책

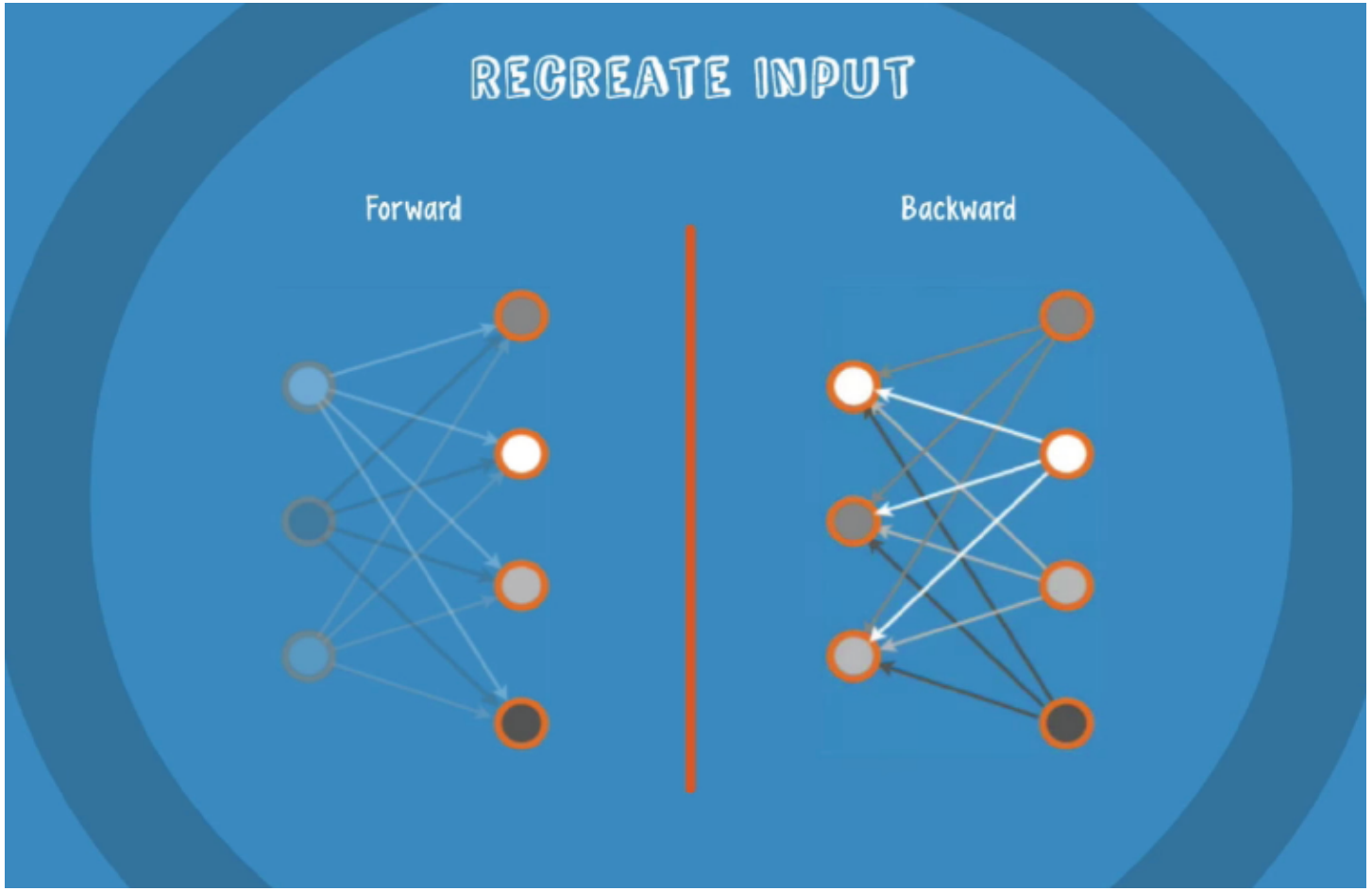
- Activation function을 sigmoid뿐만 아니라 다른 함수 써보자

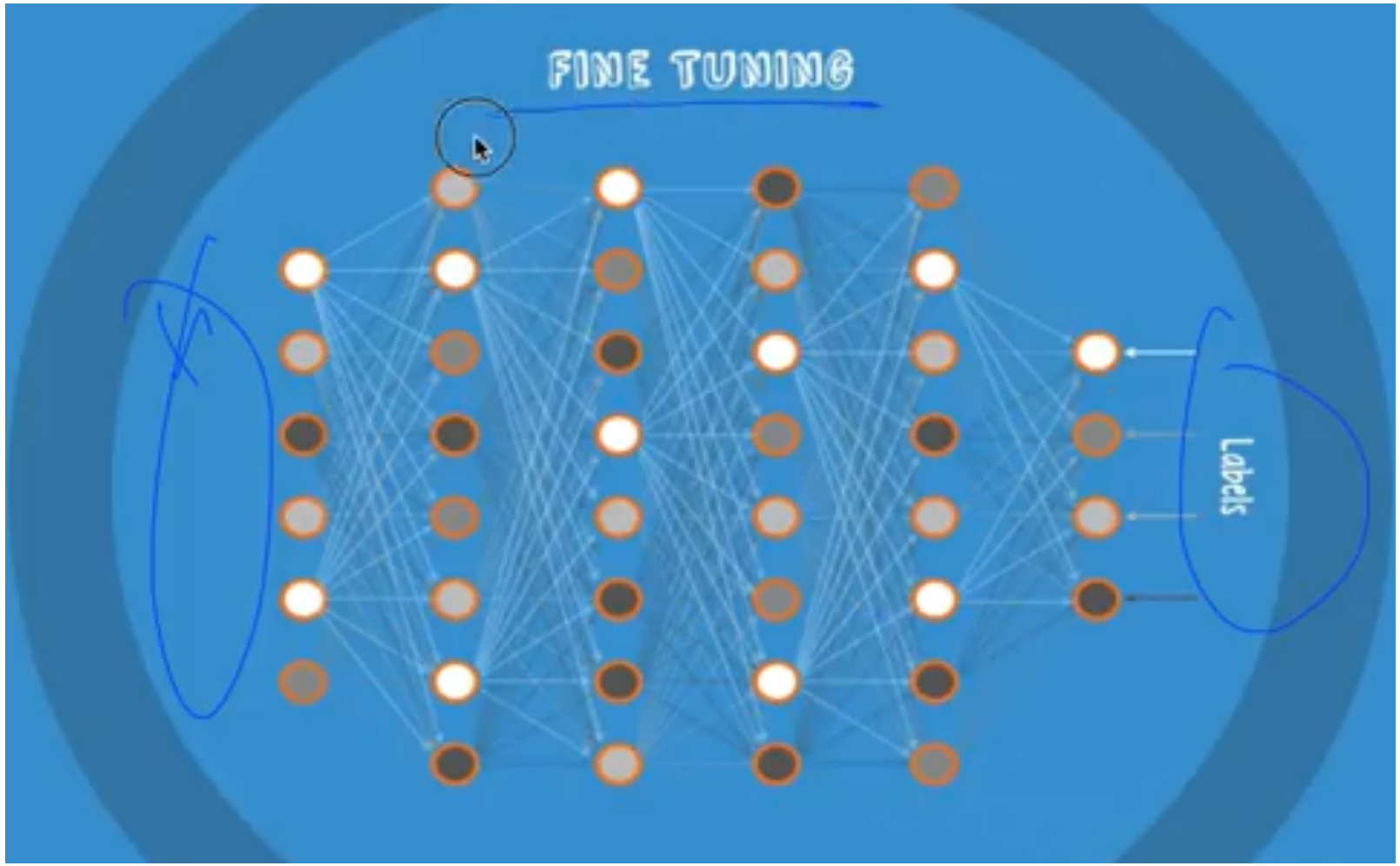


# Weight 초기화

- 이전에는 Initial weight을 랜덤하게 지정
- (당연하게도 weight가 0이면 learning이 안된다)
- 위의 문제점은 local minima에 빠지기 쉬우며 시간도 오래걸림
- Initial weight에 대한 다양한 방법이 제시
- 1. Restricted Boatman Machine(RBM)
- 2. Xavier
- 3. 혹은 이전에 학습한 데이터 활용
- 4. 혹은 작은 규모의 데이터로 학습한데이터를 초기값
- 5. 등등..

# RBM




FINE TUNING
Labels

# Xavier/He initialization

- Makes sure the weights are 'just right', not too small, not too big
- Using number of input (fan_in) and output (fan_out)

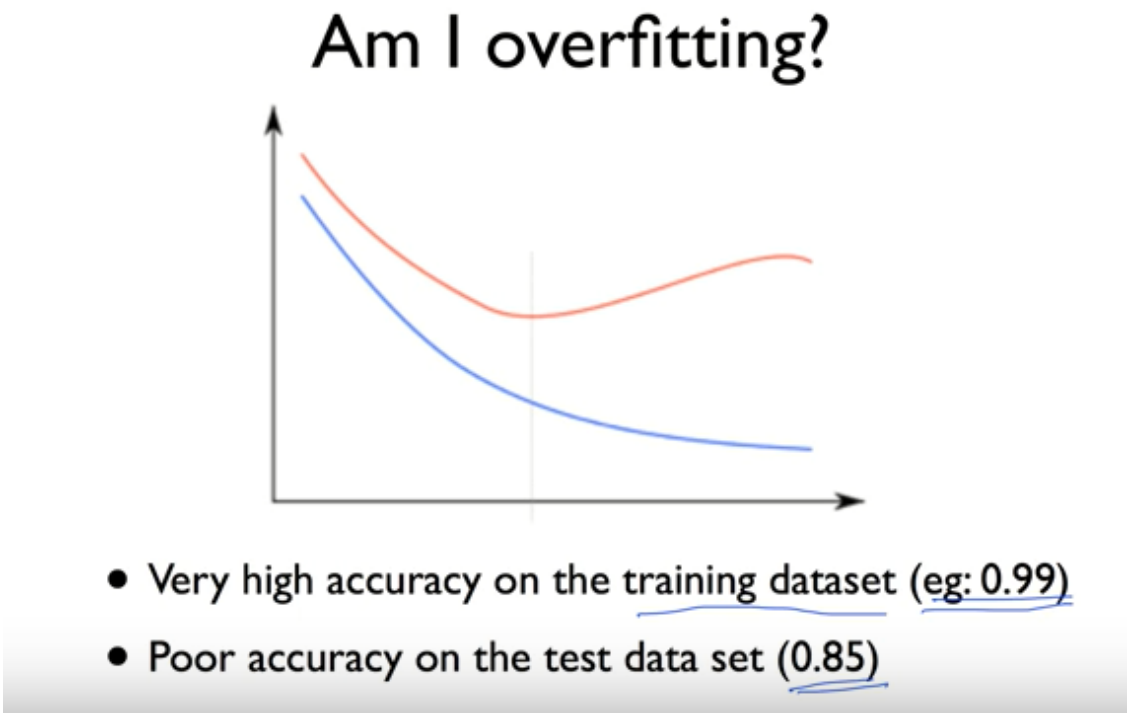
```
# Xavier initialization
# Glorot et al. 2010
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in)

# He et al. 2015
W = np.random.randn(fan_in, fan_out)/np.sqrt(fan_in/2)
```

# Still an active area of research

- We don't know how to initialize perfect weight values, yet
- Many new algorithms
  - Batch normalization
  - Layer sequential uniform variance
  - …

# Overfitting 문제점

- 해결책
- 1. data 양 늘리기 feature 수 줄이기
- 2. early stopping
- 3. Regularization



# Regularization

- 가장 간단한 regularization은 weight에 어떤 제한을 가하는 것 임
- L2-regularization example :

$$E_t(\mathbf{w}) \equiv \frac{1}{N_t}\sum_{n\in\mathcal{D}_t} E_n(\mathbf{w}) + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}$$

- $\lambda$ 는 이 규제화의 강도 control
- $\lambda$ 는 0.01 ~ 0.00001
- Weight update :

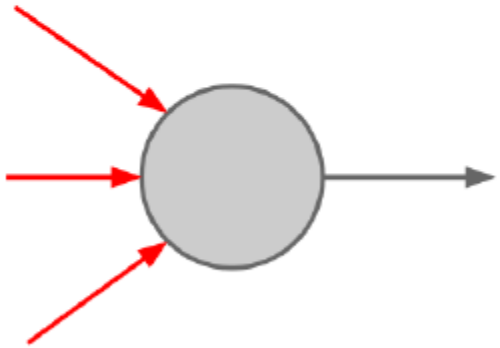
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \epsilon\left(\frac{1}{N_t}\sum \nabla E_n + \underbrace{\lambda \mathbf{w}^{(t)}}\right)$$

:weight이 자신의 크기에 비례하는 속도로 감쇠하여,
weight decay라고 부름.

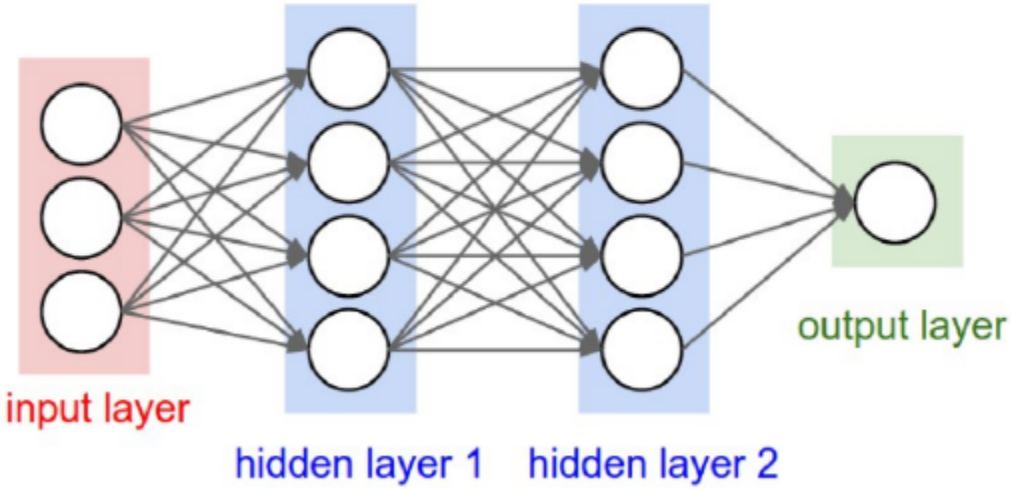
**Regularization knobs**

- L2 regularization $\frac{1}{2}\lambda w^2$
- L1 regularization $\lambda \mid w \mid$
- L1 + L2 can also be combined
- Max norm constraint $\sum_i w_{ji}^2 < c$

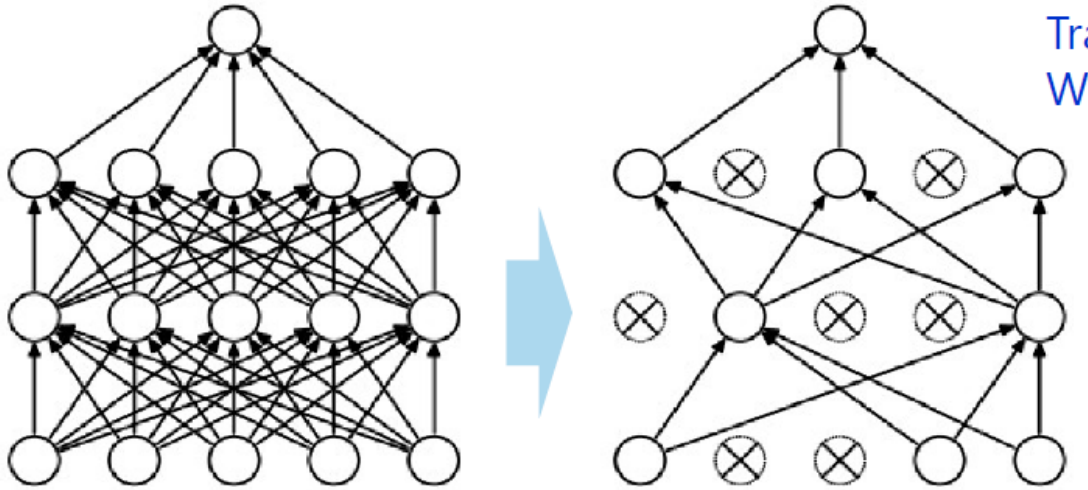
L1 is “sparsity inducing” (many weights become almost exactly zero)



enforce maximum L2 norm of the incoming weights



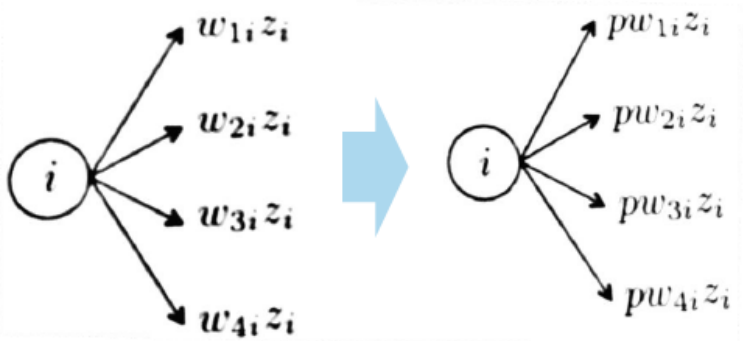
# Regularization: **Dropout**
## "randomly set some neurons to zero"



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

Training Phase
With probability p

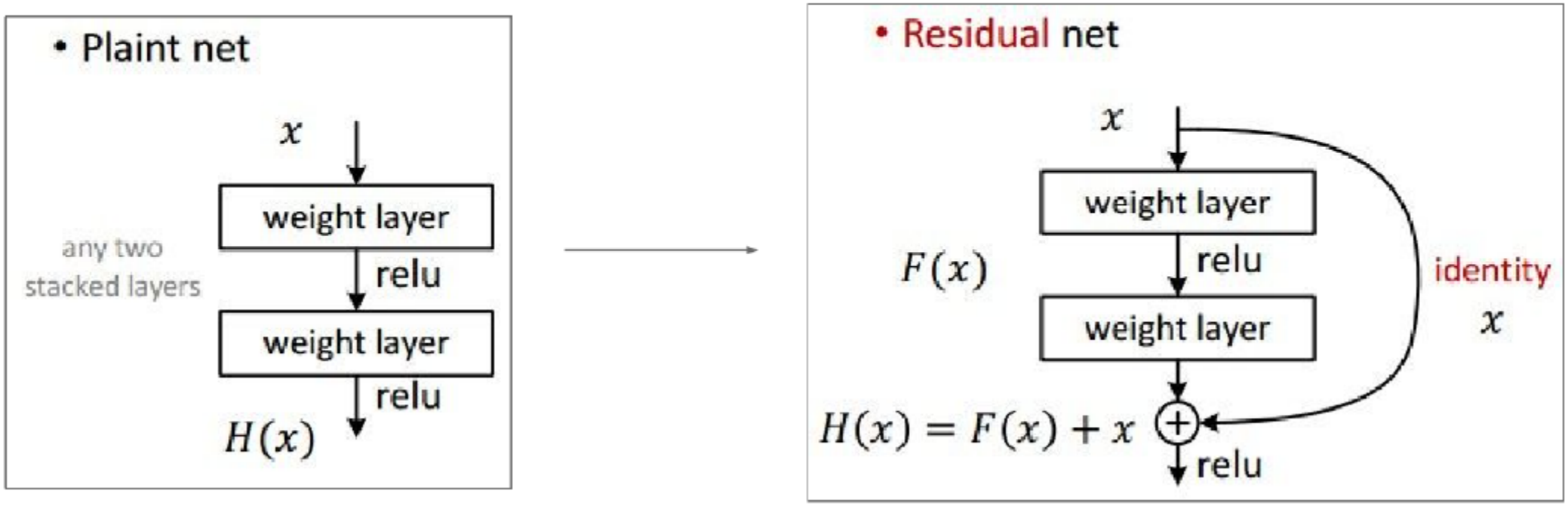
*[Srivastava et al.]*




Test Phase
With probability p

# ResNet(fast forward? 글쎄..)

## Case Study: ResNet [He et al., 2015]



# ResNet의 효과

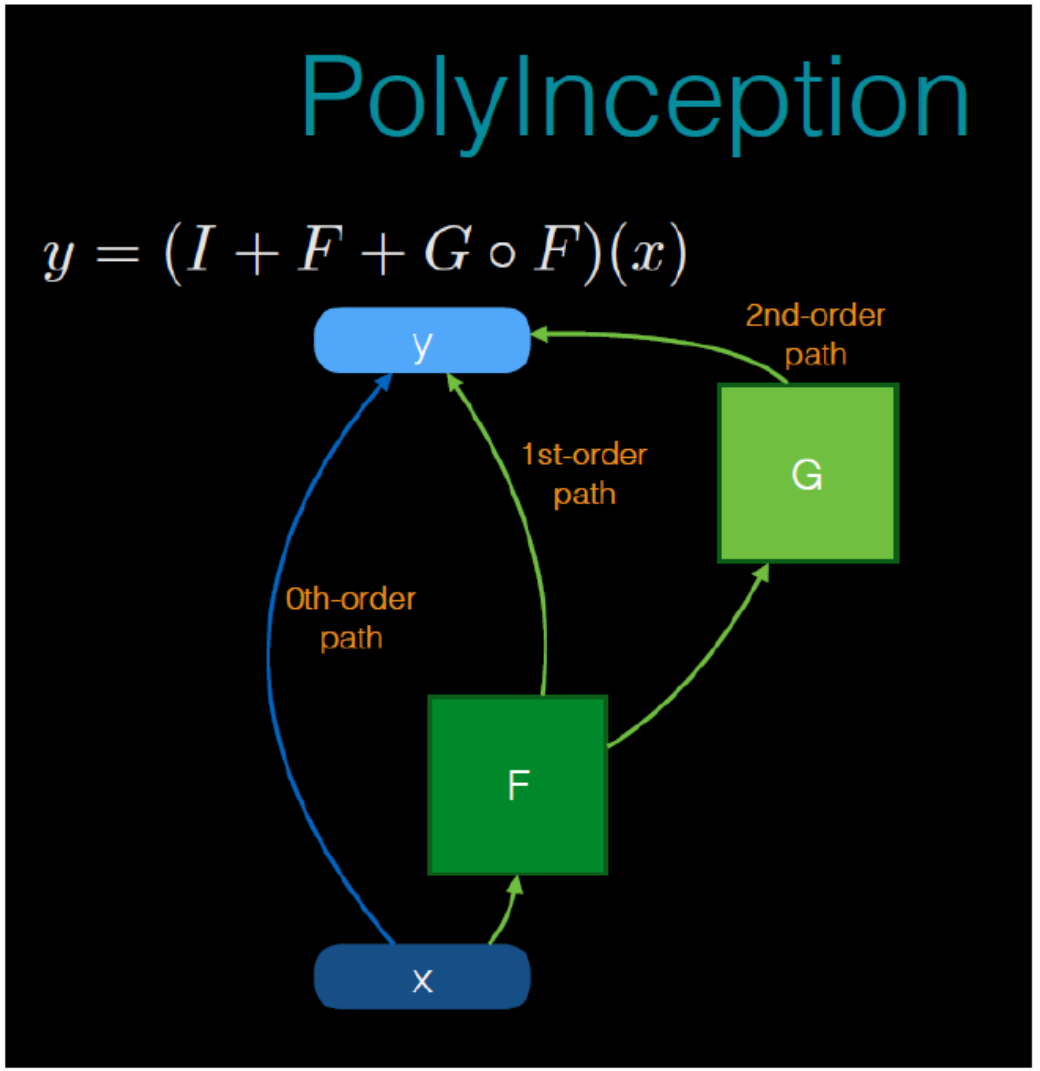
- https://laonple.blog.me/220761052425
- 위는 ResNet 설명해놓은 사이트 (CNN에 대한 깊이 있는 설명)

- 출력 : 입력+입력차이(F(x),F(x)가 0으로 가게)
- 다른 activation function이어도 vanishing gradient 완전제거 불가
- But ResNet으로 깊은 망을 효과적으로 learning 가능
- ->덕분에 100layer을 넘기면서 learning->image 오차 3%이하

# Block 구조

- 아마 block 자유성을 강조하기 위해...
- Fast forward
- Split and merge
- RNN

**PolyNet**



# Optimization(번외?)

- http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html
- 위에는 다양한 gradient descent optimization 설명
- Keyword
- 1.gradient descent
- 2.batch gradient descent
- 3.stochastic gradient descent
- 4.momentum
- http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

# Gradient Descent

- Loss function 즉 기울기가 최소가 되는 지점을 찾을 때
- 전체 train set를 사용하는 것을 Batch Gradient Descent
- ->단점 : 한번 loss function을 구할 때 마다 많은 계산량
- 많은 계산량 단점 보완 -> Stochastic Gradient Descent(SGD)
- 전체 데이터 대신 mini-batch에 대해서만 loss function 계산
- 전체 test set보다 부정확할 수도 but 계산 속도 빨라
- 같은 시간에 더 많은 loss function 계산
- Local minima에 빠지지 않을 확률이 높다.

# Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are ~100 examples.
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

Mini batch size is also hyperparameter

$$E_t(\mathbf{w}) = \frac{1}{N_t} \sum_{n \in \mathcal{D}_t} E_n(\mathbf{w})$$

# SGD

## Batch Normalization

[Ioffe and Szegedy, 2015]

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$

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

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

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

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

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

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph, get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

# momentum

- 오차 진동(oscillation)을 적게 하여 수렴 속도를 높여주기 위해 첨가한 Momentum term이다.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

- 변화량에 영향을 준다.

Some details on SGD parameters

Momentum LR Decay

$$V_{t+1} = \mu V_t - \alpha(\nabla L(W_t) + \lambda W_t)$$

$$W_{t+1} = W_t + V_{t+1}$$