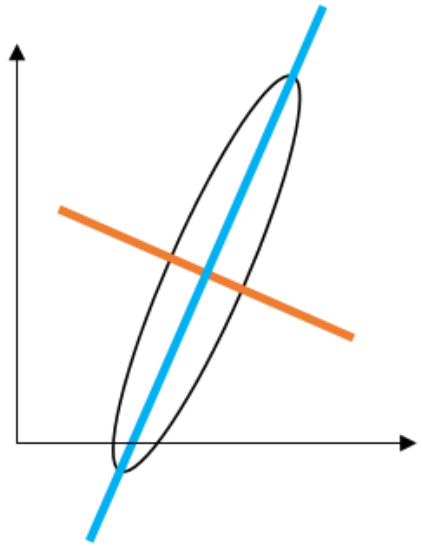
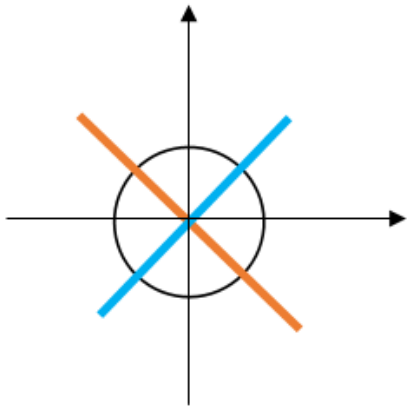
Lecture 8.
Training Neural Networks Ⅲ

Review

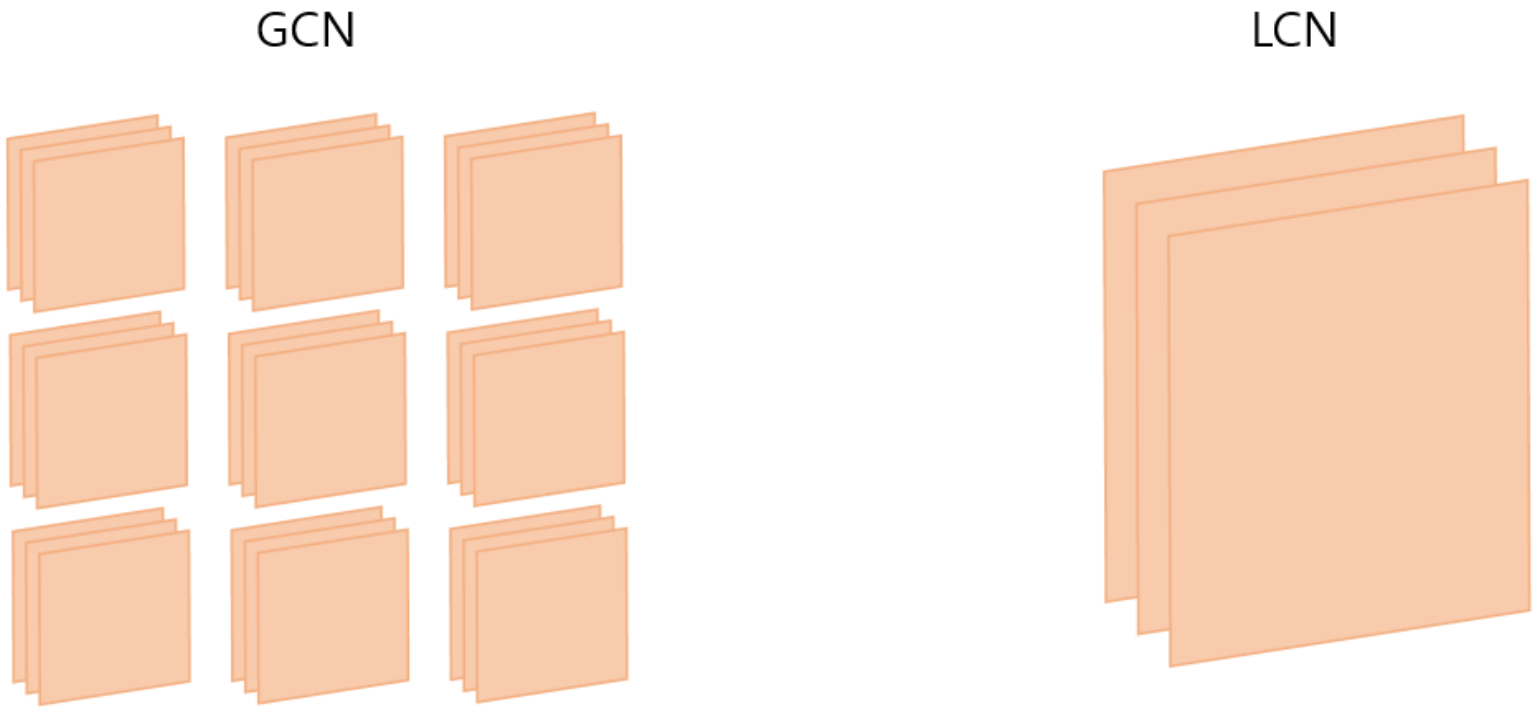
Before normalization



After normalization



Review



Review

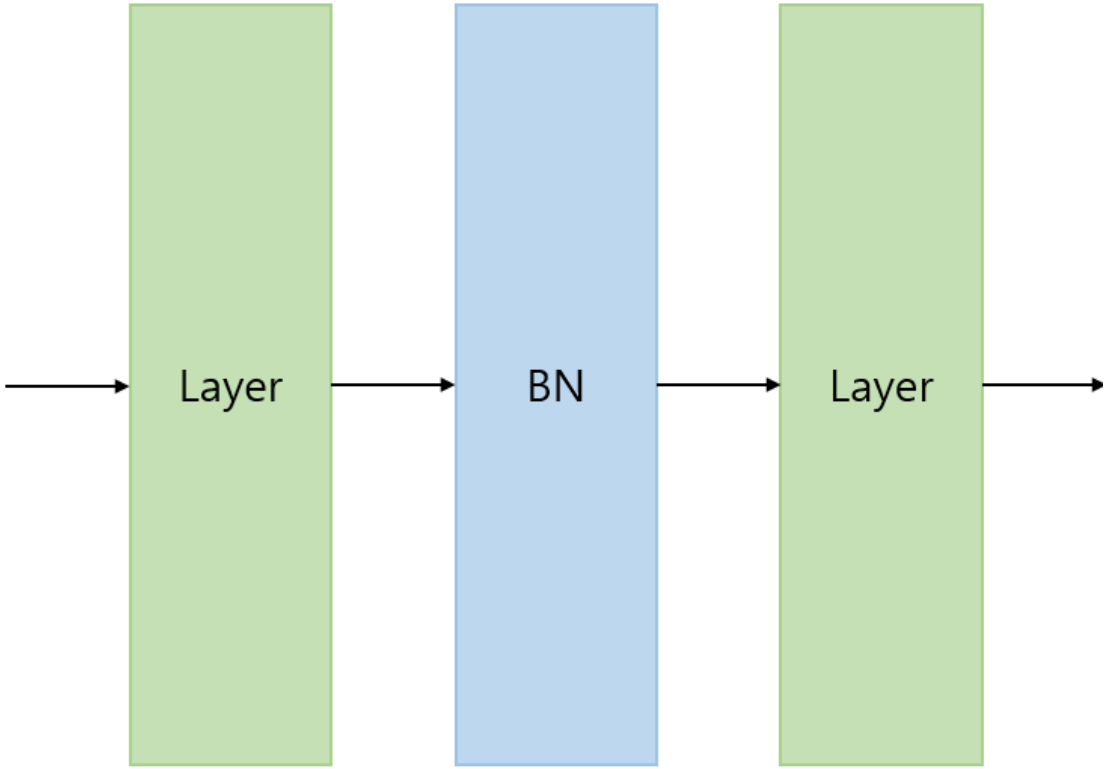
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

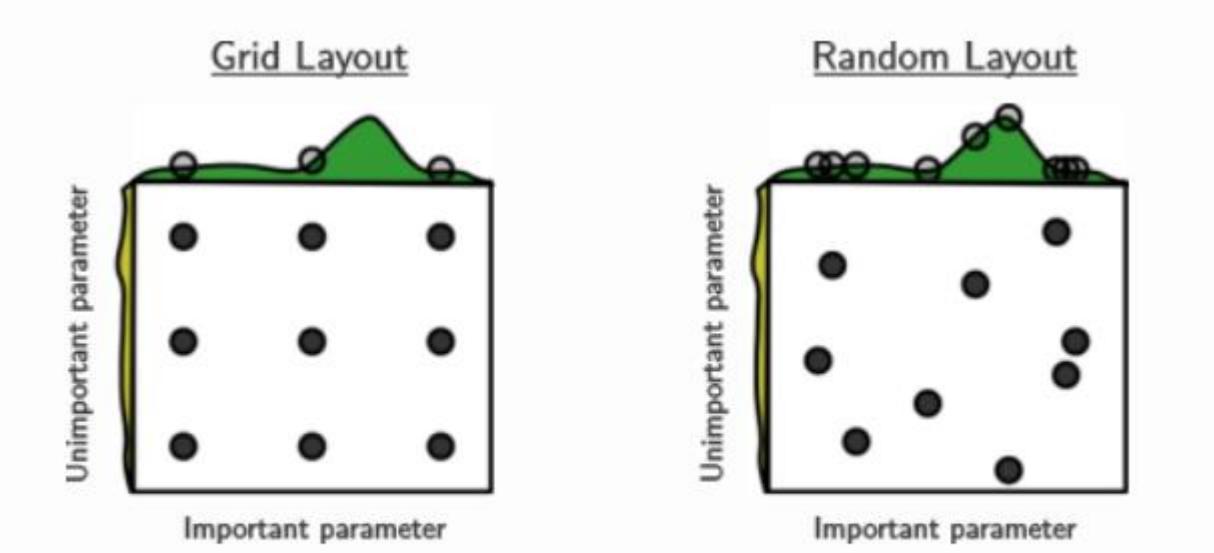
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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



Review



Grid Search 결과와 Random Search 결과 비교 예시

[Bergstra and Bengio(2012)]

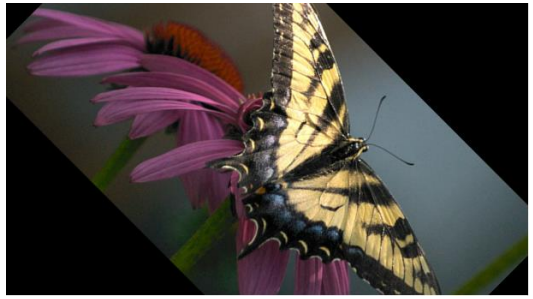
Review

Crop

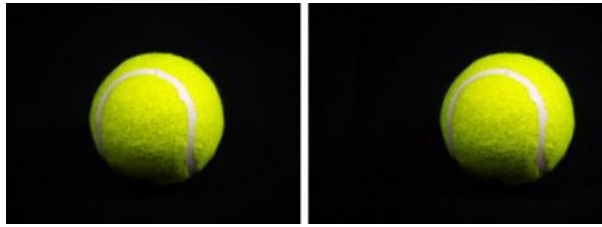
Flip

Noise

Scale



Rotate



Translation

Today's Contents

1. Change Optimization Process
    - Gradient based method - Optimizer
    - Regularization - Dropout

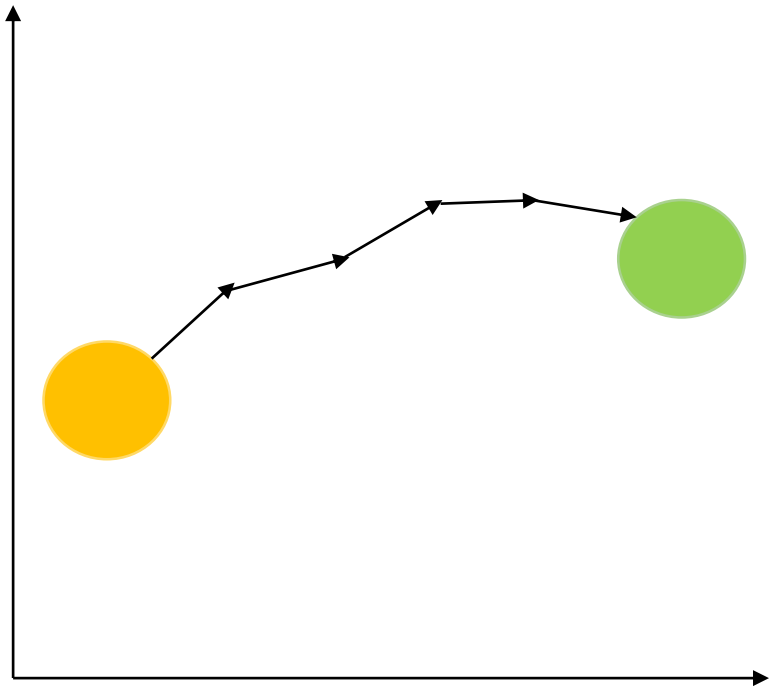
Gradient Descent Optimization Method

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Momentum
4. Nesterov Accelerated Gradient (NAG)
5. Adagrad
6. RMSProp
7. AdaDelta
8. Adam

Gradient Descent

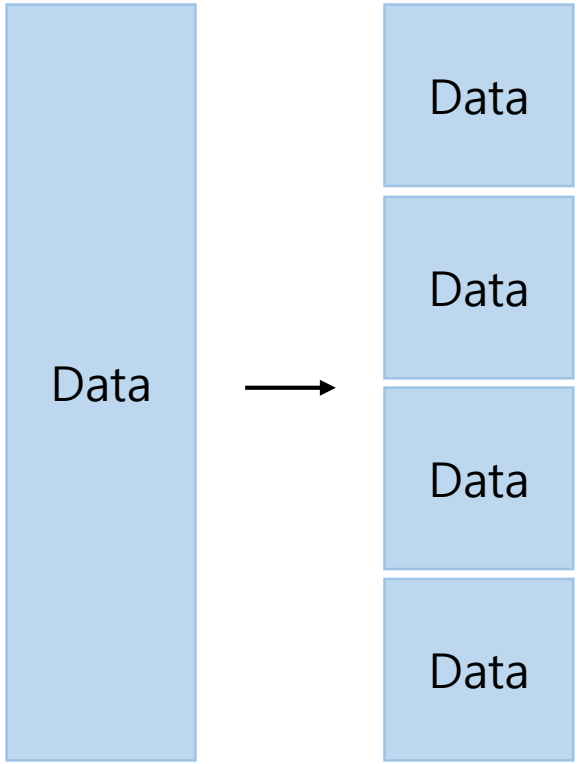
$$w_{new} = w - \eta \times \frac{\partial L}{\partial w}$$

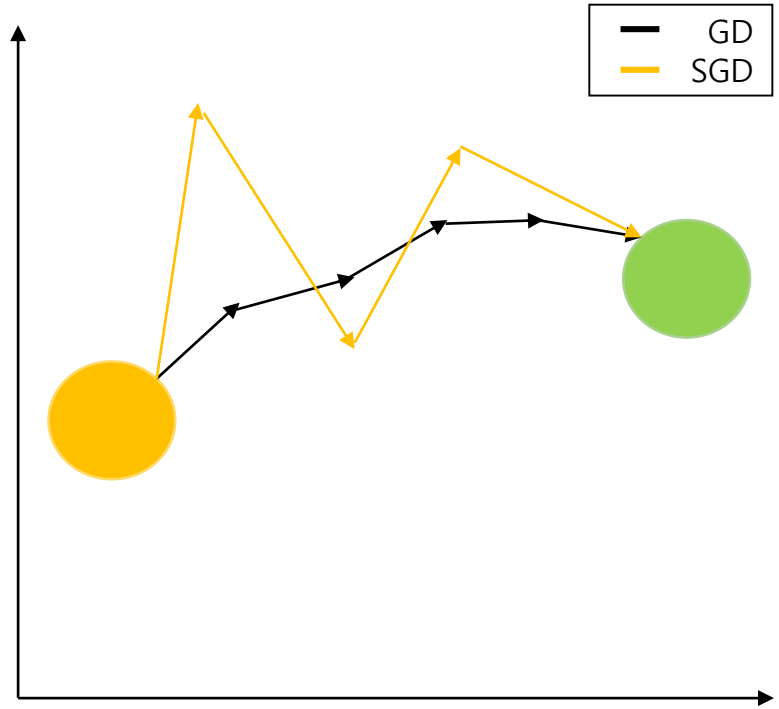
Data



Stochastic Gradient Descent

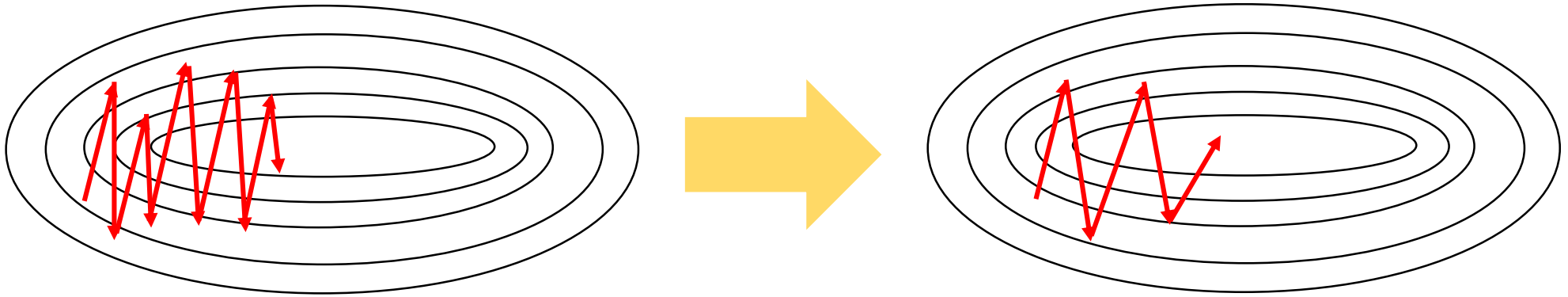
$$w_{new} = w - \eta \times \frac{\partial L}{\partial w}$$





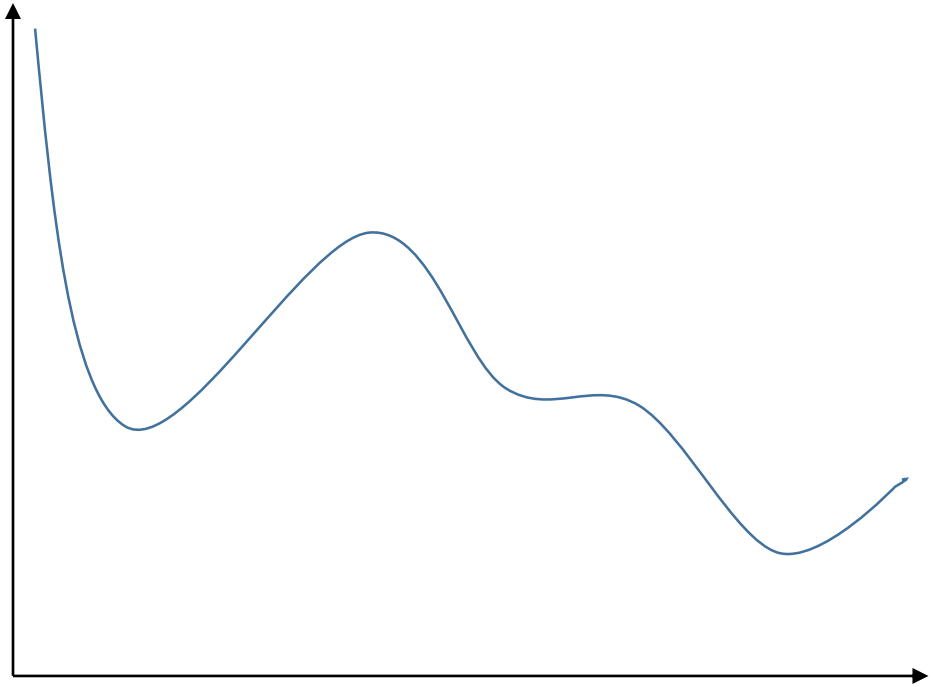
Two Problems of SGD

1. Oscilation

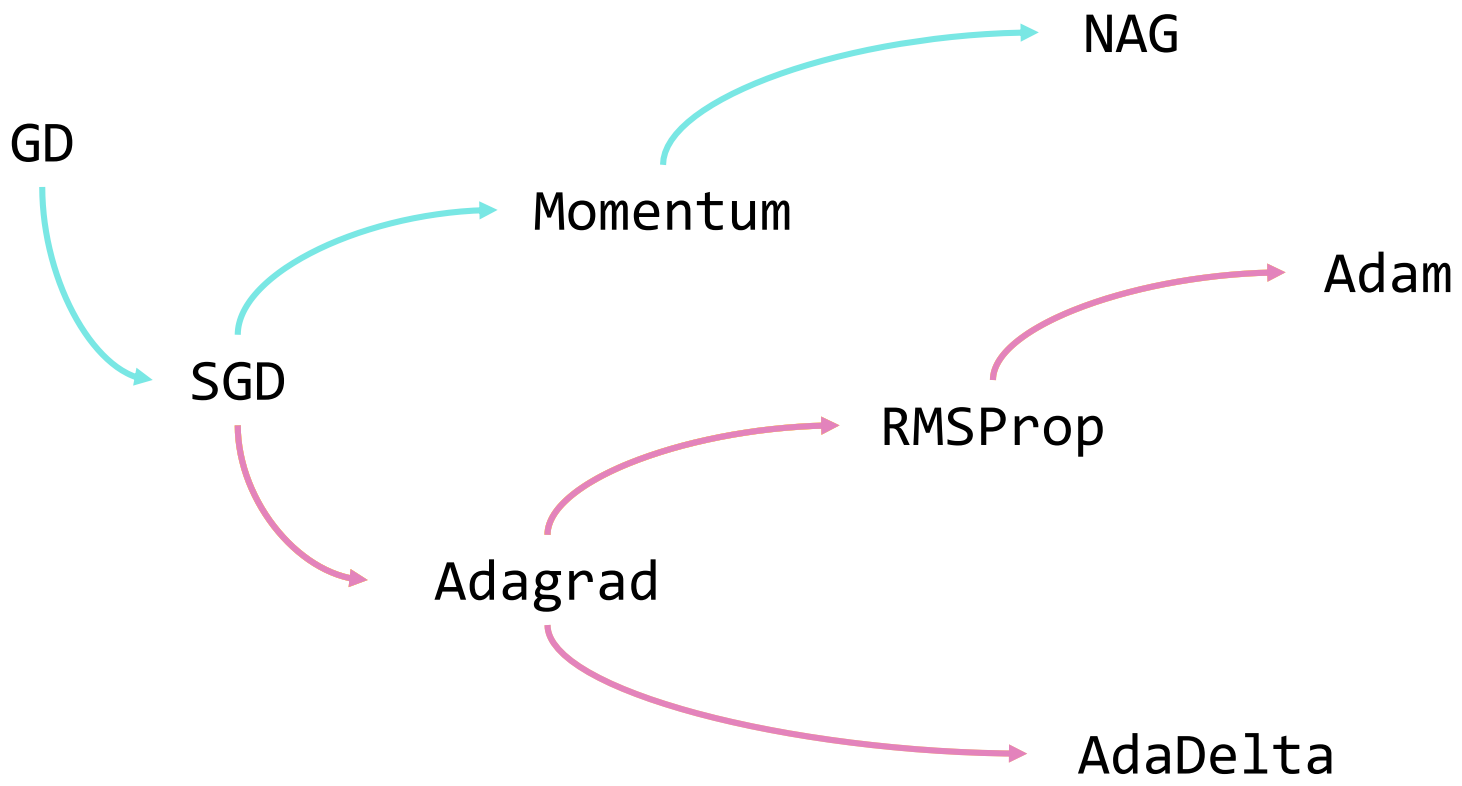


Two Problems of SGD

2. Local minimum



Optimizer 발전 과정



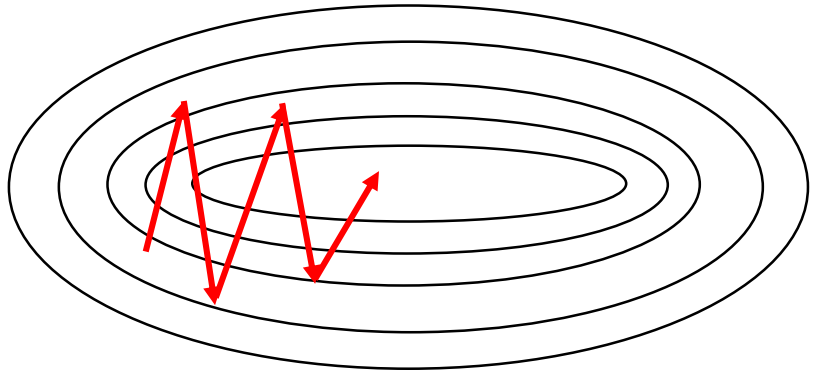
Mometum

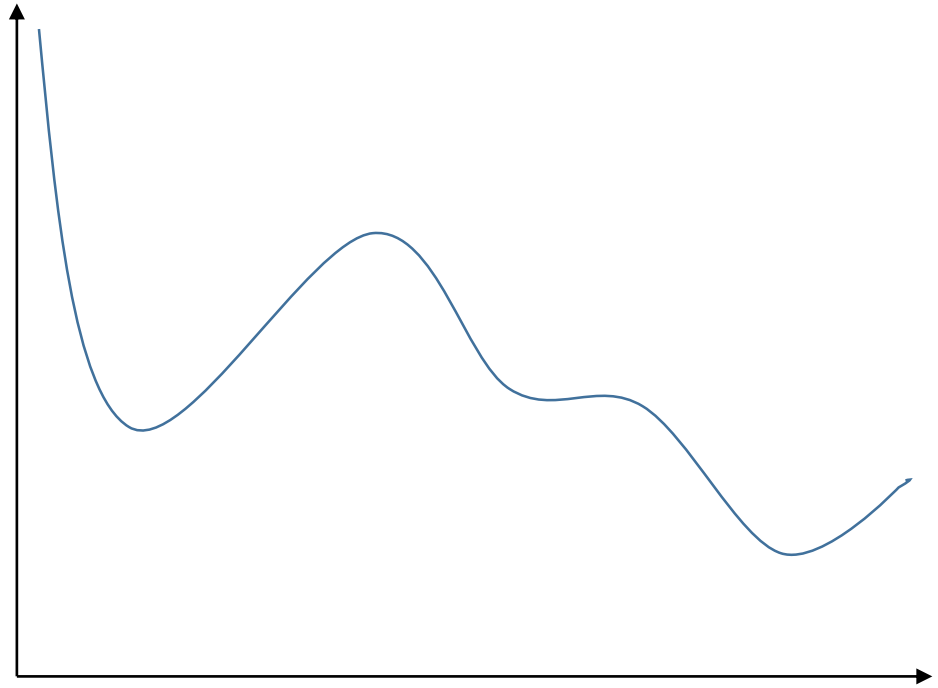
Idea : gradient update에 관성을 더하자!

$$w_{new} = w - v_t$$

$$v_t = \gamma v_{t-1} + \eta \frac{\partial L}{\partial w}$$

Momentum



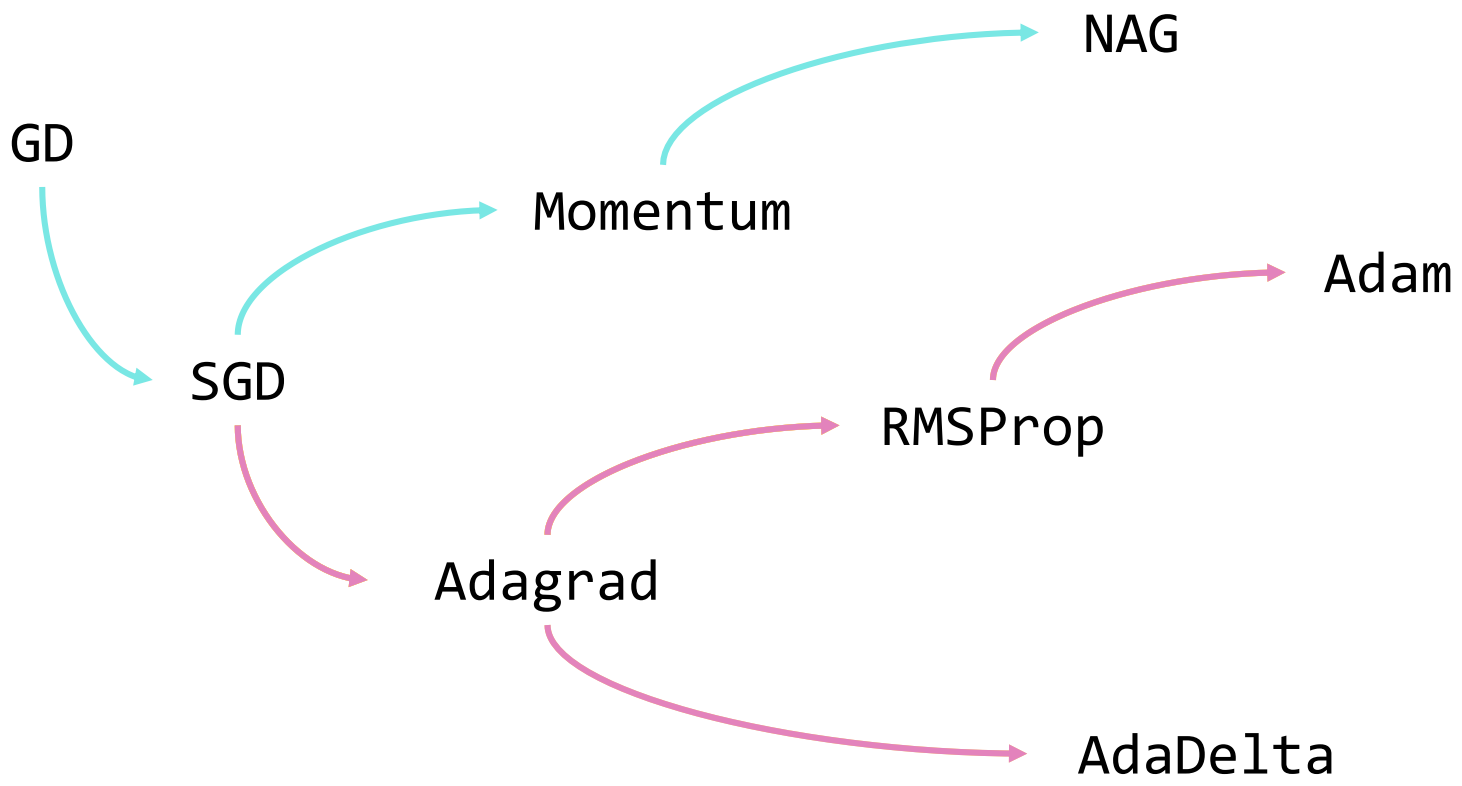


Nestrov Accelerated Gradient

$$w_{new} = w - v_t$$

$$v_t = \gamma v_{t-1} + \eta \frac{\partial L}{\partial w}\bigg|_{w=w-\gamma v_{t-1}}$$

Optimizer 발전 과정



Adagrad

Idea : learning rate를 gradient에 따라 다르게 탐색하자!

$$w_{new} = w - \frac{\eta}{\sqrt{G_t + \varepsilon}} \frac{\partial L}{\partial w}$$

$$G_t = G_{t-1} + \left(\frac{\partial L}{\partial w} \right)^2$$

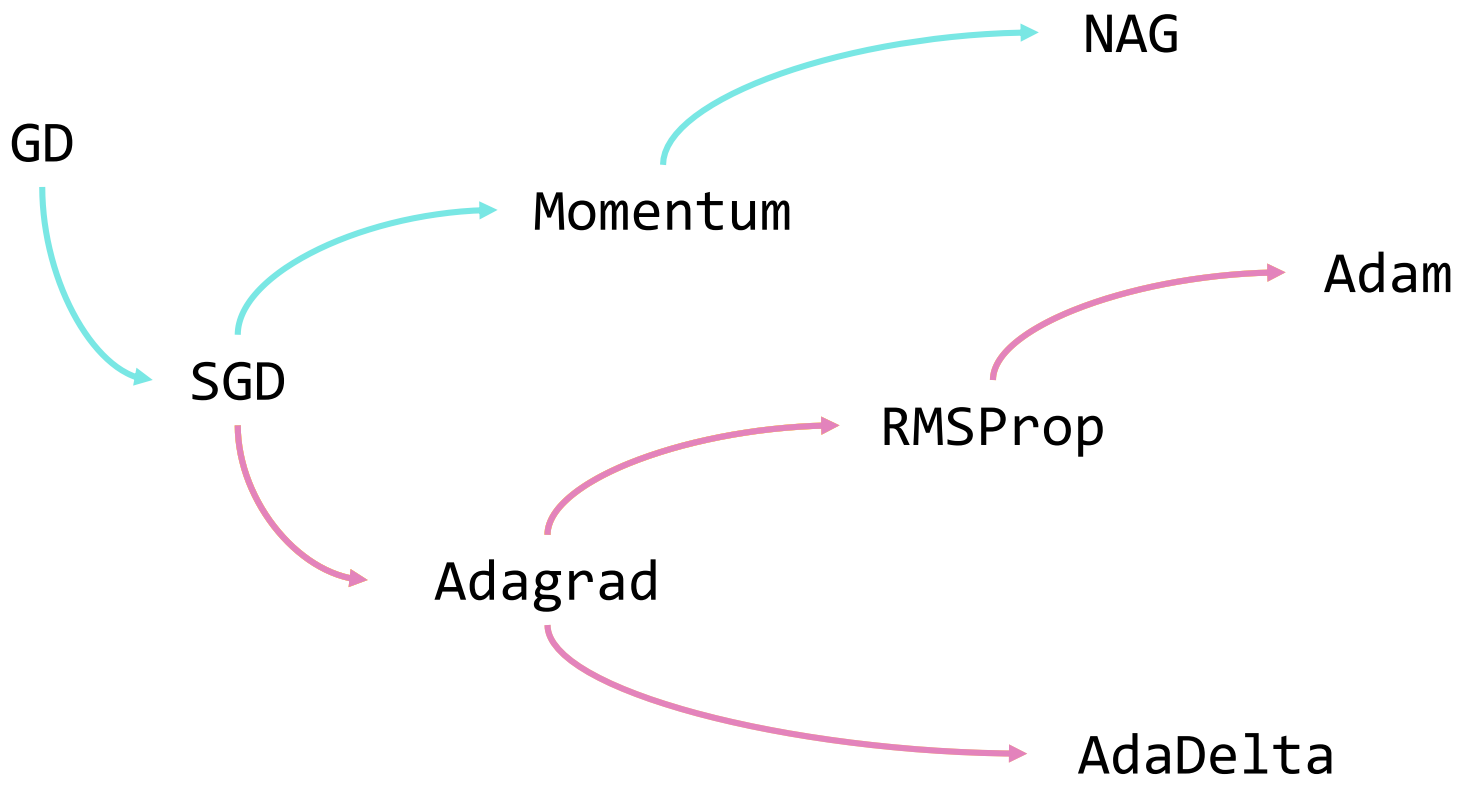
Problem of Adagrad

G_t가 무한정 커지면...?

➔ learning rate η가 점점 0에 가까워진다!

➔ 학습이 안 된다..!

Optimizer 발전 과정



RMSProp

Idea : gradient에 따라 learning rate 바꾸는 건 좋은데, 무한정 커지는 걸 방지하자!

$$w_{new} = w - \frac{\eta}{\sqrt{G + \varepsilon}} \frac{\partial L}{\partial w}$$

$$G = \gamma G + (1 - \gamma) \left(\frac{\partial L}{\partial w} \right)^2$$

AdaDelta

Idea : gradient에 따라 learning rate 바꾸는 건 좋은데, 무한정 커지는 걸 방지하자!

$$w_{new} = w - \Delta w$$

$$\Delta w = \frac{\sqrt{s + \varepsilon}}{\sqrt{G + \varepsilon}} \frac{\partial L}{\partial w}$$

$$G = \gamma G + (1 - \gamma) \left(\frac{\partial L}{\partial w} \right)^2$$

$$s = \gamma s + (1 - \gamma) \Delta w^2$$

Adam

Idea : RMSProp에 Momentum을 합친다면..?

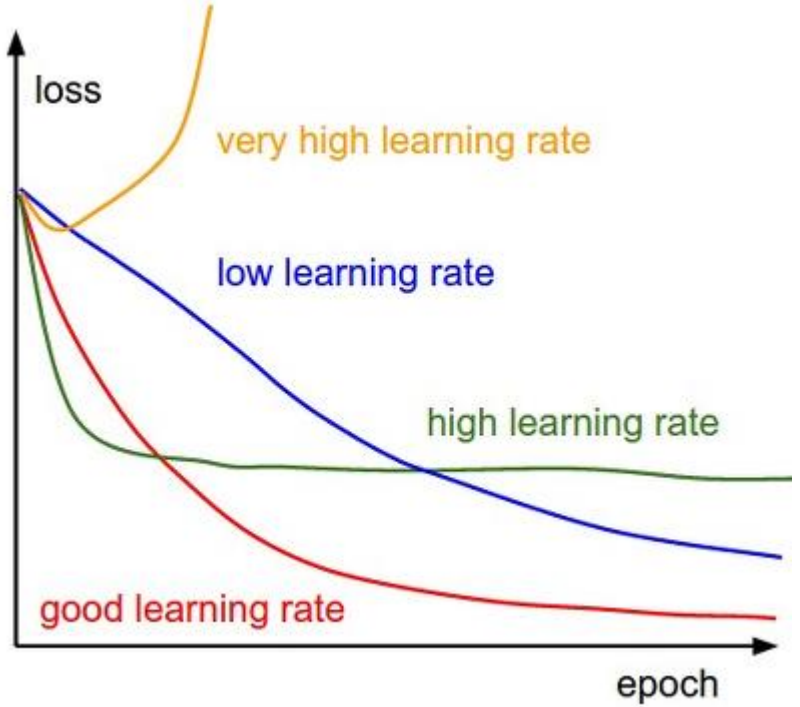
$$w_{new} = w - \frac{\eta}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial w} \right)^2$$

Learning rate decay



Optimizers in Keras

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

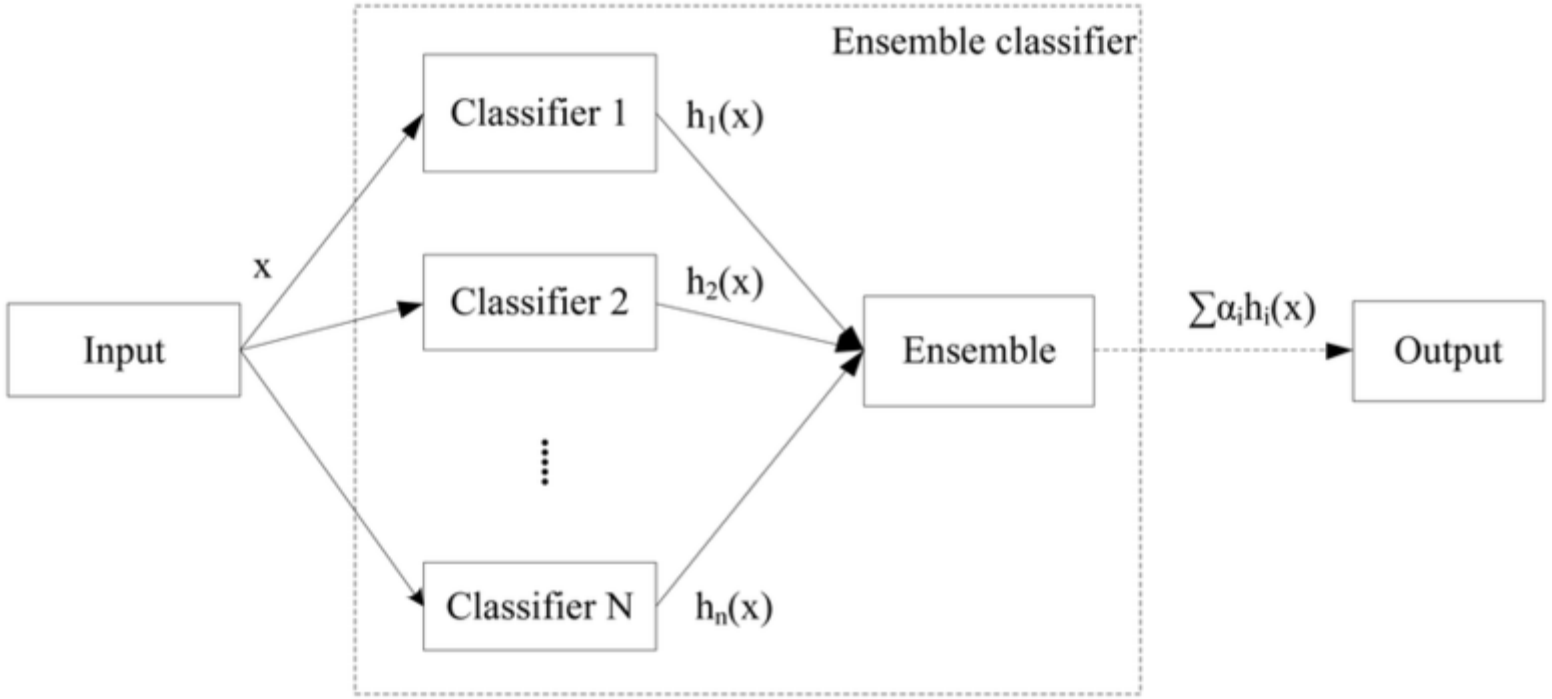
```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

```
keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
```

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

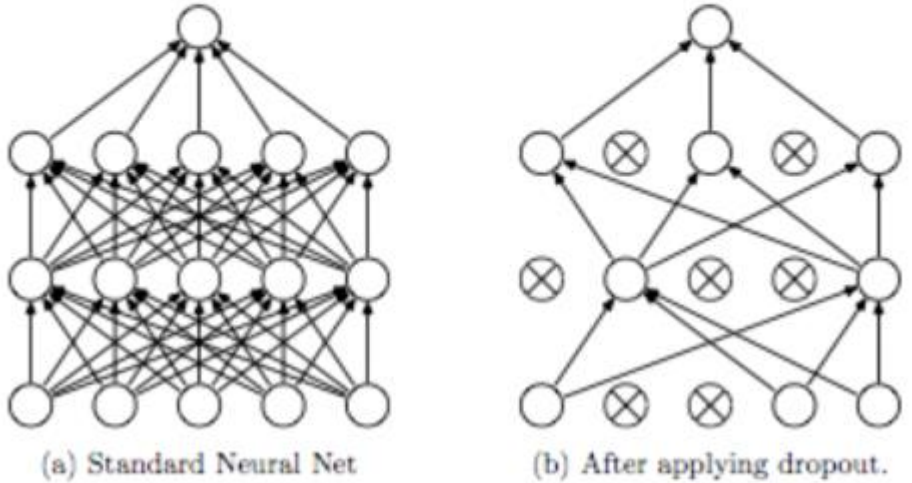
Model Ensemble

Idea : 모델 하나만 학습시킬 게 아니라, 여러 개를 학습시켜서 합하자!



Dropout

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]

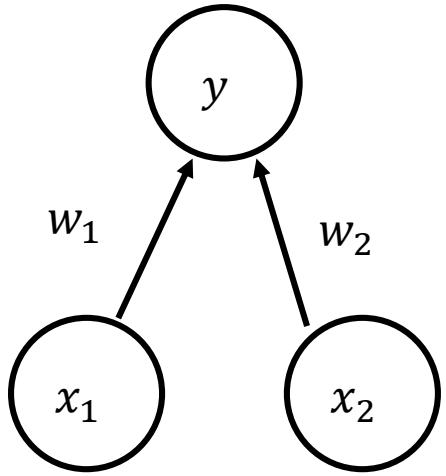


(a) Standard Neural Net (b) After applying dropout.

Dropout for test data..?

$$y = f(x, z)$$

$$y = f(x) = E_z[f(x, z)] = \int p(z) f(x, z) dz$$



Test time: $E[y] = w_1 x_1 + w_2 x_2$

During Training: $E[y] = \frac{1}{4}(w_1 x_1 + w_2 x_2) + \frac{1}{4}(w_1 x_1 + 0) + \frac{1}{4}(0 + w_2 x_2) + \frac{1}{4}(0 + 0)$

Dropout in Keras

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Preview on Next Class