4. Regularization and Optimizer

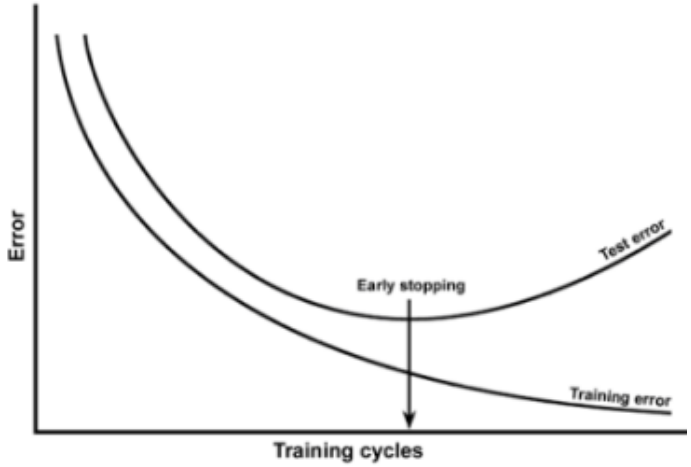
AILab
Hanyang Univ.

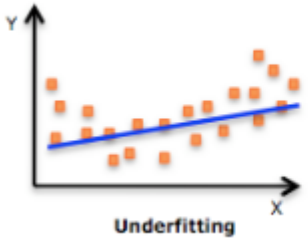
오늘 실습 내용

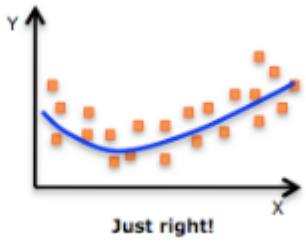
- Overfitting
  - Regularization
  - Drop-out
- Various Optimizers

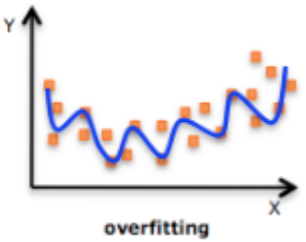
Overfitting

- **Overfitting이란?**
  - 한 데이터셋에만 지나치게 최적화된 상태
  - 아래 그래프처럼 학습 데이터에 대해서는 오차가 감소하지만 실제 데이터에 대해서는 오차가 증가하는 지점이 존재할 수 있음
  - 즉, overfitting은 학습데이터에 대해 과하게 학습하여 실제 데이터에 대한 오차가 증가할 경우 발생





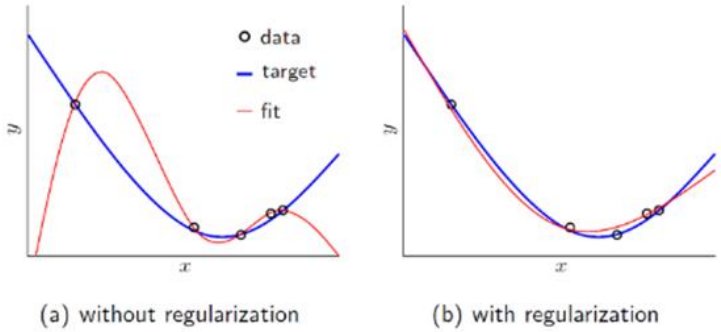




Overfitting 을 완화시키는 방법은?

- Training Data 를 늘린다.
- Regularization
- Drop-Out
- 그 외 ...

Regularization



(a) without regularization (b) with regularization

- **Regularization이란?**
  - W(weight)가 너무 큰 값들을 가지지 않도록 하는 것
  - W가 너무 큰 값을 가지게 되면 과하게 구불구불한 형태의 함수가 만들어짐
  - 즉, Regularization은 모델의 복잡도를 낮추기 위한 방법
  - 모델의 복잡도를 낮춰 줌
- **어떻게?**
  - 단순하게 cost function을 작아지는 쪽으로 학습하면 특정 가중치 값들이 커지면서 결과를 나쁘게 만들기 때문에 cost function을 바꾼다

L1 Regularization

- L1 Regularization

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|$$

C_0 : 원래 cost function
n : 훈련 data 갯수
λ : regularization 변수
W : 가중치

$$w \to w' = w - \frac{\eta \lambda}{n} \mathrm{sgn}(w) - \eta \frac{\partial C_0}{\partial w}$$

L2 Regularization

- L2 Regularization

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

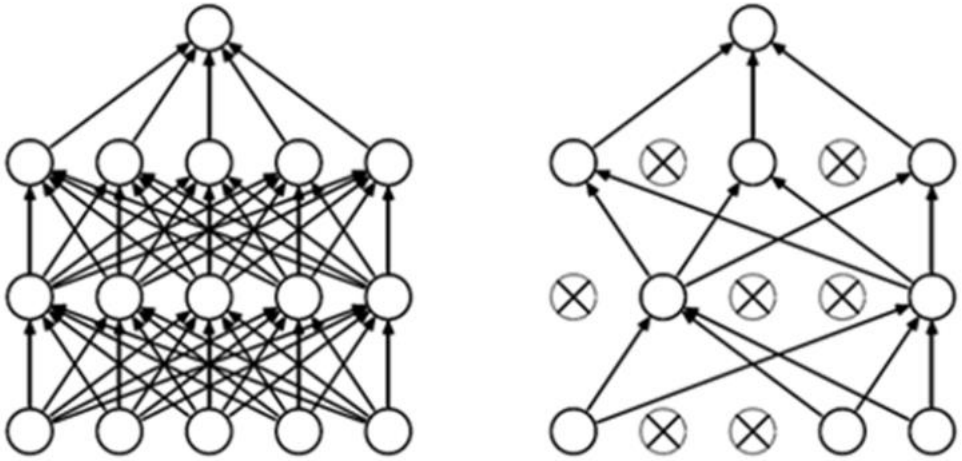
C_0 : 원래 cost function
n : 훈련 data 갯수
λ : regularization 변수
W : 가중치

$$\begin{aligned} w &\rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} w \\ &= \left(1 - \frac{\eta \lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w} \end{aligned}$$

Dropout

- 개념 설명

학습과정에서 Layer의 node를 random하게 drop함으로써, over-fitting문제를 완화시키는 방법

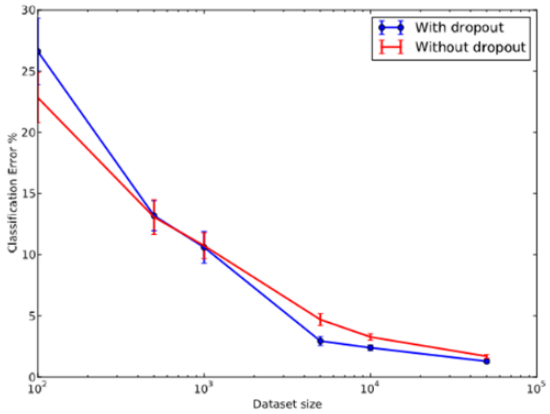


(a) Standard Neural Net

(b) After applying dropout.

Dropout

- **Regularization의 기대효과**
  - **Voting 효과**
  - **Co-adaptation 방지** : 다른 파라미터와 같이 cost function을 minimize하면 파라미터간의 동조화 현상(Co-adaptation)이 일어날 수 있으므로 Dropout을 하여 뉴런이 서로 의지하던 것을 막아서 더욱 의미있는 feature를 끄집어 낼 수 있음
  - hidden 뉴런들의 활성도(activity)를 sparse하게 만듬으로 hidden layer에 있는 중요하지 않은 뉴런의 수를 줄여주는 효과가 있음



=> 결론적으로 선명한 특질(salient feature)을 끌어낼 수 있음

Dropout

#dropout을 얼마나 시킬 것인지에 대한 placeholder 설정

keep_prob = tf.placeholder(tf.float32)

(생략)

#layer에 dropout 설정

l1 = tf.sigmoid(tf.matmul(X, W1) + b1)

l1 = tf.nn.dropout(l1, keep_prob)

(생략)

#학습 할 때 dropout 얼마나 시킬 것인지 설정 ex) 0.75 -> 75% node를 활성화 시키겠다.

_, cost_val = sess.run([optimizer, cost], feed_dict={X:batch_x, Y:batch_y, keep_prob:0.75})

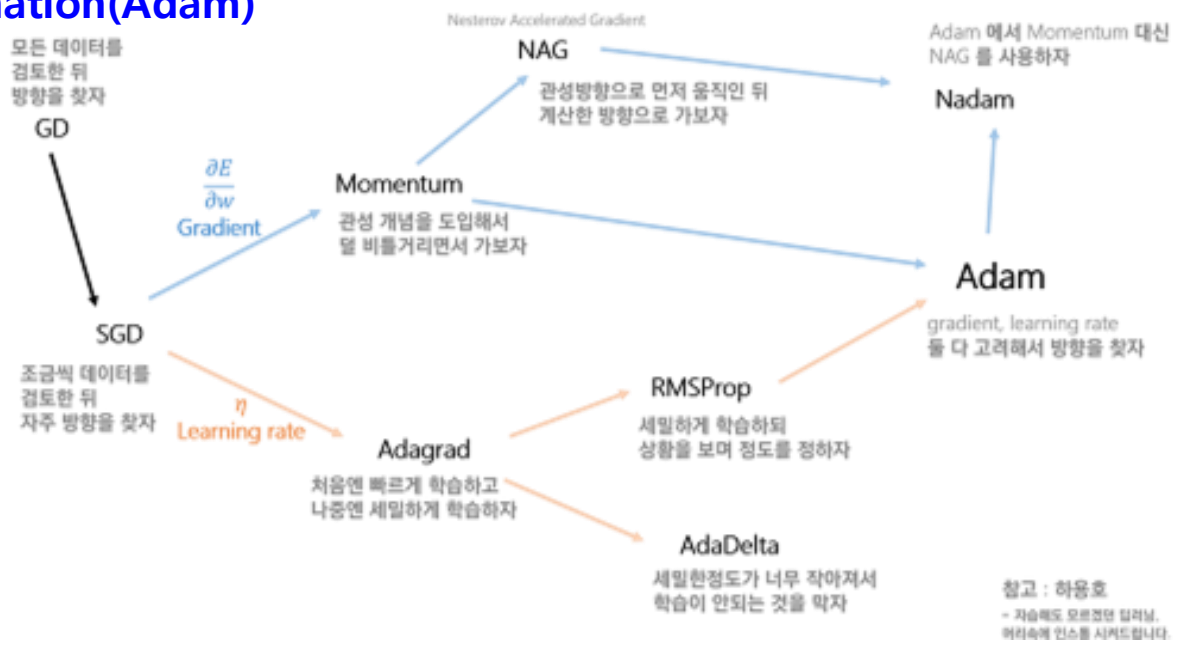
(생략)

#테스트 할 때 dropout을 적용 안 할 것이다 -> keep_porb: 1

sess.run(accuracy, feed_dict={X:mnist.test.images, Y:mnist.test.labesl, keep_prob:1})

optimizer

- Gradient Descent(GD)
- Stochastic Gradient Descent (SGD)
- Momentum
- Nesterov Accelerated Gradient (NAG)
- Adagrad
- RMSProp
- AdaDelta
- Adaptive Moment Estimation(Adam)



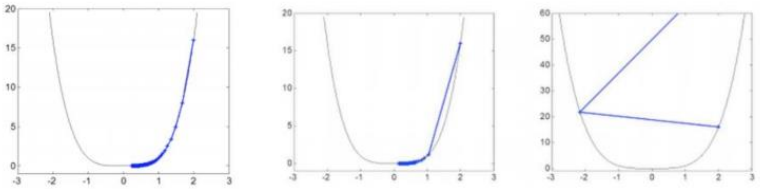
Optimizer 1 : Gradient Descent(GD)

- 개념

1회의 학습 step시에, 현재 모델의 모든 data에 대해서 예측 값에 대한 loss 미분을 learining rate 만큼 보정해서 반영하는 방법으로 gradient의 반대 방향으로 일정 크기만큼 이동해내는 것을 반복하여 Loss function J(θ) 의 값을 최소화하는 θ 의 값을 찾음

- 함수

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta)$$



$\eta = 0.01$ $\eta = 0.03$ $\eta = 0.13$

- 텐서코드

```
# GD
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

Args:

- `learning_rate` : A Tensor or a floating point value. The learning rate to use.
- `use_locking` : If True use locks for update operations.
- `name` : Optional name prefix for the operations created when applying gradients. Defaults to "GradientDescent".

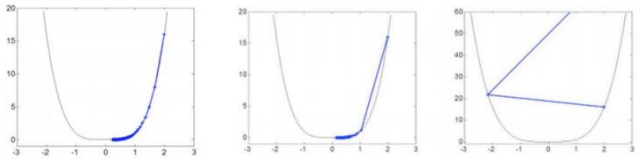
Optimizer 2 : Stochastic Gradient Descent (SGD)

- 개념

한번 step을 내딛을 때 전체 데이터에 대한 Loss Function을 계산하므로 매우 느리므로 이를 방지하기 위해 일부의 data sample이 전체 data set의 gradient와 유사할 것이라는 가정하에 일부에 대해서만 loss function을 계산

- 함수

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$



$\eta = 0.01$ $\eta = 0.03$ $\eta = 0.13$

- 텐서코드

```
# SGD
optimizer = tf.keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False).minimize(cost)
```

Arguments:

- `lr` : float >= 0. Learning rate.
- `momentum` : float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- `decay` : float >= 0. Learning rate decay over each update.
- `nesterov` : boolean. Whether to apply Nesterov momentum.

Optimizer 3 : Adaptive Moment Estimation (Adam)

- 개념

Adam 지금까지 계산해온 기울기의 지수평균을 저장하여 기울기의 제곱값의 지수평균을 저장한다.
학습에 초반부에 m과 v가 0에 가깝게 bias되어 있을 것이라고 판단해 unbiased 작업을 거친 후에 계산.

- 함수

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta)$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta))^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

- 텐서코드

```
# Adam
optimizer = tf.train.AdamOptimizer(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam').minimize(cost)
```

Args:

- `learning_rate` : A Tensor or a floating point value. The learning rate.
- `beta1` : A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
- `beta2` : A float value or a constant float tensor. The exponential decay rate for the 2nd moment estimates.
- `epsilon` : A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper.
- `use_locking` : If True use locks for update operations.
- `name` : Optional name for the operations created when applying gradients. Defaults to "Adam".

과제

- 저번 시간에 과제로 한 MNIST classifier에 dropout과 adam optimizer 적용하기

- 소스와 결과 캡쳐, GitLab에 제출
- 과제기한 : 다음주 수요일 23:59 까지
- 수업시간에 한 경우 바로 검사 받고 gitlab 제출