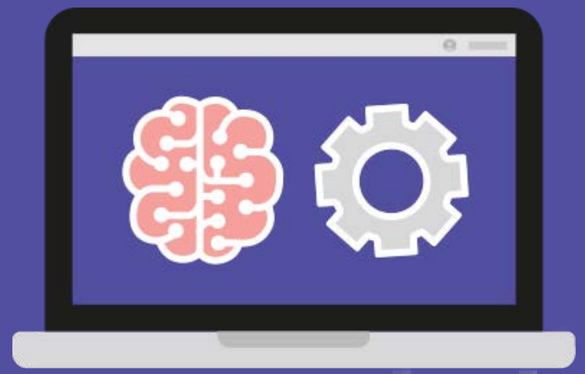
/* elice */

양재 AI School 인공지능 캠프

Lecture 14

최적화(Optimization)



김도경 선생님

수업 목표

1 딥러닝 학습의 문제점과 해결방법

1. 기울기 소실(Gradient Vanishing) => 활성화 함수
2. 가중치 초기화
3. 과적합(Overfitting)

   => 정규화, 드랍아웃(Dropout) (다음 시간에)

   배치 정규화(Batch Normalization) (추후에)

2 최적화(Optimization) 알고리즘

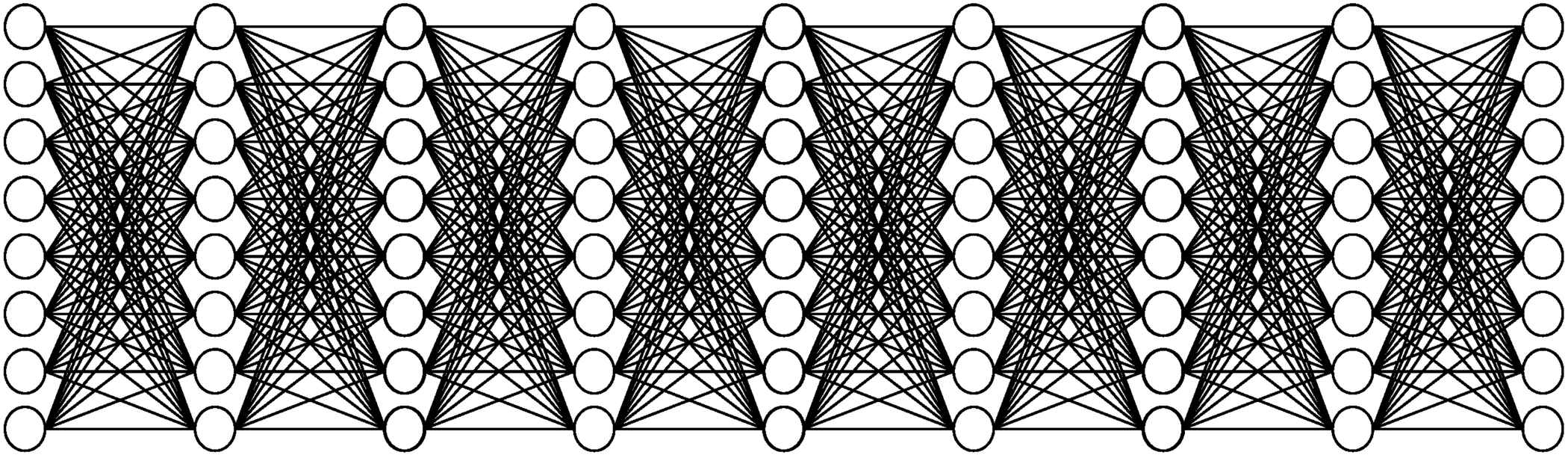
1. 알고리즘의 종류

   GD, SGD, Momentum, AdaGrad, RMSProp, Adam 등
2. 알고리즘 구현(코드)

1. 딥러닝 학습의 문제점과 해결방법

1. 딥러닝 학습의 문제점과 해결방법

- 딥러닝이 나왔으니 복잡한 문제도 층을 더 깊게 하고, 더 넓게 하면 다 풀수 있겠구나!



1. 딥러닝 학습의 문제점과 해결방법

- 하지만 학습이 잘 안되는걸?

원인

1. 기울기 소실(잘못된 활성화 함수)
2. 잘못된 초기값 설정
3. 과적합
4. 불안정한 학습 과정

해결책

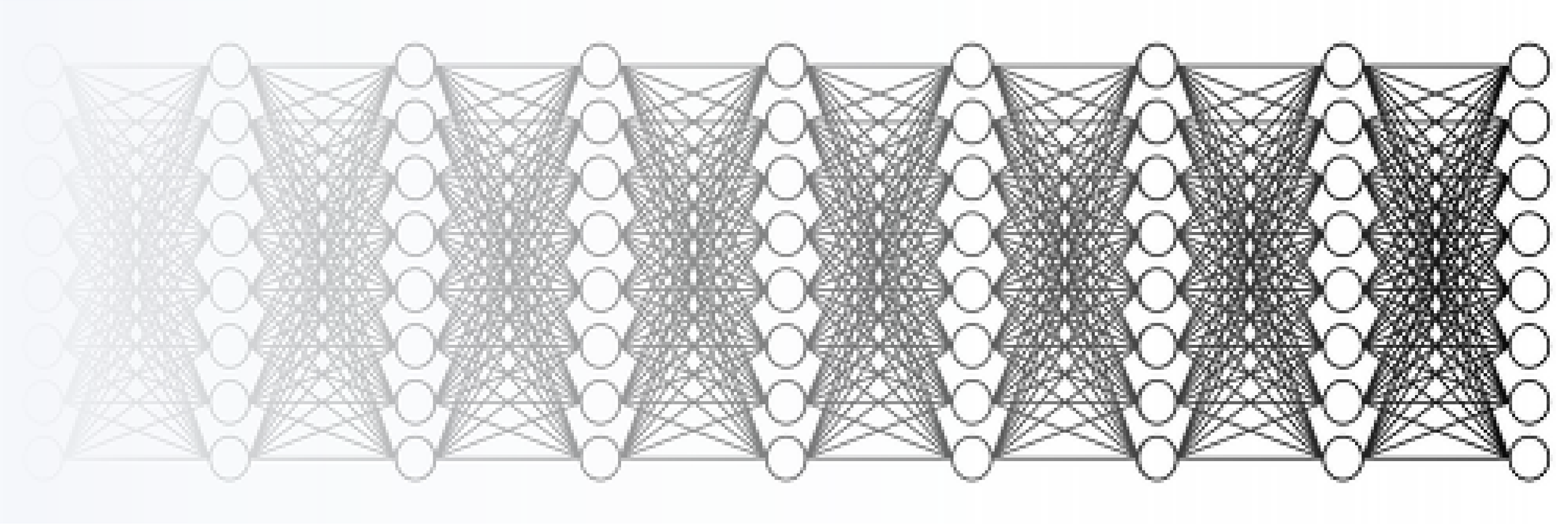
1. Sigmoid => ReLU(2006)
2. Xavier Initialization(2010), He Initialization(2015)
3. 정규화, Dropout(2014)
4. Batch Normalization(2015)

1-1. 기울기 소실(Gradient Vanishing)

- **기울기 소실(Gradient Vanishing)**

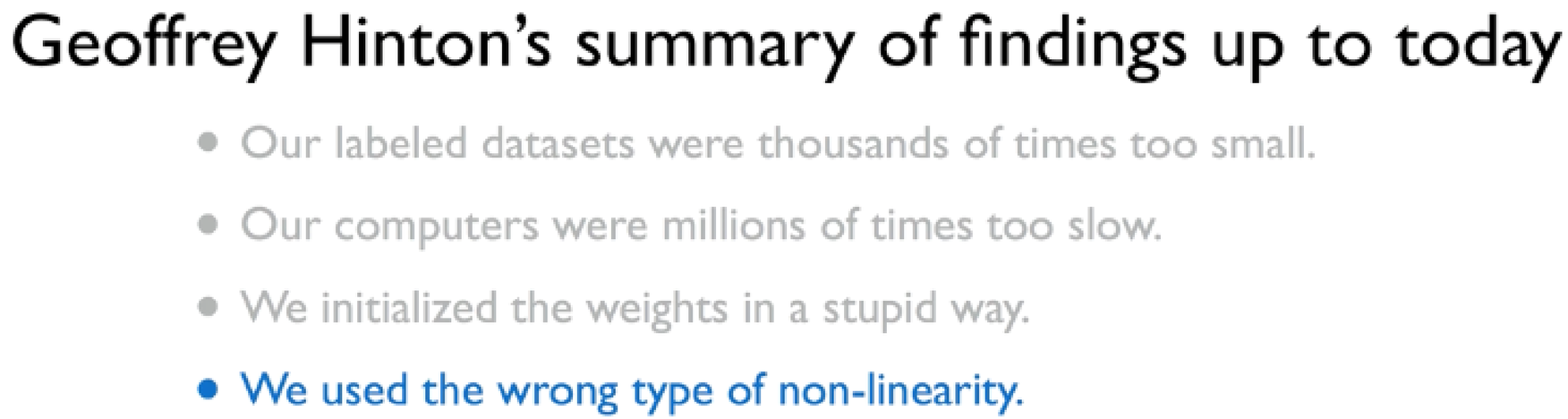
  더 깊고 더 넓은 망을 학습시키는 과정에서 Output 값과 멀어질 수록 학습이 잘 안되는 현상

  => 다시 인공신경망 연구에 침체기(제 2의 겨울, 1986~2006)가 도래



1-1. 기울기 소실(Gradient Vanishing)

- 2006년, 힌튼이 “Wrong type of non-singularity”를 사용하고 있었다며 Sigmoid 함수 대신 ReLU(Rectified Linear Unit)을 사용할 것을 제안



- 내부 hidden layer에는 ReLU를 적용하고, output layer에서만 Tanh 함수를 적용하면 정확도가 훨씬 올라감

1-1. 기울기 소실(Gradient Vanishing)

- Sigmoid

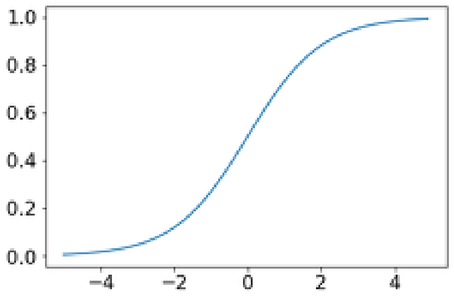
  도함수의 최대값은 0.25

  => 망이 깊어질 수록 Gradient가 1/4 씩 줄어듬

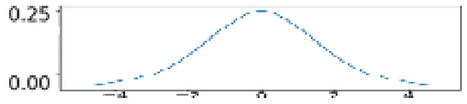
- **ReLU**

  도함수 값이 0이나 1이기 때문에 컴퓨팅 측면에서 경제적

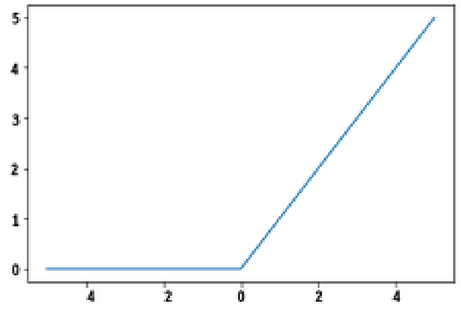
  최대값이 1보다 큰 값도 가능하기 때문에 학습이 빠르다는 장점



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

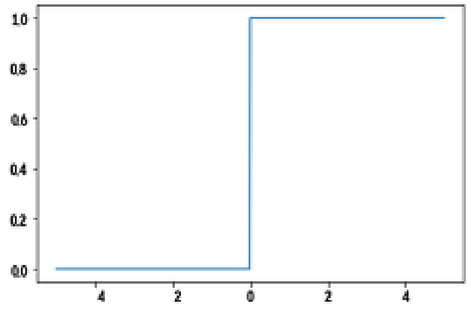


$$S'(x) = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)$$



$$ReLU(x) = \max(0,\ x)$$

Rectified Linear Unit

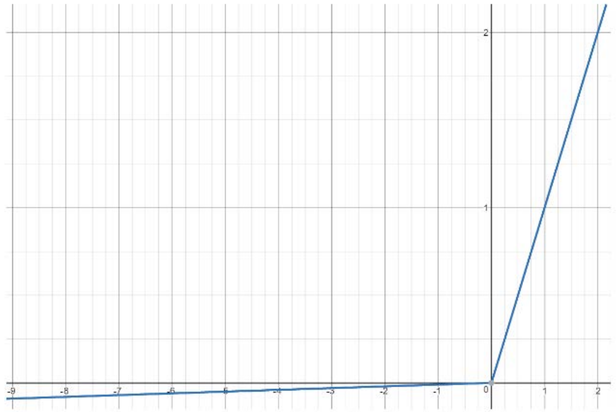


$$R'(y) = \begin{cases} 1 & (x \geq 0) \\ 0 & (x < 0) \end{cases}$$

1-1. 기울기 소실(Gradient Vanishing)

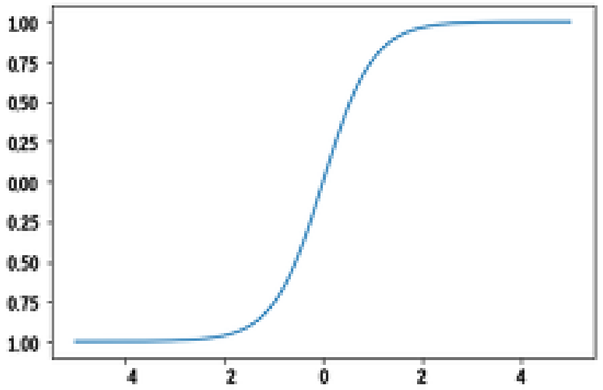
- Leaky ReLU

  0보다 작은 경우 ReLU에서 신경이 죽어버리는 현상을 극복

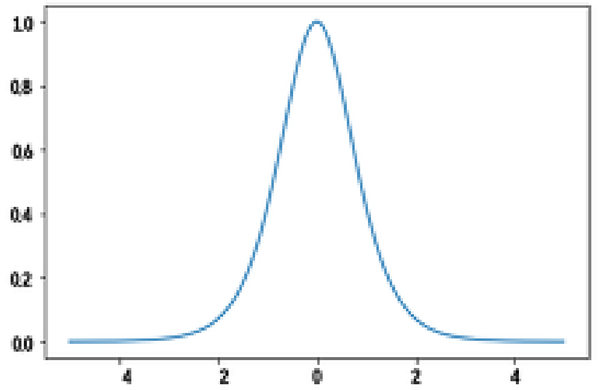


- **Tanh**

  함수값의 범위가 (-1, 1), 도함수의 최댓값이 1



$$Tanh(x) = \frac{e^{2x}+1}{e^{2x}-1}$$



$$T'(x) = 1 - \tanh^2(x)$$

1-2. 가중치 초기화

초기화의 중요성

- t=wx+b에서 w가 100, b가 50이라면 x가 0.01이더라도 t는 51이 됨
  역전파시 시그모이드 함수를 통과시키면 σ′(51)가 반환
  t가 5만 넘어도 σ′(t)는 0에 수렴 => 기울기 소실 문제 발생
- 입력층의 가중치 W를 모두 0으로 초기화한다면?
  순전파 때 두번째 층의 뉴런에 모두 같은 값이 전달됨
  역전파 때 두번째 층의 가중치가 모두 똑같이 갱신
  => 신경망의 표현력을 제한
- Bias는 0으로 초기화 하는 것이 일반적으로 효율적

1-2. 가중치 초기화

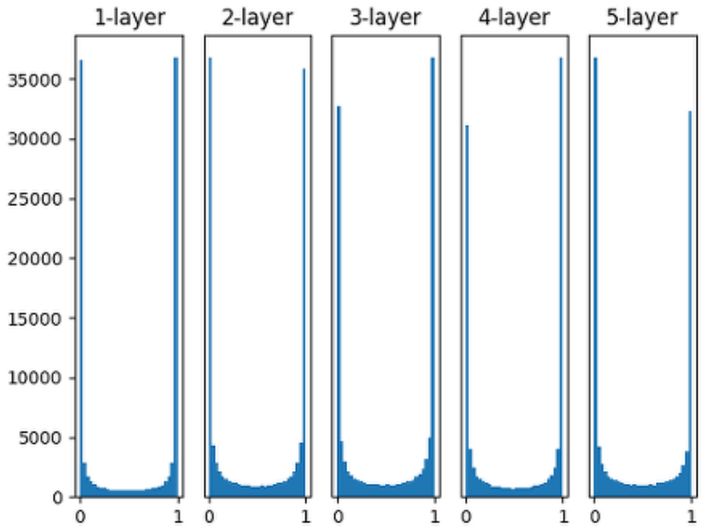
- 가중치를 표준 정규분포를 이용해 초기화

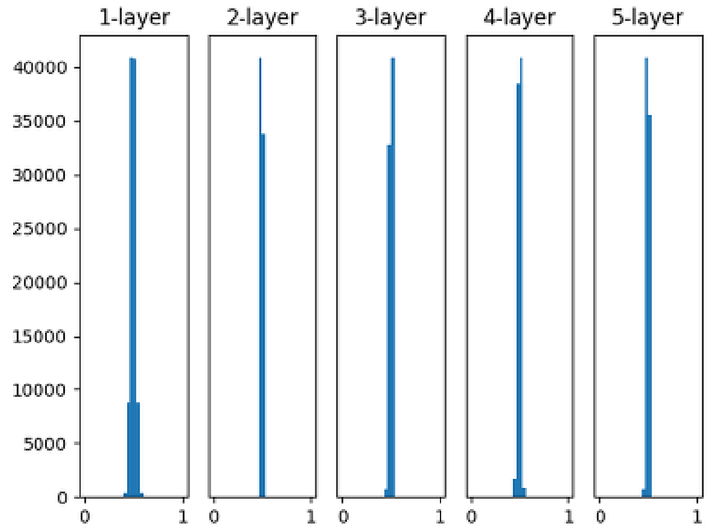
  Sigmoid 함수의 출력값이 0 과 1에 치우치는 현상이 발생

  => 기울기 소실 문제 발생

- 표준편차를 0.01로 하는 정규분포로 초기화(Naïve)

  가중치가 0.5 중심으로 모여 있음 => 기울기 소실 문제 완화





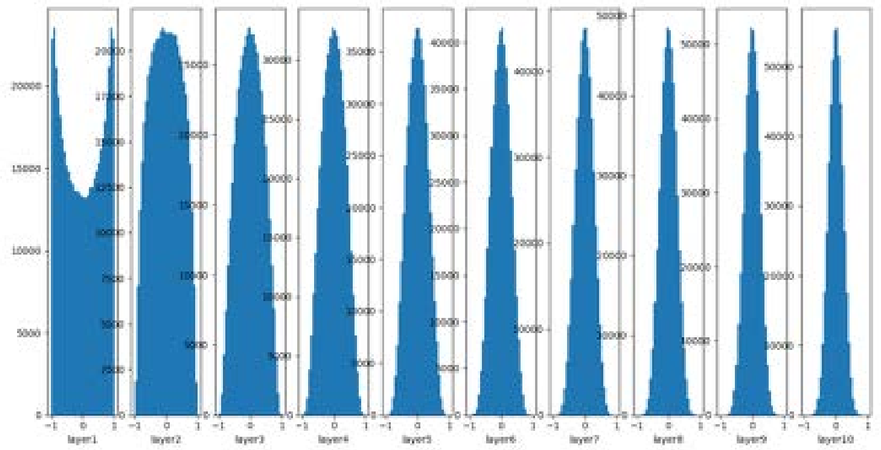
1-2. 가중치 초기화

Xavier 초기화 방법(2010)

- 표준 정규 분포를 입력 개수의 제곱근으로 나누어 줌

```
w = np.random.randn(n_input, n_output) / sqrt(n_input)
```

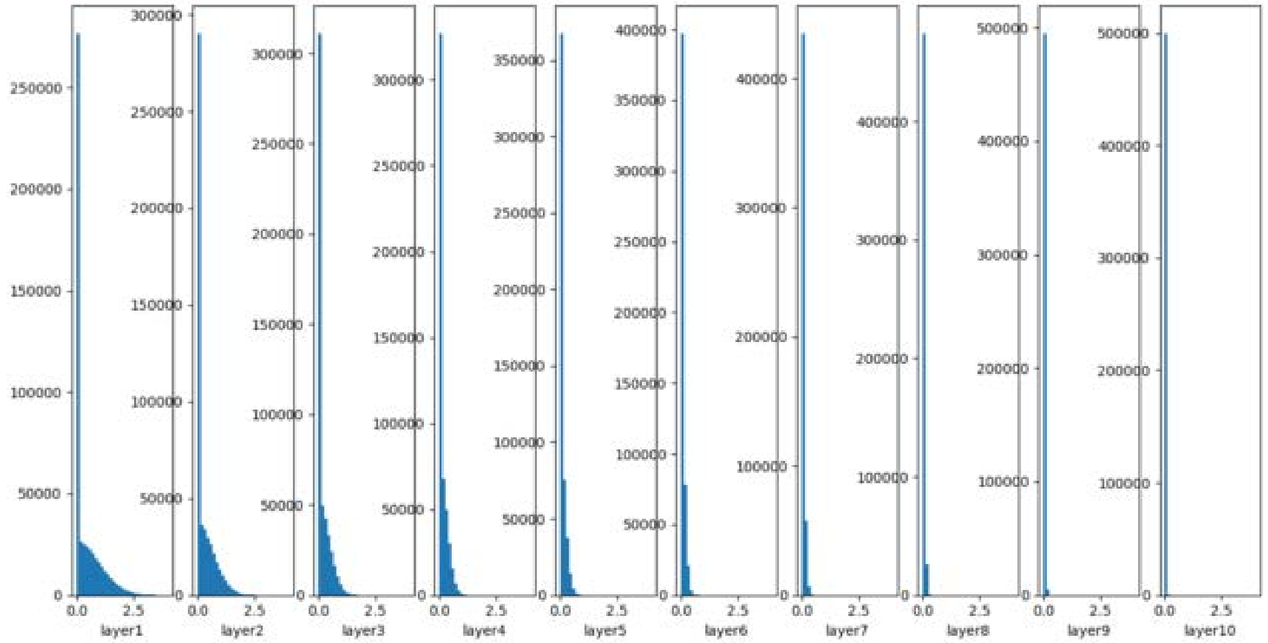
- Sigmoid와 같은 S자 함수의 경우 출력값들이 정규 분포 형태를 가져야 안정적으로 학습 가능
- Sigmoid 함수와 Xavier 초기화 방법을 사용했을 경우의 그래프



1-2. 가중치 초기화

ReLU 함수에는 Xavier 초기화가 부적합

- ReLU 함수와 Xavier 초기화 방법을 사용했을 경우의 그래프
  레이어를 거쳐갈수록 값이 0에 수렴



1-2. 가중치 초기화

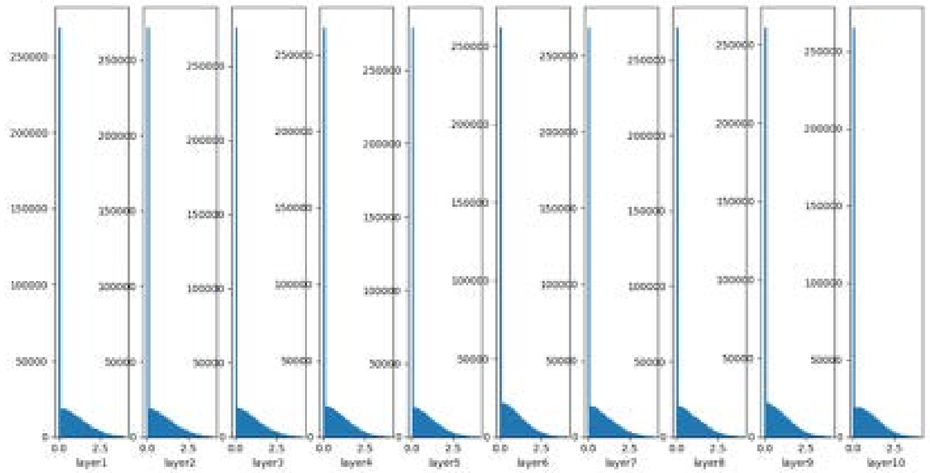
He 초기화 방법(2015)

- 표준 정규 분포를 입력 개수 절반의 제곱근으로 나누어 줌

```
w = np.random.randn(n_input, n_output) / sqrt(n_input / 2)
```

- ReLU 함수와 He 초기화 방법을 사용했을 경우의 그래프

  10층 레이어에서도 평균과 표준편차가 0 으로 수렴하지 않음



1-2. 가중치 초기화

요약

- 가중치 초기화는 매우 중요함

- Sigmoid, tanh의 경우 Xavier 초기화 방법이 효율적
- ReLU계의 활성화 함수 사용 시
  Xavier 초기화보다는 He 초기화 방법이 효율적
- 최근의 대부분의 모델에서는 He초기화를 주로 선택

2. 최적화(Optimization) 알고리즘

2. 최적화(Optimization) 알고리즘

- **경사하강법(Gradient Descent)**

  1. 네트워크의 parameter들을 θ 라고 했을 때, 손실함수 J(θ)의 값을 최소화하기 위해 기울기(gradient) ∇J(θ)를 이용하는 방법

  2. GD에서는 gradient의 반대 방향으로 일정 크기만큼 이동하는 것을 반복하여 손실함수의 값을 최소화하는 θ의 값을 찾음

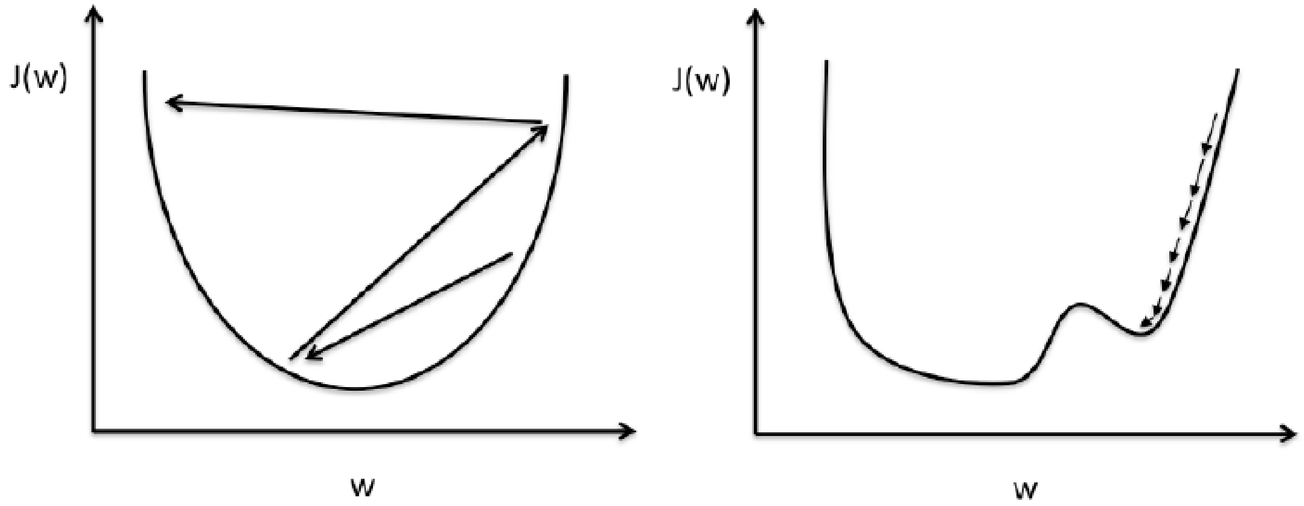
$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

  3. 이 때 η는 미리 정해진 걸음의 크기(step size)로, 학습률(learning rate)이라고 함. 보통 0.01~0.001 정도를 사용

2. 최적화(Optimization) 알고리즘

학습률(Learning rate)

- 학습률이 너무 크면 듬성듬성하고 최소값(global minimum)을 지나쳐 갈수 있음
- 학습률이 너무 작으면 학습을 촘촘히 해서 학습속도가 느려지고 극소값(Local minimum)에 빠질 수 있음



2-1. 알고리즘의 종류

- **학습 알고리즘의 종류**

  GD(Gradient Descent)

  SGD(Stochastic Gradient Descent)

  Momentum

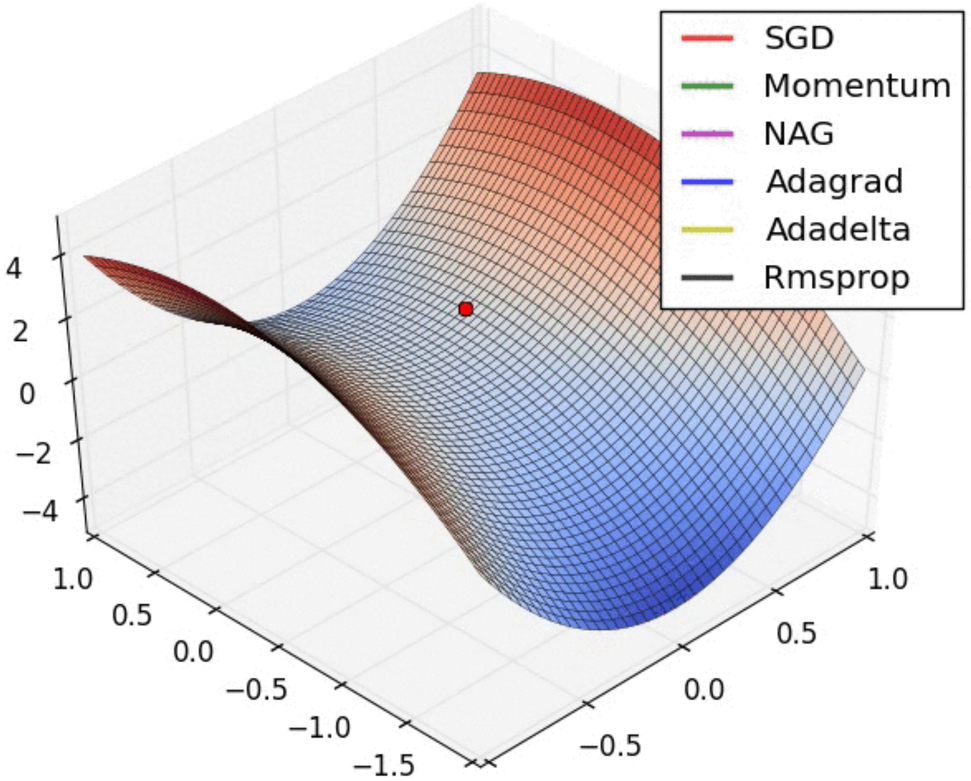
  AdaGrad

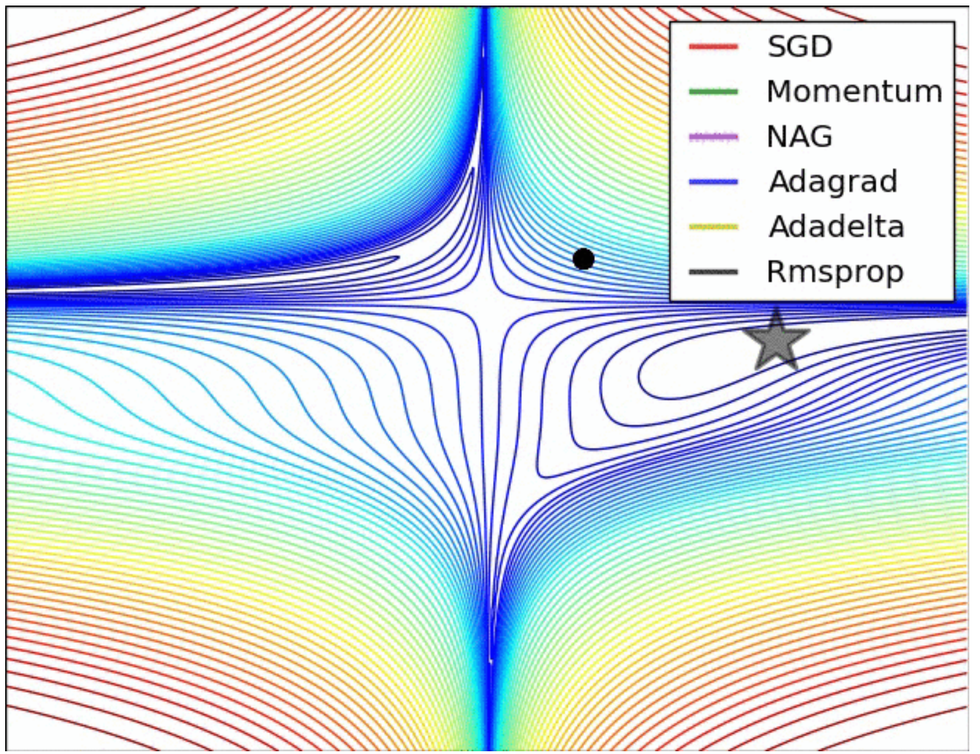
  RMSProp

  Adam

2-1. 알고리즘의 종류

- 여러 학습 알고리즘간의 **수렴속도 비교**

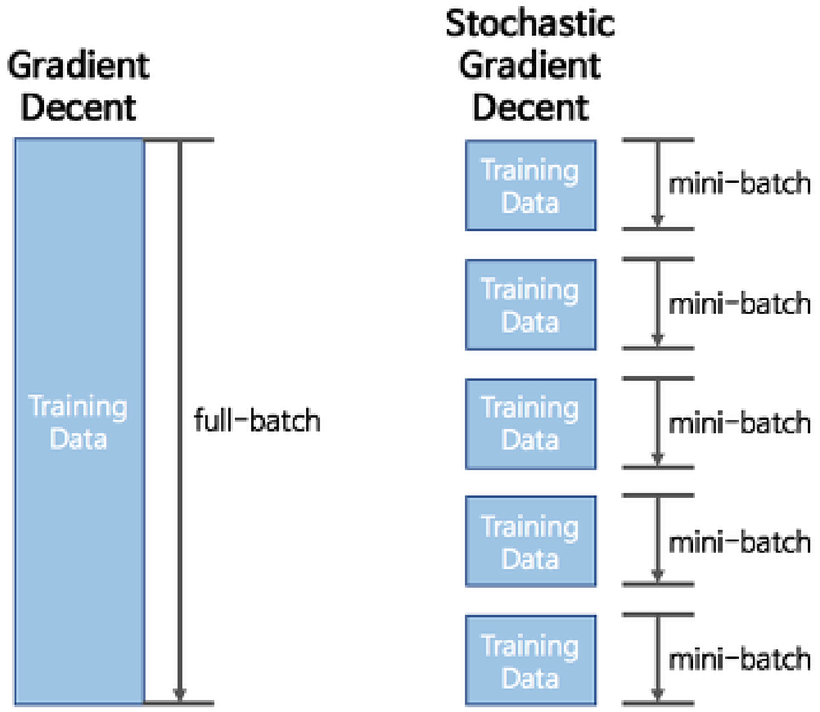




2-1. 알고리즘의 종류

SGD(Stochastic Gradient Descent)

- 손실함수를 계산할 때 전체 training set을 사용하는 것을 Batch Gradient Descent라 함
- 계산량이 너무 많아지는 것을 방지하기 위해 보통 SGD를 사용
- 전체 데이터(batch) 대신 일부 조그마한 데이터의 모음인 미니배치(mini-batch)에 대해서만 손실함수를 계산



2-1. 알고리즘의 종류

SGD(Stochastic Gradient Descent)

- 다소 부정확할 수 있지만, 훨씬 계산 속도가 빠르기 때문에 같은 시간에 더 많은 step을 갈 수 있음
- 여러 번 반복할 경우 보통 batch의 결과와 유사한 결과로 수렴

2-1. 알고리즘의 종류

- **GD**

  모든 데이터를 계산 => 1시간

  6 스텝 * 1시간 = 6시간

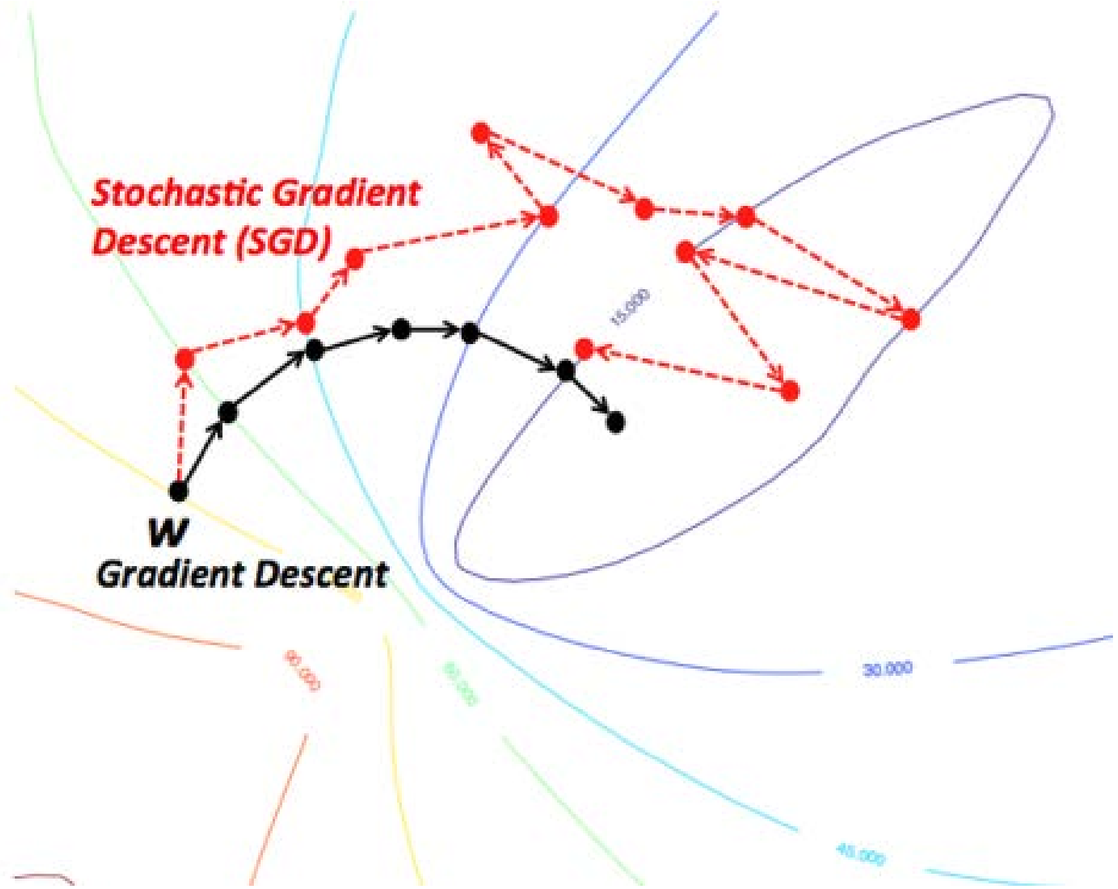
  확실한데 너무 느림

- **SGD**

  일부 데이터만 계산 => 5분

  10 스텝 * 5분 => 50분

  조금 헤메지만 빠름



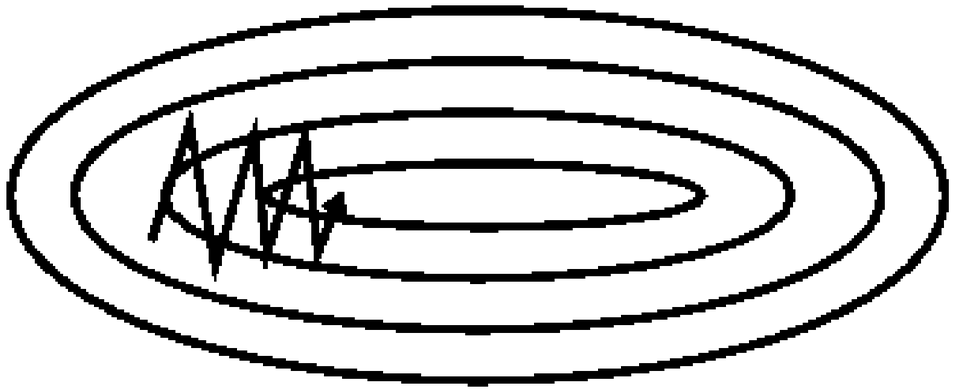
2-1. 알고리즘의 종류

Momentum

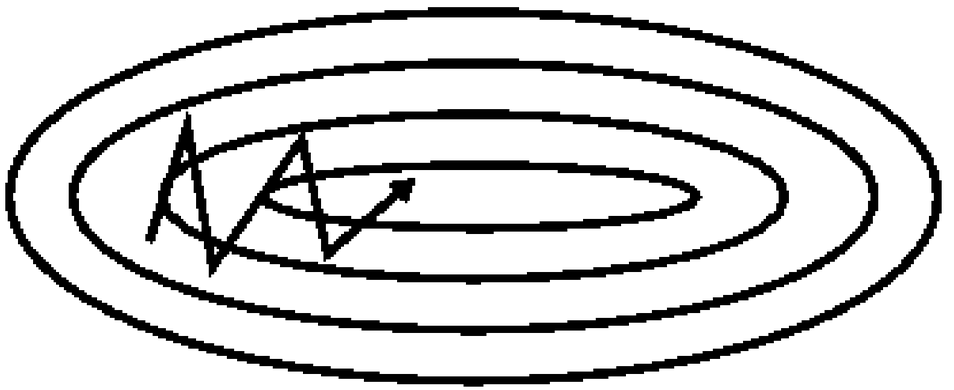
- 현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식, 즉 일종의 관성을 주는 방식

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

$$\theta = \theta - v_t$$



without momentum



with momentum

2-1. 알고리즘의 종류

AdaGrad(Adaptive Gradient)

- 기본적인 아이디어는 '많이 변화하지 않은 변수들은 step size를 크게 하고, 많이 변화했던 변수들은 step size를 작게 하는 것'

  1. 자주 등장하거나 변화를 많이 한 변수들은 optimum에 가까이 있을 확률이 높기 때문에 작은 크기로 이동하면서 세밀하게 조절

  2. 적게 변화한 변수들은 많이 이동해야할 확률이 높기 때문에 먼저 빠르게 loss 값을 줄이는 방향으로 이동하려는 방식

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

- 학습을 계속 진행하면 step size가 너무 줄어든다는 문제점이 있음

2-1. 알고리즘의 종류

RMSProp

- 제프리 힌튼(Geoffrey Hinton)이 제안
- Adagrad의 단점을 해결하기 위해 합을 지수평균으로 대체
- G가 무한정 커지지는 않으면서 최근 변화량의 변수간 상대적인 크기 차이는 유지할 수 있음

$$G = \gamma G + (1-\gamma)(\nabla_\theta J(\theta_t))^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{G+\epsilon}} \cdot \nabla_\theta J(\theta_t)$$

2-1. 알고리즘의 종류

Adam(Adaptive Moment Estimation)

- Momentum과 RMSProp을 합친 알고리즘

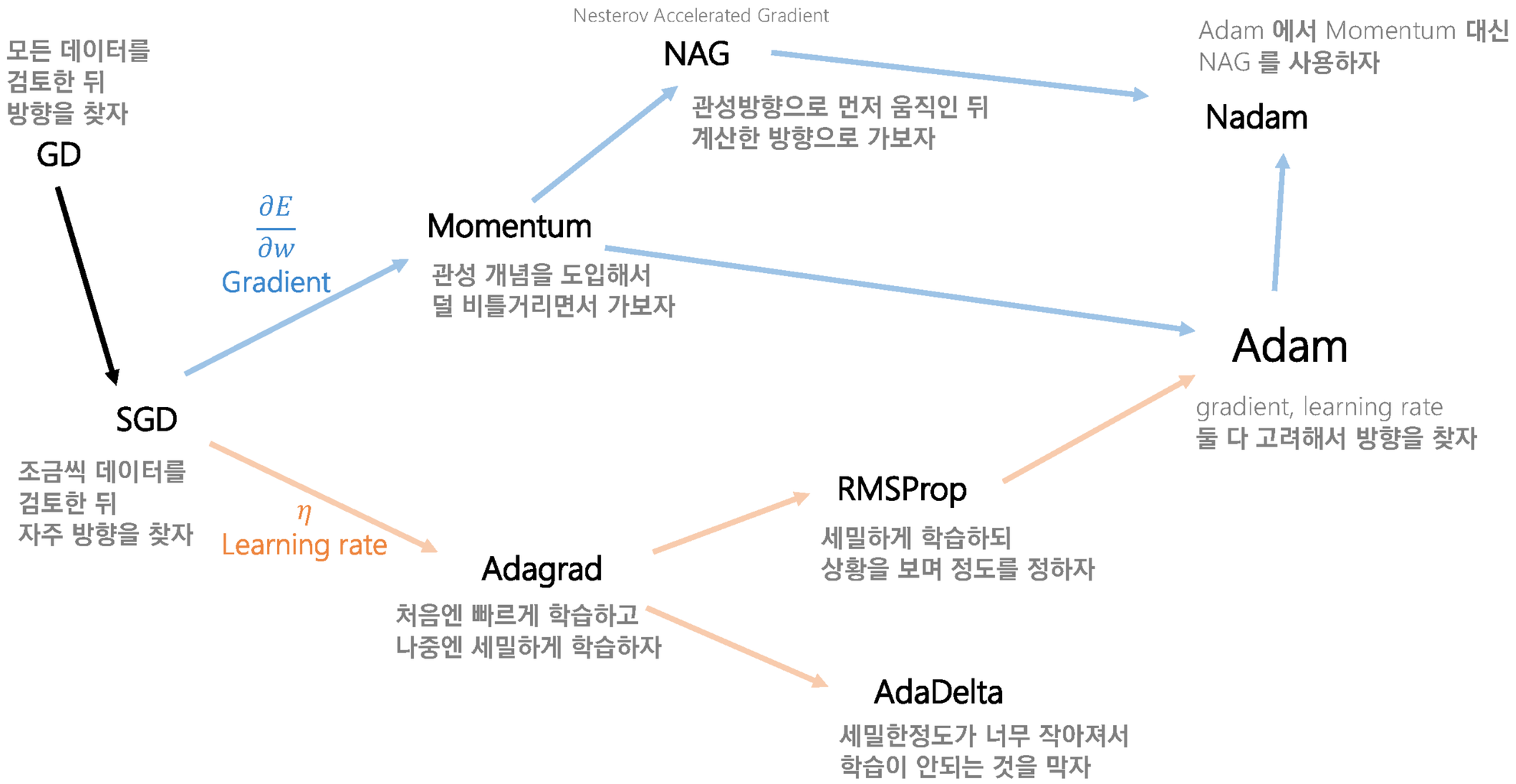
  1. Momentum 방식과 유사하게 지금까지 계산해온 기울기의 지수평균을 저장

  2. RMSProp과 유사하게 기울기의 제곱값의 지수평균을 저장

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

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

2-1. 알고리즘의 종류


모든 데이터를
검토한 뒤
방향을 찾자
GD
SGD
조금씩 데이터를
검토한 뒤
자주 방향을 찾자
$\frac{\partial E}{\partial w}$
Gradient
η
Learning rate
Momentum
관성 개념을 도입해서
덜 비틀거리면서 가보자
Nesterov Accelerated Gradient
NAG
관성방향으로 먼저 움직인 뒤
계산한 방향으로 가보자
Adagrad
처음엔 빠르게 학습하고
나중엔 세밀하게 학습하자
RMSProp
세밀하게 학습하되
상황을 보며 정도를 정하자
AdaDelta
세밀한정도가 너무 작아져서
학습이 안되는 것을 막자
Adam 에서 Momentum 대신
NAG 를 사용하자
Nadam
Adam
gradient, learning rate
둘 다 고려해서 방향을 찾자

2-2. 알고리즘 구현(코드)

```
from scratch.dataset.mnist import load_mnist
from scratch.common.util import smooth_curve
from scratch.common.multi_layer_net import MultiLayerNet
from scratch.common.optimizer import *
import matplotlib.pyplot as plt


# 0. MNIST 데이터 읽기==========
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True)


train_size = x_train.shape[0]
batch_size = 128
max_iterations = 2000
```

2-2. 알고리즘 구현(코드)

```
# 1. 실험용 설정==========
optimizers = {}
optimizers['SGD'] = SGD()
optimizers['Momentum'] = Momentum()
optimizers['AdaGrad'] = AdaGrad()
optimizers['Adam'] = Adam()
# optimizers['RMSprop'] = RMSprop()

networks = {}
train_loss = {}
for key in optimizers.keys():
    networks[key] = MultiLayerNet(
        input_size=784, hidden_size_list=[100, 100, 100, 100],
        output_size=10)
    train_loss[key] = []
```

2-2. 알고리즘 구현(코드)

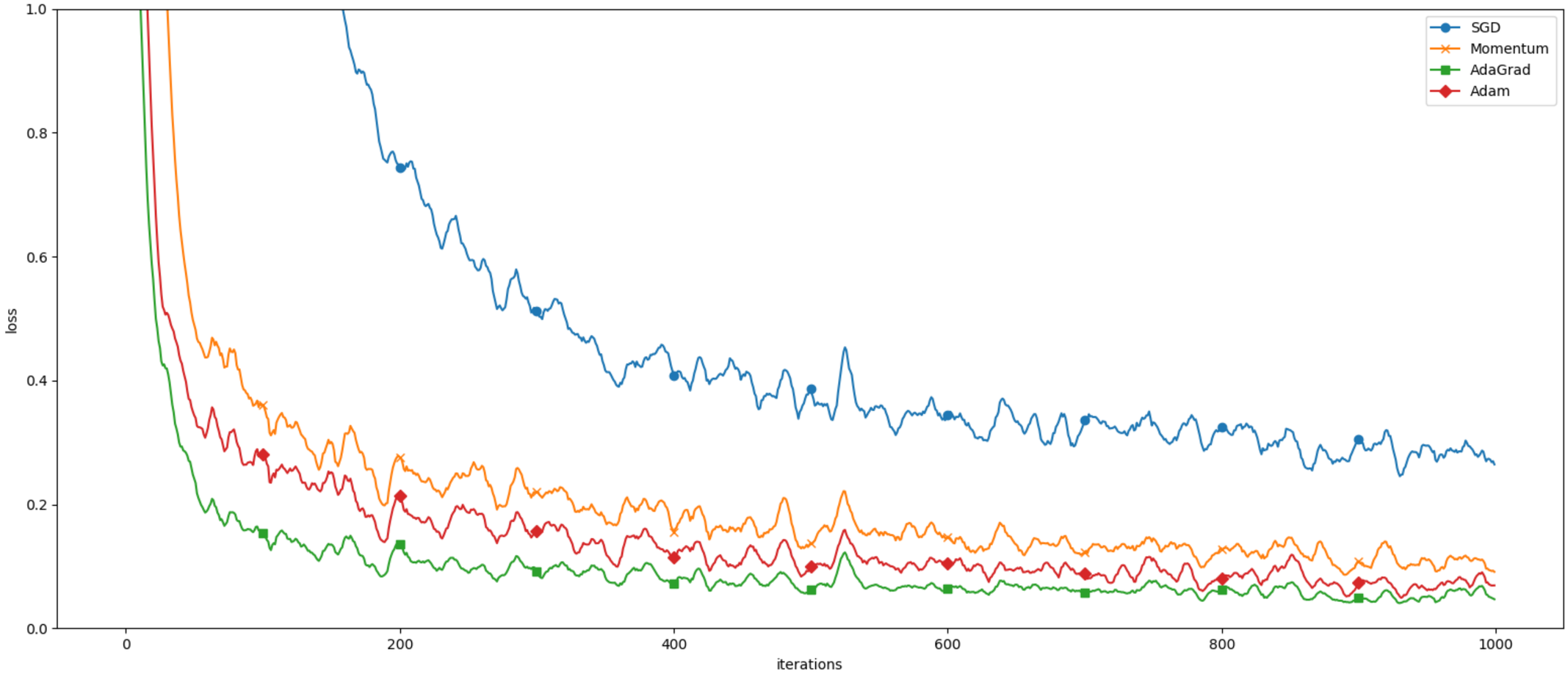
```
# 2. 훈련 시작==========
for i in range(max_iterations):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    for key in optimizers.keys():
        grads = networks[key].gradient(x_batch, t_batch)
        optimizers[key].update(networks[key].params, grads)

        loss = networks[key].loss(x_batch, t_batch)
        train_loss[key].append(loss)

    if i % 100 == 0:
        print("===========" + "iteration:" + str(i) + "===========")
        for key in optimizers.keys():
            loss = networks[key].loss(x_batch, t_batch)
            print(key + ":" + str(loss))
```

2-2. 알고리즘 구현(코드)

```
# 3. 그래프 그리기==========
markers = {"SGD": "o", "Momentum": "x", "AdaGrad": "s", "Adam": "D"}
x = np.arange(max_iterations)
plt.figure(figsize=(20, 8))
for key in optimizers.keys():
    plt.plot(x, smooth_curve(train_loss[key]), marker=markers[key], markevery=100, label=key)
plt.xlabel("iterations")
plt.ylabel("loss")
plt.ylim(0, 1)
plt.legend()
plt.show()
```

2-2. 알고리즘 구현(코드)

- 실행결과



참고자료

1. 활성화 함수
   https://gomguard.tistory.com/183

2. 가중치 초기화
   https://gomguard.tistory.com/184

3. 학습 알고리즘
   1. http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html
   2. https://seamless.tistory.com/38
   3. http://intelliz.co.kr/?p=196 - 코드

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice