Ch 8.3 – 8.5

강남웅

Gradient Descent

- Batch Gradient Descent
- Mini-batch Gradient Descent
- Stochastic Gradient Descent

Gradient Descent

- Batch Gradient Descent (BGD)
  - 모든 Training Set의 Gradient의 평균으로 update
- Mini-batch Gradient Descent (MSGD)
  - Training Set의 일부를 sampling한 뒤 (=mini-batch) gradient 평균으로 update
- Stochastic Gradient Descent (SGD)
  - Training Set중 하나를 sampling한 뒤 gradient를 구해서 update

SGD (Stochastic Gradient Decent)

- Training set의 일부를 sampling하여 Gradient Descent를 수행하는 알고리즘

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  end while

SGD (Stochastic Gradient Decent)

- Learning Rate ϵ_k (at iteration k)
  - 매 iteration마다 random sample을 하기 때문에 noise가 포함
    - noise가 global minimum에 도달해도 계속 존재 → ϵ_k를 점진적으로 줄여주는 것이 중요하다
  - Convergence condition
    - $\sum_{k=1}^{\infty} \epsilon_k = \infty$
    - $\sum_{k=1}^{\infty} {\epsilon_k}^2 < \infty$

SGD (Stochastic Gradient Decent)

- Learning Rate ϵ_k
  - Learning curve를 monitor하면서 최적의 learning rate를 구한다

  - linear schedule
    - 시작 후 τ iteration 동안

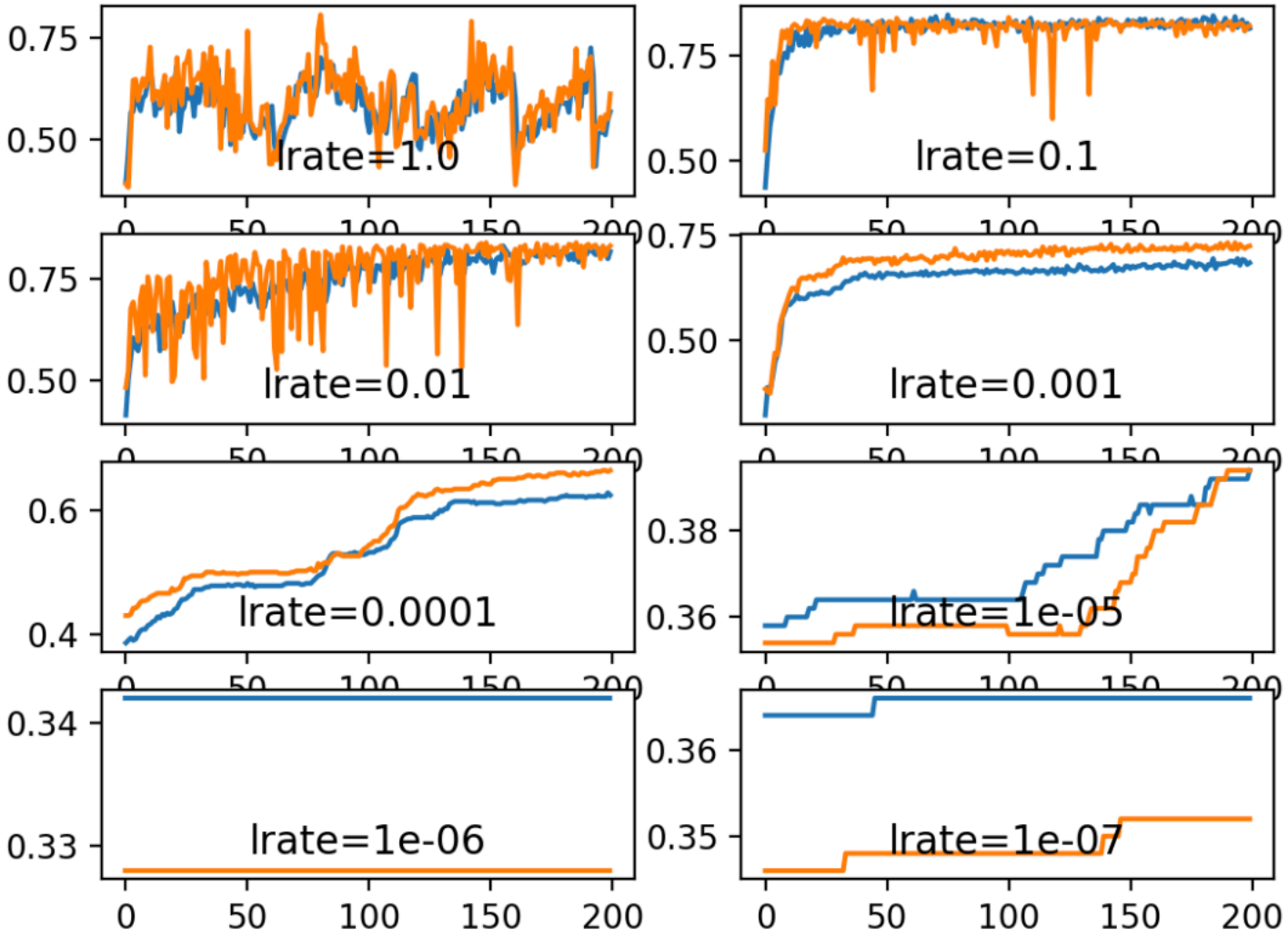
      $$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \quad where\ \alpha = \frac{k}{\tau}$$

      그 이후로는 ϵ을 고정해서 사용
      (보통 $\epsilon_\tau = \epsilon_0 * 0.01$ 로 설정)
    - ϵ_0이 크면 violent oscillation이 발생해 cost function이 급증하는 경우가 발생
    - ϵ_0이 작으면 learning이 매우 느린 속도로 진행

SGD (Stochastic Gradient Decent)

- 주황색 : Test set Acc
  파란색 : Validation Acc

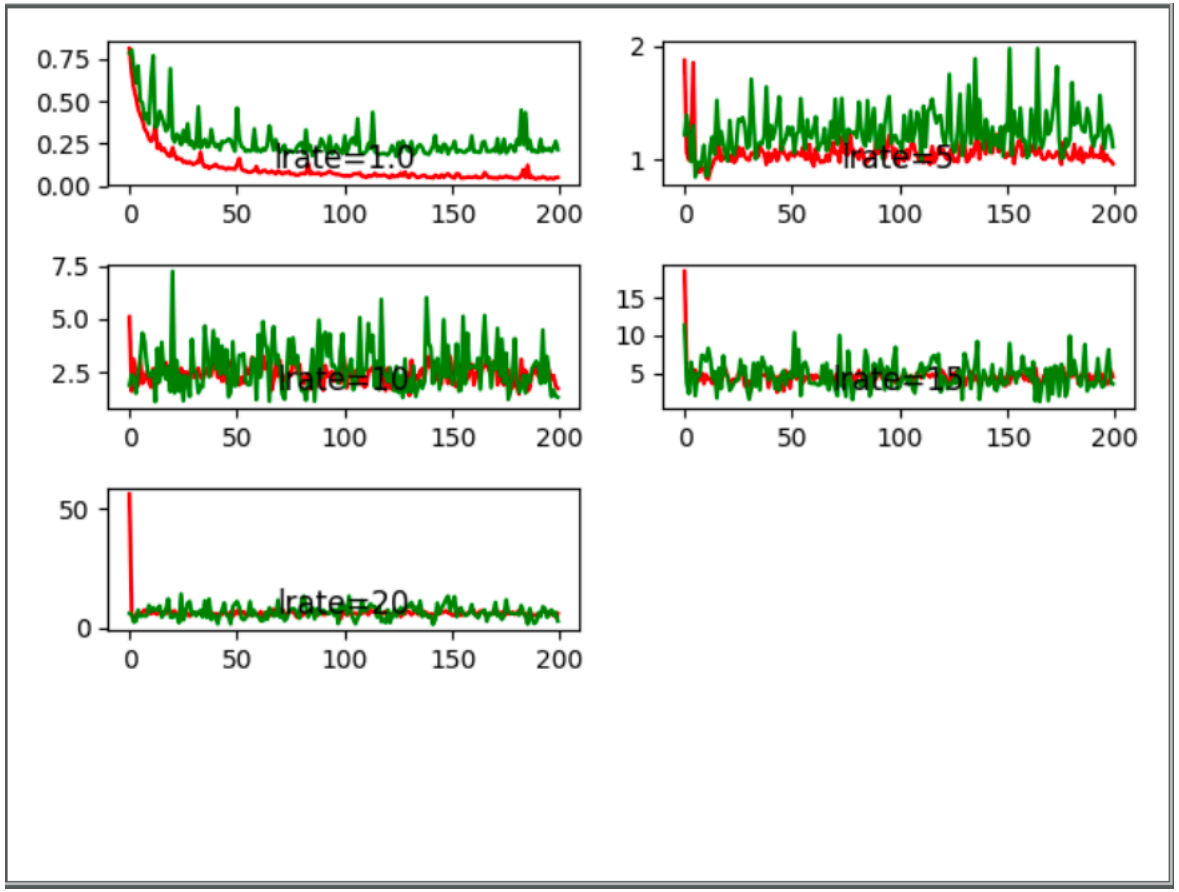
- 즉, Learning rate가 낮아지면 train이 거의 일어나지 않는다



SGD (Stochastic Gradient Decent)

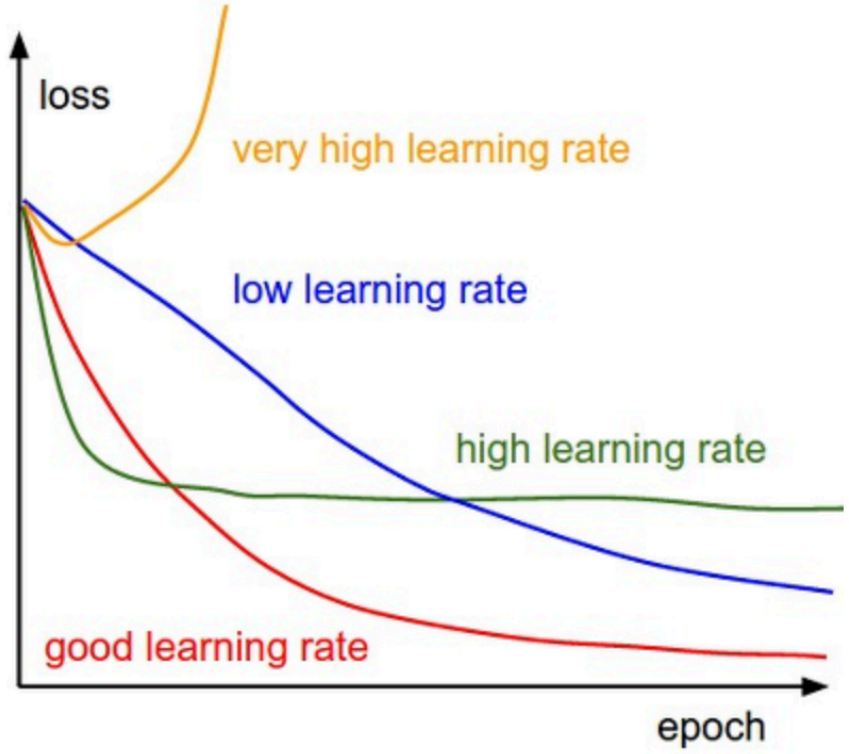
- 빨간색 : Test set loss
  초록색 : Validation loss

- 즉, Learning rate가 높아지면 loss에 차이가 심해진다



SGD (Stochastic Gradient Decent)

- loss-epoch



Gradient Descent Summary

- Batch Gradient Descent
  - 장점
    - 전체 데이터를 사용하기 때문에 optimal 로의 수렴이 안정적으로 진행
    - SGD, MGD보다 update 횟수가 적다
    - 병렬처리에 유리하다
  - 단점
    - 전체 데이터를 사용하기 때문에 느리다
    - error를 누적해야하기 때문에, memory가 많이 필요하다
    - local minimum에서 빠져나오기 힘들다

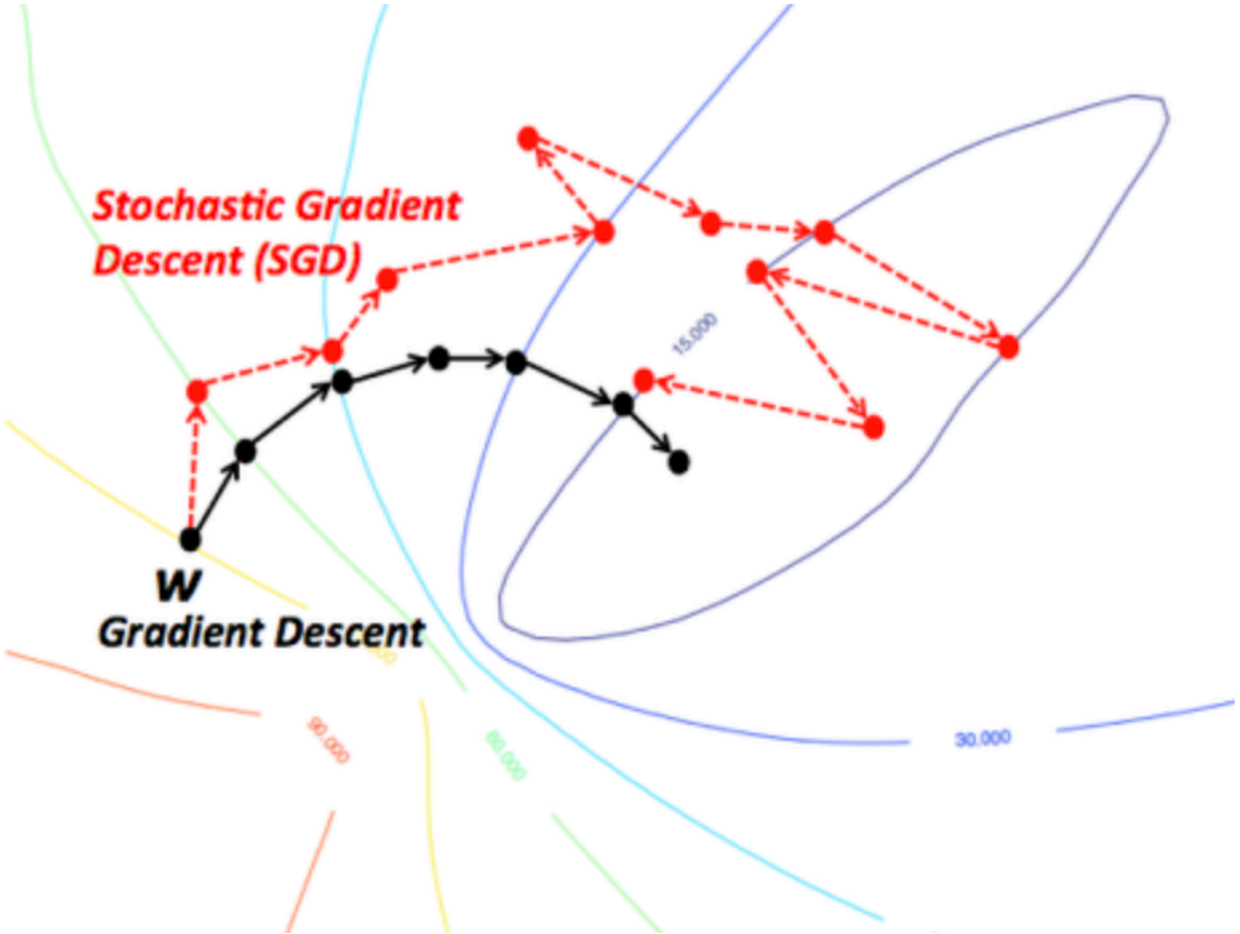
Gradient Descent Summary

- Stochastic gradient descent (SGD)
  - 장점
    - 하나의 데이터를 사용하기 때문에 매우 빠르게 진행
    - local minimum에 빠질 리스크가 낮다
    - 수렴속도가 상대적으로 빠름
  - 단점
    - global minimum을 찾지 못 할 수도 있다
    - 하나의 데이터만 사용하기 때문에 하드웨어적인 이점을 살릴 수 없다 (GPU)

Gradient Descent Summary

- Minibatch Gradient Descent
  - 장점
    - BGD보다 local minimum에 빠질 위험이 적다
    - SGD보다 효율적이다
  - 단점
    - batch size를 정해줘야 한다 (hyperparameter)

BGD vs SGD trace



Oscillation 발생

=> 느리다

Momentum

- 같은 방향으로 움직이려는 경향이 보이면, 더 빠르게 가게 해주는 알고리즘
  - Hyperparameter v (velocity) 도입

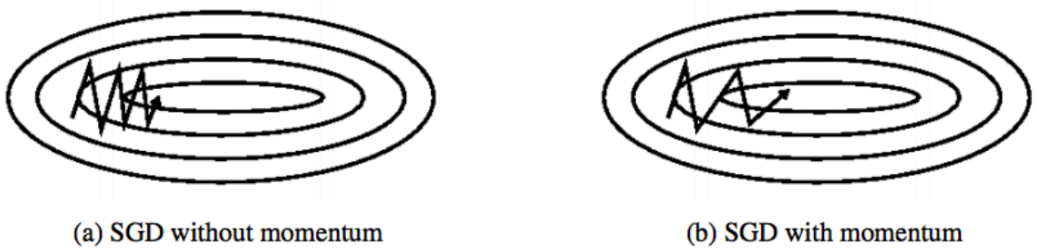
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Momentum

- Hyperparameter
  - Learning Rate ϵ
  - Contributions of previous gradients α
    (모멘트 효과에 대한 가중치; 보통 0.5, 0.9, 0.99를 사용)
  - initial velocity v

- Update Rule
  - $v_t \leftarrow \alpha v_{t-1} - \epsilon g$
  - $\theta_t \leftarrow \theta_{t-1} + v_t$



(a) SGD without momentum (b) SGD with momentum

Figure 2: Source: Genevieve B. Orr

Nesterov Momentum

- 그 전까지의 속도를 고려한 수정값의 gradient를 계산
  - 조금 더 정확한 방향으로 update

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

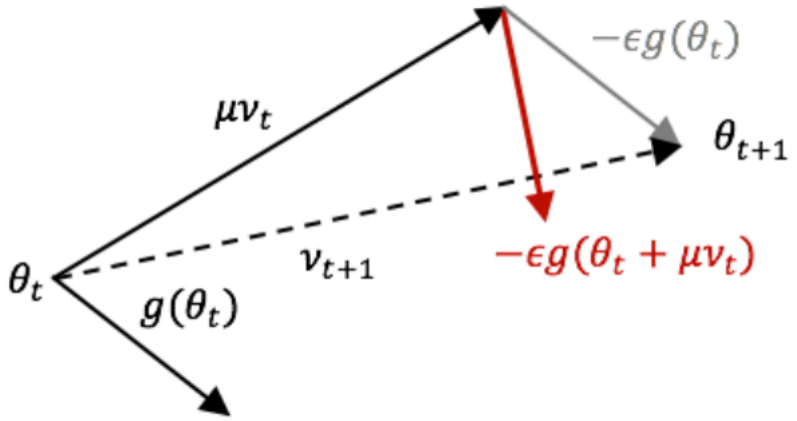
Nesterov Momentum

- Update Rule
  - $v \leftarrow \alpha v - \epsilon g'$

    $where\ g'\ is\ expectation\ of\ loss\ function\ of\ f(x^{(i)}; \theta + \alpha v)$

  - $f\left(x^{(i)}; \theta\right) \rightarrow f(x^{(i)}; \theta + \alpha v)$

    => gradient 보정효과를 얻을 수 있다



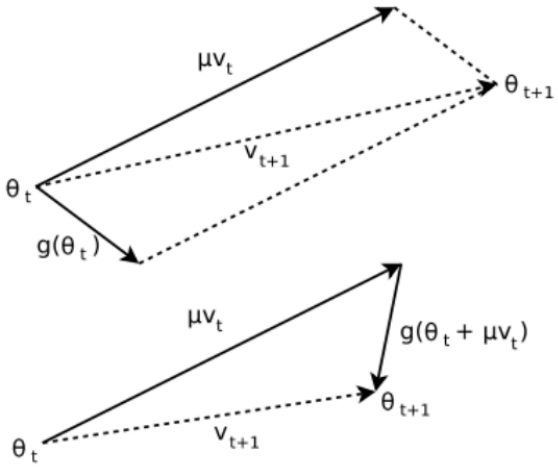


Figure 1. **(Top)** Classical Momentum **(Bottom)** Nesterov Accelerated Gradient

Parameter Initialization Strategy

- 정확하게 정착된 이론은 없다.
  - “Parameter need to break symmetry btw units”
    - 서로 다른 두 개의 hidden layer가 같은 input을 받는 경우 반드시 different initial parameter를 갖아야 한다

      => 계속 동일하게 update가 진행되기 때문

  - 주로 Weight만 random하게 initialize하고, bias를 포함한 다른 parameter들은 heuristically chosen constant를 사용한다

Parameter Initialization Strategy

- Weight를 서로 다르게 Initialize하는 방식
  - Explicitly search for set of basis function
    - Computational cost가 너무 크다
  - 대안 : Random initialization from high-entropy distribution
    - cost가 훨씬 줄어들고, 중복되는 경우가 매우 적다
    - Uniform distribution일 때 entropy가 maximize 된다.
      (Reference : https://stats.stackexchange.com/questions/66108/why-is-entropy-maximised-when-the-probability-distribution-is-uniform
      && PROBABILITY DISTRIBUTIONS AND MAXIMUM ENTROPY by. KEITH CONRAD
      https://kconrad.math.uconn.edu/blurbs/analysis/entropypost.pdf)
    - **Theorem 3.1.** *For a probability density function p on a finite set $\{x_1, \ldots, x_n\}$,*

$$h(p) \leq \log n,$$

with equality if and only if p is uniform, i.e., $p(x_i) = 1/n$ for all i.

Parameter Initialization Strategy

- Initial value of weight
  - Large initial value
    - 장점
      - Symmetry breaking effect가 커진다 => redundant unit의 생성을 피할 수 있다
      - Signal을 propagation을 거치면서 사라질 위험이 없다 (losing signal)
    - 단점
      - 너무 커지면 exploding value가 발생해 input에 너무 sensitive해진다 (Chaos)
      - activation function이 saturate 될 수 있다 => gradient loss
  - Low initial value
    - learning이 매우 느린 속도로 이뤄질 수 있다

Parameter Initialization Strategy

- Initial value of weight
  - Normalized Initialization
    - Sampling weight of fully connected layer with m inputs and n outputs
    - $W_{i,j} \sim U(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}})$
    - 유도는 linear network에서 하였지만, nonlinear에서도 잘 작용한다고 알려짐
  - Orthogonal Initialization
    - 주로 RNN을 사용할 때 이용한다
    - Weight를 Orthogonal matrix로 초기화하는 방법

Parameter Initialization Strategy

- Initial value of weight
  - Sparse Initialization
    - k개의 non-zero weight를 제외하고 모두 0으로 initialize하는 방법
    - gradient가 매우 느리게 줄어든다
  - Treat weight as hyperparameter
    - Computational resource가 충분하다면 고려할 수 있는 방법

Parameter Initialization Strategy

- Initial value of other parameters
  - Bias
    - 주로 0으로 initialize하는 경우가 많다
    - 0으로 initialize하기 불편한 경우
      - Output unit에 대한 bias일 경우 (activation function : g 일 때, g(bias) = output)
      - Initialization 단계에서 saturation을 피하고 싶은 경우
        - ReLU와 같은 함수에서 bias를 0으로 두고, weight가 sparse initialization과 같은 0이 다량 포함되어있다면, saturation이 많이 일어나기 때문
  - Mean
    - 주로 0으로 설정한다
  - Variance
    - 주로 1으로 설정한다

AdaGrad

- Concept
  - 각 변수마다 다른 성질 (sparsity, alignment)을 갖고 있기 때문에, update를 따로 시켜주는 방식
  - 전 단계에서 많이 변화하였다면, 다음 단계에서는 적게 변화하는 방식
  - Hyperparameter를 도입하지 않고 자동으로 step size를 적합하게 변화시켜주려는 방식

- Update Rule
  - Accumulation variable
    - $r \leftarrow r + g \odot g$
  - Update
    - $\theta \leftarrow \theta - \frac{\epsilon}{\delta + r^{\frac{1}{2}}} \odot g$

AdaGrad

- 장점
  - works well with sparse gradients
- 문제점
  - Accumulation variable r이 점진적으로 계속 커지게 되면, update가 이뤄지지 않는다
- 활용 예시
  - NLP 분야에서, word2vec이나 GloVe 같이 word representation을 학습시킬 경우 단어의 등장 확률에 따라 variable의 사용 비율이 확연하게 차이나기 때문에 Adagrad와 같은 학습 방식을 이용하면 훨씬 더 좋은 성능을 보여준다

RMSProp

- Concept
  - AdaGrad에서의 문제점이었던 accumulation variable을 수정
  - 해결책
    - Exponential average 사용
      - 현재의 value를 가장 크게 사용, 점진적으로(exponential) 과거의 값들을 사용
    - decay rate ρ 추가

$$\mathrm{r} \leftarrow \rho r + (1-\rho) * g \odot g$$

- Update Rule
  - $\mathrm{r} \leftarrow \rho r + (1-\rho) * g \odot g$
  - $\theta \leftarrow \theta - \frac{\epsilon}{\delta + r^{\frac{1}{2}}} \odot g$

if $\gamma = 0.9$

$$x_t = 0.9x_{t-1} + 0.1g_t$$
$$= 0.081x_{t-2} + 0.09g_{t-1} + 0.1g_t$$
$$= 0.0729x_{t-3} + 0.081g_{t-2} + 0.09g_{t-1} + 0.1g_t$$

RMSProp

- 장점
  - works well in on-line and non-stationary settings

- 문제점
  - ρ가 1에 근접한 값일 때 exponential average of gradient의 bias를 수정해주지 않으면

    i) 매우 큰 stepsize를 갖게 되거나

    ii) (초기 iteration에서) 발산하는 문제가 있을 수 있음

Adam

- Concept
  - RMSProp의 exponential average of gradient값의 bias term이 update가 되지 않는다면 문제가 생긴다는 점을 보완

  - stochastic optimization, high dimensional의 경우 SGD, momentum method에서 사용하는1st order method가 higher-order method보다 적합하므로 1st order method만 사용

    (“higher-order optimization methods are ill-suited” ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION)

  - Momentum method와 RMSProp의 장점을 혼합한 모델
    - First moment of gradient -> Momentum의 v (velocity)를 모방
    - Second moment of gradient -> RMSProp의 r (accumulation variable)을 모방

Adam

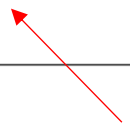
- Update Rule
  - first moment
    - $s \leftarrow \rho_1 s + (1 - \rho_1) g$
  - second moment
    - $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
  - bias correction
    - $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$
    - $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$

Adam

- Update Term
  - $\Delta\theta = -\epsilon * \frac{\hat{s}}{\frac{1}{\hat{r}^2}+\delta}$
  - $\hat{s}$: momentum이 적용된 gradient
  - $\hat{r}$: Accumulation variable

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}+\delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while



Adam

- 장점
  - bounded step size
    - stepsize ϵ이 α 근처에서 bounded 된다
    - step size가 bounded되어있다면, 예측 가능하기 때문에 하이퍼파라미터 설정 시 미리 적절한 값으로 세팅가능하다
  - it does not require a stationary objective
  - it works with sparse gradients
  - it naturally performs a form of step size annealing.

Adam

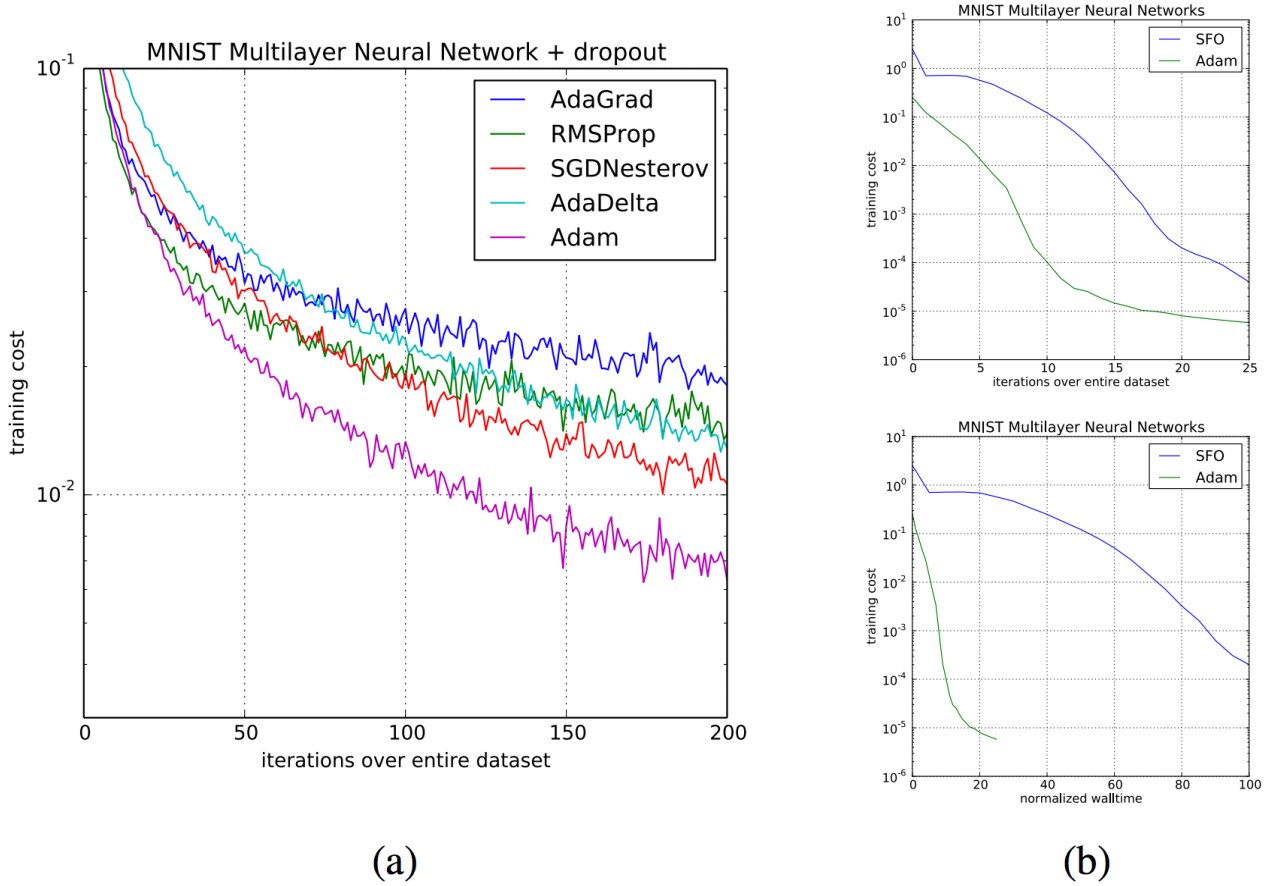


Figure 2: Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function. We compare with the sum-of-functions (SFO) optimizer (Sohl-Dickstein et al., 2014)

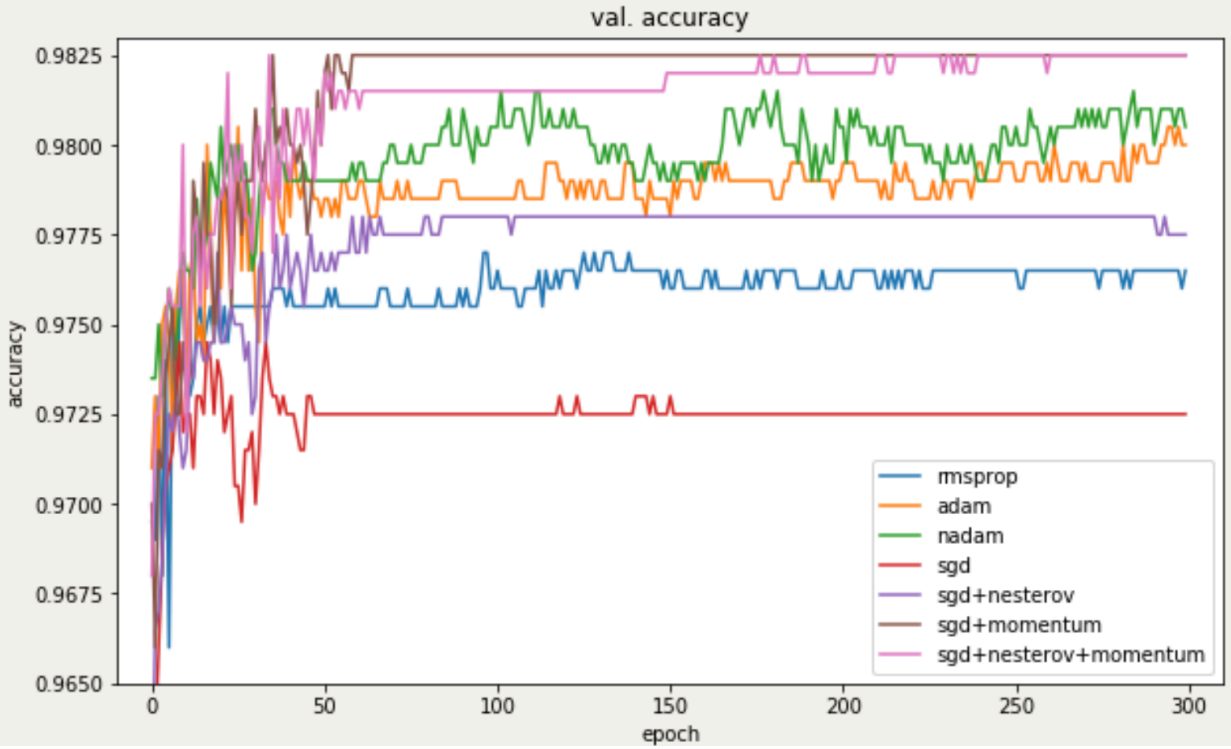
비교

- Adaptive methods vs SGD
  - We observe that the solutions found by adaptive methods generalize worse (often significantly worse) than SGD, even when these solutions have better training performance.
    - (Reference : The Marginal Value of Adaptive Gradient Methods in Machine Learning)
  - Adam have lowest training error/loss, but not val. error/loss.

- Adam vs RMSProp
  - RMSprop towards the end of optimization as gradients become sparser

비교

1. Validation accuracy



참고

- AdaBound
  - https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34
- The Marginal Value of Adaptive Gradient Methods in Machine Learning
  - https://arxiv.org/pdf/1705.08292.pdf
- Adam
  - https://arxiv.org/pdf/1412.6980.pdf