Stochastic Gradient Descent

EE807: Recent Advances in Deep Learning

Lecture 2

Slide made by

Insu Han and Jongheon Jeong

KAIST EE

Table of Contents

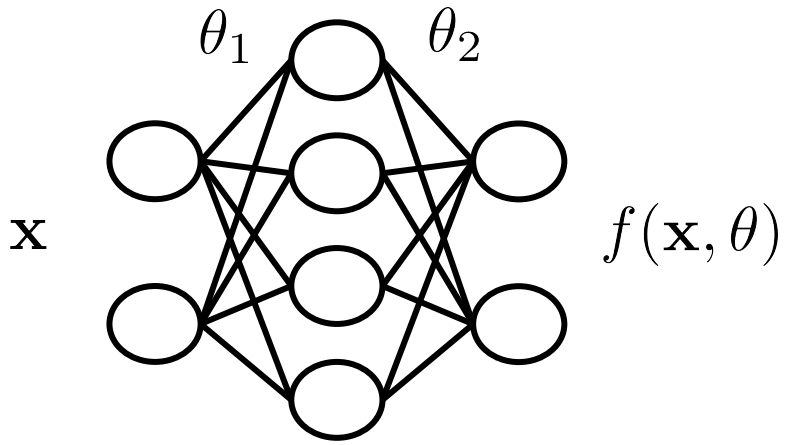
Empirical Risk Minimization (ERM)

- Given training set $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
- Prediction function $f(\mathbf{x}_i, \theta) \in \mathbb{R}$ parameterized by θ
- **Empirical risk minimization:** Find a paramater that minimizes the loss function

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i, \theta), y_i) := L(\theta)$$

where $\ell(\cdot, \cdot)$ is a loss function e.g., MSE, cross entropy,

- For example, neural network has $f(\mathbf{x}, \theta) = \theta_k^\top \sigma\left(\theta_{k-1}^\top \sigma(\cdots \sigma(\theta_1^\top \mathbf{x}))\right)$



$$L(\theta) = \frac{1}{n} \sum_i (f(\mathbf{x}_i, \theta) - y_i)^2$$

Next, how to solve ERM?

Gradient Descent (GD)

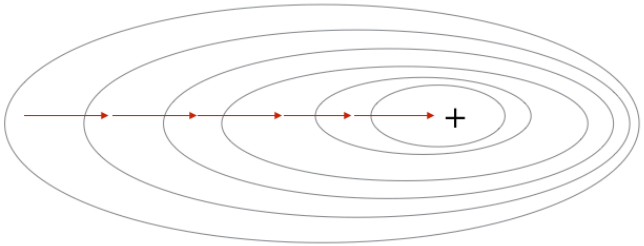
- **Gradient descent (GD)** updates parameters iteratively by taking gradient.

parameters loss function

$$\theta_{t+1} = \theta_t - \gamma \nabla L(\theta_t)$$

learning rate

$$:= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(\theta_t; \mathbf{x}_i, y_i)$$



- (+) Converges to global (local) minimum for convex (non-convex) problem.
- (−) Not efficient with respect to computation time and memory space for huge n.
- For example, ImageNet dataset has n =**1,281,167** images for training.

1.2M of 256x256 RGB images
≈ 236 GB memory

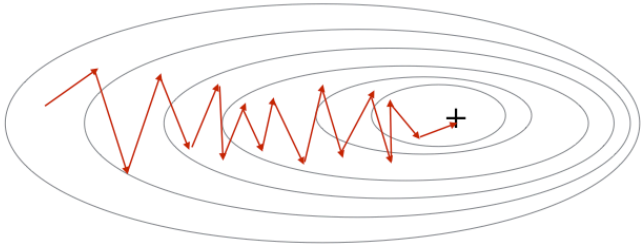
Next, efficient GD

Stochastic Gradient Descent (SGD)

- **Stochastic gradient descent (SGD)** use samples to approximate GD

$$\nabla L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(\theta; \mathbf{x}_i, y_i)$$

$$\backsimeq \frac{1}{|\mathcal{B}|} \sum_{\text{sample}\ i \in \mathcal{B}} \nabla \ell\left(\theta; \mathbf{x}_i, y_i\right)$$



- In practice, minibatch sizes $|\mathcal{B}|$ can be 32/64/128.

- Main practical challenges and current solutions:
  1. SGD can be too noisy and might be unstable ⟶ momentum
  2. hard to find a good learning rate ⟶ adaptive learning rate

Next, momentum

*source : https://lovesnowbest.site/2018/02/16/Improving-Deep-Neural-Networks-Assignment-2/

Momentum Methods

1. Momentum gradient descent
   - Add decaying previous gradients (momentum).

$$\theta_{t+1} = \theta_t - \mathbf{m}_t \qquad \mathbf{m}_t = \mu \mathbf{m}_{t-1} + \gamma \nabla L(\theta_t)$$

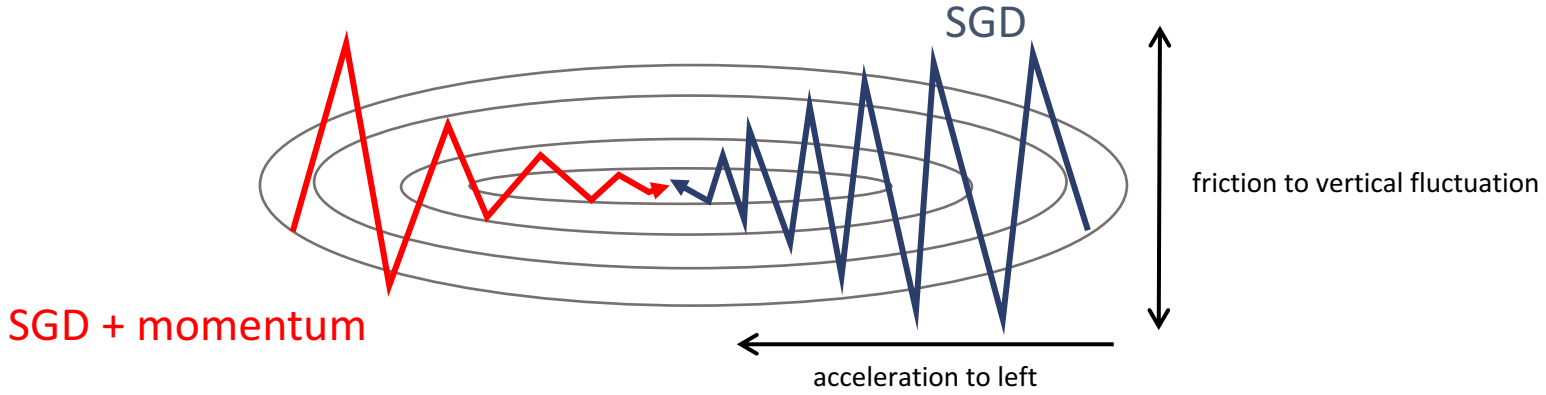
momentum

preservation ratio $\mu \in [0, 1]$

   - Equivalent to the weighted-sum of the fraction μ of previous update.

$$\theta_{t+1} = \theta_t - \gamma \left(\nabla L(\theta_t) + \mu \nabla L(\theta_{t-1}) + \mu^2 \nabla L(\theta_{t-2}) + \cdots \right)$$

   - (+) Momentum reduces the oscillation and accelerates the convergence.



Momentum Methods: Nesterov's Momentum

1. Momentum gradient descent
   - Add decaying previous gradients (momentum).

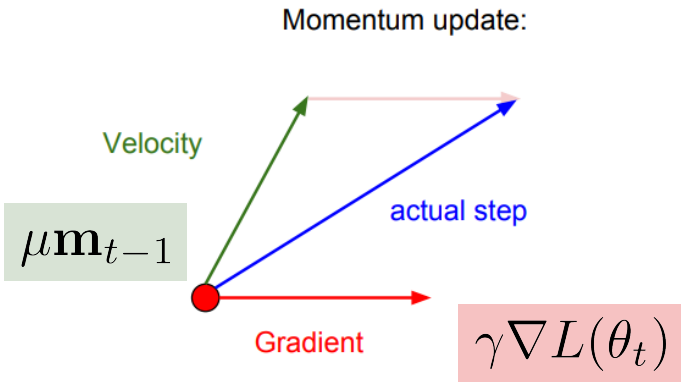
$$\theta_{t+1} = \theta_t - \mathbf{m}_t \qquad \mathbf{m}_t = \mu \mathbf{m}_{t-1} + \gamma \nabla L(\theta_t)$$

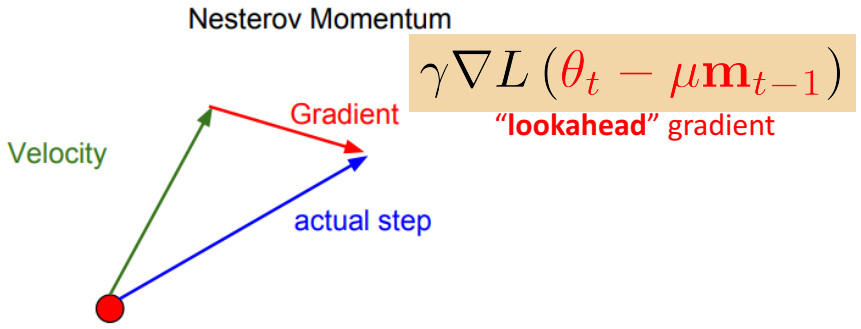
momentum

preservation ratio $\mu \in [0, 1]$

   - (−) Momentum can fail to converge even for simple convex optimizations.
   - **Nestrov's accelerated gradient** (NAG) [Nesterov' 1983] use gradient for approximate future position, i.e.,

$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \gamma \nabla L(\theta_t - \mu \mathbf{m}_{t-1})$$





Momentum Methods: Nesterov's Momentum

1. Momentum gradient descent
   - Add decaying previous gradients (momentum).

$$\theta_{t+1} = \theta_t - \mathbf{m}_t \qquad \mathbf{m}_t = \mu \mathbf{m}_{t-1} + \gamma \nabla L\left(\theta_t\right)$$

momentum

preservation ratio $\mu \in [0, 1]$

   - **Nesterov's accelerated gradient** (NAG) [Nesterov' 1983] use gradient for approximate future position, i.e.,

$$\mathbf{m}_t \leftarrow \mu \mathbf{m}_{t-1} + \gamma \nabla L\left(\theta_t - \mu \mathbf{m}_{t-1}\right)$$

```
while True:
    dtheta = compute_gradient(theta, batch_size)
    theta = theta - learning_rate * dtheta
```

SGD

```
m = 0
while True:
    dtheta = compute_gradient(theta, batch_size)
    m = mu * m + learning_rate * dtheta
    theta = theta - m
```

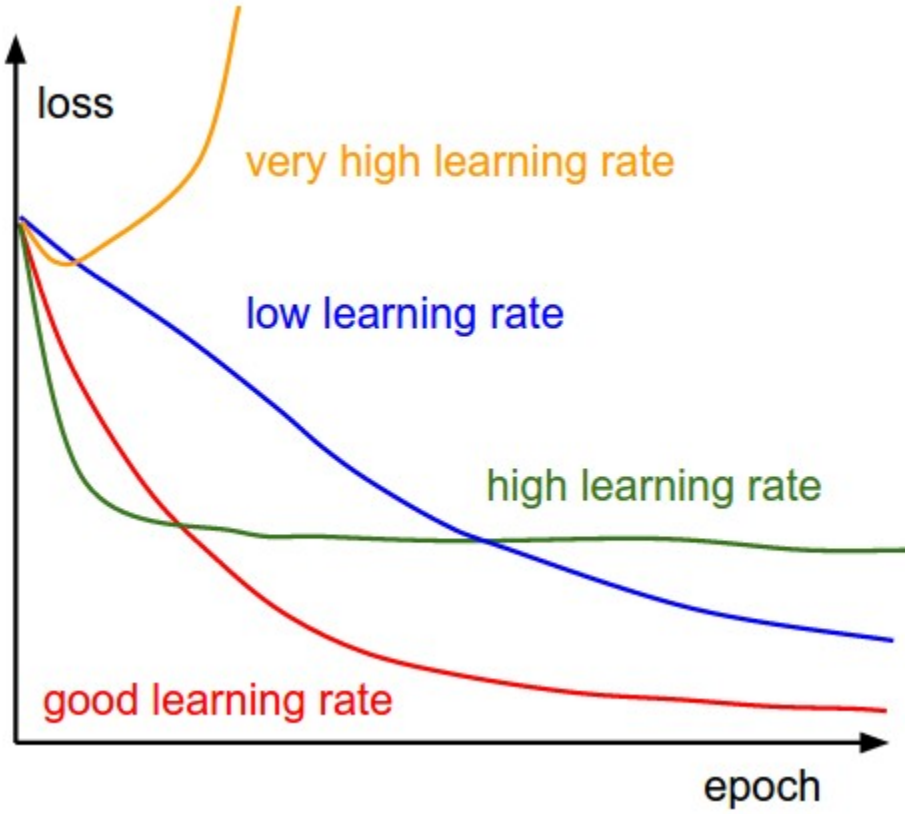
SGD + momentum

```
m = 0
while True:
    dtheta = compute_gradient(theta, batch_size)
    m_old = m
    m = mu * m + learning_rate * dtheta
    x =
```

NAG

Quiz: fill in the pseudo code of Nesterov' accelerated gradient

2. Learning rate scheduling
   - **Learning rate is critical for minimizing loss !**



Too **high** → May ignore the narrow valley, can diverge

Too **low** → May fall into the local minima, slow converge

Next, learning rate scheduling

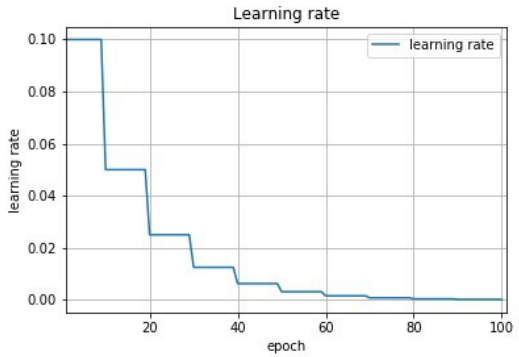
*source : http://cs231n.github.io/neural-networks-3/

Adaptive Learning Rate Methods: Learning rate annealing

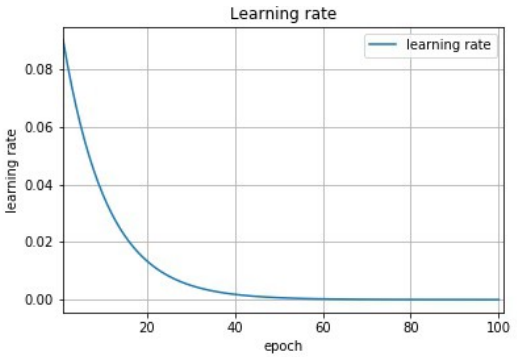
2. Learning rate scheduling : decay methods
   - A naive choice is the **constant** learning rate
   - Common learning rate schedules include time-based/step/exponential decay

	Time-based	Exponential	Step (most popular in practice)
γ_t	$\frac{\gamma_0}{1+kt}$	$\gamma_0 \exp(-kt)$	$\gamma_0 \exp(-k\lfloor \frac{t}{T_{\text{epoch}}} \rfloor)$

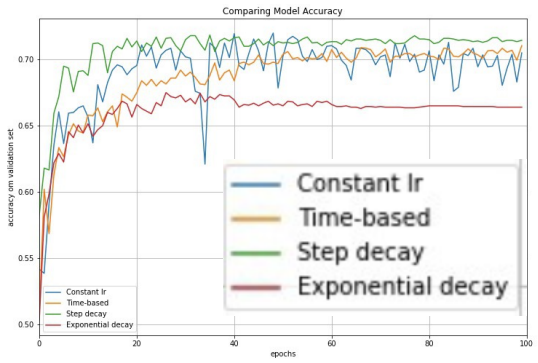
   - "Step decay" decreases learning rate by a factor every few epochs
   - Typically, it is set $\gamma_0 = 0.01$ and drops by half ever $T_{\text{epoch}} = 10$ epoch



step decay



exponential decay

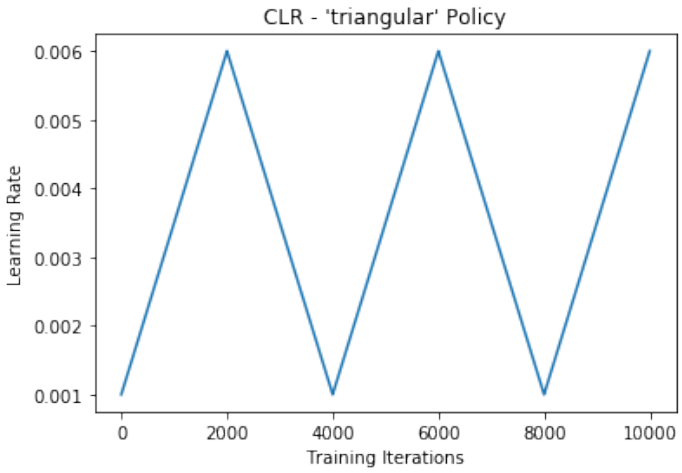


accuracy

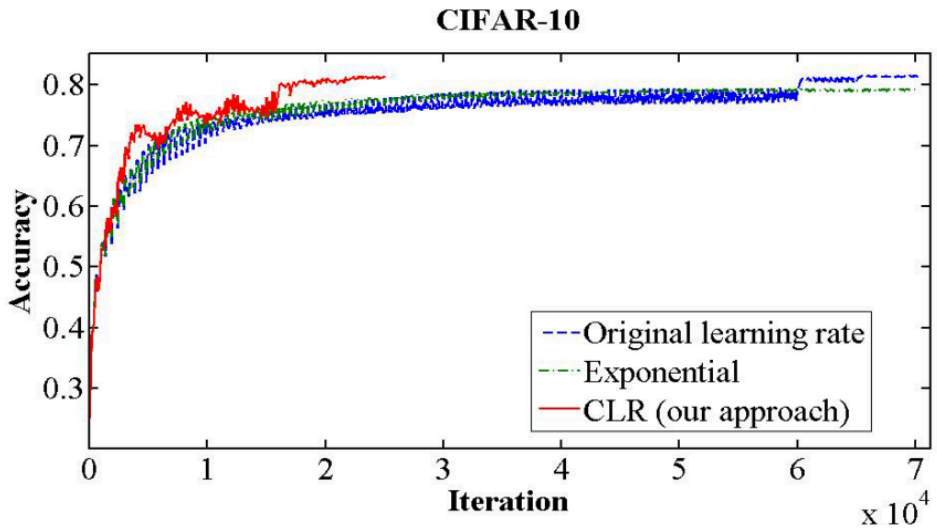
*source : https://towardsdatascience.com/

Adaptive Learning Rate Methods: Learning rate annealing

2. Learning rate scheduling : cycling method
   - [Smith' 2015] proposed **cycling** learning rate (triangular)
   - Why "**cycling**" learning rate?
     - Sometimes, **increasing learning rate** is helpful to escape the saddle points
   - It can be combined with exponential decay or periodic decay



cycling (triangular) decay



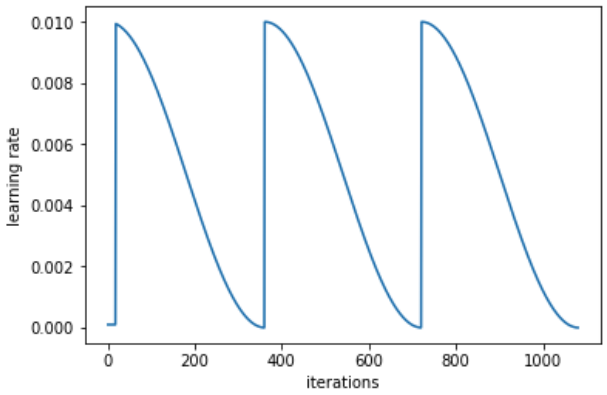
*source : https://github.com/bckenstler/CLR

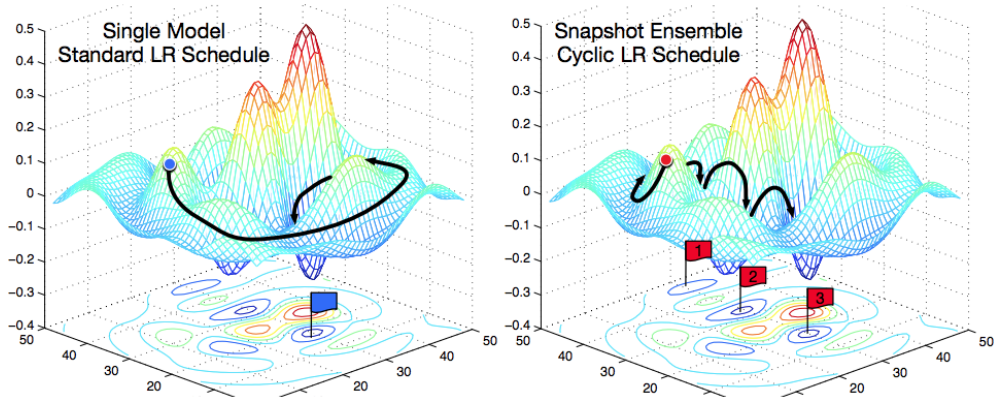
Adaptive Learning Rate Methods: Learning rate annealing

2. Learning rate scheduling : cycling method
   - [Loshchilov' 2017] use **cosine cycling** and **restart** the maximum at each cycle
   - Why "**cosine**" ?
     - It decays slowly at the half of cycle and drop quickly at the rest

   - (+) can climb down and up the loss surface, thus can traverse several local minima
   - (+) same as restarting at good points with an initial learning rate $\gamma_{\max}$

$$\gamma_t = \gamma_{\min} + \frac{1}{2}\left(\gamma_{\max} - \gamma_{\min}\right)\left(1 + \cos(\mathrm{mod}(t, T)\pi)\right) \qquad T : \text{period}$$



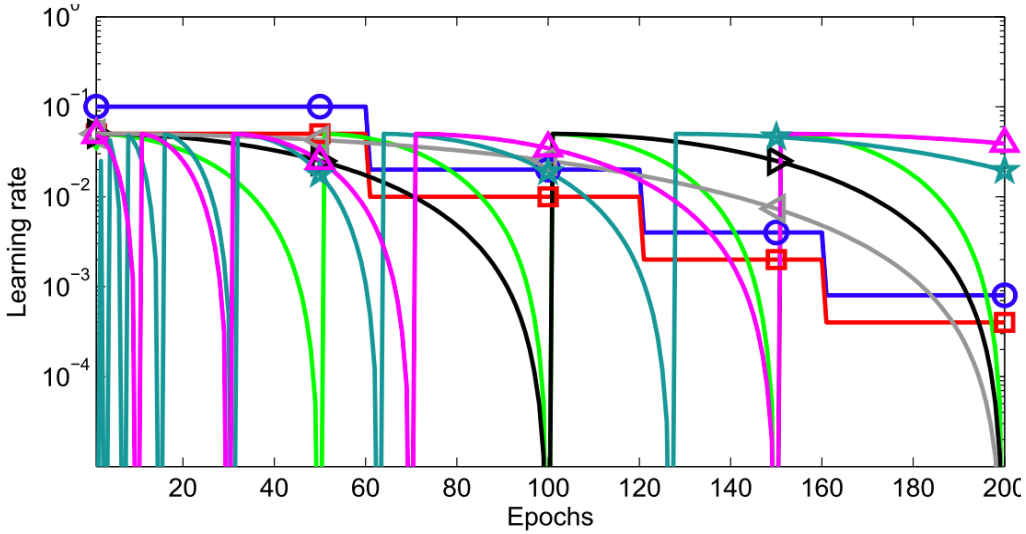


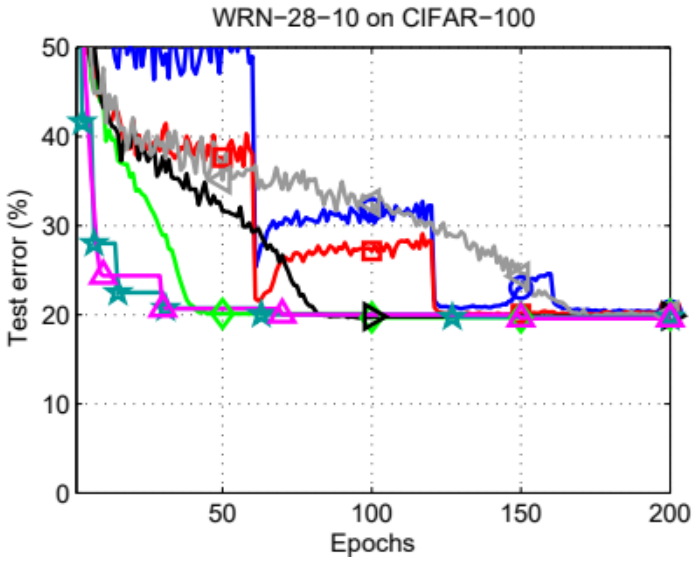
* source : Loshchilov et al., SGDR: Stochastic Gradient Descent with Warm Restarts. ICLR 2017

Adaptive Learning Rate Methods: Learning rate annealing

2. Learning rate scheduling : cycling method
   - [Loshchilov' 2017] also proposed **warm restart** in cycling learning rate

     *Warm restart : frequently restart in early iterations
   - (+) It help to escape saddle points since it is more likely to stuck in early iteration





: step decay

: cycling with no restart

: cycling with restart

But, there is no perfect learning rate scheduling! It depends on specific task.

Next, adaptive learning rate

* source : Loshchilov et al., SGDR: Stochastic Gradient Descent with Warm Restarts. ICLR 2017

3. Adaptively changing learning rate (AdaGrad, RMSProp)
   - **AdaGrad** [Duchi' 11] downscales a learning rate by magnitude of previous gradients.

$$\theta_{t+1} = \theta_t - \frac{\gamma}{\sqrt{v_t}} \nabla L(\theta_t) \qquad v_{t+1} = v_t + \nabla L(\theta_t)^2$$

v_{t+1}: sum of all previous squared gradients

   - (–) the learning rate strictly decreases and becomes too small for large iterations.
   - **RMSProp** [Tieleman' 12] uses the moving averages of squared gradient.

$$v_{t+1} = \mu v_t + (1-\mu) \nabla L(\theta_t)^2$$

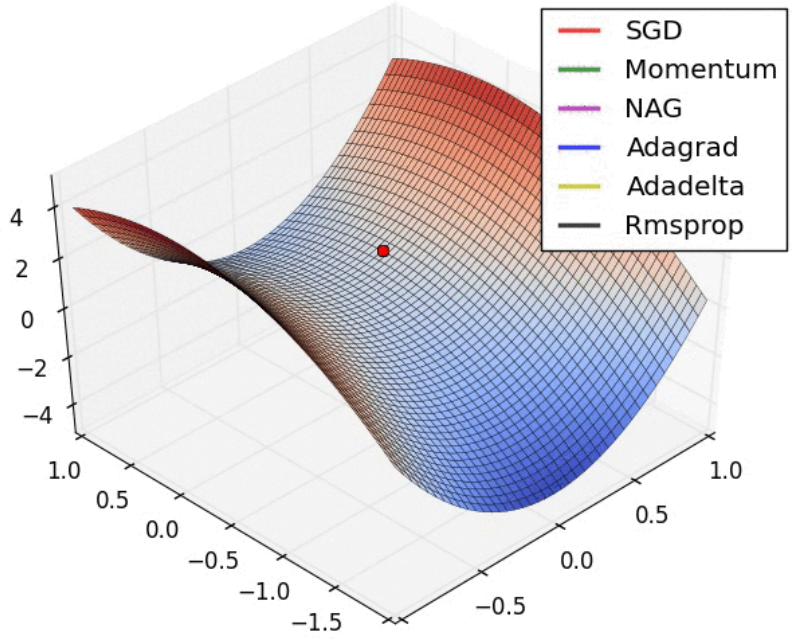
μ: preservation ratio $\mu \in [0, 1]$

   - Other variants also exist, e.g., Adadelta [Zeiler' 2012]

Adaptive Learning Rate Methods

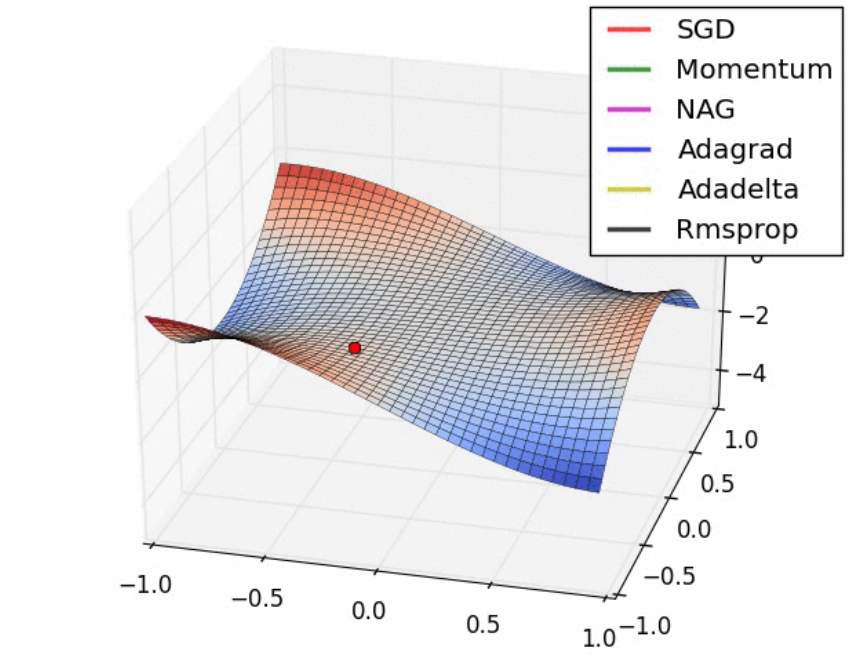
- Visualization of algorithms

optimization on saddle point



https://imgur.com/s25RsOr

optimization on local optimum



https://imgur.com/a/Hqolp

- Adaptive learning-rate methods, i.e., Adadelta and RMSprop are most suitable and provide the best convergence for these scenarios

Next, momentum + adaptive learning rate

* source: animations from from Alec Radford' blog

Adaptive Learning Rate Methods: ADAM

3. Combination of momentum and adaptive learning rate
   - **Adam** (ADAptive Moment estimation) [Kingma' 2015]

$$\theta_{t+1} \leftarrow \theta_t - \frac{\gamma}{\sqrt{v_t}}\mathbf{m}_t$$

momentum

$$\mathbf{m}_{t+1} \leftarrow \mu_1 \mathbf{m}_t + (1-\mu_1)\nabla L(\theta_t)$$

$$v_{t+1} \leftarrow \mu_2 v_t + (1-\mu_2)\nabla L(\theta_t)^2$$

average of squared gradients

   - Can be seen as momentum + RMSprop update.
   - Other variants exist, e.g., Adamax [Kingma' 14], Nadam [Dozat' 16]

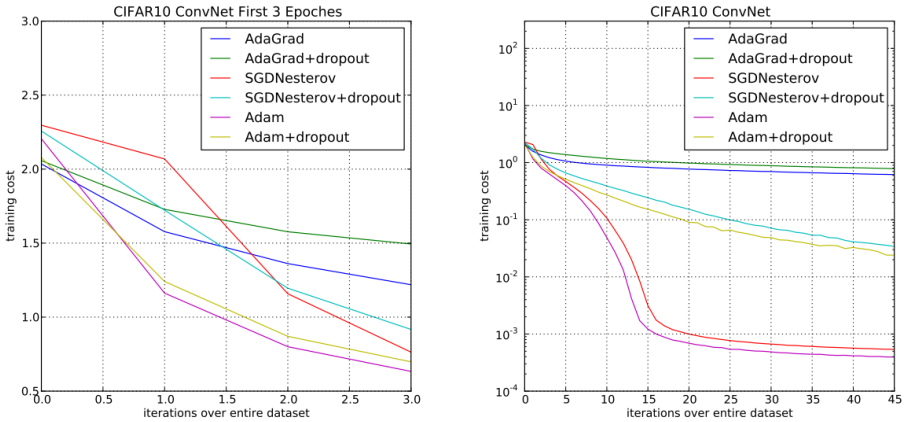
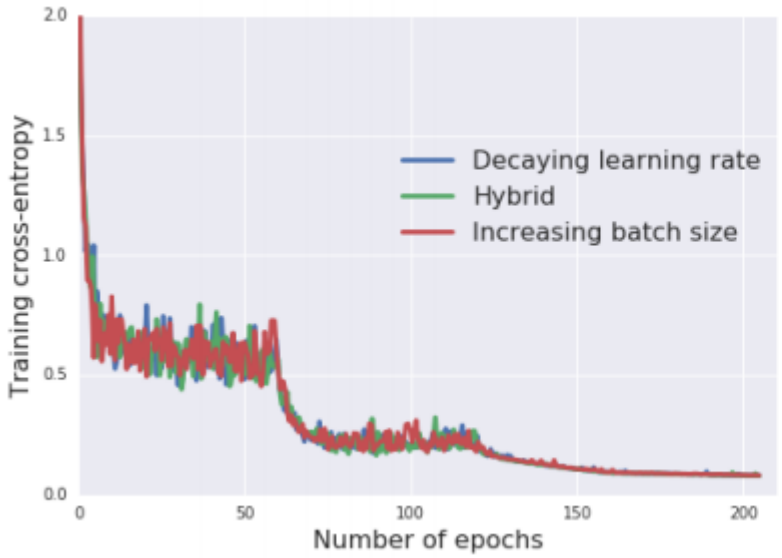


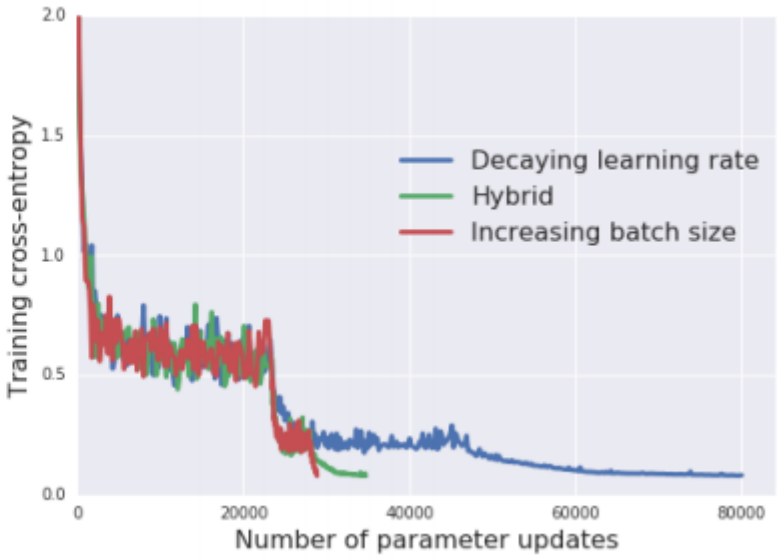
Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

* source : Kingma and Ba. Adam: A method for stochastic optimization. ICLR 2015

Decaying the Learning Rate = Increasing the Batch Size

- In practice, **SGD + Momentum** and **Adam** works well in many applications.
- But, scheduling learning rates is still critical! (should be decay appropriately)
- [Smith' 2017] shows that decaying learning rate = increasing batch size,
  - (+) A large batch size allows fewer parameter updates, leading to parallelism!





* source : Smith et al., "Don't Decay the Learning Rate, Increase the Batch Size.", ICLR 2017

Summary

- SGD have been used as essential algorithms to deep learning as back-propagation.

- Momentum methods improve the performance of gradient descend algorithms.
  - Nesterov's momentum

- Annealing learning rates are critical for training loss functions
  - Exponential, harmonic, cyclic decaying methods
  - Adaptive learning rate methods (RMSProp, AdaGrad, AdaDelta, Adam, etc)

- In practice, **SGD + momentum** shows successful results, outperforming Adam!
  - For example, NLP (Huang et al., 2017) or machine translation (Wu et al., 2016)

References

- [Nesterov' 1983] Nesterov. A method of solving a convex programming problem with convergence rate O(1/k^2). 1983
  link: http://mpawankumar.info/teaching/cdt-big-data/nesterov83.pdf
- [Duchi et al 2011], "Adaptive subgradient methods for online learning and stochastic optimization", JMLR 2011
  link : http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf
- [Tieleman' 2012] Geoff Hinton's Lecture 6e of Coursera Class
  link : http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
- [Zeiler' 2012] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method
  link : https://arxiv.org/pdf/1212.5701.pdf
- [Smith' 2015] Smith, Leslie N. "Cyclical learning rates for training neural networks."
  link : https://arxiv.org/pdf/1506.01186.pdf
- [Kingma and Ba., 2015] Kingma and Ba. Adam: A method for stochastic optimization. ICLR 2015
  link : https://arxiv.org/pdf/1412.6980.pdf
- [Dozat' 2016] Dozat, T. (2016). Incorporating Nesterov Momentum into Adam. ICLR Workshop,
  link : http://cs229.stanford.edu/proj2015/054_report.pdf
- [Smith et al., 2017] Smith, Samuel L., Pieter-Jan Kindermans and Quoc V. Le. Don't Decay the Learning Rate, Increase the Batch Size. ICLR 2017.
  link : https://openreview.net/pdf?id=B1Yy1BxCZ
- [Loshchilov et al., 2017] Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. ICLR 2017.
  link : https://arxiv.org/pdf/1608.03983.pdf