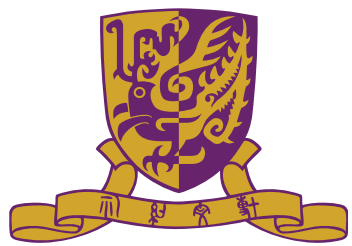
DDA5001 Machine Learning

Neural Networks (Part III): SGD and Adam

Xiao Li

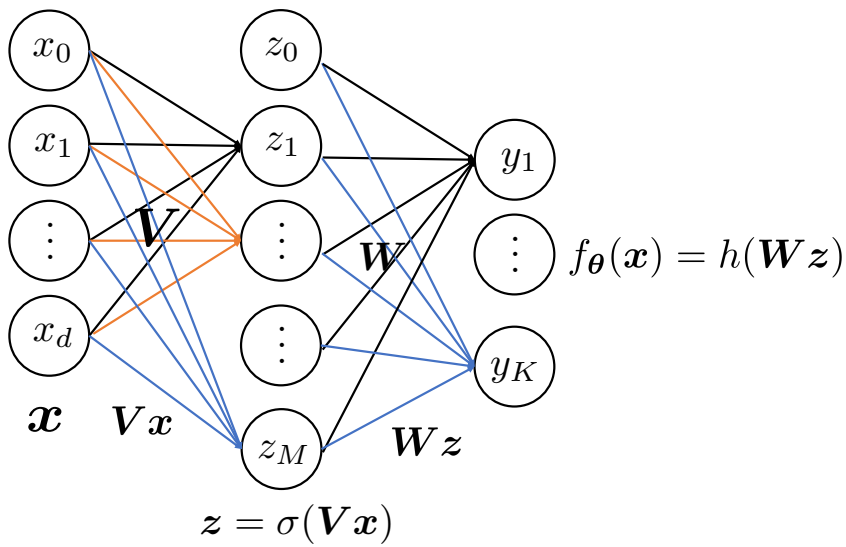
School of Data Science
The Chinese University of Hong Kong, Shenzhen



Recap: Forward Pass in Backpropagation

Forward pass: Feed data sample $\boldsymbol{x}_i$ into the network, use the current parameters $\boldsymbol{V}$ and $\boldsymbol{W}$ to compute and store

- $\boldsymbol{V}\boldsymbol{x}_i$
- $\boldsymbol{z}_i = \sigma(\boldsymbol{V}\boldsymbol{x}_i)$
- $\boldsymbol{W}\boldsymbol{z}_i$
- $f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) = h(\boldsymbol{W}\boldsymbol{z}_i),\ \ \ell_i(\boldsymbol{\theta}) = \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2$



Recap: Backward Pass in Backpropagation

Backward pass:

Compute the gradient of the second layer

- $\boldsymbol{\delta}_i = 2(h(\boldsymbol{W}\boldsymbol{z}_i) - \boldsymbol{y}_i) \odot h'(\boldsymbol{W}\boldsymbol{z}_i)$
- $\frac{\partial \ell_i}{\partial \boldsymbol{W}} = \boldsymbol{\delta}_i \boldsymbol{z}_i^\top$

Backpropagate the gradient to $\boldsymbol{z}_i$:

- $\frac{\partial \ell}{\partial \boldsymbol{z}_i} = \boldsymbol{W}^\top \boldsymbol{\delta}_i$

Compute the gradient of the first layer:

- $\frac{\partial \ell_i}{\partial \boldsymbol{V}} = \left(\frac{\partial \ell}{\partial \boldsymbol{z}_i} \odot \sigma'(\boldsymbol{V}\boldsymbol{x}_i) \right) \boldsymbol{x}_i^\top$

backpropagation is an efficient way of computing the gradient of NN learning problem.

Given the computed gradient, we can apply gradient-based training algorithm for training NN. $\rightsquigarrow$ Which algorithm should we choose?

Stochastic Gradient Descent (SGD)

Adam Family

Structured Learning Problem: Finite-Sum Structure

In LS, LR, SVM, and NN, we always have

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\ell(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)}_{\ell_i(\boldsymbol{\theta})}$$

each $\ell_i(\boldsymbol{\theta})$ is called a component function. We can use BP to compute $\nabla\mathcal{L}(\boldsymbol{\theta})$ if it is NN.

Example: Least squares

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \ \frac{1}{n}\|\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y}\|_2^2 = \frac{1}{n}\sum_{i=1}^{n} \underbrace{(\boldsymbol{\theta}^\top \boldsymbol{x}_i - y_i)^2}_{\ell_i(\boldsymbol{\theta})}$$

- Each ℓ_i corresponds to one training data point $(\boldsymbol{x}_i, y_i)$.
- The above structure is called finite-sum and the problem is called finite-sum optimization.
- Finite-sum optimization is ubiquitous in machine learning, including most of the (supervised) learning problems.

Modern Large-Scale Machine Learning Problems

One of the key features of modern machine learning problems is large-scale, i.e., the number of training data n is very large (guided by VC analysis). Such a problem is called large-scale machine learning.

- One notable example is training NN.

We have the following two observations:

- The property we want for an algorithm is fast convergence. In terms of optimization, we should use fast algorithm (fast in terms of iteration) like first-order method (GD/AGD) or even second-order method (Newton).
- Guided by VC analysis, we just need to choose a fine $\mathcal{H}$ with small generalization error and then design a fast learning algorithm (GD/AGD or Newton) to learn $\widehat{\boldsymbol{\theta}}$ by solving the learning problem. Then, the learning process is done.

What is missed? We have not considered the computational issue of the training algorithm when n is large-scale.

Let us take GD as an example.

Gradient Descent in Large-scale Finite-sum Structure

GD:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \nabla \mathcal{L}(\boldsymbol{\theta}_k)$$

- What is the structure of $\nabla \mathcal{L}(\boldsymbol{\theta}_k)$?

Due to the linearity of gradient operator, we have

$$\nabla \mathcal{L}(\boldsymbol{\theta}_k) = \nabla \left(\frac{1}{n} \sum_{i=1}^{n} \ell_i(\boldsymbol{\theta}_k) \right)$$
$$= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(\boldsymbol{\theta}_k)$$

- Observation: The gradient of $\mathcal{L}$ is the summation of the gradients of all the component functions.
- It can be costly and even prohibitive to the compute the full gradient when n is very large.
- Newton's method is even much worse when n is so large.

$\rightsquigarrow$ Motivates the design of stochastic optimization methods.

Stochastic Gradient Descent

Consider $\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^n \ell_i(\boldsymbol{\theta})$, where n is very large.

The idea: Can we just use gradient information of only one component function rather than the full gradient for algorithmic update?

Algorithmic design framework

$$\boldsymbol{\theta}_{k+1} = \operatorname*{argmin}_{\boldsymbol{\theta}\in\mathbb{R}^d} \left\{ q_k(\boldsymbol{\theta}) + \frac{1}{2\mu_k}\|\boldsymbol{\theta} - \boldsymbol{\theta}_k\|_2^2 \right\}$$

- The q_k is the linear approximation of $\mathcal{L}$ using the gradient of only one component function

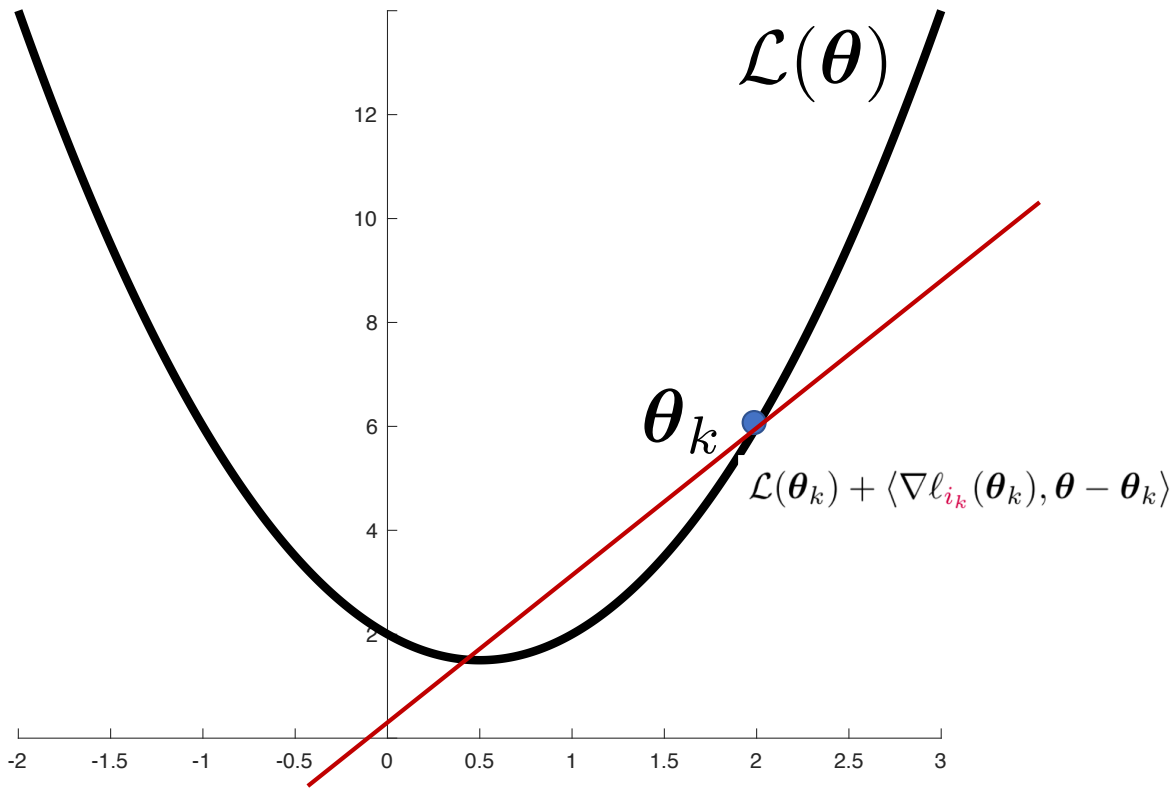
$$q_k(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}_k) + \langle \nabla \ell_{i_k}(\boldsymbol{\theta}_k), \boldsymbol{\theta} - \boldsymbol{\theta}_k \rangle,$$

  ℓ_{i_k} is the i_k-th component function at k-th iteration, $\nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ is called stochastic gradient.

Stochastic gradient descent: SGD

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k).$$

SGD: Geometric Illustration



SGD vs. GD

GD:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \underbrace{\nabla \mathcal{L}(\boldsymbol{\theta}_k)}_{\frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\boldsymbol{\theta}_k)}$$

- Expensive iteration when n is very large.
- Opportunities for parallelism.

SGD:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$$

- Very cheap iteration.
- Sequential use of data, nonparallel.

Efficiency of SGD over GD:

- Assume all the n training data $(\boldsymbol{x}_i, y_i)$ are the copies of one data (somewhat extreme).
- GD is n times more expensive than SGD.

The Choice of i_k

SGD:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$$

Parameters in SGD needed to be preset:

- The usual ones: learning rate μ_k, initial point $\boldsymbol{\theta}_0$, etc.
- The additional one: The component function index i_k at each iteration.

Random choice: At k-th iteration choose i_k uniformly at random from $\{1, \ldots, n\}$.

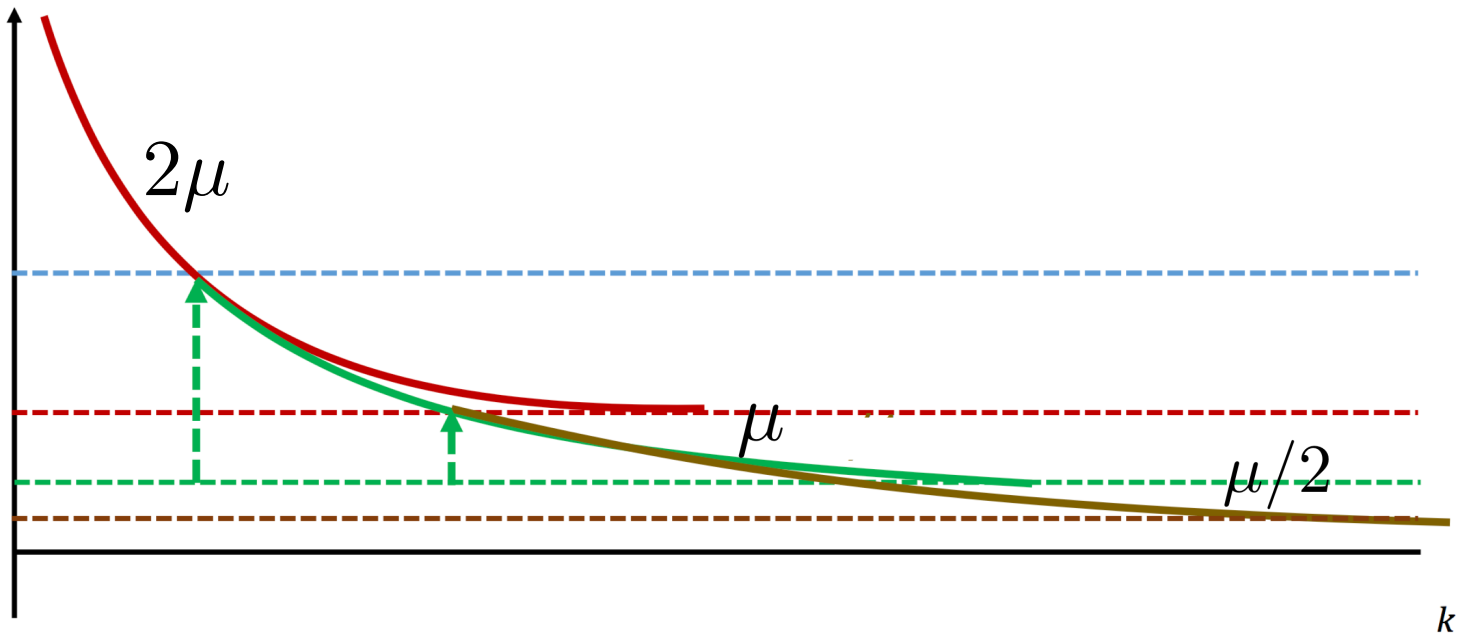
Theoretical justification: Unbiased approximation of the full gradient.

$$\mathbb{E}_{i_k}\left[\nabla \ell_{i_k}(\boldsymbol{\theta}_k)\right] = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(\boldsymbol{\theta}_k) = \nabla \mathcal{L}(\boldsymbol{\theta}_k)$$

- In expectation, SGD is GD.

Learning Rate in SGD

- One choice is $\mu_k = \frac{c}{\sqrt{k}}$. Similar to subgradient method, SGD is not a descent method. Hence, we need to use decaying learning rate in SGD.
- Apart from the decaying learning rate $\mu_k = \frac{c}{\sqrt{k}}$, step-decay is also utilized in practice.



- Step-decay: Whenever progress slows down, we half the learning rate and continue update.

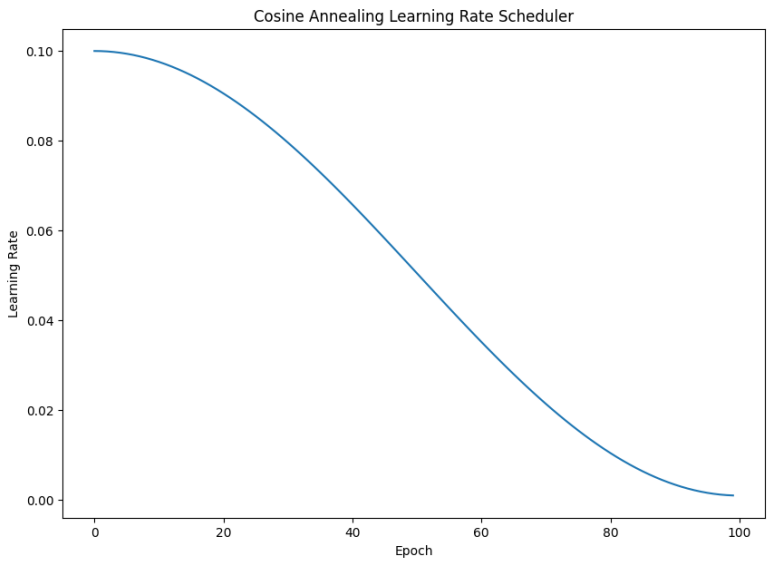
Cosine Scheduler Learning Rate

There is another famous and widely used learning rate schedule, called cosine scheduler. It requires to specify:

- The maximum initial learning rate μ_0.
- The minimal final learning rate $\mu_{\min}$.
- The preset total number of iteration T.

Then, cosine scheduler is to automatically decay μ_0 to $\mu_{\min}$ in T iterations:

$$\mu_k = \mu_{\min} + \frac{1}{2}(\mu_0 - \mu_{\min})\left(1 + \cos\left(\frac{k}{T}\pi\right)\right).$$



Application: Text Classification with SVM

- Dataset
  - RCV1 document corpus.
  - 781265 training examples ($= n$), 23149 testing examples.
  - 47152 features ($= d$).
- Task
  - Recognizing documents of category CCAT.
- Machine Learning model: Train soft-margin SVM classifier

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d, b\in\mathbb{R}} \mathcal{L}(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n}\left\{\max(0, 1 - y_i(\boldsymbol{\theta}^\top \boldsymbol{x}_i + b)) + \lambda\|\boldsymbol{\theta}\|_2^2\right\}$$

	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	*0.2275*	*6.02%*
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	*0.2275*	*6.02%*

Random Reshuffling

Random Reshuffling: The More Practical SGD

- The uniform at random choice of i_k in SGD is not practical. It is mainly for theoretical analysis / motivation of SGD.
- In SGD, we need to generate a random number i_k from $\{1, \ldots, n\}$ at each iteration. This can be time costly.
- In addition, uniformly random sampling may not utilize the training data fully in one epoch. Here, one epoch implies n iterations of SGD (one epoch means utilizing all n data points, hence n iterations).

Random Reshuffling (RR)

In k-th epoch, we first uniformly at random permute the order $\{1, \ldots, n\}$ to get σ_k. Then, we update by visiting sequentially the component function indexed by σ_k. The resulting algorithm is called Random Reshuffling (RR).

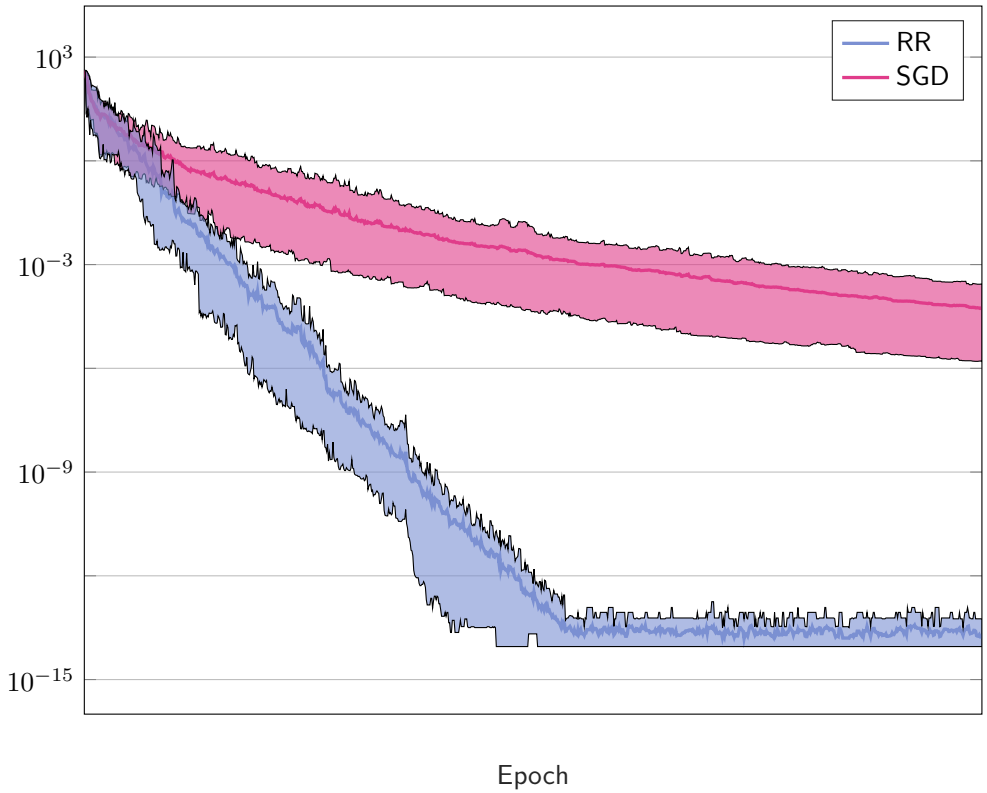
RR vs SGD: Suppose $n = 5$, one epoch of SGD and RR may be

RR	f_2	f_5	f_4	f_1	f_3
SGD	f_3	f_2	f_2	f_1	f_3

- RR uses all the samples in one epoch.

Random Reshuffling: Experiment

Implement two algorithms on simple LS.



In summary:

- RR converges faster than SGD empirically.
- RR is easy to implement due to simple permutation in one epoch.
- What is commonly implemented in practice is indeed RR; people just call it SGD. Check PyTorch's SGD implementation, it is RR.

SGD with Momentum

SGD with Momentum

We can also include momentum acceleration technique to speed up the convergence speed of SGD.

SGD with momentum:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{m}_k$$
$$\boldsymbol{m}_k = \mu_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k) + \beta_k \boldsymbol{m}_{k-1}$$

SGD with Nesterov's momentum:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{w}_k - \mu_k \nabla \ell_{i_k}(\boldsymbol{w}_k)$$
$$\boldsymbol{w}_k = \boldsymbol{\theta}_k + \frac{k-1}{k+2}(\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1})$$

SGD with momentum can substantially improve the training speed compared to SGD only.

Stochastic Gradient Descent (SGD)

Adam Family

Adaptive Learning Rate Method: AdaGrad

Motivations:

- All the optimization algorithms we studied now assign the same learning rate to all coordinates in $\boldsymbol{\theta}$.
- We might need different learning rates for different coordinates in $\boldsymbol{\theta}$.

AdaGrad is an important method that utilizes adaptive learning rate.

- The idea: Normalize each gradient coordinate by the past gradients.
- It has the following update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \frac{\boldsymbol{g}_k}{\sqrt{\sum_{t=1}^{k} \boldsymbol{g}_t^2}},$$

  where $\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ is the stochastic gradient. Vector division, square root, and square operations are all element-wise.
- AdaGrad penalizes the gradient coordinate when it has too large value, while increase the small one for exploration.

AdaGrad reduces the burden for tuning learning rate, and it is often more reliable than SGD.

Adam: Momentum Meets Adaptive Learning Rate

Observations:

- SGD with momentum can improve convergence speed.
- Adaptive learning rate like AdaGrad can reduce learning rate tuning and stablize convergence.

How about combine these two techniques? $\rightsquigarrow$Adam method.

- Adam uses a moving averaging for updating the momentum and adaptive learning rate.

Momentum update:

$$\boldsymbol{m}_k = \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\boldsymbol{g}_k, \quad \text{with} \quad \boldsymbol{m}_0 = 0.$$

Adaptive learning rate update:

$$\boldsymbol{v}_k = \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)\boldsymbol{g}_k^2, \quad \text{with} \quad \boldsymbol{v}_0 = 0.$$

$\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ is the stochastic gradient.

Rule of thumb: $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Understanding Moving Averaging

If we expand the update of momentum and adaptive learning rate in Adam, we will have

$$\boldsymbol{m}_k = \beta_1^k \boldsymbol{m}_0 + \beta_1^{k-1}(1-\beta_1)\boldsymbol{g}_1 + \cdots + \beta_1(1-\beta_1)\boldsymbol{g}_{k-1} + (1-\beta_1)\boldsymbol{g}_k,$$

and

$$\boldsymbol{v}_k = \beta_2^k \boldsymbol{v}_0 + \beta_2^{k-1}(1-\beta_2)\boldsymbol{g}_1^2 + \cdots + \beta_1(1-\beta_2)\boldsymbol{g}_{k-1}^2 + (1-\beta_2)\boldsymbol{g}_k^2,$$

Interpretations: The moving averaging exponentially decays the importance of the historical records, while emphasizes more on the most recent ones.

Adam: Algorithmic Procedure

Adam

1. Compute a stochastic gradient $\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$;
2. Update momentum: $\boldsymbol{m}_k = \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\boldsymbol{g}_k$;
3. Update adaptive learning rate $\boldsymbol{v}_k = \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)\boldsymbol{g}_k^2$;
4. Bias-corrected momentum: $\boldsymbol{m}_k = \boldsymbol{m}_k/(1-\beta_1^k)$;
5. Bias-corrected adaptive learning rate: $\boldsymbol{v}_k = \boldsymbol{v}_k/(1-\beta_2^k)$;
6. Update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \frac{\boldsymbol{m}_k}{\sqrt{\boldsymbol{v}_k + 10^{-8}}}.$$

Repeat until convergence.

- The bias-correction term is used to make $\boldsymbol{m}_k$ and $\boldsymbol{v}_k$ unbiased estimators for $\mathbb{E}[g_k]$ and $\mathbb{E}[g_k^2]$. Not sure if this step is crucial for practical performance.
- Adam is widely used in practice for training NN.

Weight Decay Issue in Adam

Weight decay regularization is widely utilized in NN training, since otherwise it easily has overfitting due to strong representation power of NN.

- Therefore, Adam is often used together with weight decay.
- However, how people use it is through changing the stochastic gradient:

$$\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k) + \lambda \boldsymbol{\theta}_k.$$

  Recall that the gradient of weight decay term $\frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$ is $\lambda\boldsymbol{\theta}$.

Issues:

- $\boldsymbol{g}_k$ is not directly used for updating, it is used to update momentum and also normalized by adaptive learning rate.
- This makes it unclear whether we have weight decay nor not.
- It is not be equivalent to the ℓ_2-regularization we studied in the regularization part.

AdamW: Decoupled Weight Decay

Idea: Decouple the weight decay term and use the true ℓ_2-regularization.
$\rightsquigarrow$ the AdamW method.

- All the steps of AdamW keeps the same as that of the Adam, except for the update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \frac{\boldsymbol{m}_k}{\sqrt{\boldsymbol{v}_k + 10^{-8}}} - \mu_k \lambda \boldsymbol{\theta}_{k-1}.$$

- This additional $\lambda \boldsymbol{\theta}_{k-1}$ corresponds to the gradient of the weight decay term added to training loss function directly, namely, for NN training

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2,$$

  AdamW applies Adam for $\min \mathcal{L}$ while GD for $\min$ the weight decay regularization.

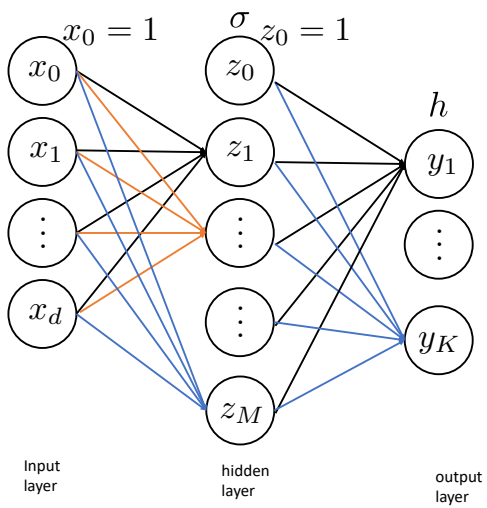
- AdamW is truly applying ℓ_2-regularization. Thus, for huge NN training problems, AdamW often has better generalization than Adam, as better regularization improves generalization.

Experiments: Setup

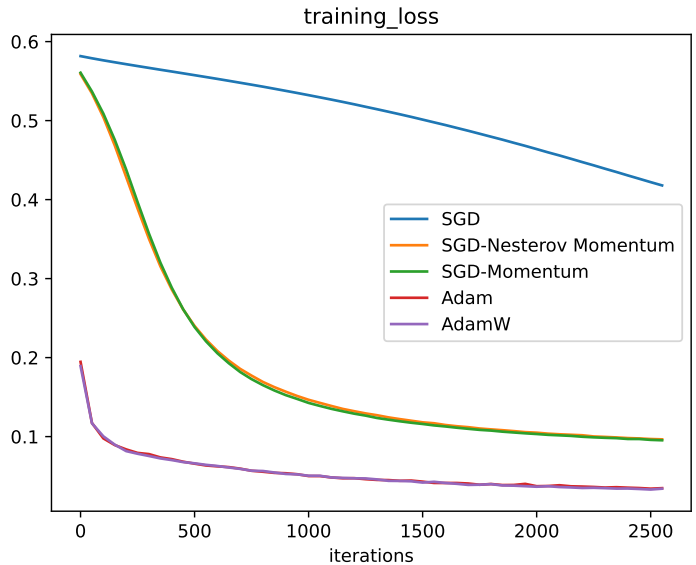
We now conduct experiments on SGD, SGD momentum, SGD nesterov momentum, Adam, AdamW for training NN.

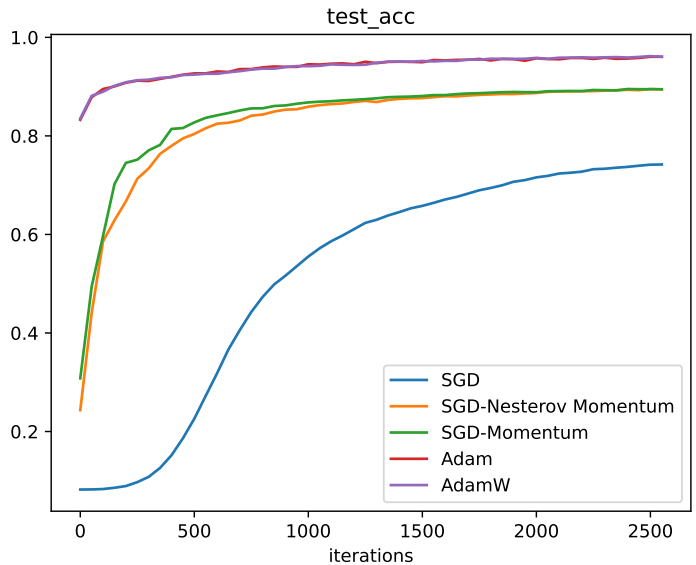
- Task: Classification on the MNIST dataset:

- Model: One hidden layer (two layer) NN, with $M = 64$, number of classes $K = 10$, σ is ReLU, h is softmax.



Experiments: Results[1]





- SGD is slow.
- Adam family has the best performance, illustrating why they are quite ubiquitous in practice.

[1] Code is available at `https://www.kaggle.com/code/jonery/lecture-demo`. You need to register Kaggle to use it.

Further Study and Reading

- Stochastic optimization methods are at the core of modern large-scale machine learning, serving as one of the foundations of deep learning.
- A very active research area in machine learning and optimization.

Further readings:

- Bottou, L., Curtis, F. E., Nocedal, J. (2018). Optimization methods for large-scale machine learning. Siam Review, 60(2), 223-311.
- Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980
- Loshchilov, I., Hutter, F. (2017). Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101.

$\rightsquigarrow$ Next lecture: Introductory deep learning.