Optimization algorithms in deep learning

Sanzhar Askaruly (San)

Ulsan National Institute of Science and Technology
Ph.D. Candidate in Biomedical Engineering

CodeSeoul ML Meetup
November 5, 2022

Overview

What we'll cover today:

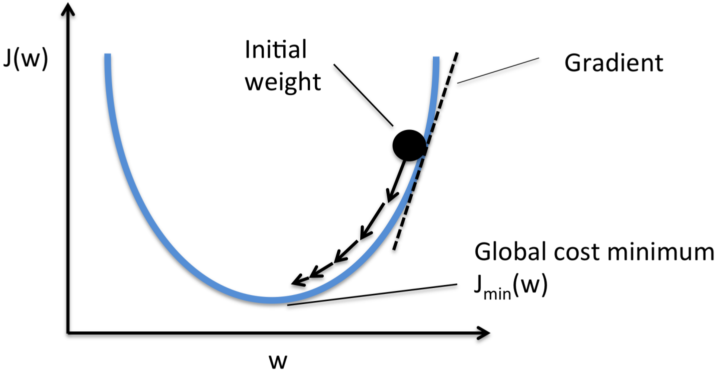
1. What is optimization?
2. Algorithms
   - SGD (Stochastic gradient descent)
   - SGD with Momentum
   - Adagrad (Adaptive learning rate)
3. Experiments
4. Summary
   - Discussion
   - Practicum

Optimization

In *context* of deep learning,
goal is to **minimize loss function**

$$w^* = \arg\min_{w} L(w) \tag{1}$$

What is gradient descent optimization?



Stochastic Gradient Descent (SGD)

Algorithm

Update step [1]:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta_t) \tag{2}$$

where,

θ_t:	current model parameters
$\nabla_\theta J(\theta_t)$:	gradient of these model parameters
η:	learning rate (fixed)

Stochastic Gradient Descent (SGD)

How we usually call in PyTorch:

```
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

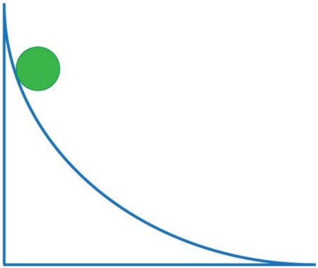
How we can create our "native" class [2]:

```
from torch.optim.optimizer import Optimizer

class CustomSGD(Optimizer):
  def __init__(self, model_params, lr=1e-3):
      self.model_params = list(model_params)
      self.lr = lr

  def zero_grad(self):
      for param in self.model_params:
          param.grad = None

  @torch.no_grad()
  def step(self):
      for param in self.model_params:
          param.sub_(self.lr * param.grad)
```

SGD with Momentum

General idea:

- Overcome small gradients near flat areas
- Build up from previous "velocity"
- Faster learning



SGD with Momentum

Algorithm

Update step [3]:

$$v_{t,i} = \gamma \cdot v_{t-1,i} + \nabla_\theta J(\theta_{t,i}) \tag{3}$$

$$\theta_{t+1} = \theta_t - \eta \cdot v_{t,i} \tag{4}$$

where,

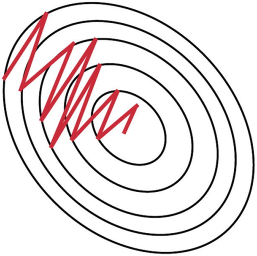
γ:	friction (or momentum, fixed)
v_t:	velocity
$\nabla_\theta J(\theta_t)$:	gradient of these model parameters
η:	learning rate (fixed)

SGD with Momentum

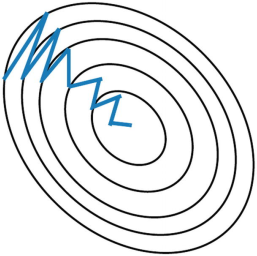
How we can create "native" `SGDMomentum` class:

```
from torch.optim.optimizer import Optimizer

class CustomSGDMomentum(Optimizer):
  def __init__(self, model_params, lr=1e-3, momentum=0.9):
      self.model_params = list(model_params)
      self.lr = lr
      self.momentum = momentum
      self.v = [torch.zeros_like(p) for p in self.model_params]

  def zero_grad(self):
      for param in self.model_params:
          param.grad = None

  @torch.no_grad()
  def step(self):
      for param, v in zip(self.model_params, self.v):
          v.mul_(self.momentum).add_(param.grad)
          param.sub_(self.lr * v)
```

SGD with Momentum



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Adagrad

Algorithm

Update step [4]:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}} + \epsilon} \cdot \nabla_\theta J(\theta_{t,i}) \tag{5}$$

where,

$$G_{t,i} = G_{t-1,i} + (\nabla_\theta J(\theta_{t,i}))^2 \tag{6}$$

and,

$G_{t,i}$:	the sum of the squared gradients
ϵ:	a small number, to avoid division by zero
θ_t:	current model parameters
$\nabla_\theta J(\theta_t)$:	gradient of these model parameters
η:	learning rate (fixed)

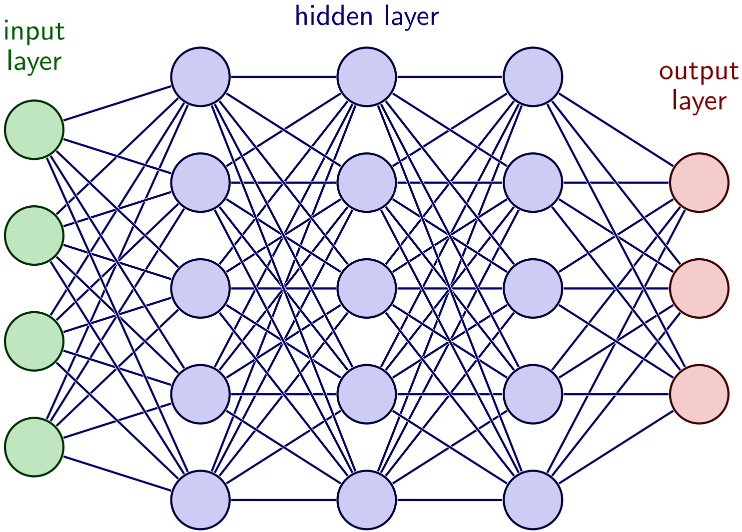
Adagrad

How we can create "native" `Adagrad` class:

```
from torch.optim.optimizer import Optimizer

class CustomAdagrad(Optimizer):
  def __init__(self, model_params, lr=1e-2, init_acc_sqr_grad=0, eps=1e-10)
      self.model_params = list(model_params)
      self.lr = lr
      self.acc_sqr_grads = [torch.full_like(p, init_acc_sqr_grad) for p in
      self.eps = eps

  def zero_grad(self):
      for param in self.model_params:
          param.grad = None

  @torch.no_grad()
  def step(self):
      for param, acc_sqr_grad in zip(self.model_params, self.acc_sqr_grads)
          acc_sqr_grad.add_(param.grad * param.grad)
          std = acc_sqr_grad.sqrt().add(self.eps)
          param.sub_((self.lr / std) * param.grad)
```

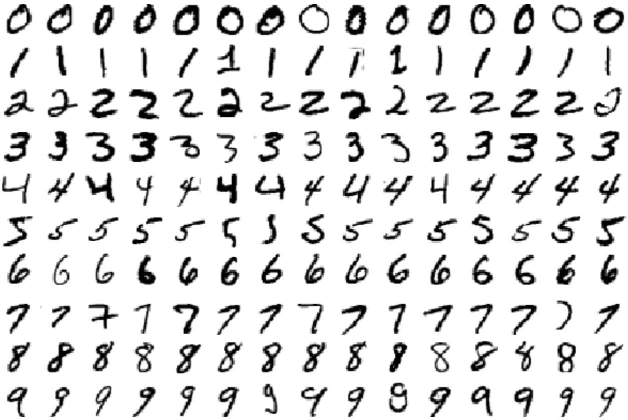
Experiment

A vanilla MLP (Multilayer Perceptron)



Experiment

MNIST dataset [5]



Experiment

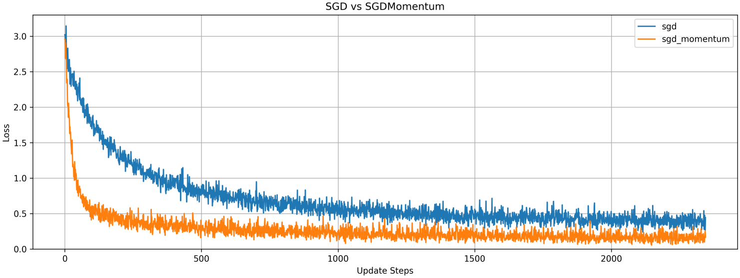


Figure 1: SGD vs momentum [git: CodeSeoul/machine-learning]

Experiment

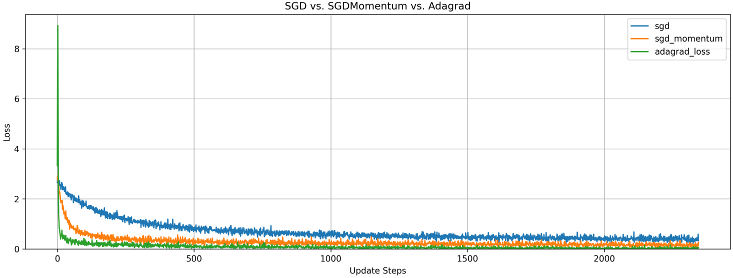


Figure 2: SGD vs momentum vs Adagrad [git: CodeSeoul/machine-learning]

Discussion

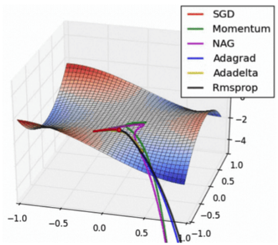


Figure 3: Trajectories of optimization algorithms in high-dimensional space [1]

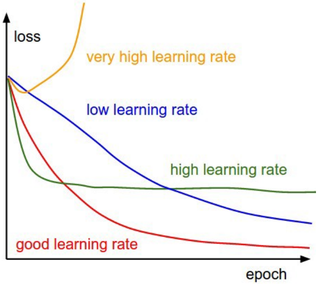


Figure 4: Selecting learning rate (lr) [6]

- The randomness introduced by SGD allows to reach better minimum. But in cases with many local minima, SGD may still get stuck.
- A systematic way to choose a good lr is by initially assigning it a very low value and increasing it slowly until the loss stops decreasing.

Practicum

Thank you for your attention!

- Workshop contents: https://github.com/CodeSeoul/machine-learning/tree/master/221105-optimization
- Follow-up QA? http://discord.com/users/tuttelikz

References

- Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).
- Mykola Novik. *torch-optimizer – collection of optimization algorithms for PyTorch.* Version 1.0.1. Jan. 2020.
- Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.
- John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization.". In: *Journal of machine learning research* 12.7 (2011).
- Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- Abinash Mohanty. *The Subtle Art of Fixing and Modifying Learning Rate*. Towards Data Science, July 2019. URL: https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303.