Machine Learning
Lecture 3

Yang Yuan

Review of the last lecture

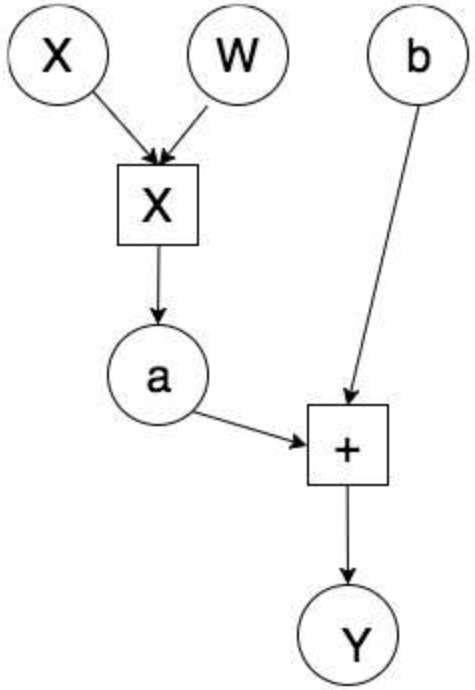
- How to create dataset (crowd sourcing, pipeline, recaptcha)
- Training loss, test loss, validation loss, population loss
- Overfit vs underfit
  - Regularization
- Unsupervised learning
  - Clustering
  - PCA
  - Generative model
  - Anomaly detection
  - Dimension reduction
- Semi-supervised learning

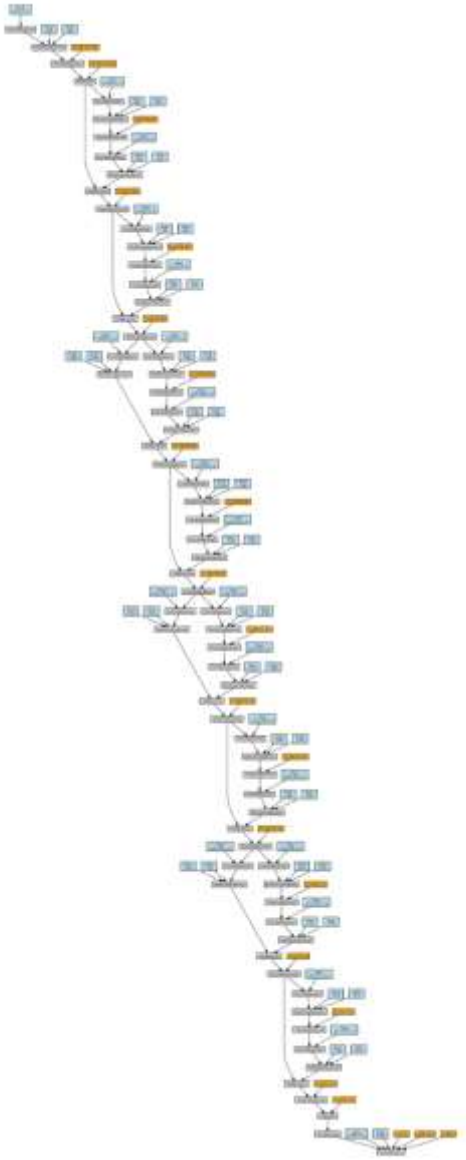
Review of the last lecture

- Use github to download code online
- Set up pycharm (connect to the server)
- Run jupyter notebook to write code interactively
- How to write a simple neural network model using pytorch

Pytorch: autograd feature

- When do the calculation, pytorch automatically calculate the computational graph
- The gradient will be computed automatically
- Takes time to understand what is differentiable, what is not
- All torch.nn stuff, and F.relu etc are differentiable!
- Therefore, only need to define function, not the **gradient of** function!





Arguments

- Clean and efficient way to organize hyperparameters
  - Recommended for big projects
- You may set default value, type of the argument, etc.

```
def main():
    # Training settings
    parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
    parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                        help='input batch size for training (default: 64)')
    parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                        help='input batch size for testing (default: 1000)')
    parser.add_argument('--epochs', type=int, default=10, metavar='N',
                        help='number of epochs to train (default: 10)')
    parser.add_argument('--lr', type=float, default=0.01, metavar='LR',
                        help='learning rate (default: 0.01)')
    parser.add_argument('--momentum', type=float, default=0.5, metavar='M',
                        help='SGD momentum (default: 0.5)')
    parser.add_argument('--no-cuda', action='store_true', default=False,
                        help='disables CUDA training')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')
    parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                        help='how many batches to wait before logging training status')

    parser.add_argument('--save-model', action='store_true', default=False,
                        help='For Saving the current Model')
    args = parser.parse_args()
    use_cuda = not args.no_cuda and torch.cuda.is_available()
```

xx.cuda()?

- Run a.cuda() / a.cpu() move the model/data to/from GPU
- GPU has much faster running time, but CPU (usually) has much larger memory
- Use GPU for computation, use CPU for storing data
- CPU model can’t process GPU data, and GPU model can’t process CPU data

```
device = torch.device("cuda" if use_cuda else "cpu")

model = Net().to(device)
```

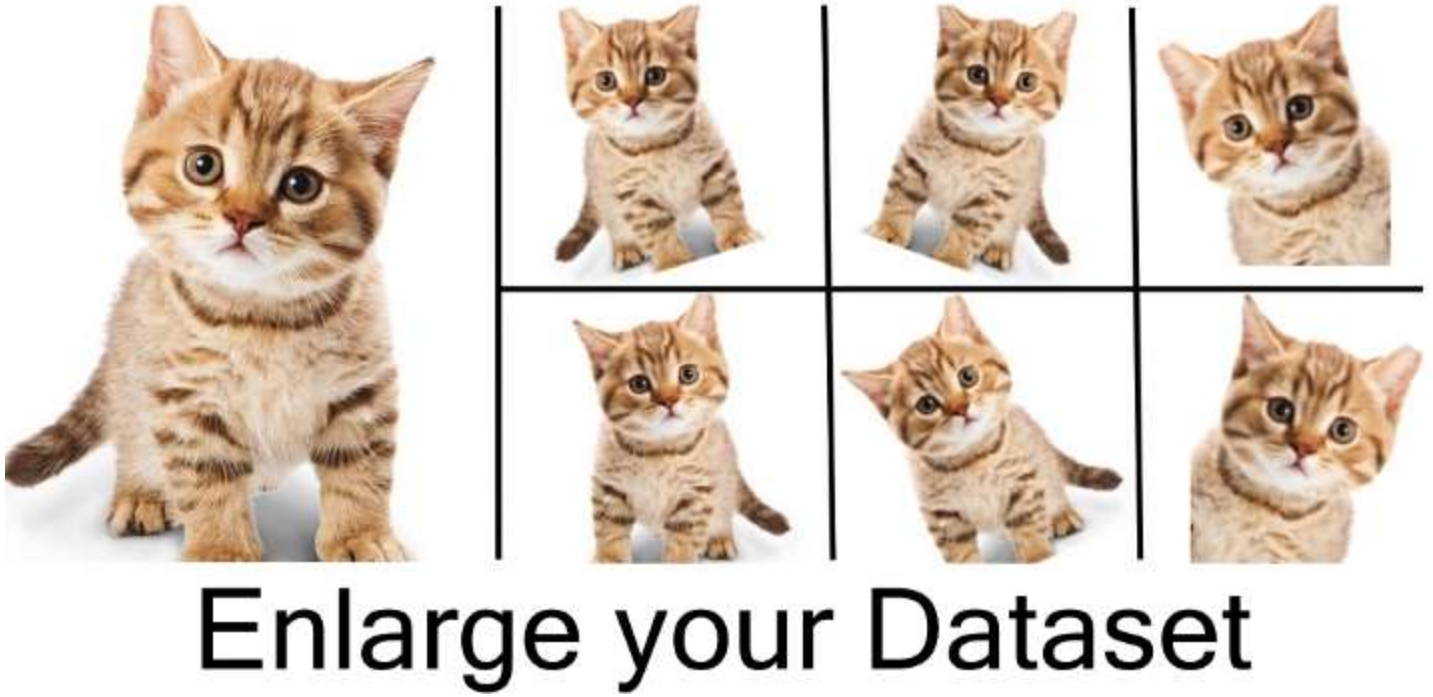
```
data, target = data.to(device), target.to(device)
```

Read data

```
data, target = data.to(device), target.to(device)
```

- Naïve way: put data into Tensor, then run cuda() to move to GPU, then use model(data) to compute
- Better way: use dataloader to read dataset automatically
  - Supports shuffling the data
  - Set batch size
  - Add transformation
    - Data augmentation
  - Download from the Internet

```
train_loader = torch.utils.data.DataLoader(
```



Optimizer and epochs

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

for epoch in range(1, args.epochs + 1):
    train(args, model, device, train_loader, optimizer, epoch)
    test(args, model, device, test_loader)
```

- Optimizer:
  - Run SGD/GD/all other variants automatically
  - Add weight decay, momentum, Nesterov's acceleration, etc.
  - Convenient to use
- Epochs:
  - Every epoch, run training, then run testing (eval on test set)

Training

- Model.train()
  - Training mode and test mode are different from some layers
  - Batch norm, dropout, etc
- Zero_grad
  - Clear gradient in tensors
  - Otherwise gradient will accumulate
- Output=model(data)
  - Forward pass

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

Training

- Loss=F.nll_loss(output,target)
  - Compute the distance using some loss function
- Loss.backward()
  - Compute gradient of loss (automatically)
  - Run SGD/GD for loss
- Optimizer.step()
  - Using grad to update the weights of the network
  - Loss decreases

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```

Testing

```
def test(args, model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
```

- With torch.no_grad()
  - No need to compute gradient below (since we are not training)
  - Pytorch runs faster (not building the computational graph)
- Pred=output.argmax()
  - Output probability
  - Our prediction is the maximum prob class

How to write your own optimizer

In [10]:
```
for i in range(100):
    if not (a.grad is None):
        a.grad.fill_(0)
    loss=torch.norm(a-1)
    loss.backward()
    a.data=a.data-0.01*a.grad
    if i%10==0:
        print(loss.data,a.data)
```

```
tensor(1.1328) tensor([0.2303, 0.4524, 0.8465, 0.6403, 0.5358])
tensor(1.0328) tensor([0.2989, 0.5012, 0.8602, 0.6723, 0.5771])
tensor(0.9328) tensor([0.3674, 0.5499, 0.8738, 0.7043, 0.6185])
tensor(0.8328) tensor([0.4360, 0.5987, 0.8875, 0.7364, 0.6598])
tensor(0.7328) tensor([0.5045, 0.6475, 0.9012, 0.7684, 0.7012])
tensor(0.6328) tensor([0.5731, 0.6962, 0.9148, 0.8005, 0.7425])
tensor(0.5328) tensor([0.6416, 0.7450, 0.9285, 0.8325, 0.7839])
tensor(0.4328) tensor([0.7102, 0.7938, 0.9422, 0.8645, 0.8252])
tensor(0.3328) tensor([0.7787, 0.8426, 0.9559, 0.8966, 0.8665])
tensor(0.2328) tensor([0.8473, 0.8913, 0.9695, 0.9286, 0.9079])
```

投票 最多可选1项

设置

Is it clear? Do we need to go back?

A It is clear

B Let us go back

提交

Optimization methods

Why do we need optimization?

- Human-beings can only create simple stuff, do simple things
  - Nothing complicated
- Unfortunately, simple model cannot fit the complicated world!
  - 请不要使用一条直线丈量世界
- What shall we do, then? We human-beings,
  - Design **simple** structure (neural network), with millions/billions of parameters
  - Initialize the parameters with **simple** random numbers
  - Collect lots of **simple** data
  - Run a **simple** training method (optimization)
  - Finally get a really **complicated** model that fits the data!
- Optimization is the key: a simple method to find set of parameters to fit the data, but the parameter space is huge!

Optimization methods

- Assume we try to optimize $f(x)$
- Zero-th order methods: only has the information of $f(x)$
- First order methods: has information of $f(x)$ and $\nabla f(x)$
- Second order methods: has information of $f(x), \nabla f(x), \nabla^2 f(\mathrm{x})$ (i.e., Hessian matrix)
- Usually no one uses higher order methods (too time consuming)

Optimization methods

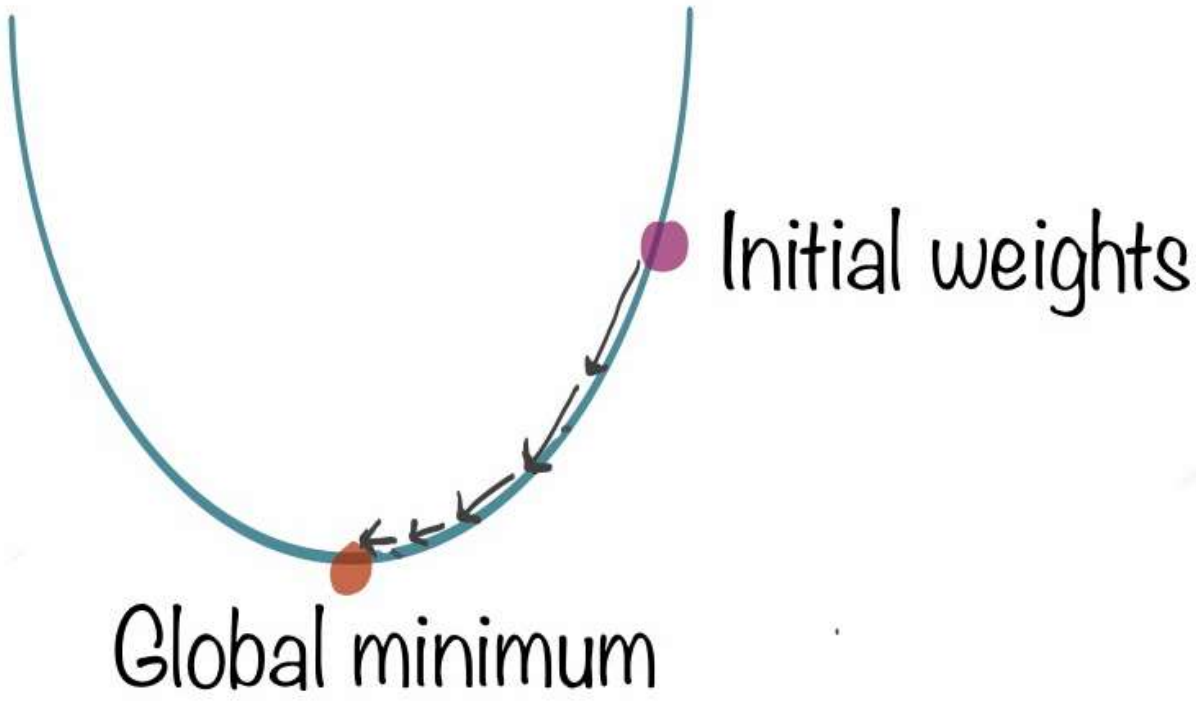
- Zeroth-order method: hard to optimize, only has the information of the current point!
  - Hyperparameter tuning
  - Function not differentiable
  - Sometimes this is the only possible method, without further information
- Second-order method:
  - The naïve way is too time consuming
  - Hessian matrix has size $O(d^2)$, where d is #parameters
    - Huge!!!
  - 1.5-th order method, say BFGS methods, using Hessian-vector product

Optimization≈(Stochastic) Gradient Descent

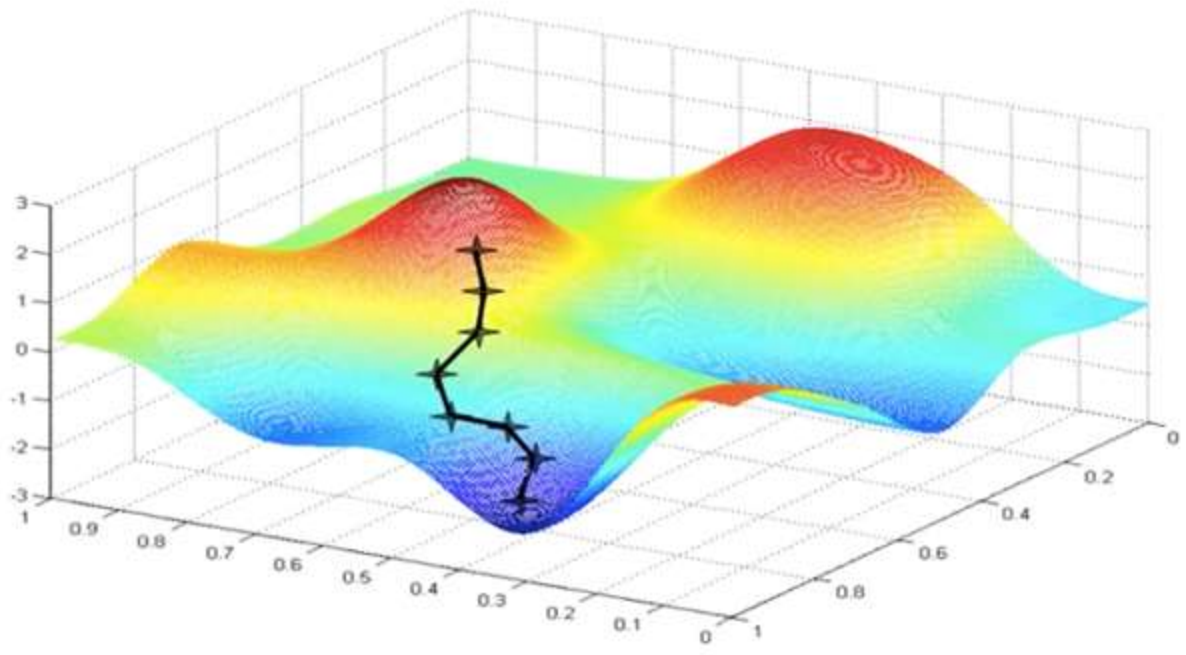
- There are millions of ML problems/target functions
  - But (almost) just one algorithm
- Unlike Algorithms and Data Structure course
  - Every problem has a unique algorithm/data structure
- Not because ML researchers are lazy/stupid!
- GD algorithm is powerful and fundamental
  - Someone claimed that it's "mother of all algorithms"
  - Can solve lots of classical problems
    - maximum flow, bipartite matching, the k-server, etc.

Gradient descent

- To minimize a function $f(x)$
  - $x_{t+1} = x_t - \eta \nabla f(x_t)$
  - η is step size (learning rate)
    - How much we go for each step?
- Intuition:
  - Gradient gives the direction to decrease the function
  - Take one step of that direction
  - Do this iteratively (gradient direction may change over time)



More figures about gradient descent

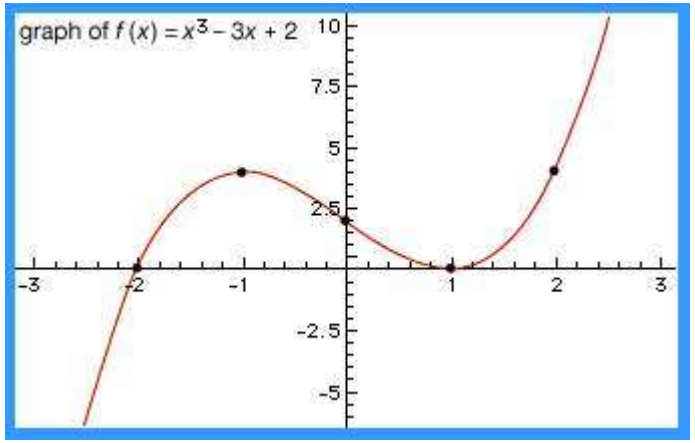


Some theory behind

- L-Lipschitz:

$$\forall\, x, y, |f(x) - f(y)| \leq L\|x - y\|$$

  - It means, the function f does not change too fast
- Local minimum:
  - If x^* is a local minimum of a differentiable function $f(\mathrm{x})$, then $\nabla f(x^*) = 0$
- Notice that this is a necessary condition! Not sufficient.



Some theory behind

- Local minimum sufficient condition:
  - $\nabla f(x) = 0,\ \nabla^2 f(x) > 0$ (positive definite)
  - Why?
- Hessian matrix $\nabla^2 f(x)$ is “gradient for gradient”
  - If we look at the eigenspace of $\nabla^2 f(x)$
  - Oh, look at each eigenvector
  - It measures how gradient $\nabla f\ (x)$ changes along the direction
- So, $\nabla^2 f(x) > 0$ means, no matter which direction you go, $\nabla f(x)$ increases (from 0 to positive value)
- $\nabla f(x)$ measures how $f(x)$ changes, so we know $f(x)$ always increases along all directions

投票 最多可选1项

设置

Is it clear? Do we need to go back?

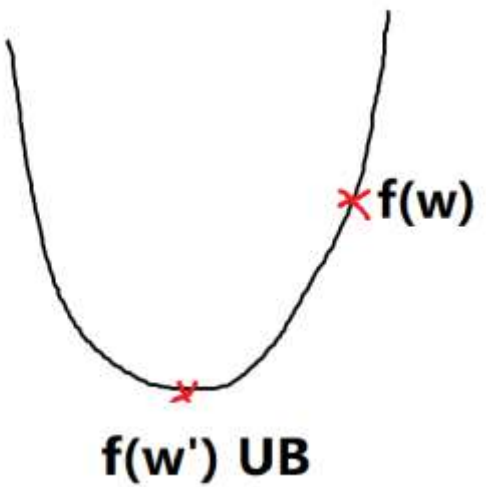
A It is clear

B Let us go back

提交

Deeper reasons for gradient descent

- It's based on first order Taylor expansion:
  - $f(w') = f(w) + \langle \nabla f(w), w' - w \rangle + \frac{g(w')}{2} \|w' - w\|^2$
  - We are at w, want to estimate w'
  - We approximate $f(w')$ with $f(w) + \langle \nabla f(w), w' - w \rangle$
    - With a tail $\frac{g(w')}{2} \|w' - w\|^2$
- **Smoothness assumption**: $\exists L, |g(w')| \leq L$ for all w'
- $f(w') \leq f(w) + \langle \nabla f(w), w' - w \rangle + \frac{L}{2} \|w' - w\|^2$
- Right hand side is a quadratic function
  - Has a global minimum, upper bound for $f(w')$



Smoothness assumption

- Alternative view of smoothness assumption:
  - Gradient is Lipschitz
    - $||\nabla f(w) - \nabla f(w')|| \leq L||w - w'||$ for all w, w'
  - Equivalent to $|f(w') - f(w) - \langle \nabla f(w), w' - w \rangle| \leq \frac{L}{2}||w - w'||^2$
    - Because the function's gradient is Lipschitz, it cannot grow faster than rate L

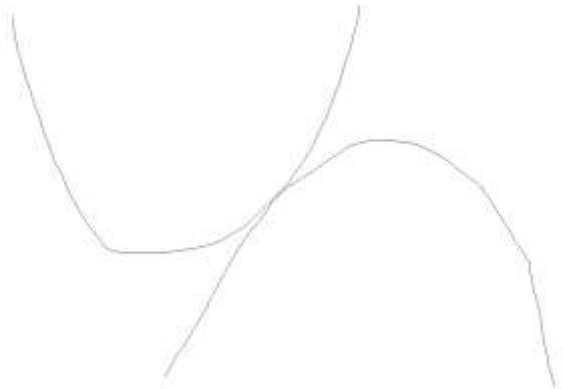
Smoothness condition (Lem 1.2.3 in Nesterov)

- Gradient Lipschitz is equivalent as $|f(y) - f(x) - \langle \nabla f(x), y - x \rangle| \leq \frac{L}{2}||y - x||^2$
- $\Rightarrow: f(y) = f(x) + \int_0^1 \langle \nabla f\big(x + \tau(y - x)\big), y - x \rangle d\tau = f(x) + \langle \nabla f(x), y - x \rangle + \int_0^1 \langle \nabla f\big(x + \tau(y - x)\big) - \nabla f(x), y - x \rangle d\tau$
- Therefore $|f(y) - f(x) - \langle \nabla f(x), y - x \rangle| \leq \int_0^1 \left\| \nabla f\big(x + \tau(y - x)\big) - \nabla f(x) \right\| \cdot \|y - x\| d\tau = \frac{L}{2}\|y - x\|^2$
- The other direction is also true: if not, there must exists a point with larger Lipschitz condition on gradient

Smoothness give lower/upper bounds

- Given x_0, consider two quadratic functions
- $\phi_1(x) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle + \frac{L}{2}\|x - x_0\|^2$
- $\phi_2(x) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle - \frac{L}{2}\|x - x_0\|^2$
- Then the graph of f is between the graph of ϕ_1 and ϕ_2:

$$\phi_1(x) \geq f(x) \geq \phi_2(x)$$



Smoothness condition (Lem 1.2.2 in Nesterov)

- $\nabla f(y) = \nabla f(x) + \int_0^1 \nabla^2 f\big(x + \tau(y-x)\big)(y-x)d\tau$

$$= \nabla f(x) + \left(\int_0^1 \nabla^2 f\big(x + \tau(y-x)\big)d\tau\right) \cdot (y-x)$$

If $\|\nabla^2 f(x)\| \leq L$, we have

$$\|\nabla f(y) - \nabla f(x)\| = \left\|\left(\int_0^1 \nabla^2 f\big(x + \tau(y-x)\big)d\tau\right) \cdot (y-x)\right\|$$

$$\leq \left\|\int_0^1 \nabla^2 f\big(x + \tau(y-x)\big)d\tau\right\| \cdot \|y-x\| \leq L \cdot \|y-x\|$$

The other side is also true (gradient of gradient is bounded by L)

Deeper reasons for gradient descent

- If $w' = w - \eta \nabla f(w)$, we have

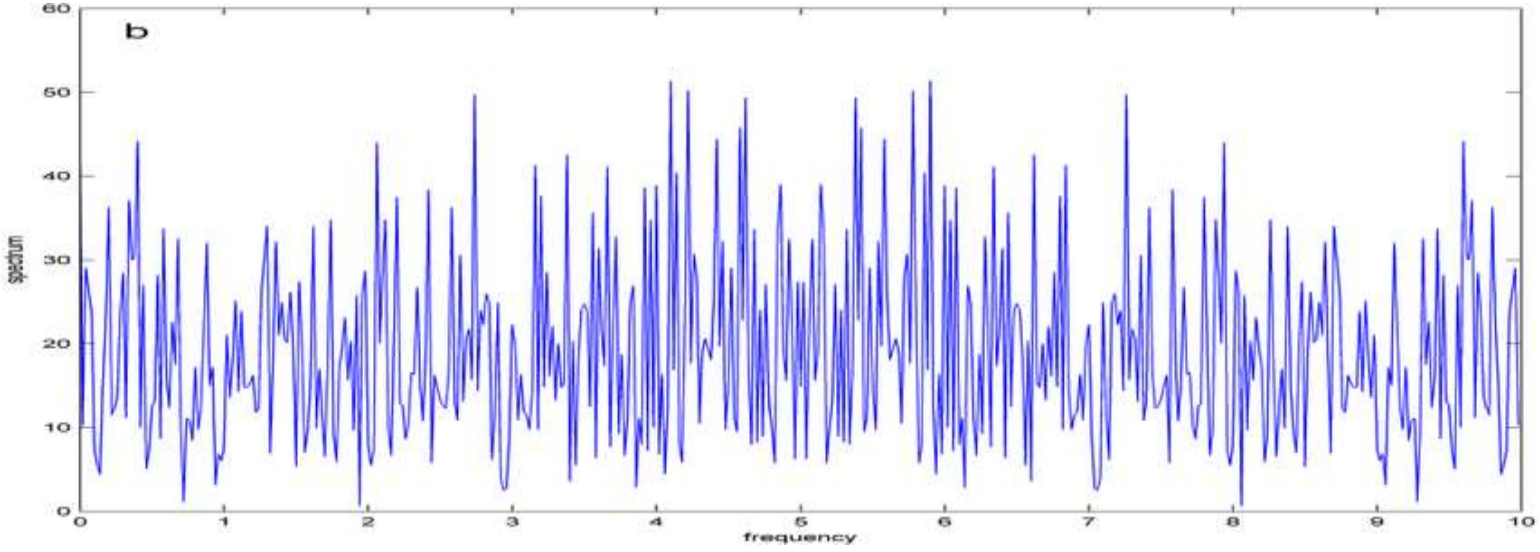
$$f(w') - f(w) \leq \langle \nabla f(w), w' - w \rangle + \frac{L}{2} \|w' - w\|^2$$

$$= \langle \nabla f(w), -\eta \nabla f(w) \rangle + \frac{L\eta^2}{2} \left|\left|\nabla f(w)\right|\right|^2 = -\eta \left(1 - \frac{L\eta}{2}\right) ||\nabla f(w)||^2$$

So we should set $\boldsymbol{\eta < \frac{2}{L}}$ to make sure $f(w') - f(w) < 0$

Smoothness and gradient descent

- Smoothness tells us how to set the step size
  - $\eta = O\left(\frac{1}{L}\right)$
  - L is large means less smooth, means smaller step size
    - It also means gradient Lipschitz constant is larger, so gradient may change more rapidly
- Smoothness -> GD decreases function value!



投票 最多可选1项

设置

Is it clear? Do we need to go back?

A It is clear

B Let us go back

提交