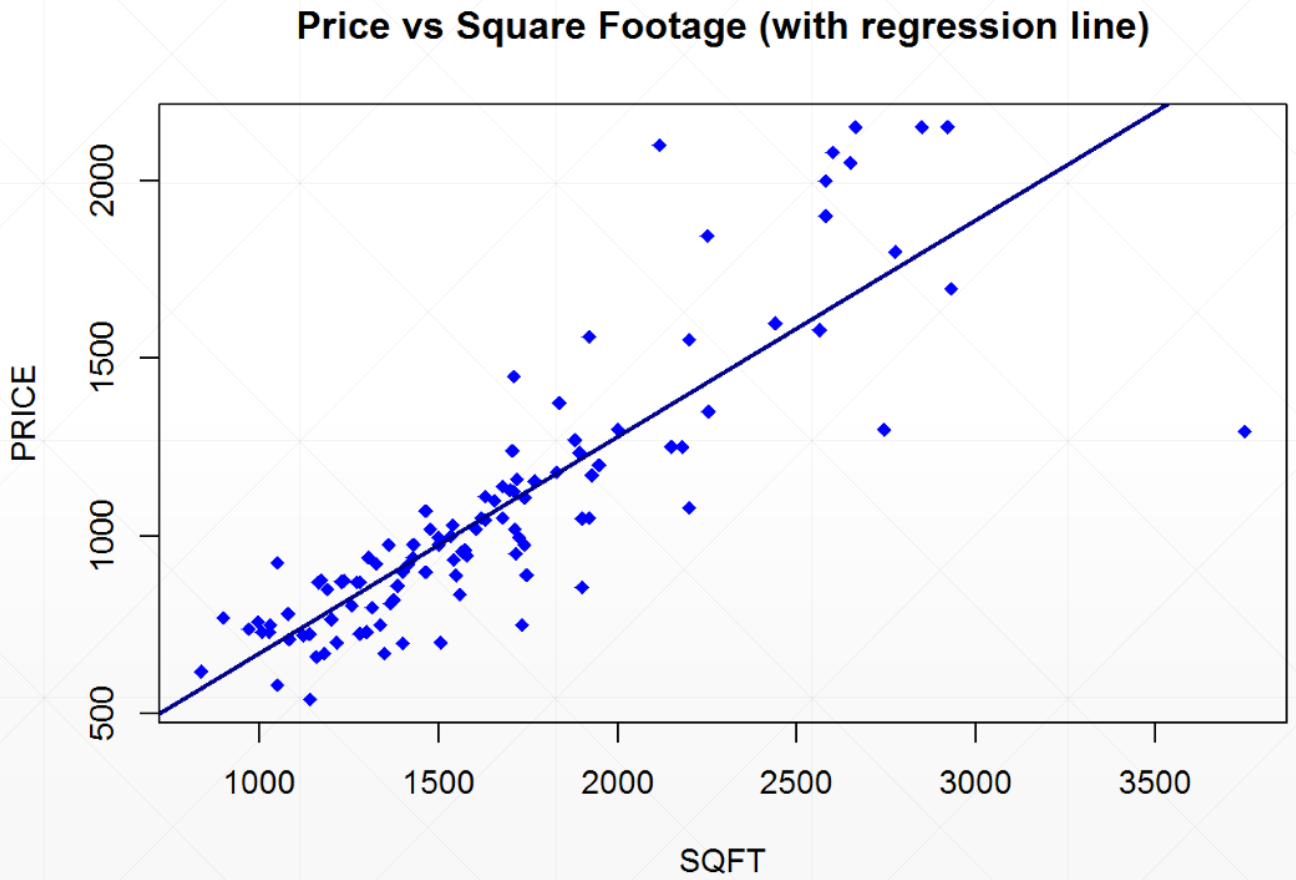
PyTorch

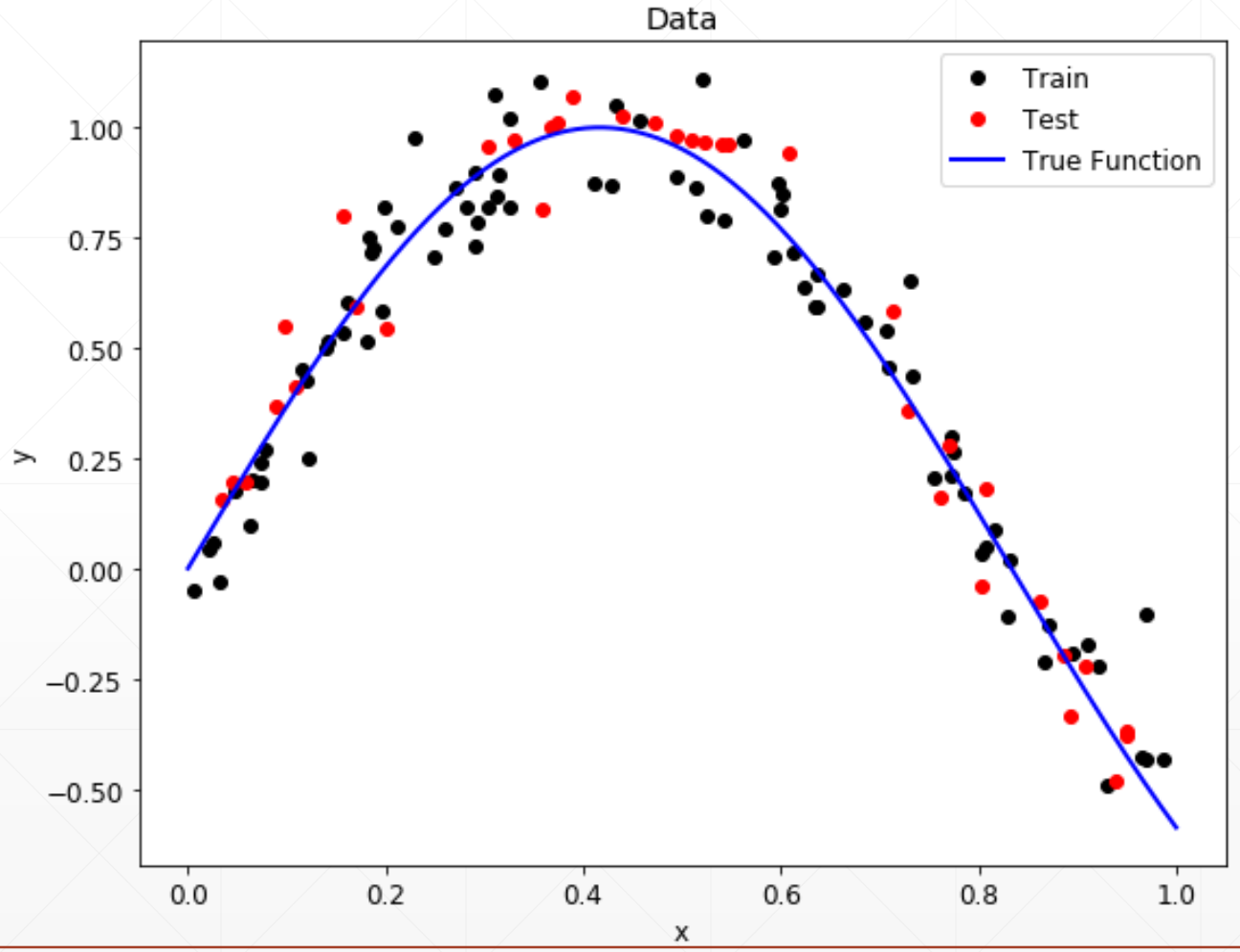
过拟合&欠拟合

主讲人：龙良曲

Scenario1: house price



Scenario2: GPA



The ground-truth distribution?

- That's perfect if known

- However

Another factor: noise

- $y = w * x + b + \epsilon$
- $\epsilon \backsim N(0.01, 1)$

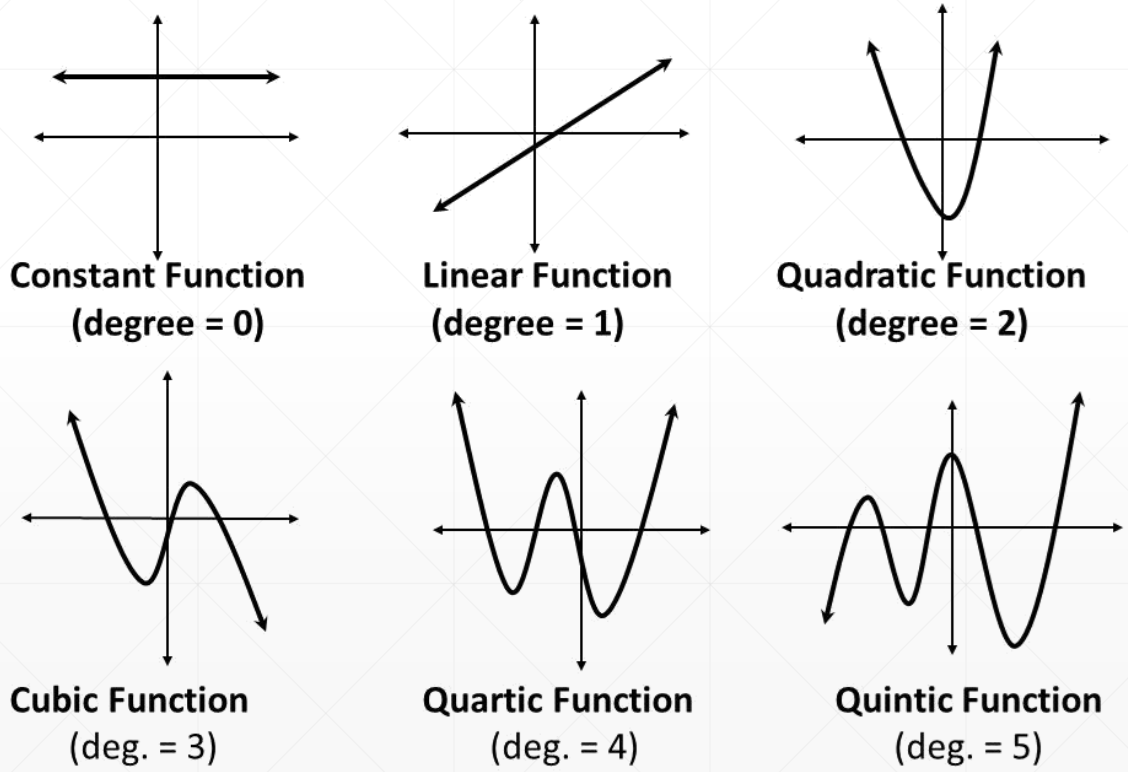
$$loss = (WX + b - y)^2$$

- 1.567 = w * 1 + b + eps
- 3.043 = w * 2 + b + eps
- 4.519 = w * 3 + b + eps
- ...

Let's assume

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

Graphs of Polynomial Functions:

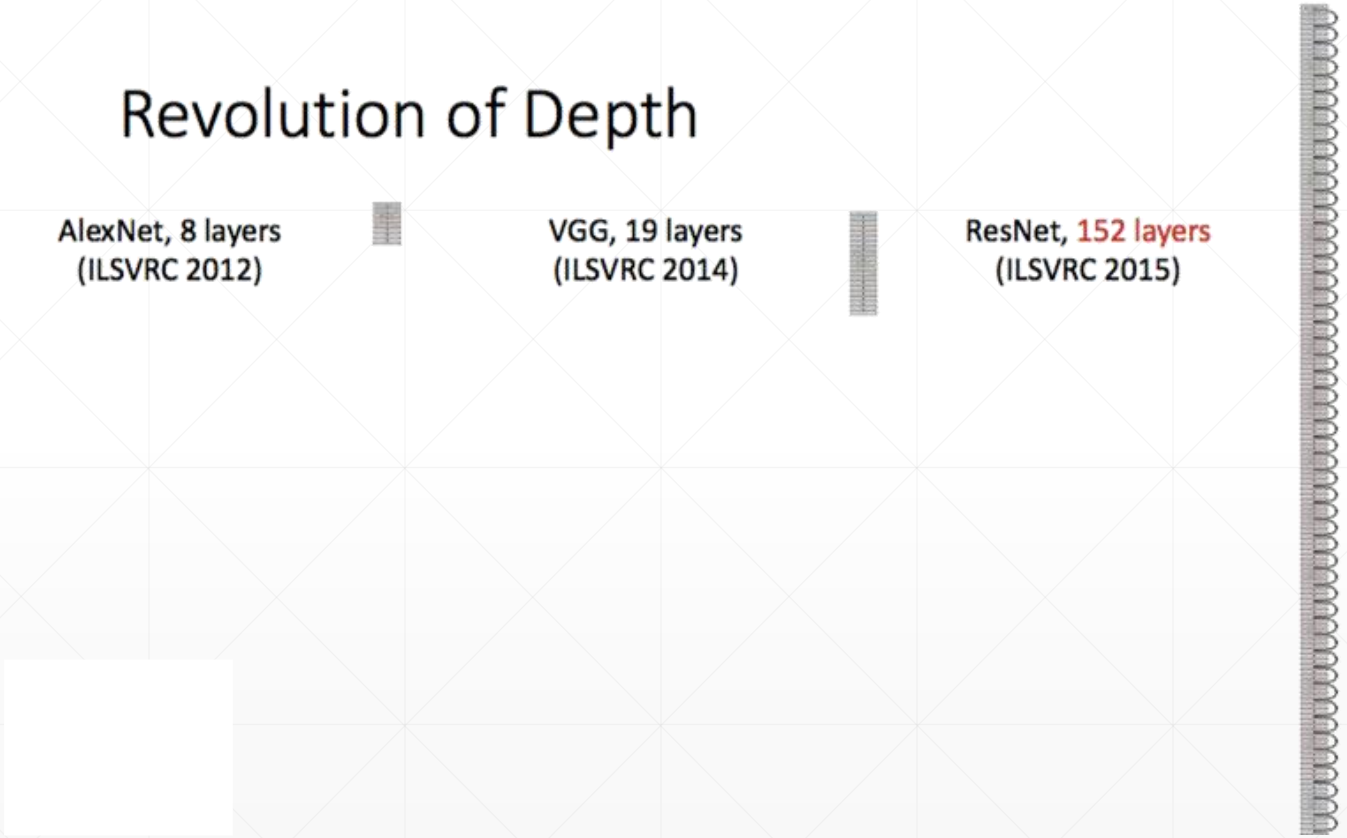


Mismatch: ground-truth VS estimated

- model capacity

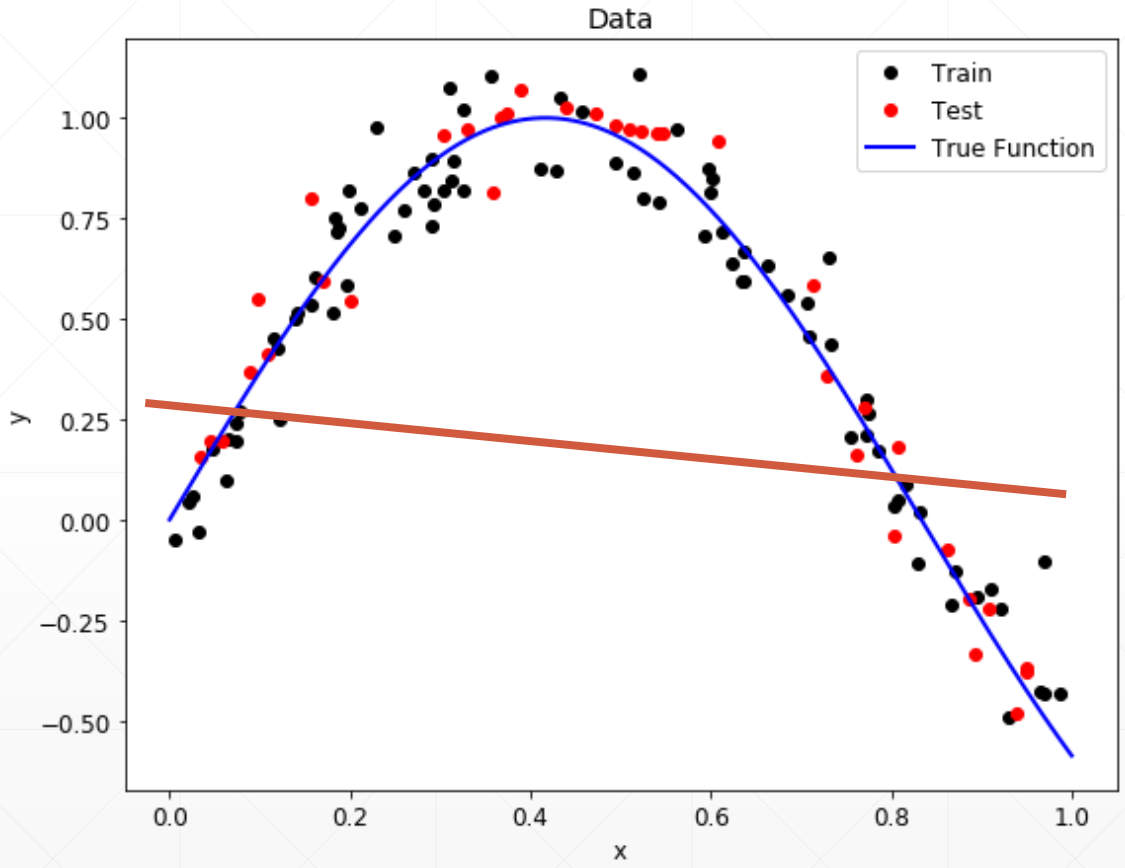
$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

Model Capacity

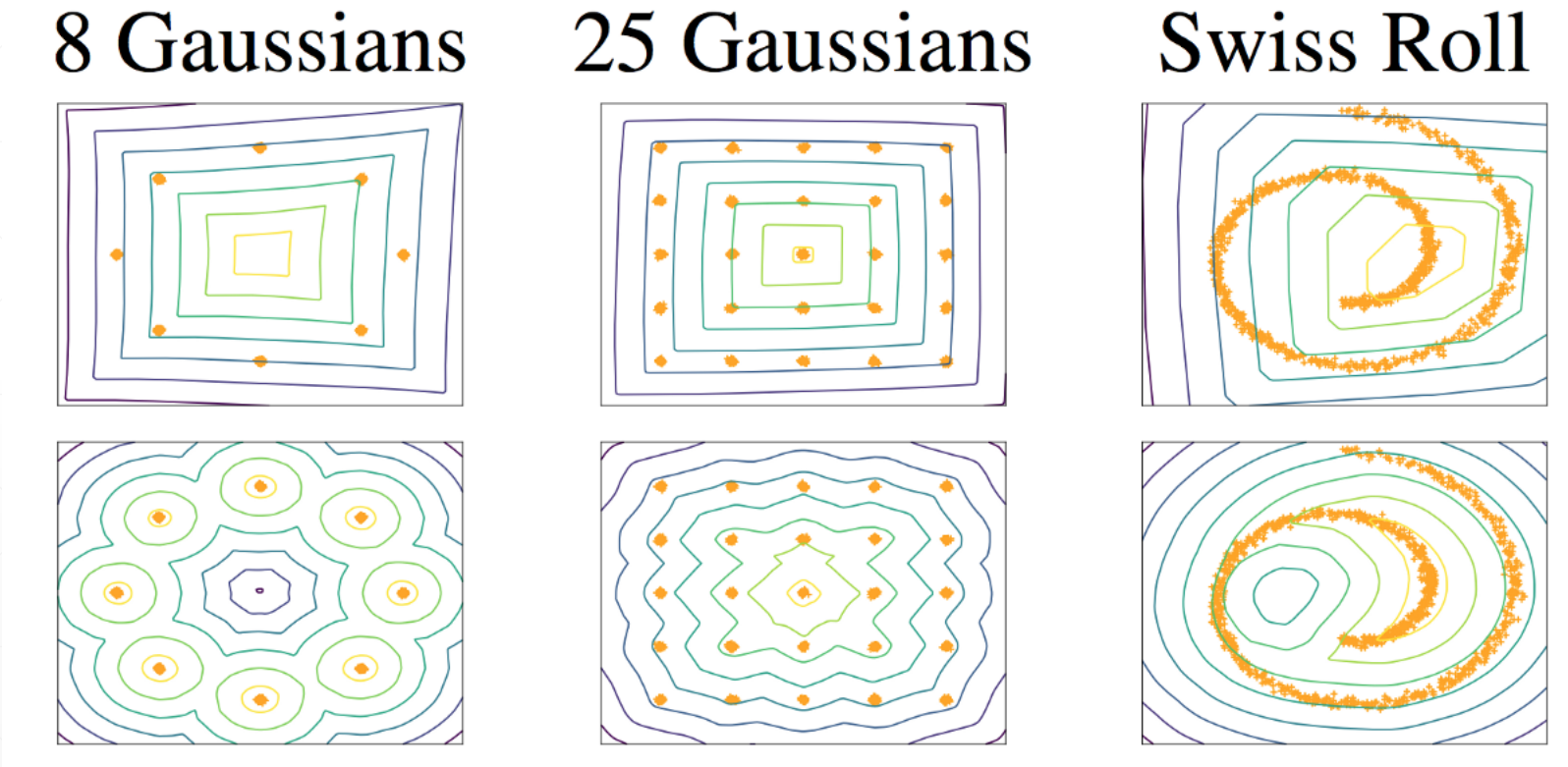


Case1: Estimated < Ground-truth

under-fitting



For example: WGAN

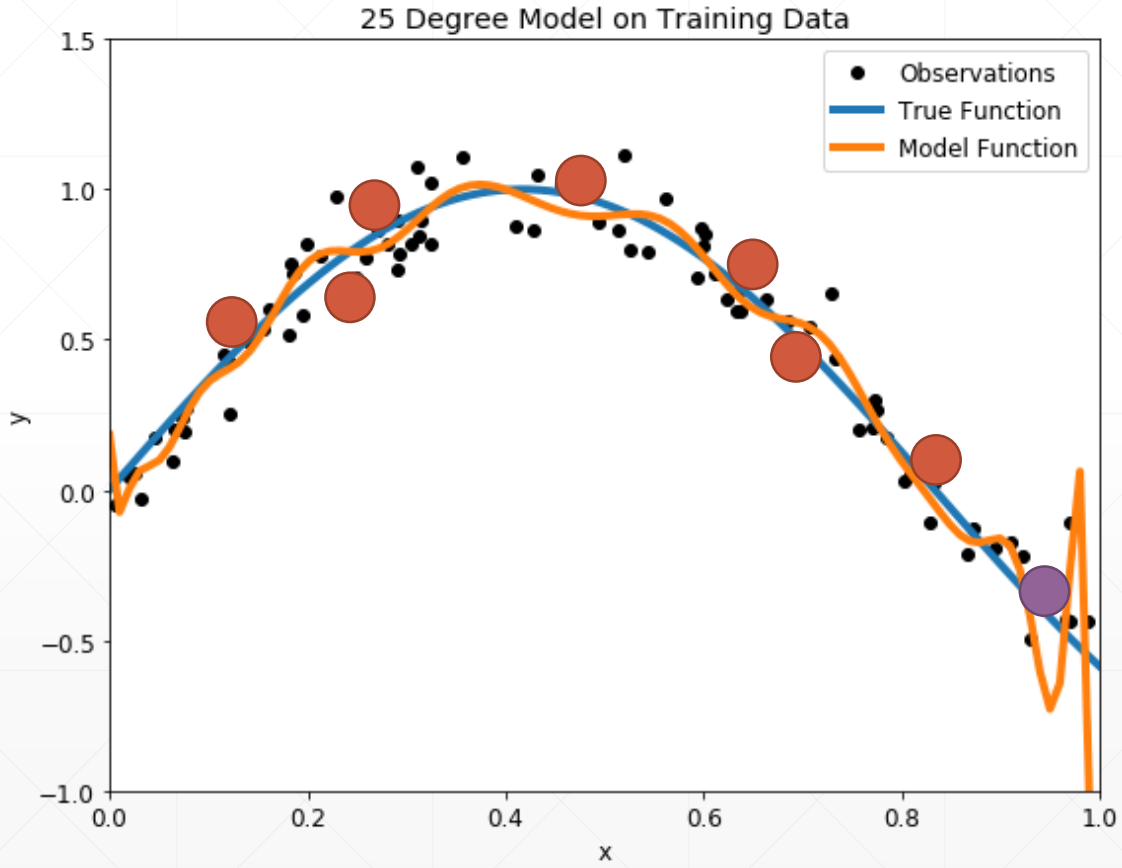


Underfitting

- train acc. is bad
- test acc. is bad as well

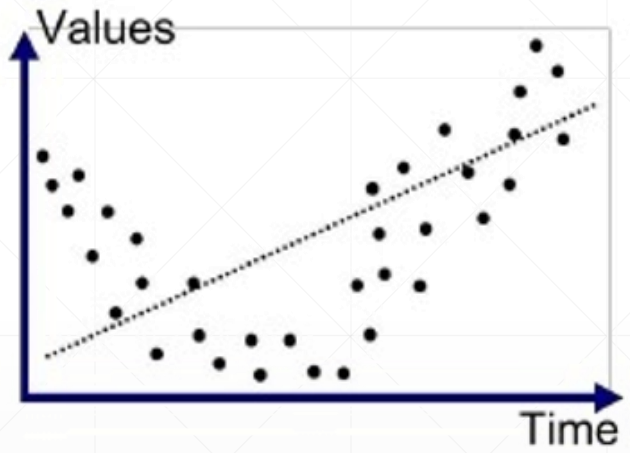
Case2: Ground-truth < Estimated

over-fitting

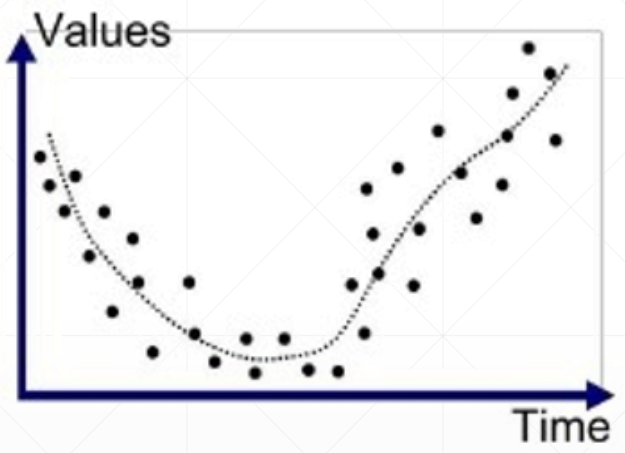


Overfitting

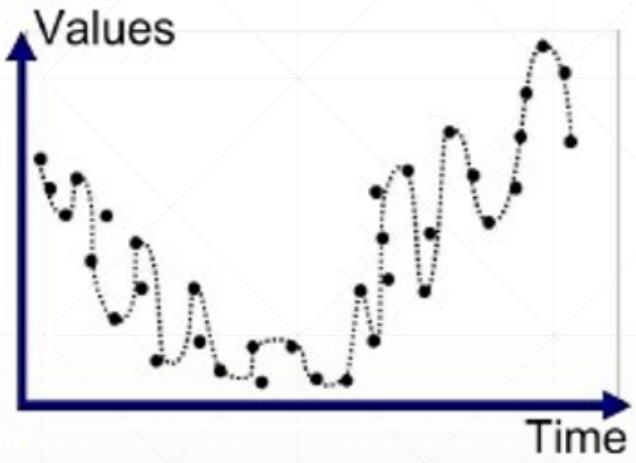
- train loss and acc. is much better
- test acc. is worse
- => Generalization Performance



Underfitted



Good Fit/Robust



Overfitted

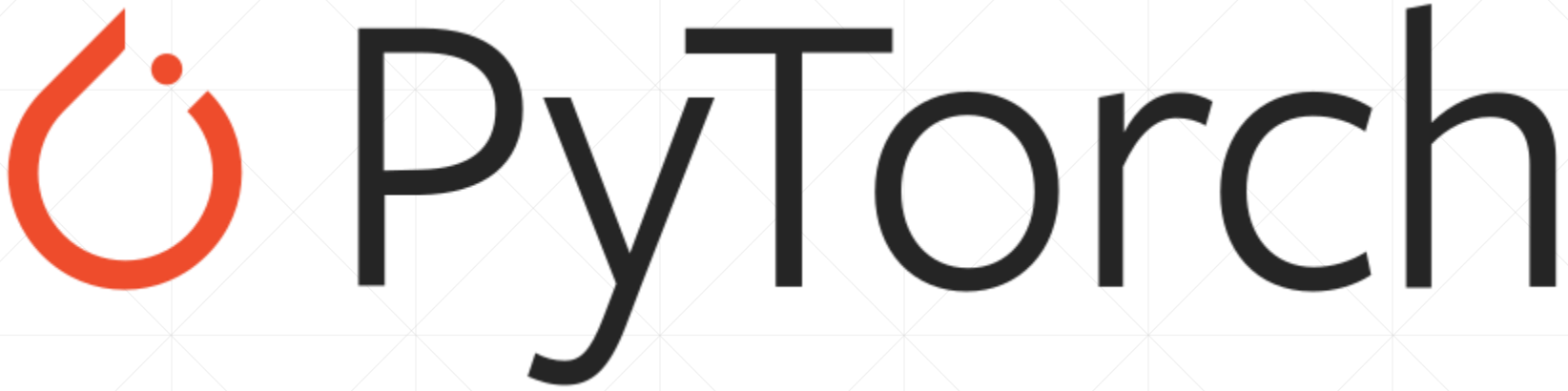
Overfitting !

- how to detect
- how to reduce

下一课时

train-val-test
划分

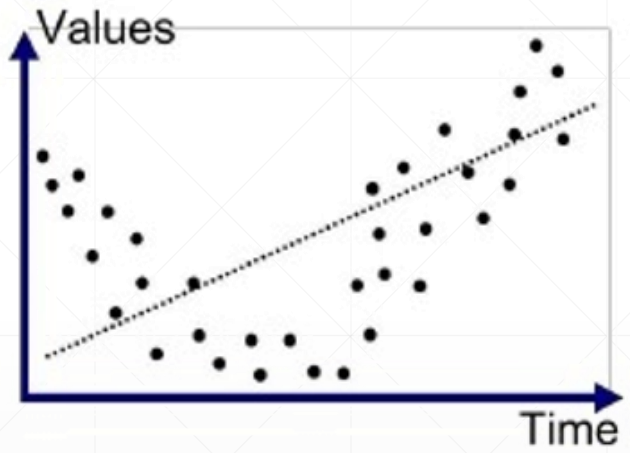
Thank You.



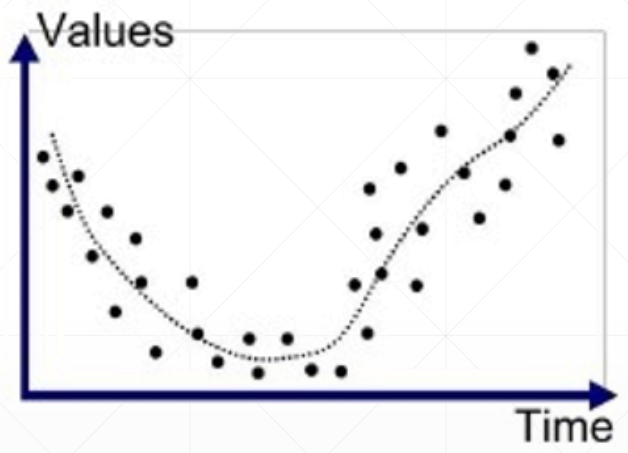
Train-Val-Test划分

主讲人：龙良曲

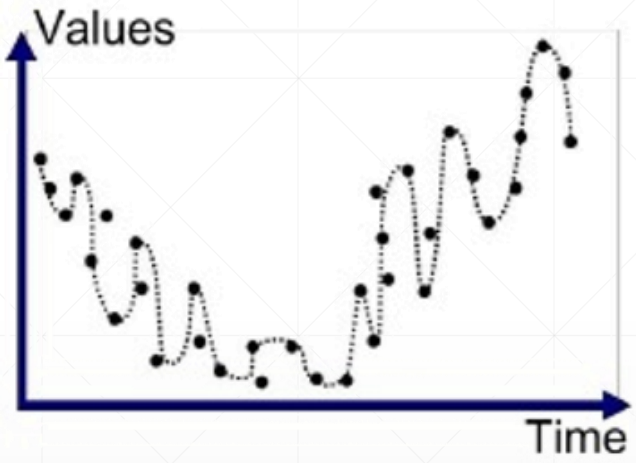
Recap



Underfitted

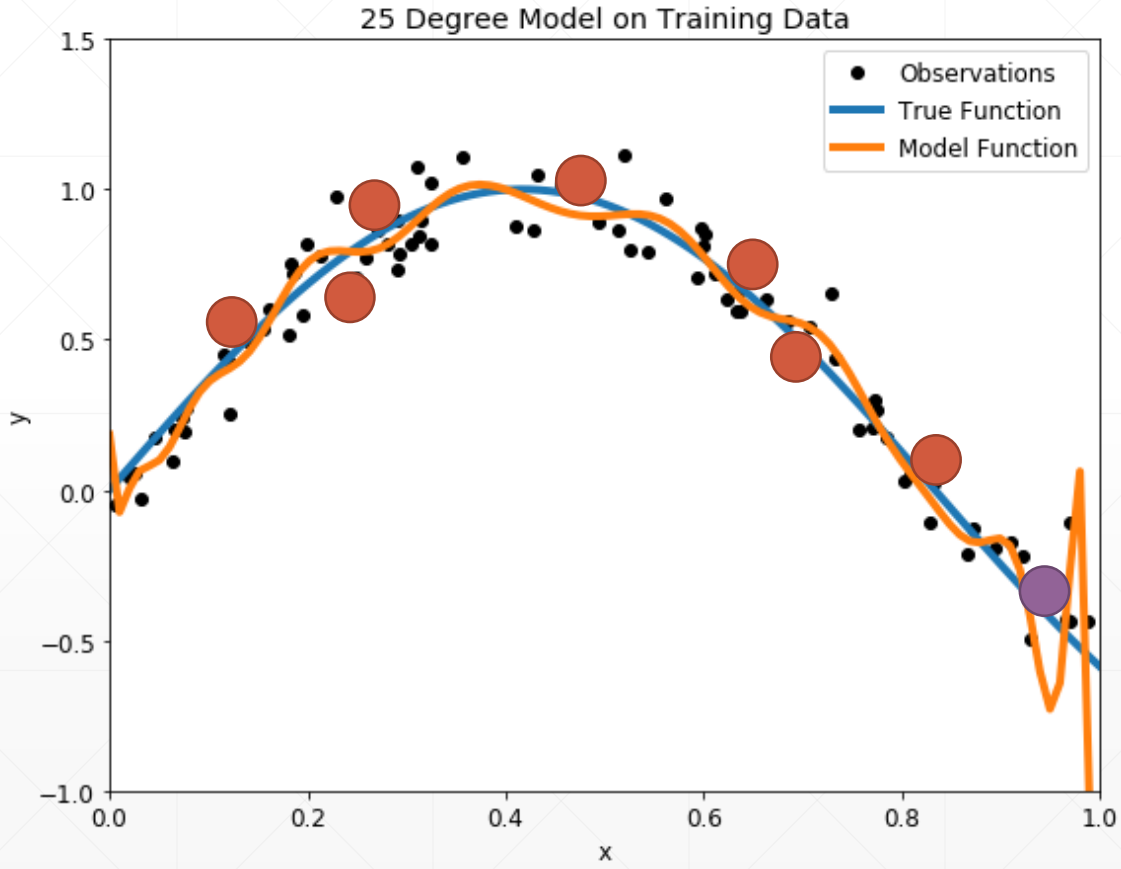


Good Fit/Robust



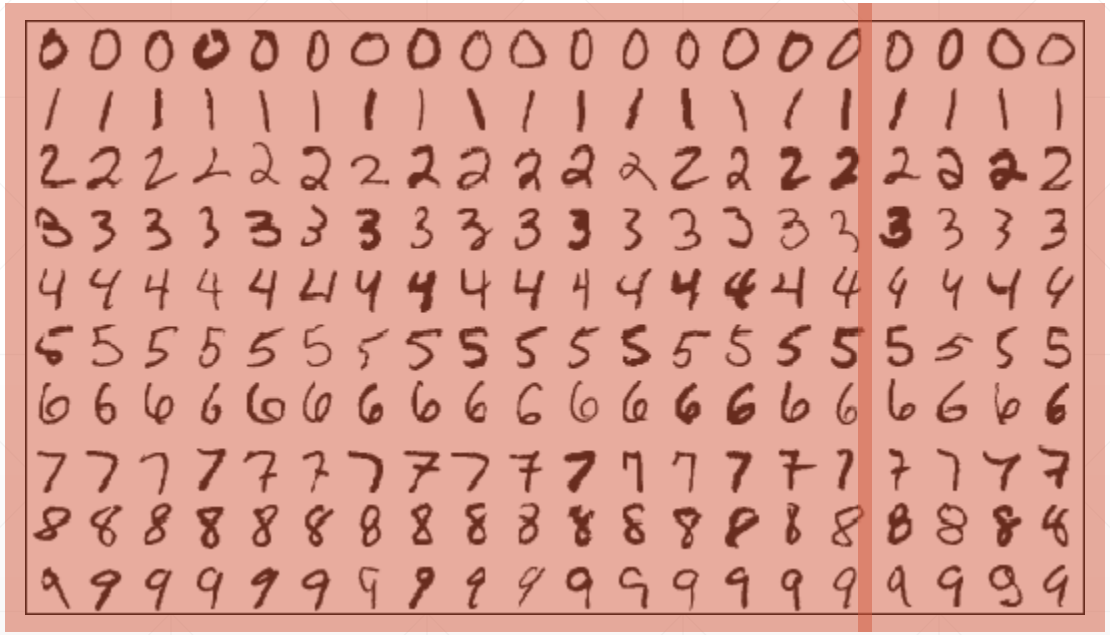
Overfitted

How to detect



Splitting

Train Set



Test Set

For example

60K

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])),
    batch_size=batch_size, shuffle=True)
```

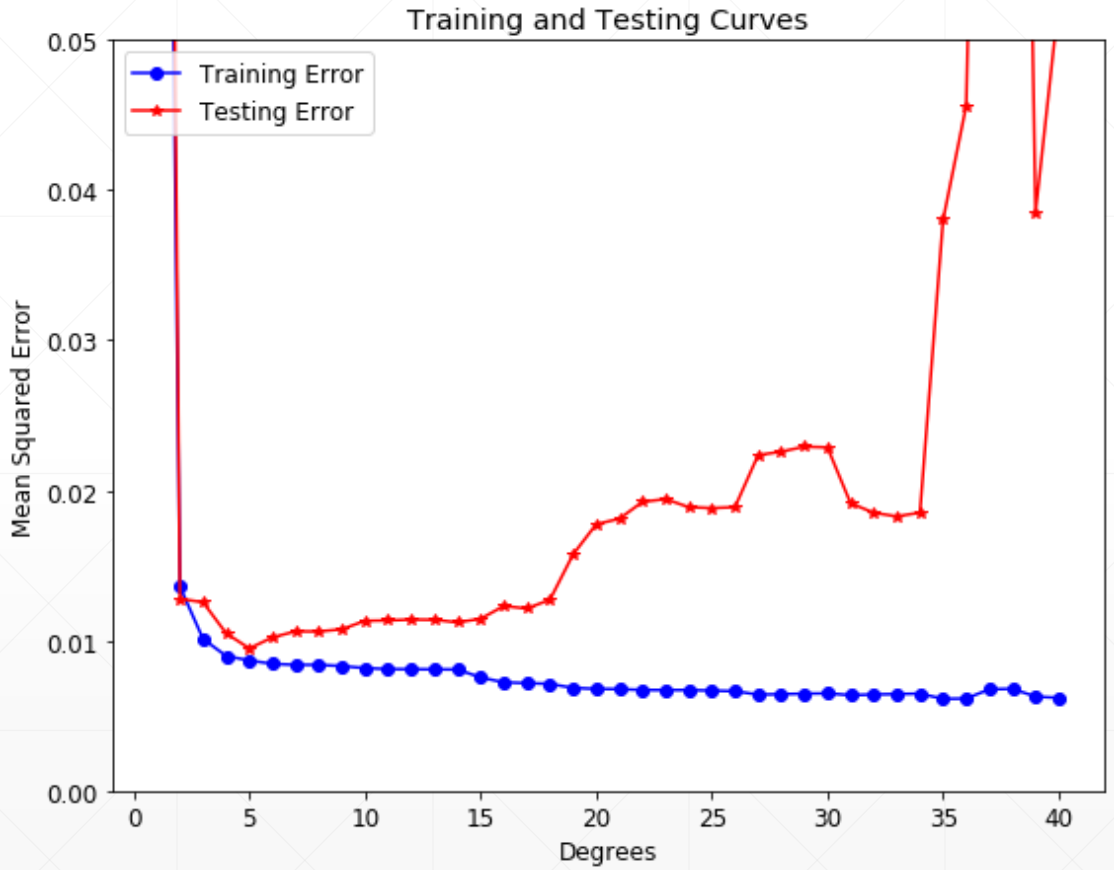
10K

test while train

```
for epoch in range(epochs):

    for batch_idx, (data, target) in enumerate(train_loader):
        ...
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 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()))

    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data = data.view(-1, 28 * 28)
        pred = logits.data.max(1)[1]
        correct += pred.eq(target.data).sum()
        ...

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

train test trade-off

Overfitting

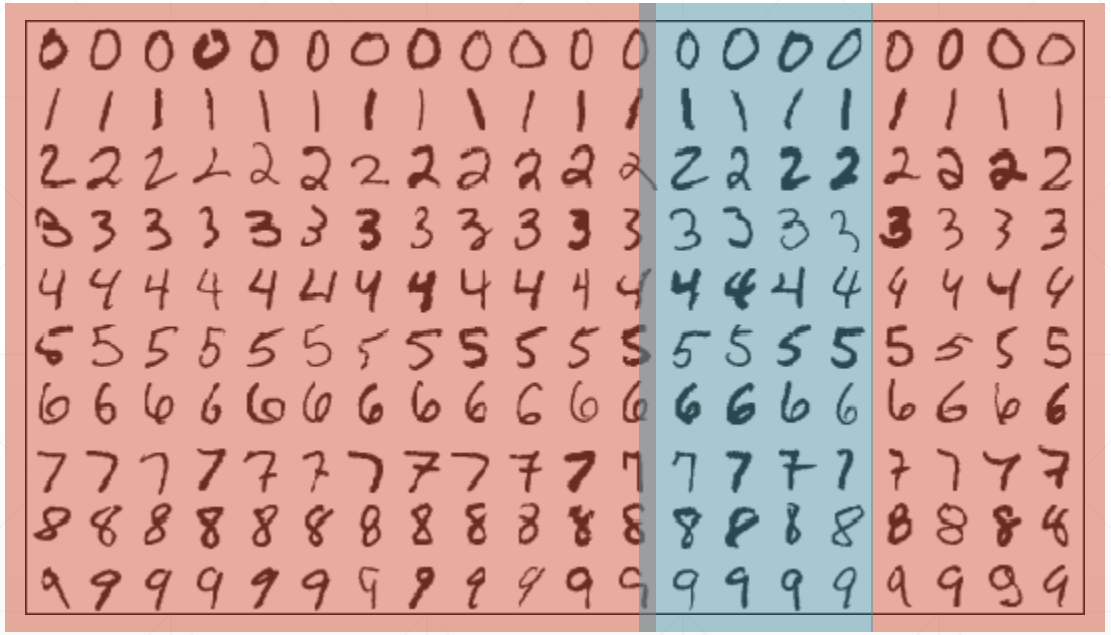


For others judge

- Kaggle

Train Set

Val Set



Test Set

Unavailable

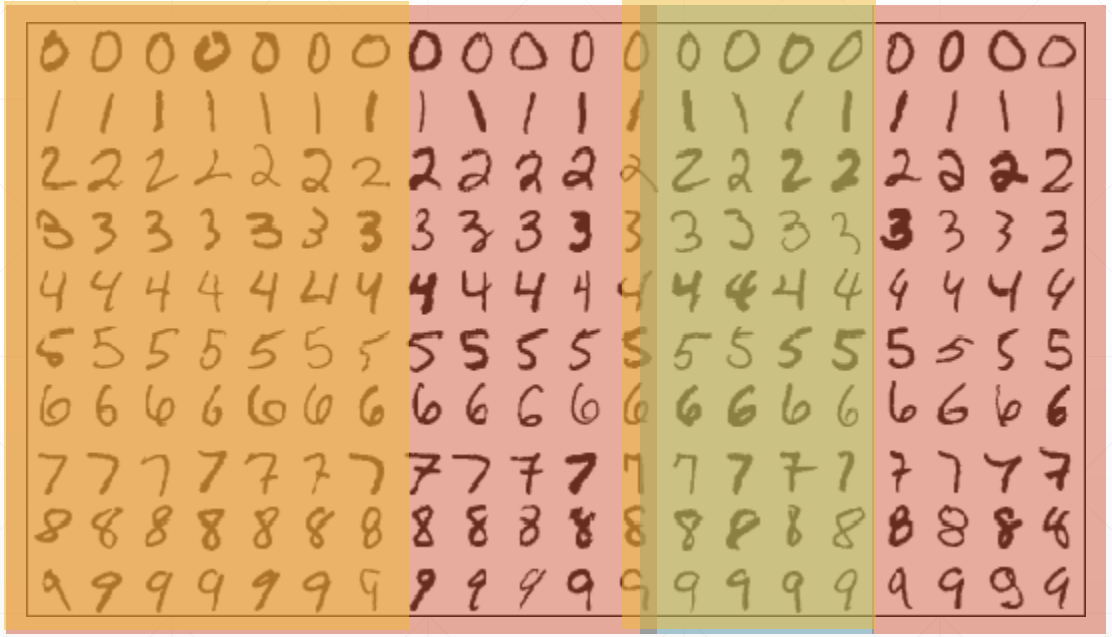
train-val-test

```
print('train:', len(train_db), 'test:', len(test_db))
train_db, val_db = torch.utils.data.random_split(train_db, [50000, 10000])
print('db1:', len(train_db), 'db2:', len(val_db))
train_loader = torch.utils.data.DataLoader(
    train_db,
    batch_size=batch_size, shuffle=True)
val_loader = torch.utils.data.DataLoader(
    val_db,
    batch_size=batch_size, shuffle=True)
```

K-fold cross-validation

Val Set

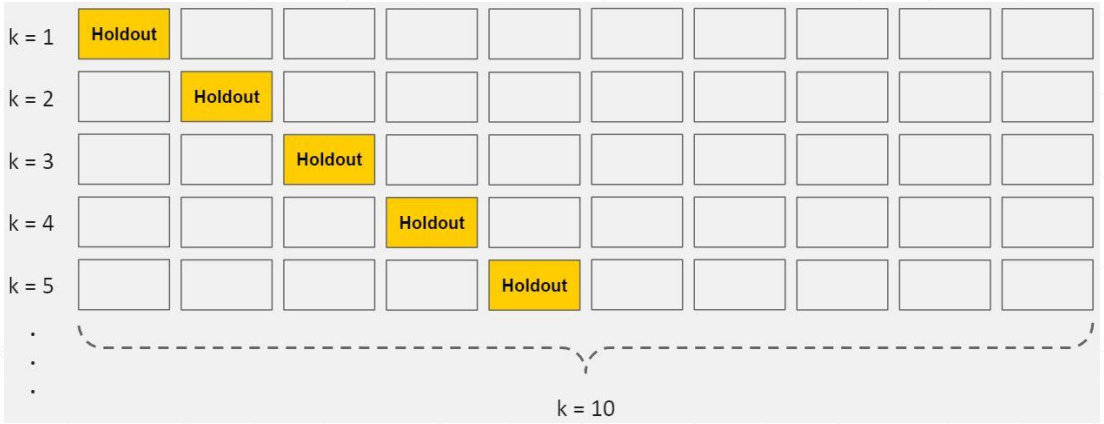
Train Set



Test Set

k-fold cross validation

- merge train/val sets
- randomly sample 1/k as val set



下一课时

减轻Overfitting

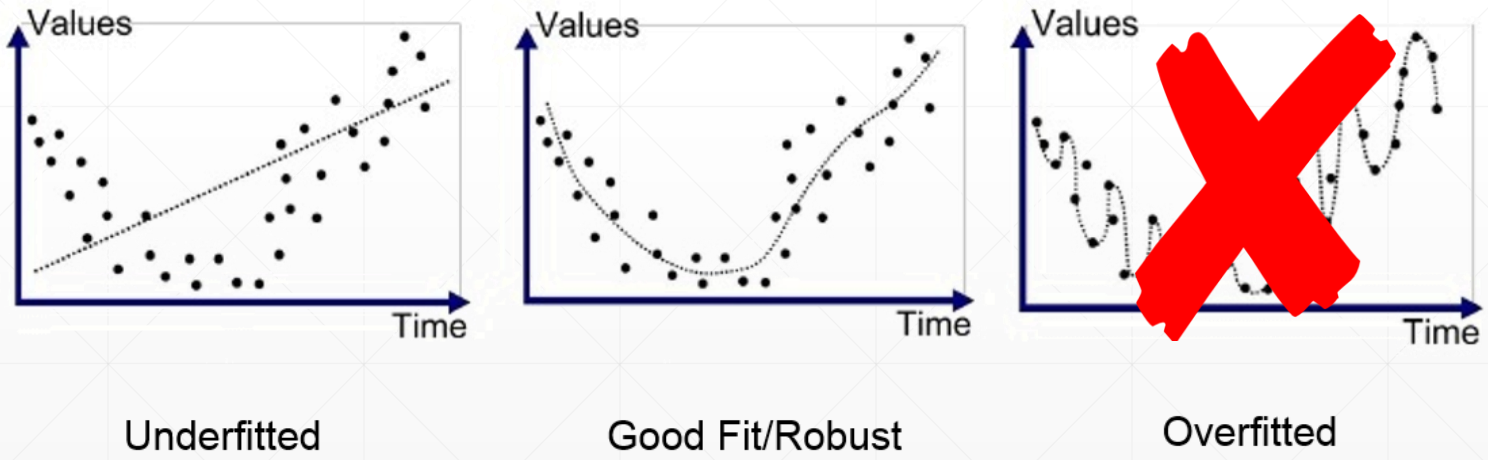
Thank You.

Regularization

主讲人：龙良曲

Occam's Razor

- *More things should not be used than are necessary.*



Underfitted | Good Fit/Robust | Overfitted

Reduce Overfitting

- More data
- Constraint model complexity
  - shallow
  - regularization
- Dropout
- Data argumentation
- Early Stopping

Regularization

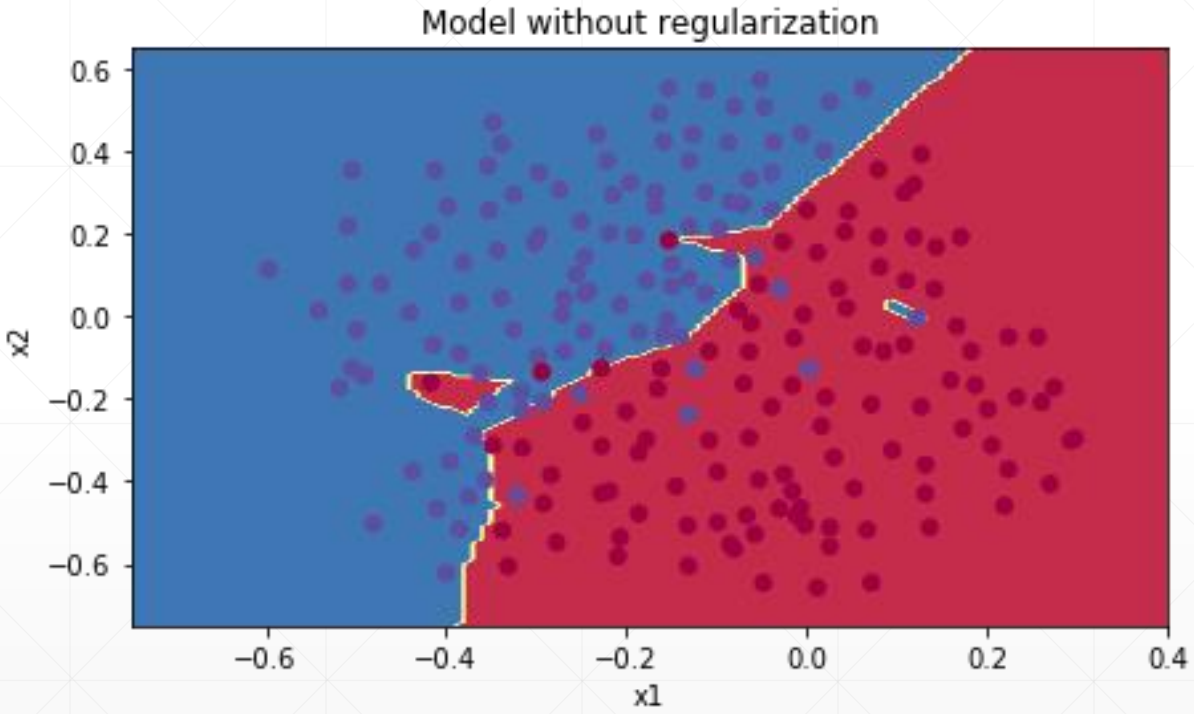
Weight Decay

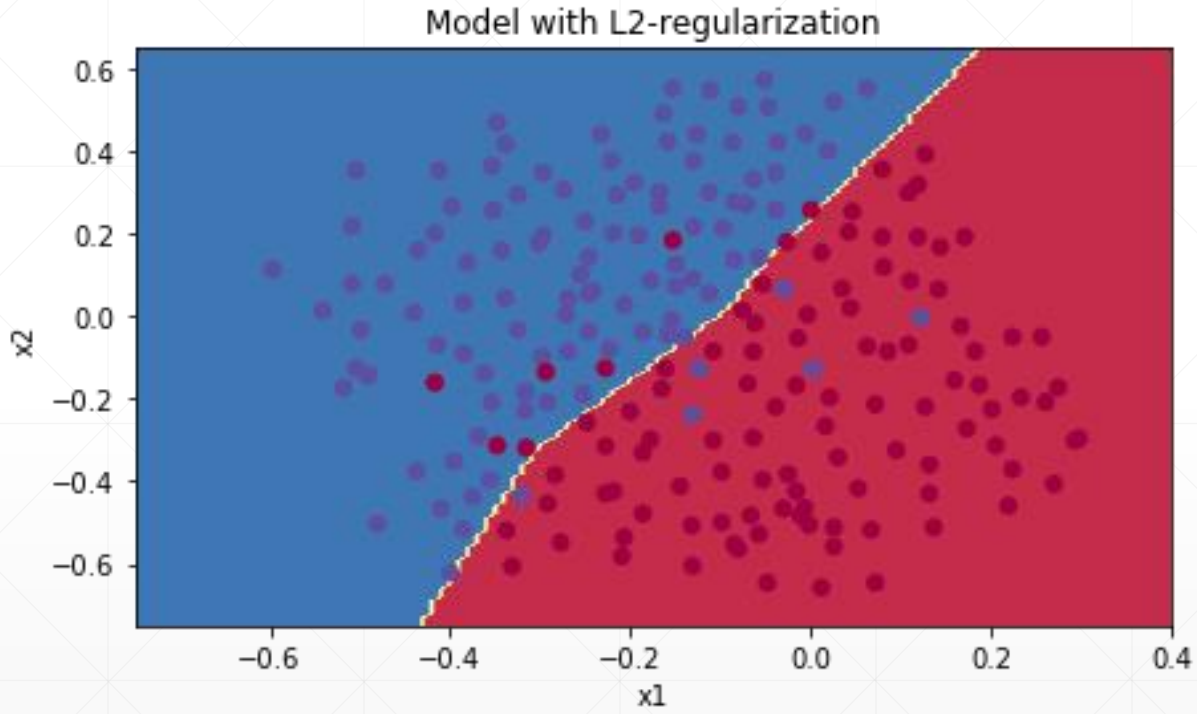
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)]$$

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x^n + \varepsilon.$$

Enforce Weights close to 0

Intuition


Model without regularization
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
x2
−0.6
−0.4
−0.2
0.0
0.2
0.4
x1


Model with L2-regularization
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
x2
−0.6
−0.4
−0.2
0.0
0.2
0.4
x1

How

- L1-regularization

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)] + \lambda \sum_{i=1}^{n}|\theta_i|$$

- L2-regularization

$$J(W;X,y) + \frac{1}{2}\lambda \cdot ||W||^2$$

L2-regularization

```
device = torch.device('cuda:0')
net = MLP().to(device)
optimizer = optim.SGD(net.parameters(), lr=learning_rate, weight_decay=0.01)
criteon = nn.CrossEntropyLoss().to(device)
```

L1-regularization

```
regularization_loss = 0
for param in model.parameters():
    regularization_loss += torch.sum(torch.abs(param))

classify_loss = criteon(logits, target)
loss = classify_loss + 0.01 * regularization_loss

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

下一课时

动量与学习率衰减

Thank You.

PyTorch

动量与学习率衰减

主讲人：龙良曲

Tricks

- momentum

- learning rate decay

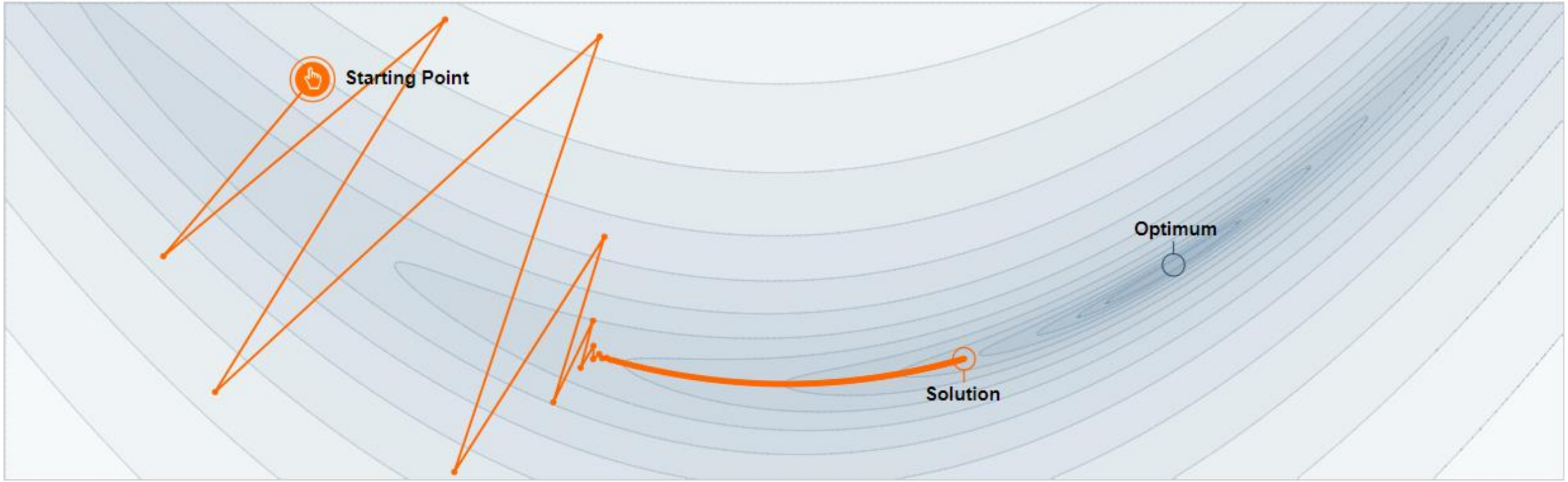
Momentum

$$w^{k+1} = w^k - \alpha \nabla f(w^k).$$

$$z^{k+1} = \beta z^k + \nabla f(w^k)$$
$$w^{k+1} = w^k - \alpha z^{k+1}$$

https://distill.pub/2017/momentum/

No momentum



Step-size α = 0.0038

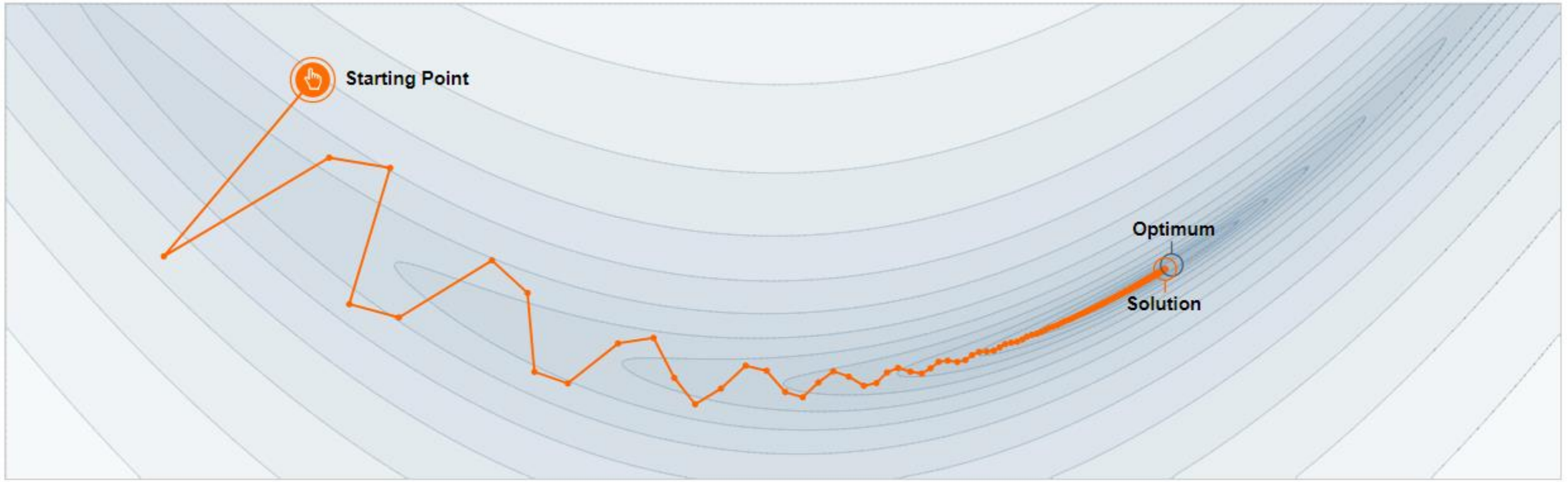
0 0.003 0.006

Momentum β = 0.0

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

With appr. momentum



Step-size α = 0.0038

0 0.003 0.006

Momentum β = 0.78

0.00 0.500 0.990

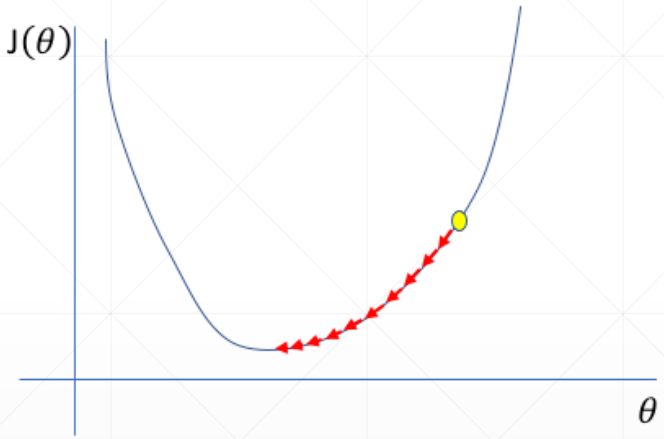
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

momentum

```
optimizer = torch.optim.SGD(model.parameters(), args.lr,
                                momentum=args.momentum,
                                weight_decay=args.weight_decay)
scheduler = ReduceLROnPlateau(optimizer, 'min')

for epoch in xrange(args.start_epoch, args.epochs):
        train(train_loader, model, criterion, optimizer, epoch)
        result_avg, loss_val = validate(val_loader, model, criterion, epoch)
        scheduler.step(loss_val)
```

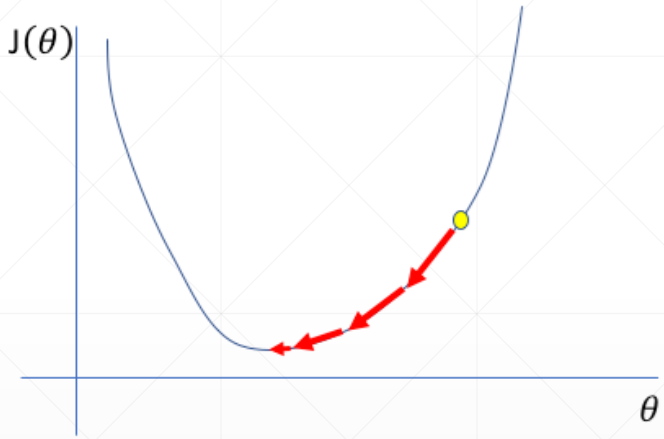
Learning rate tunning

Too low



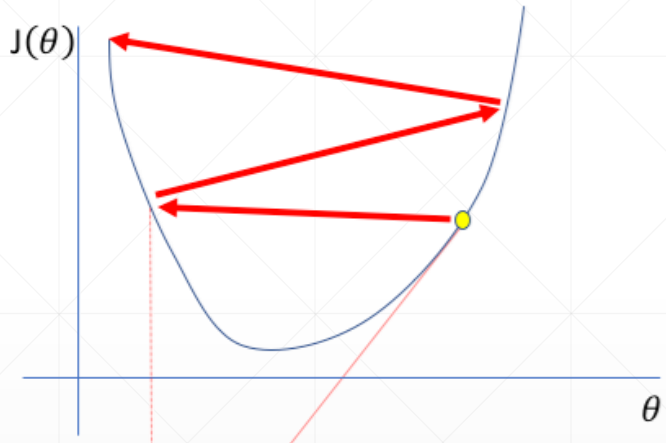
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

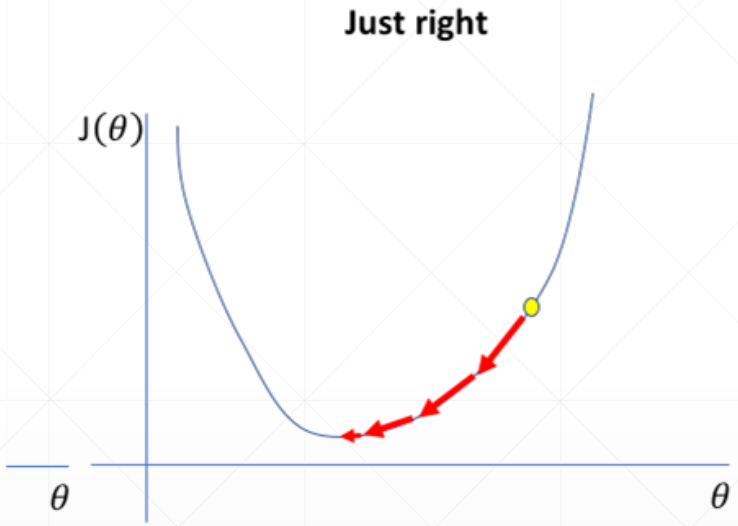
Too high



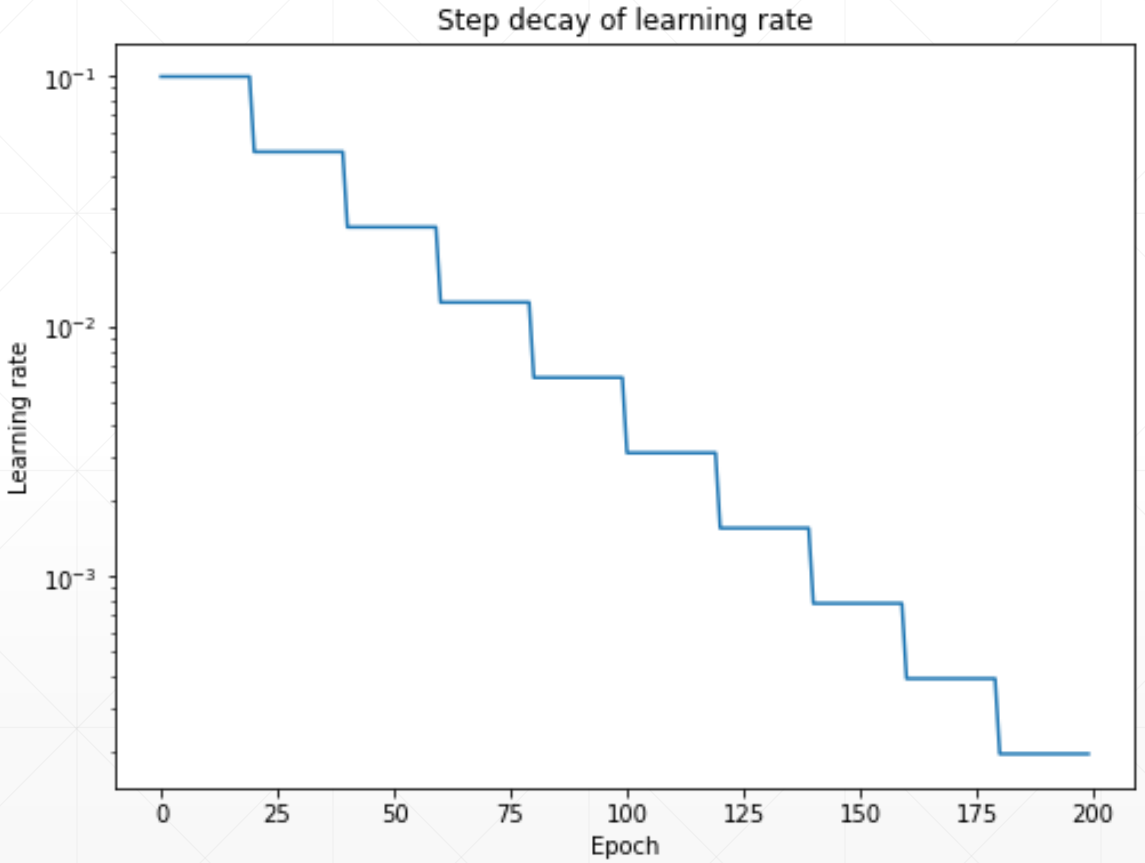
Too large of a learning rate causes drastic updates which lead to divergent behaviors

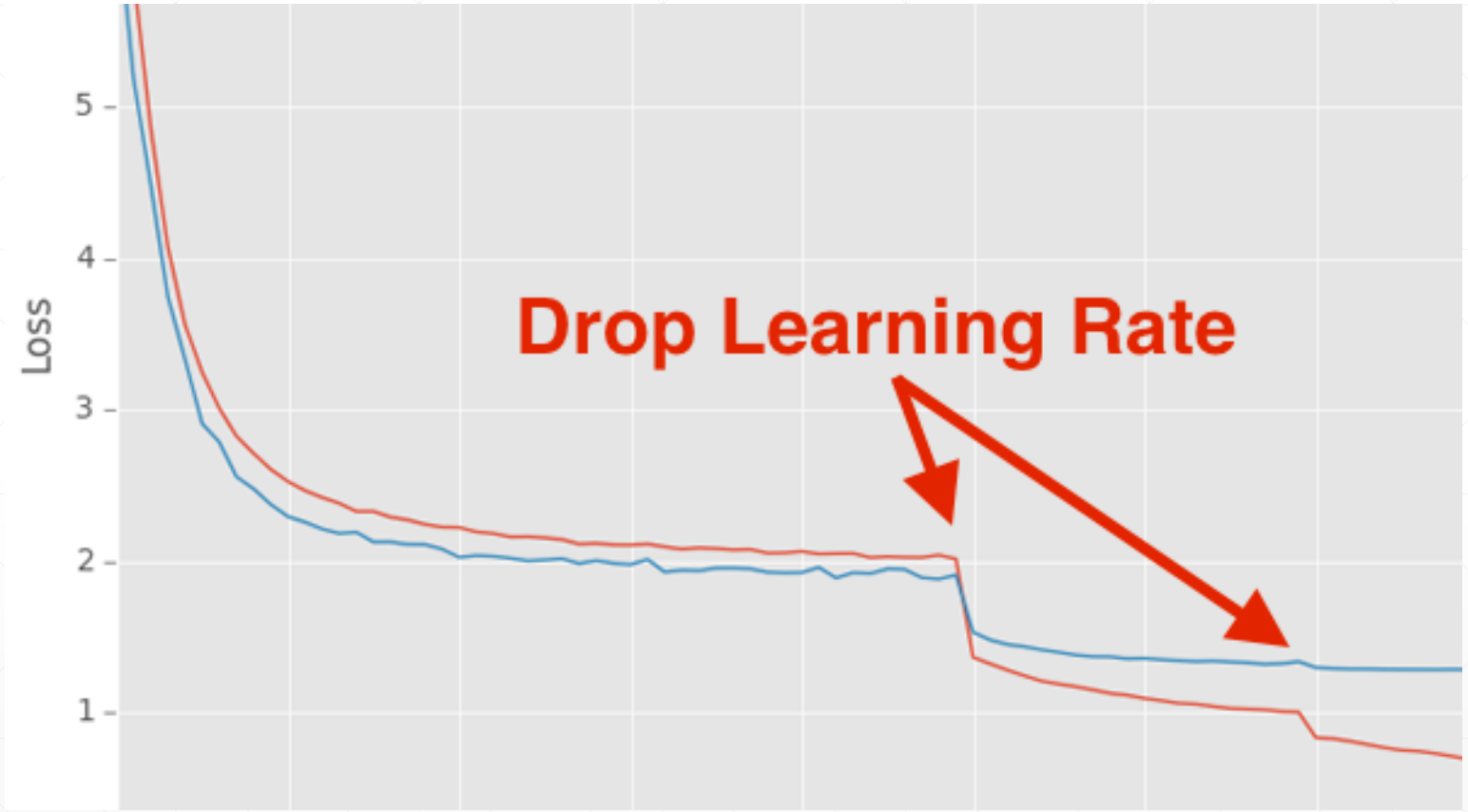

Andrej Karpathy
@karpathy
3e-4 is the best learning rate for Adam, hands down.
408 11:01 AM - Nov 24, 2016
124 people are talking about this

Learning rate decay



The optimal learning rate swiftly reaches the minimum point




Loss
5
4
3
2
1
Drop Learning Rate

Scheme 1.

CLASS **torch.optim.lr_scheduler.ReduceLROnPlateau**(*optimizer, mode='min', factor=0.1, patience=10, verbose=False, threshold=0.0001, threshold_mode='rel', cooldown=0, min_lr=0, eps=1e-08*) [SOURCE]

```
optimizer = torch.optim.SGD(model.parameters(), args.lr,
                                momentum=args.momentum,
                                weight_decay=args.weight_decay)
scheduler = ReduceLROnPlateau(optimizer, 'min')

for epoch in xrange(args.start_epoch, args.epochs):
        train(train_loader, model, criterion, optimizer, epoch)
        result_avg, loss_val = validate(val_loader, model, criterion, epoch)
        scheduler.step(loss_val)
```

Scheme 2.

```
# Assuming optimizer uses lr = 0.05 for all groups
# lr = 0.05     if epoch < 30
# lr = 0.005    if 30 <= epoch < 60
# lr = 0.0005   if 60 <= epoch < 90
# ...
scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
for epoch in range(100):
    scheduler.step()
    train(...)
    validate(...)
```

下一课时

其他Tricks

Thank You.

Early Stop,Dropout

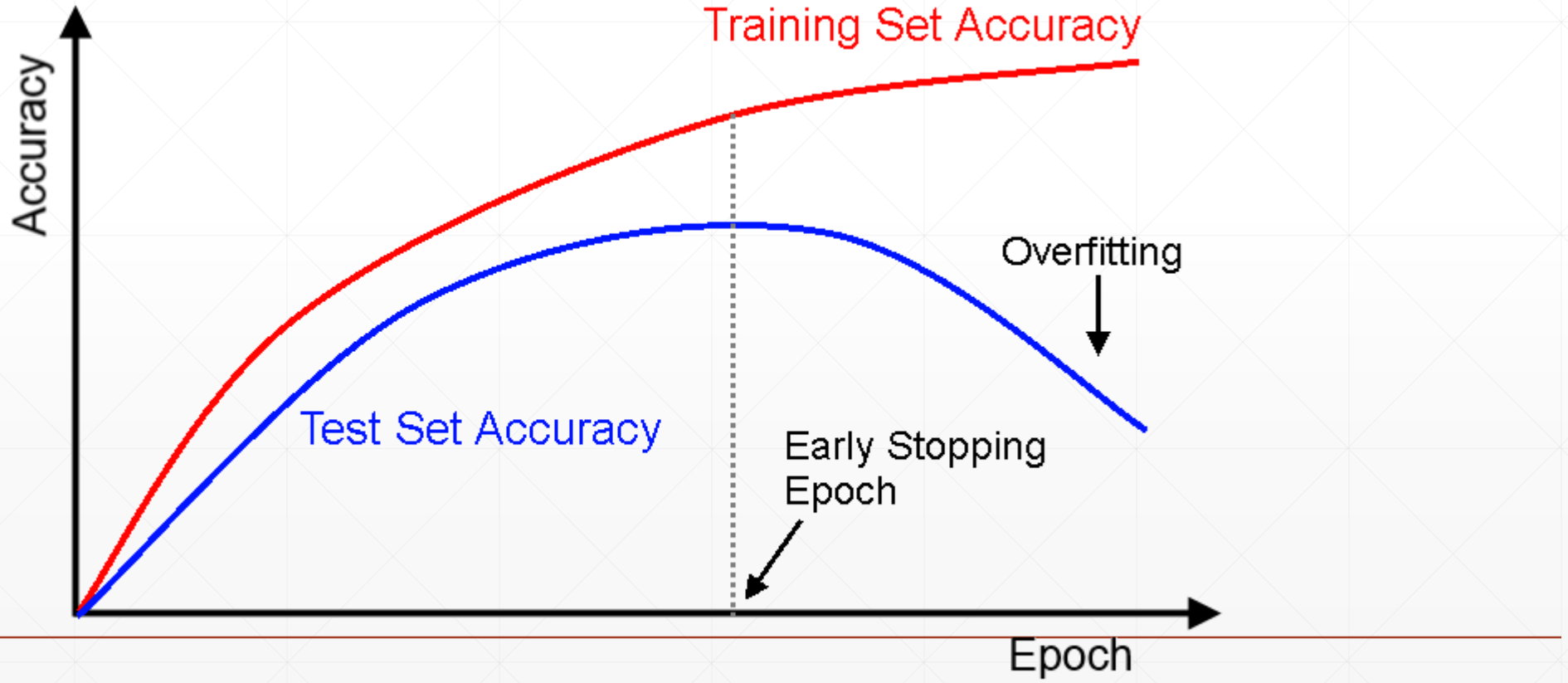
主讲人：龙良曲

Tricks

- Early Stopping
- Dropout
- Stochastic Gradient Descent

Early Stopping

- Regularization

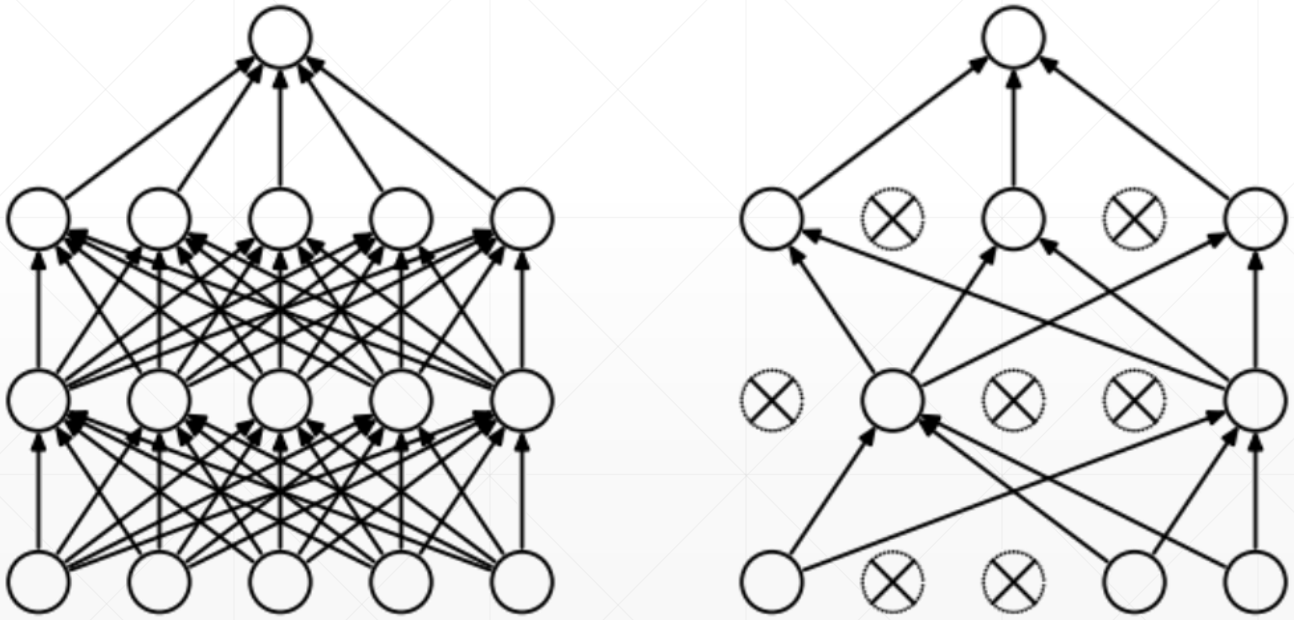


How-To

- Validation set to select parameters
- Monitor validation performance
- Stop at the highest val perf.

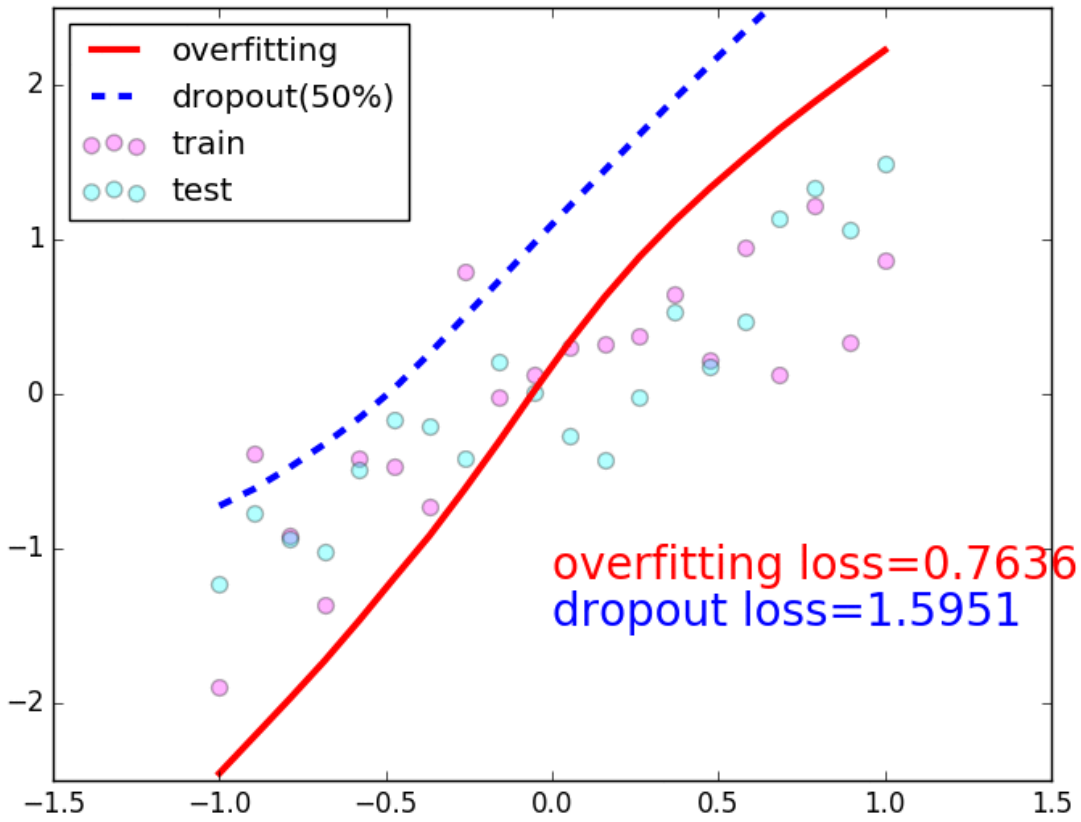
Dropout

- *Learning less to learn better*
- Each connection has $p = [0, 1]$ to lose



(a) Standard Neural Net

(b) After applying dropout.


overfitting
dropout(50%)
train
test
overfitting loss=0.7636
dropout loss=1.5951
2
1
0
−1
−2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

```
net_dropped = torch.nn.Sequential(
    torch.nn.Linear(784, 200),
    torch.nn.Dropout(0.5),  # drop 50% of the neuron
    torch.nn.ReLU(),
    torch.nn.Linear(200, 200),
    torch.nn.Dropout(0.5),  # drop 50% of the neuron
    torch.nn.ReLU(),
    torch.nn.Linear(200, 10),
)
```

Clarification

- *torch.nn.Dropout(p=dropout_prob)*
- *tf.nn.dropout(keep_prob)*

Behavior between train and test

Batch-Norm

```
for epoch in range(epochs):

    # train
    net_dropped.train()
    for batch_idx, (data, target) in enumerate(train_loader):
      ...
    net_dropped.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
      ...
```

Stochastic Gradient Descent

- Stochastic
  - not random!
- Deterministic

Gradient Descent

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^i - y^i) \cdot x_j^i$$

② Vanilla (Batch) G.D.

$$\theta_j := \theta_j - \alpha \cdot \underbrace{\frac{\partial}{\partial \theta_j} J(\theta)}_{\frac{1}{m} \sum_{i=1}^{m} (\hat{y}^i - y^i) x_j^i}$$

https://towardsdatascience.com/difference-between-batch-gradient-descent-and-stochastic-gradient-descent-1187f1291aa1

Gradient Descent

③ Stochastic G.D.

for i in range(m):

$$\theta_j := \theta_j - \alpha \cdot \underbrace{(\hat{y}^i - y^i) x_j^i}_{\text{only one example}} \frac{\nabla J}{\nabla \theta_j}$$

https://towardsdatascience.com/difference-between-batch-gradient-descent-and-stochastic-gradient-descent-1187f1291aa1

Stochastic Gradient Descent

- Not single usually
- batch = 16, 32, 64, 128…

Why

¥1789.00

索泰（ZOTAC）GeForce GTX1060 X-GAMING OC吃鸡显卡/游戏电竞台式机独

6.6万+条评价　二手有售

索泰京东自营旗舰店

自营 秒杀 券1500-200

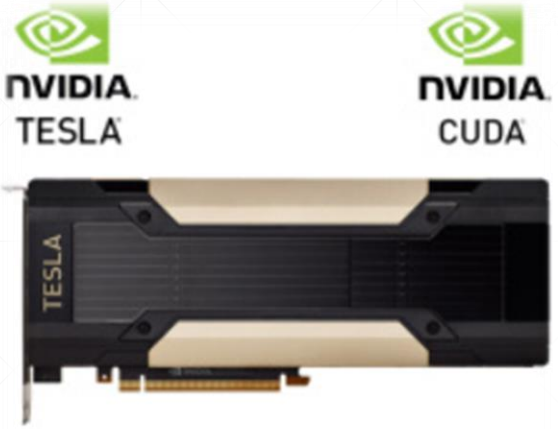
¥7599.00

索泰（ZOTAC） GTX1080Ti-11GD5X至尊PLUS OC绝地求生/吃鸡显卡

40+条评价

云轩 DIY整机专营店

满减 免邮 满199-100 险



¥80999.00

NVIDIA TESLA V100 16GB/32GB HBM2/人工智能深度学习GPU运算 16GB HBM2

0条评价

子希DIY电脑专营店

免邮

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)},$$

下一课时

贝叶斯定理

Thank You.