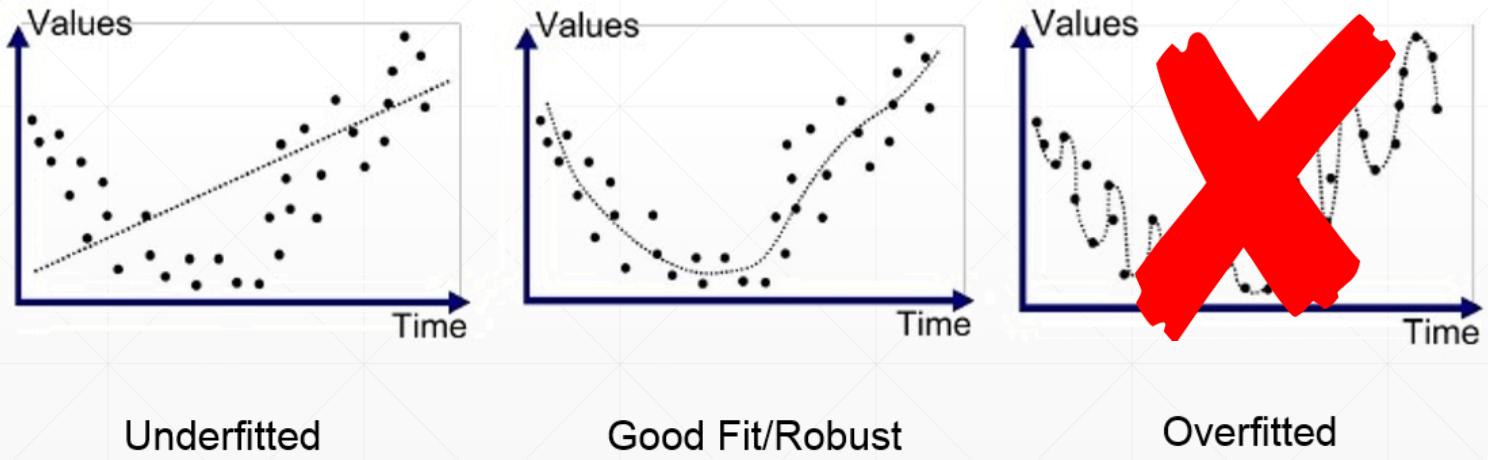

# Regularization

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

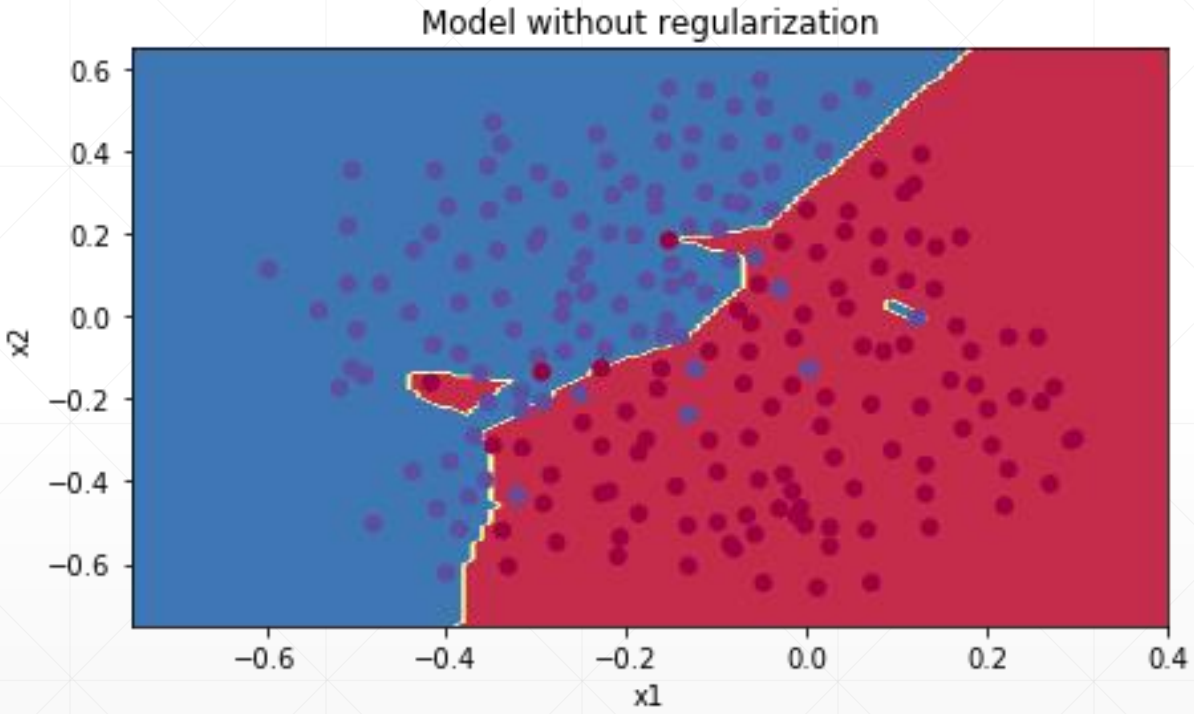
Weight Decay

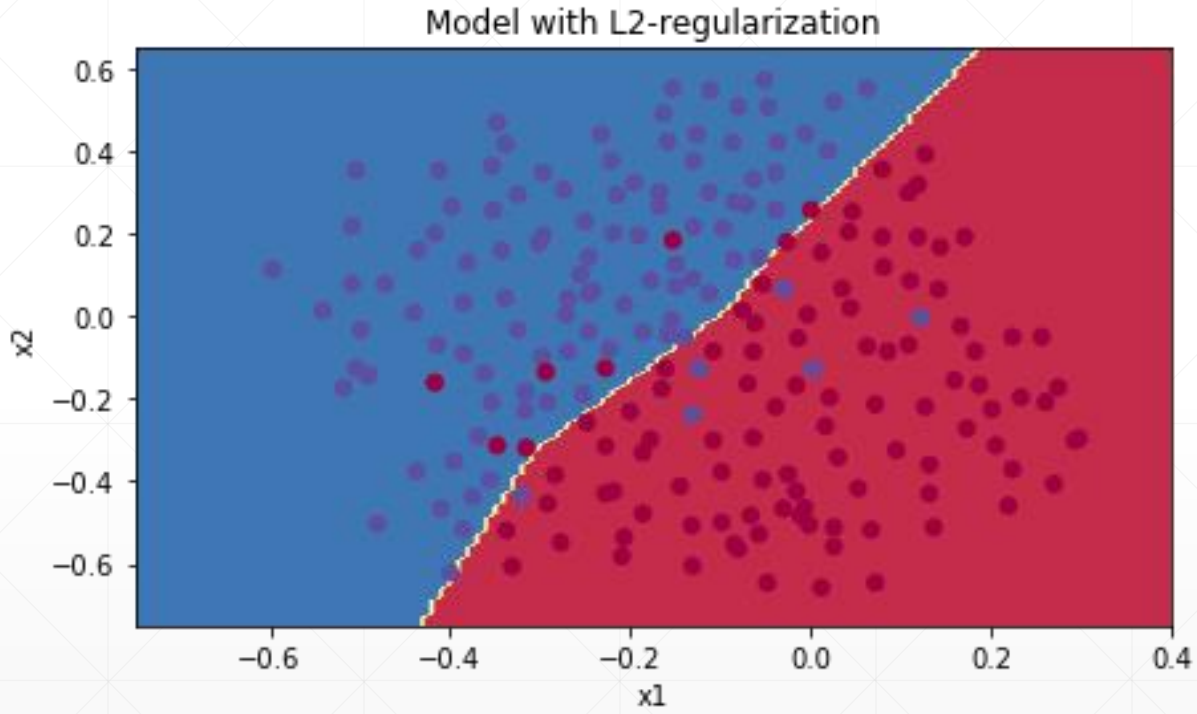
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i)]$$

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

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

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

lambda

# 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.