

《人工智能》实验报告

学号: SA18225034
姓名: 陈建虎
班级: 18 级大数据 2 班

一、对于给定的模型，比如二阶模型 $y=w_1*x^2 + w_2*x + b$ ，尝试找出该 model 中的最佳 function。

1. 验证数据归一化技术的效果。采用 BGD。

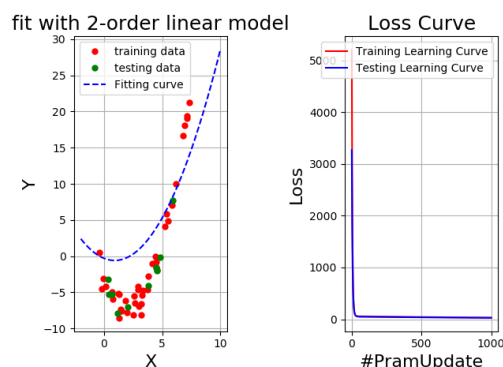
超参数设置如下：

training_epochs	Learning Rate	N	TestingDataRatio	OrderNum	Seed	BatchSize
int(1000)	1*1e-4	50	0.2	2	16225151	$N*(1-TestingDataRatio)$

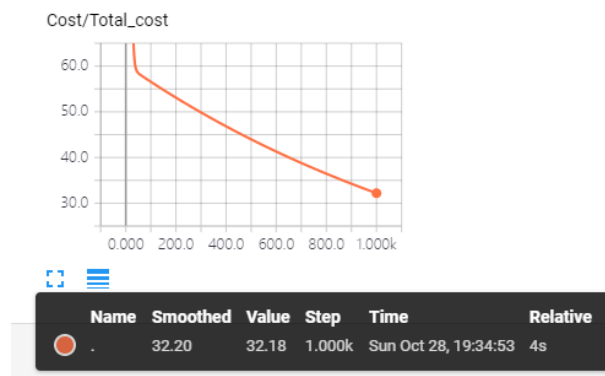
a) 设置 `DataScaleFlag=0` 时：

运行结果：

```
Initial w,b: [array([[ 3.15309572],  
 [ 1.28588533]], dtype=float32), array([ 0.64348024], dtype=float32)]  
TrainingDataNum|TestingDataNum= 40.0 | 10.0  
fit with 2-order linear model...  
prev_cost = 5201.47 Initial_test_cost = 3268.09  
最佳参数 W,b: [[ 0.35269144]  
 [-0.64401829]] [-0.29999003]  
4.078130722045898 秒  
Training...Cost= 32.1841  
Testing...Cost= 28.4143
```



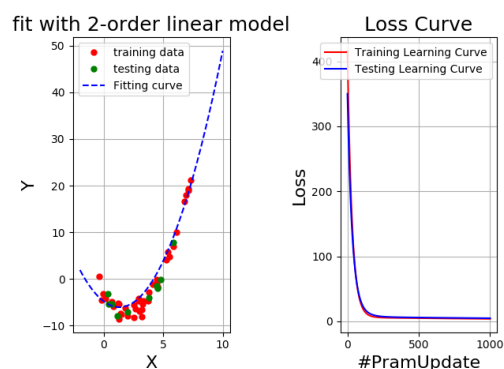
tensorboard: (train_set)



b) 设置 `DataScaleFlag=1` 时:

运行结果:

```
Initial w,b: [array([[ 3.15309572],
 [ 1.28588533]], dtype=float32), array([ 0.64348024], dtype=float32)]
TrainingDataNum|TestingDataNum= 40.0 | 10.0
fit with 2-order linear model...
prev_cost = 416.062 Initial_test_cost = 349.951
最佳参数 W,b: [[ 0.72680497]
 [-1.90241973]] [-4.78916645]
3.3763463497161865 秒
Training...Cost= 3.83901
Testing...Cost= 4.78157
```



tensorboard: (train_set)



总结:

通过观察上面的实验结果，明显能感觉到使用了数据归一化技术时 `Total_cost` 的收敛速

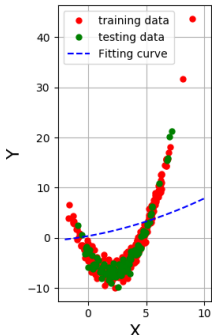
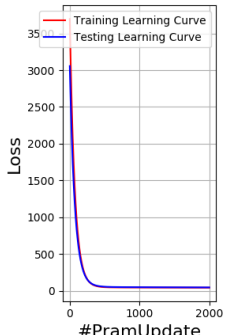
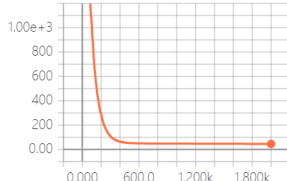
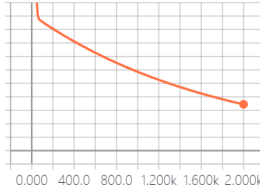
度明显比没有使用此技术时快。大约在第 300 轮左右时，使用数据归一化技术的模型 Total_cost 基本收敛，而未使用此技术的模型到最后（epochs=1000）也没能收敛。

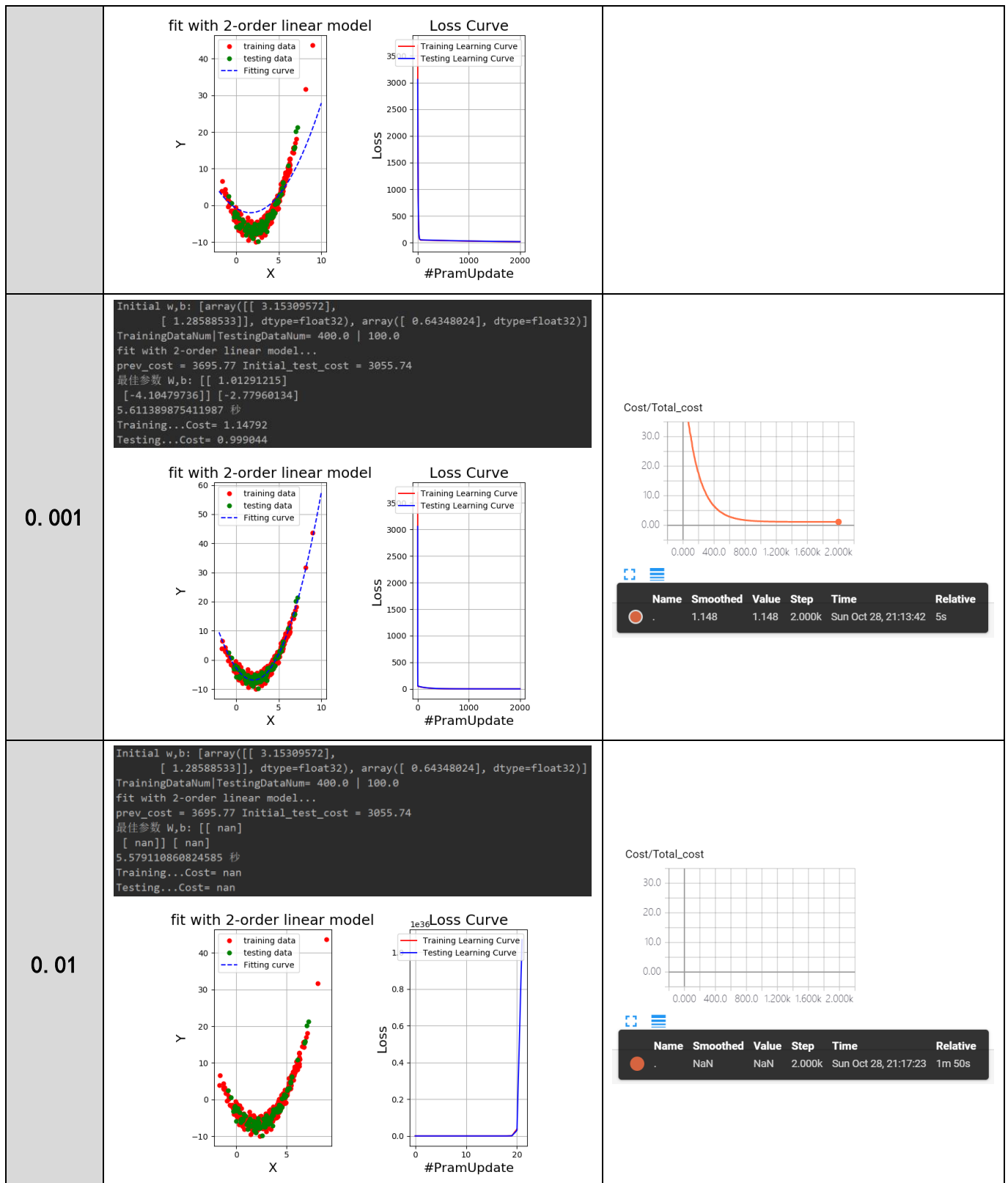
2. 学习率对收敛速度的影响。

2.1) 采用 BGD。超参数设置如下：

training_epochs	BatchSize	N	TestingDataRatio	OrderNum	Seed	DataScaleFlag
int(2000)	$N \times (1 - \text{TestingDataRatio})$	500	0.2	2	16225151	0

不同 learning rate 运行结果如下

Learn ngrate	运行结果	tensorboard (train_set)												
0.0000 1	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[0.033882] [0.41199756]] [0.35269091] 6.096155881881714 秒 Training...Cost= 45.1929 Testing...Cost= 48.7977</pre></div> <div><div>fit with 2-order linear model</div><div>Loss Curve</div></div>	<div><div>Cost/Total_cost</div><div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>45.20</td><td>45.19</td><td>2.000k</td><td>Sun Oct 28, 20:59:05</td><td>6s</td></tr></tbody></table></div></div>	Name	Smoothed	Value	Step	Time	Relative	.	45.20	45.19	2.000k	Sun Oct 28, 20:59:05	6s
Name	Smoothed	Value	Step	Time	Relative									
.	45.20	45.19	2.000k	Sun Oct 28, 20:59:05	6s									
0.0001	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[0.43169847] [-1.44959557]] [-0.79387891] 5.571177959442139 秒 Training...Cost= 17.1771 Testing...Cost= 18.9705</pre></div>	<div><div>Cost/Total_cost</div><div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>17.21</td><td>17.18</td><td>2.000k</td><td>Sun Oct 28, 21:07:12</td><td>5s</td></tr></tbody></table></div></div>	Name	Smoothed	Value	Step	Time	Relative	.	17.21	17.18	2.000k	Sun Oct 28, 21:07:12	5s
Name	Smoothed	Value	Step	Time	Relative									
.	17.21	17.18	2.000k	Sun Oct 28, 21:07:12	5s									



总结:

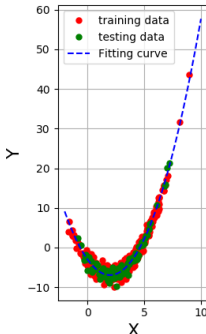
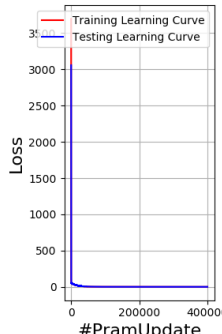
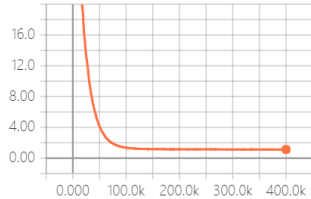
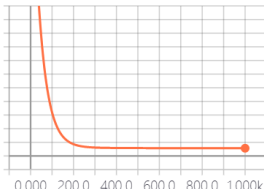
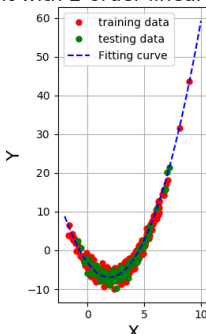
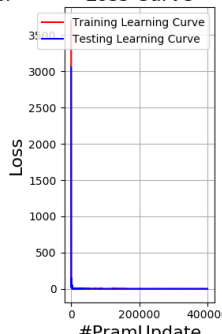
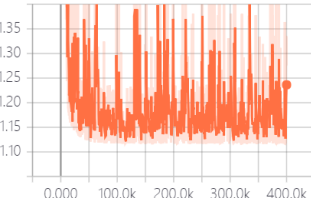
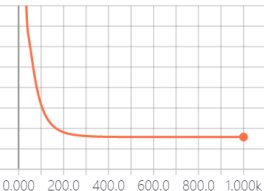
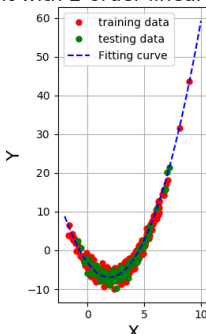
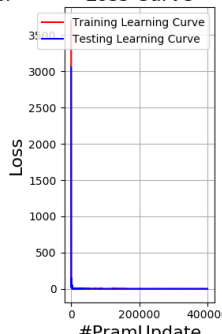
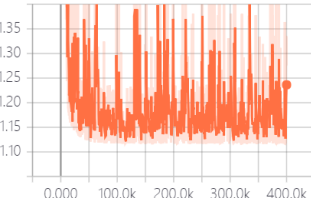
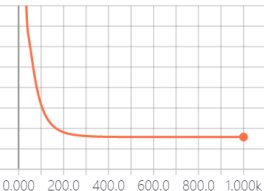
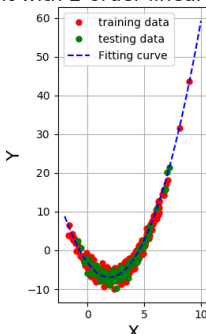
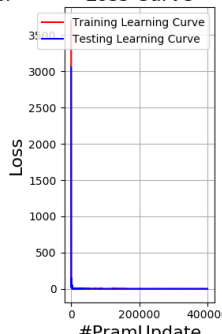
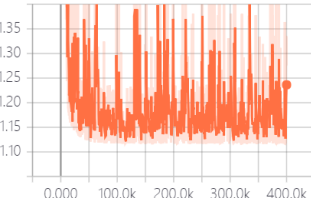
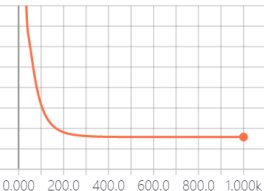
通过观察上面的实验结果发现，当 lr 太小时 (1×10^{-5})， $Total_cost$ 收敛得极慢，2000 轮训练完成时 $cost$ 依然很高，这时应该适当增大 lr (从 1×10^{-5} 到 1×10^{-4} 再到 1×10^{-3})，适

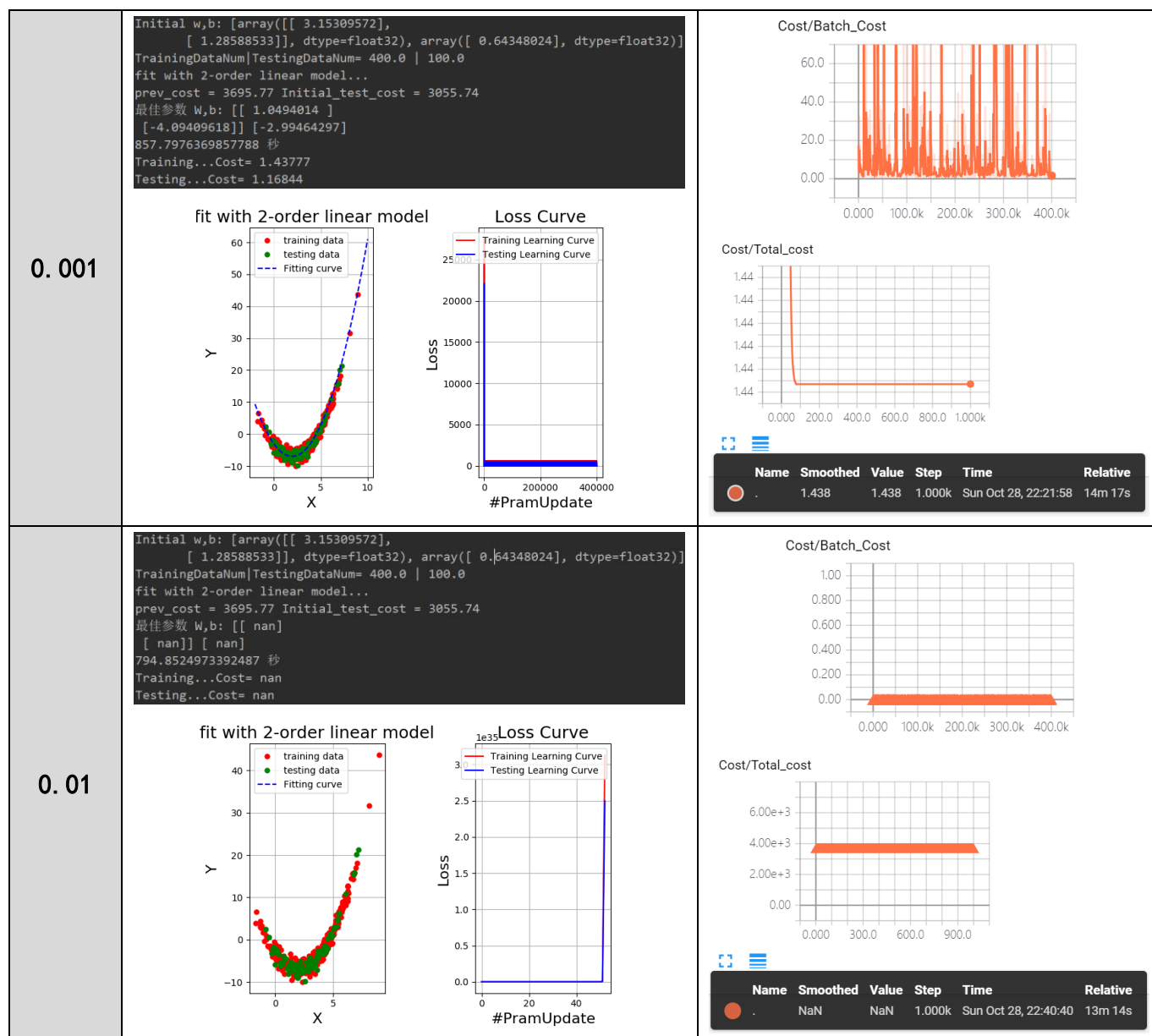
当增大后发现 Total_cost 收敛速度增加, 当 $lr=0.001$ 时, cost 最小, 同时收敛速度也最快。当尝试再增大 lr 到 0.01 时, 发现 cost 直接跨过最小值, 朝着无限大的方向去了。出现此原因就是 lr 太大, 即更新参数的步长太大。

2.2) 采用 **SGD**。超参数设置如下:

training_epochs	BatchSize	N	TestingDataRatio	OrderNum	Seed	DataScaleFlag
int(1000)	1	500	0.2	2	16225151	0

不同 **learning rate** 运行结果如下

Learn nrate	运行结果:	tensorboard (train_set)																									
0.0000 1	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.0088501] [-4.02007437]] [-2.97418761] 766.1627781391144 秒 Training...Cost= 1.12981 Testing...Cost= 0.987424</pre></div> <div><div>fit with 2-order linear model</div></div> <div><div>Loss Curve</div></div> <div><div>Cost/Batch_Cost</div></div> <div><div>Cost/Total_cost</div></div> <table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.130</td><td>1.130</td><td>1.000k</td><td>Sun Oct 28, 21:46:27</td><td>12m 46s</td></tr></tbody></table> <tr><td>0.0001</td><td><div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.01881862] [-3.94540596]] [-3.14303136] 770.622985124588 秒 Training...Cost= 1.2758 Testing...Cost= 1.09446</pre></div><div><div>fit with 2-order linear model</div></div><div><div>Loss Curve</div></div><div><div>Cost/Batch_Cost</div></div><div><div>Cost/Total_cost</div></div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.276</td><td>1.276</td><td>1.000k</td><td>Sun Oct 28, 22:06:30</td><td>12m 50s</td></tr></tbody></table></td></tr>	Name	Smoothed	Value	Step	Time	Relative	.	1.130	1.130	1.000k	Sun Oct 28, 21:46:27	12m 46s	0.0001	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.01881862] [-3.94540596]] [-3.14303136] 770.622985124588 秒 Training...Cost= 1.2758 Testing...Cost= 1.09446</pre></div> <div><div>fit with 2-order linear model</div></div> <div><div>Loss Curve</div></div> <div><div>Cost/Batch_Cost</div></div> <div><div>Cost/Total_cost</div></div> <table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.276</td><td>1.276</td><td>1.000k</td><td>Sun Oct 28, 22:06:30</td><td>12m 50s</td></tr></tbody></table>	Name	Smoothed	Value	Step	Time	Relative	.	1.276	1.276	1.000k	Sun Oct 28, 22:06:30	12m 50s
Name	Smoothed	Value	Step	Time	Relative																						
.	1.130	1.130	1.000k	Sun Oct 28, 21:46:27	12m 46s																						
0.0001	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.01881862] [-3.94540596]] [-3.14303136] 770.622985124588 秒 Training...Cost= 1.2758 Testing...Cost= 1.09446</pre></div> <div><div>fit with 2-order linear model</div></div> <div><div>Loss Curve</div></div> <div><div>Cost/Batch_Cost</div></div> <div><div>Cost/Total_cost</div></div> <table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.276</td><td>1.276</td><td>1.000k</td><td>Sun Oct 28, 22:06:30</td><td>12m 50s</td></tr></tbody></table>	Name	Smoothed	Value	Step	Time	Relative	.	1.276	1.276	1.000k	Sun Oct 28, 22:06:30	12m 50s														
Name	Smoothed	Value	Step	Time	Relative																						
.	1.276	1.276	1.000k	Sun Oct 28, 22:06:30	12m 50s																						



总结:

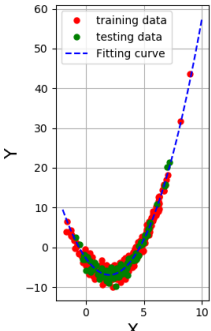
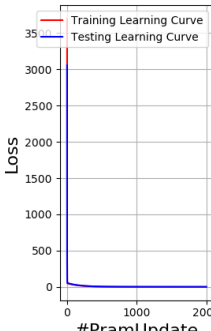
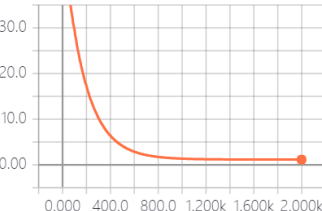
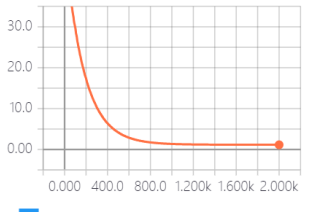
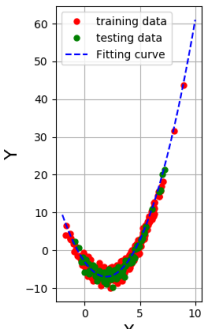
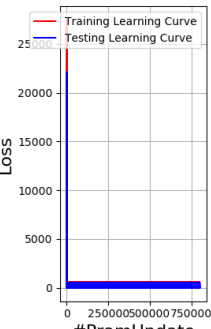
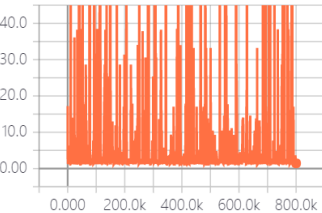
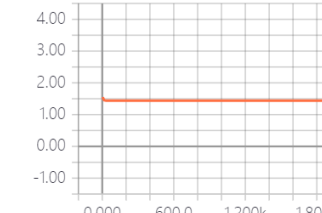
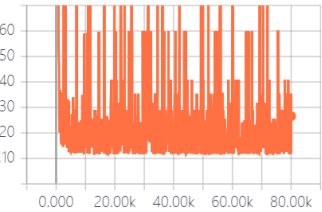
通过观察上面的实验结果发现,当使用随机梯度下降算法时,lr 对 Total_cost 收敛速度影响与使用 BGD 时基本一致,即随着 lr 的逐步增大,Total_cost 收敛速度越来越快,但当超过某一阈值(0.01)时,情况会变得非常糟。

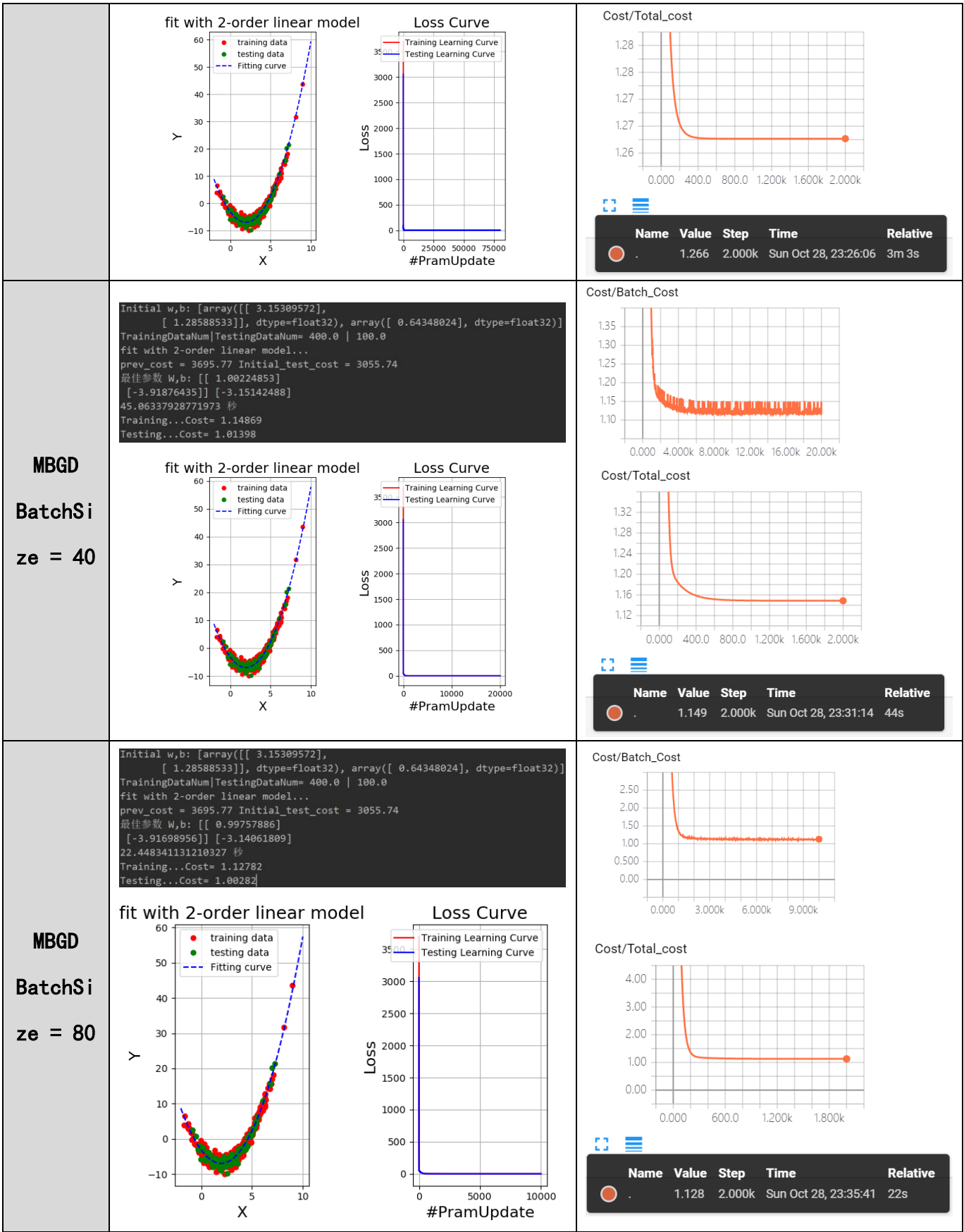
除此之外,还能观察到 Batch_cost 会出现大幅抖动情况,主要原因就是随机梯度下降算法每次更新参数时,只使用一个实例。

3. 对比 BGD, SGD 和 MBGD, 并进一步观察分析 MBGD 中 BatchSize 对收敛稳定性和收敛速度的影响。使用基本的 GD 方法, 超参数设置如下:

training_epochs	Learning Rate	N	TestingDataRatio	OrderNum	Seed	DataScaleFlag
int(2000)	1*1e-3	500	0.2	2	16225151	0

不同 **GD 方法** 运行结果如下

GD 方法	运行结果	tensorboard (train_set)												
BGD	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.01291215] [-4.10479736]] [-2.77960134] 5.845089912414551 秒 Training...Cost= 1.14792 Testing...Cost= 0.999044</pre></div> <div><div>fit with 2-order linear model</div><div>Loss Curve</div></div>	<div><div>Cost/Batch_Cost</div><div>Cost/Total_cost</div><div><div></div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.148</td><td>1.148</td><td>2.000k</td><td>Sun Oct 28, 22:50:16</td><td>5s</td></tr></tbody></table></div></div>	Name	Smoothed	Value	Step	Time	Relative	.	1.148	1.148	2.000k	Sun Oct 28, 22:50:16	5s
Name	Smoothed	Value	Step	Time	Relative									
.	1.148	1.148	2.000k	Sun Oct 28, 22:50:16	5s									
SGD	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.0494014] [-4.09409618]] [-2.99464297] 1628.9480073451996 秒 Training...Cost= 1.43777 Testing...Cost= 1.16844</pre></div> <div><div>fit with 2-order linear model</div><div>Loss Curve</div></div>	<div><div>Cost/Batch_Cost</div><div>Cost/Total_cost</div><div><div></div><table><thead><tr><th>Name</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.438</td><td>2.000k</td><td>Sun Oct 28, 23:20:23</td><td>27m 3s</td></tr></tbody></table></div></div>	Name	Value	Step	Time	Relative	.	1.438	2.000k	Sun Oct 28, 23:20:23	27m 3s		
Name	Value	Step	Time	Relative										
.	1.438	2.000k	Sun Oct 28, 23:20:23	27m 3s										
MBGD BatchSize = 10	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[1.02087653] [-3.96244621]] [-3.13220763] 183.8192036151886 秒 Training...Cost= 1.26614 Testing...Cost= 1.08141</pre></div>	<div><div>Cost/Batch_Cost</div></div>												



通过观察发现如下现象：

- 1) Batchsize=80 时，Total Loss 收敛最快。
- 2) BGD 下，Total Loss 稳定的下降，不断朝极小值接近，但收敛速度慢。
- 3) SGD 和 Batchsize=10 这两种情况下，很早就出现了 Total Loss 值不再下降的现象，但实际上此时并没有找到 Total Loss 的极小值。
- 4) MBGD 下，随着 Batchsize 的取值不同，Batch_Cost 出现了不同程度的震荡。

问题

Q1：现象 3）中，为何会出现 Total Loss 值不再更新的现象呢？

答：因为当 SGD 和 Batchsize=10 这两种情况时，参数更新太过频繁，很可能会找到最值后因为某些噪声实例，导致 Total_cost 又跳出最小值。

Q2：现象 4）中，为何会出现震荡？

答：当 batchsize=1 时，MBGD 变为 SGD，此时更新一次参数只随机从训练集中抽取一个实例，所以 Batch_Cost 抖动程度最大；当 batchsize=10 时，通过 10 个实例的 loss 求平均的方法来更新参数，此时会比 batchsize=1 稍微稳定一点点，但是还是存在较大抖动，以此类推，当 batchsize 逐渐增大到 40 和 80 时，Batch_Cost 抖动越来越小，但 batchsize 过大时也不好，因为这样会导致更新参数的速度变得很慢，即导致 cost 收敛很慢。

Q3：你觉得以上 5 种不同的 BatchSize 中，哪种取值最好？为什么？

答：我会选择 BatchSize=40。因为此时 Batch_Cost 震荡幅度不会太大，Total_cost 的收敛速度也比较可观，即能较好地平衡 Batch_Cost 抖动和 Total_cost 的收敛速度。

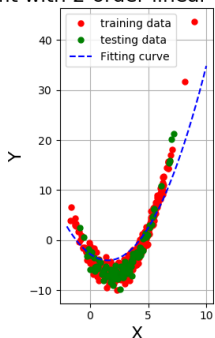
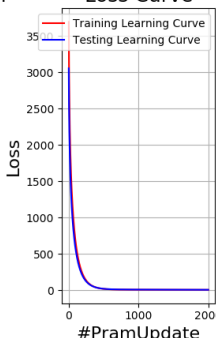
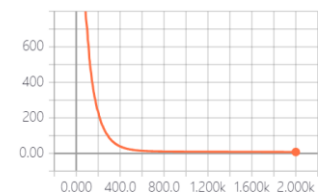
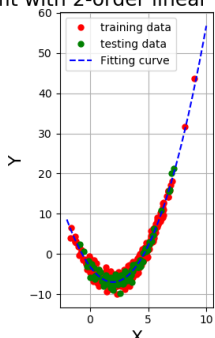
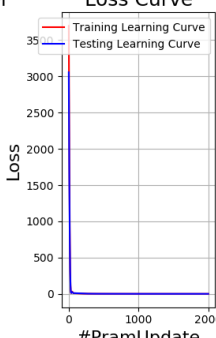
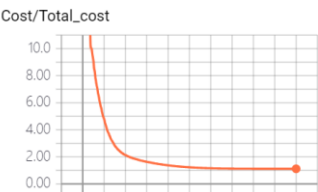
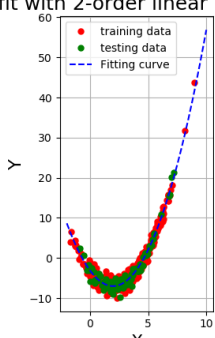
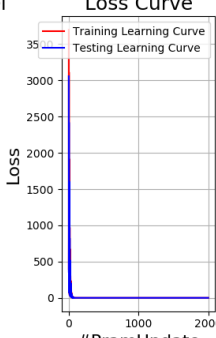
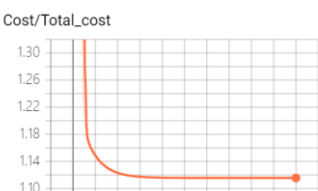
4. 验证常用梯度优化算法的效果。

超参数设置如下：

training_epochs	Learning Rate	BatchSize	N	TestingDataRatio	OrderNum	Seed	DataScaleFlag
int(2000)	1e-3	$N \times (1 - \text{TestingDataRatio})$	500	0.2	2	16225151	0

不同**优化方法**运行结果如下

优化方法	运行结果：	tensorboard (train_set)
------	-------	-------------------------

<div>AdaGrad</div> <div>Lr=0.1</div>	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[0.54280442] [-1.66816723]] [-2.76331615] 7.419820547103882 秒 Training...Cost= 7.96502 Testing...Cost= 8.63265</pre></div> <div><div>fit with 2-order linear model</div></div> <div><div>Loss Curve</div></div>	<div><div>Cost/Total_cost</div></div> <div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>7.971</td><td>7.965</td><td>2.000k</td><td>Mon Oct 29, 14:22:46</td><td>7s</td></tr></tbody></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	7.971	7.965	2.000k	Mon Oct 29, 14:22:46	7s
Name	Smoothed	Value	Step	Time	Relative									
.	7.971	7.965	2.000k	Mon Oct 29, 14:22:46	7s									
<div>Adam</div> <div>Lr=0.1</div>	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[0.98938459] [-3.8976903]] [-3.17854118] 5.919700860977173 秒 Training...Cost= 1.11573 Testing...Cost= 1.00037</pre></div> <div><div>fit with 2-order linear model</div></div> <div><div>Loss Curve</div></div>	<div><div>Cost/Total_cost</div></div> <div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.116</td><td>1.116</td><td>2.000k</td><td>Mon Oct 29, 14:28:07</td><td>5s</td></tr></tbody></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	1.116	1.116	2.000k	Mon Oct 29, 14:28:07	5s
Name	Smoothed	Value	Step	Time	Relative									
.	1.116	1.116	2.000k	Mon Oct 29, 14:28:07	5s									
<div>Moment</div> <div>Lr=1e-3</div> <div>momentum</div> <div>= 0.9</div>	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... prev_cost = 3695.77 Initial_test_cost = 3055.74 最佳参数 W,b: [[0.99170125] [-3.91454053]] [-3.15608668] 6.341227293014526 秒 Training...Cost= 1.11558 Testing...Cost= 0.996771</pre></div> <div><div>fit with 2-order linear model</div></div> <div><div>Loss Curve</div></div>	<div><div>Cost/Total_cost</div></div> <div><table><thead><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr></thead><tbody><tr><td>.</td><td>1.116</td><td>1.116</td><td>2.000k</td><td>Mon Oct 29, 14:30:15</td><td>6s</td></tr></tbody></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	1.116	1.116	2.000k	Mon Oct 29, 14:30:15	6s
Name	Smoothed	Value	Step	Time	Relative									
.	1.116	1.116	2.000k	Mon Oct 29, 14:30:15	6s									

Q: 通过观察分析, 你认为, 对于当前代码, 哪种梯度优化算法最好? 为什么?

答: 我 Moment 算法最好。因为在使用 Moment 算法时, Total_cost 的收敛速度最快, 它引入了物理学中动量因素, 从而可以使得参数在梯度非常大的地方更新的非常快, 并且也能使 loss 避免陷入局部最小值和鞍点, 更容易到达全局最小值

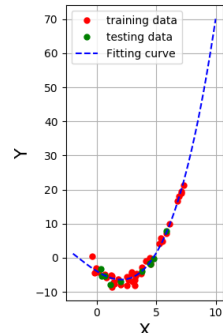
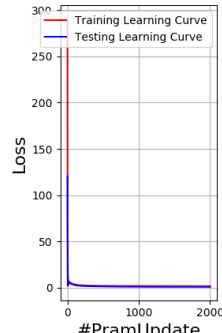


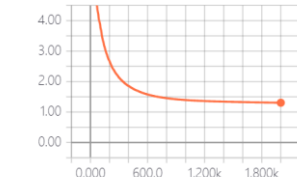
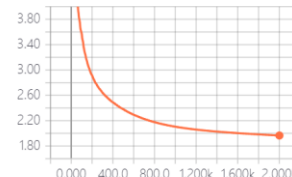
5. 数据集和验证集 CV 划分对“过拟合”的影响。采用 BGD 作为梯度下降方法。

超参数设置如下:

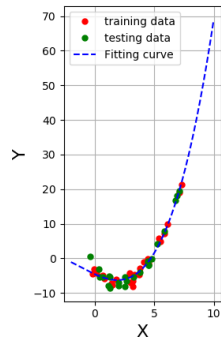
training_epochs	Learning Rate	BatchSize	N	OrderNum	Seed	DataScaleFlag
int(2000)	0.001	$N \times (1 - \text{TestingDataRatio})$	50	3	16225151	1

优化器选择: GradientDescentOptimizer

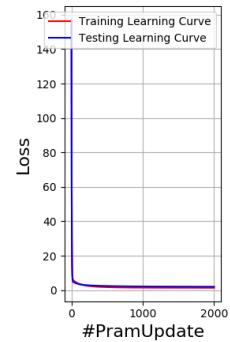
不同 TestingDataRatio 运行结果如下:

TestingD ataRatio	运行结果:	tensorboard																									
		train_set	test_set																								
0.2	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533], [-0.48727676]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 40.0 10.0 fit with 3-order linear model... prev_cost = 290.614 Initial_test_cost = 120.284 最佳参数 W,b: [[0.06226044] [0.33284118] [-2.15305597]] [-3.9324069] 6.15789794921875 秒 Training...Cost= 1.57914 Testing...Cost= 0.844565</pre></div> <div><div>fit with 3-order linear model</div><div>Loss Curve</div></div>	<div><table><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr><tr><td>.</td><td>1.580</td><td>1.579</td><td>2.000k</td><td>Mon Oct 29, 14:38:24</td><td>6s</td></tr></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	1.580	1.579	2.000k	Mon Oct 29, 14:38:24	6s	<div><table><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr><tr><td>.</td><td>0.8448</td><td>0.8446</td><td>2.000k</td><td>Mon Oct 29, 14:38:24</td><td>6s</td></tr></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	0.8448	0.8446	2.000k	Mon Oct 29, 14:38:24	6s
Name	Smoothed	Value	Step	Time	Relative																						
.	1.580	1.579	2.000k	Mon Oct 29, 14:38:24	6s																						
Name	Smoothed	Value	Step	Time	Relative																						
.	0.8448	0.8446	2.000k	Mon Oct 29, 14:38:24	6s																						
0.5	<div><pre>Initial w,b: [array([[3.15309572], [1.28588533], [-0.48727676]], dtype=float32), array([0.64348024], dtype=float32)] TrainingDataNum TestingDataNum= 25.0 25.0 fit with 3-order linear model... prev_cost = 156.422 Initial_test_cost = 157.037 最佳参数 W,b: [[0.06666175] [0.22492861] [-1.56640919]] [-4.57957458] 5.809838056564331 秒 Training...Cost= 1.30498 Testing...Cost= 1.96533</pre></div>	<div><table><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr><tr><td>.</td><td>1.305</td><td>1.305</td><td>2.000k</td><td>Mon Oct 29, 14:45:02</td><td>5s</td></tr></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	1.305	1.305	2.000k	Mon Oct 29, 14:45:02	5s	<div><table><tr><th>Name</th><th>Smoothed</th><th>Value</th><th>Step</th><th>Time</th><th>Relative</th></tr><tr><td>.</td><td>1.966</td><td>1.965</td><td>2.000k</td><td>Mon Oct 29, 14:45:02</td><td>5s</td></tr></table></div>	Name	Smoothed	Value	Step	Time	Relative	.	1.966	1.965	2.000k	Mon Oct 29, 14:45:02	5s
Name	Smoothed	Value	Step	Time	Relative																						
.	1.305	1.305	2.000k	Mon Oct 29, 14:45:02	5s																						
Name	Smoothed	Value	Step	Time	Relative																						
.	1.966	1.965	2.000k	Mon Oct 29, 14:45:02	5s																						

fit with 3-order linear model

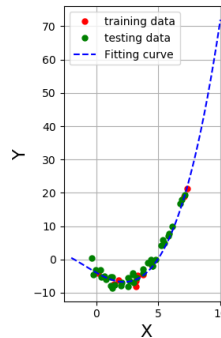


Loss Curve

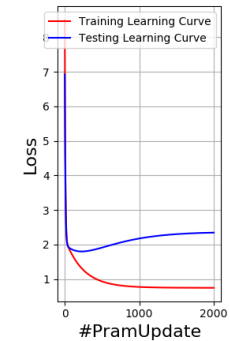


```
Initial w,b: [array([[ 3.15309572],
 [ 1.28588533],
 [-0.48727676]], dtype=float32), array([ 0.64348024], dtype=float32)]
TrainingDataNum|TestingDataNum= 9.999999999999998 | 40.0
fit with 3-order linear model...
prev_cost = 8.50283 Initial_test_cost = 6.92366
最佳参数 W,b: [[ 0.07627806]
 [ 0.20613954]
 [-2.10145584]] [-3.93530178]
7.0269551277160645 秒
Training...Cost= 0.746928
Testing...Cost= 2.34633
```

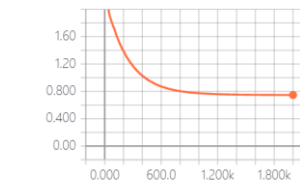
fit with 3-order linear model



Loss Curve

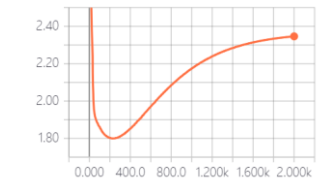


Cost/Total_cost



Name	Smoothed	Value	Step	Time	Relative
.	0.7469	0.7469	2.000k	Mon Oct 29, 14:48:49	6s

Cost/Total_cost



Name	Smoothed	Value	Step	Time	Relative
.	2.346	2.346	2.000k	Mon Oct 29, 14:48:49	6s

0.8

总结

过拟合的情况: `TestingDataRatio=0.8`

训练数据集的数据量对过拟合的影响: 通过实验观察可知: 训练集数量越少, 越容易产生过拟合。

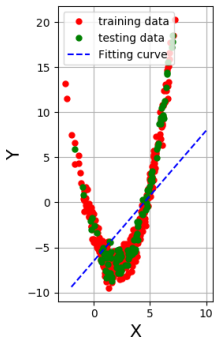
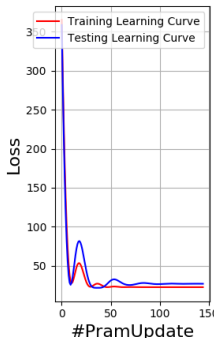
二、跨越不同模型进行比较, 从而找出终极的最佳 function。

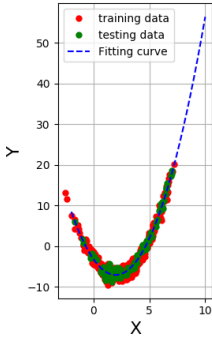
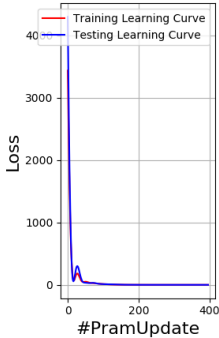
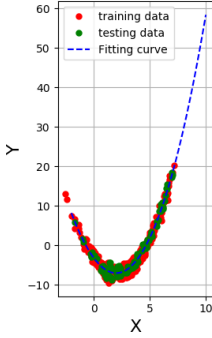
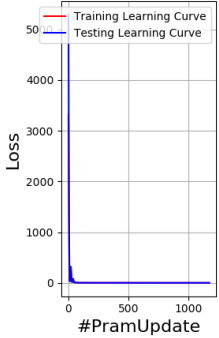
超参数设置如下:

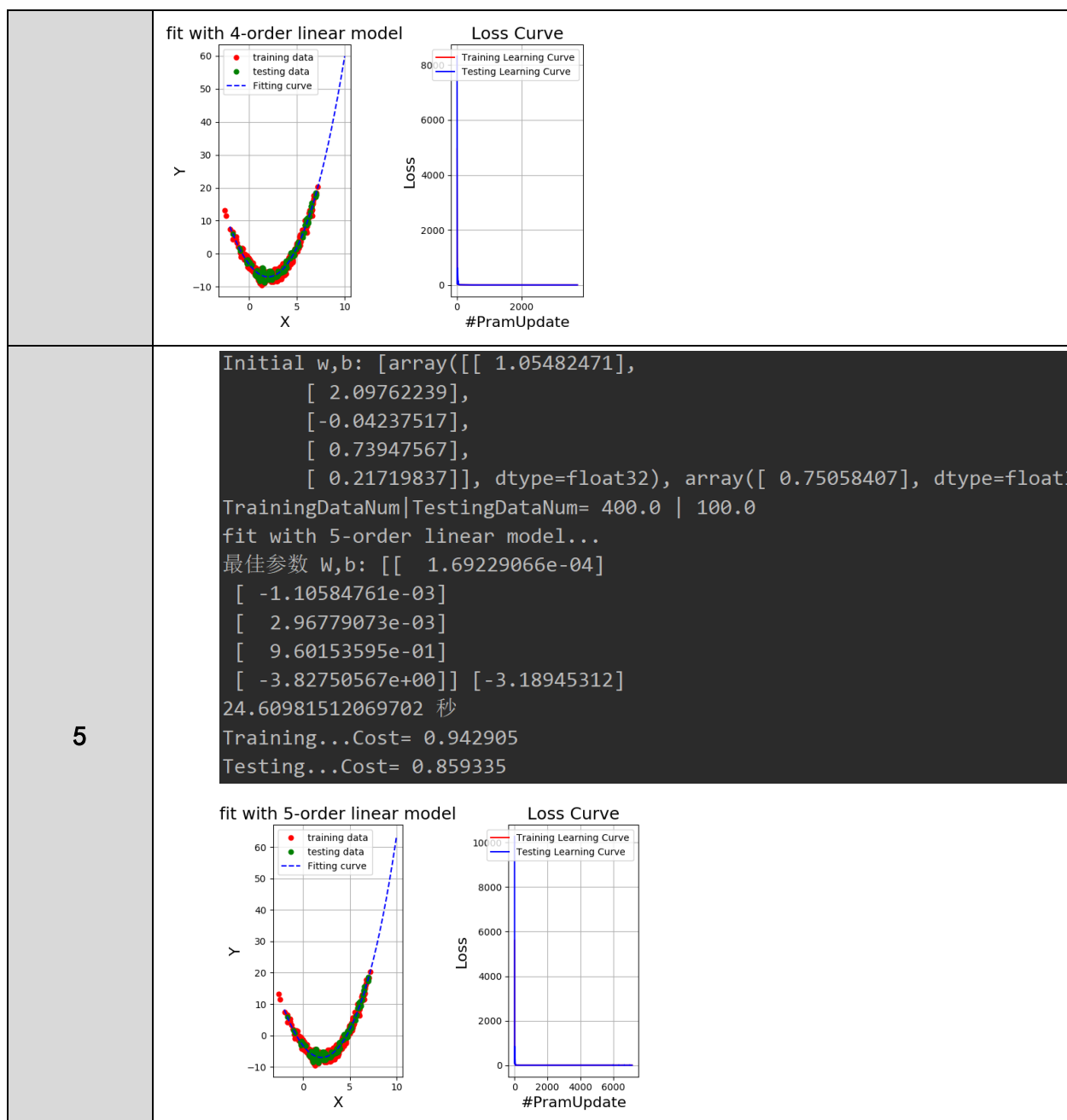
training_epochs	BatchSize	N	TestingDataRatio	Learning Rate	Seed	DataScaleFlag
<code>int(1e20)</code>	<code>N*(1-TestingDataRatio)</code>	500	0.2	0.1	18225034	1

优化器: `Adam`

不同阶数运行结果如下: `seed=18225034`

阶数	运行结果
1	<div><pre>Initial w,b: [array([[1.05482471]], dtype=float32), array([0.75058407], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 1-order linear model... 最佳参数 W,b: [[1.44410295]] [-6.48789215] 0.4378819465637207 秒 Training...Cost= 22.7445 Testing...Cost= 27.099</pre><div><div>fit with 1-order linear model</div></div><div><div>Loss Curve</div></div></div>
2	<div><pre>Initial w,b: [array([[1.05482471], [2.09762239]], dtype=float32), array([0.75058407], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 2-order linear model... 最佳参数 W,b: [[0.9840245] [-3.88252681]] [-3.20840335] 1.2812178134918213 秒 Training...Cost= 0.949577 Testing...Cost= 0.870324</pre></div>

	<div><div>fit with 2-order linear model</div><div>Loss Curve</div></div>
3	<div><div><pre>Initial w,b: [array([[1.05482471], [2.09762239], [-0.04237517]], dtype=float32), array([0.75058407], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 3-order linear model... 最佳参数 W,b: [[0.00576481] [0.94008369] [-3.82704389]] [-3.16074848] 3.8295583724975586 秒 Training...Cost= 0.944043 Testing...Cost= 0.861646</pre></div><div><div>fit with 3-order linear model</div><div>Loss Curve</div></div></div>
4	<div><div><pre>Initial w,b: [array([[1.05482471], [2.09762239], [-0.04237517], [0.73947567]], dtype=float32), array([0.75058407], dtype=float32)] TrainingDataNum TestingDataNum= 400.0 100.0 fit with 4-order linear model... 最佳参数 W,b: [[6.02278994e-04] [3.84316235e-04] [9.47643793e-01] [-3.80683538e+00]] [-3.18080592] 10.910299062728882 秒 Training...Cost= 0.943449 Testing...Cost= 0.859579</pre></div></div>



将上表数据整理后得下表:

Model & Optimized function	Training Loss	Testing Loss
1 阶: $y=1.44x-6.49$	22.7445	27.099
2 阶: $y=0.98x^2-3.88x-3.21$	0.949577	0.870324
3 阶: $y=5.8e-3x^3+0.94x^2-3.82x-3.16$	0.944043	0.861646
4 阶: $y=6.02e-4x^4+3.84e-4x^3+0.95x^2-3.80x-3.18$	0.943449	0.859579
5 阶: $y=1.69e-4x^5-1.11e-3x^4+2.97e-3x^3+0.96x^2-3.82x-3.19$	0.942905	0.859335

Q: 观察上表中你的记录结果，试找出跨模型的最佳 function，并说出你的选择理由。

答: 最佳模型为 2 阶: $y=0.98*x^2-3.88*x-3.21$ 。

原因如下: 第一, 虽然在增加阶数到 3/4/5 时, 看不出明显的过拟合, 但是可以明显发现, 阶数升高时, 最终训练结果中, 3 次及 3 次以上的项参数都几乎接近于 0, 所以相当于还是 2 阶模型。第二, 如果我们选择高阶模型, 有可能会增加过拟合的几率。