组会汇报

2.4

FASHION-MNIST

```
njh@GPU:~$ python3 train.py
epoch 0, loss 43.0246, train accuracy 0.614, test accuracy 0.754, use time 37.249
epoch 1, loss 0.5672, train accuracy 0.788, test accuracy 0.790, use time 34.270
epoch 2, loss 0.4934, train accuracy 0.816, test accuracy 0.819, use time 34.075
epoch 3, loss 0.4477, train accuracy 0.832, test accuracy 0.827, use time 33.299
epoch 4, loss 0.6111, train accuracy 0.806, test accuracy 0.665, use time 33.340
epoch 5, loss 0.5595, train accuracy 0.791, test accuracy 0.810, use time 33.849
epoch 6, loss 0.4676, train accuracy 0.825, test accuracy 0.821, use time 33.659
epoch 7, loss 0.4263, train accuracy 0.839, test accuracy 0.837, use time 33.423
epoch 8, loss 0.4075, train accuracy 0.846, test accuracy 0.838, use time 33.811
epoch 9, loss 0.3894, train accuracy 0.851, test accuracy 0.843, use time 33.418
epoch 10, loss 0.3800, train accuracy 0.856, test accuracy 0.844, use time 33.737
epoch 11, loss 0.3744, train accuracy 0.858, test accuracy 0.839, use time 33.519
epoch 12, loss 0.3690, train accuracy 0.861, test accuracy 0.850, use time 33.683
epoch 13, loss 0.3615, train accuracy 0.863, test accuracy 0.845, use time 34.430
epoch 14, loss 0.3607, train accuracy 0.863, test accuracy 0.851, use time 33.962
```

SGD

```
epoch 0, loss 0.5144, train accuracy 0.812, test accuracy 0.873, use time 42.696
epoch 1, loss 0.3144, train accuracy 0.884, test accuracy 0.888, use time 38.177
epoch 2, loss 0.2686, train accuracy 0.901, test accuracy 0.899, use time 38.287
epoch 3, loss 0.2376, train accuracy 0.911, test accuracy 0.908, use time 38.331
epoch 4, loss 0.2147, train accuracy 0.920, test accuracy 0.912, use time 38.034
epoch 5, loss 0.1927, train accuracy 0.929, test accuracy 0.909, use time 38.215
epoch 6, loss 0.1775, train accuracy 0.935, test accuracy 0.913, use time 38.557
epoch 7, loss 0.1612, train accuracy 0.941, test accuracy 0.923, use time 38.074
epoch 8, loss 0.1463, train accuracy 0.946, test accuracy 0.925, use time 38.026
epoch 9, loss 0.1312, train accuracy 0.951, test accuracy 0.918, use time 38.451
epoch 10, loss 0.1207, train accuracy 0.956, test accuracy 0.927, use time 38.218
epoch 11, loss 0.1108, train accuracy 0.959, test accuracy 0.921, use time 38.279
epoch 12, loss 0.1014, train accuracy 0.963, test accuracy 0.922, use time 39.782
epoch 13, loss 0.0886, train accuracy 0.967, test accuracy 0.918, use time 38.239
epoch 14, loss 0.0850, train accuracy 0.969, test accuracy 0.927, use time 38.413
```

BatchNorm

Adam

```
epoch 0, loss 0.7249, train accuracy 0.768, test accuracy 0.838, use time 42.216
epoch 1, loss 0.3749, train accuracy 0.860, test accuracy 0.866, use time 37.492
epoch 2, loss 0.3195, train accuracy 0.881, test accuracy 0.886, use time 37.456
epoch 3, loss 0.2839, train accuracy 0.895, test accuracy 0.890, use time 37.893
epoch 4, loss 0.2672, train accuracy 0.900, test accuracy 0.889, use time 37.509
epoch 5, loss 0.2498, train accuracy 0.907, test accuracy 0.898, use time 37.428
epoch 6, loss 0.2329, train accuracy 0.915, test accuracy 0.894, use time 37.776
epoch 7, loss 0.2232, train accuracy 0.918, test accuracy 0.904, use time 37.254
epoch 8, loss 0.2081, train accuracy 0.923, test accuracy 0.904, use time 37.471
epoch 9, loss 0.1937, train accuracy 0.928, test accuracy 0.902, use time 37.752
epoch 10, loss 0.1806, train accuracy 0.932, test accuracy 0.909, use time 37.508
epoch 11, loss 0.1677, train accuracy 0.938, test accuracy 0.908, use time 37.706
epoch 12, loss 0.1549, train accuracy 0.942, test accuracy 0.909, use time 37.630
epoch 13, loss 0.1440, train accuracy 0.945, test accuracy 0.910, use time 37.904
epoch 14, loss 0.1287, train accuracy 0.952, test accuracy 0.913, use time 38.090
```

Batch Normalization

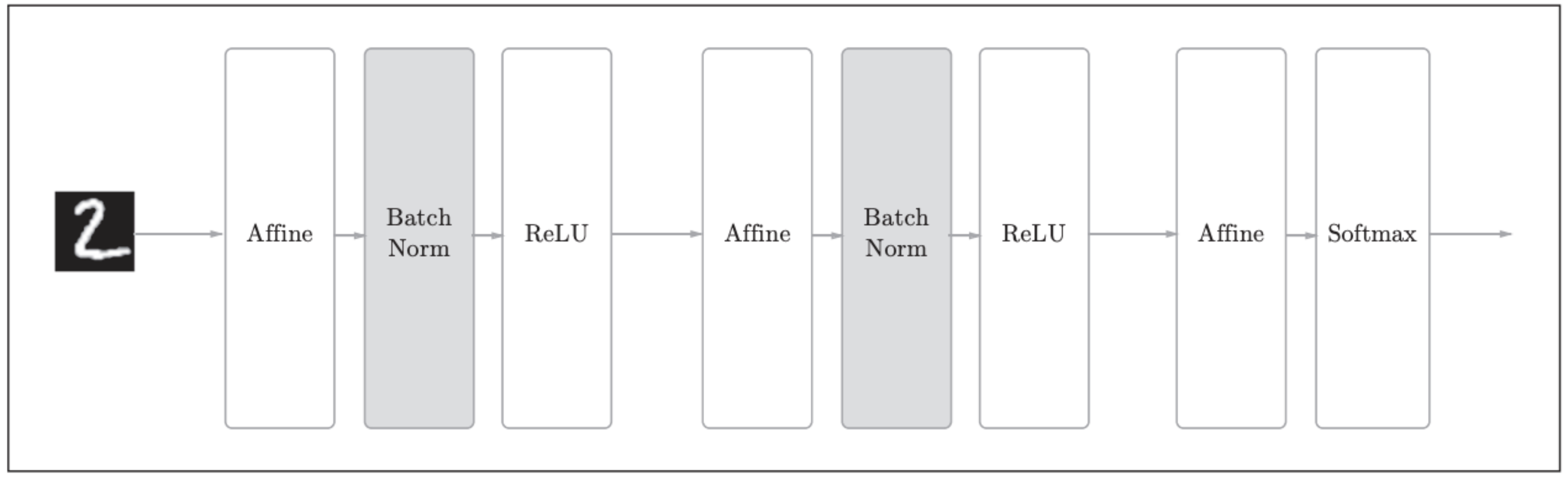


图6-16　使用了Batch Normalization的神经网络的例子（Batch Norm层的背景为灰色）

Batch Norm，顾名思义，以进行学习时的mini-batch为单位，按mini-batch进行正规化。具体而言，就是进行使数据分布的均值为0、方差为1的正规化。用数学式表示的话，如下所示。

CLASS torch.nn.BatchNorm2d(*num_features*, *eps=1e-05*, *momentum=0.1*, *affine=True*, *track_running_stats=True*, *device=None*, *dtype=None*) [SOURCE]

Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

CIFAR-10

```
epoch 0, loss 2.2944, train accuracy 0.114, test accuracy 0.152, use time 84.213
epoch 1, loss 2.0281, train accuracy 0.250, test accuracy 0.301, use time 90.211
epoch 2, loss 1.7210, train accuracy 0.366, test accuracy 0.404, use time 75.184
epoch 3, loss 1.5447, train accuracy 0.433, test accuracy 0.457, use time 83.158
epoch 4, loss 1.4065, train accuracy 0.491, test accuracy 0.522, use time 98.114
epoch 5, loss 1.2980, train accuracy 0.536, test accuracy 0.553, use time 100.965
epoch 6, loss 1.2019, train accuracy 0.575, test accuracy 0.584, use time 100.421
epoch 7, loss 1.1140, train accuracy 0.609, test accuracy 0.600, use time 98.588
epoch 8, loss 1.0213, train accuracy 0.643, test accuracy 0.637, use time 100.570
epoch 9, loss 0.9409, train accuracy 0.672, test accuracy 0.664, use time 96.640
epoch 10, loss 0.8717, train accuracy 0.698, test accuracy 0.684, use time 99.888
epoch 11, loss 0.8044, train accuracy 0.719, test accuracy 0.694, use time 97.455
epoch 12, loss 0.7362, train accuracy 0.745, test accuracy 0.727, use time 102.919
epoch 13, loss 0.6734, train accuracy 0.765, test accuracy 0.737, use time 93.242
epoch 14, loss 0.6209, train accuracy 0.785, test accuracy 0.753, use time 99.412
```

which optimizer is best for cifar-10?

全部 图片 视频 新闻 购物 更多 工具

找到约 115,000 条结果 （用时 0.39 秒）

The Adamax optimizer.

By now I had observed that the highest accuracy rate can be achieved using Adamax optimizer by making a few tweeks in the hyperparameters.

SGD →

```
epoch 0, loss 1.4537, train accuracy 0.469, test accuracy 0.577, use time 54.414
epoch 1, loss 1.0228, train accuracy 0.637, test accuracy 0.680, use time 48.995
epoch 2, loss 0.8156, train accuracy 0.715, test accuracy 0.717, use time 51.254
epoch 3, loss 0.6828, train accuracy 0.763, test accuracy 0.757, use time 52.961
epoch 4, loss 0.5808, train accuracy 0.799, test accuracy 0.777, use time 54.321
epoch 5, loss 0.5003, train accuracy 0.827, test accuracy 0.780, use time 53.460
epoch 6, loss 0.4303, train accuracy 0.852, test accuracy 0.798, use time 53.435
epoch 7, loss 0.3762, train accuracy 0.870, test accuracy 0.812, use time 53.828
epoch 8, loss 0.3221, train accuracy 0.888, test accuracy 0.808, use time 53.324
epoch 9, loss 0.2761, train accuracy 0.904, test accuracy 0.815, use time 53.568
epoch 10, loss 0.2290, train accuracy 0.921, test accuracy 0.822, use time 53.205
epoch 11, loss 0.1878, train accuracy 0.934, test accuracy 0.820, use time 53.791
epoch 12, loss 0.1583, train accuracy 0.945, test accuracy 0.829, use time 53.025
epoch 13, loss 0.1297, train accuracy 0.955, test accuracy 0.818, use time 52.535
epoch 14, loss 0.1131, train accuracy 0.961, test accuracy 0.824, use time 52.286
```

Adam →

```
epoch 0, loss 1.9542, train accuracy 0.296, test accuracy 0.389, use time 59.933
epoch 1, loss 1.5158, train accuracy 0.439, test accuracy 0.492, use time 56.756
epoch 2, loss 1.3214, train accuracy 0.524, test accuracy 0.565, use time 56.949
epoch 3, loss 1.1687, train accuracy 0.584, test accuracy 0.604, use time 60.224
epoch 4, loss 1.0314, train accuracy 0.642, test accuracy 0.651, use time 63.520
epoch 5, loss 0.9164, train accuracy 0.681, test accuracy 0.702, use time 63.810
epoch 6, loss 0.8177, train accuracy 0.720, test accuracy 0.705, use time 64.023
epoch 7, loss 0.7231, train accuracy 0.752, test accuracy 0.733, use time 63.928
epoch 8, loss 0.6490, train accuracy 0.782, test accuracy 0.752, use time 63.110
epoch 9, loss 0.5817, train accuracy 0.804, test accuracy 0.764, use time 63.558
epoch 10, loss 0.5241, train accuracy 0.824, test accuracy 0.769, use time 60.898
epoch 11, loss 0.4712, train accuracy 0.843, test accuracy 0.784, use time 57.280
epoch 12, loss 0.4143, train accuracy 0.860, test accuracy 0.793, use time 54.727
epoch 13, loss 0.3696, train accuracy 0.876, test accuracy 0.791, use time 54.695
epoch 14, loss 0.3296, train accuracy 0.890, test accuracy 0.788, use time 55.180
```

Adamax →

```
epoch 0, loss 2.0259, train accuracy 0.286, test accuracy 0.376, use time 54.305
epoch 1, loss 1.5349, train accuracy 0.436, test accuracy 0.466, use time 50.971
epoch 2, loss 1.3206, train accuracy 0.523, test accuracy 0.565, use time 50.794
epoch 3, loss 1.1456, train accuracy 0.599, test accuracy 0.626, use time 50.888
epoch 4, loss 1.0188, train accuracy 0.648, test accuracy 0.668, use time 50.957
epoch 5, loss 0.8957, train accuracy 0.692, test accuracy 0.687, use time 51.151
epoch 6, loss 0.8042, train accuracy 0.725, test accuracy 0.711, use time 53.184
epoch 7, loss 0.7097, train accuracy 0.756, test accuracy 0.737, use time 52.757
epoch 8, loss 0.6402, train accuracy 0.783, test accuracy 0.746, use time 54.271
epoch 9, loss 0.5696, train accuracy 0.807, test accuracy 0.759, use time 53.953
epoch 10, loss 0.5153, train accuracy 0.825, test accuracy 0.774, use time 51.632
epoch 11, loss 0.4669, train accuracy 0.842, test accuracy 0.785, use time 53.168
epoch 12, loss 0.4201, train accuracy 0.857, test accuracy 0.780, use time 53.361
epoch 13, loss 0.3738, train accuracy 0.874, test accuracy 0.787, use time 53.124
epoch 14, loss 0.3395, train accuracy 0.887, test accuracy 0.788, use time 53.449
```

CIFAR-10

```
epoch 0, loss 0.8929, train accuracy 0.691, test accuracy 0.773, use time 82.349
epoch 1, loss 0.5525, train accuracy 0.809, test accuracy 0.814, use time 74.668
epoch 2, loss 0.4361, train accuracy 0.850, test accuracy 0.815, use time 79.936
epoch 3, loss 0.3586, train accuracy 0.876, test accuracy 0.839, use time 70.987
epoch 4, loss 0.3027, train accuracy 0.894, test accuracy 0.843, use time 72.816
epoch 5, loss 0.2604, train accuracy 0.911, test accuracy 0.822, use time 67.790
epoch 6, loss 0.2156, train accuracy 0.924, test accuracy 0.815, use time 48.528
epoch 7, loss 0.1962, train accuracy 0.930, test accuracy 0.859, use time 49.300
epoch 8, loss 0.1857, train accuracy 0.935, test accuracy 0.840, use time 49.706
epoch 9, loss 0.1586, train accuracy 0.945, test accuracy 0.856, use time 50.147
epoch 10, loss 0.1536, train accuracy 0.948, test accuracy 0.849, use time 48.788
epoch 11, loss 0.1352, train accuracy 0.954, test accuracy 0.849, use time 49.331
epoch 12, loss 0.1322, train accuracy 0.955, test accuracy 0.837, use time 48.535
epoch 13, loss 0.1185, train accuracy 0.960, test accuracy 0.846, use time 48.859
epoch 14, loss 0.1114, train accuracy 0.962, test accuracy 0.855, use time 48.045
```

超参数调整

```
epoch 0, loss 0.7610, train accuracy 0.734, test accuracy 0.797, use time 47.502
epoch 1, loss 0.4498, train accuracy 0.843, test accuracy 0.841, use time 43.612
epoch 2, loss 0.3445, train accuracy 0.882, test accuracy 0.846, use time 44.195
epoch 3, loss 0.2780, train accuracy 0.903, test accuracy 0.842, use time 43.485
epoch 4, loss 0.2260, train accuracy 0.922, test accuracy 0.862, use time 44.408
epoch 5, loss 0.1807, train accuracy 0.936, test accuracy 0.873, use time 44.024
epoch 6, loss 0.1600, train accuracy 0.943, test accuracy 0.876, use time 43.165
epoch 7, loss 0.1276, train accuracy 0.955, test accuracy 0.865, use time 43.973
epoch 8, loss 0.1100, train accuracy 0.960, test accuracy 0.872, use time 44.883
epoch 9, loss 0.0919, train accuracy 0.967, test accuracy 0.874, use time 45.547
epoch 10, loss 0.0810, train accuracy 0.971, test accuracy 0.878, use time 46.303
epoch 11, loss 0.0739, train accuracy 0.974, test accuracy 0.873, use time 45.254
epoch 12, loss 0.0634, train accuracy 0.978, test accuracy 0.878, use time 43.828
epoch 13, loss 0.0636, train accuracy 0.978, test accuracy 0.876, use time 43.901
epoch 14, loss 0.0549, train accuracy 0.981, test accuracy 0.881, use time 43.742
```

Model builders

The following model builders can be used to instanciate an AlexNet model, with or without pre-trained weights. All the model builders internally rely on the `torchvision.models.alexnet.AlexNet` base class. Please refer to the source code for more details about this class.

`alexnet`(*[, weights, progress])	AlexNet model architecture from One weird trick for parallelizing convolutional neural networks.

Alexnet

Alexnet was introduced in the paper ImageNet Classification with Deep Convolutional Neural Networks and was the first very successful CNN on the ImageNet dataset. When we print the model architecture, we see the model output comes from the 6th layer of the classifier

```
(classifier): Sequential(
    ...
    (6): Linear(in_features=4096, out_features=1000, bias=True)
 )
```

To use the model with our dataset we reinitialize this layer as

```
model.classifier[6] = nn.Linear(4096,num_classes)
```

有无归一化

```
epoch 0, loss 1.9424, train accuracy 0.271, test accuracy 0.522, use time 49.315
epoch 1, loss 0.8763, train accuracy 0.692, test accuracy 0.759, use time 43.809
epoch 2, loss 0.5777, train accuracy 0.799, test accuracy 0.796, use time 44.151
epoch 3, loss 0.4500, train accuracy 0.844, test accuracy 0.834, use time 41.860
epoch 4, loss 0.3737, train accuracy 0.870, test accuracy 0.836, use time 43.011
epoch 5, loss 0.3111, train accuracy 0.892, test accuracy 0.837, use time 42.267
epoch 6, loss 0.2592, train accuracy 0.908, test accuracy 0.843, use time 42.631
epoch 7, loss 0.2197, train accuracy 0.922, test accuracy 0.857, use time 43.221
epoch 8, loss 0.1839, train accuracy 0.936, test accuracy 0.852, use time 43.483
epoch 9, loss 0.1510, train accuracy 0.947, test accuracy 0.859, use time 43.555
epoch 10, loss 0.1331, train accuracy 0.953, test accuracy 0.861, use time 44.473
epoch 11, loss 0.1094, train accuracy 0.961, test accuracy 0.868, use time 44.116
epoch 12, loss 0.1008, train accuracy 0.965, test accuracy 0.870, use time 43.070
epoch 13, loss 0.0852, train accuracy 0.970, test accuracy 0.861, use time 42.911
epoch 14, loss 0.0747, train accuracy 0.975, test accuracy 0.868, use time 45.040
```

CIFAR-10

Adam

```
epoch 0, loss 1.3609, train accuracy 0.508, test accuracy 0.639, use time 53.103
epoch 1, loss 0.8843, train accuracy 0.697, test accuracy 0.695, use time 50.215
epoch 2, loss 0.7295, train accuracy 0.751, test accuracy 0.687, use time 50.180
epoch 3, loss 0.6472, train accuracy 0.780, test accuracy 0.759, use time 50.081
epoch 4, loss 0.5860, train accuracy 0.802, test accuracy 0.768, use time 47.601
epoch 5, loss 0.5338, train accuracy 0.819, test accuracy 0.768, use time 48.452
epoch 6, loss 0.5074, train accuracy 0.828, test accuracy 0.770, use time 49.825
epoch 7, loss 0.4843, train accuracy 0.837, test accuracy 0.763, use time 48.281
epoch 8, loss 0.4600, train accuracy 0.846, test accuracy 0.771, use time 48.986
epoch 9, loss 0.4418, train accuracy 0.852, test accuracy 0.763, use time 54.760
epoch 10, loss 0.4253, train accuracy 0.858, test accuracy 0.765, use time 62.808
epoch 11, loss 0.4192, train accuracy 0.861, test accuracy 0.773, use time 62.982
epoch 12, loss 0.3852, train accuracy 0.872, test accuracy 0.789, use time 63.091
epoch 13, loss 0.3708, train accuracy 0.878, test accuracy 0.783, use time 64.363
epoch 14, loss 0.3674, train accuracy 0.880, test accuracy 0.793, use time 64.912
```

Adamax

```
epoch 0, loss 1.7564, train accuracy 0.347, test accuracy 0.514, use time 70.241
epoch 1, loss 1.0135, train accuracy 0.640, test accuracy 0.705, use time 61.340
epoch 2, loss 0.7046, train accuracy 0.758, test accuracy 0.777, use time 63.772
epoch 3, loss 0.5423, train accuracy 0.813, test accuracy 0.788, use time 61.027
epoch 4, loss 0.4415, train accuracy 0.848, test accuracy 0.821, use time 65.661
epoch 5, loss 0.3612, train accuracy 0.875, test accuracy 0.823, use time 56.317
epoch 6, loss 0.2964, train accuracy 0.896, test accuracy 0.825, use time 46.991
epoch 7, loss 0.2492, train accuracy 0.913, test accuracy 0.833, use time 43.516
epoch 8, loss 0.2073, train accuracy 0.929, test accuracy 0.828, use time 43.993
epoch 9, loss 0.1716, train accuracy 0.942, test accuracy 0.838, use time 44.385
epoch 10, loss 0.1523, train accuracy 0.948, test accuracy 0.840, use time 43.796
epoch 11, loss 0.1348, train accuracy 0.953, test accuracy 0.843, use time 43.867
epoch 12, loss 0.1170, train accuracy 0.960, test accuracy 0.835, use time 43.952
epoch 13, loss 0.1075, train accuracy 0.964, test accuracy 0.849, use time 43.756
epoch 14, loss 0.0976, train accuracy 0.968, test accuracy 0.838, use time 43.818
```

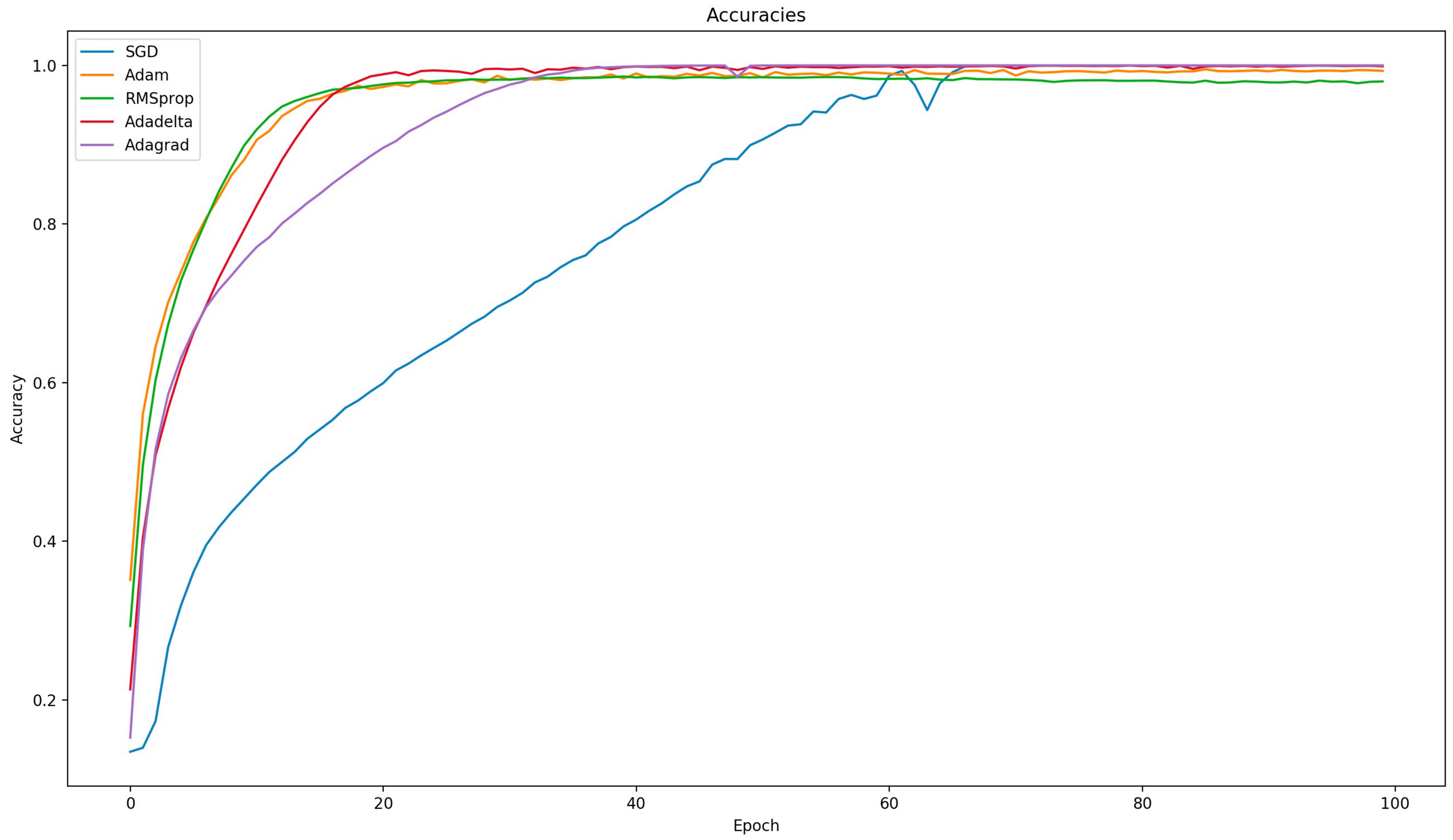
```
epoch 15, loss 0.1031, train accuracy 0.965, test accuracy 0.839, use time 46.991
epoch 16, loss 0.0907, train accuracy 0.970, test accuracy 0.843, use time 44.981
epoch 17, loss 0.0862, train accuracy 0.970, test accuracy 0.846, use time 46.403
epoch 18, loss 0.0818, train accuracy 0.973, test accuracy 0.853, use time 46.870
epoch 19, loss 0.0766, train accuracy 0.975, test accuracy 0.844, use time 48.358
epoch 20, loss 0.0737, train accuracy 0.975, test accuracy 0.851, use time 47.961
epoch 21, loss 0.0713, train accuracy 0.977, test accuracy 0.858, use time 46.382
epoch 22, loss 0.0720, train accuracy 0.976, test accuracy 0.850, use time 55.391
epoch 23, loss 0.0633, train accuracy 0.979, test accuracy 0.853, use time 76.011
epoch 24, loss 0.0660, train accuracy 0.978, test accuracy 0.843, use time 72.167
epoch 25, loss 0.0664, train accuracy 0.978, test accuracy 0.849, use time 72.037
epoch 26, loss 0.0558, train accuracy 0.981, test accuracy 0.843, use time 71.044
epoch 27, loss 0.0576, train accuracy 0.980, test accuracy 0.851, use time 78.266
epoch 28, loss 0.0558, train accuracy 0.982, test accuracy 0.854, use time 70.550
epoch 29, loss 0.0523, train accuracy 0.983, test accuracy 0.836, use time 76.267
```

SGD 30epoch

```
epoch 27, loss 0.0215, train accuracy 0.993, test accuracy 0.895, use time 66.261
epoch 28, loss 0.0191, train accuracy 0.994, test accuracy 0.889, use time 67.603
epoch 29, loss 0.0218, train accuracy 0.993, test accuracy 0.887, use time 71.037
```

总结

- 更宽的网络、增加BN、Drop对模型泛化能力影响较大
- 优化器、学习率等超参数也能有小幅提升



优化器

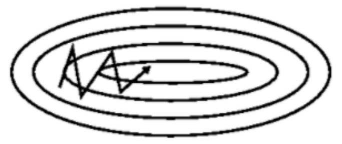
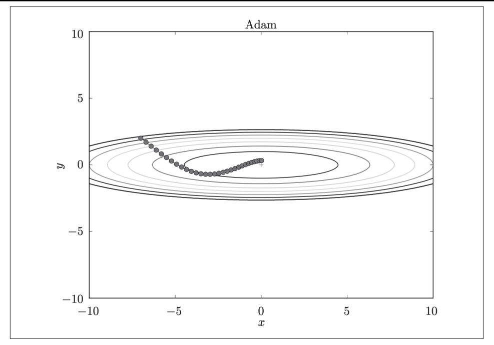
An overview of gradient descent optimization algorithms

An overview of gradient descent optimization algorithms*

Sebastian Ruder
Insight Centre for Data Analytics, NUI Galway
Aylien Ltd., Dublin
ruder.sebastian@gmail.com

Abstract

Gradient descent optimization algorithms, while increasingly popular, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by. This article aims to provide the reader with intuitions with regard to the behaviour of different algorithms that will allow her to put them to use. In the course of this overview, we look at different variants of gradient descent, summarize challenges, introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent.

	特点	缺点	
BGD	整个训练集数据计算梯度	慢	陷入局部最小值或者鞍点
SGD/MBGD	随机一个样本/每次一小批	不一定是全局最优	
Momentum/NAG	$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$ $\theta = \theta - v_t$	缺乏适应性	 Image 2: SGD without momentum  Image 3: SGD with momentum
Adagrad/Adadelta	为参数的每个元素适当地调整学习率	学习越深入，更新的幅度就越小	$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$
Adam	计算每个参数的自适应学习率		 图6-7 基于Adam的最优化的更新路径

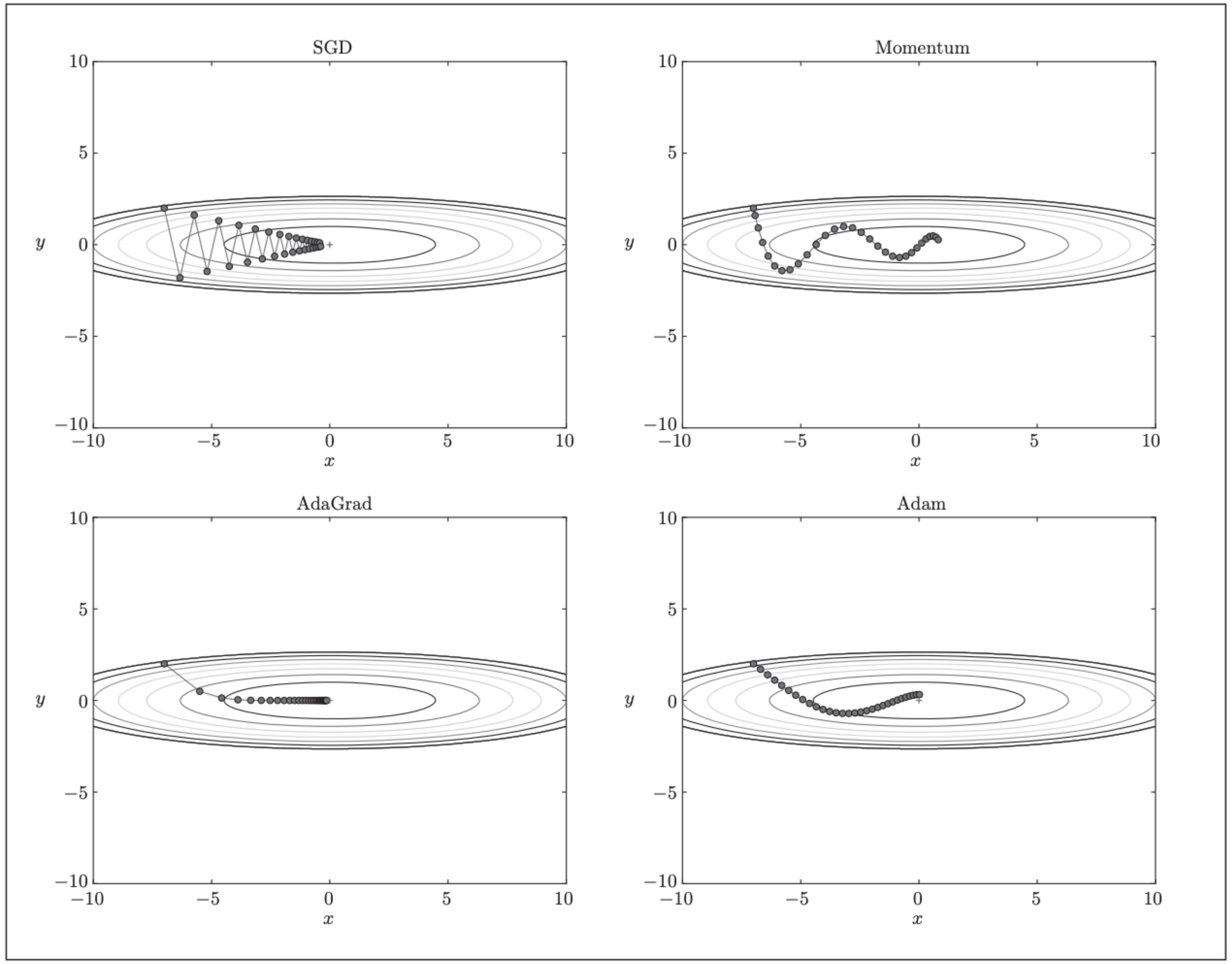


图6-8　最优化方法的比较：SGD、Momentum、AdaGrad、Adam