【PyTorch深度学习实战案例】70个练手项目合集，七天练完，练完即可就业！还不拿下 ！！！（很少有人可以把pytorch讲的如此通俗易懂了）

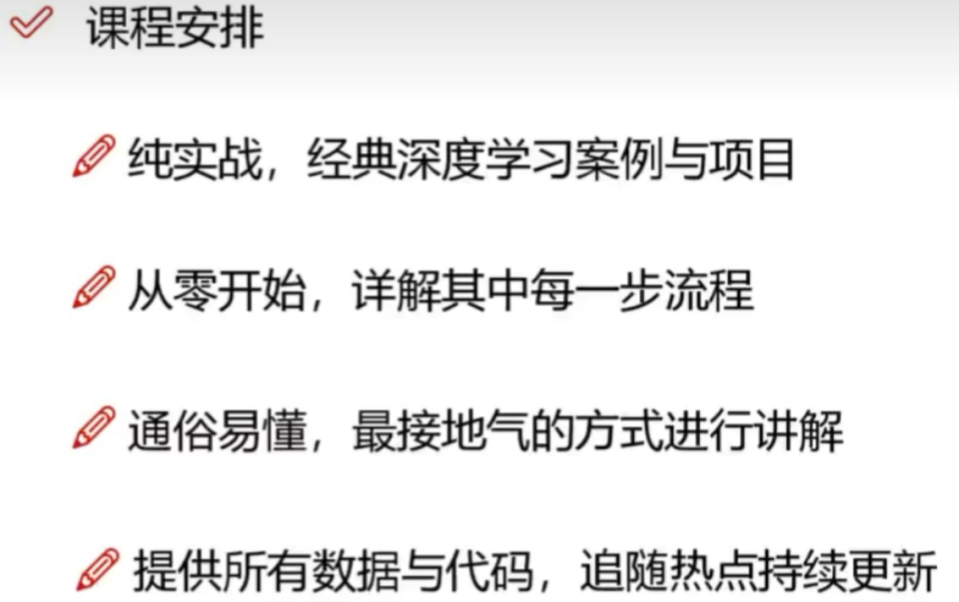
2023-02-13 17:35:11

作者 : [歌舞町女王hhhh](https://space.bilibili.com/1831048913" \t "https://www.bilibili.com/video/BV1a14y1c72R/_blank)

<https://www.bilibili.com/video/BV1a14y1c72R/?spm_id_from=333.337.search-card.all.click&vd_source=4212b105520112daf65694a1e5944e23>

# P1 1.01-PyTorch实战课程简介

04:57



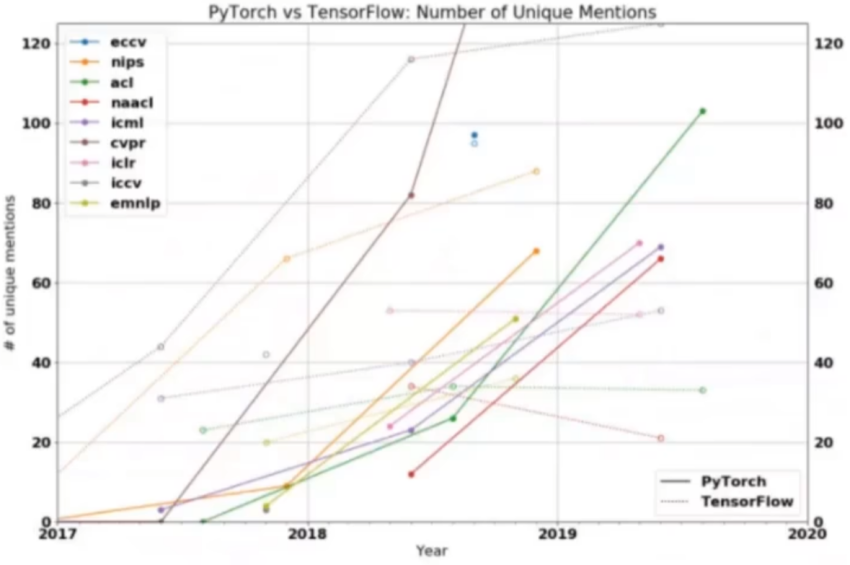
# P2 2. 02 - 002 PyTorch框架发展趋势简介

08:26

# 1.PyTorch登场

Torch是什么?一个火炬!其实跟Tensoriow中Tensor是一个意思，当做是能在GPU中计算的矩阵就可以啦!如果没玩过Tensorfow，Numpy总用过吧，也可以当

做是ndarray的GPU版!



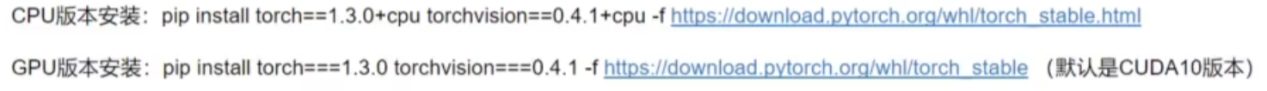
PyTorch可以说是现阶段主流的深度学习框架了，武林盟主之争大概是这个历史,。。。15年底之前Cafe是老大哥，随着Tensorow的诞生，霸占江湖数载，19年起

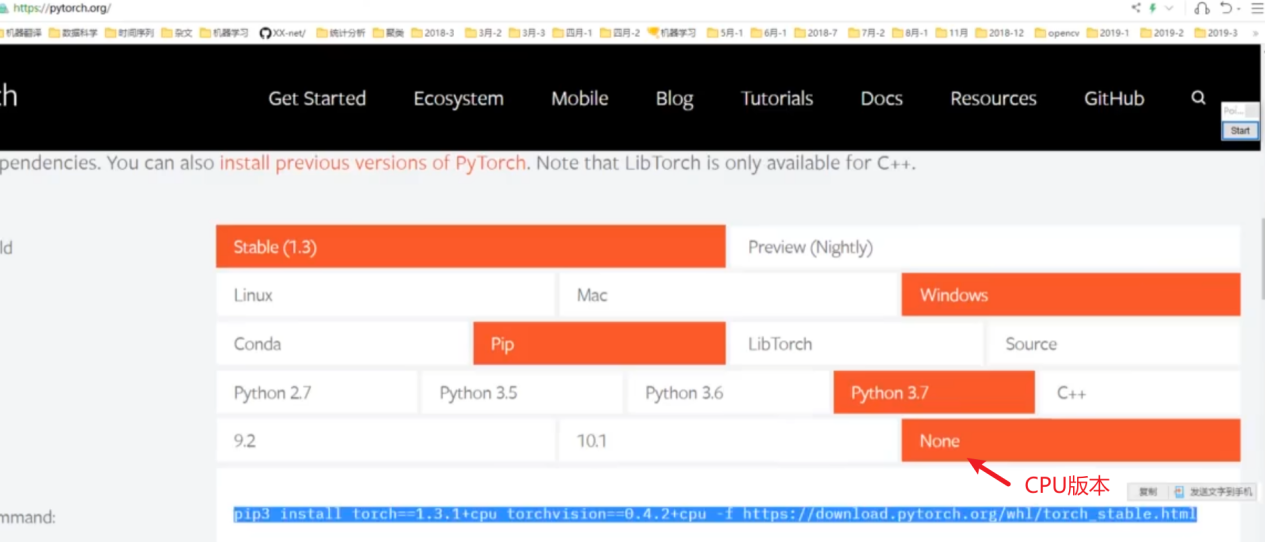
无论从学术界还是工程界PyTorch已经霸占了半壁江山!

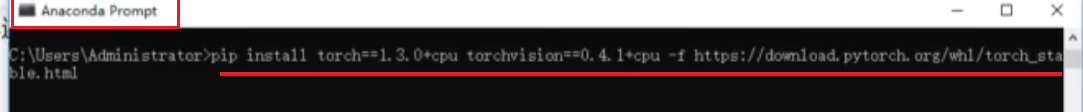
# P3 3. 03 - 003 框架安装方法（CPU与GPU版本）

05:14

## 1.安装PyTorch:使用PIP的方法比较简单



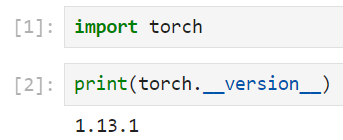


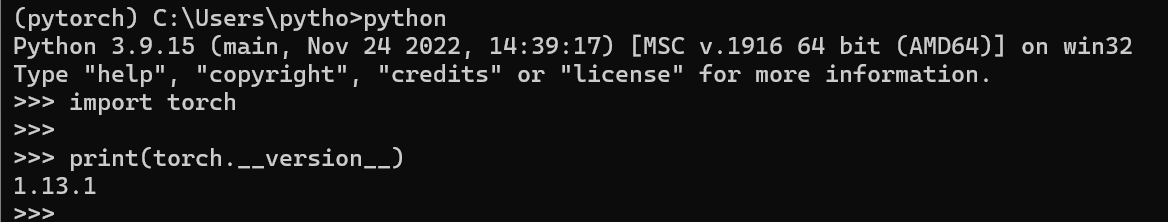


安装cuda



## 2.验证是否安装成功：



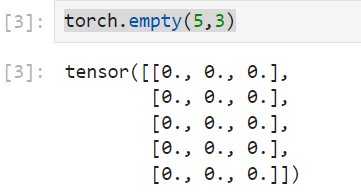


# P4 4.04-004 PyTorch基本操作

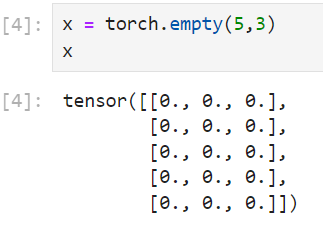
09:26

## 1.基本使用方法

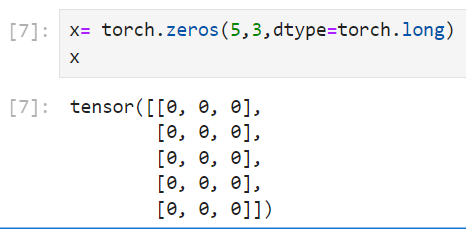
创一个矩阵，有木有觉得很爽，如果用tensorfow的同学可能会这么觉得。



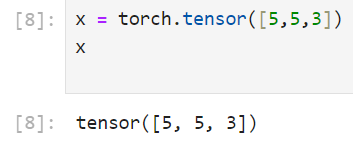
来个随机值试试水



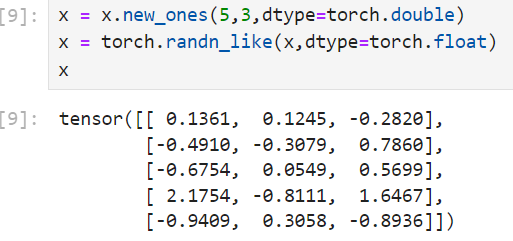
初始化一个全零的矩阵



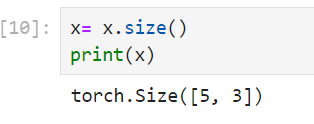
直接传入数据



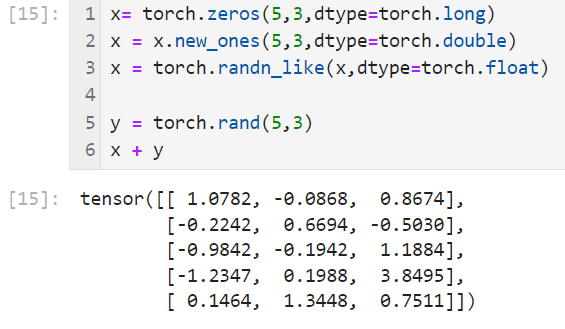
感觉是不是跟Numpy差不多，其实这些框架的基本操作基本都是大同小异

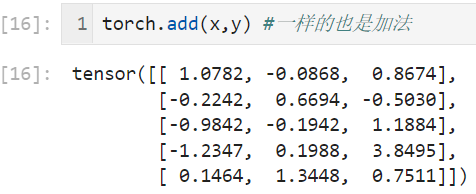


展示矩阵大小（用于查看运行结果是否正确）

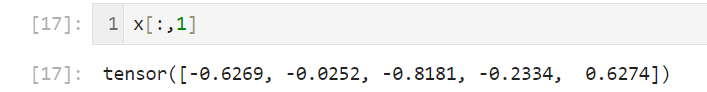


## 2.基本运算

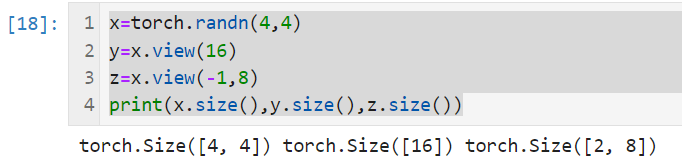




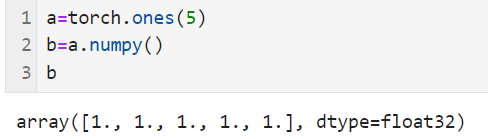
## 3.索引

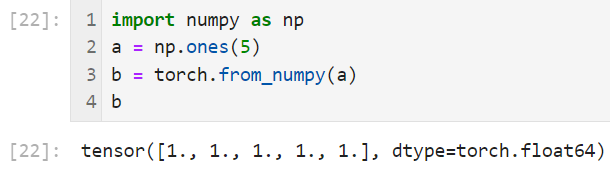


## 4.view操作可以改变矩阵维度



## 5.与Numpy的协同操作





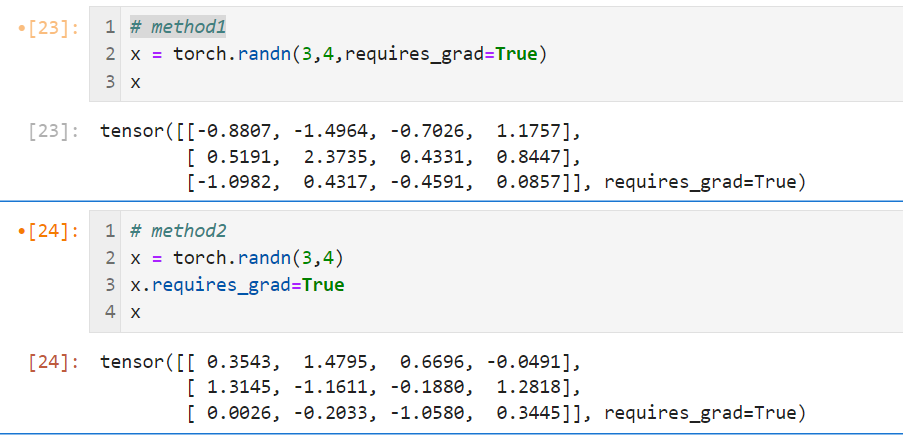
# P5 5.05-005 自动求导机制

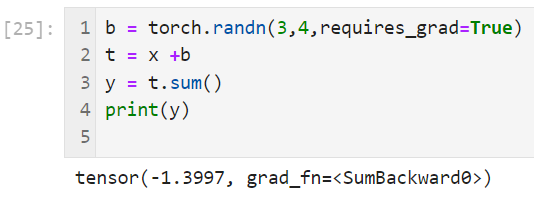
11:00

## 1.反向传播

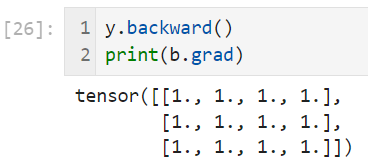
框架干的最厉害的一件事就是帮我们把返向传播全部计算好了

需要**求导**的，可以手动定义:

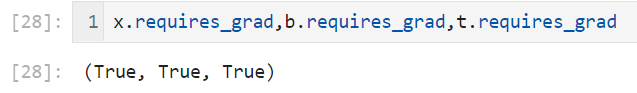




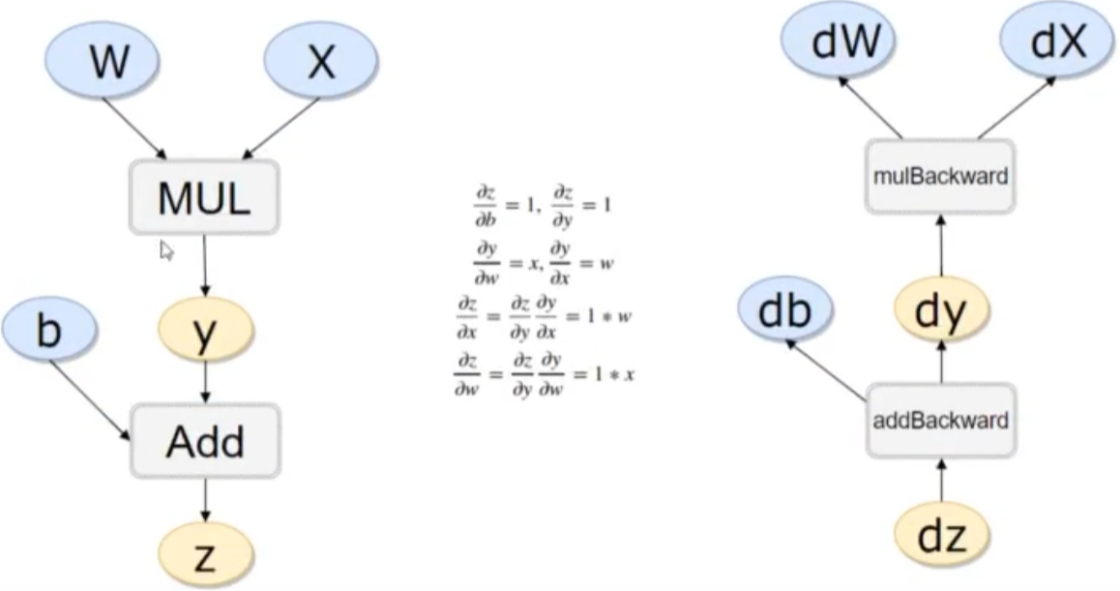
B.brad是导数

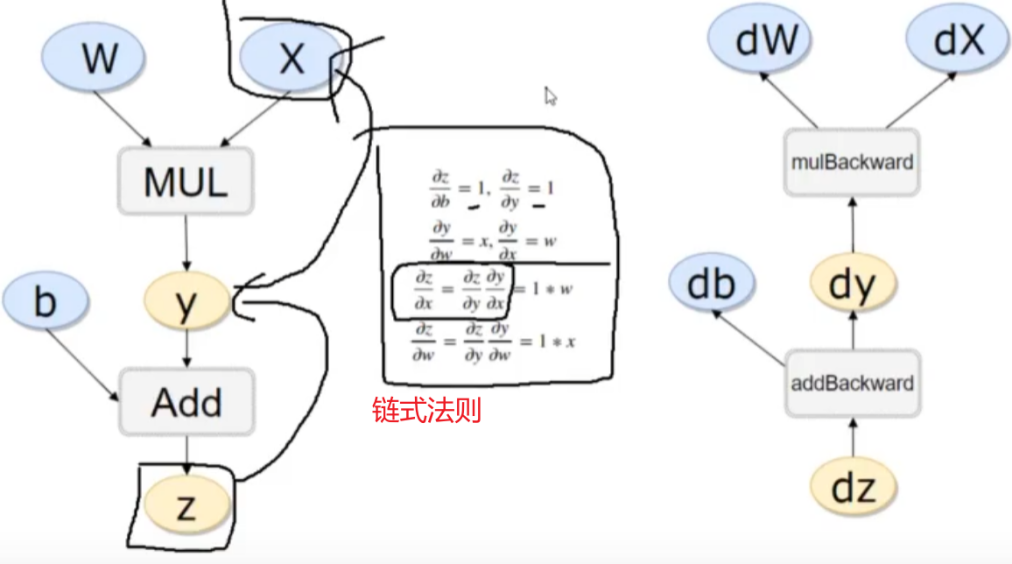


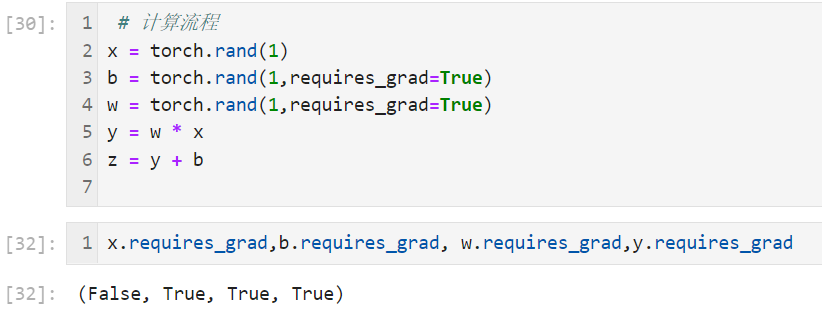
虽然没有指定t的requires grad但是需要用到它，也会默认的

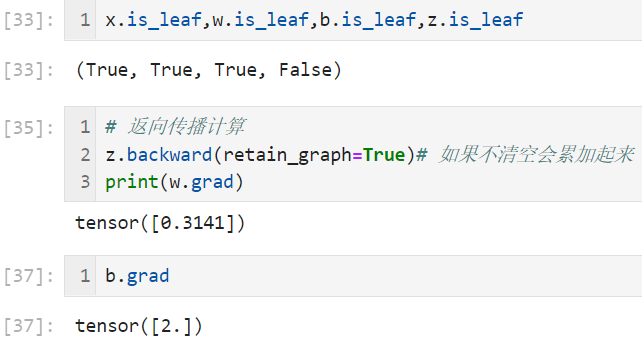


## 2.举个例子看一下







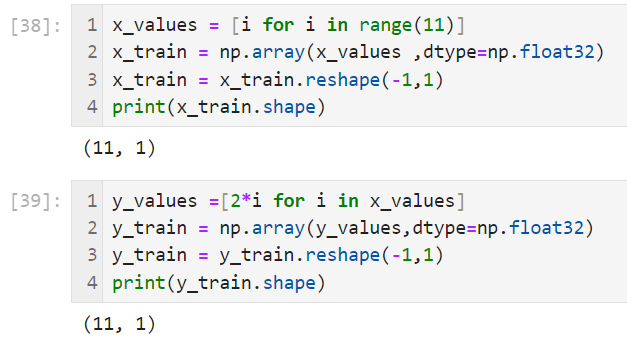


# P6 6. 06 - 006 线性回归DEMO-数据与参数配置

08:58

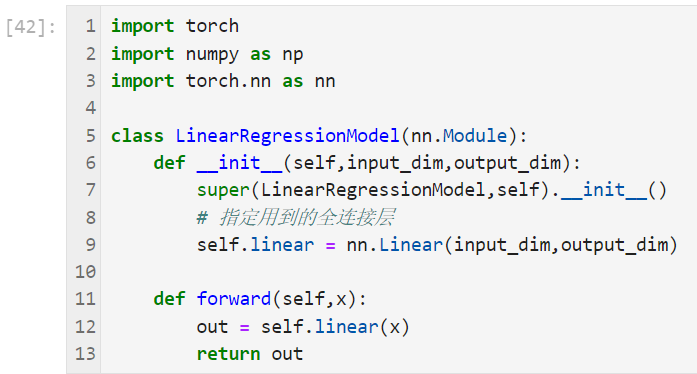
## 1.做一个线性回归试试水

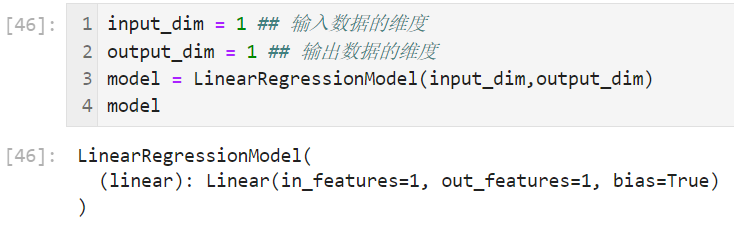
构造一组输入数据X和其对应的标签y



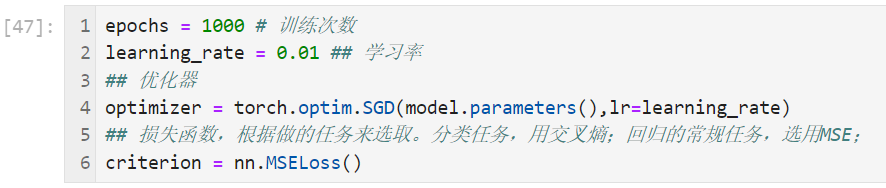
## 2.线性回归模型

其实线性回归就是一个不加激活函数的全连接层。





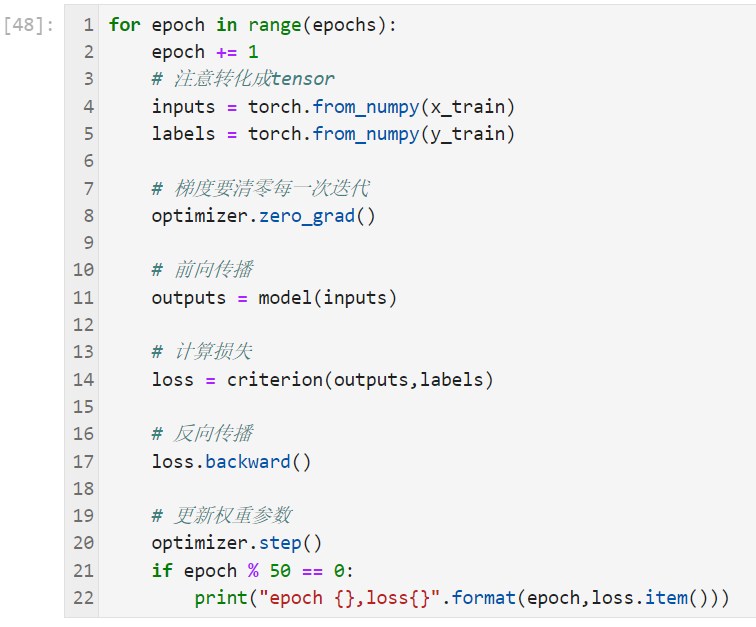
指定好参数和损失函数



# P7 7. 07 - 007 线性回归DEMO-训练回归模型

10:09

## 3.训练模型（接上章）



训练输出

epoch 50,loss0.01796736568212509

epoch 100,loss0.010247877798974514

epoch 150,loss0.005845008417963982

epoch 200,loss0.003333774395287037

epoch 250,loss0.0019014835124835372

epoch 300,loss0.001084525021724403

epoch 350,loss0.0006185758393257856

epoch 400,loss0.0003528103115968406

epoch 450,loss0.0002012289041886106

epoch 500,loss0.00011477460793685168

epoch 550,loss6.546464283019304e-05

epoch 600,loss3.733710400410928e-05

epoch 650,loss2.129636959580239e-05

epoch 700,loss1.214610892930068e-05

epoch 750,loss6.928933999006404e-06

epoch 800,loss3.951005510316463e-06

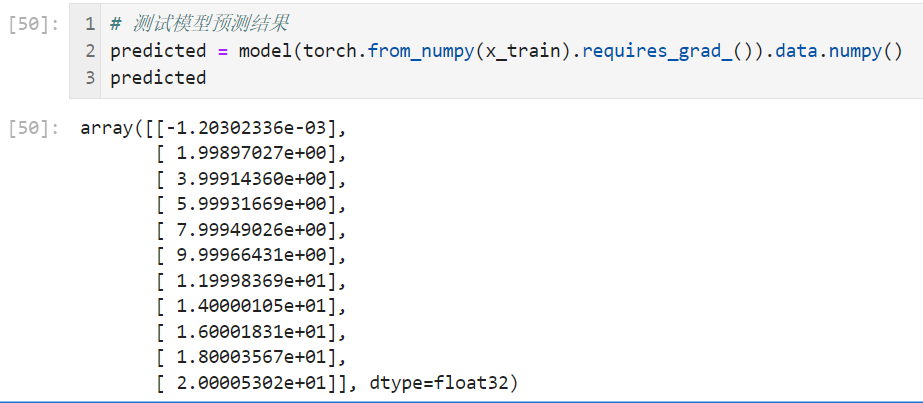
epoch 850,loss2.2537014956469648e-06

epoch 900,loss1.2853381576860556e-06

epoch 950,loss7.331084361794638e-07

epoch 1000,loss4.1838367792479403e-07

## 4.测试模型预测结果



## 5.模型的保存和读取

torch.save(model.state\_dict(),"model.pkl")

model.load\_state\_dict(torch.load("model.pkl"))



## 6.使用GPU进行训练

。只需要把数据和模型传入到cuda里面就可以了

##################################################

# 2.线性回归模型

#  其实线性回归就是一个不加激活函数的全连接层。

##################################################

class LinearRegressionModel(nn.Module):

    '''

      input\_dim: 输入数据的维度

      output\_dim: 输出数据的维度

    '''

    def \_\_init\_\_(self,input\_dim,output\_dim):

        super(LinearRegressionModel,self).\_\_init\_\_()

        # 指定用到的全连接层

        self.linear = nn.Linear(input\_dim,output\_dim)

    ## 前向传播函数。

    ## \*\*\*\*做任何项目，都是改这个函数的实现，其他的不变。\*\*\*\*

    def forward(self,x):

        out = self.linear(x)

        return out

#####---------------

input\_dim = 1 ## 输入数据的维度

output\_dim = 1 ## 输出数据的维度

model = LinearRegressionModel(input\_dim,output\_dim)

##使用GPU进行训练:只需要把数据和模型传入到cuda里面就可以了(1/2)

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

model.to(device)

####指定好参数和损失函数

# 损失函数，根据做的任务来选取。分类任务，用交叉熵；回归的常规任务，选用MSE；

criterion = nn.MSELoss()

## 学习率

learning\_rate = 0.01

# 优化器

optimizer = torch.optim.SGD(model.parameters(),lr=learning\_rate)

# 训练次数

epochs = 1000

for epoch in range(epochs):

    epoch += 1

    # 注意转化成tensor

    ##使用GPU进行训练:只需要把数据和模型传入到cuda里面就可以了(2/2)

    # inputs = torch.from\_numpy(x\_train) ## 用 cpu

    # labels = torch.from\_numpy(y\_train) ## 用 cpu

    inputs = torch.from\_numpy(x\_train).to(device)## 用 gpu

    labels = torch.from\_numpy(y\_train).to(device)## 用 gpu

    # 梯度要清零每一次迭代

    optimizer.zero\_grad()

    # 前向传播

    outputs = model(inputs)

    # 计算损失

    loss = criterion(outputs,labels)

    # 反向传播

    loss.backward()

    # 更新权重参数

    optimizer.step()

    if epoch % 50 == 0:

        print("epoch {},loss{}".format(epoch,loss.item()))

训练输出结果

epoch 50,loss0.163279190659523

epoch 100,loss0.09312834590673447

epoch 150,loss0.05311690270900726

epoch 200,loss0.030295919626951218

epoch 250,loss0.017279651015996933

epoch 300,loss0.009855687618255615

epoch 350,loss0.005621299147605896

epoch 400,loss0.00320617388933897

epoch 450,loss0.0018286893609911203

epoch 500,loss0.0010430163238197565

epoch 550,loss0.0005948917241767049

epoch 600,loss0.0003393067163415253

epoch 650,loss0.00019352862727828324

epoch 700,loss0.0001103798349504359

epoch 750,loss6.295608181972057e-05

epoch 800,loss3.590919732232578e-05

epoch 850,loss2.04806183319306e-05

epoch 900,loss1.168106246041134e-05

epoch 950,loss6.66270034344052e-06

epoch 1000,loss3.8000227959855692e-06

# P8 8. 08 - 008 补充：常见tensor格式

07:11

## 1.Tensor常见的形式有哪些

0: scalar

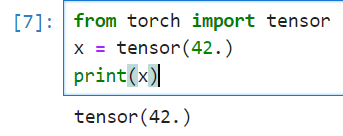
1: vector

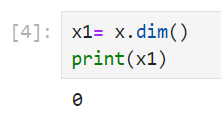
2: matrix

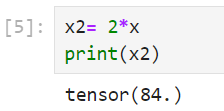
3:n-dimensional tensor

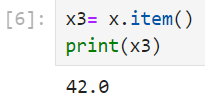
### 0.Scalar

# 通常就是一个数值





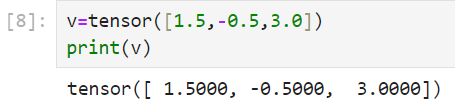


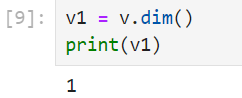


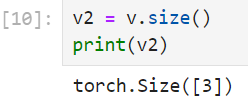
### 1. Vector

例如:[-5.,2.,0.]，在深度学习中通常指特征，例如词向量特征，某一维度特征等



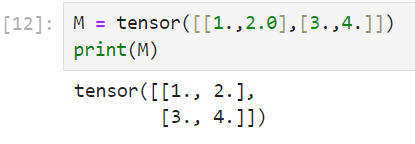


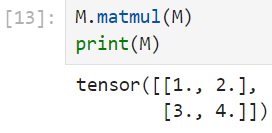


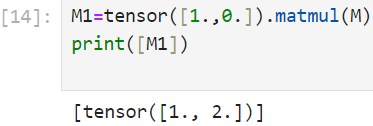


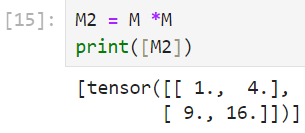
### 2.Matrix

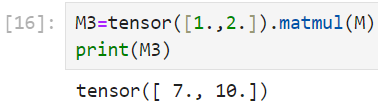
一般计算的都是矩阵，通常都是多维的



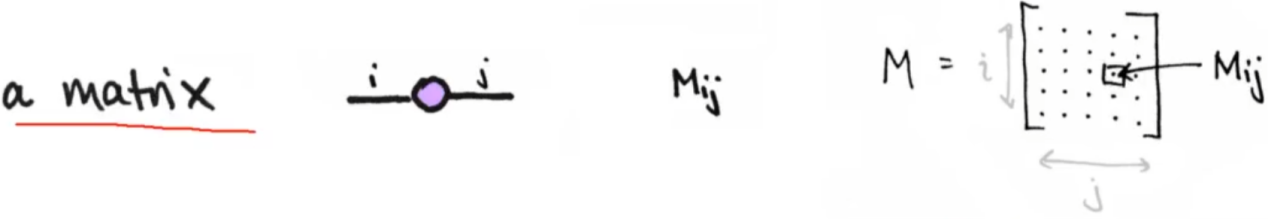


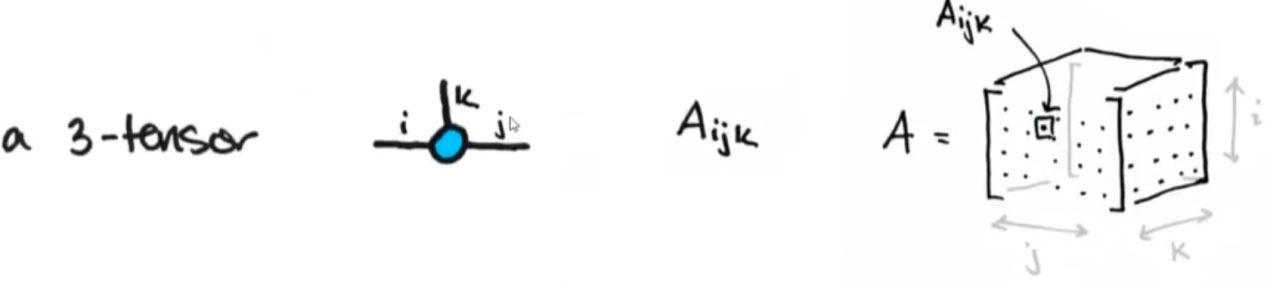


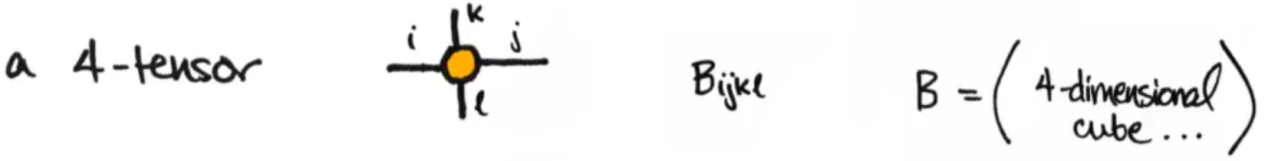








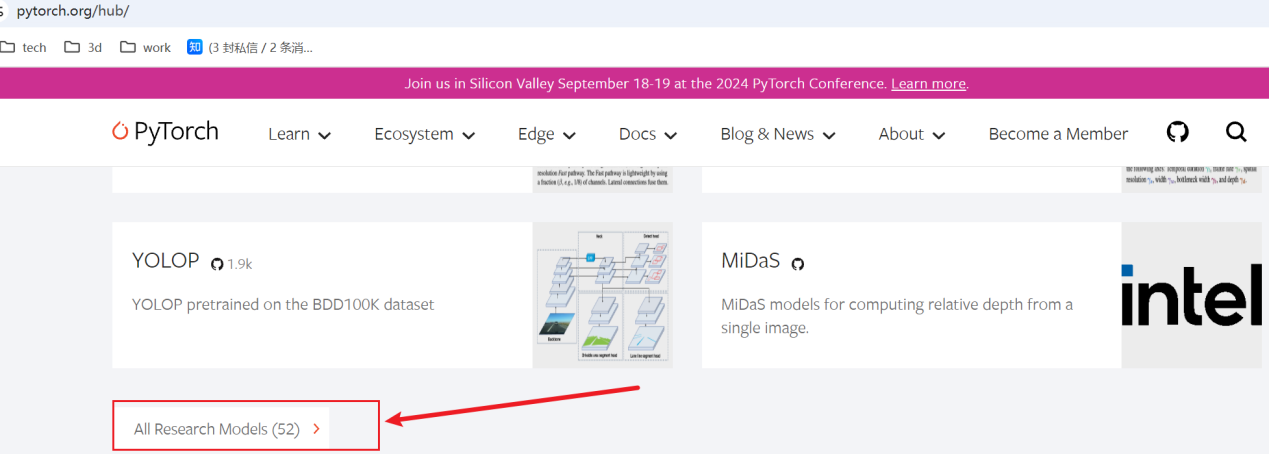


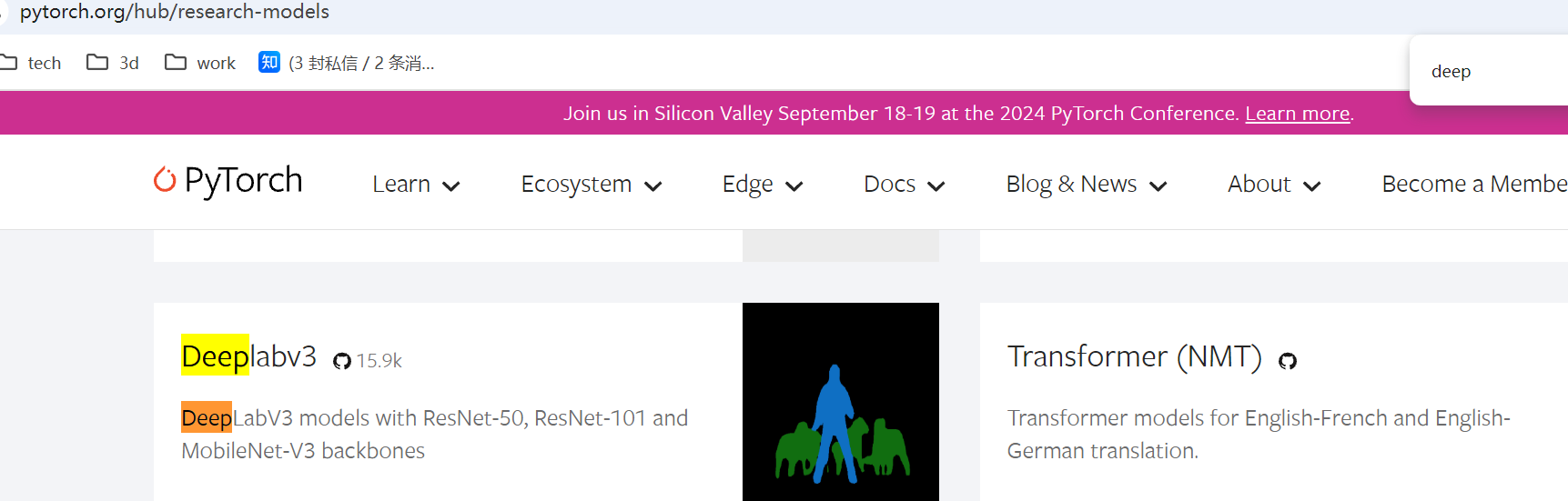


# P9 9. 09 - 009 补充：Hub模块简介

08:27

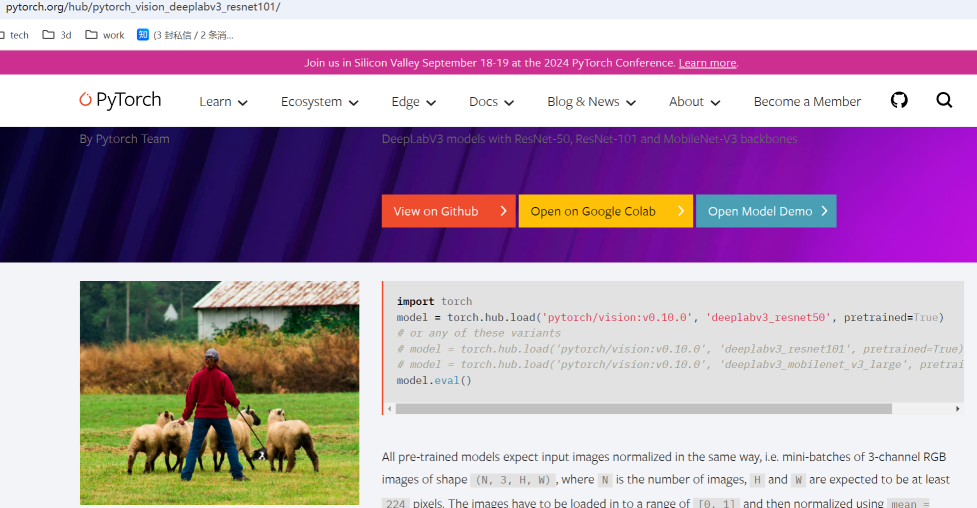
## 1）https://pytorch.org/hub/





1. deeplabv3\_resnet101

https://pytorch.org/hub/pytorch\_vision\_deeplabv3\_resnet101/



**import** torchmodel **=** torch.hub.load('pytorch/vision:v0.10.0', 'deeplabv3\_resnet50', pretrained**=**True)*# or any of these variants*

*# model = torch.hub.load('pytorch/vision:v0.10.0', 'deeplabv3\_resnet101', pretrained=True)*

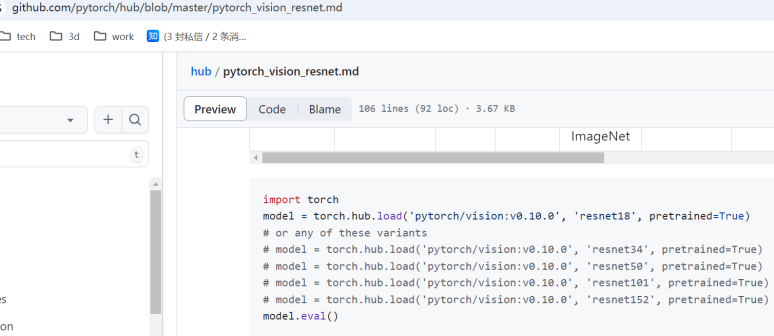
*# model = torch.hub.load('pytorch/vision:v0.10.0', 'deeplabv3\_mobilenet\_v3\_large', pretrained=True)*model.eval()

3)GITHUB地址

GITHUB: <https://github.com/pytorch/hub>

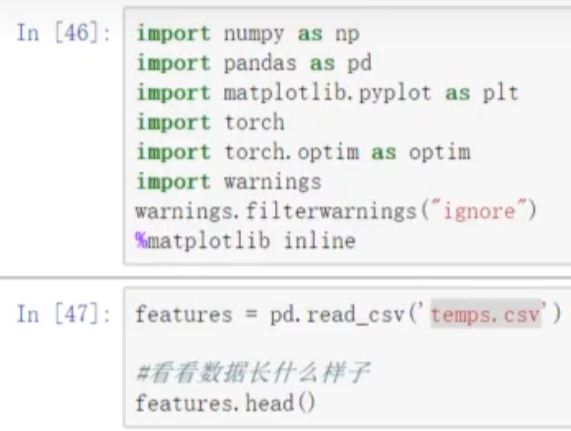
模型: <https://pytorch.org/hub/research-models>

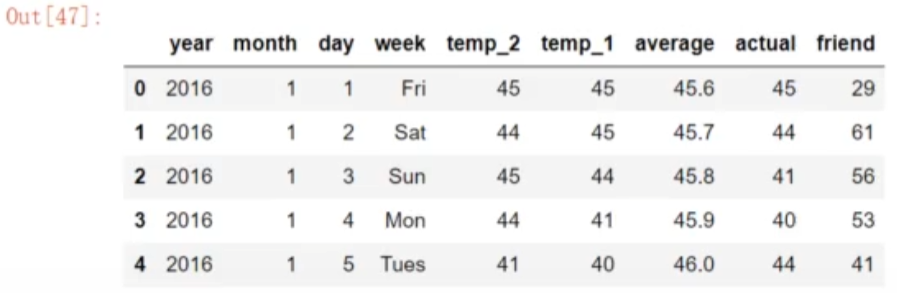
https://github.com/pytorch/hub/blob/master/pytorch\_vision\_resnet.md



# P10 10. 10 - 010 气温数据集与任务介绍

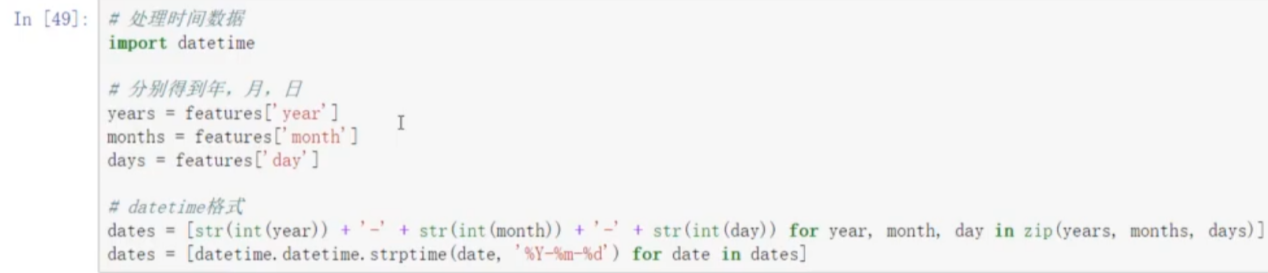
06:43

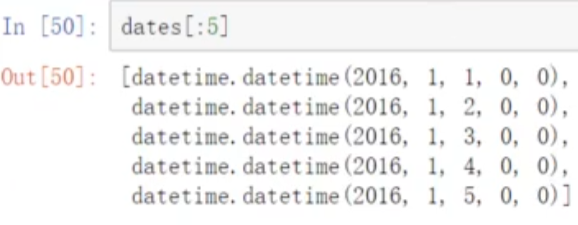


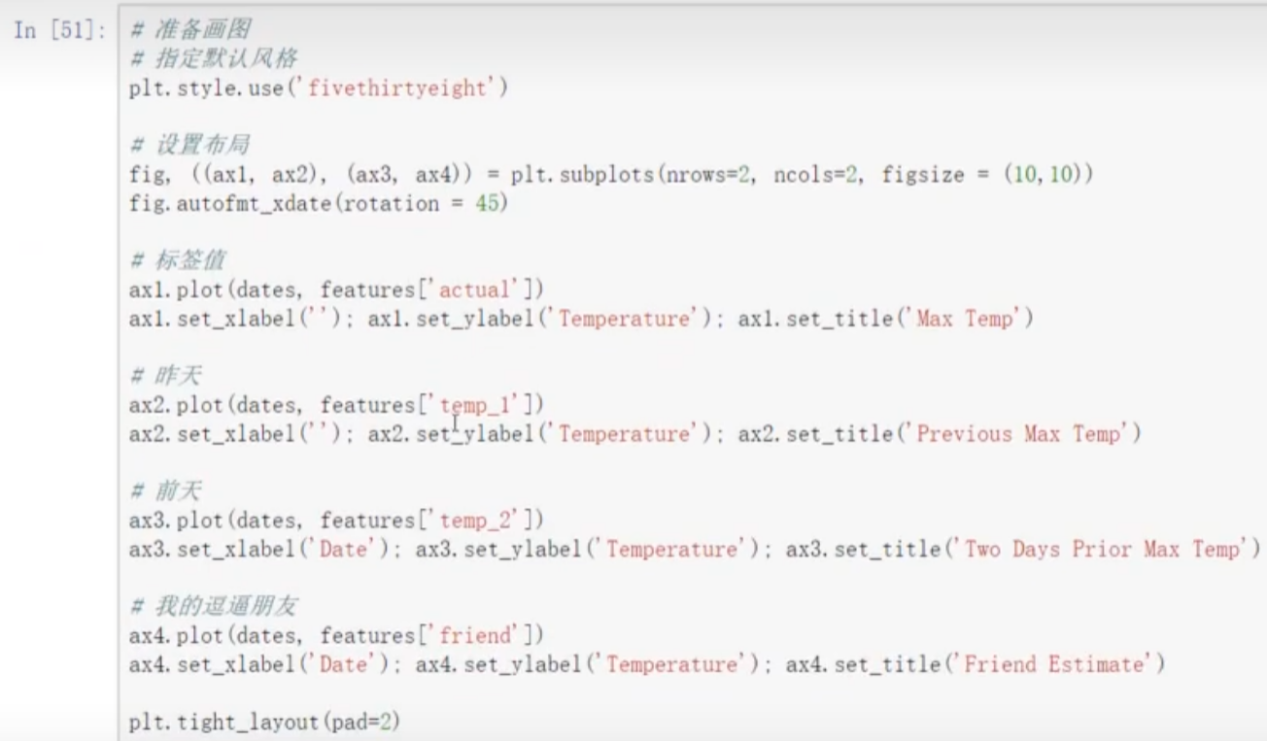


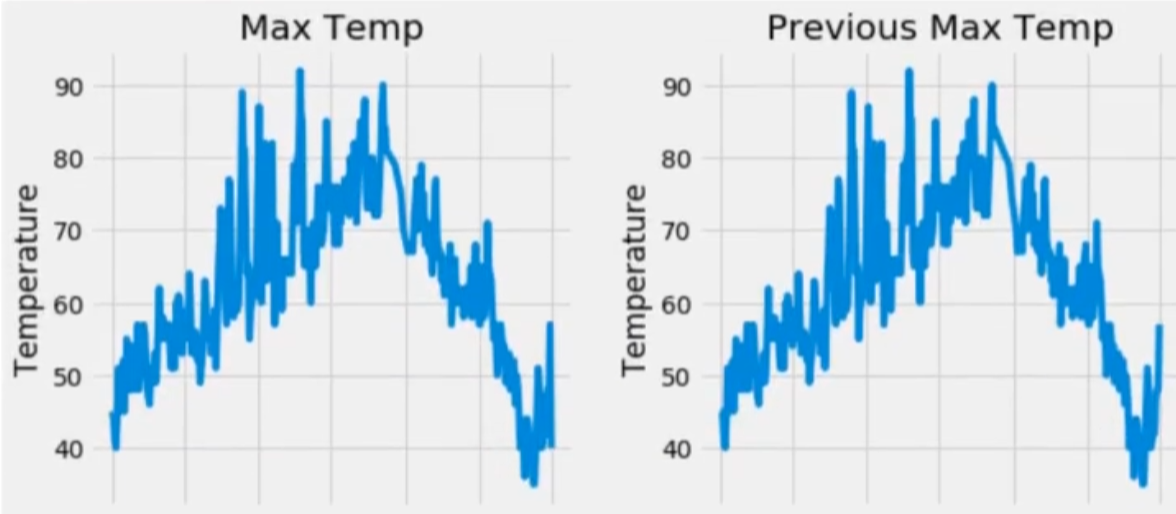


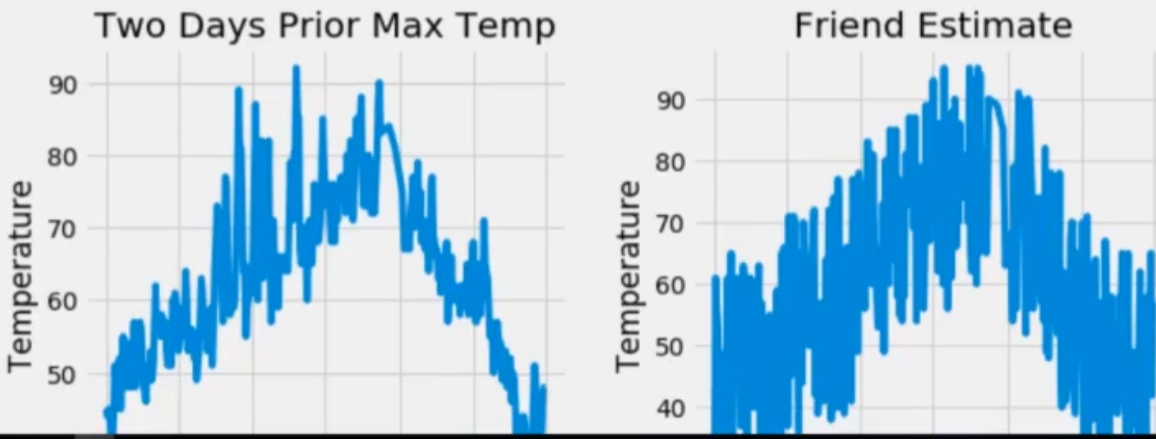


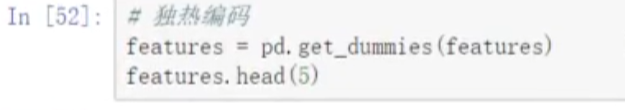


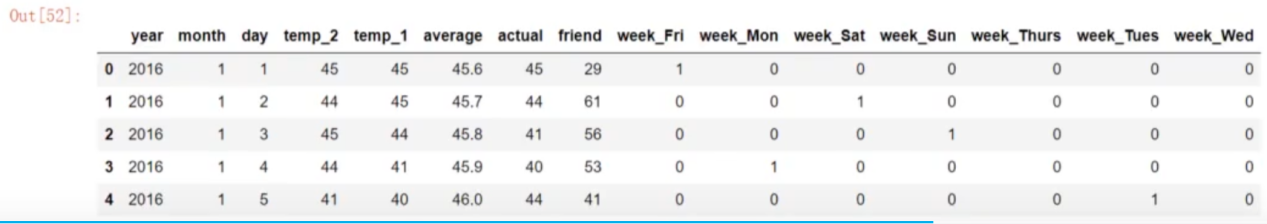






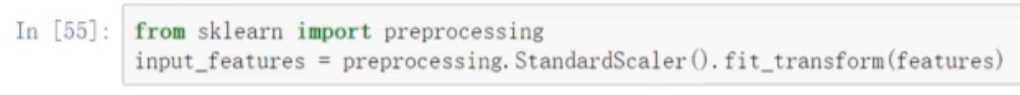












# P11 11. 11 - 011 按建模顺序构建完成网络架构

11:40

# P12 12. 12 - 012 简化代码训练网络模型

11:05

# P13 13. 13 - 013 分类任务概述

05:14

P14 14. 14 - 014 构建分类网络模型

09:41

P15 15. 15 - 015 DataSet模块介绍与应用方法

10:13

P16 16. 16 - 016 卷积神经网络应用领域

07:26

P17 17. 17 - 017 卷积的作用

09:25

P18 18. 18 - 018 卷积特征值计算方法

08:08

P19 19. 19 - 019 得到特征图表示

07:00

P20 20. 20 - 020 步长与卷积核大小对结果的影响

08:12

P21

21. 21 - 021 边缘填充方法

06:32

P22

22. 22 - 022 特征图尺寸计算与参数共享

07:03

P23

23. 23 - 023 池化层的作用

05:39

P24

24. 24 - 024 整体网络架构

06:21

P25

25. 25 - 025 VGG网络架构

06:17

P26

26. 26 - 026 残差网络Resnet

07:42

P27

27. 27 - 027 感受野的作用

05:47

P28

28. 28 - 028 卷积网络参数定义

07:22

P29

29. 29 - 029 网络流程解读

07:27

P30

30. 30 - 030 Vision模块功能解读

05:12

P31

31. 31 - 031 分类任务数据集定义与配置

06:28

P32

32. 32 - 032 图像增强的作用

04:52

P33

33. 33 - 033 数据预处理与数据增强模块

09:26

P34

34. 34 - 034 Batch数据制作

08:39

P35

35. 35 - 035 迁移学习的目标

05:33

P36

36. 36 - 036 迁移学习策略

07:12

P37

37. 37 - 037 加载训练好的网络模型

09:55

P38

38. 38 - 038 优化器模块配置

05:15

P39

39. 39 - 039 实现训练模块

08:17

P40

40. 40 - 040 训练结果与模型保存

09:32

P41

41. 41 - 041 加载模型对测试数据进行预测

09:11

P42

42. 42 - 042 额外补充-Resnet论文解读

11:48

P43

43. 43 - 043 额外补充-Resnet网络架构解读

08:27

P44

44. 44 - 044 RNN网络架构解读

11:28

P45

45. 45 - 045 词向量模型通俗解释

08:15

P46

46. 46 - 046 模型整体框架

10:11

P47

47. 47 - 047 训练数据构建

05:12

P48

48. 48 - 048 CBOW与Skip-gram模型

08:21

P49

49. 49 - 049 负采样方案

07:41

P50

50. 50 - 050 任务目标与数据简介

07:19

P51

51. 51 - 051 RNN模型所需输入格式解析

06:55

P52

52. 52 - 052 项目配置参数设置

10:28

P53

53. 53 - 053 新闻数据读取与预处理方法

08:08

P54

54. 54 - 054 LSTM网络模块定义与参数解析

09:37

P55

55. 55 - 055 训练LSTM文本分类模型

08:56

P56

56. 56 - 056 Tensorboardx可视化展示模块搭建

09:18

P57

57. 57 - 057 CNN应用于文本任务原理解析

10:47

P58

58. 58 - 058 网络模型架构与效果展示

10:59

P59

59. 59 - 059 对抗生成网络通俗解释

08:26

P60

60. 60 - 060 GAN网络组成

05:15

P61

61. 61 - 061 损失函数解释说明

10:07

P62

62. 62 - 062 数据读取模块

08:28

P63

63. 63 - 063 生成与判别网络定义

08:40

P64

64. 64 - 064 CycleGan网络所需数据

06:51

P65

65. 65 - 065 CycleGan整体网络架构

10:04

P66

66. 66 - 066 PatchGan判别网络原理

04:41

P67

67. 67 - 067 Cycle开源项目简介

07:08

P68

68. 68 - 068 数据读取与预处理操作

10:18

P69

69. 69 - 069 生成网络模块构造

12:13

P70

70. 70 - 070 判别网络模块构造

05:03

P71

71. 71 - 071 损失函数：identity loss计算方法

09:14

P72

72. 72 - 072 生成与判别损失函数指定

11:41

P73

73. 73 - 073 额外补充：VISDOM可视化配置

05:55

P74

74. 74 - 074 OCR文字识别要完成的任务

06:30

P75

75. 75 - 075 CTPN文字检测网络概述

08:06

P76

76. 76 - 076 序列网络的作用

09:21

P77

77. 77 - 077 输出结果含义解析

07:10

P78

78. 78 - 078 CTPN细节概述

09:07

P79

79. 79 - 079 CRNN识别网络架构

06:17

P80

80. 80 - 080 CTC模块的作用

04:30

P81

81. 81 - 081 OCR文字检测识别项目效果展示

04:22

P82

82. 82 - 082 OCR文字检测识别项目效果展示

06:50

P83

83. 83 - 083 检测模块候选框生成

08:08

P84

84. 84 - 084 候选框标签制作

08:24

P85

85. 85 - 085 整体网络所需模块

04:57

P86

86. 86 - 086 网络架构各模块完成的任务解读

08:40

P87

87. 87 - 087 CRNN识别模块所需数据与标签

05:13

P88

88. 88 - 088 识别模块网络架构解读

10:42

P89

89. 89 - 089 3D卷积原理解读

07:44

P90

90. 90 - 090 UCF101动作识别数据集简介

06:03

P91

91. 91 - 091 测试效果与项目配置

12:03

P92

92. 92 - 092 视频数据预处理方法

07:25

P93

93. 93 - 093 数据Batch制作方法

09:03

P94

94. 94 - 094 3D卷积网络所涉及模块

07:52

P95

95. 95 - 095 训练网络模型

08:33

P96

96. 96 - 096 BERT任务目标概述

05:28

P97

97. 97 - 097 传统解决方案遇到的问题

11:10

P98

98. 98 - 098 注意力机制的作用

06:58

P99

99. 99 - 099 self-attention计算方法

11:26

P100

100. 100 - 100 特征分配与softmax机制

09:21

P101

101. 101 - 101 Multi-head的作用

09:10

P102

102. 102 - 102 位置编码与多层堆叠

07:19

P103

103. 103 - 103 transformer整体架构梳理

10:58

P104

104. 104 - 104 BERT模型训练方法

09:38

P105

105. 105 - 105 训练实例

09:48

P106

106. 106 - 106 BERT开源项目简介

07:36

P107

107. 107 - 107 项目参数配置

12:09

P108

108. 108 - 108 数据读取模块

07:41

P109

109. 109 - 109 数据预处理模块

09:38

P110

110. 110 - 110 tfrecord制作

11:36

P111

111. 111 - 111 Embedding层的作用

07:30

P112

112. 112 - 112 加入额外编码特征

09:24

P113

113. 113 - 113 加入位置编码特征

05:13

P114

114. 114 - 114 mask机制

08:51

P115

115. 115 - 115 构建QKV矩阵

12:39

P116

116. 116 - 116 完成Transformer模块构建

09:57

P117

117. 117 - 117 训练BERT模型

08:52

P118

118. 118 - 118 项目配置与环境概述

06:53

P119

119. 119 - 119 数据读取与预处理

05:46

P120

120. 120 - 120 网络结构定义

07:02

P121

121. 121 - 121 训练网络模型

08:20

P122

122. 122 - 122 项目模板各模块概述

08:45

P123

123. 123 - 123 各模块配置参数解析

09:16

P124

124. 124 - 124 数据读取与预处理模块功能解读

11:46

P125

125. 125 - 125 模型架构模块

06:47

P126

126. 126 - 126 训练模块功能

11:24

P127

127. 127 - 127 训练结果可视化展示模块

07:20

P128

128. 128 - 128 模块应用与BenckMark解读

11:54