# 深度学习系统

Tensorflow、Pytorch和MindSpore都是当前深度学习领域最受欢迎和广泛应用的深度学习框架，它们都支持图形化计算图，可以方便地搭建神经网络模型，同时也提供了丰富的工具和API，方便用户进行模型训练、评估和部署。

## TensorFlow

### 什么是TensorFlow

TensorFlow是由Google公司在2015年推出的开源深度学习框架，是最受关注的深度学习框架之一。TensorFlow的名字源于其核心数据结构Tensor（张量）和流动的计算图（Flow）。

TensorFlow的发展历程：

* 2011年，Google Brain团队在其内部使用的深度学习基础架构 DistBelief 框架上进行改进，逐渐形成了 TensorFlow，此时TensorFlow还没有开源。
* 2015年，TensorFlow 正式开源，此时处在 0.x 的非正式版本。
* 2016年12月期间，TensorFlow 逐渐支持了分布式、移动设备 与Windows 系统。
* 2017年，TensorFlow 正式发布了 1.0.0 版本。
* 2019年，TensorFlow 逐步加入了 TPU、张量处理单元等工具。
* 2019年10月，TensorFlow2.0 正式发布，内部嵌入了 Keras。

截至本书书写时，TensorFlow的最新稳定版本为2.12版本。

### TensorFlow的核心概念与架构

TensorFlow的核心概念是“计算图”（Computation Graph）和“张量”（Tensor）。在TensorFlow中，所有的计算都被表示为一个有向无环图（DAG）的形式，这个图被称为计算图，计算图中的节点表示计算操作，边表示数据流动的方向。张量则是TensorFlow中的基本数据类型，它可以被看作是多维数组或矩阵。

TensorFlow的系统结构以C API为界，将整个系统分为前端和后端两个子系统。

1. 前端系统：提供编程模型，负责构造计算图
2. 后端系统：提供运行时环境，负责执行计算图

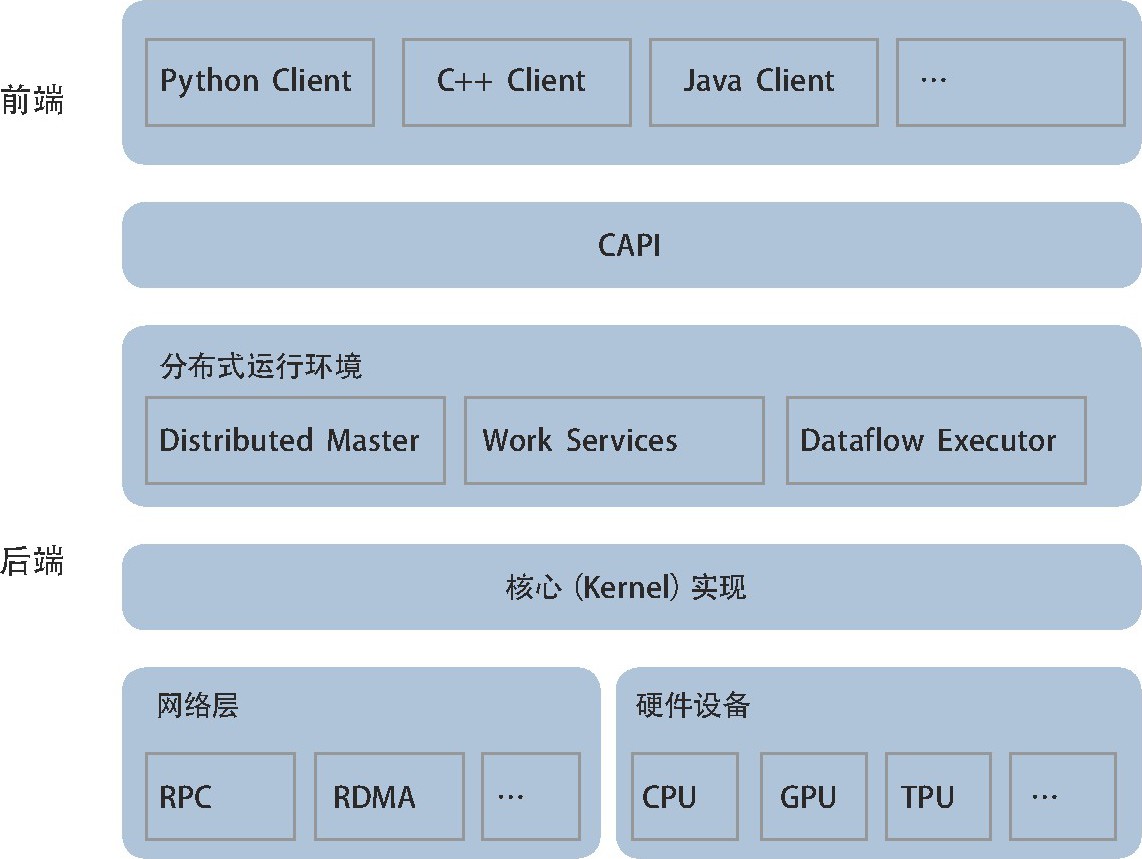


图 8‑1 TensorFlow系统架构图

TensorFlow的计算图和张量的特点使其具有高度的可移植性和可扩展性，可以运行在不同的硬件和平台上，例如CPU、GPU、TPU等，也可以在不同的操作系统上运行。同时，TensorFlow也具有丰富的社区和生态系统，支持各种第三方库的集成和使用，如TensorFlow Hub、TensorFlow Serving等。

### TensorFlow示例代码

代码基于tensorflow2的环境编写。针对minist手写数字识别数据集进行梯度下降算法的实现。

1. 导入依赖包
2. import tensorflow as tf
3. from tensorflow import keras
4. from keras import layers, optimizers, datasets
5. 加载数据集
6. #加载minist数据集，分成训练集和测试集，每个样本包含图像和标签
7. (x, y), (x\_val, y\_val) = datasets.mnist.load\_data()
8. print('datasets', x.shape, y.shape, x.min(), y.min())
9. 处理数据
10. #训练集图像数据归一化到0-1之前
11. x = tf.convert\_to\_tensor(x, dtype=tf.float32) / 255.
12. #构建数据集对象
13. db = tf.data.Dataset.from\_tensor\_slices((x, y))
14. #批量训练，并行计算一次32个样本、所有数据集迭代20次
15. db = db.batch(32).repeat(10)
16. 构建网络模型
17. # 构建Sequential窗口，共3层网络，前一个网络的输出作为后一个网络的输入
18. model = keras.Sequential([
19. layers.Dense(256, activation='relu'),
20. layers.Dense(128, activation='relu'),
21. layers.Dense(10)
22. ])
23. # 指定输入大小
24. model.build(input\_shape=(None, 28\*28))
25. # 打印出网络的结构和参数量
26. model.summary()
27. # optimizers用于更新梯度下降算法参数，0.01为学习率
28. optimizer = optimizers.SGD(lr=0.01)
29. # acc\_meter用于计算正确率
30. acc\_meter = keras.metrics.Accuracy()
31. # 创建参数文件
32. summary\_writer = tf.summary.create\_file\_writer('./tf\_log')
33. 进行训练测试
34. #循环数据集
35. for step, (xx, yy) in enumerate(db):
36. with tf.GradientTape() as tape:
37. #图像样本大小重置(-1, 28\*28)
38. xx = tf.reshape(xx, (-1, 28\*28))
39. out = model(xx)
40. y\_onehot = tf.one\_hot(yy, depth=10)
41. loss = tf.square(out-y\_onehot)
42. loss = tf.reduce\_sum(loss/xx.shape[0])
43. acc\_meter.update\_state(tf.argmax(out, axis=1), yy)
44. grads = tape.gradient(loss, model.trainable\_variables)
45. optimizer.apply\_gradients(zip(grads, model.trainable\_variables))
46. with summary\_writer.as\_default():
47. tf.summary.scalar('train-loss', float(loss), step=step)
48. tf.summary.scalar('test-acc', acc\_meter.result().numpy(), step=step)
49. if step % 1000 == 0:
50. print(step, 'loss:', float(loss), end=' ')
51. print('acc:', acc\_meter.result().numpy())
52. acc\_meter.reset\_states()

经过训练，最终在minist手写数字识别数据集上能够表现出0.97的正确率。

## Pytorch

### 什么是Pytorch

Pytorch是由Facebook开发的一款基于Python的深度学习框架。它于2016年首次发布，其前身是Torch框架。

Pytorch最初的设计目的是为了能够更好地进行动态计算图，即使在运行中改变计算图，也能够自动地构建梯度。这使得Pytorch在一些需要动态计算图的领域，如自然语言处理和图像处理方面，具有很大的优势。另外，Pytorch也支持静态计算图，这是由于其后端使用了TorchScript进行编译。同时，Pytorch也提供了强大的GPU加速能力。

### Pytorch的核心概念与架构

PyTorch是一个基于Python的深度学习框架，其核心概念包括张量（Tensor）、自动微分（Autograd）和动态计算图（Dynamic Computational Graph）。

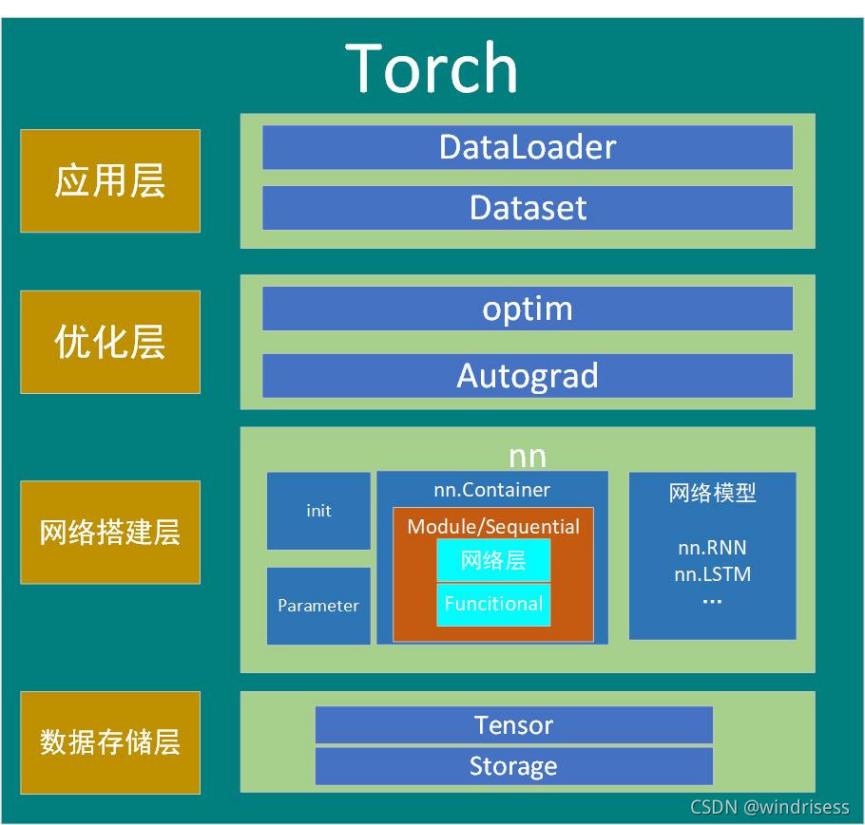


图 8‑2 Pytorch系统架构图

### Pytorch示例代码

代码基于torch1.7的环境编写。针对minist手写数字识别数据集进行梯度下降算法的实现。

1. 引入依赖包
2. import torch
3. import torchvision
4. from torch.utils.data import DataLoader
5. import torch.nn as nn
6. import torch.nn.functional as F
7. import torch.optim as optim
8. 设置参数
9. n\_epochs = 2
10. batch\_size\_train = 64
11. batch\_size\_test = 1000
12. learning\_rate = 0.01
13. momentum = 0.5
14. log\_interval = 10
15. random\_seed = 1
16. torch.manual\_seed(random\_seed)
17. 导入数据
18. train\_loader = torch.utils.data.DataLoader(
19. torchvision.datasets.MNIST('./data/', train=True, download=True,
20. transform=torchvision.transforms.Compose([
21. torchvision.transforms.ToTensor(),
22. torchvision.transforms.Normalize(
23. (0.1307,), (0.3081,))
24. ])),
25. batch\_size=batch\_size\_train,
26. shuffle=True)
27. test\_loader = torch.utils.data.DataLoader(
28. torchvision.datasets.MNIST('./data/', train=False, download=True,
29. transform=torchvision.transforms.Compose([
30. torchvision.transforms.ToTensor(),
31. torchvision.transforms.Normalize(
32. (0.1307,), (0.3081,))
33. ])),
34. batch\_size=batch\_size\_test,
35. shuffle=True)
36. 创建网络模型
37. class Net(nn.Module):
38. def \_\_init\_\_(self):
39. super(Net, self).\_\_init\_\_()
40. self.conv1 = nn.Conv2d(1, 10, kernel\_size=5)
41. self.conv2 = nn.Conv2d(10, 20, kernel\_size=5)
42. self.conv2\_drop = nn.Dropout2d()
43. self.fc1 = nn.Linear(320, 50)
44. self.fc2 = nn.Linear(50, 10)
45. def forward(self, x):
46. x = F.relu(F.max\_pool2d(self.conv1(x), 2))
47. x = F.relu(F.max\_pool2d(self.conv2\_drop(self.conv2(x)), 2))
48. x = x.view(-1, 320)
49. x = F.relu(self.fc1(x))
50. x = F.dropout(x, training=self.training)
51. x = self.fc2(x)
52. return F.log\_softmax(x, dim=1)
53. 初始化网络和优化器
54. network = Net()
55. optimizer = optim.SGD(network.parameters(), lr=learning\_rate,
56. momentum=momentum)
57. 记录训练测试数据
58. train\_losses = []
59. train\_counter = []
60. test\_losses = []
61. test\_counter = [i\*len(train\_loader.dataset) for i in range(1, n\_epochs + 1)]
62. 训练和测试
63. for epoch in range(1, n\_epochs + 1):
64. network.train()
65. for batch\_idx, (data, target) in enumerate(train\_loader):
66. optimizer.zero\_grad()
67. output = network(data)
68. loss = F.nll\_loss(output, target)
69. loss.backward()
70. optimizer.step()
71. if batch\_idx % log\_interval == 0:
72. if batch\_idx \* len(data) % 1000 == 0:
73. print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
74. epoch,
75. batch\_idx \* len(data),
76. len(train\_loader.dataset),
77. 100. \* batch\_idx / len(train\_loader),
78. loss.item())
79. )
80. train\_losses.append(loss.item())
81. train\_counter.append((batch\_idx\*64) + ((epoch-1)\*len(train\_loader.dataset)))
82. torch.save(network.state\_dict(), './model.pth')
83. torch.save(optimizer.state\_dict(), './optimizer.pth')
84. network.eval()
85. test\_loss = 0
86. correct = 0
87. with torch.no\_grad():
88. for data, target in test\_loader:
89. output = network(data)
90. test\_loss += F.nll\_loss(output, target, size\_average=False).item()
91. pred = output.data.max(1, keepdim=True)[1]
92. correct += pred.eq(target.data.view\_as(pred)).sum()
93. test\_loss /= len(test\_loader.dataset)
94. test\_losses.append(test\_loss)
95. print('\nTest set: Avg. loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
96. test\_loss, correct, len(test\_loader.dataset),
97. 100. \* correct / len(test\_loader.dataset)))
98. 绘制训练测试数据
99. import matplotlib.pyplot as plt
100. fig = plt.figure()
101. plt.plot(train\_counter, train\_losses, color='blue')
102. plt.scatter(test\_counter, test\_losses, color='red')
103. plt.legend(['Train Loss', 'Test Loss'], loc='upper right')
104. plt.xlabel('number of training examples seen')
105. plt.ylabel('negative log likelihood loss')
106. plt.show()

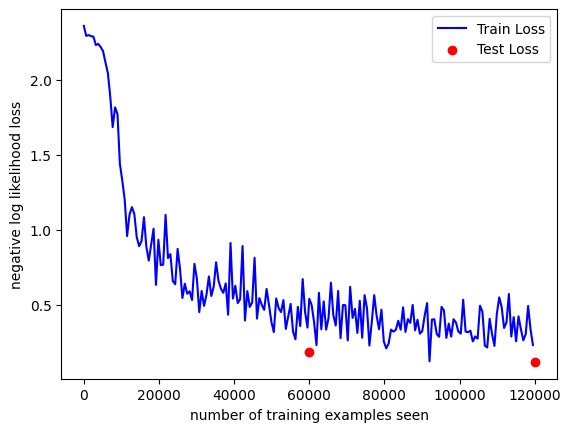


图 8‑3 Pytorch在Minist数据集上的表现

## MindSpore

### 什么是MindSpore

MindSpore是华为于2019年发布的深度学习框架，并于2020年宣布开源。该框架是面向“端-边-云”全场景设计的AI框架，旨在弥合AI算法研究与生产部署之间的鸿沟。

### MindSpore的核心概念与架构

MindSpore在算法研究阶段，为开发者提供动静统一的编程体验以提升算法的开发效率；在生产阶段，自动并行可以极大加快分布式训练的开发和调试效率，同时充分挖掘异构硬件的算力；在部署阶段，基于“端-边-云”统一架构，应对企业级部署和安全可信方面的挑战。

MindSpore整体架构分为四层：

1. 模型层，为用户提供开箱即用的功能，该层主要包含预置的模型和开发套件，以及图神经网络（GNN）、深度概率编程、科学计算库等热点研究领域拓展库；
2. 表达层 （MindExpression），为用户提供AI模型开发、训练、推理的接口，支持用户用原生 Python语法开发和调试神经网络，其特有的动静态图统一能力使开发者可以兼顾开发效率和执行性能，同时该层在生产和部署阶段提供全场景统一的C++/Python接口；
3. 编译优化（MindCompiler），作为AI框架的核心，以全场景统一中间表达（MindIR）为媒介，将前端表达编译成执行效率更高的底层语言，同时进行全局性能优化，包括自动微分、代数化简等硬件无关优化，以及图算融合、算子生成等硬件相关优化；
4. 运行时，按照上层编译优化的结果对接并调用底层硬件算子，同时通过“端-边-云”统一的运行时架构，支持包括联邦学习在内的“端-边-云”AI协同。

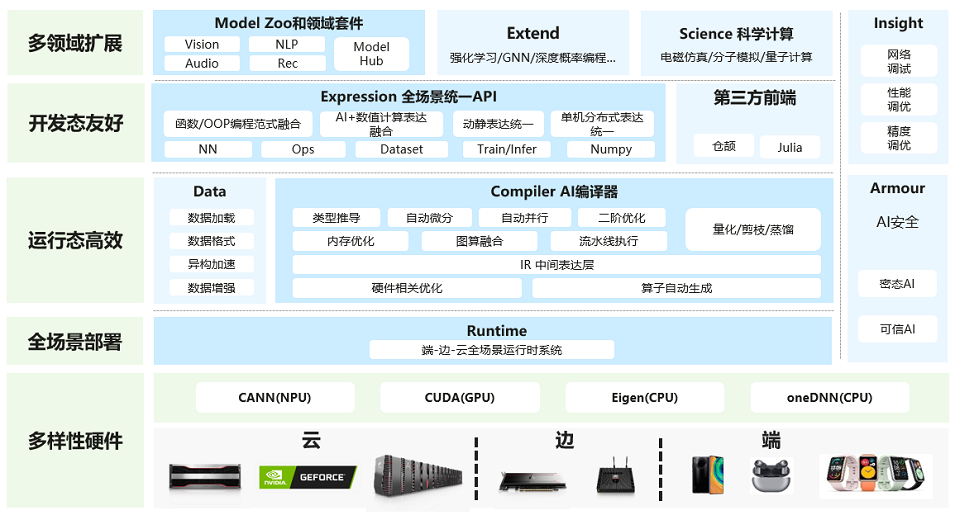


图 8‑4 MindSpore系统架构图

### MindSpore的使用和示例代码

1. 导入模块
2. import os
3. # os.environ['DEVICE\_ID'] = '0'
4. import mindspore as ms
5. import mindspore.context as context
6. import mindspore.dataset.transforms.c\_transforms as C
7. import mindspore.dataset.vision.c\_transforms as CV
8. from mindspore import nn
9. from mindspore.train import Model
10. from mindspore.train.callback import LossMonitor
11. context.set\_context(mode=context.GRAPH\_MODE, device\_target='Ascend') # Ascend, CPU, GPU
12. 数据处理
13. def create\_dataset(data\_dir, training=True,
14. batch\_size=32, resize=(32, 32),
15. rescale=1/(255\*0.3081), shift=-0.1307/0.3081,
16. buffer\_size=64):
17. data\_train = os.path.join(data\_dir, 'train') # train set
18. data\_test = os.path.join(data\_dir, 'test') # test set
19. ds = ms.dataset.MnistDataset(
20. data\_train if training else data\_test)
21. ds = ds.map(input\_columns=["image"],
22. operations=[CV.Resize(resize),
23. CV.Rescale(rescale, shift),
24. CV.HWC2CHW()])
25. ds = ds.map(input\_columns=["label"],
26. operations=C.TypeCast(ms.int32))
27. ds = ds.shuffle(buffer\_size=buffer\_size).batch(
28. batch\_size, drop\_remainder=True)
29. return ds
30. 定义模型
31. class LeNet5(nn.Cell):
32. def \_\_init\_\_(self):
33. super(LeNet5, self).\_\_init\_\_()
34. self.conv1 = nn.Conv2d(1, 6, 5, stride=1, pad\_mode='valid')
35. self.conv2 = nn.Conv2d(6, 16, 5, stride=1, pad\_mode='valid')
36. self.relu = nn.ReLU()
37. self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)
38. self.flatten = nn.Flatten()
39. self.fc1 = nn.Dense(400, 120)
40. self.fc2 = nn.Dense(120, 84)
41. self.fc3 = nn.Dense(84, 10)
42. def construct(self, x):
43. x = self.relu(self.conv1(x))
44. x = self.pool(x)
45. x = self.relu(self.conv2(x))
46. x = self.pool(x)
47. x = self.flatten(x)
48. x = self.fc1(x)
49. x = self.fc2(x)
50. x = self.fc3(x)
51. return x
52. 训练
53. def train(data\_dir, lr=0.01, momentum=0.9, num\_epochs=3):
54. ds\_train = create\_dataset(data\_dir)
55. ds\_eval = create\_dataset(data\_dir, training=False)
56. net = LeNet5()
57. loss = nn.loss.SoftmaxCrossEntropyWithLogits(
58. sparse=True, reduction='mean')
59. opt = nn.Momentum(
60. net.trainable\_params(), lr, momentum)
61. loss\_cb = LossMonitor(
62. per\_print\_times=ds\_train.get\_dataset\_size())
63. model = Model(net, loss, opt, metrics={'acc', 'loss'})
64. # dataset\_sink\_mode can be True when using Ascend
65. model.train(num\_epochs, ds\_train,
66. callbacks=[loss\_cb], dataset\_sink\_mode=False)
67. metrics = model.eval(ds\_eval, dataset\_sink\_mode=False)
68. print('Metrics:', metrics)
69. 训练效果

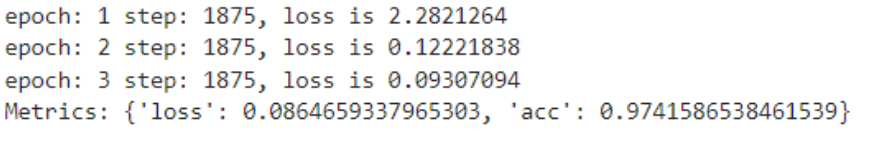


图 8-5 MindSpore在Minist数据集上的表现

## 区别与联系

### 区别

1. TensorFlow 是一个灵活、可移植、面向超大数据量的训练的深度学习框架，支持多种语言。缺点是调试较难，底层接口写起来繁琐，版本兼容性差。
2. PyTorch 是一个动态计算图的深度学习框架，具有简单易用、灵活性强的优点，但是效率可能稍低，缺乏成熟的生态系统，不适合部署在移动端。
3. MindSpore 是一个支持昇腾芯片、GPU、CPU等异构平台的深度学习框架，在昇腾+MindSpore组合下性能领先。API易用，但缺点是社区较小，API接口友好度需要进一步提升。

### 联系

1. 三个框架都是深度学习框架，支持GPU加速、分布式训练等特性。
2. 三个框架都支持动态构建计算图，灵活性强，适合研究和实验。
3. 三个框架都具有一定的缺点和优点，在不同的场景下使用可以根据实际需求进行选择。

## 参考文献

[1]加日拉·买买提热衣木,常富蓉,刘晨等.主流深度学习框架对比[J].电子技术与软件工程,2018,No.129(07):74.

[2]庞涛.开源深度学习框架发展现状与趋势研究[J].互联网天地,2018(04):46-54.