PyTorch 卷积网络配置与训练（一）

人工智能与Python程序设计 教研组

提纲

1. 卷积网络的训练流程
2. CIFAR-10数据介绍与加载
3. 网络搭建与模型优化
4. 图像识别大作业

人工智能与
Python程序设计

提纲

1. 卷积网络的训练流程
2. CIFAR-10数据介绍与加载
3. 网络搭建与模型优化
4. 图像识别大作业

人工智能与
Python程序设计

卷积网络的训练流程

卷积网络的训练流程

Network

卷积网络的训练流程

网络前馈预测

→ Network → Dog!

卷积网络的训练流程

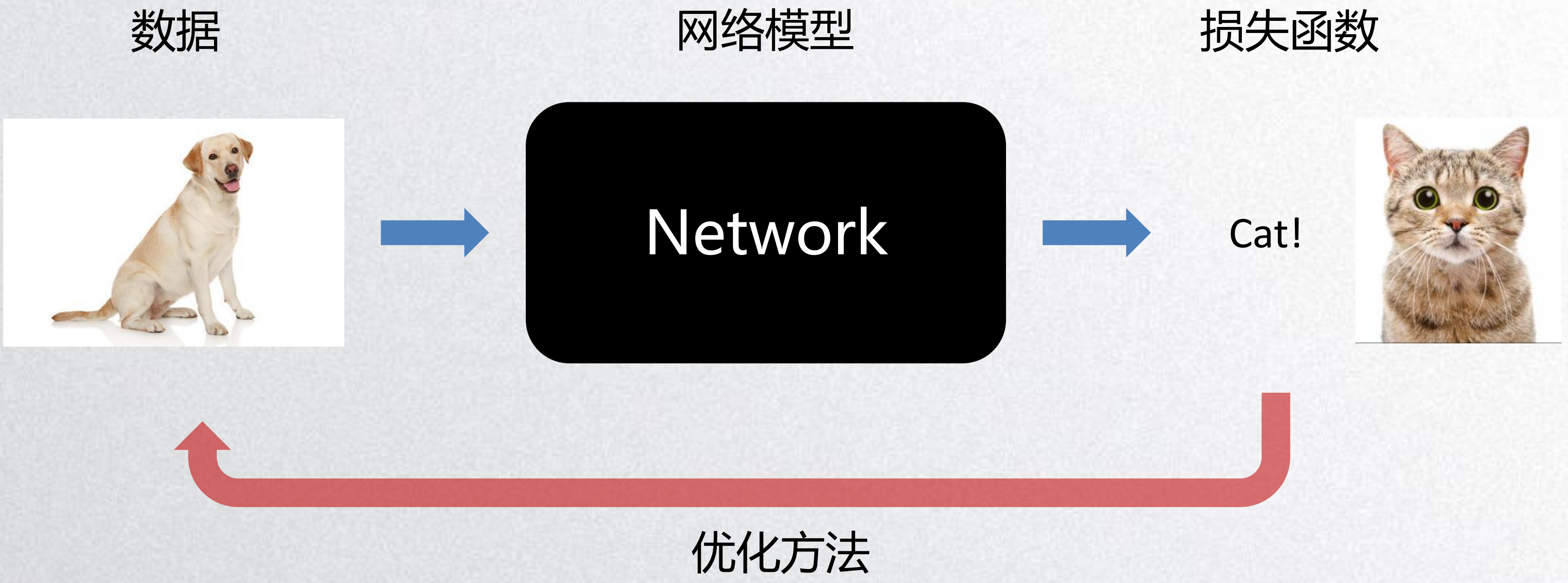
网络前馈预测

Network

Cat!

误差反向传播

卷积网络的训练流程



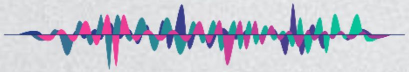
卷积网络的训练流程

数据准备

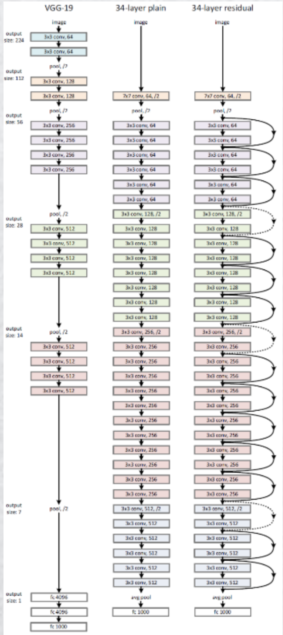
网络定义

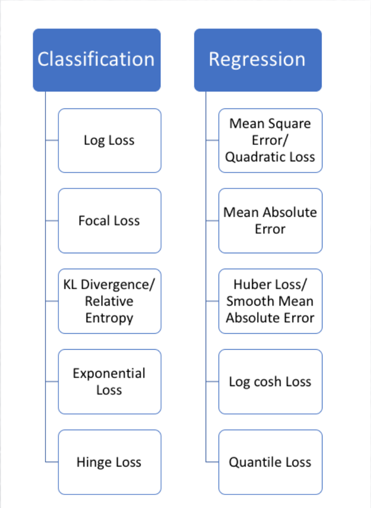
损失函数

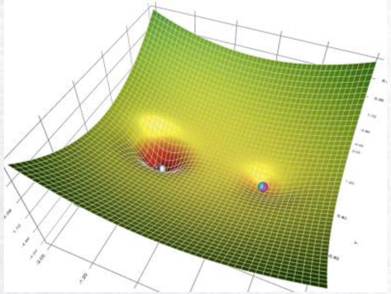
优化方法



It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness...







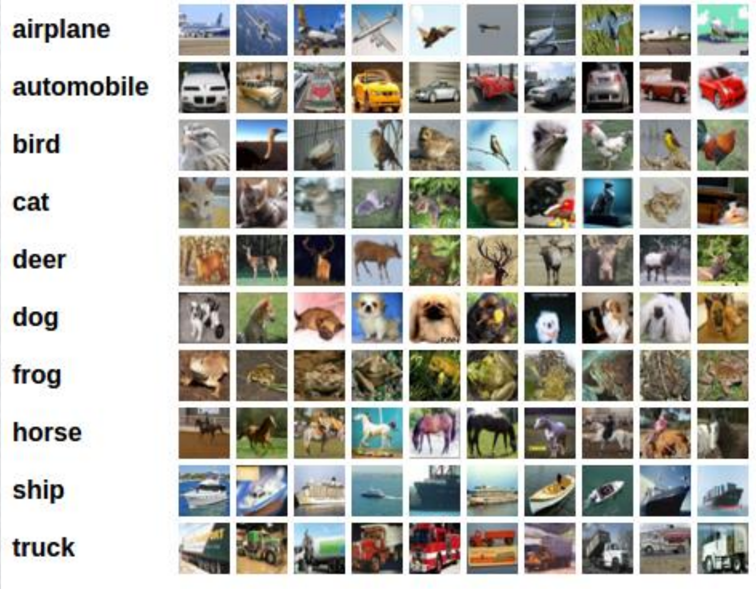
提纲

1. 卷积网络的训练流程
2. CIFAR-10数据介绍与加载
3. 网络搭建与模型优化
4. 图像识别大作业

人工智能与
Python程序设计

CIFAR-10数据介绍与加载

- CIFAR-10数据介绍
  - 10个类别，每个类别6000张图像，共计60000张。
  - 图像分辨率为32x32, 均为RGB三通道图像。



CIFAR-10数据介绍与加载

- torchvision库
  - 集成计算机视觉相关的数据集，模型架构，以及常用的图像处理操作等

- torchvision.datasets
  - CelebA
  - CIFAR
  - Cityscapes
  - COCO
  - DatasetFolder
  - EMNIST
  - FakeData
  - Fashion-MNIST
  - Flickr
  - HMDB51
  - ImageFolder
  - ImageNet
  - Kinetics-400
  - KMNIST

- torchvision.io
  - Video
  - Fine-grained video API
  - Image

- torchvision.models
  - Classification
  - Semantic Segmentation
  - Object Detection, Instance Segmentation and Person Keypoint Detection
  - Video classification

- torchvision.transforms
  - Scriptable transforms
  - Compositions of transforms
  - Transforms on PIL Image and torch.*Tensor
  - Transforms on PIL Image only
  - Transforms on torch.*Tensor only
  - Conversion Transforms
  - Generic Transforms
  - Functional Transforms

CIFAR-10数据介绍与加载

- 基于torchvision库对CIFAR-10数据加载

数据集类实例化

```
import torch
import torchvision


trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)
```

CIFAR-10数据介绍与加载

- 构建CIFAR-10数据类

```
class CIFAR10(VisionDataset):
.
.
.
    def __getitem__(self, index: int) -> Tuple[Any, Any]:
        """
        Args:
            index (int): Index

        Returns:
            tuple: (image, target) where target is index of the target class.
        """
        img, target = self.data[index], self.targets[index]

        # doing this so that it is consistent with all other datasets
        # to return a PIL Image
        img = Image.fromarray(img)

        if self.transform is not None:
            img = self.transform(img)

        if self.target_transform is not None:
            target = self.target_transform(target)

        return img, target

    def __len__(self) -> int:
        return len(self.data)
```

根据index获取对应数据样本，并返回到数据队列

数据个数声明

CIFAR-10数据介绍与加载

- 基于torchvision库对CIFAR-10数据加载

```python
import torch
import torchvision


trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)
```

DataLoader实例化

提纲

1. 卷积网络的训练流程
2. CIFAR-10数据介绍与加载
3. 网络搭建与模型优化
4. 图像识别大作业

人工智能与
Python程序设计

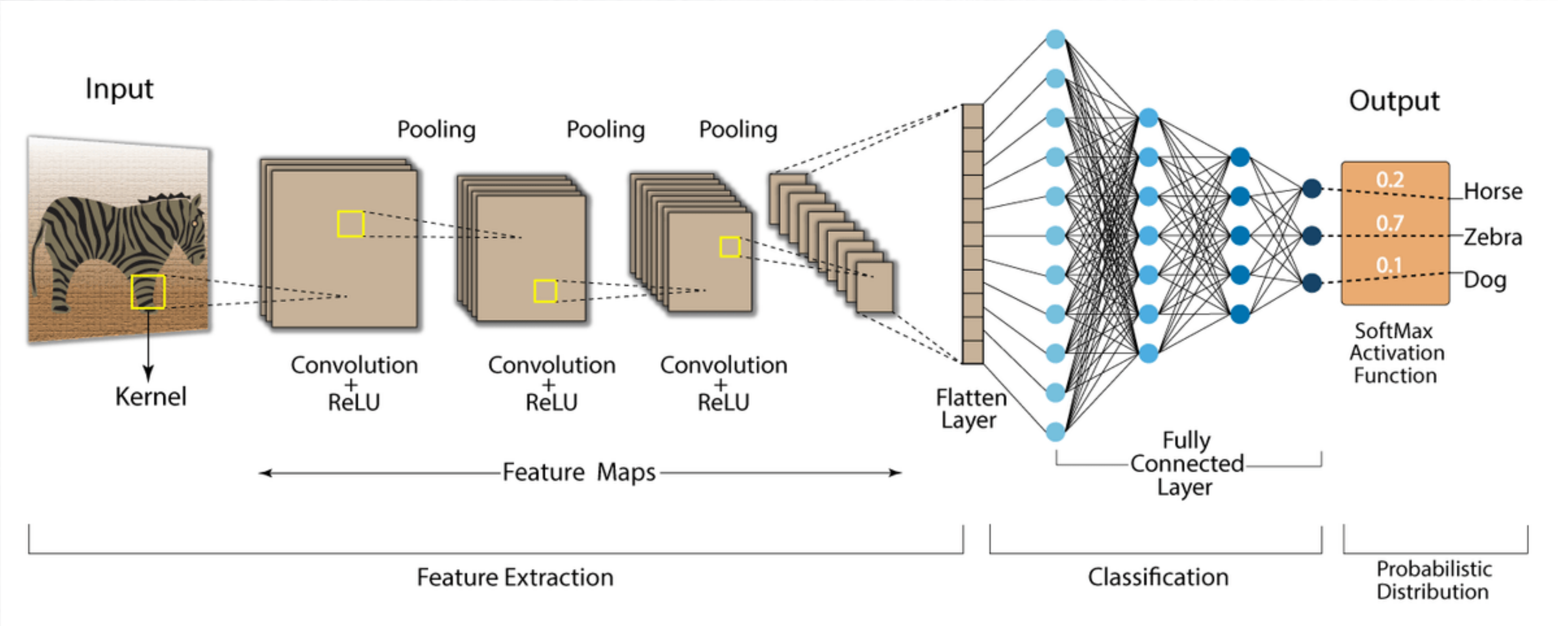
网络搭建与模型优化

网络模型

卷积 → 卷积 → 全连接 → 全连接 → 分类

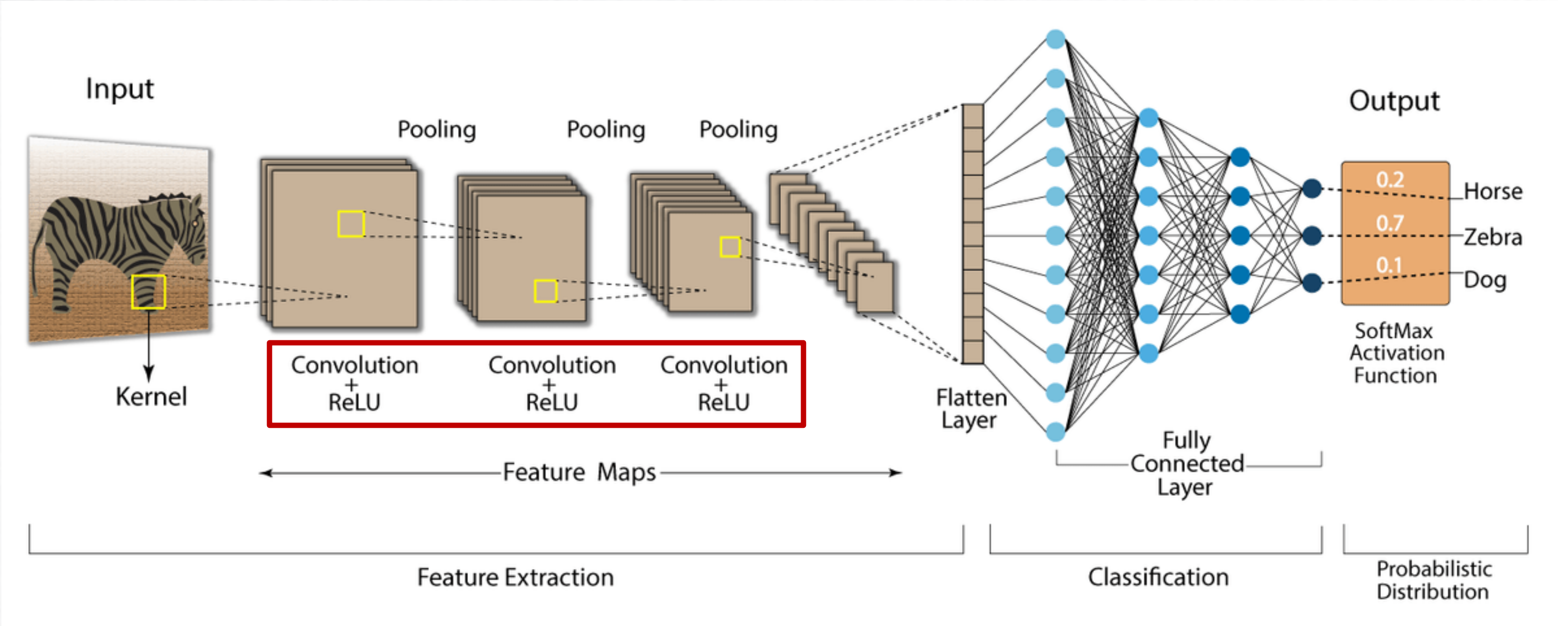
类别预测

网络搭建与模型优化



Input
Pooling
Pooling
Pooling
Output
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Flatten Layer
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Fully Connected Layer
Feature Maps
Feature Extraction
Classification
Probabilistic Distribution

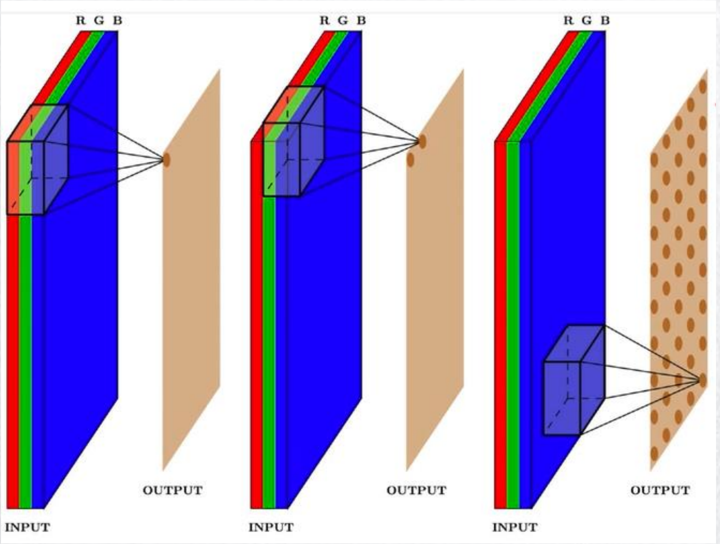
网络搭建与模型优化



Input
Pooling
Pooling
Pooling
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Feature Maps
Flatten Layer
Fully Connected Layer
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Feature Extraction
Classification
Probabilistic Distribution

网络搭建与模型优化

- 卷积

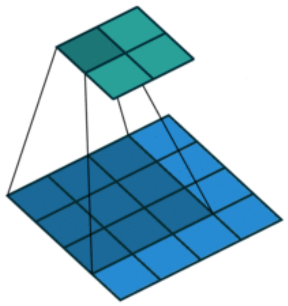


网络搭建与模型优化

- 卷积

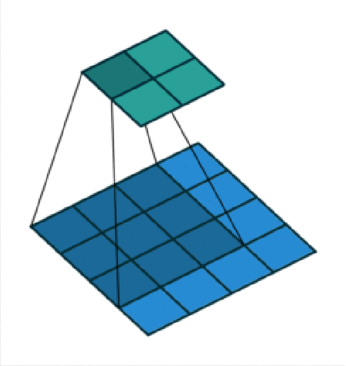
CLASS `torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')` [SOURCE]

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int* or *tuple*) – Size of the convolving kernel
- **stride** (*int* or *tuple*, *optional*) – Stride of the convolution. Default: 1
- **padding** (*int* or *tuple*, *optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string*, *optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int* or *tuple*, *optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int*, *optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool*, *optional*) – If `True`, adds a learnable bias to the output. Default: `True`

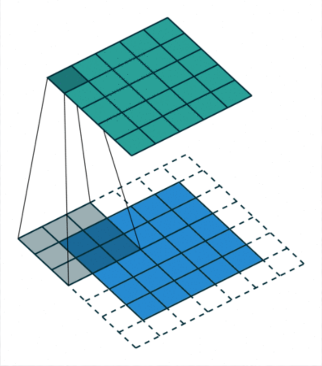


网络搭建与模型优化

- 卷积

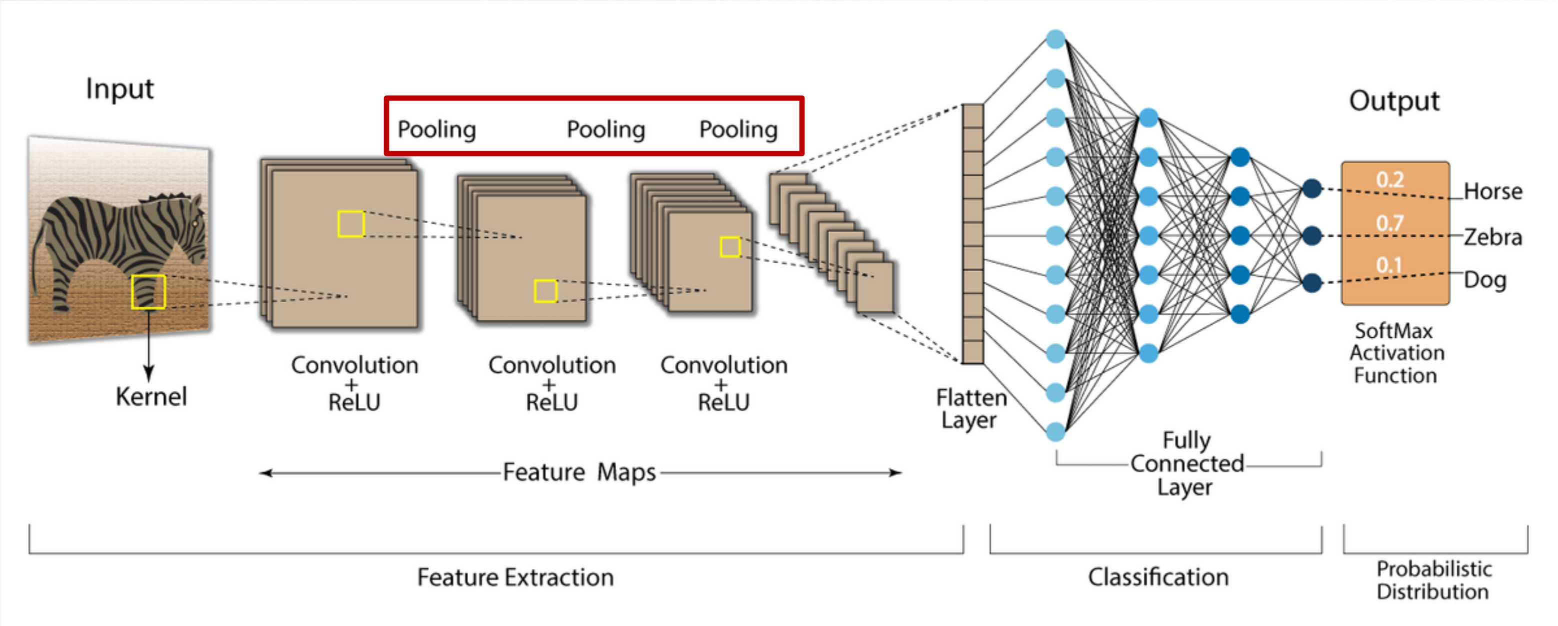


No padding,
stride 1



padding: 1
stride: 1

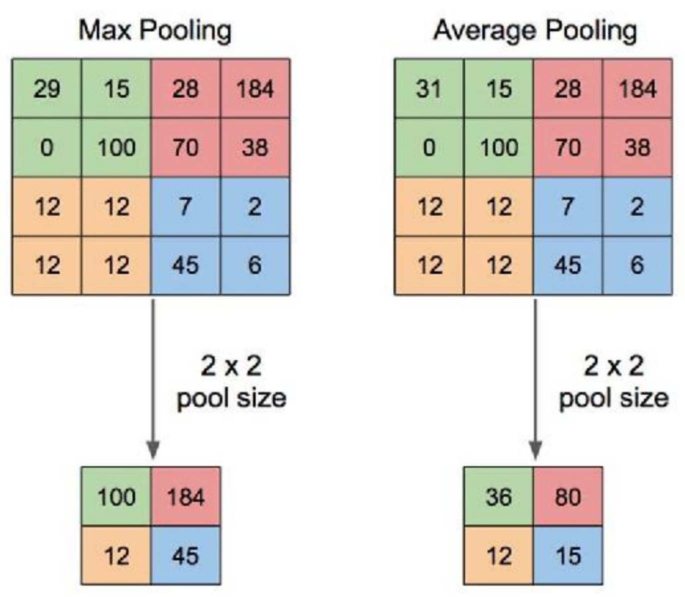
网络搭建与模型优化



Input
Pooling
Pooling
Pooling
Kernel
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Flatten Layer
Feature Maps
Fully Connected Layer
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Feature Extraction
Classification
Probabilistic Distribution

网络搭建与模型优化

- 池化(Pooling)



网络搭建与模型优化

- 池化(Pooling)

最大池化

CLASS `torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)`

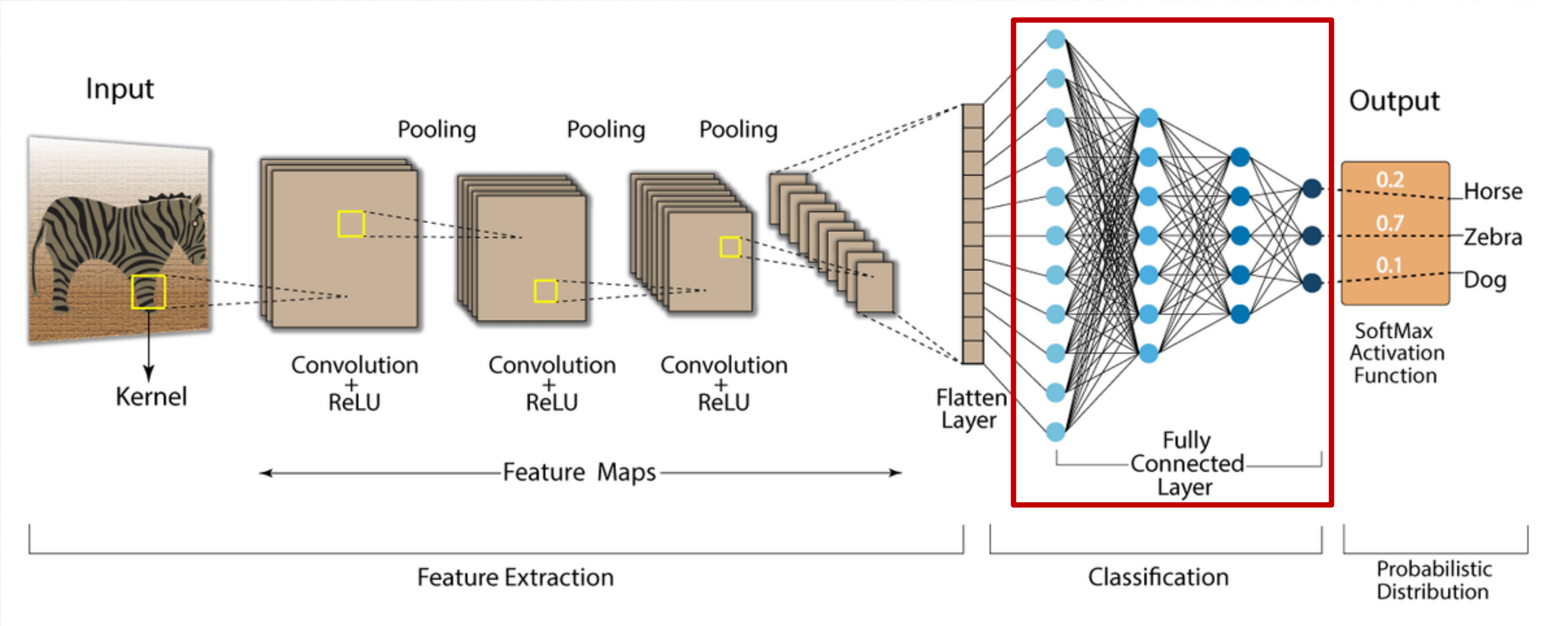
- **kernel_size** – the size of the window to take a max over
- **stride** – the stride of the window. Default value is `kernel_size`
- **padding** – implicit zero padding to be added on both sides
- **dilation** – a parameter that controls the stride of elements in the window
- **return_indices** – if `True`, will return the max indices along with the outputs. Useful for `torch.nn.MaxUnpool2d` later
- **ceil_mode** – when True, will use *ceil* instead of *floor* to compute the output shape

平均池化

CLASS `torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)`

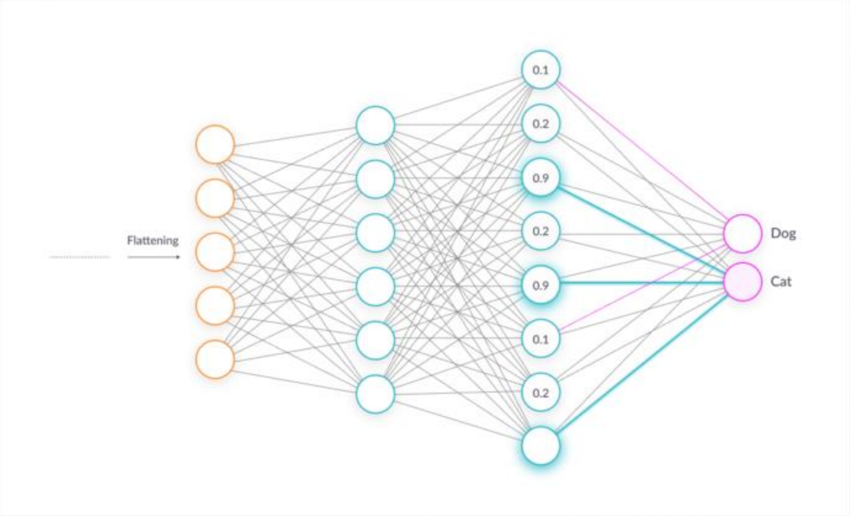
Applies a 2D average pooling over an input signal composed of several input planes.

网络搭建与模型优化


Input
Pooling
Pooling
Pooling
Kernel
Convolution
+
ReLU
Convolution
+
ReLU
Convolution
+
ReLU
Flatten
Layer
Fully
Connected
Layer
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax
Activation
Function
Feature Maps
Feature Extraction
Classification
Probabilistic
Distribution

网络搭建与模型优化

- 全连接(Linear, Fully Connected Layer)



网络搭建与模型优化

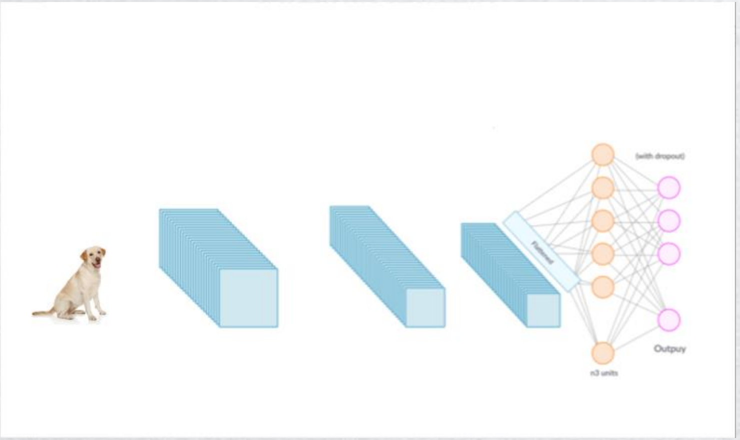
- 全连接(Linear, Fully Connected Layer)

CLASS `torch.nn.Linear(in_features: int, out_features: int, bias: bool = True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

网络搭建与模型优化




```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

网络搭建与模型优化

在构造函数中，实例化不同的layer组件，并赋给类成员变量

```
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

在前馈函数中，利用实例化的组件对网络进行搭建，并对输入Tensor进行操作，并返回Tensor类型的输出结果

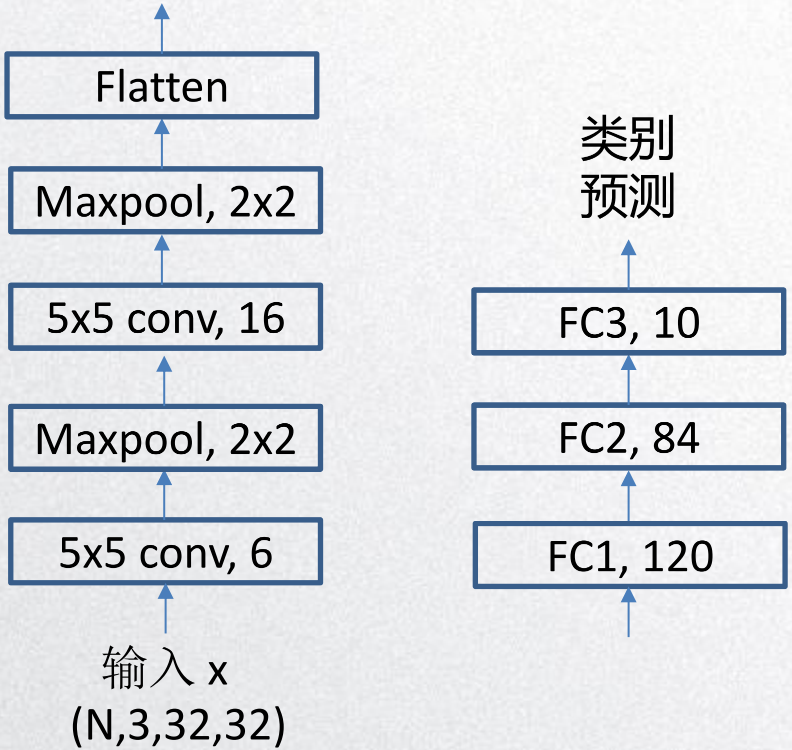
网络搭建与模型优化

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```



PyTorch中的损失函数

- 常用损失函数：
  - 分类模型：
    - 虽然可以强行通过回归方式解决，但是效果较差
    - 目前主流方法都是基于**概率相关方法**进行建模
      - 使得样本的分类概率达到最大

下雨	不下雨	下雨	不下雨	下雨	不下雨
0.1	0.9	0.3	0.7	0.4	0.6

使用概率的方法更符合真实情况

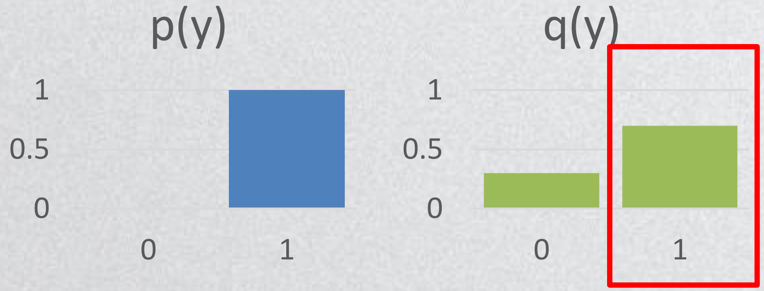
概率基础

- **多分类**概率分布
  - 样本空间：一个事件所有发生的可能情况
    - $S = \{$第0类，第1类，…，第$C-1$类$\}$
  - 样本点概率：$P(s)$，满足 $P(s) \geq 0$且$P(s) \leq 1$
  - 定义在一个样本空间S的概率分布，满足
    - $P(s) \geq 0$，$\forall\, s \in S$
    - $P(\text{第0类}) + P(\text{第1类}) + \cdots + P(\text{第}C-1\text{类}) = 1$
    - 可以用K个概率数表示对于一个事件发生的概率的估计
      - {0.1, 0.3, 0.2, 0.4}
      - {1, 0, 0, 0} 必定为第0类的概率
      - {0, 1, 0, 0} 必定为第1类的概率

PyTorch中的损失函数

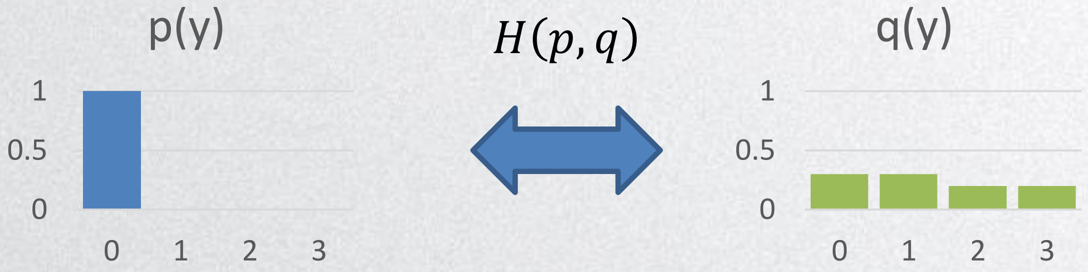
- 二分类交叉熵： $y \in \{0, 1\}$
  - $H(p,q) = -\sum_y p(y) \cdot \log q(y)$
    $= -\ \left(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right)$

    逻辑回归：$\hat{y}_i = \frac{1}{1+e^{-(xw+b)}}$
  - 在所有数据上取平均： $-\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$



PyTorch中的损失函数

- 多分类交叉熵：
  - $H(p,q) = -\sum_y p(y) \cdot \log q(y)$
  - 衡量真实分布$p(y)$和预测分布$q(y)$之间的差异
    - 如果$p(y) = q(y)$，那么交叉熵$H(p,q)$最小，且刚好等于$p(y)$的熵

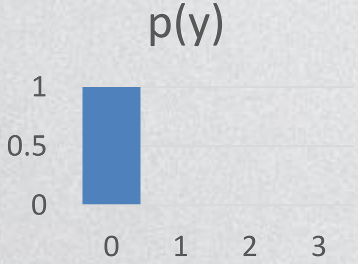


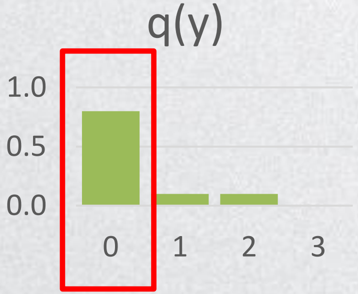
1、枚举两个概率分布的值，对应点$p(y)$和$\log q(y)$相乘

2、看起来很复杂，但是对于分类任务来说，只会有一项“**被激活**”（非0）

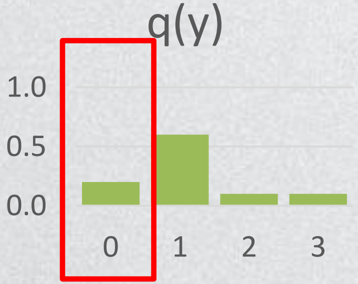
PyTorch中的损失函数

- 多分类： 假设总共有C类, $y \in \{0,1,2,\ldots,C-1\}$
  - $H(p,q) = -\sum_y p(y) \cdot \log q(y) = -\sum_{y=0}^{C-1} p(y) \cdot \log q(y)$





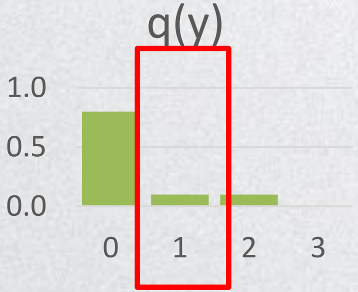
$H(p,q) = -\log 0.8 = 0.2231$



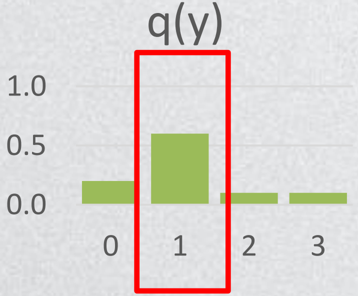
$H(p,q) = -\log 0.2 = 1.6094$

PyTorch中的损失函数

- 多分类： 假设总共有C类, $y \in \{0,1,2,\dots,C-1\}$
  - $H(p,q) = -\sum_y p(y)\cdot\log q(y) = -\sum_{y=0}^{C-1} p(y)\cdot\log q(y)$



$$H(p,q) = -\log 0.1 = 2.3026$$



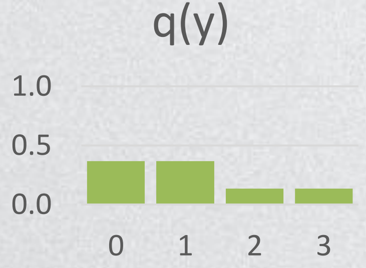
$$H(p,q) = -\log 0.6 = 0.5108$$

PyTorch中的损失函数

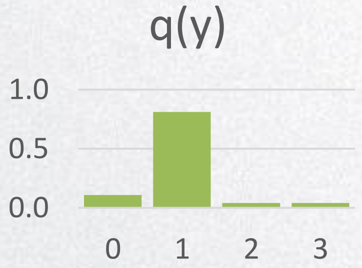
- 多分类： 假设总共有C类, $y \in \{0,1,2,\dots,C-1\}$
  - $H(p,q) = -\sum_y p(y)\cdot\log q(y) = -\sum_{y=0}^{C-1} p(y)\cdot\log q(y)$
- 问题：如何得到$q(y)$?
  - 模型输出一个C维向量$\boldsymbol{z} = [z[0], z[1], \dots, z[C-1]] \in R^C$
  - 利用softmax函数计算:

$$q(y=i) = \frac{e^{z[i]}}{\sum_j e^{z[j]}}$$

y	z	exp(z)	q(y)
0	1.000	2.718	0.366
1	1.000	2.718	0.366
2	0.000	1.000	0.134
3	0.000	1.000	0.134



y	z	exp(z)	q(y)
0	1.000	2.718	0.110
1	3.000	20.086	0.810
2	0.000	1.000	0.040
3	0.000	1.000	0.040



网络搭建与模型优化

- 损失函数
  - PyTorch中的交叉熵函数

CLASS `torch.nn.CrossEntropyLoss(weight: Optional[torch.Tensor] = None, size_average=None, ignore_index: int = -100, reduce=None, reduction: str = 'mean')` [SOURCE]

The loss can be described as:

$$\operatorname{loss}(x, class) = -\log\left(\boxed{\frac{\exp(x[class])}{\sum_j \exp(x[j])}}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

softmax
function

PyTorch中的损失函数

- PyTorch提供的交叉熵损失函数:
  - nn.CrossEntropyLoss
  - 输入:
    - NxC维矩阵$\boldsymbol{Z}$，其中每一行为$\boldsymbol{z} = [z[0], z[1], \dots, z[C-1]] \in R^C$
      - 每个数据点在C个类别上的“确信度”
    - N维向量$\boldsymbol{y}$, 其中每个元素为$y \in \{0, 1,2, \dots, C-1\}$
      - 标准答案

$$loss(\boldsymbol{Z}, \boldsymbol{y}) = -\sum_{i=0}^{N-1} \log(\frac{e^{\boldsymbol{Z}[i,y[i]]}}{\sum_j e^{\boldsymbol{Z}[i,j]}})$$

  - 注意：该损失函数同时计算了softmax函数和交叉熵函数:

网络搭建与模型优化

- 损失函数
  - PyTorch中的交叉熵函数
  - 交叉熵损失函数的实例化

```
import nn

criterion = nn.CrossEntropyLoss()
```

- 优化方法的声明与实例化

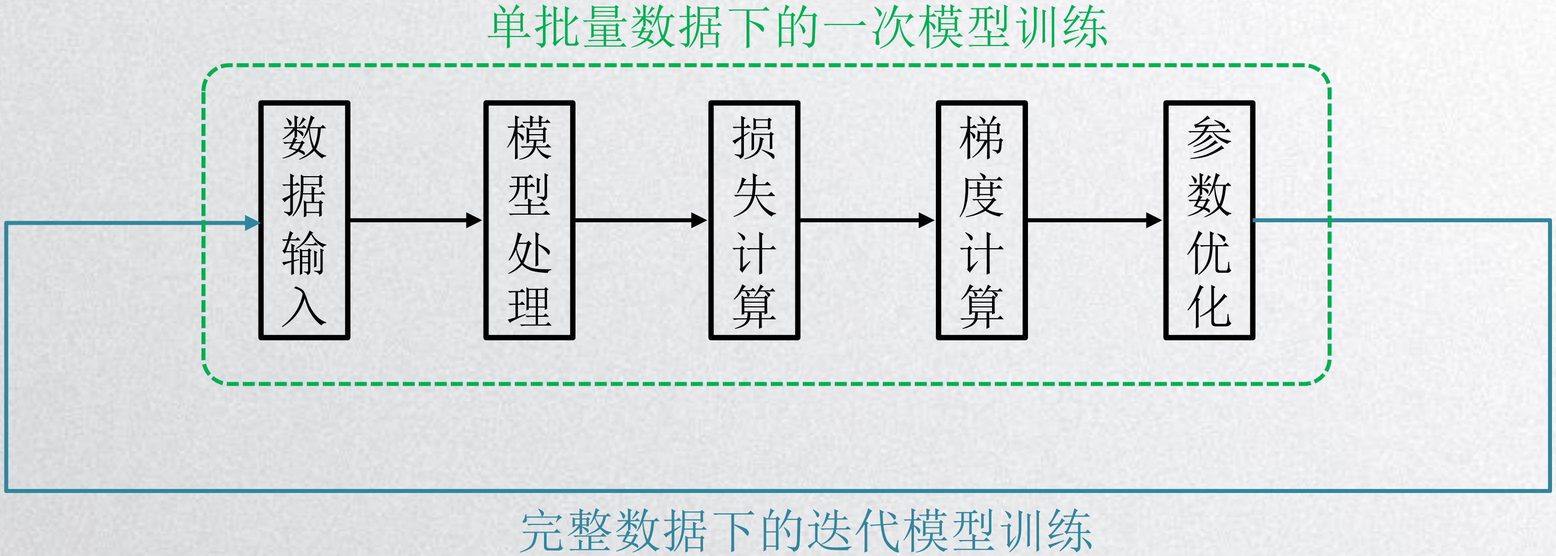
```
import torch.optim as optim

optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

网络搭建与模型优化

- 模型训练

网络定义　　损失函数　　优化方法



网络搭建与模型优化

- 模型训练

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

网络搭建与模型优化

- 模型训练

全部数据迭代次数

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

获取当前批次数据

清空模型参数的梯度

模型预测与参数优化

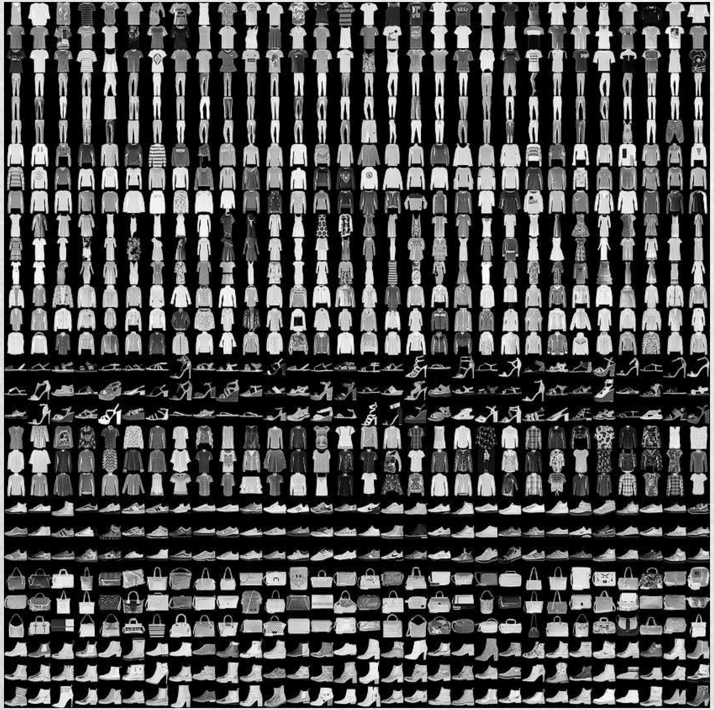
网络搭建与模型优化

- 简单卷积网络训练流程回顾
  - 参见simpleConvNet.py

提纲

人工智能与
Python程序设计

商品图像（Fashion-Mnist）分类



FashionMNIST 是一个图像数据集。 它是由 Zalando（一家德国的时尚科技公司）旗下的研究部门提供，涵盖了来自 10 种类别的共 7 万个不同商品的正面图片。

商品图像（Fashion-Mnist）分类

- 实现一个基于CNN的图像分类模型
- 具体要求：
  - 模型构建：继承nn.Module，实现FashionMnistModel类
  - 模型训练：基于Fashion-Mnist数据集，完成FashionMnistModel的训练
  - 模型测试：实现test函数，完成图像分类测试

机器学习的基本流程

数据准备

- 数据标注
- 训练集/验证集/测试集分割
- 特征提取

模型训练

- 分类损失函数
- 损失函数优化和参数调优

模型测试

- 性能评价指标
- 交叉验证

数据准备

- 构建数据集类：

构造函数：
实例化数据加载的相关类属性

```
class FashionDataset(Dataset):
    '''
    定义Dataset:
    - 用于加载训练和测试数据，请勿改动
    - 返回一张图片(3维Tensor)以及对应的标签(0-9)
    '''
    def __init__(self,datadir,transform,is_train = True):
        super().__init__()
        self.datadir = datadir
        self.img,self.label = self.load_data(self.datadir,is_train = is_train)
        self.len_data = len(self.img)
        self.transform = transform

    def __getitem__(self,index):
        return self.transform(self.img[index]), self.label[index]

    def __len__(self):
        return self.len_data

    def load_data(self,datadir,is_train):
        dirname = os.path.join(datadir)
        files = ['train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',
            't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz']

        paths = []
        for fname in files:
            paths.append(os.path.join(dirname,fname))
        if is_train:

            with gzip.open(paths[0], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)
            with gzip.open(paths[1], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                   offset=16).reshape(len(label), 28, 28)
        else:
            with gzip.open(paths[2], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)

            with gzip.open(paths[3], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                      offset=16).reshape(len(label), 28, 28)
        return img, label
```

数据准备

- 构建数据集类：

构造函数：
实例化数据加载的相关类属性

```
class FashionDataset(Dataset):
    '''
    定义Dataset:
    - 用于加载训练和测试数据，请勿改动
    - 返回一张图片(3维Tensor)以及对应的标签(0-9)
    '''
    def __init__(self,datadir,transform,is_train = True):
        super().__init__()
        self.datadir = datadir
        self.img,self.label = self.load_data(self.datadir,is_train = is_train)
        self.len_data = len(self.img)
        self.transform = transform

    def __getitem__(self,index):
        return self.transform(self.img[index]), self.label[index]

    def __len__(self):
        return self.len_data

    def load_data(self,datadir,is_train):
        dirname = os.path.join(datadir)
        files = ['train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',
            't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz']

        paths = []
        for fname in files:
            paths.append(os.path.join(dirname,fname))
        if is_train:

            with gzip.open(paths[0], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)
            with gzip.open(paths[1], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                   offset=16).reshape(len(label), 28, 28)
        else:
            with gzip.open(paths[2], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)

            with gzip.open(paths[3], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                      offset=16).reshape(len(label), 28, 28)
        return img, label
```

根据index获取对应数据样本，并返回回到数据队列

数据准备

- 构建数据集类：

构造函数：
实例化数据加载的相关类属性

获取所有数据个数

```python
class FashionDataset(Dataset):
    '''
    定义Dataset:
    - 用于加载训练和测试数据，请勿改动
    - 返回一张图片(3维Tensor)以及对应的标签(0-9)
    '''
    def __init__(self,datadir,transform,is_train = True):
        super().__init__()
        self.datadir = datadir
        self.img,self.label = self.load_data(self.datadir,is_train = is_train)
        self.len_data = len(self.img)
        self.transform = transform

    def __getitem__(self,index):
        return self.transform(self.img[index]), self.label[index]

    def __len__(self):
        return self.len_data

    def load_data(self,datadir,is_train):
        dirname = os.path.join(datadir)
        files = ['train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',
            't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz']

        paths = []
        for fname in files:
            paths.append(os.path.join(dirname,fname))
        if is_train:

            with gzip.open(paths[0], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)
            with gzip.open(paths[1], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                offset=16).reshape(len(label), 28, 28)
        else:
            with gzip.open(paths[2], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)

            with gzip.open(paths[3], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                   offset=16).reshape(len(label), 28, 28)
        return img, label
```

根据index获取对应数据样本，并返回到数据队列

数据准备

- 构建数据集类：

构造函数：
实例化数据加载的相关类属性

获取所有数据个数

读取所有数据

```python
class FashionDataset(Dataset):
    '''
    定义Dataset:
    - 用于加载训练和测试数据，请勿改动
    - 返回一张图片(3维Tensor)以及对应的标签(0-9)
    '''
    def __init__(self,datadir,transform,is_train = True):
        super().__init__()
        self.datadir = datadir
        self.img,self.label = self.load_data(self.datadir,is_train = is_train)
        self.len_data = len(self.img)
        self.transform = transform

    def __getitem__(self,index):
        return self.transform(self.img[index]), self.label[index]

    def __len__(self):
        return self.len_data

    def load_data(self,datadir,is_train):
        dirname = os.path.join(datadir)
        files = ['train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',
            't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz']

        paths = []
        for fname in files:
            paths.append(os.path.join(dirname,fname))
        if is_train:

            with gzip.open(paths[0], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)
            with gzip.open(paths[1], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                    offset=16).reshape(len(label), 28, 28)
        else:
            with gzip.open(paths[2], 'rb') as lbpath:
                label = np.frombuffer(lbpath.read(), np.uint8, offset=8)

            with gzip.open(paths[3], 'rb') as imgpath:
                img = np.frombuffer(imgpath.read(), np.uint8,
                                       offset=16).reshape(len(label), 28, 28)
        return img, label
```

根据index获取对应数据样本，并返回到数据队列

数据准备

- 实例化DataLoader：
  - Train_loader
  - Test_loader

```python
# 定义data loader
train_dataset = FashionDataset('data',
                          transform=transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.1307,), (0.3081,))
                          ])
                         )



train_loader = DataLoader(train_dataset,batch_size=320, shuffle=True, num_workers= 4)

test_dataset = FashionDataset('data',
                          transform=transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.1307,), (0.3081,))
                          ]),
                         is_train = False
                         )
test_loader = DataLoader(test_dataset,batch_size=32, shuffle=False, num_workers= 1)
```

模型搭建与训练

- 模型训练与测试类：

```python
class Model():
    def __init__(self):
        '''
        创建模型和优化器，设置模型超参数
        * 参数
            * learning_rate
            * epoches
            * model_save_path
            * device: cuda or cpu
        * 模型
            * 创建FashionMnistModel的实例，命名为model
            * 定义optimizer
            * 定义loss function
        '''
        self.lr = 0.01
        self.epoches = 20
        self.model_save_path = './model'
        # 指定训练的device，优先使用GPU，GPU不可用时加载CPU
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = FashionMnistModel().to(self.device)
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=self.lr)
        self.loss_function = nn.CrossEntropyLoss()
```

模型搭建与训练

- 模型训练与测试类:

```python
class Model():
    def __init__(self):
        '''
        创建模型和优化器，设置模型超参数
        * 参数
            * learning_rate
            * epoches
            * model_save_path
            * device: cuda or cpu
        * 模型
            * 创建FashionMnistModel的实例，命名为model
            * 定义optimizer
            * 定义loss function
        '''
        self.lr = 0.01
        self.epoches = 20
        self.model_save_path = './model'
        # 指定训练的device，优先使用GPU，GPU不可用时加载CPU
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = FashionMnistModel().to(self.device)
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=self.lr)
        self.loss_function = nn.CrossEntropyLoss()
```

需同学自主完成

FashionMnistModel（）模型搭建

模型搭建与训练

- 模型搭建：

```python
class FashionMnistModel(nn.Module):
    def __init__(self):
        '''
        ***********请在此写入你的代码**********
        定义模型
        '''

    def forward(self, x):
        '''
        *********请在此处输入你的代码*********
        输入: input, 它的size是(batch_size, img_h, img_w, img_c)
        输出（返回值）: output(预测值), hidden(隐藏层的值)
            * output的size是(batch_size, num_label)

        定义模型函数:
            * 将输入经过卷积层和激活函数
            * 使用pooling降低通道数
            * 对卷积层的输出做适当的维度变换
            * 用线性层将output映射到num_label的维度上
            * 返回output
        '''
```

模型搭建与训练

- 模型训练：

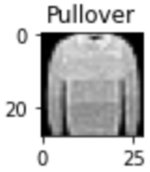
```python
def train(self,train_loader,test_loader):
    '''
    训练函数
    '''
    self.model.train()
    for epoch in range(self.epoches):
        loss_list = []
        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(self.device), target.long().to(self.device)
            self.optimizer.zero_grad()
            output = self.model(data)
            loss = self.loss_function(output, target)
            loss.backward()
            self.optimizer.step()
            loss_list.append(loss.item())
            if batch_idx % 50 == 0:
                print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                    epoch, batch_idx * len(data), len(train_loader.dataset),
                           100. * batch_idx / len(train_loader), loss.item()))
        self.test(test_loader)
        # 保存模型参数
        if epoch+1 % 5 == 0:
            self._save_model(epoch+1)
```

模型搭建与训练

- 模型测试：

```python
img,label = test_dataset.__getitem__(20)
pred = inference(model2,img)
fig = plt.figure(figsize=(1,1))
plt.imshow(np.squeeze(img), cmap='gray')
plt.title(label_classes[pred])
```

```
Text(0.5, 1.0, 'Pullover')
```




```python
def test(self,test_loader):
    '''
    检验模型测试集上的效果
    '''
    self.model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(self.device), target.long().to(self.device)
            output = self.model(data)
            test_loss += self.loss_function(output, target).item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

回顾

1. 卷积网络的训练流程
2. 简单卷积网络的搭建
3. 损失定义与模型优化
4. 图像识别大作业

人工智能与
Python程序设计

谢谢！