第一天

2019年6月6日 9:27

1. **深度学习的概念**
    a. 机器学习的分支，以神经网络为基础，对数据的特征进行学习。
2. **和机器学习的区别**
    a. 深度学习不需要手动的进行特征工程
    b. 深度学习需要的数量多，需要的计算性能强

深度学习的应用场景

常见框架

神经网络

1. 概念
    a. 模仿生物的神经系统实现的模型，能够对数据的特征进行学习
2. 神经元
    a. 神经网络中的最小的单元。不同的神经元组合能够得到神经网络
    b. 结构：t = f(Wx+b)
    c. 内积：点积，向量的乘法（对应位置相乘相加），得到一个标量。
3. 单层的神经网络
4. 两层的神经网络：
    a. 感知机：两层神经网络。输入层有多个神经元，输出层一个神经元。
    b. 感知机的作用：进行一个二分类
5. 多层神经网络：
    a. 每一层的神经元之间没有连接
    b. 全连接层：当N层的每一个神经元和前一层的每一个神经元都有链接的时候，第N层就是全连接层。就是在进行矩阵乘法，进行特征的变换，就在进行y = wx+b。

6. 激活函数：把原来的数据进行变换
    a. 为什么要使用非线性的激活函数
        i. 线性的激活函数或者是不是用激活函数，多层神经网络和两层神经网络没有区别
        ii. 是用非线性的激活函数能够增加模型的非线性分割能力
    b. 为什么要使用简单的激活函数（RELU）
        i. RELU方便求导
        ii. sigmoid在取值很大或者很小的时候，导数非常小，会导致参数的更新速度很慢

c. 激活函数的作用：
    i. 增加非线性分割能力
    ii. 增加模型的稳健性（让模型能够拟合不同的数据）
    iii. 缓解梯度消失
    iv. 加速模型收敛
d. 线性
    i. f(kx) = kf(x)
    ii. f(x+y) = f(x) + f(y)

pytorch的使用

1. 张量
    a. 0阶张量：常数，eg:1
    b. 1阶张量：向量，eg：[1,2,3]
    c. 2阶张量：矩阵，eg: [[1,2]]

2. 张量的创建
    a. torch.tensor(list/array)
    b. torch.ones/empty/zeros([3,4])
    c. torch.rand([3,4]) ,[0,1)
    d. torch.randint(low,high,size=[]):生成size个取值从low到high的随机整数
    e. torch.randn([3,4]) 均值 为0标准差为1的数组
3. 张量相关的属性和方法
    a. tensor中只有一个元素的时候，获取数据
        i. tensor.item()
    b. tensor转化为ndarray
        i. tensor.numpy()
    c. 获取tensor的形状
        i. tensor.size(dim)
    d. tensor的变形和转置
        i. tensor.view()
        ii. tensor.t(0,1)
    e. tensor的切片和索引
        i. 和numpy相同
    f. 获取tensor中的最大值
        i. tensor.max()
        ii. tensor.max(dim=-1) #dim=-1获取行方向的最大值
4. tensor的数据类型和修改方法

a. int32 -->int
b. int64 ---> long
c. float32--->float
d. float64 ---> double
e. tensor.float().long().double().int()

5. tensor的计算
    a. tensor+tensor，形状相同对应位置计算
    b. 形状不同：可以进行广播,([1,2,3],[10,5,1])
    c. tensor+数字 ,tensor中的每个值和数字计算
    d. x.add_(y) x的值会直接被修改
6. CUDA类型的tensor
    a. device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    b. torch.tensor([1,,3],device) #创建cuda类型的tensor
    c. tensor.to(device) #把tensor转化为cuda支持 的tensor
    d. tensor.cpu() #把cuda的tensor转化为cpu上的tensor

第二天

2019年6月8日 9:53

梯度下降

1. 梯度：向量，值是导数，方向是变化最快的方向
2. 导数的计算：▽w =f(w+d)-f(w-d)/2d（d非常小)
3. 更新参数w： w = w-α▽w

链式法则：把复杂的函数用更小的变量去表示包含x的部分

反向传播：先计算最后一层的偏导，之后再计算倒数第二层...

1. 链式法则：
2. 计算图：使用图来表示一个计算，用一个中间变量来表示每一次的计算结果

pytorch实现线性回归

1. tensor 中的require_grad参数
    a. 设置为True，表示会记录该tensor的计算过程
2. tensor中的grad_fn属性
    a. 用来保存计算的过程
3. tensor不保留计算过程
    a. with torch.no_grad():
4. 反向传播：
    a. out.backward()
    b. 导数保存在tensor.grad,默认梯度会累加
5. tensor.data
    a. 获取tensor中值的引用操作
6. tensor.numpy（）
    a. 当tensor中需要计算梯度的时候，grad_fn不为None的时候，tensor.data.numpy()、tensor.detach().numpy()
7. 实现线性回归的逻辑
    a. 准备数据
    b. 初始化参数，进入循环（参数的梯度置为0），计算预测值
    c. 计算loss，loss.backward()计算梯度
    d. 更新参数
8. pytorch通过api完成模型和训练

a. api
    i. nn.Module 构造模型
        1) init：自定义的方法实现的位置
        2) forward：完成一次向前计算的过程
    ii. optimizer优化器类
        1) torch.optim.SGD/Aadm
        2) 流程：
            a) 实例化 Aadm(model.paramters(),lr)
            b) 梯度置为0，optimizer.zero_grad()
            c) 反向传播，计算梯度：loss.backward()
            d) optimizer.step() 参数的更新
    iii. 损失函数
        1) 损失函数对象torch.nn提供
b. 训练的过程：
    i. 实例化模型
    ii. 实例化损失函数
    iii. 实例化优化器类
    iv. 进入循环：
        1) 梯度置为0
        2) 调用模型得到预测值
        3) 调用loss函数，得到损失
        4) loss.backward() 进行梯度计算
        5) optimizer.step()
c. 注意点
    i. model.eval() #把模型置为评估模型
        1) model.training = False
    ii. GPU上运行代码
        1) 自定义的tensor.to(device)
        2) model.to(device)

优化算法：

1. 梯度下降：把所有的数据传入模型，计算平均梯度，更新参数，缺点：慢
2. 随机梯度下降：选一条数据进行参数的更新。缺点：容易受到噪声数据的影响
3. 批梯度下降：每次选择一波数据进行参数的更新。缺点：梯度的变化幅度可能会很大，在最小值附件徘徊
4. 动量法：把历史的梯度考虑进去。更新参数时候使用梯度=历史梯度的指数加权平均
5. adagrad : 更新参数的时候，使用自适应的学习率，= 学习率/历史梯度的平方和
6. rmsprop：使用自适应的学习率，= 学习率/历史梯度的平方的指数加权平均
7. adam：动量法和RMSprop的结合版本，既考虑梯度（使用梯度的指数加权平均），又考

虑学习率（让学习率除以历史梯度的平方的指数加权平均）

第三天

2019年6月9日 9:22

pytorch中数据加载

batch：数据打乱顺序，组成一波一波的数据，批处理

epoch：拿所有的数据训练一次

Dataset基类，数据集类

1. torch.utils.data.Dataset
2. 两个重要的方法：
    a. __getitem__(index) :能够对实例进行索引
    b. __len__: len(实例) 调用实例的__len__方法

迭代数据集

1. torch.utils.data.DataLoader（dataset，batch_size,shuffle）

手写数字识别的思路：

1. 准备数据，通过dataset和DataLoader准备
2. 模型构建
3. 模型训练，模型保存和加载
4. 模型的评估

a. 准备Mnist数据
    a. torchvision.transforms.ToTensor
        i. 把ndarray转化为tensor
        ii. PIL中iamge对象对转化为tensor
    b. torchvision.transforms.Normlize(mean,str)
        i. mean，std的形状和通道数相同
    c. torchvision.transforms.Compose
        i. 把不同的实例组合使用
b. 交叉熵损失
    a. nn.CrossEntropyLoss()
    b. 使用带权损失计算交叉熵损失
        i. softmax(out)
        ii. $$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_K}}, j = 1 \cdots k$$

iii. output = F.log_softmax(out) #log(P)
iv. F.nll_loss(output,target) #$-\sum Y\log(p)$

c. 带权损失
i. $loss = -\sum w_i x_i$

c. 训练
a. 遍历dataloader
b. tqdm(可迭代对象,total=迭代总次数)

d. 模型的评估
a. 不需要计算梯度
b. 计算损失和准确率
c. 准确率的计算
i. 获取概率最大值的位置作为预测值
ii. 预测值和真实值判断相等，结果取均值

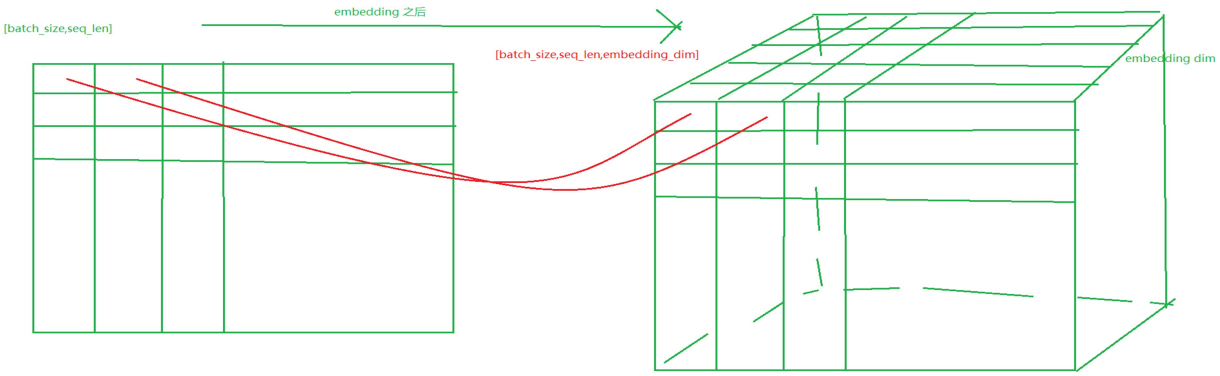
文本分词

N-gram：用连续的N个token左右一个特征，N往往取2或者3。

1. 考虑了句子中词语的顺序

文本向量化

1. one-hot
2. word emebdding:
   a. 用一个向量表示每一个词语，向量中的每个值都是参数，都会在后续通过训练得到
   b.



作业

构造数据（有三个特征值，一个目标值），构造模型进行训练，进行模型评估

第四天

2019年6月11日 9:30

文本情感分类

1. 准备数据
    1. 如何完成基础打Dataset的构建和Dataloader的准备
        i. 注意点：dataloader中的collate_fn的实现
            1) collate_fn(batch):batch中是一个个的getItem的结果
    2. 每个batch中文本的长度不一致的问题如何解决
        i. 长句子裁剪
        ii. 短句子填充
    3. 每个batch中的文本如何转化为数字序列
        i. {“词”:int}

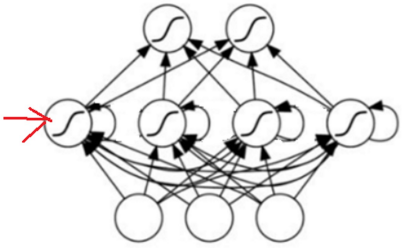
RNN：具有短期记忆的网络

1. 和普通网络的区别
    1. 能够更擅长解决时间序列问题
    2. 当前时刻神经元的输入有两个：
        i. 当前时刻的输入
        ii. 前一时刻的输出
2. RNN的常见类别
    1. one-one:图像分类
    2. one-many：图像描述
    3. many-one：文本的分类
    4. many-many：翻译
3. RNN的长依赖：带预测词语的位置比较远，很难影响到当前时间步的输出
4. LSTM：解决长依赖
    1. Cell state：细胞状态，存储的是记忆信息
    2. 遗忘门：决定什么信息会被忘记
    3. 输入门：把信息输入到cell state中
    4. 输出门：决定什么信息会被输出
5. GRU：更新门+ 输出门
6. 双向LSTM：让每个时间步既包含正向的信息，也包含反向的信息

pytorch中RNN的API

1. LSTM：
    1. torch.nn.LSTM

为什么LSTM默认batch_first=False
需要一个【seq_len,batch_size,embedding_dim】

2. 

lstm = LSMT(input_size=embedding_dim,hidden_size=4,num_layers=1,batch_first=True,bidriectional=False)
初始化 h_0,C_0
调用lstm：lstm(input,(h_0,c_0))
output,(h_n,c_n):
output:[batch_size,seq_len,4]
h_n,c_n,h_0,c_0： [1,batch_size,4]
embedding: [batch_size,seq_len,embeding_dim]
input:[batch_size,seq_len]

3. 双lstm中数据的拼接顺序
    i. **output：按照正反计算的结果顺序在第2个维度进行拼接，正向第一个拼接反向的最后一个输出**
    ii. **hidden state:按照得到的结果在第0个维度进行拼接，正向第一个之后接着是反向第一个**

第四天重点：

1. collate_fn(batch):
    1. batch ---> [（一个getitem的结果），() ()]
2. word sequence：实现方法把字符串转化为数字序列
    1. 文本padding，让batch中文本长度一致
    2. 对词频进行过滤
    3. 字符串--》数字，数字--》字符串
3. RNN
    1. 是什么：具有短期记忆的网络
        i. 为什么具有短期记忆：把前一次的输出作为当前的输入
    2. LSTM：能够解决长依赖问题
        i. 使用三个门结构实现长依赖
            1) 遗忘门：决定信息的遗忘
            2) 输入门：输入信息
            3) 输出门：输出信息
        ii. 门的理解：通过sigmoid和参数控制输出输入和输出的大小
    3. GRU和双向LSTM，双向GRU和LSTM的区别
        i. GRU2个门：
            1) 更新门
            2) 输出门
        ii. 双向LSTM、GRU
            1) 正向的第一个输出和方向的最后一个输出进行拼接
            2) 结果同时包含正向和方向信息

4. pytorch中LSTM的API
    i. torch.nn.LSTM(input_size,hidden_size,num_layers,batch_first,dropout,bidirectional)
    ii. 经常会用LSTM的最后一个时间步的输出表示文本

第五天

2019年6月12日 9:27

梯度消失：

1. 初始的参数权重太小
2. 使用了易饱和神经元

梯度爆炸

1. 初始的权重过大，导致梯度呈指数倍增加

解决梯度消失爆炸：

1. 使用易训练的神经元：relu
2. 改进优化器算法
3. 使用batch norm：对数据进行归一化，把数据拉回到[0,1]范围，让计算出的梯度不会过小

pytorch中的序列化容器

api:nn.sequential

隐马尔可夫

1. 应用场景
    a. 分词
    b. 词性标注
    c. 实体识别
2. 原理
    a. 两个假设：
        i. 1阶马尔科夫假设
            1) 当前的状态只受到之前有限个状态的影响
        ii. 输出独立性假设
            1) 观察值只受到状态的的影响
    b. 隐马尔克夫假设只要做什么？
        i. 通过语料（输入一个句子）寻找（概率最大）隐藏序列
3. 马尔科夫的三个问题
    a. 似然度问题：计算给定观察序列概率
    b. 解码问题：给定的观察序列，找概率最大的隐藏序列
    c. 学习问题：计算转移概率和观察似然度，通过统计语料得到的

第六天

2019年6月14日 10:09

聊天机器人的类型

1. QA bot ： 问答机器人
    a. 信息检索，搜索
    b. 知识图谱
2. TASK BOT：任务机器人，帮人做事情
    a. 槽位填充
3. Chat BOT：闲聊机器人
    a. 信息检索
    b. seq2seq模型

项目的实现逻辑：

1. 问题进行基础的处理
2. 判断用户意图
3. 如果用户进行闲聊，调用闲聊模型返回结果
4. 如果用户要进行问答，调用问答模型返回结果

词典的来源：

1. 输入法的词典
2. 手动收集

收集停用词

jieba

1. jieba.lcut
2. jieba.load_userdict(path)
3. jieba.posseg.cut() #返回词性

fasttext的使用方法

1. fasttext数据格式
    a. 分词后的句子\t__lable__label
2. import fastText
    a. model = fastText.trainsupervisored(path,dim....)
    b. model.save_model(path)

c. model = fastText.laod_model(path)
d. ([predict],array([prob])) = mode.predict(sentence)

第七天

2019年6月15日 9:38

fasttext的原理

1. 架构
    a. 输入层：词语+N-gram之后的词语 ---》embedding
    b. 隐藏层
    c. 输出层
2. 层次化的softmax（对传统softmax的优化）
    a. 哈夫曼树：树的带权路径最短的二叉树
    b. 带权路径长度：从根结点到该结点之间的路径长度与该结点的权的**乘积**
    c. 哈夫曼树的构造方法：
        i. 每次选择根节点权值最小的两棵树进行合并
    d. 哈夫曼编码：
        i. 满足前缀编码的要求
        ii. 把字母的出现次数叶子节点的权值，构造哈夫曼树
    e. 好处：减少计算的时间复杂度，提高效率
3. fasttext的负采样（对传统的softmax的优化）
    a. 采样：
        i. 数据根据label的数量分为V份，每一份是和数量的3/4次方成比例
        ii. 之后从数据中随机的选择neg个负样本
    b. 训练：
        i. 整个训练过程通过二分类的方式进行训练，提高当前正样本的概率
        ii. 损失=正样本的损失+neg个负样本的损失
    c. 好处：
        i. 提高训练速度
            1）训练数据少
            2）把多分类转化为2分类进行进行训练
        ii. 提高模型的稳健性，负样本可以模拟噪声数据
4. Seq2seq模型的原理
    a. 编码器：RNN
        i. 去理解句子
    b. 解码器：RNN
        i. 根据句子向量，得到输出

ii. 为什么可以有多个输出
    1）把前一次的输出和hidden_state作为下一次的输入
iii. 什么时候停止多个输出
    1）在训练的时候训练数据的target最后添加<EOS>
    2）在预测的时候预测结果为EOS，则停止

4. seq2seq的DEMO
    a. 文本转化为序列（数字序列，torch.LongTensor）
    b. 使用序列，准备数据集，准备Dataloader
        i. 随机种子，random.seed(9)
            1）使用同一个随机种子，每次随机的结果是一样的
        ii. add_eos:
            1）训练集中，target需要有EOS，告诉模型EOS是结束
            2）预测的时候，预测出EOS，意味着该结束
    c. 完成编码器
        i. 为了加速gru、lstm的计算，可以对句子进行打包和解包的操作
            1）torch.nn.utils.rnn.pack_paded_sequence
            2）nn.utils.rnn.pad_packed_sequence
            3）打包的方法需要batch的数据按照句子的长度降序排序
    d. 完成解码器
        i. decoder最开始的输入是一个[batch_size,1]的SOS，
        ii. 是一个在每个时间步循环
            1）output_t,hidden = forward_step(input,hidden)
            2）input是[batch_size,1],hidden和encoder hidden相同
            3）input先进行embedding
            4）结果通过gru处理得到output,[batch_size,1,hidden_size]
            5）结果进行变形为，【batch_size,vocab_Size】,进行log_softmax转化为概率
        iii. 把每个时间步的输出保存，用来和target计算loss
        iv. 当前时间步的输出，取概率最大的位置，作为预测值，作为下一个时间步的输入
    e. 完成seq2seq模型
        i. encode和decoder放到一起使用
    f. 完成模型训练的逻辑，进行训练
        i. loss怎么定义

1) 每个时间步是在进行多分类，类别数量=词典数量
2) outputs:[batch_Szie,max_Len,vocab_size],target:[batch_size,max_len]
3) 对outputs,和target进行变形，让batch_size*max_len,
4) 之后的损失计算和普通多分类一样

g. 完成模型评估的逻辑，进行模型评估
    i. 需要保存每个时间步的预测结果，供后续进行准确率的计算

作业：
使用对联数据，训练模型，能够输入上联，预测下联
数据地址：https://github.com/wb14123/couplet-dataset

第八天

2019年6月17日 9:32

seq2seq的优化

1. teacher forcing机制：
    a. 在seq2seq中避免一步错，步步错的情况
    b. 把真实值作为下一步的输入，能够加快模型的训练速度
    c. 在输出确定的情况下：可以在循环中使用teacher forcing
    d. 如果输出不确定，应该在循环外使用

2. Attention
    a. 初始化decoder的隐藏状态Z_i
    b. 和encoder每个时间步进行计算，得到的结果进行softmax，转化为概率
    c. 概率和encoder每个时间步记性相乘相加，得到C_i
    d. C_i作为decoder的输入，得到输出
3. attention的计算：
    a. 计算attention weight
    b. context vector
    c. attention result
4. badhanau 和Luong attention的区别
    a. 计算attention结果的位置不同
        i. baha 在decoder的每个时间步之前计算attention的结果，bahd使用的双向GRU计算attention的结果
        ii. Luong 在decoder每个时间步之后计算attention的二级果，使用单向多层GRU
    b. attention weight的区别
        i. bahd： V* tanh（Wzi-I + Uh_j）
        ii. luong:
            1）general :进行矩阵变换后进行矩阵乘法
            2）dot：对应位置相乘
            3）concat：
5. beam search
    a. decoder中的用来优化预测结果的
    b. decoder中
        i. 输入sos，得到多个个输出
        ii. 选择其中的概率最大的beam width个保留，分别作为下一次的输入

iii. 从所有的输出中选择概率最大的beam width个保留，分别作为下一次的输入
iv. 重复2-3步骤，知道到达max_len，获取到达EOS

6. 梯度裁剪
   a. 限制梯度的大小，最终抑制梯度爆炸
   b. nn.utils.clip_grad_norm_(model.paramters(),5)
7. 模型的优化方案：
   a. 参数的初始化
   b. 优化现有的数据，语料
      i. 数据进行清洗
         1）标点、表情、外文的处理
         2）把时间、人名、地点等名词替换成对应的各自符号
      ii. 从不同角度，不同复杂程度去准备语料
         1）角度：天气，吃饭，性别
         2）复杂度：简单、一般、复杂
   c. 工程的角度出发优化
      i. 使用模板，对常见的问题进行匹配，返回预设的答案
      ii. 使用分类模型，进行对问题的分类，返回预设的答案
      iii. 使用搜索模型，从现有的语料库中，返回相似问题对应的答案

问答机器人：

1. 实现逻辑
   a. 问题处理
   b. 相似问题的召回（可能相似的前K个问题）
   c. 相似问题的排序（把具体的相似度进行排序）
2. 问题的处理
   a. 基础的处理（清理）
   b. 实体识别，判断用户的问题中主语（用主语来过滤结果）
   c. 获取句子向量（计算相似度）
3. 相似问题的召回
   a. 海选，可能相似的结果
   b. 海选之后进行排序，速度更快
   c. 相似度计算的方法
      i. 在对应的主题中进行召回
      ii. 对数据进行聚类，在某一个类别中进行相似度的计算
   d. 思考：没有考虑词语的顺序?
4. 相似问题的排序
   a. 使用深度学习建立模型，输入用户的问题，和召回的问题，返回相似度

b. 思考：
    i. 数据的来源
        1) 抓取的百度知道
        2) 手动构造
    ii. 模型如何构建
        1) 孪生神经网络
        2) embedding+lstm

召回的流程

1. 准备数据
    a. 存到本地
    b. 存到数据库
2. 问题和语料转化为向量
    a. 使用tfidf
3. 计算相似度
    a. Pysparnn

```
# 1、 原始数据构造索引
cp = ci.MultiClusterIndex(features_vec, data)

#2. 索引中传入带搜索数据，返回结果
cp.search(search_features_vec, k=1, k_clusters=2, return_distance=False)
```

pysparnn的原理

1. 簇修建，简单的聚类后从个类别中进行数据的搜索，提高效率
2. 步骤
    a. 数据预处理：
        i. 随机选择根号N个样本作为learder
        ii. 剩下的数据每个找到和她最相近的leader，作为一簇
    b. 查询
        i. 计算leader和问题q的相似度，找到最相近的leader
        ii. 再计算q和leader中最相似的K个结果，返回
    c. 优化：
        i. 每个follower属于多个leader
        ii. 每次查询，查询最相似的多个leader
3. BM25的原理
    a. bm25 是一种最佳匹配方法
    b. 让词语的重要程度随着数量的增加而衰减
    c. 对句子的长度进行归一化，计算的结果受到句子长度的影响会变弱

d. bm25(i) = tf*中间项*idf
e. 中间项= (k+1)tf/tf + k(1-b+b d/avdl)