

4.5 实践：基于前馈神经网络完成鸢尾花分类

在本实践中，我们继续使用第三章中的鸢尾花分类任务，将Softmax分类器替换为本章介绍的前馈神经网络。 在本实验中，我们使用的损失函数为交叉熵损失；优化器为随机梯度下降法；评价指标为准确率。

4.5.1 小批量梯度下降法

在梯度下降法中，目标函数是整个训练集上的风险函数，这种方式称为**批量梯度下降法（Batch Gradient Descent，BGD）**。批量梯度下降法在每次迭代时需要计算每个样本上损失函数的梯度并求和。当训练集中的样本数量 N 很大时，空间复杂度比较高，每次迭代的计算开销也很大。

为了减少每次迭代的计算复杂度，我们可以在每次迭代时只采集一小部分样本，计算在这组样本上损失函数的梯度并更新参数，这种优化方式称为小批量梯度下降法（Mini-Batch Gradient Descent，Mini-Batch GD）。

第 t 次迭代时，随机选取一个包含 K 个样本的子集 \mathcal{B}_t ，计算这个子集上每个样本损失函数的梯度并进行平均，然后再进行参数更新。

$$\theta_{t+1} \leftarrow \theta_t - \alpha \frac{1}{K} \sum_{(\mathbf{x}, y) \in \mathcal{B}_t} \frac{\partial \mathcal{L}(y, f(\mathbf{x}; \theta))}{\partial \theta},$$

其中 K 为**批量大小(Batch Size)**。 K 通常不会设置很大，一般在 $1 \sim 100$ 之间。在实际应用中为了提高计算效率，通常设置为2的幂 2^n 。

在实际应用中，小批量随机梯度下降法有收敛快、计算开销小的优点，因此逐渐成为大规模的机器学习中的主要优化算法。此外，随机梯度下降相当于在批量梯度下降的梯度上引入了随机噪声。在非凸优化问题中，随机梯度下降更容易逃离局部最优点。

小批量随机梯度下降法的训练过程如下：

算法 4.1 小批量随机梯度下降法

输入: 训练集 $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^N$, 验证集 \mathcal{V} , 学习率 α

1 随机初始化 θ ;

2 **repeat**

3 对训练集 \mathcal{D} 中的进行随机分组, 每组大小为 K 个样本, 得到分组 $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_T$;

4 **for** $epoch = 1 \dots E$ **do**

5 **for** $iter = 1 \dots T$ **do**

6 计算 \mathcal{B}_t 上的梯度, 并更新参数;

7 **end**

8 **end**

9 **until** 模型 $f(\mathbf{x}; \theta)$ 在验证集 \mathcal{V} 上的错误率不再下降;

输出: θ

4.5.1.1 数据分组

为了小批量梯度下降法，我们需要对数据进行随机分组。目前，机器学习中通常做法是构建一个**数据迭代器**，每个迭代过程中从全部数据集中获取一批指定数量的数据。

数据迭代器的实现原理如下图所示：

- 1. 首先，将数据集封装为Dataset类，传入一组索引值，根据索引从数据集中获取数据；
- 2. 其次，构建DataLoader类，需要指定数据批量的大小和是否需要将数据进行乱序，通过该类即可批量获取数据。

在实践过程中，通常使用进行参数优化。在飞桨中，使用 `paddle.io.DataLoader` 加载minibatch的数据，`paddle.io.DataLoader` API可以生成一个迭代器，其中通过设置 `batch_size` 参数来指定minibatch的长度，通过设置shuffle参数为True，可以在生成 minibatch 的索引列表时将索引顺序打乱。

4.5.2 数据处理

构造IrisDataset类进行数据读取，继承自 `paddle.io.Dataset` 类。`paddle.io.Dataset` 是用来封装 Dataset的方法和行为的抽象类，通过一个索引获取指定的样本，同时对该样本进行数据处理。当继承 `paddle.io.Dataset` 来定义数据读取类时，实现如下方法：

file:///D:/Downloads/第4章（下）：基于前馈神经网络完成鸢尾花分类任务.html

1/7

- `__getitem__`：根据给定索引获取数据集中指定样本，并对样本进行数据处理；
- `__len__`：返回数据集样本个数。

代码实现如下：

```
In [1]: import numpy as np
import paddle
import paddle.io as io
from nndl import load_data

class IrisDataset(io.Dataset):
    def __init__(self, mode='train', num_train=120, num_dev=15):
        super(IrisDataset, self).__init__()
        # 调用第三章中的数据读取函数，其中不需要将标签转成one-hot类型
        X, y = load_data(shuffle=True)
        if mode == 'train':
            self.X, self.y = X[:num_train], y[:num_train]
        elif mode == 'dev':
            self.X, self.y = X[num_train:num_train + num_dev], y[num_train:num_train + num_dev]
        else:
            self.X, self.y = X[num_train + num_dev:], y[num_train + num_dev:]

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

    def __len__(self):
        return len(self.y)
```

/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/__init__.py:107: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working

from collections import MutableMapping

/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/rcsetup.py:20: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working

from collections import Iterable, Mapping

/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/colors.py:53: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working

from collections import Sized

```
In [2]: paddle.seed(12)
train_dataset = IrisDataset(mode='train')
dev_dataset = IrisDataset(mode='dev')
test_dataset = IrisDataset(mode='test')
```

W0516 19:23:24.529255 3381 device_context.cc:447] Please NOTE: device: 0, GPU Compute Capability: 8.0, Driver API Version: 11.2, Runtime API Version: 11.2

W0516 19:23:24.532312 3381 device_context.cc:465] device: 0, cuDNN Version: 8.2.

```
In [3]: # 打印训练集长度
print ("length of train set: ", len(train_dataset))

length of train set: 120
```

4.5.2.2 用DataLoader进行封装

```
In [4]: # 批量大小
batch_size = 16

# 加载数据
train_loader = io.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
dev_loader = io.DataLoader(dev_dataset, batch_size=batch_size)
test_loader = io.DataLoader(test_dataset, batch_size=batch_size)
```

4.5.3 模型构建

构建一个简单的前馈神经网络进行鸢尾花分类实验。其中输入层神经元个数为4，输出层神经元个数为3，隐含层神经元个数为6。代码实现如下：

```
In [5]: from paddle import nn

# 定义前馈神经网络
class Model_MLP_L2_V3(nn.Layer):
    def __init__(self, input_size, output_size, hidden_size):
        super(Model_MLP_L2_V3, self).__init__()
        # 构建第一个全连接层
        self.fc1 = nn.Linear(
            input_size,
            hidden_size,
            weight_attr=paddle.ParamAttr(initializer=nn.initializer.Normal(mean=0.0, std=0.01)),
            bias_attr=paddle.ParamAttr(initializer=nn.initializer.Constant(value=1.0))
        )
        # 构建第二全连接层
        self.fc2 = nn.Linear(
            hidden_size,
            output_size,
            weight_attr=paddle.ParamAttr(initializer=nn.initializer.Normal(mean=0.0, std=0.01)),
            bias_attr=paddle.ParamAttr(initializer=nn.initializer.Constant(value=1.0))
        )
        # 定义网络使用的激活函数
        self.act = nn.Sigmoid()
```

```
def forward(self, inputs):
    outputs = self.fcl(inputs)
    outputs = self.act(outputs)
    outputs = self.fc2(outputs)
    return outputs

fnn_model = Model_MLP_L2_V3(input_size=4, output_size=3, hidden_size=6)
```

4.5.4 完善Runner类

基于RunnerV2类进行完善实现了RunnerV3类。其中训练过程使用自动梯度计算，使用 DataLoader 加载批量数据，使用随机梯度下降法进行参数优化；模型保存时，使用 state_dict 方法获取模型参数；模型加载时，使用 set_state_dict 方法加载模型参数。

由于这里使用随机梯度下降法对参数优化，所以数据以批次的形式输入到模型中进行训练，那么评价指标计算也是分别在每个批次进行的，要想获得每个epoch整体的评价结果，需要对历史评价结果进行累积。这里定义 Accuracy 类实现该功能。

```
In [6]: from paddle.metric import Metric

class Accuracy(Metric):
    def __init__(self, is_logist=True):
        """
        输入：
        - is_logist: outputs是logist还是激活后的值
        """

        # 用于统计正确的样本个数
        self.num_correct = 0
        # 用于统计样本的总数
        self.num_count = 0

        self.is_logist = is_logist

    def update(self, outputs, labels):
        """
        输入：
        - outputs: 预测值， shape=[N, class_num]
        - labels: 标签值， shape=[N, 1]
        """

        # 判断是二分类任务还是多分类任务， shape[1]=1时为二分类任务， shape[1]>1时为多分类任务
        if outputs.shape[1] == 1: # 二分类
            outputs = paddle.squeeze(outputs, axis=-1)
            if self.is_logist:
                # logist判断是否大于0
                preds = paddle.cast((outputs>=0), dtype='float32')
            else:
                # 如果不是logist，判断每个概率值是否大于0.5，当大于0.5时，类别为1，否则类别为0
                preds = paddle.cast((outputs>=0.5), dtype='float32')
        else:
            # 多分类时，使用'paddle.argmax'计算最大元素索引作为类别
            preds = paddle.argmax(outputs, axis=1, dtype='int64')

        # 获取本批数据中预测正确的样本个数
        labels = paddle.squeeze(labels, axis=-1)
        batch_correct = paddle.sum(paddle.cast(preds==labels, dtype="float32")).numpy()[0]
        batch_count = len(labels)

        # 更新num_correct 和 num_count
        self.num_correct += batch_correct
        self.num_count += batch_count

    def accumulate(self):
        # 使用累计的数据，计算总的指标
        if self.num_count == 0:
            return 0
        return self.num_correct / self.num_count

    def reset(self):
        # 重置正确的数目和总数
        self.num_correct = 0
        self.num_count = 0

    def name(self):
        return "Accuracy"
```

RunnerV3类的代码实现如下：

```
In [7]: import paddle.nn.functional as F

class RunnerV3(object):
    def __init__(self, model, optimizer, loss_fn, metric, **kwargs):
        self.model = model
        self.optimizer = optimizer
        self.loss_fn = loss_fn
        self.metric = metric # 只用于计算评价指标

        # 记录训练过程中的评价指标变化情况
        self.dev_scores = []
```

```
# 记录训练过程中的损失函数变化情况
self.train_epoch_losses = [] # 一个epoch记录一次loss
self.train_step_losses = [] # 一个step记录一次loss
self.dev_losses = []

# 记录全局最优指标
self.best_score = 0

def train(self, train_loader, dev_loader=None, **kwargs):
    # 将模型切换为训练模式
    self.model.train()

    # 传入训练轮数，如果没有传入值则默认为0
    num_epochs = kwargs.get("num_epochs", 0)
    # 传入log打印频率，如果没有传入值则默认为100
    log_steps = kwargs.get("log_steps", 100)
    # 评价频率
    eval_steps = kwargs.get("eval_steps", 0)

    # 传入模型保存路径，如果没有传入值则默认为"best_model.pdparams"
    save_path = kwargs.get("save_path", "best_model.pdparams")

    custom_print_log = kwargs.get("custom_print_log", None)

    # 训练总的步数
    num_training_steps = num_epochs * len(train_loader)

    if eval_steps:
        if self.metric is None:
            raise RuntimeError('Error: Metric can not be None!')
        if dev_loader is None:
            raise RuntimeError('Error: dev_loader can not be None!')

    # 运行的step数目
    global_step = 0

    # 进行num_epochs轮训练
    for epoch in range(num_epochs):
        # 用于统计训练集的损失
        total_loss = 0
        for step, data in enumerate(train_loader):
            X, y = data
            # 获取模型预测
            logits = self.model(X)
            loss = self.loss_fn(logits, y) # 默认求mean
            total_loss += loss

            # 训练过程中，每个step的loss进行保存
            self.train_step_losses.append((global_step, loss.item()))

            if log_steps and global_step%log_steps==0:
                print(f"[Train] epoch: {epoch}/{num_epochs}, step: {global_step}/{num_training_steps}, loss: {loss.item():.5f}")

            # 梯度反向传播，计算每个参数的梯度值
            loss.backward()

            if custom_print_log:
                custom_print_log(self)

            # 小批量梯度下降进行参数更新
            self.optimizer.step()
            # 梯度归零
            self.optimizer.clear_grad()

            # 判断是否需要评价
            if eval_steps>0 and global_step>0 and \
                (global_step%eval_steps == 0 or global_step==(num_training_steps-1)):

                dev_score, dev_loss = self.evaluate(dev_loader, global_step=global_step)
                print(f"[Evaluate] dev score: {dev_score:.5f}, dev loss: {dev_loss:.5f}")

                # 将模型切换为训练模式
                self.model.train()

                # 如果当前指标为最优指标，保存该模型
                if dev_score > self.best_score:
                    self.save_model(save_path)
                    print(f"[Evaluate] best accuracy performance has been updated: {self.best_score:.5f} --> {dev_score:.5f}")
                    self.best_score = dev_score

                global_step += 1

            # 当前epoch 训练loss累计值
            trn_loss = (total_loss / len(train_loader)).item()
            # epoch粒度的训练loss保存
            self.train_epoch_losses.append(trn_loss)

        print(f"[Train] Training done!")

    # 模型评估阶段，使用'paddle.no_grad()'控制不计算和存储梯度
    @paddle.no_grad()
    def evaluate(self, dev_loader, **kwargs):
```

```

    assert self.metric is not None

    # 将模型设置为评估模式
    self.model.eval()

    global_step = kwargs.get("global_step", -1)

    # 用于统计训练集的损失
    total_loss = 0

    # 重置评价
    self.metric.reset()

    # 遍历验证集每个批次
    for batch_id, data in enumerate(dev_loader):
        X, y = data

        # 计算模型输出
        logits = self.model(X)

        # 计算损失函数
        loss = self.loss_fn(logits, y).item()
        # 累积损失
        total_loss += loss

        # 累积评价
        self.metric.update(logits, y)

    dev_loss = (total_loss/len(dev_loader))
    dev_score = self.metric.accumulate()

    # 记录验证集loss
    if global_step!=-1:
        self.dev_losses.append((global_step, dev_loss))
        self.dev_scores.append(dev_score)

    return dev_score, dev_loss

# 模型评估阶段，使用'paddle.no_grad()'控制不计算和存储梯度
@paddle.no_grad()
def predict(self, x, **kwargs):
    # 将模型设置为评估模式
    self.model.eval()
    # 运行模型前向计算，得到预测值
    logits = self.model(x)
    return logits

def save_model(self, save_path):
    paddle.save(self.model.state_dict(), save_path)

def load_model(self, model_path):
    model_state_dict = paddle.load(model_path)
    self.model.set_state_dict(model_state_dict)
```

4.5.5 模型训练

实例化RunnerV3类，并传入训练配置，代码实现如下：

```
In [8]: import paddle.optimizer as opt

lr = 0.2

# 定义网络
model = fnn_model

# 定义优化器
optimizer = opt.SGD(learning_rate=lr, parameters=model.parameters())

# 定义损失函数。softmax+交叉熵
loss_fn = F.cross_entropy

# 定义评价指标
metric = Accuracy(is_logist=True)

runner = RunnerV3(model, optimizer, loss_fn, metric)
```

使用训练集和验证集进行模型训练，共训练150个epoch。在实验中，保存准确率最高的模型作为最佳模型。代码实现如下：

```
In [9]: # 启动训练
log_steps = 100
eval_steps = 50
runner.train(train_loader, dev_loader,
             num_epochs=150, log_steps=log_steps, eval_steps = eval_steps,
             save_path="best_model.pdparams")
```



```
[Train] epoch: 0/150, step: 0/1200, loss: 1.09505
[Evaluate] dev score: 0.40000, dev loss: 1.09034
[Evaluate] best accuracy performance has been updated: 0.00000 --> 0.40000
[Train] epoch: 12/150, step: 100/1200, loss: 1.17283
[Evaluate] dev score: 0.40000, dev loss: 1.08822
[Evaluate] dev score: 0.33333, dev loss: 1.11525
[Train] epoch: 25/150, step: 200/1200, loss: 1.10550
[Evaluate] dev score: 0.40000, dev loss: 1.09292
[Evaluate] dev score: 0.40000, dev loss: 1.09518
[Train] epoch: 37/150, step: 300/1200, loss: 1.11925
[Evaluate] dev score: 0.33333, dev loss: 1.07696
[Evaluate] dev score: 0.60000, dev loss: 1.04172
[Evaluate] best accuracy performance has been updated: 0.40000 --> 0.60000
[Train] epoch: 50/150, step: 400/1200, loss: 0.98805
[Evaluate] dev score: 0.40000, dev loss: 0.94627
[Evaluate] dev score: 1.00000, dev loss: 0.79737
[Evaluate] best accuracy performance has been updated: 0.60000 --> 1.00000
[Train] epoch: 62/150, step: 500/1200, loss: 0.64998
[Evaluate] dev score: 0.80000, dev loss: 0.65108
[Evaluate] dev score: 0.93333, dev loss: 0.55086
[Train] epoch: 75/150, step: 600/1200, loss: 0.45059
[Evaluate] dev score: 0.80000, dev loss: 0.48998
[Evaluate] dev score: 0.93333, dev loss: 0.44585
[Train] epoch: 87/150, step: 700/1200, loss: 0.40733
[Evaluate] dev score: 0.93333, dev loss: 0.40810
[Evaluate] dev score: 1.00000, dev loss: 0.38619
[Train] epoch: 100/150, step: 800/1200, loss: 0.38417
[Evaluate] dev score: 0.93333, dev loss: 0.35828
[Evaluate] dev score: 1.00000, dev loss: 0.34140
[Train] epoch: 112/150, step: 900/1200, loss: 0.28922
[Evaluate] dev score: 0.93333, dev loss: 0.31777
[Evaluate] dev score: 1.00000, dev loss: 0.30399
[Train] epoch: 125/150, step: 1000/1200, loss: 0.28038
[Evaluate] dev score: 1.00000, dev loss: 0.28455
[Evaluate] dev score: 1.00000, dev loss: 0.27460
[Train] epoch: 137/150, step: 1100/1200, loss: 0.19444
[Evaluate] dev score: 0.93333, dev loss: 0.25200
[Evaluate] dev score: 1.00000, dev loss: 0.24346
[Evaluate] dev score: 1.00000, dev loss: 0.22806
[Train] Training done!
```

可视化观察训练集损失和训练集loss变化情况。

```
In [13]: import matplotlib.pyplot as plt

# 绘制训练集和验证集的损失变化以及验证集上的准确率变化曲线
def plot_training_loss_acc(runner, fig_name,
    fig_size=(16, 6),
    sample_step=20,
    loss_legend_loc="upper right",
    acc_legend_loc="lower right",
    train_color="#8E004D",
    dev_color='#E20079',
    fontsize='x-large',
    train_linestyle="--",
    dev_linestyle='--'):

    plt.figure(figsize=fig_size)

    plt.subplot(1, 2, 1)
    train_items = runner.train_step_losses[:, sample_step:]
    train_steps=[x[0] for x in train_items]
    train_losses = [x[1] for x in train_items]

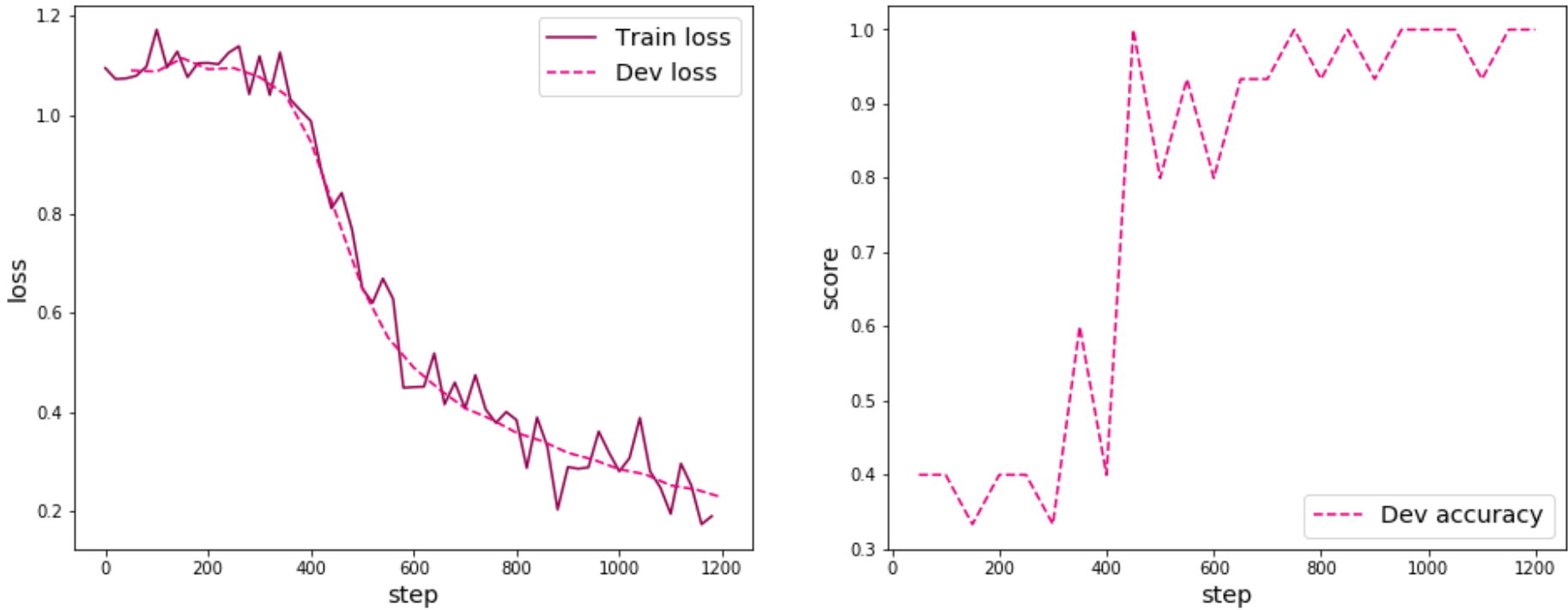
    plt.plot(train_steps, train_losses, color=train_color, linestyle=train_linestyle, label="Train loss")
    if len(runner.dev_losses)>0:
        dev_steps=[x[0] for x in runner.dev_losses]
        dev_losses = [x[1] for x in runner.dev_losses]
        plt.plot(dev_steps, dev_losses, color=dev_color, linestyle=dev_linestyle, label="Dev loss")
    # 绘制坐标轴和图例
    plt.ylabel("loss", fontsize=fontsize)
    plt.xlabel("step", fontsize=fontsize)
    plt.legend(loc=loss_legend_loc, fontsize=fontsize)

    # 绘制评价准确率变化曲线
    if len(runner.dev_scores)>0:
        plt.subplot(1, 2, 2)
        plt.plot(dev_steps, runner.dev_scores,
            color=dev_color, linestyle=dev_linestyle, label="Dev accuracy")

        # 绘制坐标轴和图例
        plt.ylabel("score", fontsize=fontsize)
        plt.xlabel("step", fontsize=fontsize)
        plt.legend(loc=acc_legend_loc, fontsize=fontsize)

    plt.savefig(fig_name)
    plt.show()

plot_training_loss_acc(runner, 'fw-loss.pdf')
```



从输出结果可以看出准确率随着迭代次数增加逐渐上升，损失函数下降。

4.5.6 模型评价

使用测试数据对在训练过程中保存的最佳模型进行评价，观察模型在测试集上的准确率以及Loss情况。代码实现如下：

```
In [11]: # 加载最优模型
runner.load_model('best_model.pdparams')
# 模型评价
score, loss = runner.evaluate(test_loader)
print("[Test] accuracy/loss: {:.4f}/{:.4f}".format(score, loss))

[Test] accuracy/loss: 0.8667/0.8319
```

4.5.7 模型预测

同样地，也可以使用保存好的模型，对测试集中的某一个数据进行模型预测，观察模型效果。代码实现如下：

```
In [12]: # 获取测试集中第一条数据
X, label = next(test_loader())
logits = runner.predict(X)

pred_class = paddle.argmax(logits[0]).numpy()
label = label[0][0].numpy()

# 输出真实类别与预测类别
print("The true category is {} and the predicted category is {}".format(label, pred_class))

The true category is [2] and the predicted category is [2]
```

4.6 小结

本章介绍前馈神经网络的基本概念、网络结构及代码实现，利用前馈神经网络完成一个分类任务，并通过两个简单的实验，观察前馈神经网络的梯度消失问题和死亡ReLU问题，以及对应的优化策略。此外，还实践了基于前馈神经网络完成鸢尾花分类任务。

4.7 实验拓展

尝试基于MNIST手写数字识别数据集，设计合适的前馈神经网络进行实验，并取得95%以上的准确率。