基于MindSpore的鸢尾花分类任务对比实验指导书

一、实验环境构建

1、登录华为云官网https://www.huaweicloud.com/?ticket=ST-4217587-ZMyac1H4moJ0pjSOHq4ubtbh-sso&locale=zh-cn，在搜索框输入对象存储服务进入控制台。（若已经创建过对象存储服务则直接跳到第4步项目创建）

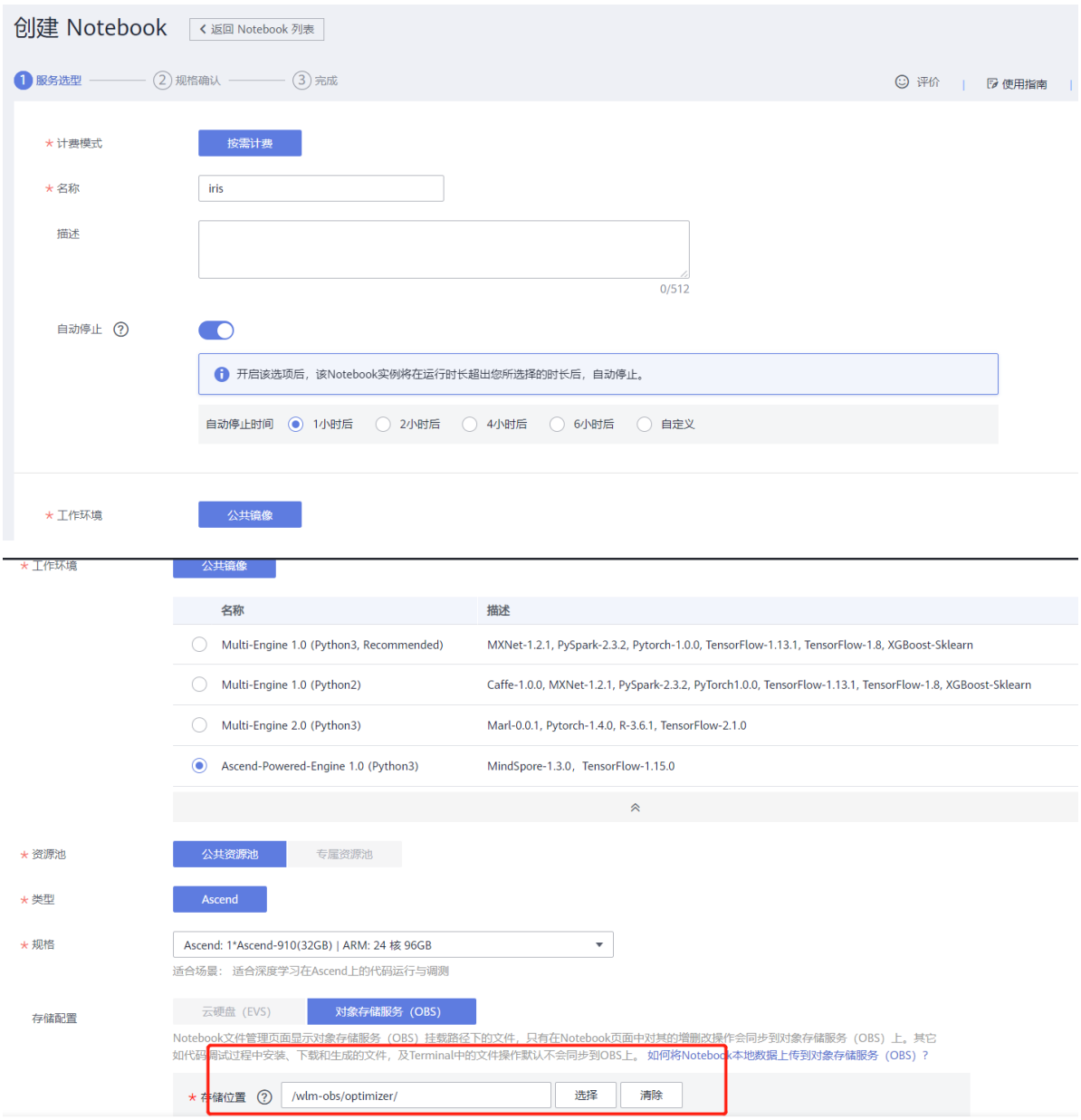
2、选择桶列表中的创建桶操作。

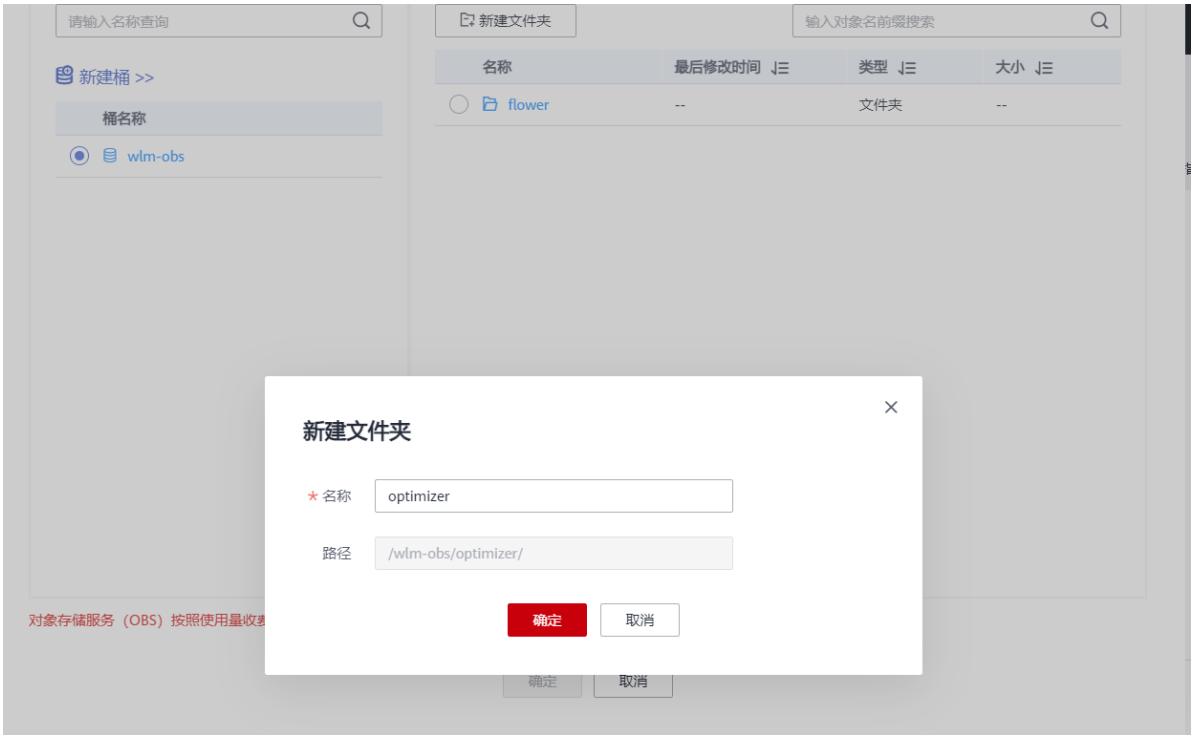
3、资源的相关配置情况如下，区域选择华北-北京四，桶名称需设置为全局唯一，桶默认私有，点击立即创建。

4、项目创建，在搜索框输入ModelArts进入控制台。

5、点击开发环境，创建Notebook实例，注意此处采用旧版Notebook创建实例界面。

6、实例配置情况如下所示，存储位置选择之前创建的桶资源，选择新建一个optimizer的文件夹用于存储相关资源。

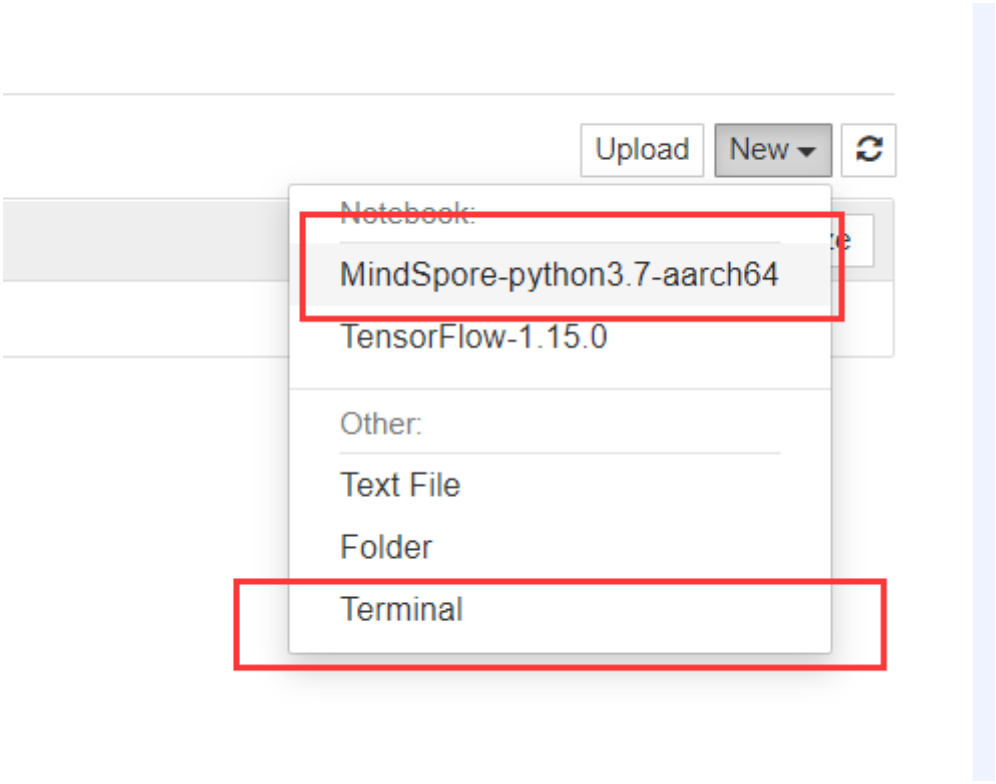




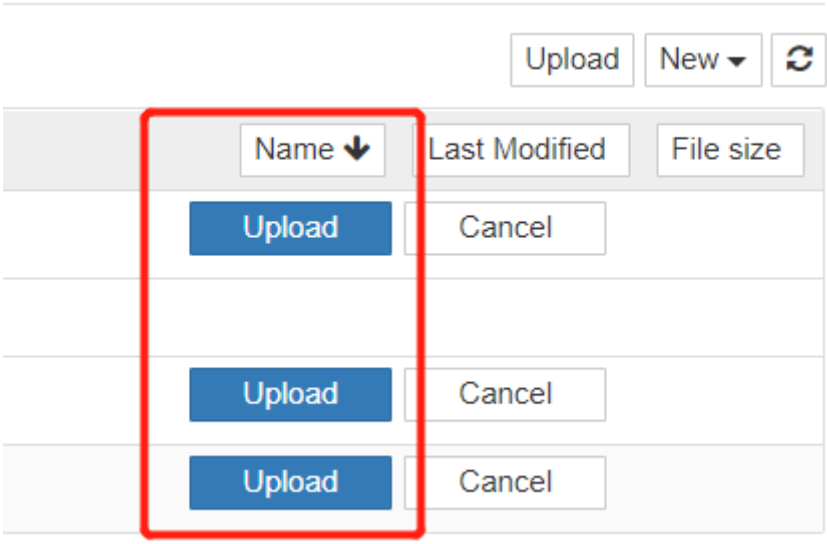
7、各项信息确认无误后，创建新环境。

8、打开Notebook实验环境进行实验。

9、在实验中可以创建相关的终端和代码实例，需要注意的是，本实验环境应选择MindSpore-python3.7的内核。



10、若有额外资源可通过Upload上传，需要注意的是，需要点击蓝色的Upload按钮才能正式将资源上传至服务器。



二、实验设计

实验核心步骤如下：

- 不同优化器的求解函数极值点效果对比，要求手写实现至少三种优化器模型。自定义待优化函数并在函数上对优化器效果进行测试。
- 不同优化器在分类实验中的效果对比，调用相关库实现至少三种优化器的分类效果对比即可

三、实验流程

1、数据集准备

本次实验使用简单iris数据进行，仅需要将其上传至服务器以备读取所需即可

2、定义待优化函数，以Beala函数为例进行分析

该步骤主要对神经网络中的常见优化器进行研究，以Beala公式为目标函数进行优化，该方程的极值点应为(3, 0.5)

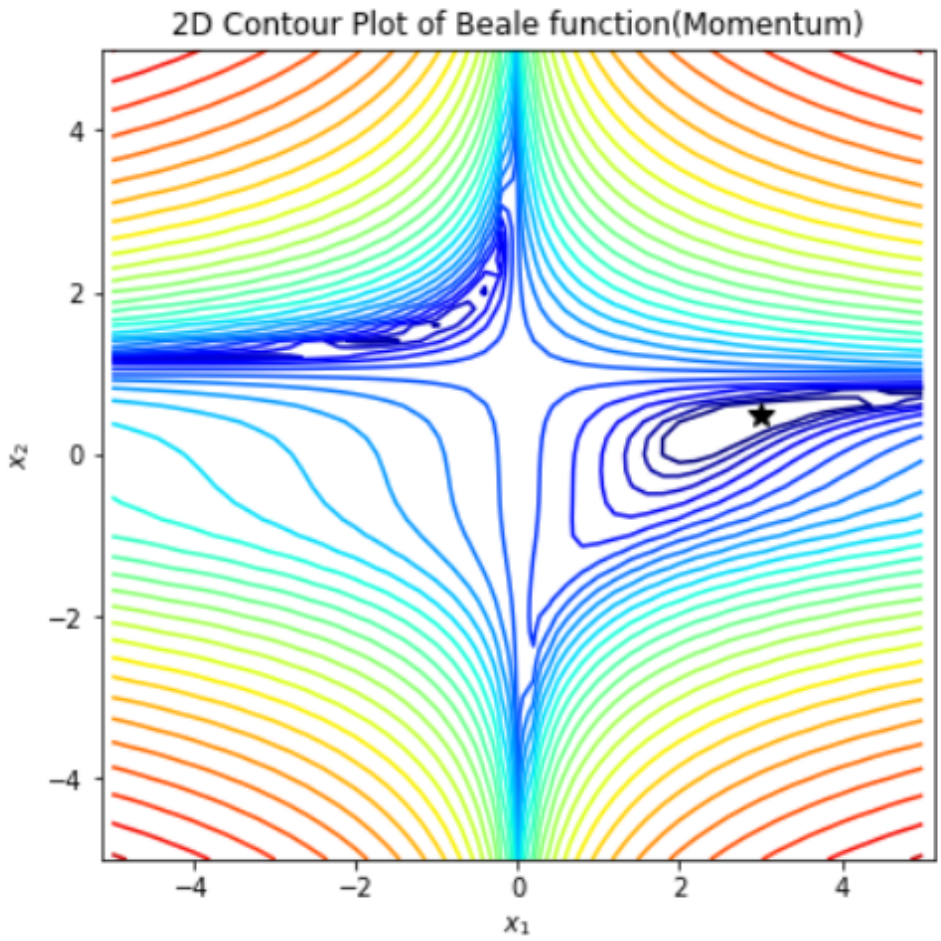
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as plt_cl  # Matplotlib的色阶条

# ------------------定义目标函数beale、目标函数的偏导函数dbeale_dx，并画出目标函数-------
--------------
#定义beale公式
def beale(x1,x2):
    return (1.5-x1+x1*x2)**2+(2.25-x1+x1*x2**2)**2+(2.625-x1+x1*x2**3)**2
#定义beale公式的偏导函数
def dbeale_dx(x1, x2):
    dfdx1 = 2*(1.5-x1+x1*x2)*(x2-1)+2*(2.25-x1+x1*x2**2)*(x2**2-1)+2*(2.625-
x1+x1*x2**3)*(x2**3-1) # 求beale公式关于x1的偏导数
    dfdx2 = 2*(1.5-x1+x1*x2)*x1+2*(2.25-x1+x1*x2**2)*(2*x1*x2)+2*(2.625-
x1+x1*x2**3)*(3*x1*x2**2) # 求beale公式关于x2的偏导数
    return dfdx1, dfdx2


# 定义画图函数
def gd_plot(x_traj):
    plt.rcParams['figure.figsize'] = [6, 6] # 窗口大小
    plt.contour(X1, X2, Y, levels=np.logspace(0, 6, 30),
                norm=plt_cl.LogNorm(), cmap=plt.cm.jet)  # 画等高线图
    plt.title('2D Contour Plot of Beale function(Momentum)') # 添加标题
    plt.xlabel('$x_1$') # x轴标签
    plt.ylabel('$x_2$') # y轴标签
    plt.axis('equal') # 设置坐标轴为正方形
    plt.plot(3, 0.5, 'k*', markersize=10) # 画出最低点
    if x_traj is not None:
        x_traj = np.array(x_traj) # 将x_traj转为数组
        plt.plot(x_traj[:, 0], x_traj[:, 1], 'k-')
# 以x_traj的第一列为x轴坐标，第二列为y轴坐标进行画图
plt.show() # 显示图像

step_x1, step_x2 = 0.2, 0.2
X1, X2 = np.meshgrid(np.arange(-5, 5 + step_x1, step_x1),
                     np.arange(-5, 5 + step_x2, step_x2))  # 将图形从-5 到 5.2，步
长为0.2 划分成网格点
Y = beale(X1, X2) # 将x1,x2坐标带入beale公式
print("目标结果 (x_1, x_2) = (3, 0.5)")
gd_plot(None) # 调用函数
```

目标结果（x_1，x_2）=（3，0.5）



3、以上述框架为模板，给出无优化器，以及如SGD等优化器在该问题上的求解结果，并对结果进行对比分析。

一些可供参考的优化器原理如下：

- SGD优化器

  梯度下降法：梯度下降（gradient descent）在机器学习中应用十分的广泛，是求解无约束优化问题最简单和最古老的方法之一。通过迭代，参数向梯度的反方向更新，直到收敛。

  $$W_{new} = W - \eta \frac{\partial J(W)}{\partial W}$$

  缺点：

  - 有可能会陷入局部最小值；
  - 不会收敛，最终会一直在最小值附近波动，并不会达到最小值并停留在此；
  - 下降速度慢；
  - 选择合适的学习率比较困难；
  - 在所有方向上统一的缩放梯度，不适用于稀疏数据；

- Momentum优化器

  Momentum：是动量优化法中的一种（Momentum、NAG），即使用动量(Momentum)的随机梯度下降法(SGD)，主要思想是引入一个积攒历史梯度信息的动量来加速SGD。其参数优化公式如下所示：

$$v_{new} = \gamma v - \eta \frac{\partial J(W)}{\partial W}$$
$$W_{new} = W + v_{new}$$

其中$\frac{\partial J(W)}{\partial W}$ 表示损失函数 J 关于参数W的梯度；η表示学习率；γ表示动量的大小，一般取值为0.9。

这个算法和之前的梯度下降法(SGD)相比，唯一不同的就是多了一个γv。这一改动使Momentum会观察历史梯度，若当前梯度的方向与历史梯度一致（表明当前样本不太可能为异常点），则会增强这个方向的梯度；若当前梯度与历史梯方向不一致，则梯度会衰减。一种形象的解释是：我们把一个球推下山，球在下坡时积聚动量，在途中变得越来越快，γ可视为空气阻力，若球的方向发生变化，则动量会衰减。

优点：

- 参考了历史梯度，增加了稳定性；
- 由于引入加速动量，加快收敛速度。下降初期时，使用上一次参数更新，下降方向一致，乘上较大的γ能够进行很好的加速；
- 还有一定摆脱局部最优的能力。下降中后期时，在局部最小值来回震荡的时候，梯度趋近于0，γ使得更新幅度增大，跳出陷阱（局部最优）

- 自适应优化器

  自适应学习率优化算法主要有：AdaGrad算法，RMSProp算法，Adam算法以及AdaDelta算法。

  AdaGrad

  AdaGrad的基本思想是对每个变量用不同的学习率。这个学习率在一开始比较大，用于快速梯度下降。随着优化过程的进行，对于已经下降很多的变量，则减缓学习率，对于还没怎么下降的变量，则保持一个较大的学习率。其参数优化公式如下所示：

$$G_{new} = G + (\frac{\partial J(W)}{\partial W})^2$$
$$W_{new} = W - \frac{\eta}{(\sqrt{G_{new}} + \varepsilon)} \cdot \frac{\partial J(W)}{\partial W}$$

  其中$\frac{\partial J(W)}{\partial W}$ 表示损失函数 J 关于参数W的梯度；η表示学习率，一般取值0.01；ε是一个很小的数，防止分母为0；G_{new}表示了前t 步参数W梯度的平方累加。把沿路的Gradient的平方根，作为Regularizer。分母作为Regularizer项的工作机制如下：

  1. 训练前期，梯度较小，使得Regularizer项很大，放大梯度。[激励阶段]
  2. 训练后期，梯度较大，使得Regularizer项很小，缩小梯度。[惩罚阶段]

  优点：

  - 在数据分布稀疏的场景，能更好利用稀疏梯度的信息，比标准的SGD算法更有效地收敛；
  - 对每个变量用不同的学习率，对输入参数学习率的依赖小，容易调节参数；

  缺点：

  - 主要缺陷来自分母项的对梯度平方不断累积，随之时间步地增加，分母项越来越大，最终导致学习率收缩到太小无法进行有效更新；

  RMSProp

  为了解决 Adagrad 学习率急剧下降问题，RMSProp保留过去梯度的微分平方数项，旨在消除梯度下降中的摆动。与Momentum的效果一样，某一维度的导数比较大，则指数加权平均就大，某一维度的导数比较小，则其指数加权平均就小，这样就保证了各维度导数都在一个量级，进而减少了摆动。允许使用一个更大的学习率η。其参数优化公式如下所示：

$$v_{new} = \gamma \cdot v + (1-\gamma) \cdot (\frac{\partial J(W)}{\partial W})^2$$

$$W_{new} = W - \frac{\eta}{(\sqrt{v_{new}} + \varepsilon)} (\frac{\partial J(W)}{\partial W})$$

其中$\frac{\partial J(W)}{\partial W}$ 表示损失函数 J 关于参数W的梯度；η表示学习率，一般取值0.001；ε是一个很小的数，防止分母为0；γ表示动量的大小,一般取值为0.9。

Adam

Adam算法是另一种计算每个参数的自适应学习率的方法。相当于 RMSprop + Momentum。除了像RMSprop存储了过去梯度的平方 v_t 的指数衰减平均值 ，也像 momentum 一样保持了过去梯度 m_t 的指数衰减平均值。其参数优化公式如下所示：

$$m_{new} = \beta_1 m + (1-\beta_1)(\frac{\partial J(W)}{\partial W})$$

$$v_{new} = \beta_2 v + (1-\beta_2)(\frac{\partial J(W)}{\partial W})^2$$

由于$\frac{m_0}{v_0}$初始化为0，会导致$\frac{m_{new}}{v_{new}}$偏向于0，尤其在训练初期阶段，所以，此处需要对梯度均值$\frac{m_{new}}{v_{new}}$进行偏差纠正，降低偏差对训练初期的影响。

$$\hat{m_{new}} = m_{new} / (1-\beta_1)$$

$$\hat{v_{new}} = v_{new} / (1-\beta_2)$$

$$W_{new} = W - \eta \frac{1}{\sqrt{\hat{v_{new}}} + \varepsilon} \hat{m_{new}}$$

其中$\frac{\partial J(W)}{\partial W}$ 表示损失函数 J 关于参数W的梯度；η表示学习率，一般取值0.001；ε是一个很小的数，一般取值10e-8，防止分母为0；β_1 β_2分别表示一阶和二阶动量的大小，一般取值为$\beta_1 = 0.9$ $\beta_2 = 0.99$。

优点

- 能够克服AdaGrad梯度急剧减小的问题，在很多应用中都展示出优秀的学习率自适应能力；
- 实现简单，计算高效，对内存需求少；
- 参数的更新不受梯度的伸缩变换影响；
- 超参数具有很好的解释性，且通常无需调整或仅需很少的微调；
- 更新的步长能够被限制在大致的范围内（初始学习率）；
- 能自然地实现步长退火过程（自动调整学习率）；
- 很适合应用于大规模的数据及参数的场景；
- 适用于不稳定目标函数；
- 适用于梯度稀疏或梯度存在很大噪声的问题

4、鸢尾花分类实验，导入MindSpore模块和辅助模块

```
import csv
import os
import time

import numpy as np
from easydict import EasyDict as edict
```

```
from matplotlib import pyplot as plt

import mindspore
from mindspore import nn
from mindspore import context
from mindspore import dataset
from mindspore.train.callback import TimeMonitor, LossMonitor
from mindspore import Tensor
from mindspore.train import Model
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

context.set_context(mode=context.GRAPH_MODE, device_target="Ascend")  # 设定运行模式为静态图模式，并且运行设备为昇腾芯片
```

5、模型参数设置，仅给出一个格式样例（具体可根据自己需求进行调整）

```
#变量定义
cfg = edict({
    'data_size': 150,
    'train_size': 120,       #训练集大小
    'test_size': 30 ,        #测试集大小
    'feature_number': 4,        #输入特征数
    'num_class': 3,      #分类类别
    'batch_size': 30,    #批次大小
    'data_dir':    'iris.data',      # 数据集路径
    'save_checkpoint_steps': 5,                    #多少步保存一次模型
    'keep_checkpoint_max': 1,                        #最多保存多少个模型
    'out_dir_no_opt':    './model_iris/no_opt',            #保存模型路径，无优化器模型
    'out_dir_sgd':    './model_iris/sgd',           #保存模型路径,SGD优化器模型
    'out_dir_momentum':    './model_iris/momentum',           #保存模型路径，momentum模型
    'out_dir_adam':    './model_iris/adam',           #保存模型路径，adam优化器模型
    'output_prefix': "checkpoint_fashion_forward"      #保存模型文件名
})
```

6、读取数据并根据MindSpore模型需求进行预处理，共150条数据，将数据集的4个属性作为自变量X。将数据集的3个类别映射为{0, 1，2}，作为因变量Y。使用MindSpore GeneratorDataset接口将numpy.ndarray类型的数据转换为 Dataset。

```
#鸢尾花数据集，本数据集共有150个带标签的数据
with open(cfg.data_dir) as csv_file:
    data = list(csv.reader(csv_file, delimiter=','))

label_map = {'setosa': 0,'versicolor': 1,'virginica':2 }
#分别获取数据中的特征值X和标签值Y
X = np.array([[float(x) for x in s[:-1]] for s in data[:cfg.data_size]],
np.float32)
Y = np.array([label_map[s[-1]] for s in data[:cfg.data_size]], np.int32)

# 将数据集分为训练集120条，测试集30条。
train_idx = np.random.choice(cfg.data_size, cfg.train_size, replace=False)
test_idx = np.array(list(set(range(cfg.data_size)) - set(train_idx)))
```

```
X_train, Y_train = X[train_idx], Y[train_idx]
X_test, Y_test = X[test_idx], Y[test_idx]
```

7、定义训练，测试以及预测过程。

构建训练函数，定义训练及测试的输出格式。对比分析不同优化器下对鸢尾花数据的分类实际效果情况。一个可供参考的输出样例如下：

```
------------------Momentum优化器----------------------
============== Starting Training ==============
epoch: 1 step: 4, loss is 1.0456764
epoch: 2 step: 4, loss is 0.9358957
epoch: 3 step: 4, loss is 0.84239113
epoch: 4 step: 4, loss is 0.72849303
epoch: 5 step: 4, loss is 0.6306475
epoch: 6 step: 4, loss is 0.59610367
epoch: 7 step: 4, loss is 0.530419
epoch: 8 step: 4, loss is 0.5638479
epoch: 9 step: 4, loss is 0.47823143
epoch: 10 step: 4, loss is 0.46555394
epoch: 11 step: 4, loss is 0.49670792
epoch: 12 step: 4, loss is 0.5009627
epoch: 13 step: 4, loss is 0.45189294
epoch: 14 step: 4, loss is 0.45233986
epoch: 15 step: 4, loss is 0.39280808
epoch: 16 step: 4, loss is 0.34712094
epoch: 17 step: 4, loss is 0.40795138
epoch: 18 step: 4, loss is 0.38752627
epoch: 19 step: 4, loss is 0.383623
epoch: 20 step: 4, loss is 0.40174365
{'acc': 0.7666666666666667}
第0个sample预测结果： versicolor    真实结果： versicolor
第1个sample预测结果： setosa    真实结果： setosa
第2个sample预测结果： virginica    真实结果： virginica
第3个sample预测结果： setosa    真实结果： setosa
第4个sample预测结果： setosa    真实结果： setosa
第5个sample预测结果： virginica    真实结果： virginica
第6个sample预测结果： virginica    真实结果： virginica
第7个sample预测结果： virginica    真实结果： versicolor
第8个sample预测结果： virginica    真实结果： versicolor
```

8、实验总结，对比分析不同优化器适用范围及性能优劣。

四、实验提交材料及评分标准

1、实验提交材料

- 可运行的notebook代码（手写优化器函数优化实验，MindSpore框架调库鸢尾花优化器对比分析实验）
- 实验报告（是评定最终实验成绩的主要依据）

2、评分标准

得分	评价依据
0分	抄袭或被抄袭，未提交实验相关材料
1-4分	未完成第二节实验设计的全部内容，根据完成内容量情况进行评分
5分	完成实验全部内容，但实验报告缺少必要的叙述以及中间结果，例如函数极值点的可视化对比等
6分	完成全部实验内容，实验报告内容详实，有自己对模型的探索和尝试（优化器的参数探索，不同优化器间不同场景下的探索）