

Exercise 1

Increasing the range of all parameters for iterations and the max resources to 100 seem to yield a better score although very marginally, 0.97333 to 0.98. Perhaps increasing the amount of resources into the thousands will improve the score further although that will consume a lot of time.

```
In [1]: from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.experimental import enable_halving_search_cv # noqa
from sklearn.model_selection import HalvingRandomSearchCV
from scipy.stats import randint
import numpy as np

#Original code
X, y = load_iris(return_X_y=True)
clf = RandomForestClassifier(random_state=0)
np.random.seed(0)

param_distributions = {"max_depth": [3, None],
                       "min_samples_split": randint(2, 11)}

search = HalvingRandomSearchCV(clf, param_distributions,
                               resource='n_estimators',
                               max_resources=10,
                               random_state=0).fit(X, y)

print(search.best_params_)

print(search.score(X, y))

{'max_depth': 3, 'min_samples_split': 3, 'n_estimators': 9}
0.9733333333333334
```

```
In [ ]: #Altered parameters

param_distributions = {"max_depth": [11,13,15,17,19,21,23,25],
                       "min_samples_split": randint(2, 60)}

search = HalvingRandomSearchCV(clf, param_distributions,
                               resource='n_estimators',
                               max_resources=100,
                               random_state=0).fit(X, y)

print(search.best_params_)

print(search.score(X, y))

{'max_depth': 11, 'min_samples_split': 10, 'n_estimators': 81}
0.98
```

Exercise 2

After much attempt in trying to improve the score by changing the various range of estimator numbers, learning rate and number of iterations, the best results seem to remain the one provided by SVC (the best result from Adaboost classifier is the exact same as the SVC). Not only did the Adaboost take significantly longer, the best result possible is the same as SVC classifier's result; any different result are always lower. So, the SVC classifier is highly optimized and very fast.

```
In [3]: from skopt import BayesSearchCV
# parameter ranges are specified by one of below
from skopt.space import Real, Categorical, Integer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

data= load_iris()
X=data.data
y=data.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.75,
                                                    random_state=0)

# Log-uniform: understand as search over  $p = \exp(x)$  by varying  $x$ 
opt = BayesSearchCV(
    SVC(),
    {
        'C': Real(1e-6, 1e+6, prior='log-uniform'),
        'gamma': Real(1e-6, 1e+1, prior='log-uniform'),
        'degree': Integer(1,8),
        'kernel': Categorical(['linear', 'poly', 'rbf']),
    },
    n_iter=32,
    random_state=0
)

# executes bayesian optimization
_ = opt.fit(X_train, y_train)

# model can be saved, used for predictions or scoring
print("SVC classifier results:",opt.score(X_test, y_test))
```

SVC classifier results: 0.9736842105263158

```
In [19]: opt = BayesSearchCV(
    AdaBoostClassifier(),
    {
        'n_estimators': Integer(50, 500),
        'learning_rate': Real(0.01, 1.0,prior='log-uniform'),

    },
    n_iter=100,
    random_state=0
)

# executes bayesian optimization
_ = opt.fit(X_train, y_train)

# model can be saved, used for predictions or scoring
print("Adaboost classifier results:",opt.score(X_test, y_test))
```

Adaboost classifier results: 0.9736842105263158

Exercise 3

Reduced the number of epochs and trials in order to save time, we can still compare the functions since they're all running the same parameters.

Results: CELU > ReLU > Tanh > Sigmoid. There seems to be a consistent approximately 0.03 difference between one activation and its neighbor.

```
In [24]: import os

import optuna
from optuna.trial import TrialState
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.utils.data
from torchvision import datasets
from torchvision import transforms

DEVICE = torch.device("cpu")
BATCHSIZE = 128
CLASSES = 10
DIR = os.getcwd()
EPOCHS = 5
N_TRAIN_EXAMPLES = BATCHSIZE * 10
N_VALID_EXAMPLES = BATCHSIZE * 5
ACTIVATIONS = {
    "ReLU": nn.ReLU,
    "Tanh": nn.Tanh,
    "CELU": nn.CELU,
    "Sigmoid": nn.Sigmoid,
}

def define_model(trial, activation_cls):
    n_layers = trial.suggest_int("n_layers", 1, 3)
    layers = []

    in_features = 28 * 28
    for i in range(n_layers):
        out_features = trial.suggest_int(f"n_units_l{i}", 4, 128)
        layers.append(nn.Linear(in_features, out_features))
        layers.append(activation_cls())

        p = trial.suggest_float(f"dropout_l{i}", 0.2, 0.5)
        layers.append(nn.Dropout(p))

        in_features = out_features

    layers.append(nn.Linear(in_features, CLASSES))
    layers.append(nn.LogSoftmax(dim=1))

    return nn.Sequential(*layers)

def get_mnist():
    # Load FashionMNIST dataset.
    train_loader = torch.utils.data.DataLoader(
        datasets.FashionMNIST(DIR, train=True, download=True, transform=transforms.ToTensor(),
        batch_size=BATCHSIZE,
        shuffle=True,
    )
    valid_loader = torch.utils.data.DataLoader(
        datasets.FashionMNIST(DIR, train=False, transform=transforms.ToTensor()),
        batch_size=BATCHSIZE,
        shuffle=True,
    )
```

```

    return train_loader, valid_loader

def objective(trial, activation_cls):
    model = define_model(trial, activation_cls).to(DEVICE)

    optimizer_name = trial.suggest_categorical("optimizer", ["Adam", "RMSprop", "SGD"])
    lr = trial.suggest_float("lr", 1e-5, 1e-1, log=True)
    optimizer = getattr(optim, optimizer_name)(model.parameters(), lr=lr)

    train_loader, valid_loader = get_mnist()

    for epoch in range(EPOCHS):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            if batch_idx * BATCHSIZE >= N_TRAIN_EXAMPLES:
                break

            data = data.view(data.size(0), -1).to(DEVICE)
            target = target.to(DEVICE)

            optimizer.zero_grad()
            output = model(data)
            loss = F.nll_loss(output, target)
            loss.backward()
            optimizer.step()

        model.eval()
        correct = 0
        with torch.no_grad():
            for batch_idx, (data, target) in enumerate(valid_loader):
                if batch_idx * BATCHSIZE >= N_VALID_EXAMPLES:
                    break

                data = data.view(data.size(0), -1).to(DEVICE)
                target = target.to(DEVICE)

                output = model(data)
                pred = output.argmax(dim=1, keepdim=True)
                correct += pred.eq(target.view_as(pred)).sum().item()

        accuracy = correct / min(len(valid_loader.dataset), N_VALID_EXAMPLES)
        trial.report(accuracy, epoch)

        if trial.should_prune():
            raise optuna.exceptions.TrialPruned()

    return accuracy

if __name__ == "__main__":
    #optuna.logging.set_verbosity(optuna.logging.ERROR) #Silence progress output to not fi
    results = {}

    for name, activation_cls in ACTIVATIONS.items():
        print(f"\nRunning study for {name}")

        study = optuna.create_study(direction="maximize")
        study.optimize(
            lambda trial: objective(trial, activation_cls),
            n_trials=30,
            timeout=600,
        )

        best_trial = study.best_trial

```

```
results[name] = best_trial.value

print(f"Best accuracy for {name}: {best_trial.value:.4f}")

print("Activation comparison")
for act, acc in results.items():
    print(f"{act:8s}: {acc:.4f}")
```

Running study for ReLU
Best accuracy for ReLU: 0.7688

Running study for Tanh
Best accuracy for Tanh: 0.7359

Running study for CELU
Best accuracy for CELU: 0.8031

Running study for Sigmoid
Best accuracy for Sigmoid: 0.6969
Activation comparison
ReLU : 0.7688
Tanh : 0.7359
CELU : 0.8031
Sigmoid : 0.6969