Miniproject 1: Bayesian Optimization

02463 Active machine learning and agency

Ida Lund Ragaart (s204010), Simon Stenbæk Jensen (s214592),
Katharina Strauss Søgaard (s214634) & Cecilie Dahl Hvilsted (s214605)

1 Abstract

When classifying images using a neural network as a classification model, the value of several different hyperparameters must be known. But the optimization process used for choosing the optimal hyperparameters can be very computationally expensive. We will consider Bayesian Optimization as the method of performing the optimization process. This project aims to find the optimal number of neurons in 2 hidden layers and the optimal activation function of 'ReLu' and 'Sigmoid' for each hidden layer. We found that, when using the optimal hyperparameters found by Bayesian Optimization, the classification model performed with a maximum accuracy of 82 % by using one of the acquisition functions: Expected Improvement and Lower Confidence Bound.

2 Introduction

This mini project explores a classification model of the Fashion-MNIST dataset [4] made from Zalando's article images. Using a neural network as our machine learning model, we seek to find the optimal value for 4 different hyperparameters in the 2 hidden layers of the neural network. When the optimal hyperparameters are determined, a comparison of the 3 different acquisition functions 'Maximum Probability of Improvement' (MPI), 'Expected Improvement' (EI), and 'Lower confidence bound' (LCB) is then performed to determine which of the 3 acquisition functions solves the classification problem with the highest accuracy.

3 Methods

The goal of this project is to optimize the hyperparameters of a neural network using the Python GPyOpt package. The GPyOpt package holds a lot of functionality for Bayesian optimization, including finding the best combination of hyperparameters that maximize the performance of the neural network. The Jupyter notebook is written in Python 3 and the neural network is trained and tested on the Fashion MNIST dataset, which consists of 70,000 grayscale images, split into a test set (10,000 rows) and a training set (60,000 rows) [4].

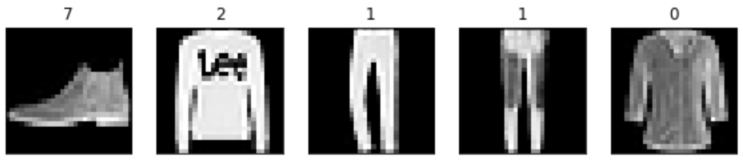

Figure 1: Subset of dataset

The images are of size 28x28 pixels, and each pixel is represented as an integer value between 0 and 255. The dataset is divided into 10 classes, each corresponding to a specific type of clothing item. A few examples of these can be seen on figure 1. The hyperparameters that are optimized are the number of neurons in the 2 hidden layers of the neural network as well as the activation function used in each of them. The range of neurons being considered is from 1 to 100, and the two activation functions being tested are 'Sigmoid' and 'ReLu'.

Using the Python package Keras, we define a neural network architecture consisting of an input layer, two dense layers and an output layer with 10 neurons holding the Softmax activation function. The neural network is implemented using code inspired by an article by Muhammad Ardi[1] but has been modified to fit our specific objective. Then, a Gaussian Process is fitted to the data to estimate the objective function, where Bayesian Optimization is performed on the neural network, to find the optimal hyperparameters. The optimization procedure runs for a maximum of 15 iterations and with an exploration rate equal to 0.5, meaning equal exploration and exploitation. The loss and accuracy is computed for every epoch and the best set of hyperparameters obtained using the acquisition function in question is returned. This is done for each acquisition function. Each of the acquisition functions uses the Gaussian process to output the next sample point in order to find the minimum of the objective function. The Bayesian Optimization is implemented by a modification of the code given in Exercise 3 by Kristoffer Madsen [2] which uses the GPyOpt module for Bayesian Optimization [3]. This method uses a standard Gaussian Process (GP) with the Matérn 5/2 kernel, as seen in appendix A, which is a standard kernel for modeling real data and smooth functions. The labeled images in the data set make it suitable for supervised learning algorithms. The complete code is given in appendix B.

4 Results

The Bayesian Optimization returned optimal hyperparameters, which are given below, showing the number of neurons in each hidden layer, the activation function for each layer, as well as the classification accuracy for each acquisition function:

Acquisition function	**MPI**	**EI**	**LCB**
No. neurons (1st layer)	38	99	49
No. neurons (2nd layer)	80	99	38
Activation function (1st layer)	Sigmoid	ReLu	ReLu
Activation function (2nd layer)	Sigmoid	ReLu	ReLu
Accuracy of classification	0.74	0.82	0.82

Table 1: Results

In the Bayesian Optimization process, the choice of activation function is a categorical value, meaning that it takes a binary number such that when equal to 0 it is using the 'Sigmoid' activation function, and when equal to 1 the 'ReLu' activation function.

For each acquisition function a plot of the optimal number of neurons in each hidden layer is plotted against each other in figure 2 to inspect the certainty of their outcome when the exploration rate and exploitation rate are equal.

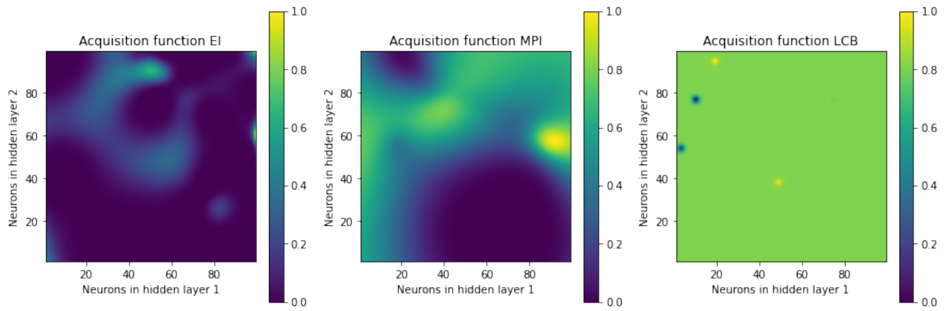

Figure 2: Acquisition functions

5 Discussion

Considering the acquisition plots in figure 2 as well as the accuracy's in table 1 it seems that the acquisition function EI is more certain than the others of where to sample the next point, as it holds very little yellow shades which restricts the amount of possible new sample points. Compared to acquisition function LCB which reveals 2 next sample points to be optimal, while almost every other combination of the number of neurons in the hidden layers also seems to be a fair new sample point (despite the 2 dark spots). When using the optimal hyperparameters in the neural network, the classification models using EI and LCB perform with an equal accuracy of 82 %. This can be explained by the fact that they are very certain of which of the next sample points will be optimal (due to the small amount of yellow on the plots), while the acquisition function MPI shows more optimal points to choose between and thereby has a harder time choosing the actual optimal next sample point (resulting in an accuracy of 74 %).

Through the optimization of the hyperparameters it is, for all 3 acquisition functions, found that they perform the best when the same activation function is used on both the hidden layers in the neural network. The three methods used quite a different number of neurons, ranging from 38 all the way to 99 in one layer. This indicates that there is not one optimal solution, as both EI and LCB ended up with the same accuracy despite having very different numbers of neurons. We could therefore have explored how the number of neurons affects the results. Ideally, we would use fewer neurons to avoid having unnecessary neurons.

6 Conclusion

From this study it is found that in a classification problem on the Fashion MNIST dataset, a neural network performs the best if the activation functions used in the all hidden layers of a neural network are the same, as well as using one of the acquisition functions 'Expected Improvement' or 'Lower Confidence Bound' during the optimization of the objective function.

7 References

[1] Muhammad Ardi. Simple neural network on mnist handwritten digit dataset. 2020.

[2] Kristoffer Madsen. *BayesianOptimization_Exercise_3_solutions*, 2023.

[3] Sheffield University. *GpyOpt: Bayesian optimization in Python*, 2018.

[4] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

8 Learning outcome

During this mini project, we have learned how Bayesian Optimization is applied to a neural network in order to solve a classification problem optimally (and in this case choosing a large number of different hyperparameters, that are possible to optimize).

We have learned how different the results of using the different acquisition functions can be, despite ending up with similar accuracies. This showed us that it can be challenging to choose just one to use and that it might be beneficial to test all of them before choosing.

As we were running our script the plots showing the acquisition functions changed drastically, which gave us an insight into how different it could behave each time. This was difficult to understand as it sometimes seemed as if the functions were random as to how well they behaved.

Throughout the project, we also gained an understanding of optimizing neural networks, which can be complex. The large number of parameters involved and the nature of the model can make it difficult to find the optimal set of hyperparameters, even with Bayesian Optimization.

We have also learned to implement Bayesian optimization in python, on a neural network, and construct a neural network with `Keras`.

Overall, the mini-project provided insights on the challenges and advantages of optimizing neural networks with Bayesian optimization.

9 Appendix

A Matérn 5/2 kernel

The Matérn 5/2 kernel is given by the regular Matérn kernel:

$$K_{\text{Matérn},v}(d) = \sigma^2 \frac{2^{I-v}}{\Gamma(v)} (\sqrt{2v}\frac{d}{l})^v K_v(\sqrt{2v}\frac{d}{l}) \tag{1}$$

where the parameter v is fixed at 5/2, making the kernel:

$$K_{\text{Matérn},5/2}(d) = \sigma^2 (1 + \frac{\sqrt{5}d}{l} + \frac{5d^2}{3l^2}) \exp(-\frac{\sqrt{5}d}{l}) \tag{2}$$

where d is the distance measure between x_i and x_j by the euclidian distance $\sqrt{||x_i - x_j||^2}$ and l is a parameter affecting the smoothness.

B Code

```
# %%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.autograd import Variable

import torchvision
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
from sklearn.metrics import confusion_matrix

import keras
from keras import layers
from keras.models import Sequential
from keras.utils import to_categorical
from matplotlib import pyplot
from keras.datasets import fashion_mnist
#!pip install GPy
#!pip install GPyOpt
import GPyOpt

# %%
# Dataset
# https://becominghuman.ai/simple-neural-network-on-mnist-handwritten-digit-
    dataset-61e47702ed25
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
print(X_train.shape)
print(X_test.shape)

# %%
# Display some images
fig, axes = plt.subplots(ncols=5, sharex=False,
    sharey=True, figsize=(10, 4))
for i in range(5):
    axes[i].set_title(y_train[i])
    axes[i].imshow(X_train[i], cmap='gray')
    axes[i].get_xaxis().set_visible(False)
    axes[i].get_yaxis().set_visible(False)
plt.show()

# %%
# One of K coding
# Convert y_train into one-hot format
temp = []
for i in range(len(y_train)):
    temp.append(to_categorical(y_train[i], num_classes=10))
y_train = np.array(temp)
# Convert y_test into one-hot format
temp = []
for i in range(len(y_test)):
    temp.append(to_categorical(y_test[i], num_classes=10))
y_test = np.array(temp)

# %%
print(y_train.shape)
print(y_test.shape)

# %%
# Neural network
model = Sequential()
model.add(layers.Flatten(input_shape=(28,28)))
model.add(layers.Dense(5, activation='sigmoid'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()

# %%
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['
    accuracy'])

# %%
model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))

# %%
predictions = model.predict(X_test)
predictions = np.argmax(predictions, axis=1)
print(predictions)

# %%
fig, axes = plt.subplots(ncols=10, sharex=False,
                         sharey=True, figsize=(20, 4))
for i in range(10):
    axes[i].set_title(predictions[i])
    axes[i].imshow(X_test[i], cmap='gray')
    axes[i].get_xaxis().set_visible(False)
    axes[i].get_yaxis().set_visible(False)
plt.show()

# %%
#activation
activation1 = (0,1)
activation2 = (0,1)
```

```
# max_features = ('log2', 'sqrt', None)
neuron1 = tuple(np.arange(1,100,1, dtype= np.int))
# criterion = ('gini', 'entropy')
neuron2 = tuple(np.arange(1,100,1, dtype= np.int))


# define the dictionary for GPyOpt
domain = [{'name': 'neuron1', 'type': 'discrete', 'domain':neuron1},
          {'name': 'neuron2', 'type': 'discrete', 'domain': neuron2},
          {'name': 'activation1', 'type': 'categorical', 'domain': activation1},
          {'name': 'activation2', 'type': 'categorical', 'domain': activation2}]


## we have to define the function we want to maximize --> validation accuracy,
## note it should take a 2D ndarray but it is ok that it assumes only one point
## in this setting
def objective_function(x):
    #print(x)
    # we have to handle the categorical variables that is convert 0/1 to labels
    # log2/sqrt and gini/entropy
    param = x[0]
    print(param)
    # we have to handle the categorical variables
    if param[2] == 0:
        act1 = 'sigmoid'
    elif param[2] == 1:
        act1 = 'relu'
    else:
        act1 = None

    if param[3] == 0:
        act2 = 'sigmoid'
    elif param[3] == 1:
        act2 = 'relu'
    else:
        act2 = None

    #create the model
    model = Sequential()
    model.add(layers.Flatten(input_shape=(28,28)))
    model.add(layers.Dense(int(param[0]), activation=act1))
    model.add(layers.Dense(int(param[1]), activation=act2))
    model.add(layers.Dense(10, activation='softmax'))
    # fit the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['
        accuracy'])
    model = model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test
        ))
    print(model.history['val_accuracy'][-1])
    return - model.history['val_accuracy'][-1]


opt = GPyOpt.methods.BayesianOptimization(f = objective_function,   # function to
    optimize
```

```
                                                domain = domain,          # box-
                                                    constrains of the problem
                                                acquisition_type = 'EI' ,       #
                                                    Select acquisition function MPI,
                                                     EI, LCB
                                               )
opt.acquisition.exploration_weight=0.5

opt.run_optimization(max_iter = 15)

x_best = opt.X[np.argmin(opt.Y)]
print('Done 1')

opt2 = GPyOpt.methods.BayesianOptimization(f = objective_function,   # function
    to optimize
                                                domain = domain,          # box-
                                                    constrains of the problem
                                                acquisition_type = 'MPI' ,       #
                                                    Select acquisition function MPI,
                                                     EI, LCB
                                               )
opt2.acquisition.exploration_weight=0.5

opt2.run_optimization(max_iter = 15)

x_best2 = opt2.X[np.argmin(opt2.Y)]
print('DONE 2')


opt3 = GPyOpt.methods.BayesianOptimization(f = objective_function,   # function
    to optimize
                                                domain = domain,          # box-
                                                    constrains of the problem
                                                acquisition_type = 'LCB' ,       #
                                                    Select acquisition function MPI,
                                                     EI, LCB
                                               )
opt3.acquisition.exploration_weight=0.5

opt3.run_optimization(max_iter = 15)

x_best3 = opt3.X[np.argmin(opt3.Y)]

print("The best parameters obtained (EI): neuron1=" + str(x_best[0]) + ", neuron2
    =" + str(x_best[1]) + ", activation1=" + str(
     x_best[2])+ ", activation2=" + str(
     x_best[3]))

print("The best parameters obtained (MPI): neuron1=" + str(x_best2[0]) + ", 
    neuron2=" + str(x_best2[1]) + ", activation1=" + str(
     x_best2[2])+ ", activation2=" + str(
     x_best2[3]))

print("The best parameters obtained (LCB): neuron1=" + str(x_best3[0]) + ", 
    neuron2=" + str(x_best3[1]) + ", activation1=" + str(
     x_best3[2])+ ", activation2=" + str(
    x_best3[3]))

# %%
#We start by taking a look a the model subclass of our Bayesian optimization
    object
print(opt.model.input_dim)
#so there is 6 dimensions, that is 1 for each of n_estimators and depth which are
     discrete variables.
# the remaining 4 is because one-out-of-K encoding is using for the categorical
    variables max_features and criterion
#we can also look at kernel parameters
print(opt.model.get_model_parameters_names())
#and get the current fitted values
print(opt.model.get_model_parameters())
#To get a plot of the acquisition function we use the function opt.acquisition.
    acquisition_function
#first we define a sensible grid for the first to parameters
#indexing='ij' ensures that x/y axes are not flipped (which is default):
#we also add two extra axes for the categorical varibles and here fix these to 0
    ('log2' and 'gini')
#note that the acqusition function can actually take any value not only integers
    as it lives in the GP space (here 0.5 intervals)
#and it is quite fast to evaluate - here in 40000 points
n_feat = np.arange(1,100,0.5)
max_d = np.arange(1,100,0.5)
pgrid = np.array(np.meshgrid(n_feat, max_d,[1],[0],[1],[0],indexing='ij'))

print(pgrid.reshape(6,-1).T.shape)
#we then unfold the 4D array and simply pass it to the acqusition function
acq_img = opt.acquisition.acquisition_function(pgrid.reshape(6,-1).T)
acq_img2 = opt2.acquisition.acquisition_function(pgrid.reshape(6,-1).T)
acq_img3 = opt3.acquisition.acquisition_function(pgrid.reshape(6,-1).T)

#it is typical to scale this between 0 and 1:
acq_img = (-acq_img - np.min(-acq_img))/(np.max(-acq_img - np.min(-acq_img)))
acq_img2 = (-acq_img2 - np.min(-acq_img2))/(np.max(-acq_img2 - np.min(-acq_img2))
    )
acq_img3 = (-acq_img3 - np.min(-acq_img3))/(np.max(-acq_img3 - np.min(-acq_img3))
    )



#then fold it back into an image and plot
acq_img = acq_img.reshape(pgrid[0].shape[:2])
acq_img2 = acq_img2.reshape(pgrid[0].shape[:2])
acq_img3 = acq_img3.reshape(pgrid[0].shape[:2])

fig, ax = plt.subplots(1,3, figsize=(15,5))

im1 = ax[0].imshow(acq_img.T, origin='lower',extent=[n_feat[0],n_feat[-1],max_d
    [0],max_d[-1]])
plt.colorbar(im1,ax=ax[0])
ax[0].set_xlabel('Neurons in hidden layer 1')
ax[0].set_ylabel('Neurons in hidden layer 2')
ax[0].set_title('Acquisition function EI');
```

```
im2 = ax[1].imshow(acq_img2.T, origin='lower',extent=[n_feat[0],n_feat[-1],max_d
    [0],max_d[-1]])
plt.colorbar(im2,ax=ax[1])
#ax[1].set_colorbar()
ax[1].set_xlabel('Neurons in hidden layer 1')
ax[1].set_ylabel('Neurons in hidden layer 2')
ax[1].set_title('Acquisition function MPI');

im3 =ax[2].imshow(acq_img3.T, origin='lower',extent=[n_feat[0],n_feat[-1],max_d
    [0],max_d[-1]])
plt.colorbar(im3,ax=ax[2])
#ax[2].set_colorbar()
ax[2].set_xlabel('Neurons in hidden layer 1')
ax[2].set_ylabel('Neurons in hidden layer 2')
ax[2].set_title('Acquisition function LCB');
```