Genetic Alogrithm for CNN Hyperparametr Optimization

Tianrui Guo

Department of Applied Mathematics

April 20, 2019

Abstract

In this big data era, we developed many efficient and effective ways to analyze our database. Among various techniques, computer vision is a fancy and important one. One of the most significant technique in modern computer vision is convolutional neural network which is a typical method in deep learning. When we process image classification or image recognition, we inevitably encounter a problem of determine the hyperparamters such as the number of Conv channels, the size of filters or the dimension of the dense layers. The process of finding a set of hyperparameters which can improve the performance of convolutional neural network is known as hyperparameter optimization, and it can be also called as tuning. In this Capstone project we are using genetic algorithm for finding an optimal CNN architecture. When the convolutional neural network goes deeper and more complicated, the number of hyperparameters increases dramatically. In order to increase the performance of tuning, we no longer use manual tuning methods like grid search, instead we use genetic algorithm which is a kind of random search method for tuning, because we want the optimization algorithm searching automatically in a larger hyperparamter space. To test the performance of genetic tuning, we implement genetic algorithm on MNIST which is a relative small data set include 60,000 examples in training set and 10,000 examples in test set. If this algorithm performs well in this dataset, we will have possibility to implement it on other significative datasets.

1 Introduction

Machine learning is the cutting edge technique in modern data science, and it is dominated by deep neural network. These great inventions enable us have much better performance in data analysis. Inspired by the deep neural network architecture, we obtain a rapid development in computer vision. Computer vision is an cutting edge technique making computer to see visual object and extract information from images and videos in the similar way as human does. Image classification and image recognition are the art-of-state field in computer vision, and they have significant contribution to human face recognition, automatic driving and neural style transfer. In this capstone project, we are using convolutional neural network to train MNIST dataset, and this is a kind of image classification. MNIST dataset is a large database of handwritten digits that is commonly used for training various image processing systems. The examples in MNIST dataset are black and white images of human handwritten digits and they are normalized to fit into a 28×28 pixel bounding box and anti-aliased, which introduced grayscale levels. Hence, we can encode the images into matrices which have all entries equals to one or zero, this problem is transferred as matrix regression. The framework we use to process image classification of MNIST dataset is convolutional neural network (we called CNN or Conv neural network later). Convolutional neural network (CNN, or ConvNet) is a class of deep neural

networks, most commonly applied to analyzing visual imagery. Convolutional networks were inspired by biological mechanism in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. The advantage of CNN of image classification is CNN has a weight-sharing feature. Comparing with fully connected neural networks, CNN has conspicuous fewer parameters which makes CNN extract information from dataset at lower computational cost.

The main purpose of this project is to find an optimal hyperparameter set for CNN framework. In deep learning, a hyperparameter is a parameter whose value is set before the training begins. Different models have different types of hyperparameters, in our CNN model we have filter size, Conv layer channel, pooling size, drop out rate and dense layer size as hyperparameters. Hyperparameters are integers and continuous numbers in our model, leading to mixed-type optimization problems. The choice of hyperparameters can affect the performance of deep neural network significantly. The accuracy and the time required for training process will depend on the hyperparameters we choose. To improve performance of our CNN model, we can apply various tuning methods to find a set of optimal hyperparameters. In this case we are applying genetic algorithm for hyperparameter optimization. Genetic algorithm for CNN tuning is a kind of population based training which is inspired by the process of natural selection. We consider a

tuple of hyperparameters as a chromosome of an individual. In our model, we assume every individual has only one chromosome. Since one chromosome represents a tuple of hyperparameters, a gene on the chromosome represents one hyperparameter. If a model has ten hyperparameter, the individual will have one chromosome consists ten genes. For our genetic algorithm, all the individuals in the same generation constitute the whole population. In next section, we will discuss each gene on the chromosome and how genetic algorithm works. For the first generation, we randomize the initial population using the hyperparameter distribution that is pre-defined. In this case, we regulate the distribution of hyperparameter to save computational resource, since genetic algorithm is at high level of computational cost, and we will also discuss later. Compare to using grid search method, for the first generation, we use random method to generate will enable us to explore more combination of hyperparameters. In most cases, we know little about the properties of deep neural ntework model, so that it is hard to directly give empirical combinations which have great performance. Even though grid search method seems more robust, we will not use it here. Figure 1 shows difference between grid search and random search.

The remainder of this Capstone project is organized as follows. In Section 2 we will discuss the model and tuning method we use. Section 2.1 is about CNN model for this problem, and Section 2.2 is about genetic algorithm. Results after

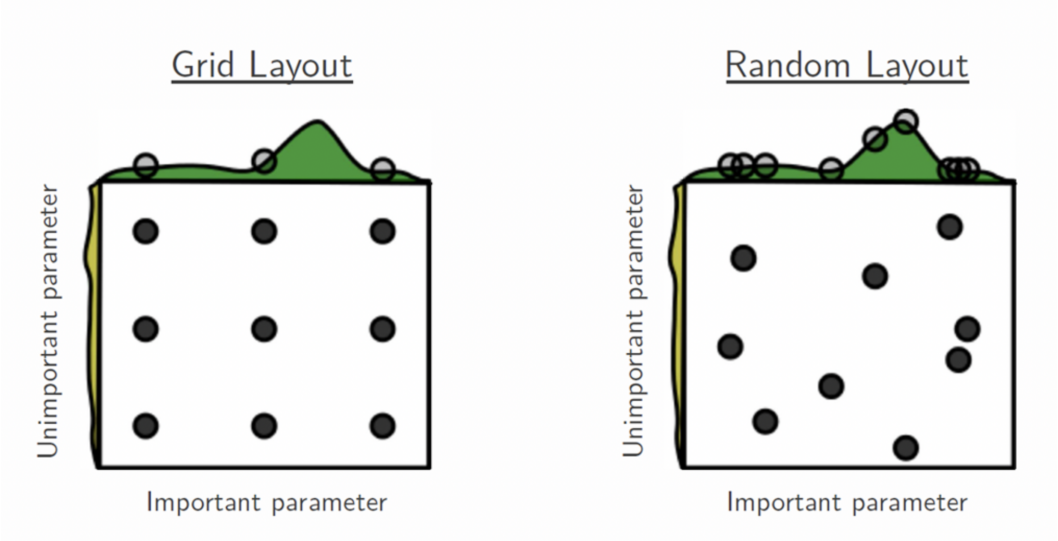

Figure 1: Grid Search vs Random Search

testing by MNIST dataset are included in Section 3 and we draw conclusion in Section 4.

2 Modeling

In the first part of this section, we will discuss the structure of our CNN model, and explain the functions of every part of CNN architecture. In the second part of this section we will introduce genetic algorithm, and we will explain how genetic algorithm works for CNN hyperparameter tuning. In our model for training MNIST dataset, CNN model is implemented as the inner part for genetic algorithm, and genetic algorithm works as outer structure for selecting optimal hyperparameter combination.

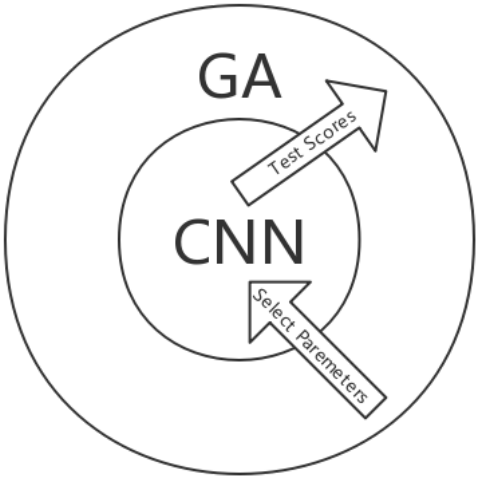

Figure 2: Relationship bewteen genetic algorithm and CNN in our model

2.1 CNN Model

As the inner structure of our model, CNN model is the classifier that test the performance of the model by a tuple of given hyperparameters from genetic algorithm. In our experiment, we use fixed layer structure to train MNIST dataset. In other words, our neural layers such as Conv layers, max pooling layers, drop-out layers, dense layers and softmax layers are in fixed order and the number of each layer is also fixed. The reason that we are not using a dynamic structure CNN model is because we are tuning the model by genetic algorithm, therefore the individual in a certain generation will have the same type of chromosome which indicate they have the same type of genes representing

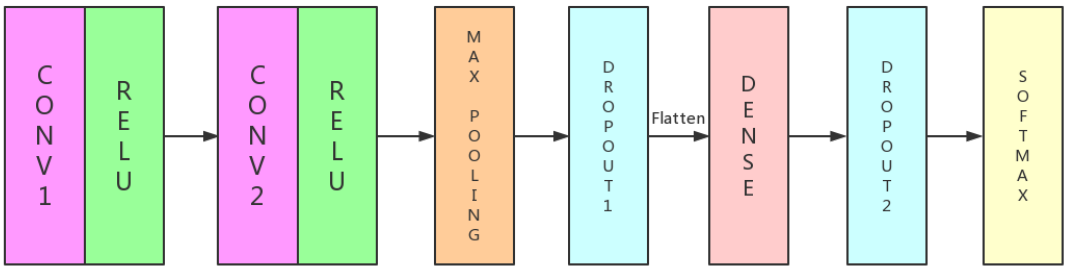

Figure 3: The structure of our CNN model

hyperparameters. If we change the order or number of our hidden layers in CNN model, the chromosome of individuals will change simultaneously. Individual with different type of chromosome cannot reproduce the next generation.

The structure that we are applying is shown in Figure 3. This model is inspired by LeNet-5, which is developed by Yann Lecun. The first hidden unit is consisted of a convolution operator and a ReLU activation function. The convolution operator calculates the summation of the production between each entry in the filter matrix and the corresponding area in the input matrix. Here we applied 1 as the strides of filter since our input is just a 28×28 gray scale image, we do not use bigger strides. Consider A is the input matrix, A_R is the sub-matrix of A whose upper left entry is A_{ij} and dimension equals to m which is the dimension of filter maxtix and F is the filter matrix. Then we have:

$$Conv(A)_{ij} = \sum_{p=1}^{m} \sum_{q=1}^{m} A_{Rpq} F_{pq}$$

Figure 4 shows how the convolution operator works. In this case, the first hidden layer provides us two hyperparameters which are the size of filter(or kernel) and the number of filters. It worth to notice that we actually apply same padding to the input matrix. For MNIST dataset, the inputs of data 28×28 pixel image, this is of a very small dimension. We basically don not want the image shrink when processing Conv layers, so we just add zero entries around the input matrix to get the output matrix ad the same dimension after Conv layers, and this is what we do for same padding. Additionally, if we do not use any padding, the entries near the boundaries of input matrix will become less import.

After the convolution operation, we put the matrix into an activation function. In our cases, we use ReLU. This activation function was first introduced to a dynamical network by Hahnloser et al. in 2000. The mathematical notation of ReLU is:

$$f(x) = max(0, x) = x^{+}$$

ReLU function helps us map the output of previous layer to the range of 0 to 1. The reason that we use ReLU as activation function is:

- **Sparsity**: When the input of ReLU is non-positive, the output turns to be zero, which sparse activation.

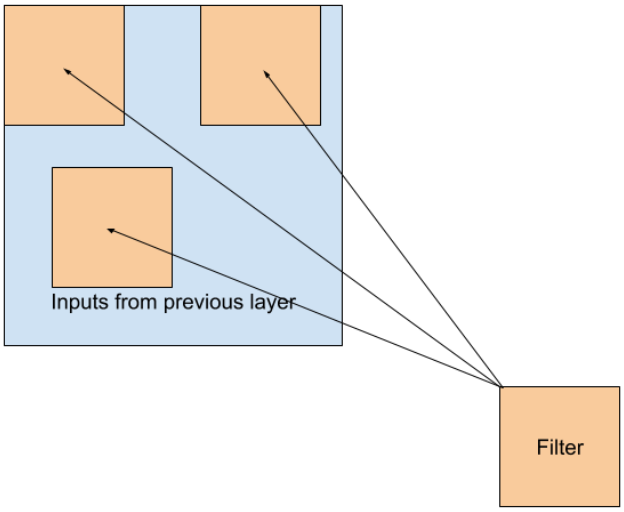

Figure 4: How CNN filter works

- **Better gradient propagation**: ReLU can effectively reduce vanishing gradient problem. For example, the gradient of sigmoid activate function vanishes when the absolute value of input goes very large.

- **Easy to compute**: ReLU save a lot of computational resource when doing propagation.

For the second hidden unit, it is totally the same with the first one, and also gives us another two hyperparameters which are the size of the filers and the number of filters.

2.2 Genetic Algorithm

There are many advantages of using genetic algorithm for hyperparameter optimization:

3 Result

4 Conclusion