AI Project 1: Drug Molecular Toxicity Prediction

KAR CHUN TEONG
518030990014
Department of Computer Science
Shanghai Jiao Tong University
`teongkarchun@sjtu.edu.cn`

Abstract

Develop a Deep Neural Network which learn useful patterns from the data provided and predict the toxicity of a new list of molecules based on learned knowledge using TensorFlow package.

1 Introduction

This project is the individual project of CS410 Artificial Intelligence 2020 Fall in Shanghai Jiao Tong University named Drug Molecular Toxicity Prediction. Given a dataset of drug molecules and possibilities of their toxicity, develop a Deep Neural Network which can learn from the data provided and predict the toxicity of other molecules.

2 Neural Network Architecture

2.1 Choosing the Suitable Neural Network Architectures

There are several types of neural networks with different usages, to get the best results, we need to choose a suitable neural network. First of all, we need to inspect the requirements of the project. The features given are the one-hots of molecules, which is essentially a two-dimension matrix, and we need to predict the possibilities of the molecule being toxic based on the two-dimension matrix. Based on this information, we did some researches on several types of neural networks, which are convolutional neural network, and fully connected network.

Convolutional neural network consist of one or more convolutional layers, which are either interconnected or pooled. The convolutional layer performs a convolutional operation, which is using a kernel or filter with weights attached on it to scan through the area of input and put the convolution results at corresponding positions, then passes the result to the next layer. By the use of kernel, it could effieciently be used to classify features within input matrix, for example, to derive significant feature from an image matrix, therefore, it is commonly used for classification of images, clustering of images, and etc.. Convolutional neural network is effiecient in spotting significant feature within matrices, which could be used to spot the possible toxic traits of molecules within the one-hot matrices. Therefore, we could deduct that convolutional neural network is suitable for this task.

Secondly, fully connected network consists of with every neuron in one layer connecting to every neuron in the next layer. This network structure basically don't have a specific strong suit, which also means that it could virtually be used for every kind of tasks, so we decided to use it too.

2.2 Convolutional Neural Network Model 1

2.2.1 Model Overview

To start off, we build a convolutional neural network model, the architecture of the model can be found in "model_figures/CNN_train.png", since the picture took up too much space so that we can't include it in the report, the model consists of the layers as shown below:

1. layer 1: a convolutional layer of 16 filters with kernel size of $(3, 3)$, with the activation function of relu, with same padding.
2. layer 2: a max pooling layer with kernel size of $(2, 2)$, with same padding.
3. layer 3: a convolutional layer of 32 filters with kernel size of $(3, 3)$, with the activation function of relu, with same padding.
4. layer 4: a max pooling layer with kernel size of $(2, 2)$, with same padding.
5. layer 5: a convolutional layer of 64 filters with kernel size of $(3, 3)$, with the activation function of relu, with same padding.
6. layer 6: a flatten layer used to flatten the input for the dense layer.
7. layer 7: a dense layer with output of dimension of 64 with the activation function of relu.
8. layer 8: a dense layer with output of dimension of 2.

We would justify our model design in the following sections.

2.2.2 Kernel Size

Researchers stated that a small kernel size is generally better[1], also, given that we are trying to find a toxic part within molecular structure, it's logical to deduct that a smaller kernel size will do a better job since the molecular size is small in general. Among small kernel size, a $(1, 1)$ kernel basically do nothing, $(2, 2)$ could be considered, but it is even-sized, and a even number for kernel size is not desirable, since odd-sized filter always produce a symmetrical output layer while even-sized filter might distort the size of output matrices, which makes the following processing harder. Therefore, our final choice of kernel size is $(3, 3)$. Do note that we use a kernel size of $(2, 2)$ for the max pooling layers, in concern of the number of operations involved.

2.2.3 Number of Kernels

We start off with convolutional layer with small number of kernels, which is 16, followed by 32, then 64. Notice that the number of kernels is increasing as we go deep into the network, this is to ensure that the earlier layers capture local patterns first with the smaller number of kernels, then, as the network goes deeper, the deeper layers could characterize more global patterns with a larger number of kernels.

2.2.4 Choice of padding

We use the same padding for all convolutional layers, which would ensure the output size is the same as input size, this is to ensure that the toxic traits of the molecule would not be omitted as the output size get squeezed into a smaller size.

2.2.5 Choice of Strides

Notice that we use stride of 1 in all of our layers, this is to prevent downsampling of our input, so that the toxic traits of the molecule would not be omitted when the kernel use a large stride.

2.2.6 Setting Seed

One of the project requirements is to ensure the results to be reproducible. To achieve this in tensorflow, we would need to set a fixed value of seed, otherwise, tensorflow would produce different training weights each time we run the program. A problem arises as setting seed is neccessary, which is the value of seed to choose. Depends on the value of the seeds, we could have produce drasticly different results. Based on my research, it seems that the only way to find a seed to produce the better results is to iterate over different value of seeds on the model and choose the seed with the best result. However, given the limited time and the poor performance of my laptop, we could not afford to do the iteration. Therefore, we just choose a random value for the seed. However, do note that researchers stated that many experimental results indicate that randomized models can reach sound performance compared to fully adaptable ones, with a number of favorable benefits, including simplicity of implementation, faster learning with less intervention from human beings, and possibility of leveraging overall linear regression and classification algorithms[2]. Therefore, setting a seed to limit the randomness of model is not always a good practice.

2.2.7 Other Details

Other details of the code, which includes reading and preprocessing the data, predicting and generating output, and so on are simple enough and the code explains itself, therefore, we will not do any further explanation on them. It is worth noting that we build the model largely based on this tensorflow tutorial on convolutional neural network[3].

2.3 Convolutional Neural Network Model 2

2.3.1 Model Overview

However, we find out that the performance of the initial model is limited, so we rebuild the model, the model consists of the layers as shown below:

1. layer 1: a convolutional layer of 32 filters with kernel size of $(5, 5)$, with the activation function of relu, with same padding.
2. layer 2: a max pooling layer with kernel size of $(2, 2)$.
3. layer 3: a convolutional layer of 64 filters with kernel size of $(3, 3)$, with the activation function of relu, with same padding, and with a stride of $(1, 2)$.
4. layer 4: a max pooling layer with kernel size of $(2, 2)$.
5. layer 5: a flatten layer used to flatten the input for the dense layer.
6. layer 6: a dense layer with output of dimension of 2.
7. layer 7: an activation function of softmax to filter the output.

A special note is to be made for layer 6 and 7, in this model, we return two outputs, one go through from layer 1 to layer 6, while another one go through from layer 1 to layer 7. In other words, one output pass through the softmax activation function while the other one doesn't. Again, we justify the changes of model in the below section.

2.3.2 Kernel Size

We argued about the use of small kernel size could give us better results previously in section 2.2.2, however, we noticed that by using a kernel size of $(5, 5)$ in the first convolutional layer could help the model to quickly grasp the overview of the molecule, then in the second convolutional layer we use a kernel size of $(3, 3)$ to capture the local details of feature. As shown in section 3.1, this approach brings better results.

2.3.3 Number of Kernels

As mentioned previously, we start off with convolutional layer with smaller number of kernels, which is 32, followed by 64.

2.3.4 Choice of padding

We use the same padding setting for the convolutional layers as before, the distinct difference is that we did not use same padding for the max pooling layers, because the entire purpose of max pooling layers is to downsample the input layer, by using the same padding in max pooling layers in the previous model, we completely missed out the advantages of pooling layers, so we fix it in this model.

2.3.5 Choice of Strides

The only difference of stride is in the second convolutional layer, we use a stride of $(1, 2)$ to downsample the height of input layer.

2.3.6 Setting Seed

As mentioned previously, we still need to set a fixed value seed to ensure that the results are reproducible.

2.3.7 Other Details

As mentioned previously, the other details of the code, which includes reading and preprocessing the data, predicting and generating output, and so on are simple enough and the code explains itself, therefore, we will not do any further explanation on them.

2.4 Fully Connected Neural Networks Model

2.4.1 Model Overview

We build another model using the fully connected neural network architecture, the architecture of the model can be found in "model_figures/FCN_train.png", the model consists of the layers as shown below:

1. layer 1: Dense layer with parameter 1024.
2. layer 2: ReLU activation layer.
3. layer 3: Flatten layer.
4. layer 4: Dense layer with parameter 64 with ReLU activation layer.
5. layer 5: Dense layer with parameter 1 with sigmoid activation layer.

The architecture of this model is inituitive thus doesn't really need any further explanation, we just pass through the input data with several fully connected dense layer intersected with activation layers.

3 Performance

3.1 Performance of each model

CNN model 1	0.78533
CNN model 1 with batch normalization and dropout	0.77245
CNN model 2	0.85237
CNN model 2 with batch normalization and dropout	0.83148
FCN model	0.79972

Table 1: Performance of different models in AUC score

As shown in Table 1, the performance of CNN model 2 is clearly superior than the other models, so we decided to use it in our final submission.

3.2 Performance Analysis

3.2.1 The effect of batch normalization and dropout in CNN models

Curiously enough, the addition of batch normalization and dropout layer in both CNN models actually had an adverse effect on their performance. Our deduction on this occurence is that the main effect of batch normalization and dropout layer is to prevent overfitting, and since we used a quite low number of epochs for both CNN models in this project, there is actually a low chance of overfitting, so using batch normalization and dropout layer is actually causing underfitting for our models and thus lowering their performance. Therefore, we decided to remove the batch normalization and dropout layer from our final submission code.

4 Submission Details

We submitted all models mentioned above, the corresponding files to each models are shown below:

1. CNN model 1: CNN_train.py and CNN_test.py
2. CNN model 2: train.py and test.py
3. FCN model: FCN_train.py and FCN_test.py

We decided to use the CNN model 2 as the final model since it has the best performance, however, we still included the other models as supplement materials. Note that the reason behind the huge discrepancy of style between the different models is we build our initial model from scratch and later find out that there exists a demo model.

5 References

[1] Sabyasachi Sahoo. Deciding optimal kernel size for CNN. https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

[2] Simone Scardapane, Dianhui Wang. Randomness in neural networks: an overview. https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1200

[3] Tensorflow official tutorial of convulutional neural network. https://www.tensorflow.org/tutorials/images/cnn