A HILL-CLIMBING APPROACH TO CONSTRUCT NEURAL NETWORK AUTOMATICALLY – TESTED ON MNIST

OPEN SOURCE PAPER

Chung-Chen Chen
Department of Computer Science and Information Engineering
Nation Quemoy University
Kinmen, Taiwan
ccc@nqu.edu.tw

October 17, 2021

ABSTRACT

Deep learning models are usually constructed by human, based on their intutition.

If we can develop a automatically method to construct neural network model, maybe we can construct some models that's difficult for human to build.

A method based on Hill Climbing Algorithm is use to build Neural Network model automatically.

In this paper, we propose a method based on Hill-Climbing Algorithm to construct neural network model automatically.

Our experiment shows that Hill-Climbing Algorithm works on handwritten digit recognition problem, and build a model with 98% accuracy.

You may get the open source code at: `https://github.com/cccresearch/nnModelAuto/`

***K*eywords** Neural Network · Deep Learning · MNIST · Auto Construct

1 Introduction

Most neural network learn by gradient descendent approach. Backpropagation algorithm is used to calculate the gradient, and adjust weight step by step, until the gradient down to zero.

But gradient descendent approach only works for continuous function, and cannot work on discrete case.

Neural network models are usually composed by several layers, such as Linear/ReLU/Conv2D/Pool/Sigmoid etc. However, we can use gradient descendent algorithm to learn weight parameters, but cannot learn the structure of layer directly.

On the other hand, search or optimization approach may be works for it. For example, Hill-Climbing Algorithm and Probabilistic Search may be used to adjust neural network model automatically.

In order to verify it, we try to use the Hill-Climbing Algorithm to build neural network model automatically on MNIST test set.

Our experiment shows that Hill-Climbing Algorithm, started from a single layer network, and evolve into multilayer networks. Finally a multilayer neural network with accuracy 98.05% was builted.

Hill-Climbing is not the only method can be used to construct neural network models automatically. We will try some other optimization algorithm in the future.

2 Background

Neural network research started from the single layer perceptron Frank Rosenblatt's research in 1957 [1]. In 1986, Hinton reinvent the backpropagation algorithm [2] and works good on voice recognition.

In 1998, LeCun [3] propose Convolutional Neural Network (CNN) works on handwritten digit recognition problem well.

After 2011, the GPU hardware and the ImageNet test set accelarate the research of neural network. There are many deeper neural network models developed and have good performance. These new development was called Deep Learning Technology [4] [5] [6].

However, these neural networks were usually designed by human, not by machine.

We wonder if computer can design neural network automatically, so we have an experiment on the MNIST test set.

There are some research on building neural network automatically [7] [8] . This paper is the first step of our resarch on this topic.

3 Method

The Hill-Climbing Algorithm simulate the action of climbing a hill. When there are neighbors higher than here, we move to a higher neighbor.

```
def HillClimbing(x)
    x = a random solution
    while x have higher neighbor nx:
        x = nx
    return x
```

We need a height() function to compare which is higher. Whenever there is a higher neighbor, x move to it. If there are no higher neighbor, the algorithm stop.

The following python code shows the detail of hill-climbing algorithm.

```
def hillClimbing(s, maxGens, maxFails):
    fails = 0
    for gens in range(maxGens):
        snew = s.neighbor()
        sheight = s.height()
        nheight = snew.height()
        if (nheight > sheight):
            s = snew
            fails = 0
        else:
            fails = fails + 1
        if (fails >= maxFails):
            break
    return s
```

The height() function was defined as following in our code.

```
def height(self):
    net = self.net
    if not net.exist():
        trainer.run(net)
    else:
        net.load()
    return net.accuracy()-(net.parameter_count()/1000000)
```

The height() above works well on MNIST case, but may be adjust for the other cases.

Another function, neighbor(), works by random select from the [insert, update] operations. Insert will add a new layer, and update will modify a layer in the network.

```
def neighbor(self):
    model = copy.deepcopy(self.net.model)
    layers = model["layers"]
    in_shapes = self.net.in_shapes
    ops = ["insert", "update"]
    success = False
    while not success:
        i = random.randint(0, len(layers)-1)
        layer = layers[i]
        op = random.choice(ops)
        newLayer = randomLayer()
        if not compatable(in_shapes[i], newLayer["type"]):
            continue
        if op == "insert":
            layers.insert(i, newLayer)
        elif op == "update":
            if layers[i]["type"] == "Flatten":
                continue
            else:
                layers[i] = newLayer
        break

    nNet = Net()
    nNet.build(model)
    return SolutionNet(nNet)
```

The training of neural network takes a lot of time. So we try to minimize the number of neighbors. There are only six kind of network layers in our code, and the parameter size and channel are all restricted to power of 2.

```
types = ["ReLU", "Linear", "Conv2d", "AvgPool2d", "LinearReLU", "ConvPool2d"]
sizes = [ 8, 16, 32, 64, 128, 256 ]
channels = [ 1, 2, 4, 8, 16, 32 ]

def randomLayer():
    type1 = random.choice(types)
    if type1 in ["Linear", "LinearReLU"]:
        k = random.choice(sizes)
        return {"type":type1, "out_features":k}
    elif type1 in ["Conv2d", "ConvPool2d"]:
        out_channels = random.choice(channels)
        return {"type":type1, "out_channels": out_channels}
    else:
        return {"type":type1}
```

The code will use the compatable() function to check the compatability of dimension before insert a new layer.

Table 1: The accuracy of models in our Hill-Climbing Algorithm on MNIST

step	model	accuracy	parameter_count	height	comment
0	Flatten	91.79%	7850	91.782151	start
1	ReLU+Flatten	91.91%	7850	91.902154	
5	ConvPool2d(8)+Flatten	92.08%	13610	92.066392	
9	ConvPool2d(8)+ReLU+Flatten	93.37%	13610	93.356393	
40	Conv2d(4)+ReLU+Flatten	95.98%	27090	95.952913	
47	Conv2d(4)+Conv2d(8)+ReLU+Flatten	97.64%	46426	97.593573	
49	Conv2d(4)+Conv2d(8)+Conv2d(32)+ReLU+Flatten	98.05%	157562	97.892441	

```
types2d = ["Conv2d", "ConvPool2d", "AvgPool2d", "Flatten"]
types1d = ["Linear", "LinearReLU"]

def compatable(in_shape, newLayerType):
    if newLayerType in ["ReLU"]:
        return True
    elif len(in_shape) == 4 and newLayerType in types2d:
        return True
    elif len(in_shape) == 2 and newLayerType in types1d:
        return True
    return False
```

Finally, the Hill-Climbing Algorithm start from a simple network with Flatten+Linear layers. The Flatten layer turn multidimensional structure into one dimensional, and the linear layer fully connect from the flatten layer to the outputs.

From the start, Hill-Climbing Algorithm will add new layer or update existed layer step by step. Whenever find a better network (higher), move to it (the new solution).

4 Experiments

In our code, whenever a better model is found, it will be recorded in a file.

The follow table shows the accuracy and height of an experiment.

In table 1, all model ends with an Linear layer appeneded.

The Hill-Climbing Algorithm start from the Flatten(+Linear) model, and move to better model step by step. In step 49, a model "Conv2d(4)+Conv2d(8)+Conv2d(32)+ReLU+Flatten" was builted with accuracy 98.05% .

However, the paramter_count increase from 7850 to 157562.

Our height() function is "accuracy()-parameter_count()/1000000", so, the height of the last model (98.05-0.158) is still better than the others, include the first one (91.79-0.07).

If we prefer small model, we may modify the height() function to "accuracy()-parameter_count()/100000". If we don't care about the size of model, we may just use the "accuracy()" as the height().

5 Conclusion and Future Works

The experiment above shows that the idea of building neural network automatically by Hill-Climbing Algorithm really works.

However, the Hill-Climbing Algorithm is not the only method can be used. We will try some other optimization methods in the future.

Besides, our computer without GPU is too slow to test it on bigger test set, such as CIFAR and ImageNet. More experiments should be done in the future.

In this paper, we focus on MNIST Image Recognition domain. But we guess that Hill-Clmbing Algorithm may works for the other domain.

For example, build Recurrent Neural Network (RNN) automatically for language processing domain is on the top of our future work list.

References

[1] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[2] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[3] Yann André LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 1998.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[6] François Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2017.

[7] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, Matthias Urban, Michael Burkart, Maximilian Dippel, Marius Thomas Lindauer, and Frank Hutter. Towards automatically-tuned deep neural networks. In *Automated Machine Learning*, 2019.

[8] Steven Abreu. Automated architecture design for deep neural networks. *ArXiv*, abs/1908.10714, 2019.