Abstract

Neural networks trained using transfer learning are vulnerable. According to the method of extracting 3-layer FCNN, NNs trained with transfer learning can also be extracted using the similar method. This paper proposes a method to extract the parameters of NNs trained using transfer learning and some tips to avoid it. Although the method proposed contains a certain degree of randomness (this randomness is not critical), it still has the ability to successfully extract (with 99.99% similarity) NN parameters.

Motivation

Because more and more versatile backbone NNs (such as ResNet, Inception, etc.) have been discovered, researchers have begun to use transfer learning as the major training method. Using this technique can reduce the time to obtain results. However, this technique will also put these NNs in danger of being stolen.

This paper attempts to **reveal the specific operation of this method of stealing NNs** (I don't recommend that you use this method to steal a commercial NN). Starting from the prerequisites to deploy this kind of attach, this paper also **shows how to safely use the transfer learning technique**.

Introduction

In the theoretical background of NNs (this has been rigorously proven mathematically), a 3-layer FCNN can simulate any functions if it is designed properly. But not all tasks can be implemented with a 3-layer FCNN. For example, for an image classification problem, we cannot simply convert an image into a vector directly. This is because after the conversion, the spatial information is lost. On the other hand, convolutional neural network (CNN) is a good choice. We can use an appropriate CNN before a 3-layer FCNN to extract a vector that contains spatial information. However, the training of this CNN is usually very difficult. Therefore, some research teams will publish some versatile CNNs when dealing with similar tasks. This is the origin of transfer learning (not all).

But leaving the head of the pretrained NN, a NN will only have a 3-layer FCNN which can be extracted completely. Unfortunately, the pretrained head of transfer learning is usually public,

which creates conditions for extracting the parameters of the whole NN trained using transfer learning.

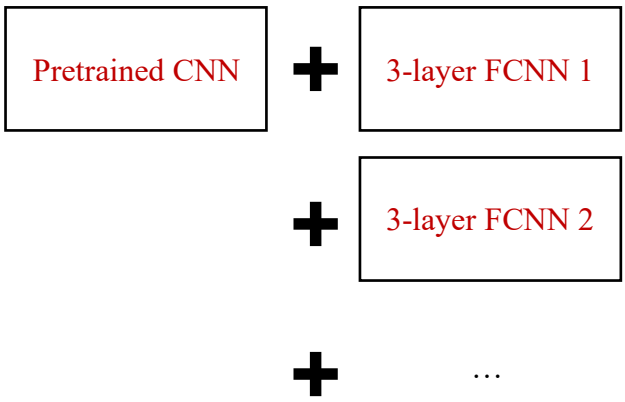


Figure 1: Transfer learning

Extract the parameters of a 3-layer FCNN

The structure of a 3-layer NN is as follows.

In which the matrix of $A_1, A_2, A_3, A_4, A_5, A_6$ is called the *first A matrix*, the matrix of B_1, B_2 is called the *first B matrix*, the matrix of A_7, A_8 is called the *second A matrix*, the matrix of B_3 is called the *second B matrix*, and:

$$y_1 = ReLU(A_1x_1 + A_2x_2 + A_3x_3 + B_1)$$
$$y_2 = ReLU(A_2x_1 + A_4x_2 + A_6x_3 + B_2)$$
$$z = A_7y_1 + A_8y_2 + B_3$$
$$ReLU(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

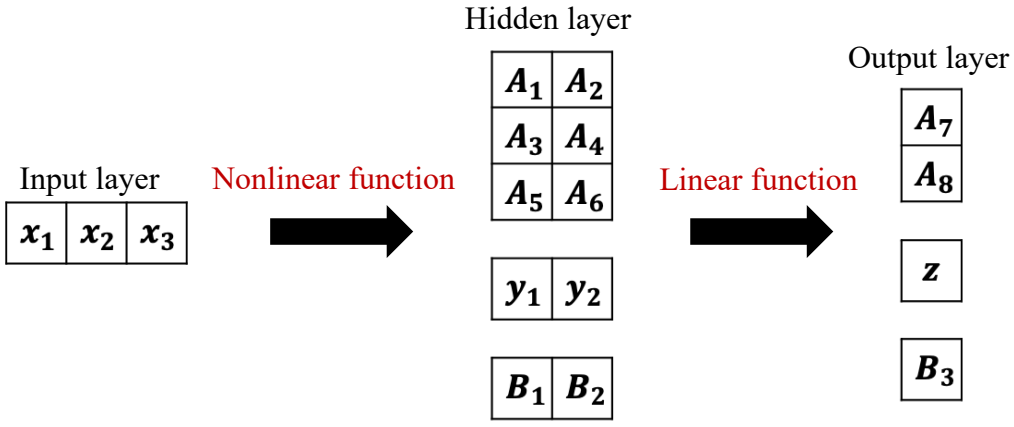


Figure 2: a 3-layer FCNN

There are only two functions, one of which is a nonlinear function, and the other is a linear function. And although there are only 3 layers, the number of FCNN parameters still accounts for a large part of all the parameters of a NN (even in deep NNs, the last three layers still account for a large proportion of parameters). Although the number of parameters is large, it does not affect the principle of calculation. For this reason, in the next toy NN, the number of parameters is deliberately reduced, which is also for better visualization.

Take the above NN for example, because of the different values of y_1 and y_2, there will eventually be four different cases for z:

$$z = A_7 y_1 + A_8 y_2 + B_3$$
$$z = A_8 y_2 + B_3$$
$$z = A_7 y_1 + B_3$$
$$z = B_3$$

Because of the selectivity of ReLU (only ReLU is discussed here, because most of the current NNs only use ReLU as the activation function, especially those deep NNs), the nonlinear function is actually piecewise linear. In order to obtain the gradient of different linear sections and then determine the parameters of the linear functions of different sections, it is necessary to determine the positions of the dividing points of different sections at first. In this paper, these dividing points are referred to as *critical points*.

In order to find the critical points, we need to enumerate the points on a straight line in the *input space* of the 3-layer FCNN. (Similarly, the space corresponding to the hidden layer is called *hidden space*, and the space corresponding to the output layers is called *output space*) The intersection of this straight line and the dividing boundaries of different linear sections is the critical point.

Depending on different straight lines, the critical points will be different, but there are two characteristics:

1. On each critical point, one and only one dimension in the hidden space is 0. (In the example above $A_1 x_1 + A_2 x_2 + A_3 x_3 = -B_1$, in which x_1, x_2, x_3 are of one critical point.)
2. Once the critical point is found in one dimension, this critical point is also the critical point for the other dimensions.

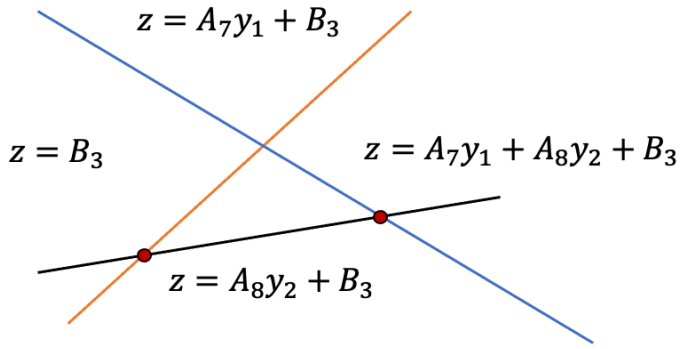


Figure 3: two critical points and the way to find them

Therefore, on both sides of the critical point in one dimension, there will be completely different gradients. Take x_1 in the previous NN as an example:

$$\frac{\partial z}{\partial x_1} = A_7 \frac{\partial y_1}{\partial x_1} + A_8 \frac{\partial y_2}{\partial x_1}$$

$$\frac{\partial y_1}{\partial x_1} = \begin{cases} A_1 & A_1x_1 + A_3x_2 + A_5x_3 > 0 \\ 0 & A_1x_1 + A_3x_2 + A_5x_3 \leq 0 \end{cases}$$

Suppose that for a certain critical point (say c_1), in a certain dimension (say x_1), we have found two related points, one of which is a bit left in this dimension (say c_1l) and the other is a bit right (say c_1r). Through the *gradient difference* between these two points, the following important information can be obtained:

$$\left.\frac{\partial z}{\partial x_1}\right|_{c_1l} - \left.\frac{\partial z}{\partial x_1}\right|_{c_1r} = A_7A_1 + A_8\frac{\partial y_2}{\partial x_1} - A_8\frac{\partial y_2}{\partial x_1} = A_7A_1$$

Similarly, it follows that:

$$\left.\frac{\partial z}{\partial x_1}\right|_{c_1l} - \left.\frac{\partial z}{\partial x_1}\right|_{c_1r} = A_7A_1 \quad \left.\frac{\partial z}{\partial x_1}\right|_{c_2l} - \left.\frac{\partial z}{\partial x_1}\right|_{c_2r} = A_8A_2$$

$$\left.\frac{\partial z}{\partial x_2}\right|_{c_1l} - \left.\frac{\partial z}{\partial x_2}\right|_{c_1r} = A_7A_3 \quad \left.\frac{\partial z}{\partial x_2}\right|_{c_2l} - \left.\frac{\partial z}{\partial x_2}\right|_{c_2r} = A_8A_4$$

$$\left.\frac{\partial z}{\partial x_3}\right|_{c_1l} - \left.\frac{\partial z}{\partial x_3}\right|_{c_1r} = A_7A_5 \quad \left.\frac{\partial z}{\partial x_2}\right|_{c_2l} - \left.\frac{\partial z}{\partial x_2}\right|_{c_2r} = A_8A_6$$

By assign A_1 and A_2 to 1 (this will cause no problems), we can recover A_3, A_4, A_5, A_6 of the first A matrix. In the first characteristic of the definition of the critical point, after the first A matrix is obtained by the above method, the first B matrix can be easily obtained.

Up to now, the first nonlinear function has been obtained. Then, the method of obtaining the parameters of a linear function is solving a system of linear equations. This is a very basic

linear algebra method. In the example above, since there are only three parameters that need to be extracted in the end, one can simply use the API for 3 times to obtain the second A and B matrices.

Extract the parameters of a NN trained with transfer learning

In some NNs, the parameters of the pretrained head will not change at all (some NNs using transfer learning will greatly change the parameters of the pretrained head, but some will not), so the parameters that need to be extracted are only the last three layers. The only difference is that if you don't use transfer learning, you can easily find a straight line in the input space. If you use transfer learning, since we don't have the complete NN model, we need to find a straight line in the output space of the pretrained head (which is also the input space of the 3-layer FCNN). Unfortunately, most of the current NNs do not support searching for the inverse function (cannot know what input was used through the output). On the other hand, since we need to find gradients in all dimensions, the best straight line direction should be the diagonal which is $[1, 1, 1, \dots]$, but it will take a lot of time to find a straight line in this direction, because most NN outputs are not in this direction.

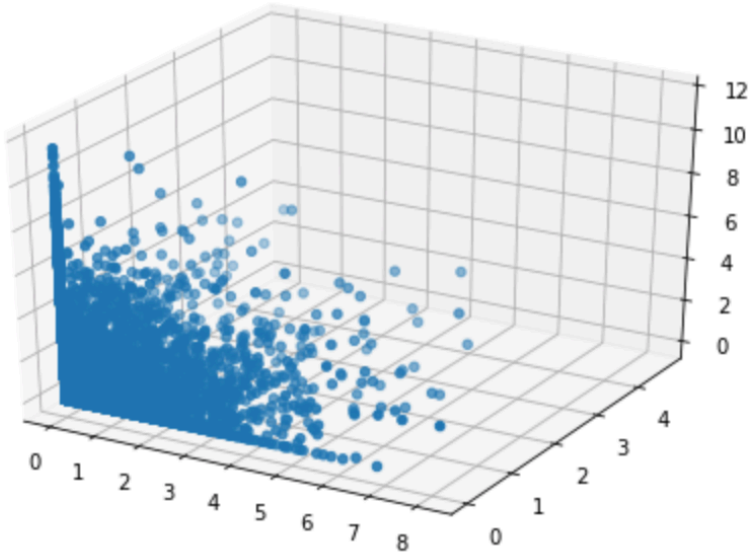


Figure 4: some outputs of a pretrained head

Therefore, I choose to find straight lines in the directions of each axis $[1, 0, 0, \dots], [0, 1, 0, \dots], \dots$. These two above methods of finding straight lines are completely the same in effect.

The method of finding points is *random enumeration*, that is, an input is randomly selected in the input space of the pretrained head and is defined as an anchor, and if the output of the out random selected points are in the desired direction according to the anchor, this input and output pair is retained.

In each direction for each dimension in the input space, the gradient difference follows that:

Table 1

	Critical point 1	Critical point 2	...
Direction 1	A_7A_1	A_8A_2	...
Direction 2	A_7A_3	A_8A_4	...
Direction 3	A_7A_5	A_8A_6	...
...	...	...	...

If we can find the points on the diagonal, the corresponding table is:

Table 2

	Critical point 1	Critical point 2	...
Diagonal	A_7A_1	A_8A_2	...
Diagonal	A_7A_3	A_8A_4	...
Diagonal	A_7A_5	A_8A_6	...
...	...	...	...

With this information, we can recover the parameter matrices by following the above steps.

But in the above method, there is a critical problem. Because of ReLU, it is impossible for the pretrained head to output any negative values. As a result, in the output space of the pretrained head, if the critical points corresponding to the subsequent 3-layer FCNN have negative values, these critical points will never be found. But in the actual neural network, it is already impossible to say there are no negative critical points. So, this can be regarded as the biggest problem of this method.

Experiments

The experiment will be carried out on a toy NN, this network has two 2×2 convolutional layers as the pretrained head, and the rest is a 3-layer FCNN. This network will provide the API of the head and the API of the entire neural network. No other APIs available.

```
self.conv = nn.Sequential(nn.Conv2d(1,3,2),
                          nn.ReLU(),
                          nn.Conv2d(3,3,2),
                          nn.ReLU())
```

```
A_0 = torch.tensor([[-1,-2,-3.],[-2.5,-4.5,-5.5]])
B_0 = torch.tensor([[1.],[1.]])
A_1 = torch.tensor([[3,3.]])
B_1 = torch.tensor([[1.]])
```

As mentioned above, the first mission is to find a straight line in the output space of the pretrained head. Instead of finding the points on the diagonal, we look for points on axis. In this example, there are three dimensions:

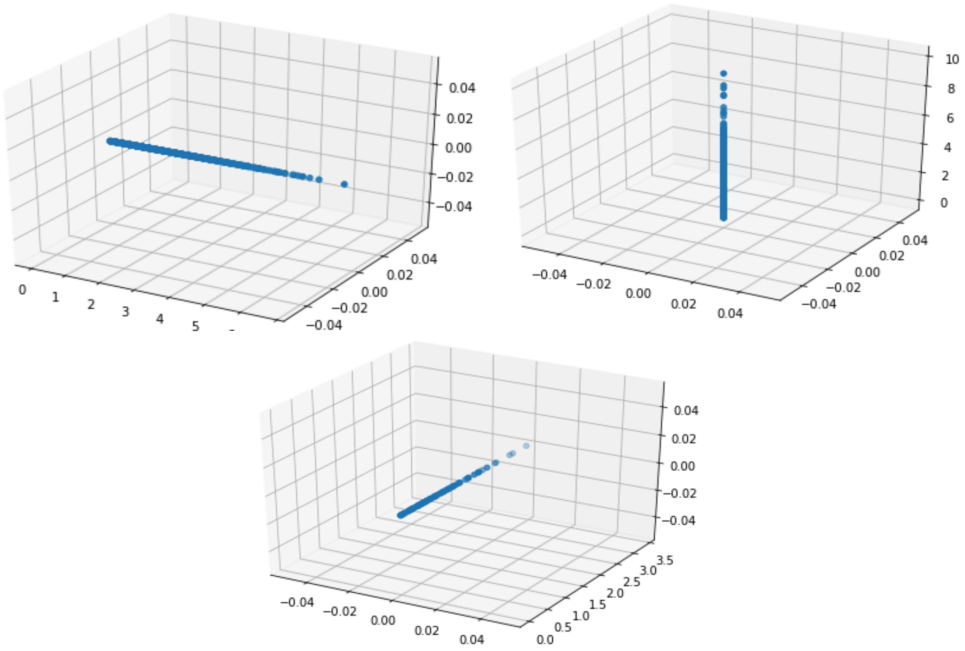


Figure 5: points on axis

Then the second mission is to calculate the gradient. In the pure 3-layer FCNN, we can control the interval between each point to be the same, but the points obtained by enumeration using the above random method cannot guarantee this property. Therefore, the curve of the gradient might have some glitches. But we can make it smooth (the orange curve).

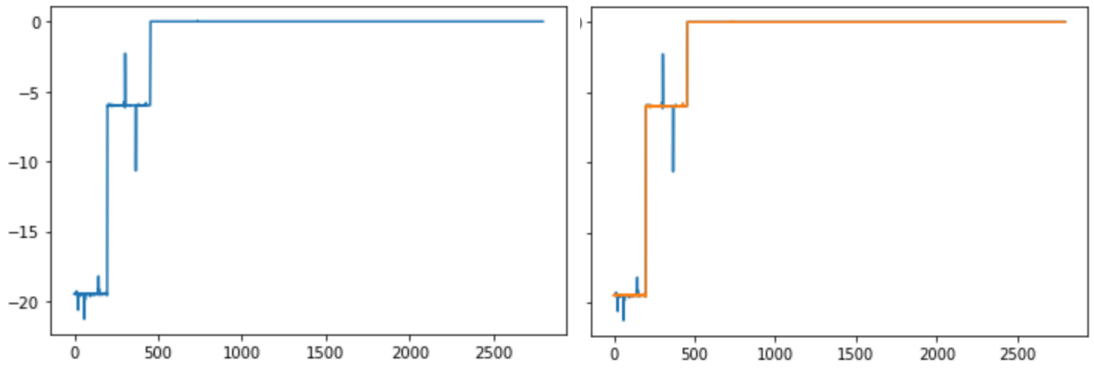


Figure 6: curve of the gradient

Using the smoothed curve of the gradient, we can find the critical points.

```
{198: [197, 198], 454: [453, 454]}
tensor([0.0000, 0.2221, 0.0000])
tensor([0.0000, 0.4996, 0.0000])
```

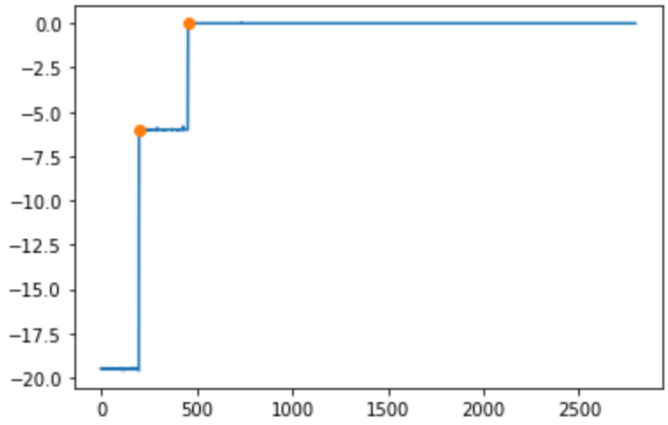


Figure 7: the critical point

Sometimes it is not accurate enough. If you are satisfied enough, you can randomly enumerate more points. But since this is a random process, more points cannot guarantee the absolute accuracy. In this example, there are two critical points $[0,0.2221,0]$, $[0,0.4996,0]$. The accurate critical points are $[0,0.25,0]$, $[0,0.5,0]$. It is not exactly the same, but it is acceptable. Then we can use the critical points to recover the first A matrix.

```
tensor([[-1.0000, -1.9994, -2.9990],
        [-2.5000, -4.4988, -5.4988]])
tensor([[-1.0000, -2.0000, -3.0000],
        [-2.5000, -4.5000, -5.5000]])
```

The upper one is the recovered matrix, and the lower one is the accurate matrix. We can see they are almost the same, which show the high fidelity and the high accuracy of this methods. Then we recover the first B matrix. The result is also acceptable.

```
tensor([[1.0097],
        [1.0017]], grad_fn=<PermuteBackward>)
tensor([[1.],
        [1.]])
```

Finally, we solve a set of linear equations to obtain the other two parameter matrices at the same time.

```
matrix([[-33.86135571],
        [ 23.10923634],
        [ 17.31328639]])
```

At here, the result seems NOT acceptable at all! But this is because I only enumerate few points, so that the result is not accurate enough. But if we use the accurate parameter to calculate the result reversely (substitute the accurate parameters into these linear equations), the residual is still acceptable, which shows that the method works.

```
abs(A[0][0]*3+A[0][1]*3+1-B[0])
```

```
0.032819271087646484
```

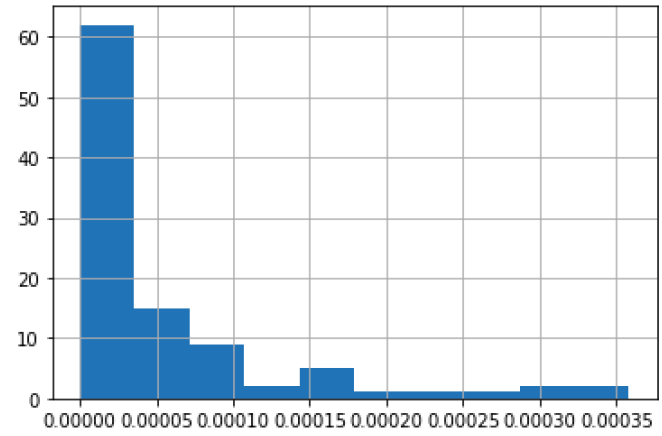
```
abs(A[1][0]*3+A[1][1]*3+1-B[1])
```

```
0.031827449798583984
```

```
abs(A[2][0]*3+A[2][1]*3+1-B[2])
```

```
0.032481253147125244
```

Run 100 random examples. Sometimes the error is also acceptable. (**Since this method includes some randomness, including the initialization of the CNN and the random inputs selected. I cannot guarantee this method will always yield the correct result**.)




```
err 0.004789423666815623 %
```

Conclusion

The method proposed in this paper requires the pretrained head to remain unchanged. Therefore, when using transfer learning, you must set a certain learning rate for the pretrained head, so that the pretrained head is different from the network that can be downloaded on the Internet. Secondly, the method proposed in this paper has strict requirements on the location of the critical point of a NN. Although the caveat is difficult to be satisfied in most NNs, it is still necessary to detect the location of these critical points to completely avoid this kind of attack.

Lessons I have learned

Even though the structure of NN cannot be explained, its calculation method has been transparent, which brings some security risks. For commercial NNs, this potential risk must be taken seriously. In fact, this project is a relatively theoretical project, and it not only has higher requirements on the understanding of the calculation principle of NNs, but also highly depends on the corresponding code to verify the results. But through my efforts, my abilities have been improved through this project. I am proud of myself! Similarly, through projects and paper presentations from the other students, I learned that cyber security is a very promising and very active research field. In future research, I learned how to evaluate my research project from the perspective of cyber security.

here is the opensource demo

https://github.com/ParanoiaPrime/ACS-Project/blob/main/project.ipynb

References

Jagielski, Matthew, et al. "High Accuracy and High Fidelity Extraction of Neural Networks." *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 2020.