ResNet

Table of Contents

1. ResNet

1.1. Background

When we train a shallow neural network, the performance on the training set would quickly decrease in the early stage of training. However, when we train a deep neural network, the performance on the training set would decrease slowly in the early stage of training.

The reason is that, when we initialize the weights of the neural network, the original input would be scrambled by the early layers to somehow a random vector. So then the signal in the last layers would be nothing but random noise. That means **we take the original random noise to compute the loss and gradients, and the final few layers, which get meaningless input, would learn a meaningless update. But the early layers that get meaningful input would also learn a nothing update** Resnet Tutorial.

To wrap up, deeper models show a degradation problem, which means the performance of them are worse than the shallow models though we use batchnorm to solve the gradient vanishing problem.

Researchers try to solve this problem by introducing a new type of block, called **residual block**. They do not just simply stack simple layers but create a structure of block, whose contains two paths to process the information. The first path is the **original path**, which is the same as the general neural network. The second path is the **shortcut path**, add directly the input to the block to the output of the block.

The philosophy behind this is that **if we could use a simpler model to learn the function, then we just change the later layers to a identity mapping and it would at least not make the performance worse.**

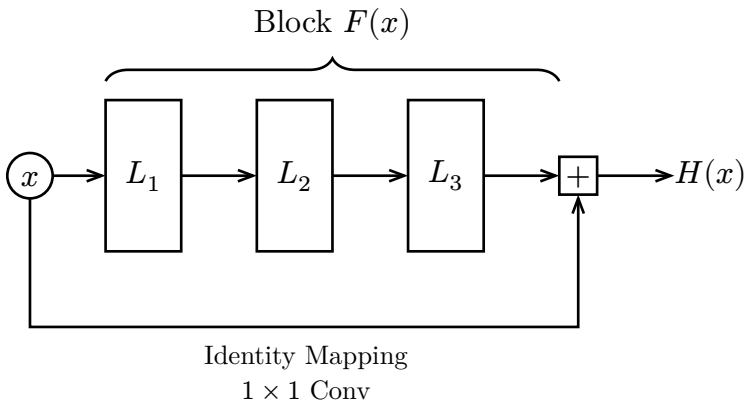
1.2. Mathmatical Formulation

Since it is a 'net' architecture, the basic idea is the same with the general neural network. So we only model the important new features.

What a neural network actually does is to find a function $f(x)$ that maps the input x to the output, such as a valuea or a class. For a deep network, the late layers receive

the input which is already highly integrated by the early layers and it needs to map this informatic expression to a more complex one. This mapping is quite hard to learn.

Imagine we want the later layers to learn the mapping $H(x)$, and the input is x, instead of learning the mapping directly, we could adjust the later layers to learn the mapping $H(x) - x$. This way, the later layers only need to learn the difference between the original input and the output, which is much easier to learn. And we call this function $F(x) = H(x) - x$. Then we add the output of the layers to the original input, we could get the final output. That is $H(x) = F(x) + x$ (element-wise addition). Graphically, it looks like this:



1.3. Details

Although the design of the residual block is quite simple, there are some details that need to be considered.

1. We need to add the input to the block to the output of the block, which means **the shape of the two tensors need to be exactly the same**. Otherwise, we could only **concatenate them** through the dimension of which is the same.

If the dimension is not the same, we could use a **1x1 convolution** to adjust the dimension to the same.

First let go a detour of how a convolution works by only look at the dimensions. Imagine we have a convolutional layer `Conv2d(in_channels=3, out_channels=6, kernel_size=3)`. It means we have 3 input channels, 6 output channels, and a kernel size of 3x3. What it does is just use 6 kernels of $3 \times 3 \times 3$ to operate on the input and get a tensor of $6 \times H \times W$.

In Residual Neural Network, instead of using basic layers, we use **bottleneck layers** to reduce the dimension. A bottleneck layer is list of convolutional layers, usually 3 layers, with the first layer to reduce the dimension, the second layer to downsample and the third layer to increase the dimension. It keeps most of the opearations in the middle layer, where the dimension is the smallest, to reduce the computation cost.

So generally, after a block, the dimension would become 4 times the original dimension and the height and width would be halved. So we just use a **1x1 convolution layer**

to downsample the original input and increase the dimension to the same as the output of the block, this helps us in the shortcut path where we add the identity mapping element-wise to the output of the block.