Qixin Wang

Mentor: Ziheng, Chen

Direct Reading Program

12/6/2020

**The design of Residual Block**

This semester, we work on the Machine Learning class by taking the free courses on Coursera. Then I figure out that the convolutional neural networks are very interesting, so we then study the specific convolutional neuronal networks course on Coursera. Within the lecture, I want to talk about the reason we need to use Residual Network, or Resnet.

The neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The neural networks include two networks:

Forward network, where the information moves in only one direction—forward—from the input nodes, to the hidden nodes and finally to the output nodes. There are no cycles or loops in the network.



The gradient back propagation which computes the gradient of the loss function with respect to the weight or parameter of the network for a single input–output example, and does so efficiency, unlike a naive direct (forward network) computation of the gradient with respect to each weight individually.



However, in these network, there is a problem called vanishing gradient, which means the major issues we have when we train our networks with backpropagations. Each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of training. The problem is that in some cases, the gradient will be vanishingly small such as ResNets, effectively preventing the weight from changing its value.

So, what is residual networks, or ResNets? In common neural networks with many layers, deep layers of the networks are difficult to train because of vanishing and exploding gradient types of problems, so we skip connections which allows us to take the activation from one layer and suddenly feed it to another layer even much deeper in the neural network. By using that, we build ResNets which enables us to train very deep networks. ResNets are built out of something called a residual block, so using residual blocks allows us to train much deeper neural networks. And the way we build a residual network is by taking many of these residual blocks, blocks like these, and stacking them together to form a deep network.

For network we discussed in forward network and most backpropagation, the network looks like this:



We call such network a plain network and this is the terminology of the ResNets paper.

To turn this into ResNets, we add all those skip connections although those short like a connection like so. Within the residual networks, every two layers ends up with that additional change that we saw on the previous picture to turn each of these into residual block. Therefore, this picture shows five residual blocks stacked together, and this is a residual network.



 If we draw our network without all the extra residual and all the extra short cuts or skip connections, it turns out that if we use our standard optimization algorithm such as a gradient descent or one of the fancier optimization algorithms to train on plain network will causes great training error.



When we look to our residual block, suppose our input is x. When the input activation goes to first ReLu with parameter F1, we change the activation from x to F1(x). When the F1(x) reaches the second ReLu with parameter F2, we change the activation from F1(x) to F2(F1(x)). Then, when the network leaves this block, we will add the input activation x to F2(F1(x)), so the output is F2(F1(x))+x. Here our F1 and F2 functions are not linear, and the reason why they are not linear is for training. If our parameter is linear, not matter how many layers our networks went through, the final output is F2\*F1\*x, which is as the same effect we can train with single layer. Thus, non-linear activations make our network complex and capable to perform complicated tasks.



From the picture above, empirically increasing the number of layers results the training error will tend to decrease after a while but then they'll tend to go back up. And in theory as we make a neural network deeper, it should only do better on the training set. In theory, having a deeper network should only have positive effects, but in practice, having a plain network that is very deep means that all the optimization algorithm just has a much harder time training. Thus, in reality, the training error gets worse if we pick a network that's too deep. But what happens with ResNet is that even as the number of layers gets deeper, we can have the performance of the training error kind of keep on going down. But by allowing it to go much deeper in the neural network, this really helps with the vanishing and exploding gradient problems and allows us to train much deeper neural networks without appreciable loss in performance.

So, why do ResNets work so well? If we make a plain network deeper, it can hurt the ability to train the network to do well on the training set. And that's why sometimes we don't want a plain network that is too deep.

Picture 1:



As in picture shown above, in our plain network, if the input is a1 and suppose the layer is Fn; the activation after going through each layer is an; the parameter of each layer is ωn; the output is L, and the activation an = Fn (an-1, ωn) for all n be integer.

Picture 2: 

As I mentioned before, there is a vanishing gradient if our network goes deep. The vanishing gradient of the output $\frac{∂y}{∂ωn}$ can be written as the partial derivative of the partial derivative of each layer’s gradient, $\frac{∂L}{∂ωn}= \frac{∂L}{∂an}\*\frac{∂an}{∂an-1}\*\frac{∂an-1}{∂an-2}\*…\*\frac{∂a3}{∂a2}\*\frac{∂a2}{∂ωn}$ , where $\frac{∂an}{∂an-1}$is the gradient of each layer. For our plain network, if we want to have a result, the gradient of output $\frac{∂y}{∂L}$ cannot be 0, which implies the none of the multiplication factors are 0. Thus, we suppose P ($\frac{∂L}{∂ωn}$= 0) = p, and we want to see what is going on with this p. P($\frac{∂L}{∂ωn}$= 0) = 1- $P(\frac{∂L}{∂ωn}\ne 0)$ ≥ 1 – P($\frac{∂L}{∂an}\ne 0$)\* P($\frac{∂an}{∂an-1}\ne 0$)\*…\* P($\frac{∂a2}{∂ωn}\ne 0$). Without loss of generality, we suppose the probability of each gradient is q, so P ($\frac{∂L}{∂ωn}$= 0)≥1-(1-q)n-1. Since q is a probability, q≤1, which means by geometry formula, (1-q) n-1=0 as n approach ∞. Therefore, the probability of the plain network to have a vanishing gradient will be large if we train our network very deep. So how does our neural network reduce this vanishing gradient?

Picture 3:

In the residual block, since previously we know residual block requires the activation to be the summation of previous activation and function F, which is Ḟ = F (x, ωT) +x, where the F (x, ωT) = g(ωT\*x) and g is the activation function, so partial derivation of this Ḟ is $\frac{dḞ}{dx}=1+\frac{dF}{dx}=1+ω\*g'(ωT\*x)$. Here the $\frac{dF}{dx}$ is very small, says $\frac{dF}{dx}$<<1, so $\frac{dḞ}{dx}$ is close to 1, and unlikely be 0.

Picture 4:

Usually, the activation function g is ReLU, and the ReLU function is a separated function which equals to 0 for x<0 and x for x>0. So the g’ is a discrete function which equals to 0 for x<0 and 1 for x>0. Therefore, we can tell half of the probability of g’ to be 0.

In conclusion, since the residual network has a very unlike happening vanishing gradient as the same deep network as plain network, it is more efficiency than plain network to train networks with deep layers.