Confidence estimates for neural networks

Rishabh Krishnan

May 13, 2018

Neural networks, right now, are mostly like a blackbox. What that means is that when we get the result from a neural network, we have no idea how the neural network computed it.One way to find a solution to this problem is to find a confidence interval for the neural network's answer to the question; how sure the network is of its answer. These confidence intervals can be used to measure the uncertainty of neural networks, which can be used in both scientific and industry applications.

Review

Confidence Intervals

To get a confidence interval for a distribution, if you remember from RS1, in order to get a confidence interval for distribution, we need to know the shape of the distribution, and the standard deviation. For example, for a normal distribution with standard deviation σ and mean μ, the confidence interval is :

$$\mu \pm \sigma * z^*$$

where z^* is the critical value, depending on how confident we want our interval to be. Note that the only variables in this equation are μ and σ (since z^* is a constant we can look up).

Applying this to Neural Networks

Let us model our actual values $y(x)$ in the following way:

$$y(x) = f(x) + n(x)$$

where $f(x)$ is the function that our neural network can model, and $n(x)$ is some additional noise that it can't process.
Note that n is a function of x, which means we assume that the noise will change depending on the input. That means we can't just do one universal calculation to figure out the standard deviation for the neural network.
So, what we need is, on average, for a given input, how much will our prediction differ from the actual value? There are many ways to find this.

Ways to calculate uncertainty

Maximum Likelihood

The basic idea behind this part is to maximize the likelihood that we would observe the training data , given the parameters we choose and the neural network that we have . Thus, we are trying to maximize the function:

$$P\left(y(x)|x,\mu,\sigma,N\right)$$

Note that, since the natural log function preserves order (if $x > y$, then $\ln(x) > \ln(y)$) this is equivalent to minimizing the following function:

$$-\ln\left(P\left(y(x)|x,\mu,\sigma,N\right)\right) \tag{1}$$

How will we make sure that our neural network maximizes this, we will change the architecture in the following way. Although we still have the same

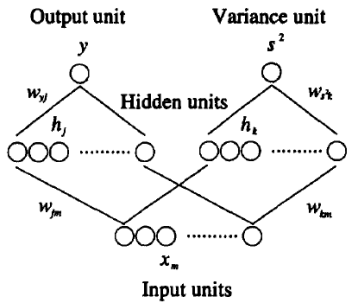

Figure 1: Neural network architecture for maximum likelihood method

inputs as before, we know have two sets of hidden units that come off of them. In effect,we have two networks with the same inputs. These separate networks train towards two different outputs; μ and σ, or the predicted value, and its uncertainty.
Since there are two outputs now, we cannot simply use the cost function that we usually use for backpropogation,($C(x) = \|y - f(x)\|$, where $f(x)$ is the prediction of the neural network, and y is the actual value for input x)), since the uncertainty network would never get trained.

Rather, we will use (1) as the cost function, so:

$$C(x) = -\ln\left(P\left(y(x)|x,\mu(x),\sigma(x),N\right)\right)$$

Note that the algebraic form of the cost function will depend on the specific distribution that we assume for the noise. For example, if we assume that the noise is normally distributed, then:

$$C(x) = -\ln\left(\frac{1}{\sqrt{2\pi\left(\sigma(x)\right)^2}} * \exp\left(\frac{-\left(y(x)-f(x)\right)^2}{2*\left(\sigma(x)\right)^2}\right)\right)$$

Thus, after training this network, we will have a network that can give us both the uncertainty and the predicted value for any input.

Bootstrapping

Statistics part

So, to understand why we do this, we first want to go through what bootstrapping is in statistics. Let us have a probability distribution P. (More informally, consider a box, in which we have a certain number of red balls, blue balls, and green balls). Now, consider a sample X of size$n$ (drawing n balls out of the box).

We want to estimate the standard deviation of the overall distribution P by using the standard deviation of the sample, $s(X)$. If we knew the distribution of P, we could then figure out how accurately our sample's standard deviation was at predicting the population standard deviation.
However, the problem is that in most situations, we don't know the probability distribution P. What we would ideally like to do is take many samples from P, which would allow us to gauge how the sample standard deviation varies across different samples. Again, we usually can't do this, because that takes a lot of time and space.
To solve this problem, the "bootstrapping" process causes us to *resample* the sample X, with the assumption that the sample is representative of the overall distribution. With the resample, we use this to determine the standard deviation of the overall population, using the variance between the samples.
Going back to the box example, if we originally drew 100 balls from the box, but then we lost access to the box, so we couldn't sample any more and see how different samples varied from each other. In order to then get a good approximation, we would assume that our 100 balls had a similar distribution to the sample in the box; then, we would take samples of 10 balls from our sample of 100 balls, and try to find out how these samples vary.

Applying this to machine learning

Consider the regression problem where we want to predict a person's adult height based on the adult height of their family members.
To solve this problem, there is technically an infinite dataset that we could use to train a neural network that we were using. If we had access to the height of a person, and the height of their family members, for every person that ever lived or will live, we would have all possible data for this problem.

However, we obviously don't have all that data, we only have a sample of it. This is our sample X. Usually, what we do is take one neural network and train it on our whole sample. This gives us an estimate for the expected value of the adult height for each possible person, but it doesn't give us any estimate of the variation between people.

The bootstrapping method creates k different neural networks, and trains each one on a random sample of our training data (a sample of our sample). Then, for every input, we pass it into each of the neural networks. This gives a set of results, which we will call S. To get the estimated standard deviation for that input, we take the standard deviation of the results from all of the networks.

Note that this does not require us to assume that the distribution is normal,

like the maximum likelihood method did. However, it also requires much more time, since have to train more neural networks.