Distilling the Knowledge in a Neural Network

Problem Statement:

In large-scale machine learning, we typically use very similar models for the training stage and the deployment stage despite their very different requirements: For tasks like speech and object recognition, training must extract structure from very large, highly redundant datasets. Deployment to a large number of users, however, has much more stringent requirements on latency and computational resources. Making predictions using an ensemble of models or a single very large model is cumbersome and may be too computationally expensive to allow deployment to a large number of users.

Contribution:

In an earlier work, Caruana et al. have shown that it is possible to compress the knowledge in an ensemble into a single model which is much easier to deploy. The authors of this paper develop this approach further using a different compression technique - “distillation”.

Method:

Knowledge is simply a learned mapping from input vectors to output vectors. For cumbersome models that learn to discriminate between a large number of classes, the normal training objective is to maximize the average log probability of the correct answer, but a side-effect of the learning is that the trained model assigns probabilities to all of the incorrect answers and even when these probabilities are very small, some of them are much larger than others. The relative probabilities of incorrect answers tell us a lot about how the cumbersome model tends to generalize. An image of a BMW, for example, may only have a very small chance of being mistaken for a garbage truck, but that mistake is still many times more probable than mistaking it for a carrot. When we are distilling the knowledge from a large model into a small one, we can train the small model to generalize in the same way as the large model.

An obvious way to transfer the generalization ability of the cumbersome model to a small model is to use the class probabilities produced by the cumbersome model as “soft targets” for training the small model. But there is a problem with that approach. Take an example of MNIST dataset, one version of a 2 may be given a probability of 10^{-6} of being a 3 and 10^{-9} of being a 7 whereas for another version it may be the other way around. This is valuable information that defines a rich similarity structure over the data (i.e., it says which 2’s look like 3’s and which look like 7’s) but it has very little influence on the cross-entropy cost function during the transfer stage because the probabilities are so close to zero. Caruana et al. circumvent this problem by using the **logits** (the inputs to the final softmax) rather than the probabilities produced by the softmax as the targets for learning the small model and they minimize the squared difference between the logits produced by the cumbersome model and the logits produced by the small model.

The authors present a more general solution, called “distillation”, which raises the temperature of the final softmax until the cumbersome model produces a suitably soft set of targets. Then they use the same high temperature when training the small model to match these soft targets.

Distillation:

Neural networks typically produce class probabilities by using a **softmax** output layer that converts the logit, z_i, computed for each class into a probability, q_i, by comparing z_i with the other logits.

$$q_i = \frac{exp(\frac{z_i}{T})}{\sum_j exp(\frac{z_j}{T})}$$

where T is a temperature that is normally set to 1. Using a higher value for T produces a softer probability distribution over classes.

Knowledge is transferred to the distilled model by training it on a transfer set and using a soft target distribution for each case in the transfer set that is produced by using the cumbersome model with a high temperature in its softmax. The same high temperature is used when training the distilled model, but after the training is complete the distilled model uses a temperature of 1 during test time.

When the correct labels are known for all or some of the transfer set, this method can be significantly improved by also training the distilled model to produce the correct labels. One way to do this is to use the correct labels to modify the soft targets, but the authors suggest that a better way is to simply use a weighted average of two different objective functions. The first objective function is the cross entropy with the soft targets and this cross entropy is computed using the same high temperature in the softmax of the distilled model. The second objective function is the cross entropy with the correct labels. This is computed using exactly the same logits in softmax of the distilled model but at a temperature of 1. The authors found that the best results were generally obtained by using a considerably lower weight on the second objective function. Since the magnitudes of the gradients produced by the soft targets scale as $1/T^2$ it is important to multiply them by T^2 when using both hard and soft targets. This ensures that the relative contributions of the hard and soft targets remain roughly unchanged if the temperature used for distillation is changed while experimenting with meta-parameters.