Neural Network for Machine Learning - Geoffrey Hinton

Contents

1 Ways to make neural networks generalize better

Different regularization methods have different effects on the learning process. For example L2 regularization penalizes high weight values. L1 regularization penalizes weight values that do not equal zero. Adding noise to the weights during learning ensures that the learned hidden representations take extreme values. Sampling the hidden representations regularizes the network by pushing the hidden representation to be binary during the forward pass which limits the modeling capacity of the network.

2 Combining multiple neural networks to improve generalization

So we should aim to make the individual predictors disagree without making them being poor predictors. The art is to have individual predictors that make errors very different from one another but each one is fairly accurate.

In case of mixtures of experts, the aim of data clustering is not to find clusters of input vectors which are similar. Rather, what we are interested in is the similarity in the input output mappings. In other words, we want each cluster to have a relationship between input and output that can be well-modeled by one local model.

2.1 Dropout

There are 2 ways to combine the outputs of multiple models. The first one is the mixture model. In a mixture, we combine the models by averaging their output probabilities. The second way is using the product of the individual model probabilities. Here, we use the geometric mean of the outputs from all the models. The geometric mean generally adds up to less than one. So, we need to normalize by the sum. In a product, the low probability output by one model has the veto power over the other models which is not substantially true for the mixture.

Now, let us talk about the dropout. This is an alternative procedure to Bayesian learning. It does not work quite as well as the correct Bayesian thing, but it is much more practical. In dropout, each time we present a training example, we randomly omit each hidden unit with probability 0.5. So, we are randomly sampling from 2^H different architectures where H is the number of hidden units. All the hidden units share weights over the architectures. What it means is that when one hidden unit is present in one architecture, it has the same weight as that of other architectures. So, we can think of dropout as a form of model averaging. Theoretically, we sample from 2^H models although in practice, many of them will never be sampled and the models which are sampled get only one training example. This can be thought of as the most extreme form

of bagging. The sharing of weights over the models means that each model is very strongly regularized by the others. This is much better regularizer than L_1 or L_2 penalties that pull the weights towards zero, whereas dropout makes the model weights tend to pull towards the correct value. At test time, we use all the hidden units with all the outgoing weights halved. This is not exactly the same as averaging all the separate dropped out models, but it is a pretty good approximation, and it is fast.

Now, if you have a deep net which is significantly overfitting, dropout will usually reduce the number of errors by a lot. Also, any net that uses early stopping, can do better by using dropout at the cost of taking quite a lot longer to train. If your deep net is not overfitting, you should use a bigger one with dropout. This ensures that you have enough computational power.

2.1.1 Hinton's Idea for Dropout

There is another way to think about dropout which is originally how Geoffrey Hinton arrived at the idea. If a hidden unit knows which other hidden units are present, it can co-adapt to them on the training data. What it means is the real signal that is training a hidden unit is to try to fix up the errors leftover when all the other hidden units have had their say. That is what is being backpropagated to train the weights of each hidden unit. This causes complex co-adaptation between the hidden units. And, this is likely to go wrong when the data changes, i.e. for the new test data. So, if you depend on the complex co-adaptation of hidden units to get things right on the training data, it is quite likely to not work so well on the new test data. It is like the idea that a big, complex conspiracy involving lots of people is almost certain to go wrong because there is always things they overlook. And, if there is a large number of people involved, one of them will behave in an unexpected way, then others will be doing the wrong thing. It is much better if you want conspiracies, to have lots of little conspiracies. Then, in case any unexpected things happen and many of the little conspiracies fail, some will still succeed. So, in dropout, if a hidden unit has to work with combinatorially many sets of other units, it is more likely to do something that is individually useful, rather then only useful because of the way particula other hidden units are collaborating with it. It will tend to do something that is marginally useful given what its co-workers also want to achieve. This is the reason Hinton opines why large nets with dropout work better.

3 Modeling Binary Data with Boltzmann Machine

Let us discuss about 2 ways of producing models of data in particular binary vectors. The most natural way to think about generating a binary vector is first to generate the states of some latent variables and then use these latent variables to generate the binary vector. So in a causal model, we use 2 sequential steps.

We first pick the states of the hidden units from their prior distributions. Often in the causal model, these will be independent in the prior. So their probability of turning on, if they are binary latent variables, would just depend on some bias that each one of them has. Then we pick the visible states from their conditional distribution given the hidden states. The probability of generating a visible vector, $\mathbf{v}$, is computed by summing over all possible hidden states. Each hidden state is an explanation of $\mathbf{v}$.

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{h}) p(\mathbf{v}|\mathbf{h}) \tag{1}$$

This is probably the most natural way of thinking about generating data. Another way of generating data is using Boltzmann machine, which is an energy based model and does not generate data causally. Instead, everything is defined in terms of the energies of joint configurations of the visible and hidden units. There are two ways of relating the energy of a joint configuration to its probability. We can simply define the probability of the joint configuration to be proportional to the exponential of the negative energy of that joint configuration.

$$p(\mathbf{v}, \mathbf{h}) \propto \exp(-E(\mathbf{v}, \mathbf{h})) \tag{2}$$

The energy of a joint configuration is given by

$$-E(\mathbf{v}, \mathbf{h}) = \sum_{i \in vis} v_i b_i + \sum_{k \in hid} h_k b_k + \sum_{i<j} v_i v_j w_{ij} + \sum_{i,k} v_i h_k w_{ik} + \sum_{k<l} h_k h_l w_{kl} \tag{3}$$

Here, v_i is the binary state of unit i in vector $\mathbf{v}$. The ***less than*** operator in the 3^{rd} and 5^{th} terms on RHS is used to index every non-identical pair of (i, j) and (k, l) once.

Now, to make the proportion sign in equation 2 an equality, we need to normalize the RHS by all possible joint configurations of visible and hidden units.

$$p(\mathbf{v}, \mathbf{h}) = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-E(\mathbf{u}, \mathbf{g}))} \tag{4}$$

Physicists often call the normalizer the partition function. Notice that it has exponentially many terms. Now the marginal probability distribution of visible units $\mathbf{v}$ can be written as

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{u},\mathbf{g}} \exp(-E(\mathbf{u}, \mathbf{g}))} \tag{5}$$

In reality, we cannot compute the normalizing term or the partition function for a network with more than a few hidden units, because it will have exponentially many terms. So, we use Markov Chain Monte Carlo to get samples from

the model starting from a random global configuration with probability proportional the exponential of the negative energy of that joint configuration as in equation 2.

3.1 Restricted Boltzmann Machine

In the previous subsection, we show that how Boltzmann Machine (BM) can be used as a probabilistic model of a set of binary data vectors. Now we are going to get around the learning algorithm of BM. This algorithm is theoretically well-justified, but too slow and noisy to be used in practice. However, recently Hinton devised several ways to greatly speed up the algorithm. Now it is much more practical and even been used as part of the winning entry of a million dollar machine learning competition.

The BM learning algorithm is an unsupervised learning algorithm. What it means is that unlike the backpropagation algorithm, where we provide a set of input data with associated target labels to train, in BM, we provide only the input data without any labels. The input data vectors earn the labels through the algorithm. The learning algorithm builds a model of a set of input vectors.

The goal of learning is to maximize the product of probabilities that BM assigns to the binary vectors in the training set, which is equivalent to maximizing the sum of the *log* probabilities. It is also equivalent to maximizing the probability that we would obtain exactly the N training cases if we did the following.

- Let the network settle to its stationary distribution N different times with no external input.
- Sample the visible vector once each time.

What is exactly being learned is the set of parameters that define a distribution over the visible vectors.

Now, from equation 5, we can derive the following equations. It might seem a little bit confusing, but I am going to describe them both mathematically and intuitively.

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{v}} - \langle s_i s_j \rangle_{model} \tag{6}$$

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model} \tag{7}$$

In equation 6, the term on LHS is the derivative of log probability of the training vector $\mathbf{v}$ under the model. The first term on RHS is the expected value of product of states at thermal equilibrium when $\mathbf{v}$ is clamped on the visible units. The second term is the expected value of product of states at thermal equilibrium with no clamping. This derivative is so simple because the probability of a global configuration at thermal equilibrium is an exponential function of its energy. So, setting to equilibrium makes the log probability a linear function of the energy. A significant difference between backpropagation and BM

learning is that in BM, the process of setting to thermal equilibrium propagates information about the weights and so, we do not need to back propagate the derivatives.

Let us take a look at equations 6 and 7 from another perspective. We call the first and second terms on RHS as positive and negative phases, respectively. Now, from equation 5 , we can say that the positive phase finds hidden configurations that work well with $\mathbf{v}$ and lowers their energies. On the other hand, the negative phase finds the joint configurations that are the best competitors to the positive ones and raises their energies.

Now in order to learn this learning rule, we need to collect these positive and negative statistics. The positive statistics are the ones when you have data clamped on the visible units, and the negative statistics are the ones when we do not have data clamped and that we are going to use for unlearning (Hopfield nets). An inefficient way of collecting these statistics was proposed by Hinton and Sejnowski in 1983. The idea is summarized below.

An Inefficient Way of Collecting the Statistics for BM Learning
Hinton and Sejnowski (1983)

Positive Phase

- Clamp a data vector on the visible units and set the hidden units to random binary states.
- Update the hidden units one at a time until the network reaches thermal equilibrium at a temperature of 1.
- Sample $\langle s_i s_j \rangle$ for every connected pair of units.
- Repeat for all data vectors in the training set and average.

Negative Phase

- Set all the units to random binary states.
- Update all the units one at a time until the network reaches thermal equilibrium at a temperature of 1.
- Sample $\langle s_i s_j \rangle$ for every connected pair of units.
- Repeat many times (how many?!) and average to get good estimates.

As stated in the last step of the negative phase, we have to repeat many times and then average to get a reasonable estimate. It is very difficult to know how many times we need to repeat because certainly in the negative phase, the energy landscape contains many local minima, which are fairly separated and have about the same energy. As an example, if we use BM to model a set of images, we expect there to be reasonable images, all of which have about the same energy as well as unreasonable images with very high energy contents. And, a large fraction of the image space belongs to the bad energy states. Also, if multiple modes are present, it is very unclear how many times we need to repeat the process to be able to sample those modes.