

Role of ERROR SURFACES in WEIGHT SPACE for WEIGHT MODIFICATION RULES

The following is a written version of the (loose) formalism I presented in class to help us think about the role of "error surfaces" (Criterion Function surfaces) in "weight space" vis-a-vis the weight update rules used in some training algorithms.

Assume we have a neural network for which we wish to develop a training algorithm. Assume there are N weights, and these are the only variables available to us.

1. Conceptualize a "weight space" of N dimensions, each dimension corresponding to one of the N adjustable weights (we thus have a way of representing all possible combinations of weights).
2. Decide on a mapping (or, class of mappings) we want the net to "learn." This could be as general a statement as "any mapping this net is capable of learning."
3. Create (per one's own whim) some function--call it a Criterion Function--which generates a real value for each possible combination of the N weights. Define the Criterion Function such that it has (at least a local) minimum value for all those settings of the N weights which yield a "satisfactory" implementation of the desired mapping. (Of course, we also require that larger values are assigned to weight combinations that yield "unsatisfactory" results.)
4. Add a dimension to the N -dimensional weight space, and in this extra dimension, represent the real value associated with each possible combination of values in the other N dimensions. This can be thought of as creating a (hyper)surface in the weight space. In general, this surface will have "hills", "valleys", "ravines", "planes", etc., depending on the Criterion Function that was defined, the architecture of the net, the transfer function of the elements in the net, and the specific function being learned.

For example, if the network consists of a single element with N input weights and a linear transfer function, and if the Criterion Function defined is the simple "squared error function" = $(\text{desired output} - \text{actual output})^2$, then the shape of the surface will be a simple quadratic--i.e., one which has a unique minimum, and all "paths" leading to it are monotonic decreasing.

5. Assuming the mathematics is tractable, determine an analytic expression for the partial derivatives of the Criterion Function with respect to each of the N adjustable weights. These derivatives determine the "direction" of maximum change on the Criterion Function surface, i.e., the gradient.
6. Now, look at the actual implementation that will be used in doing the "training." Determine what information can be made available to the trainer during the process. Then, develop a computation schema for updating the variables of the problem (the N weights), wherein the only data used are those available to the trainer. If it is possible to come up with a computation schema that is exactly analogous to the expression for the negative of the partial derivatives developed above, then the update rule would construct a gradient descent ("steepest descent") path on the Criterion Function surface. Clearly, this will be possible for only special combinations of elements, architectures, and Criterion Functions. [In practice, the gradient descent is only approximated, since finite weight changes are made, instead of the required infinitesimal ones.]

For the example given above, the partial derivatives turn out containing only the product of the element's output error and the value of the input feeding the weight being adjusted (and a possible multiplicative constant). The inputs to the net are assumed available for measurement, and since the net only has one element (layer), these are the inputs needed for the gradient calculation. Similarly, the output is assumed to be available for measurement. Further, it is assumed that the desired output is available to the trainer. Thus, all the components of the partial derivative are available to the trainer in the assumed implementation. The resulting computation schema is the well known Delta Rule.

It is only because the Delta Rule satisfies all of the above steps relative to a "squared error" Criterion Function that we can say that the Rule accomplishes a gradient descent on the "squared error" surface...and hence, yields a Least Squared Error solution.

When the computation rule is expanded to apply to the case of many output elements (but still one layer) and the Criterion Function taken to be the sum of the squared errors for each of the output elements, then we have a Least Sum of Squared Error solution.

If the computation rule is expanded to hold off making changes in the weights until all input patterns in the training set are polled, and properly normalized, then the word "mean" is added, so (for the one element case) we have a Least Mean Squared Error solution.

In the case of the back-propagation algorithm, a specific architecture, element transfer function type, and Criterion Function are given. Though the squared-error Criterion Function is the same as for the simple one-element example given earlier, the network architecture is different, so the shape of the surface is no longer the simple convex shape with only one minimum value. In fact, I suspect it is only by faith that the developers of the algorithm assumed this Criterion Function would fulfill the requirement stated in step 3 (i.e., that the function assign a minimal value to each "satisfactory" implementation of the desired mapping, with higher values for "unsatisfactory" solutions). They did not (to my knowledge) prove that their Criterion Function has this property. Empirically, however, there are ample experimental results to demonstrate that often, the minima do yield "satisfactory" implementations. [This is not the case for the Hopfield algorithm.] Nevertheless, the authors of this algorithm went through the steps of determining an analytic expression for the partial derivatives of the Criterion Function with respect to each of the N weights (this entailed putting a requirement on the transfer function of each element that it be differentiable). They then stipulated the kind of information the trainer is allowed to have concerning the operation of the net, and based on this arrived at a (clever) algorithm for updating weights which "matched" the partial derivative expression arrived at earlier. Thus, this algorithm achieves a gradient descent path on the surface of the Criterion Function. Since this surface does not have one unique minimum, the algorithm stops at any local minimum it happens to be in, and, as mentioned above, most of these turn out being "satisfactory" implementations of the desired mapping.