Feature induction by backpropagation

Edmund Ronald, Marc Schoenauer, Michèle Sebag

Feature Induction by Backpropagation

Edmund Ronald, Marc Schoenauer, Martine Sebag

CMAP, Ecole Polytechnique,91 128 Palaiseau, France
email: eronald@cmapx.polytechnique.fr

0. Abstract

A method for investigating the internal knowledge representation constructed by neural net learning is described: It is shown how from a given weight matrix defining a feed-forward artificial neural net, we can induce patterns characteristic of each of the classes of inputs classified by that net. These characteristic patterns, called prototypes, are found by a gradient descent search of the space of inputs. After an exposition of the theory, results are given for the well known LED recognition problem where a network simulates recognition of decimal digits displayed on a seven-segment LED display. Contrary to theoretical intuition, the experimental results indicate that the computed prototypes retain only some of the features of the original input patterns. Thus it appears that the indicated method extracts those features deemed significant by the net.

1. Introduction

Feature induction attempts to recover from a given trained net some input pattern whose presentation would elicit a desired output. Such an input is called a prototype. More formally, given a trained net N_W (W stands for the weights matrix), and a desired target output vector t, we wish to solve for a prototypical input vector i such that the net's computed output vector $o = N_W(i)$ approximates the target t. Feature induction is a search through input space.

Note that in a supervised learning algorithm such as back-propagation [PDP], we are given a training ensemble P of input-output pairs $<i_p, t_p>$, and search for a net – i.e. a matrix W of synaptic weights (and biases) such that for each input exemplar i_p, $N_W(i_p)$ approximates the corresponding desired output t_p. Learning is thus a search that trawls through the space of weight matrices.

To summarize the preceding discussion, both back-propagation and feature induction by prototype search can be seen as search algorithms that solve for different variables in the equation

$$N_W(i_p) = t_p \qquad \text{(EQ 1)}$$

Both methods employ gradient descent, moving through the search space in discrete steps.

The core sections of this paper are sections 2 and 3. In section 2 we derive the value of the steps by which feature induction travels through the input vector space. The latter section 3 is devoted to the description of a numerical experiment.

2. Feature Induction by Error Back-Propagation

In this section we give a fairly informal derivation for our method of prototype-search by gradient descent[1]. Our procedure for finding prototypical inputs is derived from the back-propagation search procedure in W-space (the space of weight matrices); this procedure as described in [Rumelhart Hinton & Williams 86] has been frequently and successfully employed in the training of nets, and has since seen numerous improvements. Back-propagation relies on a recursive procedure which propagates the so-called error or δ terms backwards from the output of the net. Our feature inductionmethod builds on this same set of propagated δ error terms,which are described in detail in the reference above pp 325-326. As this error term calculation is well understood, we shall not justify it in the derivation below, which only highlights the novelties of our feature induction method with respect to classical back-propagation.

In back-prop net training, for a given training exemplar $<i_p, t_p>$ in the training ensemble P, the weight matrix W gives rise to a computational error which is the distance between what the net computes –i.e. $o_p = N_W(i_p)$ – and the desired target output vector t_p. The actual distance is usually chosen to be the square euclidean distance. Back-propagation computes the gradient of this error function for each test exemplar in turn, and travels in the opposite direction in order to minimize the error over the whole training set. Feature induction, is a similar gradient descent search procedure,

1. A search method other than gradient descent, e.g. a genetic algorithm could also be employed to search the space of weights [de Garis 90], [Schoenauer, Ronald &Damour 93]

which takes place in the space of input vectors. It is now assumed that the net has already been trained, and thus the W matrix is fixed. Assume that we have a desired target output t_d, and wish to search for an input i_d such that

$$N_W(i_d) \approx t_d \qquad \text{(EQ 2)}$$

Now let E stand for the value taken by the square error function, i.e. the square of the euclidean distance, in output space, separating the desired target output t_d and the actual output $o_d=N_W(i_d)$ computed by the net with weight matrix W. Then E_d is defined by

$$E_d = \sum_n (t_{d_n} - o_{d_n})^2 \qquad \text{(EQ 3)}$$

where the index n ranges over the dimensions of the input space, or over the set of input neurons if the reader prefer the concrete over the abstract. Minimizing the error E_p by an iterative algorithm like gradient descent will allow us (provided the iterations converge) to satisfy (2).

Whereas the usual [PDP] back-propagation learning methods varies the weights, our modified algorithm searches iteratively through input space, yielding corrections to the input vector i, corrections we shall denote by $[\Delta i]$. Gradient descent means that these corrections are steps taken in opposite direction to the gradient of the error function, with the partial derivatives being taken with respect to the input neurons' activations[1]. Proportionality in the opposite direction is written as

$$[\Delta i] = -\eta \nabla E \qquad \text{(EQ 4)}$$

where η is some small fixed positive parameter called the learning rate. The error function E is a mapping from net-input space into the set of reals. Thus equation (4) is equivalent to a system of equations in partial derivatives:

$$[\Delta i]_n = -\eta \frac{\partial E}{\partial a_n} \qquad \text{(EQ 5)}$$

where n ranges over the set of input neurons, and a_n are the corresponding activation values. For the purposes of our algorithm, let us assume however that n ranges over the set of all neurons of the net. The algorithm thus involves computing all the partial derivatives in this extended system, although only those for the input neurons are of ultimate relevance. Now back-propagation learning [Rumelhart] already gives us a recursive method for estimating the weight–error derivatives δ_n defined by:

$$\delta_n = -\frac{\partial E}{\partial net_n} \qquad \text{(EQ 6)}$$

where net_n is the net-input to neuron n defined by

$$net_n = \sum_k w_{kn} a_n \qquad \text{(EQ 7)}$$

where the w_{kn} are the synaptic weights leading from the output of the precursors of neuron n to neuron n's input. The back-propagation method for deriving the δ_n iterates through the layers of the net, starting from the output, moving backwards. At each neuron n, it involves computing the weight-error derivative δ_n, and then assigning credit for this error to the preceding neurons in the net. Now if we look at some neuron k that is a direct successor of neuron n in the input layer, and we are interested in the partial derivative of the error function with respect to this input neuron's activation, (dE/d an) then, assuming that neuron j admits only neuron k as successor, though k can have many inputs as antecedents, we can obtain this partial derivative by employing the chain rule for derivation in order to rewrite equation (5) as:

1. This straightforward gradient-descent could be enhanced by employing a momentum term to smooth the search and accelerate convergence.

$$-\eta\frac{\partial}{\partial a_j}E = -\varepsilon\left(\frac{\partial}{\partial net_k}E\right)\left(\frac{\partial}{\partial a_j}net_k\right) = \delta_k\left(\frac{\partial}{\partial a_j}net_k\right) \qquad \text{(EQ 8)}$$

And as by definition (7), $net_k = \sum w_{nk}a_n$, and in this special case the summation has a single term, we finally obtain

$$-\eta\frac{\partial}{\partial a_j}E = \delta_k w_{jk} \qquad \text{(EQ 9)}$$

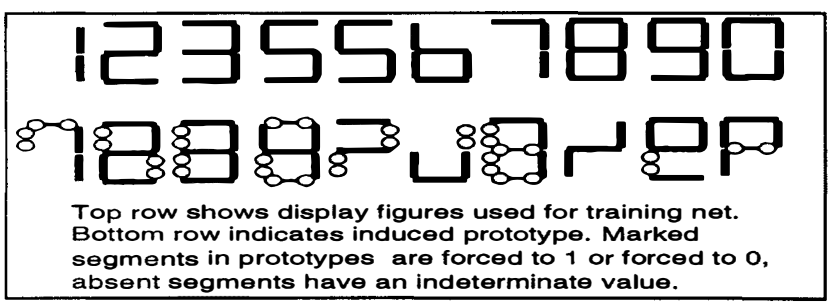
Now, in the general case, the input neuron j is of course connected to, and feeding forwards towards, several successor neurons such as k. Thus it is reasonable to complete the algorithm by crediting it with the ponderated error that each successor propagates backwards. We thus obtain our $[\Delta i]$ vector's components as

$$[\Delta i]_j = \sum_k \delta_k w_{jk} \qquad \text{(EQ 10)}$$

As classical back-prop training already gives us a method for recursively deriving the error terms δ_k, the above description suffices to define our feature induction algorithm. This is born out by experiment, as described in the next section.

The Seven-Segments example.

To investigate the usefulness of the technique on a small yet non-trivial example, a numerical experiment was performed on the so-called LED problem, simulating recognition of digits displayed on a seven-segment light emitting diode display such as are commonly employed in pocket calculators. The network topology for effecting recognition was a classical 3 layer feed-forward net: Seven neurons in the input layer (one per segment), 10 hidden neurons in a single hidden layer totally connected to the input layer, and 10 again (one per decimal digit) in the output layer, this being also completely connected to the hidden layer. The neurons embodied the usual sigmoid logistical activation function F(x) =1/(1+exp(-x)).

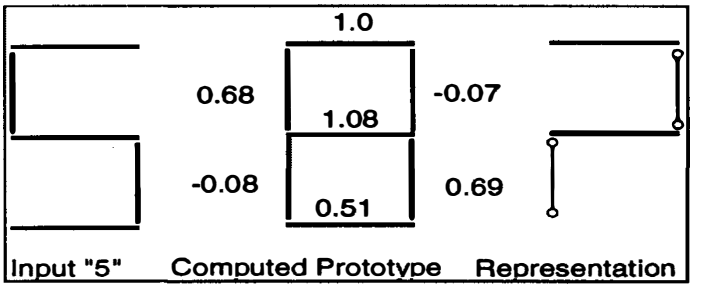


Top row shows display figures used for training net. Bottom row indicates induced prototype. Marked segments in prototypes are forced to 1 or forced to 0, absent segments have an indeterminate value.

The simulator software supplied with [Mc Lelland & Rumelhart 88] was employed as a basis for rewriting a suite of programs . The first is a standard back-propagation simulator which trains a 3-layer net and writes the weight matrix out to a file. The second program does the actual feature extraction. It reads in the net topology and the weights matrix as generated by the back-prop simulator. It then reads in the definition of a target vector, and performs a prescribed number (NITER) of forward-backwards iterations of the prototype search algorithm described in the preceding section, starting from some random input vector. The prototype input corresponding to the target vector is assumed to be that input vector which the algorithm has obtained after NITER steps.

Back-propagation training yielded a net capable of recognizing the digits in the LED problem, feature extraction then retrieved the digits' shape from the weight matrix. This may seem a trivial example; but imagine that a human were faced with digits presented as a bar-chart with the order of bars fixed but jumbled! This task would be conceptually identical to reading. However a human attempting to solve it needs an explicit representation of knowledge that is usually internalized (the features of the digits), and subconscious. This knowledge, stating which features (segments) are essential for recognition and which may be neglected, is exactly that which our method retrieves from the trained net. It can be seen in the figure that the prototype shapes retrieved by the feature induction method differ strongly from the

digits that served as training input. Clearly, the net has eliminated some redundant information, and is therefore relying on a subset of features to effect the recognition of the digits. In our figure, we have marked in black those segments whose value in the prototype exceeds 0.75, and with bubbles those whose value lies under 0.25. A typical prototype is that corresponding to the digit "5":



4. Conclusion.

Explanation of reasoning is an area where neural nets are weak, compared to expert systems: Weight matrices are hard to interpret directly, and the generalization process by which a net categorizes novel input is not yet well understood. Prototype induction can be seen as a tool to enlighten the user of a net.

In our experiments, the speed of convergence of prototype search seems to be much greater than for back-prop learning. We believe this is because the search-space is much smaller in dimensionality than in standard backprop where the search-space is weight-matrix-space. In the case of the seven segments problem, as solved by a 7-10-10 net, the input space has dimensionality of 7, while (7*10 +10*10) weights and 10+10 biases, i.e. 190 parameters are needed to define a weight matrix.

Our experiments with this prototype recovery method have encountered some potential problem areas which stem from the mathematics of the algorithm. Our method shares with classical back-prop the well known drawbacks of the gradient-descent approach: Local minima can trap the search, and oscillations can occur when the learning rate is too steep. The speed of convergence seems however to be much greater than for back-prop, perhaps because the search-space is smaller in dimensionality. Like back-prop learning, prototype search starts from a random initial input state, thus the algorithm might converge to different prototypes on subsequent runs.

A mathematics-induced problem which we have seen occur in prototype recovery, but which cannot appear in back-propagation is run-away. By this we mean convergence to fictitious prototype input vectors outside the n-dimensional unit cube. Such a prototype vector has no signification as an input, yet we have encountered this problem in the seven-segments example. It is particularly vexing, because the direction to which such a ghost points can be entirely spurious. This is a problem which we hope to investigate further.

5. Acknowledgments

The first author thanks Steve Suddarth for introducing him to back-propagation. Patrick Greussay found time to look at a rough outline of this paper

6. References

[de Garis 90] de Garis H: Genetic Programming, Proceedings of the 7th international conference on machine learning, B. Porter, R. Mooney Eds, Morgan Kaufman 1990, p. 132-139.

[Schoenauer, Ronald and Damour 93]: Schoenauer M., Ronald E. & Damour S. Evolving Nets for Control in Neuro-Nimes 93, EC2 .,

[McClelland &Rumelhart 88] McClelland J.L. & Rumelhart D.E. Explorations in parallel distributed processing, a handbook of models, programs and exercises, MIT Press 1988.

[Rumelhart Hinton & Williams 86] Rumelhart D.E., Hinton G.E. & Williams R.J.: Learning internal representations by error propagation in Parallel distributed processing, explorations in the microstructure of cognition, Vol. 1: Foundations, McClelland J.L. & Rumelhart D.E Eds, MIT Press 1986.