

1986:1 SKIPPED CONNECTIONS

## Learning representations by back-propagating errors

David E. Rumelhart\*, Geoffrey E. Hinton† & Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure<sup>1</sup>.

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors<sup>2</sup>. Learning becomes more interesting but

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

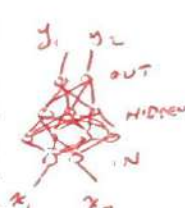
The total input,  $x_j$ , to unit  $j$  is a linear function of the outputs,  $y_i$ , of the units that are connected to  $j$  and of the weights,  $w_{ji}$ , on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output,  $y_j$ , which is a non-linear function of its total input

$$y_j = \frac{1}{1 + e^{-x_j}} \quad (2)$$



SKIP CONNECTIONS

DEFINITION OF SIMPLE NEURON (NO BIAS)

BIAS ≈ Θ

NONLINEAR ACTIVATION FUNCTION

† To whom correspondence should be addressed



$$w_{ji} = \frac{\partial x_j}{\partial y_i}$$

LETTERS TO NATURE

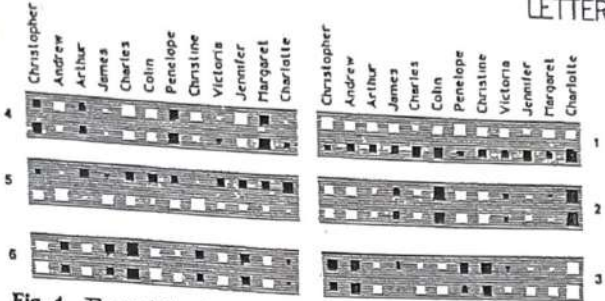


Fig. 4 The weights from the 24 input units that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles, excitatory weights; black rectangles, inhibitory weights; area of the rectangle encodes the magnitude of the weight. The weights from the 12 English people are in the top row of each unit. Unit 1 is primarily concerned with the distinction between English and Italian and most of the other units ignore this distinction. This means that the representation of an English person is very similar to the representation of their Italian equivalent. The network is making use of the isomorphism between the two family trees to allow it to share structure and it will therefore tend to generalize sensibly from one tree to the other. Unit 2 encodes which generation a person belongs to, and unit 6 encodes which branch of the family they come from. The features captured by the hidden units are not at all explicit in the input and output encodings, since these use a separate unit for each person. Because the hidden features capture the underlying structure of the task domain, the network generalizes correctly to the four triples on which it was not trained. We trained the network for 1500 sweeps, using  $\epsilon = 0.005$  and  $\alpha = 0.5$  for the first 20 sweeps and  $\epsilon = 0.01$  and  $\alpha = 0.9$  for the remaining sweeps. To make it easier to interpret the weights we introduced 'weight-decay' by decrementing every weight by 0.2% after each weight change. After prolonged learning, the decay was balanced by  $\partial E/\partial w$ , so the final magnitude of each weight indicates its usefulness in reducing the error. To prevent the network needing large weights to drive the outputs to 1 or 0, the error was considered to be zero if output units that should be on had activities above 0.8 and output units that should be off had activities below 0.2.

WEIGHT DECAY

resulting from the effect of  $i$  on  $j$  is simply

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} w_{ji}$$

DEFINITION OF  $\frac{\partial E}{\partial y_i}$

so taking into account all the connections emanating from unit  $i$  we have

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j} w_{ji} \quad (7)$$

DEFINITION OF  $\frac{\partial E}{\partial y_i}$

We have now seen how to compute  $\partial E/\partial y$  for any unit in the penultimate layer when given  $\partial E/\partial y$  for all units in the last layer. We can therefore repeat this procedure to compute this term for successively earlier layers, computing  $\partial E/\partial w$  for the weights as we go.

One way of using  $\partial E/\partial w$  is to change the weights after every input-output case. This has the advantage that no separate memory is required for the derivatives. An alternative scheme, which we used in the research reported here, is to accumulate  $\partial E/\partial w$  over all the input-output cases before changing the weights. The simplest version of gradient descent is to change each weight by an amount proportional to the accumulated  $\partial E/\partial w$

$$\Delta w = -\epsilon \partial E/\partial w \quad (8)$$

GRADIENT UPDATE

This method does not converge as rapidly as methods which make use of the second derivatives, but it is much simpler and can easily be implemented by local computations in parallel hardware. It can be significantly improved, without sacrificing the simplicity and locality, by using an acceleration method in which the current gradient is used to modify the velocity of the point in weight space instead of its position

$$\Delta w(t) = -\epsilon \partial E/\partial w(t) + \alpha \Delta w(t-1) \quad (9)$$

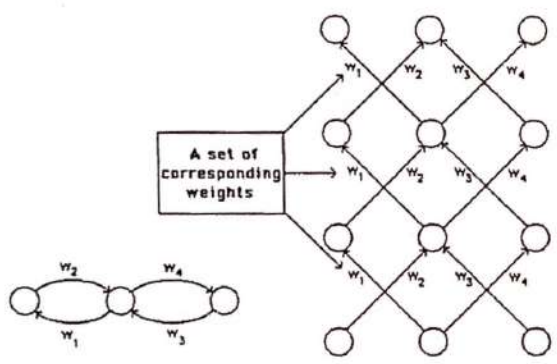
ACCELERATION GRADIENT UPDATE

where  $t$  is incremented by 1 for each sweep through the whole set of input-output cases, and  $\alpha$  is an exponential decay factor between 0 and 1 that determines the relative contribution of the current gradient and earlier gradients to the weight change.

To break symmetry we start with small random weights. Variants on the learning procedure have been discovered independently by David Parker (personal communication) and by Yann Le Cun<sup>3</sup>.

One simple task that cannot be done by just connecting the input units to the output units is the detection of symmetry. To detect whether the binary activity levels of a one-dimensional array of input units are symmetrical about the centre point, it is essential to use an intermediate layer because the activity in an individual input unit, considered alone, provides no evidence about the symmetry or non-symmetry of the whole input vector, so simply adding up the evidence from the individual input units is insufficient. (A more formal proof that intermediate units are required is given in ref. 2.) The learning procedure discovered an elegant solution using just two intermediate units, as shown in Fig. 1.

SYMMETRY INTERMEDIATE UNITS



RECURRENT NET

Fig. 5 A synchronous iterative net that is run for three iterations and the equivalent layered net. Each time-step in the recurrent net corresponds to a layer in the layered net. The learning procedure for layered nets can be mapped into a learning procedure for iterative nets. Two complications arise in performing this mapping: first, in a layered net the output levels of the units in the intermediate layers during the forward pass are required for performing the backward pass (see equations (5) and (6)). So in an iterative net it is necessary to store the history of output states of each unit. Second, for a layered net to be equivalent to an iterative net, corresponding weights between different layers must have the same value. To preserve this property, we average  $\partial E/\partial w$  for all the weights in each set of corresponding weights and then change each weight in the set by an amount proportional to this average gradient. With these two provisos, the learning procedure can be applied directly to iterative nets. These nets can then either learn to perform iterative searches or learn sequential structures<sup>4</sup>.

Another interesting task is to store the information in the two family trees (Fig. 2). Figure 3 shows the network we used, and Fig. 4 shows the 'receptive fields' of some of the hidden units after the network was trained on 100 of the 104 possible triples.

RECEPTIVE FIELDS

So far, we have only dealt with layered, feed-forward networks. The equivalence between layered networks and recurrent networks that are run iteratively is shown in Fig. 5.

The most obvious drawback of the learning procedure is that the error-surface may contain local minima so that gradient descent is not guaranteed to find a global minimum. However, experience with many tasks shows that the network very rarely gets stuck in poor local minima that are significantly worse than the global minimum. We have only encountered this undesirable behaviour in networks that have just enough connections to perform the task. Adding a few more connections creates extra dimensions in weight-space and these dimensions provide paths around the barriers that create poor local minima in the lower dimensional subspaces.

LOCAL MINIMA

MORE CONNECTIONS

The learning procedure, in its current form, is not a plausible model of learning in brains. However, applying the procedure to various tasks shows that interesting internal representations can be constructed by gradient descent in weight-space, and this suggests that it is worth looking for more biologically plausible ways of doing gradient descent in neural networks.

We thank the System Development Foundation and the Office of Naval Research for financial support.

Received 1 May; accepted 31 July 1986

1. Rosenblatt, F. *Principles of Neurodynamics* (Spartan, Washington, DC, 1961)
2. Minsky, M. L. & Papert, S. *Perceptrons* (MIT, Cambridge, 1969)
3. Le Cun, Y. *Proc. Cognitive* 85, 599-604 (1985).
4. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (eds Rumelhart, D. E. & McClelland, J. L.) 318-362 (MIT, Cambridge, 1986)