Training wheels for encoder networks

Peter Anderson

Follow this and additional works at: http://scholarworks.rit.edu/article

Recommended Citation

This Article is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Articles by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Training Wheels for Encoder Networks

Peter G. Anderson*
Computer Science Department
Rochester Institute of Technology
Rochester, NY 14623-5608

Abstract

We develop a new approach to training encoder feed-forward neural networks and apply it to two classes of problems.

Our approach is to initially train the network with a related, relatively easy-to-learn problem, and then gradually replace the training set with harder problems, until the network learns the problem we originally intended to solve.

The problems we address are modifications of the common $N$-$2$-$N$ encoder network problem with $N$ exemplars, the unit vectors, $e_k$ in $N$-space.

Our first modification of the problem is to use objects consisting of paired 1’s $(e_k + e_{k+1})$, with subscripts taken mod $N$). This requires an $N$-$2$-$N$ net to organize the images of the exemplars in 2-space ordered around a circle.

Our second modification is to use patterns consisting of two objects; each object is a pair of adjacent 1’s, the objects must be separated from each other. This problem can be learned by a $N$-$4$-$N$ network which must organize the images of the exemplars in 4-space in the form of a mobius strip.

The easy-to-learn problem in both cases involves replacing the the two-ones signal $e_k + e_{k+1}$ with a block signal of length $B$: $e_k + e_{k+1} + \cdots + e_{k+B-1}$.

In several cases, our method allowed us to train networks that otherwise fail to train. In some other cases, our method proved to be ten times faster than otherwise.

1 Training wheels and topology

$N$-$2$-$N$ encoder networks can be difficult to train, and the method we use in this paper often makes the training easier, and in some cases make it possible (see [1] for a related result). Our method also allows the networks to build their internal representations appreciating an aspect of the topological relationships among the data. To illustrate this relationship, consider the following three exemplars: $e_1$, $e_2$, and $e_3$ in $R^3$:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\]

If we consider them as three "images," we may regard the first two as similar to each other and dissimilar from the third. Usual distance metrics—Euclidean, Hamming, correlation—give these three points pairwise equal separations. Our method addresses this by "blurring" the images, smearing the single-spike signal to a block:

\[
\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}
\]

The Hamming distances (disagreement count) between pairs of these three blurred exemplars now reflects the similarity we desire: the distance between the first two is 2, and the distance between the third and either of the first two is 6. We call the blurred exemplars the original exemplars with "training wheels."

By using the blurred exemplars to start network training, we find that training often succeeds in cases for which it previously failed, and is many times faster than without this aid. Furthermore, the behavior of the resulting network can be considered improved. The improved behavior shows up in case the input is noisy

*anderson@cs.rit.edu
or the network is somehow damaged. In cases where the output should have a single positive activation but for some reason has a second one, the second one will very likely be adjacent to the desired positive one. Furthermore, a continuous change of input from \( e_k \) to \( e_{k+1} \) (such as \((1-t)e_k + te_{k+1}\) for \(0 \leq t \leq 1 \)) will have a similar smooth change in the network’s output. A network trained without this method may have a jumpy change in output values corresponding to a smooth change in input values. (The details of this experimentation will be presented elsewhere.)

2 Encoder networks

Encoder networks (also known as encoder-decoder, compression, and hour-glass networks) are feed-forward neural networks with \( N \) input nodes, \( M \) hidden nodes, and \( N \) output nodes (i.e., \( N-M-N \) networks). Training these networks can be very challenging when \( M \ll N \). They are trained so their output values (approximately) match their input values on a training set. A favorite toy problem is to train an \( N-2-N \) network on \( N \)-tuple exemplars of the form \((0, \ldots, 0, 1, 0, \ldots, 0)\) (this problem and generalizations are discussed below).

Encoder networks may be valuable for a wide variety of applications; here we list four.

The network may learn to reproduce a certain class of signals. Then, given a signal that deviates from those in its training set, it may produce a signal similar to the ones in the training set. For example, a trained network may be able to convert letters into a given font or remove scratches from phonograph records.

An \( N-M-N \) net compresses a signal of \( N \) values into one of \( M \) values, so these systems are potentially useful for data compression. The first half (encoder) of the network is a compression function, \( c : \mathbb{R}^N \rightarrow \mathbb{R}^M \), and the second half (decoder) is the decompression function, \( d : \mathbb{R}^M \rightarrow \mathbb{R}^N \). Neural network training organizes the compression and decompression algorithms according to the statistics of the training exemplars.

In the area of computer vision or pattern recognition, one may want to construct an \( N-M-K \) network, say, to classify \( N \)-pixel images into one of \( K \) patterns. One approach to this problem is: first train an \( N-M-N \) network to reproduce the input training exemplars, then use the encoder half of the network as an initial approximation to the first layer of weights of the \( N-M-K \) net. The idea is that the hidden layer captures the “essential” aspects of the patterns.

A fourth application area is that of training a system to determine whether a system is operating correctly. In some cases it is easy to obtain data representing “correct operation” but nearly impossible to obtain data for “incorrect operation” (i.e., anticipating imminent failure). Examples are instrumenting operation of an expensive machine and monitoring a patient in intensive care. An \( N-M-N \) net may be trained to respond with output approximately equal to its input for correct-operation data. This net then can be used to constantly monitor an operating system, and, in case the net’s output fails to match its input with some specified tolerance, an alarm can sound.

3 Single dots

The traditional “single dot” problem refers to the training of an \( N-M-N \) feed-forward neural network that is (nearly) the identity function on a set of \( N \) exemplars, the standard unit vectors in \( \mathbb{R}^N \). Such a network operates by mapping the training exemplars \( e_1, \ldots, e_N \) into \( \mathbb{R}^M \) in such a way that \( M \) \(-1\)-dimensional hyperplanes each separates one of these points in \( \mathbb{R}^M \) from the other \( N-1 \) points. Surprisingly, this is possible for \( M \) as low as 2. The images of the \( N \) training exemplars can be roughly organized onto points on the boundary of a circle in \( \mathbb{R}^2 \), and \( N \) lines (\( M-1\)-dimensional hyperplanes) can each separate one of these points from the other \( N-1 \). Figure 1 illustrates how this is done for the \( 8-2-8 \) network; the \( N-2-N \) case is an obvious generalization.

It is possible for a \( N-2-N \) network to achieve this task for any positive \( N \), although it is very difficult for the backprop algorithm to discover such a network for \( N > 14 \) (see [1]).

4 Double dots

Fig. 2 shows five of the forty training exemplars, as binary row-vectors, for the compression problem we call “double dots.” Each exemplar consists of two 1’s and fourteen 0’s, with the 1’s separated by a gap of at least five 0’s (we consider the positions of the 0’s and 1’s to be integers “mod 16,” so the gaps of 0’s are on both sides of the 1’s).

Fig. 3 shows how those forty training exemplar can be organized to form a Mobius strip. This Mobius strip is a graph, \( G = (V_G, E_G) \), where the vertex set \( V_G \) is in one-to-one correspondence with the set of forty
Figure 1: A possible organization of images of eight training exemplars, $\sigma \mathbf{W} \epsilon_k$, for $k = 1, \ldots, 8$, in $\mathbb{R}^2$. The dotted straight line shows how a single hyperplane separates one point from the others.

exemplars. Two vertices are adjacent if they agree in one of their two values and disagree by exactly one in the other value. That is,

$$E_G = \{ \{p, q\}, \{p, q + 1\} \}$$

These edges connect nearest neighbors in the sense described in the discussion on distances in the Introduction.

A 16–M–16 feed-forward neural network that is (nearly) the identity function on this set of forty exemplars operates by mapping the exemplars into $\mathbb{R}^M$ in such a way that sixteen $M-1$-dimensional hyperplanes each separates the five points corresponding to the value $k$ from the other points, for each $k$, $0 \leq k \leq 15$. Precisely, denote by $\mathbf{y}_{jk}$ the image in $\mathbb{R}^M$ corresponding to the exemplar $\mathbf{e}_j + \mathbf{e}_k$; then for every $j$, there must be a linear function $F_j$ on $\mathbb{R}^M$ such that

$$F_j(\mathbf{y}_{pq}) > 0 \quad \text{iff} \quad p = j \quad \text{or} \quad q = j$$

(We use the bipolar form for convenience.) For such a neural network to exist, we must be able to embed the Mobius strip in $\mathbb{R}^M$ and have the sixteen linear functions.

A Mobius strip can not be embedded in the plane $\mathbb{R}^2$. It can be embedded in $\mathbb{R}^3$, but in a way that does not allow the sixteen cutting planes. Consequently, we have devoted our search to the case $N = 4$–N. (A Mobius strip can be embedded in three-sphere, $S^3$, the set of unit vectors in $\mathbb{R}^4$.) Indeed, such networks do exist—we have trained several and hand-constructed others. We have not proved that the $N$–4–N encoder network for these problems exists in general.

$$
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
$$

Figure 2: Five of the forty training exemplars for a 16–4–16 encoder network.

5 Training wheels & morphing

The one-dot problem is difficult for $N > 14$, but there are several related problems that are considerably easier. One of these problems is illustrated in Fig. 4. Instead of trying to solve the 8–2–8 network for the training exemplar set

$$
\begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
$$

we “blur” the dot, and use the following training set:

$$
\begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
$$

The more difficult training set requires that the $N$ exemplars become organized in $\mathbb{R}^2$ in a roughly circular manner, as shown in Fig. 1—the image of each exemplar must be separable from the others by a straight line.

The modified problem, using blurs of length three rather than spikes of length one, is easier to learn for two reasons, and harder for one reason (see Fig. 4). It is easier, because lines that separate three points from the other five are easier to draw than lines that separate just one point (see Fig. 1); and the points do not need to be organized as corners of a convex body, but can sketch out a non-convex body (see Fig. 4). The blurred problem may be harder, because
the points must be circularly ordered (either clockwise or counter clockwise) in $\mathbb{R}^2$; the original problem allows any permutation of the points around the circle.

We initiate training, as usual, by setting the network’s synaptic weights to small pseudo-random values. The training set, then, originally consists of one of the easy, blurry versions of the problem we want to solve. As soon as the network learns this easy version of the training set, we “remove one training wheel” by reducing the size of the blur block by one. This is “morphing,” an image processing trick in which one image is gradually changed to another—such as a person becoming a werewolf.

6 Results

6.1 Single dots

In the experiments reported in this section we used the “bipolar” representation of the data: +1 and −1 instead of 1 and 0. We considered a pattern “learned” when the sign of all the outputs was correct. We “removed a set of training wheels” or considered training complete when all patterns were learned.

Table 1 shows how many epochs it took us to train $N$−2−$N$ networks “using training wheels” starting with a block size of $B = [N/4]$ 1’s and working down to two 1’s. For unknown reasons, it appears better to start with $B = [N/4]$ than with the larger $B = [N/2]$.

Compare the “training wheels” method with the epoch counts given in Table 2 which shows how long it takes without training wheels to train the $N$−2−$N$ networks for $9 \leq N \leq 30$ with blocks of size $B$, $[N/4] \geq B \geq 2$. The final number in each row is the number of epochs it took to train the $N$−2−$N$ network to solve the problem with two 1’s in the field. For $N = 16$ and $N = 18$, the training wheels method cut the training epochs by a factor of more than ten. Many other cases failed to produce trained networks within 70,000 epochs, the limit of our patience for this experiment.

6.2 Double dots

Table 3 shows the results of several experiments training $N$−4−$N$ networks to reproduce input signals with two subfields of $B$ 1’s in a field of size $N$. $G$ indicates a “gap” size—we required at least $G$ values between the leftmost 1’s in each block (measured both clockwise
<table>
<thead>
<tr>
<th>$N$</th>
<th>$B_0$</th>
<th>$\text{Epochs to learn problem with } B = B_0 = \lfloor N/4 \rfloor, \ldots, 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3</td>
<td>239 255</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>229 243 301</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>235 249 288</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>239 255 279</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>256 269 286</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>383 472 507 630</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>345 643 694 800</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>406 990 1038 1151</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>389 1154 1204 1461</td>
</tr>
<tr>
<td>21</td>
<td>6</td>
<td>588 970 1987 2071 2179</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
<td>473 952 1684 1732 2241</td>
</tr>
<tr>
<td>23</td>
<td>6</td>
<td>520 1817 2552 2627 3508</td>
</tr>
<tr>
<td>24</td>
<td>6</td>
<td>1015 2456 2651 2707 11509</td>
</tr>
<tr>
<td>25</td>
<td>7</td>
<td>686 1001 11463 12251 12485 13097</td>
</tr>
<tr>
<td>26</td>
<td>7</td>
<td>607 1206 9582 10035 10142 10719</td>
</tr>
<tr>
<td>27</td>
<td>7</td>
<td>1002 1628 5284 6989 7079 65437</td>
</tr>
<tr>
<td>28</td>
<td>7</td>
<td>810 2082 4421 4713 4773 55114</td>
</tr>
<tr>
<td>29</td>
<td>8</td>
<td>844 1690 3799 5534 34385 34896 37305</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>1226 1834 11762 12250 12283 12296 158019</td>
</tr>
<tr>
<td>31</td>
<td>8</td>
<td>1735 2327 90793 92669 93618 94675 97632</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>2050 3356 7476 7993 8195 8216 239090</td>
</tr>
</tbody>
</table>

Table 1: The number of epochs to solve the encoder problem for $N$ points and a block of 1’s beginning with $B = \lfloor N/4 \rfloor$ 1’s (the leftmost entry in each row), working down to two 1’s, “removing one training wheel” at a time.

and counter-clockwise; i.e., regarding field positions as indexed $\text{mod } N$).

7 Implementation

Our base algorithm was standard backwards error propagation (“backprop”), such as that given in [3]. The programs were all written in the programming language J.

7.1 The language “J”

J is a recent dialect of APL developed by Iverson [4]. In the APL tradition, J allows and encourages users to work in terms of arrays and vectors. Consequently, J programs are remarkably concise; algorithms are visible, and details are hidden. J programs may be 10% as large as C or Fortran programs; J’s operations on arrays generally dispense with subscripts and loops.

We work with a feed forward neural network with an input vector, $X$, and hidden layer vector, $Y$, and an output vector, $Z$. $W$ is the matrix of synaptic weights connecting $X$ and $Y$, and $V$ is the matrix of synaptic weights connecting $Y$ and $Z$. The nonlinear squashing function is $\sigma$. The neural net transfer function

$$Z = \sigma(V \times \sigma(W \times X))$$

is easily expressed in J as

$$Z = \cdot \text{sig} V \cdot \text{ip} Y = \cdot \text{sig} W \cdot \text{ip} X \quad (1)$$

The J assignment operator is “=.,” and “ip” is the name we have assigned to the matrix multiplication operation.

Operations take one or two operands and are written as prefix (such as our squashing function sig) or infix (such as ip and =.), respectively. Infix operations all are right associative, so Eq. 1 has the following understood parentheses:

$$Z = \cdot \text{sig}(V\cdot\text{ip}(Y = (\text{sig}(W\cdot\text{ip} X))))$$

In our backprop implementation, $X$ is the matrix whose columns are the input values of the training exemplars. For the problem in which the input exemplars are the standard unit vectors, $e_k$, $1 \leq k \leq N$, $X$ is the $N \times N$ identity matrix (or the bipolar representation, with 0’s replaced by -1’s).
<table>
<thead>
<tr>
<th>( N )</th>
<th>( B_0 )</th>
<th>Epochs to learn problem with ( B = B_0 = \lfloor N/4 \rfloor, \ldots, 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3</td>
<td>239 406</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>229 153 425</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>235 147 &gt;70000</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>239 151 &gt;70000</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>256 211 2931</td>
</tr>
<tr>
<td>17</td>
<td>5</td>
<td>383 199 467 &gt;70000</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>345 518 514 9656</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>406 1063 381 &gt;70000</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>389 1272 802 &gt;70000</td>
</tr>
<tr>
<td>21</td>
<td>6</td>
<td>588 975 2401 797 &gt;70000</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
<td>473 898 2771 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>23</td>
<td>6</td>
<td>520 1846 2719 2396 &gt;70000</td>
</tr>
<tr>
<td>24</td>
<td>6</td>
<td>1015 2904 1120 14084 &gt;70000</td>
</tr>
<tr>
<td>25</td>
<td>7</td>
<td>686 881 11922 4588 27910 &gt;70000</td>
</tr>
<tr>
<td>26</td>
<td>7</td>
<td>607 1110 6973 &gt;70000 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>27</td>
<td>7</td>
<td>1002 1781 5660 20297 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>28</td>
<td>7</td>
<td>810 2843 3190 3807 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>29</td>
<td>8</td>
<td>844 1632 11113 4389 &gt;70000 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>30</td>
<td>8</td>
<td>1226 2023 54151 7342 &gt;70000 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>31</td>
<td>8</td>
<td>1735 2426 &gt;70000 9983 39576 &gt;70000 &gt;70000</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
<td>2050 3515 7120 12084 &gt;70000 &gt;70000 &gt;70000</td>
</tr>
</tbody>
</table>

Table 2: The number of epochs to solve the encoder problem for \( N \) points and a block of 1’s beginning with \( B = \lfloor N/4 \rfloor \) 1’s (the leftmost entry in each row), down to 2 1’s. Each problem was started by random weights initialization—“training wheels” were not used.

### 7.2 The training algorithm

The backward error propagation step is expressed in \( J \) as

\[
Dy = \left(t \cdot Y \right) \cdot \left(| V \right) ip Dz = \left( X - Z \right) \cdot t \cdot Z \tag{2}
\]
i.e.,

\[
\delta y = \tau(Y) \cdot V^T \times \left( X - Z \right) \cdot \tau(Z) / \delta z
\]

(We use “\( \cdot \)” to denote component-by-component multiplication of vectors and matrices and “\( / \)” to denote ordinary matrix multiplication.) The variables \( Dy = \delta y \) and \( Dz = \delta z \) are the errors associated with the hidden layer and the output layer, respectively. In Eq. 2, “\( \cdot \)” is ordinary, component-wise multiplication; “\( / \)” is matrix transpose; and “\( t \cdot y = \tau(y) \)” is \((y-1) \cdot (y+1)\), corresponding to the derivative of the squashing function \( \text{sig} = \tan h \) at \( y \), where \( \text{sig}(a)=y \).

Because of \( J \)’s ease of using arrays, we implemented the learning algorithm using the batch update rule rather than on-line learning. That is, we presented the exemplars to the net—all at once—and then applied the cumulative error adjustments. For example, the change, \( \Delta W \), to the first synaptic weights layer is

\[
\Delta W = \mu \Delta W + \epsilon \delta y \times X^T
\]
i.e.,

\[
\Delta W = \mu \Delta W + \epsilon \delta y \times X^T
\]

\( \mu = \mu \) is the momentum rate (we used 0.5); \( \Delta W \) stands for “old \( \Delta W \)” and is saved each epoch for use the next epoch (as shown in Eq. 3).

The learning rate, \( \epsilon = \epsilon \), is set adaptively [5]. For the hardest problem we attempted successfully, we initialized \( \epsilon \) to 2 \(-45\); whenever an update improved the network, we increased \( \epsilon \) by

\[
\epsilon = \epsilon \cdot 0.0000001 + \epsilon \cdot 0.0000001 \tag{4}
\]

In case an update failed to improve the network (we measured network improvement in terms of the sum of the absolute errors; i.e., the \( L_1 \) norm), we decrease \( \epsilon \) by

\[
\epsilon = \epsilon \cdot 0.0000001 + \epsilon \cdot 0.3 \tag{5}
\]

We repeatedly decrease \( \epsilon \) until the update step is sufficiently small to improve the network. The numerical parameters in Eqs. 4 and 5 were determined by trial
and error; the additive constant, 0.0000001, keeps eps from decaying to zero. Whenever we decrease eps, we also reset the momentum terms ODW and ODV to zero.

8 Acknowledgments

Many of the training wheels experiments were first investigated by my graduate students: David Cox [2], Kathy Rainero [7], and Sanjay Raghavendra [6]. Jeff Pink [5] also experimented with and showed me the adaptive learning rate technique.

References


Proceedings of the 1988 Connectionist Models Summer School,


