A Fast C++ Implementation of Neural Network Backpropagation Training Algorithm: Application to Bayesian Optimal Image Demosaicing

Yi-Qing Wang, Nicolas Limare

CMLA, ENS Cachan, France (yiqing.wang, nicolas.limare@cmla.ens-cachan.fr)

Abstract

Recent years have seen a surge of interest in multilayer neural networks fueled by their successful applications in numerous image processing and computer vision tasks. In this article, we describe a C++ implementation of the stochastic gradient descent to train a multilayer neural network, where a fast and accurate acceleration of tanh(.) is achieved with linear interpolation. As an example of application, we present a neural network able to deliver state-of-the-art performance in image demosaicing.

Source Code

The source code and an online demo are accessible at the IPOL web page of this article.

Keywords: feedforward neural networks; code optimisation; image demosaicing

1 Introduction

Multilayer neural networks have recently ignited much interest in the signal processing community thanks to their broad range of applications [2]. For example, it has been shown that a multilayer neural network, when endowed with sufficient capacity, could be trained to produce state-of-the-art performance in denoising [3,11] as well as in demosaicing [10].

This superior performance does come at a price. Because of its non-convex structure, no efficient algorithm exists to solve a typical empirical risk minimization program involving a multilayer neural network as the regression function. As a result, stochastic gradient descent becomes the de facto tool of choice although it implies high computational cost on a large training dataset.
2 Algorithm

A feedforward neural network is a succession of hidden layers followed by an application-dependent decoder

\[ f(\cdot, \theta) = \varnothing \circ h_n \circ \cdots \circ h_1(\cdot), \quad n \geq 1 \quad (1) \]

with

\[ a_{l+1} = h_l(a_l) = a(W_l a_l + b_l), \quad \forall 1 \leq l \leq n \quad (2) \]

and

\[ \varnothing(a_{n+1}) = W_{n+1} a_{n+1} + b_{n+1}, \]

in case of a linear decoder. The parameter vector \( \theta \) comprises the connection weights \( W_l \) and biases \( b_l \) indexed by their respective layer \( l \). The non-linear activation function \( a(\cdot) \) is understood to apply to its input vector element-wise.

Mathematically speaking, neural networks can approximate any continuous function on a compact set to any desired degree of accuracy [7], hence their applications in regression tasks

\[
\theta^* = \arg\min_{\theta} \mathbb{E}[\|f(\tilde{x}, \theta) - x\|^2_2] = \arg\min_{\theta} \mathbb{E}[\|f(\tilde{x}, \theta) - \mathbb{E}[x|\tilde{x}]\|^2_2],
\]

where \((\tilde{x}, x) \in \mathbb{R}^d \times \mathbb{R}^d\) denotes a random supervised pair of observation and teaching signal, whose behavior is governed by some probability law \( \mathbb{P} \) serving to define the expectation \( \mathbb{E} \) and the conditional expectation \( \mathbb{E}[x|\tilde{x}] \). The rightmost equality of (3) follows directly from the conditional expectation’s definition. In practice, though we can sample from it, the probability \( \mathbb{P} \) either does not have a closed form expression or is formed by such a large number of examples that an exact expectation based on it cannot be computed in real time. Moreover, the function \( \theta \mapsto \|f(\tilde{x}, \theta) - x\|^2_2 \) is not convex, leaving us with little choice but to substitute the expectation above with an empirical surrogate, that is, the average mean square error (MSE)

\[
\frac{1}{m} \sum_{i=1}^{m} \|f(\tilde{x}_i, \theta) - x_i\|^2_2
\]

with a relatively small batch size and to rely on the method of steepest descent to conduct the minimization. Here \((\tilde{x}_i, x_i)_{i \geq 1}\) denotes a sequence of identically distributed and independent realisations from \( \mathbb{P} \). Though such a procedure cannot help us reach \( \theta^* \) in general, it does quite well in practice, when the trained neural network is tested on another dataset distinct from the one used in its training but sharing a similar example distribution.

To derive the derivative of (4) with respect to \( \theta \)

\[
\frac{\partial}{\partial \theta} \frac{1}{m} \sum_{i=1}^{m} \|f(\tilde{x}_i, \theta) - x_i\|^2_2 = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta} \|f(\tilde{x}_i, \theta) - x_i\|^2_2
\]

we focus on the generic term

\[
\frac{\partial}{\partial \theta} \|f(\tilde{x}, \theta) - x\|^2_2.
\]
Without loss of generality, in what follows we assume that the neural network has a linear decoder and all of its non-linear units are implemented by $a(\cdot) := \tanh(\cdot)$ whose derivative enjoys a particular property
\[
\tanh'(\cdot) = 1 - \tanh^2(\cdot).
\]
Let us also mention that other popular activation functions could also be used such as the sigmoid \[9\] and the rectifier \[6\].

As a consequence of the chain rule, one can compute the derivative \((5)\) one layer at a time, hence the term backpropagation (see Algorithm 1). To continue using the previously introduced notation, we identify $a_1$ with $\hat{x}$. In addition, let us denote the real-valued objective
\[
J := \frac{1}{2} \| f(a_1, \theta) - x \|^2_2
\]
to further simplify the notation. Substituting $f(a_1, \theta)$ by \((1)\), we obtain the first part of the derivative
\[
\frac{\partial J}{\partial W_{n+1}} = (W_{n+1}a_{n+1} + b_{n+1} - x) a^T_{n+1}
\]
\[
\frac{\partial J}{\partial b_{n+1}} = W_{n+1}a_{n+1} + b_{n+1} - x,
\]
as well as
\[
\frac{\partial J}{\partial a_{n+1}} = W^T_{n+1}(W_{n+1}a_{n+1} + b_{n+1} - x),
\]
which, when combined with the observation \((6)\) and the chain rule, leads to
\[
\frac{\partial J}{\partial W^i_{nj}} = \frac{\partial J}{\partial a^i_{n+1}} \frac{\partial a^i_{n+1}}{\partial W^i_{nj}} = \frac{\partial J}{\partial a^i_{n+1}} [1 - (a^i_{n+1})^2] a^i_{n},
\]
where $W^i_{nj}$ and $a^i_{n+1}$ denote respectively the element at the $i$th row and $j$th column of $W_n$ and the $i$th row of $a_{n+1}$. In matrix form, we thus have
\[
\frac{\partial J}{\partial W_n} = \left[ \frac{\partial J}{\partial a_{n+1}} \odot (1 - a_{n+1} \odot a_{n+1}) \right] a^T_n,
\]
with $\odot$ meaning the element-wise product and $1$ a matrix of the same dimension as $a_{n+1}$ but filled with ones. Similarly, it follows
\[
\frac{\partial J}{\partial b_n} = \frac{\partial J}{\partial a_{n+1}} \odot (1 - a_{n+1} \odot a_{n+1})
\]
\[
\frac{\partial J}{\partial a_n} = W^T_n \left[ \frac{\partial J}{\partial a_{n+1}} \odot (1 - a_{n+1} \odot a_{n+1}) \right],
\]
hence the recursive relationship that lies at the heart of the algorithm which we now detail.

With backpropagation, stochastic gradient descent is straightforward (see Algorithm 2): following \[5\], we first initialize a multilayer neural network’s parameter vector $\theta_0$ with the bias $b_l$ set to zero and the entries of $W_l$ sampled independently from the uniform law on the interval
\[
(-\sqrt{\frac{6}{n_l + n_{l+1}}}, \sqrt{\frac{6}{n_l + n_{l+1}}})
\]
in which $n_l$ denotes the number of hidden units at its $l$th layer. Next, a few supervised examples are drawn from the training dataset to calculate the stochastic gradient, which we use to update the neural network. This step repeats until a prefixed number of training cycles are completed. Every once in a while, we test the trained neural network on a distinct and fixed validation dataset to assess its generalization error. The version returned at the end of the training is the one which realises the smallest estimated generalization error.
Algorithm 1 Backpropagation

1: Input: \( m \) pairs of i.i.d. supervised examples \((\tilde{x}_i, x_i)_{1 \leq i \leq m}\).
2: Parameter: connection weights and biases of the neural network \( \theta := (W_l, b_l)_{1 \leq l \leq n+1} \).
3: Output: derivative of the objective

\[
J_m = \frac{1}{2m} \sum_{i=1}^{m} \| f(\tilde{x}_i, \theta) - x_i \|_2^2
\]

with respect to \( \theta \).
4: Feedforward to get a \( m \)-column matrix \( D_{n+1} \) whose \( i \)th column represents \( f(\tilde{x}_i, \theta) - x_i \) as well as the \( m \)-column activation matrices \((A_l)_{1 \leq l \leq n+1}\) whose \( i \)th column is formed by the values computed at layer \( l \) from the input \( \tilde{x}_i \) according to (2).
5: for \( l = n + 1 \) to 1 do
6: Compute the derivatives

\[
\begin{align*}
\frac{\partial J_m}{\partial b_l} &= \frac{1}{m} D_l \mathbf{1}_{m \times 1} \\
\frac{\partial J_m}{\partial W_l} &= \frac{1}{m} D_l A_l^T \\
\frac{\partial J_m}{\partial A_l} &= W_l^T D_l
\end{align*}
\]

where \( \mathbf{1}_{m \times 1} \) is the column vector of \( m \) ones.
7: Set \( \mathbf{1}_l \) to be a matrix filled with ones and of the same dimension as \( A_l \)

\[
D_{l-1} = \frac{\partial J_m}{\partial A_l} \odot (\mathbf{1}_l - A_l \odot A_l).
\]
8: end for
9: Order \((\frac{\partial J_m}{\partial W_l}, \frac{\partial J_m}{\partial b_l})_{1 \leq l \leq n+1}\) the same way as \((W_l, b_l)_{1 \leq l \leq n+1}\) in \( \theta \) to form \( \frac{\partial J_m}{\partial \theta} \).

3 A Faster \( \tanh(\cdot) \) Implementation

Applying \( \tanh(\cdot) \) at a neural network’s hidden nodes is the most time consuming operation in backpropagation, an observation which led us to implement a faster version than the one in the standard C/C++ numerics library but with equal accuracy. The key is to observe that the range of \( \tanh(\cdot) \) is bounded in the interval \((-1, 1)\) and floating point numbers have limited precision.

IEEE Standard 754 specifies that a single precision floating point number has 23 fraction bits, meaning that the largest number strictly smaller than 1 that it can represent is \( 1 - 2^{-24} \). In other words, it incurs at most an error equal to \( 2^{-24} \) if we return 1 for all \( x \) such that

\[
\tanh(x) > 1 - 2^{-24},
\]

which, thanks to the equality

\[
\tanh(x) = 1 - \frac{2}{1 + e^{2x}},
\]

implies

\[
1 + e^{2x} > 2^{25} \iff x > \frac{\ln(2^{25} - 1)}{2} \approx 12.5 \ln(2).
\]
Algorithm 2 Stochastic gradient descent

1: Input: initial parameter $\theta_0$, a training dataset and a distinct validation set $(\tilde{x}_i, x_i)_{1 \leq i \leq N_1}$.
2: Parameter: learning rate $\alpha$, iteration number $T$, observation interval $S$, batch size $N_2$.
3: Output: a trained neural network $\theta_*$. 
4: Evaluate the current error on the validation set 
   $$\epsilon := \frac{1}{N_1} \sum_{i=1}^{N_1} ||f(\tilde{x}_i, \theta) - x_i||_2^2.$$ 
5: for $t = 0$ to $T - 1$ do
6: Draw $N_2$ pairs of supervised examples from the training dataset.
7: Calculate the stochastic gradient with Algorithm 1: $\frac{\partial}{\partial \theta} J_{N_2}(\theta_t)$.
8: Update the neural network 
   $$\theta_{t+1} = \theta_t - \alpha \frac{\partial}{\partial \theta} J_{N_2}(\theta_t).$$ 
   Note that one may also want to set the learning rate in a layer-wise fashion, in which case $\alpha$ is a positive diagonal matrix.
9: if $\text{mod}(t + 1, S) = 0$ then
10: Evaluate the current error on the validation set 
   $$e := \frac{1}{N_1} \sum_{i=1}^{N_1} ||f(\tilde{x}_i, \theta_{t+1}) - x_i||_2^2.$$ 
11: if $\epsilon > e$ then 
12: $\epsilon = e$ and $\theta_* = \theta_{t+1}$ 
13: end if
14: end if
15: end for

Since $\tanh(\cdot)$ is odd, we may only focus on its domain ranging from 0 to $12.5 \ln(2)$. However, to avoid the cost of sign determination, we choose to symmetrize the domain and use a linear interpolation to minimise computing cost. Hence it is useful to recall a classical polynomial interpolation error bound. To simplify notations, we only concern ourselves with the linear interpolation though the result can be readily extended to higher degree polynomials.

Let $f(\cdot)$ be a twice differentiable function defined on $[a, b]$ with $a < b$. Let $p(\cdot)$ be its linear interpolation going through $(a, f(a))$ and $(b, f(b))$ and $W(x) = (x - a)(x - b)$. Then for any fixed $x \in (a, b)$, the function

$$g : t \in [a, b] \mapsto f(t) - p(t) - \frac{f(x) - p(x)}{W(x)} W(t)$$

has at least three roots, which, by Rolle’s theorem, implies at least two roots for $g'$ and one root for $g''$. We can therefore denote $c \in (a, b)$ to be $g''$’s root, or $g''(c) = 0$, which leads to the interpolation error bound

$$f''(c) = 2 \frac{f(x) - p(x)}{W(x)} \Rightarrow \sup_{x \in [a, b]} |f(x) - p(x)| \leq \sup_{c \in (a, b)} |f''(c)| \frac{(b - a)^2}{8}.$$  

261
With $|\tanh''(\cdot)|$ bounded by $\frac{4}{3\sqrt{3}}$, we can thus set equally spaced interpolating nodes so as to ensure an interpolation error less than $10^{-8}$ ($2^{-24} \approx 6 \times 10^{-8}$).

With compiler options −02 and −fno-vectorize enabled, our tanh(·) runs on average 5 to 8 times faster than the standard implementation.

4 Application

Here we show an application of the neural network in image demosaicing. A typical digital camera has a Bayer color filter array (CFA) [1] placed in front of its sensor so that at each pixel location, either green, or blue, or red light intensity is recorded. To recover a color image, one needs to estimate missing color intensities from such a mosaiced image.

Neural networks find a perfect setting in this problem because the four building blocks (see Figure 4) of the Bayer CFA are either one or two 90 degrees rotation away from one another. Since it is reasonable to postulate that natural image patch statistics are roughly invariant with respect to a 90 degrees rotation, a neural network trained to demosaic one pattern can also be used to do the same for the other three patterns. This forms the basis of our approach (see Algorithm 4).

![Figure 1: Four basic Bayer patterns (a) RGGB (b) GRBG (c) BGGR (d) GBRG](image)

Since the fundamental object that a neural network attempts to approximate is the conditional expectation under its training distribution, it is useful to bias the data in such a way as to raise the statistical importance of its high frequency patterns at the expense of their slow-varying counterparts. Without resorting to a refined statistical modeling to define what the high frequency patterns are, we simply whiten the input (see Algorithm 3). As the principal component analysis (PCA) implicitly assumes that data behaves like a Gaussian vector, its concept of high frequency patterns may not coincide with ours. However, experiments showed that whitening is conducive to good training.

**Algorithm 3 Whitening**

1. **Input:** $n$ vectors $p_j \in \mathbb{R}^d$.
2. **Output:** the whitening operator $\mathcal{W}$.
3. **Parameter:** smoothing factor $\sigma_Z = 0.01$.
4. Center the inputs $\tilde{p}_j = (p_j - \bar{p})/51$ with $\bar{p} = \frac{1}{n} \sum_{j=1}^{n} p_j$.
5. Run the principal component analysis (PCA) on $(\tilde{p}_j)_{1 \leq j \leq n}$ to obtain their eigenvectors and eigenvalues $(\phi_i, \lambda_i)_{1 \leq i \leq d}$.
6. $\mathcal{W}(p) = 51^{-1} \sum_{i=1}^{d} \sum_{j=1}^{n} \lambda_i^{1/2} \langle \phi_i, p - \bar{p} \rangle \phi_i$ with $\langle \cdot, \cdot \rangle$ a standard scalar product.

For an illustration, we trained a neural network which, observing a 6-by-6 Bayer RGGB mosaiced patch, attempted to predict the 8 missing color intensities in its central 2-by-2 area. The neural network has 2 hidden layers, each having 108 non-linear encoding units. Its basic learning rate was set to 0.1. Following [3], the layer specific learning rate was the basic learning rate divided by
A Fast C++ Implementation of Neural Network Backpropagation Training Algorithm: Application to Bayesian Optimal Image Demosaicing

Algorithm 4 Neural network demosaicing
1: Input: a Bayer color filter array masked image.
2: Output: a demosaiced color image.
3: Parameter: a neural network trained to demosaic $\kappa$-by-$\kappa$ RGGB filtered patches and its associated whitening operator $W$.
4: Decompose the mosaiced image into overlapping $\kappa$-by-$\kappa$ patches with spatial offset equal to 1.
5: Rotate these patches, if necessary, so that their Bayer CFA patterns become RGGB.
6: Whiten the patches and run them through the neural network to form estimated color patches.
7: Rotate patches again to have the previously rotated patches back to their initial positions.
8: Aggregate patches with equal weights to form the final image.

The number of input units at that layer. To compute the stochastic gradient $10^3$ random supervised examples were drawn at a time from an image database composed of 400 randomly chosen images from the Berkeley Segmentation Dataset (BSD500) and a selection of 781 images from INRIA Holidays Dataset [8]. During its $2 \times 10^8$ round training, the neural network was tested every $S = 10^4$ rounds on a validation dataset of $10^4$ examples from the remaining 100 images of BSD500. As shown in Figure 2, the neural network gained the most, as measured in validation MSE, from its initial $10^6$ rounds of training. But later rounds did improve the neural network as well.

![NN Learning Curve](image)

Figure 2: The neural network’s validation RMSE (root mean square error, that is, the square root of MSE) was recorded every $10^4$ backpropagation iterations during its training. The figure’s horizontal axis is plotted in the logarithmic scale.

Most of the images selected from INRIA Holidays Dataset were taken outdoors in good lighting conditions and have occasionally vibrant color contrast. Having been demosaiced one way or another, they were downsampled by 4 after being convolved with a Gaussian kernel of standard deviation equal to 1.7 for quality enhancement. They are available at http://dev.ipol.im/~yiqing/inria.tar.gz.

We compare our trained neural network with DLMMSE [12], which has a known bias by design [4] to do very well on the gray-looking Kodak PhotoCD benchmark images\(^2\) at the expense of more colorful ones like those in the McMaster dataset\(^3\). The results are reported in terms of RMSE in Table 1. Overall our neural network outperforms DLMMSE on both datasets (see Figure 4).

\(^2\)http://r0k.us/graphics/kodak/
\(^3\)http://www4.comp.polyu.edu.hk/~cslzhang/CDM_Dataset.htm
Figure 3: Demosaicing examples. Images (a) and (g) have been cropped from mcm01 and kodim19 respectively (see Figure 4) and image (d) depicts the right wing of the butterfly shown on this article’s demo web page. Though it fails to restore the fence as well as DLMMSE did, in overall RMSE and visual terms, the neural network is superior.

5 Image Credits

McMaster Dataset, image 1, http://www4.comp.polyu.edu.hk/~cslzhang/CDM_Dataset.htm

Kodak Image Suite, image 19, http://r0k.us/graphics/kodak/
A Fast C++ Implementation of Neural Network Backpropagation Training Algorithm: Application to Bayesian Optimal Image Demosaicing

Figure 4: The two benchmark datasets: Kodak (above) and McMaster (below).

Thomas Barnes, U.S. Fish and Wildlife Service

References


Table 1: Algorithm comparison in RMSE on McMaster (mcmxx) and Kodak (kodimxx). Better results are marked in bold.

<table>
<thead>
<tr>
<th>IMAGEID</th>
<th>NN</th>
<th>DLMMSE</th>
<th>IMAGEID</th>
<th>NN</th>
<th>DLMMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mcm01</td>
<td>9.03</td>
<td>11.84</td>
<td>kodim01</td>
<td>2.97</td>
<td>2.72</td>
</tr>
<tr>
<td>mcm02</td>
<td>4.52</td>
<td>5.46</td>
<td>kodim02</td>
<td>2.38</td>
<td>2.31</td>
</tr>
<tr>
<td>mcm03</td>
<td>4.98</td>
<td>6.16</td>
<td>kodim03</td>
<td>1.67</td>
<td>1.89</td>
</tr>
<tr>
<td>mcm04</td>
<td>3.57</td>
<td>5.23</td>
<td>kodim04</td>
<td>2.23</td>
<td>2.47</td>
</tr>
<tr>
<td>mcm05</td>
<td>4.84</td>
<td>7.37</td>
<td>kodim05</td>
<td>2.83</td>
<td>3.29</td>
</tr>
<tr>
<td>mcm06</td>
<td>2.85</td>
<td>5.57</td>
<td>kodim06</td>
<td>2.44</td>
<td>2.36</td>
</tr>
<tr>
<td>mcm07</td>
<td>3.03</td>
<td>2.82</td>
<td>kodim07</td>
<td>1.63</td>
<td>2.03</td>
</tr>
<tr>
<td>mcm08</td>
<td>2.65</td>
<td>3.56</td>
<td>kodim08</td>
<td>3.84</td>
<td>3.76</td>
</tr>
<tr>
<td>mcm09</td>
<td>3.07</td>
<td>5.14</td>
<td>kodim09</td>
<td>1.74</td>
<td>1.78</td>
</tr>
<tr>
<td>mcm10</td>
<td>2.73</td>
<td>4.05</td>
<td>kodim10</td>
<td>1.81</td>
<td>1.92</td>
</tr>
<tr>
<td>mcm11</td>
<td>2.49</td>
<td>3.65</td>
<td>kodim11</td>
<td>2.48</td>
<td>2.46</td>
</tr>
<tr>
<td>mcm12</td>
<td>2.83</td>
<td>3.92</td>
<td>kodim12</td>
<td>1.65</td>
<td>1.69</td>
</tr>
<tr>
<td>mcm13</td>
<td>2.23</td>
<td>3.06</td>
<td>kodim13</td>
<td>4.26</td>
<td>4.25</td>
</tr>
<tr>
<td>mcm14</td>
<td>2.82</td>
<td>3.64</td>
<td>kodim14</td>
<td>2.99</td>
<td>3.76</td>
</tr>
<tr>
<td>mcm15</td>
<td>2.67</td>
<td>3.62</td>
<td>kodim15</td>
<td>2.36</td>
<td>2.64</td>
</tr>
<tr>
<td>mcm16</td>
<td>4.94</td>
<td>8.10</td>
<td>kodim16</td>
<td>1.71</td>
<td>1.59</td>
</tr>
<tr>
<td>mcm17</td>
<td>5.12</td>
<td>9.22</td>
<td>kodim17</td>
<td>2.11</td>
<td>2.07</td>
</tr>
<tr>
<td>mcm18</td>
<td>4.18</td>
<td>5.28</td>
<td>kodim18</td>
<td>3.29</td>
<td>3.37</td>
</tr>
<tr>
<td>average</td>
<td>3.81</td>
<td>5.43</td>
<td>kodim19</td>
<td>2.33</td>
<td>2.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kodim20</td>
<td>2.07</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kodim21</td>
<td>2.66</td>
<td>2.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kodim22</td>
<td>2.86</td>
<td>3.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kodim23</td>
<td>1.56</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>kodim24</td>
<td>4.17</td>
<td>4.25</td>
</tr>
<tr>
<td>average</td>
<td>2.50</td>
<td>2.61</td>
<td>kodim25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


