Stochastic Gradient Descent Learning and the Backpropagation Algorithm

Oliver K. Ernst
Department of Physics
University of California, San Diego
La Jolla, CA 92093-0354
oernst@ucsd.edu

Abstract

Many learning rules minimize an error or energy function, both in supervised and unsupervised learning. We review common learning rules and their relation to gradient and stochastic gradient descent methods. Recent work generalizes the mean square error rule for supervised learning to an Ising-like energy function. For a specific set of parameters, the energy and error landscapes are compared, and convergence behavior is examined in the context of single linear neuron experiments. We discuss the physical interpretation of the energy function, and the limitations of this description. The backpropgation algorithm is modified to accommodate the energy function, and numerical simulations demonstrate that the learning rule captures the distribution of the network's inputs.

1 Gradient and Stochastic Gradient Descent

A learning rule is an algorithm for updating the weights of a network in order to achieve a particular goal. One particular common goal is to minimize a *error function* associated with the network, also referred to as an objective or cost function.

Let $Q(\mathbf{w})$ denote the error function associated with a network with connection weights $\mathbf{w}$. A first approach to minimize the error is the method of gradient descent, where at each iteration a step is taken in the direction corresponding to $-\nabla Q(\mathbf{w})$. The learning rule that follows is

$$\mathbf{w} := \mathbf{w} - \eta \nabla Q(\mathbf{w}), \tag{1}$$

where η is a constant parameter called the *learning rate*. Consider being given a set of n observations $\{\mathbf{x}_i, \mathbf{t}_i\}$, where $i = 1, 2...n$. Each observation consists of inputs $\mathbf{x}$ into the network and corresponding outputs $\mathbf{t}$. A particular case of gradient descent may be formulated for an error function of the form

$$Q(\mathbf{w}) = \sum_{i=1}^{n} Q_i(\mathbf{w}), \tag{2}$$

where $Q_i(\mathbf{w})$ is the error corresponding to the i-th observation. In this case, the learning rule becomes

$$\mathbf{w} := \mathbf{w} - \eta \sum_{i=1}^{n} \nabla Q_i(\mathbf{w}). \tag{3}$$

For most error functions, this standard (or batch) gradient descent method is guaranteed to converge to a local minimum, provided one exists and the learning rate is small. Furthermore, it does so along the fastest route, taking steps at every iteration perpendicular to the contour lines.

A key challenge is computational power - evaluating the sum in 3 may be computationally intensive, particularly for large training sets. A more economical method for minimizing Q is that of *stochastic*

IPR2025-00194
Digital Global Systems Exhibit 1013

(or on-line) gradient descent. At each iteration, the gradient of $Q(\mathbf{w})$ is approximated by the gradient of the single observation's associated error, $Q_i(\mathbf{w})$. The learning rule at each step becomes

$$\mathbf{w} := \mathbf{w} - \eta \nabla Q_i(\mathbf{w}), \tag{4}$$

where we implicitly iterate over the data set, with the possibility of numerous passes. While at each iteration, the algorithm takes a step in a direction perpendicular to the contour lines for that observations error function Q_i, it does not necessarily move perpendicular to the contour lines of Q. Furthermore convergence to the minimum of Q is not guaranteed, further demonstrated in Section 2.1.

1.1 Mean Square Error (MSE)

A logical choice for the error function is the mean square error,

$$E(\mathbf{w}) = \sum_{i=1}^{n} E_i(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} ||\mathbf{t}_i - \mathbf{o}_i(\mathbf{w})||^2, \tag{5}$$

where $\mathbf{t}_i$ is the expected outcome of the network for the i-th observation, and $\mathbf{o}_i(\mathbf{w})$ is the actual output of the network, and $||\mathbf{z}||^2$ denotes the 2-norm [2].

As an illustrative example, consider the case of a network consisting of a single linear neuron. Let $\mathbf{x}_i$ denote the vector of inputs, $\mathbf{w}$ the weights corresponding to each input, and $h_i = \mathbf{w}^\top \mathbf{x}_i$ the weighted sum of neuron's inputs. For simplicity, assume a linear activation function $g(h_i) = h_i$ such that the actual output of the neuron is $y_i = g(h_i) = h_i = \mathbf{w}^\top \mathbf{x}_i$.

Given a set of n inputs and expected outputs $\{\mathbf{x}_i, t_i\}$, the mean square error thus becomes

$$E(\mathbf{w}) = \sum_{i=1}^{n} E_i(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (t_i - \mathbf{w}^\top \mathbf{x}_i)^2. \tag{6}$$

Note that since $t_i, y_i \in \mathcal{R}$ we may drop the absolute value in 5. The learning rule from the stochastic gradient descent method 4 becomes

$$\mathbf{w} := \mathbf{w} + \eta (t_i - \mathbf{w}^\top \mathbf{x}_i) \mathbf{x}_i. \tag{7}$$

2 Energy Cost Function

Begin by expanding the MSE for a single observation of a linear neuron as

$$E_i(\mathbf{w}) = \frac{1}{2}(t_i - \mathbf{w}^\top \mathbf{x}_i)^2 = \frac{1}{2} t_i^2 - t_i \mathbf{w}^\top \mathbf{x}_i + \frac{1}{2} \mathbf{w}^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{w}. \tag{8}$$

Define a similar error function

$$\mathcal{E}_i(\mathbf{w}) = -c t_i \mathbf{w}^\top \mathbf{x}_i - b \mathbf{w}^\top \mathbf{V} \mathbf{w}, \tag{9}$$

where $c, b \in \mathcal{R}$ are constants and $\mathbf{V} \in \mathcal{R}^{n \times n}$. Note that for $c = 1, b = -1/2, \mathbf{V} = \mathbf{x}_i \mathbf{x}_i^\top$, we recover the MSE function, up to a constant term $t_i^2/2$ which does not affect the learning rule [6].

The learning rule associated with this error function is

$$\mathbf{w} := \mathbf{w} + \eta \left[c t_i \mathbf{x}_i + b(\mathbf{V} + \mathbf{V}^\top) \mathbf{w} \right], \tag{10}$$

derived using 4 and the identity $\partial \mathbf{x}^\top \mathbf{B} \mathbf{x} / \partial \mathbf{x} = (\mathbf{B} + \mathbf{B}^\top) \mathbf{x}$.

This work explores a particular choice for the parameters $b, c, \mathbf{V}$, and it's relation to the MSE rule. As before for the MSE, let $c = 1, b = -1/2$. Note that the covariance matrix for general random vector $\mathbf{x}$ with mean $\overline{\mathbf{x}}$ is

$$\Sigma_{\mathbf{xx}} = \langle \mathbf{x}\mathbf{x}^\top \rangle - \overline{\mathbf{x}}\, \overline{\mathbf{x}}^\top. \tag{11}$$

As we iterate over data pairs $\{\mathbf{x}_i, t_i\}$ in stochastic gradient descent, at each step we add a correction term to the weights according to 10. This suggests that if the covariance matrix $\Sigma_{\mathbf{xx}}$ were known,

an intuitive choice for the learning rule is to replace $\mathbf{V} = \mathbf{x}\mathbf{x}^\top$ by $\mathbf{V} = \langle \mathbf{x}\mathbf{x}^\top \rangle = \Sigma_{\mathbf{xx}} + \overline{\mathbf{x}}\overline{\mathbf{x}}^\top$. The error function and learning rule then become

$$\mathcal{E}_i(\mathbf{w}) = -t_i \mathbf{w}^\top \mathbf{x}_i + \frac{1}{2}\mathbf{w}^\top \left(\Sigma_{\mathbf{xx}} + \overline{\mathbf{x}}\overline{\mathbf{x}}^\top\right)\mathbf{w}, \tag{12}$$

$$\mathbf{w} := \mathbf{w} + \eta \left[t_i \mathbf{x}_i - \frac{1}{2}(\Sigma_{\mathbf{xx}} + \Sigma_{\mathbf{xx}}^\top + 2\,\overline{\mathbf{x}}\,\overline{\mathbf{x}}^\top)\mathbf{w} \right]. \tag{13}$$

This similar form has been discussed previously [6] - we briefly depart to clear up several misconceptions. It has been pointed out that this error for resembles the energy of the Ising model in statistical mechanics. This view is limited in several points. In 9, the analogous role of the state of the system is played by $\mathbf{w}$, where generally elements $w_i \in \mathcal{R}$, while in the Ising model the state is described by a vector σ where $\sigma_i \in \{-1, 1\}$. This suggests that an analogy to a continuous spin Ising model may be more appropriate. The constants c, b would play the analogous roles of the magnetic moment and spin-spin coupling strength, respectively, and $t_i \mathbf{x}_i$ in the first term the local magnetic field. The role of the matrix V, however, presents a refutation of this analogy. In the Ising model, the structure of the matrix designates which spins in the lattice couple to another. Several conditions are physically imposed upon the matrix, in particular that it is symmetric (spins cannot couple in only one direction) and that diagonal elements are zero (spins cannot couple with themselves). The choice of $\mathbf{V} = \mathbf{x}_i \mathbf{x}_i^\top$ violates the second condition, and in general the symmetry condition need not be met.

Nonetheless, the remainder of this work maintains the title of *energy function* for 9 solely for clarity, rather than a physical interpretation.

It has been suggested [6] that *Hebb's* and *Oja's* rules may be recovered for particular choices of $b, c, \mathbf{V}$. However, this claim is based on poor terminology, as these are unsupervised learning rules, while the energy form is clearly a supervised method.

The true Hebb's rule may indeed be understood as maximizing the variance of the output, or identically minimizing $E(\mathbf{w}) = -\langle (\mathbf{w}^\top \mathbf{x})^2 \rangle$ [2]. On the other hand, it has been extensively shown that Oja's rule may not be expressed as the gradient of *any* function [7]. As a brief summary, recall that a conservative vector field satisfies $\partial(\Delta\mathbf{w})_i/\partial w_j = \partial(\Delta\mathbf{w})_j/\partial w_i$. Oja's learning rule is $\Delta\mathbf{w} = (\mathbf{x} - \mathbf{w}y)y$. Since $\partial(yx_i - y^2 w_i)/\partial w_j = x_i x_j - 2w_i y x_j - \delta_{ij} y^2$, the last condition is clearly not met. Consequently the field is not conservative and equivalently can not be interpreted as the gradient of some function [2].

2.1 Single Neuron Comparisons of the Energy and MSE Functions

As a simple comparison of the energy 12 and the MSE 6 error functions, consider the case of a single linear neuron with two inputs, i.e. $\mathbf{x}, \mathbf{w} \in \mathcal{R}^2$. Given a large set of observations $\{\mathbf{x}_i, t_i\}$, consider first the case where there exists a unique solution $\tilde{\mathbf{w}}$ in weight space where the MSE is zero. In the implementation, this corresponds to generating the expected outputs from given inputs as $t_i = x_{i,0}\tilde{w}_0 + x_{i,1}\tilde{w}_1$ for some fixed $\tilde{\mathbf{w}}$.

Figure 1 shows contour plots of the energy and MSE surfaces in $\mathbf{w}$-space. The MSE surface for a single observation pair is shown in Figure 1a, and is the familiar trench shape. Summing over all observations to attain the total MSE function yields the bowl shape shown in Figure 1b. A sample stochastic gradient descent path is shown to converge to the exact solution, indicated by the green 'X'. Given sufficient passes over the data set, any desired precision may be attained. The energy function surface for the same single observation pair is shown in 1c, and is a bowl shape similar to the total error function. However, while the total function in 1d has a similar minimum to the MSE function, the individual observation's minima indicated by the red 'X' do not necessarily coincide the total minimum in the green. This leads to the non-converging behavior shown for the stochastic gradient descent.

As a second case, consider a data set that consists of two subsets of observations, each with a unique zero MSE solution as previously. Figure 2a shows the total MSE function's contour plot. Indicated is the minimum of the energy function, and the solutions to the two subsets. The stochastic gradient descent shows that rather than converge to the total MSE function's minimum, the solution oscillates between the solutions of the two subsets as the algorithm iterates over data pairs belonging to a random one of these subsets. Figure 2b shows the same behavior for the total energy function.

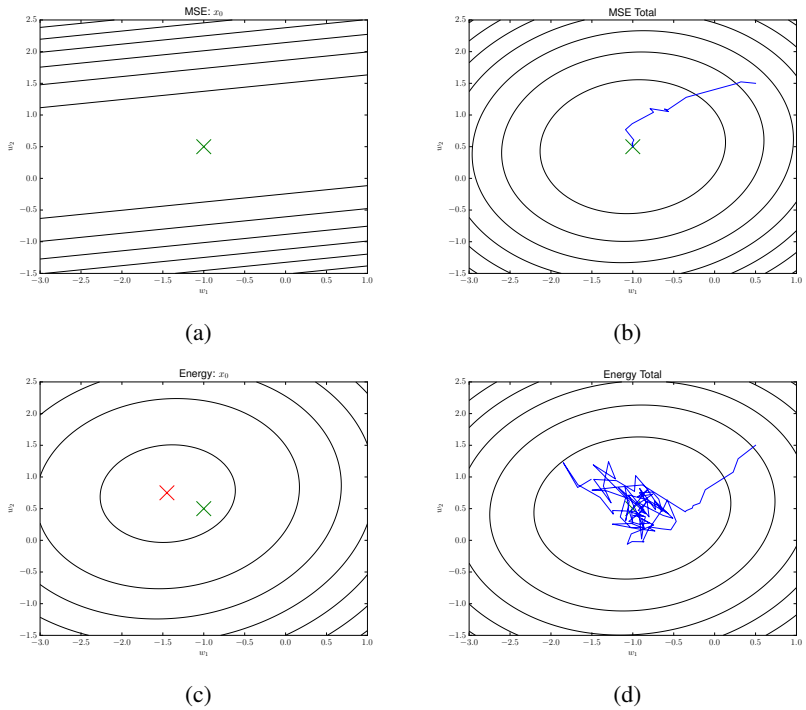


Figure 1: The error surfaces in weight space for a single linear neuron with two inputs. The observations $\{\mathbf{x}, t\}$ have a single solution in weight space with zero MSE. (a) A single observation $\mathbf{x}_0, t_0$ for the MSE function, corresponding to a trench. The green 'X' denotes the minimum of the total MSE function. (b) The total MSE function for all data sets. A sample stochastic gradient descent path from a random starting point is shown. (c) As (a), for the energy function. The red 'X' denotes the minimum of this observations energy function. (d) As (b), for the energy function.

The combination of these two simulations demonstrates that while the energy function will not necessarily converge to the same point as the MSE function, it generally exhibits oscillatory behavior in the same neighborhood in weight space. Furthermore, the bowl nature of the energy function may be exploited for a faster decrease toward the minimum by avoiding slow crawls along the trench as in the case of the MSE.

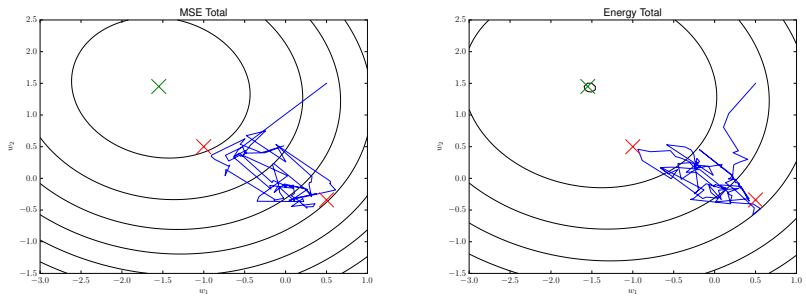
3 Backpropagation

3.1 Brief Review

A common application of gradient descent methods is to the backpropagation algorithm [2, 4, 5]. Altering the set of weights in a network to attain a desired result is generally difficult, and increases substantially with the size of the network and the complexity of its topology.

In the present work, a simplified procedure of the backpropagation algorithm is reviewed and its application discussed for the energy function. A more general treatment is deferred to other literature [2].

Begin by considering a general network as shown in Figure 3, consisting of an input layer, an output layer, and any number of hidden layers in-between. Associate some weight function $Q(\mathbf{w})$ with the network, where $\mathbf{w}$ is the vector of all connection's weights. Analogous to the case of a single neuron, the solution of the backpropgation algorithm is the set of weights which minimizes Q. The training of the network is based on a set of data pairs $\{\mathbf{x}, \mathbf{t}\}$ with $\mathbf{x}$ being the inputs fed into the network in the input layer, and $\mathbf{t}$ the expected outputs. The backpropagation is given in Algorithm 1, and proceeds by iterating over the set of observation pairs $(\mathbf{x}, \mathbf{t})$. At each iteration:



(a) (b)

Figure 2: Total error surfaces for a set observations $\{\mathbf{x}, t\}$ that consists of two subsets, each with a single solution in weight space with zero MSE. The green 'X' denotes the minimum of the total error function. The red 'X' denote the exact solutions of the two subsets. (a) The total MSE function for all data sets. A sample stochastic gradient descent path from a random starting point is shown. (b) As (a), for the energy function.

Algorithm 1 Backpropagation

procedure BACKPROPAGATION
  for each observation pair $(\mathbf{x}, \mathbf{t})$, **do**
    Feed-forward: Feed the input $\mathbf{x}$ into the network, store all outputs o_i.
    Error at the output layer: Calculate the error δ_j at the output layer $\delta_j o_i = \frac{\partial Q}{\partial w_{ij}}$.
    Backpropagate to hidden layers: Calculate the hidden layer's errors as $\delta_h = \sum_q w_{hq}\delta_q$.
    Weight updates: Update the weights of all connections as $w_{ij} := w_{ij} - \eta o_i \delta_j$.

1. *Feed-forward:* The input $\mathbf{x}$ is fed into the input layer, and forward propagate through the network. Store the output of each neuron i as o_i.
2. *Error at the output layer:* The error δ_j of the j-th neuron in the output layer is computed as
$$o_i \delta_j = \frac{\partial Q}{\partial w_{ij}}, \tag{14}$$
where w_{ij} is the weight from the i-th neuron to the j-th neuron, and o_i is the output of the i-th neuron, or conversely, the input into the j-th.
3. *Backpropagate to the hidden layers:* Since the expected output of neurons in the hidden layers is not known, it is instead estimated by errors in the layer closer to the output (in the case of the final hidden layer, the output layer). If w_{hq} is the connection from the h-th to the q-th neuron in the hidden layer, then the error is estimated as
$$\delta_h = \sum_q w_{hq}\delta_q \tag{15}$$
4. *Weight updates:* Update the weights of each connection as
$$w_{ij} := w_{ij} - \eta o_i \delta_j \tag{16}$$

3.2 MSE and Energy Formalism

In the remaining text, consider the case of a network of linear neurons, where the output is given as the weighted sum of the inputs $y_i = \mathbf{w}_i^\top \mathbf{x}_i$ where the elements x_{ji} correspond to the j-th input into the i-th neuron with weight w_{ji}.

The gradient of the MSE for the i-th neuron in the output layer is given as

$$\nabla_{\mathbf{w}_i} E_i(\mathbf{w}_i) = \mathbf{x}_i(\mathbf{w}_i^\top \mathbf{x}_i - t_i). \tag{17}$$

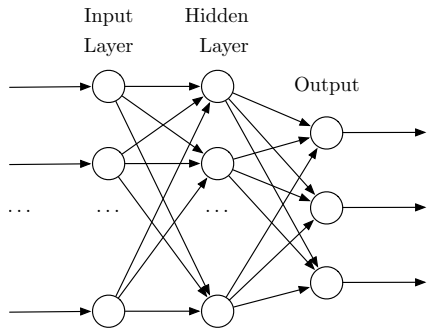


Figure 3: The neural network used for the Iris dataset. The input layer has four input neurons, each hidden layer has four neurons, and the output layer has three. One and three hidden layer cases were examined in the simulations.

Identifying this with 14, identify the error as

$$\delta_i = \mathbf{w}_i^\top \mathbf{x}_i - t_i. \tag{18}$$

Recall the gradient of the energy function is

$$\nabla_{\mathbf{w}_i} \mathcal{E}_i(\mathbf{w}_i) = -\mathbf{x}_i t_i + \frac{1}{2}(\Sigma_{\mathbf{x}_i\mathbf{x}_i} + \Sigma_{\mathbf{x}_i\mathbf{x}_i}^\top + 2\,\overline{\mathbf{x}_i}\,\overline{\mathbf{x}_i}^\top)\mathbf{w}_i. \tag{19}$$

A difficulty presents itself in defining an error δ_i as before, since it is not possible to factor $\mathbf{x}_i$ out of this expression. However, in order to backpropagate the error to the hidden layers, a definite expression is required. One work-around is to define an inverse for the vector $\mathbf{x}_i$ as the *Moore-Penrose pseudoinverse*

$$\mathbf{x}^+ \equiv \frac{\mathbf{x}^\top}{\mathbf{x}^\top \mathbf{x}}. \tag{20}$$

Note in particular that $\mathbf{x} \otimes \mathbf{x}^+ = \mathcal{I}$ where $\mathcal{I}$ is the identity matrix of appropriate dimension.

In light of this assumption, write

$$\nabla_{\mathbf{w}_i} \mathcal{E}_i(\mathbf{w}_i) = \mathbf{x}_i \left(\frac{1}{2}\mathbf{x}_i^+(\Sigma_{\mathbf{x}_i\mathbf{x}_i} + \Sigma_{\mathbf{x}_i\mathbf{x}_i}^\top)\mathbf{w}_i - t_i \right).$$

and consequently define the error as

$$\delta_i = \frac{1}{2}\mathbf{x}_i^+(\Sigma_{\mathbf{x}_i\mathbf{x}_i} + \Sigma_{\mathbf{x}_i\mathbf{x}_i}^\top)\mathbf{w}_i - t_i. \tag{21}$$

The validity of this approach is considered further in Section 5. The final rules for updating the weights of the i-th neuron with inputs $\mathbf{x}_i$ and input weights $\mathbf{w}_i$ are

$$\mathbf{w}_i := \mathbf{w}_i - \eta \mathbf{x}_i \begin{cases} \mathbf{w}_i^\top \mathbf{x}_i - t_i \text{ for MSE method,} \\ \frac{1}{2}\mathbf{x}_i^+(\Sigma_{\mathbf{x}_i\mathbf{x}_i} + \Sigma_{\mathbf{x}_i\mathbf{x}_i}^\top)\mathbf{w}_i - t_i \text{ for energy method.} \end{cases} \tag{22}$$

3.3 Bootstrapping the Covariance Matrix

Calculating the error from the gradient of the energy function 9 requires the covariance matrix of the inputs into the neurons in the output layer, that is, the outputs of neurons in the final hidden layer. While the inputs at the input layer of the network are known, the inputs into the final output layer are not. The naive approach of propagating the entire set of inputs through the network may be hindered if the size of data set is large, or the network contains a large number of hidden layers. Furthermore, the calculation needs to be performed after every weight update, as the outputs in the final layer will have changed.

A slight relaxation in precision allows for a more elegant approach using bootstrapping. Algorithm 2 outlines the procedure used to estimate the input covariance at the output layer. A total of N_r sets is constructed, each consisting of n_r random input vectors $\mathbf{x}$. The choice of N_r and n_r clearly dictates the covariance obtained, and their choice is discussed in Section 4. For each set, the input vectors are in turn fed forward through the network to obtain the input vectors to the output neurons, from which the covariance matrix for the set is calculated. Finally, the true covariance is approximated as the mean of the set of covariances $\{\sigma_i\}$, where $i = 1, 2, \ldots, N_r$.

Algorithm 2 Bootstrapping for Output Layer Covariance

```
procedure BOOTSTRAPPING
    Draw N_r sets of input vectors {x}, each set of size n_r
    for each set {x}_i, i = 1, 2, ..., N_r, do
        for each element x_j, j = 1, 2, ..., n_r, do
            Forward propagate x_j to the input into the final neuron y_j
        Compute the covariance matrix σ_i of the set {y_j}
    Approximate the true covariance as σ = mean({σ_i})
```

4 Numerical Simulations

The backpropagation algorithm was applied to learn the Iris data set [1]. The dataset consists of 150 observations, each with four attributes that denote properties of the iris flower, and a single output as classification of one of three types of iris flower. The four input attributes are approximately Gaußian distributed.

The network used is shown in Figure 3. The input layer has four input neurons, each hidden layer has four neurons, and the output layer has three. One and three hidden layer cases were examined in the simulations. The network is fully connected in that the output from each neuron is fed into the input of each neuron in the next layer. Each output neuron represents a single flower type, with the ideal outputs being represented by outputs $(n_1, n_2, n_3) = (0, 0, 1), (0, 1, 0)$ or $(1, 0, 0)$ for the three output neurons n_1, n_2, n_3, with each combination being assigned to a single classification of iris.

The network is run over 100 randomly selected training data pairs, and tested on 50 randomly selected test pairs. The parameters used in the bootstrapping were $N_r = n_r = 5$, and the learning rate was set at $\eta = 10^{-3}$. However, these parameters were not finely tuned to increase the performance of the network, but rather focusing on the comparison of the energy and MSE error functions. A future analysis to yield optimal results would utilize an activation function to create a network of binary neurons. Presently, the use of linear neurons is of interest for the output distributions.

Figure 4 shows histograms of the three output neurons for the MSE and energy error functions. The output of the energy function more closely resembles a Gaußian distribution, particularly for the single hidden layer case. The use of the covariance matrix in the learning rule captures some of the overall distribution of the input data. Ideally, the bootstrapping method then estimates a similar distribution at the output layer, leading to a learning rule that attempts to mimic the input distribution. For a greater number of hidden layers shown in the right panel of Figure4, the similarity between the MSE and energy distributions increases. This is likely due the estimation of the covariance matrix, and is further discussed below.

Figure 5 shows the error of the output neurons for each observation $|(n_1, n_2, n_3) - (t_1, t_2, t_3)|$, where (t_1, t_2, t_3) refers to the ideal output of the output neurons for the data observation. While any quantitative comparisons are unclear, it can be seen that the error for the two methods follows a similar trend, i.e. is of similar magnitude for the same observation pair. This indicates that the energy function considered balances capturing the Gaußian nature of the inputs with minimizing the square of the error.

5 Discussion

The poor performance of the network in classifying the Iris data set is primarily a result of working with linear neurons. The classification problem is more suited for binary neurons, which may attained by implementing an activation function. However, this limits the capacity to compare the MSE and energy cost function, and is therefore generally optimizing the performance is left for future analysis.

The validity of several assumptions need to be addressed. Since the inverse of a vector is not uniquely defined, the Moore-Penrose pseudoinverse was used to define the error associated with the energy cost function. This inverse is unique and guaranteed to exist, and therefore constitutes a reasonable choice. However, it is one choice out of many for the error, and may not be the optimal.

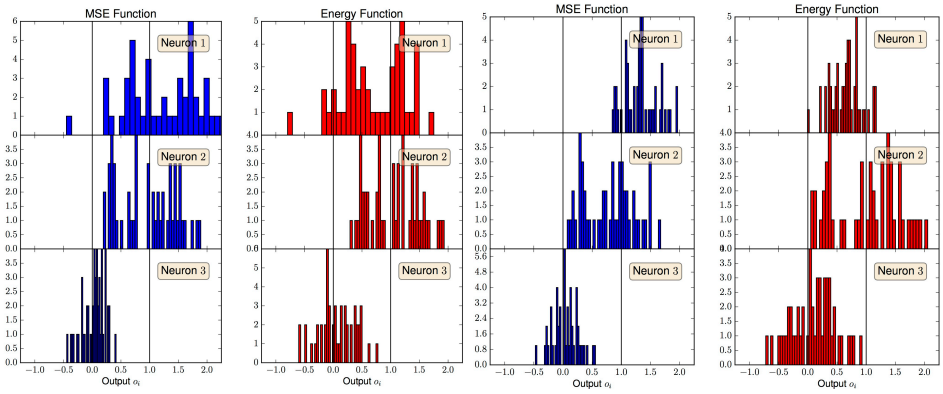
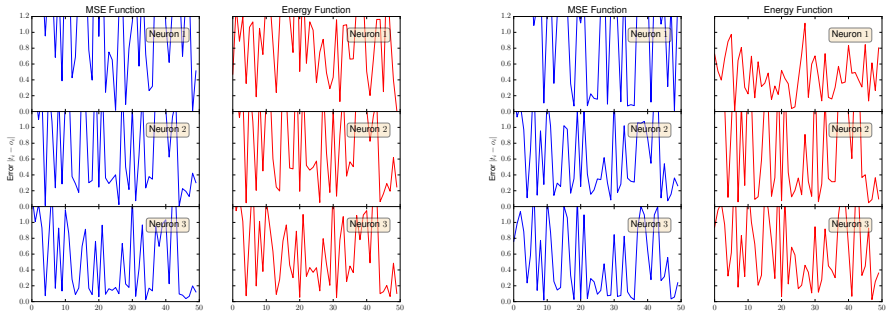


Figure 4: Histograms of the three output neurons for test pairs. Shown for networks with one hidden layer (left two panels) and three hidden layers (right two panels).



(a) (b)

Figure 5: Errors of each output neuron $|t_i - o_i|$ for a random order of test pairs.

A bootstrapping method was used to estimate the covariance of the inputs to the output neurons. Since the numerical tests were performed on networks with one and three hidden layers, it is expected that this approximation is reasonably accurate, despite the size parameters N_r and n_r not being finely tuned to optimal values. However, it is expected that the approximation's performance decreases as the size of the network grows. Alternative approximations [3] to estimate the covariance matrix may lead to better results.

The current study is primarily qualitative in examining the results of backpropagation algorithm. One possibility for future analysis for quantifying these outcomes is to use the Kolmogorov-Smirnov to compare the distributions obtained to the expected.

6 Conclusions

The MSE learning rule is a popular example of training a neural network by minimizing an associated error function. Here we show a related form which uses the covariance of the input data to capture the structure of the data and expected solutions. Applied to the backpropagation algorithm, the network balances the Gaußian nature of the input data with minimizing the MSE. We anticipate that further optimizations may lead to modified backpropagation algorithms that preserve the behavior of the networks inputs, and that are capable of learning over smaller training data sets.

Acknowledgments

I would like to thank the entire BENG 260 class, especially helpful discussions with Prof. Cauwenberghs and the TAs, and for plenty of help programming.

References

[1] K Bache and M Lichman. {UCI} Machine Learning Repository, 2013.

[2] Pierre Baldi, Yves Chauvin, and Kurt Hornik. Backpropagation: Theory, Architectures, and Applications. chapter 12, pages 389–432. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[3] Evren Imre and Adrian Hilton. Covariance estimation for minimal geometry solvers via scaled unscented transformation. *Computer Vision and Image Understanding*, 130:18–34, January 2015.

[4] Tom Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[5] Raul Rojas. Neural Networks: A Systematic Introduction. *Springer-Verlag*, 1996.

[6] Jakub M Tomczak. Associative Learning Using Ising-Like Model. 240:295–304, 2014.

[7] L Xu. Least mean square error reconstruction principle for self-organizing neural-nets. *Neural Networks*, 1993.