Chapter 13 Unconstrained Optimization and Neural Networks

An Introduction to Optimization
Spring, 2015

Wei-Ta Chu
The essence of neural networks lies in the connection weights between neurons. The selection of these weights is referred to as training or learning. We often refer to the weights as the learning parameters. A popular method for training a neural network is the backpropagation algorithm, based on an unconstrained optimization problem and an associated gradient algorithm applied to the problem.

An artificial neural network is a circuit composed of interconnected simple circuit elements called neurons. Each neuron represents a map, typically with multiple inputs and a single output.
Introduction

- Specifically, the output of the neuron is a function of the sum of the inputs. The function at the output of the neuron is called the activation function. Note that the single output of the neuron may be used as an input to several other neurons, and therefore the symbol for a single neuron has multiple arrows emanating from it. (Figure 13.1 and 13.2)

- A neural network may be implemented using an analog circuit. In this case inputs and outputs may be represented by currents and voltages.
Introduction

- In a *feed-forward neural network*, the neurons are interconnected in layers, so that the data flow in only one direction. The inputs to each neuron are weighted outputs of neurons in the preceding layer. (Figure 13.3)

- The first layer in the network is called the *input layer*, and the last layer is called the *output layer*. The layers in between the input and output layers are called *hidden layers*.

![Figure 13.3 Structure of a feedforward neural network](image-url)
Introduction

- We can view a neural network as simple a particular implementation of a map from $\mathbb{R}^n$ to $\mathbb{R}^m$, where $n$ is the number of inputs $x_1, \ldots, x_n$ and $m$ is the number of outputs $y_1, \ldots, y_m$.

- The information about the mapping is “stored” in the weights over all the neurons, and thus the neural network is a distributed representation of the mapping.

- For any given input, computation of the corresponding output is achieved through the collective effect of individual input-output characteristics of each neuron; therefore, the neural network can be considered as a parallel computation device.
Introduction

- Suppose that we are given a map $F : \mathbb{R}^n \to \mathbb{R}^m$ that we wish to implement using a given neural network. Let $(x_{d,1}, y_{d,1}), \ldots, (x_{d,p}, y_{d,p}) \in \mathbb{R}^n \times \mathbb{R}^m$, where each $y_{d,i}$ is the output of the map $F$ corresponding to the input $x_{d,i}$; that is, $y_{d,i} = F(x_{d,i})$.

- We refer to the set $\{(x_{d,1}, y_{d,1}), \ldots, (x_{d,p}, y_{d,p})\}$ as the **training set**. We train the neural network by adjusting the weights such that the map implemented by the network is close to the desired map $F$. For this reason, we can think of neural networks as function approximators.
Introduction

The form of learning described above can be thought of as learning with a teacher. The teacher supplies questions to the network in the form of $x_{d,1}, \ldots, x_{d,p}$ and tells the network the correct answers $y_{d,1}, \ldots, y_{d,p}$. Training of the network then comprises applying a training algorithm that adjusts weights based on the error between the computed and desired outputs; that is, the difference between $y_{d,i} = F(x_{d,i})$ and the output of neural network corresponding to $x_{d,i}$.
Consider a single neuron. For this particular neuron, the activation function is simply the identity (linear function with unit slope). The neuron implements the following (linear) map from $\mathbb{R}^n$ to $\mathbb{R}$. (Figure 13.4)

$$y = \sum_{i=1}^{n} w_i x_i = \mathbf{x}^T \mathbf{w}$$

where $\mathbf{x} = [x_1, \ldots, x_n]^T \in \mathbb{R}^n$ is the vector of inputs, $y \in \mathbb{R}$ is the output, and $\mathbf{w} = [w_1, \ldots, w_n]^T \in \mathbb{R}^n$ is the vector of weights.

Suppose that we are given a map $F : \mathbb{R}^n \rightarrow \mathbb{R}$. We wish to find the value of the weights $w_1, \ldots, w_n$ such that the neuron approximates the map $F$ as closely as possible.
Single-Neuron Training

To do this, we use a training set consisting of \( p \) pairs \( \{(x_{d,1}, y_{d,1}), ..., (x_{d,p}, y_{d,p})\} \) where \( x_{d,i} \in \mathbb{R}^n \) and \( y_{d,i} \in \mathbb{R}, \ i = 1, ..., p \). For each \( i \), \( y_{d,i} = F(x_{d,i}) \) is the “desired” output corresponding to the given input \( x_{d,i} \). The training problem can then be formulated as the following optimization problem:

\[
\text{minimize} \quad \frac{1}{2} \sum_{i=1}^{p} (y_{d,i} - x_{d,i}^T w)^2
\]

where the minimization is taken over all \( w = [w_1, ..., w_n]^T \in \mathbb{R}^n \)

The factor of \( \frac{1}{2} \) is added for notational convenience and does not change the minimizer.
Single-Neuron Training

The objective function above can be written in matrix form as follows. First define the matrix \( X_d \in \mathbb{R}^{n \times p} \) and vector \( y_d \in \mathbb{R}^p \) by

\[
X_d = [x_{d,1}, \ldots, x_{d,p}] \quad y_d = \begin{bmatrix} y_{d,1} \\ \vdots \\ y_{d,p} \end{bmatrix}
\]

Then, the optimization problem becomes

\[
\text{minimize } \frac{1}{2} \| y_d - X_d^T w \|^2
\]

There are two cases to consider in this optimization problem:

\( p \leq n \) and \( p > n \). In the first case, we have at most as many training pairs as the number of weights. Assume that \( \text{rank}(X_d^T) = p \) in this case there are an infinitely many points satisfying \( y_d = X_d^T w \). Hence, there are infinitely many solutions to the optimization problem, with the optimal objective function value of 0.
A possible criterion for this selection is that of minimizing the solution norm. This is exactly the problem considered in Section 12.3. Recall that the minimum-norm solution is $\mathbf{w}^* = \mathbf{X}_d (\mathbf{X}_d^T \mathbf{X}_d)^{-1} \mathbf{y}_d$. An efficient iterative algorithm for finding this solution is Kaczmarz’s algorithm.

Kaczmarz’s algorithm in this setting takes the form

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mu \frac{e_k \mathbf{x}_{d,R(k)+1}}{\| \mathbf{x}_{d,R(k)+1} \|^2}$$

where $\mathbf{w}^{(0)} = 0$ and

$$e_k = y_{d,R(k)+1} - \mathbf{x}_{d,R(k)+1}^T \mathbf{w}^{(k)}$$

Recall that $R(k)$ is the unique integer in $\{0, ..., p - 1\}$ satisfying $k = l p + R(k)$ for some integer $l$; that is, $R(k)$ is the remainder that results if we divide $k$ by $p$. 
The single neuron together with the training algorithm is often called *Adaline*, an acronym for *adaptive linear element*.

(Figure 13.5)

We now consider the case where \( p > n \). We assume that \( \text{rank}(X_d^T) = n \). In this case the objective function \( \frac{1}{2} \| y_d - X_d^T w \| ^2 \) is simply a strictly convex quadratic function of \( w \), because the matrix \( X_d X_d^T \) is a positive definite matrix.  

To solve this optimization problem, we have at our disposal the whole slew of unconstrained optimization algorithms considered in earlier chapters.
Single-Neuron Training

For example, we can use a gradient algorithm, which in this case takes the form

$$w^{(k+1)} = w^{(k)} + \alpha_k X_d e^{(k)}, \quad e^{(k)} = y_d - X_d^T w^{(k)}$$

The discussion above assumed that the activation function for the neuron is the identity map. The derivation and analysis of the algorithms can be extended to the case of a general differentiable activation function $f_a$. Specifically, the output of the neuron in this case is given by

$$y = f_a \left( \sum_{i=1}^n w_i x_i \right) = f_a(x^T w)$$

The algorithm for the case of a single training pair $(x_d, y_d)$ has the form

$$w^{(k+1)} = w^{(k)} + \mu \frac{e_k x_d}{\|x_d\|^2}$$

$$e_k = y_d - f_a(x_d^T w^{(k)})$$
Now we consider a neural network consisting of many layers. For simplicity, we restrict our attention to networks with three layers. (Figure 13.6)

There are $n$ inputs $x_i$, where $i = 1, \ldots, n$. We have $m$ outputs $y_s$, $s = 1, \ldots, m$. There are $l$ neurons in the hidden layer. The outputs of the neurons in the hidden layer are $z_j$, $j = 1, \ldots, l$. The inputs $x_1, \ldots, x_n$ are distributed to the neurons in the hidden layer. We may think of the neurons in the input layer as single-input-single-output linear elements, with each activation function being the identity map.
The Backpropagation Algorithm

- We denote the activation functions of the neurons in the hidden layer by \( f_j^h \), where \( j = 1, ..., l \), and the activation functions of the neurons in the output layer by \( f_s^o \), \( s = 1, ..., m \). Note that each activation function is a function from \( R \) to \( R \).

- We denote the weights for inputs into the hidden layer by \( w_{ji}^h \), we denote the weights for inputs from the hidden layer into the output layer by \( w_{sj}^o \). Given the weights \( w_{ji}^h \) and \( w_{sj}^o \), the neural network implements a map from \( R^n \) to \( R^m \). To find an explicit formula for this map, let us denote the input to the \( j \)th neuron in the hidden layer by \( v_j \) and the output of the \( j \)th neuron in the hidden layer \( z_j \).
The Backpropagation Algorithm

- We have

\[ v_j = \sum_{i=1}^{n} w_{ji}^h x_i \]

\[ z_j = f_j^h \left( \sum_{i=1}^{n} w_{ji}^h x_i \right) \]

- The output from the \( s \)th neuron of the output layer is

\[ y_s = f_s^o \left( \sum_{j=1}^{l} w_{sj}^o z_j \right) \]

Therefore, the relationship between the inputs and the \( s \)th output \( y_s \) is given by

\[ y_s = f_s^o \left( \sum_{j=1}^{l} w_{sj}^o f_j^h \left( \sum_{i=1}^{n} w_{ji}^h x_i \right) \right) = F_s(x_1, \ldots, x_n) \]

- The overall mapping is therefore given by

\[
\begin{bmatrix}
y_1 \\
\vdots \\
y_m
\end{bmatrix} =
\begin{bmatrix}
F_1(x_1, \ldots, x_n) \\
\vdots \\
F_m(x_1, \ldots, x_n)
\end{bmatrix}
\]
The Backpropagation Algorithm

- We analyze the case where the training set consists of a single pair \((x_d, y_d)\), where \(x_d \in \mathbb{R}^n\) and \(y_d \in \mathbb{R}^m\). In practice, the training set consists of many such pairs, and training is typically performed with each pair at a time.

- The training of the neural network involves adjusting the weights of the network such that the output generated by the network for the given input \(x_d = [x_{d1}, \ldots, x_{dn}]^T\) is as close to \(y_d\) as possible. Formally, this can be formulated as

\[
\text{minimize} \quad \frac{1}{2} \sum_{s=1}^{m} (y_{ds} - y_s)^2
\]

where \(y_s, s = 1, \ldots, m\), are the actual outputs of the neural network in response to the inputs \(x_{d1}, \ldots, x_{dn}\), as given by

\[
y_s = f_s^o \left( \sum_{j=1}^{l} w^o_{sj} f_j^h \left( \sum_{i=1}^{n} w^h_{ji} x_i \right) \right)
\]
The Backpropagation Algorithm

- This minimization is taken over all \( w^h_{ji} \), \( w^o_{sj} \), \( i = 1, \ldots, n \), \( j = 1, \ldots, l \), \( s = 1, \ldots, m \). For simplicity of notation, we use the symbol \( \mathbf{w} \) for the vector
  \[
  \mathbf{w} = \{ w^h_{ji}, w^o_{sj} : i = 1, \ldots, n, j = 1, \ldots, l, s = 1, \ldots, m \}
  \]
  and the symbol \( E \) for the objective function to be minimized; that is,

  \[
  E(\mathbf{w}) = \frac{1}{2} \sum_{s=1}^{m} (y_{ds} - y_s)^2 = \frac{1}{2} \sum_{s=1}^{m} \left( y_{ds} - f^o_s \left( \sum_{j=1}^{l} w^o_{sj} f^h_j \left( \sum_{i=1}^{n} w^h_{ji} x_{di} \right) \right) \right)^2
  \]
The Backpropagation Algorithm

- To solve the optimization problem above, we use a gradient algorithm with fixed step size. To formulate the algorithm, we need to compute the partial derivatives of $E$ with respect to each component of $\mathbf{w}$. For this, let us first fix the indices $i$, $j$, and $s$. We first compute the partial derivative of $E$ with respect to $w_{sj}^o$. For this, we write

$$E(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^{m} \left( y_{dp} - f_p^o \left( \sum_{q=1}^{l} w_{pq}^o z_q \right) \right)^2$$

where for each $q = 1, ..., l$,

$$z_q = f_q^h \left( \sum_{i=1}^{n} w_{qi}^h x_{di} \right)$$
The Backpropagation Algorithm

Using the chain rule, we obtain

$$\frac{\partial E}{\partial w_{sj}^o}(\mathbf{w}) = -(y_{ds} - y_s) f_s'(\sum_{q=1}^{l} w_{sq}^o z_q) z_j$$

where $f_s' : R \rightarrow R$ is the derivative of $f_s^o$. For simplicity of notation, we write

$$\delta_s = (y_{ds} - y_s) f_s'(\sum_{q=1}^{l} w_{sq}^o z_q)$$

We can think of each $\delta_s$ as a scaled output error, because it is the difference between the actual output $y_s$ of the neural network and the desired output $y_{ds}$, scaled by $f_s'(\sum_{q=1}^{l} w_{sq}^o z_q)$.

Using the $\delta_s$ notation, we have

$$\frac{\partial E}{\partial w_{sj}^o}(\mathbf{w}) = -\delta_s z_j$$
We next compute the partial derivative of $E$ with respect to $w_{ji}^h$. We start with the equation

$$E(w) = \frac{1}{2} \sum_{p=1}^{m} \left( y_{dp} - f_p^o \left( \sum_{q=1}^{l} w_{pq}^o f_q^h \left( \sum_{i=1}^{n} w_{qr}^h x_{dr} \right) \right) \right)^2$$

Using the chain rule again, we get

$$\frac{\partial E}{\partial w_{ji}^h}(w) = - \sum_{p=1}^{m} (y_{dp} - y_p) f_{p}^{o'} \left( \sum_{q=1}^{l} w_{pq}^{o} z_{q} \right) w_{pj}^{o} f_{j}^{h'} \left( \sum_{r=1}^{n} w_{jr}^{h} x_{dr} \right) x_{di}$$

where $f_{j}^{h'} : R \rightarrow R$ is the derivative of $f_{j}^{h}$. Simplifying the above yields

$$\frac{\partial E}{\partial w_{ji}^h}(w) = - \left( \sum_{p=1}^{m} \delta_{p} w_{pj}^{o} \right) f_{j}^{h'}(v_{j}) x_{di}$$

where $v_{j} = \sum_{i=1}^{n} w_{ji}^{h} x_{i}$
The Backpropagation Algorithm

- We are now ready to formulate the gradient algorithm for updating the weights of the neural network. We write the update equations for the two sets of weights $w_{sj}^o$ and $w_{ji}^h$ separately. We have

$$w_{sj}^{o(k+1)} = w_{sj}^{o(k)} + \eta \delta_{s}^{(k)} z_{j}^{(k)}$$

$$w_{ji}^{h(k+1)} = w_{ji}^{h(k)} + \eta \left( \sum_{p=1}^{m} \delta_{p}^{(k)} w_{pj}^{o(k)} \right) f_{j}'(v_{j}^{(k)}) x_{di}$$

where $\eta$ is the (fixed) step size and

$$v_{j}^{(k)} = \sum_{i=1}^{n} w_{ji}^{h(k)} x_{di}$$

$$z_{j}^{(k)} = f_{j}'(v_{j}^{(k)})$$

$$y_{s}^{(k)} = f_{s}^{o} \left( \sum_{q=1}^{l} w_{sq}^{o(k)} z_{q}^{(k)} \right)$$

$$\delta_{s}^{(k)} = (y_{ds} - y_{s}^{(k)}) f_{s}' \left( \sum_{q=1}^{l} w_{sq}^{o(k)} z_{q}^{(k)} \right)$$
The Backpropagation Algorithm

- The update equation for the weights $w^o_{sj}$ of the output layer neurons is illustrated in Figure 13.7, whereas the update equation for the weights $w^h_{ji}$ of the hidden layer neurons is illustrated in Figure 13.8.
The Backpropagation Algorithm

- The update equations are referred to the *backpropagation algorithm*. The reason for the name backpropagation is that the output errors $\delta_1^{(k)}, \ldots, \delta_m^{(k)}$ are propagated back from the output layer to the hidden layer and are used in the update equation for the hidden layer weights.

- In the discussion above we assumed only a single hidden layer. If we have multiple hidden layers, the update equations for the weights will resemble the equations derived above. The output errors are propagated backward from layer to layer and are used to update the weights at each layer.
The Backpropagation Algorithm

- Summary: Using the inputs $x_{di}$ and the current set of weights, we first compute the quantities $v_j^{(k)}$, $z_j^{(k)}$, $y_i^{(k)}$, $\delta_s^{(k)}$ in turn. This is called the **forward pass** of the algorithm, because it involves propagating the input forward from the input layer to the output layer.

- Next, we compute the updated weights using the quantities computed in the forward pass. This is called the **reverse pass** of the algorithm, because it involves propagating the computed output errors $\delta_s^{(k)}$ backward through the network.
Example

- Consider the neural network shown in Figure 13.9. The activation functions for all the neurons are given by $f(v) = \frac{1}{1 + e^{-v}}$. This particular activation function has the convenient property that $f'(v) = f(v)(1 - f(v))$. Therefore, using this property, we can write

$$
\delta_1 = (y_d - y_1)f'(\sum_{q=1}^{2} w_{1q}^o z_q) \\
= (y_d - y_1)f(\sum_{q=1}^{2} w_{1q}^o z_q)\left(1 - f(\sum_{q=1}^{2} w_{1q}^o z_q)\right) \\
= (y_d - y_1)y_1(1 - y_1)
$$
Example

Suppose that the initial weights are \( w_{11}^{h(0)} = 0.1, \ w_{12}^{h(0)} = 0.3, \ w_{21}^{h(0)} = 0.3, \ w_{22}^{h(0)} = 0.4, \ w_{11}^{o(0)} = 0.4 \) and \( w_{12}^{o(0)} = 0.6 \). Let \( x_d = [0.2, 0.6]^T \) and \( y_d = 0.7 \). Perform one iteration of the backpropagation algorithm to update the weights of the network. Use a step size of \( \eta = 10 \).

To proceed, we first compute

\[
\begin{align*}
    v_1^{(0)} &= w_{11}^{h(0)} x_{d1} + w_{12}^{h(0)} x_{d2} = 0.2 \\
    v_2^{(0)} &= w_{21}^{h(0)} x_{d1} + w_{22}^{h(0)} x_{d2} = 0.3
\end{align*}
\]

Next, we compute

\[
\begin{align*}
    z_1^{(0)} &= f(v_1^{(0)}) = \frac{1}{1 + e^{-0.2}} = 0.5498 \\
    z_2^{(0)} &= f(v_2^{(0)}) = \frac{1}{1 + e^{-0.3}} = 0.5744
\end{align*}
\]
Example

- We then compute
  \[ y_1^{(0)} = f(w_{11}^{(0)} z_1^{(0)} + w_{12}^{(0)} z_2^{(0)}) = f(0.5646) = 0.6375 \]
  which gives an output error of
  \[ \delta_1^{(0)} = (y_d - y_1^{(0)})y_1^{(0)}(1 - y_1^{(0)}) = 0.01444 \]
  This completes the forward pass.

- To update the weights, we use
  \[ w_{11}^{(1)} = w_{11}^{(0)} + \eta \delta_1^{(0)} z_1^{(0)} = 0.4794 \]
  \[ w_{12}^{(1)} = w_{12}^{(0)} + \eta \delta_1^{(0)} z_2^{(0)} = 0.6830 \]

  and, using the fact that \( f'(v_j^{(0)}) = f(v_j^{(0)})(1 - f(v_j^{(0)})) = z_j^{(0)}(1 - z_j^{(0)}) \)
  we get
  \[ w_{11}^{h(1)} = w_{11}^{h(0)} + \eta \delta_1^{(0)} w_{11}^{o(0)} z_1^{(0)}(1 - z_1^{(0)}) x_{d1} = 0.1029 \]
  \[ w_{12}^{h(1)} = w_{12}^{h(0)} + \eta \delta_1^{(0)} w_{11}^{o(0)} z_1^{(0)}(1 - z_1^{(0)}) x_{d2} = 0.3086 \]
  \[ w_{21}^{h(1)} = w_{21}^{h(0)} + \eta \delta_1^{(0)} w_{12}^{o(0)} z_2^{(0)}(1 - z_2^{(0)}) x_{d1} = 0.3042 \]
  \[ w_{22}^{h(1)} = w_{22}^{h(0)} + \eta \delta_1^{(0)} w_{12}^{o(0)} z_2^{(0)}(1 - z_2^{(0)}) x_{d2} = 0.4127 \]
Example

Thus, we have completed one iteration of the backpropagation algorithm. We can easily check that $y^{(1)}_1 = 0.6588$, and hence $|y_d - y^{(1)}_1| < |y_d - y^{(0)}_1|$; that is, the actual output of the neural network has become closer to the desired output as a result of updating the weights.

After 15 iterations of the backpropagation algorithm, we get

$$
w^{o(15)}_{11} = 0.6365 \quad w^{h(15)}_{12} = 0.3315$$
$$w^{o(15)}_{12} = 0.8474 \quad w^{h(15)}_{21} = 0.3146$$
$$w^{h(15)}_{11} = 0.1105 \quad w^{h(15)}_{22} = 0.4439$$

The resulting value of the output corresponding to the input $\mathbf{x}_d = [0.2, 0.6]^T$ is $y^{(15)}_1 = 0.6997$. 
The function of the form \( f(v) = 1/(1 + e^{-v}) \) is called a **sigmoid** and is a popular activation function used in practice. It is possible to use a more general version of the sigmoid function, of the form (Figure 13.10)

\[
g(v) = \frac{\beta}{1 + e^{-(v-\theta)}}
\]

The parameters \( \beta \) and \( \theta \) represent *scale* and *shift* (or *location*) parameters respectively.
The parameter $\theta$ is often interpreted as a threshold. If such an activation function is used in a neural network, we would also want to adjust the values of the parameters $\beta$ and $\theta$, which also affect the value of the objective function to be minimized.

However, it turns out that these parameters can be incorporated into the backpropagation algorithm simply by treating them as additional weights in the network. Specifically, we can represent a neuron with activation function $g$ as one with activation function $f$ with the addition of two extra weights.
Example

Consider the same neural network as in the previous example. We introduce shift parameters $\theta_1, \theta_2, \theta_3$ to the activation functions in the neurons. Using the configuration illustrated in Figure 13.11, we can incorporate the shift parameters into the backpropagation algorithm. We have

\[ v_1 = w_{11}^h x_{d1} + w_{12}^h x_{d2} - \theta_1 \]
\[ v_2 = w_{21}^h x_{d1} + w_{22}^h x_{d2} - \theta_2 \]
\[ z_1 = f(v_1) \]
\[ z_2 = f(v_2) \]
\[ y_1 = f(w_{11}^o z_1 + w_{12}^o z_2 - \theta_3) \]
\[ \delta_1 = (y_d - y_1)y_1(1 - y_1) \]

where $f$ is the sigmoid function $f(v) = 1/(1 + e^{-v})$
Example

- The components of the gradient of the objective function $E$ with respect to the shift parameters are

$$\frac{\partial E}{\partial \theta_1} = \delta_1 w_{11}^o z_1 (1 - z_1)$$

$$\frac{\partial E}{\partial \theta_2} = \delta_1 w_{12}^o z_2 (1 - z_2)$$

$$\frac{\partial E}{\partial \theta_3} = \delta_1$$
Example

Consider the neural network of the previous example. We wish to train the neural network to approximate the exclusive OR (XOR) function. Note that the XOR function has two inputs and one output.

To train the neural network, we use the following training pairs

\[ x_{d,1} = [0, 0]^T \quad y_{d,1} = 0 \]
\[ x_{d,2} = [0, 1]^T \quad y_{d,2} = 1 \]
\[ x_{d,3} = [1, 0]^T \quad y_{d,3} = 1 \]
\[ x_{d,4} = [1, 1]^T \quad y_{d,4} = 0 \]

We now apply the backpropagation algorithm to train the network using the training pairs above.
Example

- To do this, we apply one pair per iteration in a cycle fashion. In other words, in the $k$ th iteration of the algorithm, we apply the pair $(x_{d,R(k)+1}, y_{d,R(k)+1})$, where, as in Kaczmarz’s algorithm, $R(k)$ is the unique integer in $\{0,1,2,3\}$ satisfying $k = 4l + R(k)$ for some integer $l$; that is, $R(k)$ is the remainder that results if we divide $k$ by 4.

- The experiment yields the following weights

\[
\begin{align*}
  w_{11}^o &= -11.01 & w_{11}^h &= -7.777 \\
  w_{12}^o &= 10.92 & w_{12}^h &= -8.403 \\
  \theta_1 &= -3.277 & w_{21}^h &= -5.593 \\
  \theta_2 &= -8.357 & w_{22}^h &= -5.638 \\
  \theta_3 &= 5.261
\end{align*}
\]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.007</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.99</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.009</td>
</tr>
</tbody>
</table>