A neural network implementation in FORTRAN

J. Ravnik*

February 27, 2020

Abstract

This technical note briefly describes a neural network implementation and gradient descent and genetic training algorithms.

1 Neural networks

The neural net is composed of nodes distributed in layers. Values of nodes for each layer are stored as vectors, i.e. $\vec{n}_i$, where i is the layer number. Neuron $\vec{n}_0$ represents the input to the neural network. σ_i is an activation function and works on vectors. $\vec{b}_i$ are scalar values called bias. W_i are weight matrices. Number of rows in W_i is equal to the number of elements in $\vec{n}_i$ neuron, number of columns in W_i matrix is equal to the number of elements in $\vec{n}_{i-1}$ neuron. Popular choices for the activation function include the sigmoid and the identity function

$$\sigma(x) = \frac{1}{1+\exp(-x)}, \qquad \sigma(x) = x \tag{1}$$

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)\left[1-\sigma(x)\right], \qquad \frac{\partial \sigma(x)}{\partial x} = 1 \tag{2}$$

An input is propagated through the network to output layer by a sequence of the following algebraic expressions

$$\begin{aligned}
\vec{n}_1 &= \sigma_1\left(\vec{b}_1 + W_1\vec{n}_0\right) \\
\vec{n}_2 &= \sigma_2\left(\vec{b}_2 + W_2\vec{n}_1\right) \\
&\vdots \\
\vec{n}_i &= \sigma_i\left(\vec{b}_i + W_i\vec{n}_{i-1}\right) \\
&\vdots \\
\vec{n}_k &= \sigma_k\left(\vec{b}_k + W_k\vec{n}_{k-1}\right)
\end{aligned} \tag{3}$$

*Faculty of Mechanical Engineering, University of Maribor, Slovenia, jure.ravnik@um.si

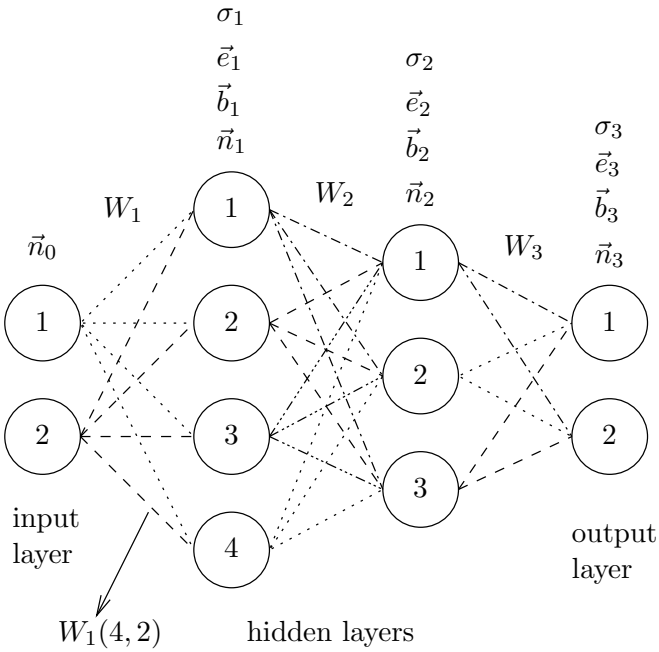

Figure 1: A four-layer neural network. The neurons are labeled $\vec{n}_0$ to $\vec{n}_3$, where $\vec{n}_0$ is the input neuron and $\vec{n}_3$ the output neuron. Hidden and output layers have a bias $\vec{b}$, an activation function σ and an error $\vec{e}$ ascossiated with them. Connections between layers are defined using wiehgts, which are stored in matrices W_1, W_2 and W_3.

For example, based on Figure 1, the value of the output neuron is

$$\vec{n}_3 = \sigma_3\left(\vec{b}_3 + W_3\sigma_2\left(\vec{b}_2 + W_2\sigma_1\left(\vec{b}_1 + W_1\vec{n}_0\right)\right)\right) \tag{4}$$

Let the number of layers in the network be $k+1$. Thus, the output neuron is $\vec{n}_k$. Considering a known training data pair $\vec{t}_0$ and $\vec{t}_k$, we define the error vector at the output layer as

$$\vec{e}_k = \vec{t}_k - \vec{n}_k, \tag{5}$$

where $\vec{n}_k$ is the output of the network based on the input $\vec{n}_0 = \vec{t}_0$ obtained using eq. (3).

The error of the output layer can be back propagated to hidden layers by distributing it based on the weights. This is achieved by multiplication of the transposed weight matrix

$$\begin{aligned} \vec{e}_{k-1} &= W_k^T\vec{e}_k \\ &\vdots \\ \vec{e}_2 &= W_3^T\vec{e}_3 \\ \vec{e}_1 &= W_2^T\vec{e}_2 \end{aligned} \tag{6}$$

For example, based on Figure 1, the error vectors can be calculated as

$$\begin{aligned} \vec{e}_3 &= \vec{t}_3 - \sigma_3\left(\vec{b}_3 + W_3\sigma_2\left(\vec{b}_2 + W_2\sigma_1\left(\vec{b}_1 + W_1\vec{t}_0\right)\right)\right) \\ \vec{e}_2 &= W_3^T\vec{e}_3 \\ \vec{e}_1 &= W_2^T\left(W_3^T\vec{e}_3\right), \end{aligned} \tag{7}$$

where $\vec{t}_0$ and $\vec{t}_3$ are the training data pair.

Next, we define the L_2 error norm of the neural network for a single training pair as

$$L_2 = \vec{e}_k \cdot \vec{e}_k \tag{8}$$

In general we posses N training pairs, $\vec{t}_0^{(j)}$ and $\vec{t}_k^{(j)}$, where j denotes the training pair number. Then, we define the error of the network via the following norm

$$E = \frac{1}{N}\sum_{j=1}^{N} L_2^{(j)} = \frac{1}{N}\sum_{j=1}^{N} \vec{e}_k^{(j)} \cdot \vec{e}_k^{(j)} = \frac{1}{N}\sum_{j=1}^{N}\left(\vec{t}_k^{(j)} - \vec{n}_k^{(j)}\right)^2, \tag{9}$$

where $\vec{n}_k^{(j)}$ is obtained by propagating $\vec{n}_0^{(j)} = \vec{t}_0^{(j)}$ through the network. When training the network, our task is to minimize E by choosing an appropriate network structure (number and size of layers and activation function types) and then finding the optimal weights and biases.

2 Optimization of weights and biases

2.1 Monte Carlo strategy

We randomly choose weights and biases and calculate the norm E using eq. (9). After many attempts, we choose the weights and biases that produce the smallest E.

2.2 Gradient descent strategy

We consider one training data pair at a time, i.e. we minimize L_2 in eq. (8). For weights, the following strategy is used

$$W_i^{(r,c)} \leftarrow W_i^{(r,c)} - \alpha \frac{\partial L_2}{\partial W_i^{(r,c)}}, \tag{10}$$

where α is the learning rate, $0 < \alpha \leq 1$ and $W_i^{(r,c)}$ is the row r, column c entry in the weight matrix W_i.

$$\frac{\partial L_2}{\partial W_i^{(r,c)}} = \frac{\partial(\vec{e}_i \cdot \vec{e}_i)}{\partial W_i^{(r,c)}} = -2e_i^{(r)} \frac{\partial n_i^{(r)}}{\partial W_i^{(r,c)}} \tag{11}$$

$$= -2e_i^{(r)} \frac{\partial \sigma_i \left(\vec{b}_i + W_i \vec{n}_{i-1}\right)^{(r)}}{\partial W_i^{(r,c)}} \tag{12}$$

When the activation function is a sigmoid, then according to (2) we obtain

$$= -2e_i^{(r)} n_i^{(r)} \left(1 - n_i^{(r)}\right) \frac{\partial \left(\vec{b}_i + W_i \vec{n}_{i-1}\right)^{(r)}}{\partial W_i^{(r,c)}} \tag{13}$$

$$= -2e_i^{(r)} n_i^{(r)} \left(1 - n_i^{(r)}\right) n_{i-1}^{(c)} \tag{14}$$

Finally, for a sigmoid activation function we obtain the following weight update expression

$$W_i^{(r,c)} \leftarrow W_i^{(r,c)} + 2\alpha e_i^{(r)} n_i^{(r)} \left(1 - n_i^{(r)}\right) n_{i-1}^{(c)} \tag{15}$$

For identity activation function we have

$$W_i^{(r,c)} \leftarrow W_i^{(r,c)} + 2\alpha e_i^{(r)} n_{i-1}^{(c)} \tag{16}$$

Biases are updated using

$$b_i^{(r)} \leftarrow b_i^{(r)} - \alpha \frac{\partial L_2}{\partial b_i^{(r)}} \tag{17}$$

act. function	$\Delta W_i^{(r,c)}$	$\Delta b_i^{(r)}$
sigmoid	$\alpha e_i^{(r)} n_i^{(r)} \left(1 - n_i^{(r)}\right) n_{i-1}^{(c)}$	$\alpha e_i^{(r)} n_i^{(r)} \left(1 - n_i^{(r)}\right)$
identity	$\alpha e_i^{(r)} n_{i-1}^{(c)}$	$\alpha e_i^{(r)}$

Table 1: Formulas used to update weights and biases based on a single training pair.

where

$$\frac{\partial L_2}{\partial b_i^{(r)}} = -2e_i^{(r)} \frac{\partial n_i^{(r)}}{\partial b_i^{(r)}} = -2e_i^{(r)} \frac{\partial \sigma_i \left(\vec{b}_i + W_i \vec{n}_{i-1}\right)^{(r)}}{\partial b_i^{(r)}} \tag{18}$$

which leads to

$$b_i^{(r)} \leftarrow b_i^{(r)} + 2\alpha e_i^{(r)} n_i^{(r)} \left(1 - n_i^{(r)}\right) \tag{19}$$

in the case of sigmoid activation function and to

$$b_i^{(r)} \leftarrow b_i^{(r)} + 2\alpha e_i^{(r)} \tag{20}$$

in the case of identity activation function.

2.3 Genetic strategy

Let a neural network be defined by a chromosome, which is collection of genes. Genes are values of weights and biases. Using (8) we defined the L_2 norm of the network. We recognize that we aim to train the network in such a way that the L_2 norm will be minimized.

We consider a population of N chromosomes. We define fitness of i-th the chromosome as

$$F_i = \frac{1}{1 + L_{2,i}} \tag{21}$$

The total fitness is

$$F = \sum_{i=1}^{N} F_i \tag{22}$$

Chromosome probability is defined as

$$P_i = \frac{F_i}{F} \tag{23}$$

Chromosome cumulative probability is

$$C_i = \sum_{j=1}^{i} P_j \tag{24}$$

Chromosome cumulative probability values span from 0.0 to 1.0. When a random number is chosen between zero and one, we can use it to select fitter chromosomes with greater probability than not-fit chromosomes. The following algorithm is used:

`Algorithm I.`

- let r be a random number between zero and one
- for $i = 1$ to N
  - if $r < C_i$ then we select this chromosome

The crossover algorithm is `Algorithm II.`

- let COR be a user defined parameter called Crossover Rate
- for $i = 1$ to N
  - let r be a random number between zero and one
  - if $r < COR$ then
    - choose another chromosome (j-th)
    - create a new population chromosome by randomly mixing genes of both parents (i and j)
  - else
    - copy i-th chromosome to new population
  - end if

The mutation algorithm is `Algorithm III.`

- let MR be a user defined parameter called mutation rate
- for $i = 1$ to $N \cdot$ number of weight and biases $\cdot MR$
  - set a random gene of a random chromosome to a random value

The complete genetic optimization algorithm works in the following way `Algorithm IV.`

- choose population size N, number of iterations n, crossover rate COR and mutation rate MR
- initialize population with random genes
- loop n times
  - evaluate cumulative probability for each chromosome in population using eq. (24).
  - choose N fittest chromosomes using the algorithm I.

  - cross over chromosomes using the algorithm II.
  - mutate chromosomes using the algorithm III.

- return chromosome with best L_2 norm.

3 Example

A neural network was trained to approximate the following function $y = sin(4\pi x) + 4x$, $x \in (0, 1)$. The network had two hidden layer of size 10. 66 training points were used. The result is shown in Figure 2. Gradient descent optimization strategy was used.

References

[1] Tariq Rashid. *Make Your Own Neural Network*. 2016.

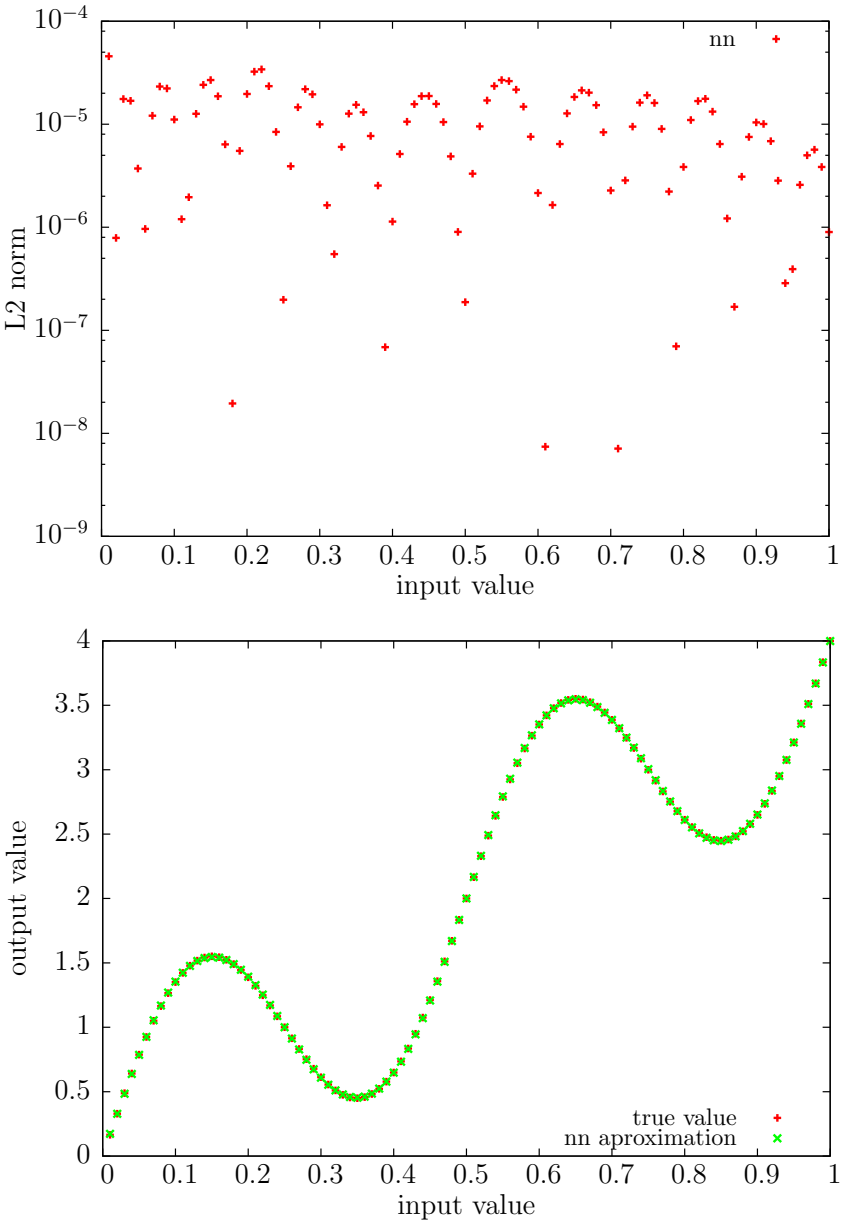

Figure 2: A neural network trained to approximate the $y = sin(4\pi x) + 4x$ function.