

Multi-Class Binary Neural Network for Handwritten Digits


Problem ID: mnist10clas

CPU Time limit: 1 second

Memory limit: 1024 MB

Difficulty: 1.3

Source: International Co
Programming Contest (IC
Challenge 2019

License: 

The objective of this problem is to train a Binary Neural Network to classify images. The training occurs on your own computer, and you should submit a program which simply outputs the trained weights.

The images are of handwritten digits from the well-known MNIST dataset. Your task is to classify each image into one of **ten** classes: digits 0 through 9.

An MNIST image is originally a collection of 28×28 pixels, each pixel being a byte, but for this problem it is given to us in a compressed form of 51 bits. For the m^{th} image, the input is denoted by



MNIST handwritten digits – 0 through 9.

$$x^m = \{x_n^m\}_{n=0,\dots,50} \in \{-1, 1\}^{51},$$

that is, x^m is a vector of 51 values (either -1 or 1). Its corresponding label is $y^m \in \{0, 1, \dots, 9\}$. There are a number of labelled images given for training, and a separate set of images reserved for testing (and known only to the judging software). The goal is to maximize the classification accuracy on the test images using a binary neural network, which we explain next.

Binary Neural Network Architecture

For this problem, we use the following binary neural network architecture, illustrated below. After the input layer, there are three layers:

1. a layer with 150 binary ‘perceptron nodes’, each fully connected to the 51 inputs via configurable weighted connections;
2. a layer with 10 ‘sum nodes’ (one per class), each connected to the outputs of 15 different perceptron nodes; and
3. a layer with one ‘argmax node’ connected to the 10 sum nodes.

Next, we describe how these nodes work in detail.

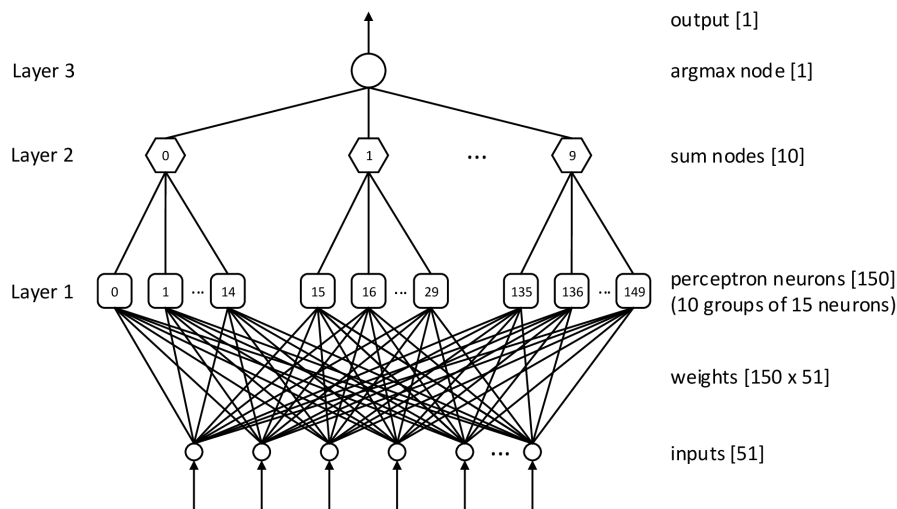


Figure 1: The binary neural network architecture for classifying digits.

Perceptron Node

The weights for each perceptron node are configurable. Each node k has 51 weights

$$w^k = \{w_n^k\}_{n=0,\dots,50} \in \{-1, 1\}^{51}.$$

The output $y^{km} \in \{-1, 1\}$ of perceptron node k for input x^m is:

$$y^{km} = \text{sign}\left(\sum_{n=0}^{50} w_n^k x_n^m\right),$$

where the $\text{sign}(\cdot)$ function takes value 1 if its argument is positive and -1 otherwise (note: according to this definition, $\text{sign}(0) = -1$). For example, if $\sum_{n=0}^{50} w_n^k x_n^m = -7$, then $y^{km} = -1$.

Sum Node

There is one sum node per class. The behavior of the sum nodes is always the same. Each produces the sum of the outputs of the 15 perceptron nodes it is connected to. Thus, its output is an integer in the range $[-15, 15]$, where a high value indicates confidence in class corresponding to the sum node.

Argmax Node

The output of the argmax node is the output of the network. It simply produces the class corresponding to the sum node with the largest output. For example, if the outputs of the sum nodes are given by the vector $[-15, -15, +9, -1, +1, -7, +5, -3, +3, -11]$, then the output of the argmax node is 2 (corresponding to the value $+9$).

For a set of weights and an input x^m , the network computes the results of the perceptron and sum nodes to produce the *predicted* label \hat{y}^m :

$$\hat{y}^m = \operatorname{argmax}_{i \in \{0, \dots, 9\}} \left(\sum_{k=15i}^{15i+14} y^{km} \right).$$

In case multiple sum nodes produce the maximum value, the argmax node breaks the tie by choosing the lowest-numbered of those nodes that are tied.

Classification Accuracy and Training

The goal is to train the binary neural network and maximize the ‘accuracy’ A on the test dataset of M images. Accuracy is defined as:

$$A = \frac{100}{M} \sum_{m=1}^M 1(y^m = \hat{y}^m),$$

where \hat{y}^m is the predicted label of the binary neural network on input x^m , y^m is the true label of x^m , and $1(\cdot)$ is the indicator function, which produces 1 if its argument is true and 0 otherwise.

You are provided with training images with their correct classes and should devise a training algorithm to choose perceptron node weights w^0, w^1, \dots, w^{149} that give good accuracy on the secret testing data.

Training Data and Evaluation Script

Use the links on the top right of the web page for this problem to download the training data to your own computer and a script to help evaluate accuracy on the training data. The training data comes as a text file with one example per line. Line number m has example x^m followed by y^m . Therefore, it has 51 values from $\{-1, 1\}$, followed by a label of 0 to 9.

Input

The program you submit should read no input.

Output and Scoring

Your program must output 150 lines, one per perceptron weight vector, each containing 51 space-separated numbers (either -1 or 1). (If your program produces some other number $v \notin \{-1, 1\}$, the judging software converts it to $\text{sign}(v)$ according to the above definition of sign .)

The judge uses your weights on a different dataset, the test dataset, which consists of 10 000 images. The judge calculates the accuracy of your algorithm using the equation given earlier.

The distribution of image labels in the training data is the same as the distribution in the secret testing data. All 10 digits appear with roughly the same frequency.

Your score on the problem is the accuracy on the test dataset, measured in percent, rounded to two decimal digits of precision.

Motivation

The motivation for using binary neural networks is that they greatly simplify the forward convolution operations and dramatically reduce the on-chip consumed energy (60 times improvement) over more general networks. Therefore, a candidate architecture for performing inference in future mobile phones is to train the binary neural network model on the cloud and then download the weights to the mobile phone for inference. Indeed, in this problem, your algorithm plays the role of the cloud training, and our tester plays the role of the mobile phone inference.

A brute force search would reveal the best weights, but unfortunately there are $2^{150 \cdot 51} = 2^{7650}$ possible weight combinations!