TP 6 Data Mining: 2-layer Neural Network and back propagation

Kieran Schubert

21.05.2019

1 Introduction

In this TP, we will implement backpropagation in a 2-layer Neural Network for image classification on the CIFAR-10 dataset, using the softmax function as an output and the ReLU function for the hidden layer activations. We will derive the analytical gradients in order to use Stochastic Gradient Descent to update the network weights, and compare how well the network performs as a function of different hidden layer sizes (1, 10, 100, 1000).

2 Mathematical Notation

In our problem, we define $j = 1, ..., K$ the class we are refering to, with K the total number of classes (10), y_j the true class of a training example represented in one-hot vector form, S_j our softmax prediction for the j^{th} class and $Z_{2,j}$ the scores:

$$S_j = \frac{e^{Z_{2,j}}}{\sum_{k=1}^{K} e^{Z_{2,k}}}. \tag{1}$$

We define the loss function as the cross-entropy function, which compares the "distance" between distributions, in our case our softmax predictions to the true distribution from the one-hot vector:

$$E = -\sum_{j=1}^{K} y_j ln(S_j). \tag{2}$$

We define the ReLU function as:

$$ReLU(x) = max(0, x). \tag{3}$$

And it's derivative:

$$\frac{\delta ReLU(x)}{\delta x} = \left\{ \begin{array}{ll} 0, & \text{for } x < 0 \\ 1, & \text{for } x > 0 \end{array} \right\}$$

3 Neural Network specification

Our CIFAR-10 dataset is a collection of images belonging to 10 different classes. We divide the dataset in the following way:

- Train data shape: (49000, 3072)
- Train labels shape: (49000, 1)
- Validation data shape: (1000, 3072)
- Validation labels shape: (1000, 1)
- Test data shape: (1000, 3072)
- Test labels shape: (1000, 1)

The network we are implementing consists of an input layer (3072x1), a hidden layer (1x1, 10x1, 100x1, 1000x1 will be tested) and an output layer (10x1). In the following section, we consider a hidden layer size of (10x1). The input layer is connected to the hidden layer by weights, which are stored in a matrix W_1. This matrix is of dimention (3072x10), as each node of the hidden layer is connected to the input layer. We also include a bias term b_1 of dimension (10x1). The hidden layer Z_1 of dimension (10x1) is a linear combination of inputs and W_1 and the bias:

$$Z_1 = W_1^T X + b_1. \tag{4}$$

We introduce non-linearity by passing Z_1 through the ReLU function to obtain a hidden layer node activation $a_1 = ReLU(Z_1)$ of the same dimension. We then perform another linear combination of the activations with the second weight matrix W_2 of dimension (10x10) and a second bias term b_2 of dimension (10x1):

$$Z_2 = W_2^T a_1 + b_2. \tag{5}$$

Z_2 (what we call the scores) is then passed through the softmax function to obtain class probabilities S_j, and our final ouptput is of dimension (10x1). We predict the class based on the maximum probability of the output layer.

4 Gradient Computations

In order to train our Neural Network, we seek to minimize the cross-entropy error by updating the network weights using backpropagation. As the cross-entropy loss is computed on the softmax prediction S_j which arises from function compositions, we use the chain rule to compute all necessary partial derivatives with respect to the cross-entropy error. The process of training consists in computing a forward pass through the network to produce a cross-entropy error, then compute a backwards pass to propagate the error we made (backpropagation) and use the gradients to adjust the network weights using Stochastic Gradient Descent. We repeat this process for a certain number of epochs (number of iterations required relative to batch size to go through the entire training sample) to complete the training process.

In this practical, we were provided the gradient of the error with respect to the scores Z_2 ($\frac{\delta E}{\delta Z_2}$), and the gradient of the error with respect to Z_1 ($\frac{\delta E}{\delta Z_1}$). Using the chain rule, we can obtain the gradients for W_1, b_1, W_2 and b_2.

$$\frac{\delta E}{\delta W_1} = \frac{\delta E}{\delta S_j}\frac{\delta S_j}{\delta Z_2}\frac{\delta Z_2}{\delta a_1}\frac{\delta a_1}{\delta Z_1}\frac{\delta Z_1}{\delta W_1} = \frac{\delta E}{\delta Z_2}\frac{\delta Z_2}{\delta Z_1}\frac{\delta Z_1}{\delta W_1} = \frac{\delta E}{\delta Z_1}\frac{\delta Z_1}{\delta W_1}. \tag{6}$$

Here, $\frac{\delta Z_1}{\delta W_1} = X$, our training dataset. Using the same logic, we obtain:

$$\frac{\delta E}{\delta W_2} = \frac{\delta E}{\delta Z_2}\frac{\delta Z_2}{\delta W_2}. \tag{7}$$

Where $\frac{\delta Z_2}{\delta W_2} = a_1$ And for the bias terms:

$$\frac{\delta E}{\delta b_2} = \frac{\delta E}{\delta Z_2}\frac{\delta Z_2}{\delta b_2}. \tag{8}$$

$$\frac{\delta E}{\delta b_1} = \frac{\delta E}{\delta Z_1}\frac{\delta Z_1}{\delta b_1}. \tag{9}$$

Where $\frac{\delta Z_2}{\delta b_2} = 1$ and $\frac{\delta Z_1}{\delta b_1} = 1$. Having obtained the gradients, we will update the weights and biases using Stochastic Gradient Descent. The general idea is to randomly sample a subset from the training set to speed up computations and iteratively update the parameters in the following way:

$$\omega^{it} = \omega^{it-1} - \gamma\nu\frac{\delta E}{\delta \omega}. \tag{10}$$

This way, we seek to minimize the cross-entropy loss by tweaking the parameters using the information from the gradient. Here, ω is the parameter of interest and ν is the learning rate. We will also reduce the learning rate as the number of iterations increases with a decay parameter γ to allow for convergence. We will not conduct any hyper-parameter tuning but will use the algorithm with the supplied information.

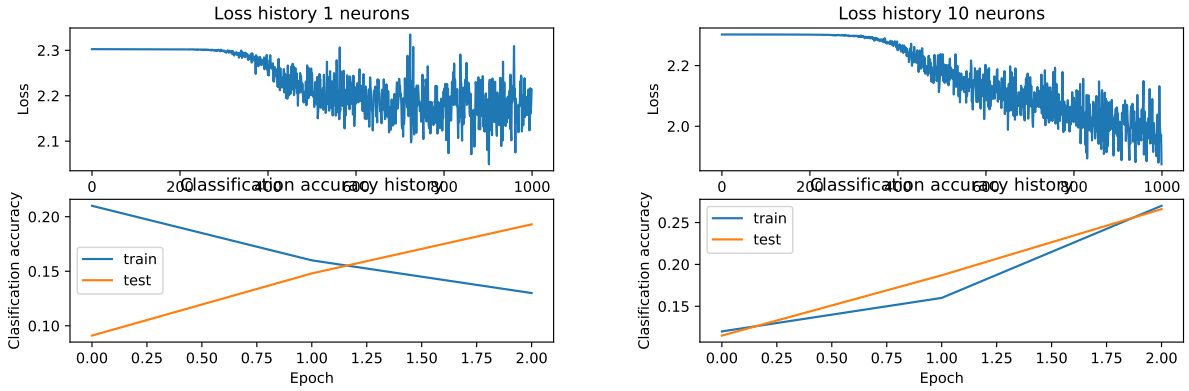
5 Results and Conclusion

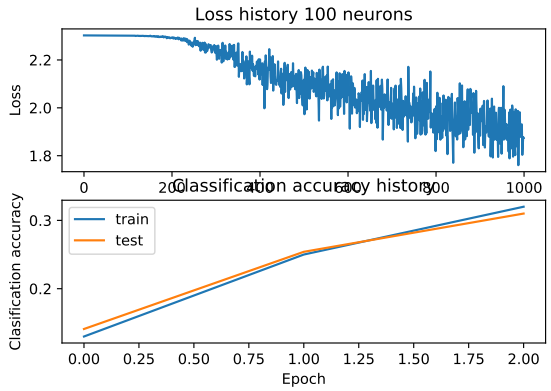
We train our network using 1, 10, 100 and 1000 neurons in the hidden layer and compare classification results as well as training time. For each network, we use the following parameters:

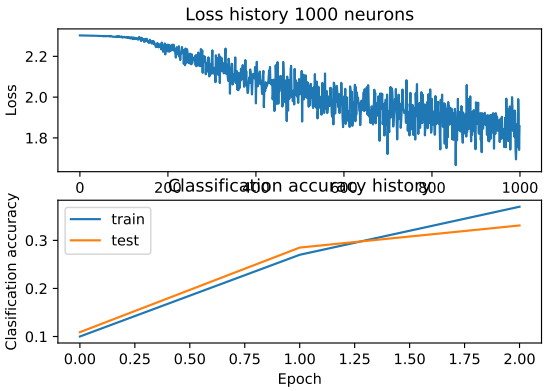
- learning rate ν = 1e-4
- learning rate decay $\gamma = 0.95$
- batch size = 100
- number of iterations = 1000

The following figures show how computation time for training increases from 1.93 seconds (1 hidden neuron), 2.89 seconds (10 hidden neurons), 9.83 seconds (100 hidden neurons) to 82.74 seconds (1000 hidden neurons). We also see that there is an increase in classification accuracy when we increase the number of neurons in the hidden layer:

- 1 hidden neurons has train_acc_history: [0.21, 0.16, 0.13] and test_acc_history: [0.091, 0.148, 0.193]
- 10 hidden neurons have train_acc_history: [0.12, 0.16, 0.27] and test_acc_history: [0.115, 0.187, 0.266]
- 100 hidden neurons have train_acc_history: [0.13, 0.25, 0.32] and test_acc_history: [0.141, 0.254, 0.31]
- 1000 hidden neurons have train_acc_history: [0.1, 0.27, 0.37] and test_acc_history: [0.109, 0.285, 0.331]







The test accuracy increases with the number of hidden neurons to achieve 33% of correct classification with 1000 hidden neurons. Although this result is not remarkable, we note that the CIFAR-10 dataset is a pretty difficult problem for classification and a simple 2-layer neural network may be too naive to achieve good performance. In the future, we may want to increase the number of hidden layers and the number of neurons per hidden layer to increase our model complexity. However, neural network tuning is somewhat heuristic in nature and requires some experience.