IFT725 - Assignment #1

Marc-Alexandre Côté
Département d'informatique
Université de Sherbrooke
Sherbrooke, Québec, Canada
`marc-alexandre.cote@usherbrooke.ca`

Abstract

This version of the report was edited by Hugo Larochelle.

1 Introduction

This report summarizes the work done to accomplish the assignment #1 of the course IFT725. The main goal was to understand how multi-layer neural network works. To do so, we had to code one from a given template. The methods we needed to implement (or complete) were: *initialize*, *fprop*, *bprop*, *training_loss*, *update*, *use* and *test*. The forward propagation, the backward propagation and the training objective are the main parts of this model. This neural network was used in context of a multiclassification problem. The model will be fully detailed in section 2.

Moreover, we were given two scripts to help us debug and test our implementation. The first one is *run_verify_gradients.py*, which checks if the function *bprop* worked as it should given the implementation of *fprop* and indirectly the function *training_loss*. The second script is *run_nnet.py*, which helps us test different configurations for the hyper-parameters used by the neural network.

Once the implementation done, we were asked to provide some results for different configurations of hyper-parameters. Those results show the impact of hyper-parameters on classification error rates. They also report the progression of both the classification error and the average regularized negative log-likelihood for a specific configuration. Finally, the classification error on the test set with a 95% confidence interval for the best configuration based on the classification error of the validation set is given. Those results will be presented in details in section 3.

2 Description of the approach

This section mathematically describes a multi-layer neural network and the learning algorithm used to train it: stochastic gradient. There is five subsections. The first three describes different parts of the model: forward propagation (Section 2.1), the training objective (Section 2.2) and backpropagation (Section 2.3). The fourth subsection explains the stochastic gradient in the context of a neural network (Section 2.4). The fifth details hyper-parameters used by the model 2.5.

But first, a brief reminder of what a neural network model is. The neural network can be seen as an artificial brain's neural network where there are neurons that are connected with each other using axons. Here, in this model[1], neurons are represented in layers $\mathbf{h}^{(k)}$ where a neuron can only communicate with neurons of the layer above $\mathbf{h}^{(k+1)}$ it. Usually the first layer $\mathbf{h}^{(0)}$ is called the *input*, the last layer $\mathbf{h}^{(K)}$ is the *output* and the rest of them are called hidden layers. It is important to note that layers can have different number of neurons. Another thing borrowed from neurology are the axons. In this model, they are represented as matrices $\mathbf{W}^{(k)}$ containing the weights of all links

[1]In equations bold symbols represent a vector/matrix or function returning a vector/matrix.

between neurons of adjacent layers. For example, $\mathbf{W}^{(0)}$ will be a matrix of dimension $|\mathbf{h}^{(1)}| \times |\mathbf{h}^{(0)}|$, where $|\mathbf{x}|$ is the cardinality of the vector $\mathbf{x}$. Because they assure communications between layers, the last matrix of weights is $\mathbf{W}^{(K-1)}$. The last element required by this model is the neuron bias $\mathbf{b}^{(k)}$. The bias represents the threshold needed for a neuron to fire, meaning to propagate its information to neurons of the above layer. There is a biases vector per layer except for the first one since it is the input, so $\mathbf{b}^{(K-1)}$ is the last vector of biases.

Also, we will call Θ the set of all parameters $\boldsymbol{\theta}^{(k)} = \{\mathbf{W}^{(k)}; \mathbf{b}^{(k)}\}$.

2.1 Forward Propagation

In the context of classification, the forward propagation is the process of calculating the output $f(\mathbf{x}; \Theta)$ given an input $\mathbf{x}$ using a neural network model. Initially, the input layer, which is $\mathbf{h}^{(0)} = \mathbf{x}$, will propagate the information to the first layer using the weighted axones linking these two layers $\mathbf{W}^{(0)}$ and the biases of the first hidden layer $\mathbf{b}^{(0)}$. In subsequent layers, a neuron will propagate its information with respect to its activation. So, the summation of all incoming information of a neuron is passed to a function called the activation function $\mathbf{g}(\cdot)$ before the neuron fires to the above layer.

Usually the activation function is nonlinear and can vary from a layer to another. Three activation functions were implemented for this assignment. The first one is the *tanh activation function* (1) which ”squashes” the neuron's input between -1 and 1. The second one is the *sigmoid activation function* (2) which ”squashes” neuron's input between 0 and 1. The last one is used for multi-class classification and is called the *softmax* (3). It gives the probability of having the class c given the input $\mathbf{x}$: $f(\mathbf{x})_c = p(y = c|\mathbf{x})$.

$$\mathbf{g}(\mathbf{x}) = \quad \mathbf{tanh}(\mathbf{x}) \quad = \left[\frac{\exp(2x_1) - 1}{\exp(2x_1) + 1} \cdots \frac{\exp(2x_C) - 1}{\exp(2x_C) + 1}\right] \tag{1}$$

$$\mathbf{g}(\mathbf{x}) = \quad \mathbf{sigm}(\mathbf{x}) \quad = \left[\frac{1}{1 + \exp(x_1)} \cdots \frac{1}{1 + \exp(x_C)}\right] \tag{2}$$

$$\mathbf{g}(\mathbf{x}) = \quad \mathbf{softmax}(\mathrm{x}) \quad = \left[\frac{\exp(x_1)}{\sum_c \exp(x_c)} \cdots \frac{\exp(x_C)}{\sum_c \exp(x_c)}\right] \tag{3}$$

Overall, the general propagation formulas are given as follow:

$$\mathbf{a}^{(k+1)} = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\, \mathbf{h}^{(k)} \tag{4}$$

$$\mathbf{h}^{(k+1)} = \mathbf{g}\left(\mathbf{a}^{(k+1)}\right), \tag{5}$$

where $k \in [0, K)$ and $\mathbf{h}^{(0)} = \mathbf{x}$ and $\mathbf{f}(\mathbf{x}) = \mathbf{h}^{(K)}$. Here, $\mathbf{a}^{(k)}$ represent a neuron before activation and $\mathbf{h}^{(k)}$ after the activation.

In the context of this assignment, the activation function $\mathbf{g}(\cdot)$ for the hidden layers is either $\tanh(\cdot)$ or $\mathrm{sigm}(\cdot)$ depending on an hyper-parameter. But, the activation function for the ouput layer is the $\mathrm{softmax}(\cdot)$.

After applying the forward propagation given an input $\mathbf{x}$, the class y chosen by the model is the one represented by the output neuron which has the highest activation: $y = \mathrm{argmax}(\mathbf{f}(\mathbf{x}))$.

2.2 Training Objective

The training objective, also called the loss function, is a function $l(\mathbf{f}(\mathbf{x}), y)$ that represents some cost associated with the model's predicted conditional distribution $f(\mathbf{x})_c = p(y = c|\mathbf{x})$ for the class, compared to the observed (true) class or target y. In this assignment, the loss function used was the regularized negative log-likelihood:

$$\begin{aligned} l(\mathbf{f}(\mathbf{x}), y) &= -\mathbf{e}(y)^T \ln \mathbf{f}(\mathbf{x}) + \lambda_1 \Omega_1(\Theta) + \lambda_2 \Omega_2(\Theta) && (6) \\ &= -\ln f(\mathbf{x})_y + \lambda_1 \Omega_1(\Theta) + \lambda_2 \Omega_2(\Theta) && (7) \end{aligned}$$

where $\Omega_1(\Theta), \Omega_2(\Theta)$ represent respectively the ℓ^1 and ℓ^2 regularization and are defined by:

$$\Omega_1(\Theta) = \sum_{\boldsymbol{\theta} \in \Theta} \|\boldsymbol{\theta}\|_1 \qquad \Omega_2(\Theta) = \sum_{\boldsymbol{\theta} \in \Theta} \|\boldsymbol{\theta}\|_2^2$$

The goal when training a neural network is to minimize that loss function (6). We used the stochastic gradient descent (see section 2.4), which requires the gradient of the function to minimize w.r.t. each parameters in Θ. Computing them can be done using the chain rule technique. Here are the general formulas of the gradients needed by the backward propagation (Section 2.3):

$$\begin{aligned} \nabla_{\mathbf{h}^{(k)}} - \ln f(\mathbf{x})_y &= \mathbf{W}^{(k)^T} (\nabla_{\mathbf{a}^{(k+1)}} - \ln f(\mathbf{x})_y) && (8) \\ \nabla_{\mathbf{a}^{(k)}} - \ln f(\mathbf{x})_y &= (\nabla_{\mathbf{h}^{(k)}} - \ln f(\mathbf{x})_y) \odot \dot{\mathbf{g}}(\mathbf{a}^{(k)}) && (9) \\ \nabla_{\mathbf{W}^{(k)}} - \ln f(\mathbf{x})_y &= (\nabla_{\mathbf{a}^{(k)}} - \ln f(\mathbf{x})_y)\, \mathbf{h}^{(k-1)^T} + \lambda_1 \mathbf{sign}(\mathbf{W}^{(k)}) + \lambda_2 2\mathbf{W}^{(k)} && (10) \\ \nabla_{\mathbf{b}^{(k)}} - \ln f(\mathbf{x})_y &= (\nabla_{\mathbf{a}^{(k)}} - \ln f(\mathbf{x})_y) && (11) \end{aligned}$$

Where $\odot$ is the element-wise product. From that, we can easily compute the gradient of the loss function w.r.t. the output layer K:

$$\begin{aligned} \nabla_{\mathbf{h}^{(K)}} - \ln f(\mathbf{x})_y &= \frac{-\mathbf{e}(y)}{f(\mathbf{x})_y} && (12) \\ \nabla_{\mathbf{a}^{(K)}} - \ln f(\mathbf{x})_y &= -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x})_y) && (13) \end{aligned}$$

The derivative of $\mathbf{g}(\cdot)$ are:

$$\begin{aligned} \dot{\mathbf{g}}(\mathrm{x}) = \quad & \mathbf{tanh}'(\mathrm{x}) && = 1 - \mathbf{tanh}(\mathrm{x})^2 && (14) \\ \dot{\mathbf{g}}(\mathrm{x}) = \quad & \mathbf{sigm}'(\mathrm{x}) && = \mathbf{sigm}(\mathrm{x})(1 - \mathbf{sigm}(\mathrm{x})) && (15) \\ \mathbf{g}(\mathrm{x}) = \quad & \mathbf{softmax}'(\mathrm{x}) && = \mathbf{softmax}(\mathrm{x})(1 - \mathbf{softmax}(\mathrm{x})) && (16) \end{aligned}$$

2.3 Backward Propagation

The backward propagation is a way of calculating the gradients needed to minimize the training objective function (described in section 2.2). It takes place after having performed a forward propagation on a given input $\mathbf{x}$ associated to a target y. This technique works by computing the gradient of the output layer before its activation using equation (13). Then using equations from (8) to (11), we compute the gradients of all remaining hidden layers. The algorithm for the backward propagation goes like this:

1. Compute the output gradient before activation using equation (13):
$$\nabla_{\mathbf{a}^{(K)}} - \ln f(\mathbf{x})_y \Leftarrow -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x})_y)$$
2. Next, for k from $K-1$ to 0:
   (a) Compute gradients of parameters using equations (10) and (11):
$$\nabla_{\mathbf{W}^{(k)}} - \ln f(\mathbf{x})_y \Leftarrow (\nabla_{\mathbf{a}^{(k)}} - \ln f(\mathbf{x})_y)\mathbf{h}^{(k-1)^T}$$
$$\nabla_{\mathbf{b}^{(k)}} - \ln f(\mathbf{x})_y \Leftarrow (\nabla_{\mathbf{a}^{(k)}} - \ln f(\mathbf{x})_y)$$
   (b) Compute gradient of layer below using equations (8) and (9):
$$\nabla_{\mathbf{h}^{(k)}} - \ln f(\mathbf{x})_y \Leftarrow \mathbf{W}^{(k)^T} (\nabla_{\mathbf{a}^{(k+1)}} - \ln f(\mathbf{x})_y)$$
$$\nabla_{\mathbf{a}^{(k)}} - \ln f(\mathbf{x})_y \Leftarrow (\nabla_{\mathbf{h}^{(k)}} - \ln f(\mathbf{x})_y) \odot \dot{\mathbf{g}}(\mathbf{a}^{(k)})$$

2.4 Stochastic Gradient

The stochastic gradient descent is an algorithm to find (possibly local) minima in a function. We need a special algorithm like this one because the function we try to minimize, the training objective, is higly non-convex. The main idea is to perform a gradient step using only a single training example $(\mathbf{x}^{(t)}, y^{(t)})$ and update the parameters. Then we do another gradient step using the next training example $(\mathbf{x}^{(t+1)}, y^{(t+1)})$ and so on, updating the parameters after each step.

The general algorithm repeats the following steps, for every training example $(\mathbf{x}^{(t)}, y^{(t)})$:

$$\begin{aligned} \mathbf{\Delta} &= -\nabla_{\Theta} l(f(\mathbf{x}^{(t)}; \Theta), y^{(t)}) \qquad (17) \\ \Theta &= \Theta + \alpha \mathbf{\Delta} \qquad (18) \end{aligned}$$

where α is the learning rate. Before training starts, the parameters of the neural network must be initialized. All biases are initialized to 0. The weight matrices can't be initialized to 0, otherwise it can be shown that all gradients will always be equal to 0 (i.e. it corresponds to a saddle point of the optimization problem). The weight matrices also must not be initialized to the same value, otherwise it can be shown that the hidden units in a given layer will alway behave exactly the same (i.e. their incoming weights will remain the same throughout training). So, we opt for a randomized initialization close to 0, where each weight $W_{i,j}^{(k)}$ is sampled from a uniform distribution in the interval $[-b, b]$, where $b = \sqrt{6/(|\mathbf{h}^{(k)}| + |\mathbf{h}^{(k+1)}|)}$.

In the case of training a neural network, equation (17) is equivalent to performing a step of forward propagation (Section 2.1) followed by a step of backward propagation (Section 2.3). We then use the gradients calculated by the backward propagation to update each parameter of Θ.

2.5 Hyper-parameters

Here are a short description of all hyper-parameters used by the model.

lr is the learning rate α. If too small, stochastic gradient descent will take more time before converging. If too big, it could diverge.

dc is the decrease constant δ for the learning rate. It could be wise to start with high learning rate, but reduce it as it converges. A way to do it, is to use a decrease factor. For example, in this assignment to technique used is: $\alpha_t = \frac{\alpha}{1+\delta t}$, where t is the number of updates.

sizes is the list of hidden layer sizes. The size of the list is the number of hidden layers and each element corresponds to the number of neurons on a specific hidden layer.

L2 is the L2 regularization weight λ_2 (weight decay). Controls the weight given to the ℓ^2 norm of the regularization terms in the training objective function (see section 2.2).

L1 is the L1 regularization weight λ_1 (weight decay). Controls the weight given to the ℓ^1 norm of the regularization terms in the training objective function (see section 2.2)

seed is the seed of the random number generator. It makes it possible to reproduce the results of a given experiment and to vary the specific randomized initialization used.

tanh is a boolean indicating whether to use the hyperbolic tangent activation function (True) instead of the sigmoid activation function (False). In other words, it tells the algorithm whether to use equation (1) or equation (2) as activation function for neurons on hidden layers.

n_epochs is the number of training epochs. This hyper-parameter is not used in this assignment because the early stopping technique is used (not explained in this report, but more information can be found in [3]).

Table 1: Results of the training and validation errors of 20 different configurations of hyper-parameters

lr	dc	sizes	L2	L1	tanh	train	valid
0.01	1e-5	[100]	0	0	True	0.0570	0.1263
0.001	1e-10	[100]	0	0	False	0.0822	0.1283
0.001	1e-10	[100]	0	0	True	0.0631	0.1284
0.001	1e-7	[100]	0	0	True	0.0669	0.1287
0.001	1e-7	[100]	0	0	False	0.1012	0.1381
0.01	1e-7	[100]	0	0	True	0.0750	0.1393
0.01	1e-5	[100]	0.001	0	True	0.1381	0.1610
0.001	1e-7	[100]	0.001	0	True	0.1393	0.1636
0.01	1e-10	[20]	0	0	False	0.1651	0.1927
0.1	1e-5	[20,10]	0.001	0	True	0.1786	0.1946
0.001	1e-7	[20]	0	0	True	0.1688	0.1952
0.001	1e-7	[20,10]	0.001	0	True	0.1797	0.2003
0.001	1e-10	[20]	0	0	False	0.1768	0.2008
0.001	1e-10	[100]	0.001	0.001	True	0.2243	0.2381
1	1e-5	[100]	0.001	0	False	0.2513	0.2619
0.001	1e-5	[100]	0	0.001	False	0.3601	0.3727
0.001	1e-5	[20]	0.001	0.001	False	0.3994	0.4041
1	1e-7	[20]	0.001	0.001	True	0.9939	0.9917
1	1e-10	[20]	0.001	0.001	True	0.9939	0.9917
1	1e-10	[20,10]	0.001	0	True	0.9939	0.9917

3 Experiments

This section presents the results of some experiments made while testing different configurations of hyper-parameters. The metric used to compare those results is the classification error which is essentially the percent of example of a dataset that have been misclassified.

The table 1 shows the training and validation classification errors of 20 different configurations of hyper-parameters. The table is sorted by the validation error. It is worth mentioning that the selected configurations are not the ”Top Best 20” but rather a manual selection to see the impact of some hyper-parameters. Clearly, we see that having a learning rate of 1 while not having a high decrease constant tends to diverge. We can also see that the regularization is not really useful here and that having only one layer with a lot of neurons is preferred. The reason behind the non-impact of regularization is related to the fact we are using an early stopping technique, which might be seen has a kind of regularization.

In addition, figure 1 shows two plots reporting the progression of the classification error (figure 1(a)) and the average regularized negative log-likelihood (figure 1(b)) on the training and validation sets for the hyper-parameter configuration with the best performance on the validation set.

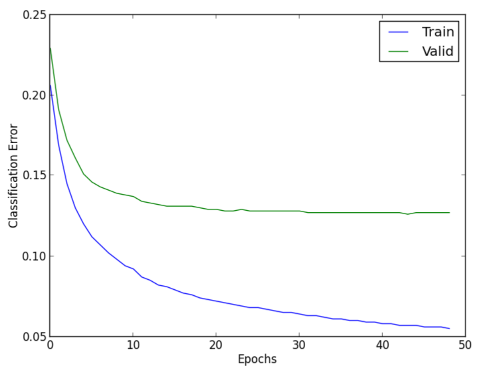
The classification error on the test set for the hyper-parameter configuration with the best performance on the validation set is: 0.1208 ± 0.0064 with a 95% confidence interval. Also, the average negative log-likelihood is 0.4182. The hyper-parameters configuration is shown as the first row in the table 1.

4 Conclusion

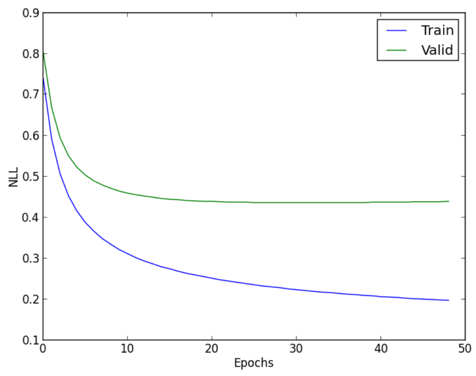
This report mathematically described the multi-layer neural network model used in the context of multi-class classification problem. Also, it reported results obtained from different configuration of hyper-parameters.

References

[1] Larochelle, Hugo (2012) Review of fundamentals, *IFT725 - Réseaux neuronaux*, UdeS.



(a) Classification error



(b) Regularized Negative Log-Likelihood

Figure 1: Progression of the classification error (a) and the average regularized negative log-likelihood (b) on the training and validation sets for the hyper-parameter configuration with the best performance on the validation set.

[2] Larochelle, Hugo (2012) Feedforward neural network, *IFT725 - Réseaux neuronaux*, UdeS.

[3] Larochelle, Hugo (2012) Training neural networks, *IFT725 - Réseaux neuronaux*, UdeS.