Neural network with hidden layers implementation

Taycir Yahmed
taycir.yahmed@telecom-paristech.fr

December 2017

1 Introduction

Deep neural networks represent an effective and universal model capable of solving a wide variety of tasks. In this report I will implement a simple n-layer neural network from scratch. I won't derive all the math that's required, but I will try to give intuitive explanations. But why implement a Neural Network from scratch at all? Even if we plan on using Neural Network libraries like *pytorch* or *tensorflow*, implementing a network from scratch at least once is an extremely valuable exercise. It helps us gain an understanding of how neural networks work, and that is essential for designing effective models.

2 Datasets

This report comes with code to load various datasets. You can find a dataset loader in the python file named *dataset_loader.py*. Here, we will be mainly working with the following two datasets.

2.1 MNIST handwritten digits

The MNIST database is a large database of handwritten digits that is commonly used for training and testing in the field of machine learning. It contains 60,000 training images and 10,000 testing images. There have been a number of scientific papers on attempts to achieve the lowest error rate on this dataset.

2.2 CIFAR-10 images

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order. Between them, the training batches contain exactly 5000 images from each class: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.*

3 Goal

Our goal is to train a machine learning classifier that predicts the correct class given the x and y coordinates: a typical supervised learning task.

4 Softmax regression

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. In logistic regression we assume that the labels were binary: y(i)∈{0,1}. We use such a classifier to distinguish between two kinds of hand-written digits. Softmax regression allows us to handle y(i)∈{1,. . . ,K} where K is the number of classes. Moreover, we can think of logistic regression as a two-layer neural network model. The figure below explains the corresponding architecture.

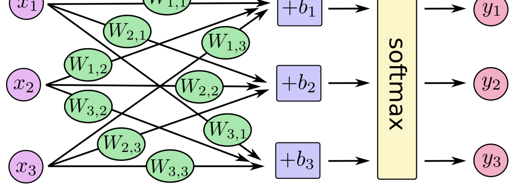

Figure 1: Softmax regression

4.1 Softmax regression on MNIST dataset

First of all, I tested the implementation with a neural network with no hidden layers, which corresponds to a softmax regression, on the MNIST dataset. You can find below the results for various scenarios:

Important remark: *for all the experiments presented in this report, I use 20 epochs, a λ value of 1e-4 (used in the regularization), an η value of 0.01 with an adaptation rate equal to 0.2, relu as an activation function and the MNIST dataset (except or the case where it is explicitly stated otherwise). Also, the accuracy stated below is calculated on the validation set in the last (20th) epoch.*

lr adaptation	regularization	accuracy
no	none	0.918
no	l1	0.9183
no	l2	0.9181
yes	none	0.922
yes	l1	0.9249
yes	l2	0.921

Table 1: Softmax regression results

We notice that the results are fairly comparable. Indeed, we have approximately the same performance when varying the regularization approach. However, using the learning rate adaptation slightly enhances the results.

Influence of the learning rate adaptation: Learning rate adaptation consists of changing the value of the learning rate from one epoch to another. So, if the loss of the current epoch is higher than that of the previous one, we slightly decrease it (by a factor of 0.8 for instance). Otherwise, we increase it a bit (by a factor of 1.2 for instance).

We can visualize below the learning curves for the best performing scenario (using learning rate adaptation and l1 regularization).

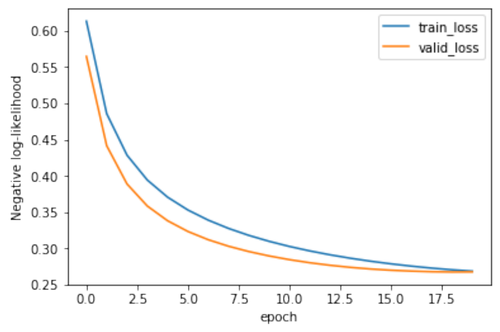

Figure 2: The negative log-likelihood

With the negative log-likelihood being defined as the following :

$$c = -\sum_{(x_i,y_i)\in\mathcal{D}} \log P(y = y_i | x_i, W, b)$$

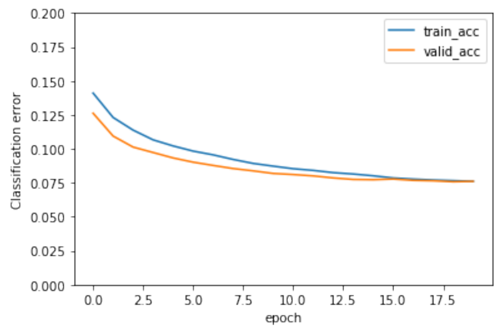

Figure 3: The classification error

These graphs ensure a basic sanity check: We can easily check that the negative log-likelihood is decreasing, as well as the classification error.

5 Neural network with hidden layers

The implementation that comes with this report can be used with any desired design of neural networks hidden layers. You can simply modify the variable *n_hidden* to specify the architecture you want to consider.

5.1 Three-layer neural network

Let's now build a 3-layer neural network with one input layer, one hidden layer, and one output layer. The number of nodes in the input layer is determined by the dimensionality of our data,

784. Similarly, the number of nodes in the output layer is determined by the number of classes we have, 10. The input to the network will be x and y coordinates and its output will be 10 probabilities, one for each class.

Choosing the size of the hidden layer: We can choose the dimensionality (the number of nodes) of the hidden layer. The more nodes we put into the hidden layer the more complex functions we will fit. But higher dimensionality comes at a cost. How to choose the size of the hidden layer? While there are some general guidelines and recommendations, it always depends on our specific problem and it is more of an art than a science.

In the experiments mentioned below, I choose a hidden layer size of **100 nodes**, with different combination of learning rate adaptation and regularization approaches.

lr adaptation	regularization	accuracy
no	none	0.9509
no	l1	0.9505
no	l2	0.9509
yes	l2	0.9685
yes	none	0.9754
yes	l1	0.9655

Table 2: Three-layer network results

We notice a general increase in performance when using a 3-layer neural network. For all the scenarios, the performance is higher, namely for the case where we use the learning rate adaptation and no regularization. Below, you can find the learning curves for this scenario.

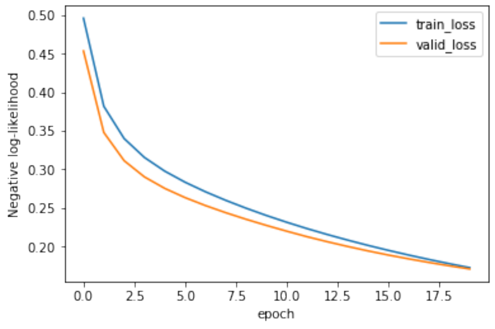

Figure 4: The negative log-likelihood

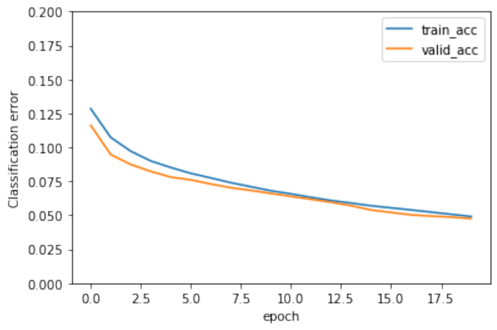

Figure 5: The classification error

5.2 Multiple-layer neural network

Multi-layered networks consist of layers of several networks, where nodes appear in at least one of these layers. The networks are connected by inter-layer links. In this report, I test a 4-layer network with a size of 128 and 64 nodes in the hidden layers, respectively. Below, you can find the results:

lr adaptation	regularization	accuracy
yes	l1	0.9609
yes	l2	0.9811
yes	none	0.9305

Table 3: Multiple-layer network results

We can see that using a 4-layer neural network with learning adaptation gives better results than the previously seen, with a performance over 0.98.

Influence of the regularization: Given the previous results, we can see that using the regularization increases the performance. Indeed, regularization can be motivated as a technique to improve the generalizability of a learned model.

We can also note that there is a slight tendency to overfit the data as the network grow deeper (with only 2 hidden layers). The below figures illustrate that.

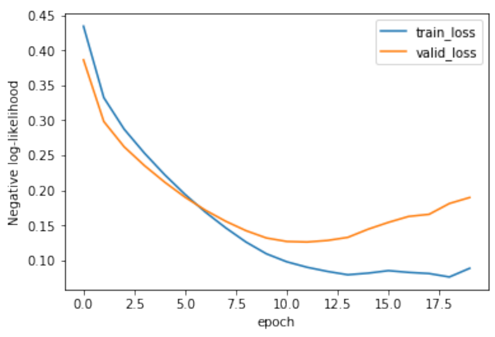

Figure 6: The negative log-likelihood

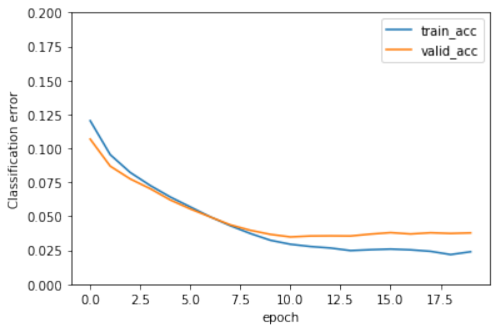

Figure 7: The classification error

We can see that the training error decreases while the validation error increases, which indicates an overfitting of the training data.

Choosing the activation function: We also need to pick an activation function for our hidden layers. The activation function transforms the inputs of the layer into its outputs. A nonlinear activation function is what allows us to fit nonlinear hypotheses. Common choices for activation functions are tanh, the sigmoid function, or relu. I used relu with **clipping**, which performs quite well in many scenarios. A nice property of these functions is that their derivate can be computed using the original function value (which I considered in the implementation).

Because we want our network to output probabilities the activation function for the output layer will be the softmax, which is simply a way to convert raw scores to probabilities.

5.3 Test on Cifar-10 dataset

When testing the code on the Cifar-10 dataset with a neural network of 4 layers (2 hidden layers of sizes 64 and 128), I notice that it doesn't give a good performance: 45%. This is due to the fact this dataset is highly complex for a simple 2-hidden-layer network to learn.

6 Conclusion

We noticed during various experiments that neural networks can easily overfit the training data, leading to lower performance on the validation set. One of the most commonly used ways to avoid/solve this problem is to use the **dropout** technique. Indeed, dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term ”dropout” refers to dropping out units (both hidden and visible) in a neural network. A possible improvement of this implementation would be to add the possibility of using dropout.