Computer Science Department, Ben Gurion University of the Negev

Practical Deep Learning - Spring 2021

Home Assignment: Deep Learning from Scratch

Due on 15/01/2021.

1 Project Description

The goal of this project is to gain an idea of how deep learning frameworks work behind the scenes (the back-prop, optimization, influence of hyper parameters), and to gain experience in scientific programming. This will help you understand how to extend standard libraries when certain modules you need do not exist. A secondary (equally as important) goal of this project is to provide the methodologies and gain experience in the development of complex optimization frameworks with a lot of moving parts.

In this project we will implement a simple neural network for classification of small vectors. We will be given with "training" labeled data samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ and another set of "validation" data $\{(\mathbf{x}_i^v, y_i^v)\}_{i=1}^{m_v}$. The goal is to define and train a neural network that knows how to classify the data samples (given some sample $\mathbf{x}$, predict its label y). You will define some structure of a network (called architecture) which has unknown weights. Then, you will train the network to classify the data samples by optimizing an objective function that tries to fit the predictions with the given labels.

Our network will have the following architecture:

$$L(\{\theta_l\}_{l=1}^{L}) = \frac{1}{m}\sum_{i=1}^{m}\ell(\theta^{(L)}, \mathbf{x}_i^{(L)}, y_i) \quad (1)$$

$$s.t \qquad \mathbf{x}_i^{(l+1)} = f(\theta^{(l)}, \mathbf{x}_i^{(l)}); \quad l = 1, ..., L-1 \quad (2)$$

where $\ell()$ is the softmax objective function defined in the lectures, and $\mathbf{x}_i^{(1)}$ denotes the given data sample i. The architecture be either a standard Neural Network

$$f(\theta^{(l)}, \mathbf{x}_i^{(l)}) = \sigma(\mathbf{W}^{(l)}\mathbf{x}_i^{(l)} + \mathbf{b}^{(l)});$$

or a simple ResNet (you may add an activation at the end or beginning of this block.)

$$f(\theta^{(l)}, \mathbf{x}_i^{(l)}) = \mathbf{x}_i^{(l)} + \mathbf{W}_2^{(l)}\sigma(\mathbf{W}_1^{(l)}\mathbf{x}_i^{(l)} + \mathbf{b}^{(l)});$$

We will denote by $\theta^{(l)}$ the set of weights of layer l that we need to learn. For the last layer L we have to learn the softmax weights:

$$\theta^{(L)} = \left\{ \{\mathbf{w}_j^{(L)}\}_{j=1}^{n_{labels}} \in \mathbb{R}^n, \text{ and the bias vector } \mathbf{b}^{(L)} \in \mathbb{R}^{n_{labels}}. \right\}$$

For each data sample $\mathbf{x}_i$, the predicted label is the label with highest probability in the softmax function.

For the standard layers we have

$$\theta^{(l)} = \{\mathbf{W}^{(l)} \in \mathbb{R}^{n_2 \times n_1}, \mathbf{b}^{(l)} \in \mathbb{R}^{n_2}\},$$

where n_1 and n_2 may differ. In particular, it is common (and necessary) to enlarge the width of the network (the size of the vectors $\mathbf{x}_i^{(l)}$ as the layers progress) as the For the residual layers $l = L-1, ..., 1$ we have

$$\theta^{(l)} = \{\mathbf{W}_1^{(l)} \in \mathbb{R}^{n \times n}, \mathbf{W}_2^{(l)} \in \mathbb{R}^{n \times n}, \mathbf{b}^{(l)} \in \mathbb{R}^n\},$$

which needs to be learned.

2 Specific Tasks

This project is large and has many moving parts. Below are the steps that you need follow on your way to the big goal: a trained network that can classify in high accuracy. All the steps should be submitted in a report, and you get part of the grade for each of them.

2.1 Part I: the classifier and optimizer (40%)

1. Write the code for computing the loss function "soft-max regression" and its gradient with respect to $\mathbf{w}_j$ and the biases. Make sure that the derivatives are correct using the gradient test (See the subsection "Gradient and Jacobian Verification" in the notes). You should demonstrate and submit the results of the gradient test.

2. Write the code for minimizing an objective function using SGD or some other SGD variant (SGD with momentum, for example).

3. Demonstrate the minimization of the softmax function using your SGD variant. Plot a graph of the success percentages of the data classification after each epoch—for both the training data and the validation data. You may only randomly subsample the two data sets for the plots as these are extra computations that are not related to the optimization itself. Try a few learning rates and min-batch sizes and see how they influence the performance (submit the graphs only for your best option, but also write about your tries). Run as many iterations of SGD as you see that is needed (i.e., more iterations do not improve the accuracy, even if the learning rate decreases).

2.2 Part II: the neural network (60%)

1. Write the code for the standard neural network. Including the forward pass and backward pass (the computation of the "Jacobian transpose times vector"). See that the Jacobian tests work and submit the tests. **This part should not be overlooked** Remark: for the Jacobian tests use the tanh() activation function, as it is differentiable and will behave properly in the gradient tests. The ReLU function is piecewise linear and non-smooth and may lead to weird-looking gradient tests. After the gradient test passes, you may use either activation functions in the network.

2. (Elective) Repeat the previous section for the residual neural network.

3. Compute a forward pass and a backward pass of a network with L layers (L is a parameter). See that the gradient of the whole network (softmax + layers) passes the gradient test. **Submit this verification.**

4. Repeat section 3 for the entire network. Try a few network lengths and see how this influences the performance. Write your conclusions and demonstrate them.

5. (Elective) In some cases you'll encounter situations where the training accuracy is improving but the validation accuracy is deteriorating. One common reason for this is that the network "overfits" the training data, which may happens when you'll use too many parameters. In such a case - try to add some regularization term to your network and see if you can prevent the overfitting or just improve the results (if you try and don't succeed - also report that). A default choice would be the squared ℓ_2 norm of all the weights in the system, but you may be more creative with this.

6. (Elective) As an alternative to the previous section, you may explore different optimization strategies. For example, separating the optimization of the classifier and the network (making alternating SGD steps on each, for example). Training layer by layer can be another option. Trying different weighting for the gradient etc. Try to be creative and see if you can do better than SGD. Creative tryouts which fail to beat SGD (but work) are also welcome.

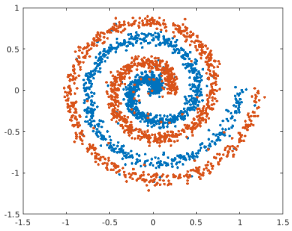
7. (Elective) If you opt to include the residual network in your code - compare between the optimization process of a standard network and a network that most of its steps are residual steps. Use about 10-20 steps in your network, and use similar vector sizes between the two options.

3 Data sets

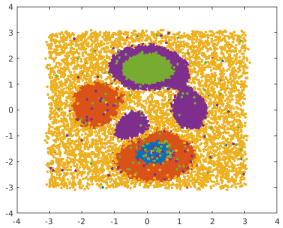
Run your experiments in the sections above on one of the first two data sets where the dimension $n = 2$ (either option 1 or 2) and on the the data set where $n = 5$ (option 3). For

each data set mat file, the matrices `Yt,Ct` represent the training data and `Yv,Cv` represent the validation data.

1. A two dimensional data "SwissRoll" with two classes (colored red and blue).



2. A two dimensional data "Peaks" with 5 classes (five colors).



3. A five dimensional data "GMM" data with 5 classes (five colors). For illustration, the figure below has $n = 3$, but in the actual data $n = 5$.