Middle East Technical University — Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Fall 2017-2018

Take Home Exam 2

Due date: **29.11.2017 - 23:59**

1 Objectives

To familiarize yourselves with artificial neural networks for regression problem through implementing and applying modular artificial neural networks on datasets, which you may have to perform **data preprocessing** on, and with analysis of error history.

2 Specifications

This homework invites you to implement a modular artificial neural network framework, and experiment with it on the given datasets, which will be described in detail subsequently.

Training datasets for this take home exam are provided in the files **set1.dat** and **set2.dat**. You can find a function named *read_data* for reading the data from files, in *data.py* within the provided base code. In the same file, you are expected to implement a feature scaling method such as normalization, minmax scaling, standardization etc. That function returns the scaled training and test examples. In your experiments, you are **NOT** allowed to use test examples in training part. You need to divide the training examples in to a training and validation set according to your validation algorithm.

In this assignment, your aim is to implement a modular artificial neural network class using the provided base code. You are required to complete missing parts in the code which are explained in the comments. You are provided with some test functions that you can utilize while developing different layers of the network.

Homework file continues with a review of theory behind artificial neural networks and the steps you should take into consideration for your implementation of the task. Assume from now on that you are given the training data set $T = \{(\mathbf{x}_i, y_i) \,|\, i \in [1..N]\}$, in which each $\mathbf{x}_i$ is an d-dimensional multivariate vector, and each y_i denotes the ground truth value of target (dependent) variable of the regression task.

2.1 Affine Layer

This is the well known fully connected layer which will be used as hidden layers in the regression network. In order to implement the affine layer you should complete the *affine_forward* and *affine_backward* functions in *layers.py*. You can use *test_affine_forward* and *test_affine_backward* functions to test your implementation. The error values should be less than 10^{-6}. Equation (1) gives the forward propagation of an affine layer

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}, \tag{1}$$

where $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias term, $\mathbf{x}$ is the input vector, and $\mathbf{o}$ is the output vector that holds the output of each neuron in the layer.

2.2 ReLU Layer

This is the rectified linear unit layer which will be used as activation function in the hidden layers. In order to implement the ReLU layer you should complete the *relu_forward* and *relu_backward* functions in *layers.py*. You can use *test_relu_forward* and *test_relu_backward* functions to test your implementation. The error values should be less than 10^{-6}. Equation (2) gives the forward propagation of a rectified linear unit.

$$a = \begin{cases} o & \text{if } o > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

where o is the output of a neuron, and a is the activation of that neuron.

2.3 Loss Layer

This will be the loss layer of the network where many loss functions can be implemented according to the purpose. The most suitable loss function for the regression task is the L2 loss which is also known as mean squared error. In order to implement the loss layer you should complete the *L2_loss* function in *layers.py*. You can use *test_L2_loss* function to test your implementation. The error values should be less than 10^{-6}. Equation (3) gives the L2 loss function for the training set.

$$L = \frac{1}{N} * \sum_{i=1}^{N} \frac{1}{2}(y_i - \hat{y}_i)^2, \tag{3}$$

where $\hat{y}_i$ is the prediction, y_i is the ground truth value for training example x_i, and N is the number of training examples.

2.4 Artificial Neural Network

Artificial Neural Networks are models inspired from biological neurons, and their network that we call brain. They are capable of representing any function if enough capacity is provided and trained wisely. In this part of the take home exam, you will implement a regression network using the layers described in previous sections. The general architecture of the network will be the input layer followed by L-1 affine layer-ReLU layer pairs, an affine layer and a loss layer. In this architecture there will be L hidden layers which may vary in different experiments.

The implementation will be done in the provided *ann.py* file. For all the functions that you are required to complete, there are comments in the base code. Follow those comments to understand the details. In the *__init__* method, you should create the parameters of the layers to construct the neural network architecture provided in the arguments. In the *loss* method you should implement the forward and backward passes of the network. If the labels are provided the function computes the loss and gradients of it with respect to all parameters. Otherwise, it computes only the predictions and returns them. In the forward pass, a sample should pass through all of the layers and a prediction should be computed using the combination of those fully connected and ReLU layers. After computing the loss and its derivative with respect to the prediction, the backpropagation algorithm should be applied in the backward pass.

Assume that the network has the architecture,

$$\text{input}, A_{W_0,b_0}, R_0, A_{W_1,b_1}, R_1, A_{W_2,b_2}, \text{loss}.$$

In the forward pass, the network computes o_0, a_0, o_1, a_1, o_2 as the outputs of the layers. The following equation gives the backward pass for the parameter W_0, and the computation of remaining derivatives is your responsibility.

$$\frac{\partial L}{\partial W_0} = \frac{\partial L}{\partial o_2}\frac{\partial o_2}{\partial a_1}\frac{\partial a_1}{\partial o_1}\frac{\partial o_1}{\partial a_0}\frac{\partial a_0}{\partial o_0}\frac{\partial o_0}{\partial W_0}.$$

If you write the derivatives with respect to other parameters, you will see that there are many common parts. In order not to compute same expressions in all computations, saving them once computed is highly recommended. Recomputation will end-up with long training time which will make doing experiments difficult.

After all, you should implement an optimization algorithm to train the network in *train_validate* method. In THE1, you implemented traditional gradient descent algorithm. It is still applicable here. You can make research about optimization methods and apply some other techniques. If you do so, explain the method and its advantages in the report. The signature of the function is prepared according to gradient descent algorithm. If you implement another one, you can change the signature. After training, this method returns the training and validation loss histories so that the experimenter can investigate the analyze the change of loss with respect to training epochs. The trained network can be used with the readily implemented *predict* method of the class.

2.5 Programming and Interpretation Tasks

First, you need to complete all the base code to implement a modular artificial neural network. While completing the code, do not change the signatures of the functions/methods except for the optimization method.

In order to start gradient descent algorithm we need an initial parameters which are chosen to be comprised of small random numbers in general. You will follow this manner with a small difference which will make your results reproducible and comparable. Before starting each experiment set the random number generator's seed to "499" and get a random parameters. You can use *numpy.random.seed(499).*

You are given two different datasets. For each of them, train networks with a single hidden layer, two hidden layers and more than two hidden layers. While training try to optimize the hyperparameters such as network architecture, learning rate, number of epochs or other hyperparameters if you use another optimization method then traditional gradient descent. For each configuration, do your best with justifications in the report.

Before training, divide your data into training and validation parts meaningfully and describe your justification for this division in your report. Note that you will use the first element of the tuple returned by *read_data* function for training and validation sets. The remaining part will be used for testing. While training the regression network, keep histories of loss value. Draw plots showing the change in loss. After training ends, predict the target values for the test data. Compute the minimum, maximum and average absolute distance with the ground truth, and compare those distances with the target scale. Discuss overfitting and underfitting issues, convergence time and local/global minimum issues. Note that, in order to get some meaningful information from the anlysis tools (the plots and the differences), you should do considerable amount of updates. Then, discuss the time point where training should be stopped. In order to increase the accuracy, you may save the trained weights, load them to the network and retrain with new hyperparameters.

The following bullet points summarize the key points that you should address in your report.

- The feature scaling technique that you utilized, and the effects of it
- Discussion on the different architectures and their results on the datasets
- Detailed explanation about the optimization method if you implement something different than traditional gradient descent
- Detailed explanation of the decision about when to stop updates
- Detailed explanation of how you choose the training and validation sets
- Interpretations of the analysis tools and discussion on overfitting, underfitting, learning capacity of the network architectures compared to dataset complexity
- Any other detail that needs to be explained

3 Restrictions and Tips

- Do not use any available Python repository files without referring to them in your report.
- Toolbox and library function use is not allowed in this homework. Do not use any ML-related toolbox/library. However, you may cross-check your results utilizing toolboxes. Your homework submission must not include any high-level toolbox/library function. Since the base code depends on NumPy, it is not restricted. Likewise you can use any library for plots.
- If you encounter trouble regarding vectorization, first try to implement the tasks by using loops. Vectorization is required, since typical ML projects deal with massive amounts of data. If at the end you fail to vectorize your code, submit the current version. Although vectorization appears to be essential in Python-ML programming, since this is not a Python course, unvectorized versions will only result in a minor decline (at most 10%) in the grade you are going to receive.
- Implementation should be of your own. Readily-used codes should not exceed a reasonable threshold within your total work. In fact you shouldn't need any code on repositories.
- Don't forget that the code you are going to submit will also be subject to manual inspection.

4 Submission

- **Late Submission:** You have a total of 3 days for all the homeworks with a penalty of 10 points for each day.
- Your implementation files together with a 3-to-4 pages long report (prepared according to provided template) focusing on theoretical and practical aspects you observed regarding this task should be uploaded on COW before the specified deadline as a compressed archive file whose name should be **<student_id>_the2.tar.gz**, e.g. 1234567_the2.tar.gz.
- The archive must contain **no directories** on top of implementation files.
- Do not send the datasets in the archive. Evaluation will be done using the original datasets.

5 Regulations

1. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.

2. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.