Deep Learning - H.W 2

May 2, 2018

1 Submission Instruction

1. Submission deadline: 15/05/2018 23:59
2. TA in charge: Yonatan Geifman
3. Submit a zip file containing only your code and the written answers, in the webcourse site: http://webcourse.cs.technion.ac.il/236606/
4. Answers must be submitted as a PDF file, typed using any document editor (not a scan of handwritten answers)
5. Submission is in pairs only
6. No late submissions

2 Introduction

In this homework your goal is to implement a (restricted) deep neural network package using Python and Numpy only. The implementation will be based on the building blocks we have learned so far.

In general, your solution should be able to construct, train and apply any user-defined deep architecture containing any number of fully-connected layers whose activation function can be ReLu, Sigmoid and/or Softmax functions (Softmax will be covered in Lecture 5). Training will be implemented using stochastic gradient descent (SGD) with either the cross-entropy loss or mean squared error, and will be applied on either full or mini-batches. As part of the exercise, you will use your package to solve several learning problems using various architectures and compare their performance.

In the next section we will define all the methods of the object "mydnn"

3 Development:

Instantiation

The constructor:

```
def __init__(self, architecture, loss,weight_decay=0):
```

- Architecture: A list of dictionaries, each dictionary represents a layer, for each layer the dictionary will consist
  - "input" - int, the dimension of the input
  - "output" int, the dimension of the output
  - "nonlinear" string, whose possible values are: "relu", "sigmoid", "sotmax" or "none"
  - "regularization" string, whose possible values are: "l1" (L_1 norm), or "l2" (L_2 norm)
- Loss - string, could be one of "MSE" or "cross-entropy"
- weight_decay - float, the lambda parameter for the regularization.

The constructor should build the network and initialize the weights and biases, for example: for layer r initialize $b^{[r]} = 0$, $W^{[r]}_{i,j} \sim U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$, where $n = i^{[r]}$ is the input dimension of the layer.

Fit

This is the method that trains the network

```
def fit(self, x_train, y_train, epochs, batch_size, learning_rate, x_val =
None, y_val = None):
```

Input Arguments:

- x_train - a Numpy nd-array where each row is a sample
- y_train - a 2d array, the labels of X in one-hot representation for classification or a value for each sample for regression.
- epochs - number of epochs to run
- batch_size - batch size for SGD
- learning_rate - float, a fixed learning rate that will be used in SGD.
- x_val - the validation x data (same structure as train data) default is None. When validation data is given, evaluation over this data will be made at the end of every epoch.
- y_val - the corresponding validation y data (labels) whose structure is identical to y_train.

Output:

- history - intermediate optimization results, which is a list of dictionaries, such that each epoch has a corresponding dictionary containing all relevant results. These dictionaries do not contain formatting information (you will later use the history to print various things including plots of learning and convergence curves for your networks).

Description:

The function will run SGD for a user-defined number of epochs, with the defined batch size. On every epoch the data will be reshuffled (make sure you shuffle the x's and y's together).

For every batch the data should be passed forward and gradients pass backward. After gradients are computed, weights update will be performed using the learning rate.

After every epoch the following line will be printed on screen with the values of the last epoch.

```
Epoch 3/5 - 30 seconds - loss: 0.3 - acc: 0.9 - val_loss: 0.4 - val_acc: 0.85
```

(The accuracy will be printed only for classification networks.)

Predict

The predict function will get an nd-array of inputs and return the network prediction

```
def predict(self,X, batch_size=None):
```

Arguments:

- X - a 2d array, with valid dimensions for the network.
- batch_size - an optional variable for splitting the prediction into batches for memory compliances; if none all the samples will be processed in one batch.

Returns -

- pred - a 2d array where each row is a prediction of the corespondent sample.

Evaluate

This function will evaluate a set (x's and y's) and returns the loss and accuracy (accuracy will be calculated only for classification).

```
def evaluate(self,X,y, batch_size = None):
```

Arguments:

- X - a 2d array, valid as an input to the network.

- y - a 2d array, the labels of X in one-hot representation for classification or a value for each sample for regression.
- batch_size - an optional variable for splitting the prediction into batches for memory compliances.

Output:

- [loss, accuracy] - for regression a list with the loss, for classification the loss and the accuracy.

Other Information

- You should normalize the data before training as similar to H.W. 1
- Running the experiments will take time, don't wait for the last day to begin.
- Your file should be named "mydnn.py", this will also be the name of the class.
- You can assume that the user will use MSE for regression and cross-entropy for classification.

4 Experiments

Consider the MNIST dataset (for which you already have a loading routine from HW 1). In the first set of experiments your goal is discover how the learning rate, the batch size, and the number of epochs affect the training. In each of these experiments, the results should be depicted in two figures (one for loss and one for accuracy), where the X-axis is the number of iterations (number of backward iterations during training) and the Y-axis will be the loss or accuracy. In each figure show both the training and validation curves.

Batch size:

We first consider a basic architecture with one hidden layer containing 128 neurons, a ReLU activation function, and softmax output layer. Your experiment should reveal the relationship between the batch size (with values 128, 1024 and 60000) to the learning performance. Discuss your results, and design and run more experiments to support your hypothesis, if needed.

Regularization:

Consider the last (one hidden layer) architecture and run it first without regularization, and compare to applications with L_1 and L_2 norms regularization (optimize the weight decay parameter λ on the validation set; an initial recommended value is $\lambda = 5e - 4$). Discuss how the use of regularization affects generalization.

Architecture:

Architecture selection is major and challenging task when constructing DNNs. Research how the architecture selection affects the results for the MNIST dataset. Find the best architecture from all architectures up to depth of 3 and width of 512 (conduct a grid search of at least 9 different architectures). In your report discuss how the width, depth and the overhaul number of weights affect the train and test accuracy. Motivate your selection of architecture from a statistical learning theory perspective (hypothesis set size, training set size, overfitting etc...).

Regression:

For this section we will create a synthetic dataset using the function

$$f(x) = x_1 \exp(-x_1^2 - x_2^2)$$

Sample uniformly at random m training points in the range $x_1 \in [-2, 2], x_2 \in [-2, 2]$. For the test set, take the linear grid using `np.linspace (-2,2, 1000)`.

Find the best architecture for the case where $m = 100$ and for the case $m = 1000$. In your results show the final MSE on the test set and also plot a 3d graph showing $\hat{y}_{test}$ (predicted values for the test points) as function of $x = (x_1, x_2)$.