Homework 0
11-785 Spring 2019

Updated: Dec 30th 2018
On Time Submission Deadline: 20 Jan 2019, 11:59:59 EDT
Late Submission Deadline: 31 Jan 2019, 11:59:59 EDT

Jan 1st 2019

1 Introduction

In this assignment you will be introduced to basic numpy functionality, vectorization, and slicing/indexing. The goals of the assignment are as follows:

- Understand the advantages of vectorization using numpy
- Learn basic and useful numpy functions
- Most importantly, no more loops!

You will be given a set of problems in this homework to test your basics. You have to finish all of them to get full points.

Although, this homework is worth only 1% of your final grade, it is essential that you do it fully! This homework acts as an introduction to python, if you cant solve this homework then you will be struggling with the coming assignments. Make sure you understand the concepts introduced here and in recitation 0 to determine your initial level in the course.

1.1 Autograder Submission

Your solutions will be autograded by Autolab. For this reason, it is important that you do not change the signature of any of the functions contained in the template.

In order to submit your solution, create a tar file containing your code. The root of the tar file should have a directory named hw0 containing your module code.

Creating the tar could be done through using the *tar* command in the command line. You can use this command to create the file,

```
tar --create --file=handin.tar files_to_include ...
# --create: will create a new archive.
# --file: name of the archived tar file.
```

You can also untar the handout using the following command in the command line.

```
tar --extract handout.tar
# --extract: extract files from an archive.
```

For more information on using tar please refer to this website, https://www.computerhope.com/unix/utar.htm .

2 Vectorization (30 points)

In this problem you will be given snippets of code. The snippets will be functions that you will be introduced to through out the course and famous functions you might use in basic machine learning algorithms. These functions will not be vectorized.

Your task is to vectorize the functions. That is, you have to replace the loop with numpy functions while maintaining its functionality.

- `vectorize_sumproducts`: Takes two 1-dimensional arrays and sums the products of all the pairs.
- `vectorize_Relu`: Takes one 2-dimensional array and apply the relu function on all the values of the array.
- `vectorize_PrimeRelu`: Takes one 2-dimensional array and apply the derivative of relu function on all the values of the array.

3 Variable length (60 points)

In this problem you will be given a variable length synthetic dataset. You will be given two different types of data, uni-variate time-series data and multivariate time-series data.

Uni-variate time-series data will look something like this (N, -) where N is the number of instances and - is the variable depending on the length of each instance.

Multivariate time-series data will look something like (N, -, F) where N is the number of instances, - is the variable depending on the length of each instance and F is the dimension of the features of an instance.

Your task will revolve around processing the data so that time-series arrays have the same length. You can use loops in this part.

3.1 Slicing (36 points)

In this part of the problem you are required to slice the data to a smaller lengths. That is you will be chopping part of an instance to make all the instances in

the dataset of the same length.To do that you have multiple options as to how to chop the dataset:

- `slice_fixed_point`: Takes one 3-dimensional array with the starting position and the length of the output instances. Your task is to slice the instances from the same starting position for the given length.
- `slice_last_point`: Takes one 3-dimensional array with the length of the output instances. Your task is to keeping only the l last points for each instances in the dataset.
- `slice_random_point`: Takes one 3-dimensional array with the length of the output instances. Your task is to slice the instances from a random point in each of the utterances with the given length. Please use function `numpy.random.randint` for generating the starting position.

Note that no matter what method you use you need to make sure that the length you choose to reduce the sizes to is larger than or equal to the size of any instance in the dataset. In this problem we give you the correct length that is possible to achieve for all the utterances in the dataset.

Here are some examples of how the functions should behave like . The examples are 2-dimensional arrays. You should implement methods for 3-dimensional array.

```
data = [
[u0l0, u0l1, u0l2, u0l3, u0l4],
[u1l0, u1l1, u1l2, u1l3],
[u2l0, u2l1, u2l2, u2l3, u2l4, u2l5],
[u3l0, u3l1, u3l2, u3l3, u3l4]
]

# For any (uXlY) in data, X stands for the index of the utterance and Y
    stands for the index of the feature in the feature vector of the
    utterance X.

result_slice_fixed_point = slice_fixed_point(data, 2, 1)
>>>> print(result_slice_fixed_point)
>>>>
    [[u0l1, u0l2],
     [u1l1, u1l2],
     [u2l1, u2l2],
     [u3l1, u3l2]]

result_slice_last_point = slice_last_point(data, 3)
>>>> print(result_slice_last_point)
>>>>
    [[u0l2, u0l3, u0l4],
     [u1l1, u1l2, u1l3],
     [u2l3, u2l4, u2l5],
     [u3l2, u3l3, u3l4]]
```

Note that we cannot give you an example for random point because for each utterance there will be a different starting position of each utterance.

3.2 Padding (24 points)

In this part of the problem you are required to pad the data to a larger/same lengths. That is you will be adding values to an instance to make all the instances in the dataset of the same length. To do that you have multiple options:

- `pad_pattern_end`: Takes one 3-dimensional array. Your task is to pad the instances from the end position as shown in the example below. That is, you need to pad the reflection of the utterance mirrored along the edge of the array.
- `pad_constant_central`: Takes one 3-dimensional array with the constant value of padding. Your task is to pad the instances with the given constant value while maintaining the array at the center of the padding.

Here are some examples of how the functions should behave like.

```
data = [
[u010, u011, u012, u013, u014],
[u110, u111, u112, u113],
[u210, u211, u212, u213, u214, u215],
[u310, u311]
]

# For any (uXlY) in data, X stands for the index of the utterance and Y
    stands for the index of the feature in the feature vector of the
    utterance X.

result_pad_pattern_end = pad_pattern_end(data)
>>>> print(result_pad_pattern_end)
>>>>
    [[u010, u011, u012, u013, u014, u014],
     [u110, u111, u112, u113, u113, u112],
     [u210, u211, u212, u213, u214, u215],
     [u310, u311, u311, u310, u310, u311]]

result_pad_constant_central = pad_constant_central(data, cval)
>>>> print(result_pad_constant_central)
>>>>
    [[u010, u011, u012, u013, u014, cval],
     [cval, u110, u111, u112, u113, cval],
     [u210, u211, u212, u213, u214, u215],
     [cval, cval, u310, u311, cval, cval]]
```

4 PyTorch (10 points)

PyTorch is an open-source deep learning library for python, and will be the primary framework throughout the course. You can install PyTorch referring to `https://PyTorch.org/get-started/locally/`. One of the fundamental concepts in PyTorch is the Tensor, a multi-dimensional matrix containing elements of a single type. Tensors are similar to numpy nd-arrays and tensors support most of the functionality that numpy matrices do.

In following exercises, you will familiarize yourself with tensors and more importantly, the PyTorch documentation. It is important to note that for this section we are simply using PyTorch's tensors as a matrix library, just like numpy. So please do not use functions in `torch.nn`, like `torch.nn.ReLU`.

4.1 Numpy and PyTorch Conversion (2 Points)

In PyTorch, it is very simple to convert between numpy arrays and tensors. PyTorch's tensor library provides functions to perform the conversion in either direction. In this task, you are to write 2 functions:

- `numpy2tensor`: Takes a numpy nd-array and converts it to a PyTorch tensor. Function `torch.tensor` is one of the simple ways to implement it but please do not use it this time. The PyTorch environment installed on Autolab is not an up-to-date version and does not support this function.
- `tensor2numpy`: Takes a PyTorch tensor and converts it to a numpy nd-array.

4.2 Tensor Sum-Products (4 Points)

In this task, you are to implement the function `tensor_sumproducts` that takes two tensors as input, and returns the sum of the element-wise products of the two tensors.

4.3 Tensor ReLU and ReLU Prime (4 Points)

In this task, you are to implement the ReLU and ReLU Prime function for PyTorch Tensors.

- `tensor_ReLU`: Takes a tensor and applies the ReLU function on all elements.
- `tensor_ReLU_prime`: Takes a tensor and applies the derivative of the ReLU function on all elements.