Homework 0

Due: 2020-09-27. 12pm

1 Introduction

In this homework you will be introduced to basic numpy functionality, vectorization, and slicing/indexing. The goals of the homework 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.

1.1 GitHub Submission

GitHub is a web-based hosting service for version control using Git. It is mostly used for computer code. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features.

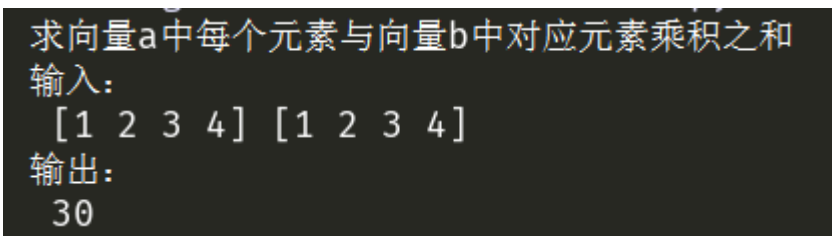
This semester, you'll need to upload your homework to GitHub. Each homework will be stored in a separate folder containing code and lab reports.

2 Vectorization

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.



```
使用numpy实现对一个二维数组每个元素的relu函数
输入：
 [[ 0.43319064 -0.27410033  1.7346154  -0.4458454   0.48723817]
 [-1.8072646  -0.61172277 -0.8146078  -2.046327   -0.12262674]
 [-0.72999716  0.55212754 -0.07076364 -1.1316797  -0.90253824]
 [ 0.39476725 -0.07755557 -0.23782122 -0.1616919  -0.02895901]
 [-1.0090414  -2.0450685  -0.3979849   0.42365417 -0.38129404]]
输出：
 [[0.43319064 0.         1.7346154  0.         0.48723817]
 [0.         0.         0.         0.         0.        ]
 [0.         0.55212754 0.         0.         0.        ]
 [0.39476725 0.         0.         0.         0.        ]
 [0.         0.         0.         0.42365417 0.        ]]
```

```
使用numpy实现对一个二维数组每个元素的relu函数导数
输入：
 [[-0.5052219   0.8363793   0.15555482  1.3820144  -0.806064  ]
 [-1.209871   -3.207883   -0.252078    2.2459204  -0.21223529]
 [-0.95360094 -0.830034   -1.5741748   0.66613966 -0.51535445]
 [-1.0048797  -0.40345213  0.40193152 -0.53335613 -0.6286738 ]
 [ 1.0651809   0.5166048  -0.99889266  0.9002582   1.6369417 ]]
输出：
 [[0 1 1 1 0]
 [0 0 0 1 0]
 [0 0 0 1 0]
 [0 0 1 0 0]
 [1 1 0 1 1]]
```

3 Variable length

In this problem you will be given a variable length synthetic dataset. You will be given two different types of data, univariate timeseries data and multivariate timeseries data.

Univariate timeseries data will look something like this (N, v) where N is the number of instances and v is the variable depending on the length of each instance.

Multivariate timeseries data will look something like (N, v, F) where N is the number of instances, v 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 timeseries arrays have the same length. You can use loops in this part.

3.1 Slicing

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 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 = [
[u010, u011, u012, u013, u014],
[u110, u111, u112, u113],
[u210, u211, u212, u213, u214, u215],
[u310, u311, u312, u313, u314]
]

# 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)
>>>>
    [[u011, u012],
     [u111, u112],
     [u211, u212],
     [u311, u312]]

result_slice_last_point = slice_last_point(data, 3)
>>>> print(result_slice_last_point)
>>>>
    [[u012, u013, u014],
     [u111, u112, u113],
     [u213, u214, u215],
     [u312, u313, u314]]
```

3.2 Padding

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

PyTorch is an open-source machine learning library for Python, based on Torch, used for applications such as natural language processing.It is primarily developed by Facebook's artificial-intelligence research group, and Uber's "Pyro" software for probabilistic programming is built on it.

PyTorch provides two high-level features:

- Tensor computation (like NumPy) with strong GPU acceleration.
- Deep neural networks built on a tape-based autodiff system.

4.1 Numpy and PyTorch Conversion

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 ndarray 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.
- tensor2numpy: Takes a PyTorch tensor and converts it to a numpy ndarray.

4.2 Tensor Sum-products

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.

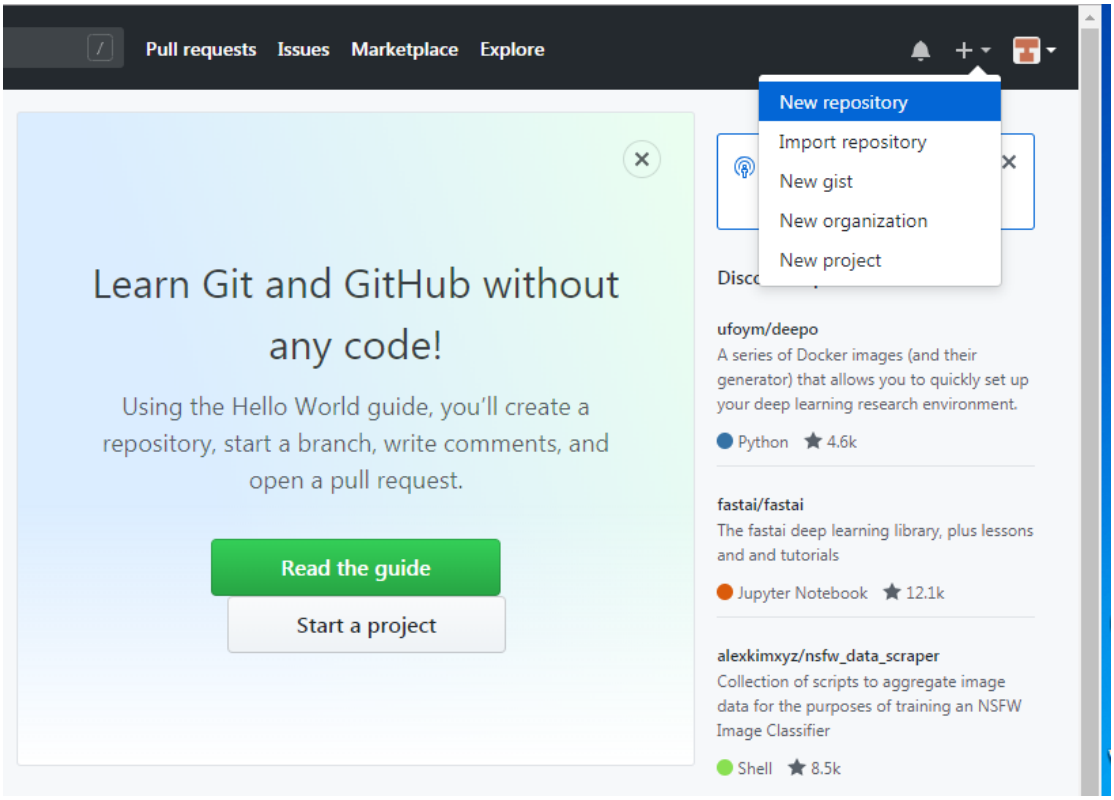
4.3 Tensor ReLu and ReLu prime

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

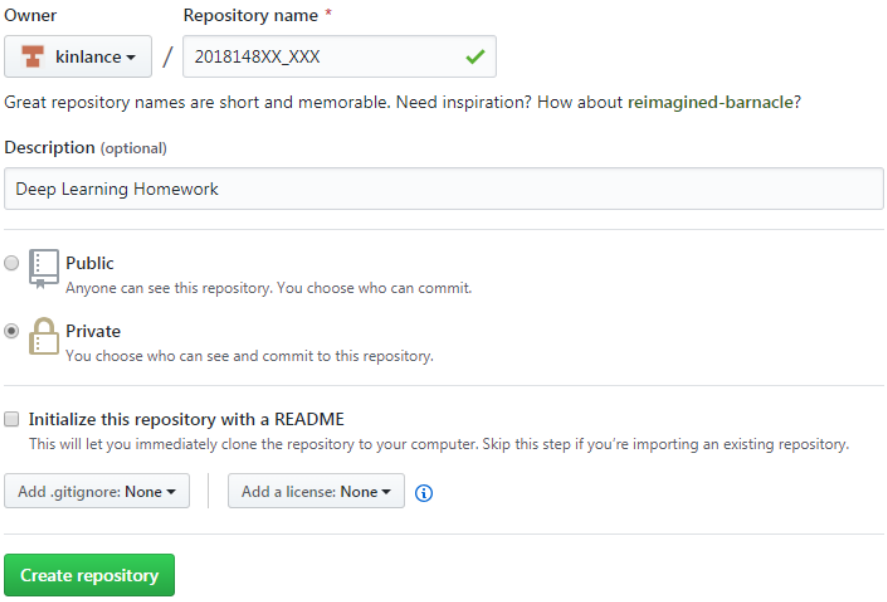
- *tensor_ReLU:* Takes a tensor and applies the ReLU function on all elements.
- *tensor_ReLU_prime:* Takes a tensor and applies the derivative of the ReLU function on all elements.

Github 操作指南

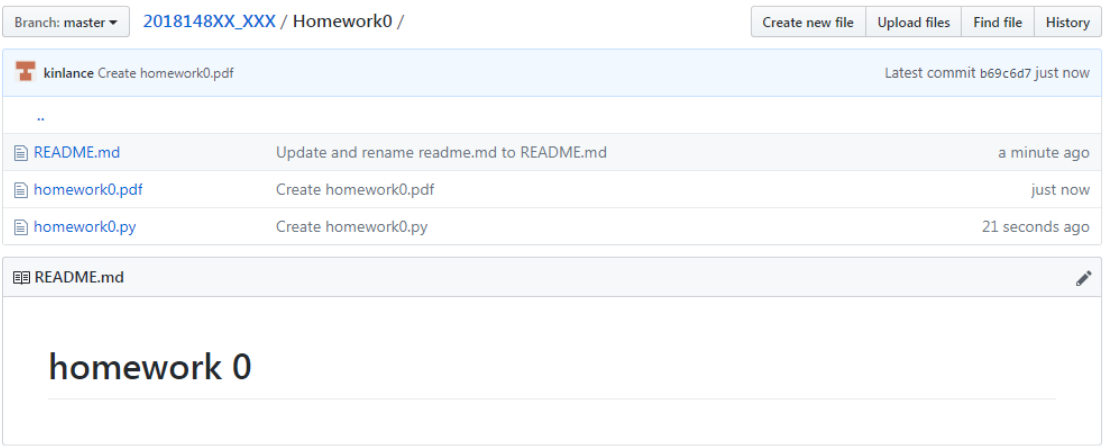
1. 建立新的 repository



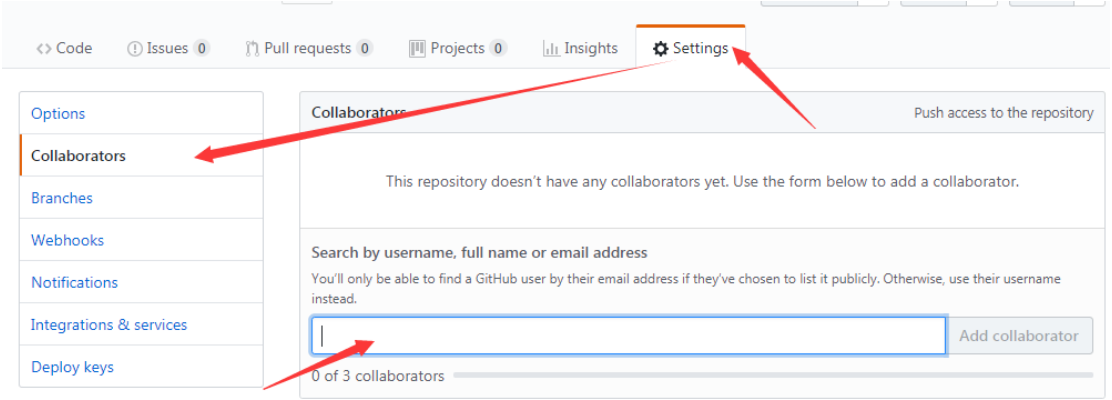
2. 建立新工程



3. 作业提交内容



4. 邀请老师、TA 加入组



GitHub 账号

甘甜：gantian
董兴宁：dongxingning
俞旭铮：OMRailgun

请将个人 GitHub Private repository 的 URL 提交至：
https://docs.qq.com/sheet/DWWJTVVpCQlJFU2Z1