fairgrad

Release 0.1.1

Gaurav Maheshwari, Michael Perrot

Sep 06, 2022

CONTENTS:

CHAPTER ONE

INTRODUCTION

FairGrad, is an easy to use general purpose approach to enforce fairness for gradient descent based methods. The core idea is to enforce fairness by iteratively learns group specific weights based on whether they are advantaged or not. FairGrad is:

- Simple and easy to integrate with no significant overhead
- Supports Multiclass problems and can work with any gradient based methods
- Supports various group fairness notions including exact and approximate fairness
- Is competitive on various tasks including complex NLP and CV ones

CHAPTER TWO

INSTALLATION

You can install FairGrad from PyPI with *pip* or your favorite package manager:

```
pip install fairgrad
```

CHAPTER

THREE

QUICK START

To use fairgrad simply replace your pytorch cross entropy loss with fairgrad cross entropy loss. Alongside, regular pytorch cross entropy arguments, it expects following *extra* arguments:

```
y_train (np.asarray[int], Tensor, optional): All train example's corresponding label.
         Note that the label space must start from 0.
s_train (np.asarray[int], Tensor, optional): All train example's corresponding sensitive␣
↪attribute. This means if there
        are 2 sensitive attributes, with each of them being binary. For instance gender -
↪ (male and female) and
        age (above 45, below 45). Total unique sentive attributes are 4.
        Note that the protected space must start from 0.
fairness_measure (string): Currently we support "equal_odds", "equal_opportunity",
↪"accuracy_parity", and "demographic_parity".
                           Note that demographic parity is only supported for binary␣
↪case.
epsilon (float, optional): The slack which is allowed for the final fairness level.
fairness_rate (float, optional): Parameter which intertwines current fairness weights␣
↪with sum of previous fairness rates.
```

CHAPTER FOUR

USAGE EXAMPLE

Below is a simple example:

```
>>> from fairgrad.torch import CrossEntropyLoss
>>> input = torch.randn(10, 5, requires_grad=True)
>>> target = torch.empty(10, dtype=torch.long).random_(2)
>>> s = torch.empty(10, dtype=torch.long).random_(2) # protected attribute
>>> loss = CrossEntropyLoss(y_train = target, s_train = s, fairness_measure = 'equal_odds
↪')
>>> output = loss(input, target, s, mode='train')
>>> output.backward()
```

For complete worked out example refer to example folder on github.

CHAPTER

FIVE

CITATION

Citation for this work:

```
@article{maheshwari2022fairgrad,
         title={FairGrad: Fairness Aware Gradient Descent},
         author={Maheshwari, Gaurav and Perrot, Micha{\"e}l},
         journal={arXiv preprint arXiv:2206.10923},
         year={2022}}
```

CHAPTER

SIX

REFERENCE

6.1 fairness.functions

class fairgrad.fairness_functions.**AccuracyParity**(*y_unique*, *s_unique*, *y*, *s*)

The function implements the accuracy parity fairness function. A model h is fair for Accuracy Parity when theprobability of being correct is independent of the sensitive attribute.

Parameters

- **y_unique** (*npt.ndarray[int]*) – all unique labels in all label space.
- **s_unique** (*npt.ndarray[int]*) – all unique protected attributes in all protected attribute space.
- **y** (*npt.ndarray[int]*) – all label space
- **s** (*npt.ndarray[int]*) – all protected attribute space

class fairgrad.fairness_functions.**DemographicParity**(*y_unique*, *s_unique*, *y*, *s*)

The function implements the demographic parity fairness function. A model h is fair for Demographic Parity when the probability of predicting each label is independent of the sensitive attribute.

Parameters

- **y_unique** (*npt.ndarray[int]*) – all unique labels in all binary label space.
- **s_unique** (*npt.ndarray[int]*) – all unique protected attributes in all protected attribute space.
- **y** (*npt.ndarray[int]*) – all binary label space
- **s** (*npt.ndarray[int]*) – all protected attribute space

class fairgrad.fairness_functions.**EqualityOpportunity**(*y_unique*, *s_unique*, *y*, *s*, *y_desirable*)

The function implements the accuracy parity fairness function. A model h is fair for Equality of Opportunity when the probability of predicting the correct label is independent of the sensitive attribute for a given subset of labels called the desirable outcomes

Parameters

- **y_unique** (*npt.ndarray[int]*) – all unique labels in all label space.
- **s_unique** (*npt.ndarray[int]*) – all unique protected attributes in all protected attribute space.
- **y** (*npt.ndarray[int]*) – all label space
- **s** (*npt.ndarray[int]*) – all protected attribute space

- **y_desirable** (*npt.ndarray[int]*) – the label for which the fairness needs to be enforced.

`class fairgrad.fairness_functions.EqualizedOdds`(*y_unique*, *s_unique*, *y*, *s*)

The function implements the equal odds fairness function. A model h is fair for Equalized Odds when the probability of predicting the correct label is independent of the sensitive attribute.

Parameters

- **y_unique** (*npt.ndarray[int]*) – all unique labels in all label space.
- **s_unique** (*npt.ndarray[int]*) – all unique protected attributes in all protected attribute space.
- **y** (*npt.ndarray[int]*) – all label space
- **s** (*npt.ndarray[int]*) – all protected attribute space

`class fairgrad.fairness_functions.FairnessSetupArguments`(*fairness_function_name: str*, *y_unique: <built-in function asarray>*, *s_unique: <built-in function asarray>*, *all_train_y: <built-in function asarray>*, *all_train_s: <built-in function asarray>*, *y_desirable: Union[<built-in function asarray>, NoneType] = None*)

Parameters

- **fairness_function_name** (*str*) –
- **y_unique** (*asarray*) –
- **s_unique** (*asarray*) –
- **all_train_y** (*asarray*) –
- **all_train_s** (*asarray*) –
- **y_desirable** (*Optional[asarray]*) –

6.2 fairgrad.torch.cross_entropy

`class fairgrad.torch.cross_entropy.CrossEntropyLoss`(*weight=None*, *size_average=None*, *ignore_index=-100*, *reduce=None*, *reduction='mean'*, *y_train=None*, *s_train=None*, *fairness_measure=None*, *epsilon=0.0*, *fairness_rate=0.01*)

This is an extension of the CrossEntropyLoss provided by pytorch. Please check pytorch documentation for understanding the cross entropy loss. Here, we augment the cross entropy to enforce fairness. The exact algorithm can be found in Fairgrad paper <https://arxiv.org/abs/2206.10923>.

Parameters

- **weight** (*Tensor, optional*) – a manual rescaling weight given to each class. If given, has to be a Tensor of size C
- **size_average** (*bool, optional*) – Deprecated (see `reduction`). By default, the losses are averaged over each loss element in the batch. Note that for some losses, there are multiple elements per sample. If the field `size_average` is set to `False`, the losses are instead summed for each minibatch. Ignored when reduce is `False`. Default: `True`

- **ignore_index** (*int, optional*) – Specifies a target value that is ignored and does not contribute to the input gradient. When `size_average` is `True`, the loss is averaged over non-ignored targets.
- **reduce** (*bool, optional*) – Deprecated (see `reduction`). By default, the losses are averaged or summed over observations for each minibatch depending on `size_average`. When `reduce` is `False`, returns a loss per batch element instead and ignores `size_average`. Default: `True`
- **reduction** (*string, optional*) – Specifies the reduction to apply to the output: `'none'` | `'mean'` | `'sum'`. `'none'`: no reduction will be applied, `'mean'`: the weighted mean of the output is taken, `'sum'`: the output will be summed. Note: `size_average` and `reduce` are in the process of being deprecated, and in the meantime, specifying either of those two args will override `reduction`. Default: `'mean'`
- **y_train** (*np.asarray[int], Tensor, optional*) – All train example's corresponding label
- **s_train** (*np.asarray[int], Tensor, optional*) – All train example's corresponding sensitive attribute. This means if there are 2 sensitive attributes, with each of them being binary. For instance gender - (male and female) and age (above 45, below 45). Total unique sentive attributes are 4.
- **fairness_measure** (*string*) – Currently we support "equal_odds", "equal_opportunity", and "accuracy_parity".
- **epsilon** (*float, optional*) – The slack which is allowed for the final fairness level.
- **fairness_rate** (*float, optional*) – Parameter which intertwines current fairness weights with sum of previous fairness rates.

Examples:

```
>>> input = torch.randn(10, 5, requires_grad=True)
>>> target = torch.empty(10, dtype=torch.long).random_(2)
>>> s = torch.empty(10, dtype=torch.long).random_(2) # protected attribute
>>> loss = CrossEntropyLoss(y_train = target, s_train = s, fairness_measure =
↪'equal_odds')
>>> output = loss(input, target, s, mode='train')
>>> output.backward()
```

forward(*input*, *target*, *s=None*, *mode='train'*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

Parameters

- **input** (*Tensor*) –
- **target** (*Tensor*) –
- **s** (*Optional[Tensor]*) –
- **mode** (*str*) –

Return type
Tensor

PYTHON MODULE INDEX

INDEX