higher

Release 0.2.1

Facebook AI Research (FAIR)

Jul 14, 2020

LIBRARY REFERENCE:

`higher` is a library providing support for higher-order optimization, e.g. through unrolled first-order optimization loops, of "meta" aspects of these loops. It provides tools for turning existing *torch.nn.Module* instances "stateless", meaning that changes to the parameters thereof can be tracked, and gradient with regard to intermediate parameters can be taken. It also provides a suite of differentiable optimizers, to facilitate the implementation of various meta-learning approaches.

CHAPTER

ONE

TOP-LEVEL FUNCTIONS

higher.**innerloop_ctx**(*model*, *opt*, *device=None*, *copy_initial_weights=True*, *override=None*, *track_higher_grads=True*)

A context manager for writing differentiable inner loops.

Parameters

- **model** – a `torch.nn.Module` subclass instance.
- **opt** – an existing optimizer, assumed to be an instance of `torch.optim.Optimizer`, of a supported type which is either defined in `torch.optim`, or a custom implemantation which has been added to higher at runtime by using `higher.register_optim`. We assume this optimizer tracks the parameters (or some subset thereof) of a single `torch.nn.Module` instance, with support for parameter groups.
- **device** (*optional*) – a device to cast the fast weights and state to. If not specified, the device used for corresponding weights of `model` will be used.
- **copy_initial_weights** – if true, the weights of the patched module are copied to form the initial weights of the patched module, and thus are not part of the gradient tape when unrolling the patched module. If this is set to False, the actual module weights will be the initial weights of the patched module. This is useful when doing MAML, for example.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows `innerloop_ctx` to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

Yields A `(fmodule, diffopt)` tuple. where `fmodule` is a "stateless" version of the original module, for which calls to forward take the additional kwarg-only parameter `params`, which should be a list of torch tensors requiring gradients, ideally provided by this function (see below) or by an update step from one of the optimizers in `higher.optim`. And `diffopt` is an initialized `DifferentiableOptimizer` instance of the right subtype.

CHAPTER

TWO

MONKEY-PATCHING FUNCTIONS

Functions for making `torch.nn.Module` subclass instances stateless.

`higher.patch.`**`buffer_sync`**(*module*, *fmodule*, *device=None*)
One off sync (copy) of buffers in `fmodule` with those from `module`.

Return type `None`

`higher.patch.`**`make_functional`**(*module*, *encapsulator=None*)
Returns a stateless version of an `nn.Module` instance.

Return type `_MonkeyPatchBase`

`higher.patch.`**`monkeypatch`**(*module*, *device=None*, *copy_initial_weights=True*, *track_higher_grads=True*)
Create a monkey-patched stateless version of a module.

This function produces a monkey-patched version of a module, and returns a copy of its parameters for use as fast weights. Where the original module or any of its submodules have state (e.g. batch norm), this will be copied too, but further updates (e.g. during inner loop training) will cause these to diverge without changing the state of the original module.

Parameters

- **`module`** – a `torch.nn.Module` subclass instance.
- **`device`** (*`optional`*) – a device to cast the fast weights and state to.
- **`copy_initial_weights`** – if True, the weights of the patched module are copied to form the initial weights of the patched module, and thus are not part of the gradient tape when unrolling the patched module. If this is set to False, the actual module weights will be the initial weights of the patched module. This is useful when doing MAML, for example.
- **`track_higher_grads`** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows `monkeypatch` to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

Returns a "stateless" version of the original module, for which calls to forward take the additional kwarg-only parameter `params`, which should be a list of torch tensors requiring gradients, ideally provided by this function (see below) or by an update step from one of the optimizers in `higher.optim`.

Return type `fmodule`

Return type `_MonkeyPatchBase`

CHAPTER

THREE

DIFFERENTIABLE OPTIMIZERS

Differentiable optimizer wrappers around `torch.optim` instances.

class higher.optim.**DifferentiableASGD**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

A differentiable version of the ASGD optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableAdadelta**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

A differentiable version of the Adadelta optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableAdagrad**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

A differentiable version of the Adagrad optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.

- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableAdam**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

A differentiable version of the Adam optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableAdamax**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

A differentiable version of the Adamax optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableOptimizer**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.

- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

step(*loss*, *params=None*, *override=None*, *grad_callback=None*, ***kwargs*)

Perform a model update.

This would be used by replacing the normal sequence:

```
opt.zero_grad()
loss.backward()
opt.step()
```

with:

```
diffopt.step(loss)
```

Parameters

- **loss** – the loss tensor.
- **params** (*optional*) – the parameters with regard to which we measure the loss. These must be provided if the differentiable optimizer did not receive a patched model with a view over its own fast weights at initialisation. If there is such a model, and params are provided, they will overwrite the params of the encapsulated model.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings. Setting override here has highest precedence, i.e. it will override any tensors provided as override during the creation of the differentiable optimizer, where there is name clash.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. This callback overrides the default provided when constructing the differentiable optimizer.

Returns The updated parameters, which will individually have `grad_fn`s of their own. If the optimizer has an encapsulated patched model, its view over its own fast weights will be updated with these params.

Return type `Iterable[Tensor]`

class higher.optim.**DifferentiableRMSprop**(**args*, ***kwargs*)

A differentiable version of the RMSprop optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableRprop**(**args*, ***kwargs*)

A differentiable version of the Rprop optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by `reference_params`. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the op-

timization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

class higher.optim.**DifferentiableSGD**(*other*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *grad_callback=None*, *track_higher_grads=True*, ***kwargs*)

A differentiable version of the SGD optimizer.

This optimizer creates a gradient tape as it updates parameters.

Initialize the optimizer with the state of an existing optimizer.

Parameters

- **other** – an existing optimizer instance.
- **reference_params** – an iterable over the parameters of the original model.
- **fmodel** (*optional*) – a patched stateless module with a view on weights.
- **device** (*optional*) – the device to cast state tensors to.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the ith element of the list overrides the corresponding setting in the ith parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **grad_callback** – (optional) a single argument function which will be applied to a list of gradients of parameters, which respects the order specified by reference_params. This can be used to apply a function, such as gradient clipping, to all (or a subset) of these gradients every time the step function is called. If this keyword argument is provided when calling the step method, its value will override the default specified here.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

higher.optim.**apply_trainable_opt_params**(*opt*, *override*)

Apply learned hyperparameters back to original optimizer.

Parameters

- **opt** – the original optimizer. The hyperparameters in its parameter groups will be modified in place.
- **override** – dictionary of the format used for the override kwarg of differentiable optimizers.

Return type None

higher.optim.**create_diff_optim**(*opt_type*, *opt_kwargs=None*, *params=None*, *fmodel=None*, *device=None*, *override=None*, *track_higher_grads=True*, ***kwargs*)

Construct a differentiable version of an new optimizer.

Parameters

- **opt_type** – the type (constructor) for a torch.optim.Optimizer subtype from amongst the types supported by the library, or registered with it a runtime.
- **opt_kwargs** – a dictionary of keywords to be passed to the optimizer constructor.
- **params** (*optional*) – a list of (fast) weights which the differentiable optimizer will update. These must be provided if fmodel is not provided. If both, these will be used in lieu. These will only be used for shape inference when initializing the optimizer. This argument can also take the same format as parameter groups, i.e. an iterable over dictionaries which contain the 'params' key with fast weights as value, and group-specific hyperparameters.
- **fmodel** (*optional*) – a patched version of the `module` tracked by `opt`. It is assumed this patched instance has a view on its latest fast weights through `fmodel.parameters()`. If provided, it is not necessary to pass the fast weights explicitly to the differentiable optimizer's `step` function via the keyword arg `params`. If not provided, the fast weights to update must be provided to `step`.
- **device** (*optional*) – the device to cast the optimizer state to when creating the differentiable optimizer. If not provided, the same device as used for the parameters tracked by `opt` will be used.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the returned differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

Returns An initialized `DifferentiableOptimizer` instance of the right subtype.

Return type *DifferentiableOptimizer*

higher.optim.**get_diff_optim**(*opt*, *reference_params*, *fmodel=None*, *device=None*, *override=None*, *track_higher_grads=True*, ***kwargs*)

Construct/initialize a differentiable version of an existing optimizer.

Parameters

- **opt** – an existing optimizer, assumed to be an instance of `torch.optim.Optimizer`, of a supported type which is either defined in `torch.optim`, or a custom implemantation which has been added to higher at runtime by using `higher.register_optim`. We assume this optimizer tracks the parameters (or some subset thereof) of a single `torch.nn.Module` instance, with support for parameter groups.
- **reference_params** – the parameters of the module tracked by `opt`, as returned by `module.parameters()`.
- **fmodel** (*optional*) – a patched version of the `module` tracked by `opt`. It is assumed this patched instance has a view on its latest fast weights through `fmodel.parameters()`. If provided, it is not necessary to pass the fast weights explicitly to

the differentiable optimizer's `step` function via the keyword arg `params`. If not provided, the fast weights to update must be provided to `step`.

- **device** (*optional*) – the device to cast the optimizer state to when creating the differentiable optimizer. If not provided, the same device as used for the parameters tracked by `opt` will be used.
- **override** (*optional*) – a dictionary mapping optimizer settings (i.e. those which would be passed to the optimizer constructor or provided within parameter groups) to either singleton lists of override values, or to a list of override values of length equal to the number of parameter groups. If a single override is provided for a keyword, it is used for all parameter groups. If a list is provided, the `i`th element of the list overrides the corresponding setting in the `i`th parameter group. This permits the passing of tensors requiring gradient to differentiable optimizers for use as optimizer settings.
- **track_higher_grads** – if True, during unrolled optimization the graph be retained, and the fast weights will bear grad funcs, so as to permit backpropagation through the optimization process. Setting this to False allows the returned differentiable optimizer to be used in "test mode", without potentially tracking higher order gradients. This can be useful when running the training loop at test time, e.g. in k-shot learning experiments, without incurring a significant memory overhead.

Returns An initialized `DifferentiableOptimizer` instance of the right subtype.

Return type *DifferentiableOptimizer*

higher.optim.**get_trainable_opt_params**(*opt*, *device=None*)

Get an override dictionary from an optimizer instance.

Parameters

- **opt** – the optimizer to obtain an override dictionary from.
- **device** (*optional*) – the device to cast the learnable tensors to.

Returns A dictionary of the format expected for the override kwarg of differentiable optimizers. It is initialized with trainable tensors with as values those float and int hyperparameters found in the optimizer's parameter groups (or stuctures containing these). Heuristically, hyperparameters containing mixtures of differentiable and non-differentiable types will be ignored (and must be manually specified when constructing an override dict).

Return type `Dict[str, List[Any]]`

higher.optim.**register_optim**(*optim_type*, *diff_optim_type*)

Registers a new optimizer type for use with higher functions.

Parameters

- **optim_type** – the type of a new optimizer, assumed to be an instance of `torch.optim.Optimizer`.
- **diff_optim_type** – the type of a new differentiable optimizer, assumed to be an instance of `higher.optim.DifferentiableOptimizer` with functionally equivalent logic to `optim_type`.

Return type `None`

CHAPTER

FOUR

UTILITY FUNCTIONS

Utility functions for components of `higher`.

`higher.utils.`**`flatten`**(*x*)
Returns a flattened list of objects from a nested structure.

Return type `List`[`Any`]

`higher.utils.`**`get_func_params`**(*module*, *device=None*, *safe_copy=True*)
Returns a detached copy of module parameters which requires gradient.

Return type `List`[`Tensor`]

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX