# Neural Monitor

**Duc Nguyen**

**May 17, 2021**

# CONTENTS:

CHAPTER

ONE

# MONITOR – MONITOR TRAINING OF NEURAL NETWORKS

**class** neural_monitor.monitor.**Monitor**

Collects statistics and displays the results using various backends. The collected stats are stored in <root>/<model_name>/<prefix><#id> where #id is automatically assigned each time a new run starts.

The following snippet shows how to plot smoothed training losses and save images from the current iteration, and then display them every 100 iterations.

```
from neural_monitor import monitor as mon

# Tensorboard is turned on by default
mon.initialize(model_name='foo-model', print_freq=100, use_tensorboard=True)
...

def calculate_loss(pred, gt):
    ...
    training_loss = ...
    mon.plot('training loss', loss, smooth=.99, filter_outliers=True)

def calculate_acc(pred, gt):
    accuracy = ...
    mon.plot('training acc', accuracy, smooth=.99, filter_outliers=True)

...
for epoch in mon.iter_epoch(range(n_epochs)):
    for data in mon.iter_batch(data_loader):
        pred = net(data)
        calculate_loss(pred, gt)
        calculate_acc(pred, gt)
        mon.imwrite('input images', data['images'], latest_only=True)

    mon.dump('checkpoint.pt', {
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        ...
    }, method='torch', keep=5)  # keep only 5 latest checkpoints
...
```

**current_folder** path to the current run.

**writer** an instance of Tensorboard's `SummaryWriter` when *use_tensorboard* is set to `True`.

**plot_folder** path to the folder containing the collected plots.

**file_folder** path to the folder containing the collected files.

**image_folder** path to the folder containing the collected images.

**hist_folder** path to the folder containing the collected histograms.

**backup**(*files_or_folders: Union[str, List[str]]*, *ignores: Union[str, List[str]] = None*, *includes: Union[str, List[str]] = None*)
saves a copy of the given files to `current_folder`. Accepts a str or list/tuple of file or folder names. You can backup your codes and/or config files for later use.

**Parameters**

- **files_or_folders** – files or folders to be saved.
- **ignores** – files or patterns to ignore. Default: `None`.
- **includes** – files or patterns to include. Default: `None`.

**Returns** `None`.

**clear_hist_stats**(*key: Union[int, str, Tuple]*)
removes the collected statistics for histogram plot of the specified *key*.

**Parameters** **key** – the name of the histogram collection.

**Returns** `None`.

**clear_num_stats**(*key*)
removes the collected statistics for scalar plot of the specified *key*.

**Parameters** **key** – the name of the scalar collection.

**Returns** `None`.

**dump**(*name: str*, *obj: Any*, *method: str = 'pickle'*, *keep: int = - 1*, ***kwargs*)
saves the given object.

**Parameters**

- **name** – name of the file to be saved.
- **obj** – object to be saved.
- **method** – `str` or `callable`. If `callable`, it should be a custom method to dump object. There are 3 types of `str`.

  `'pickle'`: use `pickle.dump()` to store object.

  `'torch'`: use `torch.save()` to store object.

  `'txt'`: use `numpy.savetxt()` to store object.

  Default: `'pickle'`.
- **keep** – the number of versions of the saved file to keep. Default: -1 (keeps only the latest version).
- **kwargs** – additional keyword arguments to the underlying save function.

**Returns** `None`.

**dump_model**(*network*, *use_tensorboard=False*, **args*, ***kwargs*)
saves a string representation of the given neural net.

**Parameters**

- **network** – neural net to be saved as string representation.

- **use_tensorboard** – use tensorboard to save *network*'s graph.
- **args** – additional arguments to Tensorboard's `SummaryWriter()` when *use_tensorboard* is `True`.
- **kwargs** – additional keyword arguments to Tensorboard's `SummaryWriter()` when *~se_tensorboard* is `True`.

**Returns** `None`.

**dump_rep**(*name*, *obj*)
saves a string representation of the given object.

**Parameters**

- **name** – name of the txt file containing the string representation.
- **obj** – object to saved as string representation.

**Returns** `None`.

**property epoch: int**
returns the current epoch.

**Returns** `_last_epoch`.

**flush**()
executes all the scheduled plots. Do not call this if using *Monitor*'s context manager mode.

**Returns** `None`.

**hist**(*name*, *value: Union[torch.Tensor, numpy.ndarray]*, *n_bins: int = 20*, *latest_only: bool = False*, ***kwargs*)
schedules a histogram plot of (a batch of) points. A `matplotlib` figure will be rendered and saved every `print_freq` iterations.

**Parameters**

- **name** – name of the figure to be saved. Must be unique among plots.
- **value** – any-dim tensor to be histogrammed.
- **n_bins** – number of bins of the histogram.
- **latest_only** – whether to save only the latest statistics or keep everything from beginning.
- **kwargs** – additional options to tensorboard

**Returns** `None`.

**property hist_stats**
returns the collected tensors from beginning.

**Returns** `_hist_since_beginning`.

**imwrite**(*name: str*, *value: Union[torch.Tensor, numpy.ndarray]*, *latest_only: Optional[bool] = False*, ***kwargs*)
schedules to save images. The images will be rendered and saved every `print_freq` iterations. There are some assumptions about input data:

- If the input is `'uint8'` it is an 8-bit image.
- If the input is `'float32'`, its values lie between `0` and `1`.
- If the input has 3 dims, the shape is `[h, w, 3]` or `[h, w, 1]`.
- If the channel dim is different from 3 or 1, it will be considered as multiple gray images.

**Parameters**

- **name** – name of the figure to be saved. Must be unique among plots.
- **value** – 2D, 3D or 4D tensor to be plotted. The expected shape is `(H, W)` for 2D tensor, `(H, W, C)` for 3D tensor and `(N, C, H, W)` for 4D tensor. If the number of channels is other than 3 or 1, each channel is saved as a gray image.
- **latest_only** – whether to save only the latest statistics or keep everything from beginning.
- **kwargs** – additional options to tensorboard.

**Returns** `None`.

**initialize**(*model_name: Optional[str] = None*, *root: Optional[str] = None*, *current_folder: Optional[str] = None*, *print_freq: Optional[int] = 1*, *num_iters: Optional[int] = None*, *prefix: Optional[str] = 'run'*, *use_tensorboard: Optional[bool] = True*, *with_git: Optional[bool] = False*) → None

**Parameters**

- **model_name** – name of the experimented model. Default: `'my-model'`.
- **root** – root path to store results. Default: `'results'`.
- **current_folder** – the folder that the experiment is currently dump to. Note that if *current_folder* already exists, all the contents will be loaded. This option can be used for loading a trained model. Default: `None`.
- **print_freq** – frequency of stdout. Default: 1.
- **num_iters** – number of iterations per epoch. If not provided, it will be calculated after one epoch. Default: `None`.
- **prefix** – a common prefix that is shared between folder names of different runs. Default: `'run'`.
- **use_tensorboard** – whether to use Tensorboard. Default: `True`.
- **with_git** – whether to retrieve some Git information. Should be used only when the project is initialized with Git. Default: `False`.

**Returns** `None`.

**property iter: int**

returns the current iteration.

**Returns** `_iter`.

**iter_batch**(*iterator: Iterable*) → Any

tracks training iteration and returns the item in *iterator*.

**Parameters** **iterator** – the batch iterator. For e.g., `enumerator(loader)`.

**Returns** a generator over *iterator*.

```
>>> from neuralnet_pytorch import monitor as mon
>>> mon.print_freq = 1000
>>> data_loader = ...
>>> num_epochs = 10
>>> for epoch in mon.iter_epoch(range(num_epochs)):
...     for idx, data in mon.iter_batch(enumerate(data_loader)):
...         # do something here
```

*iter_epoch()*

**iter_epoch**(*iterator: Iterable*) → Any
tracks training epoch and returns the item in *iterator*.

**Parameters** **iterator** – the epoch iterator. For e.g., `range(num_epochs)`.

**Returns** a generator over *iterator*.

```
>>> from neuralnet_pytorch import monitor as mon
>>> mon.print_freq = 1000
>>> num_epochs = 10
>>> for epoch in mon.iter_epoch(range(mon.epoch, num_epochs))
...     # do something here
```

*iter_batch()*

**load**(*file: str*, *method: str = 'pickle'*, *version: int = - 1*, ***kwargs*)
loads from the given file.

**Parameters**

- **file** – name of the saved file without version.
- **method** – `str` or `callable`. If `callable`, it should be a custom method to load object. There are 3 types of `str`.

  `'pickle'`: use `pickle.dump()` to store object.

  `'torch'`: use `torch.save()` to store object.

  `'txt'`: use `numpy.savetxt()` to store object.

  Default: `'pickle'`.
- **version** – the version of the saved file to load. Default: -1 (loads the latest version of the saved file).
- **kwargs** – additional keyword arguments to the underlying load function.

**Returns** `None`.

**property model_name: str**
returns the name of the model.

**Returns** `_model_name`.

**property num_stats**
returns the collected scalar statistics from beginning.

**Returns** `_num_since_beginning`.

**plot**(*name: str*, *value: Union[torch.Tensor, numpy.ndarray, float]*, *smooth: Optional[float] = 0*, *filter_outliers: Optional[bool] = True*, ***kwargs*)
schedules a plot of scalar value. A `matplotlib` figure will be rendered and saved every `print_freq` iterations.

**Parameters**

- **name** – name of the figure to be saved. Must be unique among plots.
- **value** – scalar value to be plotted.
- **smooth** – a value between `0` and `1` to define the smoothing window size. See `smooth()`. Default: `0`.

- **filter_outliers** – whether to filter out outliers in plot. This affects only the plot and not the raw statistics. Default: True.
- **kwargs** – additional options to tensorboard.

**Returns** None.

**plot_matrix**(*name: str*, *value: Union[torch.Tensor, numpy.ndarray, float]*, *labels: Union[List[str], List[List[str]]] = None*, *show_values: bool = False*)
plots the given matrix with colorbar and labels if provided.

**Parameters**

- **name** – name of the figure to be saved. Must be unique among plots.
- **value** – matrix value to be plotted.
- **labels** – labels of each axis. Can be a list/tuple of strings or a nested list/tuple. Defaults: None.
- **show_values** – show values of the matrix

**Returns** None.

**property prefix: str**
returns the prefix of saved folders.

**Returns** _prefix.

**read_log**()
reads the saved log file.

**Returns** contents of the log file.

**reset**()
factory-resets the monitor object. This includes clearing all the collected data, set the iteration and epoch counters to 0, and reset the timer.

**Returns** None.

**scatter**(*name: str*, *value: Union[torch.Tensor, numpy.ndarray]*, *latest_only: bool = False*, ***kwargs*)
schedules a scattor plot of (a batch of) points. A 3D matplotlib figure will be rendered and saved every print_freq iterations.

**Parameters**

- **name** – name of the figure to be saved. Must be unique among plots.
- **value** – 2D or 3D tensor to be plotted. The last dim should be 3.
- **latest_only** – whether to save only the latest statistics or keep everything from beginning.
- **kwargs** – additional options to tensorboard.

**Returns** None.

neural_monitor.monitor.**collect_tracked_variables**(*name=None*, *return_name=False*)
Gets tracked variable given name.

**Parameters**

- **name** – name of the tracked variable. can be str or``list``/tuple of str``s. If ``None, all the tracked variables will be returned.
- **return_name** – whether to return the names of the tracked variables.

**Returns** the tracked variables.

neural_monitor.monitor.**get_tracked_variables**() → Dict
Retrieves the values of tracked variables.

**Returns** a dictionary containing the values of tracked variables associated with the given names.

neural_monitor.monitor.**track**(*name: str*, *x: Union[torch.Tensor, torch.nn.Module]*, *direction: Optional[str] = None*) → Union[torch.Tensor, torch.nn.Module]
An identity function that registers hooks to track the value and gradient of the specified tensor.

Here is an example of how to track an intermediate output

```
from neural_monitor import track, get_tracked_variables
import nueralnet_pytorch as nnt

input = ...
conv1 = track('op', nn.Conv2d(dim, 4, 3), 'all')
conv2 = nn.Conv2d(4, 5, 3)
intermediate = conv1(input)
output = track('conv2_output', conv2(intermediate), 'all')
loss = T.sum(output ** 2)
loss.backward(retain_graph=True)
d_inter = T.autograd.grad(loss, intermediate, retain_graph=True)
d_out = T.autograd.grad(loss, output)
tracked = get_tracked_variables()

testing.assert_allclose(tracked['conv2_output'], nnt.utils.to_numpy(output))
testing.assert_allclose(np.stack(tracked['grad_conv2_output']), nnt.utils.to_
↪numpy(d_out[0]))
testing.assert_allclose(tracked['op'], nnt.utils.to_numpy(intermediate))
for d_inter_, tracked_d_inter_ in zip(d_inter, tracked['grad_op_output']):
    testing.assert_allclose(tracked_d_inter_, nnt.utils.to_numpy(d_inter_))
```

**Parameters**

- **name** – name of the tracked tensor.
- **x** – tensor or module to be tracked. If module, the output of the module will be tracked.
- **direction** – there are 4 options

  None: tracks only value.

  'forward': tracks only value.

  'backward': tracks only gradient.

  'all': tracks both value and gradient.

  Default: None.

**Returns** *x*.

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX