Pytorch Hooks for Analysis and Debugging

Alexander Binder

February 8, 2021

Recap: PyTorch Modules

e.g. `nn.Conv2d`, `nn.Linear` – a class
see `https://pytorch.org/docs/master/generated/torch.nn.Module.html#torch.nn.Module`

- takes a feature map as input, compute next feature map (via `.forward(self,*args)`)
- contain trainable parameters – `torch.nn.parameter.Parameter`
  - a tensor with *requires_grad* = *True*
  - meaning: to be used for updating its values during training,
  - will be saved in `state_dict` (also buffers)
  - has a its own iterators (e.g. `.named_parameters()`), will be given to the optimizer for updating its values
- has convenience methods
  - `.to(device)` to move all tensors to another device
  - iterators over modules,parameters, buffers inside

Forward hooks (for a module)

- can be registered to a module, executed after the forward pass of this module (see pre-hook)
- signature:
  `hook(module, inputtensor, outputtensor) -> None`
- how to register them ?
  below example for a module `some_module`
  `handle=net.some_module.register_forward_hook(hook)`

Forward hooks (for a module)

good for ?

- printing stats of feature maps (see trivial example)
- saving feature maps to disk for further analysis
- saving intermediate tensors into the module for later reading them out (e.g. running means)
- suitable for feature maps which never appear explicitly in the network forward

Backward hooks (for a module or a tensor)

focus here: hooks for a module

- executed after the backward pass
- good for:
  - printing and saving gradient stats!
  - some simple network attribution models
- signature:
  `hook(module, grad_in, grad_out) -> Tensor or None`
- how to register them ?
  `handle=net.some_module.register_backward_hook(hook)`

parametrized hooks

when you need to pass parameters, e.g. info on filepaths for saving ...

- create a function *a* which has hook signature + extra parameters: `a(*hooksig,*additionalparams)`
- create a function `b(*additionalparams) -> hook(*hooksig)` which returns something with the signature of a hook
  - internally `b(*additionalparams)` defines a function `hook(*hooksig)`
  - `hook(*hooksig)` calls `a(*hooksig,*additionalparams)` and returns the value of the call `a(*inputsig,*additionalparams)`
  - `b(*additionalparams)` returns the name of the string hook which is the handle to call the function
- see example

usage of the handles?

```
...
handles.append(handle)
...
use code with handles
...
#remove handles
for h in handles:
    h.remove()
handles=[]
```