Fastai is a great library with great resources implemented in an intuitive style. There are various kinds of callbacks which makes the training of the model easier as well as interesting. There is still a concept of Runner which is talked about in the Fastai Deep Learning Part2 which is kind of hard to understand. But not anymore. Today we will go through the Runner class and try to understand its concepts under the hood.

**What are Callbacks?**

Callbacks are a little mystery object which helps the training of the model. In simple language Callbacks are the special type of functions which are used to inject special behavior at different places in the training loop. There are various things which we might want to do while training our model,like:

* Print the loss and the accuracy metrics.
* Record the Losses for each batch and plot them.
* Find a good learning rate for the model.

All these things can be implemented with the help of Callbacks, in short, Callbacks help to make the training loop infinitely customizable like Sylvain Gugger said.

No matter how good a thing is, it can always be improved. And the improvement is the Runner class. But to fully understand the improved version, you have to be familiar with the basics of the CallbackHandler and its limitations. These are some great resources to understand the Callbacks. So I would recommend you to go through these blog posts before reading further.

[Understanding the Callbacks](https://pouannes.github.io/blog/callbacks-fastai/) [Implementing Callbacks in Fastai](https://medium.com/@edwardeasling/implementing-callbacks-in-fast-ai-1c23de25b6eb)

After going through the above articles you must have noticed that the Callbacks has to inherit from the base Callback which has different methods of its own as seen below. It's also worth noting that the only reason we inherit from this class is to handle the exceptions which might rise

when someone calls a method onto a callback which is not defined in its scope.

|  |
| --- |
| *class* Callback():  *def* begin\_fit(*self*, *learn*):  self.learn = learn  return True  *def* after\_fit(*self*): return True  *def* begin\_epoch(*self*, *epoch*):  self.epoch=epoch  return True  *def* begin\_validate(*self*): return True  *def* after\_epoch(*self*): return True  *def* begin\_batch(*self*, *xb*, *yb*):  self.xb,self.yb = xb,yb  return True  *def* after\_loss(*self*, *loss*):  self.loss = loss  return True  *def* after\_backward(*self*): return True  *def* after\_step(*self*): return True |

Suppose you define a TestCallback which will stop the execution once the number of iterations is greater than 10. The Callback is defined as seen below.

|  |
| --- |
| *class* TestCallback(*Callback*):  *def* begin\_fit(*self*,*learn*):  super().begin\_fit(learn)  self.n\_iters = 0  return True    *def* after\_step(*self*):  self.n\_iters += 1  print(self.n\_iters)  if self.n\_iters>=10: self.learn.stop = True  return True |

So if we call a method like “after\_loss” on the TestCallback then it would look for the method in its own scope and if not found then will look for the super() class which is Callback and would perform that method. Thus, an error is averted.

Hence this inheritance has to be improved in future.

Another problem which needs to be addressed is the duplication of the code in the CallbackHandler() which can be avoided with a different class.

**Now let’s have a look at the Runner class.**

The Runner class can be seen below has the training loops inside it with only callbacks and the callback functions initialised. It can look a bit intimidating but don't worry we will break down the class and get through it. The main magic is happening inside the \_\_init\_\_( ) as well as the \_\_call\_\_( ) methods.

|  |
| --- |
| *class* Runner():  *def* \_\_init\_\_(*self*, *cbs*=None, *cb\_funcs*=None):  cbs = listify(cbs) # converts the arguments into type list.  for cbf in listify(cb\_funcs):  cb = cbf()  setattr(self, cb.name, cb)  cbs.append(cb)  self.stop,self.cbs = False,[TrainEvalCallback()]+cbs  @property  *def* opt(*self*): return self.learn.opt  @property  *def* model(*self*): return self.learn.model  @property  *def* loss\_func(*self*): return self.learn.loss\_func  @property  *def* data(*self*): return self.learn.data  *def* one\_batch(*self*, *xb*, *yb*):  self.xb,self.yb = xb,yb  if self('begin\_batch'): return  self.pred = self.model(self.xb)  if self('after\_pred'): return  self.loss = self.loss\_func(self.pred, self.yb)  if self('after\_loss') or not self.in\_train: return  self.loss.backward()  if self('after\_backward'): return  self.opt.step()  if self('after\_step'): return  self.opt.zero\_grad()  *def* all\_batches(*self*, *dl*):  self.iters = len(dl)  for xb,yb in dl:  if self.stop: break  self.one\_batch(xb, yb)  self('after\_batch')  self.stop=False  *def* fit(*self*, *epochs*, *learn*):  self.epochs,self.learn = epochs,learn  try:  for cb in self.cbs: cb.set\_runner(self)  if self('begin\_fit'): return  for epoch in range(epochs):  self.epoch = epoch  if not self('begin\_epoch'): self.all\_batches(self.data.train\_dl)  with torch.no\_grad():  if not self('begin\_validate'): self.all\_batches(self.data.valid\_dl)  if self('after\_epoch'): break    finally:  self('after\_fit')  self.learn = None  *def* \_\_call\_\_(*self*, *cb\_name*):  for cb in sorted(self.cbs, key=*lambda* x: x.\_order):  f = getattr(cb, cb\_name, None)  if f and f(): return True  return False |

**Upgraded Callback Class:**

|  |
| --- |
| *class* Callback():  \_order=0  *def* set\_runner(*self*, *run*): self.run=run  *def* \_\_getattr\_\_(*self*, *k*): return getattr(self.run, k)  @property  *def* name(*self*):  name = re.sub(r'Callback$', '', self.\_\_class\_\_.\_\_name\_\_)  return camel2snake(name or 'callback') |